

Table of Contents

Overview

 What's Machine Learning?

 Machine Learning Studio

 What's the Studio?

 Studio capabilities

 Infographic: ML basics

 Team Data Science Process

 Overview

 Lifecycle

 Walkthroughs

 Frequently asked questions

 What's new?

Get Started

 Create your first experiment

 Example walkthrough

 Create a predictive solution

 1: Create a workspace

 2: Upload data

 3: Create experiment

 4: Train and evaluate

 5: Deploy web service

 6: Access web service

Data Science for Beginners

 1: Five questions

 2: Is your data ready?

 3: Ask the right question

 4: Predict an answer

 5: Copy other people's work

R quick start

How To

[Set up tools and utilities](#)

[Set up environments](#)

[Set up virtual machines](#)

[Customize Hadoop](#)

[Set up a virtual machine](#)

[Manage a workspace](#)

[Analyze business needs](#)

[Technical needs](#)

[Identify your scenario](#)

[Acquire and understand data](#)

[Load data into storage](#)

[Import training data](#)

[Explore and visualize data](#)

[Develop models](#)

[Engineer and select features](#)

[Create and train models](#)

[Deploy and consume models](#)

[Overview](#)

[Deploy models](#)

[Manage web services](#)

[Retrain models](#)

[Consume models](#)

[Examples](#)

[Sample experiments](#)

[Sample datasets](#)

[Customer churn example](#)

[End-to-end scenarios](#)

[Web service examples](#)

[Reference](#)

[PowerShell](#)

[Algorithm & Module reference](#)

[REST](#)

Related

[Cortana Intelligence Gallery](#)

[Overview](#)

[Industries](#)

[Solutions](#)

[Experiments](#)

[Jupyter Notebooks](#)

[Competitions](#)

[Competitions FAQ](#)

[Tutorials](#)

[Collections](#)

[Custom Modules](#)

[Cortana Analytics](#)

[APIs](#)

Resources

[Machine Learning REST Error Codes](#)

[Net# Neural Networks Language](#)

[Pricing](#)

[Service updates](#)

[Blog](#)

[MSDN forum](#)

[Stack Overflow](#)

[Videos](#)

[Get help from live chat](#)

Introduction to machine learning in the cloud

1/17/2017 • 6 min to read • [Edit on GitHub](#)

What is machine learning?

Machine learning is a technique of data science that helps computers learn from existing data in order to forecast future behaviors, outcomes, and trends.

These forecasts or predictions from machine learning can make apps and devices smarter. When you shop online, machine learning helps recommend other products you might like based on what you've purchased. When your credit card is swiped, machine learning compares the transaction to a database of transactions and helps detect fraud. When your robot vacuum cleaner vacuums a room, machine learning helps it decide whether the job is done.

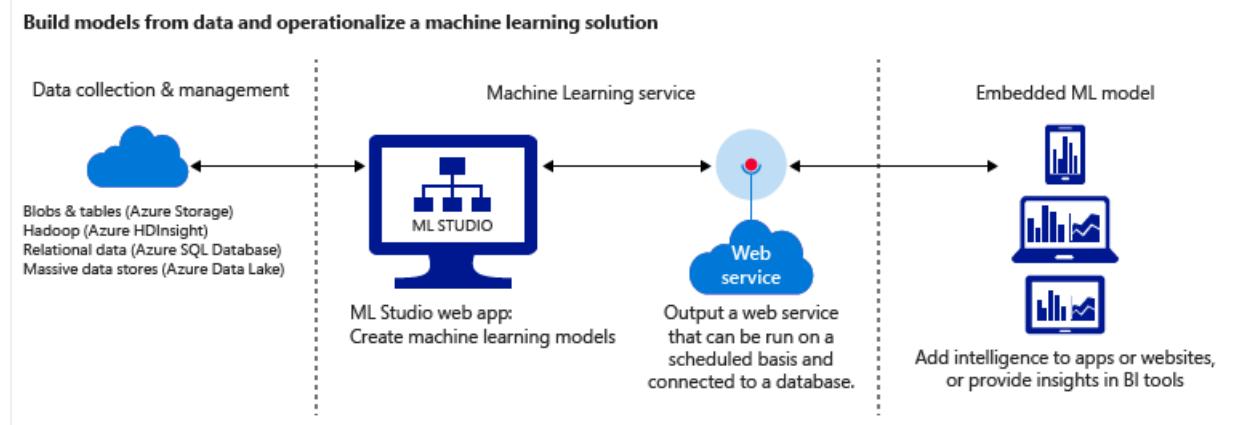
For a brief overview, try the video series [Data Science for Beginners](#). Without using jargon or math, Data Science for Beginners introduces machine learning and steps you through a simple predictive model.

What is Machine Learning in the Microsoft Azure cloud?

Azure Machine Learning is a cloud predictive analytics service that makes it possible to quickly create and deploy predictive models as analytics solutions.

You can work from a ready-to-use library of algorithms, use them to create models on an internet-connected PC, and deploy your predictive solution quickly. Start from ready-to-use examples and solutions in the [Cortana Intelligence Gallery](#).

Azure Machine Learning: Basic workflow



Azure Machine Learning not only provides tools to model predictive analytics, but also provides a fully managed service you can use to deploy your predictive models as ready-to-consume web services.

NOTE

[Try Azure Machine Learning for free](#)

No credit card or Azure subscription needed. [Get started now >](#)

What is predictive analytics?

Predictive analytics uses math formulas called algorithms that analyze historical or current data to identify

patterns or trends in order to forecast future events.

Tools to build complete machine learning solutions in the cloud

Azure Machine Learning has everything you need to create complete predictive analytics solutions in the cloud, from a large algorithm library, to a studio for building models, to an easy way to deploy your model as a web service. Quickly create, test, operationalize, and manage predictive models.

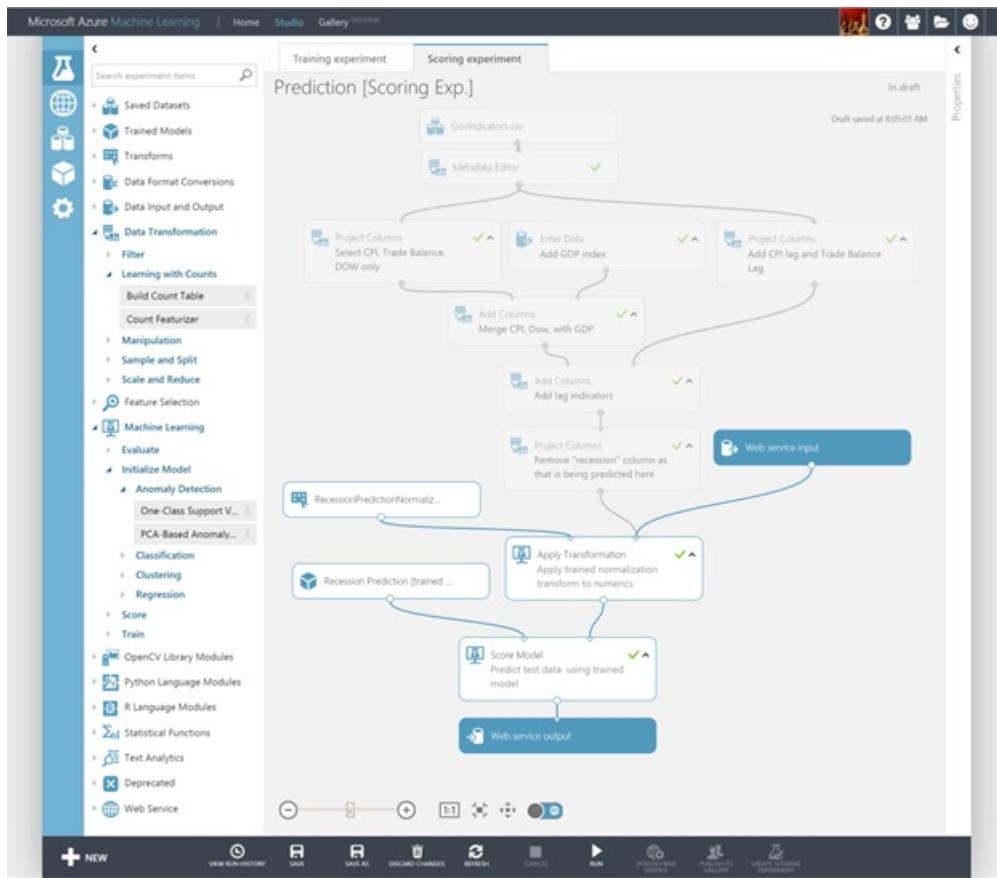
Machine Learning Studio: Create predictive models

In [Machine Learning Studio](#), you can quickly create predictive models by dragging, dropping, and connecting modules. You can experiment with different combinations, and [try it out for free](#).

- In [Cortana Intelligence Gallery](#), you can try analytics solutions authored by others or contribute your own. Post questions or comments about experiments to the community, or share links to experiments via social networks such as LinkedIn and Twitter.

The screenshot shows the Cortana Intelligence Gallery interface. At the top, there's a navigation bar with 'Cortana Intelligence Gallery' on the left, a search icon, and 'Sign in' on the right. Below the navigation bar, there's a menu with 'Browse all', 'Solution Templates', 'Experiments', 'Machine Learning APIs', 'Notebooks', 'Competitions', 'Tutorials', and 'Collections'. A main message says: 'Cortana Intelligence Gallery enables our growing community of developers and data scientists to share their analytics solutions. [Learn how to contribute](#)'. There are four main cards displayed: 1) 'COMPETITION Decoding Brain Signals Microsoft' featuring a brain and neural network graphic; 2) 'COLLECTION Cognitive Services Microsoft' featuring a brain and circuit board graphic; 3) 'NOTEBOOK Introduction to Azure ML R Microsoft' featuring a globe and R logo; 4) 'NOTEBOOK Predicting breast cancer using data from AzureML Microsoft' featuring a pink ribbon graphic. Below these, under 'Recently added', there are four smaller cards: 1) 'COLLECTION Cognitive Services' (empty), 2) 'EXPERIMENT Decoding Brain Signals - RD H Attempt 2' (empty), 3) 'COLLECTION SuperFamily' (empty), and 4) 'EXPERIMENT lab City-Market Clustering - Copy' (empty). A 'See all' button is located in the top right corner of the recently added section.

- Use a large library of [Machine Learning algorithms and modules](#) in Machine Learning Studio to jump-start your predictive models. Choose from sample experiments, R and Python packages, and best-in-class algorithms from Microsoft businesses like Xbox and Bing. Extend Studio modules with your own custom R and Python scripts.



Operationalize predictive analytics solutions by publishing your own

The following tutorials show you how to operationalize your predictive analytics models:

- [Deploy web services](#)
- [Train and retrain models through APIs](#)
- [Manage web service endpoints](#)
- [Scale a web service](#)
- [Consume web services](#)

Key machine learning terms and concepts

Machine learning terms can be confusing. Here are definitions of key terms to help you. Use comments following to tell us about any other term you'd like defined.

Data exploration, descriptive analytics, and predictive analytics

Data exploration is the process of gathering information about a large and often unstructured data set in order to find characteristics for focused analysis. **Data mining** refers to automated data exploration.

Descriptive analytics is the process of analyzing a data set in order to summarize what happened. The vast majority of business analytics - such as sales reports, web metrics, and social networks analysis - are descriptive.

Predictive analytics is the process of building models from historical or current data in order to forecast future outcomes.

Supervised and unsupervised learning

Supervised learning algorithms are trained with labeled data - in other words, data comprised of examples of the answers wanted. For instance, a model that identifies fraudulent credit card use would be trained from a data set with labeled data points of known fraudulent and valid charges. Most machine learning is supervised.

Unsupervised learning is used on data with no labels, and the goal is to find relationships in the data. For instance, you might want to find groupings of customer demographics with similar buying habits.

Model training and evaluation

A machine learning model is an abstraction of the question you are trying to answer or the outcome you want to predict. Models are trained and evaluated from existing data.

Training data

When you train a model from data, you use a known data set and make adjustments to the model based on the data characteristics to get the most accurate answer. In Azure Machine Learning, a model is built from an algorithm module that processes training data and functional modules, such as a scoring module.

In supervised learning, if you're training a fraud detection model, you use a set of transactions that are labeled as either fraudulent or valid. You split your data set randomly, and use part to train the model and part to test or evaluate the model.

Evaluation data

Once you have a trained model, evaluate the model using the remaining test data. You use data you already know the outcomes for, so that you can tell whether your model predicts accurately.

Other common machine learning terms

- **algorithm:** A self-contained set of rules used to solve problems through data processing, math, or automated reasoning.
- **anomaly detection:** A model that flags unusual events or values and helps you discover problems. For example, credit card fraud detection looks for unusual purchases.
- **categorical data:** Data that is organized by categories and that can be divided into groups. For example a categorical data set for autos could specify year, make, model, and price.
- **classification:** A model for organizing data points into categories based on a data set for which category groupings are already known.
- **feature engineering:** The process of extracting or selecting features related to a data set in order to enhance the data set and improve outcomes. For instance, airfare data could be enhanced by days of the week and holidays. See [Feature selection and engineering in Azure Machine Learning](#).
- **module:** A functional part in a Machine Learning Studio model, such as the Enter Data module that enables entering and editing small data sets. An algorithm is also a type of module in Machine Learning Studio.
- **model:** A supervised learning model is the product of a machine learning experiment comprised of training data, an algorithm module, and functional modules, such as a Score Model module.
- **numerical data:** Data that has meaning as measurements (continuous data) or counts (discrete data). Also referred to as *quantitative data*.
- **partition:** The method by which you divide data into samples. See [Partition and Sample](#) for more information.
- **prediction:** A prediction is a forecast of a value or values from a machine learning model. You might also see the term "predicted score." However, predicted scores are not the final output of a model. An evaluation of the model follows the score.
- **regression:** A model for predicting a value based on independent variables, such as predicting the price of a car based on its year and make.
- **score:** A predicted value generated from a trained classification or regression model, using the [Score Model module](#) in Machine Learning Studio. Classification models also return a score for the probability of the predicted value. Once you've generated scores from a model, you can evaluate the model's accuracy using the [Evaluate Model module](#).
- **sample:** A part of a data set intended to be representative of the whole. Samples can be selected randomly or based on specific features of the data set.

Next steps

You can learn the basics of predictive analytics and machine learning using a [step-by-step tutorial](#) and by [building](#)

on samples.

What is Azure Machine Learning Studio?

1/17/2017 • 5 min to read • [Edit on GitHub](#)

Microsoft Azure Machine Learning Studio is a collaborative, drag-and-drop tool you can use to build, test, and deploy predictive analytics solutions on your data. Machine Learning Studio publishes models as web services that can easily be consumed by custom apps or BI tools such as Excel.

Machine Learning Studio is where data science, predictive analytics, cloud resources, and your data meet.

NOTE

[Try Azure Machine Learning for free](#)

No credit card or Azure subscription needed. [Get started now >](#)

The Machine Learning Studio interactive workspace

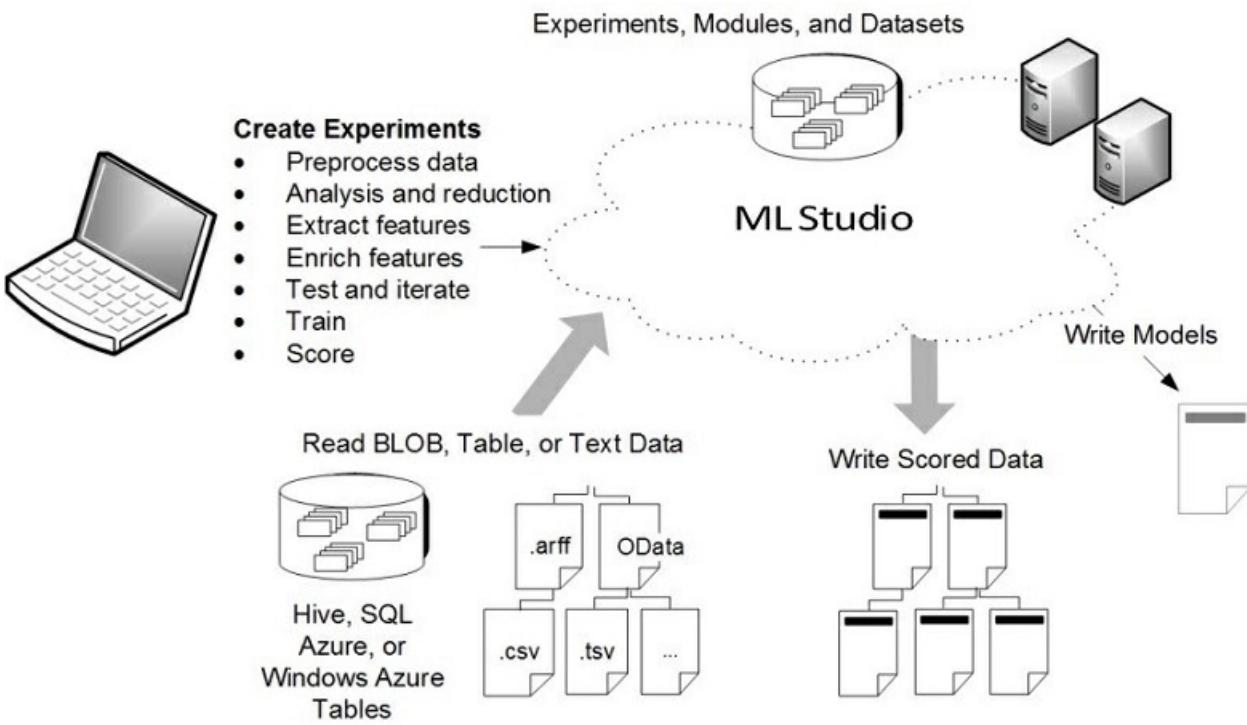
To develop a predictive analysis model, you typically use data from one or more sources, transform and analyze that data through various data manipulation and statistical functions, and generate a set of results. Developing a model like this is an iterative process. As you modify the various functions and their parameters, your results converge until you are satisfied that you have a trained, effective model.

Azure Machine Learning Studio gives you an interactive, visual workspace to easily build, test, and iterate on a predictive analysis model. You drag-and-drop **datasets** and analysis **modules** onto an interactive canvas, connecting them together to form an **experiment**, which you run in Machine Learning Studio. To iterate on your model design, you edit the experiment, save a copy if desired, and run it again. When you're ready, you can convert your **training experiment** to a **predictive experiment**, and then publish it as a **web service** so that your model can be accessed by others.

There is no programming required, just visually connecting datasets and modules to construct your predictive analysis model.

TIP

To download and print a diagram that gives an overview of the capabilities of Machine Learning Studio, see [Overview diagram of Azure Machine Learning Studio capabilities](#).



Get started with Machine Learning Studio

When you first enter [Machine Learning Studio](#) you see the **Home** page. From here you can view documentation, videos, webinars, and find other valuable resources.

Click the upper-left menu  and you'll see several options.

Cortana Intelligence

Click **Cortana Intelligence** and you'll be taken to the home page of the [Cortana Intelligence Suite](#). The Cortana Intelligence Suite is a fully managed big data and advanced analytics suite to transform your data into intelligent action. See the Suite home page for full documentation, including customer stories.

Azure Machine Learning

There are two options here, **Home**, the page where you started, and **Studio**.

Click **Studio** and you'll be taken to the [Azure Machine Learning Studio](#). First you'll be asked to sign in using your Microsoft account, or your work or school account. Once signed in, you'll see the following tabs on the left:

- **PROJECTS** - Collections of experiments, datasets, notebooks, and other resources representing a single project
- **EXPERIMENTS** - Experiments that you have created and run or saved as drafts
- **WEB SERVICES** - Web services that you have deployed from your experiments
- **NOTEBOOKS** - Jupyter notebooks that you have created
- **DATASETS** - Datasets that you have uploaded into Studio
- **TRAINED MODELS** - Models that you have trained in experiments and saved in Studio
- **SETTINGS** - A collection of settings that you can use to configure your account and resources.

Gallery

Click **Gallery** and you'll be taken to the [Cortana Intelligence Gallery](#). The Gallery is a place where a community of data scientists and developers share solutions created using components of the Cortana Intelligence Suite.

For more information about the Gallery, see [Share and discover solutions in the Cortana Intelligence Gallery](#).

Components of an experiment

An experiment consists of datasets that provide data to analytical modules, which you connect together to construct a predictive analysis model. Specifically, a valid experiment has these characteristics:

- The experiment has at least one dataset and one module
- Datasets may be connected only to modules
- Modules may be connected to either datasets or other modules
- All input ports for modules must have some connection to the data flow
- All required parameters for each module must be set

You can create an experiment from scratch, or you can use an existing sample experiment as a template. For more information, see [Use sample experiments to create new experiments](#).

For an example of creating a simple experiment, see [Create a simple experiment in Azure Machine Learning Studio](#).

For a more complete walkthrough of creating a predictive analytics solution, see [Develop a predictive solution with Azure Machine Learning](#).

Datasets

A dataset is data that has been uploaded to Machine Learning Studio so that it can be used in the modeling process. A number of sample datasets are included with Machine Learning Studio for you to experiment with, and you can upload more datasets as you need them. Here are some examples of included datasets:

- **MPG data for various automobiles** - Miles per gallon (MPG) values for automobiles identified by number of cylinders, horsepower, etc.
- **Breast cancer data** - Breast cancer diagnosis data.
- **Forest fires data** - Forest fire sizes in northeast Portugal.

As you build an experiment you can choose from the list of datasets available to the left of the canvas.

For a list of sample datasets included in Machine Learning Studio, see [Use the sample data sets in Azure Machine Learning Studio](#).

Modules

A module is an algorithm that you can perform on your data. Machine Learning Studio has a number of modules ranging from data ingress functions to training, scoring, and validation processes. Here are some examples of included modules:

- [Convert to ARFF](#) - Converts a .NET serialized dataset to Attribute-Relation File Format (ARFF).
- [Compute Elementary Statistics](#) - Calculates elementary statistics such as mean, standard deviation, etc.
- [Linear Regression](#) - Creates an online gradient descent-based linear regression model.
- [Score Model](#) - Scores a trained classification or regression model.

As you build an experiment you can choose from the list of modules available to the left of the canvas.

A module may have a set of parameters that you can use to configure the module's internal algorithms. When you select a module on the canvas, the module's parameters are displayed in the **Properties** pane to the right of the canvas. You can modify the parameters in that pane to tune your model.

For some help navigating through the large library of machine learning algorithms available, see [How to choose algorithms for Microsoft Azure Machine Learning](#).

Deploying a predictive analytics web service

Once your predictive analytics model is ready, you can deploy it as a web service right from Machine Learning Studio. For more details on this process, see [Deploy an Azure Machine Learning web service](#).

Overview diagram of Azure Machine Learning Studio capabilities

1/17/2017 • 1 min to read • [Edit on GitHub](#)

The [Microsoft Azure Machine Learning Studio Capabilities Overview](#) diagram gives you a high-level overview of how you can use Machine Learning Studio to develop a predictive analytics model and operationalize it in the Azure cloud.

Azure Machine Learning Studio has available a large number of machine learning algorithms, along with modules that help with data input, output, preparation, and visualization. Using these components you can develop a predictive analytics experiment, iterate on it, and use it to train your model. Then with one click you can operationalize your model in the Azure cloud so that it can be used to score new data.

This diagram demonstrates how all those pieces fit together.

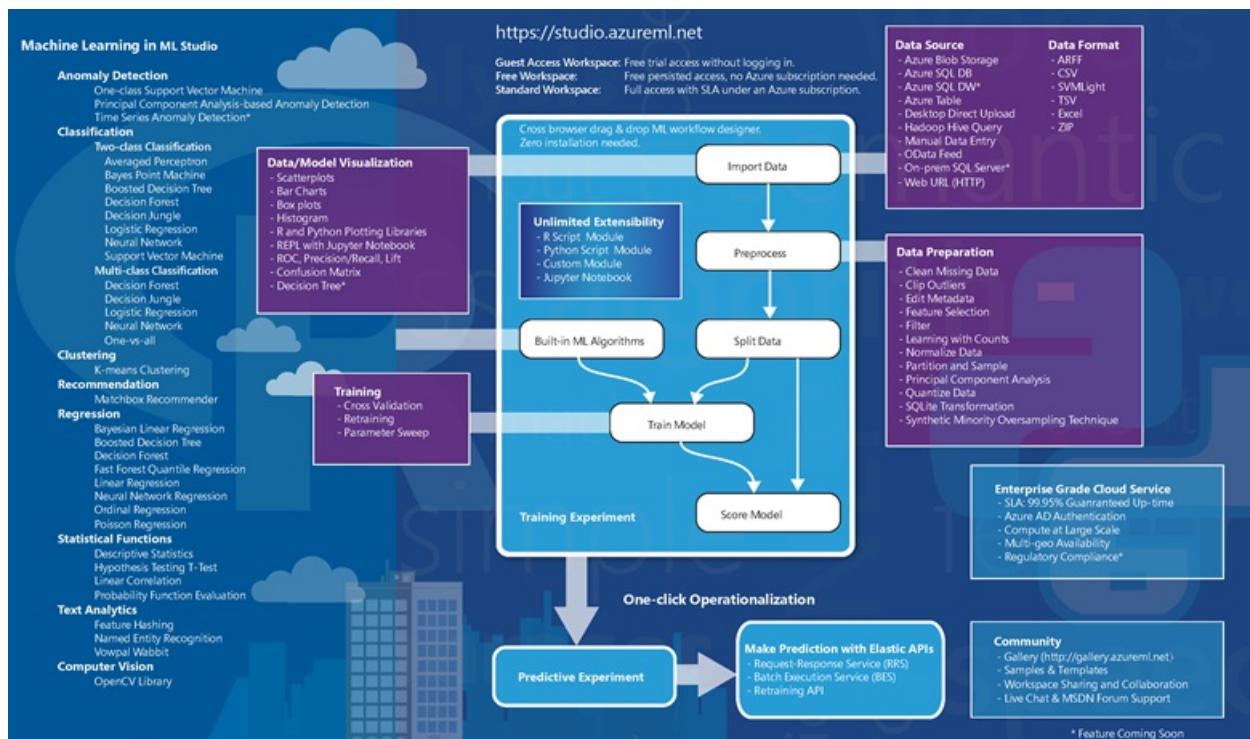
NOTE

See [Machine learning algorithm cheat sheet for Microsoft Azure Machine Learning Studio](#) for additional help in navigating through and choosing the machine learning algorithms available in Machine Learning Studio.

Download the Machine Learning Studio overview diagram

Download the [Microsoft Azure Machine Learning Studio Capabilities Overview](#) diagram and get a high-level view of the capabilities of Machine Learning Studio. To keep it nearby, you can print the diagram in tabloid size (11 x 17 in.).

Download the diagram here: [Microsoft Azure Machine Learning Studio Capabilities Overview](#)



More help with Machine Learning Studio

- For an overview of Microsoft Azure Machine Learning, see [Introduction to machine learning on Microsoft Azure](#).
- For an overview of Machine Learning Studio, see [What is Azure Machine Learning Studio?](#).
- For a detailed discussion of the machine learning algorithms available in Machine Learning Studio, see [How to choose algorithms for Microsoft Azure Machine Learning](#).

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

Downloadable Infographic: Machine learning basics with algorithm examples

1/17/2017 • 1 min to read • [Edit on GitHub](#)

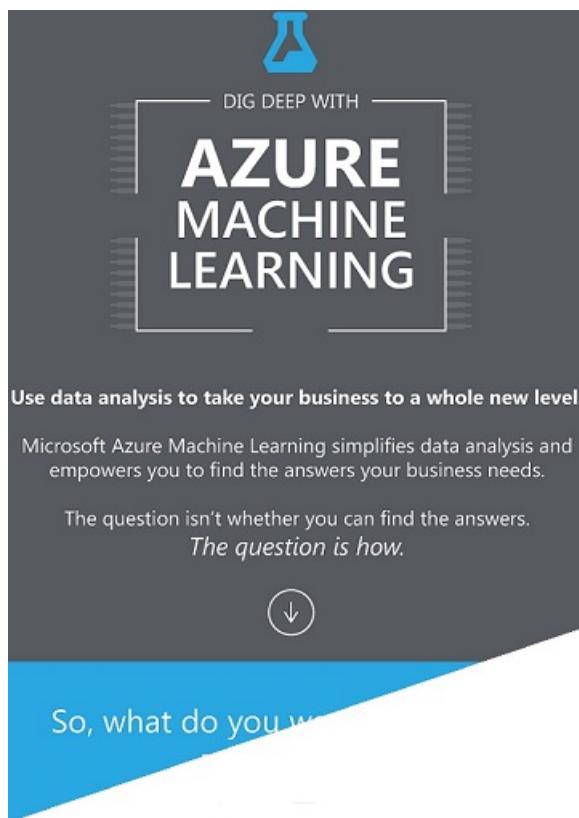
Download this easy-to-understand infographic overview of machine learning basics to learn about popular algorithms used to answer common machine learning questions. Algorithm examples help the machine learning beginner understand which algorithms to use and what they're used for.

Popular algorithms in Machine Learning Studio

Azure Machine Learning Studio comes with a large library of algorithms for predictive analytics. The infographic identifies four popular families of algorithms - regression, anomaly detection, clustering, and classification - and provides links to working examples in the [Cortana Intelligence Gallery](#). The Gallery contains example experiments and tutorials that demonstrate how these algorithms can be applied in many real-world solutions.

Download the infographic with algorithm examples

[Download: Infographic of machine learning basics with links to algorithm examples \(PDF\)](#)



More help with algorithms for beginners and advanced users

- For a deeper discussion of the different types of machine learning algorithms, how they're used, and how to choose the right one for your solution, see [How to choose algorithms for Microsoft Azure Machine Learning](#).
- For a list by category of all the machine learning algorithms available in Machine Learning Studio, see [Initialize Model](#) in the Machine Learning Studio Algorithm and Module Help.
- For a complete alphabetical list of algorithms and modules in Machine Learning Studio, see [A-Z list of Machine Learning Studio modules](#) in Machine Learning Studio Algorithm and Module Help.

- To download and print a diagram that gives an overview of the capabilities of Machine Learning Studio, see [Overview diagram of Azure Machine Learning Studio capabilities](#).
- For an overview of the Cortana Intelligence Gallery and the many community-generated resources available there, see [Share and discover resources in the Cortana Intelligence Gallery](#).

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

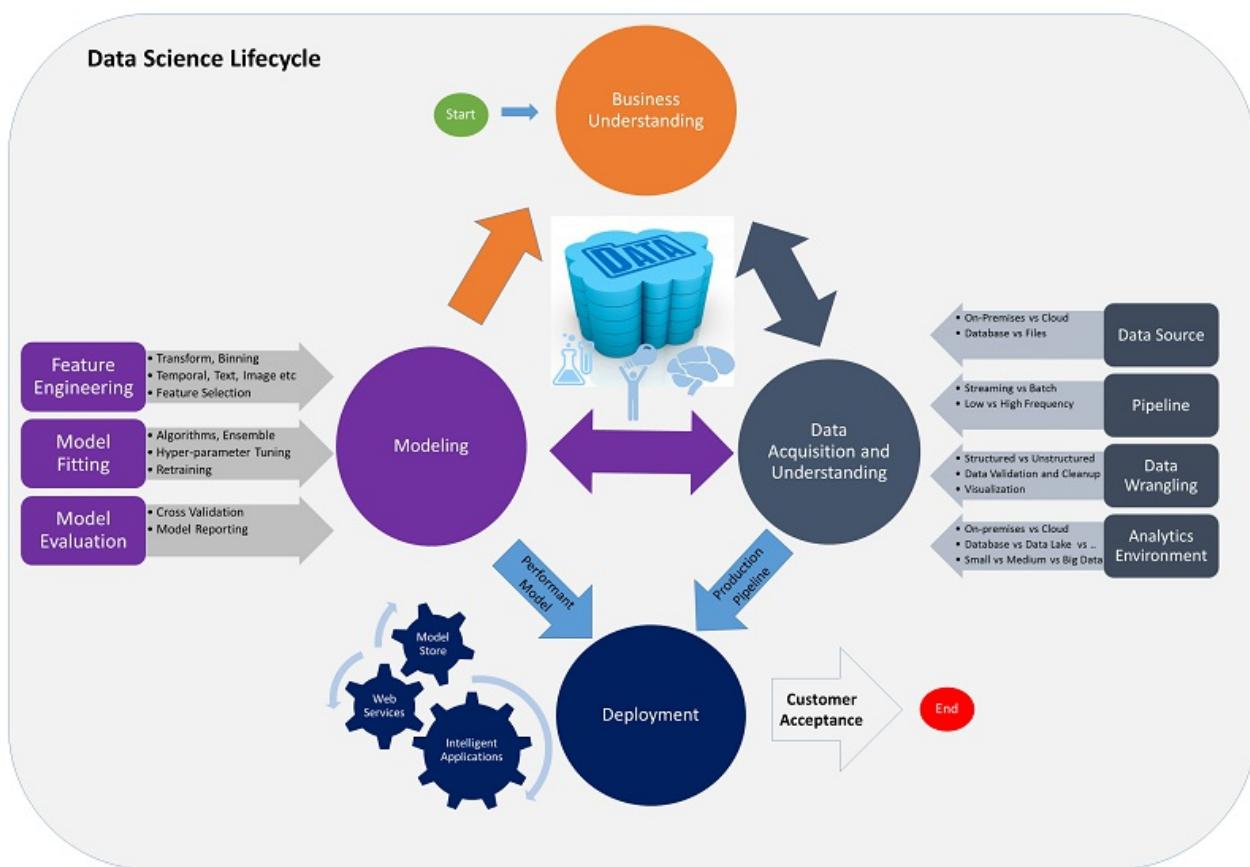
Team Data Science Process lifecycle

1/17/2017 • 13 min to read • [Edit on GitHub](#)

The Team Data Science Process (TDSP) provides a recommended lifecycle that you can use to structure the development of your data science projects. The lifecycle outlines the steps, from start to finish, that projects usually follow when they are executed. If you are using another data science lifecycle, such as CRISP-DM, KDD or your organization's own custom process, you can still use the task-based TDSP in the context of those development lifecycles.

This lifecycle has been designed for data science projects that are intended to ship as part of intelligent applications. These applications deploy machine learning or artificial intelligence models for predictive analytics. Exploratory data science projects and ad hoc or on-off analytics projects can also benefit from using this process, but in such cases some steps described may not be needed.

Here is a visual representation of the **Team Data Science Process lifecycle**.



The TDSP lifecycle is composed of five major stages that are executed iteratively. These include:

- **Business Understanding**
- **Data Acquisition and Understanding**
- **Modeling**
- **Deployment**
- **Customer Acceptance**

For each stage, we provide the following information:

- **Goals**: the specific objectives itemized.
- **How to do it**: the specific tasks outlined and guidance provided on completing them.

- **Artifacts:** the deliverables and the support for producing them.

1. Business Understanding

Goals

- The **key variables** are specified that are to serve as the **model targets** and whose related metrics are used determine the success for the project.
- The relevant **data sources** are identified that the business has access to and or from other sources when needed.

How to do it

There are two main tasks addressed in this stage:

- **Define Objectives:** Work with your customer and other stakeholders to understand and identify the business problems. Formulate questions that define the business goals and that data science techniques can target.
- **Identify data sources:** Find the relevant data that helps you answer the questions that define the objectives of the project.

1.1 Define Objectives

1. A central objective of this step is to identify the key **business variables** that the analysis needs to predict. These variables are referred to as the **model targets** and the metrics associated with them are used to determine the success of the project. Sales forecast or the probability of an order being fraudulent are examples of such targets.
2. Define the **project goals** by asking and refining "sharp" questions that are relevant and specific and unambiguous. Data science is the process of using names and numbers to answer such questions. "*When choosing your question, imagine that you are approaching an oracle that can tell you anything in the universe, as long as the answer is a number or a name*". For additional guidance, see the **Ask a Sharp Question** section of the [How to do Data Science](#) blog. Data science / machine learning is typically used to answer five types of questions:
 - How much or how many? (regression)
 - Which category? (classification)
 - Which group? (clustering)
 - Is this weird? (anomaly detection)
 - Which option should be taken? (recommendation)
3. Define the **project team** by specifying the roles and responsibilities of its members. Develop a high-level milestone plan that you iterate on as more information is discovered.
4. **Define success metrics.** For example: Achieve customer churn prediction accuracy of X% by the end of this 3-month project, so that we can offer promotions to reduce churn. The metrics must be **SMART**:
 - Specific
 - Measurable
 - Achievable
 - Relevant
 - Time-bound

1.2 Identify Data Sources

Identify data sources that contain known examples of answers to your sharp questions. Look for the following data:

- Data that is **Relevant** to the question. Do we have measures of the target and features that are related to the target?
- Data that is an **Accurate measure** of our model target and the features of interest.

It is not uncommon, for example, to find that existing systems need to collect and log additional kinds of data to address the problem and achieve the project goals. In this case, you may want to look for external data sources or update your systems to collect newer data.

Artifacts

Here are the deliverables in this stage:

- **Charter Document:** A standard template is provided in the TDSP project structure definition. This is a living document that is updated throughout the project as new discoveries are made and as business requirements change. The key is to iterate upon this document, adding more detail, as you progress through the discovery process. Keep the customer and other stakeholders involved in making the changes and clearly communicate the reasons for the changes to them.
- **Data Sources:** This is part of the Data Report that is found in the TDSP project structure. It describes the sources for the raw data. In later stages, you fill in additional details like scripts to move the data to your analytic environment.
- **Data:** This document provides the descriptions and the schema (data types, information on validation rules, if any) for the data that is used to answer the question. The entity-relation diagrams or descriptions are included too, if available.

2. Data Acquisition and Understanding

Goals

- A clean, high-quality dataset whose relations to the target variables are understood that are located in the analytics environment, ready to model.
- A solution architecture of the data pipeline to refresh and score data regularly has been developed.

How to do it

There are three main tasks addressed in this stage:

- **Ingest the data** into the target analytic environment.
- **Explore the data** to determine if the data quality is adequate to answer the question.
- **Set up a data pipeline** to score new or regularly refreshed data.

2.1 Ingest the data

Set up the process to move the data from source locations to the target locations where analytics operations like training and predictions are to be executed. For technical details and options on how to do this with various Azure data services, see [Load data into storage environments for analytics](#).

2.2 Explore the data

Before you train your models, you need to develop a sound understanding of the data. Real-world datasets are often noisy or are missing values or have a host of other discrepancies. Data summarization and visualization can be used to audit the quality of your data and provide the information needed to process the data before it is ready for modeling. For guidance on cleaning the data, see [Tasks to prepare data for enhanced machine learning](#). This process is often iterative.

TDSP provides an automated utility called [IDEAR](#) to help visualize the data and prepare data summary reports. We recommend starting with IDEAR first to explore the data to help develop initial data understanding interactively with no coding and then write custom code for data exploration and visualization.

Once you are satisfied with the quality of the cleansed data, the next step is to better understand the patterns that are inherent in the data that help you choose and develop an appropriate predictive model for your target. Look for evidence for how well connected the data is to the target and whether there is sufficient data to move forward with the next modeling steps. Again, this process is often iterative. You may need to find new data sources with more accurate or more relevant data to augment the dataset initially identified in the previous stage.

2.3 Set up a data pipeline

In addition to the initial ingestion and cleaning of the data, you typically need to set up a process to score new data or refresh the data regularly as part of an ongoing learning process. This can be done by setting up a data pipeline or workflow. Here is an [example](#) of how to set up a pipeline with [Azure Data Factory](#). A solution architecture of the data pipeline is developed in this stage. The pipeline is also developed in parallel with the following stages of the data science project. The pipeline may be batch-based or a streaming/real-time or a hybrid depending on your business needs and the constraints of your existing systems into which this solution is being integrated.

Artifacts

The following are the deliverables in this stage.

- **Data Quality Report:** This report contains data summaries, relationships between each attribute and target, variable ranking etc. The [IDEAR](#) tool provided as part of TDSP can quickly generate this report on any tabular dataset such as a CSV file or a relational table.
- **Solution Architecture:** This can be a diagram or description of your data pipeline used to run scoring or predictions on new data once you have built a model. It also contains the pipeline to retrain your model based on new data. The document is stored in this [directory](#) when using the TDSP directory structure template.
- **Checkpoint Decision:** Before you begin full feature engineering and model building, you can reevaluate the project to determine whether there is sufficient value expected to continue pursuing it. You may, for example, be ready to proceed, need to collect more data, or abandon the project as the data does not exist to answer the question.

3. Modeling

Goals

- Optimal data features for the machine learning model.
- An informative ML model that predicts the target most accurately.
- An ML model that is suitable for production.

How to do it

There are three main tasks addressed in this stage:

- **Feature Engineering:** create data features from the raw data to facilitate model training.
- **Model training:** find the model that answers the question most accurately by comparing their success metrics.
- Determine if your model is **suitable for production**.

3.1 Feature Engineering

Feature engineering involves inclusion, aggregation and transformation of raw variables to create the features used in the analysis. If you want insight into what is driving a model, then you need to understand how features are related to each other and how the machine learning algorithms are to use those features. This step requires a creative combination of domain expertise and insights obtained from the data exploration step. This is a balancing act of finding and including informative variables while avoiding too many unrelated variables. Informative variables improve our result; unrelated variables introduce unnecessary noise into the model. You also need to generate these features for any new data obtained during scoring. So the generation of these features can only depend on data that is available at the time of scoring. For technical guidance on feature engineering when using various Azure data technologies, see [Feature engineering in the Data Science Process](#).

3.2 Model Training

Depending on type of question you are trying answer, there are many modeling algorithms available. For guidance on choosing the algorithms, see [How to choose algorithms for Microsoft Azure Machine Learning](#). Although this article is written for Azure Machine Learning, the guidance it provides is useful for other ML frameworks.

The process for model training includes the following steps:

- Split the input data randomly for modeling into a training data set and a test data set.
- Build the models using the training data set.
- Evaluate (training and test dataset) a series of competing machine learning algorithms along with the various associated tuning parameters (known as parameter sweep) that are geared toward answering the question of interest with the current data.
- Determine the “best” solution to answer the question by comparing the success metric between alternative methods.

NOTE

Avoid leakage: Data Leakage can be caused by the inclusion of data from outside the training dataset that allows a model or machine learning algorithm to make unrealistically good predictions. Leakage is a common reason why data scientists get nervous when they get predictive results that seem too good to be true. These dependencies can be hard to detect. To avoid this often requires iterating between building an analysis data set, creating a model, and evaluating the accuracy.

We provide an [Automated Modeling and Reporting tool](#) with TDSP that is able to run through multiple algorithms and parameter sweeps to produce a baseline model. It also produces a baseline modeling report summarizing performance of each model and parameter combination including variable importance. This process is also iterative as it can drive further feature engineering.

Artifacts

The artifacts produced in this stage include:

- **Feature Sets:** The features developed for the modeling are described in the Feature Set section of the Data Definition report. It contains pointers to the code to generate the features and description on how the feature was generated.
- **Modeling Report:** For each model that is tried, a standard report following a specified TDSP template is produced.
- **Checkpoint Decision:** Evaluate whether the model is performing well enough to deploy it to a production system. Some key questions to ask are:
 - Does the model answer the question with sufficient confidence given the test data?
 - Should try any alternative approaches: collect additional data, do more feature engineering, or experiment with other algorithms?

4. Deployment

Goal

- Models and pipeline are deployed to a production or production-like environment for final user acceptance.

How to do it

The main task addressed in this stage:

- **Operationalize the model:** Deploy the model and pipeline to a production or production-like environment for application consumption.

4.1 Operationalize a model

Once you have a set of models that perform well, they can be operationalized for other applications to consume. Depending on the business requirements, predictions are made either in real-time or on a batch basis. To be operationalized, the models have to be exposed with an open API interface that is easily consumed from various applications such online website, spreadsheets, dashboards, or line of business and backend applications. For examples of model operationalization with an Azure Machine Learning web service, see [Deploy an Azure Machine Learning web service](#). It is also a good idea to build telemetry and monitoring into the production model and the data pipeline deployed to help with system status reporting and troubleshooting.

Artifacts

- Status dashboard of system health and key metrics.
- Final modeling report with deployment details.
- Final solution architecture document.

5. Customer Acceptance

Goal

- **Finalize the project deliverables:** confirm that the pipeline, the model, and their deployment in a production environment are satisfying customer objectives.

How to do it

There are three main tasks addressed in this stage:

- **System validation:** confirm the deployed model and pipeline are meeting customer needs.
- **Project hand-off:** to the entity that will run the system in production.

The customer would validate that the system meets their business needs and the answers the questions with acceptable accuracy to deploy the system to production for use by their client application. All the documentation is finalized and reviewed. A hand-off of the project to the entity responsible for operations is completed. This could be, for example, an IT or customer data science team or an agent of the customer that is responsible for running the system in production.

Artifacts

The main artifact produced in this final stage is the [Project Final Report](#). This is the project technical report containing all details of the project that useful to learn and operate the system. A [template](#) is provided by TDSP that can be used as is or customized for specific client needs.

Summary

The [Team Data Science Process lifecycle](#) is modeled as a sequence of iterated steps that provide guidance on the tasks needed to use predictive models. These models can be deployed in a production environment to be leveraged to build intelligent applications. The goal of this process lifecycle is to continue to move a data science project forward towards a clear engagement end point. While it is true that data science is an exercise in research and discovery, being able to clearly communicate this to your team and your customers using a well defined set of artifacts that employees standardized templates can help avoid misunderstanding and increase the chance of a successful completion of a complex data science project.

Next steps

Full end-to-end walkthroughs that demonstrate all the steps in the process for [specific scenarios](#) are also provided. They are listed and linked with thumbnail descriptions in the [Team Data Science Process walkthroughs](#) topic.

Team Data Science Process walkthroughs

1/17/2017 • 2 min to read • [Edit on GitHub](#)

The end-to-end walkthroughs itemized here each demonstrate the steps in the Team Data Science Process for **specific scenarios**. They illustrate how to combine cloud, on-premise tools, and services into a workflow or pipeline to create an intelligent application.

Use SQL Data Warehouse

The [Team Data Science Process in action: using SQL Data Warehouse](#) walkthrough shows you how to build and deploy machine learning classification and regression models using SQL Data Warehouse (SQL DW) for a publicly available NYC taxi trip and fare dataset.

Use SQL Server

The [Team Data Science Process in action: using SQL Server](#) walkthrough shows you build and deploy machine learning classification and regression models using SQL Server and a publicly available NYC taxi trip and fare dataset.

Use HDInsight Hadoop clusters

The [Team Data Science Process in action: using HDInsight Hadoop clusters](#) walkthrough uses an [Azure HDInsight Hadoop cluster](#) to store, explore and feature engineer data from a publicly available NYC taxi trip and fare dataset.

Use Azure HDInsight Hadoop Clusters on a 1-TB dataset

The [Team Data Science Process in action: using Azure HDInsight Hadoop Clusters on a 1-TB dataset](#) walkthrough presents an end-to-end scenario that uses an [Azure HDInsight Hadoop cluster](#) to store, explore, feature engineer, and down sample data from a publicly available [Criteo](#) dataset.

Data Science using Python with Spark on Azure

The [Data Science using Spark on Azure HDInsight](#) walkthrough uses the Team Data Science Process in an end-to-end scenario using an [Azure HDInsight Spark cluster](#) to store, explore and feature engineer data from the publicly available NYC taxi trip and fare dataset.

Data Science using Scala with Spark on Azure

The [Data Science using Scala with Spark on Azure](#) walkthrough shows how to use Scala for supervised machine learning tasks with the Spark scalable machine learning library (MLlib) and SparkML packages on an Azure HDInsight Spark cluster. It walks you through the tasks that constitute the [Data Science Process](#): data ingestion and exploration, visualization, feature engineering, modeling, and model consumption. The models built include logistic and linear regression, random forests, and gradient boosted trees.

Use Azure Data Lake Storage and Analytics

The [Scalable Data Science in Azure Data Lake: An end-to-end Walkthrough](#) shows how to use Azure Data Lake to do data exploration and binary classification tasks on a sample of the NYC taxi dataset to predict whether or not a tip is paid by a customer.

Use R with SQL Server R Services

The [Data Science End-to-End Walkthrough using SQL Server R Services](#) walkthrough provides data scientists with a combination of R code, SQL Server data, and custom SQL functions to build and deploy an R model to SQL Server.

Use T-SQL with SQL Server R Services

The [In-Database Advanced Analytics for SQL Developers](#) walkthrough provides SQL programmers with experience building an advanced analytics solution with Transact-SQL using SQL Server R Services to operationalize an R solution.

What's next?

For an overview of topics that walk you through the tasks that comprise the data science process in Azure, see [Data Science Process](#).

Azure Machine Learning Frequently Asked Questions (FAQ): Billing, capabilities, limitations, and support

1/17/2017 • 29 min to read • [Edit on GitHub](#)

This FAQ answers questions about Azure Machine Learning, a cloud service for developing predictive models and operationalizing solutions through web services. This FAQ covers questions about using the service, including the billing model, capabilities, limitations, and support.

General questions

What is Azure Machine Learning?

Azure Machine Learning is a fully managed service that you can use to create, test, operate, and manage predictive analytic solutions in the cloud. With only a browser, you can sign-in, upload data, and immediately start machine learning experiments. Drag-and-drop predictive modeling, a large pallet of modules, and a library of starting templates makes common machine learning tasks simple and quick. For more information, see the [Azure Machine Learning service overview](#). For a machine learning introduction covering key terminology and concepts, see [Introduction to Azure Machine Learning](#).

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

What is Machine Learning Studio?

Machine Learning Studio is a workbench environment you access through a web browser. Machine Learning Studio hosts a pallet of modules with a visual composition interface that enables you to build an end-to-end, data science workflow in the form of an experiment.

For more information about Machine Learning Studio, see [What is Machine Learning Studio?](#)

What is the Machine Learning API service?

The Machine Learning API service enables you to deploy predictive models, such as those built in Machine Learning Studio, as scalable, fault-tolerant, web services. The web services created by the Machine Learning API service are REST APIs that provide an interface for communication between external applications and your predictive analytics models.

For more information, see [Connect to a Machine Learning web service](#).

Where are my classic web services listed? Where are my new Azure Resource Manager based web services listed?

Classic Web services and New Azure Resource Manager based Web services are listed in the [Microsoft Azure Machine Learning Web Services](#) portal.

Classic Web services are also listed in [Machine Learning Studio](#) on the Web services tab.

Microsoft Azure Machine Learning Web Service questions

What are Azure Machine Learning Web Services?

Machine Learning Web Services provide an interface between an application and a Machine Learning workflow scoring model. Using the Azure Machine Learning web service, an external application can communicate with a Machine Learning workflow scoring model in real time. A Machine Learning web service call returns prediction results to an external application. To make a Machine Learning web service call, you pass an API key that was created when you deployed the web service. The Machine Learning web service is based on REST, a popular architecture choice for web programming projects.

Azure Machine Learning has two types of services:

- Request-Response Service (RRS) - A low latency, highly scalable service that provides an interface to the stateless models created and deployed from the Machine Learning Studio.
- Batch Execution Service (BES) - An asynchronous service that scores a batch for data records.

There are several ways to consume the REST API and access the web service. For example, you can write an application in C#, R, or Python using the sample code generated for you when you deployed the web service.

The sample code is available on: The Consume page for the Web service in the Azure Machine Learning Web Services portal. The API Help Page in the Web service dashboard in Machine Learning Studio.

Or you can use the sample Microsoft Excel workbook created for you (also available in the web service dashboard in Studio).

What are the main updates with the new Azure ML Web Services?

For more information about the New Azure Machine Learning Web Services, refer to the [related documentation](#).

Machine Learning Studio questions

Importing and exporting data for Machine Learning

What data sources does Machine Learning support?

Data can be loaded into a Machine Learning Studio experiment in one of three ways: by uploading a local file as a dataset, by using a module to import data from cloud data services, or by importing a dataset saved from another experiment. To learn more about supported file formats, see [Import training data into Machine Learning Studio](#).

How large can the data set be for my modules?

Modules in Machine Learning Studio support datasets of up to 10 GB of dense numerical data for common use cases. If a module takes more than one input, the 10 GB is the total of all input sizes. You can also sample larger datasets via Hive or Azure SQL Database queries, or by Learning by Counts pre-processing, before ingestion.

The following types of data can expand into larger datasets during feature normalization, and are limited to less than 10 GB:

- Sparse
- Categorical
- Strings
- Binary data

The following modules are limited to datasets less than 10GB:

- Recommender modules
- SMOTE module
- Scripting modules: R, Python, SQL
- Modules where the output data size can be larger than input data size, such as Join or Feature Hashing.
- Cross-validation, Tune Model Hyperparameters, Ordinal Regression, and One-vs-All Multiclass, when number of

iterations is very large.

For datasets larger than a few GB, you should upload data to Azure storage or Azure SQL Database or use HDInsight, rather than directly uploading from local file.

What are the limits for data upload?

For datasets larger than a couple GB, upload data to Azure storage or Azure SQL Database or use HDInsight, rather than directly uploading from local file.

Can I read data from Amazon S3?

If you have a small amount of data and want to expose it via an http URL, then you can use the [Import Data](#) module. For any larger amounts of data to transfer it to Azure Storage first and then use the [Import Data](#) module to bring it into your experiment.

Is there a built-in image input capability?

You can learn about image input capability in the [Import Images](#) reference.

Modules

The algorithm, data source, data format, or data transformation operation I am looking for isn't in Azure Machine Learning Studio. What are my options?

You can visit the [user feedback forum](#) to see feature requests that we are tracking. Add your vote to a request if a capability you're looking for has already been requested. If the capability you're looking for doesn't exist, create a new request. You can view the status of your request in this forum too. We track this list closely and update the status of feature availability frequently. In addition, with the built-in support for R and Python custom transformations can be created as needed.

Can I bring my existing code into Machine Learning Studio?

Yes, you can bring your existing R or Python code into Machine Learning Studio, run it in the same experiment with Azure Machine Learning learners, and deploy the solution as a web service via Azure Machine Learning. For more information, see [Extend your experiment with R](#) and [Execute Python machine learning scripts in Azure Machine Learning Studio](#).

Is it possible to use something like PMML to define a model?

No, that is not supported, however custom R and Python code can be used to define a module.

How many modules can I execute in parallel in my experiment?

You can execute up to four modules in parallel in an experiment.

Data processing

Is there an ability to visualize data (beyond R visualizations) interactively within the experiment?

By clicking the output of a module, you can visualize the data and get statistics.

When previewing results or data in the browser, the number of rows and columns is limited, why?

Since the data is being transmitted to the browser and may be large, the data size is limited to prevent slowing down Machine Learning Studio. To visualize all the data/result, it's better to download the data and use Excel or another tool.

Algorithms

What existing algorithms are supported in Machine Learning Studio?

Machine Learning Studio provides state-of-the-art algorithms, such as Scalable Boosted Decision trees, Bayesian Recommendation systems, Deep Neural Networks, and Decision Jungles developed at Microsoft Research. Scalable

open-source machine learning packages like Vowpal Wabbit are also included. Machine Learning Studio supports machine learning algorithms for multiclass and binary classification, regression, and clustering. See the complete list of [Machine Learning Modules](#).

Do you automatically suggest the right Machine Learning algorithm to use for my data?

No, however there are various ways in Machine Learning Studio to compare the results of each algorithm to determine the right one for your problem.

Do you have any guidelines on picking one algorithm over another for the provided algorithms? See [How to choose an algorithm](#).

Are the provided algorithms written in R or Python?

No, these algorithms are mostly written in compiled languages to provide higher performance.

Are any details of the algorithms provided?

The documentation provides some information about the algorithms, and the parameters provided for tuning are described to optimize the algorithm for your use.

Is there any support for online learning?

No, currently only programmatic retraining is supported.

Can I visualize the layers of a Neural Net Model using the built-in Module?

No.

Can I create my own modules in C# or some other language?

Currently new custom modules can only be created in R.

R module

What R packages are available in Machine Learning Studio?

Machine Learning Studio supports 400+ CRAN R packages today, and here is the [current list](#) of all included packages. Also, see [Extend your experiment with R](#) to learn how to retrieve this list yourself. If the package you want is not in this list, provide the name of package at [user feedback forum](#).

Is it possible to build a custom R module?

Yes, see [Author custom R modules in Azure Machine Learning](#) for more information.

Is there a REPL environment for R?

No, there is no REPL environment for R in the studio.

Python module

Is it possible to build a custom Python module?

Not currently, but you can use one or more [Execute Python Script](#) modules to get the same result.

Is there a REPL environment for Python?

You can use the Jupyter Notebooks in Machine Learning Studio. For more information, see [Introducing Jupyter Notebooks in Azure Machine Learning Studio](#).

Web service

Retraining Models Programmatically

How do I Retrain Azure Machine Learning models programmatically?

Use the Retraining APIs. for more information, see [Retrain Machine Learning models programmatically](#). Sample code is also available in the [Microsoft Azure Machine Learning Retraining Demo](#).

Create

Can I deploy the model locally or in an application without an internet connection?

No.

Is there a baseline latency that is expected for all web services?

See the [Azure subscription limits](#)

Use

When would I want to run my predictive model as a Batch Execution service versus a Request Response service?

The Request Response service (RRS) is a low-latency, high-scale web service that is used to provide an interface to stateless models that are created and deployed from the experimentation environment. The Batch Execution service (BES) is a service for asynchronously scoring a batch of data records. The input for BES is similar to data input used in RRS. The main difference is that BES reads a block of records from a variety of sources, such as the Blob service and Table service in Azure, Azure SQL Database, HDInsight (hive query), and HTTP sources. For more information, see [How to consume Machine Learning web services](#).

How do I update the model for the deployed web service?

Updating a predictive model for an already deployed service is as simple as modifying and rerunning the experiment that you used to author and save the trained model. Once you have a new version of the trained model available, Machine Learning Studio asks you if you want to update your web service. For details on how to update a deployed web service, see [Deploy a Machine Learning web service](#).

You can also use the Retraining APIs. For more information, see [Retrain Machine Learning models programmatically](#). Sample code is also available in the [Microsoft Azure Machine Learning Retraining Demo](#).

How do I monitor my web service deployed in production?

Once a predictive model has been deployed, you can monitor it from the Azure classic portal (Classic web services only) or the Azure Machine Learning Web Services portal. Each deployed service has its own dashboard where you can see monitoring information for that service. For more information on managing your deployed web services, see [Manage a Web service using the Azure Machine Learning Web Services portal](#) and [Manage an Azure Machine Learning workspace](#).

Is there a place where I can see the output of my RRS/BES?

For RRS, the web service response is typically where you see the result. You can also write it to Azure blob storage. For BES, the output is written to a blob by default. You can also write the output to a database or table using the [Export Data](#) module.

Can I create web services only from models created in Machine Learning Studio?

No, you can also create web services directly from Jupyter Notebooks and RStudio.

Where can I find information about error codes?

See [Machine Learning Module Error Codes](#) for a list of error codes and descriptions.

Scalability

What is the scalability of the web service?

Currently, the default endpoint is provisioned with 20 concurrent RRS requests per endpoint. You can scale this to

200 concurrent requests per endpoint and you can scale each web service to 10,000 endpoints per web service as described in [Scaling a Web Service](#). For BES, each endpoint allows processing 40 requests at a time and additional requests beyond 40 requests are queued. These queued requests run automatically as the queue drains.

Are R jobs spread across nodes?

No.

How much data can I use for training?

Modules in Machine Learning Studio support datasets of up to 10 GB of dense numerical data for common use cases. If a module takes more than one input, the total size for all inputs together is 10 GB. You can also sample larger datasets via Hive or Azure SQL Database queries, or by pre-processing with [Learning with Counts](#) modules before ingestion.

The following types of data can expand into larger datasets during feature normalization, and are limited to less than 10 GB:

- sparse
- categorical
- strings
- binary data

The following modules are limited to datasets less than 10 GB:

- Recommender modules
- SMOTE module
- Scripting modules: R, Python, SQL
- Modules where the output data size can be larger than input data size, such as Join or Feature Hashing.
- Cross-Validate, Tune Model Hyperparameters, Ordinal Regression, and One-vs-All Multiclass, when number of iterations is very large.

For datasets larger than a few GB, you should upload data to Azure storage or Azure SQL Database, or use HDInsight, rather than directly uploading from a local file.

Are there any vector size limitations?

Rows and columns are each limited to the .NET limitation of Max Int: 2,147,483,647.

Can the size of the virtual machine being used to run the web service be adjusted?

No.

Security and availability

Who has access to the http endpoint for the web service by default? How do I restrict access to the endpoint?

After a web service is deployed, a default endpoint is created for that service. The default endpoint can be called using its API Key. Additional endpoints can be added with their own keys from the Azure classic portal or programmatically using the Web Service Management APIs. Access keys are needed to make calls to the web service. For more information, see [Connect to a Machine Learning web service](#).

What happens if my Azure storage account can't be found?

Machine Learning Studio relies on a user supplied Azure storage account to save intermediary data when executing the workflow. This storage account is provided to Machine Learning Studio at the time a workspace is created. After the workspace is created, if the storage account is deleted and can no longer be found, the workspace will stop

functioning and all experiments in that workspace will fail.

If you accidentally deleted the storage account, recreate the storage account with the same name in the same region as the deleted storage account. After that, resync the Access Key.

What happens if my storage account access key is out of sync?

Machine Learning Studio relies on a user supplied Azure storage account to store intermediary data when executing the workflow. This storage account is provided to Machine Learning Studio at the time a workspace is created and the Access Keys are associated with that workspace. If the Access Keys are changed, after the workspace is created, the workspace can no longer access the storage account. It will stop functioning and all experiments in that workspace will fail.

If you have changed storage account Access Keys, resync the Access Keys in the workspace using the Azure classic portal.

Support and training

Where can I get training for Azure Machine Learning?

[Azure Machine Learning Documentation Center](#) hosts video tutorials and how-to guides. These step-by-step guides provide an introduction to the services and walk through the data science life cycle of importing data, cleaning data, building predictive models and deploying them in production with Azure Machine Learning.

We are adding new material to the Machine Learning Center on an ongoing basis. You can submit requests for additional learning material on Machine Learning Center at the [user feedback forum](#).

You can also find training at [Microsoft Virtual Academy](#).

How do I get support for Azure Machine Learning?

To get technical support for Azure Machine Learning, go to [Azure Support](#) and select **Machine Learning**.

Azure Machine Learning also has a community forum on MSDN where you can ask questions related to Azure Machine Learning. The forum is monitored by the Azure Machine Learning team. Visit [Azure Forum](#).

Billing questions

How does Machine Learning billing work?

There are two components to the Azure Machine Learning service. The Machine Learning Studio and Machine Learning Web Services.

While you are evaluating Machine Learning Studio, you can use the free billing tier. The free tier also allows you to deploy a classic web service with limited capacity.

Once you have decided that Azure Machine Learning meets your needs, you can sign up for the standard tier. To sign up, you must have a Microsoft Azure Subscription.

In the standard tier you are billed a monthly for each workspace you define in Machine Learning Studio. When you run an experiment in the studio, you are billed for compute resources when you are running an experiment. When you deploy a Classic Web Service, transactions and compute hours are billed on a Pay As You Go (PAYG) basis.

The New Machine Learning Web Services introduce Billing Plans that allow for more predictability in costs. Tiered pricing offers discounted rates to customers that need a large amount of capacity.

When you create a plan, you commit to a fixed cost that comes with an included quantity of API compute hours and API transactions. If you need more included quantities, you can add additional instances to your plan. If you need a lot more included quantities, you can choose a higher tier plan that provides considerably more included quantities

and a better discounted rate.

After the included quantities in existing instances are used up, additional usage is charged at the overage rate associated with the billing plan tier.

Note: Included quantities are reallocated every 30 days and unused included quantities do not roll over to the next period.

For additional billing and pricing information, see [Machine Learning Pricing](#).

Does Machine Learning have a free trial?

Azure Machine Learning has a free subscription option (see [Machine Learning Pricing](#) for details), and Machine Learning Studio has an 8-hour quick evaluation trial available (log in to [Machine Learning Studio](#) for this trial).

In addition, when you sign up for an Azure free trial, you can try any Azure services for a month. To learn more about the Azure free trial, visit [Azure Free Trial FAQ](#).

What is a transaction?

A transaction represents an API call that Azure Machine Learning responds to. Transactions from Request-Response Service (RRS) and Batch Execution Service (BES) calls are aggregated and charged against your billing plan.

Can I use the included transaction quantities in a plan for both RRS and BES transactions?

Yes, your transactions from your RRS and BES are aggregated and charged against your billing plan.

What is an API compute hour?

An API compute hour is the billing unit for the time API Calls take to run using the ML compute resources. All your calls are aggregated for billing purposes.

How long does a typical production API call take?

Production API call times can vary significantly, generally ranging from hundreds of milliseconds to a few seconds, but may require minutes depending on the complexity of the data processing and machine learning model. The best way to estimate production API call times is to benchmark a model on the Machine Learning service.

What is a Studio Compute hour?

A Studio Compute hour is the billing unit for the aggregate time your experiments use compute resources in studio.

In the New Web Services, what is the dev/test tier meant for?

The Azure ML new Web Services provide multiple tiers that you can use to provision your billing plan. The dev/test tier is a tier that provides limited included quantities that allow you to test your experiment as new web service without incurring costs. You have the opportunity to "Kick the Tires" to see how it works.

Are there separate storage charges?

The Machine Learning Free tier does not require or allow separate storage. The Machine Learning Standard tier requires users to have an Azure storage account. Azure storage is [billed separately](#).

How does Machine Learning support high-availability work?

Production API call times can vary significantly, generally ranging from hundreds of milliseconds to a few seconds, but may require minutes depending on the complexity of the data processing and machine learning model. The best way to estimate production API call times is to benchmark a model on the Machine Learning service.

What specific kind of compute resources will my production API calls be run on?

The Machine Learning service is a multitenant service, and actual compute resources used on the backend vary and

are optimized for performance and predictability.

Management of New Web Services

What happens if I delete my plan?

The plan is removed from your subscription and you are billed for a prorated usage.

Note: You cannot delete a plan that is in use by a web service. To delete the plan, you must either assign a new plan to the web service or delete the web service.

What is a plan instance?

A plan instance is a unit of included quantities that you can add to your billing plan. When you select a billing tier for your billing plan, it comes with one instance. If you need more included quantities, you can add instances of the selected billing tier to your plan.

How many plan instances can I add?

You can have one instance of the dev/test tier in a subscription.

For tiers S1, S2, and S3 you can add as many as necessary.

Note: Depending on your anticipated usage, it may be more cost effective to upgrade to a higher included quantities tier rather than add instances to the current tier.

What happens when I change plan tiers (Upgrade / downgrade)?

The old plan is deleted and the current usage is billed on a prorated basis. A new plan with the full included quantities of the upgraded/downgraded tier is created for the rest of the period.

Note: Included quantities are allocated per period and unused quantities do not roll over.

What happens when I increase the instances in a plan?

Quantities are included on a prorated basis and may take 24 hours to be effective.

What happens when I delete an instance of a plan?

The instance is removed from your subscription and you are billed for a prorated usage.

Signing up for New Web Services plans

How do I sign up for a plan?

You have two ways to create billing plans.

When you first deploy a new web service, you can choose an existing plan or create a new plan.

Plans created in this manner are in your default region and your web service will be deployed to that region.

If you want to deploy services to regions other than your default region, you may want to define your billing plans before you deploy your service,

In that case you can log into the Azure Machine Learning Web Services portal and navigate to the plans page. From there, you can Add and Delete plans, as well as modify existing plans.

Which plan should I choose to start off with?

We recommend you that you start with the standard S1 tier and monitor your service for usage. If you find you are using your included quantities rapidly, you can add instances or move to a higher tier and get better discounted rates. You can adjust your billing plan as needed throughout your billing cycle.

Which regions are the new plans available in?

The new billing plans are available in the three production regions in which we support the new web services:

- South Central US
- West Europe
- South East Asia

I have web services in multiple regions. Do I need a plan for every region?

Yes. Plan pricing varies by region. When you deploy a web service to another region, you need to assign it a plan specific to that region.

New Web Services - Overages

How do I check if my web service usage is in overage?

You can view the usage on all your plans on the Plans page in the Azure Machine Learning Web Services portal. sign in to the portal and click the Plans menu option.

In the Transactions and Compute columns of the table, you can see the included quantities of the plan and the percentage used.

What happens when I use up the include quantities in the dev/test tier?

Services that have a dev/test tier assigned them are stopped until the next period or you move them to one of the paid tiers.

For Classic Web Services and Overages of New Web Services, how are prices calculated for Request Response (RRS) and Batch (BES) workloads?

For an RRS workload, you are charged for every API transaction call that you make and for the compute time associated with those requests. Your RRS Production API Transaction costs are calculated as the total number of API calls that you make multiplied by the price per 1,000 transactions (prorated by individual transaction). Your RRS API Production API Compute Hour costs are calculated as the amount of time required for each API call to run, multiplied by the total number of API transactions multiplied by the price per Production API Compute Hour.

For example for Standard S1 overage, 1,000,000 API Transactions that take 0.72 seconds each to run would result in $(1,000,000 * \$0.50/1K \text{ API transactions})$ in \$500 in Production API Transaction costs and $(1,000,000 * 0.72 \text{ sec} * \$2/\text{hr})$ \$400 in Production API Compute Hours, for a total of \$900.

For a BES workload, you are charged in the same manner, however, the API transaction costs represent the number of batch jobs that you submit and the compute costs represent the compute time associated with those batch jobs. Your BES Production API Transaction costs are calculated as the total number of jobs submitted multiplied by the price per 1,000 transactions (prorated by individual transaction). Your BES API Production API Compute Hour costs are calculated as the amount of time required for each row in your job to run multiplied by the total number of rows in your job multiplied by the total number of jobs multiplied by the price per Production API Compute Hour. When using the Machine Learning calculator, the transaction meter represents the number of jobs that you plan to submit and the time per transaction field represents the combined time needed for all the rows in each job to run.

For example with Standard S1 overage, if you submit 100 jobs per day that each consist of 500 rows that take 0.72 seconds each, then your monthly overage costs would be $(100 \text{ jobs per day} = 3,100 \text{ jobs/mo} * \$0.50/1K \text{ API transactions})$ \$1.55 in Production API Transaction costs and $(500 \text{ rows} * 0.72 \text{ sec} * 3,100 \text{ Jobs} * \$2/\text{hr})$ \$620 in Production API Compute Hours, for a total of \$621.55.

Azure ML Classic Web Services

Is the Pay As You Go still available? Yes, Classic Web Services are still available in Azure Machine Learning.

Azure Machine Learning Free and Standard Tier

What is included in the Azure Machine Learning Free tier?

The Azure Machine Learning Free tier is intended to provide an in-depth introduction to the Azure Machine Learning Studio. All you need is a Microsoft account to sign up. The Free tier includes free access to one Azure Machine Learning Studio workspace per [Microsoft account](#). It includes the ability to use up to 10 GB of storage and the ability to operationalize models as staging APIs. Free tier workloads are not covered by an SLA and are intended for development and personal use only. Free tier workloads can't access data by connecting to an on-premises SQL server.

What is included in the Azure Machine Learning Standard tier and plans?

The Azure Machine Learning Standard tier is a paid production version of Azure Machine Learning Studio. The Azure ML service Studio monthly fee is billed on a per workspace per month basis and prorated for partial months. Azure ML Studio experiment hours are billed per compute hour for active experimentation. Billing is prorated for partial hours.

The Azure ML API service is billed depending on whether it's a classic web services or a new web service.

The following charges are aggregated per workspace for your subscription.

- Machine Learning Workspace Subscription - The Machine Learning Workspace Subscription is a monthly fee that provides access to an ML Studio workspace and is required to run experiments both in the studio and utilizing the production APIs.
- Studio Experiment Hours - this meter aggregates all compute charges accrued by running experiments in ML Studio and running production API calls in the staging environment.
- Access data by connecting to an on-premises SQL server in your models for your training and scoring.
- For Classic Web Services:
 - Production API Compute Hours - this meter includes compute charges accrued by Web services running in production.
 - Production API Transactions (in 1000s) - this meter includes charges accrued per call to your production web service.

Apart from the preceding charges, in the case of New Web Services, charges are aggregated to the plan selected:

- Standard S1/S2/S3 API Plan (Units) - this meter represents the type of instance selected for new Web Services
- Standard S1/S2/S3 Overage API Compute Hours - this meter includes compute charges accrued by the New Web Services running in production after the included quantities in existing instance(s) are used up. The additional usage is charged at the overate rate associated with S1/S2/S3 plan tier.
- Standard S1/S2/S3 Overage API Transactions (in 1,000s) - this meter includes charges accrued per call to your production New Web Service after the included quantities in existing instance(s) are used up. The additional usage is charged at the overate rate associated with S1/S2/S3 plan tier.
- Included Quantity API Compute Hours - with the New Web Services, this meter represents the included quantity of API Compute Hours
- Included Quantity API Transactions (in 1,000s) - with the New Web Services, this meter represents the included quantity of API Transactions

How do I sign up for Azure ML Free tier?

All you need is a Microsoft account. Go to [Azure Machine Learning home](#), and click **Start Now**. Log in with your Microsoft account and a workspace in Free tier is created for you. You can start to explore and create Machine Learning experiments right away.

How do I sign up for Azure ML Standard tier?

You must first have access to an Azure subscription in order to create a Standard ML workspace. You can sign up for a 30-day free trial Azure subscription and later upgrade to a paid Azure subscription, or purchase a paid Azure subscription outright. You can then create a Machine Learning workspace from the Microsoft Azure classic portal

after gaining access to the subscription. View the [step-by-step instructions](#).

Alternatively, you can be invited by a Standard ML workspace owner to access the owner's workspace.

Can I specify my own Azure blob storage account to use with the Free tier?

No, the Standard tier is equivalent to the version of the Machine Learning service that was available before the tiers were introduced.

Can I deploy my machine learning models as APIs in the Free tier?

Yes, you can operationalize machine learning models to staging API services as part of the free tier. In order to put the staging API service into production and get a production end point for the operationalized service, you must use the Standard tier.

What is the difference between Azure Free trial and Azure Machine Learning Free tier?

The [Microsoft Azure free trial](#) offers credits that can be applied to any Azure service for one month. The Azure Machine Learning Free tier offers continuous access specifically to the Azure Machine Learning service for non-production workloads.

How do I move an experiment from the Free tier to the Standard tier?

To copy your experiments from the Free tier to the Standard tier:

1. Log into Azure Machine Learning Studio and make sure you can see both the Free workspace and the Standard workspace in the workspace selector in the top navigation bar.
2. Switch to Free workspace if you are in the Standard workspace.
3. In the experiment list view, select an experiment you'd like to copy, and click the Copy command button.
4. Select the Standard workspace from the pop-up dialog box and click the Copy button. All the associated datasets, trained model, etc. are copied together with the experiment into the Standard workspace.
5. You need to rerun the experiment and republish your web service in the Standard workspace.

Studio Workspace

Will I see different bills for different workspaces?

Workspace charges are broken out separately for each applicable meter on a single bill.

What specific kind of compute resources will my experiments be run on?

The Machine Learning service is a multitenant service, and actual compute resources used on the backend vary and are optimized for performance and predictability.

Guest access

What is Guest Access to Azure Machine Learning Studio?

Guest Access is a restricted trial experience that allows you to create and run experiments in the Azure Machine Learning Studio at no cost and without authentication. Guest sessions are non-persistent (cannot be saved) and limited to 8 hours. Other limitations include lack of R and Python support, lack of staging APIs and restricted dataset size and storage capacity. By comparison, users who choose to sign in with a Microsoft account have full access to the Free-tier of Machine Learning Studio described above which includes a persistent workspace and more comprehensive capabilities. Choose your free Machine Learning experience by clicking **Get started** on <https://studio.azureml.net>, and selecting either Guess Access or sign in with a Microsoft account.

What's New in Azure Machine Learning

1/17/2017 • 1 min to read • [Edit on GitHub](#)

The August 2016 release of Microsoft Azure Machine Learning updates provide the following features:

- Classic Web services can now be managed in the new [Microsoft Azure Machine Learning Web Services](#) portal that provides one place to manage all aspects of your Web service.
 - Which provides web service [usage statistics](#).
 - Simplifies testing of Azure Machine Learning Remote-Request calls using sample data.
 - Provides a new Batch Execution Service test page with sample data and job submission history.
 - Provides easier endpoint management.

The July 2016 release of Microsoft Azure Machine Learning updates provide the following features:

- Web services are now managed as Azure resources managed through [Azure Resource Manager](#) interfaces, allowing for the following enhancements:
 - There are new [REST APIs](#) to deploy and manage your Resource Manager based Web services.
 - There is a new [Microsoft Azure Machine Learning Web Services](#) portal that provides one place to manage all aspects of your Web service.
- Incorporates a new subscription-based, multi-region web service deployment model using Resource Manager based APIs leveraging the Resource Manager Resource Provider for Web Services.
- Introduces new [pricing plans](#) and plan management capabilities using the new Resource Manager RP for Billing.
 - You can now [deploy your web service to multiple regions](#) without needing to create a subscription in each region.
- Provides web service [usage statistics](#).
- Simplifies testing of Azure Machine Learning Remote-Request calls using sample data.
- Provides a new Batch Execution Service test page with sample data and job submission history.

In addition, the Machine Learning Studio has been updated to allow you to deploy to the new Web service model or continue to deploy to the classic Web service model.

Machine learning tutorial: Create your first data science experiment in Azure Machine Learning Studio

1/17/2017 • 16 min to read • [Edit on GitHub](#)

If you've never used **Azure Machine Learning Studio** before, this tutorial is for you.

In this tutorial, we'll walk through how to use Studio for the first time to create a machine learning experiment. The experiment will test an analytical model that predicts the price of an automobile based on different variables such as make and technical specifications.

NOTE

This tutorial shows you the basics of how to drag-and-drop modules onto your experiment, connect them together, run the experiment, and look at the results. We're not going to discuss the general topic of machine learning or how to select and use the 100+ built-in algorithms and data manipulation modules included in Studio.

If you're new to machine learning, the video series [Data Science for Beginners](#) might be a good place to start. This video series is a great introduction to machine learning using everyday language and concepts.

If you're familiar with machine learning, but you're looking for more general information about Machine Learning Studio, and the machine learning algorithms it contains, here are some good resources:

- [What is Machine Learning Studio?](#) - This is a high-level overview of Studio.
- [Machine learning basics with algorithm examples](#) - This infographic is useful if you want to learn more about the different types of machine learning algorithms included with Machine Learning Studio.
- [Machine Learning Guide](#) - This guide covers similar information as the infographic above, but in an interactive format.
- [Machine learning algorithm cheat sheet](#) and [How to choose algorithms for Microsoft Azure Machine Learning](#) - This downloadable poster and accompanying article discuss the Studio algorithms in depth.
- [Machine Learning Studio: Algorithm and Module Help](#) - This is the complete reference for all Studio modules, including machine learning algorithms,

NOTE

[Try Azure Machine Learning for free](#)

No credit card or Azure subscription needed. [Get started now >](#)

How does Machine Learning Studio help?

Machine Learning Studio makes it easy to set up an experiment using drag-and-drop modules preprogrammed with predictive modeling techniques.

Using an interactive, visual workspace, you drag-and-drop **datasets** and analysis **modules** onto an interactive canvas. You connect them together to form an **experiment** that you run in Machine Learning Studio. You **create a model, train the model, and score and test the model**.

You can iterate on your model design, editing the experiment and running it until it gives you the results you're looking for. When your model is ready, you can publish it as a **web service** so that others can send it new data and get predictions in return.

Open Machine Learning Studio

To get started with Studio, go to <https://studio.azureml.net>. If you've signed into Machine Learning Studio before, click **Sign In**. Otherwise, click **Sign up here** and choose between free and paid options.

Sign in to Machine Learning Studio

Five steps to create an experiment

In this machine learning tutorial, you'll follow five basic steps to build an experiment in Machine Learning Studio to create, train, and score your model:

- **Create a model**
 - [Step 1: Get data](#)
 - [Step 2: Prepare the data](#)
 - [Step 3: Define features](#)
- **Train the model**
 - [Step 4: Choose and apply a learning algorithm](#)
- **Score and test the model**
 - [Step 5: Predict new automobile prices](#)

TIP

You can find a working copy of the following experiment in the [Cortana Intelligence Gallery](#). Go to [Your first data science experiment - Automobile price prediction](#) and click **Open in Studio** to download a copy of the experiment into your Machine Learning Studio workspace.

Step 1: Get data

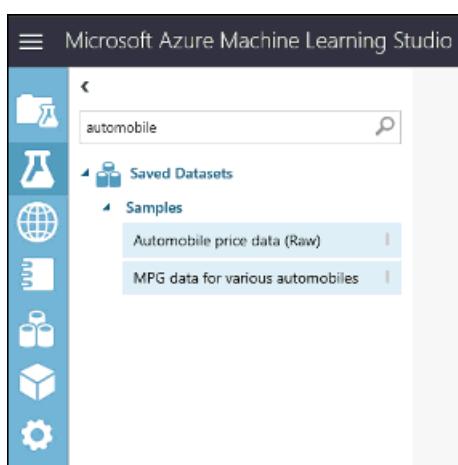
The first thing you need to perform machine learning is data. There are several sample datasets included with Machine Learning Studio that you can use, or you can import data from many sources. For this example, we'll use the sample dataset, **Automobile price data (Raw)**, that's included in your workspace. This dataset includes entries for various individual automobiles, including information such as make, model, technical specifications, and price.

Here's how to get the dataset into your experiment.

1. Create a new experiment by clicking **+NEW** at the bottom of the Machine Learning Studio window, select **EXPERIMENT**, and then select **Blank Experiment**.
2. The experiment is given a default name that you can see at the top of the canvas. Select this text and rename it to something meaningful, for example, **Automobile price prediction**. The name doesn't need to be unique.

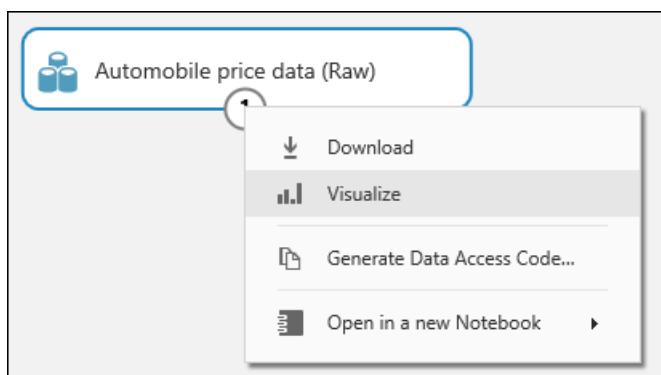


3. To the left of the experiment canvas is a palette of datasets and modules. Type **automobile** in the Search box at the top of this palette to find the dataset labeled **Automobile price data (Raw)**. Drag this dataset to the experiment canvas.



Find the automobile dataset and drag it onto the experiment canvas

To see what this data looks like, click the output port at the bottom of the automobile dataset, and then select **Visualize**.



Click the output port and select "Visualize"

TIP

Datasets and modules have input and output ports represented by small circles - input ports at the top, output ports at the bottom. To create a flow of data through your experiment, you'll connect an output port of one module to an input port of another. At any time, you can click the output port of a dataset or module to see what the data looks like at that point in the data flow.

In this sample dataset, each instance of an automobile appears as a row, and the variables associated with each automobile appear as columns. Given the variables for a specific automobile, we're going to try to predict the price in far-right column (column 26, titled "price").

Automobile price prediction ➤ Automobile price data (Raw) ➤ dataset											
rows	columns										
205	26										
symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	er	peak-rpm	city-mpg	highway-mpg	price
3		alfa-romero	gas	std	two	convertible		5000	21	27	13495
3		alfa-romero	gas	std	two	conven		5000	21	27	16500
1		alfa-romero	gas	std	two	hatch	54	5000	19	26	16500
2	164	audi	gas	std	four	sedan	102	5500	24	30	13950
2	164	audi	gas	std	four	sed	115	5500	18	22	17450
2		audi	gas	std	two	sp	110	5500	19	25	15250
1	158	audi	gas	std	four		110	5500	19	25	17710
1		audi	gas	std	four		110	5500	19	25	18920
1	158	audi	gas	turbo	four		140	5500	17	20	23875
0		audi	gas	turbo	two		160	5500	16	22	
2	192	bmw	gas	std	two		101	5800	23	29	16430
0	192	bmw	gas	std	four		101	5800	23	29	16925
0	188	bmw	gas	std	two		121	4250	21	28	20970
0	188	bmw	gas	std	fo		121	4250	21	28	21105
1		bmw	gas	std	fi		121	4250	20	25	24565

View the automobile data in the data visualization window

Close the visualization window by clicking the "x" in the upper-right corner.

Step 2: Prepare the data

A dataset usually requires some preprocessing before it can be analyzed. For example, you might have noticed the missing values present in the columns of various rows. These missing values need to be cleaned so the model can analyze the data correctly. In our case, we'll remove any rows that have missing values. Also, the **normalized-losses** column has a large proportion of missing values, so we'll exclude that column from the model altogether.

TIP

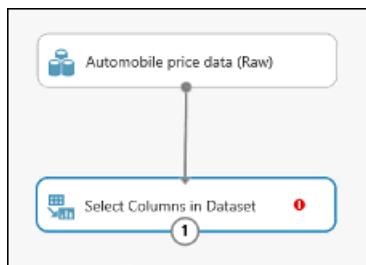
Cleaning the missing values from input data is a prerequisite for using most of the modules.

First we add a module that removes the **normalized-losses** column completely, and then we add another module that removes any row that has missing data.

1. Type **select columns** in the Search box at the top of the module palette to find the [Select Columns in](#)

[Dataset](#) module, then drag it to the experiment canvas. This module allows us to select which columns of data we want to include or exclude in the model.

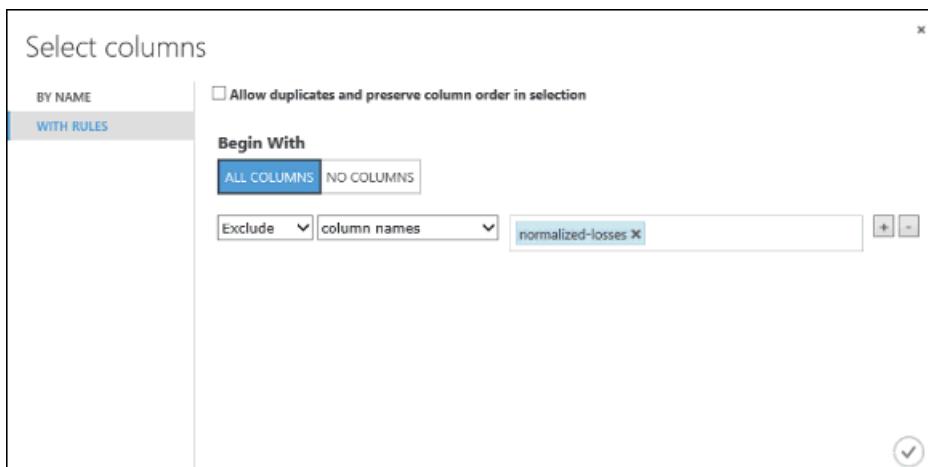
2. Connect the output port of the **Automobile price data (Raw)** dataset to the input port of the [Select Columns in Dataset](#) module.



Add the "Select Columns in Dataset" module to the experiment canvas and connect it

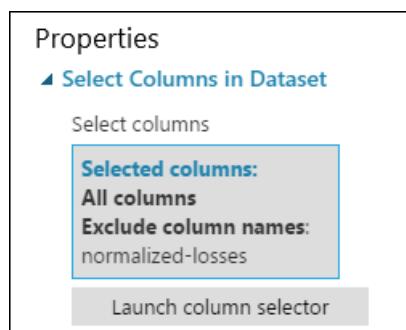
3. Click the [Select Columns in Dataset](#) module and click **Launch column selector** in the **Properties** pane.

- On the left, click **With rules**
- Under **Begin With**, click **All columns**. This directs [Select Columns in Dataset](#) to pass through all the columns (except those columns we're about to exclude).
- From the drop-downs, select **Exclude** and **column names**, and then click inside the text box. A list of columns is displayed. Select **normalized-losses**, and it's added to the text box.
- Click the check mark (OK) button to close the column selector (on the lower-right).



Launch the column selector and exclude the "normalized-losses" column

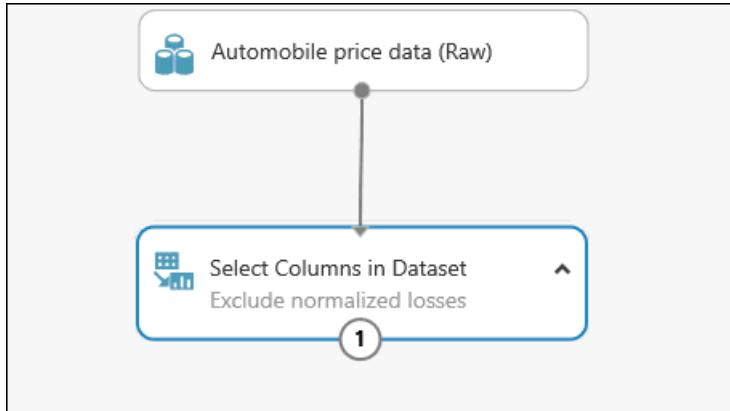
Now the properties pane for **Select Columns in Dataset** indicates that it will pass through all columns from the dataset except **normalized-losses**.



The properties pane shows that the "normalized-losses" column is excluded

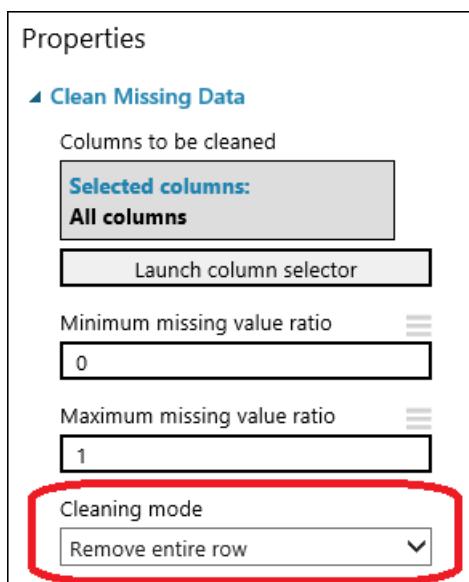
TIP

You can add a comment to a module by double-clicking the module and entering text. This can help you see at a glance what the module is doing in your experiment. In this case, double-click the [Select Columns in Dataset](#) module and type the comment "Exclude normalized losses."



Double-click a module to add a comment

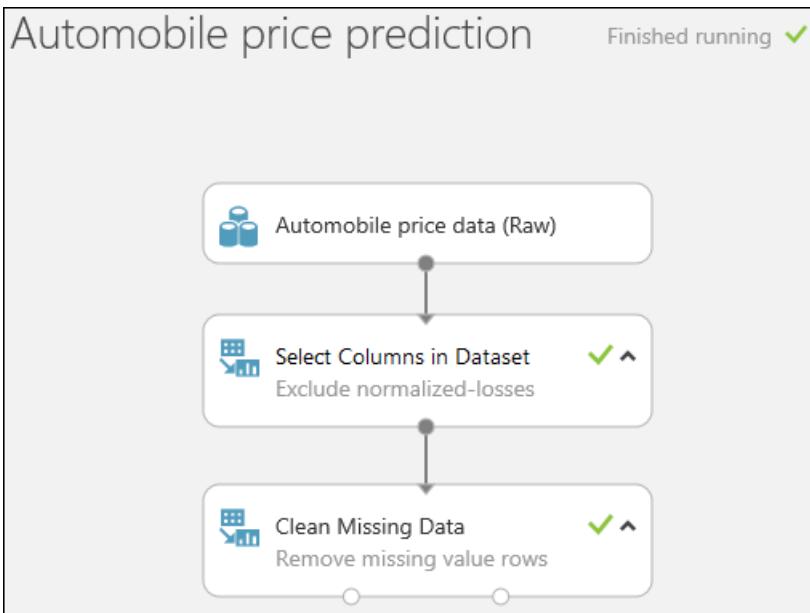
4. Drag the [Clean Missing Data](#) module to the experiment canvas and connect it to the [Select Columns in Dataset](#) module. In the **Properties** pane, select **Remove entire row** under **Cleaning mode**. This directs [Clean Missing Data](#) to clean the data by removing rows that have any missing values. Double-click the module and type the comment "Remove missing value rows."



Set the cleaning mode to "Remove entire row" for the "Clean Missing Data" module

5. Run the experiment by clicking **RUN** at the bottom of the page.

When the experiment has finished running, all the modules have a green check mark to indicate that they finished successfully. Notice also the **Finished running** status in the upper-right corner.



After running it, the experiment should look something like this

TIP

Why did we run the experiment now? By running the experiment, the column definitions for our data pass from the dataset, through the [Select Columns in Dataset](#) module, and through the [Clean Missing Data](#) module. This means that any modules we connect to [Clean Missing Data](#) will also have this same information.

All we have done in the experiment up to this point is clean the data. If you want to view the cleaned dataset, click the left output port of the [Clean Missing Data](#) module and select **Visualize**. Notice that the **normalized-losses** column is no longer included, and there are no missing values.

Now that the data is clean, we're ready to specify what features we're going to use in the predictive model.

Step 3: Define features

In machine learning, *features* are individual measurable properties of something you're interested in. In our dataset, each row represents one automobile, and each column is a feature of that automobile.

Finding a good set of features for creating a predictive model requires experimentation and knowledge about the problem you want to solve. Some features are better for predicting the target than others. Also, some features have a strong correlation with other features and can be removed. For example, city-mpg and highway-mpg are closely related so we can keep one and remove the other without significantly affecting the prediction.

Let's build a model that uses a subset of the features in our dataset. You can come back later and select different features, run the experiment again, and see if you get better results. But to start, let's try the following features:

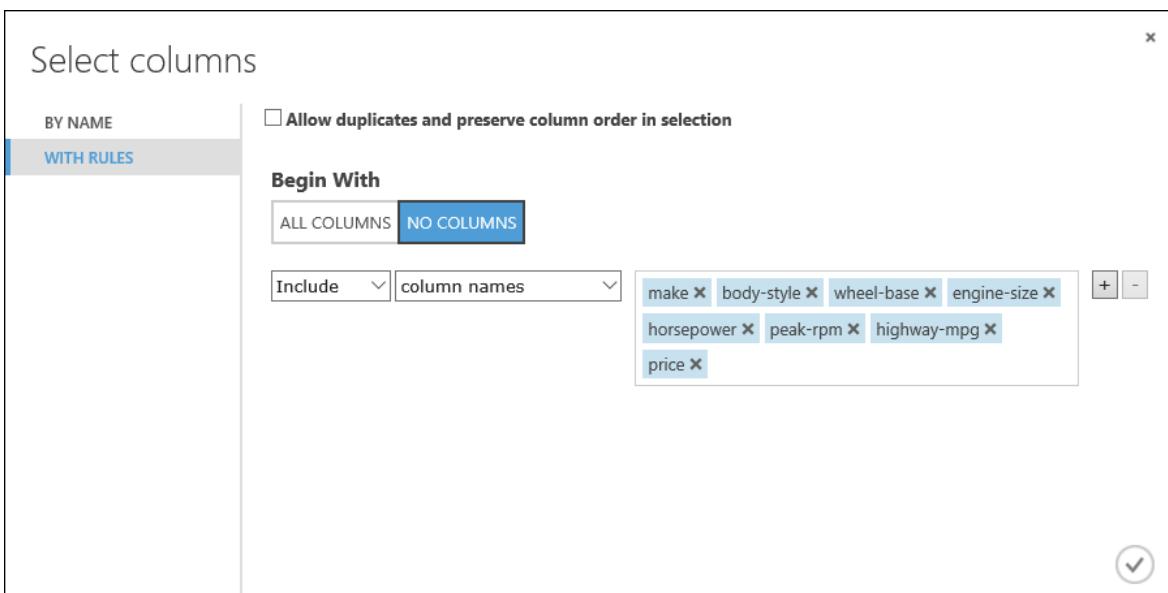
make, body-style, wheel-base, engine-size, horsepower, peak-rpm, highway-mpg, price

1. Drag another [Select Columns in Dataset](#) module to the experiment canvas. Connect the left output port of the [Clean Missing Data](#) module to the input of the [Select Columns in Dataset](#) module.



Connect the "Select Columns in Dataset" module to the "Clean Missing Data" module

2. Double-click the module and type "Select features for prediction."
3. Click **Launch column selector** in the **Properties** pane.
4. Click **With rules**.
5. Under **Begin With**, click **No columns**. In the filter row, select **Include** and **column names** and select our list of column names in the text box. This directs the module to not pass through any columns (features) except the ones that we specify.
6. Click the check mark (OK) button.



Select the columns (features) to include in the prediction

This produces a filtered dataset containing only the features we want to pass to the learning algorithm we'll use in the next step. Later, you can return and try again with a different selection of features.

Step 4: Choose and apply a learning algorithm

Now that the data is ready, constructing a predictive model consists of training and testing. We'll use our data to train the model, and then we'll test the model to see how closely it's able to predict prices.

Classification and *regression* are two types of supervised machine learning algorithms. Classification predicts an answer from a defined set of categories, such as a color (red, blue, or green). Regression is used to predict a number.

Because we want to predict price, which is a number, we'll use a regression algorithm. For this example, we'll use a simple *linear regression* model.

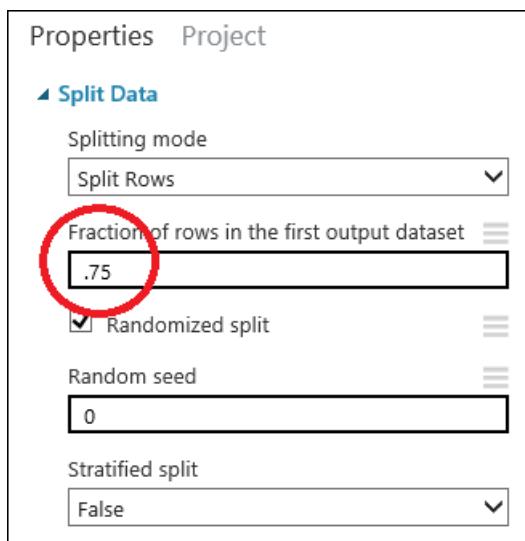
TIP

If you want to learn more about different types of machine learning algorithms and when to use them, you might view the first video in the Data Science for Beginners series, [The five questions data science answers](#). You might also look at the infographic [Machine learning basics with algorithm examples](#), or check out the [Machine learning algorithm cheat sheet](#).

We train the model by giving it a set of data that includes the price. The model scans the data and look for correlations between an automobile's features and its price. Then we'll test the model - we'll give it a set of features for automobiles we're familiar with and see how close the model comes to predicting the known price.

We'll use our data for both training the model and testing it by splitting the data into separate training and testing datasets.

1. Select and drag the [Split Data](#) module to the experiment canvas and connect it to the last [Select Columns in Dataset](#) module.
2. Click the [Split Data](#) module to select it. Find the **Fraction of rows in the first output dataset** (in the **Properties** pane to the right of the canvas) and set it to 0.75. This way, we'll use 75 percent of the data to train the model, and hold back 25 percent for testing (later, you can experiment with using different percentages).

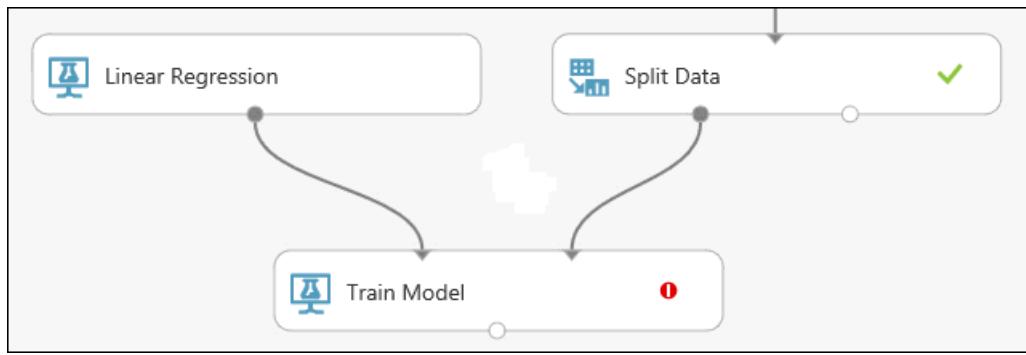


Set the split fraction of the "Split Data" module to 0.75

TIP

By changing the **Random seed** parameter, you can produce different random samples for training and testing. This parameter controls the seeding of the pseudo-random number generator.

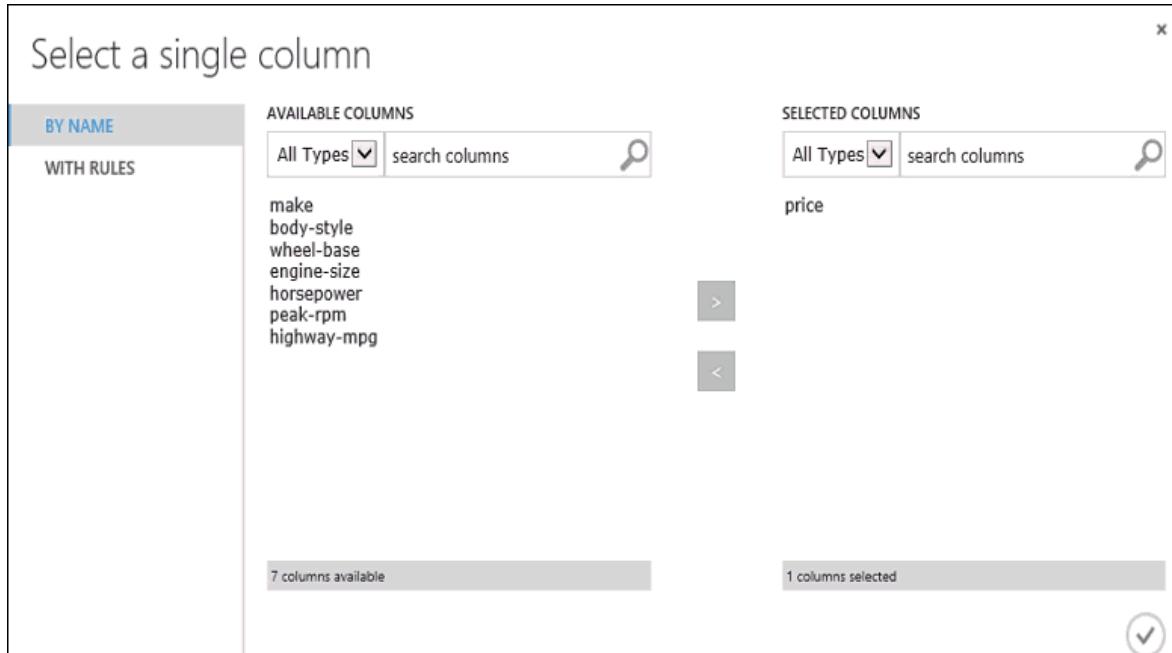
3. Run the experiment. When the experiment is run, the [Select Columns in Dataset](#) and [Split Data](#) modules pass column definitions to the modules we'll be adding next.
4. To select the learning algorithm, expand the **Machine Learning** category in the module palette to the left of the canvas, and then expand **Initialize Model**. This displays several categories of modules that can be used to initialize machine learning algorithms. For this experiment, select the [Linear Regression](#) module under the **Regression** category, and drag it to the experiment canvas. (You can also find the module by typing "linear regression" in the palette Search box.)
5. Find and drag the [Train Model](#) module to the experiment canvas. Connect the output of the [Linear Regression](#) module to the left input of the [Train Model](#) module, and connect the training data output (left port) of the [Split Data](#) module to the right input of the [Train Model](#) module.



Connect the "Train Model" module to both the "Linear Regression" and "Split Data" modules

6. Click the **Train Model** module, click **Launch column selector** in the **Properties** pane, and then select the **price** column. This is the value that our model is going to predict.

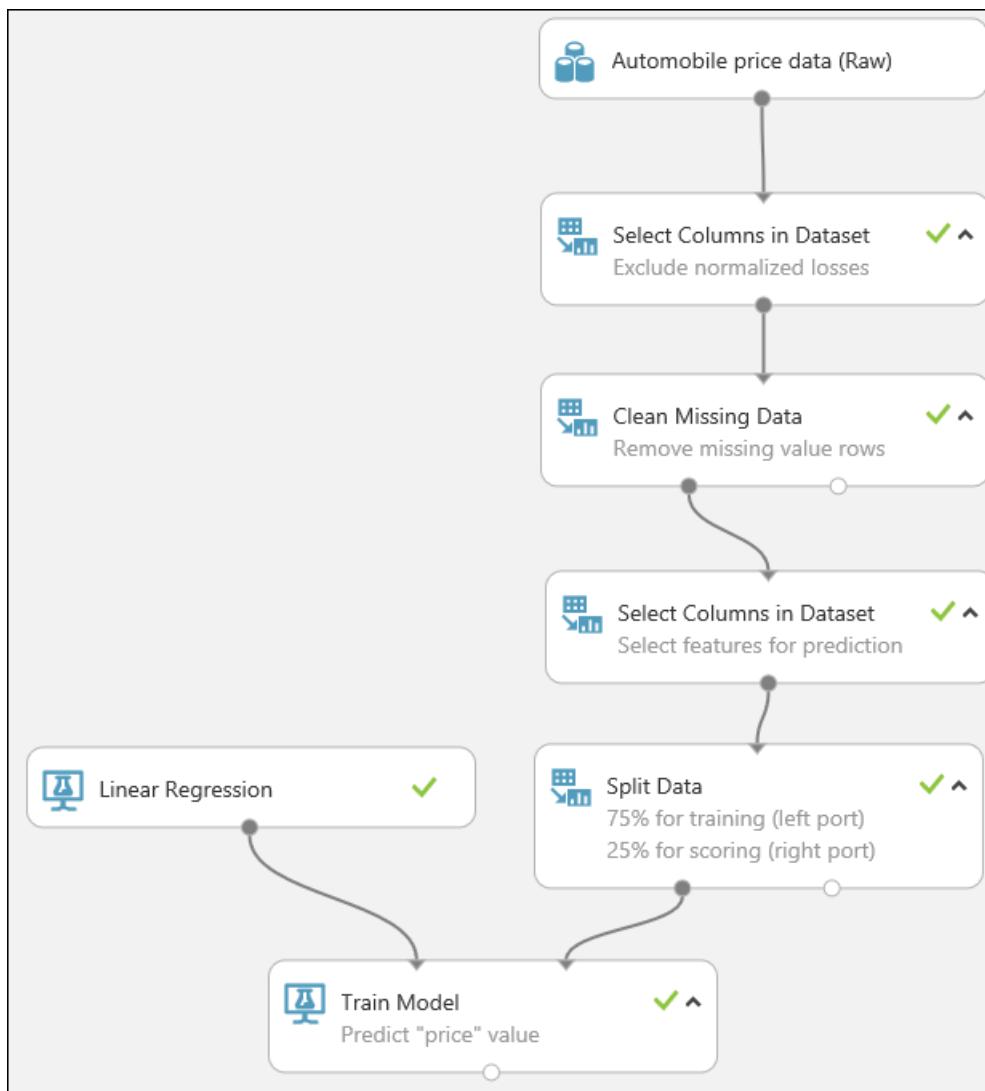
You select the **price** column in the column selector by moving it from the **Available columns** list to the **Selected columns** list.



Select the price column for the "Train Model" module

7. Run the experiment.

We now have a trained regression model that can be used to score new automobile data to make price predictions.

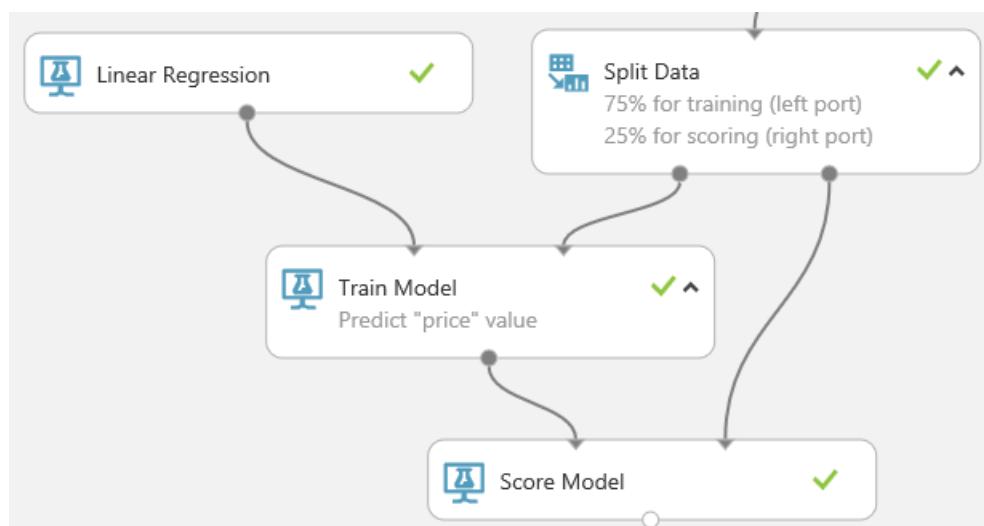


After running, the experiment should now look something like this

Step 5: Predict new automobile prices

Now that we've trained the model using 75 percent of our data, we can use it to score the other 25 percent of the data to see how well our model functions.

- Find and drag the [Score Model](#) module to the experiment canvas. Connect the output of the [Train Model](#) module to the left input port of [Score Model](#). Connect the test data output (right port) of the [Split Data](#) module to the right input port of [Score Model](#).



Connect the "Score Model" module to both the "Train Model" and "Split Data" modules

2. Run the experiment and view the output from the **Score Model** module (click the output port of **Score Model** and select **Visualize**). The output shows the predicted values for price and the known values from the test data.

Simple test experiment - Copy > Score Model > Scored dataset								
rows	columns							
48	9							
make	body-style	wheel-base	engine-size	horsepower	peak-rpm	highway-mpg	price	Scored Labels
subaru	sedan	97	108	111	4800	29	11259	10286.204819
mitsubishi	hatchback	93.7	92	68	5500	38	6669	5446.847864
dodge	hatchback	93.7	90	68	5500	38	6229	6344.800711
honda	hatchback	86.6	92	76	6000	38	6855	5528.302953
alfa-romero	convertible	88.6	130	111	5000	27	16500	13498.476233
volvo	wagon	104.3	141	114	5400	28	16515	16097.608038
isuzu	hatchback	96	119	90	5000	29	11048	8315.257218
dodge	hatchback	93.7	90	68	5500	41	5572	6630.154608
bmw	sedan	101.2	109	101	5800	29	16420	10012.409695

Output of the "Score Model" module

3. Finally, we test the quality of the results. Select and drag the **Evaluate Model** module to the experiment canvas, and connect the output of the **Score Model** module to the left input of **Evaluate Model**.

TIP

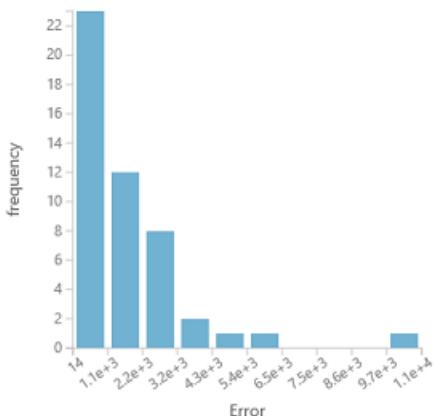
There are two input ports on the **Evaluate Model** module because it can be used to compare two models side by side. Later, you can add another algorithm to the experiment and use **Evaluate Model** to see which one gives better results.

4. Run the experiment.

To view the output from the **Evaluate Model** module, click the output port, and then select **Visualize**.

Metrics

Mean Absolute Error	1656.147651
Root Mean Squared Error	2456.983209
Relative Absolute Error	0.276606
Relative Squared Error	0.089608
Coefficient of Determination	0.910392

Error Histogram**Evaluation results for the experiment**

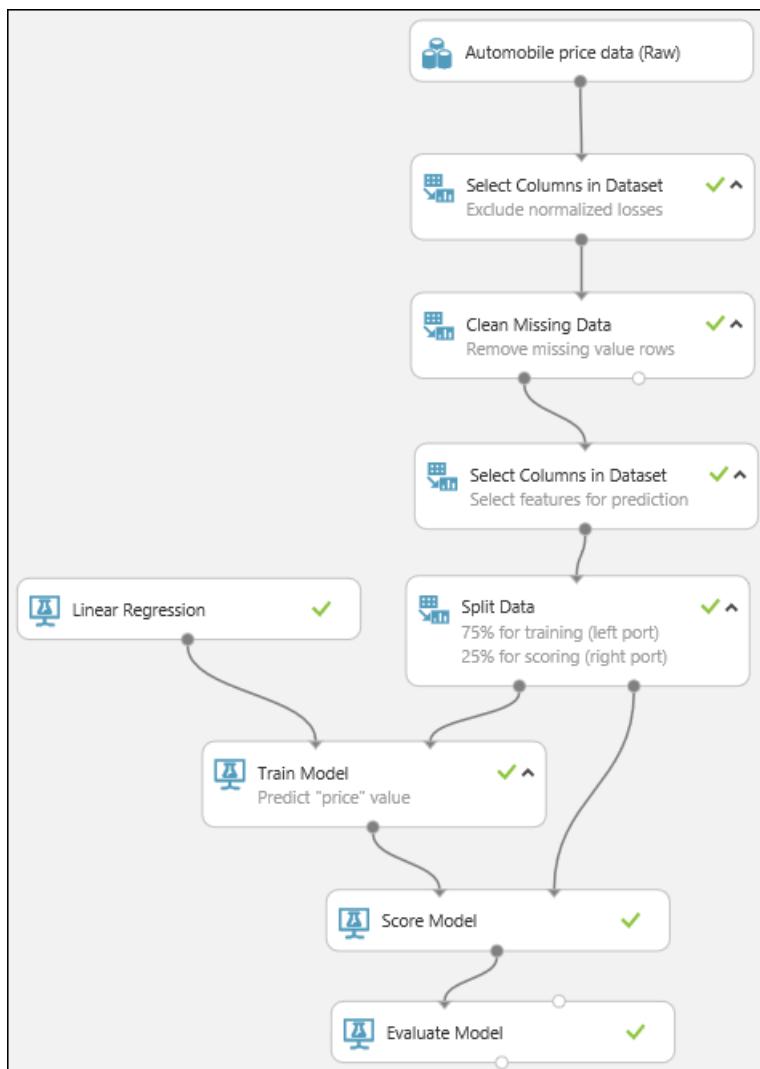
The following statistics are shown for our model:

- **Mean Absolute Error** (MAE): The average of absolute errors (an *error* is the difference between the predicted value and the actual value).
- **Root Mean Squared Error** (RMSE): The square root of the average of squared errors of predictions made on the test dataset.
- **Relative Absolute Error**: The average of absolute errors relative to the absolute difference between actual values and the average of all actual values.
- **Relative Squared Error**: The average of squared errors relative to the squared difference between the actual values and the average of all actual values.
- **Coefficient of Determination**: Also known as the **R squared value**, this is a statistical metric indicating how well a model fits the data.

For each of the error statistics, smaller is better. A smaller value indicates that the predictions more closely match the actual values. For **Coefficient of Determination**, the closer its value is to one (1.0), the better the predictions.

Final experiment

The final experiment should look something like this:



The final experiment

Next steps

Now that you've completed the first machine learning tutorial and have your experiment set up, you can continue to improve the model and then deploy it as a predictive web service.

- **Iterate to try to improve the model** - For example, you can change the features you use in your prediction. Or you can modify the properties of the [Linear Regression](#) algorithm or try a different algorithm altogether. You can even add multiple machine learning algorithms to your experiment at one time and compare two of them by using the [Evaluate Model](#) module. For an example of how to compare multiple models in a single experiment, see [Compare Regressors](#) in the Cortana Intelligence Gallery.

TIP

To copy any iteration of your experiment, use the **SAVE AS** button at the bottom of the page. You can see all the iterations of your experiment by clicking **VIEW RUN HISTORY** at the bottom of the page. For more details, see [Manage experiment iterations in Azure Machine Learning Studio](#).

- **Deploy the model as a predictive web service** - When you're satisfied with your model, you can deploy it as a web service to be used to predict automobile prices by using new data. For more details, see [Deploy an Azure Machine Learning web service](#).

Want to learn more? For a more extensive and detailed walkthrough of the process of creating, training, scoring, and deploying a model, see [Develop a predictive solution by using Azure Machine Learning](#).

Walkthrough: Develop a predictive analytics solution for credit risk assessment in Azure Machine Learning

1/17/2017 • 2 min to read • [Edit on GitHub](#)

In this walkthrough, we'll take an extended look at the process of developing a solution in Machine Learning Studio. We'll develop a predictive analytics model in Machine Learning Studio, and then deploy it as an Azure Machine Learning web service where the model can make predictions using new data.

TIP

This walkthrough assumes you've used Machine Learning Studio at least once before, and that you have some understanding of machine learning concepts, though it assumes you're not an expert in either.

If you've never used **Azure Machine Learning Studio** before, you might want to start with the tutorial, [Create your first data science experiment in Azure Machine Learning Studio](#). That tutorial takes you through Machine Learning Studio for the first time, showing you the basics of how to drag-and-drop modules onto your experiment, connect them together, run the experiment, and look at the results.

If you're new to machine learning, the video series [Data Science for Beginners](#) might be a good place to start. This video series is a great introduction to machine learning using everyday language and concepts.

The problem

Suppose you need to predict an individual's credit risk based on the information they give on a credit application.

Credit risk assessment is a complex problem, of course, but we'll simplify the parameters of the question a bit. Then, we'll use it as an example of how you can use Microsoft Azure Machine Learning with Machine Learning Studio and the Machine Learning web service to create such a predictive analytics solution.

The solution

In this detailed walkthrough, we'll start with publicly available credit risk data, develop and train a predictive model based on that data, and then deploy the model as a web service that can be used by others for credit risk assessment.

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

To create a credit risk assessment solution, we'll follow these steps:

1. [Create a Machine Learning workspace](#)
2. [Upload existing data](#)
3. [Create a new experiment](#)
4. [Train and evaluate the models](#)
5. [Deploy the web service](#)
6. [Access the web service](#)

This walkthrough is based on a simplified version of the [Binary Classification: Credit risk prediction](#) sample experiment in the [Cortana Intelligence Gallery](#).

TIP

To download and print a diagram that gives an overview of the capabilities of Machine Learning Studio, see [Overview diagram of Azure Machine Learning Studio capabilities](#).

Walkthrough Step 1: Create a Machine Learning workspace

1/17/2017 • 1 min to read • [Edit on GitHub](#)

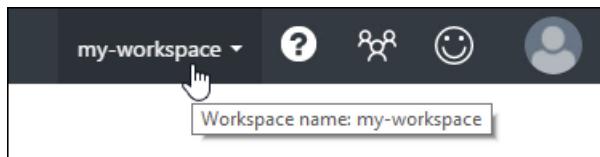
This is the first step of the walkthrough, [Develop a predictive analytics solution in Azure Machine Learning](#).

1. **Create a Machine Learning workspace**
2. [Upload existing data](#)
3. [Create a new experiment](#)
4. [Train and evaluate the models](#)
5. [Deploy the Web service](#)
6. [Access the Web service](#)

To use Machine Learning Studio, you need to have a Microsoft Azure Machine Learning workspace. This workspace contains the tools you need to create, manage, and publish experiments.

The administrator for your Azure subscription will need to create the workspace and then add you as an owner or contributor. For details, see [Create and share an Azure Machine Learning workspace](#).

After your workspace is created, open Machine Learning Studio (<https://studio.azureml.net>). If this is your only workspace, Studio will open it automatically. Otherwise, you can select the workspace in the toolbar in the upper-right corner of the window.



TIP

If you were made an owner of the workspace, you can share the experiments you're working on by inviting others to the workspace. You can do this in Machine Learning Studio on the **SETTINGS** page. You just need the Microsoft account or organizational account for each user.

On the **SETTINGS** page, click **USERS**, then click **INVITE MORE USERS** at the bottom of the window.

Next: [Upload existing data](#)

Walkthrough Step 2: Upload existing data into an Azure Machine Learning experiment

1/17/2017 • 3 min to read • [Edit on GitHub](#)

This is the second step of the walkthrough, [Develop a predictive analytics solution in Azure Machine Learning](#)

1. [Create a Machine Learning workspace](#)
2. **Upload existing data**
3. [Create a new experiment](#)
4. [Train and evaluate the models](#)
5. [Deploy the Web service](#)
6. [Access the Web service](#)

To develop a predictive model for credit risk, we need data that we can use to train and then test the model. For this walkthrough, we'll use the "UCI Statlog (German Credit Data) Data Set" from the UC Irvine Machine Learning repository. You can find it here:

[http://archive.ics.uci.edu/ml/datasets/Statlog+\(German+Credit+Data\)](http://archive.ics.uci.edu/ml/datasets/Statlog+(German+Credit+Data))

We'll use the file named **german.data**. Download this file to your local hard drive.

This dataset contains rows of 20 variables for 1000 past applicants for credit. These 20 variables represent the dataset's set of features (the feature vector), which provides identifying characteristics for each credit applicant. An additional column in each row represents the applicant's calculated credit risk, with 700 applicants identified as a low credit risk and 300 as a high risk.

The UCI website provides a description of the attributes of the feature vector for this data. This includes financial information, credit history, employment status, and personal information. For each applicant, a binary rating has been given indicating whether they are a low or high credit risk.

We'll use this data to train a predictive analytics model. When we're done, our model should be able to accept a feature vector for a new individual and predict whether he or she is a low or high credit risk.

Here's one interesting twist. The description of the dataset explains that misclassifying a person as a low credit risk when they are actually a high credit risk is 5 times more costly to the financial institution than misclassifying a low credit risk as high. One simple way to take this into account in our experiment is by duplicating (5 times) those entries that represent someone with a high credit risk. Then, if the model misclassifies that high credit risk as low, it will do that misclassification 5 times, once for each duplicate. This will increase the cost of this error in the training results.

Convert the dataset format

The original dataset uses a blank-separated format. Machine Learning Studio works better with a comma-separated value (CSV) file, so we'll convert the dataset by replacing spaces with commas.

There are many ways to convert this data. One way is by using the following Windows PowerShell command:

```
cat german.data | %{$_.Replace(" ",",")} | sc german.csv
```

Another way is by using the Unix sed command:

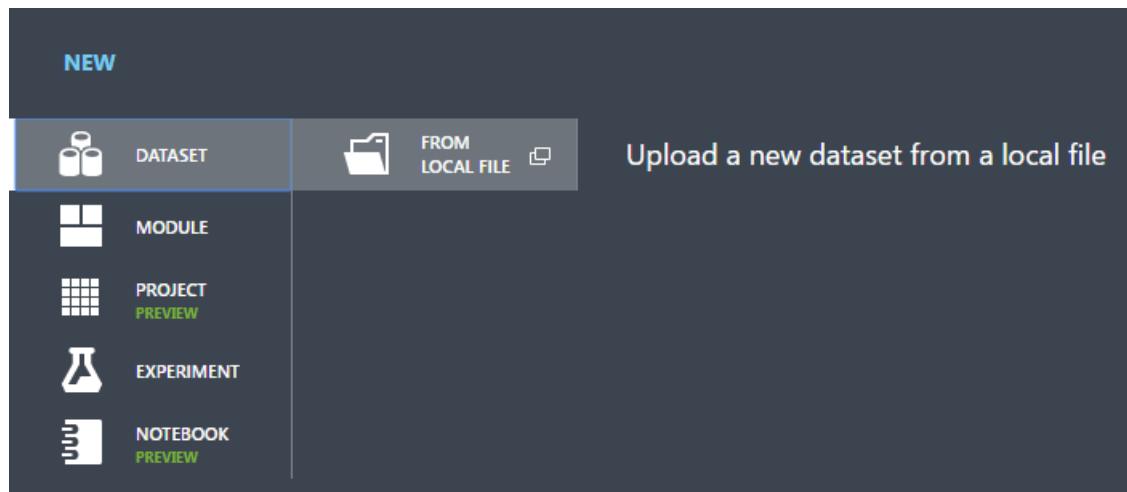
```
sed 's//g' german.data > german.csv
```

In either case, we have created a comma-separated version of the data in a file named **german.csv** that we'll use in our experiment.

Upload the dataset to Machine Learning Studio

Once the data has been converted to CSV format, we need to upload it into Machine Learning Studio.

1. Open the Machine Learning Studio home page (<https://studio.azureml.net>).
2. Click the menu  in the upper-left corner of the window, click **Azure Machine Learning**, select **Studio**, and sign in.
3. Click **+NEW** at the bottom of the window.
4. Select **DATASET**.
5. Select **FROM LOCAL FILE**.



6. In the **Upload a new dataset** dialog, click **Browse** and find the **german.csv** file you created.
7. Enter a name for the dataset. For this walkthrough, we'll call it "UCI German Credit Card Data".
8. For data type, select **Generic CSV File With no header (.nh.csv)**.
9. Add a description if you'd like.
10. Click the **OK** check mark.

Upload a new dataset

SELECT THE DATA TO UPLOAD:
 german.csv

This is the new version of an existing dataset

ENTER A NAME FOR THE NEW DATASET:
UCI German Credit Card Data

SELECT A TYPE FOR THE NEW DATASET:
Generic CSV File With no header (.nh.csv)

PROVIDE AN OPTIONAL DESCRIPTION:
Downloaded from
<http://archive.ics.uci.edu/ml/datasets>

(✓)

This uploads the data into a dataset module that we can use in an experiment.

You can manage datasets that you've uploaded to Studio by clicking the **DATASETS** tab to the left of the Studio window.

NAME	SUBMITTED BY	DESCRIPTION	DATA TYPE	CREATED	SIZE	PROJECT
UCI German Cred...	john	Downloaded fr...	GenericCSVNoHeader	12/15/2016 ...	78.9 KB	None

For more information about importing other types of data into an experiment, see [Import your training data into Azure Machine Learning Studio](#).

Next: Create a new experiment

Walkthrough Step 3: Create a new Azure Machine Learning experiment

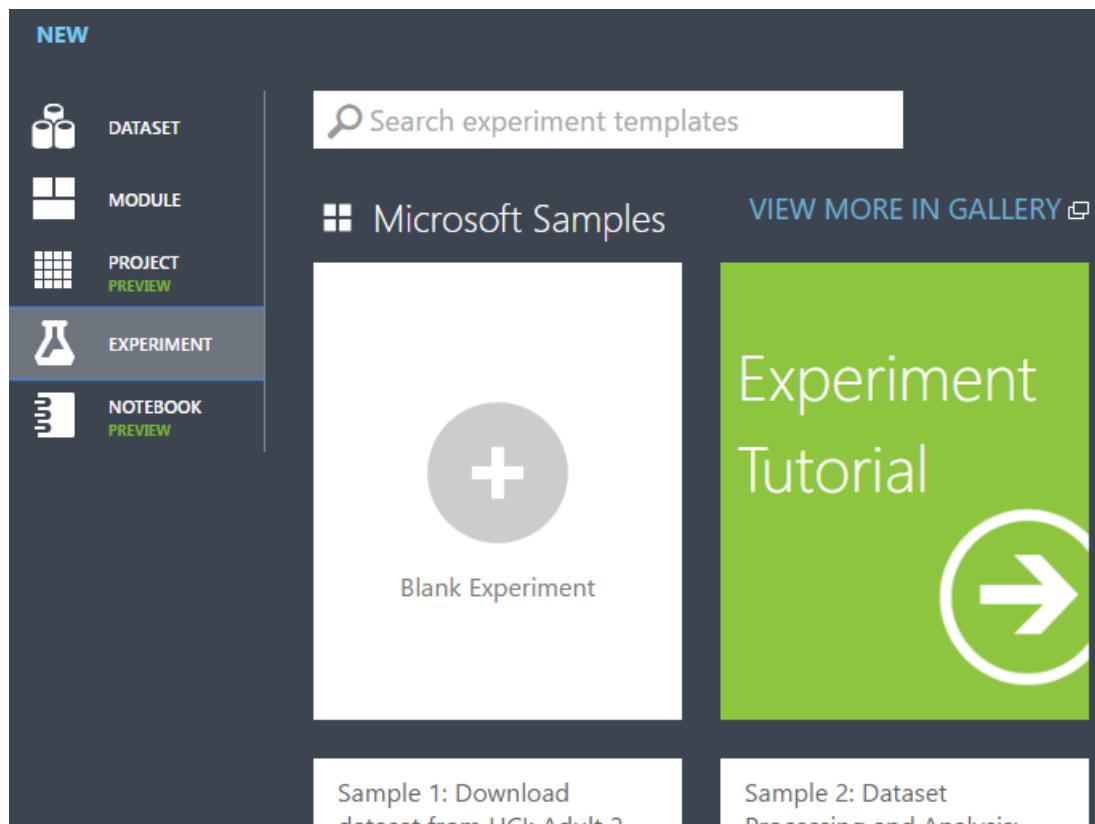
1/17/2017 • 6 min to read • [Edit on GitHub](#)

This is the third step of the walkthrough, [Develop a predictive analytics solution in Azure Machine Learning](#)

1. [Create a Machine Learning workspace](#)
2. [Upload existing data](#)
3. **Create a new experiment**
4. [Train and evaluate the models](#)
5. [Deploy the Web service](#)
6. [Access the Web service](#)

The next step in this walkthrough is to create an experiment in Machine Learning Studio that uses the dataset we uploaded.

1. In Studio, click **+NEW** at the bottom of the window.
2. Select **EXPERIMENT**, and then select "Blank Experiment".



3. Select the default experiment name at the top of the canvas and rename it to something meaningful.



TIP

It's a good practice to fill in **Summary** and **Description** for the experiment in the **Properties** pane. These properties give you the chance to document the experiment so that anyone who looks at it later will understand your goals and methodology.

Properties Project

Experiment Properties

STATUS CODE InDraft

Summary

Credit risk prediction from customer credit application data

Description

This experiment predicts credit risk based on information provided on a credit application. The training data was derived from the "UCI Statlog (German Credit Data) Data Set" from the UCI Machine Learning repository: <[http://archive.ics.uci.edu/ml/datasets/Statlog+\(German+Credit+Data\)](http://archive.ics.uci.edu/ml/datasets/Statlog+(German+Credit+Data))>.

4. In the module palette to the left of the experiment canvas, expand **Saved Datasets**.
5. Find the dataset you created under **My Datasets** and drag it onto the canvas. You can also find the dataset by entering the name in the **Search** box above the palette.

credit risk experiment

Search experiment items

Saved Datasets

My Datasets

UCI German Credit Card Data

Samples

Data Format Conversions

Prepare the data

You can view the first 100 rows of the data and some statistical information for the whole dataset: Click the

output port of the dataset (the small circle at the bottom) and select **Visualize**.

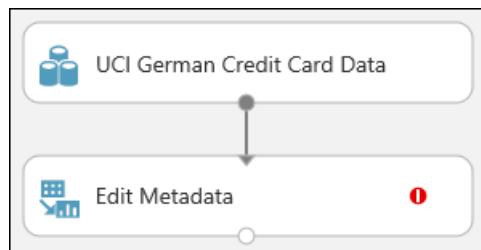
Because the data file didn't come with column headings, Studio has provided generic headings (Col1, Col2, etc.). Good headings aren't essential to creating a model, but they make it easier to work with the data in the experiment. Also, when we eventually publish this model in a web service, the headings will help identify the columns to the user of the service.

We can add column headings using the [Edit Metadata](#) module. You use the [Edit Metadata](#) module to change metadata associated with a dataset. In this case, we'll use it to provide more friendly names for column headings.

To use [Edit Metadata](#), you first specify which columns to modify (in this case, all of them.) Next, you specify the action to be performed on those columns (in this case, changing column headings.)

1. In the module palette, type "metadata" in the **Search** box. The [Edit Metadata](#) appears in the module list.
2. Click and drag the [Edit Metadata](#) module onto the canvas and drop it below the dataset we added earlier.
3. Connect the dataset to the [Edit Metadata](#): click the output port of the dataset (the small circle at the bottom of the dataset), drag to the input port of [Edit Metadata](#) (the small circle at the top of the module), then release the mouse button. The dataset and module remain connected even if you move either around on the canvas.

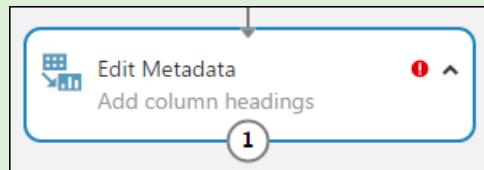
The experiment should now look something like this:



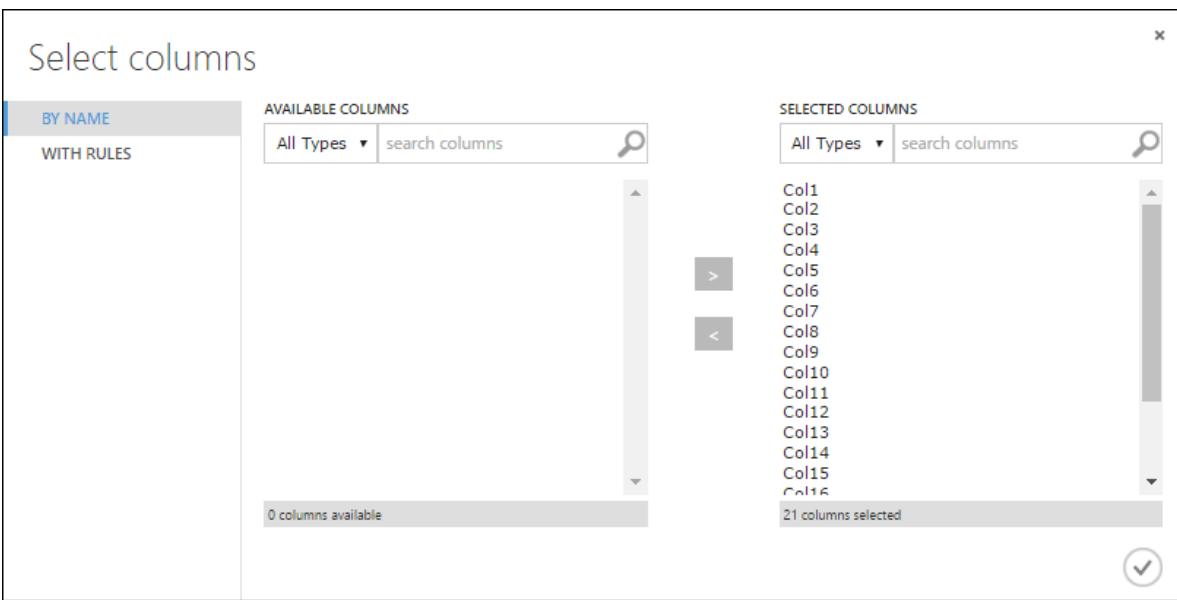
The red exclamation mark indicates that we haven't set the properties for this module yet. We'll do that next.

TIP

You can add a comment to a module by double-clicking the module and entering text. This can help you see at a glance what the module is doing in your experiment. In this case, double-click the [Edit Metadata](#) module and type the comment "Add column headings". Click anywhere else on the canvas to close the text box. To display the comment, click the down-arrow on the module.



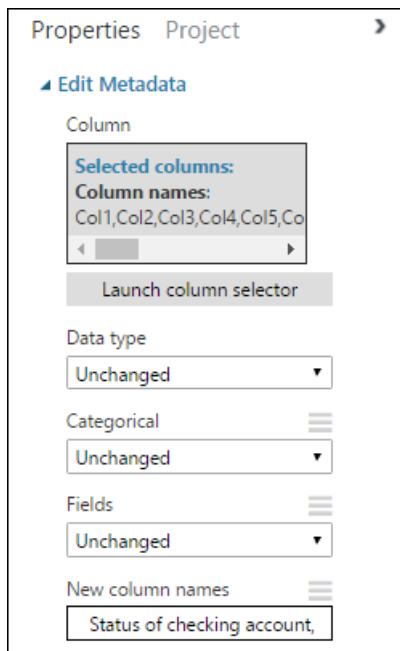
4. Select [Edit Metadata](#), and in the **Properties** pane to the right of the canvas, click **Launch column selector**.
5. In the **Select columns** dialog, select all the rows in **Available Columns** and click **>** to move them to **Selected Columns**. The dialog should look like this:



6. Click the **OK** check mark.
7. Back in the **Properties** pane, look for the **New column names** parameter. In this field, enter a list of names for the 21 columns in the dataset, separated by commas and in column order. You can obtain the columns names from the dataset documentation on the UCI website, or for convenience you can copy and paste the following list:

Status of checking account, Duration in months, Credit history, Purpose, Credit amount, Savings account/bond, Present employment since, Installment rate in percentage of disposable income, Personal status and sex, Other debtors, Present residence since, Property, Age in years, Other installment plans, Housing, Number of existing credits, Job, Number of people providing maintenance for, Telephone, Foreign worker, Credit risk

The Properties pane looks like this:



TIP

If you want to verify the column headings, run the experiment (click **RUN** below the experiment canvas). When it finishes running (a green check mark appears on **Edit Metadata**), click the output port of the **Edit Metadata** module, and select **Visualize**. You can view the output of any module in the same way to view the progress of the data through the experiment.

Create training and test datasets

The next step of the experiment is to split the dataset into two separate datasets. We'll use one for training our model and one for testing it.

To do this, we use the [Split Data](#) module.

1. Find the [Split Data](#) module, drag it onto the canvas, and connect it to the [Edit Metadata](#) module.
2. By default, the split ratio is 0.5 and the **Randomized split** parameter is set. This means that a random half of the data is output through one port of the [Split Data](#) module, and half through the other. You can adjust these parameters, as well as the **Random seed** parameter, to change the split between training and testing data. For this example, we leave them as-is.

TIP

The property **Fraction of rows in the first output dataset** determines how much of the data is output through the left output port. For instance, if you set the ratio to 0.7, then 70% of the data is output through the left port and 30% through the right port.

3. Double-click the [Split Data](#) module and enter the comment, "Training/testing data split 50%".

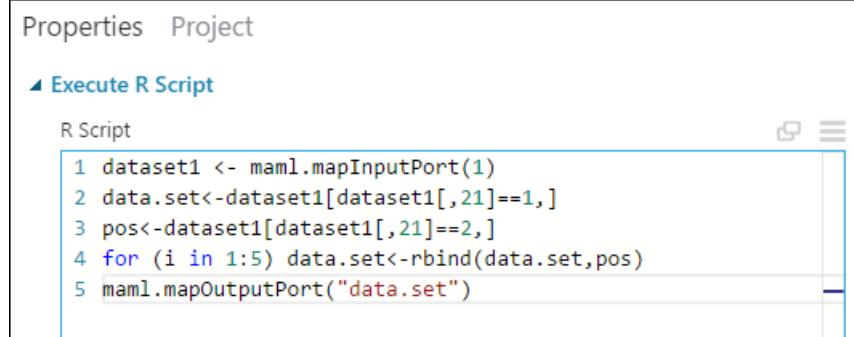
We can use the outputs of the [Split Data](#) module however we like, but let's choose to use the left output as training data and the right output as testing data.

As mentioned earlier, the cost of misclassifying a high credit risk as low is five times larger than the cost of misclassifying a low credit risk as high. To account for this, we generate a new dataset that reflects this cost function. In the new dataset, each high risk example is replicated five times, while each low risk example is not replicated.

We can do this replication using R code:

1. Find and drag the [Execute R Script](#) module onto the experiment canvas.
2. Connect the left output port of the [Split Data](#) module to the first input port ("Dataset1") of the [Execute R Script](#) module.
3. Double-click the [Execute R Script](#) module and enter the comment, "Set cost adjustment".
4. In the **Properties** pane, delete the default text in the **R Script** parameter and enter this script:

```
dataset1 <- maml.mapInputPort(1)
data.set<-dataset1[dataset1[,21]==1,]
pos<-dataset1[dataset1[,21]==2,]
for (i in 1:5) data.set<-rbind(data.set,pos)
maml.mapOutputPort("data.set")
```



We need to do this same replication operation for each output of the [Split Data](#) module so that the training and

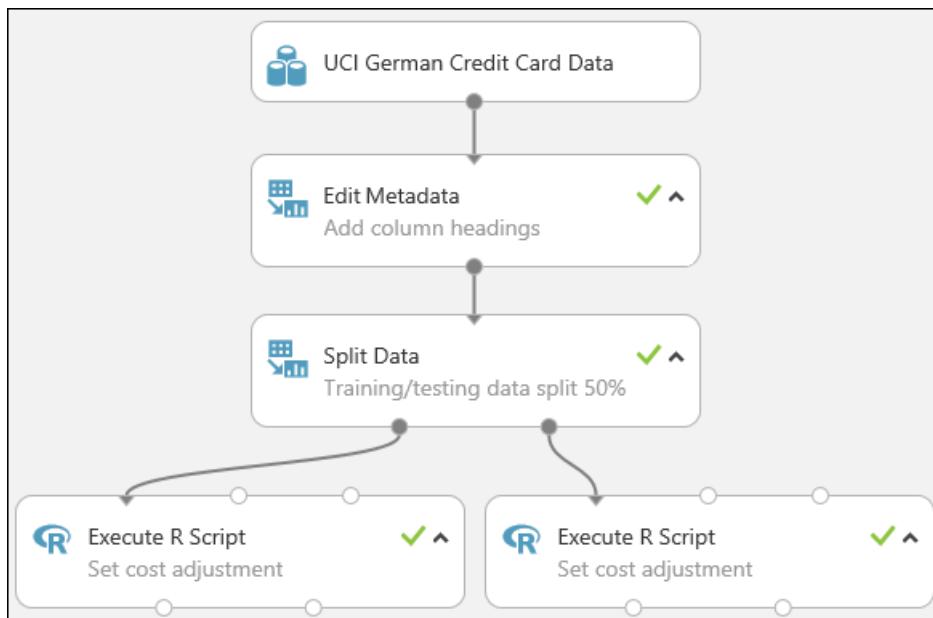
testing data have the same cost adjustment. We'll do this by duplicating the [Execute R Script](#) module we just made and connecting it to the other output port of the [Split Data](#) module.

1. Right-click the [Execute R Script](#) module and select **Copy**.
2. Right-click the experiment canvas and select **Paste**.
3. Drag the new module into position, and then connect the right output port of the [Split Data](#) module to the first input port of this new [Execute R Script](#) module.
4. At the bottom of the canvas, click **Run**.

TIP

The copy of the Execute R Script module contains the same script as the original module. When you copy and paste a module on the canvas, the copy retains all the properties of the original.

Our experiment now looks something like this:



For more information on using R scripts in your experiments, see [Extend your experiment with R](#).

Next: Train and evaluate the models

Walkthrough Step 4: Train and evaluate the predictive analytic models

1/17/2017 • 8 min to read • [Edit on GitHub](#)

This topic contains the fourth step of the walkthrough, [Develop a predictive analytics solution in Azure Machine Learning](#)

1. [Create a Machine Learning workspace](#)
2. [Upload existing data](#)
3. [Create a new experiment](#)
4. **Train and evaluate the models**
5. [Deploy the Web service](#)
6. [Access the Web service](#)

One of the benefits of using Azure Machine Learning Studio for creating machine learning models is the ability to try more than one type of model at a time in a single experiment and compare the results. This type of experimentation helps you find the best solution for your problem.

In the experiment we're developing in this walkthrough, we'll create two different types of models and then compare their scoring results to decide which algorithm we want to use in our final experiment.

There are various models we could choose from. To see the models available, expand the **Machine Learning** node in the module palette, and then expand **Initialize Model** and the nodes beneath it. For the purposes of this experiment, we'll select the [Two-Class Support Vector Machine](#) (SVM) and the [Two-Class Boosted Decision Tree](#) modules.

TIP

To get help deciding which Machine Learning algorithm best suits the particular problem you're trying to solve, see [How to choose algorithms for Microsoft Azure Machine Learning](#).

Train the models

We'll add both the [Two-Class Boosted Decision Tree](#) module and [Two-Class Support Vector Machine](#) module in this experiment.

Two-Class Boosted Decision Tree

First, let's set up the boosted decision tree model.

1. Find the [Two-Class Boosted Decision Tree](#) module in the module palette and drag it onto the canvas.
2. Find the [Train Model](#) module, drag it onto the canvas, and then connect the output of the [Two-Class Boosted Decision Tree](#) module to the left input port of the [Train Model](#) module.

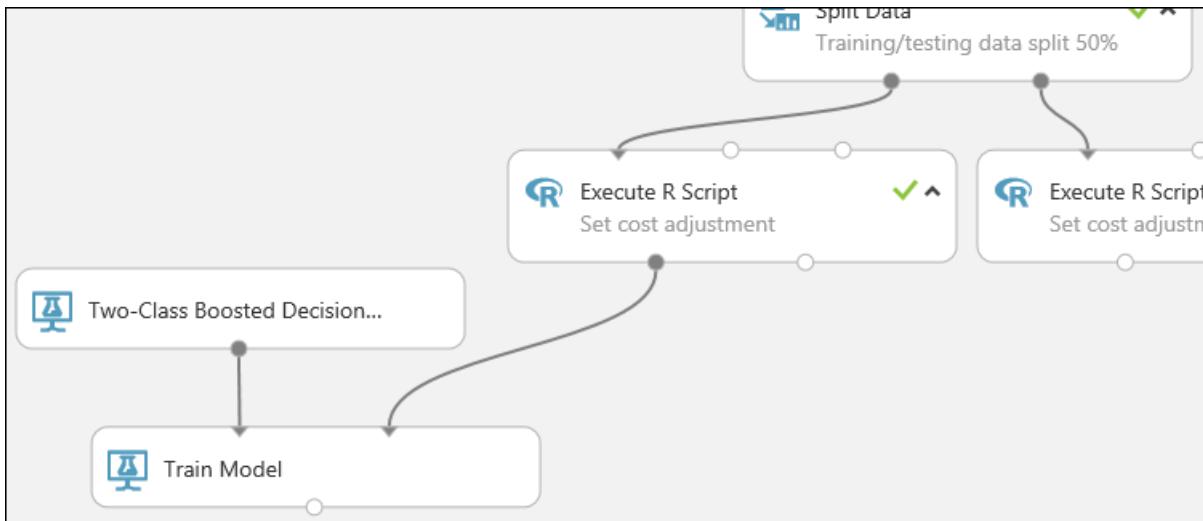
The [Two-Class Boosted Decision Tree](#) module initializes the generic model, and [Train Model](#) uses training data to train the model.

3. Connect the left output of the left [Execute R Script](#) module to the right input port of the [Train Model](#) module (we decided in [Step 3](#) of this walkthrough to use the data coming from the left side of the Split Data module for training).

TIP

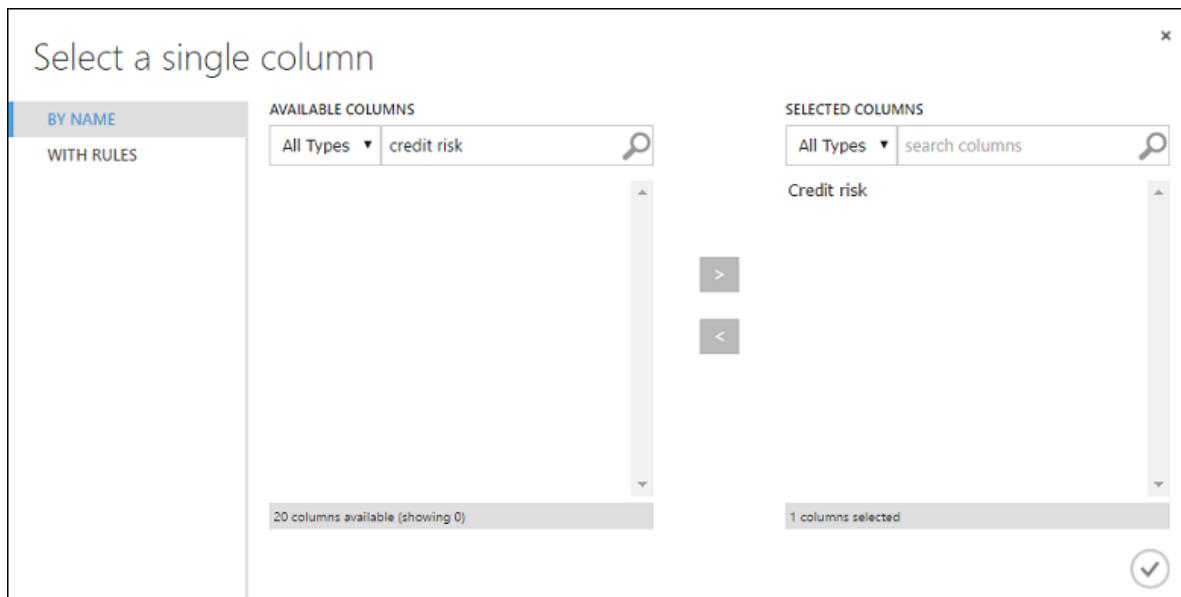
We don't need two of the inputs and one of the outputs of the [Execute R Script](#) module for this experiment, so we can leave them unattached.

This portion of the experiment now looks something like this:



Now we need to tell the [Train Model](#) module that we want the model to predict the Credit Risk value.

1. Select the [Train Model](#) module. In the **Properties** pane, click **Launch column selector**.
2. In the **Select a single column** dialog, type "credit risk" in the search field under **Available Columns**, select "Credit risk" below, and click the right arrow button (>) to move "Credit risk" to **Selected Columns**.



3. Click the **OK** check mark.

Two-Class Support Vector Machine

Next, we set up the SVM model.

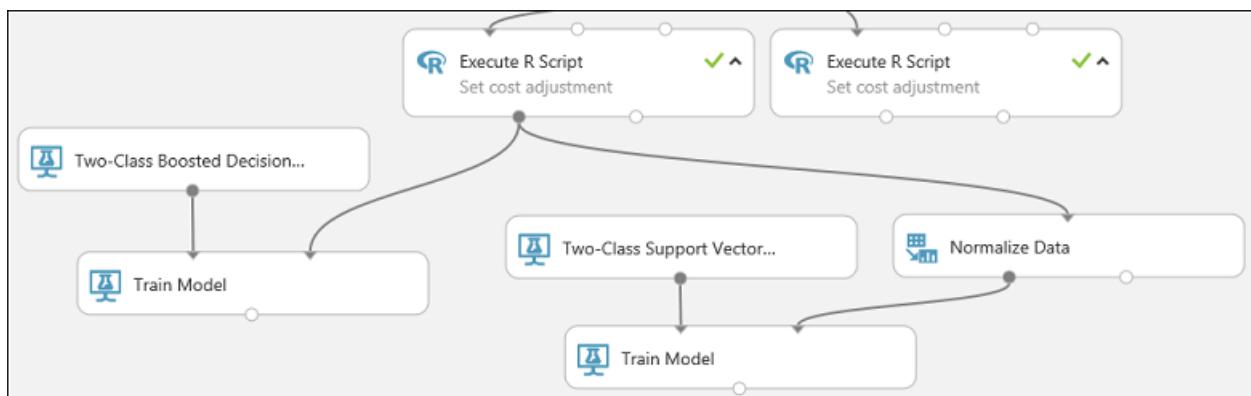
First, a little explanation about SVM. Boosted decision trees work well with features of any type. However, since the SVM module generates a linear classifier, the model that it generates has the best test error when all numeric features have the same scale. To convert all numeric features to the same scale, we use a "Tanh" transformation (with the [Normalize Data](#) module). This transforms our numbers into the [0,1] range. The SVM module converts string features to categorical features and then to binary 0/1 features, so we don't need to manually transform

string features. Also, we don't want to transform the Credit Risk column (column 21) - it's numeric, but it's the value we're training the model to predict, so we need to leave it alone.

To set up the SVM model, do the following:

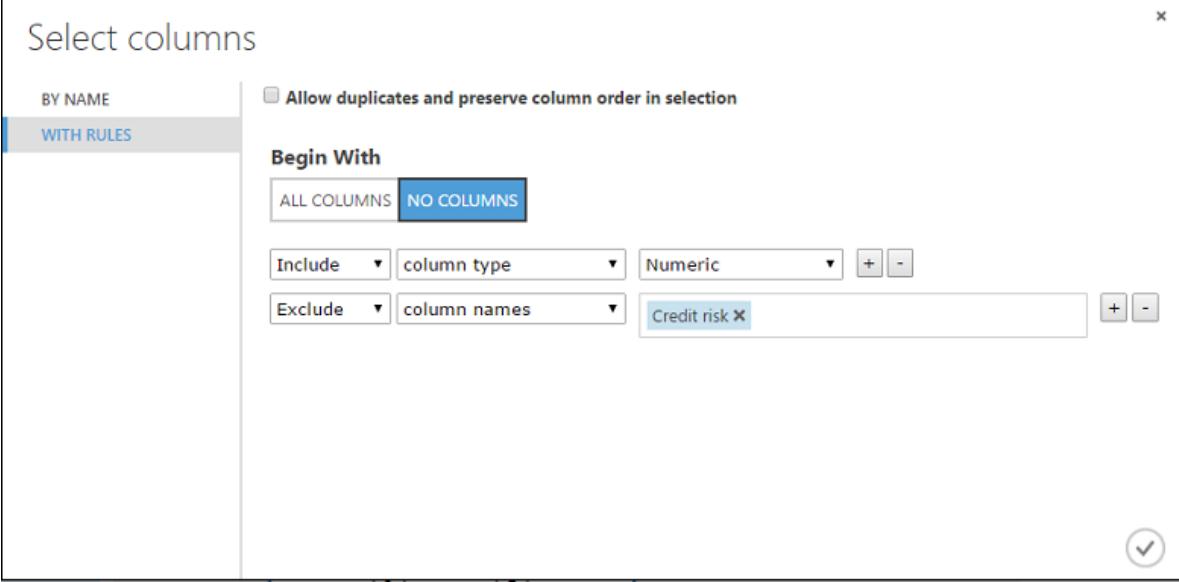
1. Find the [Two-Class Support Vector Machine](#) module in the module palette and drag it onto the canvas.
2. Right-click the [Train Model](#) module, select **Copy**, and then right-click the canvas and select **Paste**. The copy of the [Train Model](#) module has the same column selection as the original.
3. Connect the output of the [Two-Class Support Vector Machine](#) module to the left input port of the second [Train Model](#) module.
4. Find the [Normalize Data](#) module and drag it onto the canvas.
5. Connect the left output of the left [Execute R Script](#) module to the input of this module (notice that the output port of a module may be connected to more than one other module).
6. Connect the left output port of the [Normalize Data](#) module to the right input port of the second [Train Model](#) module.

This portion of our experiment should now look something like this:



Now configure the [Normalize Data](#) module:

1. Click to select the [Normalize Data](#) module. In the **Properties** pane, select **Tanh** for the **Transformation method** parameter.
2. Click **Launch column selector**, select "No columns" for **Begin With**, select **Include** in the first dropdown, select **column type** in the second dropdown, and select **Numeric** in the third dropdown. This specifies that all the numeric columns (and only numeric) are transformed.
3. Click the plus sign (+) to the right of this row - this creates a row of dropdowns. Select **Exclude** in the first dropdown, select **column names** in the second dropdown, and enter "Credit risk" in the text field. This specifies that the Credit Risk column should be ignored (we need to do this because this column is numeric and so would be transformed if we didn't exclude it).
4. Click the **OK** check mark.



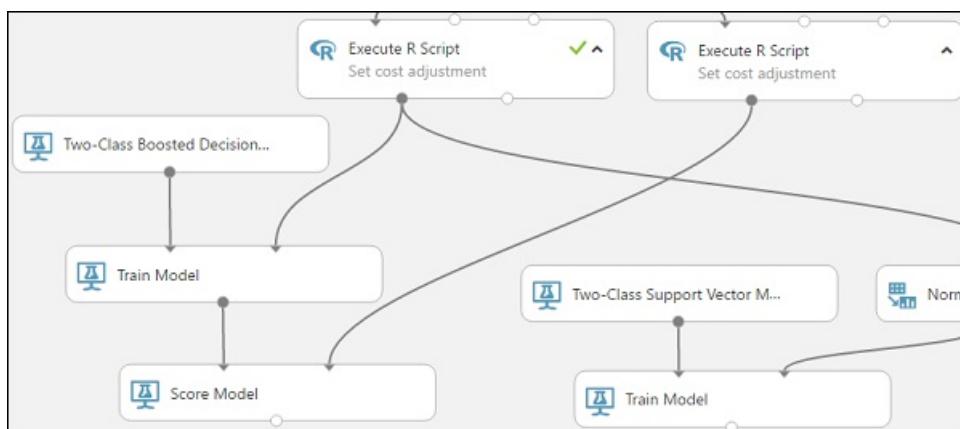
The [Normalize Data](#) module is now set to perform a Tanh transformation on all numeric columns except for the Credit Risk column.

Score and evaluate the models

We use the testing data that was separated out by the [Split Data](#) module to score our trained models. We can then compare the results of the two models to see which generated better results.

Add the Score Model modules

1. Find the [Score Model](#) module and drag it onto the canvas.
2. Connect the [Train Model](#) module that's connected to the [Two-Class Boosted Decision Tree](#) module to the left input port of the [Score Model](#) module.
3. Connect the right [Execute R Script](#) module (our testing data) to the right input port of the [Score Model](#) module.

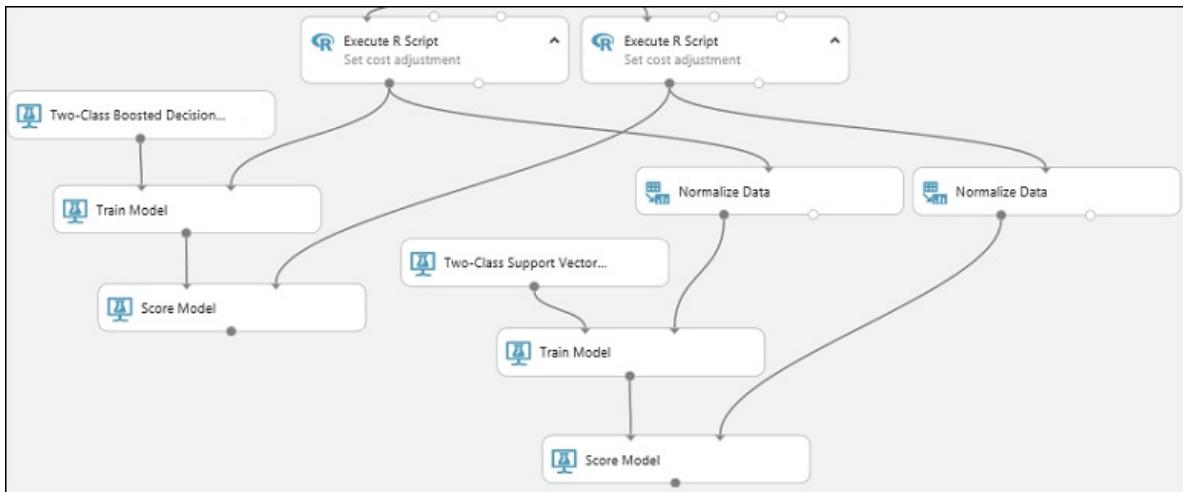


The [Score Model](#) module can now take the credit information from the testing data, run it through the model, and compare the predictions the model generates with the actual credit risk column in the testing data.

4. Copy and paste the [Score Model](#) module to create a second copy.
5. Connect the output of the SVM model (that is, the output port of the [Train Model](#) module that's connected to the [Two-Class Support Vector Machine](#) module) to the input port of the second [Score Model](#) module.
6. For the SVM model, we have to do the same transformation to the test data as we did to the training data. So copy and paste the [Normalize Data](#) module to create a second copy and connect it to the right [Execute R](#)

Script module.

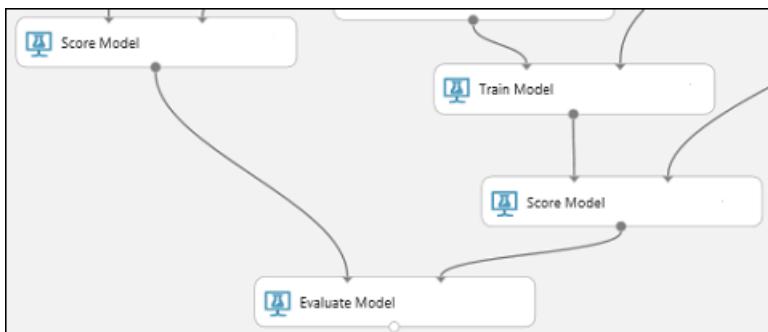
7. Connect the left output of the second Normalize Data module to the right input port of the second Score Model module.



Add the Evaluate Model module

To evaluate the two scoring results and compare them, we use an [Evaluate Model](#) module.

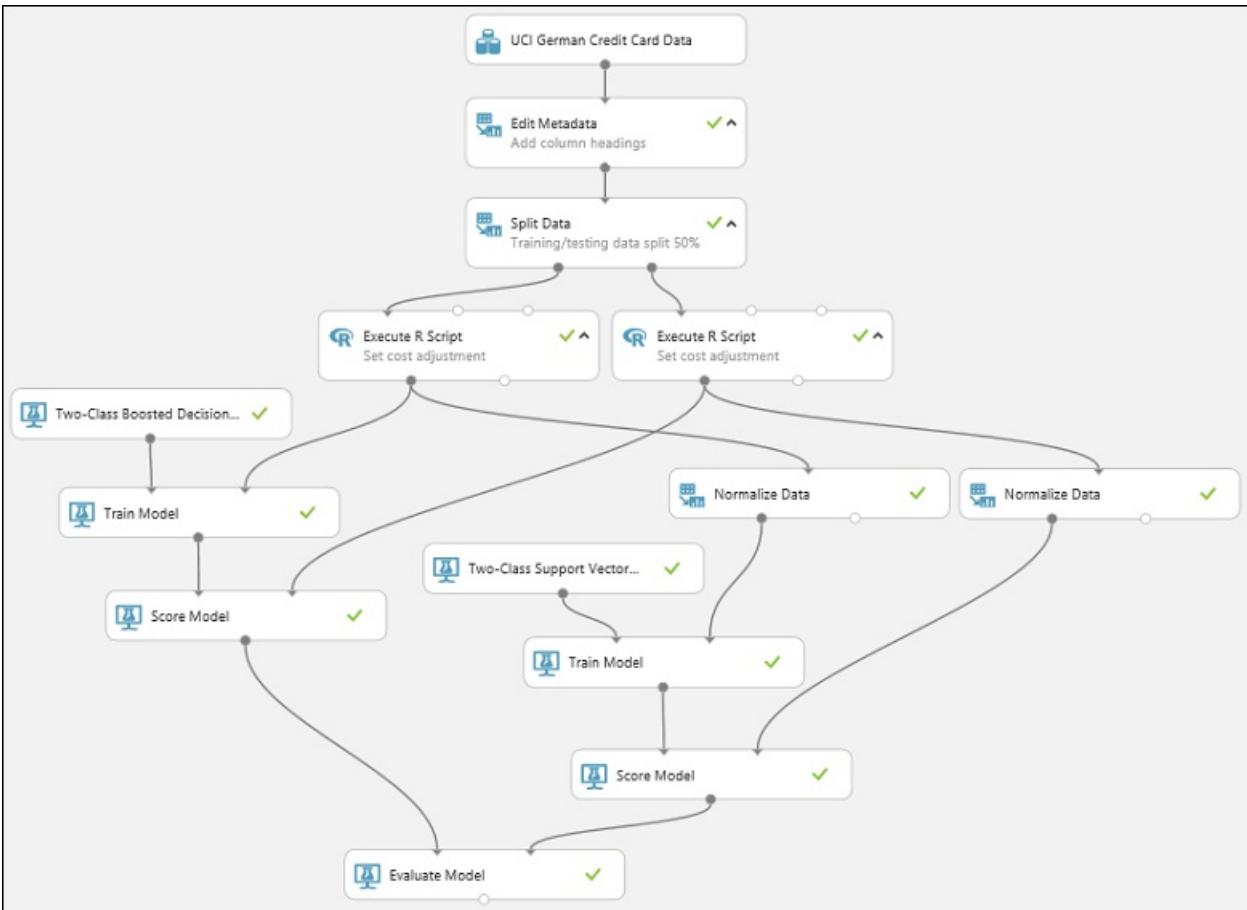
1. Find the [Evaluate Model](#) module and drag it onto the canvas.
2. Connect the output port of the [Score Model](#) module associated with the boosted decision tree model to the left input port of the [Evaluate Model](#) module.
3. Connect the other [Score Model](#) module to the right input port.



Run the experiment and check the results

To run the experiment, click the **RUN** button below the canvas. It may take a few minutes. A spinning indicator on each module shows that it's running, and then a green check mark shows when the module is finished. When all the modules have a check mark, the experiment has finished running.

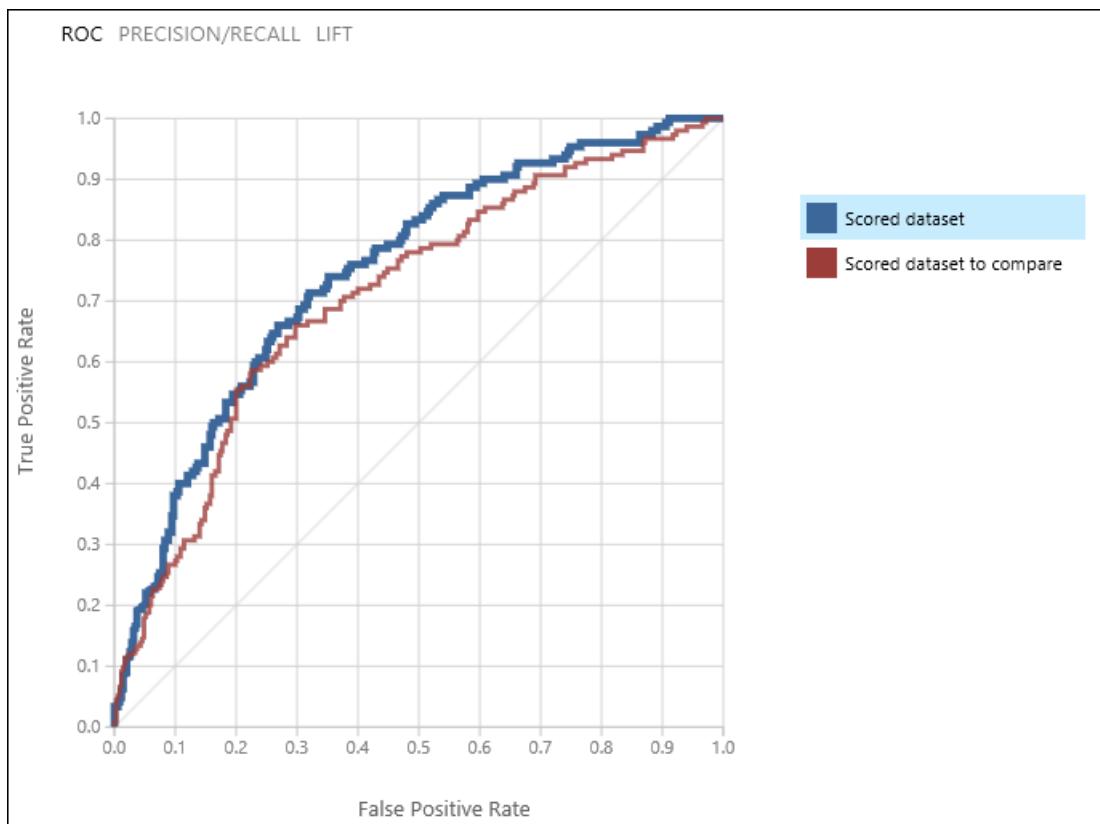
The experiment should now look something like this:



To check the results, click the output port of the [Evaluate Model](#) module and select **Visualize**.

The [Evaluate Model](#) module produces a pair of curves and metrics that allow you to compare the results of the two scored models. You can view the results as Receiver Operator Characteristic (ROC) curves, Precision/Recall curves, or Lift curves. Additional data displayed includes a confusion matrix, cumulative values for the area under the curve (AUC), and other metrics. You can change the threshold value by moving the slider left or right and see how it affects the set of metrics.

To the right of the graph, click **Scored dataset** or **Scored dataset to compare** to highlight the associated curve and to display the associated metrics below. In the legend for the curves, "Scored dataset" corresponds to the left input port of the [Evaluate Model](#) module - in our case, this is the boosted decision tree model. "Scored dataset to compare" corresponds to the right input port - the SVM model in our case. When you click one of these labels, the curve for that model is highlighted and the corresponding metrics are displayed, as shown in the following graphic.



True Positive	False Negative	Accuracy	Precision	Threshold	Cumulative AUC
441	309	0.646	0.846	0.5	0.748
False Positive	True Negative	Recall	F1 Score		
80	270	0.588	0.694		

By examining these values, you can decide which model is closest to giving you the results you're looking for. You can go back and iterate on your experiment by changing parameter values in the different models.

The science and art of interpreting these results and tuning the model performance is outside the scope of this walkthrough. For additional help, you might read the following articles:

- [How to evaluate model performance in Azure Machine Learning](#)
- [Choose parameters to optimize your algorithms in Azure Machine Learning](#)
- [Interpret model results in Azure Machine Learning](#)

TIP

Each time you run the experiment a record of that iteration is kept in the Run History. You can view these iterations, and return to any of them, by clicking **VIEW RUN HISTORY** below the canvas. You can also click **Prior Run** in the **Properties** pane to return to the iteration immediately preceding the one you have open.

You can make a copy of any iteration of your experiment by clicking **SAVE AS** below the canvas. Use the experiment's **Summary** and **Description** properties to keep a record of what you've tried in your experiment iterations.

For more details, see [Manage experiment iterations in Azure Machine Learning Studio](#).

Next: Deploy the web service

Walkthrough Step 5: Deploy the Azure Machine Learning Web service

1/17/2017 • 9 min to read • [Edit on GitHub](#)

This is the fifth step of the walkthrough, [Develop a predictive analytics solution in Azure Machine Learning](#)

1. [Create a Machine Learning workspace](#)
2. [Upload existing data](#)
3. [Create a new experiment](#)
4. [Train and evaluate the models](#)
5. **Deploy the Web service**
6. [Access the Web service](#)

To give others a chance to use the predictive model we've developed in this walkthrough, we can deploy it as a Web service on Azure.

Up to this point we've been experimenting with training our model. But the deployed service is no longer going to do training - it generates predictions by scoring the user's input based on our model. So we're going to do some preparation to convert this experiment from a **training** experiment to a **predictive** experiment.

This is a two-step process:

1. Convert the *training experiment* we've created into a *predictive experiment*
2. Deploy the predictive experiment as a Web service

But first, we need to trim this experiment down a little. We currently have two different models in the experiment, but we only want one model when we deploy this as a Web service.

Let's say we've decided that the boosted tree model performed better than the SVM model. So the first thing to do is remove the [Two-Class Support Vector Machine](#) module and the modules that were used for training it. You may want to make a copy of the experiment first by clicking **Save As** at the bottom of the experiment canvas.

We need to delete the following modules:

- [Two-Class Support Vector Machine](#)
- [Train Model](#) and [Score Model](#) modules that were connected to it
- [Normalize Data](#) (both of them)
- [Evaluate Model](#) (because we're finished evaluating the models)

Select each module and press the Delete key, or right-click the module and select **Delete**.

Now we're ready to deploy this model using the [Two-Class Boosted Decision Tree](#).

Convert the training experiment to a predictive experiment

Converting to a predictive experiment involves three steps:

1. Save the model we've trained and then replace our training modules
2. Trim the experiment to remove modules that were only needed for training
3. Define where the Web service will accept input and where it generates the output

We could do this manually, but fortunately all three steps can be accomplished by clicking **Set Up Web service**

at the bottom of the experiment canvas (and selecting the **Predictive Web service** option).

TIP

If you want more details on what happens when you convert a training experiment to a predictive experiment, see [Convert a Machine Learning training experiment to a predictive experiment](#).

When you click **Set Up Web service**, several things happen:

- The trained model is converted to a single **Trained Model** module and stored in the module palette to the left of the experiment canvas (you can find it under **Trained Models**)
- Modules that were used for training are removed; specifically:
 - Two-Class Boosted Decision Tree
 - Train Model
 - Split Data
 - the second [Execute R Script](#) module that was used for test data
- The saved trained model is added back into the experiment
- **Web service input** and **Web service output** modules are added (these identify where the user's data will enter the model, and what data is returned, when the Web service is accessed)

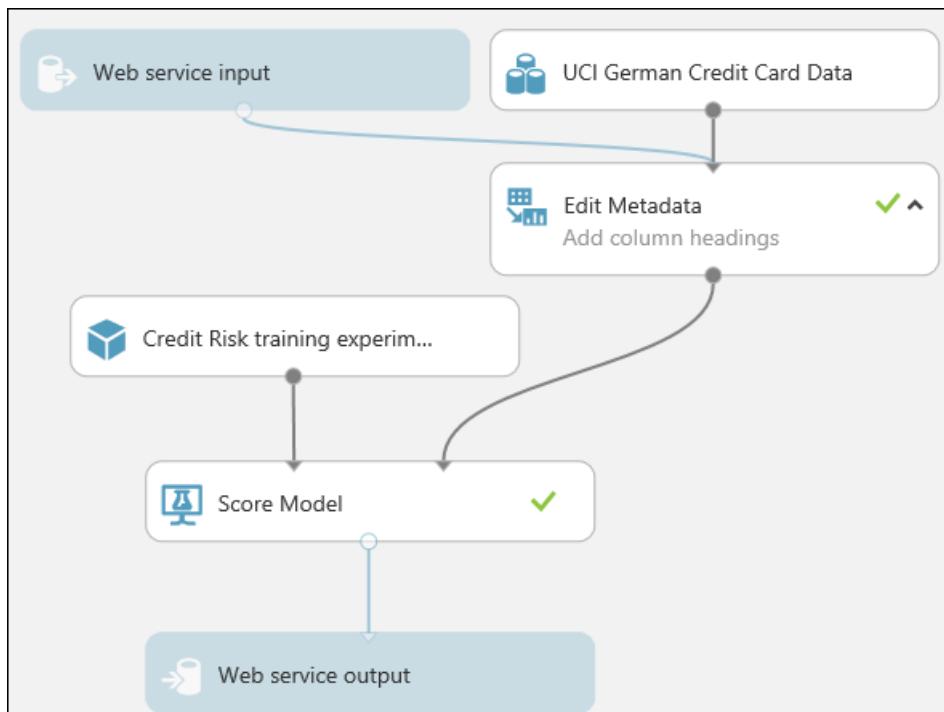
NOTE

You can see that the experiment is saved in two parts under tabs that have been added at the top of the experiment canvas. The original training experiment is under the tab **Training experiment**, and the newly created predictive experiment is under **Predictive experiment**. The predictive experiment is the one we'll deploy as a Web service.

We need to take one additional step with this particular experiment. We added two [Execute R Script](#) modules to provide a weighting function to the data. That was just a trick we needed for training and testing, so we can take those modules out in the final model.

Machine Learning Studio removed one [Execute R Script](#) module when it removed the [Split](#) module. Now we can remove the other and connect [Metadata Editor](#) directly to [Score Model](#).

Our experiment should now look like this:



NOTE

You may be wondering why we left the UCI German Credit Card Data dataset in the predictive experiment. The service is going to score the user's data, not the original dataset, so why leave the original dataset in the model?

It's true that the service doesn't need the original credit card data. But it does need the schema for that data, which includes information such as how many columns there are and which columns are numeric. This schema information is necessary to interpret the user's data. We leave these components connected so that the scoring module has the dataset schema when the service is running. The data isn't used, just the schema.

Run the experiment one last time (click **Run**.) If you want to verify that the model is still working, click the output of the **Score Model** module and select **View Results**. You'll see that the original data is displayed, along with the credit risk value ("Scored Labels") and the scoring probability value ("Scored Probabilities".)

Deploy the Web service

You can deploy the experiment as either a classic Web service or a new Web service that's based on Azure Resource Manager.

Deploy as a classic Web service

To deploy a classic Web service derived from our experiment, click **Deploy Web Service** below the canvas and select **Deploy Web Service [Classic]**. Machine Learning Studio deploys the experiment as a Web service and takes you to the dashboard for that Web service. From here, you can return to the experiment (**View snapshot** or **View latest**) and run a simple test of the Web service (See **Test the Web service** below). There is also information here for creating applications that can access the Web service (more on that in the next step of this walkthrough).

REQUEST/RESPONSE	Test	Excel 2013 or later Excel 2010 or earlier workbook	LAST UPDATED
BATCH EXECUTION		Excel 2013 or later workbook	2/5/2016 5:43:22 PM

You can configure the service by clicking the **CONFIGURATION** tab. Here you can modify the service name (it's given the experiment name by default) and give it a description. You can also give more friendly labels for the input and output data.

credit risk experiment

DASHBOARD **CONFIGURATION**

settings

GENERAL

Display Name	Credit Risk experiment
Description	No description provided for this web service.

ENABLE LOGGING

[Learn more](#)

ENABLE SAMPLE DATA?

INPUT SCHEMA

Col1 (String)
Col2 (Numeric)

Deploy as a New Web service

To deploy a New Web service derived from our experiment:

1. Click **Deploy Web Service** below the canvas and select **Deploy Web Service [New]**. Machine Learning Studio transfers you to the Azure Machine Learning Web services **Deploy Experiment** page.
2. Enter a name for the Web service.
3. For **Price Plan**, you can select an existing pricing plan, or select "Create new" and give the new plan a name and select the monthly plan option. The plan tiers default to the plans for your default region and your Web service is deployed to that region.
4. Click **Deploy**.

After a few minutes, the **Quickstart** page for your Web service opens.

You can configure the service by clicking the **Configure** tab. Here you can modify the service title and give it a description.

To test the Web service select, click the **Test** tab (see **Test the Web service** below). For information on creating applications that can access the Web service, click the **Consume** tab (the next step in this walkthrough will go into more detail).

TIP

You can update the Web service after you've deployed it. For example, if you want to change your model, then you can edit the training experiment, tweak the model parameters, and click **Deploy Web Service**, selecting **Deploy Web Service [Classic]** or **Deploy Web Service [New]**. When you deploy the experiment again, it replaces the Web service, now using your updated model.

Test the Web service

When the Web service is accessed, the user's data enters through the **Web service input** module where it's passed to the **Score Model** module and scored. The way we've set up the predictive experiment, the model expects data in the same format as the original credit risk dataset.

The results are then returned to the user from the Web service through the **Web service output** module.

TIP

The way we have the predictive experiment configured, the entire results from the **Score Model** module are returned. This includes all the input data plus the credit risk value and the scoring probability. If you wanted to return something different - for example, only the credit risk value - then you could insert a **Project Columns** module between **Score Model** and the **Web service output** to eliminate columns you don't want the Web service to return.

Test a classic Web service

You can test the Web service in Machine Learning Studio or in the Azure Machine Learning Web Services portal. Testing in the Azure Machine Learning Web Services portal has the advantage of allowing you to enable

Test in Machine Learning Studio

1. On the **DASHBOARD** page, click the **Test** button under **Default Endpoint**. A dialog pops up and asks you for the input data for the service. These are the same columns that appeared in the original credit risk dataset.
2. Enter a set of data and then click **OK**.

Test in the Azure Machine Learning Web Services portal

1. On the **DASHBOARD** page, click the **Test** preview link under **Default Endpoint**. The test page in the Azure Machine Learning Web Services portal for the Web service endpoint opens and asks you for the input data for the service. These are the same columns that appeared in the original credit risk dataset.
2. Click **Test Request-Response**.

Test a new Web service

1. In the Azure Machine Learning Web services portal, click **Test** at the top of the page. The **Test** page opens and you can input data for the service. The input fields displayed correspond to the columns that appeared in the original credit risk dataset.
2. Enter a set of data and then click **Test Request-Response**.

The results of the test will display on the right hand side of the page in the output column.

TIP

When testing in the Azure Machine Learning Web Services portal, you can have the portal create sample data that you can use to test the Request-Response service. On the **Configure** page, select "Yes" for **Sample Data Enabled?**. When you open the Request-Response tab on the **Test** page, the portal will fill in sample data taken from the original credit risk dataset.

Manage the Web service

Manage a Classic Web service in the Azure classic portal

Once you've deployed your Classic Web service, you can manage it from the [Azure classic portal](#).

1. Sign in to the [Azure classic portal](#)
2. In the Microsoft Azure services panel, click **MACHINE LEARNING**
3. Click your workspace
4. Click the **Web services** tab
5. Click the Web service we created
6. Click the "default" endpoint

From here, you can do things like monitor how the Web service is doing and make performance tweaks by changing how many concurrent calls the service can handle.

For more details, see:

- [Creating Endpoints](#)
- [Scaling Web service](#)

Manage a Web service in the Azure Machine Learning Web Services portal

Once you've deployed your Web service, whether Classic or New, you can manage it from the [Azure Machine Learning Web services portal](#).

To monitor the performance of your Web service:

1. Sign in to the [Azure Machine Learning Web services portal](#)
2. Click **Web services**
3. Click your Web service
4. Click the **Dashboard**

Next: [Access the Web service](#)

Walkthrough Step 6: Access the Azure Machine Learning Web service

1/17/2017 • 1 min to read • [Edit on GitHub](#)

This is the last step of the walkthrough, [Develop a predictive analytics solution in Azure Machine Learning](#)

1. [Create a Machine Learning workspace](#)
2. [Upload existing data](#)
3. [Create a new experiment](#)
4. [Train and evaluate the models](#)
5. [Deploy the Web service](#)
6. **Access the Web service**

In the previous step in this walkthrough we deployed a Web service that uses our credit risk prediction model. Now users are able to send data to it and receive results.

The Web service is an Azure Web service that can receive and return data using REST APIs in one of two ways:

- **Request/Response** - The user sends one or more rows of credit data to the service by using an HTTP protocol, and the service responds with one or more sets of results.
- **Batch Execution** - The user stores one or more rows of credit data in an Azure blob and then sends the blob location to the service. The service scores all the rows of data in the input blob, stores the results in another blob, and returns the URL of that container.

The quickest and easiest way to access the Web service is through the [Azure ML Request-Response Service Web App](#) or [Azure ML Batch Execution Service Web App Template](#). These web app templates can build a custom web app that knows your Web service's input data and what it will return. All you need to do is provide access to your Web service and data, and the template does the rest.

For more information on using the web app templates, see [Consume an Azure Machine Learning Web service with a web app template](#).

You can also develop a custom application to access the Web service using starter code provided for you in R, C#, and Python programming languages. You can find complete details in [How to consume an Azure Machine Learning Web service that has been published from a Machine Learning experiment](#).

Data Science for Beginners video 1: The 5 questions data science answers

1/17/2017 • 4 min to read • [Edit on GitHub](#)

Get a quick introduction to data science from *Data Science for Beginners* in five short videos from a top data scientist. These videos are basic but useful, whether you're interested in doing data science or you work with data scientists.

This first video is about the kinds of questions that data science can answer. To get the most out of the series, watch them all. [Go to the list of videos](#)



Other videos in this series

Data Science for Beginners is a quick introduction to data science taking about 25 minutes total. Check out the other four videos:

- Video 1: The 5 questions data science answers
- Video 2: [Is your data ready for data science?](#) (4 min 56 sec)
- Video 3: [Ask a question you can answer with data](#) (4 min 17 sec)
- Video 4: [Predict an answer with a simple model](#) (7 min 42 sec)
- Video 5: [Copy other people's work to do data science](#) (3 min 18 sec)

Transcript: The 5 questions data science answers

Hi! Welcome to the video series *Data Science for Beginners*.

Data Science can be intimidating, so I'll introduce the basics here without any equations or computer programming jargon.

In this first video, we'll talk about "The 5 questions data science answers."

Data Science uses numbers and names (also known as categories or labels) to predict answers to questions.

It might surprise you, but *there are only five questions that data science answers*:

- Is this A or B?
- Is this weird?
- How much – or – How many?
- How is this organized?
- What should I do next?

Each one of these questions is answered by a separate family of machine learning methods, called algorithms.

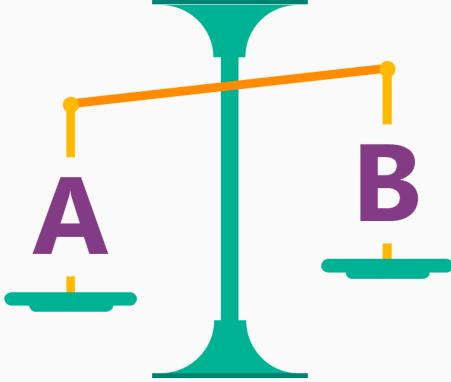
It's helpful to think about an algorithm as a recipe and your data as the ingredients. An algorithm tells how to combine and mix the data in order to get an answer. Computers are like a blender. They do most of the hard work of the algorithm for you and they do it pretty fast.

Question 1: Is this A or B? uses classification algorithms

Let's start with the question: Is this A or B?

Is this A or B?

Classification algorithms



This family of algorithms is called two-class classification.

It's useful for any question that has just two possible answers.

For example:

- Will this tire fail in the next 1,000 miles: Yes or no?
- Which brings in more customers: a \$5 coupon or a 25% discount?

This question can also be rephrased to include more than two options: Is this A or B or C or D, etc.? This is called multiclass classification and it's useful when you have several—or several thousand—possible answers.

Multiclass classification chooses the most likely one.

Question 2: Is this weird? uses anomaly detection algorithms

The next question data science can answer is: Is this weird? This question is answered by a family of algorithms called anomaly detection.

Is this weird?

Anomaly detection algorithms



If you have a credit card, you've already benefitted from anomaly detection. Your credit card company analyzes your purchase patterns, so that they can alert you to possible fraud. Charges that are "weird" might be a purchase at a store where you don't normally shop or buying an unusually pricey item.

This question can be useful in lots of ways. For instance:

- If you have a car with pressure gauges, you might want to know: Is this pressure gauge reading normal?
- If you're monitoring the internet, you'd want to know: Is this message from the internet typical?

Anomaly detection flags unexpected or unusual events or behaviors. It gives clues where to look for problems.

Question 3: How much? or How many? uses regression algorithms

Machine learning can also predict the answer to How much? or How many? The algorithm family that answers this question is called regression.

How much? How many?

Regression algorithms

Monday



72°

Tuesday



Regression algorithms make numerical predictions, such as:

- What will the temperature be next Tuesday?
- What will my fourth quarter sales be?

They help answer any question that asks for a number.

Question 4: How is this organized? uses clustering algorithms

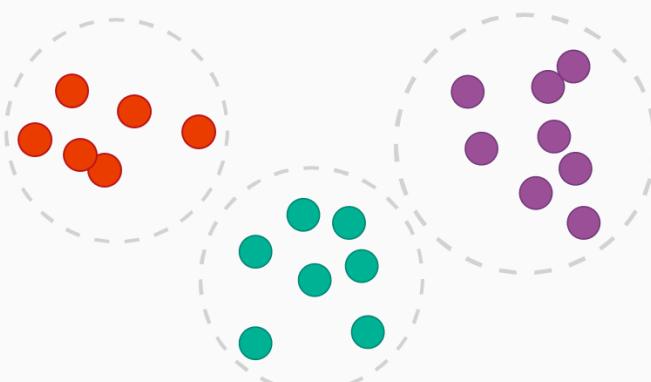
Now the last two questions are a bit more advanced.

Sometimes you want to understand the structure of a data set - How is this organized? For this question, you don't have examples that you already know outcomes for.

There are a lot of ways to tease out the structure of data. One approach is clustering. It separates data into natural "clumps," for easier interpretation. With clustering, there is no one right answer.

How is this organized?

Clustering Algorithms



The diagram shows three distinct clusters of colored dots. The top-left cluster contains five red dots. The bottom cluster contains six teal dots. The top-right cluster contains seven purple dots. Each cluster is enclosed within a dashed circle, illustrating how clustering algorithms group data points based on their proximity.

Common examples of clustering questions are:

- Which viewers like the same types of movies?
- Which printer models fail the same way?

By understanding how data is organized, you can better understand - and predict - behaviors and events.

Question 5: What should I do now? uses reinforcement learning algorithms

The last question – What should I do now? – uses a family of algorithms called reinforcement learning.

Reinforcement learning was inspired by how the brains of rats and humans respond to punishment and rewards. These algorithms learn from outcomes, and decide on the next action.

Typically, reinforcement learning is a good fit for automated systems that have to make lots of small decisions without human guidance.

What should I do now?

Reinforcement Learning Algorithms



Questions it answers are always about what action should be taken - usually by a machine or a robot. Examples are:

- If I'm a temperature control system for a house: Adjust the temperature or leave it where it is?
- If I'm a self-driving car: At a yellow light, brake or accelerate?
- For a robot vacuum: Keep vacuuming, or go back to the charging station?

Reinforcement learning algorithms gather data as they go, learning from trial and error.

So that's it - The 5 questions data science can answer.

Next steps

- [Try a first data science experiment with Machine Learning Studio](#)
- [Get an introduction to Machine Learning on Microsoft Azure](#)

Is your data ready for data science?

1/17/2017 • 4 min to read • [Edit on GitHub](#)

Video 2: Data Science for Beginners series

Learn how to evaluate your data to make sure it meets basic criteria to be ready for data science.

To get the most out of the series, watch them all. [Go to the list of videos](#)



Other videos in this series

Data Science for Beginners is a quick introduction to data science in five short videos.

- Video 1: [The 5 questions data science answers](#) (5 min 14 sec)
- Video 2: Is your data ready for data science?
- Video 3: [Ask a question you can answer with data](#) (4 min 17 sec)
- Video 4: [Predict an answer with a simple model](#) (7 min 42 sec)
- Video 5: [Copy other people's work to do data science](#) (3 min 18 sec)

Transcript: Is your data ready for data science?

Welcome to "Is your data ready for data science?" the second video in the series *Data Science for Beginners*.

Before data science can give you the answers you want, you have to give it some high-quality raw materials to work with. Just like making a pizza, the better the ingredients you start with, the better the final product.

Criteria for data

So, in the case of data science, there are some ingredients that we need to pull together.

We need data that is:

- Relevant
- Connected
- Accurate
- Enough to work with

Is your data relevant?

So the first ingredient - we need data that's relevant.

Irrelevant Data

Price of milk (\$/gal)	Red Sox batting avg.	Blood alcohol content (%)
3.79	.304	.03
3.45	.320	.09
4.06	.259	.01
3.89	.298	.05
4.12	.332	.13
3.92	.270	.06
3.23	.294	.10

Relevant Data

Body mass (kg)	Margaritas	Blood alcohol content (%)
103	3	.03
67	5	.09
87	1	.01
52	2	.05
73	5	.13
79	3	.06
110	7	.10

Look at the table on the left. We met seven people outside of Boston bars, measured their blood alcohol level, the Red Sox batting average in their last game, and the price of milk in the nearest convenience store.

This is all perfectly legitimate data. Its only fault is that it isn't relevant. There's no obvious relationship between these numbers. If I gave you the current price of milk and the Red Sox batting average, there's no way you could guess my blood alcohol content.

Now look at the table on the right. This time we measured each person's body mass and counted the number of drinks they've had. The numbers in each row are now relevant to each other. If I gave you my body mass and the number of Margaritas I've had, you could make a guess at my blood alcohol content.

Do you have connected data?

The next ingredient is connected data.

Disconnected Data

Grill temp. (Fahrenheit)	Weight of beef patty (lb)	Burger rating (out of 10)
	.33	8.2
	.24	5.6
550		7.8
725	.45	9.4
600		8.2
625		6.8
	.49	4.2

Connected Data

Grill temp. (Fahrenheit)	Weight of beef patty (lb)	Burger rating (out of 10)
575	.33	8.2
550	.24	5.6
550	.69	7.8
725	.45	9.4
600	.57	8.2
625	.36	6.8
550	.49	4.2

Here is some relevant data on the quality of hamburgers: grill temperature, patty weight, and rating in the local food magazine. But notice the gaps in the table on the left.

Most data sets are missing some values. It's common to have holes like this and there are ways to work around them. But if there's too much missing, your data begins to look like Swiss cheese.

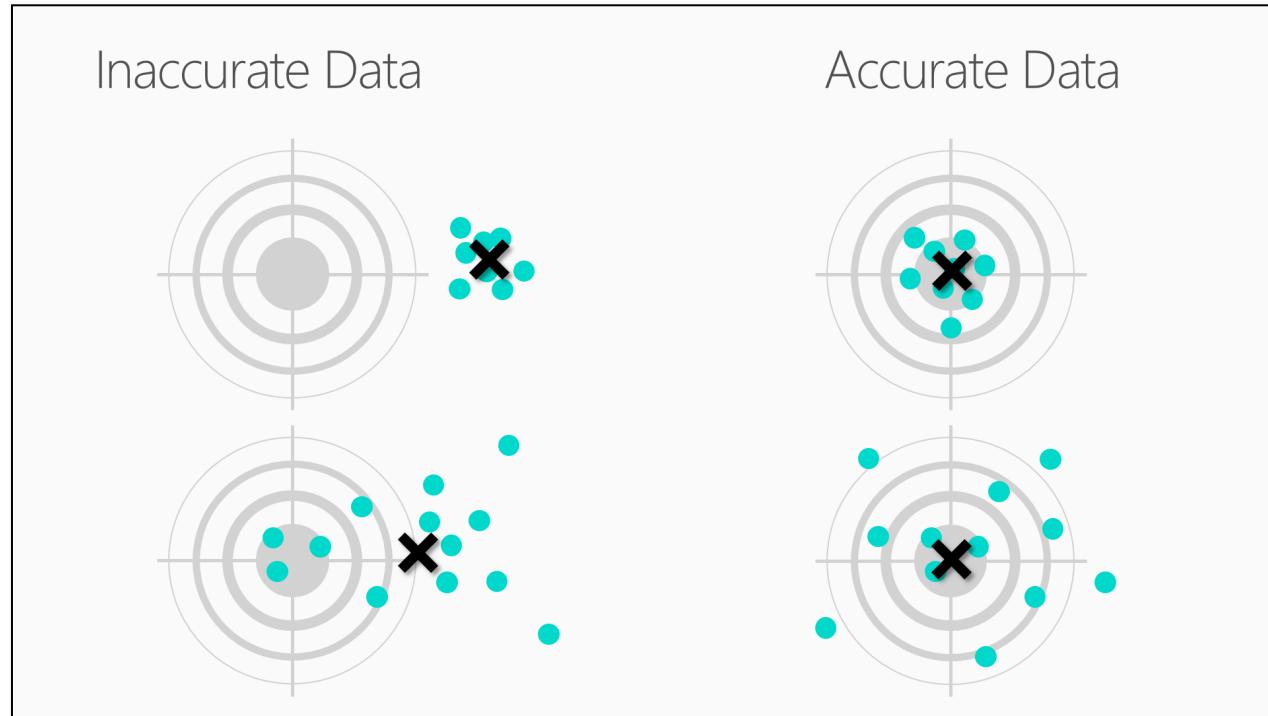
If you look at the table on the left, there's so much missing data, it's hard to come up with any kind of relationship

between grill temperature and patty weight. This is an example of disconnected data.

The table on the right, though, is full and complete - an example of connected data.

Is your data accurate?

The next ingredient we need is accuracy. Here are four targets that we'd like to hit with arrows.



Look at the target in the upper right. We've got a tight grouping right around the bullseye. That, of course, is accurate. Oddly, in the language of data science, our performance on the target right below it is also considered accurate.

If you were to map out the center of these arrows, you'd see that it's very close to the bullseye. The arrows are spread out all around the target, so they're considered imprecise, but they're centered around the bullseye, so they're considered accurate.

Now look at the upper-left target. Here our arrows hit very close together, a tight grouping. They're precise, but they're inaccurate because the center is way off the bullseye. And, of course, the arrows in the bottom-left target are both inaccurate and imprecise. This archer needs more practice.

Do you have enough data to work with?

Finally, ingredient #4 - we need to have enough data.

Barely enough data



Think of each data point in your table as being a brush stroke in a painting. If you have only a few of them, the painting can be pretty fuzzy - it's hard to tell what it is.

If you add some more brush strokes, then your painting starts to get a little sharper.

When you have barely enough strokes, you can see just enough to make some broad decisions. Is it somewhere I might want to visit? It looks bright, that looks like clean water – yes, that's where I'm going on vacation.

As you add more data, the picture becomes clearer and you can make more detailed decisions. Now I can look at the three hotels on the left bank. You know, I really like the architectural features of the one in the foreground. I'll stay there, on the third floor.

With data that's relevant, connected, accurate, and enough, we have all the ingredients we need to do some high-quality data science.

Be sure to check out the other four videos in *Data Science for Beginners* from Microsoft Azure Machine Learning.

Next steps

- [Try a first data science experiment with Machine Learning Studio](#)
- [Get an introduction to Machine Learning on Microsoft Azure](#)

Ask a question you can answer with data

1/17/2017 • 4 min to read • [Edit on GitHub](#)

Video 3: Data Science for Beginners series

Learn how to formulate a data science question in Data Science for Beginners video 3. This video includes a comparison of questions for classification and regression algorithms.

To get the most out of the series, watch them all. [Go to the list of videos](#)



Other videos in this series

Data Science for Beginners is a quick introduction to data science in five short videos.

- Video 1: [The 5 questions data science answers](#) (5 min 14 sec)
- Video 2: [Is your data ready for data science?](#) (4 min 56 sec)
- Video 3: Ask a question you can answer with data
- Video 4: [Predict an answer with a simple model](#) (7 min 42 sec)
- Video 5: [Copy other people's work to do data science](#) (3 min 18 sec)

Transcript: Ask a question you can answer with data

Welcome to the third video in the series "Data Science for Beginners."

In this one, you'll get some tips for formulating a question you can answer with data.

You might get more out of this video, if you first watch the two earlier videos in this series: "The 5 questions data science can answer" and "Is your data is ready for data science?"

Ask a sharp question

We've talked about how data science is the process of using names (also called categories or labels) and numbers to predict an answer to a question. But it can't be just any question; it has to be a *sharp question*.

A vague question doesn't have to be answered with a name or a number. A sharp question must.

Imagine you found a magic lamp with a genie who will truthfully answer any question you ask. But it's a mischievous genie, and he'll try to make his answer as vague and confusing as he can get away with. You want to pin him down with a question so airtight that he can't help but tell you what you want to know.

If you were to ask a vague question, like "What's going to happen with my stock?", the genie might answer, "The price will change". That's a truthful answer, but it's not very helpful.

But if you were to ask a sharp question, like "What will my stock's sale price be next week?", the genie can't help but give you a specific answer and predict a sale price.

Examples of your answer: Target data

Once you formulate your question, check to see whether you have examples of the answer in your data.

If our question is "What will my stock's sale price be next week?" then we have to make sure our data includes the stock price history.

If our question is "Which car in my fleet is going to fail first?" then we have to make sure our data includes information about previous failures.

Examples of the answer: Target data



These examples of answers are called a target. A target is what we are trying to predict about future data points, whether it's a category or a number.

If you don't have any target data, you'll need to get some. You won't be able to answer your question without it.

Reformulate your question

Sometimes you can reword your question to get a more useful answer.

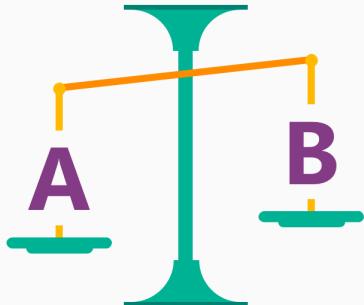
The question "Is this data point A or B?" predicts the category (or name or label) of something. To answer it, we use a *classification algorithm*.

The question "How much?" or "How many?" predicts an amount. To answer it we use a *regression algorithm*.

To see how we can transform these, let's look at the question, "Which news story is the most interesting to this reader?" It asks for a prediction of a single choice from many possibilities - in other words "Is this A or B or C or D?" - and would use a classification algorithm.

But, this question may be easier to answer if you reword it as "How interesting is each story on this list to this reader?" Now you can give each article a numerical score, and then it's easy to identify the highest-scoring article. This is a rephrasing of the classification question into a regression question or How much?

Reformulate your question



How you ask a question is a clue to which algorithm can give you an answer.

You'll find that certain families of algorithms - like the ones in our news story example - are closely related. You can reformulate your question to use the algorithm that gives you the most useful answer.

But, most important, ask that sharp question - the question that you can answer with data. And be sure you have the right data to answer it.

We've talked about some basic principles for asking a question you can answer with data.

Be sure to check out the other videos in "Data Science for Beginners" from Microsoft Azure Machine Learning.

Next steps

- [Try a first data science experiment with Machine Learning Studio](#)
- [Get an introduction to Machine Learning on Microsoft Azure](#)

Predict an answer with a simple model

1/17/2017 • 5 min to read • [Edit on GitHub](#)

Video 4: Data Science for Beginners series

Learn how to create a simple regression model to predict the price of a diamond in Data Science for Beginners video 4. We'll draw a regression model with target data.

To get the most out of the series, watch them all. [Go to the list of videos](#)



Other videos in this series

Data Science for Beginners is a quick introduction to data science in five short videos.

- Video 1: [The 5 questions data science answers](#) (5 min 14 sec)
- Video 2: [Is your data ready for data science?](#) (4 min 56 sec)
- Video 3: [Ask a question you can answer with data](#) (4 min 17 sec)
- Video 4: Predict an answer with a simple model
- Video 5: [Copy other people's work to do data science](#) (3 min 18 sec)

Transcript: Predict an answer with a simple model

Welcome to the fourth video in the "Data Science for Beginners" series. In this one, we'll build a simple model and make a prediction.

A *model* is a simplified story about our data. I'll show you what I mean.

Collect relevant, accurate, connected, enough data

Say I want to shop for a diamond. I have a ring that belonged to my grandmother with a setting for a 1.35 carat diamond, and I want to get an idea of how much it will cost. I take a notepad and pen into the jewelry store, and I write down the price of all of the diamonds in the case and how much they weigh in carats. Starting with the first diamond - it's 1.01 carats and \$7,366.

Now I go through and do this for all the other diamonds in the store.

<u>Carats</u>	<u>Price</u>
1.01	7,366
.49	985
.31	544
1.51	9,140
.37	493
.73	3,011
1.53	11,413
.56	1,814
.41	876
.74	2,690
.63	1,190
.6	4,172
2.06	11,764
1.1	4,682
1.31	6,171

Notice that our list has two columns. Each column has a different attribute - weight in carats and price - and each row is a single data point that represents a single diamond.

We've actually created a small data set here - a table. Notice that it meets our criteria for quality:

- The data is **relevant** - weight is definitely related to price
- It's **accurate** - we double-checked the prices that we write down
- It's **connected** - there are no blank spaces in either of these columns
- And, as we'll see, it's **enough** data to answer our question

Ask a sharp question

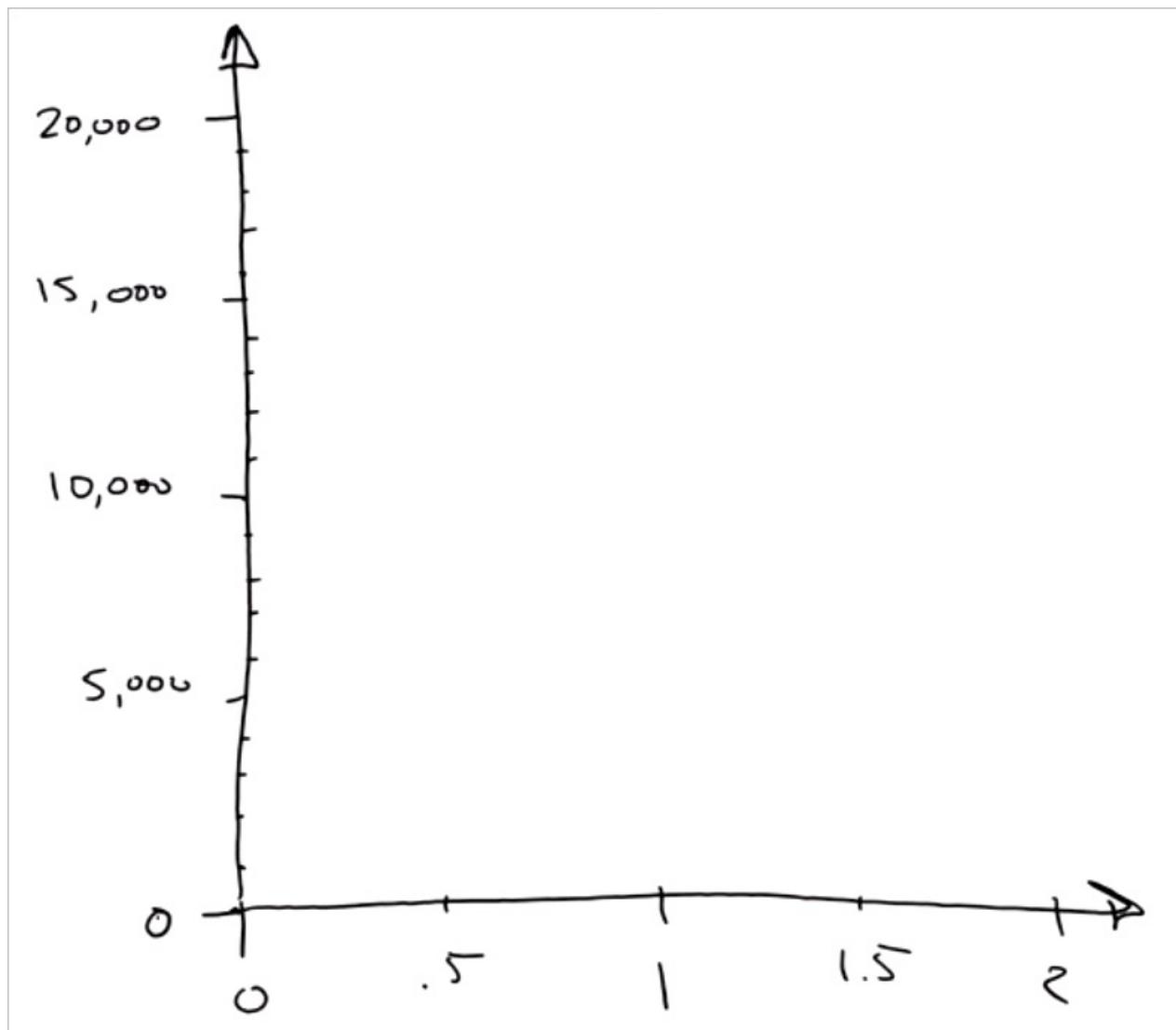
Now we'll pose our question in a sharp way: "How much will it cost to buy a 1.35 carat diamond?"

Our list doesn't have a 1.35 carat diamond in it, so we'll have to use the rest of our data to get an answer to the question.

Plot the existing data

The first thing we'll do is draw a horizontal number line, called an axis, to chart the weights. The range of the weights is 0 to 2, so we'll draw a line that covers that range and put ticks for each half carat.

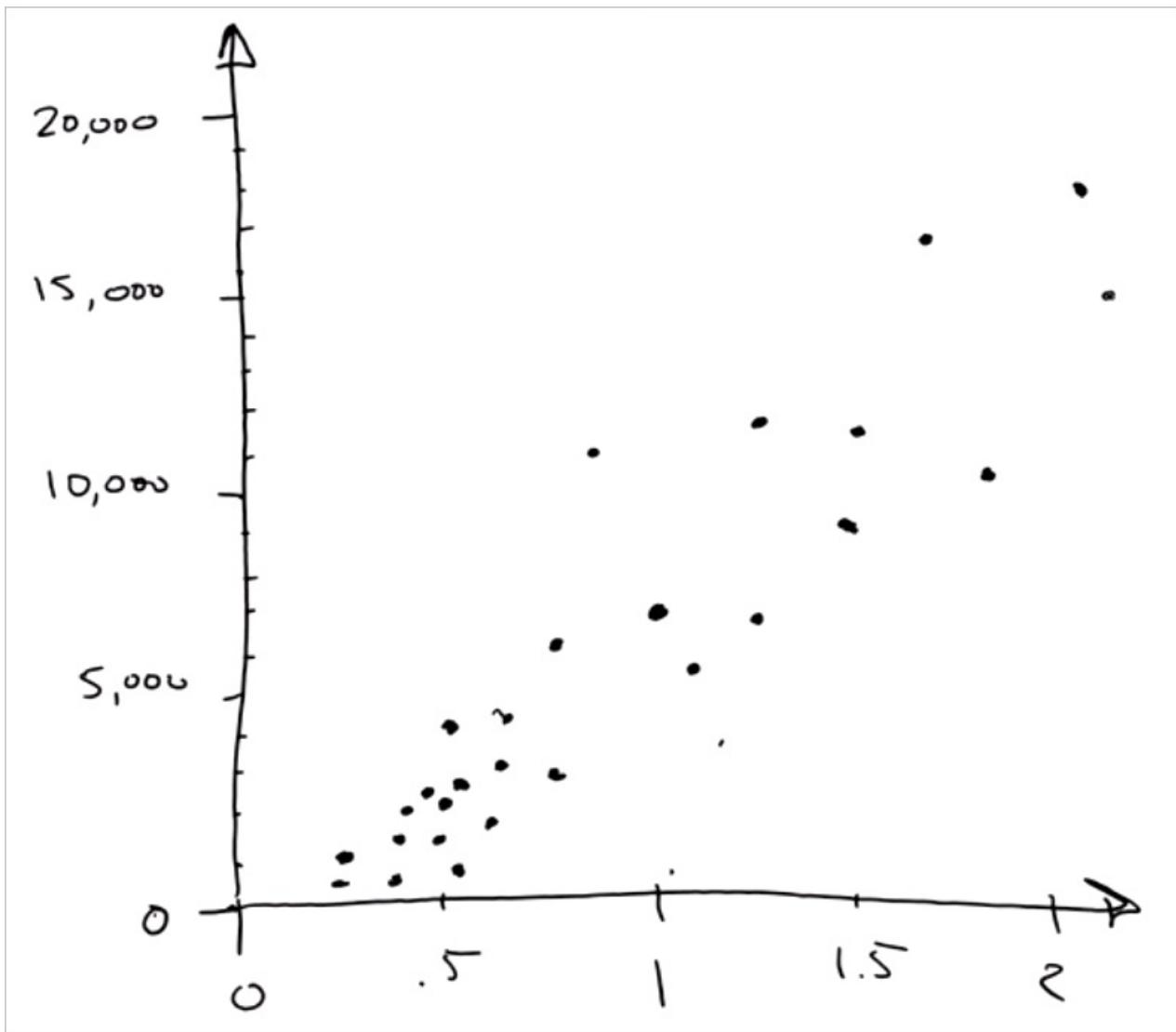
Next we'll draw a vertical axis to record the price and connect it to the horizontal weight axis. This will be in units of dollars. Now we have a set of coordinate axes.



We're going to take this data now and turn it into a *scatter plot*. This is a great way to visualize numerical data sets.

For the first data point, we eyeball a vertical line at 1.01 carats. Then, we eyeball a horizontal line at \$7,366. Where they meet, we draw a dot. This represents our first diamond.

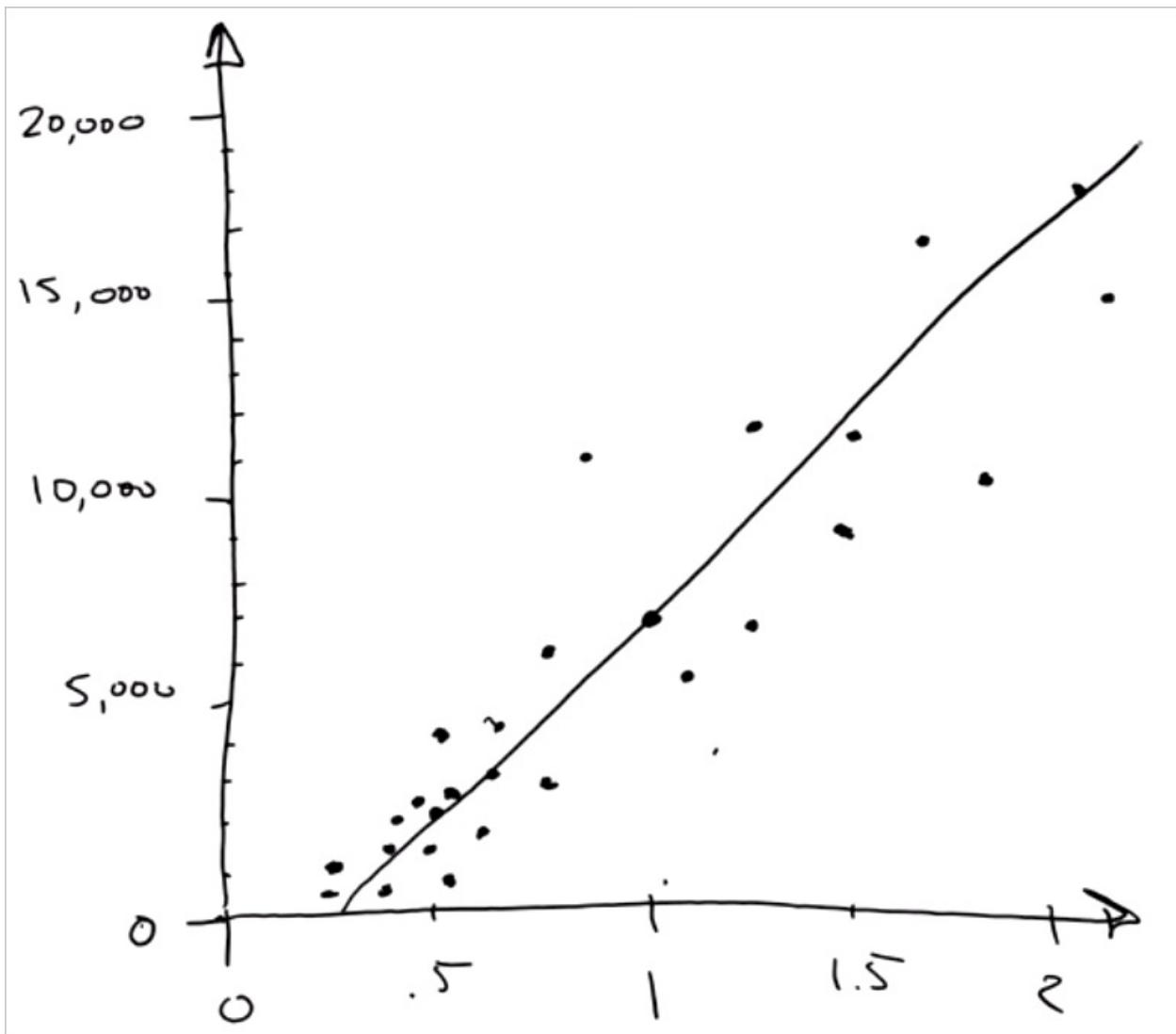
Now we go through each diamond on this list and do the same thing. When we're through, this is what we get: a bunch of dots, one for each diamond.



Draw the model through the data points

Now if you look at the dots and squint, the collection looks like a fat, fuzzy line. We can take our marker and draw a straight line through it.

By drawing a line, we created a *model*. Think of this as taking the real world and making a simplistic cartoon version of it. Now the cartoon is wrong - the line doesn't go through all the data points. But, it's a useful simplification.



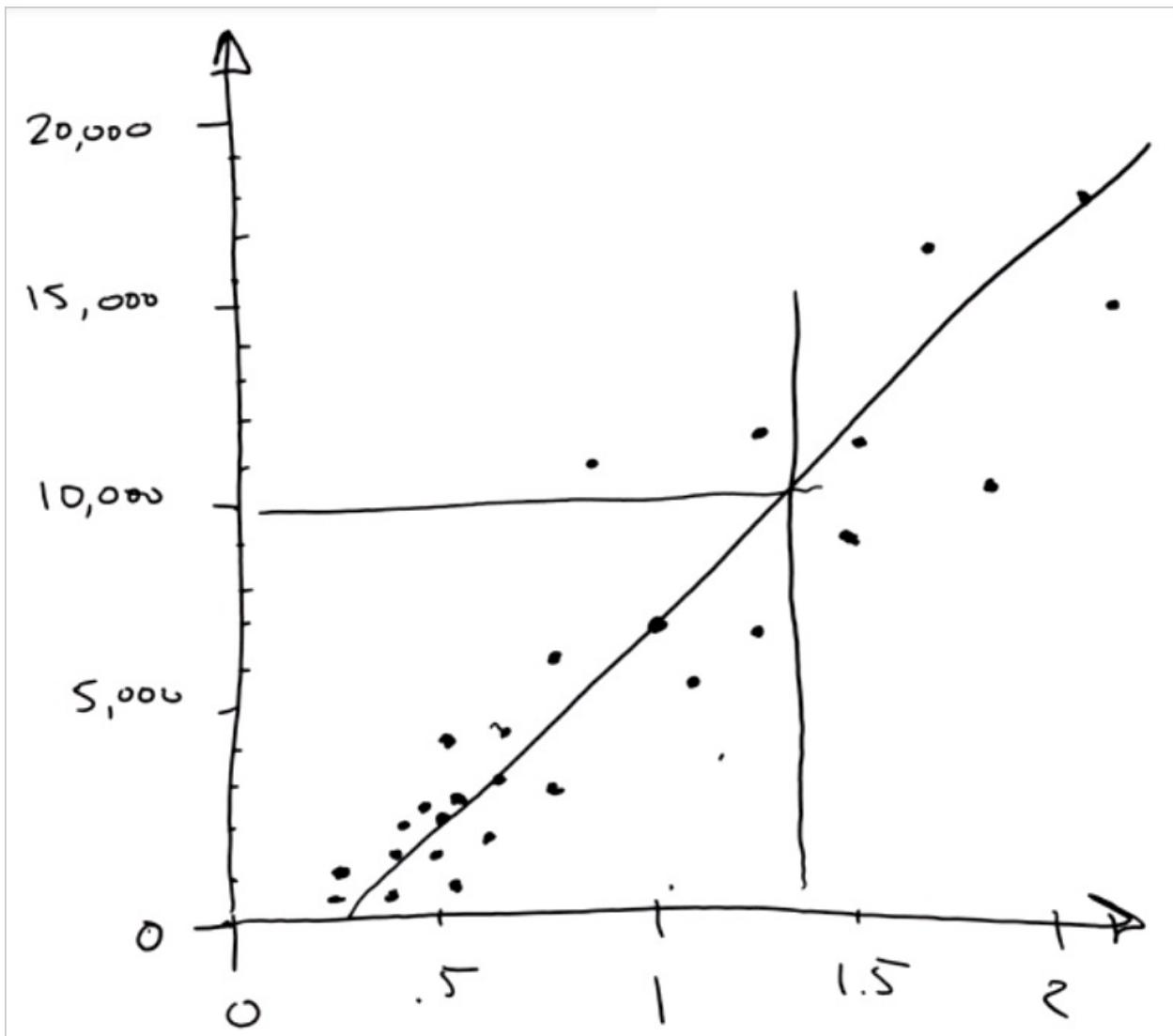
The fact that all the dots don't go exactly through the line is OK. Data scientists explain this by saying that there's the model - that's the line - and then each dot has some *noise* or *variance* associated with it. There's the underlying perfect relationship, and then there's the gritty, real world that adds noise and uncertainty.

Because we're trying to answer the question *How much?* this is called a *regression*. And because we're using a straight line, it's a *linear regression*.

Use the model to find the answer

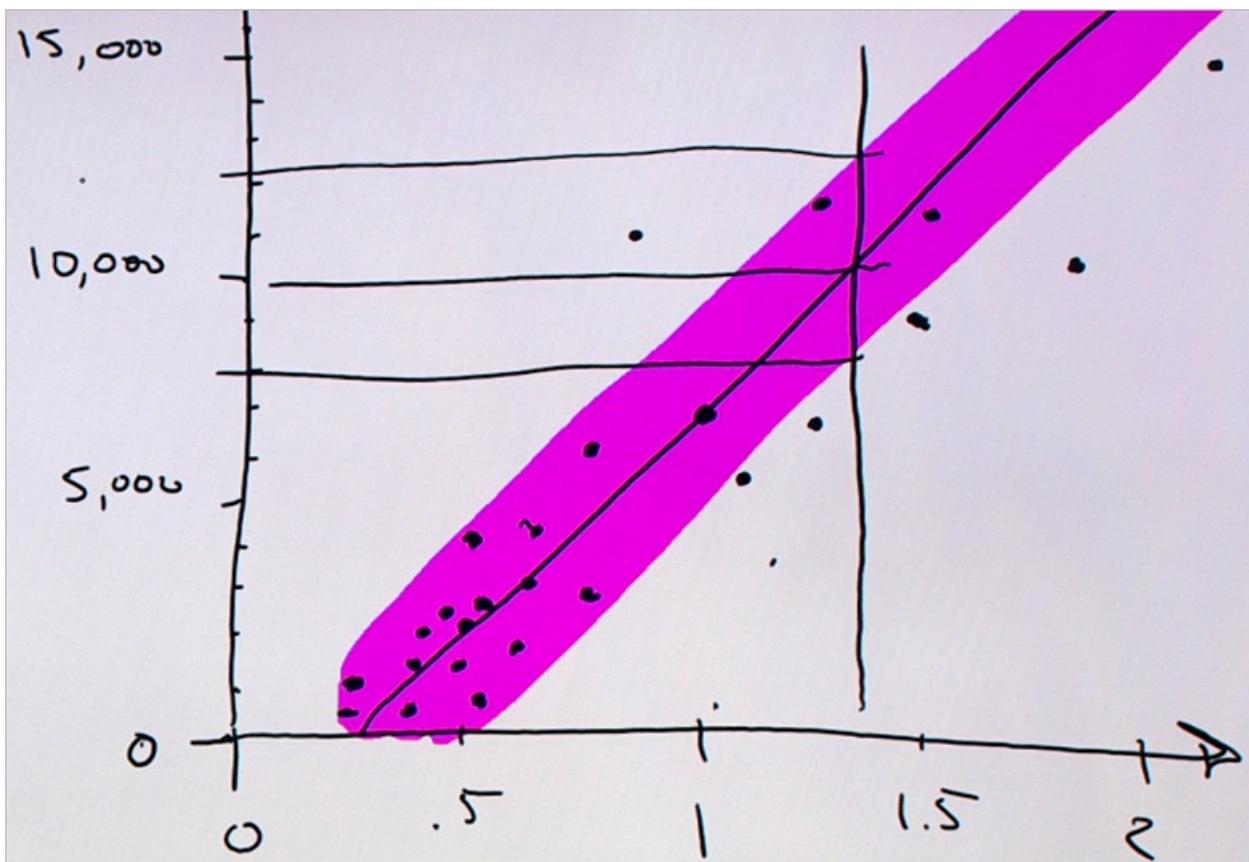
Now we have a model and we ask it our question: How much will a 1.35 carat diamond cost?

To answer our question, we eyeball 1.35 carats and draw a vertical line. Where it crosses the model line, we eyeball a horizontal line to the dollar axis. It hits right at 10,000. Boom! That's the answer: A 1.35 carat diamond costs about \$10,000.



Create a confidence interval

It's natural to wonder how precise this prediction is. It's useful to know whether the 1.35 carat diamond will be very close to \$10,000, or a lot higher or lower. To figure this out, let's draw an envelope around the regression line that includes most of the dots. This envelope is called our *confidence interval*: We're pretty confident that prices fall within this envelope, because in the past most of them have. We can draw two more horizontal lines from where the 1.35 carat line crosses the top and the bottom of that envelope.



Now we can say something about our confidence interval: We can say confidently that the price of a 1.35 carat diamond is about \$10,000 - but it might be as low as \$8,000 and it might be as high as \$12,000.

We're done, with no math or computers

We did what data scientists get paid to do, and we did it just by drawing:

- We asked a question that we could answer with data
- We built a *model* using *linear regression*
- We made a *prediction*, complete with a *confidence interval*

And we didn't use math or computers to do it.

Now if we'd had more information, like...

- the cut of the diamond
- color variations (how close the diamond is to being white)
- the number of inclusions in the diamond

...then we would have had more columns. In that case, math becomes helpful. If you have more than two columns, it's hard to draw dots on paper. The math lets you fit that line or that plane to your data very nicely.

Also, if instead of just a handful of diamonds, we had two thousand or two million, then you can do that work much faster with a computer.

Today, we've talked about how to do linear regression, and we made a prediction using data.

Be sure to check out the other videos in "Data Science for Beginners" from Microsoft Azure Machine Learning.

Next steps

- [Try a first data science experiment with Machine Learning Studio](#)
- [Get an introduction to Machine Learning on Microsoft Azure](#)

Copy other people's work to do data science

1/17/2017 • 3 min to read • [Edit on GitHub](#)

Video 5: Data Science for Beginners series

One of the trade secrets of data science is getting other people to do your work for you. Find a clustering algorithm example to use for your own machine learning experiment.

To get the most out of the series, watch them all. [Go to the list of videos](#)



Other videos in this series

Data Science for Beginners is a quick introduction to data science in five short videos.

- Video 1: [The 5 questions data science answers](#) (5 min 14 sec)
- Video 2: [Is your data ready for data science?](#) (4 min 56 sec)
- Video 3: [Ask a question you can answer with data](#) (4 min 17 sec)
- Video 4: [Predict an answer with a simple model](#) (7 min 42 sec)
- Video 5: Copy other people's work to do data science

Transcript: Copy other people's work to do data science

Welcome to the fifth video in the series "Data Science for Beginners."

In this one, you'll discover a place to find examples that you can borrow from as a starting point for your own work. You might get the most out of this video if you first watch the earlier videos in this series.

One of the trade secrets of data science is getting other people to do your work for you.

Find examples in the Cortana Intelligence Gallery

Microsoft has a cloud-based service called [Azure Machine Learning](#) that you're welcome to try for free. It provides you with a workspace where you can experiment with different machine learning algorithms, and, when you've got your solution worked out, you can launch it as a web service.

Part of this service is something called the [Cortana Intelligence Gallery](#). It contains a variety of resources, one of which is a collection of Azure Machine Learning experiments, or models, that people have built and contributed for others to use. These experiments are a great way to leverage the thought and hard work of others to get you started on your own solutions.

You can find the gallery at aka.ms/CortanaIntelligenceGallery. Everyone is welcome to browse through it.

Cortana Intelligence Gallery

Browse all Solution Templates Experiments Machine Learning APIs Notebooks Competitions Tutorials Collections

Cortana Intelligence Gallery enables our growing community of developers and data scientists to share their analytics solutions. Learn how to contribute.

TUTORIAL Setting up predictive analytics pipelines using Azure SQL Data Warehouse Microsoft

COMPETITION Decoding Brain Signals Microsoft

COLLECTION Cognitive Services Microsoft

NOTEBOOK Computing Influence Score for Twitter Users Microsoft

EXPERIMENT Binary Classification: CRM - Upselling Marian Dragt

Recently added

[See all](#)

 TUTORIAL Retail Forecasting Template with SQL Server R Services	 TUTORIAL Automatically installing JupyterHub on Linux Data Sci...	 COLLECTION Education related experiments	 COLLECTION DataKind VisionZero
--	--	---	---------------------------------------

If you click **Experiments** at the top, you'll see a number of the most recent and popular experiments in the gallery. You can search through the rest of experiments by clicking **Browse All** at the top of the screen, and there you can enter search terms and choose search filters.

Find and use a clustering algorithm example

So, for instance, let's say you want to see an example of how clustering works, so you search for "**clustering**" experiments.

Cortana Intelligence Gallery

clustering

Browse all Industries Solution Templates Experiments

Here's an interesting one that someone contributed to the gallery.

EXPERIMENT

Clustering sweep: diabetes dataset

This experiment demonstrates how to use the new Sweep Clustering module to select the best number of centroids and optimize other para...

K-Means Clustering

🕒 1430 ⬇ 733 9 months ago

Jeannine Takaki

Click on that experiment and you get a web page that describes the work that this contributor did, along with some of their results.

Cortana Intelligence Gallery

Browse all Industries Solution Templates Experiments Machine Learning APIs Notebooks Competitions More

EXPERIMENT

Clustering sweep: diabetes dataset

Jeannine Takaki • published on October 28, 2015

Summary

This experiment demonstrates how to use the new Sweep Clustering module to select the best number of centroids and optimize other parameters

Description

Summary

This experiment uses a parameter sweep with the K-means clustering algorithm to select the best number of clusters and the best initial centroid, based on a clustering metric you select. In the experiment, the original dataset is divided into two parts, to handle missing data and to compare the effect of variables on clustering. The results of the models are compared using a principal components graph. The experiment also demonstrates how you can use the [Sweep Clustering](#) module to fill in values for a label column.

Understanding the Data

The dataset contains 768 rows, from a larger dataset that studies the incidence of diabetes among different populations.

The following clinical values are included, which are often used in diagnosing diabetes.

Triceps skin fold measurements (TSF) and body mass index (BMI) are thought to be correlated with

[Open in Studio](#)

+ Add to Collection

🕒 1428 views ⬇ 730 downloads

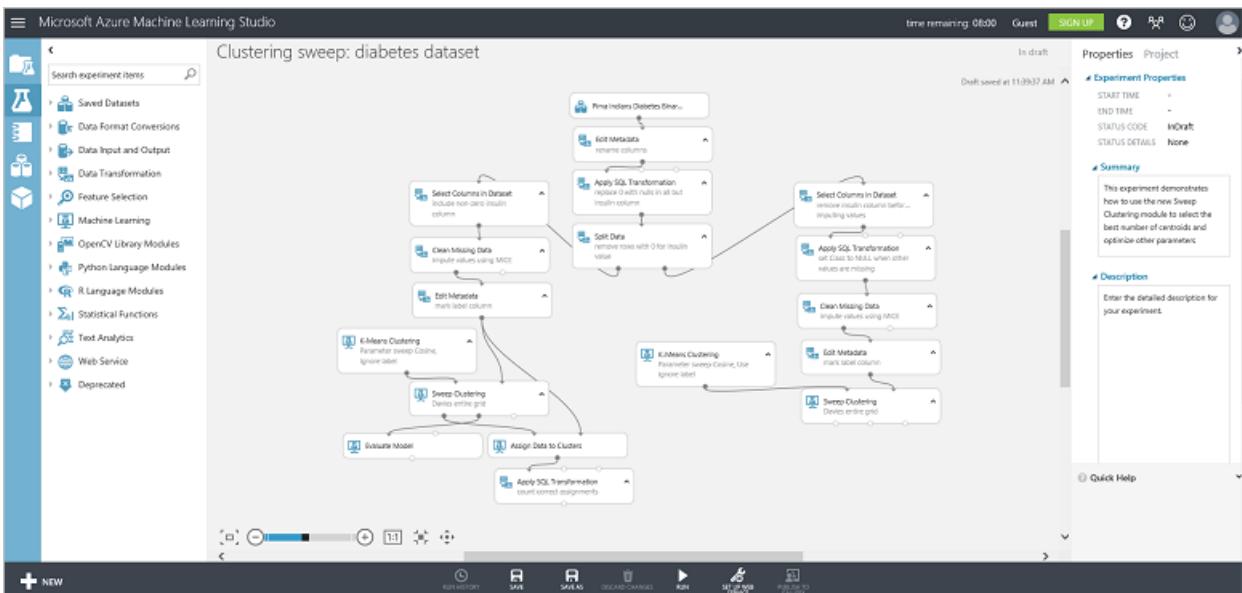
Tweet Share

ALGORITHMS
K-Means Clustering

Notice the link that says **Open in Studio**.



I can click on that and it takes me right to **Azure Machine Learning Studio**. It creates a copy of the experiment and puts it in my own workspace. This includes the contributor's dataset, all the processing that they did, all of the algorithms that they used, and how they saved out the results.



And now I have a starting point. I can swap out their data for my own and do my own tweaking of the model. This gives me a running start, and it lets me build on the work of people who really know what they're doing.

Find experiments that demonstrate machine learning techniques

There are other experiments in the [Cortana Intelligence Gallery](#) that were contributed specifically to provide how-to examples for people new to data science. For instance, there's an experiment in the gallery that demonstrates how to handle missing values ([Methods for handling missing values](#)). It walks you through 15 different ways of substituting empty values, and talks about the benefits of each method and when to use it.

Cortana Intelligence Gallery

Browse all Industries Solution Templates Experiments Machine Learning APIs Notebooks Competitions More

EXPERIMENT

Methods for handling missing values

Brandon Rohrer • published on September 28, 2015

Summary

This experiment illustrates a variety methods for handling missing data on a sample data set.

Description

Real world data is usually missing values, which trip up a lot of machine learning algorithms. There are lots of tricks for dealing with these, but you have to be careful. The way in which you fill them can change the result dramatically. Being explicit and thoughtful about how you handle missing values will get you the very best results.

I've illustrated a large handful of approaches to missing values here in a fake data set. The data shows a group of employees, some of their personal data, and some data regarding an upcoming office party. In every case, knowing what the data means is the most important part of handling it well.

Column 0	age	years_seniority	income	parking_space	attending_party	entree	pets	emergency_contact
Tony	48	27		1	5	shrimp		Pepper
Donald	67	25	86	10	2	beef		Jane
Henry	69	21	95	6	1	chicken	62	Janet
Janet	62	21	110	3	1	beef		Henry

Open in Studio

+ Add to Collection

863 views 95 downloads

[Tweet](#) [Share](#)

TAGS

missing_values data_quality method

[Cortana Intelligence Gallery](#) is a place to find working experiments that you can use as a starting point for your own solutions.

Be sure to check out the other videos in "Data Science for Beginners" from Microsoft Azure Machine Learning.

Next steps

- Try your first data science experiment with Azure Machine Learning
- Get an introduction to Machine Learning on Microsoft Azure

Quickstart tutorial for the R programming language for Azure Machine Learning

1/17/2017 • 53 min to read • [Edit on GitHub](#)

Introduction

This quickstart tutorial helps you quickly start extending Azure Machine Learning by using the R programming language. Follow this R programming tutorial to create, test and execute R code within Azure Machine Learning. As you work through tutorial, you will create a complete forecasting solution by using the R language in Azure Machine Learning.

Microsoft Azure Machine Learning contains many powerful machine learning and data manipulation modules. The powerful R language has been described as the lingua franca of analytics. Happily, analytics and data manipulation in Azure Machine Learning can be extended by using R. This combination provides the scalability and ease of deployment of Azure Machine Learning with the flexibility and deep analytics of R.

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

Forecasting and the dataset

Forecasting is a widely employed and quite useful analytical method. Common uses range from predicting sales of seasonal items, determining optimal inventory levels, to predicting macroeconomic variables. Forecasting is typically done with time series models.

Time series data is data in which the values have a time index. The time index can be regular, e.g. every month or every minute, or irregular. A time series model is based on time series data. The R programming language contains a flexible framework and extensive analytics for time series data.

In this quickstart guide we will be working with California dairy production and pricing data. This data includes monthly information on the production of several dairy products and the price of milk fat, a benchmark commodity.

The data used in this article, along with R scripts, can be [downloaded here](#). This data was originally synthesized from information available from the University of Wisconsin at <http://future.aae.wisc.edu/tab/production.html>.

Organization

We will progress through several steps as you learn how to create, test and execute analytics and data manipulation R code in the Azure Machine Learning environment.

- First we will explore the basics of using the R language in the Azure Machine Learning Studio environment.
- Then we progress to discussing various aspects of I/O for data, R code and graphics in the Azure Machine Learning environment.
- We will then construct the first part of our forecasting solution by creating code for data cleaning and transformation.
- With our data prepared we will perform an analysis of the correlations between several of the variables in our dataset.
- Finally, we will create a seasonal time series forecasting model for milk production.

Interact with R language in Machine Learning Studio

This section takes you through some basics of interacting with the R programming language in the Machine Learning Studio environment. The R language provides a powerful tool to create customized analytics and data manipulation modules within the Azure Machine Learning environment.

I will use RStudio to develop, test and debug R code on a small scale. This code is then cut and paste into an [Execute R Script](#) module in Machine Learning Studio ready to run.

The Execute R Script module

Within Machine Learning Studio, R scripts are run within the [Execute R Script](#) module. An example of the [Execute R Script](#) module in Machine Learning Studio is shown in Figure 1.

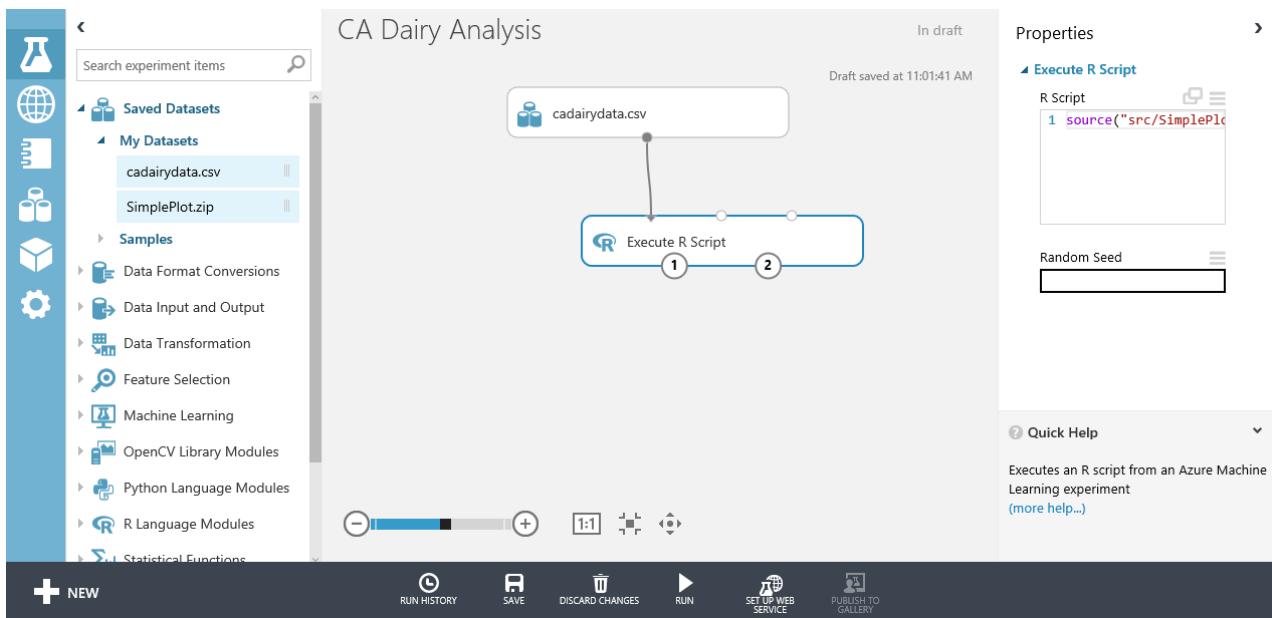


Figure 1. The Machine Learning Studio environment showing the [Execute R Script](#) module selected.

Referring to Figure 1, let's look at some of the key parts of the Machine Learning Studio environment for working with the [Execute R Script](#) module.

- The modules in the experiment are shown in the center pane.
- The upper part of the right pane contains a window to view and edit your R scripts.
- The lower part of right pane shows some properties of the [Execute R Script](#). You can view the error and output logs by clicking on the appropriate spots of this pane.

We will, of course, be discussing the [Execute R Script](#) in greater detail in the rest of this document.

When working with complex R functions, I recommend that you edit, test and debug in RStudio. As with any software development, extend your code incrementally and test it on small simple test cases. Then cut and paste your functions into the R script window of the [Execute R Script](#) module. This approach allows you to harness both the RStudio integrated development environment (IDE) and the power of Azure Machine Learning.

Execute R code

Any R code in the [Execute R Script](#) module will execute when you run the experiment by clicking on the **Run** button. When execution has completed, a check mark will appear on the [Execute R Script](#) icon.

Defensive R coding for Azure Machine Learning

If you are developing R code for, say, a web service by using Azure Machine Learning, you should definitely plan how your code will deal with an unexpected data input and exceptions. To maintain clarity, I have not included much in the way of checking or exception handling in most of the code examples shown. However, as we proceed I will give you several examples of functions by using R's exception handling capability.

If you need a more complete treatment of R exception handling, I recommend you read the applicable sections of the book by Wickham listed in [Appendix B - Further Reading](#).

Debug and test R in Machine Learning Studio

To reiterate, I recommend you test and debug your R code on a small scale in RStudio. However, there are cases where you will need to track down R code problems in the [Execute R Script](#) itself. In addition, it is good practice to check your results in Machine Learning Studio.

Output from the execution of your R code and on the Azure Machine Learning platform is found primarily in output.log. Some additional information will be seen in error.log.

If an error occurs in Machine Learning Studio while running your R code, your first course of action should be to look at error.log. This file can contain useful error messages to help you understand and correct your error. To view error.log, click on **View error log** on the **properties pane** for the [Execute R Script](#) containing the error.

For example, I ran the following R code, with an undefined variable y, in an [Execute R Script](#) module:

```
x<- 1.0  
z<- x+y
```

This code fails to execute, resulting in an error condition. Clicking on **View error log** on the **properties pane** produces the display shown in Figure 2.

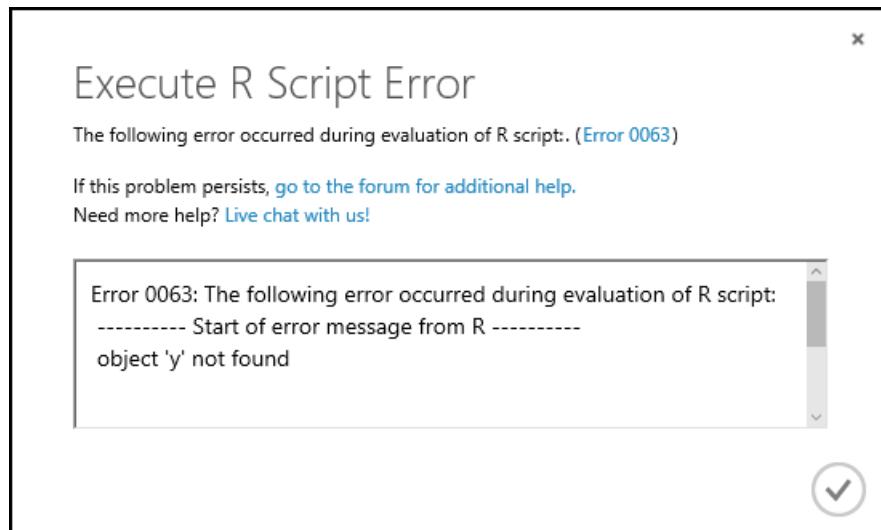


Figure 2. Error message pop-up.

It looks like we need to look in output.log to see the R error message. Click on the [Execute R Script](#) and then click on the **View output.log** item on the **properties pane** to the right. A new browser window opens, and I see the following.

```
[Critical] Error: Error 0063: The following error occurred during evaluation of R script:  
----- Start of error message from R -----  
object 'y' not found  
  
object 'y' not found  
----- End of error message from R -----
```

This error message contains no surprises and clearly identifies the problem.

To inspect the value of any object in R, you can print these values to the output.log file. The rules for examining object values are essentially the same as in an interactive R session. For example, if you type a variable name on a line, the value of the object will be printed to the output.log file.

Packages in Machine Learning Studio

Azure Machine Learning comes with over 350 preinstalled R language packages. You can use the following code in the [Execute R Script](#) module to retrieve a list of the preinstalled packages.

```
data.set <- data.frame(installed.packages())
maml.mapOutputPort("data.set")
```

If you don't understand the last line of this code at the moment, read on. In the rest of this document we will extensively discuss using R in the Azure Machine Learning environment.

Introduction to RStudio

RStudio is a widely used IDE for R. I will use RStudio for editing, testing and debugging some of the R code used in this quick start guide. Once R code is tested and ready, you simply cut and paste from the RStudio editor into a Machine Learning Studio [Execute R Script](#) module.

If you do not have the R programming language installed on your desktop machine, I recommend you do so now. Free downloads of open source R language are available at the Comprehensive R Archive Network (CRAN) at <http://www.r-project.org/>. There are downloads available for Windows, Mac OS, and Linux/UNIX. Choose a nearby mirror and follow the download directions. In addition, CRAN contains a wealth of useful analytics and data manipulation packages.

If you are new to RStudio, you should download and install the desktop version. You can find the RStudio downloads for Windows, Mac OS, and Linux/UNIX at <http://www.rstudio.com/products/RStudio/>. Follow the directions provided to install RStudio on your desktop machine.

A tutorial introduction to RStudio is available at <https://support.rstudio.com/hc/sections/200107586-Using-RStudio>.

I provide some additional information on using RStudio in [Appendix A](#).

Get data in and out of the Execute R Script module

In this section we will discuss how you get data into and out of the [Execute R Script](#) module. We will review how to handle various data types read into and out of the [Execute R Script](#) module.

The complete code for this section is in the zip file you downloaded earlier.

Load and check data in Machine Learning Studio

Load the dataset

We will start by loading the **csdairydata.csv** file into Azure Machine Learning Studio.

- Start your Azure Machine Learning Studio environment.
- Click on **+ NEW** at the lower left of your screen and select **Dataset**.
- Select **From Local File**, and then **Browse** to select the file.
- Make sure you have selected **Generic CSV file with header (.csv)** as the type for the dataset.
- Click the check mark.
- After the dataset has been uploaded, you should see the new dataset by clicking on the **Datasets** tab.

Create an experiment

Now that we have some data in Machine Learning Studio, we need to create an experiment to do the analysis.

- Click on **+ NEW** at the lower left and select **Experiment**, then **Blank Experiment**.
- You can name your experiment by selecting, and modifying, the **Experiment created on ...** title at the top of the page. For example, changing it to **CA Dairy Analysis**.
- On the left of the experiment page, expand **Saved Datasets**, and then **My Datasets**. You should see the **cadairydata.csv** that you uploaded earlier.

- Drag and drop the **csdairydata.csv dataset** onto the experiment.
- In the **Search experiment items** box on the top of the left pane, type [Execute R Script](#). You will see the module appear in the search list.
- Drag and drop the [Execute R Script](#) module onto your pallet.
- Connect the output of the **csdairydata.csv dataset** to the leftmost input (**Dataset1**) of the [Execute R Script](#).
- **Don't forget to click on 'Save'!**

At this point your experiment should look something like Figure 3.

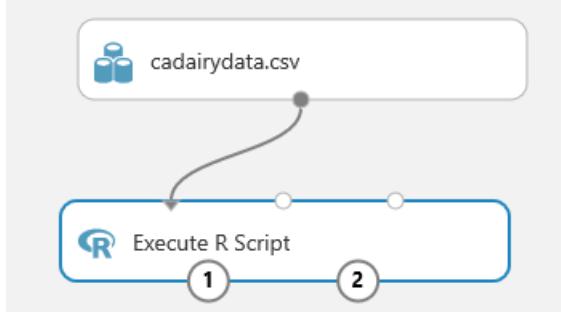


Figure 3. The CA Dairy Analysis experiment with dataset and Execute R Script module.

Check on the data

Let's have a look at the data we have loaded into our experiment. In the experiment, click on the output of the **cadairydata.csv dataset** and select **visualize**. You should see something like Figure 4.

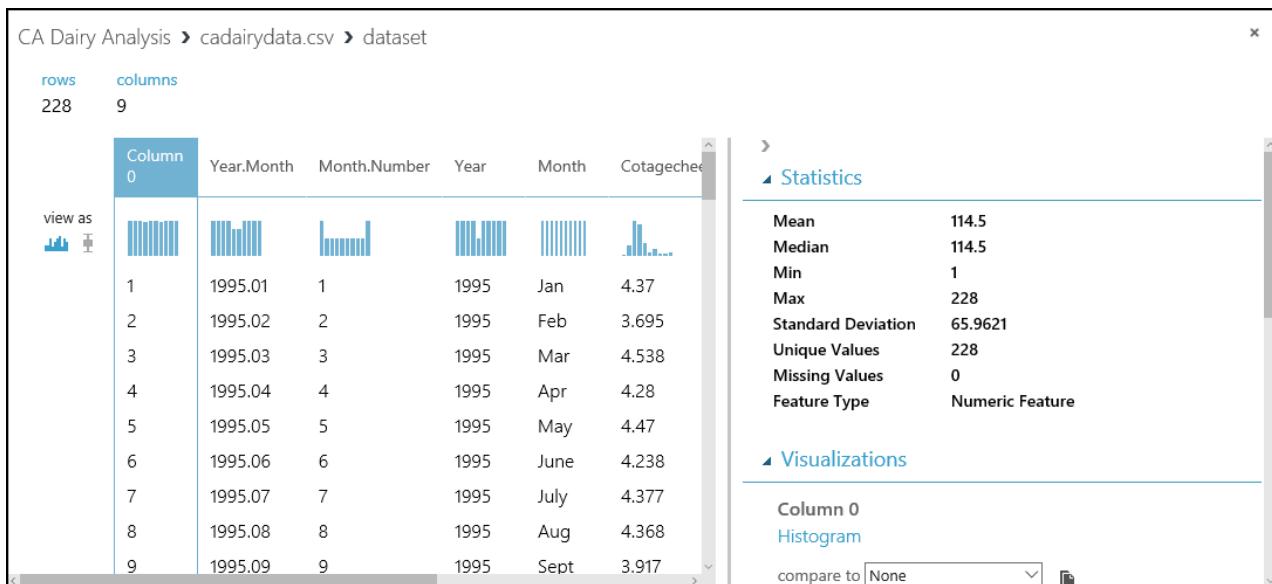


Figure 4. Summary of the cadairydata.csv dataset.

In this view we see a lot of useful information. We can see the first several rows of that dataset. If we select a column, the Statistics section shows more information about the column. For example, the Feature Type row shows us what data types Azure Machine Learning Studio assigned to the column. Having a quick look like this is a good sanity check before we start to do any serious work.

First R script

Let's create a simple first R script to experiment with in Azure Machine Learning Studio. I have created and tested the following script in RStudio.

```

## Only one of the following two lines should be used
## If running in Machine Learning Studio, use the first line with maml.mapInputPort()
## If in RStudio, use the second line with read.csv()
cadairydata <- maml.mapInputPort(1)
# cadairydata <- read.csv("cadairydata.csv", header = TRUE, stringsAsFactors = FALSE)
str(cadairydata)
pairs(~ Cotagecheese.Prod + Icecream.Prod + Milk.Prod + N.CA.Fat.Price, data = cadairydata)
## The following line should be executed only when running in
## Azure Machine Learning Studio
maml.mapOutputPort('cadairydata')

```

Now I need to transfer this script to Azure Machine Learning Studio. I could simply cut and paste. However, in this case, I will transfer my R script via a zip file.

Data input to the Execute R Script module

Let's have a look at the inputs to the [Execute R Script](#) module. In this example we will read the California dairy data into the [Execute R Script](#) module.

There are three possible inputs for the [Execute R Script](#) module. You may use any one or all of these inputs, depending on your application. It is also perfectly reasonable to use an R script that takes no input at all.

Let's look at each of these inputs, going from left to right. You can see the names of each of the inputs by placing your cursor over the input and reading the tooltip.

Script Bundle

The Script Bundle input allows you to pass the contents of a zip file into [Execute R Script](#) module. You can use one of the following commands to read the contents of the zip file into your R code.

```

source("src/yourfile.R") # Reads a zipped R script
load("src/yourData.rdata") # Reads a zipped R data file

```

NOTE

Azure Machine Learning treats files in the zip as if they are in the src/ directory, so you need to prefix your file names with this directory name. For example, if the zip contains the files `yourfile.R` and `yourData.rdata` in the root of the zip, you would address these as `src/yourfile.R` and `src/yourData.rdata` when using `source` and `load`.

We already discussed loading datasets in [Loading the dataset](#). Once you have created and tested the R script shown in the previous section, do the following:

1. Save the R script into a .R file. I call my script file "simpleplot.R". Here's the contents.

```

## Only one of the following two lines should be used
## If running in Machine Learning Studio, use the first line with maml.mapInputPort()
## If in RStudio, use the second line with read.csv()
cadairydata <- maml.mapInputPort(1)
# cadairydata <- read.csv("cadairydata.csv", header = TRUE, stringsAsFactors = FALSE)
str(cadairydata)
pairs(~ Cotagecheese.Prod + Icecream.Prod + Milk.Prod + N.CA.Fat.Price, data = cadairydata)
## The following line should be executed only when running in
## Azure Machine Learning Studio
maml.mapOutputPort('cadairydata')

```

2. Create a zip file and copy your script into this zip file. On Windows, you can right-click on the file and select **Send to**, and then **Compressed folder**. This will create a new zip file containing the "simpleplot.R" file.
3. Add your file to the **datasets** in Machine Learning Studio, specifying the type as **zip**. You should now see the zip file in your datasets.

4. Drag and drop the zip file from **datasets** onto the **ML Studio canvas**.
5. Connect the output of the **zip data** icon to the **Script Bundle** input of the **Execute R Script** module.
6. Type the `source()` function with your zip file name into the code window for the **Execute R Script** module. In my case I typed `source("src/simpleplot.R")`.
7. Make sure you click **Save**.

Once these steps are complete, the **Execute R Script** module will execute the R script in the zip file when the experiment is run. At this point your experiment should look something like Figure 5.

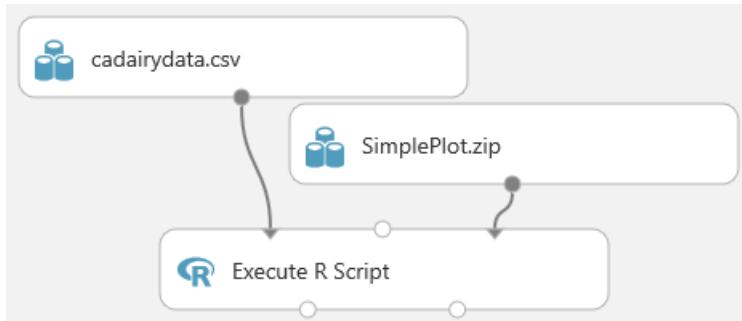


Figure 5. Experiment using zipped R script.

Dataset1

You can pass a rectangular table of data to your R code by using the **Dataset1** input. In our simple script the `maml.mapInputPort(1)` function reads the data from port 1. This data is then assigned to a dataframe variable name in your code. In our simple script the first line of code performs the assignment.

```
cadairydata <- maml.mapInputPort(1)
```

Execute your experiment by clicking on the **Run** button. When the execution finishes, click on the **Execute R Script** module and then click **View output log** on the properties pane. A new page should appear in your browser showing the contents of the `output.log` file. When you scroll down you should see something like the following.

```
[ModuleOutput] InputDataStructure
[ModuleOutput]
[ModuleOutput] {
[ModuleOutput] "InputName":Dataset1
[ModuleOutput] "Rows":228
[ModuleOutput] "Cols":9
[ModuleOutput] "ColumnTypes":System.Int32,3,System.Double,5,System.String,1
[ModuleOutput] }
```

Farther down the page is more detailed information on the columns, which will look something like the following.

```
[ModuleOutput] [1] "Loading variable port1..."  
[ModuleOutput]  
[ModuleOutput] 'data.frame': 228 obs. of 9 variables:  
[ModuleOutput]  
[ModuleOutput] $ Column 0 : int 1 2 3 4 5 6 7 8 9 10 ...  
[ModuleOutput]  
[ModuleOutput] $ Year.Month : num 1995 1995 1995 1995 1995 ...  
[ModuleOutput]  
[ModuleOutput] $ Month.Number : int 1 2 3 4 5 6 7 8 9 10 ...  
[ModuleOutput]  
[ModuleOutput] $ Year : int 1995 1995 1995 1995 1995 1995 1995 1995 1995 ...  
[ModuleOutput]  
[ModuleOutput] $ Month : chr "Jan" "Feb" "Mar" "Apr" ...  
[ModuleOutput]  
[ModuleOutput] $ Cottagecheese.Prod: num 4.37 3.69 4.54 4.28 4.47 ...  
[ModuleOutput]  
[ModuleOutput] $ Icecream.Prod : num 51.6 56.1 68.5 65.7 73.7 ...  
[ModuleOutput]  
[ModuleOutput] $ Milk.Prod : num 2.11 1.93 2.16 2.13 2.23 ...  
[ModuleOutput]  
[ModuleOutput] $ N.CA.Fat.Price : num 0.98 0.892 0.892 0.897 0.897 ...
```

These results are mostly as expected, with 228 observations and 9 columns in the dataframe. We can see the column names, the R data type and a sample of each column.

NOTE

This same printed output is conveniently available from the R Device output of the [Execute R Script](#) module. We will discuss the outputs of the [Execute R Script](#) module in the next section.

Dataset2

The behavior of the Dataset2 input is identical to that of Dataset1. Using this input you can pass a second rectangular table of data into your R code. The function `maml.mapInputPort(2)`, with the argument 2, is used to pass this data.

Execute R Script outputs

Output a dataframe

You can output the contents of an R dataframe as a rectangular table through the Result Dataset1 port by using the `maml.mapOutputPort()` function. In our simple R script this is performed by the following line.

```
maml.mapOutputPort('cadairydata')
```

After running the experiment, click on the Result Dataset1 output port and then click on **Visualize**. You should see something like Figure 6.

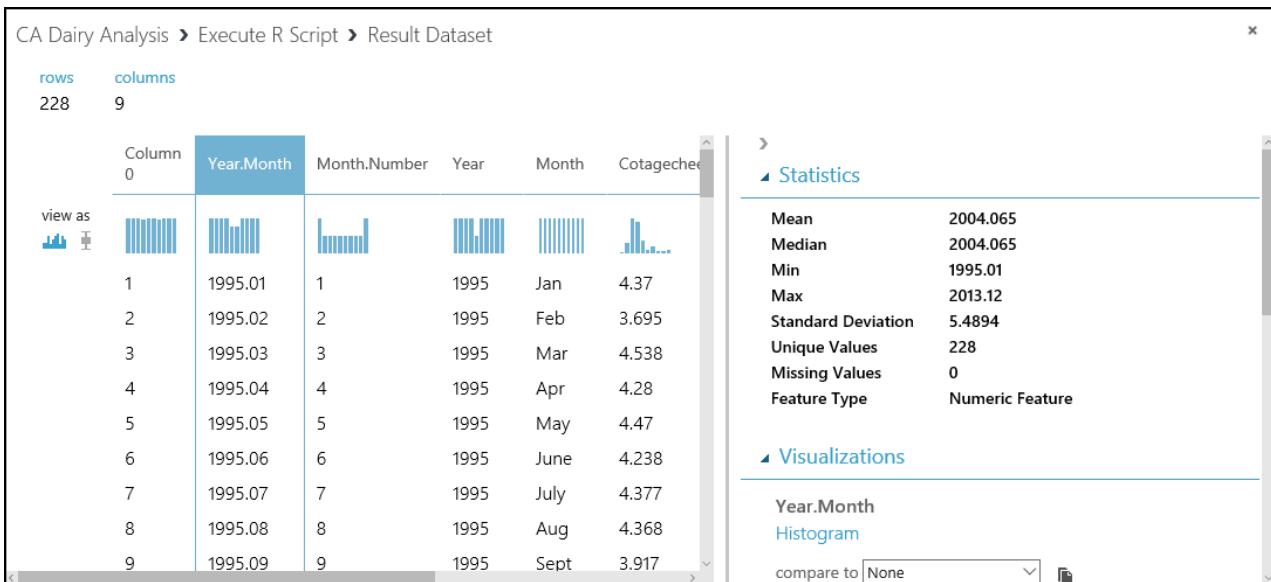


Figure 6. The visualization of the output of the California dairy data.

This output looks identical to the input, exactly as we expected.

R Device output

The Device output of the [Execute R Script](#) module contains messages and graphics output. Both standard output and standard error messages from R are sent to the R Device output port.

To view the R Device output, click on the port and then on **Visualize**. We see the standard output and standard error from the R script in Figure 7.

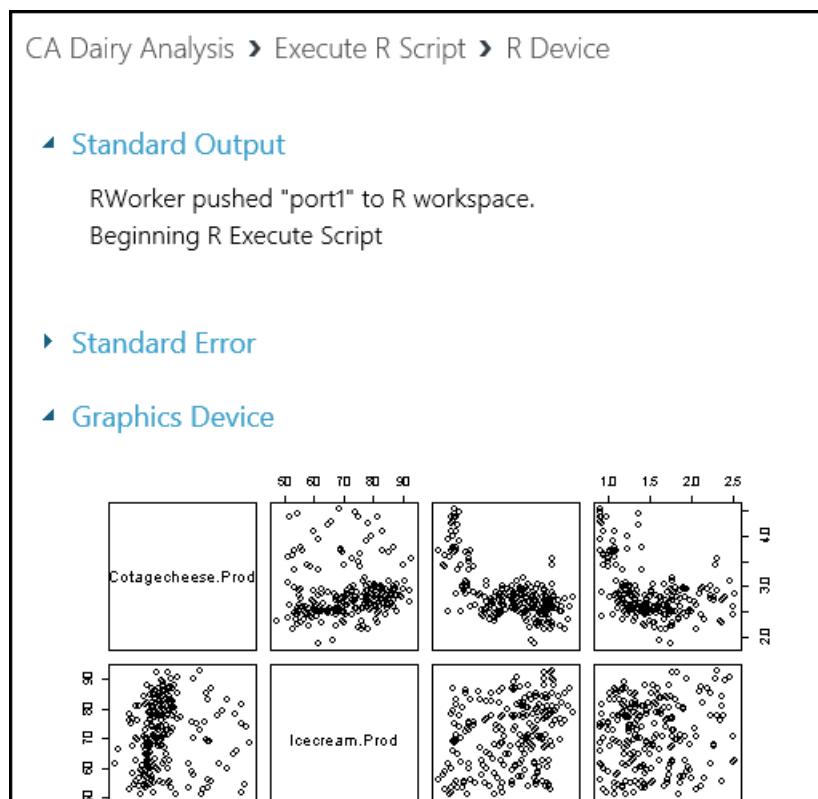


Figure 7. Standard output and standard error from the R Device port.

Scrolling down we see the graphics output from our R script in Figure 8.

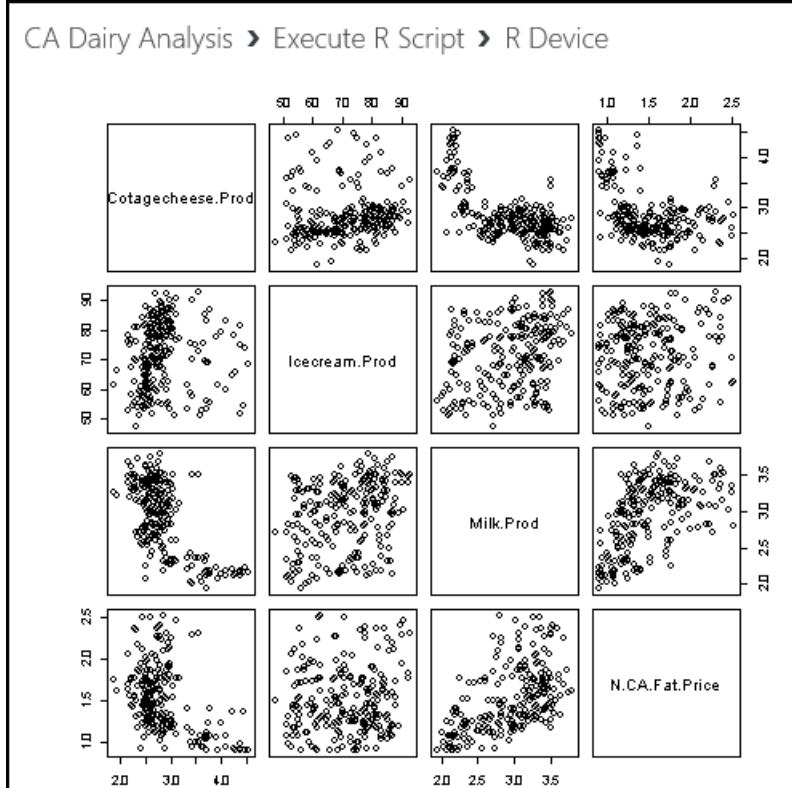


Figure 8. Graphics output from the R Device port.

Data filtering and transformation

In this section we will perform some basic data filtering and transformation operations on the California dairy data. By the end of this section we will have data in a format suitable for building an analytic model.

More specifically, in this section we will perform several common data cleaning and transformation tasks: type transformation, filtering on dataframes, adding new computed columns, and value transformations. This background should help you deal with the many variations encountered in real-world problems.

The complete R code for this section is available in the zip file you downloaded earlier.

Type transformations

Now that we can read the California dairy data into the R code in the [Execute R Script](#) module, we need to ensure that the data in the columns has the intended type and format.

R is a dynamically typed language, which means that data types are coerced from one to another as required. The atomic data types in R include numeric, logical and character. The factor type is used to compactly store categorical data. You can find much more information on data types in the references in [Appendix B - Further reading](#).

When tabular data is read into R from an external source, it is always a good idea to check the resulting types in the columns. You may want a column of type character, but in many cases this will show up as factor or vice versa. In other cases a column you think should be numeric is represented by character data, e.g. '1.23' rather than 1.23 as a floating point number.

Fortunately, it is easy to convert one type to another, as long as mapping is possible. For example, you cannot convert 'Nevada' into a numeric value, but you can convert it to a factor (categorical variable). As another example, you can convert a numeric 1 into a character '1' or a factor.

The syntax for any of these conversions is simple: `as.datatype()`. These type conversion functions include the following.

- `as.numeric()`
- `as.character()`

- `as.logical()`
- `as.factor()`

Looking at the data types of the columns we input in the previous section: all columns are of type numeric, except for the column labeled 'Month', which is of type character. Let's convert this to a factor and test the results.

I have deleted the line that created the scatterplot matrix and added a line converting the 'Month' column to a factor. In my experiment I will just cut and paste the R code into the code window of the [Execute R Script](#) Module. You could also update the zip file and upload it to Azure Machine Learning Studio, but this takes several steps.

```
## Only one of the following two lines should be used
## If running in Machine Learning Studio, use the first line with maml.mapInputPort()
## If in RStudio, use the second line with read.csv()
cadairydata <- maml.mapInputPort(1)
# cadairydata <- read.csv("cadairydata.csv", header = TRUE, stringsAsFactors = FALSE)
## Ensure the coding is consistent and convert column to a factor
cadairydata$Month <- as.factor(cadairydata$Month)
str(cadairydata) # Check the result
## The following line should be executed only when running in
## Azure Machine Learning Studio
maml.mapOutputPort('cadairydata')
```

Let's execute this code and look at the output log for the R script. The relevant data from the log is shown in Figure 9.

```
[ModuleOutput] [1] "Loading variable port1..."
[ModuleOutput]
[ModuleOutput] 'data.frame': 228 obs. of 9 variables:
[ModuleOutput]
[ModuleOutput] $ Column 0      : int 1 2 3 4 5 6 7 8 9 10 ...
[ModuleOutput]
[ModuleOutput] $ Year.Month   : num 1995 1995 1995 1995 1995 ...
[ModuleOutput]
[ModuleOutput] $ Month.Number : int 1 2 3 4 5 6 7 8 9 10 ...
[ModuleOutput]
[ModuleOutput] $ Year        : int 1995 1995 1995 1995 1995 1995 1995 1995 ...
[ModuleOutput]
[ModuleOutput] $ Month       : Factor w/ 14 levels "Apr", "April", ... : 6 5 9 1 11 8 7 3 14 13 ...
[ModuleOutput]
[ModuleOutput] $ Cottagecheese.Prod: num 4.37 3.69 4.54 4.28 4.47 ...
[ModuleOutput]
[ModuleOutput] $ Icecream.Prod  : num 51.6 56.1 68.5 65.7 73.7 ...
[ModuleOutput]
[ModuleOutput] $ Milk.Prod    : num 2.11 1.93 2.16 2.13 2.23 ...
[ModuleOutput]
[ModuleOutput] $ N.CA.Fat.Price : num 0.98 0.892 0.892 0.897 0.897 ...
[ModuleOutput]
[ModuleOutput] [1] "Saving variable cadairydata ..."
[ModuleOutput]
[ModuleOutput] [1] "Saving the following item(s): .maml.oport1"
```

Figure 9. Summary of the dataframe with a factor variable.

The type for Month should now say '**Factor w/ 14 levels**'. This is a problem since there are only 12 months in the year. You can also check to see that the type in **Visualize** of the Result Dataset port is '**Categorical**'.

The problem is that the 'Month' column has not been coded systematically. In some cases a month is called April and in others it is abbreviated as Apr. We can solve this problem by trimming the string to 3 characters. The line of code now looks like the following:

```
## Ensure the coding is consistent and convert column to a factor
cadairydata$Month <- as.factor(substr(cadairydata$Month, 1, 3))
```

Rerun the experiment and view the output log. The expected results are shown in Figure 10.

```
[ModuleOutput] [1] "Loading variable port1..."
[ModuleOutput]
[ModuleOutput] 'data.frame': 228 obs. of 9 variables:
[ModuleOutput] $ Column 0      : int 1 2 3 4 5 6 7 8 9 10 ...
[ModuleOutput]
[ModuleOutput] $ Year.Month   : num 1995 1995 1995 1995 1995 ...
[ModuleOutput]
[ModuleOutput] $ Month.Number : int 1 2 3 4 5 6 7 8 9 10 ...
[ModuleOutput]
[ModuleOutput] $ Year        : int 1995 1995 1995 1995 1995 1995 1995 1995 ...
[ModuleOutput]
[ModuleOutput] $ Month       : Factor w/ 12 levels "Apr","Aug","Dec",...: 5 4 8 1 9 7 6 2 12 11 ...
[ModuleOutput]
[ModuleOutput] $ Cottagecheese.Prod: num 4.37 3.69 4.54 4.28 4.47 ...
[ModuleOutput]
[ModuleOutput] $ Icecream.Prod  : num 51.6 56.1 68.5 65.7 73.7 ...
[ModuleOutput]
[ModuleOutput] $ Milk.Prod    : num 2.11 1.93 2.16 2.13 2.23 ...
[ModuleOutput]
[ModuleOutput] $ N.CA.Fat.Price: num 0.98 0.892 0.892 0.897 0.897 ...
[ModuleOutput]
[ModuleOutput] [1] "Saving variable cadairydata ..."
[ModuleOutput]
[ModuleOutput] [1] "Saving the following item(s): .maml.oport1"
```

Figure 10. Summary of the dataframe with correct number of factor levels.

Our factor variable now has the desired 12 levels.

Basic data frame filtering

R dataframes support powerful filtering capabilities. Datasets can be subsetted by using logical filters on either rows or columns. In many cases, complex filter criteria will be required. The references in [Appendix B - Further reading](#) contain extensive examples of filtering dataframes.

There is one bit of filtering we should do on our dataset. If you look at the columns in the `cadairydata` dataframe, you will see two unnecessary columns. The first column just holds a row number, which is not very useful. The second column, `Year.Month`, contains redundant information. We can easily exclude these columns by using the following R code.

NOTE

From now on in this section, I will just show you the additional code I am adding in the [Execute R Script](#) module. I will add each new line **before** the `str()` function. I use this function to verify my results in Azure Machine Learning Studio.

I add the following line to my R code in the [Execute R Script](#) module.

```
# Remove two columns we do not need
cadairydata <- cadairydata[, c(-1, -2)]
```

Run this code in your experiment and check the result from the output log. These results are shown in Figure 11.

```

[ModuleOutput] [1] "Loading variable port1..."
[ModuleOutput]
[ModuleOutput] 'data.frame': 228 obs. of 7 variables:
[ModuleOutput] $ Month.Number : int 1 2 3 4 5 6 7 8 9 10 ...
[ModuleOutput]
[ModuleOutput] $ Year      : int 1995 1995 1995 1995 1995 1995 1995 1995 1995 ...
[ModuleOutput]
[ModuleOutput] $ Month     : Factor w/ 12 levels "Apr","Aug","Dec",...: 5 4 8 1 9 7 6 2 12 11 ...
[ModuleOutput]
[ModuleOutput] $ Cottagecheese.Prod: num 4.37 3.69 4.54 4.28 4.47 ...
[ModuleOutput]
[ModuleOutput] $ Icecream.Prod : num 51.6 56.1 68.5 65.7 73.7 ...
[ModuleOutput]
[ModuleOutput] $ Milk.Prod   : num 2.11 1.93 2.16 2.13 2.23 ...
[ModuleOutput]
[ModuleOutput] $ N.CA.Fat.Price : num 0.98 0.892 0.892 0.897 0.897 ...
[ModuleOutput]
[ModuleOutput] [1] "Saving variable cadairydata ..."
[ModuleOutput]
[ModuleOutput] [1] "Saving the following item(s): .maml.lopert1"

```

Figure 11. Summary of the dataframe with two columns removed.

Good news! We get the expected results.

Add a new column

To create time series models it will be convenient to have a column containing the months since the start of the time series. We will create a new column 'Month.Count'.

To help organize the code we will create our first simple function, `num.month()`. We will then apply this function to create a new column in the dataframe. The new code is as follows.

```

## Create a new column with the month count
## Function to find the number of months from the first
## month of the time series
num.month <- function(Year, Month) {
  ## Find the starting year
  min.year <- min(Year)

  ## Compute the number of months from the start of the time series
  12 * (Year - min.year) + Month - 1
}

## Compute the new column for the dataframe
cadairydata$Month.Count <- num.month(cadairydata$Year, cadairydata$Month.Number)

```

Now run the updated experiment and use the output log to view the results. These results are shown in Figure 12.

```

[ModuleOutput] [1] "Loading variable port1..."
[ModuleOutput]
[ModuleOutput] 'data.frame': 228 obs. of 8 variables:
[ModuleOutput] $ Month.Number : int 1 2 3 4 5 6 7 8 9 10 ...
[ModuleOutput]
[ModuleOutput] $ Year      : int 1995 1995 1995 1995 1995 1995 1995 1995 1995 ...
[ModuleOutput]
[ModuleOutput] $ Month     : Factor w/ 12 levels "Apr","Aug","Dec",...: 5 4 8 1 9 7 6 2 12 11 ...
[ModuleOutput]
[ModuleOutput] $ Cottagecheese.Prod: num 4.37 3.69 4.54 4.28 4.47 ...
[ModuleOutput]
[ModuleOutput] $ Icecream.Prod : num 51.6 56.1 68.5 65.7 73.7 ...
[ModuleOutput]
[ModuleOutput] $ Milk.Prod   : num 2.11 1.93 2.16 2.13 2.23 ...
[ModuleOutput]
[ModuleOutput] $ N.CA.Fat.Price : num 0.98 0.892 0.892 0.897 0.897 ...
[ModuleOutput]
[ModuleOutput] $ Month.Count : num 0 1 2 3 4 5 6 7 8 9 ...
[ModuleOutput]
[ModuleOutput] [1] "Saving variable cadairydata ..."
[ModuleOutput]
[ModuleOutput] [1] "Saving the following item(s): .maml.oport1"

```

Figure 12. Summary of the dataframe with the additional column.

It looks like everything is working. We have the new column with the expected values in our dataframe.

Value transformations

In this section we will perform some simple transformations on the values in some of the columns of our dataframe. The R language supports nearly arbitrary value transformations. The references in [Appendix B - Further Reading](#) contain extensive examples.

If you look at the values in the summaries of our dataframe you should see something odd here. Is more ice cream than milk produced in California? No, of course not, as this makes no sense, sad as this fact may be to some of us ice cream lovers. The units are different. The price is in units of US pounds, milk is in units of 1 M US pounds, ice cream is in units of 1,000 US gallons, and cottage cheese is in units of 1,000 US pounds. Assuming ice cream weighs about 6.5 pounds per gallon, we can easily do the multiplication to convert these values so they are all in equal units of 1,000 pounds.

For our forecasting model we use a multiplicative model for trend and seasonal adjustment of this data. A log transformation allows us to use a linear model, simplifying this process. We can apply the log transformation in the same function where the multiplier is applied.

In the following code, I define a new function, `log.transform()`, and apply it to the rows containing the numerical values. The R `Map()` function is used to apply the `log.transform()` function to the selected columns of the dataframe. `Map()` is similar to `apply()` but allows for more than one list of arguments to the function. Note that a list of multipliers supplies the second argument to the `log.transform()` function. The `na.omit()` function is used as a bit of cleanup to ensure we do not have missing or undefined values in the dataframe.

```

log.transform<- function(invec, multiplier = 1) {
  ## Function for the transformation, which is the log
  ## of the input value times a multiplier

  warningmessages <- c("ERROR: Non-numeric argument encountered in function log.transform",
    "ERROR: Arguments to function log.transform must be greater than zero",
    "ERROR: Argument multiplier to function log.transform must be a scalar",
    "ERROR: Invalid time series value encountered in function log.transform"
  )

  ## Check the input arguments
  if(!is.numeric(invec) | !is.numeric(multiplier)) {warning(warningmessages[1]); return(NA)}
  if(any(invec < 0.0) | any(multiplier < 0.0)) {warning(warningmessages[2]); return(NA)}
  if(length(multiplier) != 1) {[warning(warningmessages[3]); return(NA}]}

  ## Wrap the multiplication in tryCatch
  ## If there is an exception, print the warning message to
  ## standard error and return NA
  tryCatch(log(multiplier * invec),
    error = function(e){warning(warningmessages[4]); NA})
}

## Apply the transformation function to the 4 columns
## of the data frame with production data
multipliers <- list(1.0, 6.5, 1000.0, 1000.0)
cadairydata[, 4:7] <- Map(log.transform, cadairydata[, 4:7], multipliers)

## Get rid of any rows with NA values
cadairydata <- na.omit(cadairydata)

```

There is quite a bit happening in the `log.transform()` function. Most of this code is checking for potential problems with the arguments or dealing with exceptions, which can still arise during the computations. Only a few lines of this code actually do the computations.

The goal of the defensive programming is to prevent the failure of a single function that prevents processing from continuing. An abrupt failure of a long-running analysis can be quite frustrating for users. To avoid this situation, default return values must be chosen that will limit damage to downstream processing. A message is also produced to alert users that something has gone wrong.

If you are not used to defensive programming in R, all this code may seem a bit overwhelming. I will walk you through the major steps:

1. A vector of four messages is defined. These messages are used to communicate information about some of the possible errors and exceptions that can occur with this code.
2. I return a value of NA for each case. There are many other possibilities that might have fewer side effects. I could return a vector of zeroes, or the original input vector, for example.
3. Checks are run on the arguments to the function. In each case, if an error is detected, a default value is returned and a message is produced by the `warning()` function. I am using `warning()` rather than `stop()` as the latter will terminate execution, exactly what I am trying to avoid. Note that I have written this code in a procedural style, as in this case a functional approach seemed complex and obscure.
4. The log computations are wrapped in `tryCatch()` so that exceptions will not cause an abrupt halt to processing. Without `tryCatch()` most errors raised by R functions result in a stop signal, which does just that.

Execute this R code in your experiment and have a look at the printed output in the `output.log` file. You will now see the transformed values of the four columns in the log, as shown in Figure 13.

```

[ModuleOutput] [1] "Loading variable port1..."
[ModuleOutput]
[ModuleOutput] 'data.frame': 228 obs. of 8 variables:
[ModuleOutput]
[ModuleOutput] $ Month.Number : int 1 2 3 4 5 6 7 8 9 10 ...
[ModuleOutput]
[ModuleOutput] $ Year      : int 1995 1995 1995 1995 1995 1995 1995 1995 1995 ...
[ModuleOutput]
[ModuleOutput] $ Month     : Factor w/ 12 levels "Apr","Aug","Dec",...: 5 4 8 1 9 7 6 2 12 11 ...
[ModuleOutput]
[ModuleOutput] $ Cottagecheese.Prod: num 1.47 1.31 1.51 1.45 1.5 ...
[ModuleOutput]
[ModuleOutput] $ Icecream.Prod : num 5.82 5.9 6.1 6.06 6.17 ...
[ModuleOutput]
[ModuleOutput] $ Milk.Prod   : num 7.66 7.57 7.68 7.66 7.71 ...
[ModuleOutput]
[ModuleOutput] $ N.CA.Fat.Price : num 6.89 6.79 6.79 6.8 6.8 ...
[ModuleOutput]
[ModuleOutput] $ Month.Count : num 0 1 2 3 4 5 6 7 8 9 ...
[ModuleOutput]
[ModuleOutput] [1] "Saving variable cadairydata ..."
[ModuleOutput]
[ModuleOutput] [1] "Saving the following item(s): .maml.oport1"

```

Figure 13. Summary of the transformed values in the dataframe.

We see the values have been transformed. Milk production now greatly exceeds all other dairy product production, recalling that we are now looking at a log scale.

At this point our data is cleaned up and we are ready for some modeling. Looking at the visualization summary for the Result Dataset output of our [Execute R Script](#) module, you will see the 'Month' column is 'Categorical' with 12 unique values, again, just as we wanted.

Time series objects and correlation analysis

In this section we will explore a few basic R time series objects and analyze the correlations between some of the variables. Our goal is to output a dataframe containing the pairwise correlation information at several lags.

The complete R code for this section is in the zip file you downloaded earlier.

Time series objects in R

As already mentioned, time series are a series of data values indexed by time. R time series objects are used to create and manage the time index. There are several advantages to using time series objects. Time series objects free you from the many details of managing the time series index values that are encapsulated in the object. In addition, time series objects allow you to use the many time series methods for plotting, printing, modeling, etc.

The POSIXct time series class is commonly used and is relatively simple. This time series class measures time from the start of the epoch, January 1, 1970. We will use POSIXct time series objects in this example. Other widely used R time series object classes include zoo and xts, extensible time series.

Time series object example

Let's get started with our example. Drag and drop a **new** [Execute R Script](#) module into your experiment. Connect the Result Dataset1 output port of the existing [Execute R Script](#) module to the Dataset1 input port of the new [Execute R Script](#) module.

As I did for the first examples, as we progress through the example, at some points I will show only the incremental additional lines of R code at each step.

Reading the dataframe

As a first step, let's read in a dataframe and make sure we get the expected results. The following code should do

the job.

```
# Comment the following if using RStudio
cadairydata <- maml.mapInputPort(1)
str(cadairydata) # Check the results
```

Now, run the experiment. The log of the new Execute R Script shape should look like Figure 14.

```
[ModuleOutput] [1] "Loading variable port1..."
[ModuleOutput]
[ModuleOutput] 'data.frame': 228 obs. of 8 variables:
[ModuleOutput] $ Month.Number : int 1 2 3 4 5 6 7 8 9 10 ...
[ModuleOutput]
[ModuleOutput] $ Year      : int 1995 1995 1995 1995 1995 1995 1995 1995 1995 ...
[ModuleOutput]
[ModuleOutput] $ Month     : Factor w/ 12 levels "Apr","Aug","Dec",...: 5 4 8 1 9 7 6 2 12 11 ...
[ModuleOutput]
[ModuleOutput] $ Cottagecheese.Prod: num 1.47 1.31 1.51 1.45 1.5 ...
[ModuleOutput]
[ModuleOutput] $ Icecream.Prod : num 5.82 5.9 6.1 6.06 6.17 ...
[ModuleOutput]
[ModuleOutput] $ Milk.Prod   : num 7.66 7.57 7.68 7.66 7.71 ...
[ModuleOutput]
[ModuleOutput] $ N.CA.Fat.Price : num 6.89 6.79 6.79 6.8 6.8 ...
[ModuleOutput]
[ModuleOutput] $ Month.Count : num 0 1 2 3 4 5 6 7 8 9 ...
```

Figure 14. Summary of the dataframe in the Execute R Script module.

This data is of the expected types and format. Note that the 'Month' column is of type factor and has the expected number of levels.

Creating a time series object

We need to add a time series object to our dataframe. Replace the current code with the following, which adds a new column of class POSIXct.

```
# Comment the following if using RStudio
cadairydata <- maml.mapInputPort(1)

## Create a new column as a POSIXct object
Sys.setenv(TZ = "PST8PDT")
cadairydata$Time <- as.POSIXct(strptime(paste(as.character(cadairydata$Year), "-", as.character(cadairydata$Month.Number), "-01 00:00:00", sep = ""), "%Y-%m-%d %H:%M:%S"))

str(cadairydata) # Check the results
```

Now, check the log. It should look like Figure 15.

```
[ModuleOutput] [1] "Loading variable port1..."  

[ModuleOutput]  

[ModuleOutput] 'data.frame': 228 obs. of 9 variables:  

[ModuleOutput]  

[ModuleOutput] $ Month.Number : int 1 2 3 4 5 6 7 8 9 10 ...  

[ModuleOutput]  

[ModuleOutput] $ Year      : int 1995 1995 1995 1995 1995 1995 1995 1995 1995 1995 ...  

[ModuleOutput]  

[ModuleOutput] $ Month     : Factor w/ 12 levels "Apr","Aug","Dec",...: 5 4 8 1 9 7 6 2 12 11 ...  

[ModuleOutput]  

[ModuleOutput] $ Cottagecheese.Prod: num 1.47 1.31 1.51 1.45 1.5 ...  

[ModuleOutput]  

[ModuleOutput] $ Icecream.Prod  : num 5.82 5.9 6.1 6.06 6.17 ...  

[ModuleOutput]  

[ModuleOutput] $ Milk.Prod    : num 7.66 7.57 7.68 7.66 7.71 ...  

[ModuleOutput]  

[ModuleOutput] $ N.CA.Fat.Price : num 6.89 6.79 6.79 6.8 6.8 ...  

[ModuleOutput]  

[ModuleOutput] $ Month.Count : num 0 1 2 3 4 5 6 7 8 9 ...  

[ModuleOutput]  

[ModuleOutput] $ Time       : POSIXct, format: "1995-01-01" "1995-02-01" ...
```

Figure 15. Summary of the dataframe with a time series object.

We can see from the summary that the new column is in fact of class `POSIXct`.

Exploring and transforming the data

Let's explore some of the variables in this dataset. A scatterplot matrix is a good way to produce a quick look. I am replacing the `str()` function in the previous R code with the following line.

```
pairs(~ Cottagecheese.Prod + Icecream.Prod + Milk.Prod + N.CA.Fat.Price, data = cadairydata, main = "Pairwise Scatterplots of dairy time series")
```

Run this code and see what happens. The plot produced at the R Device port should look like Figure 16.

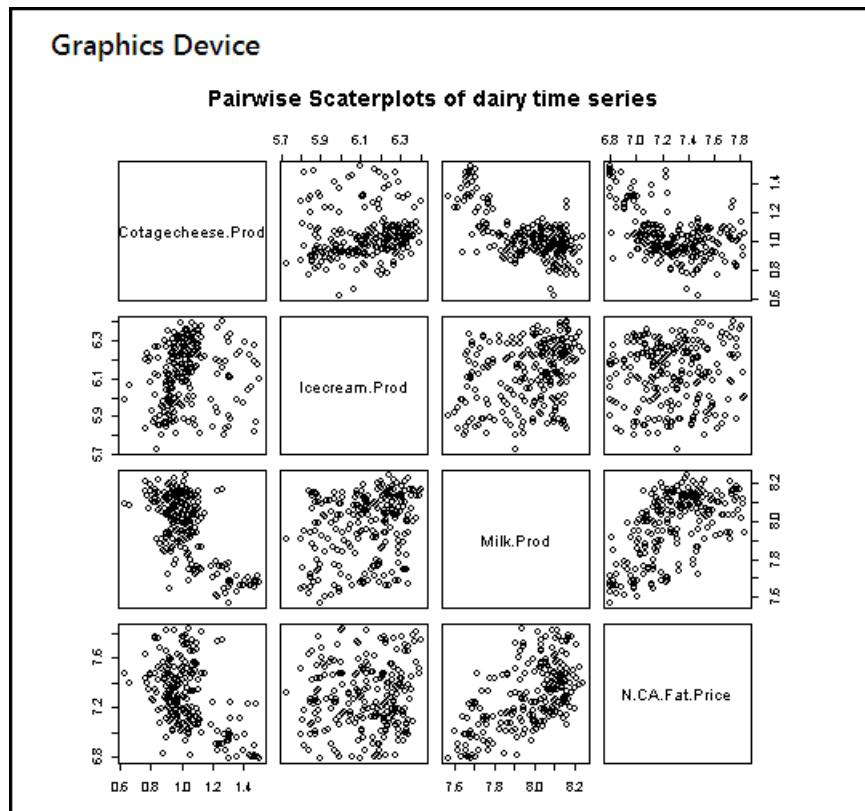


Figure 16. Scatterplot matrix of selected variables.

There is some odd-looking structure in the relationships between these variables. Perhaps this arises from trends

in the data and from the fact that we have not standardized the variables.

Correlation analysis

To perform correlation analysis we need to both de-trend and standardize the variables. We could simply use the R `scale()` function, which both centers and scales variables. This function might well run faster. However, I want to show you an example of defensive programming in R.

The `ts.detrend()` function shown below performs both of these operations. The following two lines of code de-trend the data and then standardize the values.

```
ts.detrend <- function(ts, Time, min.length = 3){  
  ## Function to de-trend and standardize a time series  
  
  ## Define some messages if they are NULL  
  messages <- c('ERROR: ts.detrend requires arguments ts and Time to have the same length',  
            'ERROR: ts.detrend requires argument ts to be of type numeric',  
            paste('WARNING: ts.detrend has encountered a time series with length less than', as.character(min.length)),  
            'ERROR: ts.detrend has encountered a Time argument not of class POSIXct',  
            'ERROR: Detrend regression has failed in ts.detrend',  
            'ERROR: Exception occurred in ts.detrend while standardizing time series in function ts.detrend'  
  )  
  # Create a vector of zeros to return as a default in some cases  
  zerovec <- rep(length(ts), 0.0)  
  
  # The input arguments are not of the same length, return ts and quit  
  if(length(Time) != length(ts)) {warning(messages[1]); return(ts)}  
  
  # If the ts is not numeric, just return a zero vector and quit  
  if(!is.numeric(ts)) {warning(messages[2]); return(zerovec)}  
  
  # If the ts is too short, just return it and quit  
  if((ts.length <- length(ts)) < min.length) {warning(messages[3]); return(ts)}  
  
  ## Check that the Time variable is of class POSIXct  
  if(class(cadairydata$Time)[1] != "POSIXct") {warning(messages[4]); return(ts)}  
  
  ## De-trend the time series by using a linear model  
  ts.frame <- data.frame(ts = ts, Time = Time)  
  tryCatch({ts <- ts - fitted(lm(ts ~ Time, data = ts.frame))},  
           error = function(e){warning(messages[5]); zerovec})  
  
  tryCatch( {stdev <- sqrt(sum((ts - mean(ts))^2))/(ts.length - 1)  
            ts <- ts/stdev},  
           error = function(e){warning(messages[6]); zerovec})  
  
  ts  
}  
## Apply the detrend.ts function to the variables of interest  
df.detrend <- data.frame(lapply(cadairydata[, 4:7], ts.detrend, cadairydata$Time))  
  
## Plot the results to look at the relationships  
pairs(~ Cotagecheese.Prod + Icecream.Prod + Milk.Prod + N.CA.Fat.Price, data = df.detrend, main = "Pairwise Scatterplots of detrended standardized time series")
```

There is quite a bit happening in the `ts.detrend()` function. Most of this code is checking for potential problems with the arguments or dealing with exceptions, which can still arise during the computations. Only a few lines of this code actually do the computations.

We have already discussed an example of defensive programming in [Value transformations](#). Both computation blocks are wrapped in `tryCatch()`. For some errors it makes sense to return the original input vector, and in other cases, I return a vector of zeros.

Note that the linear regression used for de-trending is a time series regression. The predictor variable is a time

series object.

Once `ts.detrend()` is defined we apply it to the variables of interest in our dataframe. We must coerce the resulting list created by `lapply()` to data dataframe by using `as.data.frame()`. Because of defensive aspects of `ts.detrend()`, failure to process one of the variables will not prevent correct processing of the others.

The final line of code creates a pairwise scatterplot. After running the R code, the results of the scatterplot are shown in Figure 17.

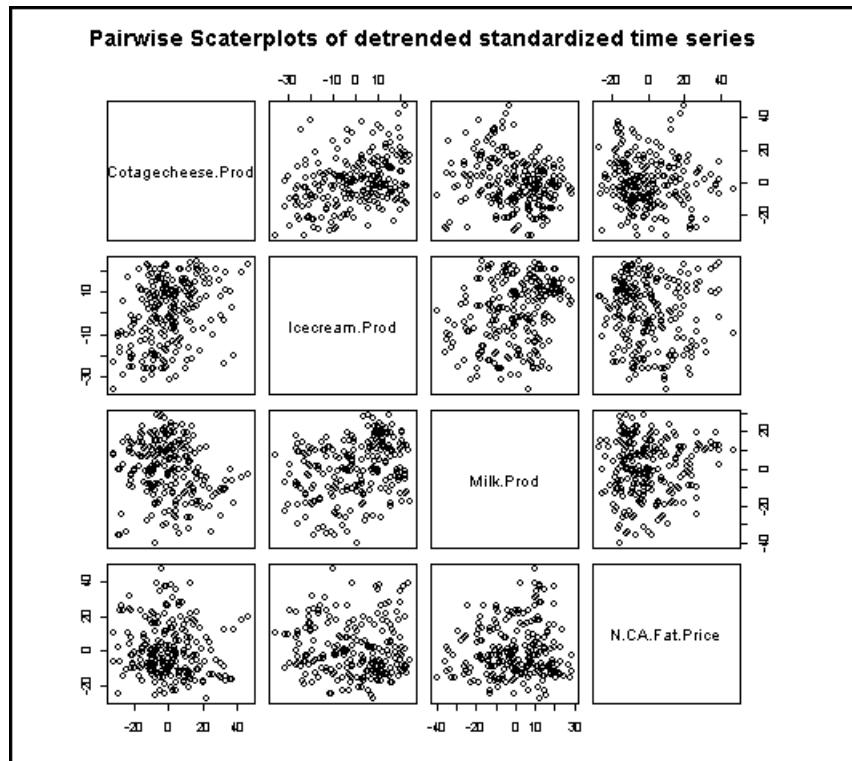


Figure 17. Pairwise scatterplot of de-trended and standardized time series.

You can compare these results to those shown in Figure 16. With the trend removed and the variables standardized, we see a lot less structure in the relationships between these variables.

The code to compute the correlations as R ccf objects is as follows.

```
## A function to compute pairwise correlations from a
## list of time series value vectors
pair.cor <- function(pair.ind, ts.list, lag.max = 1, plot = FALSE){
  ccf(ts.list[[pair.ind[1]]], ts.list[[pair.ind[2]]], lag.max = lag.max, plot = plot)
}

## A list of the pairwise indices
corpairs <- list(c(1,2), c(1,3), c(1,4), c(2,3), c(2,4), c(3,4))

## Compute the list of ccf objects
cadairycorrelations <- lapply(corpairs, pair.cor, df.detrend)

cadairycorrelations
```

Running this code produces the log shown in Figure 18.

```

[ModuleOutput] Loading objects:
[ModuleOutput] port1
[ModuleOutput] [1] "Loading variable port1..."
[ModuleOutput] [[1]]
[ModuleOutput]
[ModuleOutput]
[ModuleOutput] Autocorrelations of series 'X', by lag
[ModuleOutput]
[ModuleOutput]
[ModuleOutput] -1 0 1
[ModuleOutput] 0.148 0.358 0.317
[ModuleOutput]
[ModuleOutput]
[ModuleOutput] [[2]]
[ModuleOutput]
[ModuleOutput]
[ModuleOutput]
[ModuleOutput] Autocorrelations of series 'X', by lag
[ModuleOutput]
[ModuleOutput]
[ModuleOutput] -1 0 1
[ModuleOutput] -0.395 -0.186 -0.238
[ModuleOutput]
[ModuleOutput]
[ModuleOutput]
[ModuleOutput] [[3]]
[ModuleOutput]
[ModuleOutput]
[ModuleOutput]
[ModuleOutput] Autocorrelations of series 'X', by lag
[ModuleOutput]
[ModuleOutput]
[ModuleOutput] -1 0 1
[ModuleOutput] -0.059 -0.089 -0.127
[ModuleOutput]
[ModuleOutput]
[ModuleOutput]
[ModuleOutput] [[4]]
[ModuleOutput]
[ModuleOutput]
[ModuleOutput]
[ModuleOutput] Autocorrelations of series 'X', by lag
[ModuleOutput]
[ModuleOutput]
[ModuleOutput] -1 0 1
[ModuleOutput] 0.140 0.294 0.293
[ModuleOutput]
[ModuleOutput]
[ModuleOutput] [[5]]
[ModuleOutput]
[ModuleOutput]
[ModuleOutput]
[ModuleOutput] Autocorrelations of series 'X', by lag
[ModuleOutput]
[ModuleOutput]
[ModuleOutput] -1 0 1
[ModuleOutput] -0.002 -0.074 -0.124

```

Figure 18. List of ccf objects from the pairwise correlation analysis.

There is a correlation value for each lag. None of these correlation values is large enough to be significant. We can therefore conclude that we can model each variable independently.

Output a dataframe

We have computed the pairwise correlations as a list of R ccf objects. This presents a bit of a problem as the Result Dataset output port really requires a dataframe. Further, the ccf object is itself a list and we want only the values in the first element of this list, the correlations at the various lags.

The following code extracts the lag values from the list of ccf objects, which are themselves lists.

```

df.correlations <- data.frame(do.call(rbind, lapply(cadairycorrelations, "[[, 1])))

c.names <- c("correlation pair", "-1 lag", "0 lag", "+1 lag")
r.names <- c("Corr Cot Cheese - Ice Cream",
            "Corr Cot Cheese - Milk Prod",
            "Corr Cot Cheese - Fat Price",
            "Corr Ice Cream- Mik Prod",
            "Corr Ice Cream- Fat Price",
            "Corr Milk Prod - Fat Price")

## Build a dataframe with the row names column and the
## correlation data frame and assign the column names
outframe <- cbind(r.names, df.correlations)
colnames(outframe) <- c.names
outframe

## WARNING!
## The following line works only in Azure Machine Learning
## When running in RStudio, this code will result in an error
#maml.mapOutputPort('outframe')

```

The first line of code is a bit tricky, and some explanation may help you understand it. Working from the inside out we have the following:

1. The '`[[`' operator with the argument '`1`' selects the vector of correlations at the lags from the first element of the `ccf` object list.
2. The `do.call()` function applies the `rbind()` function over the elements of the list returns by `lapply()`.
3. The `data.frame()` function coerces the result produced by `do.call()` to a dataframe.

Note that the row names are in a column of the dataframe. Doing so preserves the row names when they are output from the [Execute R Script](#).

Running the code produces the output shown in Figure 19 when I **Visualize** the output at the Result Dataset port. The row names are in the first column, as intended.

CA Dairy Analysis > Execute R Script > Result Dataset				
rows	columns			
6	4			
	correlation pair	-1 lag	0 lag	
view as			+1 lag	
	Corr Cot Cheese - Ice Cream	0.147775	0.358106	0.317
	Corr Cot Cheese - Milk Prod	-0.395444	-0.185777	-0.238486
	Corr Cot Cheese - Fat Price	-0.059431	-0.08864	-0.127372
	Corr Ice Cream - Mik Prod	0.139825	0.294231	0.293429
	Corr Ice Cream - Fat Price	-0.001878	-0.07351	-0.124443
	Corr Milk Prod - Fat Price	0.032936	0.075419	0.052024

Figure 19. Results output from the correlation analysis.

Time series example: seasonal forecasting

Our data is now in a form suitable for analysis, and we have determined there are no significant correlations between the variables. Let's move on and create a time series forecasting model. Using this model we will forecast California milk production for the 12 months of 2013.

Our forecasting model will have two components, a trend component and a seasonal component. The complete forecast is the product of these two components. This type of model is known as a multiplicative model. The alternative is an additive model. We have already applied a log transformation to the variables of interest, which makes this analysis tractable.

The complete R code for this section is in the zip file you downloaded earlier.

Creating the dataframe for analysis

Start by adding a new [Execute R Script](#) module to your experiment. Connect the **Result Dataset** output of the existing [Execute R Script](#) module to the **Dataset1** input of the new module. The result should look something like Figure 20.

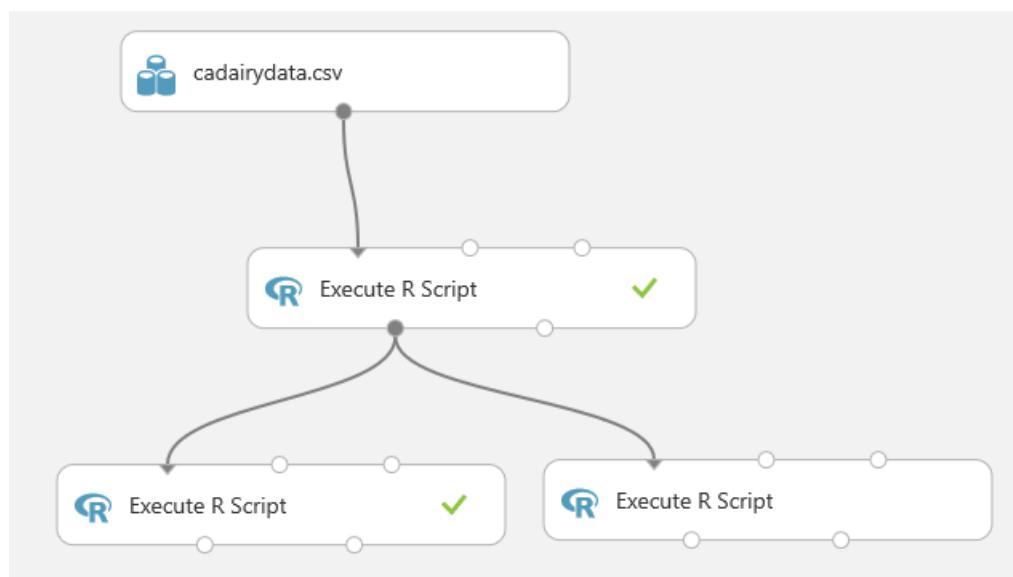


Figure 20. The experiment with the new Execute R Script module added.

As with the correlation analysis we just completed, we need to add a column with a POSIXct time series object. The following code will do just this.

```
# If running in Machine Learning Studio, uncomment the first line with maml.mapInputPort()
cadairydata <- maml.mapInputPort(1)

## Create a new column as a POSIXct object
Sys.setenv(TZ = "PST8PDT")
cadairydata$Time <- as.POSIXct(strptime(paste(as.character(cadairydata$Year), "-", as.character(cadairydata$Month.Number), "-01 00:00:00", sep = ""), "%Y-%m-%d %H:%M:%S"))

str(cadairydata)
```

Run this code and look at the log. The result should look like Figure 21.

```
[ModuleOutput] [1] "Loading variable port1..."  

[ModuleOutput]  

[ModuleOutput] 'data.frame': 228 obs. of 9 variables:  

[ModuleOutput]  

[ModuleOutput] $ Month.Number : int 1 2 3 4 5 6 7 8 9 10 ...  

[ModuleOutput]  

[ModuleOutput] $ Year      : int 1995 1995 1995 1995 1995 1995 1995 1995 1995 1995 ...  

[ModuleOutput]  

[ModuleOutput] $ Month     : Factor w/ 12 levels "Apr","Aug","Dec",...: 5 4 8 1 9 7 6 2 12 11 ...  

[ModuleOutput]  

[ModuleOutput] $ Cottagecheese.Prod: num 1.47 1.31 1.51 1.45 1.5 ...  

[ModuleOutput]  

[ModuleOutput] $ Icecream.Prod : num 5.82 5.9 6.1 6.06 6.17 ...  

[ModuleOutput]  

[ModuleOutput] $ Milk.Prod   : num 7.66 7.57 7.68 7.66 7.71 ...  

[ModuleOutput]  

[ModuleOutput] $ N.CA.Fat.Price : num 6.89 6.79 6.79 6.8 6.8 ...  

[ModuleOutput]  

[ModuleOutput] $ Month.Count : num 0 1 2 3 4 5 6 7 8 9 ...  

[ModuleOutput]  

[ModuleOutput] $ Time       : POSIXct, format: "1995-01-01" "1995-02-01" ...
```

Figure 21. A summary of the dataframe.

With this result, we are ready to start our analysis.

Create a training dataset

With the dataframe constructed we need to create a training dataset. This data will include all of the observations except the last 12, of the year 2013, which is our test dataset. The following code subsets the dataframe and creates plots of the dairy production and price variables. I then create plots of the four production and price variables. An anonymous function is used to define some augments for plot, and then iterate over the list of the other two arguments with `Map()`. If you are thinking that a for loop would have worked fine here, you are correct. But, since R is a functional language I am showing you a functional approach.

```
cadairytrain <- cadairydata[1:216, ]  

Ylabs <- list("Log CA Cottage Cheese Production, 1000s lb",  

             "Log CA Ice Cream Production, 1000s lb",  

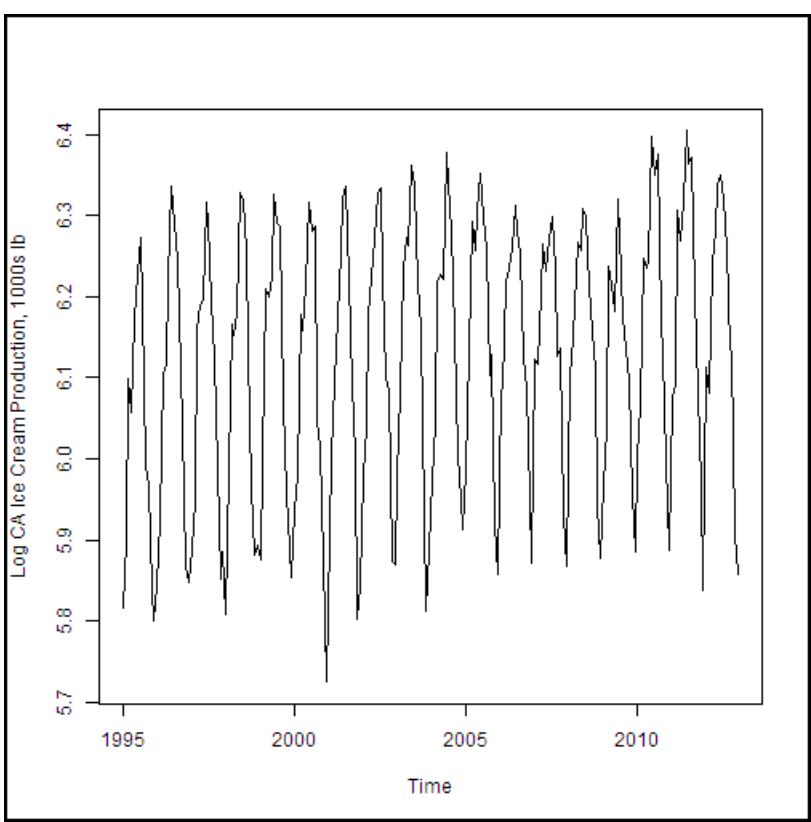
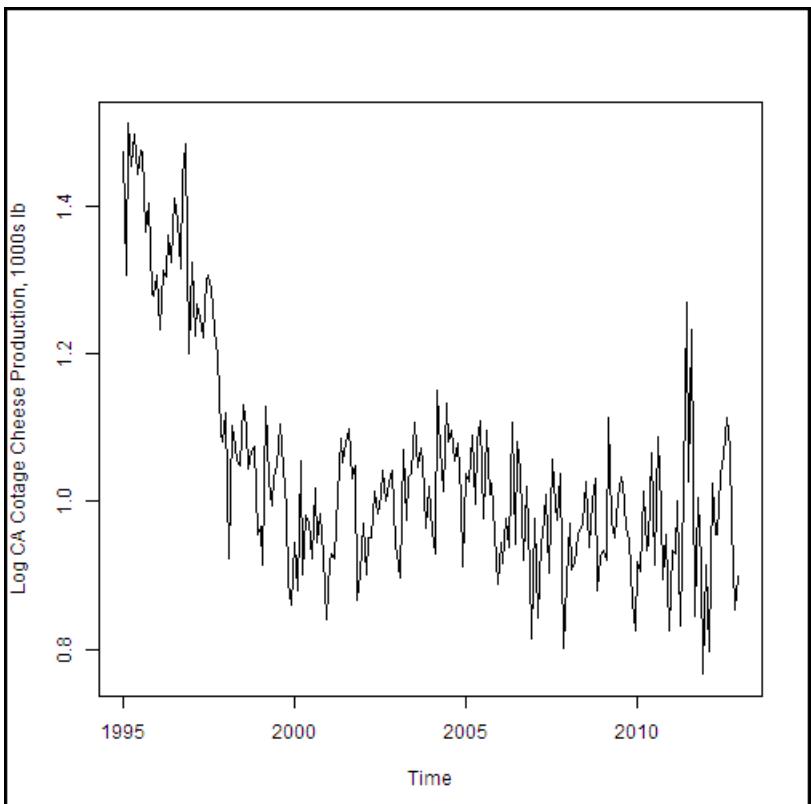
             "Log CA Milk Production 1000s lb",  

             "Log North CA Milk Milk Fat Price per 1000 lb")  

Map(function(y, Ylabs){plot(cadairytrain$Time, y, xlab = "Time", ylab = Ylabs, type = "l")}, cadairytrain[, 4:7], Ylabs)
```

Running the code produces the series of time series plots from the R Device output shown in Figure 22. Note that the time axis is in units of dates, a nice benefit of the time series plot method.



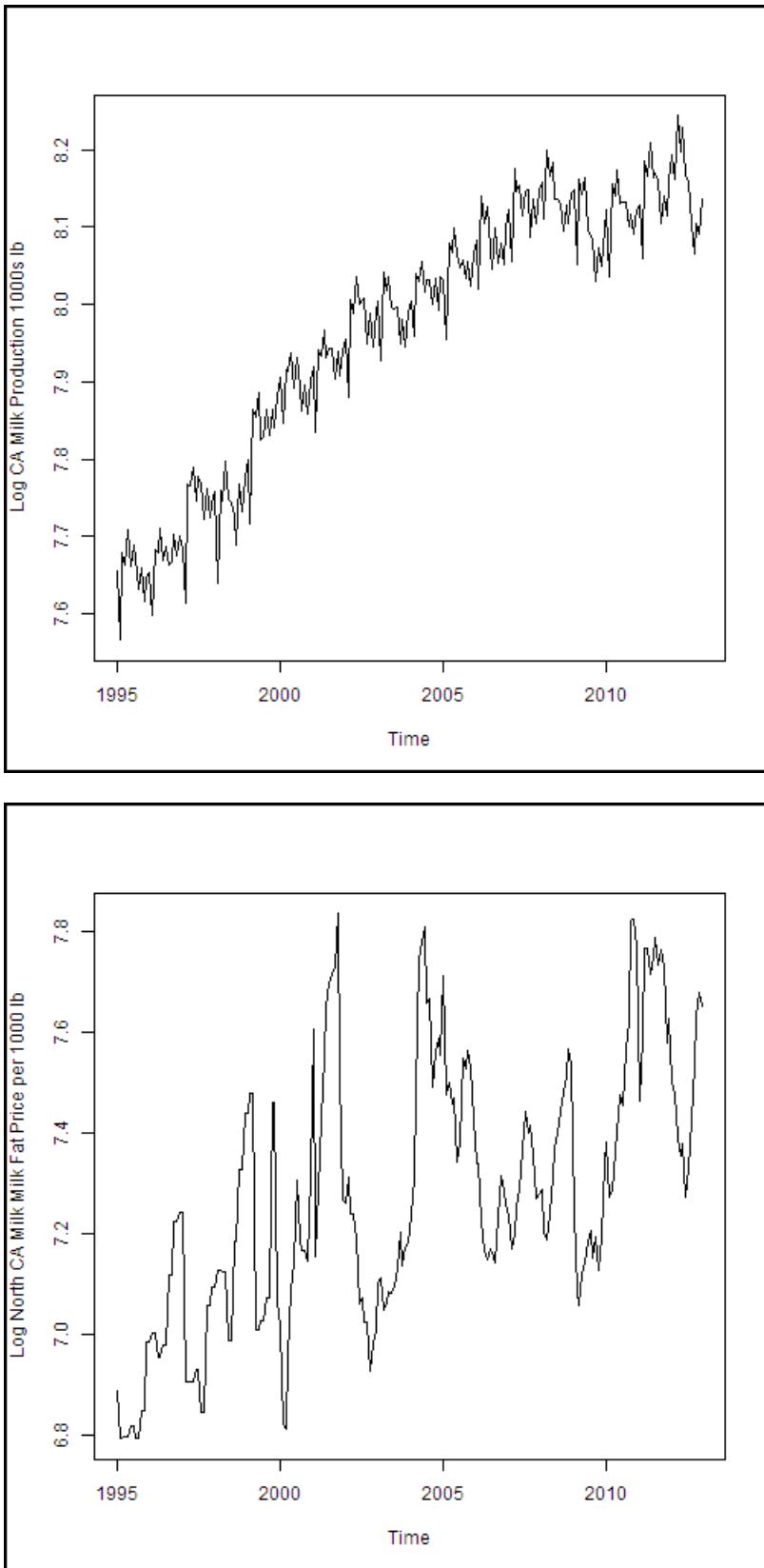


Figure 22. Time series plots of California dairy production and price data.

A trend model

Having created a time series object and having had a look at the data, let's start to construct a trend model for the California milk production data. We can do this with a time series regression. However, it is clear from the plot that we will need more than a slope and intercept to accurately model the observed trend in the training data.

Given the small scale of the data, I will build the model for trend in RStudio and then cut and paste the resulting model into Azure Machine Learning. RStudio provides an interactive environment for this type of interactive analysis.

As a first attempt, I will try a polynomial regression with powers up to 3. There is a real danger of over-fitting these kinds of models. Therefore, it is best to avoid high order terms. The `I()` function inhibits interpretation of the contents (interprets the contents 'as is') and allows you to write a literally interpreted function in a regression equation.

```
milk.lm <- lm(Milk.Prod ~ Time + I(Month.Count^2) + I(Month.Count^3), data = cadairytrain)
summary(milk.lm)
```

This generates the following.

```
##
## Call:
## lm(formula = Milk.Prod ~ Time + I(Month.Count^2) + I(Month.Count^3),
##   data = cadairytrain)
##
## Residuals:
##   Min     1Q Median     3Q    Max 
## -0.12667 -0.02730  0.00236  0.02943  0.10586
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 6.33e+00 1.45e-01 43.60 <2e-16 ***
## Time        1.63e-09 1.72e-10  9.47 <2e-16 ***
## I(Month.Count^2) -1.71e-06 4.89e-06 -0.35   0.726  
## I(Month.Count^3) -3.24e-08 1.49e-08 -2.17   0.031 *  
## ---    
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0418 on 212 degrees of freedom
## Multiple R-squared: 0.941, Adjusted R-squared: 0.94 
## F-statistic: 1.12e+03 on 3 and 212 DF, p-value: <2e-16
```

From P values ($\text{Pr}(>|t|)$) in this output, we can see that the squared term may not be significant. I will use the `update()` function to modify this model by dropping the squared term.

```
milk.lm <- update(milk.lm, . ~ . - I(Month.Count^2))
summary(milk.lm)
```

This generates the following.

```
##
## Call:
## lm(formula = Milk.Prod ~ Time + I(Month.Count^3), data = cadairytrain)
##
## Residuals:
##   Min     1Q Median     3Q    Max 
## -0.12597 -0.02659  0.00185  0.02963  0.10696
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 6.38e+00 4.07e-02 156.6 <2e-16 ***
## Time        1.57e-09 4.32e-11  36.3 <2e-16 ***
## I(Month.Count^3) -3.76e-08 2.50e-09 -15.1 <2e-16 ***
## ---    
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0417 on 213 degrees of freedom
## Multiple R-squared: 0.941, Adjusted R-squared: 0.94 
## F-statistic: 1.69e+03 on 2 and 213 DF, p-value: <2e-16
```

This looks better. All of the terms are significant. However, the $2e-16$ value is a default value, and should not be

taken too seriously.

As a sanity test, let's make a time series plot of the California dairy production data with the trend curve shown. I have added the following code in the Azure Machine Learning [Execute R Script](#) model (not RStudio) to create the model and make a plot. The result is shown in Figure 23.

```
milk.lm<- lm(Milk.Prod ~ Time + I(Month.Count^3), data = cadairytrain)

plot(cadairytrain$Time, cadairytrain$Milk.Prod, xlab = "Time", ylab = "Log CA Milk Production 1000s lb", type = "l")
lines(cadairytrain$Time, predict(milk.lm, cadairytrain), lty = 2, col = 2)
```

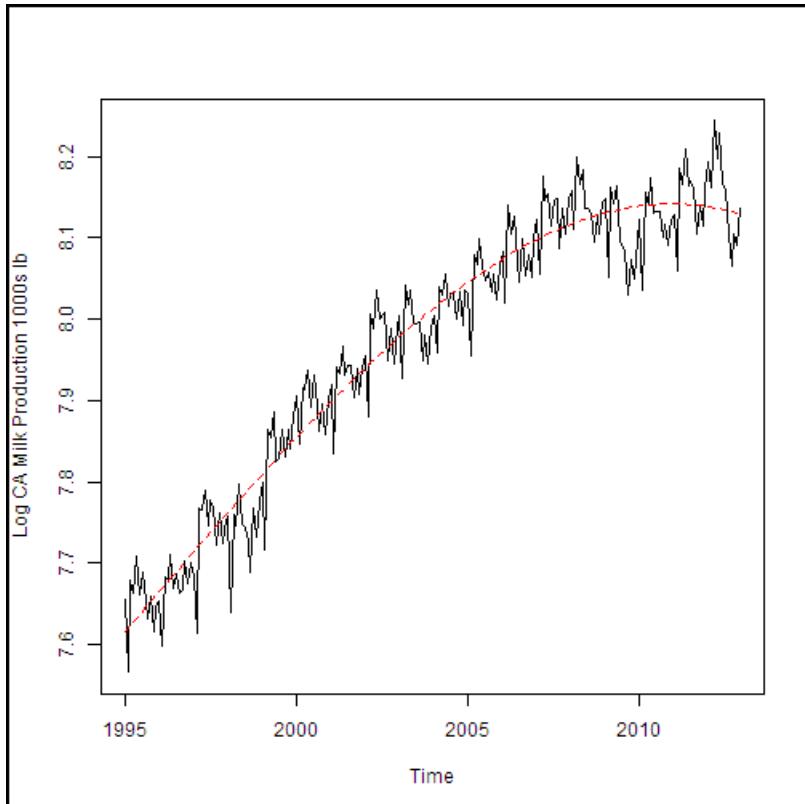


Figure 23. California milk production data with trend model shown.

It looks like the trend model fits the data fairly well. Further, there does not seem to be evidence of over-fitting, such as odd wiggles in the model curve.

Seasonal model

With a trend model in hand, we need to push on and include the seasonal effects. We will use the month of the year as a dummy variable in the linear model to capture the month-by-month effect. Note that when you introduce factor variables into a model, the intercept must not be computed. If you do not do this, the formula is over-specified and R will drop one of the desired factors but keep the intercept term.

Since we have a satisfactory trend model we can use the `update()` function to add the new terms to the existing model. The `-1` in the update formula drops the intercept term. Continuing in RStudio for the moment:

```
milk.lm2 <- update(milk.lm, . ~ . + Month - 1)
summary(milk.lm2)
```

This generates the following.

```

## 
## Call:
## lm(formula = Milk.Prod ~ Time + I(Month.Count^3) + Month - 1,
##   data = cadairytrain)
## 
## Residuals:
##   Min     1Q Median     3Q    Max 
## -0.06879 -0.01693  0.00346  0.01543  0.08726 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## Time        1.57e-09  2.72e-11  57.7 <2e-16 ***
## I(Month.Count^3) -3.74e-08  1.57e-09 -23.8 <2e-16 *** 
## MonthApr    6.40e+00  2.63e-02  243.3 <2e-16 *** 
## MonthAug    6.38e+00  2.63e-02  242.2 <2e-16 *** 
## MonthDec    6.38e+00  2.64e-02  241.9 <2e-16 *** 
## MonthFeb    6.31e+00  2.63e-02  240.1 <2e-16 *** 
## MonthJan    6.39e+00  2.63e-02  243.1 <2e-16 *** 
## MonthJul    6.39e+00  2.63e-02  242.6 <2e-16 *** 
## MonthJun    6.38e+00  2.63e-02  242.4 <2e-16 *** 
## MonthMar    6.42e+00  2.63e-02  244.2 <2e-16 *** 
## MonthMay    6.43e+00  2.63e-02  244.3 <2e-16 *** 
## MonthNov    6.34e+00  2.63e-02  240.6 <2e-16 *** 
## MonthOct    6.37e+00  2.63e-02  241.8 <2e-16 *** 
## MonthSep    6.34e+00  2.63e-02  240.6 <2e-16 *** 
## --- 
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 
## 
## Residual standard error: 0.0263 on 202 degrees of freedom 
## Multiple R-squared:  1, Adjusted R-squared:  1 
## F-statistic: 1.42e+06 on 14 and 202 DF, p-value: <2e-16

```

We see that the model no longer has an intercept term and has 12 significant month factors. This is exactly what we wanted to see.

Let's make another time series plot of the California dairy production data to see how well the seasonal model is working. I have added the following code in the Azure Machine Learning [Execute R Script](#) to create the model and make a plot.

```

milk.lm2 <- lm(Milk.Prod ~ Time + I(Month.Count^3) + Month - 1, data = cadairytrain)

plot(cadairytrain$Time, cadairytrain$Milk.Prod, xlab = "Time", ylab = "Log CA Milk Production 1000s lb", type = "l")
lines(cadairytrain$Time, predict(milk.lm2, cadairytrain), lty = 2, col = 2)

```

Running this code in Azure Machine Learning produces the plot shown in Figure 24.

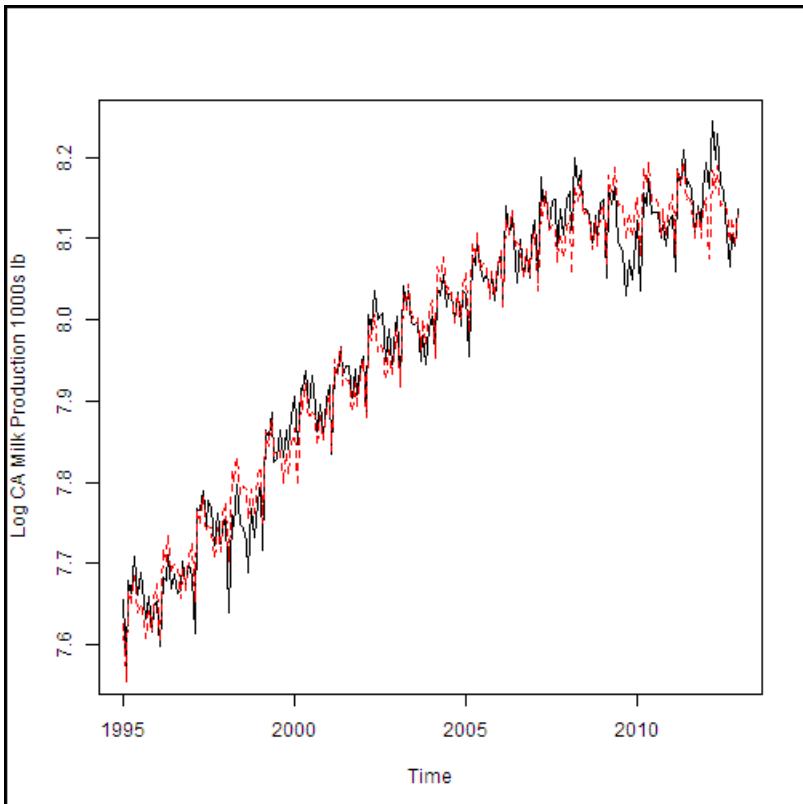


Figure 24. California milk production with model including seasonal effects.

The fit to the data shown in Figure 24 is rather encouraging. Both the trend and the seasonal effect (monthly variation) look reasonable.

As another check on our model, let's have a look at the residuals. The following code computes the predicted values from our two models, computes the residuals for the seasonal model, and then plots these residuals for the training data.

```
## Compute predictions from our models
predict1 <- predict(milk.lm, cadairydata)
predict2 <- predict(milk.lm2, cadairydata)

## Compute and plot the residuals
residuals <- cadairydata$Milk.Prod - predict2
plot(cadairytrain$Time, residuals[1:216], xlab = "Time", ylab = "Residuals of Seasonal Model")
```

The residual plot is shown in Figure 25.

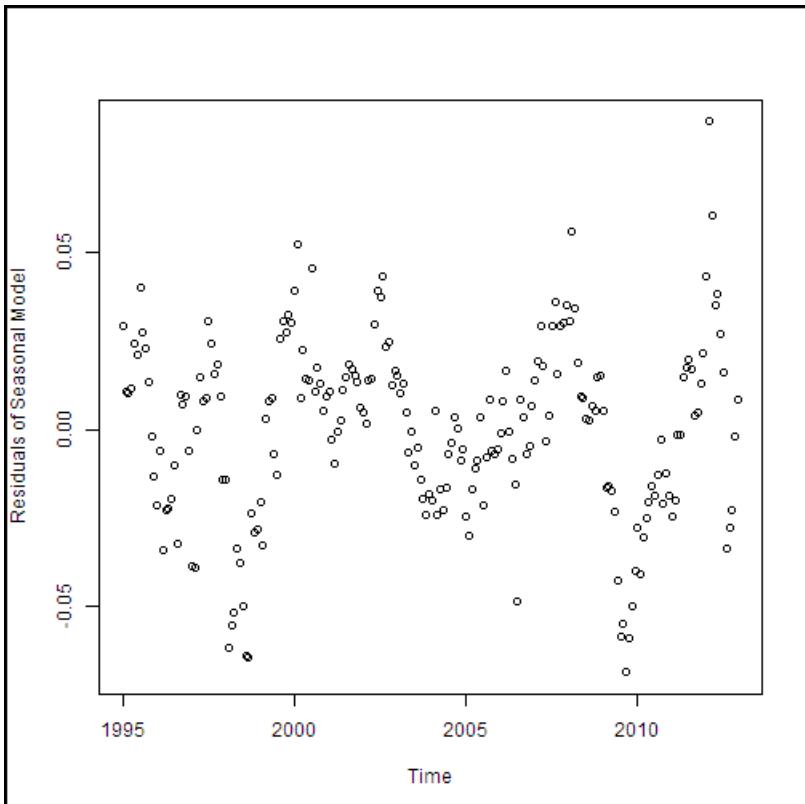


Figure 25. Residuals of the seasonal model for the training data.

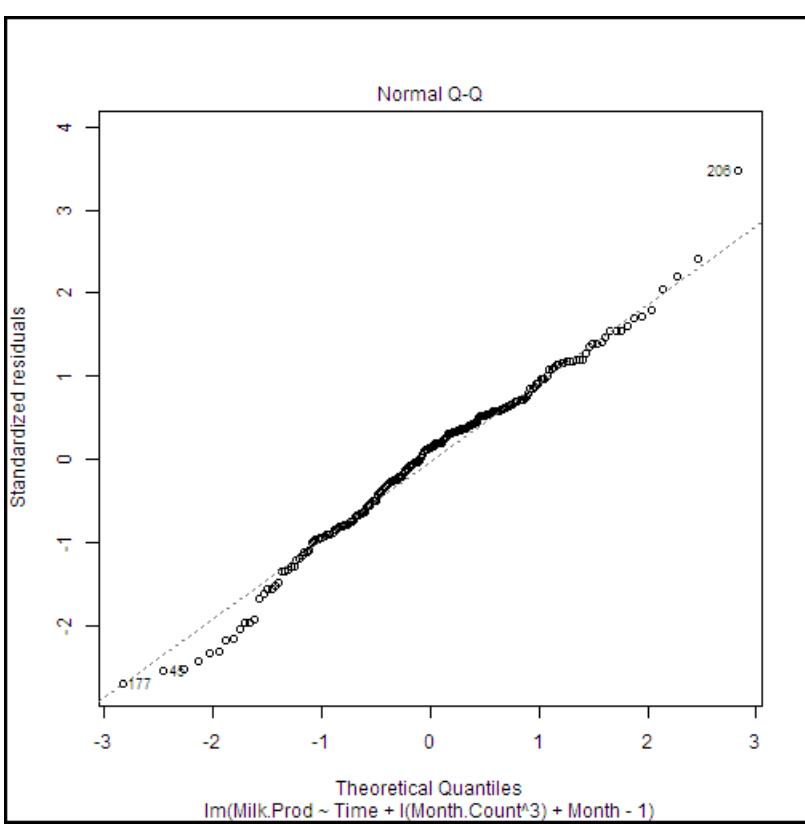
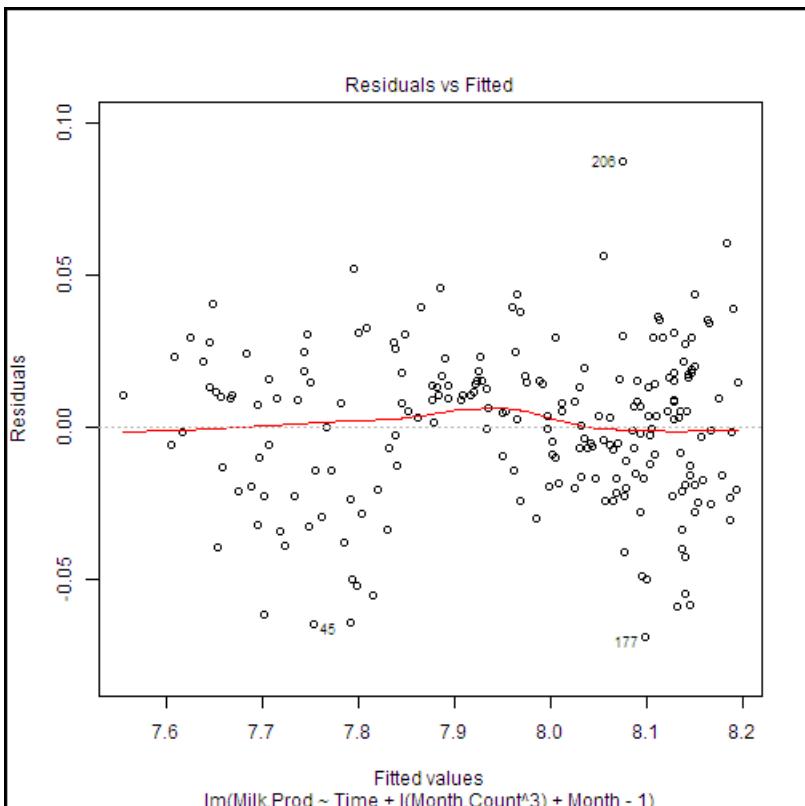
These residuals look reasonable. There is no particular structure, except the effect of the 2008-2009 recession, which our model does not account for particularly well.

The plot shown in Figure 25 is useful for detecting any time-dependent patterns in the residuals. The explicit approach of computing and plotting the residuals I used places the residuals in time order on the plot. If, on the other hand, I had plotted `milk.lm$residuals`, the plot would not have been in time order.

You can also use `plot.lm()` to produce a series of diagnostic plots.

```
## Show the diagnostic plots for the model
plot(milk.lm2, ask = FALSE)
```

This code produces a series of diagnostic plots shown in Figure 26.



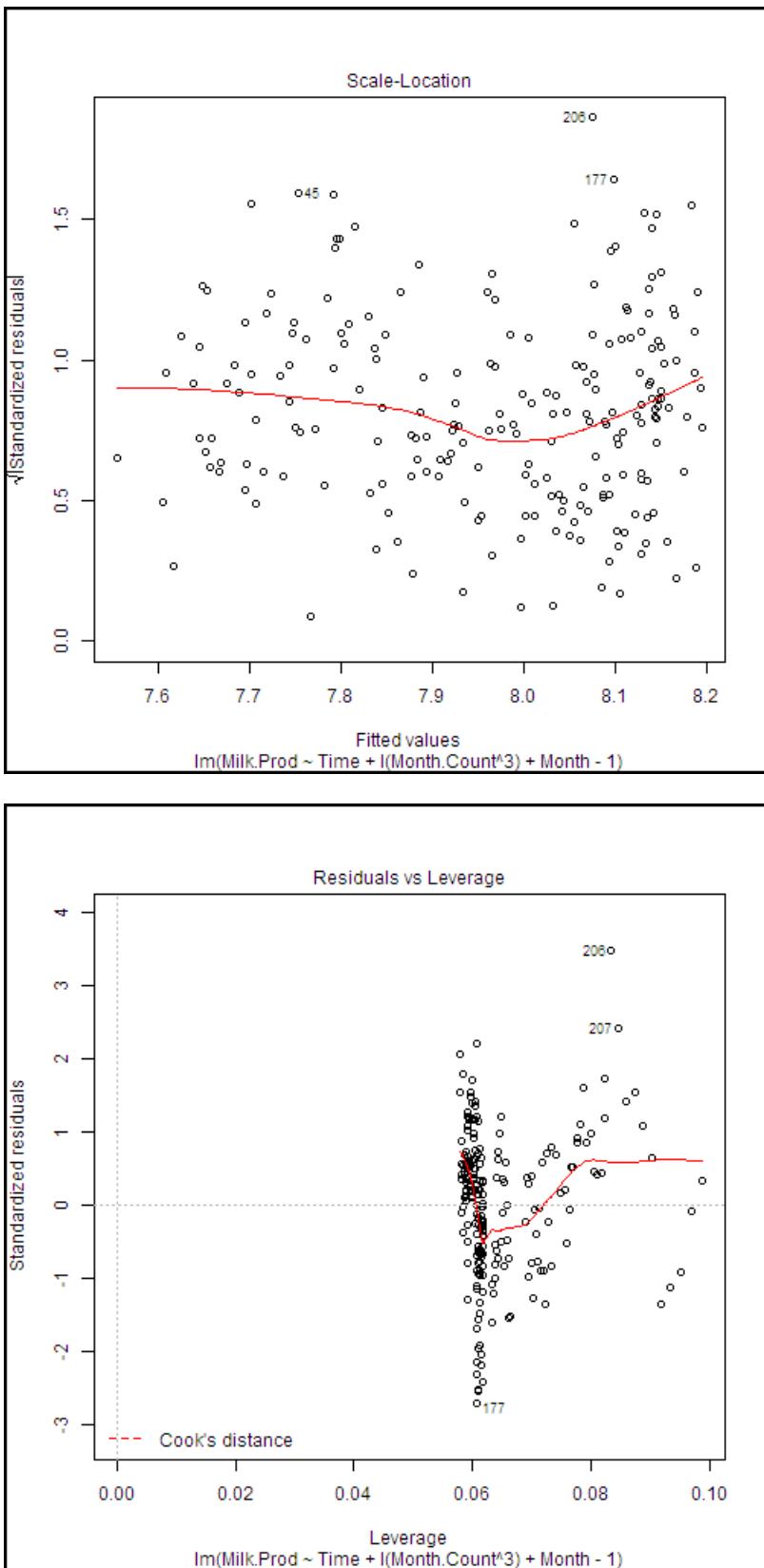


Figure 26. Diagnostic plots for the seasonal model.

There are a few highly influential points identified in these plots, but nothing to cause great concern. Further, we can see from the Normal Q-Q plot that the residuals are close to normally distributed, an important assumption for linear models.

Forecasting and model evaluation

There is just one more thing to do to complete our example. We need to compute forecasts and measure the error against the actual data. Our forecast will be for the 12 months of 2013. We can compute an error measure for this forecast to the actual data that is not part of our training dataset. Additionally, we can compare performance on the 18 years of training data to the 12 months of test data.

A number of metrics are used to measure the performance of time series models. In our case we will use the root mean square (RMS) error. The following function computes the RMS error between two series.

```
RMS.error<- function(series1, series2, is.log = TRUE, min.length = 2){  
  ## Function to compute the RMS error or difference between two  
  ## series or vectors  
  
  messages <- c("ERROR: Input arguments to function RMS.error of wrong type encountered",  
             "ERROR: Input vector to function RMS.error is too short",  
             "ERROR: Input vectors to function RMS.error must be of same length",  
             "WARNING: Function rms.error has received invalid input time series.")  
  
  ## Check the arguments  
  if(!is.numeric(series1) | !is.numeric(series2) | !is.logical(is.log) | !is.numeric(min.length)) {  
    warning(messages[1])  
    return(NA)}  
  
  if(length(series1)<min.length) {  
    warning(messages[2])  
    return(NA)}  
  
  if((length(series1) != length(series2))) {  
    warning(messages[3])  
    return(NA)}  
  
  ## If is.log is TRUE exponentiate the values, else just copy  
  if(is.log) {  
    tryCatch( {  
      temp1 <- exp(series1)  
      temp2 <- exp(series2) ,  
      error = function(e){warning(messages[4]); NA}  
    })  
  } else {  
    temp1 <- series1  
    temp2 <- series2  
  }  
  
  ## Compute predictions from our models  
  predict1 <- predict(milk.lm, cadairydata)  
  predict2 <- predict(milk.lm2, cadairydata)  
  
  ## Compute the RMS error in a dataframe  
  tryCatch( {  
    sqrt(sum((temp1 - temp2)^2) / length(temp1)),  
    error = function(e){warning(messages[4]); NA})  
  })
```

As with the `log_transform()` function we discussed in the "Value transformations" section, there is quite a lot of error checking and exception recovery code in this function. The principles employed are the same. The work is done in two places wrapped in `tryCatch()`. First, the time series are exponentiated, since we have been working with the logs of the values. Second, the actual RMS error is computed.

Equipped with a function to measure the RMS error, let's build and output a dataframe containing the RMS errors. We will include terms for the trend model alone and the complete model with seasonal factors. The following code does the job by using the two linear models we have constructed.

```

## Compute the RMS error in a dataframe
## Include the row names in the first column so they will
## appear in the output of the Execute R Script
RMS.df <- data.frame(
  rowNames = c("Trend Model", "Seasonal Model"),
  Traing = c(
    RMS.error(predict1[1:216], cadairydata$Milk.Prod[1:216]),
    RMS.error(predict2[1:216], cadairydata$Milk.Prod[1:216])),
  Forecast = c(
    RMS.error(predict1[217:228], cadairydata$Milk.Prod[217:228]),
    RMS.error(predict2[217:228], cadairydata$Milk.Prod[217:228]))
)
RMS.df

## The following line should be executed only when running in
## Azure Machine Learning Studio
maml.mapOutputPort('RMS.df')

```

Running this code produces the output shown in Figure 27 at the Result Dataset output port.

CA Dairy Analysis > Execute R Script > Result Dataset		
rows	columns	
2	3	
	rowNames	Traing
view as		
	Trend Model	122.238008
	Seasonal Model	75.115958
	Forecast	
		174.088492
		94.725325

Figure 27. Comparison of RMS errors for the models.

From these results, we see that adding the seasonal factors to the model reduces the RMS error significantly. Not too surprisingly, the RMS error for the training data is a bit less than for the forecast.

APPENDIX A: Guide to RStudio

RStudio is quite well documented, so in this appendix I will provide some links to the key sections of the RStudio documentation to get you started.

1. Creating projects

You can organize and manage your R code into projects by using RStudio. The documentation that uses projects can be found at <https://support.rstudio.com/hc/articles/200526207-Using-Projects>.

I recommend that you follow these directions and create a project for the R code examples in this document.

2. Editing and executing R code

RStudio provides an integrated environment for editing and executing R code. Documentation can be found at <https://support.rstudio.com/hc/articles/200484448-Editing-and-Executing-Code>.

3. Debugging

RStudio includes powerful debugging capabilities. Documentation for these features is at <https://support.rstudio.com/hc/articles/200713843-Debugging-with-RStudio>.

The breakpoint troubleshooting features are documented at

<https://support.rstudio.com/hc/articles/200534337-Breakpoint-Troubleshooting>.

APPENDIX B: Further reading

This R programming tutorial covers the basics of what you need to use the R language with Azure Machine Learning Studio. If you are not familiar with R, two introductions are available on CRAN:

- R for Beginners by Emmanuel Paradis is a good place to start at http://cran.r-project.org/doc/contrib/Paradis-rdebuts_en.pdf.
- An Introduction to R by W. N. Venables et. al. goes into a bit more depth, at <http://cran.r-project.org/doc/manuals/R-intro.html>.

There are many books on R that can help you get started. Here are a few I find useful:

- The Art of R Programming: A Tour of Statistical Software Design by Norman Matloff is an excellent introduction to programming in R.
- R Cookbook by Paul Teator provides a problem and solution approach to using R.
- R in Action by Robert Kabacoff is another useful introductory book. The companion Quick R website is a useful resource at <http://www.statmethods.net/>.
- R Inferno by Patrick Burns is a surprisingly humorous book that deals with a number of tricky and difficult topics that can be encountered when programming in R. The book is available for free at <http://www.burns-stat.com/documents/books/the-r-inferno/>.
- If you want a deep dive into advanced topics in R, have a look at the book Advanced R by Hadley Wickham. The online version of this book is available for free at <http://adv-r.had.co.nz/>.

A catalogue of R time series packages can be found in the CRAN Task View for time series analysis: <http://cran.r-project.org/web/views/TimeSeries.html>. For information on specific time series object packages, you should refer to the documentation for that package.

The book Introductory Time Series with R by Paul Cowpertwait and Andrew Metcalfe provides an introduction to using R for time series analysis. Many more theoretical texts provide R examples.

Some great internet resources:

- DataCamp: DataCamp teaches R in the comfort of your browser with video lessons and coding exercises. There are interactive tutorials on the latest R techniques and packages. Take the free interactive R tutorial at <https://www.datacamp.com/courses/introduction-to-r>
- A quick R tutorial by Kelly Black from Clarkson University <http://www.cyclismo.org/tutorial/R/>
- 60+ R resources listed at <http://www.computerworld.com/article/2497464/business-intelligence-60-r-resources-to-improve-your-data-skills.html>

Set up data science environments for use in the Team Data Science Process

1/17/2017 • 1 min to read • [Edit on GitHub](#)

The Team Data Science Process uses various data science environments for the storage, processing, and analysis of data. They include Azure Blob Storage, several types of Azure virtual machines, HDInsight (Hadoop) clusters, and Azure Machine Learning workspaces. The decision about which environment to use depends on the type and quantity of data to be modeled and the target destination for that data in the cloud.

- For guidance on questions to consider when making this decision, see [Plan Your Azure Machine Learning Data Science Environment](#).
- For a catalog of some of the scenarios you might encounter when doing advanced analytics, see [Scenarios for the Team Data Science Process](#)

This menu links to topics that describe how to set up the various data science environments used by the Team Data Science Process.

The **Microsoft Data Science Virtual Machine (DSVM)** is also available as an Azure virtual machine (VM) image. This VM is pre-installed and configured with several popular tools that are commonly used for data analytics and machine learning. The DSVM is available on both Windows and Linux. For further information, see [Introduction to the cloud-based Data Science Virtual Machine for Linux and Windows](#).

Data Science Virtual machines in Azure

1/17/2017 • 2 min to read • [Edit on GitHub](#)

Instructions are provided here that describe how to set up an Azure VM and an Azure VM with SQL Service as IPython Notebook servers. The Windows virtual machine is configured with supporting tools such as IPython Notebook, Azure Storage Explorer, and AzCopy, as well as other utilities that are useful for data science projects. Azure Storage Explorer and AzCopy, for example, provide convenient ways to upload data to Azure storage from your local machine or to download it to your local machine from storage.

This menu links to topics that describe how to set up the various data science environments used by the [Team Data Science Process \(TDSP\)](#).

Several types of Azure virtual machines can be provisioned and configured to be used as part of a cloud-based data science environment. The decision about which flavor of virtual machine to use depends on the type and quantity of data to be modeled with machine learning, and the target destination for that data in the cloud.

- For guidance on the questions to consider when making this decision, see [Plan Your Azure Machine Learning Data Science Environment](#).
- For a catalog of some of the scenarios you might encounter when doing advanced analytics, see [Scenarios for the Advanced Analytics Process and Technology in Azure Machine Learning](#)

Two sets of instructions are provided:

- [Set up an Azure virtual machine as an IPython Notebook server for advanced analytics](#) shows how to provision an Azure virtual machine with IPython Notebook and other tools used to do data science for cases in which a form of Azure storage other than SQL can be used to store the data.
- [Set up an Azure SQL Server virtual machine as an IPython Notebook server for advanced analytics](#) shows how to provision an Azure SQL Server virtual machine with IPython Notebook and other tools used to do data science for cases in which a SQL database can be used to store the data.

Once provisioned and configured, these virtual machines are ready for use as IPython Notebook servers for the exploration and processing of data, and for other tasks needed in conjunction with Azure Machine Learning and the Team Data Science Process (TDSP). The next steps in the data science process are mapped in the [TDSP learning path](#) and may include steps that move data into SQL Server or HDInsight, process and sample it there in preparation for learning from the data with Azure Machine Learning.

NOTE

Azure Virtual Machines are priced as **pay only for what you use**. To ensure that you are not being billed when not using your virtual machine, it has to be in the **Stopped (Deallocated)** state from the [Azure Classic Portal](#). For step-by-step instructions or how to deallocate your virtual machine, see [Shutdown and deallocate virtual machine when not in use](#)

Customize Azure HDInsight Hadoop clusters for the Team Data Science Process

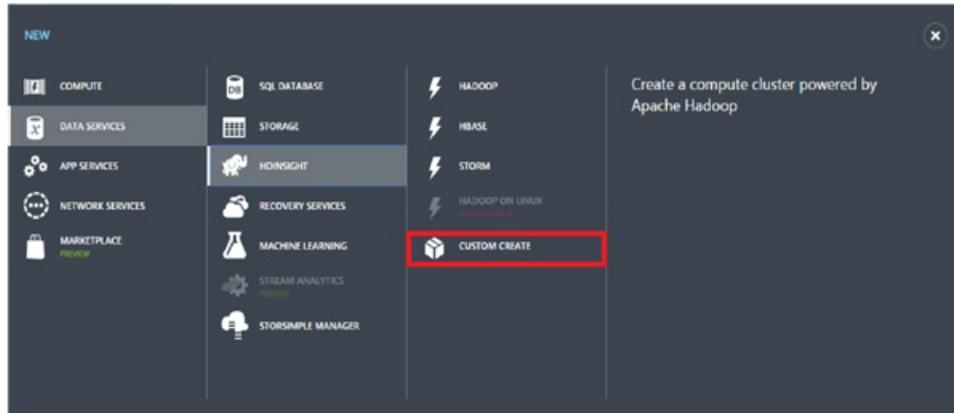
1/17/2017 • 3 min to read • [Edit on GitHub](#)

This article describes how to customize an HDInsight Hadoop cluster by installing 64-bit Anaconda (Python 2.7) on each node when the cluster is provisioned as an HDInsight service. It also shows how to access the headnode to submit custom jobs to the cluster. This customization makes many popular Python modules, that are included in Anaconda, conveniently available for use in user defined functions (UDFs) that are designed to process Hive records in the cluster. For instructions on the procedures used in this scenario, see [How to submit Hive queries](#).

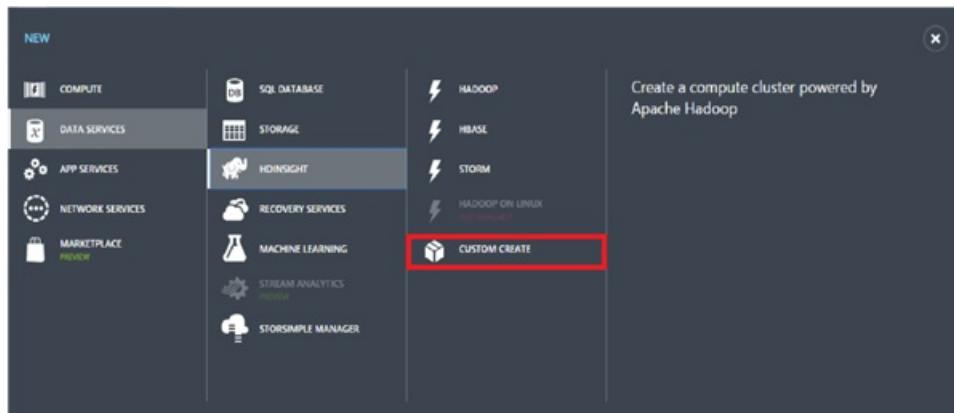
The following menu links to topics that describe how to set up the various data science environments used by the [Team Data Science Process \(TDSP\)](#).

Customize Azure HDInsight Hadoop Cluster

To create a customized HDInsight Hadoop cluster, start by logging on to [Azure classic portal](#), click **New** at the left bottom corner, and then select DATA SERVICES -> HDINSIGHT -> **CUSTOM CREATE** to bring up the **Cluster Details** window.



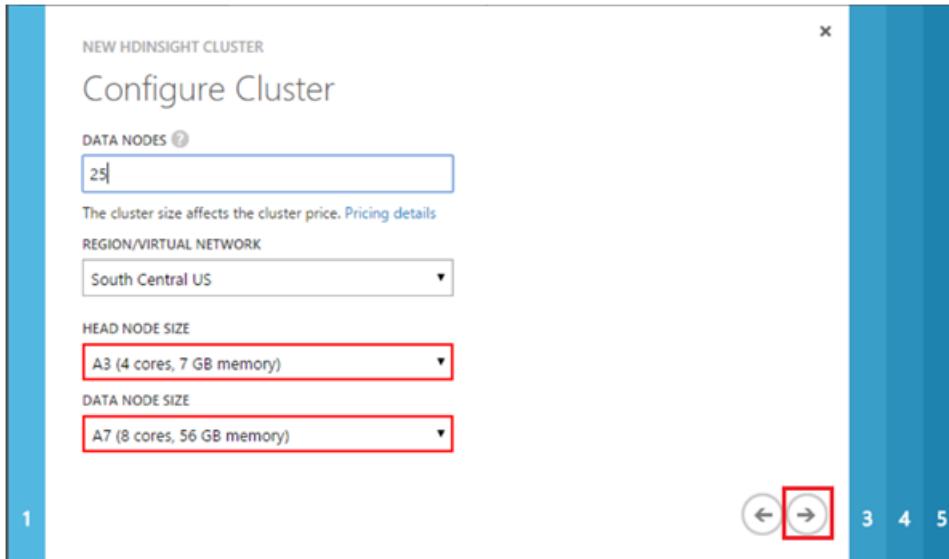
Input the name of the cluster to be created on configuration page 1, and accept default values for the other fields. Click the arrow to go to the next configuration page.



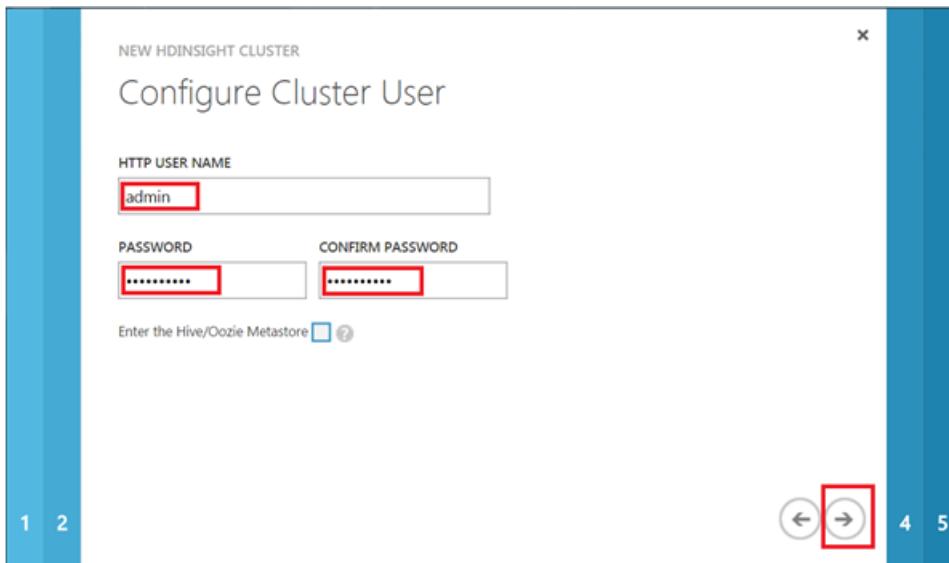
On configuration page 2, input the number of **DATA NODES**, select the **REGION/VIRTUAL NETWORK**, and select the sizes of the **HEAD NODE** and the **DATA NODE**. Click the arrow to go to the next configuration page.

NOTE

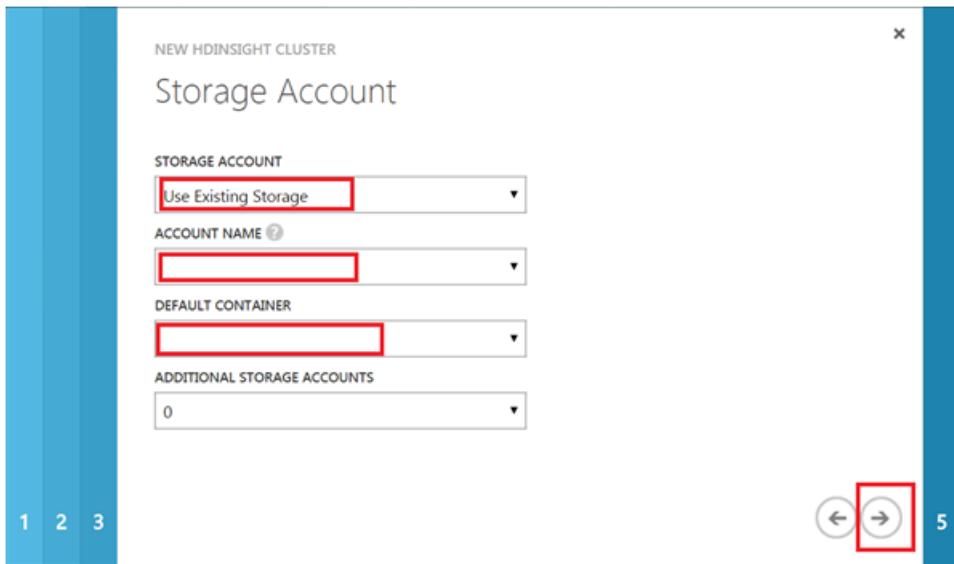
The **REGION/VIRTUAL NETWORK** has to be the same as the region of the storage account that is going to be used for the HDInsight Hadoop cluster. Otherwise, in the fourth configuration page, the storage account will not appear on the dropdown list of **ACCOUNT NAME**.



On configuration page 3, provide a user name and password for the HDInsight Hadoop cluster. **Do not** select the *Enter the Hive/Oozie Metastore*. Then, click the arrow to go to the next configuration page.



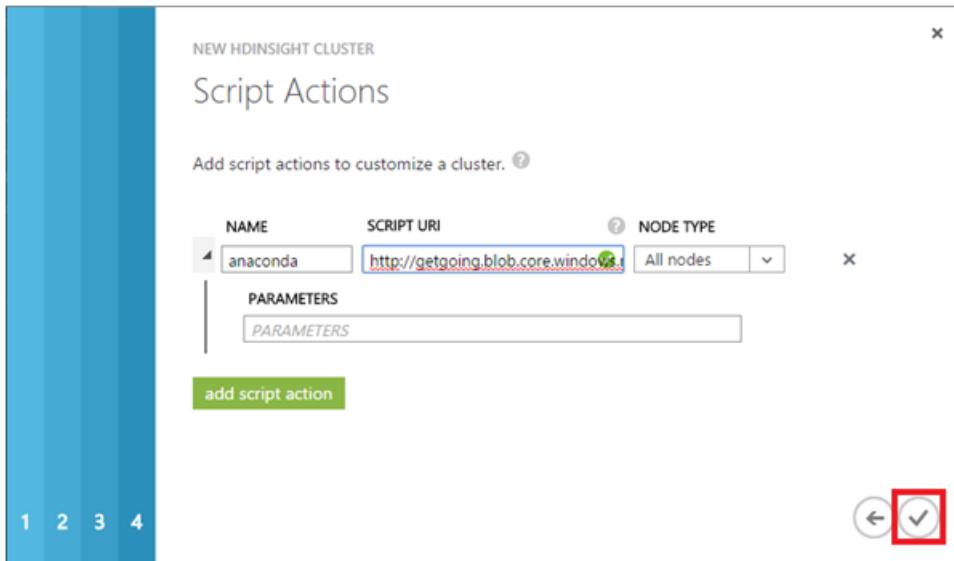
On configuration page 4, specify the storage account name, the default container of the HDInsight Hadoop cluster. If you select *Create default container* in the **DEFAULT CONTAINER** dropdown list, a container with the same name as the cluster will be created. Click the arrow to go to the last configuration page.



On the final **Script Actions** configuration page, click **add script action** button, and fill the text fields with the following values.

- **NAME** - any string as the name of this script action
- **NODE TYPE** - select **All nodes**
- **SCRIPT URI** - http://getgoing.blob.core.windows.net/publicscripts/Azure_HDI_Setup_Windows.ps1
 - *publicscripts* is a public container in the storage account
 - *getgoing* we use to share PowerShell script files to facilitate users' work in Azure
- **PARAMETERS** - (leave blank)

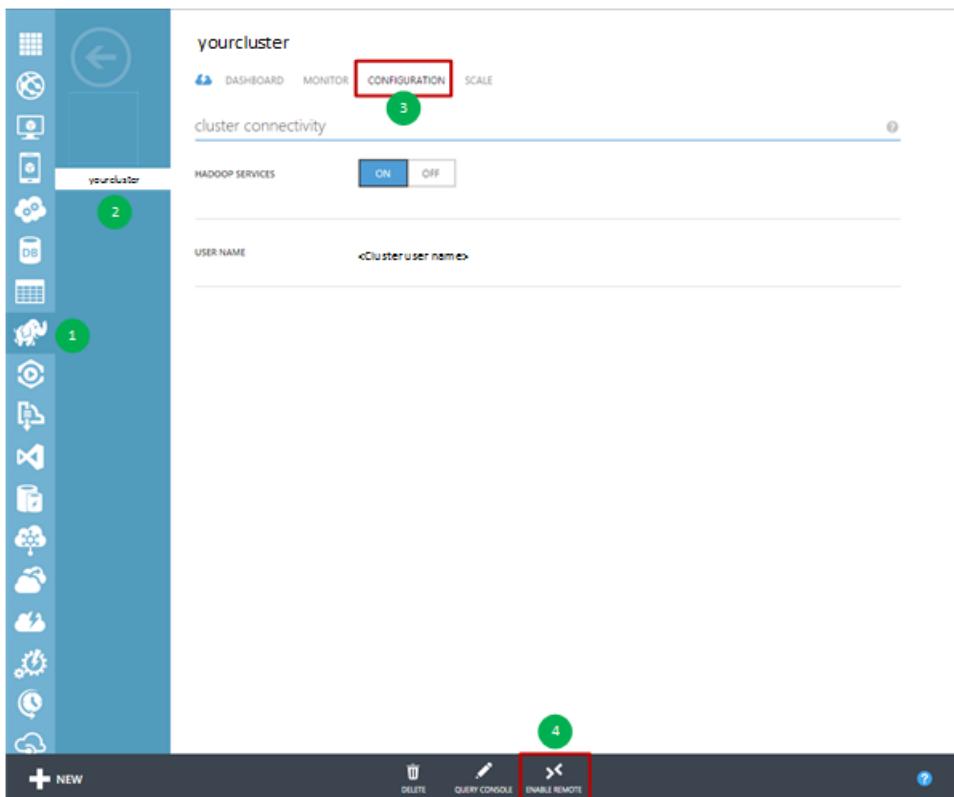
Finally, click the check mark to start the creation of the customized HDInsight Hadoop cluster.



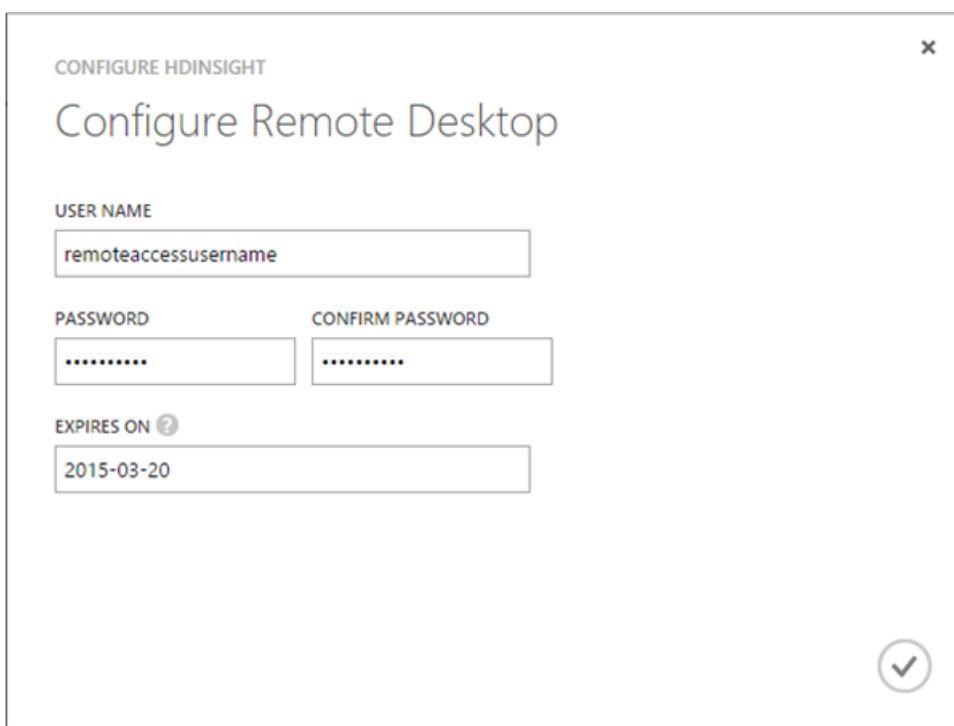
Access the Head Node of Hadoop Cluster

You must enable remote access to the Hadoop cluster in Azure before you can access the head node of the Hadoop cluster through RDP.

1. Log in to the [Azure classic portal](#), select **HDInsight** on the left, select your Hadoop cluster from the list of clusters, click the **CONFIGURATION** tab, and then click the **ENABLE REMOTE** icon at the bottom of the page.



2. In the **Configure Remote Desktop** window, enter the USER NAME and PASSWORD fields, and select the expiration date for remote access. Then click the check mark to enable the remote access to the head node of the Hadoop cluster.

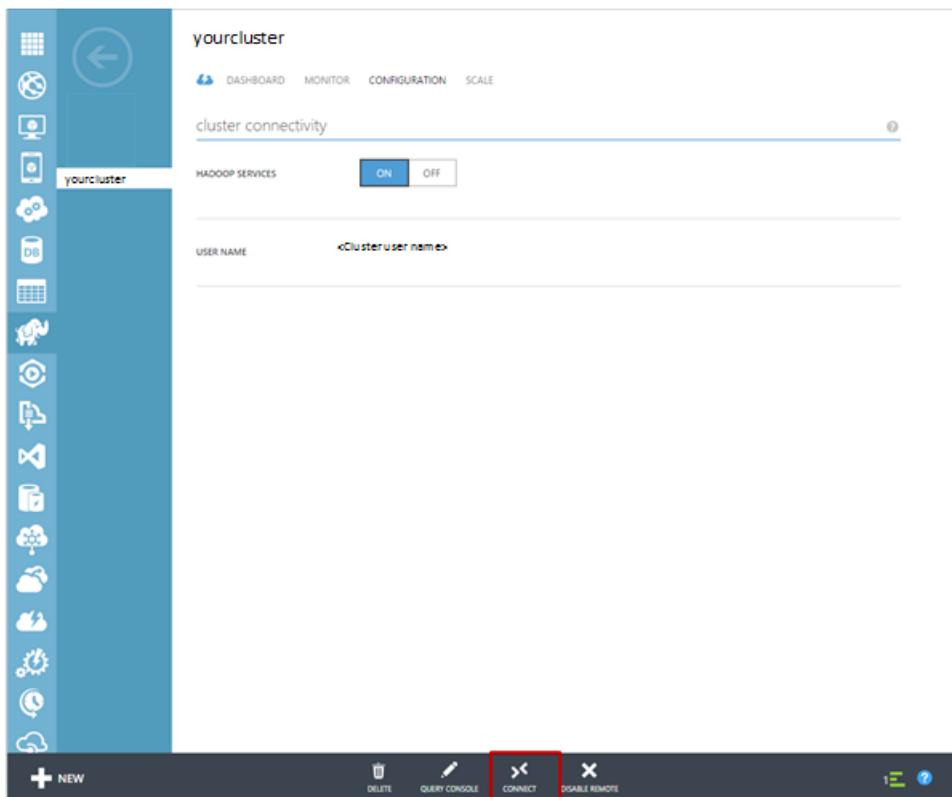


NOTE

The user name and password for the remote access are not the user name and password that you use when you created the Hadoop cluster. This is a separate set of credentials. Also, the expiration date of the remote access has to be within 7 days from the current date.

After remote access is enabled, click **CONNECT** at the bottom of the page to remote into the head node. You log on to the head node of the Hadoop cluster by entering the credentials for the remote access user that you

specified earlier.



The next steps in the advanced analytics process are mapped in the [Team Data Science Process \(TDSP\)](#) and may include steps that move data into HDInsight, then process and sample it there in preparation for learning from the data with Azure Machine Learning.

See [How to submit Hive queries](#) for instructions on how to access the Python modules that are included in Anaconda from the head node of the cluster in user-defined functions (UDFs) that are used to process Hive records stored in the cluster.

Introduction to the cloud-based Data Science Virtual Machine for Linux and Windows

1/17/2017 • 5 min to read • [Edit on GitHub](#)

The Data Science Virtual Machine is a customized VM image on Microsoft's Azure cloud built specifically for doing data science. It has many popular data science and other tools pre-installed and pre-configured to jump-start building intelligent applications for advanced analytics. It is available on Windows Server 2012 or on OpenLogic 7.2 CentOS-based Linux versions.

This topic discusses what you can do with the Data Science VM, outlines some of the key scenarios for using the VM, itemizes the key features available on the Windows and Linux versions, and provides instructions on how to get started using them.

What can I do with the Data Science Virtual Machine?

The goal of the Data Science Virtual Machine is to provide data professionals at all skill levels and roles with a friction-free data science environment. This VM saves you considerable time that you would spend if you had rolled out a comparable environment on your own. Instead, start your data science project immediately in a newly created VM instance.

The Data Science VM is designed and configured for working with a broad usage scenarios. You can scale your environment up or down as your project needs change. You are able to use your preferred language to program data science tasks. You can install other tools and customize the system for your exact needs.

Key Scenarios

This section suggests some key scenarios for which the Data Science VM can be deployed.

Preconfigured analytics desktop in the cloud

The Data Science VM provides a baseline configuration for data science teams looking to replace their local desktops with a managed cloud desktop. This baseline ensures that all the data scientists on a team have a consistent setup with which to verify experiments and promote collaboration. It also lowers costs by reducing the sysadmin burden and saving on the time needed to evaluate, install, and maintain the various software packages needed to do advanced analytics.

Data science training and education

Enterprise trainers and educators that teach data science classes usually provide a virtual machine image to ensure that their students have a consistent setup and that the samples work predictably. The Data Science VM creates an on-demand environment with a consistent setup that eases the support and incompatibility challenges. Cases where these environments need to be built frequently, especially for shorter training classes, benefit substantially.

On-demand elastic capacity for large-scale projects

Data science hackathons/competitions or large-scale data modeling and exploration require scaled out hardware capacity, typically for short duration. The Data Science VM can help replicate the data science environment quickly on demand, on scaled out servers that allow experiments requiring high-powered computing resources to be run.

Short-term experimentation and evaluation

The Data Science VM can be used to evaluate or learn tools such as Microsoft R Server, SQL Server, Visual Studio tools, Jupyter, deep learning / ML toolkits, and new tools popular in the community with minimal setup effort. Since the Data Science VM can be set up quickly, it can be applied in other short-term usage scenarios such as replicating

published experiments, executing demos, following walkthroughs in online sessions or conference tutorials.

What's included in the Data Science VM?

The Data Science Virtual Machine has many popular data science tools already installed and configured. It also includes tools that make it easy to work with various Azure data and analytics products. You can explore and build predictive models on large-scale data sets using the Microsoft R Server or using SQL Server 2016. A host of other tools from the open source community and from Microsoft are also included, as well as sample code and notebooks. The following table itemizes and compares the main components included in the Windows and Linux editions of the Data Science Virtual Machine.

WINDOWS EDITION	LINUX EDITION
Microsoft R Server Developer Edition	Microsoft R Server Developer Edition
Anaconda Python 2.7, 3.5	Anaconda Python 2.7, 3.5
Jupyter Notebook Server (R, Python)	JupyterHub: Multi-user Jupyter notebooks (R, Python, Julia)
SQL Server 2016 Developer Edition: Scalable in-database analytics with R services	Postgres, SQuirreL SQL (database tool), SQL Server drivers, and command line (bcp, sqlcmd)
Visual Studio Community Edition 2015 (IDE) - Azure HDInsight (Hadoop), Data Lake, SQL Server Data tools - Node.js, Python, and R tools for Visual Studio	IDEs and editors - Eclipse with Azure toolkit plugin - Emacs (with ESS, auctex) gedit
Power BI desktop	--
Machine Learning Tools - Integration with Azure Machine Learning - CNTK (deep learning/AI) - Xgboost (popular ML tool in data science competitions) - Vowpal Wabbit (fast online learner) - Rattle (visual quick-start data and analytics tool) - Mxnet (deep learning/AI)	Machine Learning Tools - Integrations with Azure Machine Learning - CNTK (deep learning/AI) - Xgboost (popular ML tool in data science competitions) - Vowpal Wabbit (fast online learner) - Rattle (visual quick-start data and analytics tool)
SDKs to access Azure and Cortana Intelligence Suite of services	SDKs to access Azure and Cortana Intelligence Suite of services
Tools for data movement and management of Azure and Big Data resources: Azure Storage Explorer, CLI, PowerShell, AdlCopy (Azure Data Lake), AzCopy, dtui (for DocumentDB), Microsoft Data Management Gateway	Tools for data movement and management of Azure and Big Data resources: Azure Storage Explorer, CLI
Git, Visual Studio Team Services plugin	Git
Windows port of most popular Linux/Unix command-line utilities accessible through GitBash/command prompt	--

How to get started with the Windows Data Science VM

- Create an instance of the VM on Windows by navigating to [this page](#) and selecting the green **Create Virtual Machine** button.
- Sign in to the VM from your remote desktop using the credentials you specified when you created the VM.
- To discover and launch the tools available, click the **Start** menu.

Get started with the Linux Data Science VM

- Create an instance of the VM on Linux (OpenLogic CentOS-based) by navigating to [this page](#) and selecting the **Create Virtual Machine** button.
- Sign in to the VM from an SSH client, such as Putty or SSH Command, using the credentials you specified when you created the VM.
- In the shell prompt, enter `dsvm-more-info`.
- For a graphical desktop, download the X2Go client for your client platform [here](#) and follow the instructions in the Linux Data Science VM document [Provision the Linux Data Science Virtual Machine](#).

Next steps

For the Windows Data Science VM

- For more information on how to run specific tools available on the Windows version, see [Provision the Microsoft Data Science Virtual Machine](#) and
- For more information on how to perform various tasks needed for your data science project on the Windows VM, see [Ten things you can do on the Data science Virtual Machine](#).

For the Linux Data Science VM

- For more information on how to run specific tools available on the Linux version, see [Provision the Linux Data Science Virtual Machine](#).
- For a walkthrough that shows you how to perform several common data science tasks with the Linux VM, see [Data science on the Linux Data Science Virtual Machine](#).

Ten things you can do on the Data science Virtual Machine

1/17/2017 • 27 min to read • [Edit on GitHub](#)

The Microsoft Data Science Virtual Machine (DSVM) is a powerful data science development environment that enables you to perform various data exploration and modeling tasks. The environment comes already built and bundled with several popular data analytics tools that make it easy to get started quickly with your analysis for On-premises, Cloud or hybrid deployments. The DSVM works closely with many Azure services and is able to read and process data that is already stored on Azure, in Azure SQL Data Warehouse, Azure Data Lake, Azure Storage, or in DocumentDB. It can also leverage other analytics tools such as Azure Machine Learning and Azure Data Factory.

In this article we walk you through how to use your DSVM to perform various data science tasks and interact with other Azure services. Here are some of the things you can do on the DSVM:

1. Explore data and develop models locally on the DSVM using Microsoft R Server, Python
2. Use a Jupyter notebook to experiment with your data on a browser using Python 2, Python 3, Microsoft R an enterprise ready version of R designed for scalability and performance
3. Operationalize models built using R and Python on Azure Machine Learning so client applications can access your models using a simple web services interface
4. Administer your Azure resources using Azure portal or Powershell
5. Extend your storage space and share large-scale datasets / code across your whole team by creating an Azure File Storage as a mountable drive on your DSVM
6. Share code with your team using Github and access your repository using the pre-installed Git clients - Git Bash, Git GUI.
7. Access various Azure data and analytics services like Azure blob storage, Azure Data Lake, Azure HDInsight (Hadoop), Azure DocumentDB, Azure SQL Data Warehouse & databases
8. Build reports and dashboard using the Power BI Desktop pre-installed on the DSVM and deploy them on the cloud
9. Dynamically scale your DSVM to meet your project needs
10. Install additional tools on your virtual machine

NOTE

Additional usage charges will apply for many of the additional data storage and analytics services listed in this article. Please refer to the [Azure Pricing](#) page for details.

Prerequisites

- You will need an Azure subscription. You can sign up for a free trial [here](#).
- Instructions for provisioning a Data Science Virtual Machine on the Azure portal are available at [Creating a virtual machine](#).

1. Explore data and develop models using Microsoft R Server or Python

You can use languages like R and Python to do your data analytics right on the DSVM.

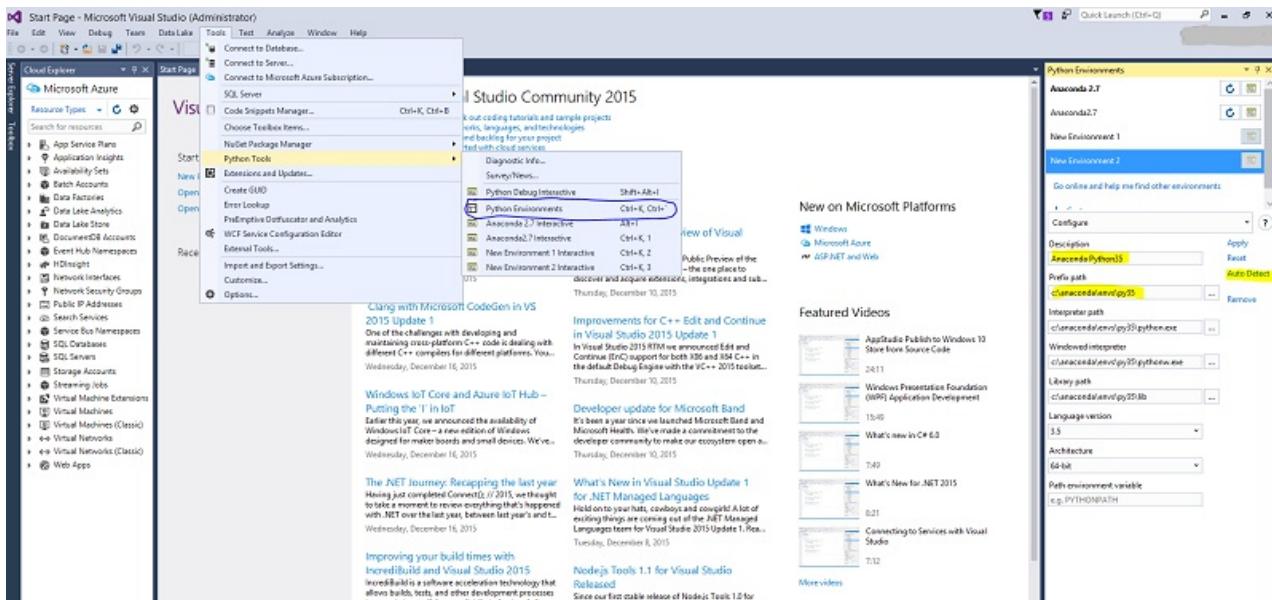
For R, you can use an IDE called "Revolution R Enterprise 8.0" that can be found on the start menu or the desktop. Microsoft has provided additional libraries on top of the Open source/CRAN-R to enable scalable analytics and the

ability to analyze data larger than the memory size allowed by doing parallel chunked analysis. You can also install an R IDE of your choice like [RStudio](#).

For Python, you can use an IDE like Visual Studio Community Edition which has the Python Tools for Visual Studio (PTVS) extension pre-installed. By default, only a basic Python 2.7 is configured on PTVS (without any analytics library like SciKit, Pandas). In order to enable Anaconda Python 2.7 and 3.5, you need to do the following:

- Create custom environments for each version by navigating to **Tools -> Python Tools -> Python Environments** and then clicking "+ Custom" in the Visual Studio 2015 Community Edition
- Give a description and set the environment prefix paths as `c:\anaconda` for Anaconda Python 2.7 OR `c:\anaconda\envs\py35` for Anaconda Python 3.5
- Click **Auto Detect** and then **Apply** to save the environment.

Here is what the custom environment setup looks like in Visual Studio.



See the [PTVS documentation](#) for additional details on how to create Python Environments.

Now you are set up to create a new Python project. Navigate to **File -> New -> Project -> Python** and select the type of Python application you are building. You can set the Python environment for the current project to the desired version (Anaconda 2.7 or 3.5): right-click the **Python environment**, select **Add/Remove Python Environments**, and then select the desired environment to associate with the project. You can find more information about working with PTVS on the product [documentation](#) page.

2. Using a Jupyter Notebook to explore and model your data with Python or R

The Jupyter Notebook is a powerful environment that provides a browser-based "IDE" for data exploration and modeling. You can use Python 2, Python 3 or R (both Open Source and the Microsoft R Server) in a Jupyter Notebook.

To launch the Jupyter Notebook click on the start menu icon / desktop icon titled **Jupyter Notebook**. On the DSVM you can also browse to "<https://localhost:9999/>" to access the Jupiter Notebook. If it prompts you for a password, use instructions provided in the **How to create a strong password on the Jupyter notebook server** section of the [Provision the Microsoft Data Science Virtual Machine](#) topic to create a strong password to access the Jupyter notebook.

Once you have opened the notebook, you will see a directory that contains a few example notebooks that are pre-packaged into the DSVM. Now you can:

- click on the notebook to see the code.
- execute each cell by pressing **SHIFT-ENTER**.
- run the entire notebook by clicking on **Cell -> Run**
- create a new notebook by clicking on the Jupyter Icon (left top corner) and then clicking **New** button on the right and then choosing the notebook language (also known as kernels).

NOTE

Currently we support Python 2.7, Python 3.5 and R. The R kernel supports programming in both Open source R as well as the enterprise scalable Microsoft R Server.

Once you are in the notebook you can explore your data, build the model, test the model using your choice of libraries.

3. Build models using R or Python and Operationalize them using Azure Machine Learning

Once you have built and validated your model the next step is usually to deploy it into production. This allows your client applications to invoke the model predictions on a real time or on a batch mode basis. Azure Machine Learning provides a mechanism to operationalize a model built in either R or Python.

When you operationalize your model in Azure Machine Learning, a web service is exposed that allows clients to make REST calls that pass in input parameters and receive predictions from the model as outputs.

NOTE

If you have not yet signed up for Azure Machine Learning, you can obtain a free workspace or a standard workspace by visiting the [Azure Machine Learning Studio](#) home page and clicking on "Get Started".

Build and Operationalize Python models

Here is a snippet of code developed in a Python Jupyter Notebook that builds a simple model using the SciKit-learn library.

```
#IRIS classification
from sklearn import datasets
from sklearn import svm
clf=svm.SVC()
iris = datasets.load_iris()
X, y = iris.data, iris.target
clf.fit(X, y)
```

The method used to deploy your python models to Azure Machine Learning wraps the prediction of the model into a function and decorates it with attributes provided by the pre-installed Azure Machine Learning python library that denote your Azure Machine Learning workspace ID, API Key, and the input and return parameters.

```
from azureml import services
@services.publish(workspaceid, auth_token)
@services.types(sep_l=float, sep_w=float, pet_l=float, pet_w=float)
@services.returns(int) #0, or 1, or 2
def predictIris(sep_l, sep_w, pet_l, pet_w):
    inputArray =[sep_l, sep_w, pet_l, pet_w]
    return clf.predict(inputArray)
```

A client can now make calls to the web service. There are convenience wrappers that construct the REST API

requests. Here is a sample code to consume the web service.

```
# Consume through web service URL and keys
from azureml import services
@services.service(url, api_key)
@services.types(sep_l=float, sep_w=float, pet_l=float, pet_w=float)
@services.returns(float)
def IrisPredictor(sep_l, sep_w, pet_l, pet_w):
    pass

IrisPredictor(3,2,3,4)
```

NOTE

The Azure Machine Learning library is only supported on Python 2.7 currently.

Build and Operationalize R models

You can deploy R models built on the Data Science Virtual Machine or elsewhere onto Azure Machine Learning in a manner that is similar to how it is done for Python. Her are the steps:

- create a settings.json file as below to provide your workspace ID and auth token.
- write a wrapper for the model's predict function.
- call `publishWebService` in the Azure Machine Learning library to pass in the function wrapper.

Here is the procedure and code snippets that can be used to set up, build, publish, and consume a model as a web service in Azure Machine Learning.

Setup

1. Install the Machine Learning R package by typing `install.packages("AzureML")` in Revolution R Enterprise 8.0 IDE or your R IDE.
2. Download RTools from [here](#). You need the zip utility in the path (and named zip.exe) to operationalize your R package into Machine Learning.
3. Create a settings.json file under a directory called `.azureml` under your home directory and enter the parameters from your Azure Machine Learning workspace:

settings.json File structure:

```
{"workspace":{
  "id" : "ENTER YOUR AZUREML WORKSPACE ID",
  "authorization_token" : "ENTER YOUR AZUREML AUTH TOKEN"
}}
```

Build a model in R and publish it in Azure Machine Learning

```

library(AzureML)
ws <- workspace(config="~/azureml/settings.json")

if(!require("lme4")) install.packages("lme4")
library(lme4)
set.seed(1)
train <- sleepstudy[sample(nrow(sleepstudy), 120),]
m <- lm(Reaction ~ Days + Subject, data = train)

# Define a prediction function to publish based on the model:
sleepyPredict <- function(newdata){
  predict(m, newdata=newdata)
}

ep <- publishWebService(ws, fun = sleepyPredict, name="sleepy lm", inputSchema = sleepstudy, data.frame=TRUE)

```

Consume the model deployed in Azure Machine Learning

To consume the model from a client application, we use the Azure Machine Learning library to look up the published web service by name using the `services` API call to determine the endpoint. Then you just call the `consume` function and pass in the data frame to be predicted. The following code is used to consume the model published as an Azure Machine Learning web service.

```

library(AzureML)
library(lme4)
ws <- workspace(config="~/azureml/settings.json")

s <- services(ws, name = "sleepy lm")
s <- tail(s, 1) # use the last published function, in case of duplicate function names

ep <- endpoints(ws, s)

# OK, try this out, and compare with raw data
ans = consume(ep, sleepstudy)$ans

```

More information about the Azure Machine Learning R library can be found [here](#).

4. Administer your Azure resources using Azure portal or Powershell

The DSVM not only allows you to build your analytics solution locally on the virtual machine, but also allows you to access services on Microsoft's Azure cloud. Azure provides several compute, storage, data analytics services and other services that you can administer and access from your DSVM.

To administer your Azure subscription and cloud resources you can use your browser and point to the [Azure portal](#). You can also use Azure Powershell to administer your Azure subscription and resources via a script. You can run Azure Powershell from a shortcut on the desktop or from the start menu titled "Microsoft Azure Powershell". Refer to [Microsoft Azure Powershell documentation](#) for more information on how you can administer your Azure subscription and resources using Windows Powershell scripts.

5. Extend your storage space with a shared file system

Data scientists can share large datasets, code or other resources within the team. The DSVM itself has about 70GB of space available. To extend your storage, you can use the Azure File Service and either mount it on the DSVM or access it via a REST API.

NOTE

The maximum space of the Azure File Service share is 5TB and individual file size limit is 1TB.

You can use Azure Powershell to create an Azure File Service share. Here is the script to run under Azure PowerShell to create an Azure File service share.

```
# Authenticate to Azure.  
Login-AzureRmAccount  
# Select your subscription  
Get-AzureRmSubscription -SubscriptionName "<your subscription name>" | Select-AzureRmSubscription  
# Create a new resource group.  
New-AzureRmResourceGroup -Name <ds vmdatarg>  
# Create a new storage account. You can reuse existing storage account if you wish.  
New-AzureRmStorageAccount -Name <mydatadisk> -ResourceGroupName <ds vmdatarg> -Location "<Azure Data Center Name For eg. South Central US>" -Type "Standard_LRS"  
# Set your current working storage account  
Set-AzureRmCurrentStorageAccount -ResourceGroupName "<ds vmdatarg>" -StorageAccountName <mydatadisk>  
  
# Create a Azure File Service Share  
$s = New-AzureStorageShare <<teamsharename>>  
# Create a directory under the File share. You can give it any name  
New-AzureStorageDirectory -Share $s -Path <directory name>  
# List the share to confirm that everything worked  
Get-AzureStorageFile -Share $s
```

Now that you have created an Azure file share, you can mount it in any virtual machine in Azure. It is highly recommended that the VM is in same Azure data center as the storage account to avoid latency and data transfer charges. Here is the commands to mount the drive on the DSVM that you can run on Azure Powershell.

```
# Get storage key of the storage account that has the Azure file share from Azure portal. Store it securely on the VM to avoid prompted in next command.  
cmdkey /add:<<mydatadisk>>.file.core.windows.net /user:<<mydatadisk>> /pass:<storage key>  
  
# Mount the Azure file share as Z: drive on the VM. You can chose another drive letter if you wish  
net use z: \\<<mydatadisk>>.file.core.windows.net\<<teamsharename>>
```

Now you can access this drive as you would any normal drive on the VM.

6. Share code with your team using Github

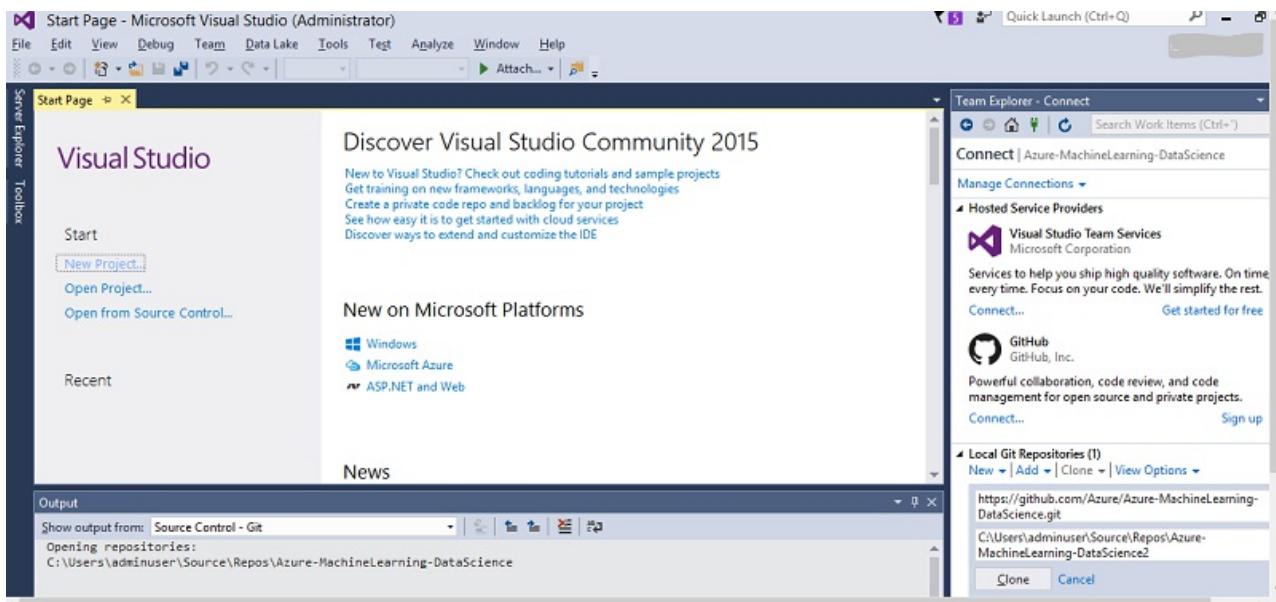
Github is a code repository where you can find a lot of sample code and sources for different tools using various technologies shared by the developer community. It uses Git as the technology to track and store versions of the code files. Github is also a platform where you can create your own repository to store your team's shared code and documentation, implement version control and also control who have access to view and contribute code. Please visit the [Github help pages](#) for more information on using Git. You can use Github as one of the ways to collaborate with your team, use code developed by the community and contribute code back to the community.

The DSVM already comes loaded with client tools on both command line as well GUI to access Github repository. The command line tool to work with Git and Github is called Git Bash. Visual Studio installed on the DSVM has the Git extensions. You can find start-up icons for these tools on the start menu and the desktop.

To download code from a Github repository you will use the `git clone` command. For example to download data science repository published by Microsoft into the current directory you can run the following command once you are in `git bash`.

```
git clone https://github.com/Azure/Azure-MachineLearning-DataScience.git
```

In Visual Studio, you can do the same clone operation. The screen-shot below shows how to access Git and Github tools in Visual Studio.



You can find more information on using Git to work with your Github repository from several resources available on github.com. The [cheat sheet](#) is a useful reference.

7. Access various Azure data and analytics services

Azure Blob

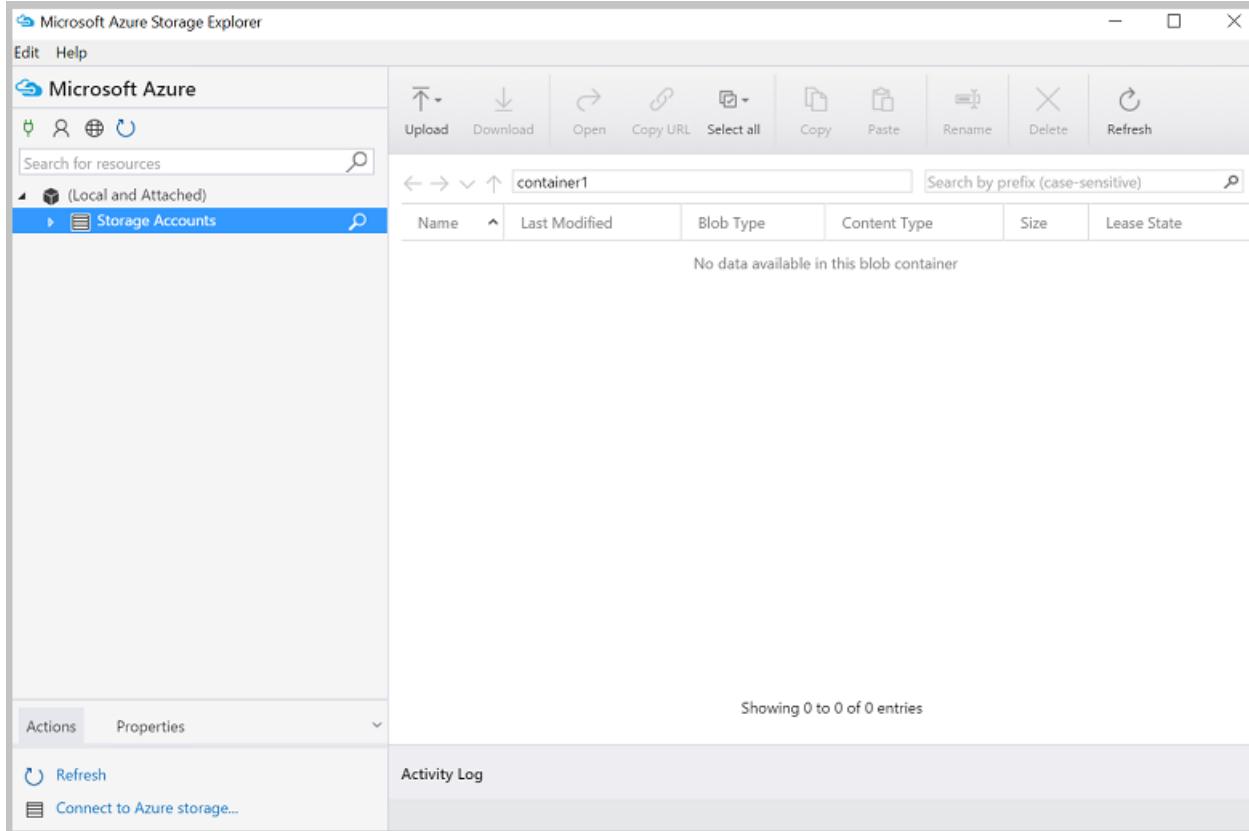
Azure blob is a reliable, economical cloud storage for data big and small. Let us look at how you can move data to Azure Blob and access data stored in an Azure Blob.

Prerequisite

- Create your Azure Blob storage account from [Azure portal](#).

The screenshot shows the Microsoft Azure portal interface. On the left, the navigation menu includes 'Internal' (highlighted in orange), 'Microsoft Azure', 'New' (with a red box around it), 'Data + Storage', 'Resource groups', 'All resources', 'Recent', 'App Services', 'Virtual machines (classic)', 'Virtual machines', 'SQL databases', 'Cloud services (classic)', 'Subscriptions', and 'Browse >'. In the center, a 'New' blade is open under 'Data + Storage'. It has a search bar and a 'MARKETPLACE' section with 'See all' and categories like 'Compute', 'Web + Mobile', 'Data + Storage' (which is highlighted with a red box), 'Data + Analytics', 'Internet of Things', 'Networking', 'Media + CDN', 'Hybrid Integration', 'Security + Identity', 'Developer Services', 'Management', and 'Container Apps'. Below this is a 'RECENT' section. On the right, the main 'Data + Storage' blade shows 'FEATURED APPS' with 'See all' and cards for 'SQL Database', 'Data Lake Store', 'SQL Data Warehouse', and 'Azure DocumentDB'. At the bottom, a 'Storage account' card is highlighted with a red box, describing it as a service for reliable, economical cloud storage.

- Confirm that the pre-installed command line AzCopy tool is found at `C:\Program Files (x86)\Microsoft SDKs\Azure\AzCopy\azcopy.exe`. You can add the directory containing the azcopy.exe to your PATH environment variable to avoid typing the full command path when running this tool. For more info on AzCopy tool please refer to [AzCopy documentation](#)
- Start the Azure Storage Explorer tool. It can be downloaded from [Microsoft Azure Storage Explorer](#).



Move data from VM to Azure Blob: AzCopy

To move data between your local files and blob storage, you can use AzCopy in command line or PowerShell:

```
AzCopy /Source:C:\myfolder /Dest:https://<mystorageaccount>.blob.core.windows.net/<mycontainer> /DestKey:<storage account key>
/Pattern:abc.txt
```

Replace **C:\myfolder** to the path where your file is stored, **mystorageaccount** to your blob storage account name, **mycontainer** to the container name, **storage account key** to your blob storage access key. You can find your storage account credentials in [Azure portal](#).

Run AzCopy command in PowerShell or from a command prompt. Here is some example usage of AzCopy command:

```
# Copy *.sql from local machine to a Azure Blob
"C:\Program Files (x86)\Microsoft SDKs\Azure\AzCopy\azcopy" /Source:"c:\Aaqs\Data Science Scripts" /Dest:https://[ENTER STORAGE ACCOUNT].blob.core.windows.net/[ENTER CONTAINER]/DestKey:[ENTER STORAGE KEY]/S /Pattern:*.sql

# Copy back all files from Azure Blob container to Local machine

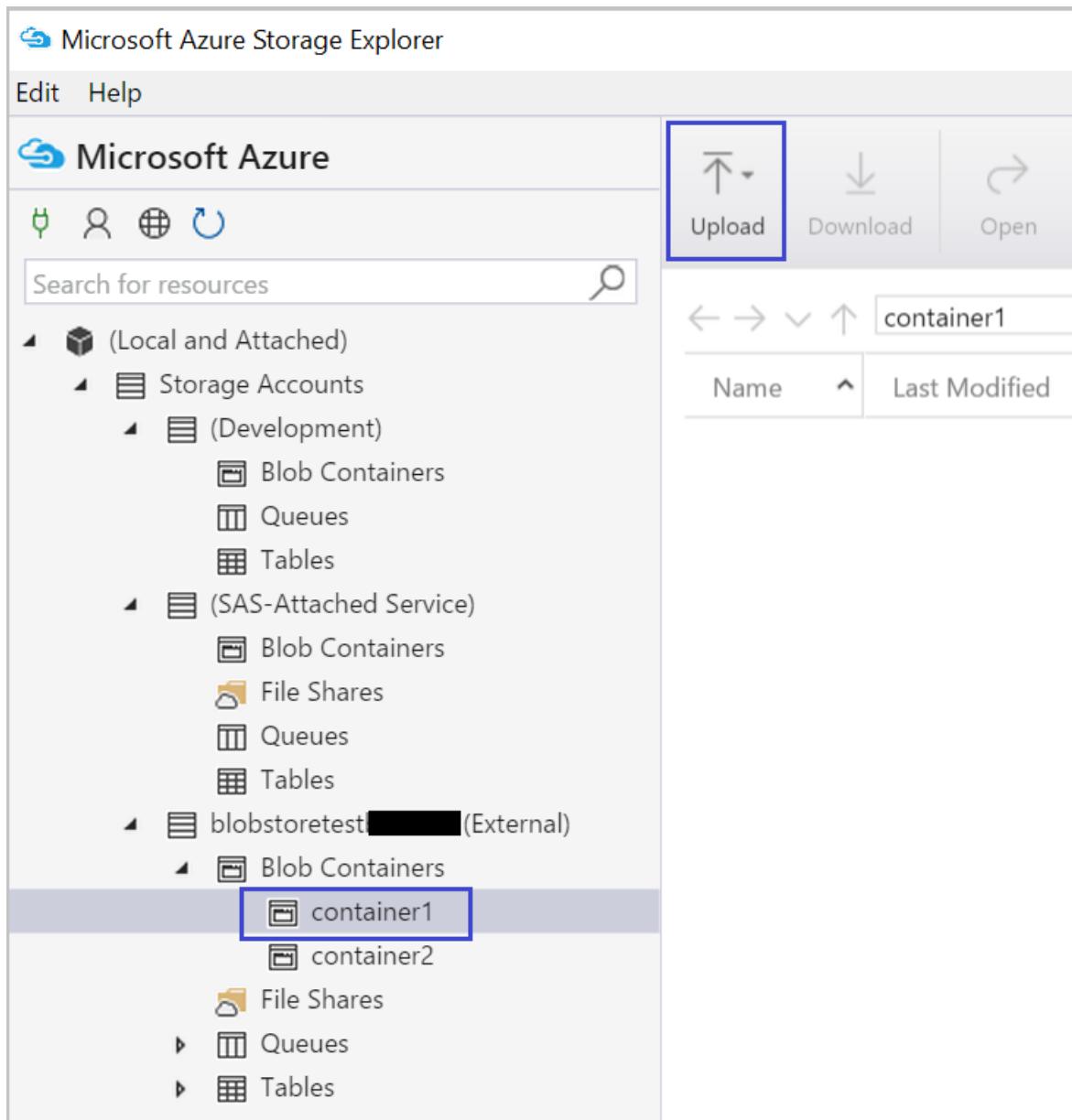
"C:\Program Files (x86)\Microsoft SDKs\Azure\AzCopy\azcopy" /Dest:"c:\Aaqs\Data Science Scripts\temp" /Source:https://[ENTER STORAGE ACCOUNT].blob.core.windows.net/[ENTER CONTAINER]/SourceKey:[ENTER STORAGE KEY]/S
```

Once you run your AzCopy command to copy to an Azure blob you see your file shows up in Azure Storage Explorer shortly.

Move data from VM to Azure Blob: Azure Storage Explorer

You can also upload data from the local file in your VM using Azure Storage Explorer:

- To upload data to a container, select the target container and click the **Upload** button.



- Click on the ... to the right of the **Files** box, select one or more files to upload from the file system and click **Upload** to begin uploading the files.

Upload files

Files

No files selected ...

Blob type

Block Blob ▼

Upload .vhdx files as page blobs (recommended)

Upload to folder (optional)

Upload Cancel

Read data from Azure Blob: Machine Learning reader module

In Azure Machine Learning Studio you can use an **Import Data module** to read data from your blob.

Properties

Reader

Data source: Azure Blob Storage

Authentication type: Account

Account name: [REDACTED]

Account key: [REDACTED]

Path to container, directory or blob: blob1.csv

Blob file format: CSV

File has header row

Use cached results

START TIME: 2/4/2016 12:58:34 PM
END TIME: 2/4/2016 12:58:47 PM
ELAPSED TIME: 0:00:12.500
STATUS CODE: Finished
STATUS DETAILS: None

[View output log](#)

Read data from Azure Blob: Python ODBC

You can use **BlobService** library to read data directly from blob in a Jupyter Notebook or Python program.

First, import required packages:

```

import pandas as pd
from pandas import Series, DataFrame
import numpy as np
import matplotlib.pyplot as plt
from time import time
import pyodbc
import os
from azure.storage.blob import BlobService
import tables
import time
import zipfile
import random

```

Then plug in your Azure Blob account credentials and read data from Blob:

```

CONTAINERNAME = 'xxx'
STORAGEACCOUNTNAME = 'xxx'
STORAGEACCOUNTKEY = 'xxxxxxxxxxxxxx'
BLOBNAME = 'nyctaxidataset/nyctaxitrip/trip_data_1.csv'
localfilename = 'trip_data_1.csv'
LOCALDIRECTORY = os.getcwd()
LOCALFILE = os.path.join(LOCALDIRECTORY, localfilename)

#download from blob
t1 = time.time()
blob_service = BlobService(account_name=STORAGEACCOUNTNAME,account_key=STORAGEACCOUNTKEY)
blob_service.get_blob_to_path(CONTAINERNAME,BLOBNAME,LOCALFILE)
t2 = time.time()
print(("It takes %s seconds to download "+BLOBNAME) % (t2 - t1))

#unzipping downloaded files if needed
#with zipfile.ZipFile(ZIPPEDLOCALFILE, "r") as z:
#    z.extractall(LOCALDIRECTORY)

df1 = pd.read_csv(LOCALFILE, header=0)
df1.columns =
['medallion','hack_license','vendor_id','rate_code','store_and_fwd_flag','pickup_datetime','dropoff_datetime','passenger_count','trip_time_in_secs','trip_distance','pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude']
print('the size of the data is: %d rows and %d columns' % df1.shape

```

The data is read in as a data frame:

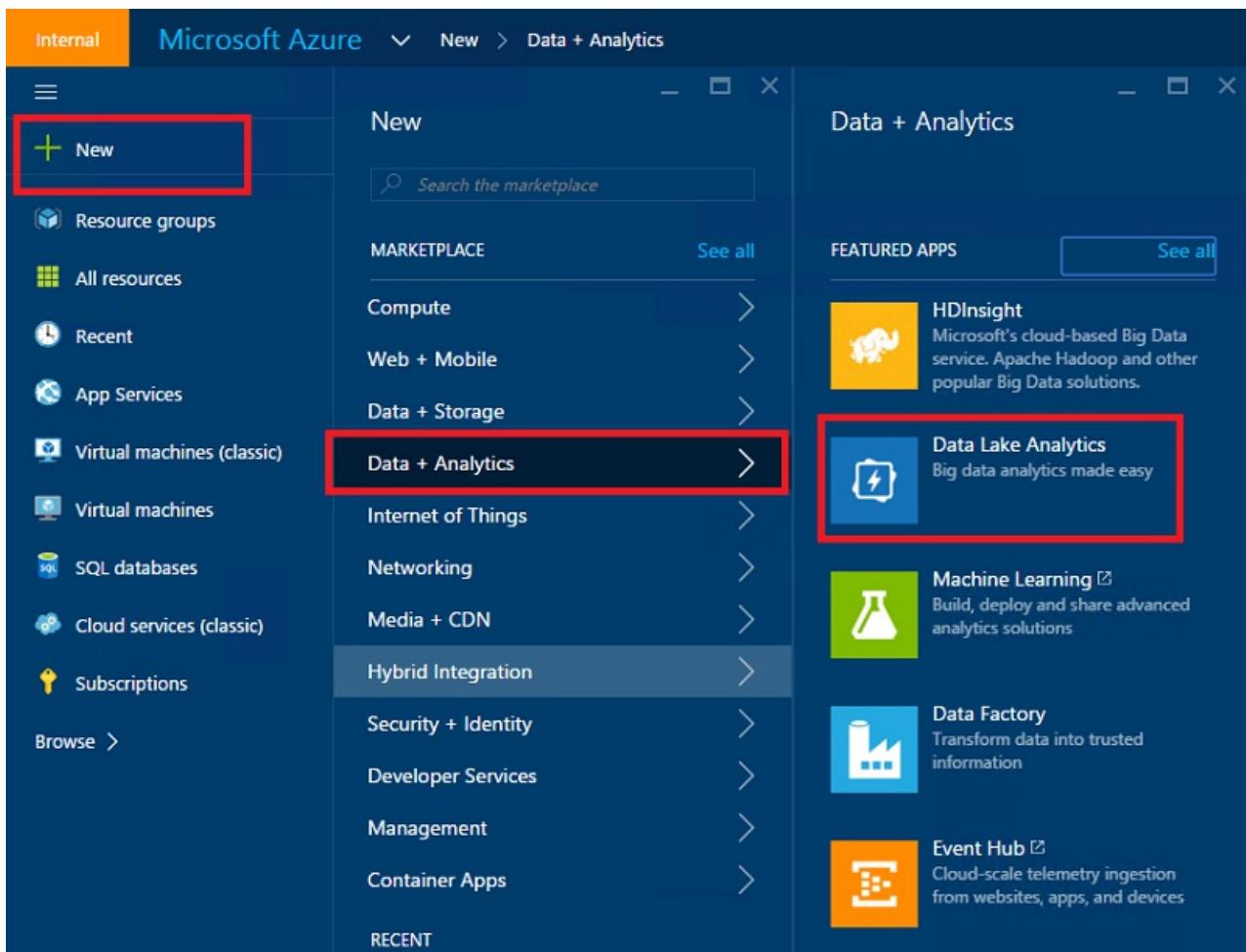
Check the first 10 rows of the data frame							
In [25]:	df1.head(10)						
Out[25]:	medallion	hack_license	vendor_id	rate_code	store_and_fwd_flag	pickup_datetime	dropoff_datetime
0	89D227B655E5C82AECF13C3F540D4CF4	BA96DE419E711691B9445D6A6307C170	CMT	1	N	2013-01-01 15:11:48	
1	0BD7C8F5BA12B88E0B67BED28BEA73D8	9FD8F69F0804BDB5549F40E9DA1BE472	CMT	1	N	2013-01-06 00:18:35	
2	0BD7C8F5BA12B88E0B67BED28BEA73D8	9FD8F69F0804BDB5549F40E9DA1BE472	CMT	1	N	2013-01-05 18:49:41	

Azure Data Lake

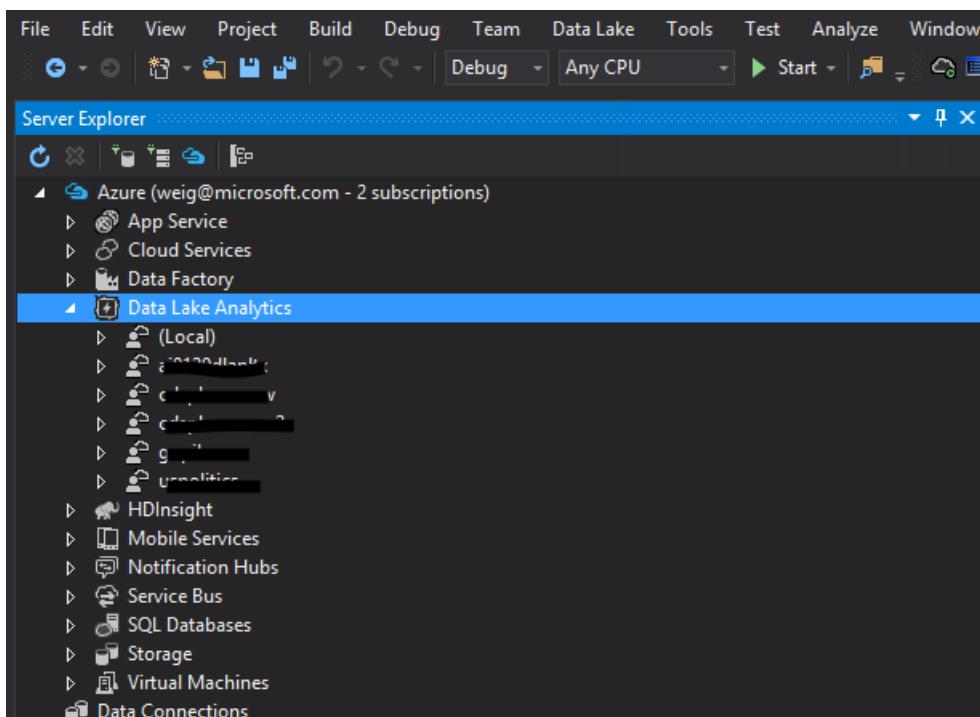
Azure Data Lake Storage is a hyper-scale repository for big data analytics workloads and compatible with Hadoop Distributed File System (HDFS). It works with both the Hadoop ecosystem and the Azure Data Lake Analytics. We show how you can move data into the Azure Data Lake Store and run analytics using Azure Data Lake Analytics.

Prerequisite

- Create your Azure Data Lake Analytics in [Azure portal](#).

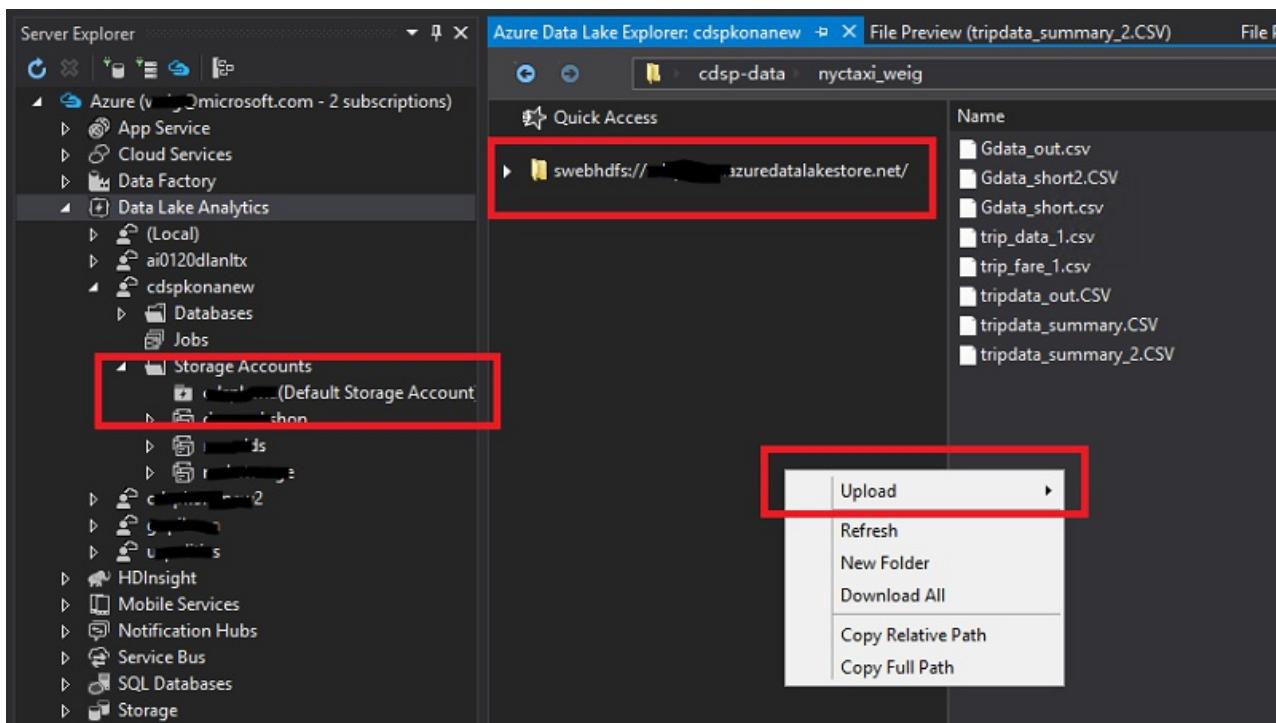


- The **Azure Data Lake Tools** in **Visual Studio** found at this [link](#) is already installed on the Visual Studio Community Edition which is on the virtual machine. After starting Visual Studio and logging in your Azure subscription, you will see your Azure Data Analytics account and storage in the left panel of Visual Studio.



Move data from VM to Data Lake: Azure Data Lake Explorer

You can use **Azure Data Lake Explorer** to upload data from the local files in your Virtual Machine to Data Lake storage.

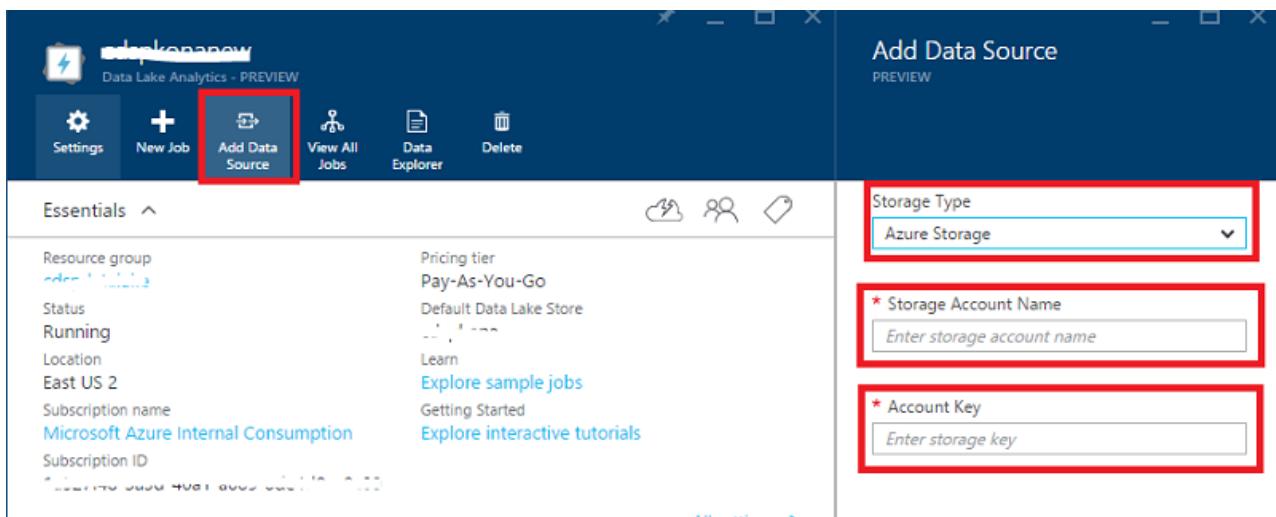


You can also build a data pipeline to productionize your data movement to or from Azure Data Lake using the [Azure Data Factory\(ADF\)](#). We refer you to this [article](#) to guide you through the steps to build the data pipelines.

Read data from Azure Blob to Data Lake: U-SQL

If your data resides in Azure Blob storage, you can directly read data from Azure storage blob in U-SQL query.

Before composing your U-SQL query, make sure your blob storage account is linked to your Azure Data Lake. Go to [Azure portal](#), find your Azure Data Lake Analytics dashboard, click **Add Data Source**, select storage type to **Azure Storage** and plug in your Azure Storage Account Name and Key. Then you will be able to reference the data stored in the storage account.



In Visual Studio, you can read data from blob storage, do some data manipulation, feature engineering, and output the resulting data to either Azure Data Lake or Azure Blob Storage. When you reference the data in blob storage, use **wasb://**; when you reference the data in Azure Data Lake, use **swebhdfs://**

```
Submit (Local) master dbo
@a =
    EXTRACT medallion string,
    hack_license string,
    vendor_id string,
    rate_code string,
    store_and_fwd_flag string,
    pickup_datetime string,
    dropoff_datetime string,
    passenger_count int,
    trip_time_in_secs double,
    trip_distance double,
    pickup_longitude string,
    pickup_latitude string,
    dropoff_longitude string,
    dropoff_latitude string

    FROM "wasb://<Container name>@<Azure Blob Storage Account Name>.blob.core.windows.net/<Input Data File Name>" 
    USING Extractors.Csv();

@b =
    SELECT vendor_id,
    COUNT(medallion) AS cnt_medallion,
    SUM(passenger_count) AS cnt_passenger,
    AVG(trip_distance) AS avg_trip_dist,
    MIN(trip_distance) AS min_trip_dist,
    MAX(trip_distance) AS max_trip_dist,
    AVG(trip_time_in_secs) AS avg_trip_time
    FROM @a
    GROUP BY vendor_id;

OUTPUT @b
TO "swebhdfs://<Azure Data Lake Storage Account Name>.azuredatalakestore.net/<Folder Name>/<Output Data File Name>" 
USING Outputters.Csv();

OUTPUT @b
TO "wasb://<Container name>@<Azure Blob Storage Account Name>.blob.core.windows.net/<Output Data File Name>" 
USING Outputters.Csv();
```

You may use the following U-SQL queries in Visual Studio:

```

@a =
EXTRACT medallion string,
    hack_license string,
    vendor_id string,
    rate_code string,
    store_and_fwd_flag string,
    pickup_datetime string,
    dropoff_datetime string,
    passenger_count int,
    trip_time_in_secs double,
    trip_distance double,
    pickup_longitude string,
    pickup_latitude string,
    dropoff_longitude string,
    dropoff_latitude string

```

FROM "wasb://<Container name>@<Azure Blob Storage Account Name>.blob.core.windows.net/<Input Data File Name>"
USING Extractors.Csv();

```

@b =
SELECT vendor_id,
COUNT(medallion) AS cnt_medallion,
SUM(passenger_count) AS cnt_passenger,
AVG(trip_distance) AS avg_trip_dist,
MIN(trip_distance) AS min_trip_dist,
MAX(trip_distance) AS max_trip_dist,
AVG(trip_time_in_secs) AS avg_trip_time
FROM @a
GROUP BY vendor_id;

OUTPUT @b
TO "swebhdfs://<Azure Data Lake Storage Account Name>.azuredatalakestore.net/<Folder Name>/<Output Data File Name>"  

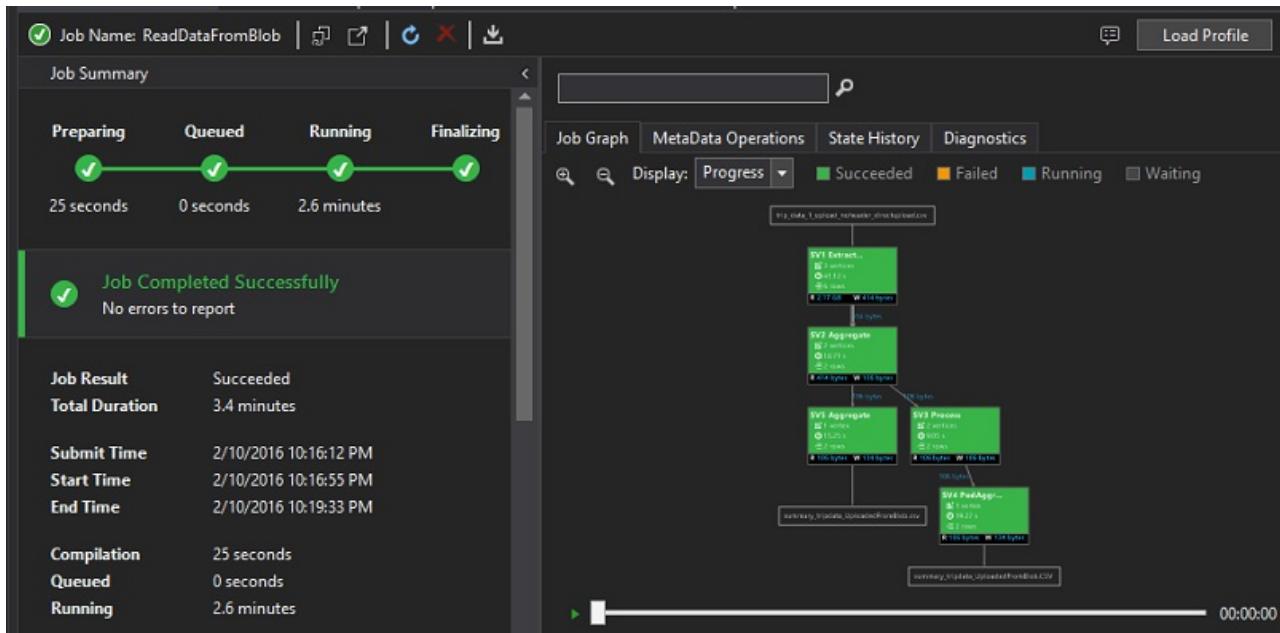
USING Outputters.Csv();

OUTPUT @b
TO "wasb://<Container name>@<Azure Blob Storage Account Name>.blob.core.windows.net/<Output Data File Name>"  

USING Outputters.Csv();

```

After your query is submitted to the server, a diagram showing the status of your job will be displayed.



Query data in Data Lake: U-SQL

After the dataset is ingested into Azure Data Lake, you can use [U-SQL language](#) to query and explore the data. U-SQL language is similar to T-SQL, but combines some features from C# so that users can write customized

modules, User Defined Functions, and etc. You can use the scripts in the previous step.

After the query is submitted to server, tripdata_summary.CSV can be found shortly in **Azure Data Lake Explorer**, you may preview the data by right-click the file.

The screenshot shows the Azure Data Lake Explorer interface. The left sidebar has a 'Quick Access' section and a connection to 'swebhdfs://cdspkona.azuredatalakestore.net/'. The main area displays a table of files with columns: Name, File Size (Logical), and Modified. The file 'tripdata_summary.CSV' is selected and highlighted in blue. The table data is as follows:

Name	File Size (Logical)	Modified
Gdata_out.csv	99 bytes	1/28/2016 9:41:11 AM
Gdata_short2.CSV	91,327 bytes	1/30/2016 12:44:03 AM
Gdata_short.csv	91,327 bytes	1/28/2016 9:29:05 AM
trip_data_1.csv	2,345 MB	1/28/2016 7:13:13 AM
trip_fare_1.csv	1,603 MB	1/28/2016 7:12:09 AM
tripdata_out.CSV	2,345 MB	2/1/2016 8:45:08 AM
tripdata_summary.CSV	121 bytes	2/1/2016 5:57:09 PM
tripdata_summary_2.CSV	121 bytes	2/1/2016 9:47:39 PM

To see the file information:

The screenshot shows the 'File Information' panel for the 'tripdata_summary.CSV' file. It displays the following details:

- Name: tripdata_summary
- Modified: 2/1/2016 5:57:09 PM
- File Size: 121 bytes
- Path: swebhdfs://[REDACTED].azuredatalakestore.net/cdsp-data/nytaxi_weight/tripdata_summary.CSV

Below this, there is a preview of the top 2 rows of the CSV file:

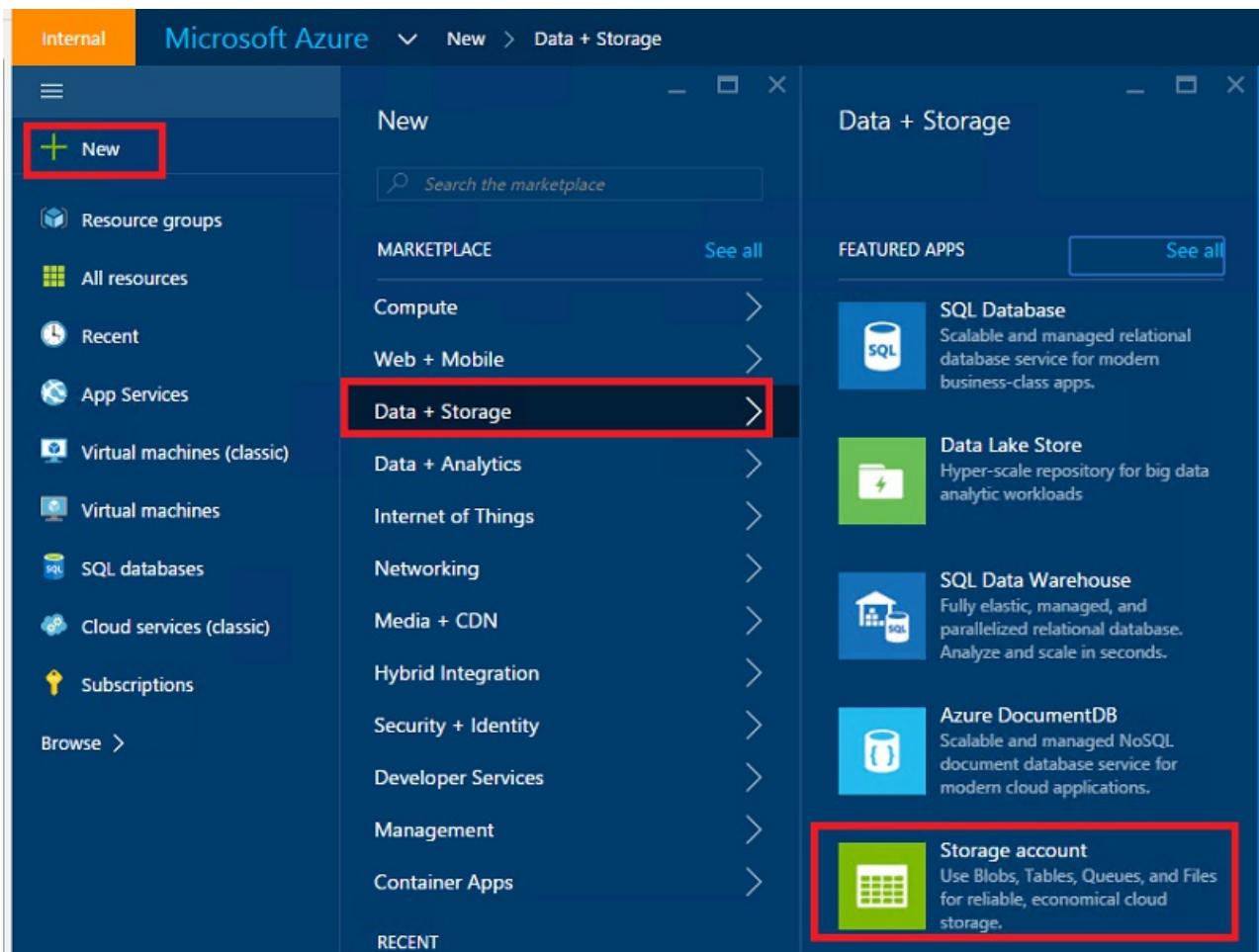
	Column_0	Column_1	Column_2	Column_3	Column_4
1	"CMT"	7450899	9432635	2.72750974077074	677.502435209496
2	"VTS"	7325716	15648779	2.81518435331084	689.445932656958

HDInsight Hadoop Clusters

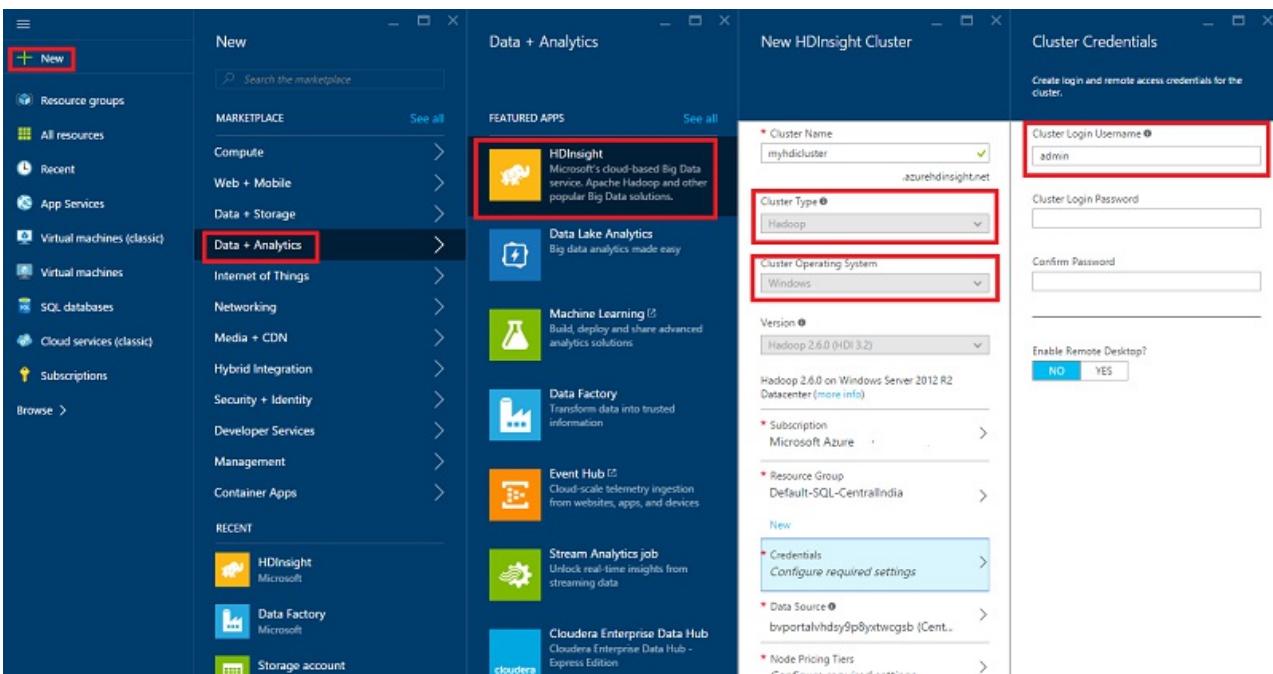
Azure HDInsight is a managed Apache Hadoop, Spark, HBase, and Storm service on the cloud. You can work easily with Azure HDInsight clusters from the data science virtual machine.

Prerequisite

- Create your Azure Blob storage account from [Azure portal](#). This storage account is used to store data for HDInsight clusters.



- Customize Azure HDInsight Hadoop Clusters from [Azure portal](#)
 - You must link the storage account created with your HDInsight cluster when it is created. This storage account is used for accessing data that can be processed within the cluster.



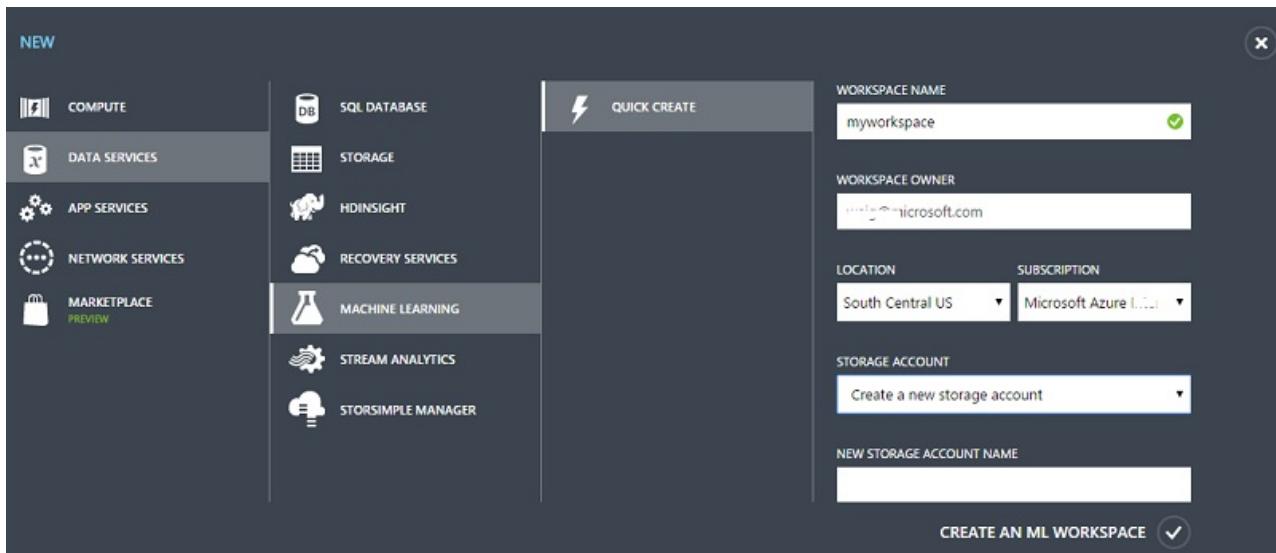
- You must enable **Remote Access** to the head node of the cluster after it is created. Remember the remote access credentials you specify here (different from those specified for the cluster at its creation): you will need them below.

The screenshot shows the Azure portal interface for managing an HDInsight cluster named 'weighdi'. The top navigation bar includes buttons for Settings, Dashboard, Remote Desktop (which is highlighted with a red box), Scale Cluster, Delete, and Move. Below the navigation bar, there's an 'Essentials' section with cluster details like URL, Cluster Type (Hadoop on Windows), and Location (West US). A 'Quick Links' section contains links for Cluster Dashboard, Documentation, and Scale Cluster, with 'Remote Desktop' also highlighted with a red box.

- Create an Azure Machine Learning workspace. Your Machine Learning Experiments will be stored in this Machine Learning workspace. Select the highlighted options in Portal as shown in the screenshot below.

The screenshot shows the Azure portal's 'New' blade. On the left, a sidebar lists options like Resource groups, All resources, Recent, App Services, Virtual machines (classic), Virtual machines, SQL databases, Cloud services (classic), Subscriptions, and Browse. The main area is titled 'New' and shows the 'Marketplace' search bar. Under 'MARKETPLACE', categories include Compute, Web + Mobile, Data + Storage, and Data + Analytics (which is highlighted with a red box). Under 'FEATURED APPS', it lists HDInsight, Data Lake Analytics, Machine Learning (which is highlighted with a red box), and Data Factory.

- Then enter the parameters for your workspace



- Upload data using IPython Notebook. First import required packages, plug in credentials, create a db in your storage account, then load data to HDI clusters.

```
#Import required Packages
import pyodbc
import time as time
import json
import os
import urllib
import urllib2
import warnings
import re
import pandas as pd
import matplotlib.pyplot as plt
from azure.storage.blob import BlobService
warnings.filterwarnings("ignore", category=UserWarning, module='urllib2')
```

```
#Create the connection to Hive using ODBC
SERVER_NAME='xxx.azurehdinsight.net'
DATABASE_NAME='nyctaxidb'
USERID='xx'
PASSWORD='xx'
DB_DRIVER='Microsoft Hive ODBC Driver'
driver = 'DRIVER={' + DB_DRIVER + '}'
server = 'Host=' + SERVER_NAME + ';Port=443'
database = 'Schema=' + DATABASE_NAME
hiveserv = 'HiveServerType=2'
auth = 'AuthMech=6'
uid = 'UID=' + USERID
pwd = 'PWD=' + PASSWORD
CONNECTION_STRING='';join([driver,server,database,hiveserv,auth,uid,pwd])
connection = pyodbc.connect(CONNECTION_STRING, autocommit=True)
cursor=connection.cursor()
```

```
#Create Hive database and tables
queryString = "create database if not exists nyctaxidb;"
```

```
cursor.execute(queryString)

queryString = """
    create external table if not exists nyctaxidb.trip
    (
        medallion string,
        hack_license string,
        vendor_id string,
        rate_code string,
        store_and_fwd_flag string,
```

```

pickup_datetime string,
dropoff_datetime string,
passenger_count int,
trip_time_in_secs double,
trip_distance double,
pickup_longitude double,
pickup_latitude double,
dropoff_longitude double,
dropoff_latitude double)
PARTITIONED BY(month int)
ROW FORMAT DELIMITED FIELDS TERMINATED BY',' lines terminated by '\n'
STORED AS TEXTFILE LOCATION 'wasb:///nyctaxidbdata/trip' TBLPROPERTIES('skip.header.line.count'=1);
"""

cursor.execute(queryString)

queryString = """
create external table if not exists nyctaxidb.fare
(
    medallion string,
    hack_license string,
    vendor_id string,
    pickup_datetime string,
    payment_type string,
    fare_amount double,
    surcharge double,
    mta_tax double,
    tip_amount double,
    tolls_amount double,
    total_amount double)
PARTITIONED BY(month int)
ROW FORMAT DELIMITED FIELDS TERMINATED BY',' lines terminated by '\n'
STORED AS TEXTFILE LOCATION 'wasb:///nyctaxidbdata/fare' TBLPROPERTIES('skip.header.line.count'=1);
"""

cursor.execute(queryString)

#Upload data from blob storage to HDI cluster
for i in range(1,13):
    queryString = "LOAD DATA INPATH 'wasb:///nyctaxitripraw2/trip_data_%d.csv' INTO TABLE nyctaxidb2.trip PARTITION (month=%d);%"%(i,i)
    cursor.execute(queryString)
    queryString = "LOAD DATA INPATH 'wasb:///nyctaxifareraw2/trip_fare_%d.csv' INTO TABLE nyctaxidb2.fare PARTITION (month=%d);%"%(i,i)
    cursor.execute(queryString)

```

- Alternately, you can follow this [walkthrough](#) to upload NYC Taxi data to HDI cluster. Major steps include:
 - AzCopy: download zipped CSV's from public blob to your local folder
 - AzCopy: upload unzipped CSV's from local folder to HDI cluster
 - Log into the head node of Hadoop cluster and prepare for exploratory data analysis

After the data is loaded to HDI cluster, you can check your data in Azure Storage Explorer. And you have a database nyctaxidb created in HDI cluster.

Data exploration: Hive Queries in Python

Since the data is in Hadoop cluster, you can use the pyodbc package to connect to Hadoop Clusters and query database using Hive to do exploration and feature engineering. You can view the existing tables we created in the prerequisite step.

```

queryString = """
    show tables in nyctaxidb2;
"""

pd.read_sql(queryString,connection)

```

Out[6]:	tab_name
0	fare
1	trip

Let's look at the number of records in each month and the frequencies of tipped or not in the trip table:

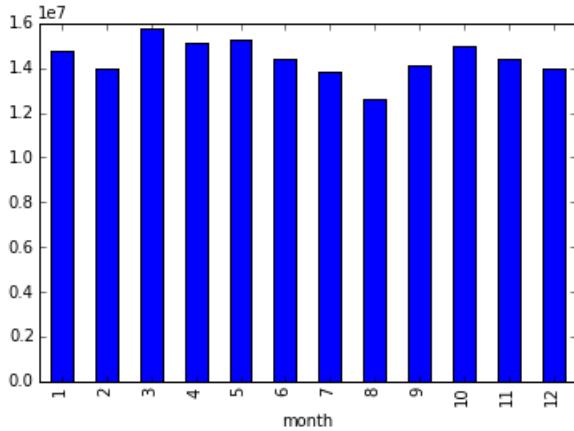
```
queryString = """
    select month, count(*) from nyctaxidb.trip group by month;
"""

results = pd.read_sql(queryString,connection)

%matplotlib inline

results.columns = ['month', 'trip_count']
df=results.copy()
df.index= df['month']
df['trip_count'].plot(kind='bar')
```

Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0xa7d3f28>

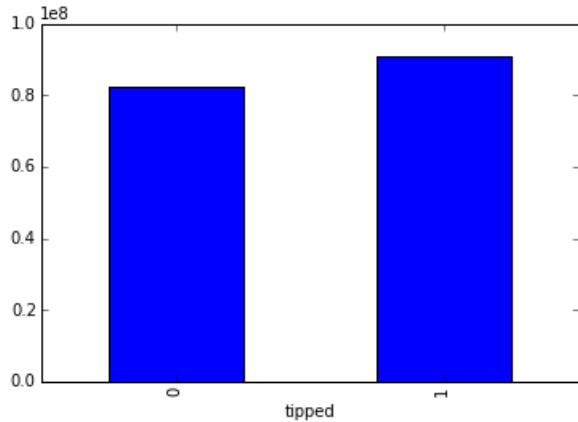


```
queryString = """
    SELECT tipped, COUNT(*) AS tip_freq
    FROM
    (
        SELECT if(tip_amount > 0, 1, 0) as tipped, tip_amount
        FROM nyctaxidb.fare
    )tc
    GROUP BY tipped;
"""

results = pd.read_sql(queryString,connection)

results.columns = ['tipped', 'trip_count']
df=results.copy()
df.index= df['tipped']
df['trip_count'].plot(kind='bar')
```

```
Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0xebbf5320>
```



We can also compute the distance between pickup location and dropoff location and then compare it to the trip distance.

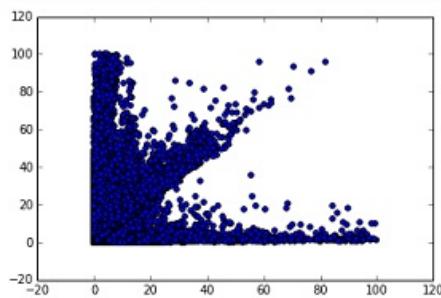
```
queryString = """
    select pickup_longitude, pickup_latitude, dropoff_longitude, dropoff_latitude, trip_distance, trip_time_in_secs,
    3959*2*2*atan((1-sqrt(1-pow(sin((dropoff_latitude-pickup_latitude)*radians(180/180/2)),2))
    *radians(180/180/2),2)-cos(pickup_latitude*radians(180/180/2))
    *cos(dropoff_latitude*radians(180/180/2))*pow(sin((dropoff_longitude-pickup_longitude)*radians(180/180/2),2)))
    /sqrt(pow(sin((dropoff_latitude-pickup_latitude)*radians(180/180/2),2)
    +cos(pickup_latitude*radians(180/180/2))*cos(dropoff_latitude*radians(180/180/2))
    *pow(sin((dropoff_longitude-pickup_longitude)*radians(180/180/2),2))) as direct_distance
    from nyctaxidb.trip
    where month=1
        and pickup_longitude between -90 and -30
        and pickup_latitude between 30 and 90
        and dropoff_longitude between -90 and -30
        and dropoff_latitude between 30 and 90;
"""

results = pd.read_sql(queryString,connection)
results.head(5)
```

Out[19]:	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	trip_distance	trip_time_in_secs	direct_distance
0	-73.971481	40.757412	-73.971840	40.750305	0.84	180	0.491436
1	-73.923576	40.798939	-73.956322	40.781368	3.56	420	2.099697
2	-73.981194	40.757748	-73.970009	40.754097	1.04	360	0.637479
3	-73.989616	40.757706	-73.970909	40.783096	2.21	480	2.009027
4	-73.995491	40.744186	-73.979599	40.777721	3.10	600	2.461942

```
results.columns = ['pickup_longitude', 'pickup_latitude', 'dropoff_longitude',
                   'dropoff_latitude', 'trip_distance', 'trip_time_in_secs', 'direct_distance']
df = results.loc[results['trip_distance']<=100] #remove outliers
df = df.loc[df['direct_distance']<=100] #remove outliers
plt.scatter(df['direct_distance'], df['trip_distance'])
```

```
Out[20]: <matplotlib.collections.PathCollection at 0xf3e47f0>
C:\Anaconda\lib\site-packages\matplotlib\collections.py:571: FutureWarning: elementwise comparison failed; returning scalar instead, but in the future will perform elementwise comparison
  if self._edgecolors == str('face')):
```



Now let's prepare a down-sampled (1%) set of data for modeling. We can use this data in Machine Learning reader module.

```
queryString = """
create table if not exists nyctaxi_downsampled_dataset_testNEW (
    medallion string,
    hack_license string,
    vendor_id string,
    rate_code string,
    store_and_fwd_flag string,
    pickup_datetime string,
    dropoff_datetime string,
    pickup_hour string,
    pickup_week string,
    weekday string,
    passenger_count int,
    trip_time_in_secs double,
    trip_distance double,
    pickup_longitude double,
    pickup_latitude double,
    dropoff_longitude double,
    dropoff_latitude double,
    direct_distance double,
    payment_type string,
    fare_amount double,
    surcharge double,
    mta_tax double,
    tip_amount double,
    tolls_amount double,
    total_amount double,
    tipped string,
    tip_class string
)
row format delimited fields terminated by ','
lines terminated by '\n'
stored as textfile;
"""

cursor.execute(queryString)
```

```
--- now insert contents of the join into the above internal table
```

```
queryString = """
insert overwrite table nyctaxi_downsampled_dataset_testNEW
select
    t.medallion,
    t.hack_license,
    t.vendor_id,
    t.rate_code,
    t.store_and_fwd_flag,
    t.pickup_datetime,
    t.dropoff_datetime,
    hour(t.pickup_datetime) as pickup_hour,
    weekofyear(t.pickup_datetime) as pickup_week,
```

```

from_unixtime(unix_timestamp(t.pickup_datetime,'yyyy-MM-dd HH:mm:ss'),'u') as weekday,
t.passenger_count,
t.trip_time_in_secs,
t.trip_distance,
t.pickup_longitude,
t.pickup_latitude,
t.dropoff_longitude,
t.dropoff_latitude,
t.direct_distance,
f.payment_type,
f.fare_amount,
f.surcharge,
f.mta_tax,
f.tip_amount,
f.tolls_amount,
f.total_amount,
if(tip_amount>0,1,0) as tipped,
if(tip_amount=0,0,
if(tip_amount>0 and tip_amount<=5,1,
if(tip_amount>5 and tip_amount<=10,2,
if(tip_amount>10 and tip_amount<=20,3,4))) as tip_class
from
(
select
medallion,
hack_license,
vendor_id,
rate_code,
store_and_fwd_flag,
pickup_datetime,
dropoff_datetime,
passenger_count,
trip_time_in_secs,
trip_distance,
pickup_longitude,
pickup_latitude,
dropoff_longitude,
dropoff_latitude,
3959*2*atan((1-sqrt(1-pow(sin((dropoff_latitude-pickup_latitude)*radians(180)/180)
*radians(180)/180/2),2)-cos(pickup_latitude*radians(180)/180)
*cos(dropoff_longitude*radians(180)/180)*pow(sin((dropoff_longitude-pickup_longitude)*radians(180)/180/2),2)))
/sqrt(pow(sin((dropoff_latitude-pickup_latitude)*radians(180)/180/2),2)
+cos(pickup_latitude*radians(180)/180)*cos(dropoff_latitude*radians(180)/180)*pow(sin((dropoff_longitude-
pickup_longitude)*radians(180)/180/2),2))) as direct_distance,
rand() as sample_key
from trip
where pickup_latitude between 30 and 90
and pickup_longitude between -90 and -30
and dropoff_latitude between 30 and 90
and dropoff_longitude between -90 and -30
)t
join
(
select
medallion,
hack_license,
vendor_id,
pickup_datetime,
payment_type,
fare_amount,
surcharge,
mta_tax,
tip_amount,
tolls_amount,
total_amount
from fare
) f
on t.medallion=f.medallion and t.hack_license=f.hack_license and t.pickup_datetime=f.pickup_datetime

```

```

where t.sample_key<=0.01
"""
cursor.execute(queryString)

```

After a while, you can see the data has been loaded in Hadoop clusters:

```

queryString = """
select * from nyctaxi_downsampled_dataset limit 10;
"""

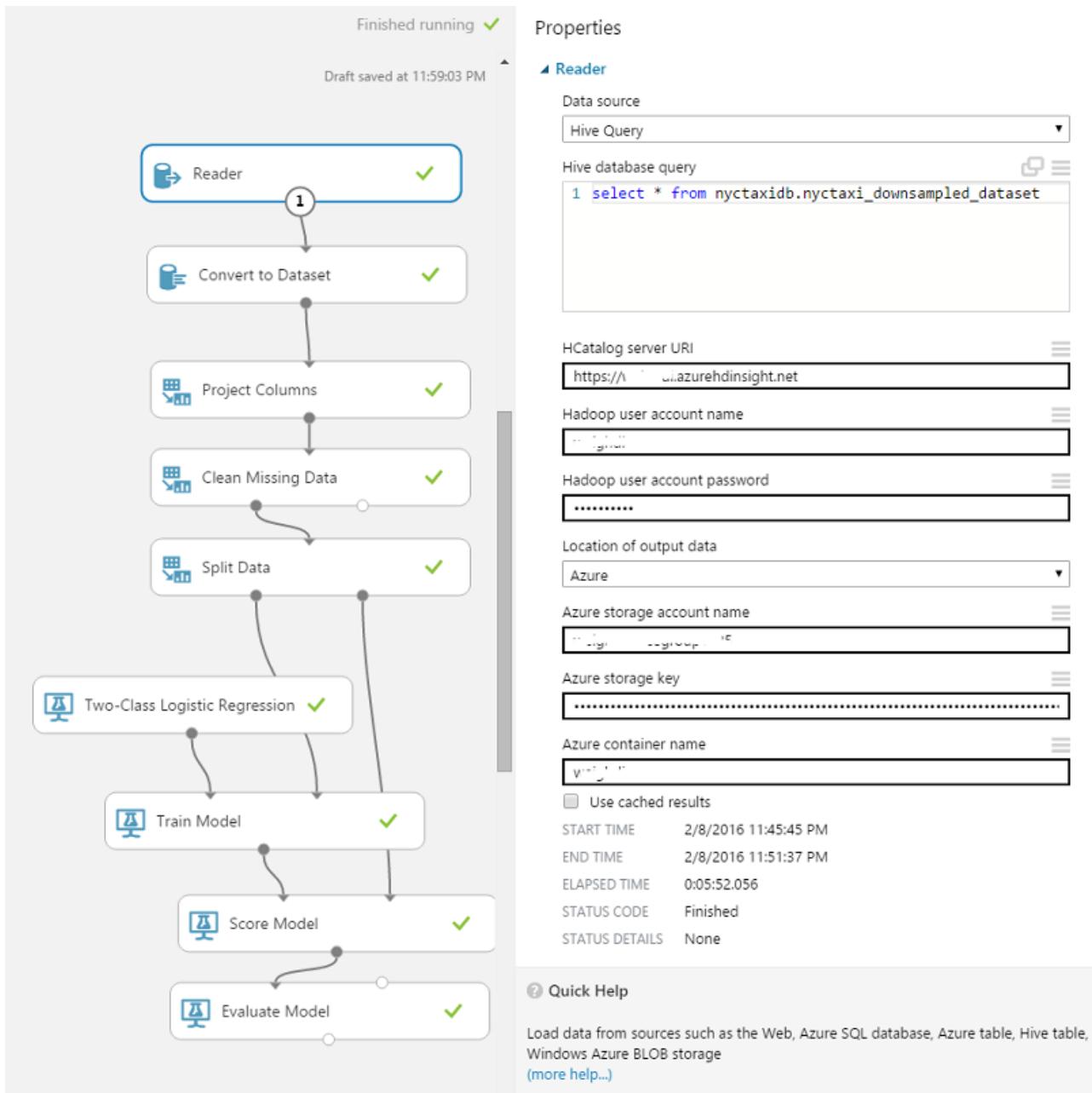
cursor.execute(queryString)
pd.read_sql(queryString,connection)

```

Out[62]:	medallion	hack_license	vendor_id	rate_code	store_and_fwd_flag	pickup_datetime
0	0009986BDBAB2F9A125FEF49D0BFCCDD	EB0BF47D51268910F10C88DC001A759B	CMT	1	N	2013-08-07 15:26:56
1	002B4CFC5B8920A87065FC131F9732D1	65938970D001F7280B2DDF268EA81D2D	VTS	1		2013-02-22 16:33:00
2	003D87DB553C6F00F774C8575BC8444A	C61317837B26954B50CEAE75D70C47BA	VTS	1		2013-06-08 21:00:00
3	0053334C798EC6C8E637657962030F99	AC901C98FCFA6109186500A50E95AEDE	VTS	2		2013-10-14 09:53:00
4	0094A03FFE6BAFBE0B966B1445B9B50B	F378DD401CBC0F5DAA9A02850B4350E6	VTS	1		2013-04-03 10:50:00
5	009D3CCA83486B03FCE736A2F642CBA8	8D9F1C84782FB48F7A32A702349F8C2E	VTS	1		2013-07-03 22:00:00
6	00A1EA0E8CD47CE24F3FE5444ACE549C	4FD770C068437BBA91947510C3A61DE7	CMT	1	N	2013-09-05 21:37:43
7	00A74F4CAB098B3A345A580BFCEEA467	A2B580922A9DC7AB01A93189B1C84076	VTS	1		2013-09-13 11:00:00

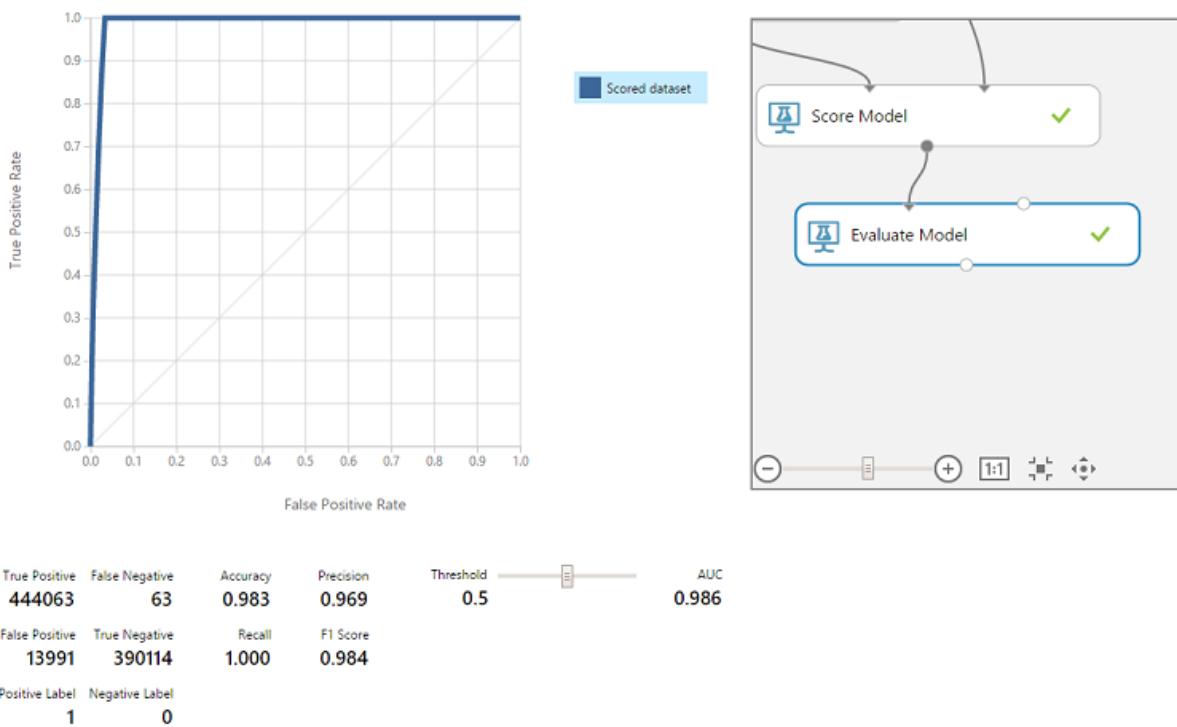
Read data from HDI using Machine Learning: reader module

You may also use the **reader** module in Machine Learning Studio to access the database in Hadoop cluster. Plug in the credentials of your HDI clusters and Azure Storage Account and you will be able to build machine learning models using database in HDI clusters.



The scored dataset can then be viewed:

ROC PRECISION/RECALL LIFT



Azure SQL Data Warehouse & databases

Azure SQL Data Warehouse is an elastic data warehouse as a service with enterprise-class SQL Server experience.

You can provision your Azure SQL Data Warehouse by following the instructions provided in this [article](#). Once you provision your Azure SQL Data Warehouse, you can use this [walkthrough](#) to do data upload, exploration and modeling using data within the SQL Data Warehouse.

Azure DocumentDB

Azure DocumentDB is a NoSQL database in the cloud. It allows you to work with documents like JSON and allows you to store and query the documents.

You need to do the following per-requisites steps to access DocumentDB from the DSVM.

1. Install DocumentDB Python SDK (Run `pip install pydocumentdb` from command prompt)
2. Create DocumentDB account and Document DB database from [Azure portal](#)
3. Download "DocumentDB Migration Tool" from [here](#) and extract to a directory of your choice
4. Import JSON data (volcano data) stored on a [public blob](#) into DocumentDB with following command parameters to the migration tool (dtui.exe from the directory where you installed the DocumentDB Migration Tool). Enter the source and target location parameters from below.

```
/s:JsonFile /s:Files:https://cahandson.blob.core.windows.net/samples/volcano.json /t:DocumentDBBulk  
/t.ConnectionString:AccountEndpoint=https://[DocDBAccountName].documents.azure.com:443;/AccountKey  
=[[KEY];Database=volcano /t.Collection:volcano1
```

Once you import the data, you can go to Jupyter and open the notebook titled *DocumentDBSample* which contains python code to access DocumentDB and do some basic querying. You can learn more about DocumentDB by visiting the service [documentation page](#)

8. Build reports and dashboard using the Power BI Desktop

Let us visualize the Volcano JSON file we saw in the DocumentDB example above in Power BI to gain visual insights into the data. Detailed steps are available in the [Power BI article](#). The high level steps are below :

1. Open Power BI Desktop and do "Get Data". Specify the URL as:
<https://cahandson.blob.core.windows.net/samples/volcano.json>
2. You should see the JSON records imported as a list
3. Convert the list to a table so Power BI can work with the same
4. Expand the columns by clicking on the expand icon (the one with the "left arrow and a right arrow" icon on the right of the column)
5. Notice that location is a "Record" field. Expand the record and select only the coordinates. Coordinate is a list column
6. Add a new column to convert the list coordinate column into a comma separate LatLong column concatenating the two elements in the coordinate list field using the formula
`Text.From([coordinates]{1})&","&Text.From([coordinates]{0})`.
7. Finally convert the `Elevation` column to Decimal and select the **Close** and **Apply**.

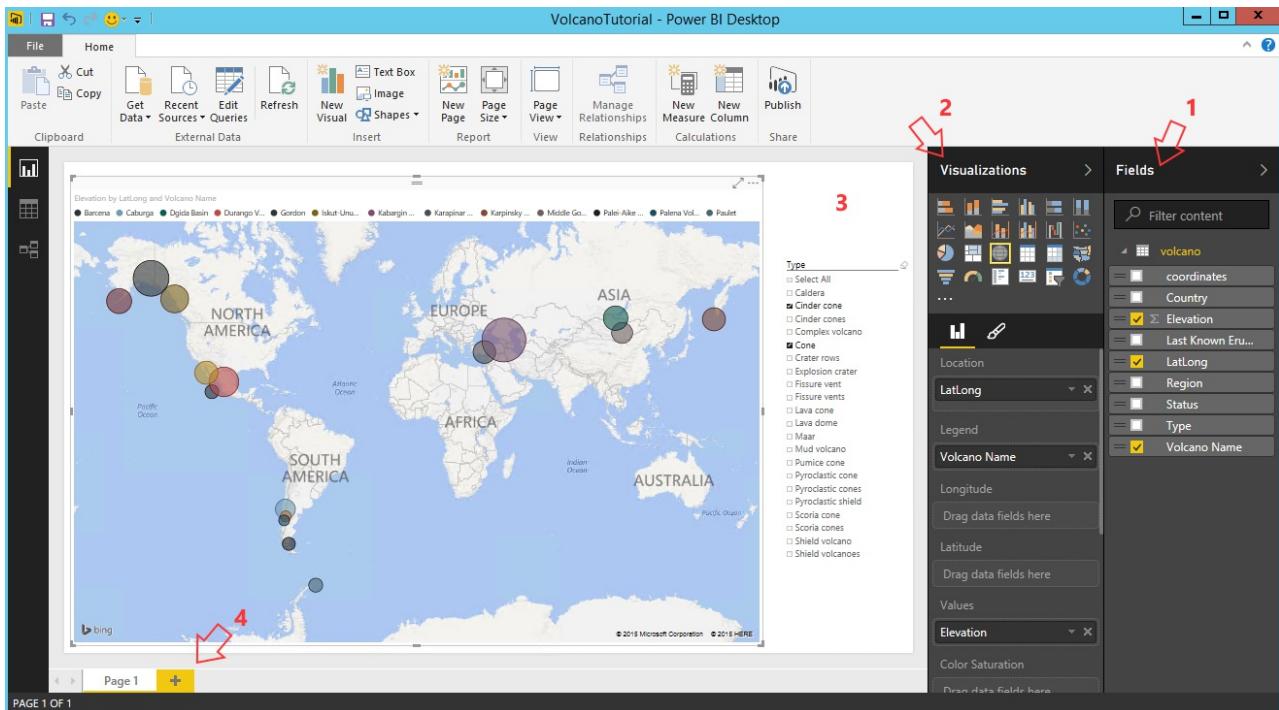
Instead of steps above, you can paste the following code that scripts out the steps above in the Advanced Editor in Power BI that allows you to write the data transformations in a query language.

```
let
    Source = Json.Document(Web.Contents("https://cahandson.blob.core.windows.net/samples/volcano.json")),
    #"Converted to Table" = Table.FromList(Source, Splitter.SplitByNothing(), null, null, ExtraValues.Error),
    #"Expanded Column1" = Table.ExpandRecordColumn(#"Converted to Table", "Column1", {"Volcano Name", "Country", "Region", "Location", "Elevation", "Type", "Status", "Last Known Eruption", "id"}, {"Volcano Name", "Country", "Region", "Location", "Elevation", "Type", "Status", "Last Known Eruption", "id"}),
    #"Expanded Location" = Table.ExpandRecordColumn(#"Expanded Column1", "Location", {"coordinates"}, {"coordinates"}),
    #"Added Custom" = Table.AddColumn(#"Expanded Location", "LatLong", each Text.From([coordinates]{1})&","&Text.From([coordinates]{0})),
    #"Changed Type" = Table.TransformColumnTypes(#"Added Custom",{{"Elevation", type number}})
in
    #"Changed Type"
```

You now have the data in your Power BI data model. Your Power BI desktop should look as shown below.

coordinates	Elevation	Type	Status	Last Known Eruption	id
List	947	Stratovolcano	Historical	Last known eruption from 1500-1699, inclusive	d44c94b6-81f8-43b1-afef-fcb1-5
List	949	Caldera	Holocene	Unknown	c3b51fe1-2c57-404e-a74-26d4
List	3477	Complex volcano	Historical	Last known eruption from 1500-1699, inclusive	1f7825f6-b06e-4fb1-c9a-61bf-5
List	-2000	Submarine volcano	Uncertain	Unknown	e404ce23b-105a-4a0b-771-1257
List	-1850	Submarine volcano	Fumarolic	Undated, but probable Holocene eruption	c4fb1c9a-61bf-4d3d-8d7e-9d9
List	458	Caldera	Historical	Last known eruption from 1500-1699, inclusive	7caf5345-9f8b-4fb5-93eb-2c57-404e-a74-26d4
List	-8	Submarine volcanoes	Historical	Last known eruption from 1800-1899, inclusive	544ce23b-105a-4a0b-771-1257
List	2518	Stratovolcano	Radiocarbon	Last known eruption from 1700-1799, inclusive	4f58dc2f-04ad-4fb5-93eb-2c57-404e-a74-26d4
List	529	Fissure vent	Holocene	Undated, but probable Holocene eruption	3bc61029-73a3-43b1-afef-fcb1-5
List	3258	Stratovolcano	Holocene	Undated, but probable Holocene eruption	08a0f8ac-106d-4fb5-93eb-2c57-404e-a74-26d4
List	5050	Stratovolcano	Tephrochronology	Last known eruption B.C. (Holocene)	4f58dc2f-04ad-4fb5-93eb-2c57-404e-a74-26d4
List	-177	Submarine volcano	Historical	Last known eruption in 1964 or later	3bc61029-73a3-43b1-afef-fcb1-5
List	670	Fissure vent	Historical	Last known eruption from 1800-1899, inclusive	3a306294-2a5c-4fb5-93eb-2c57-404e-a74-26d4
List	2291	Stratovolcano	Historical	Last known eruption in 1964 or later	08a0f8ac-106d-4fb5-93eb-2c57-404e-a74-26d4

You can start building reports and visualizations using the data model. You can follow the steps in this [Power BI article](#) to build a report. The end result will be a report that looks like the following.



9. Dynamically scale your DSVM to meet your project needs

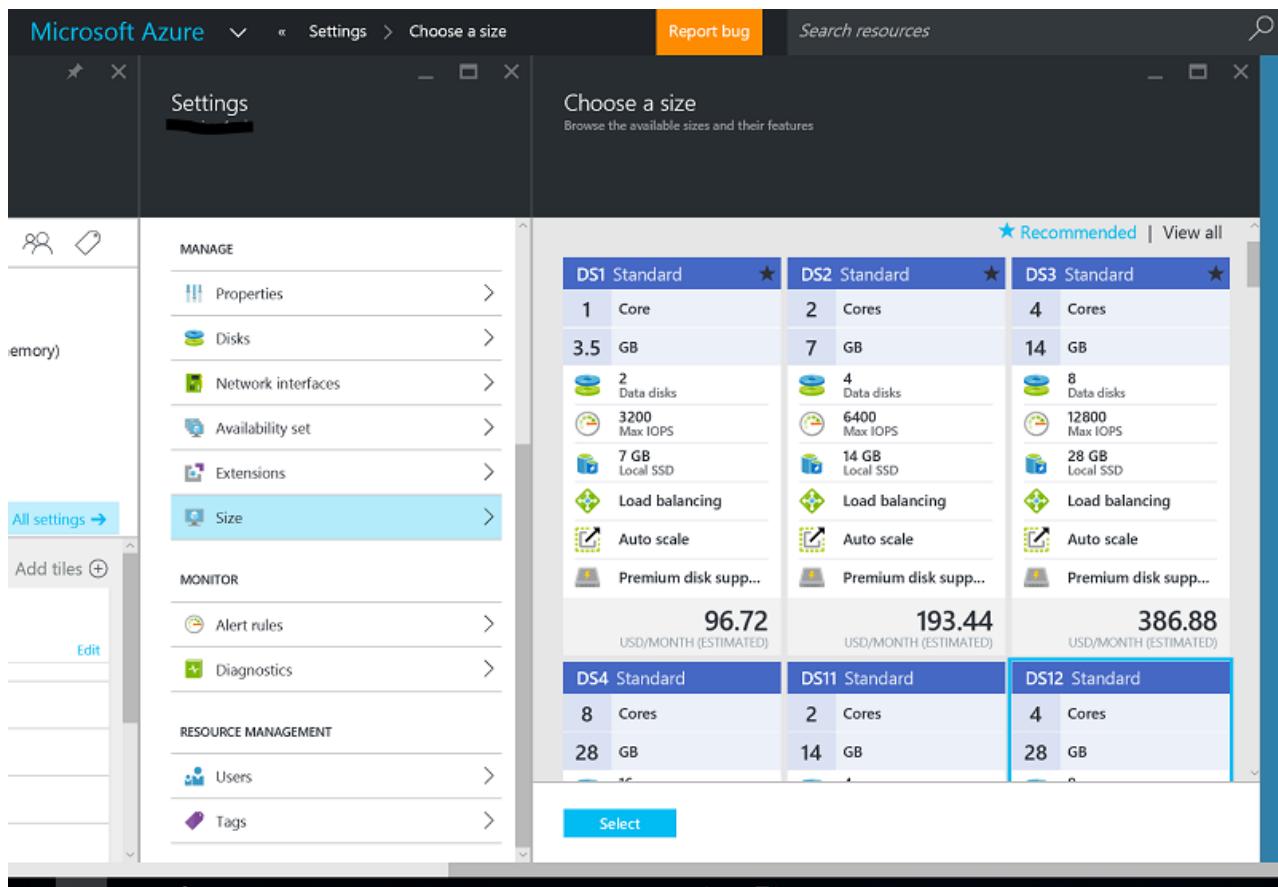
You can scale up and down the DSVM to meet your project needs. If you don't need to use the VM in the evening or weekends, you can just shut down the VM from the [Azure portal](#).

NOTE

You will incur compute charges if you use just the Operating system shutdown button on the VM.

If you need to handle some large scale analysis and need more CPU and/or memory and/or disk capacity you can find a large choice of VM sizes in terms of CPU cores, memory capacity and disk types (including Solid state drives) that meet your compute and budgetary needs. The full list of VMs along with their hourly compute pricing is available on the [Azure Virtual Machines Pricing](#) page.

Similarly, if your need for VM processing capacity reduces (for example: you moved a major workload to a Hadoop or a Spark cluster), you can scale down the cluster from the [Azure portal](#) and going to the settings of your VM instance. Here is a screenshot.



10. Install additional tools on your virtual machine

We have packaged several tools that we believe will be able to address many of the common data analytics needs and that should save you time by avoiding having to install and configure your environments one by one and save you money by paying only for resources that you use.

You can leverage other Azure data and analytics services profiled in this article to enhance your analytics environment. We understand that in some cases your needs may require additional tools, including some proprietary third party tools. You have full administrative access on the virtual machine to install new tools you need. You can also install additional packages in Python and R that are not pre-installed. For Python you can use either `conda` or `pip`. For R you can use the `install.packages()` in the R console or use the IDE and choose "**Packages -> Install Packages...**".

Summary

These are just some of the things you can do on the Microsoft Data Science Virtual Machine. There are many more things you can do to make it an effective analytics environment.

Provision the Microsoft Data Science Virtual Machine

1/17/2017 • 12 min to read • [Edit on GitHub](#)

The Microsoft Data Science Virtual Machine is an Azure virtual machine (VM) image pre-installed and configured with several popular tools that are commonly used for data analytics and machine learning. The tools included are:

- Microsoft R Server Developer Edition
- Anaconda Python distribution
- Jupyter notebook (with R, Python kernels)
- Visual Studio Community Edition
- Power BI desktop
- SQL Server 2016 Developer Edition
- Machine learning and Data Analytics tools
 - [Computational Network Toolkit \(CNTK\)](#): A deep learning software toolkit from Microsoft Research.
 - [Vowpal Wabbit](#): A fast machine learning system supporting techniques such as online, hashing, allreduce, reductions, learning2search, active, and interactive learning.
 - [XGBoost](#): A tool providing fast and accurate boosted tree implementation.
 - [Rattle](#) (the R Analytical Tool To Learn Easily): A tool that makes getting started with data analytics and machine learning in R easy, with GUI-based data exploration, and modeling with automatic R code generation.
 - [mxnet](#): a deep learning framework designed for both efficiency and flexibility
 - [Weka](#) : A visual data mining and machine learning software in Java.
 - [Apache Drill](#): A schema-free SQL Query Engine for Hadoop, NoSQL and Cloud Storage. Supports ODBC and JDBC interfaces to enable querying NoSQL and files from standard BI tools like PowerBI, Excel, Tableau.
- Libraries in R and Python for use in Azure Machine Learning and other Azure services
- Git including Git Bash to work with source code repositories including GitHub, Visual Studio Team Services
- Windows ports of several popular Linux command-line utilities (including awk, sed, perl, grep, find, wget, curl etc) accessible through command prompt.

Doing data science involves iterating on a sequence of tasks: finding, loading, and pre-processing data, building and testing models, and deploying the models for consumption in intelligent applications. Data scientists use a variety of tools to complete these tasks. It can be quite time consuming to find the appropriate versions of the software, and then download and install them. The Microsoft Data Science Virtual Machine can ease this burden by providing a ready-to-use image that can be provisioned on Azure with all several popular tools pre-installed and configured.

The Microsoft Data Science Virtual Machine jump-starts your analytics project. It enables you to work on tasks in various languages including R, Python, SQL, and C#. Visual Studio provides an IDE to develop and test your code that is easy to use. The Azure SDK included in the VM allows you to build your applications using various services on Microsoft's cloud platform.

There are no software charges for this data science VM image. You only pay for the Azure usage fees which dependent on the size of the virtual machine you provision. More details on the compute fees can be found in the Pricing details section on the [Data Science Virtual Machine](#) page.

Prerequisites

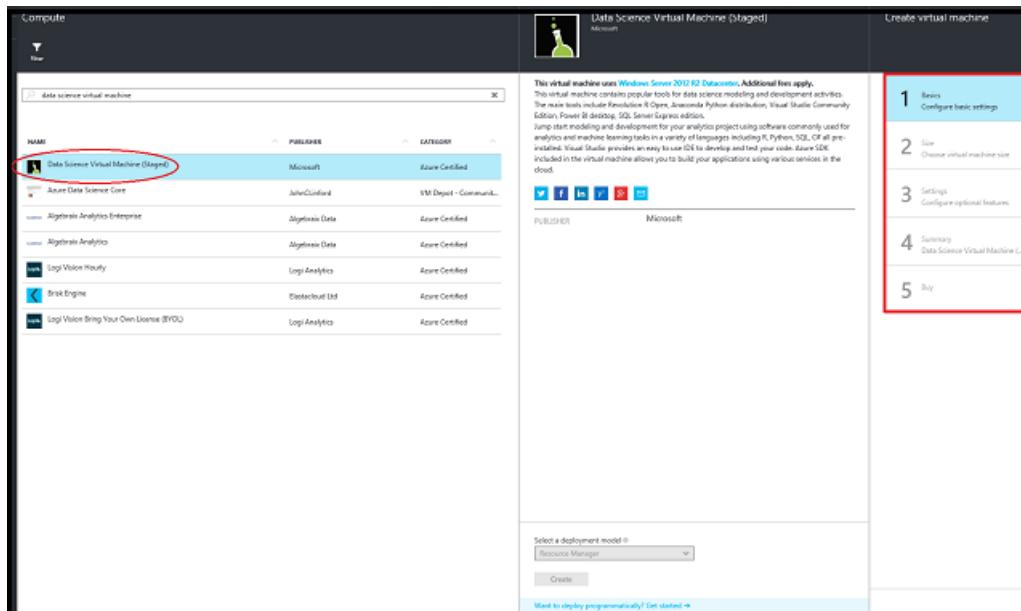
Before you can create a Microsoft Data Science Virtual Machine, you must have the following:

- **An Azure subscription:** To obtain one, see [Get Azure free trial](#).
- **An Azure storage account:** To create one, see [Create an Azure storage account](#). Alternatively, the storage account can be created as part of the process of creating the VM if you do not want to use an existing account.

Create your Microsoft Data Science Virtual Machine

Here are the steps to create an instance of the Microsoft Data Science Virtual Machine:

1. Navigate to the virtual machine listing on [Azure portal](#).
2. Select the **Create** button at the bottom to be taken into a wizard.



3. The wizard used to create the Microsoft Data Science Virtual Machine requires **inputs** for each of the **five steps** enumerated on the right of this figure. Here are the inputs needed to configure each of these steps:

a. Basics

- Name:** Name of your data science server you are creating.
- User Name:** Admin account login id.
- Password:** Admin account password.
- Subscription:** If you have more than one subscription, select the one on which the machine is to be created and billed.
- Resource Group:** You can create a new one or use an existing group.
- Location:** Select the data center that is most appropriate. Usually it is the data center that has most of your data or is closest to your physical location for fastest network access.

b. **Size:** Select one of the server types that meets your functional requirement and cost constraints. You can get more choices of VM sizes by selecting "View All".

c. Settings:

- Disk Type:** Choose Premium if you prefer a solid-state drive (SSD), else choose "Standard".
- Storage Account:** You can create a new Azure storage account in your subscription or use an existing one in the same *Location* that was chosen on the **Basics** step of the wizard.
- Other parameters:** Usually you just use the default values. You can hover over the informational link for help on the specific fields in case you want to consider the use of non-default values.
- Summary:** Verify that all information you entered is correct.
- Buy:** Click **Buy** to start the provisioning. A link is provided to the terms of the transaction. The VM does not have any additional charges beyond the compute for the server size you chose in the **Size** step.

NOTE

The provisioning should take about 10-20 minutes. The status of the provisioning is displayed on the Azure portal.

How to access the Microsoft Data Science Virtual Machine

Once the VM is created, you can remote desktop into it using the Admin account credentials that you configured in the preceding **Basics** section.

Once your VM is created and provisioned, you are ready to start using the tools that are installed and configured on it. There are start menu tiles and desktop icons for many of the tools.

How to create a strong password for Jupyter and start the notebook server

By default, the Jupyter notebook server is pre-configured but disabled on the VM until you set a Jupyter password. To create a strong password for the Jupyter notebook server installed on the machine, run the following command from a command-prompt on the Data Science Virtual Machine OR double click the desktop shortcut we have provided called **Jupyter Set Password & Start** from a local VM administrator account.

```
C:\dsvm\tools\setup\JupyterSetPasswordAndStart.cmd
```

Follow the messages and choose a strong password when prompted.

The above script will create a password hash and store it in the Jupyter configuration file located at:

C:\ProgramData\jupyter\jupyter_notebook_config.py under the parameter name **c.NotebookApp.password**.

The script will also enable and run the Jupyter server in the background. Jupyter server is created as a windows task in the Windows Task Scheduler called **Start_IPython_Notebook**. You may have to wait for a few seconds after setting the password before opening the notebook on your browser. See the section below titled **Jupyter Notebook** on how to access the Jupyter notebook server.

Tools installed on the Microsoft Data Science Virtual Machine

Microsoft R Server Developer Edition

If you wish to use R for your analytics, the VM has Microsoft R Server Developer edition installed. Microsoft R Server is a broadly deployable enterprise-class analytics platform based on R that is supported, scalable, and secure. Supporting a variety of big data statistics, predictive modeling and machine learning capabilities, R Server supports the full range of analytics – exploration, analysis, visualization, and modeling. By using and extending open source R, Microsoft R Server is fully compatible with R scripts, functions and CRAN packages, to analyze data at enterprise scale. It also addresses the in-memory limitations of Open Source R by adding parallel and chunked processing of data. This enables you to run analytics on data much bigger than what fits in main memory. Visual Studio Community Edition included on the VM contains the R Tools for Visual Studio extension that provides a full IDE for working with R. You can also download and use other IDEs as well such as [RStudio](#).

Python

For development using Python, Anaconda Python distribution 2.7 and 3.5 has been installed. This distribution contains the base Python along with about 300 of the most popular math, engineering, and data analytics packages. You can use Python Tools for Visual Studio (PTVS) that is installed within the Visual Studio 2015 Community edition or one of the IDEs bundled with Anaconda like IDLE or Spyder. You can launch one of these by searching on the search bar (**Win + S** key).

NOTE

To point the Python Tools for Visual Studio at Anaconda Python 2.7 and 3.5, you need to create custom environments for each version. To set these environment paths in the Visual Studio 2015 Community Edition, navigate to **Tools -> Python Tools -> Python Environments** and then click **+ Custom**.

Anaconda Python 2.7 is installed under C:\Anaconda and Anaconda Python 3.5 is installed under c:\Anaconda\envs\py35. See [PTVS documentation](#) for detailed steps.

Jupyter Notebook

Anaconda distribution also comes with a Jupyter notebook, an environment to share code and analysis. A Jupyter notebook server has been pre-configured with Python 2.7, Python 3.4, Python 3.5, and R kernels. There is a desktop icon named "Jupyter Notebook" to launch the browser to access the Notebook server. If you are on the VM via remote desktop, you can also visit <https://localhost:9999/> to access the Jupyter notebook server when logged in to the VM.

NOTE

Continue if you get any certificate warnings.

We have packaged several sample notebooks in Python and in R. The Jupyter notebooks show how to work with Microsoft R Server, SQL Server 2016 R Services (In-database analytics), Python, Microsoft Cognitive ToolKit (CNTK) for deep learning and other Azure technologies once you log in to Jupyter. You can see the link to the samples on the notebook home page after you authenticate to the Jupyter notebook using the password you created in an earlier step.

Visual Studio 2015 Community edition

Visual Studio Community edition installed on the VM. It is a free version of the popular IDE from Microsoft that you can use for evaluation purposes and for small teams. You can check out the licensing terms [here](#). Open Visual Studio by double-clicking the desktop icon or the **Start** menu. You can also search for programs with **Win + S** and entering "Visual Studio". Once there you can create projects in languages like C#, Python, R, node.js. Plugins are also installed that make it convenient to work with Azure services like Azure Data Catalog, Azure HDInsight (Hadoop, Spark), and Azure Data Lake.

NOTE

You may get a message stating that your evaluation period has expired. Enter your Microsoft account credentials or create a new free account to get access to the Visual Studio Community Edition.

SQL Server 2016 Developer edition

A developer version of SQL Server 2016 with R Services to run in-database analytics is provided on the VM. R Services provide a platform for developing and deploying intelligent applications. You can use the rich and powerful R language and the many packages from the community to create models and generate predictions for your SQL Server data. You can keep analytics close to the data because R Services (In-database) integrate the R language with SQL Server. This eliminates the costs and security risks associated with data movement.

NOTE

The SQL Server 2016 developer edition can only be used for development and test purposes. You need a license to run it in production.

You can access the SQL server by launching **SQL Server Management Studio**. Your VM name is populated as the

Server Name. Use Windows Authentication when logged in as the admin on Windows. Once you are in SQL Server Management Studio you can create other users, create databases, import data, and run SQL queries.

To enable In-database analytics using Microsoft R, run the following command as a one time action in SQL Server management studio after logging in as the server administrator.

```
CREATE LOGIN [%COMPUTERNAME%\SQLRUuserGroup] FROM WINDOWS
```

(Please replace the %COMPUTERNAME% with your VM name)

Azure

Several Azure tools are installed on the VM:

- There is a desktop shortcut to access the Azure SDK documentation.
- **AzCopy**: used to move data in and out of your Microsoft Azure Storage Account. To see usage, type **Azcopy** at a command prompt to see the usage.
- **Microsoft Azure Storage Explorer**: used to browse through the objects that you have stored within your Azure Storage Account and transfer data to and from Azure storage. You can type **Storage Explorer** in search or find it on the Windows Start menu to access this tool.
- **Adlcopy**: used to move data to Azure Data Lake. To see usage, type **adlcopy** in a command prompt.
- **dtui**: used to move data to and from Azure DocumentDB, a NoSQL database on the cloud. Type **dtui** on command prompt.
- **Microsoft Data Management Gateway**: enables data movement between on-premises data sources and cloud. It is used within tools like Azure Data Factory.
- **Microsoft Azure Powershell**: a tool used to administer your Azure resources in the Powershell scripting language is also installed on your VM.

Power BI

To help you build dashboards and great visualizations, the **Power BI Desktop** has been installed. Use this tool to pull data from different sources, to author your dashboards and reports, and to publish them to the cloud. For information, see the [Power BI](#) site. You can find Power BI desktop on the Start menu.

NOTE

You need an Office 365 account to access Power BI.

Additional Microsoft development tools

The [Microsoft Web Platform Installer](#) can be used to discover and download other Microsoft development tools. There is also a shortcut to the tool provided on the Microsoft Data Science Virtual Machine desktop.

Important directories on the VM

ITEM	DIRECTORY
Jupyter notebook server configurations	C:\ProgramData\jupyter
Jupyter Notebook samples home directory	c\dsvm\notebooks
Other samples	c\dsvm\samples
Anaconda (default: Python 2.7)	c\Anaconda

ITEM	DIRECTORY
Anaconda Python 3.5 environment	c:\Anaconda\envs\py35
R Server Standalone instance directory (Default R instance based on R3.2.2)	C:\Program Files\Microsoft SQL Server\130\R_SERVER
R Server In-database instance directory (R3.2.2)	C:\Program Files\Microsoft SQL Server\MSSQL13.MSSQLSERVER\R_SERVICES
Microsoft R Open (R3.3.1) instance directory	C:\Program Files\Microsoft\MRO-3.3.1
Miscellaneous tools	c\dsvm\tools

NOTE

Instances of the Microsoft Data Science Virtual Machine created before 1.5.0 (before September 3, 2016) used a slightly different directory structure than specified in the preceding table.

Next Steps

Here are some next steps to continue your learning and exploration.

- Explore the various data science tools on the data science VM by clicking the start menu and checking out the tools listed on the menu.
- Navigate to **C:\Program Files\Microsoft SQL Server\130\R_SERVER\library\RevoScaleR\demoScripts** for samples using the RevoScaleR library in R that supports data analytics at enterprise scale.
- Read the article: [10 things you can do on the Data science Virtual Machine](#)
- Learn how to build end to end analytical solutions systematically using the [Team Data Science Process](#).
- Visit the [Cortana Intelligence Gallery](#) for machine learning and data analytics samples that use the Cortana Intelligence Suite. We have also provided an icon on the **Start** menu and on the desktop of the virtual machine to this gallery.

Set up an Azure virtual machine as an IPython Notebook server for advanced analytics

1/17/2017 • 6 min to read • [Edit on GitHub](#)

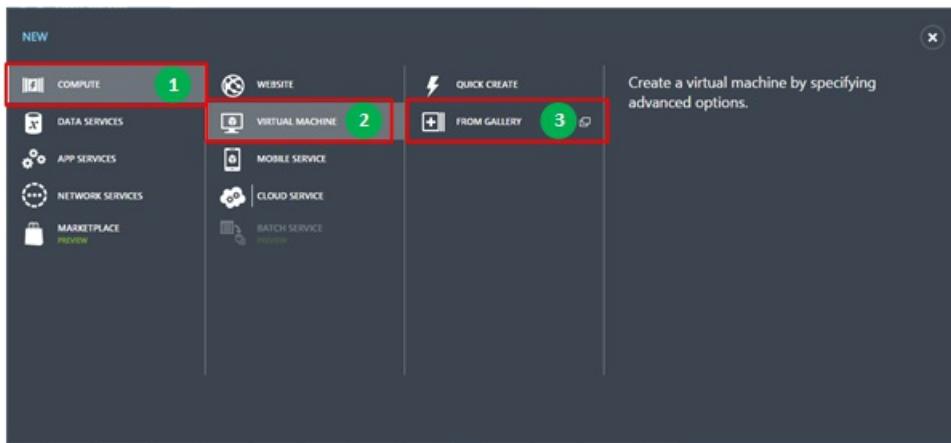
This topic shows how to provision and configure an Azure virtual machine for advanced analytics that can be used as part of a data science environment. The Windows virtual machine is configured with supporting tools such as IPython Notebook, Azure Storage Explorer, AzCopy, as well as other utilities that are useful for advanced analytics projects. Azure Storage Explorer and AzCopy, for example, provide convenient ways to upload data to Azure blob storage from your local machine or to download it to your local machine from blob storage.

Step 1: Create a general-purpose Azure virtual machine

If you already have an Azure virtual machine and just want to set up an IPython Notebook server on it, you can skip this step and proceed to [Step 2: Add an endpoint for IPython Notebooks to an existing virtual machine](#).

Before starting the process of creating a virtual machine on Azure, you need to determine the size of the machine that is needed to process the data for their project. Smaller machines have less memory and fewer CPU cores than larger machines, but they are also less expensive. For a list of machine types and prices, see the [Virtual Machines Pricing page](#)

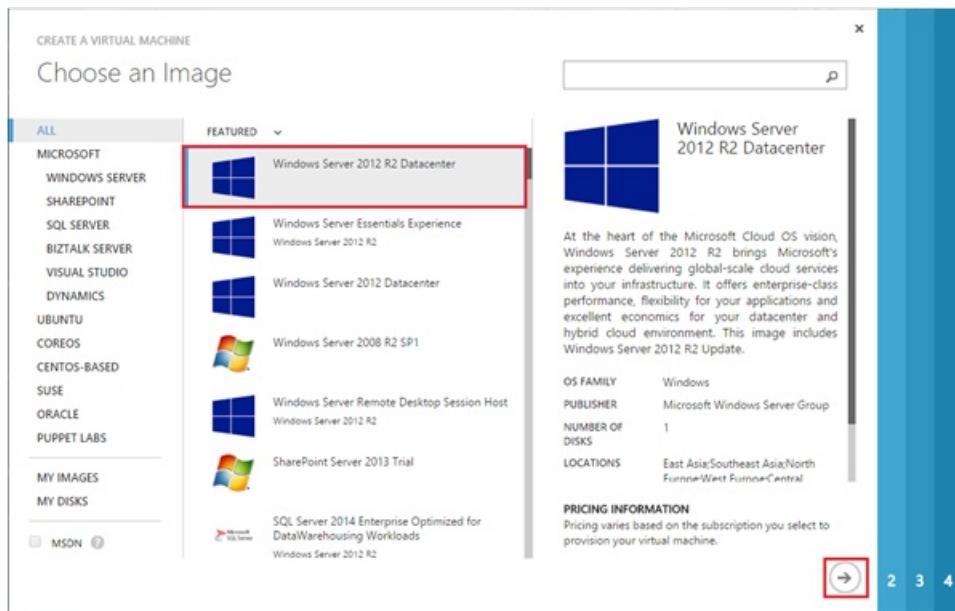
1. Log in to [Azure classic portal](#), and click **New** in the bottom left corner. A window will pop up. Select **COMPUTE -> VIRTUAL MACHINE -> FROM GALLERY**.



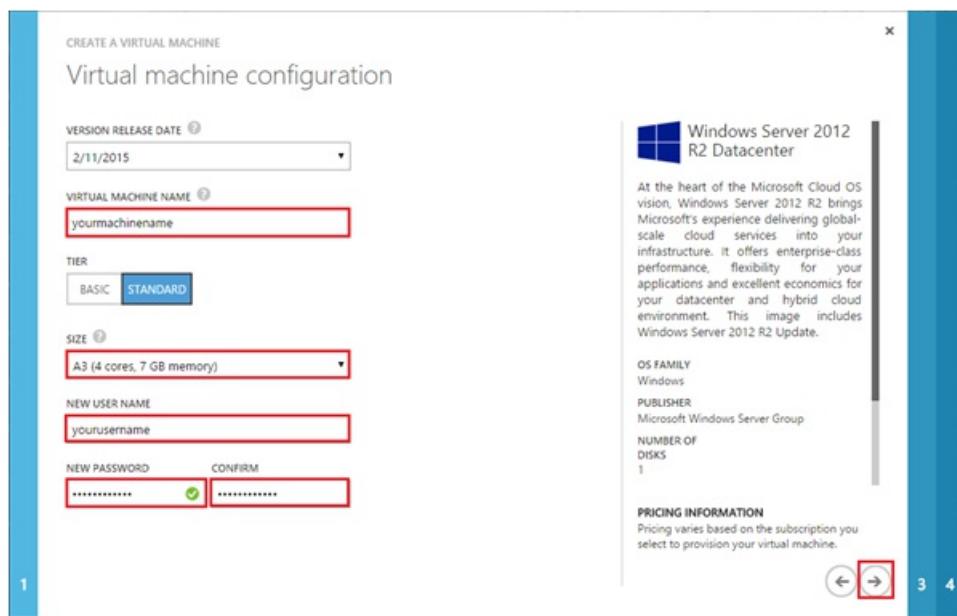
2. Choose one of the following images:

- Windows Server 2012 R2 Datacenter
- Windows Server Essentials Experience (Windows Server 2012 R2)

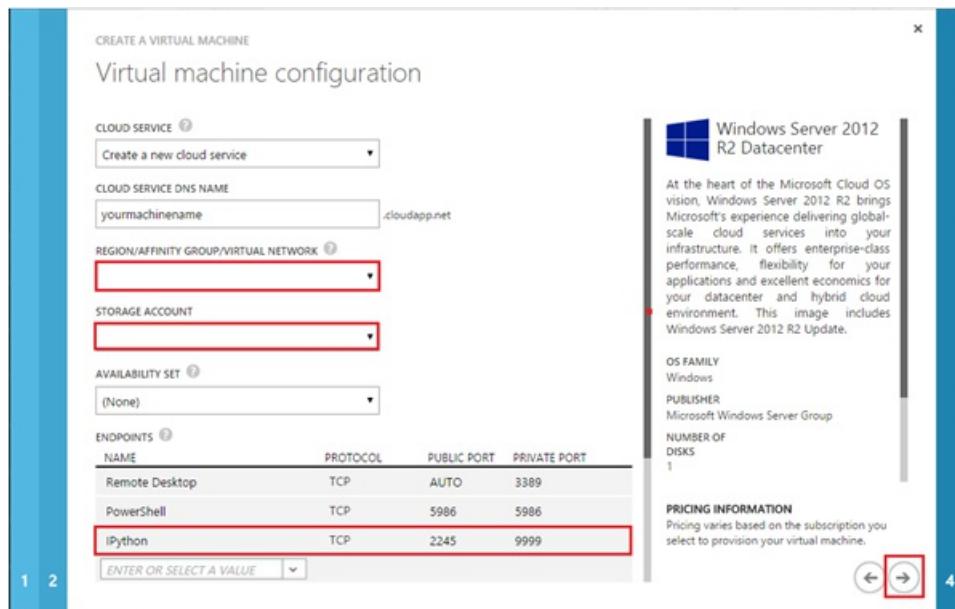
Then, click the arrow pointing right at the lower right to go the next configuration page.



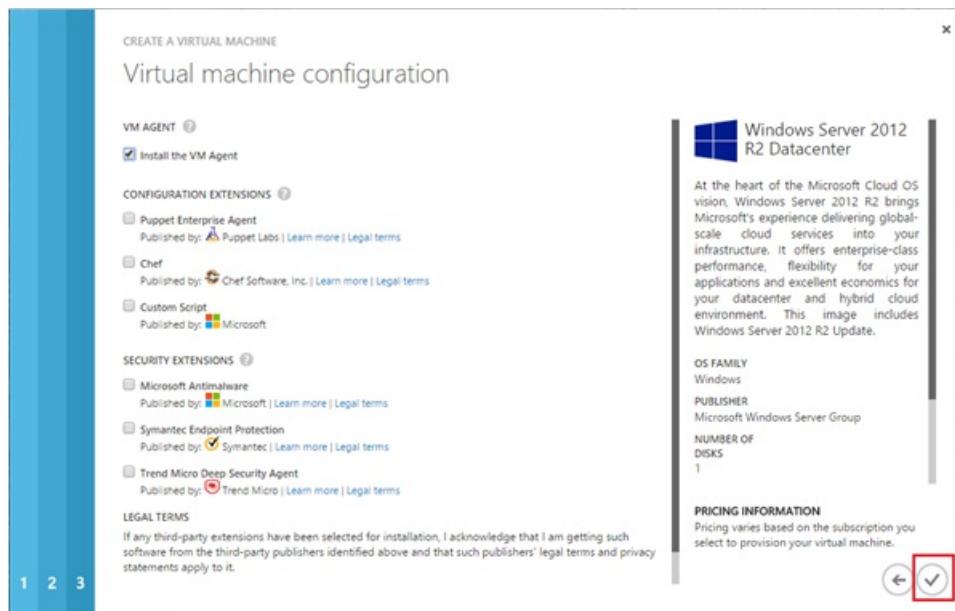
3. Enter a name for the virtual machine you want to create, select the size of the machine (Default: A3) based on the size of the data the machine is going to process and how powerful you want the machine to be (memory size and the number of compute cores), enter a user name and password for the machine. Then, click the arrow pointing right to go to the next configuration page.



4. Select the **REGION/AFFINITY GROUP/VIRTUAL NETWORK** that contains the **STORAGE ACCOUNT** that you are planning to use for this virtual machine, and then select that storage account. Add an endpoint at the bottom in the **ENDPOINTS** field by entering the name of the endpoint ("IPython" here). You can choose any string as the **NAME** of the end point, and any integer between 0 and 65536 that is **available** as the **PUBLIC PORT**. The **PRIVATE PORT** has to be **9999**. You should **avoid** using public ports that have already been assigned for internet services. [Ports for Internet Services](#) provides a list of ports that have been assigned and should be avoided.



- Click the check mark to start the virtual machine provisioning process.



It may take 15-25 minutes to complete the virtual machine provisioning process. After the virtual machine has been created, the status of this machine should show as **Running**.

Step 2: Add an endpoint for IPython Notebooks to an existing virtual machine

If you created the virtual machine by following the instructions in Step 1, then the endpoint for IPython Notebook has already been added and this step can be skipped.

If the virtual machine already exists, and you need to add an endpoint for IPython Notebook that you will install in Step 3 below, first login to Azure classic portal, select the virtual machine, and add the endpoint for IPython Notebook server. The following figure contains a screen shot of the portal after the endpoint for IPython Notebook has been added to a Windows virtual machine.

NAME	PROTOCOL	PUBLIC PORT	PRIVATE PORT	LOAD-BALANCED SET NAME
ipython	TCP	9999	-	
PowerShell	TCP	5986	5986	-
Remote Desktop	TCP	65226	3389	-

Step 3: Install IPython Notebook and other supporting tools

After the virtual machine is created, use Remote Desktop Protocol (RDP) to log on to the Windows virtual machine. For instructions, see [How to Log on to a Virtual Machine Running Windows Server](#). Open the **Command Prompt (Not the Powershell command window)** as an **Administrator** and run the following command.

```
set script='https://raw.githubusercontent.com/Azure/Azure-MachineLearning-DataScience/master/Misc/MachineSetup/Azure_VM_Setup_Windows.ps1'

@powershell -NoProfile -ExecutionPolicy unrestricted -Command "iex ((new-object net.webclient).DownloadString(%script%))"
```

When the installation completes, the IPython Notebook server is launched automatically in the `C:\Users\<user name>\Documents\IPython Notebooks` directory.

When prompted, enter a password for the IPython Notebook and the password of the machine administrator. This enables the IPython Notebook to run as a service on the machine.

Step 4: Access IPython Notebooks from a web browser

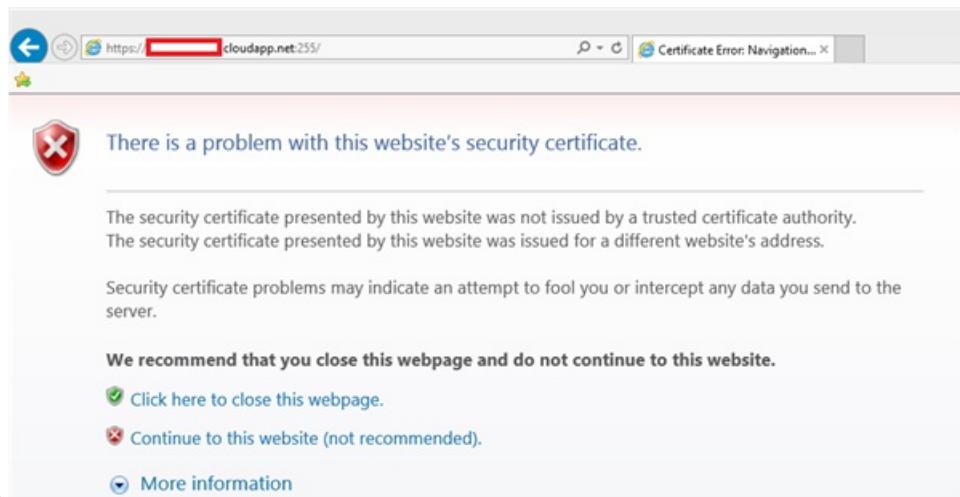
To access the IPython Notebook server, open a web browser, and input `https://<virtual machine DNS name>:<public port number>` in the URL text box. Here, the `<public port number>` should be the port number you specified when the IPython Notebook endpoint was added.

The `<virtual machine DNS name>` can be found at the Azure classic portal. After logging in to the classic portal, click **VIRTUAL MACHINES**, select the machine you created, and then select **DASHBOARD**, the DNS name will be shown as follows:

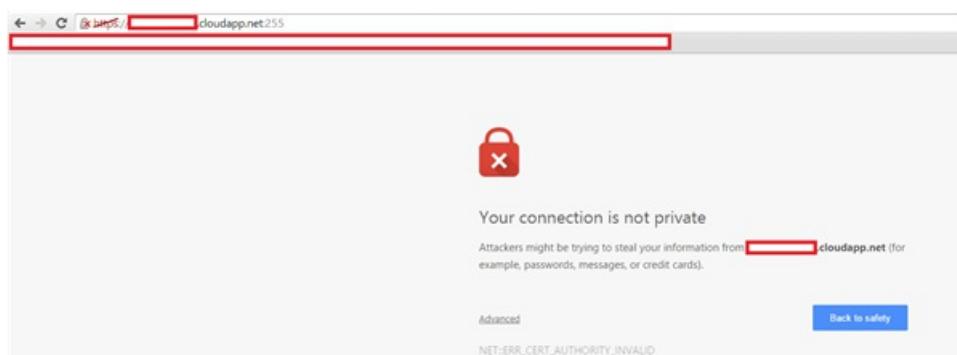


You will encounter a warning stating that *There is a problem with this website's security certificate* (Internet

Explorer) or *Your connection is not private* (Chrome), as shown in the following figures. Click **Continue to this website (not recommended)** (Internet Explorer) or **Advanced** and then **Proceed to <DNS Name> (unsafe)** (Chrome) to continue. Then input the password you specified earlier to access the IPython Notebooks.



Internet Explorer:



Chrome:

After you log on to the IPython Notebook, a directory *DataScienceSamples* will show on the browser. This directory contains sample IPython Notebooks that are shared by Microsoft to help users conduct data science tasks. These sample IPython Notebooks are checked out from **Github repository** to the virtual machines during the IPython Notebook server setup process. Microsoft maintains and updates this repository frequently. Users may visit the Github repository to get the most recently updated sample IPython Notebooks.

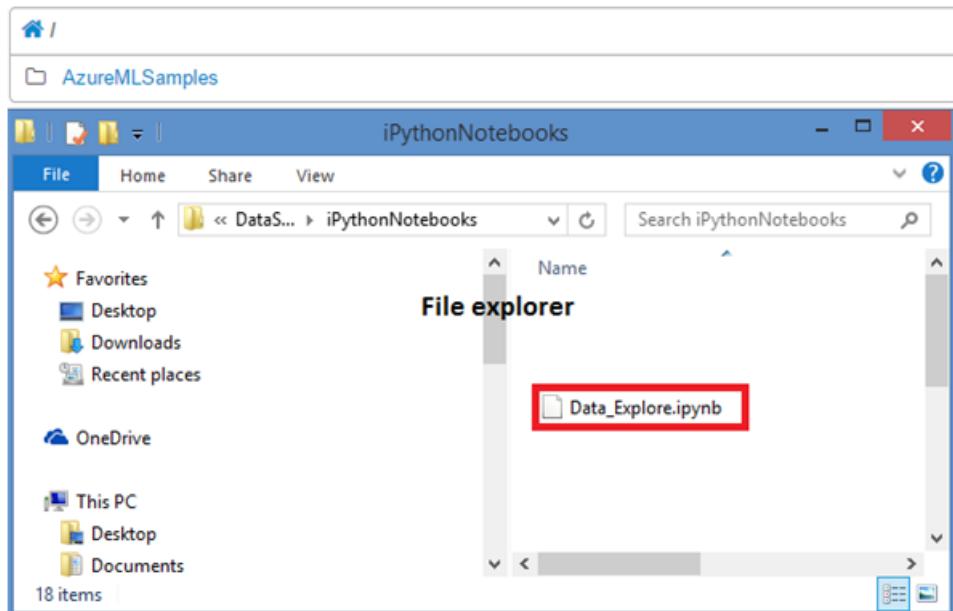


Step 5: Upload an existing IPython Notebook from a local machine to the IPython Notebook server

IPython Notebooks provide an easy way for users to upload an existing IPython Notebook on their local machines to the IPython Notebook server on the virtual machines. After you log on to the IPython Notebook in a web browser, click into the **directory** that the IPython Notebook will be uploaded to. Then, select an IPython Notebook .ipynb file to upload from the local machine in the **File Explorer**, and drag and drop it to the IPython Notebook directory on the web browser. Click the **Upload** button, to upload the .ipynb file to the IPython Notebook server. Other users can then start using it from their web browsers.

Notebooks Running Clusters

To import a notebook, drag the file onto the listing below or click here.



Logout

Notebooks Running Clusters

To import a notebook, drag the file onto the listing below or click here.

New Notebook



Shut down and de-allocate virtual machine when not in use

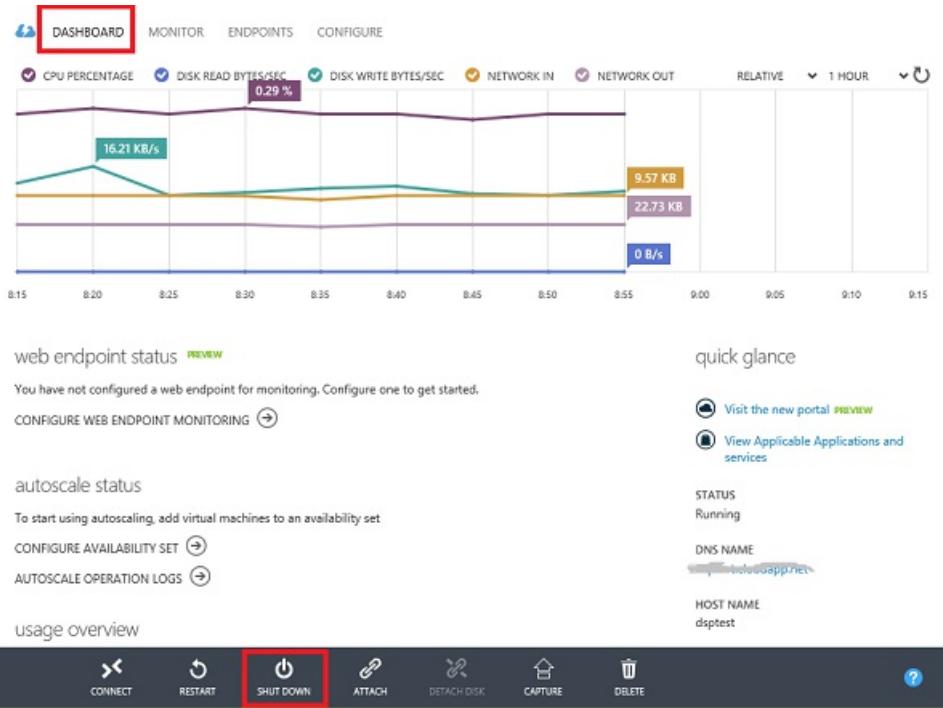
Azure Virtual Machines are priced as **pay only for what you use**. To ensure that you are not being billed when not using your virtual machine, it has to be in the **Stopped (Deallocated)** state when not in use.

NOTE

If you shut down the virtual machine from inside the VM (using Windows power options), the VM is stopped but remains allocated. To ensure you do not continue to be billed, always stop virtual machines from the [Azure classic portal](#). You can also stop the VM through Powershell by calling **ShutdownRoleOperation** with "PostShutdownAction" equal to "StoppedDeallocated".

To shut down and deallocate the virtual machine:

1. Log in to the [Azure classic portal](#) using your account.
2. Select **VIRTUAL MACHINES** from the left navigation bar.
3. In the list of virtual machines, click on the name of your virtual machine then go to the **DASHBOARD** page.
4. At the bottom of the page, click **SHUTDOWN**.



The virtual machine will be deallocated but not deleted. You may restart your virtual machine at any time from the Azure classic portal.

Your Azure VM is ready to use: what's next?

Your virtual machine is now ready to use in your data science exercises. The virtual machine is also ready for use as an IPython Notebook server for the exploration and processing of data, and other tasks in conjunction with Azure Machine Learning and the Team Data Science Process.

The next steps in the Team Data Science Process are mapped in the [Learning Path](#) and may include steps that move data into HDInsight, process and sample it there in preparation for learning from the data with Azure Machine Learning.

Set up an Azure SQL Server virtual machine as an IPython Notebook server for advanced analytics

1/17/2017 • 14 min to read • [Edit on GitHub](#)

This topic shows how to provision and configure an SQL Server virtual machine to be used as part of a cloud-based data science environment. The Windows virtual machine is configured with supporting tools such as IPython Notebook, Azure Storage Explorer, and AzCopy, as well as other utilities that are useful for data science projects. Azure Storage Explorer and AzCopy, for example, provide convenient ways to upload data to Azure blob storage from your local machine or to download it to your local machine from blob storage.

The Azure virtual machine gallery includes several images that contain Microsoft SQL Server. Select an SQL Server VM image that is suitable for your data needs. Recommended images are:

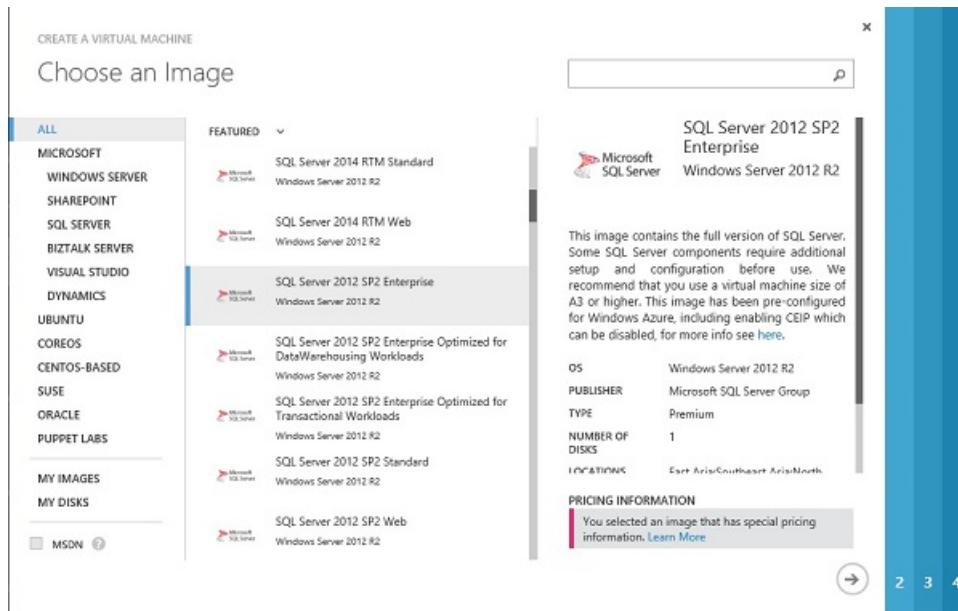
- SQL Server 2012 SP2 Enterprise for small to medium data sizes
- SQL Server 2012 SP2 Enterprise Optimized for DataWarehousing Workloads for large to very large data sizes

NOTE

SQL Server 2012 SP2 Enterprise image **does not include a data disk**. You will need to add and/or attach one or more virtual hard disks to store your data. When you create an Azure virtual machine, it has a disk for the operating system mapped to the C drive and a temporary disk mapped to the D drive. Do not use the D drive to store data. As the name implies, it provides temporary storage only. It offers no redundancy or backup because it doesn't reside in Azure storage.

Connect to the Azure Classic Portal and provision an SQL Server virtual machine

1. Log in to the [Azure Classic portal](#) using your account. If you do not have an Azure account, visit [Azure free trial](#).
2. On the Azure Classic portal, at the bottom left of the web page, click **+NEW**, click **COMPUTE**, click **VIRTUAL MACHINE**, and then click **FROM GALLERY**.
3. On the **Create a Virtual Machine** page, select a virtual machine image containing SQL Server based on your data needs, and then click the next arrow at the bottom right of the page. For the most up-to-date information on the supported SQL Server images on Azure, see [Getting Started with SQL Server in Azure Virtual Machines](#) topic in the [SQL Server in Azure Virtual Machines](#) documentation set.



4. On the first **Virtual Machine Configuration** page, provide the following information:

- Provide a **VIRTUAL MACHINE NAME**.
- In the **NEW USER NAME** box, type unique user name for the VM local administrator account.
- In the **NEW PASSWORD** box, type a strong password. For more information, see [Strong Passwords](#).
- In the **CONFIRM PASSWORD** box, retype the password.
- Select the appropriate **SIZE** from the drop down list.

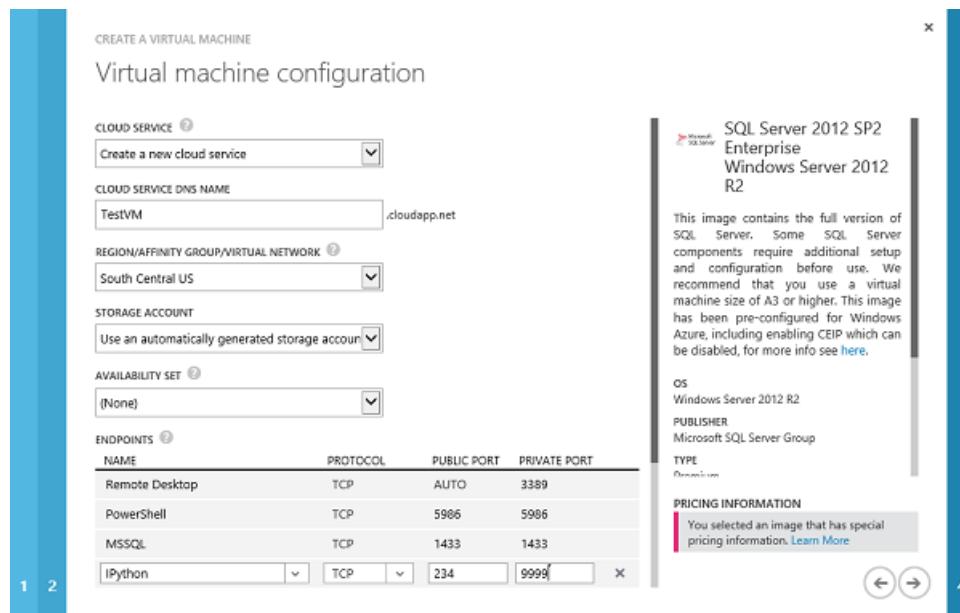
NOTE

The size of the virtual machine is specified during provisioning: A2 is the smallest size recommended for production workloads. The minimum recommended size for a virtual machine is A3 when using SQL Server Enterprise Edition. Select A3 or higher when using SQL Server Enterprise Edition. Select A4 when using SQL Server 2012 or 2014 Enterprise Optimized for Transactional Workloads images. Select A7 when using SQL Server 2012 or 2014 Enterprise Optimized for Data Warehousing Workloads images. The size selected limits the number of data disks you can configure. For most up-to-date information on available virtual machine sizes and the number of data disks that you can attach to a virtual machine, see [Virtual Machine Sizes for Azure](#). For pricing information, see [Virtual Machines Pricing](#).

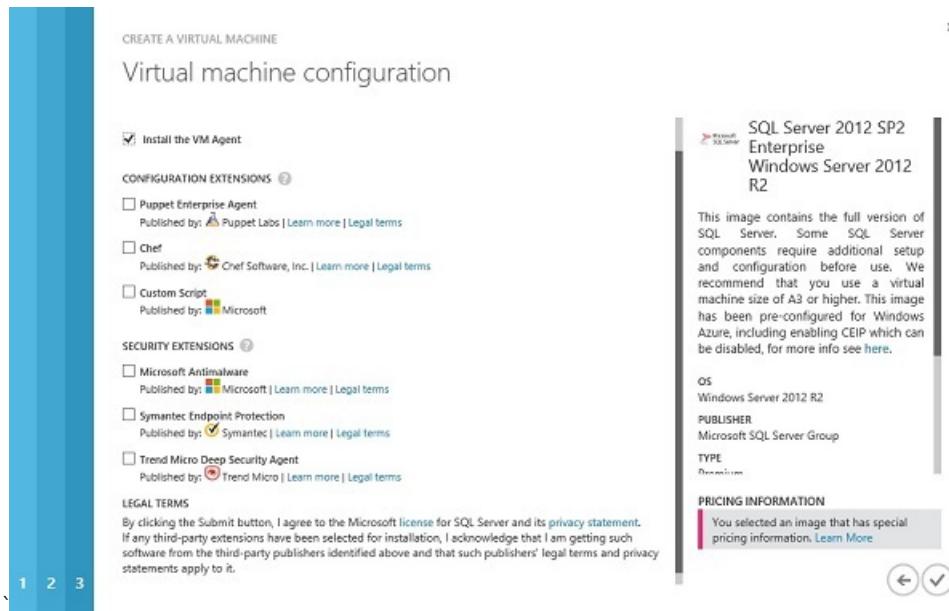
Click the next arrow on the bottom right to continue.

5. On the second **Virtual machine configuration** page, configure resources for networking, storage, and availability:
- In the **Cloud Service** box, choose **Create a new cloud service**.
 - In the **Cloud Service DNS Name** box, provide the first portion of a DNS name of your choice, so that it completes a name in the format **TESTNAME.cloudapp.net**
 - In the **REGION/AFFINITY GROUP/VIRTUAL NETWORK** box, select a region where this virtual image will be hosted.
 - In the **Storage Account**, select an existing storage account or select an automatically generated one.
 - In the **AVAILABILITY SET** box, select **(none)**.
 - Read and accept the pricing information.
6. In the **ENDPOINTS** section, click in the empty dropdown under **NAME**, and select **MSSQL** then type the port number of the instance of the Database Engine (**1433** for the default instance).
7. Your SQL Server VM can also serve as an IPython Notebook Server, which will be configured in a later step. Add a new endpoint to specify the port to use for your IPython Notebook server. Enter a name in the **NAME** column, select a port number of your choice for the public port, and **9999** for the private port.

Click the next arrow on the bottom right to continue.



8. Accept the default **Install VM agent** option checked and click the check mark in the bottom right corner of the wizard to complete the VM provisioning process.



9. Wait while Azure prepares your virtual machine. Expect the virtual machine status to proceed through:

- Starting (Provisioning)
- Stopped
- Starting (Provisioning)
- Running (Provisioning)
- Running

Open the virtual machine using Remote Desktop and complete setup

1. When provisioning completes, click on the name of your virtual machine to go to the DASHBOARD page. At the bottom of the page, click **Connect**.
2. Choose to open the rdp file using the Windows Remote Desktop program (`%windir%\system32\mstsc.exe`).
3. At the **Windows Security** dialog box, provide the password for the local administrator account that you specified in an earlier step. (You might be asked to verify the credentials of the virtual machine.)
4. The first time you log on to this virtual machine, several processes may need to complete, including setup of your desktop, Windows updates, and completion of the Windows initial configuration tasks (sysprep). After Windows sysprep completes, SQL Server setup completes configuration tasks. These tasks may cause a delay of a few minutes while they complete. `SELECT @@SERVERNAME` may not return the correct name until SQL Server setup completes, and SQL Server Management Studio may not be visible on the start page.

Once you are connected to the virtual machine with Windows Remote Desktop, the virtual machine works much like any other computer. Connect to the default instance of SQL Server with SQL Server Management Studio (running on the virtual machine) in the normal way.

Install IPython Notebook and other supporting tools

To configure your new SQL Server VM to serve as an IPython Notebook server, and install additional supporting tools such AzCopy, Azure Storage Explorer, useful Data Science Python packages, and others, a special customization script is provided to you. To install:

- Right-click the **Windows Start** icon and click **Command Prompt (Admin)**
- Copy the following commands and paste at the command prompt.

```
set script="https://raw.githubusercontent.com/Azure/Azure-MachineLearning-  
DataScience/master/Misc/MachineSetup/Azure_VM_Setup_Windows.ps1"  
@powershell -NoProfile -ExecutionPolicy unrestricted -Command "iex ((new-object net.webclient).DownloadString(%$script%))"
```

- When prompted, enter a password of your choice for the IPython Notebook server.
- The customization script automates several post-install procedures, which include:
 - Installation and setup of IPython Notebook server
 - Opening TCP ports in the Windows firewall for the endpoints created earlier:
 - For SQL Server remote connectivity
 - For IPython Notebook server remote connectivity
 - Fetching sample IPython notebooks and SQL scripts
 - Downloading and installing useful Data Science Python packages
 - Downloading and installing Azure tools such as AzCopy and Azure Storage Explorer
- You may access and run IPython Notebook from any local or remote browser using a URL of the form
`https://<virtual_machine_DNS_name>:<port>`, where port is the IPython public port you selected while provisioning the virtual machine.
- IPython Notebook server is running as a background service and will be restarted automatically when you restart the virtual machine.

Attach data disk as needed

If your VM image does not include data disks, i.e., disks other than C drive (OS disk) and D drive (temporary disk), you need to add one or more data disks to store your data. The VM image for SQL Server 2012 SP2 Enterprise Optimized for DataWarehousing Workloads comes pre-configured with additional disks for SQL Server data and log files.

NOTE

Do not use the D drive to store data. As the name implies, it provides temporary storage only. It offers no redundancy or backup because it doesn't reside in Azure storage.

To attach additional data disks, follow the steps described in [How to Attach a Data Disk to a Windows Virtual Machine](#), which will guide you through:

1. Attaching empty disk(s) to the virtual machine provisioned in earlier steps
2. Initialization of the new disk(s) in the virtual machine

Connect to SQL Server Management Studio and enable mixed mode authentication

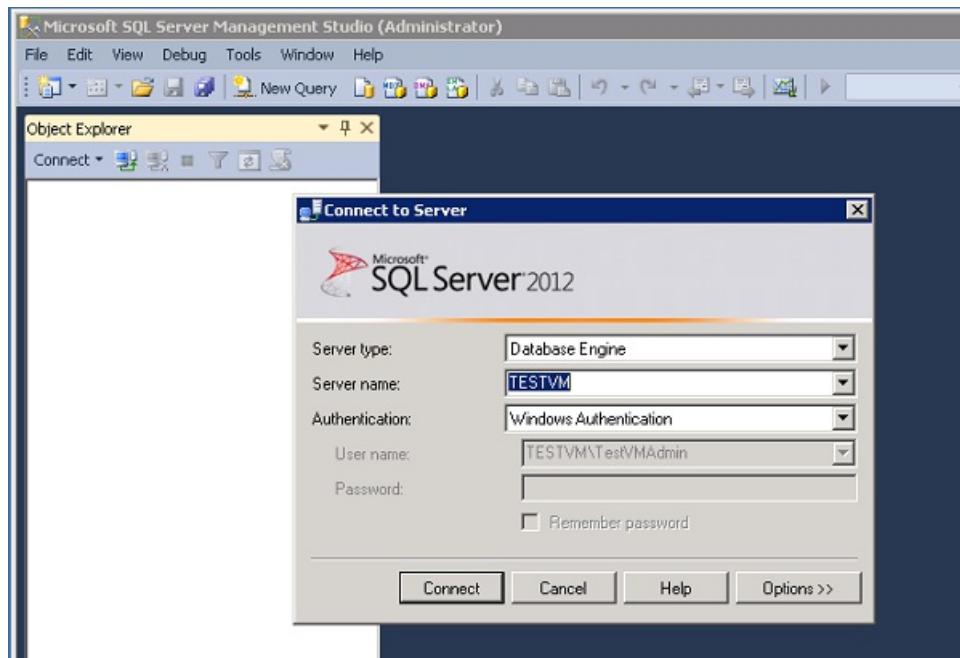
The SQL Server Database Engine cannot use Windows Authentication without domain environment. To connect to the Database Engine from another computer, configure SQL Server for mixed mode authentication. Mixed mode authentication allows both SQL Server Authentication and Windows Authentication. SQL authentication mode is required in order to ingest data directly from your SQL Server VM databases in the [Azure Machine Learning Studio](#) using the Import Data module.

1. While connected to the virtual machine by using Remote Desktop, use the Windows **Search** pane and type **SQL Server Management Studio** (SMSS). Click to start the SQL Server Management Studio (SSMS). You may want to add a shortcut to SSMS on your Desktop for future use.



The first time you open Management Studio it must create the users Management Studio environment. This may take a few moments.

- When opening, Management Studio presents the **Connect to Server** dialog box. In the **Server name** box, type the name of the virtual machine to connect to the Database Engine with the Object Explorer. (Instead of the virtual machine name you can also use **(local)** or a single period as the **Server name**. Select **Windows Authentication**, and leave **your_VM_name\your_local_administrator** in the **User name** box. Click **Connect**.

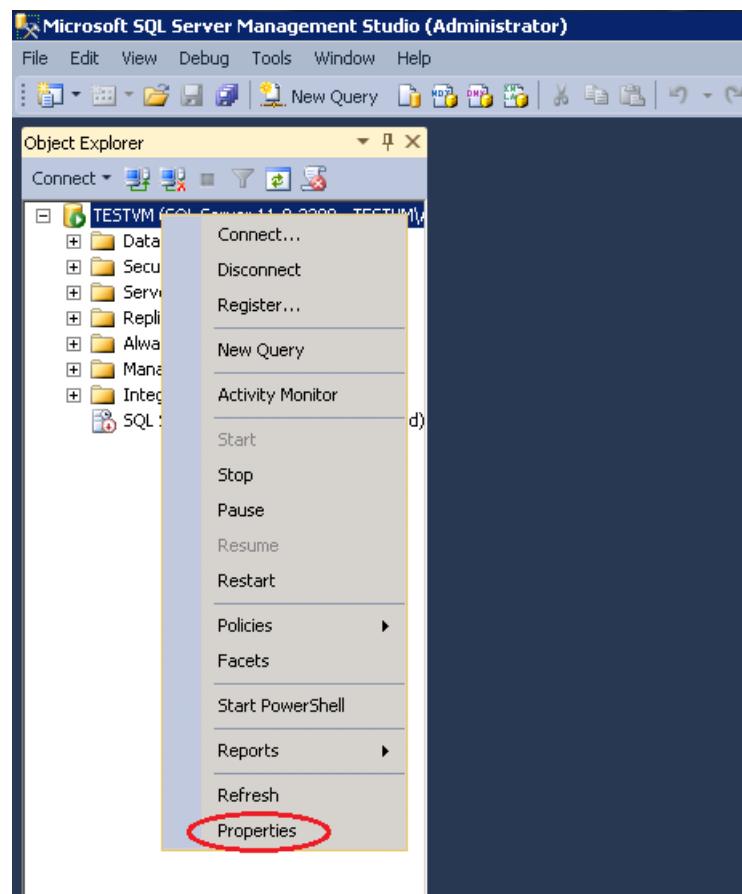


```
USE master
go

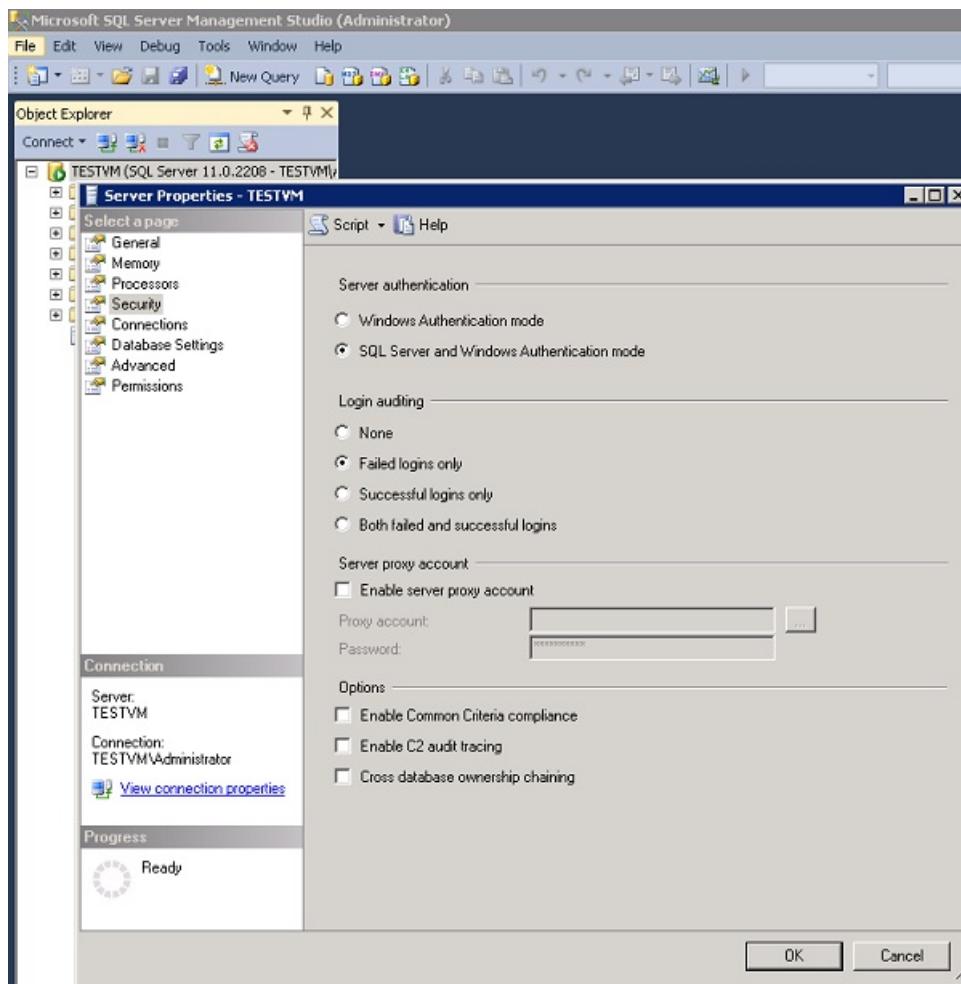
EXEC xp_instance_regwrite N'HKEY_LOCAL_MACHINE', N'Software\Microsoft\MSSQLServer\MSSQLServer', N'LoginMode',
REG_DWORD, 2
go
```

To change the authentication mode using SQL Server management Studio:

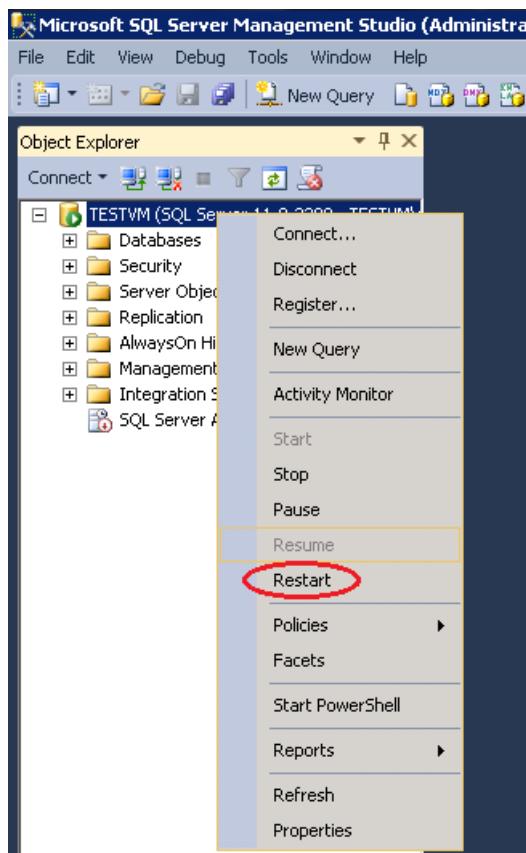
3. In **SQL Server Management Studio Object Explorer**, right-click the name of the instance of SQL Server (the virtual machine name), and then click **Properties**.



4. On the **Security** page, under **Server authentication**, select **SQL Server and Windows Authentication mode**, and then click **OK**.



5. In the **SQL Server Management Studio** dialog box, click **OK** to acknowledge the requirement to restart SQL Server.
6. In **Object Explorer**, right-click your server, and then click **Restart**. (If SQL Server Agent is running, it must also be restarted.)



7. In the **SQL Server Management Studio** dialog box, click **Yes** to agree that you want to restart SQL Server.

Create SQL Server authentication logins

To connect to the Database Engine from another computer, you must create at least one SQL Server authentication login.

You may create new SQL Server logins programmatically or using the SQL Server Management Studio. To create a new sysadmin user with SQL authentication programmatically, start a **New Query** and execute the following script. Replace and with your choice of *user name* and *password*.

```
USE master
go

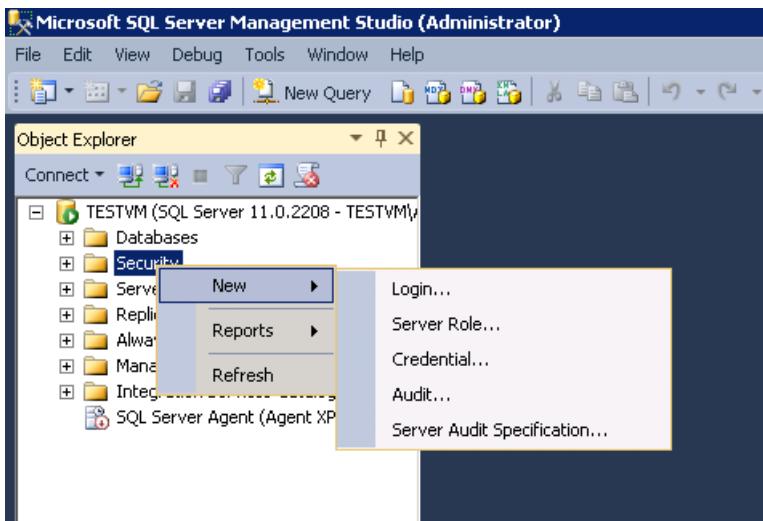
CREATE LOGIN <new user name> WITH PASSWORD = N'<new password>',
    CHECK_POLICY = OFF,
    CHECK_EXPIRATION = OFF;

EXEC sp_addsrvrolemember @loginname = N'<new user name>', @rolename = N'sysadmin';
```

Adjust the password policy as needed (the sample code turns off policy checking and password expiration). For more information about SQL Server logins, see [Create a Login](#).

To create new SQL Server logins using the SQL Server Management Studio:

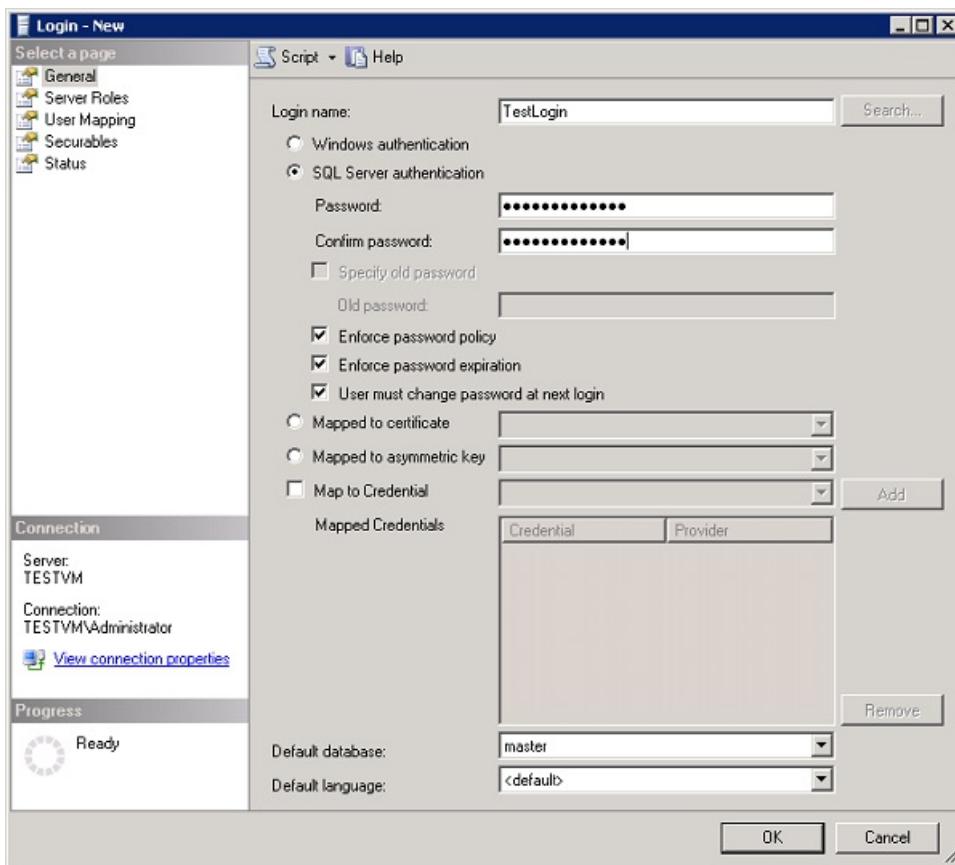
1. In **SQL Server Management Studio Object Explorer**, expand the folder of the server instance in which you want to create the new login.
2. Right-click the **Security** folder, point to **New**, and select **Login...**



3. In the **Login - New** dialog box, on the **General** page, enter the name of the new user in the **Login name** box.
4. Select **SQL Server authentication**.
5. In the **Password** box, enter a password for the new user. Enter that password again into the **Confirm Password** box.
6. To enforce password policy options for complexity and enforcement, select **Enforce password policy** (recommended). This is a default option when SQL Server authentication is selected.
7. To enforce password policy options for expiration, select **Enforce password expiration** (recommended). Enforce password policy must be selected to enable this checkbox. This is a default option when SQL Server authentication is selected.
8. To force the user to create a new password after the first time the login is used, select **User must change password at next login** (Recommended if this login is for someone else to use. If the login is for your own use, do not select this option.) Enforce password expiration must be selected to enable this checkbox. This is a default

option when SQL Server authentication is selected.

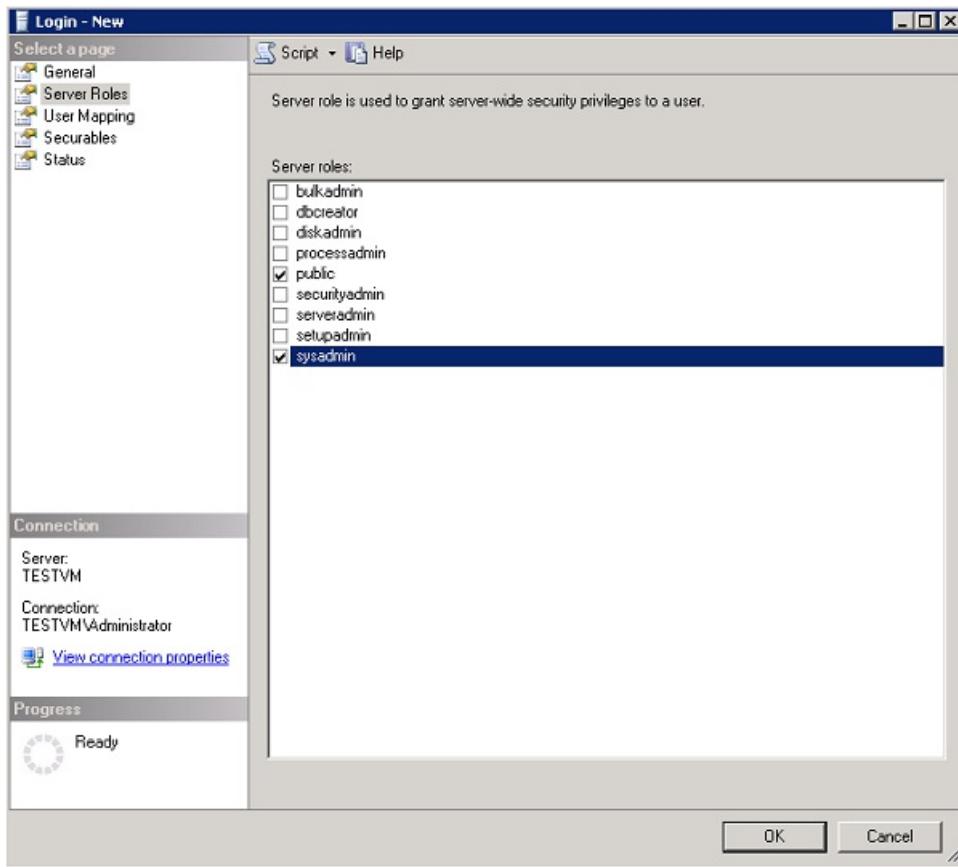
9. From the **Default database** list, select a default database for the login. **master** is the default for this option. If you have not yet created a user database, leave this set to **master**.
10. In the **Default language** list, leave **default** as the value.



11. If this is the first login you are creating, you may want to designate this login as a SQL Server administrator. If so, on the **Server Roles** page, check **sysadmin**.

IMPORTANT

Members of the sysadmin fixed server role have complete control of the Database Engine. For security reasons, you should carefully restrict membership in this role.



12. Click OK.

Determine the DNS name of the virtual machine

To connect to the SQL Server Database Engine from another computer, you must know the Domain Name System (DNS) name of the virtual machine.

(This is the name the internet uses to identify the virtual machine. You can use the IP address, but the IP address might change when Azure moves resources for redundancy or maintenance. The DNS name will be stable because it can be redirected to a new IP address.)

1. In the Azure Classic Portal (or from the previous step), select **VIRTUAL MACHINES**.
2. On the **VIRTUAL MACHINE INSTANCES** page, in the **DNS NAME** column, find and copy the DNS name for the virtual machine which appears preceded by **http://**. (The user interface might not display the entire name, but you can right-click on it, and select copy.)

Connect to the Database Engine from another computer

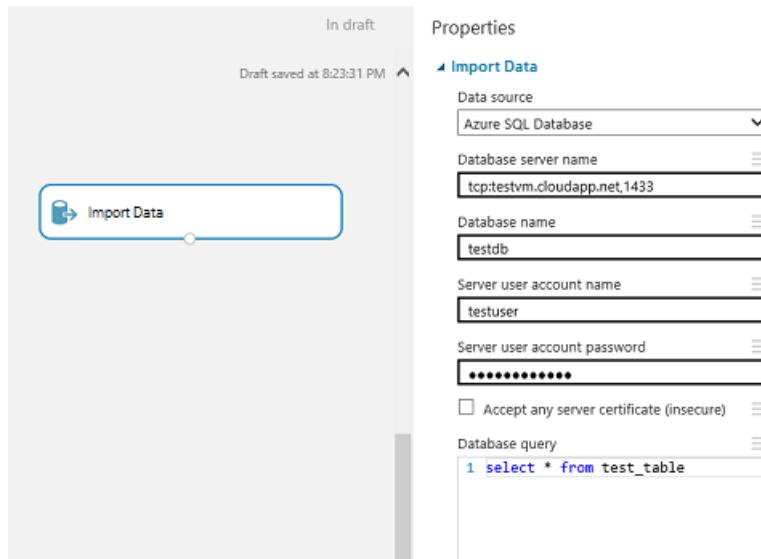
1. On a computer connected to the internet, open SQL Server Management Studio.
2. In the **Connect to Server** or **Connect to Database Engine** dialog box, in the **Server name** box, enter the DNS name of the virtual machine (determined in the previous task) and a public endpoint port number in the format *DNSName,portnumber* such as **tutorialtestVM.cloudapp.net,57500**.
3. In the **Authentication** box, select **SQL Server Authentication**.
4. In the **Login** box, type the name of a login that you created in an earlier task.
5. In the **Password** box, type the password of the login that you create in an earlier task.
6. Click **Connect**.

Connect to the Database Engine from Azure Machine Learning

In later stages of the Team Data Science Process, you will use the [Azure Machine Learning Studio](#) to build and

deploy machine learning models. To ingest data from your SQL Server VM databases directly into Azure Machine Learning for training or scoring, use the **Import Data** module in a new [Azure Machine Learning Studio](#) experiment. This topic is covered in more details through the Team Data Science Process guide links. For an introduction, see [What is Azure Machine Learning Studio?](#).

1. In the **Properties** pane of the [Import Data module](#), select **Azure SQL Database** from the **Data Source** dropdown list.
2. In the **Database server name** text box, enter `tcp:<DNS name of your virtual machine>,1433`
3. Enter the SQL user name in the **Server user account name** text box.
4. Enter the sql user's password in the **Server user account password** text box.



Shutdown and deallocate virtual machine when not in use

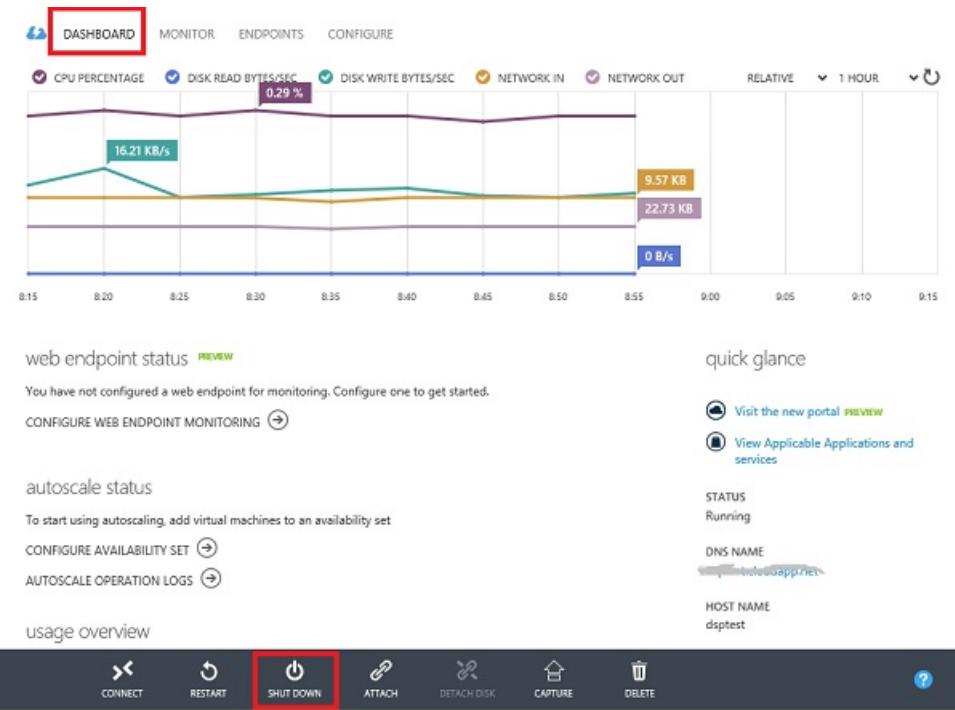
Azure Virtual Machines are priced as **pay only for what you use**. To ensure that you are not being billed when not using your virtual machine, it has to be in the **Stopped (Deallocated)** state.

NOTE

Shutting down the virtual machine from inside (using Windows power options), the VM is stopped but remains allocated. To ensure you're not being billed, always stop virtual machines from the [Azure Classic Portal](#). You can also stop the VM through Powershell by calling ShutdownRoleOperation with "PostShutdownAction" equal to "StoppedDeallocated".

To shutdown and deallocate the virtual machine:

1. Log in to the [Azure Classic Portal](#) using your account.
2. Select **VIRTUAL MACHINES** from the left navigation bar.
3. In the list of virtual machines, click on the name of your virtual machine then go to the **DASHBOARD** page.
4. At the bottom of the page, click **SHUTDOWN**.



The virtual machine will be deallocated but not deleted. You may restart your virtual machine at any time from the Azure Classic Portal.

Your Azure SQL Server VM is ready to use: what's next?

Your virtual machine is now ready to use in your data science exercises. The virtual machine is also ready for use as an IPython Notebook server for the exploration and processing of data, and other tasks in conjunction with Azure Machine Learning and the Team Data Science Process (TDSP).

The next steps in the data science process are mapped in the [Team Data Science Process](#) and may include steps that move data into HDInsight, process and sample it there in preparation for learning from the data with Azure Machine Learning.

Provision the Linux Data Science Virtual Machine

1/17/2017 • 19 min to read • [Edit on GitHub](#)

The Linux Data Science Virtual Machine is an Azure virtual machine that comes with a collection of pre-installed tools. These tools are commonly used for doing data analytics and machine learning. The key software components included are:

- Microsoft R Server Developer Edition
- Anaconda Python distribution (versions 2.7 and 3.5), including popular data analysis libraries
- JupyterHub - a multiuser Jupyter notebook server supporting R, Python, Julia kernels
- Azure Storage Explorer
- Azure command-line interface (CLI) for managing Azure resources
- PostgreSQL Database
- Machine learning tools
 - [Computational Network Toolkit \(CNTK\)](#): A deep learning software toolkit from Microsoft Research.
 - [Vowpal Wabbit](#): A fast machine learning system supporting techniques such as online, hashing, allreduce, reductions, learning2search, active, and interactive learning.
 - [XGBoost](#): A tool providing fast and accurate boosted tree implementation.
 - [Rattle](#) (the R Analytical Tool To Learn Easily): A tool that makes getting started with data analytics and machine learning in R easy, with GUI-based data exploration, and modeling with automatic R code generation.
- Azure SDK in Java, Python, node.js, Ruby, PHP
- Libraries in R and Python for use in Azure Machine Learning and other Azure services
- Development tools and editors (Eclipse, Emacs, gedit, vi)

Doing data science involves iterating on a sequence of tasks:

1. Finding, loading, and pre-processing data
2. Building and testing models
3. Deploying the models for consumption in intelligent applications

Data scientists use various tools to complete these tasks. It can be quite time consuming to find the appropriate versions of the software, and then to download, compile, and install these versions.

The Linux Data Science Virtual Machine can ease this burden substantially. Use it to jump-start your analytics project. It enables you to work on tasks in various languages, including R, Python, SQL, Java, and C++. Eclipse provides an IDE to develop and test your code that is easy to use. The Azure SDK included in the VM allows you to build your applications by using various services on Linux for the Microsoft cloud platform. In addition, you have access to other languages like Ruby, Perl, PHP, and node.js that are also pre-installed.

There are no software charges for this data science VM image. You pay only the Azure hardware usage fees that are assessed based on the size of the virtual machine that you provision with the VM image. More details on the compute fees can be found on the [VM listing page on the Azure Marketplace](#).

Prerequisites

Before you can create a Linux Data Science Virtual Machine, you must have the following:

- **An Azure subscription:** To obtain one, see [Get Azure free trial](#).
- **An Azure storage account:** To create one, see [Create an Azure storage account](#). Alternatively, the storage

account can be created as part of the process of creating the VM, if you do not want to use an existing account.

Create your Linux Data Science Virtual Machine

Here are the steps to create an instance of the Linux Data Science Virtual Machine:

1. Navigate to the virtual machine listing on the [Azure portal](#).
2. Click **Create** (at the bottom) to bring up the wizard.

The screenshot shows the Azure portal interface. On the left, there's a sidebar with options like 'New', 'Resource groups', 'All resources', 'Recent', 'App Services', 'SQL databases', 'Virtual machines (classic)', 'Cloud services (classic)', 'Subscriptions', 'Virtual machines', 'Data factories', 'Storage accounts', 'SQL servers', and 'Storage accounts (classic)'. The main area is titled 'Everything' and has a search bar with 'Data Science Virtual Machine'. Below the search bar is a table with columns 'NAME', 'PUBLISHER', and 'CATEGORY'. The table shows several entries: 'Data Science Virtual Machine' (Microsoft, Virtual Machines), 'Algebraix Analytics' (Algebraix Data, Virtual Machines), 'Algebraix Analytics Enterprise' (Algebraix Data, Virtual Machines), 'Bastacloud Ltd' (Bastacloud Ltd, Virtual Machines), 'Logi Vision Hourly' (Logi Analytics, Virtual Machines), 'Logi Vision Bring Your Own License (BYOL)' (Logi Analytics, Virtual Machines), 'Data Science Virtual Machine (Staged)' (Microsoft, Virtual Machines), and 'Linux Data Science Virtual Machine (Staged)' (Microsoft, Virtual Machines). The last two rows are highlighted with blue backgrounds. To the right of the search results is a large window titled 'Create virtual machine'. It has five numbered steps: 1. Basics (Configure basic settings), 2. Size (Choose virtual machine size), 3. Settings (Configure optional features), 4. Summary (Review the Data Science Virtual Mac...), and 5. Buy. Step 1 is currently active. The 'Basics' step contains fields for 'PUBLISHER' (set to Microsoft), 'About the Data Science Virtual Machine' (link to 'Building Data Science Solutions'), and 'Select a deployment model' (set to Resource Manager). There are also 'Create' and 'Want to deploy programmatically? Get started' buttons. Below the wizard, there are 'USEFUL LINKS' for 'Oscar API - Precognitive Analytics Platform', 'Machine Analytic Unit', and 'Databricks (Linux Image)'. At the bottom of the page, there are links for 'Databricks', 'Zoomdata Server Trial', and 'Oscar API - Precognitive Analytics Platform'.

3. The following sections provide the inputs for each of the steps in the wizard (enumerated on the right of the preceding figure) used to create the Microsoft Data Science Virtual Machine. Here are the inputs needed to configure each of these steps:

a. Basics:

- **Name:** Name of your data science server you are creating.
- **User Name:** First account sign-in ID.
- **Password:** First account password (you can use SSH public key instead of password).
- **Subscription:** If you have more than one subscription, select the one on which the machine is to be created and billed. You must have resource creation privileges for this subscription.
- **Resource Group:** You can create a new one or use an existing group.
- **Location:** Select the data center that is most appropriate. Usually it is the data center that has most of your data, or is closest to your physical location for fastest network access.

b. Size:

- Select one of the server types that meets your functional requirement and cost constraints. Select **View All** to see more choices of VM sizes.

c. Settings:

- **Disk Type:** Choose **Premium** if you prefer a solid state drive (SSD). Otherwise, choose **Standard**.
- **Storage Account:** You can create a new Azure storage account in your subscription, or use an existing one in the same location that was chosen on the **Basics** step of the wizard.
- **Other parameters:** In most cases, you just use the default values. To consider non-default values, hover over the informational link for help on the specific fields.

d. Summary:

- Verify that all information you entered is correct.

e. Buy:

- To start the provisioning, click **Buy**. A link is provided to the terms of the transaction. The VM does not have any additional charges beyond the compute for the server size you chose in the **Size** step.

The provisioning should take about 10-20 minutes. The status of the provisioning is displayed on the Azure portal.

How to access the Linux Data Science Virtual Machine

After the VM is created, you can sign in to it by using SSH. Use the account credentials that you created in the **Basics** section of step 3 for the text shell interface. On Windows, you can download an SSH client tool like [Putty](#). If you prefer a graphical desktop (X Windows System), you can use X11 forwarding on Putty or install the X2Go client.

NOTE

The X2Go client performed significantly better than X11 forwarding in testing. We recommend using the X2Go client for a graphical desktop interface.

Installing and configuring X2Go client

The Linux VM is already provisioned with X2Go server and ready to accept client connections. To connect to the Linux VM graphical desktop, do the following on your client:

1. Download and install the X2Go client for your client platform from [X2Go](#).
2. Run the X2Go client, and select **New Session**. It opens a configuration window with multiple tabs. Enter the following configuration parameters:
 - **Session tab:**
 - **Host:** The host name or IP address of your Linux Data Science VM.
 - **Login:** User name on the Linux VM.
 - **SSH Port:** Leave it at 22, the default value.
 - **Session Type:** Change the value to XFCE. Currently the Linux VM only supports XFCE desktop.
 - **Media tab:** You can turn off sound support and client printing if you don't need to use them.
 - **Shared folders:** If you want directories from your client machines mounted on the Linux VM, add the client machine directories that you want to share with the VM on this tab.

After you sign in to the VM by using either the SSH client or XFCE graphical desktop through the X2Go client, you are ready to start using the tools that are installed and configured on the VM. On XFCE, you can see applications menu shortcuts and desktop icons for many of the tools.

Tools installed on the Linux Data Science Virtual Machine

Microsoft R Open

R is one of the most popular languages for data analysis and machine learning. If you want to use R for your analytics, the VM has Microsoft R Open (MRO) with the Math Kernel Library (MKL). The MKL optimizes math operations common in analytical algorithms. MRO is 100 percent compatible with CRAN-R, and any of the R libraries published in CRAN can be installed on the MRO. You can edit your R programs in one of the default editors, like vi, Emacs, or gedit. You can also download and use other IDEs, such as [RStudio](#). For your convenience, a simple script (`installRStudio.sh`) is provided in the **/dsvm/tools** directory that installs RStudio. If you are using the Emacs editor, note that the Emacs package ESS (Emacs Speaks Statistics), which simplifies working with R files within the Emacs editor, has been pre-installed.

To launch R, you just type **R** in the shell. This takes you to an interactive environment. To develop your R program, you typically use an editor like Emacs or vi or gedit, and then run the scripts within R. If you install RStudio, you

have a full graphical IDE environment to develop your R program.

There is also an R script for you to install the [Top 20 R packages](#) if you want. This script can be run after you are in the R interactive interface, which can be entered (as mentioned) by typing **R** in the shell.

Python

For development using Python, Anaconda Python distribution 2.7 and 3.5 has been installed. This distribution contains the base Python along with about 300 of the most popular math, engineering, and data analytics packages. You can use the default text editors. In addition, you can use Spyder, a Python IDE that is bundled with Anaconda Python distributions. Spyder needs a graphical desktop or X11 forwarding. A shortcut to Spyder is provided in the graphical desktop.

Since we have both Python 2.7 and 3.5, you need to specifically activate the desired Python version you want to work on in the current session. The activation process sets the PATH variable to the desired version of Python.

To activate Python 2.7, run the following from the shell:

```
source /anaconda/bin/activate root
```

Python 2.7 is installed at */anaconda/bin*.

To activate Python 3.5, run the following from the shell:

```
source /anaconda/bin/activate py35
```

Python 3.5 is installed at */anaconda/envs/py35/bin*.

To invoke a Python interactive session, just type **python** in the shell. If you are on a graphical interface or have X11 forwarding set up, you can type **spyder** to launch the Python IDE.

Jupyter notebook

The Anaconda distribution also comes with a Jupyter notebook, an environment to share code and analysis. The Jupyter notebook is accessed through JupyterHub. You sign in using your local Linux user name and password.

The Jupyter notebook server has been pre-configured with Python 2, Python 3, and R kernels. There is a desktop icon named "Jupyter Notebook" to launch the browser to access the notebook server. If you are on the VM via SSH or X2Go client, you can also visit <https://localhost:8000/> to access the Jupyter notebook server.

NOTE

Continue if you get any certificate warnings.

You can access the Jupyter notebook server from any host. Just type <https://<VM DNS name or IP Address>:8000/>

NOTE

Port 8000 is opened in the firewall by default when the VM is provisioned.

We have packaged sample notebooks--one in Python and one in R. You can see the link to the samples on the notebook home page after you authenticate to the Jupyter notebook by using your local Linux user name and password. You can create a new notebook by selecting **New**, and then the appropriate language kernel. If you don't see the **New** button, click the **Jupyter** icon on the top left to go to the home page of the notebook server.

IDEs and editors

You have a choice of several code editors. This includes vi/VIM, Emacs, gEdit and Eclipse. gEdit and Eclipse are

graphical editors, and need you to be signed in to a graphical desktop to use them. These editors have desktop and application menu shortcuts to launch them.

VIM and **Emacs** are text-based editors. On Emacs, we have installed an add-on package called Emacs Speaks Statistics (ESS) that makes working with R easier within the Emacs editor. More information can be found at [ESS](#).

Eclipse is an open source, extensible IDE that supports multiple languages. The Java developers edition is the instance installed on the VM. There are plugins available for several popular languages that can be installed to extend the Eclipse environment. We also have a plugin installed in Eclipse called **Azure Toolkit for Eclipse**. It allows you to create, develop, test, and deploy Azure applications using the Eclipse development environment that supports languages like Java. There is also an **Azure SDK for Java** that allows access to different Azure services from within a Java environment. More information on Azure toolkit for Eclipse can be found at [Azure Toolkit for Eclipse](#).

LaTex is installed through the texlive package along with an Emacs add-on [auctex](#) package, which simplifies authoring your LaTex documents within Emacs.

Databases

Postgres

The open source database **Postgres** is available on the VM, with the services running and initdb already completed. You still need to create databases and users. For more information, see the [Postgres documentation](#).

Graphical SQL client

SQuirrel SQL, a graphical SQL client, has been provided to connect to different databases (such as Microsoft SQL Server, Postgres, and MySQL) and to run SQL queries. You can run this from a graphical desktop session (using the X2Go client, for example). To invoke SQuirrel SQL, you can either launch it from the icon on the desktop or run the following command on the shell.

```
/usr/local/squirrel-sql-3.7/squirrel-sql.sh
```

Before the first use, set up your drivers and database aliases. The JDBC drivers are located at:

```
/usr/share/java/jdbcdrivers
```

For more information, see [SQuirrel SQL](#).

Command-line tools for accessing Microsoft SQL Server

The ODBC driver package for SQL Server also comes with two command-line tools:

bcp: The bcp utility bulk copies data between an instance of Microsoft SQL Server and a data file in a user-specified format. The bcp utility can be used to import large numbers of new rows into SQL Server tables, or to export data out of tables into data files. To import data into a table, you must either use a format file created for that table, or understand the structure of the table and the types of data that are valid for its columns.

For more information, see [Connecting with bcp](#).

sqlcmd: You can enter Transact-SQL statements with the sqlcmd utility, as well as system procedures, and script files at the command prompt. This utility uses ODBC to execute Transact-SQL batches.

For more information, see [Connecting with sqlcmd](#).

NOTE

There are some differences in this utility between Linux and Windows platforms. See the documentation for details.

Database access libraries

There are libraries available in R and Python to access databases.

- In R, the **RODBC** package or **dplyr** package allows you to query or execute SQL statements on the database server.
- In Python, the **pyodbc** library provides database access with ODBC as the underlying layer.

To access **Postgres**:

- From R: Use the package **RPostgreSQL**.
- From Python: Use the **psycopg2** library.

Azure tools

The following Azure tools are installed on the VM:

- **Azure command-line interface:** The Azure CLI allows you to create and manage Azure resources through shell commands. To invoke the Azure tools, just type **azure help**. For more information, see the [Azure CLI documentation page](#).
- **Microsoft Azure Storage Explorer:** Microsoft Azure Storage Explorer is a graphical tool that is used to browse through the objects that you have stored in your Azure storage account, and to upload and download data to and from Azure blobs. You can access Storage Explorer from the desktop shortcut icon. You can invoke it from a shell prompt by typing **StorageExplorer**. You need to be signed in from an X2Go client, or have X11 forwarding set up.
- **Azure Libraries:** The following are some of the pre-installed libraries.
 - **Python:** The Azure-related libraries in Python that are installed are **azure**, **azureml**, **pydocumentdb**, and **pyodbc**. With the first three libraries, you can access Azure storage services, Azure Machine Learning, and Azure DocumentDB (a NoSQL database on Azure). The fourth library, pyodbc (along with the Microsoft ODBC driver for SQL Server), enables access to SQL Server, Azure SQL Database, and Azure SQL Data Warehouse from Python by using an ODBC interface. Enter **pip list** to see all the listed libraries. Be sure to run this command in both the Python 2.7 and 3.5 environments.
 - **R:** The Azure-related libraries in R that are installed are **AzureML** and **RODBC**.
 - **Java:** The list of Azure Java libraries can be found in the directory **/dsvm/sdk/AzureSDKJava** on the VM. The key libraries are Azure storage and management APIs, DocumentDB, and JDBC drivers for SQL Server.

You can access the [Azure portal](#) from the pre-installed Firefox browser. On the Azure portal, you can create, manage, and monitor Azure resources.

Azure Machine Learning

Azure Machine Learning is a fully managed cloud service that enables you to build, deploy, and share predictive analytics solutions. You build your experiments and models from Azure Machine Learning Studio. It can be accessed from a web browser on the data science virtual machine by visiting [Microsoft Azure Machine Learning](#).

After you sign in to Azure Machine Learning Studio, you have access to an experimentation canvas where you can build a logical flow for the machine learning algorithms. You also have access to a Jupyter notebook hosted on Azure Machine Learning and can work seamlessly with the experiments in Machine Learning Studio. Operationalize the machine learning models that you have built by wrapping them in a web service interface. This enables clients written in any language to invoke predictions from the machine learning models. For more information, see the [Machine Learning documentation](#).

You can also build your models in R or Python on the VM, and then deploy it in production on Azure Machine Learning. We have installed libraries in R (**AzureML**) and Python (**azureml**) to enable this functionality.

For information on how to deploy models in R and Python into Azure Machine Learning, see [Ten things you can do on the Data science Virtual Machine](#) (in particular, the section "Build models using R or Python and Operationalize them using Azure Machine Learning").

NOTE

These instructions were written for the Windows version of the Data Science VM. But the information provided there on deploying models to Azure Machine Learning is applicable to the Linux VM.

Machine learning tools

The VM comes with a few machine learning tools and algorithms that have been pre-compiled and pre-installed locally. These include:

- **CNTK** (Computational Network Toolkit from Microsoft Research): A deep learning toolkit.
- **Vowpal Wabbit**: A fast online learning algorithm.
- **xgboost**: A tool that provides optimized, boosted tree algorithms.
- **Python**: Anaconda Python comes bundled with machine learning algorithms with libraries like Scikit-learn. You can install other libraries by using the `pip install` command.
- **R**: A rich library of machine learning functions is available for R. Some of the libraries that are pre-installed are lm, glm, randomForest, rpart. Other libraries can be installed by running:

```
install.packages(<lib name>)
```

Here is some additional information about the first three machine learning tools in the list.

CNTK

This is an open source, deep learning toolkit. It is a command-line tool (cntk), and is already in the PATH.

To run a basic sample, execute the following commands in the shell:

```
# Copy samples to your home directory and execute cntk
cp -r /dsvm/tools/CNTK-2016-02-08-Linux-64bit-CPU-Only/Examples/Other/Simple2d cntkdemo
cd cntkdemo/Data
cntk configFile=../Config/Simple.cntk
```

The model output is in `~/cntkdemo/Output/Models`.

For more information, see the CNTK section of [GitHub](#), and the [CNTK wiki](#).

Vowpal Wabbit

Vowpal Wabbit is a machine learning system that uses techniques such as online, hashing, allreduce, reductions, learning2search, active, and interactive learning.

To run the tool on a very basic example, do the following:

```
cp -r /dsvm/tools/VowpalWabbit/demo vwdemo
cd vwdemo
vw house_dataset
```

There are other, larger demos in that directory. For more information on VW, see [this section of GitHub](#), and the [Vowpal Wabbit wiki](#).

xgboost

This is a library that is designed and optimized for boosted (tree) algorithms. The objective of this library is to push the computation limits of machines to the extremes needed to provide large-scale tree boosting that is scalable, portable, and accurate.

It is provided as a command line as well as an R library.

To use this library in R, you can start an interactive R session (just by typing `R` in the shell), and load the library.

Here is a simple example you can run in R prompt:

```
library(xgboost)

data(agaricus.train, package='xgboost')
data(agaricus.test, package='xgboost')
train <- agaricus.train
test <- agaricus.test
bst <- xgboost(data = train$data, label = train$label, max.depth = 2,
                 eta = 1, nthread = 2, nround = 2, objective = "binary:logistic")
pred <- predict(bst, test$data)
```

To run the xgboost command line, here are the commands to execute in the shell:

```
cp -r /dsvm/tools/xgboost/demo/binary_classification/ xgboostdemo
cd xgboostdemo
xgboost mushroom.conf
```

A .model file is written to the directory specified. Information about this demo example can be found [on GitHub](#).

For more information about xgboost, see the [xgboost documentation page](#), and its [Github repository](#).

Rattle

Rattle (the **R Analytical Tool To Learn Easily**) uses GUI-based data exploration and modeling. It presents statistical and visual summaries of data, transforms data that can be readily modeled, builds both unsupervised and supervised models from the data, presents the performance of models graphically, and scores new data sets. It also generates R code, replicating the operations in the UI that can be run directly in R or used as a starting point for further analysis.

To run Rattle, you need to be in a graphical desktop sign-in session. On the terminal, type **R** to enter the R environment. At the R prompt, enter the following commands:

```
library(rattle)
rattle()
```

Now a graphical interface opens up with a set of tabs. Here are the quick start steps in Rattle needed to use a sample weather data set and build a model. In some of the steps below, you are prompted to automatically install and load some required R packages that are not already on the system.

NOTE

If you don't have access to install the package in the system directory (the default), you may see a prompt on your R console window to install packages to your personal library. Answer **y** if you see these prompts.

1. Click **Execute**.
2. A dialog pops up, asking you if you like to use the example weather data set. Click **Yes** to load the example.
3. Click the **Model** tab.
4. Click **Execute** to build a decision tree.
5. Click **Draw** to display the decision tree.
6. Click the **Forest** radio button, and click **Execute** to build a random forest.
7. Click the **Evaluate** tab.
8. Click the **Risk** radio button, and click **Execute** to display two Risk (Cumulative) performance plots.
9. Click the **Log** tab to show the generate R code for the preceding operations. (Due to a bug in the current release of Rattle, you need to insert a # character in front of *Export this log ...* in the text of the log.)
10. Click the **Export** button to save the R script file named *weather_script.R* to the home folder.

You can exit Rattle and R. Now you can modify the generated R script, or use it as it is to run it anytime to repeat everything that was done within the Rattle UI. Especially for beginners in R, this is an easy way to quickly do analysis and machine learning in a simple graphical interface, while automatically generating code in R to modify and/or learn.

Next steps

Here's how you can continue your learning and exploration:

- The [Data science on the Linux Data Science Virtual Machine](#) walkthrough shows you how to perform several common data science tasks with the Linux Data Science VM provisioned here.
- Explore the various data science tools on the data science VM by trying out the tools described in this article. You can also run `dsvm-more-info` on the shell within the virtual machine for a basic introduction and pointers to more information about the tools installed on the VM.
- Learn how to build end-to-end analytical solutions systematically by using the [Team Data Science Process](#).
- Visit the [Cortana Analytics Gallery](#) for machine learning and data analytics samples that use the Cortana Analytics Suite.

Data science on the Linux Data Science Virtual Machine

1/17/2017 • 19 min to read • [Edit on GitHub](#)

This walkthrough shows you how to perform several common data science tasks with the Linux Data Science VM. The Linux Data Science Virtual Machine (DSVM) is a virtual machine image available on Azure that is pre-installed with a collection of tools commonly used for data analytics and machine learning. The key software components are itemized in the [Provision the Linux Data Science Virtual Machine](#) topic. The VM image makes it easy to get started doing data science in minutes, without having to install and configure each of the tools individually. You can easily scale up the VM, if needed, and stop it when not in use. So this resource is both elastic and cost-efficient.

The data science tasks demonstrated in this walkthrough follow the steps outlined in the [Team Data Science Process](#). This process provides a systematic approach to data science that enables teams of data scientists to effectively collaborate over the lifecycle of building intelligent applications. The data science process also provides an iterative framework for data science that can be followed by an individual.

We analyze the [spambase](#) dataset in this walkthrough. This is a set of emails that are marked as either spam or ham (meaning they are not spam), and also contains some statistics on the content of the emails. The statistics included are discussed in the next but one section.

Prerequisites

Before you can use a Linux Data Science Virtual Machine, you must have the following:

- An **Azure subscription**. If you do not already have one, see [Create your free Azure account today](#).
- A **Linux data science VM**. For information on provisioning this VM, see [Provision the Linux Data Science Virtual Machine](#).
- **X2Go** installed on your computer and opened an XFCE session. For information on installing and configuring an **X2Go client**, see [Installing and configuring X2Go client](#).
- An **AzureML account**. If you don't already have one, sign up for new one at the [AzureML homepage](#). There is a free usage tier to help you get started.

Download the spambase dataset

The [spambase](#) dataset is a relatively small set of data that contains only 4601 examples. This is a convenient size to use when demonstrating that some of the key features of the Data Science VM as it keeps the resource requirements modest.

NOTE

This walkthrough was created on a D2 v2-sized Linux Data Science Virtual Machine. This size DSVM is capable of handling the procedures in this walkthrough.

If you need more storage space, you can create additional disks and attach them to your VM. These disks use persistent Azure storage, so their data is preserved even when the server is reprovisioned due to resizing or is shut down. To add a disk and attach it to your VM, follow the instructions in [Add a disk to a Linux VM](#). These steps use the Azure Command-Line Interface (Azure CLI), which is already installed on the DSVM. So these procedures can be done entirely from the VM itself. Another option to increase storage is to use [Azure files](#).

To download the data, open a terminal window and run this command:

```
 wget http://archive.ics.uci.edu/ml/machine-learning-databases/spambase/spambase.data
```

The downloaded file does not have a header row, so let's create another file that does have a header. Run this command to create a file with the appropriate headers:

```
echo 'word_freq_make,word_freq_address,word_freq_all,word_freq_3d,word_freq_our,word_freq_over,word_freq_remove,
word_freq_internet,word_freq_order,word_freq_mail,word_freq_receive,word_freq_will,word_freq_people,word_freq_report,
word_freq_addresses,word_freq_free,word_freq_business,word_freq_email,word_freq_you,word_freq_credit,word_freq_your,word_freq_font,
word_freq_000,word_freq_money,word_freq_hp,word_freq_hpl,word_freq_george,word_freq_650,word_freq_lab,word_freq_labs,
word_freq_telnet,word_freq_857,word_freq_data,word_freq_415,word_freq_85,word_freq_technology,word_freq_1999,word_freq_parts,
word_freq_pm,word_freq_direct,word_freq_cs,word_freq_meeting,word_freq_original,word_freq_project,word_freq_re,
word_freq_edu,word_freq_table,word_freq_conference,char_freq_semicolon,char_freq_leftParen,char_freq_leftBracket,char_freq_exclamation,
char_freq_dollar,char_freq_pound,capital_run_length_average,capital_run_length_longest,capital_run_length_total,spam' > headers
```

Then concatenate the two files together with the command:

```
cat spambase.data >> headers
mv headers spambaseHeaders.data
```

The dataset has several types of statistics on each email:

- Columns like ***word_freq WORD*** indicate the percentage of words in the email that match *WORD*. For example, if *word_freq_make* is 1, then 1% of all words in the email were *make*.
- Columns like ***char_freq CHAR*** indicate the percentage of all characters in the email that were *CHAR*.
- ***capital_run_length_longest*** is the longest length of a sequence of capital letters.
- ***capital_run_length_average*** is the average length of all sequences of capital letters.
- ***capital_run_length_total*** is the total length of all sequences of capital letters.
- ***spam*** indicates whether the email was considered spam or not (1 = spam, 0 = not spam).

Explore the dataset with Microsoft R Open

Let's examine the data and do some basic machine learning with R. The Data Science VM comes with [Microsoft R Open](#) pre-installed. The multithreaded math libraries in this version of R offer better performance than various single-threaded versions. Microsoft R Open also provides reproducibility by using a snapshot of the CRAN package repository.

To get copies of the code samples used in this walkthrough, clone the **Azure-Machine-Learning-Data-Science** repository using git, which is pre-installed on the VM. From the git command line, run:

```
git clone https://github.com/Azure/Azure-MachineLearning-DataScience.git
```

Open a terminal window and start a new R session with the R interactive console.

NOTE

You can also use RStudio for the following procedures. To install RStudio, execute this command at a terminal:

```
./Desktop/DSVM/tools/installRStudio.sh
```

To import the data and set up the environment, run:

```
data <- read.csv("spambaseHeaders.data")
set.seed(123)
```

To see summary statistics about each column:

```
summary(data)
```

For a different view of the data:

```
str(data)
```

This shows you the type of each variable and the first few values in the dataset.

The *spam* column was read as an integer, but it's actually a categorical variable (or factor). To set its type:

```
data$spam <- as.factor(data$spam)
```

To do some exploratory analysis, use the [ggplot2](#) package, a popular graphing library for R that is already installed on the VM. Note, from the summary data displayed earlier, that we have summary statistics on the frequency of the exclamation mark character. Let's plot those frequencies here with the following commands:

```
library(ggplot2)
ggplot(data) + geom_histogram(aes(x=char_freq_exclamation), binwidth=0.25)
```

Since the zero bar is skewing the plot, let's get rid of it:

```
email_with_exclamation = data[data$char_freq_exclamation > 0, ]
ggplot(email_with_exclamation) + geom_histogram(aes(x=char_freq_exclamation), binwidth=0.25)
```

There is a non-trivial density above 1 that looks interesting. Let's look at just that data:

```
ggplot(data[data$char_freq_exclamation > 1, ]) + geom_histogram(aes(x=char_freq_exclamation), binwidth=0.25)
```

Then split it by spam vs ham:

```
ggplot(data[data$char_freq_exclamation > 1, ], aes(x=char_freq_exclamation)) +
  geom_density(lty=3) +
  geom_density(aes(fill=spam, colour=spam), alpha=0.55) +
  xlab("spam") +
  ggtitle("Distribution of spam\nby frequency of !") +
  labs(fill="spam", y="Density")
```

These examples should enable you to make similar plots of the other columns to explore the data contained in them.

Train and test an ML model

Now let's train a couple of machine learning models to classify the emails in the dataset as containing either spam or ham. We train a decision tree model and a random forest model in this section and then test their accuracy of their predictions.

NOTE

The rpart (Recursive Partitioning and Regression Trees) package used in the following code is already installed on the Data Science VM.

First, let's split the dataset into training and test sets:

```

md <- runif(dim(data)[1])
trainSet = subset(data, md <= 0.7)
testSet = subset(data, md > 0.7)

```

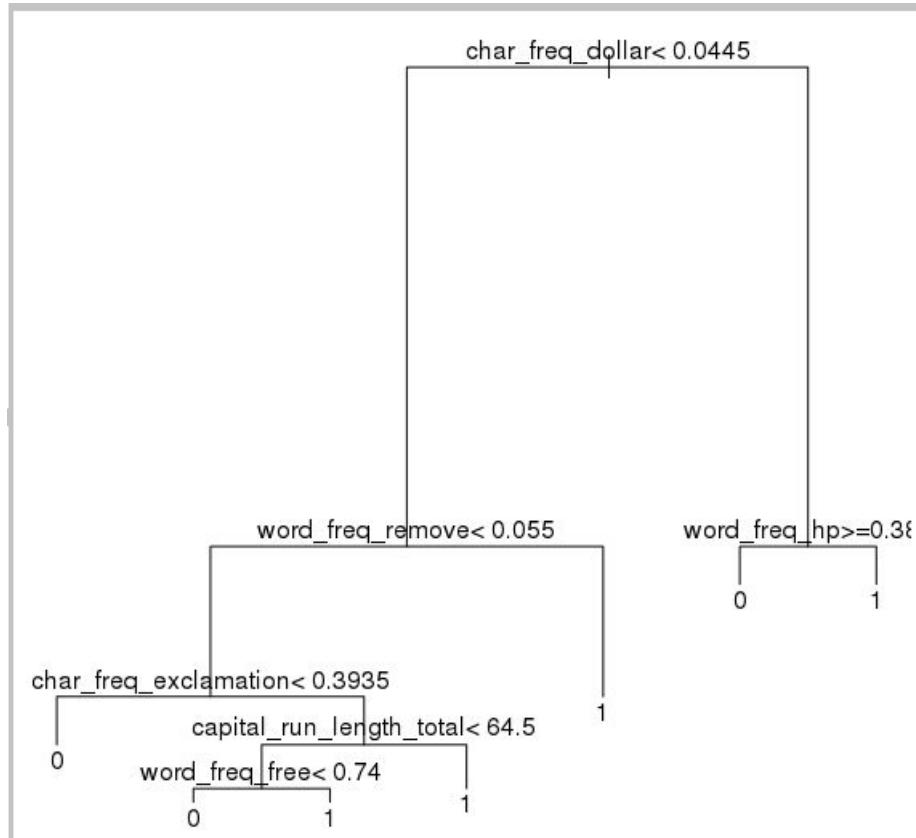
And then create a decision tree to classify the emails.

```

require(rpart)
model.rpart <- rpart(spam ~ ., method = "class", data = trainSet)
plot(model.rpart)
text(model.rpart)

```

Here is the result:



To determine how well it performs on the training set, use the following code:

```

trainSetPred <- predict(model.rpart, newdata = trainSet, type = "class")
t <- table(`Actual Class` = trainSet$spam, `Predicted Class` = trainSetPred)
accuracy <- sum(diag(t))/sum(t)
accuracy

```

To determine how well it performs on the test set:

```

testSetPred <- predict(model.rpart, newdata = testSet, type = "class")
t <- table(`Actual Class` = testSet$spam, `Predicted Class` = testSetPred)
accuracy <- sum(diag(t))/sum(t)
accuracy

```

Let's also try a random forest model. Random forests train a multitude of decision trees and output a class that is the mode of the classifications from all of the individual decision trees. They provide a more powerful machine learning approach as they correct for the tendency of a decision tree model to overfit a training dataset.

```

require(randomForest)
trainVars <- setdiff(colnames(data), 'spam')
model.rf <- randomForest(x=trainSet[, trainVars], y=trainSet$spam)

trainSetPred <- predict(model.rf, newdata = trainSet[, trainVars], type = "class")
table('Actual Class' = trainSet$spam, 'Predicted Class' = trainSetPred)

testSetPred <- predict(model.rf, newdata = testSet[, trainVars], type = "class")
t <- table('Actual Class' = testSet$spam, 'Predicted Class' = testSetPred)
accuracy <- sum(diag(t))/sum(t)
accuracy

```

Deploy a model to Azure ML

[Azure Machine Learning Studio](#) (AzureML) is a cloud service that makes it easy to build and deploy predictive analytics models. One of the nice features of AzureML is its ability to publish any R function as a web service. The AzureML R package makes deployment easy to do right from our R session on the DSVM.

To deploy the decision tree code from the previous section, you need to sign in to Azure Machine Learning Studio. You need your workspace ID and an authorization token to sign in. To find these values and initialize the AzureML variables with them:

Select **Settings** on the left-hand menu. Note your **WORKSPACE ID**.

NAME	AUTHORIZATION TOKENS	USERS	DATA GATEWAYS
WORKSPACE NAME	[REDACTED]		
WORKSPACE DESCRIPTION	Default workspace.		
WORKSPACE TYPE	Free	Learn More	
WORKSPACE ID	abcdefghijklmnopqrstuvwxyz		
WORKSPACE STORAGE	USED	AVAILABLE	

Select **Authorization Tokens** from the overhead menu and note your **Primary Authorization Token**.

Load the **AzureML** package and then set values of the variables with your token and workspace ID in your R session on the DSVM:

```
require(AzureML)
wsAuth = "<authorization-token>"
wsID = "<workspace-id>"
```

Let's simplify the model to make this demonstration easier to implement. Pick the three variables in the decision tree closest to the root and build a new tree using just those three variables:

```
colNames <- c("char_freq_dollar", "word_freq_remove", "word_freq_hp", "spam")
smallTrainSet <- trainSet[, colNames]
smallTestSet <- testSet[, colNames]
model.rpart <- rpart(spam ~ ., method = "class", data = smallTrainSet)
```

We need a prediction function that takes the features as an input and returns the predicted values:

```
predictSpam <- function(char_freq_dollar, word_freq_remove, word_freq_hp) {
  predictDF <- predict(model.rpart, data.frame("char_freq_dollar" = char_freq_dollar,
    "word_freq_remove" = word_freq_remove, "word_freq_hp" = word_freq_hp))
  return(colnames(predictDF)[apply(predictDF, 1, which.max)])
}
```

Publish the **predictSpam** function to AzureML using the **publishWebService** function:

```
spamWebService <- publishWebService("predictSpam",
  "spamWebService",
  list("char_freq_dollar" = "float", "word_freq_remove" = "float", "word_freq_hp" = "float"),
  list("spam" = "int"),
  wsID, wsAuth)
```

This function takes the **predictSpam** function, creates a web service named **spamWebService** with defined inputs and outputs, and returns information about the new endpoint.

View details of the published web service, including its API endpoint and access keys with the command:

```
spamWebService[[2]]
```

To try it out on the first 10 rows of the test set:

```
consumeDataframe(spamWebService$endpoints[[1]]$PrimaryKey, spamWebService$endpoints[[1]]$ApiLocation, smallTestSet[1:10, 1:3])
```

Use other tools available

The remaining sections show how to use some of the tools installed on the Linux Data Science VM. Here is the list of tools discussed:

- XGBoost
- Python
- Jupyterhub
- Rattle
- PostgreSQL & Squirrel SQL
- SQL Server Data Warehouse

XGBoost

[XGBoost](#) is a tool that provides a fast and accurate boosted tree implementation.

```
require(xgboost)
data <- read.csv("spambaseHeaders.data")
set.seed(123)

rnd <- runif(dim(data)[1])
trainSet = subset(data, rnd <= 0.7)
testSet = subset(data, rnd > 0.7)

bst <- xgboost(data = data.matrix(trainSet[, 0:57]), label = trainSet$spam, nthread = 2, nrounds = 2, objective = "binary:logistic")

pred <- predict(bst, data.matrix(testSet[, 0:57]))
accuracy <- 1.0 - mean(as.numeric(pred > 0.5) != testSet$spam)
print(paste("test accuracy = ", accuracy))
```

XGBoost can also call from python or a command line.

Python

For development using Python, the Anaconda Python distributions 2.7 and 3.5 have been installed in the DSVM.

NOTE

The Anaconda distribution includes [Condas](#), which can be used to create custom environments for Python that have different versions and/or packages installed in them.

Let's read in some of the spambase dataset and classify the emails with support vector machines in scikit-learn:

```
import pandas
from sklearn import svm
data = pandas.read_csv("spambaseHeaders.data", sep = '\s*')
X = data.ix[:, 0:57]
y = data.ix[:, 57]
clf = svm.SVC()
clf.fit(X, y)
```

To make predictions:

```
clf.predict(X.ix[0:20,:])
```

To show how to publish an AzureML endpoint, let's make a simpler model the three variables as we did when we published the R model previously.

```
X = data.ix[["char_freq_dollar", "word_freq_remove", "word_freq_hp"]]
y = data.ix[:, 57]
clf = svm.SVC()
clf.fit(X, y)
```

To publish the model to AzureML:

```
# Publish the model.
workspace_id = "<workspace-id>"
workspace_token = "<workspace-token>"
from azureml import services
@services.publish(workspace_id, workspace_token)
@services.types(char_freq_dollar = float, word_freq_remove = float, word_freq_hp = float)
@services.returns(int) # 0 or 1
def predictSpam(char_freq_dollar, word_freq_remove, word_freq_hp):
    inputArray = [char_freq_dollar, word_freq_remove, word_freq_hp]
    return clf.predict(inputArray)

# Get some info about the resulting model.
predictSpam.service.url
predictSpam.service.api_key

# Call the model
predictSpam.service(1, 1, 1)
```

NOTE

This is only available for python 2.7 and is not yet supported on 3.5. Run with [/anaconda/bin/python2.7](#).

Jupyterhub

The Anaconda distribution in the DSVM comes with a Jupyter notebook, a cross-platform environment to share Python, R, or Julia code and analysis. The Jupyter notebook is accessed through JupyterHub. You sign in using your local Linux user name and password at <https://<VM DNS name or IP Address>:8000/>. All configuration files for JupyterHub are found in directory **/etc/jupyterhub**.

Several sample notebooks are already installed on the VM:

- See the [IntroToJupyterPython.ipynb](#) for a sample Python notebook.
- See [IntroTutorialinR](#) for a sample R notebook.
- See the [IrisClassifierPyMLWebService](#) for another sample Python notebook.

NOTE

The Julia language is also available from the command line on the Linux Data Science VM.

Rattle

[Rattle](#) (the R Analytical Tool To Learn Easily) is a graphical R tool for data mining. It has an intuitive interface that makes it easy to load, explore, and transform data and build and evaluate models. The article [Rattle: A Data Mining GUI for R](#) provides a walkthrough that demonstrates its features.

Install and start Rattle with the following commands:

```
if(!require("rattle")) install.packages("rattle")
require(rattle)
rattle()
```

NOTE

Installation is not required on the DSVM. But Rattle may prompt you to install additional packages when it loads.

Rattle uses a tab-based interface. Most of the tabs correspond to steps in the [Data Science Process](#), like loading data or exploring it. The data science process flows from left to right through the tabs. But the last tab contains a log of the R commands run by Rattle.

To load and configure the dataset:

- To load the file, select the **Data** tab, then
- Choose the selector next to **Filename** and choose **spambaseHeaders.data**.
- To load the file, select **Execute** in the top row of buttons. You should see a summary of each column, including its identified data type, whether it's an input, a target, or other type of variable, and the number of unique values.
- Rattle has correctly identified the **spam** column as the target. Select the spam column, then set the **Target Data Type** to **Categoric**.

To explore the data:

- Select the **Explore** tab.
- Click **Summary**, then **Execute**, to see some information about the variable types and some summary statistics.
- To view other types of statistics about each variable, select other options like **Describe** or **Basics**.

The **Explore** tab also allows you to generate many insightful plots. To plot a histogram of the data:

- Select **Distributions**.
- Check **Histogram** for **word_freq_remove** and **word_freq_you**.
- Select **Execute**. You should see both density plots in a single graph window, where it is clear that the word "you" appears much more frequently in emails than "remove".

The Correlation plots are also interesting. To create one:

- Choose **Correlation** as the **Type**, then
- Select **Execute**.
- Rattle warns you that it recommends a maximum of 40 variables. Select **Yes** to view the plot.

There are some interesting correlations that come up: "technology" is strongly correlated to "HP" and "labs", for example. It is also strongly correlated to "650", because the area code of the dataset donors is 650.

The numeric values for the correlations between words are available in the Explore window. It is interesting to note, for example, that "technology" is negatively correlated with "your" and "money".

Rattle can transform the dataset to handle some common issues. For example, it allows you to rescale features, impute missing values, handle outliers, and remove variables or observations with missing data. Rattle can also identify association rules between observations and/or variables. These tabs are out of scope for this introductory walkthrough.

Rattle can also perform cluster analysis. Let's exclude some features to make the output easier to read. On the **Data** tab, choose **Ignore** next to each of the variables except these ten items:

- word_freq_hp
- word_freq_technology
- word_freq_george
- word_freq_remove
- word_freq_your
- word_freq_dollar
- word_freq_money
- capital_run_length_longest
- word_freq_business
- spam

Then go back to the **Cluster** tab, choose **KMeans**, and set the *Number of clusters* to 4. Then **Execute**. The results are displayed in the output window. One cluster has high frequency of "george" and "hp" and is probably a legitimate business email.

To build a simple decision tree machine learning model:

- Select the **Model** tab,
- Choose **Tree** as the **Type**.
- Select **Execute** to display the tree in text form in the output window.
- Select the **Draw** button to view a graphical version. This looks quite similar to the tree we obtained earlier using *rpart*.

One of the nice features of Rattle is its ability to run several machine learning methods and quickly evaluate them. Here is the procedure:

- Choose **All** for the **Type**.
- Select **Execute**.
- After it finishes you can click any single **Type**, like **SVM**, and view the results.
- You can also compare the performance of the models on the validation set using the **Evaluate** tab. For example, the **Error Matrix** selection shows you the confusion matrix, overall error, and averaged class error for each model on the validation set.
- You can also plot ROC curves, perform sensitivity analysis, and do other types of model evaluations.

Once you're finished building models, select the **Log** tab to view the R code run by Rattle during your session. You can select the **Export** button to save it.

NOTE

There is a bug in current release of Rattle. To modify the script or use it to repeat your steps later, you must insert a # character in front of *Export this log ... * in the text of the log.

PostgreSQL & Squirrel SQL

The DSVM comes with PostgreSQL installed. PostgreSQL is a sophisticated, open-source relational database. This section shows how to load our spam dataset into PostgreSQL and then query it.

Before you can load the data, you need to allow password authentication from the localhost. At a command prompt:

```
sudo gedit /var/lib/pgsql/data/pg_hba.conf
```

Near the bottom of the config file are several lines that detail the allowed connections:

```
# "local" is for Unix domain socket connections only
local all      all          trust
# IPv4 local connections:
host  all      all      127.0.0.1/32    ident
# IPv6 local connections:
host  all      all      ::1/128       ident
```

Change the "IPv4 local connections" line to use md5 instead of ident, so we can log in using a username and password:

```
# IPv4 local connections:
host  all      all      127.0.0.1/32    md5
```

And restart the postgres service:

```
sudo systemctl restart postgresql
```

To launch psql, an interactive terminal for PostgreSQL, as the built-in postgres user, run the following command from a prompt:

```
sudo -u postgres psql
```

Create a new user account, using the same username as the Linux account you're currently logged in as, and give it a password:

```
CREATE USER <username> WITH CREATEDB;
CREATE DATABASE <username>;
ALTER USER <username> password '<password>';
\quit
```

Then log in to psql as your user:

```
psql
```

And import the data into a new database:

```
CREATE DATABASE spam;
\c spam
CREATE TABLE data (word_freq_make real, word_freq_address real, word_freq_all real, word_freq_3d real, word_freq_our real, word_freq_over real, word_freq_remove real, word_freq_internet real, word_freq_order real, word_freq_mail real, word_freq_receive real, word_freq_will real, word_freq_people real, word_freq_report real, word_freq_addresses real, word_freq_free real, word_freq_business real, word_freq_email real, word_freq_you real, word_freq_credit real, word_freq_your real, word_freq_font real, word_freq_000 real, word_freq_money real, word_freq_hp real, word_freq_hpl real, word_freq_george real, word_freq_650 real, word_freq_lab real, word_freq_labs real, word_freq_telnet real, word_freq_857 real, word_freq_data real, word_freq_415 real, word_freq_85 real, word_freq_technology real, word_freq_1999 real, word_freq_parts real, word_freq_pm real, word_freq_direct real, word_freq_cs real, word_freq_meeting real, word_freq_original real, word_freq_project real, word_freq_re real, word_freq_edu real, word_freq_table real, word_freq_conference real, char_freq_semicolon real, char_freq_leftParen real, char_freq_leftBracket real, char_freq_exclamation real, char_freq_dollar real, char_freq_pound real, capital_run_length_average real, capital_run_length_longest real, capital_run_length_total real, spam integer);
\copy data FROM /home/<username>/spambase.data DELIMITER ',' CSV;
\quit
```

Now, let's explore the data and run some queries using **Squirrel SQL**, a graphical tool that lets you interact with databases via a JDBC driver.

To get started, launch Squirrel SQL from the Applications menu. To set up the driver:

- Select **Windows**, then **View Drivers**.
- Right-click on **PostgreSQL** and select **Modify Driver**.

- Select **Extra Class Path**, then **Add**.
- Enter **/usr/share/java/jdbcdrivers/postgresql-9.4.1208.jre6.jar** for the **File Name** and
- Select **Open**.
- Choose List Drivers, then select **org.postgresql.Driver** in **Class Name**, and select **OK**.

To set up the connection to the local server:

- Select **Windows**, then **View Aliases**.
- Choose the + button to make a new alias.
- Name it *Spam database*, choose **PostgreSQL** in the **Driver** drop-down.
- Set the URL to *jdbc:postgresql://localhost/spam*.
- Enter your *username* and *password*.
- Click **OK**.
- To open the **Connection** window, double-click the **Spam database** alias.
- Select **Connect**.

To run some queries:

- Select the **SQL** tab.
- Enter a simple query such as `SELECT * from data;` in the query textbox at the top of the SQL tab.
- Press **Ctrl-Enter** to run it. By default Squirrel SQL returns the first 100 rows from your query.

There are many more queries you could run to explore this data. For example, how does the frequency of the word *make* differ between spam and ham?

```
SELECT avg(word_freq_make), spam from data group by spam;
```

Or what are the characteristics of email that frequently contain *3d*?

```
SELECT * from data order by word_freq_3d desc;
```

Most emails that have a high occurrence of *3d* are apparently spam, so it could be a useful feature for building a predictive model to classify the emails.

If you wanted to perform machine learning with data stored in a PostgreSQL database, consider using [MADlib](#).

SQL Server Data Warehouse

Azure SQL Data Warehouse is a cloud-based, scale-out database capable of processing massive volumes of data, both relational and non-relational. For more information, see [What is Azure SQL Data Warehouse?](#)

To connect to the data warehouse and create the table, run the following command from a command prompt:

```
sqlcmd -S <server-name>.database.windows.net -d <database-name> -U <username> -P <password> -I
```

Then at the sqlcmd prompt:

```
CREATE TABLE spam(word_freq_make real, word_freq_address real, word_freq_all real, word_freq_3d real, word_freq_our real, word_freq_over real, word_freq_remove real, word_freq_internet real, word_freq_order real, word_freq_mail real, word_freq_receive real, word_freq_will real, word_freq_people real, word_freq_report real, word_freq_addresses real, word_freq_free real, word_freq_business real, word_freq_email real, word_freq_you real, word_freq_credit real, word_freq_your real, word_freq_font real, word_freq_000 real, word_freq_money real, word_freq_hp real, word_freq_hpl real, word_freq_george real, word_freq_650 real, word_freq_lab real, word_freq_labs real, word_freq_telnet real, word_freq_857 real, word_freq_data real, word_freq_415 real, word_freq_85 real, word_freq_technology real, word_freq_1999 real, word_freq_parts real, word_freq_pm real, word_freq_direct real, word_freq_cs real, word_freq_meeting real, word_freq_original real, word_freq_project real, word_freq_re real, word_freq_edu real, word_freq_table real, word_freq_conference real, char_freq_semicolon real, char_freq_leftParen real, char_freq_leftBracket real, char_freq_exclamation real, char_freq_dollar real, char_freq_pound real, capital_run_length_average real, capital_run_length_longest real, capital_run_length_total real, spam integer) WITH (CLUSTERED COLUMNSTORE INDEX, DISTRIBUTION = ROUND_ROBIN);  
GO
```

Copy data with bcp:

```
bcp spam in spambaseHeaders.data -q -c -t ',' -S <server-name>.database.windows.net -d <database-name> -U <username> -P <password> -F 1 -r "\r\n"
```

NOTE

The line endings in the downloaded file are Windows-style, but bcp expects UNIX-style, so we need to tell bcp that with the -r flag.

And query with sqlcmd:

```
select top 10 spam, char_freq_dollar from spam;  
GO
```

You could also query with Squirrel SQL. Follow similar steps for PostgreSQL, using the Microsoft MSSQL Server JDBC Driver, which can be found in [/usr/share/java/jdbcdrivers/sqljdbc42.jar](#).

Next steps

For an overview of topics that walk you through the tasks that comprise the Data Science process in Azure, see [Team Data Science Process](#).

For a description of other end-to-end walkthroughs that demonstrate the steps in the Team Data Science Process for specific scenarios, see [Team Data Science Process walkthroughs](#). The walkthroughs also illustrate how to combine cloud and on-premises tools and services into a workflow or pipeline to create an intelligent application.

Create and share an Azure Machine Learning workspace

1/17/2017 • 2 min to read • [Edit on GitHub](#)

This menu links to topics that describe how to set up the various data science environments used by the Cortana Analytics Process (CAPS).

To use Azure Machine Learning Studio, you need to have a Machine Learning workspace. This workspace contains the tools you need to create, manage, and publish experiments.

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

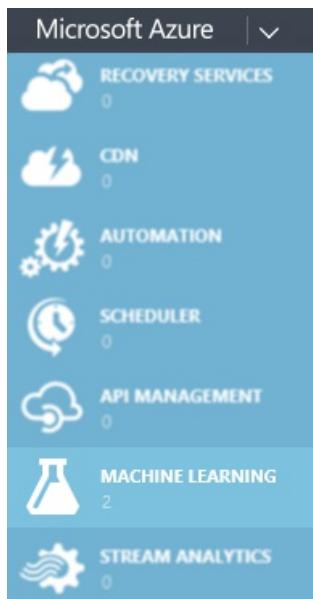
To create a workspace

1. Sign-in to the [Microsoft Azure classic portal](#).

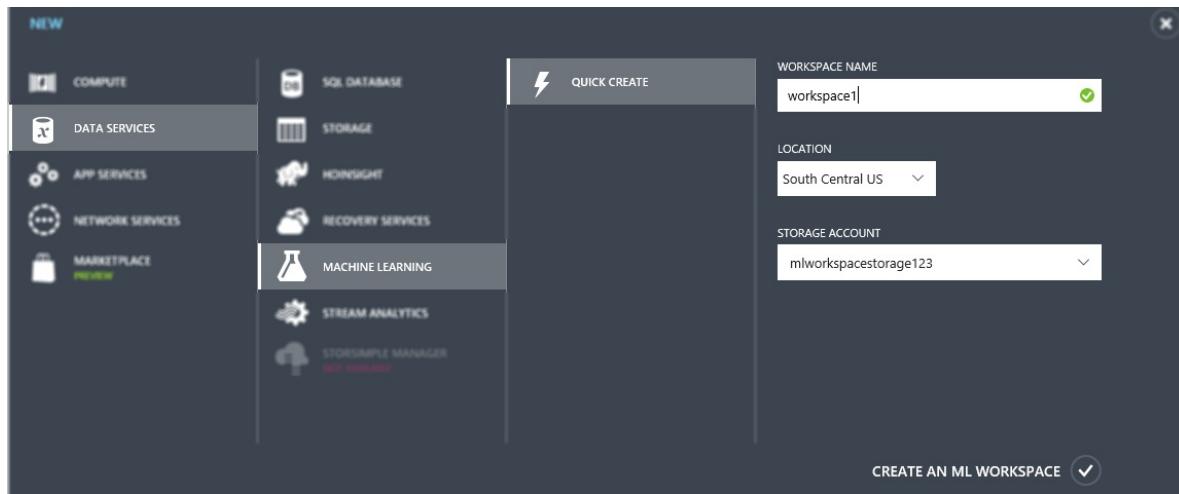
NOTE

To sign-in, you need to be an Azure subscription administrator. Being the owner of a Machine Learning workspace will not give you access to the [Microsoft Azure classic portal](#). See [Privileges of Azure subscription administrator and workspace owner](#) for more details.

1. In the Microsoft Azure services panel, click **MACHINE LEARNING**.



2. Click **+NEW** at the bottom of the window.
3. Click **DATA SERVICES**, then **MACHINE LEARNING**, then **QUICK CREATE**.



4. Enter a **WORKSPACE NAME** for your workspace.
5. Specify the Azure **LOCATION**, then enter an existing Azure **STORAGE ACCOUNT** or select **Create a new storage account** to create a new one.
6. Click **CREATE AN ML WORKSPACE**.

After your Machine Learning workspace is created, you will see it listed on the **machine learning** page.

Sharing an Azure Machine Learning workspace

Once a Machine Learning workspace is created, you can invite users to your workspace and share access to your workspace and all of its experiments. We support two roles of users:

- **User** - A workspace user can create, open, modify and delete datasets, experiments and web services in the workspace.
- **Owner** - An owner can invite, remove, and list users with access to the workspace, in addition to what a user can do. He/she also have access to Notebooks.

To share a workspace

1. Sign-in to [Machine Learning Studio](#)
2. In the Machine Learning Studio panel, click **SETTINGS**
3. Click **USERS**
4. Click **INVITE MORE USERS**

NAME	AUTHORIZATION TOKENS	USERS	
NAME	EMAIL	ROLE	STATUS
Ahmet Gyger	@hotmail.ch	User	Active
ahmet.gyger	@LIVE.COM	Owner	Active

5. Enter one or more email address. The user just need a valid Microsoft account (e.g., name@outlook.com) or an organizational account (from Azure Active Directory).
6. Click the check button.

Each user you added will receive an email with instruction to log-in to the shared workspace.

For information about managing your workspace, see [Manage an Azure Machine Learning workspace](#). If you encounter a problem creating your workspace, see [Troubleshooting guide: Create and connect to an Machine Learning workspace](#).

Privileges of Azure subscription administrator and of workspace owner

Below is a table clarifying the difference between an Azure subscription administrator and a workspace owner.

ACTIONS	AZURE SUBSCRIPTION ADMINISTRATOR	WORKSPACE OWNER
Access Microsoft Azure classic portal	Yes	No
Create a new workspace	Yes	No
Delete a workspace	Yes	No
Add endpoint to a web service	Yes	No
Delete endpoint from a web service	Yes	No
Change concurrency for a web service	Yes	No
Access Machine Learning Studio	No *	Yes

NOTE

- An Azure subscription administrator is automatically added to the workspace he/she creates as workspace Owner. However, simply being an Azure subscription administrator doesn't grant him/her access to any workspace under that subscription.

Manage an Azure Machine Learning workspace

1/17/2017 • 4 min to read • [Edit on GitHub](#)

NOTE

The procedures in this article are relevant to Azure Machine Learning Classic Web services. For information on managing Web services in the Machine Learning Web Services portal, see [Manage a Web service using the Azure Machine Learning Web Services portal](#).

Using the Azure classic portal, you can manage your Machine Learning workspaces to:

- Monitor how the workspace is being used
- Configure the workspace to allow or deny access
- Manage Web services created in the workspace
- Delete the workspace

NOTE

[Try Azure Machine Learning for free](#)

No credit card or Azure subscription needed. [Get started now >](#)

In addition, the dashboard tab provides an overview of your workspace usage and a quick glance of your workspace information.

TIP

In Azure Machine Learning Studio, on the **WEB SERVICES** tab, you can add, update, or delete a Machine Learning Web service.

To manage a workspace:

1. Sign in to the [Azure classic portal](#) using your Microsoft Azure account - use the account that's associated with the Azure subscription.
2. In the Microsoft Azure services panel, click **MACHINE LEARNING**.
3. Click the workspace you want to manage.

The workspace page has three tabs:

- **DASHBOARD** - Allows you to view workspace usage and information
- **CONFIGURE** - Allows you to manage access to the workspace
- **WEB SERVICES** - Allows you to manage Web services that have been published from this workspace

To monitor how the workspace is being used

Click the **DASHBOARD** tab.

From the dashboard, you can view overall usage of your workspace and get a quick glance of workspace information.

- The **COMPUTE** chart shows the compute resources being used by the workspace. You can change the view to display relative or absolute values, and you can change the timeframe displayed in the chart.
- **Usage overview** displays Azure storage being used by the workspace.
- **Quick glance** provides a summary of workspace information and useful links.

NOTE

The **Sign-in to ML Studio** link opens Machine Learning Studio using the Microsoft Account you are currently signed into.

The Microsoft Account you used to sign in to the Azure classic portal to create a workspace does not automatically have permission to open that workspace. To open a workspace, you must be signed in to the Microsoft Account that was defined as the owner of the workspace, or you need to receive an invitation from the owner to join the workspace.

To grant or suspend access for users

Click the **CONFIGURE** tab.

From the configuration tab you can:

- Suspend access to the Machine Learning workspace by clicking DENY. Users will no longer be able to open the workspace in Machine Learning Studio. To restore access, click ALLOW.

To manage additional accounts who have access to the workspace in Machine Learning Studio, click **Sign-in to ML Studio** in the **DASHBOARD** tab (see the preceding note regarding **Sign-in to ML Studio**). This opens the workspace in Machine Learning Studio. From here, click the **SETTINGS** tab and then **USERS**. You can click **INVITE MORE USERS** to give users access to the workspace, or select a user and click **REMOVE**.

To manage web services in this workspace

Click the **WEB SERVICES** tab.

This displays a list of web services published from this workspace. To manage a web service, click the name in the list to open the Web service page.

A Web service may have one or more endpoints defined.

- You can define more endpoints in addition to the "Default" endpoint. To add the endpoint, click **Manage Endpoints** at the bottom of the dashboard to open the Azure Machine Learning Web Services portal.
- To delete an endpoint (you cannot delete the "Default" endpoint), click the check box at the beginning of the endpoint row, and click **DELETE**. This removes the endpoint from the Web service.

NOTE

If an application is using the web service endpoint when the endpoint is deleted, the application will receive an error the next time it tries to access the service.

Click the name of a Web service endpoint to open it.

From the dashboard, you can view overall usage of your Web service over a period of time. You can select the period to view from the Period dropdown menu in the upper right of the usage charts. The dashboard shows the following information:

- **Requests Over Time** displays a step graph of the number of requests over the selected time period. It can help identify if you are experiencing spikes in usage.
- **Request-Response Requests** displays the total number of Request-Response calls the service has received over the selected time period and how many of them failed.

- **Average Request-Response Compute Time** displays an average of the time needed to execute the received requests.
- **Batch Requests** displays the total number of Batch Requests the service has received over the selected time period and how many of them failed.
- **Average Job Latency** displays an average of the time needed to execute the received requests.
- **Errors** displays the aggregate number of errors that have occurred on calls to the web service.
- **Services Costs** displays the charges for the billing plan associated with the service.

From the Configure page, you can update the following properties:

- **Description** allows you to enter a description for the Web service. Description is a required field.
- **Logging** allows you to enable or disable error logging on the endpoint. For more information on Logging, see [Enable logging for Machine Learning web services](#).
- **Enable Sample data** allows you to provide sample data that you can use to test the Request-Response service. If you created the web service in Machine Learning Studio, the sample data is taken from the data you used to train your model. If you created the service programmatically, the data is taken from the example data you provided as part of the JSON package.

Troubleshooting guide: Create and connect to an Machine Learning workspace

1/17/2017 • 1 min to read • [Edit on GitHub](#)

This guide provides solutions for some frequently encountered challenges when you are setting up Azure Machine Learning workspaces.

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

Workspace owner

To open a workspace in Machine Learning Studio, you must be signed in to the Microsoft Account you used to create the workspace, or you need to receive an invitation from the owner to join the workspace. From the Azure portal you can manage the workspace, which includes the ability to configure access.

For more information on managing a workspace, see [Manage an Azure Machine Learning workspace](#).

Allowed regions

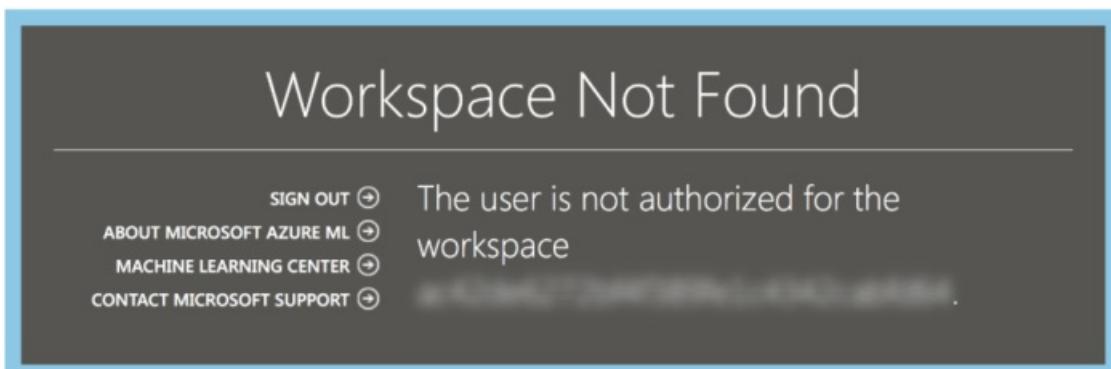
Machine Learning is currently available in a limited number of regions. If your subscription does not include one of these regions, you may see the error message, "You have no subscriptions in the allowed regions."

To request that a region be added to your subscription, create a new Microsoft support request from the Azure portal, choose **Billing** as the problem type, and follow the prompts to submit your request.

Storage account

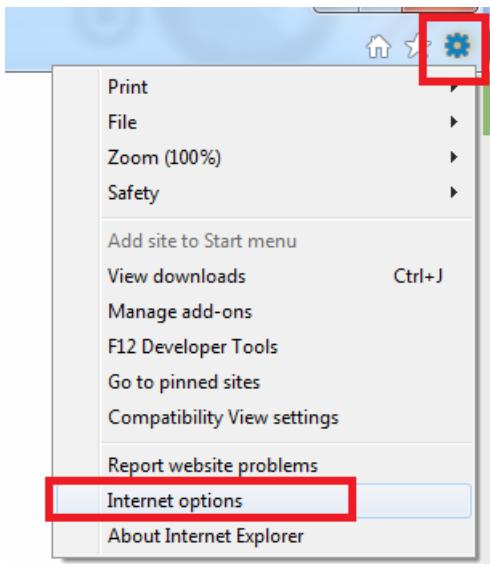
The Machine Learning service needs a storage account to store data. You can use an existing storage account, or you can create a new storage account when you create the new Machine Learning workspace (if you have quota to create a new storage account).

After the new Machine Learning workspace is created, you can sign in to Machine Learning Studio by using the Microsoft account you used to create the workspace. If you encounter the error message, "Workspace Not Found" (similar to the following screenshot), please use the following steps to delete your browser cookies.

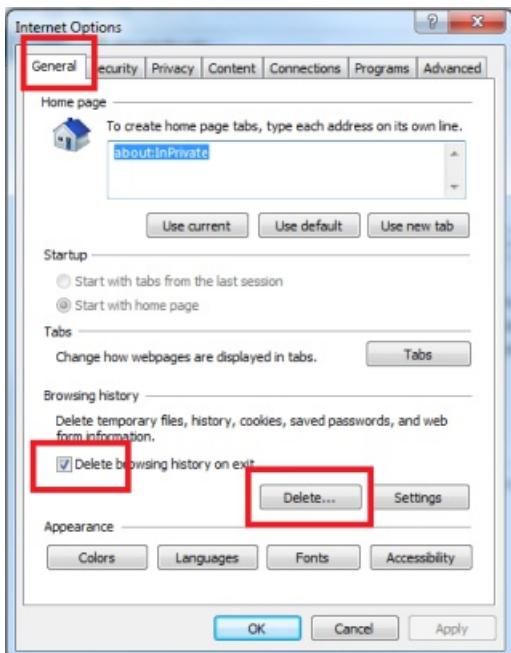


To delete browser cookies

1. If you use Internet Explorer, click the **Tools** button in the upper-right corner and select **Internet options**.



1. Under the **General** tab, click **Delete...**



1. In the **Delete Browsing History** dialog box, make sure **Cookies and website data** is selected, and click **Delete**.



After the cookies are deleted, restart the browser and then go to the [Microsoft Azure Machine Learning](#) page. When you are prompted for a user name and password, enter the same Microsoft account you used to create the workspace.

Comments

Our goal is to make the Machine Learning experience as seamless as possible. Please post any comments and issues at the [Azure Machine Learning forum](#) to help us serve you better.

Deploy Machine Learning Workspace Using Azure Resource Manager

1/17/2017 • 3 min to read • [Edit on GitHub](#)

Introduction

Using an Azure Resource Manager deployment template saves you time by giving you a scalable way to deploy interconnected components with a validation and retry mechanism. To set up Azure Machine Learning Workspaces, for example, you need to first configure an Azure storage account and then deploy your workspace. Imagine doing this manually for hundreds of workspaces. An easier alternative is to use an Azure Resource Manager template to deploy an Azure Machine Learning Workspace and all its dependencies. This article takes you through this process step-by-step. For a great overview of Azure Resource Manager, see [Azure Resource Manager overview](#).

Step-by-step: create a Machine Learning Workspace

We will create an Azure resource group, then deploy a new Azure storage account and a new Azure Machine Learning Workspace using a Resource Manager template. Once the deployment is complete, we will print out important information about the workspaces that were created (the primary key, the workspaceID, and the URL to the workspace).

Create an Azure Resource Manager template

A Machine Learning Workspace requires an Azure storage account to store the dataset linked to it. The following template uses the name of the resource group to generate the storage account name and the workspace name. It also uses the storage account name as a property when creating the workspace.

```
{
  "contentVersion": "1.0.0.0",
  "$schema": "http://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "variables": {
    "namePrefix": "[resourceGroup().name]",
    "location": "[resourceGroup().location]",
    "mlVersion": "2016-04-01",
    "stgVersion": "2015-06-15",
    "storageAccountName": "[concat(variables('namePrefix'),'stg')]",
    "mlWorkspaceName": "[concat(variables('namePrefix'),'mlwk')]",
    "mlResourceId": "[resourceId('Microsoft.MachineLearning/workspaces', variables('mlWorkspaceName'))]",
    "stgResourceId": "[resourceId('Microsoft.Storage/storageAccounts', variables('storageAccountName'))]",
    "storageAccountType": "Standard_LRS"
  },
  "resources": [
    {
      "apiVersion": "[variables('stgVersion')]",
      "name": "[variables('storageAccountName')]",
      "type": "Microsoft.Storage/storageAccounts",
      "location": "[variables('location')]",
      "properties": {
        "accountType": "[variables('storageAccountType')]"
      }
    },
    {
      "apiVersion": "[variables('mlVersion')]",
      "type": "Microsoft.MachineLearning/workspaces",
      "name": "[variables('mlWorkspaceName')]",
      "location": "[variables('location')]",
      "dependsOn": "[variables('stgResourceId')]",
      "properties": {
        "UserStorageAccountId": "[variables('stgResourceId')]"
      }
    }
  ],
  "outputs": {
    "mlWorkspaceObject": {"type": "object", "value": "[reference(variables('mlResourceId'), variables('mlVersion'))]"},
    "mlWorkspaceToken": {"type": "string", "value": "[listWorkspaceKeys(variables('mlResourceId'), variables('mlVersion')).primaryToken]"},
    "mlWorkspaceWorkspaceID": {"type": "string", "value": "[reference(variables('mlResourceId'), variables('mlVersion')).WorkspaceId]"},
    "mlWorkspaceWorkspaceLink": {"type": "string", "value": "[concat('https://studio.azureml.net/Home/ViewWorkspace/', reference(variables('mlResourceId'), variables('mlVersion')).WorkspaceId)]"}
  }
}
```

Save this template as mlworkspace.json file under c:\temp.

Deploy the resource group, based on the template

- Open PowerShell
- Install modules for Azure Resource Manager and Azure Service Management

```
# Install the Azure Resource Manager modules from the PowerShell Gallery (press "A")
Install-Module AzureRM -Scope CurrentUser

# Install the Azure Service Management modules from the PowerShell Gallery (press "A")
Install-Module Azure -Scope CurrentUser
```

These steps download and install the modules necessary to complete the remaining steps. This only needs to be done once in the environment where you are executing the PowerShell commands.

- Authenticate to Azure

```
# Authenticate (enter your credentials in the pop-up window)
Add-AzureRmAccount
```

This step needs to be repeated for each session. Once authenticated, your subscription information should be displayed.

Environment	:	AzureCloud
Account	:	
TenantId	:	[REDACTED]
SubscriptionId	:	[REDACTED]
SubscriptionName	:	Windows Azure MSDN - Visual Studio Ultimate
CurrentStorageAccount	:	

Now that we have access to Azure, we can create the resource group.

- Create a resource group

```
$rg = New-AzureRmResourceGroup -Name "uniquenamerequired523" -Location "South Central US"  
$rg
```

Verify that the resource group is correctly provisioned. **ProvisioningState** should be "Succeeded." The resource group name is used by the template to generate the storage account name. The storage account name must be between 3 and 24 characters in length and use numbers and lower-case letters only.

```
ResourceGroupName : uniquenamerequired543  
Location : southcentralus  
ProvisioningState : Succeeded  
Tags :  
ResourceId : /subscriptions/[REDACTED]/resourceGroups/uniquenamerequired543
```

- Using the resource group deployment, deploy a new Machine Learning Workspace.

```
# Create a Resource Group, TemplateFile is the location of the JSON template.  
$rgd = New-AzureRmResourceGroupDeployment -Name "demo" -TemplateFile "C:\temp\mlworkspace.json" -ResourceGroupName  
$rg.ResourceGroupName
```

Once the deployment is completed, it is straightforward to access properties of the workspace you deployed. For example, you can access the Primary Key Token.

```
# Access Azure ML Workspace Token after its deployment.  
$rgd.Outputs.mlWorkspaceToken.Value
```

Another way to retrieve tokens of existing workspace is to use the `Invoke-AzureRmResourceAction` command. For example, you can list the primary and secondary tokens of all workspaces.

```
# List the primary and secondary tokens of all workspaces  
Get-AzureRmResource |? { $_.ResourceType -Like "*MachineLearning/workspaces*" } |% { Invoke-AzureRmResourceAction -ResourceId  
$_._ResourceId -Action listworkspacekeys -Force }
```

After the workspace is provisioned, you can also automate many Azure Machine Learning Studio tasks using the [PowerShell Module for Azure Machine Learning](#).

Next Steps

- Learn more about [authoring Azure Resource Manager Templates](#).
- Have a look at the [Azure Quickstart Templates Repository](#).
- Watch this video about [Azure Resource Manager](#).

Multi-Geo Help documentation

1/17/2017 • 1 min to read • [Edit on GitHub](#)

This article describes how you can create a workspace and publish a web service in different Azure regions. The [Azure Products by Region page](#) lists regions where Azure Machine Learning is available.

Create a workspace

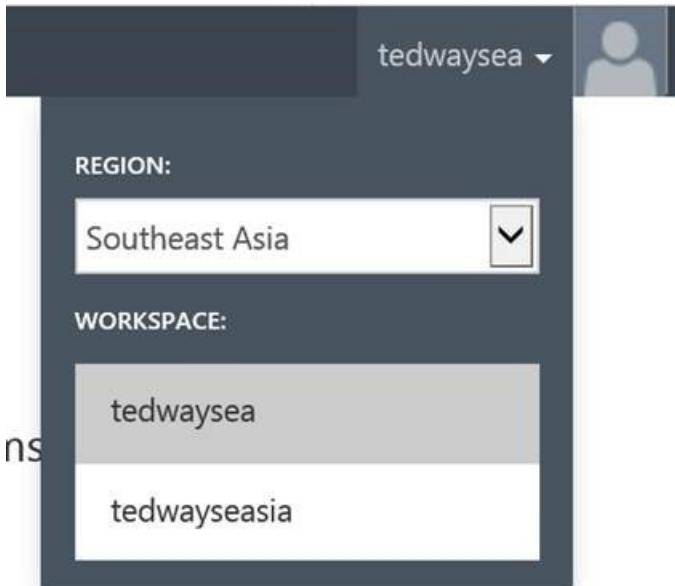
1. Sign in to the Azure Classic Portal.
2. Click **+NEW > DATA SERVICES > MACHINE LEARNING > QUICK CREATE**. Under **LOCATION** select another region, such as **Southeast Asia**.

The screenshot shows the Microsoft Azure Classic Portal interface. On the left, there's a sidebar with categories like Compute, Data Services (selected), App Services, Network Services, Marketplace, and others. In the main area, under 'DATA SERVICES', 'MACHINE LEARNING' is selected. A 'QUICK CREATE' dialog box is open, prompting for 'WORKSPACE NAME' (set to 'tedwayseas'), 'WORKSPACE OWNER' (set to 'tedway@microsoft.com'), 'LOCATION' (set to 'Southeast Asia'), and 'STORAGE ACCOUNT' (set to 'Create a new storage account'). Below these fields is a 'CREATE AN ML WORKSPACE' button with a checkmark icon. At the top right of the dialog, there are 'Subscriptions', 'Region', and a user email 'tedway@microsoft.com'.

3. Select the workspace, and then click **Sign-in to ML Studio**.

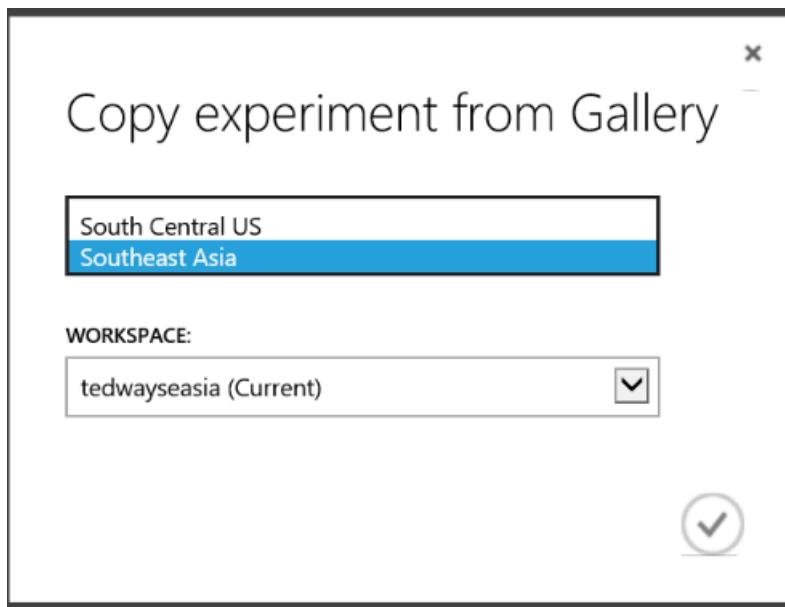
The screenshot shows the Microsoft Azure ML workspace dashboard for 'tedwayseasia'. The left sidebar lists various workspace names. The main dashboard features a large 'Your ML Workspace has been created!' message with a 'Skip Quick Start the next time I visit' checkbox. Below this, three numbered steps are listed: 1. Access your Workspace (with a 'Sign-in to ML Studio' link), 2. Manage Access (with a 'Grant or suspend access for users here' link), and 3. Manage Web Services (with a 'Start here to manage Web Services' link). At the bottom, there are buttons for '+ NEW', 'MANAGE KEYS', 'OPEN IN STUDIO', and 'DELETE'. The top right corner shows 'Subscriptions', 'Region', and the user email 'tedway@microsoft.com'.

4. You now have a workspace in another region that you may use just like any other workspace. To switch among your workspaces, look to the upper right of your screen. Click the dropdown, select the region, and then select the workspace. Everything is local to the workspace region. For example, all of your web services created from a workspace will be in the same region the workspace is located in.



Open an experiment from Gallery

If you open an experiment from Gallery, you can also select which region you want to copy the experiment to.



Web service management

To programmatically manage web services, such as for retraining, use the region-specific address:

- <https://asiasoutheast.management.azureml.net>
- <https://europewest.management.azureml.net>

Things to note

1. You can only copy experiments between workspaces that belong to the same region this way. If you need to copy experiment across workspaces in different regions, you can use the [PowerShell](#) commandlet [Copy-AmlExperiment](#) to accomplish that. Another workaround is to publish the experiment into Gallery in unlisted mode, then open it in the workspace from the other region.

2. The region selector will only show workspaces for one region at a time.
3. A free workspace or Guest Access (anonymous) workspace will be created and hosted in South Central U.S.
4. Web services deployed from a workspace in Southeast Asia will also be hosted in Southeast Asia.

More information

Ask a question on the [Azure Machine Learning forum](#).

How to identify scenarios and plan for advanced analytics data processing

1/17/2017 • 4 min to read • [Edit on GitHub](#)

What resources should you plan to include when setting up an environment to do advanced analytics processing on a dataset? This article suggests a series of questions to ask that will help identify the tasks and resources relevant your scenario. The order of high-level steps for predictive analytics is outlined in [What is the Team Data Science Process \(TDSP\)?](#). Each of these steps will require specific resources for the tasks relevant to your particular scenario. The key questions to identify your scenario concern data logistics, characteristics, the quality of the datasets, and the tools and languages you prefer to do the analysis.

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

Logistic questions: data locations and movement

The logistic questions concern the location of the **data source**, the **target destination** in Azure, and requirements for moving the data, including the schedule, amount and resources involved. The data may need to be moved several times during the analytics process. A common scenario is to move local data into some form of storage on Azure and then into Machine Learning Studio.

1. **What is your data source?** Is it local or in the cloud? For example:

- The data is publicly available at an HTTP address.
- The data resides in a local/network file location.
- The data is in a SQL Server database.
- The data is stored in an Azure storage container

2. **What is the Azure destination?** Where does it need to be for processing or modeling? For example:

- Azure Blob Storage
- SQL Azure databases
- SQL Server on Azure VM
- HDInsight (Hadoop on Azure) or Hive tables
- Azure Machine Learning
- Mountable Azure virtual hard disks.

3. **How are you going to move the data?** The procedures and resources available to ingest or load data into a variety of different storage and processing environments are outlined in the following topics.

- [Load data into storage environments for analytics](#)
- [Import your training data into Azure Machine Learning Studio from various data sources](#).

4. **Does the data need to be moved on a regular schedule or modified during migration?** Consider using Azure Data Factory (ADF) when data needs to be continually migrated, particularly if a hybrid scenario that accesses both on-premise and cloud resources is involved, or when the data is transacted or needs to be modified or have business logic added to it in the course of being migrated. For further information, see [Move data from an on-premise SQL server to SQL Azure with Azure Data Factory](#)

5. **How much of the data is to be moved to Azure?** Datasets that are very large may exceed the storage capacity of certain environments. For an example, see the discussion of size limits for Machine Learning Studio in the next section. In such cases a sample of the data may be used during the analysis. For details of how to down-sample a dataset in various Azure environments, see [Sample data in the Team Data Science Process](#).

Data characteristics questions: type, format and size

These questions are key to planning your storage and processing environments, each of which are appropriate to various types of data and each of which have certain restrictions.

1. What are the data types? For Example:

- Numerical
- Categorical
- Strings
- Binary

2. How is your data formatted? For Example:

- Comma-separated (CSV) or tab-separated (TSV) flat files
- Compressed or uncompressed
- Azure blobs
- Hadoop Hive tables
- SQL Server tables

3. How large is your data?

- Small: Less than 2GB
- Medium: Greater than 2GB and less than 10GB
- Large: Greater than 10GB

Take the Azure Machine Learning Studio environment for example:

- For a list of the data formats and types supported by Azure Machine Learning Studio, see [Data formats and data types supported](#) section.
- For information on data limitations for Azure Machine Learning Studio, see the **How large can the data set be for my modules?** section of [Importing and exporting data for Machine Learning](#)

For information on the limitations of other Azure services used in the analytics process, see [Azure Subscription and Service Limits, Quotas, and Constraints](#).

Data quality questions: exploration and pre-processing

1. **What do you know about your data?** Explore data when you need to gain an understand its basic characteristics. What patterns or trends it exhibits, what outliers it has or how many values are missing. This step is important for determining the extent of pre-processing needed, for formulating hypotheses that could suggest the most appropriate features or type of analysis, and for formulating plans for additional data collection. Calculating descriptive statistics and plotting visualizations are useful techniques for data inspection. For details of how to explore a dataset in various Azure environments, see [Explore data in the Team Data Science Process](#).
2. **Does the data require pre-processing or cleaning?** Pre-processing and cleaning data are important tasks that typically must be conducted before dataset can be used effectively for machine learning. Raw data is often noisy and unreliable, and may be missing values. Using such data for modeling can produce misleading results. For a description, see [Tasks to prepare data for enhanced machine learning](#).

Tools and languages questions

There are lots of options here depending on what languages and development environments or tools you need or are most comfortable using.

1. What languages do you prefer to use for analysis?

- R
- Python
- SQL

2. What tools should you use for data analysis?

- [Microsoft Azure Powershell](#) - a script language used to administer your Azure resources in a script language.
- [Azure Machine Learning Studio](#)
- [Revolution Analytics](#)
- [RStudio](#)
- [Python Tools for Visual Studio](#)
- [Anaconda](#)
- [Jupyter notebooks](#)
- [Microsoft Power BI](#)

Identify your advanced analytics scenario

Once you have answered the questions in the previous section, you are ready to determine which scenario best fits your case. The sample scenarios are outlined in [Scenarios for advanced analytics in Azure Machine Learning](#).

Scenarios for advanced analytics in Azure Machine Learning

1/17/2017 • 9 min to read • [Edit on GitHub](#)

This article outlines the variety of sample data sources and target scenarios that can be handled by the [Team Data Science Process \(TDSP\)](#). The TDSP provides a systematic approach for teams to collaborate on building intelligent applications. The scenarios presented here illustrate options available in the data processing workflow that depend on the data characteristics, source locations, and target repositories in Azure.

The **decision tree** for selecting the sample scenarios that is appropriate for your data and objective is presented in the last section.

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

Each of the following sections presents a sample scenario. For each scenario, a possible data science or advanced analytics flow and supporting Azure resources are listed.

NOTE

For all of the following scenarios, you need to:

- [Create a storage account](#)
- [Create an Azure Machine Learning workspace](#)

Scenario #1: Small to medium tabular dataset in a local files



Additional Azure resources: None

1. Sign in to the [Azure Machine Learning Studio](#).
2. Upload a dataset.
3. Build an Azure Machine Learning experiment flow starting with uploaded dataset(s).

Scenario #2: Small to medium dataset of local files that require processing



Additional Azure resources: Azure Virtual Machine (IPython Notebook server)

1. Create an Azure Virtual Machine running IPython Notebook.
2. Upload data to an Azure storage container.
3. Pre-process and clean data in IPython Notebook, accessing data from Azure storage container.
4. Transform data to cleaned, tabular form.
5. Save transformed data in Azure blobs.
6. Sign in to the [Azure Machine Learning Studio](#).
7. Read the data from Azure blobs using the [Import Data](#) module.
8. Build an Azure Machine Learning experiment flow starting with ingested dataset(s).

Scenario #3: Large dataset of local files, targeting Azure Blobs



Additional Azure resources: Azure Virtual Machine (IPython Notebook server)

1. Create an Azure Virtual Machine running IPython Notebook.
2. Upload data to an Azure storage container.
3. Pre-process and clean data in IPython Notebook, accessing data from Azure blobs.
4. Transform data to cleaned, tabular form, if needed.
5. Explore data, and create features as needed.
6. Extract a small-to-medium data sample.
7. Save the sampled data in Azure blobs.
8. Sign in to the [Azure Machine Learning Studio](#).
9. Read the data from Azure blobs using the [Import Data](#) module.
10. Build Azure Machine Learning experiment flow starting with ingested dataset(s).

Scenario #4: Small to medium dataset of local files, targeting SQL Server in an Azure Virtual Machine



Additional Azure resources: Azure Virtual Machine (SQL Server / IPython Notebook server)

1. Create an Azure Virtual Machine running SQL Server + IPython Notebook.
2. Upload data to an Azure storage container.
3. Pre-process and clean data in Azure storage container using IPython Notebook.
4. Transform data to cleaned, tabular form, if needed.
5. Save data to VM-local files (IPython Notebook is running on VM, local drives refer to VM drives).
6. Load data to SQL Server database running on an Azure VM.

Option #1: Using SQL Server Management Studio.

- Login to SQL Server VM
- Run SQL Server Management Studio.
- Create database and target tables.
- Use one of the bulk import methods to load the data from VM-local files.

Option #2: Using IPython Notebook – not advisable for medium and larger datasets

- Use ODBC connection string to access SQL Server on VM.
- Create database and target tables.
- Use one of the bulk import methods to load the data from VM-local files.

7. Explore data, create features as needed. Note that the features do not need to be materialized in the database tables. Only note the necessary query to create them.
8. Decide on a data sample size, if needed and/or desired.
9. Sign in to the [Azure Machine Learning Studio](#).
10. Read the data directly from the SQL Server using the [Import Data](#) module. Paste the necessary query which extracts fields, creates features, and samples data if needed directly in the [Import Data](#) query.
11. Build Azure Machine Learning experiment flow starting with ingested dataset(s).

Scenario #5: Large dataset in a local files, target SQL Server in Azure VM



Additional Azure resources: Azure Virtual Machine (SQL Server / IPython Notebook server)

1. Create an Azure Virtual Machine running SQL Server and IPython Notebook server.
2. Upload data to an Azure storage container.
3. (Optional) Pre-process and clean data.
 - a. Pre-process and clean data in IPython Notebook, accessing data from Azure blobs.
 - b. Transform data to cleaned, tabular form, if needed.
 - c. Save data to VM-local files (IPython Notebook is running on VM, local drives refer to VM drives).
4. Load data to SQL Server database running on an Azure VM.
 - a. Login to SQL Server VM.
 - b. If data not saved already, download data files from Azure storage container to local-VM folder.
 - c. Run SQL Server Management Studio.
 - d. Create database and target tables.
 - e. Use one of the bulk import methods to load the data.

f. If table joins are required, create indexes to expedite joins.

NOTE

For faster loading of large data sizes, it is recommended that you create partitioned tables and bulk import the data in parallel. For more information, see [Parallel Data Import to SQL Partitioned Tables](#).

5. Explore data, create features as needed. Note that the features do not need to be materialized in the database tables. Only note the necessary query to create them.
6. Decide on a data sample size, if needed and/or desired.
7. Sign in to the [Azure Machine Learning Studio](#).
8. Read the data directly from the SQL Server using the [Import Data](#) module. Paste the necessary query which extracts fields, creates features, and samples data if needed directly in the [Import Data](#) query.
9. Simple Azure Machine Learning experiment flow starting with uploaded dataset

Scenario #6: Large dataset in a SQL Server database on-prem, targeting SQL Server in an Azure Virtual Machine



Additional Azure resources: Azure Virtual Machine (SQL Server / IPython Notebook server)

1. Create an Azure Virtual Machine running SQL Server and IPython Notebook server.
2. Use one of the data export methods to export the data from SQL Server to dump files.

NOTE

If you decide to move all data from the on-prem database, an alternate (faster) method to move the full database to the SQL Server instance in Azure. Skip the steps to export data, create database, and load/import data to the target database and follow the alternate method.

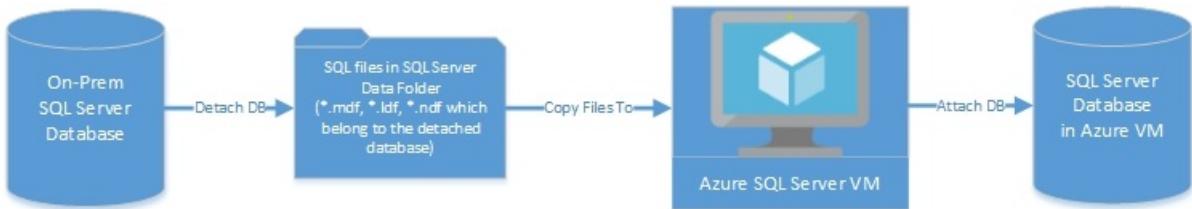
3. Upload dump files to Azure storage container.
4. Load the data to a SQL Server database running on an Azure Virtual Machine.
 - a. Login to the SQL Server VM.
 - b. Download data files from an Azure storage container to the local-VM folder.
 - c. Run SQL Server Management Studio.
 - d. Create database and target tables.
 - e. Use one of the bulk import methods to load the data.
- f. If table joins are required, create indexes to expedite joins.

NOTE

For faster loading of large data sizes, create partitioned tables and to bulk import the data in parallel. For more information, see [Parallel Data Import to SQL Partitioned Tables](#).

5. Explore data, create features as needed. Note that the features do not need to be materialized in the database tables. Only note the necessary query to create them.
6. Decide on a data sample size, if needed and/or desired.
7. Sign in to the [Azure Machine Learning Studio](#).
8. Read the data directly from the SQL Server using the [Import Data](#) module. Paste the necessary query which extracts fields, creates features, and samples data if needed directly in the [Import Data](#) query.
9. Simple Azure Machine Learning experiment flow starting with uploaded dataset.

Alternate method to copy a full database from an on-premises SQL Server to Azure SQL Database



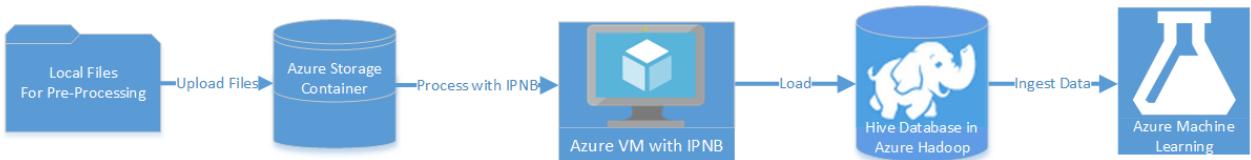
Additional Azure resources: Azure Virtual Machine (SQL Server / IPython Notebook server)

To replicate the entire SQL Server database in your SQL Server VM, you should copy a database from one location/server to another, assuming that the database can be taken temporarily offline. You do this in the SQL Server Management Studio Object Explorer, or using the equivalent Transact-SQL commands.

1. Detach the database at the source location. For more information, see [Detach a database](#).
2. In Windows Explorer or Windows Command Prompt window, copy the detached database file or files and log file or files to the target location on the SQL Server VM in Azure.
3. Attach the copied files to the target SQL Server instance. For more information, see [Attach a Database](#).

[Move a Database Using Detach and Attach \(Transact-SQL\)](#)

Scenario #7: Big data in local files, target Hive database in Azure HDInsight Hadoop clusters



Additional Azure resources: Azure HDInsight Hadoop Cluster and Azure Virtual Machine (IPython Notebook server)

1. Create an Azure Virtual Machine running IPython Notebook server.
2. Create an Azure HDInsight Hadoop cluster.
3. (Optional) Pre-process and clean data.
 - a. Pre-process and clean data in IPython Notebook, accessing data from Azure blobs.
 - b. Transform data to cleaned, tabular form, if needed.
 - c. Save data to VM-local files (IPython Notebook is running on VM, local drives refer to VM drives).
4. Upload data to the default container of the Hadoop cluster selected in the step 2.
5. Load data to Hive database in Azure HDInsight Hadoop cluster.
 - a. Log in to the head node of the Hadoop cluster

- b. Open the Hadoop Command Line.
- c. Enter the Hive root directory by command `cd %hive_home%\bin` in Hadoop Command Line.

- d. Run the Hive queries to create database and tables, and load data from blob storage to Hive tables.

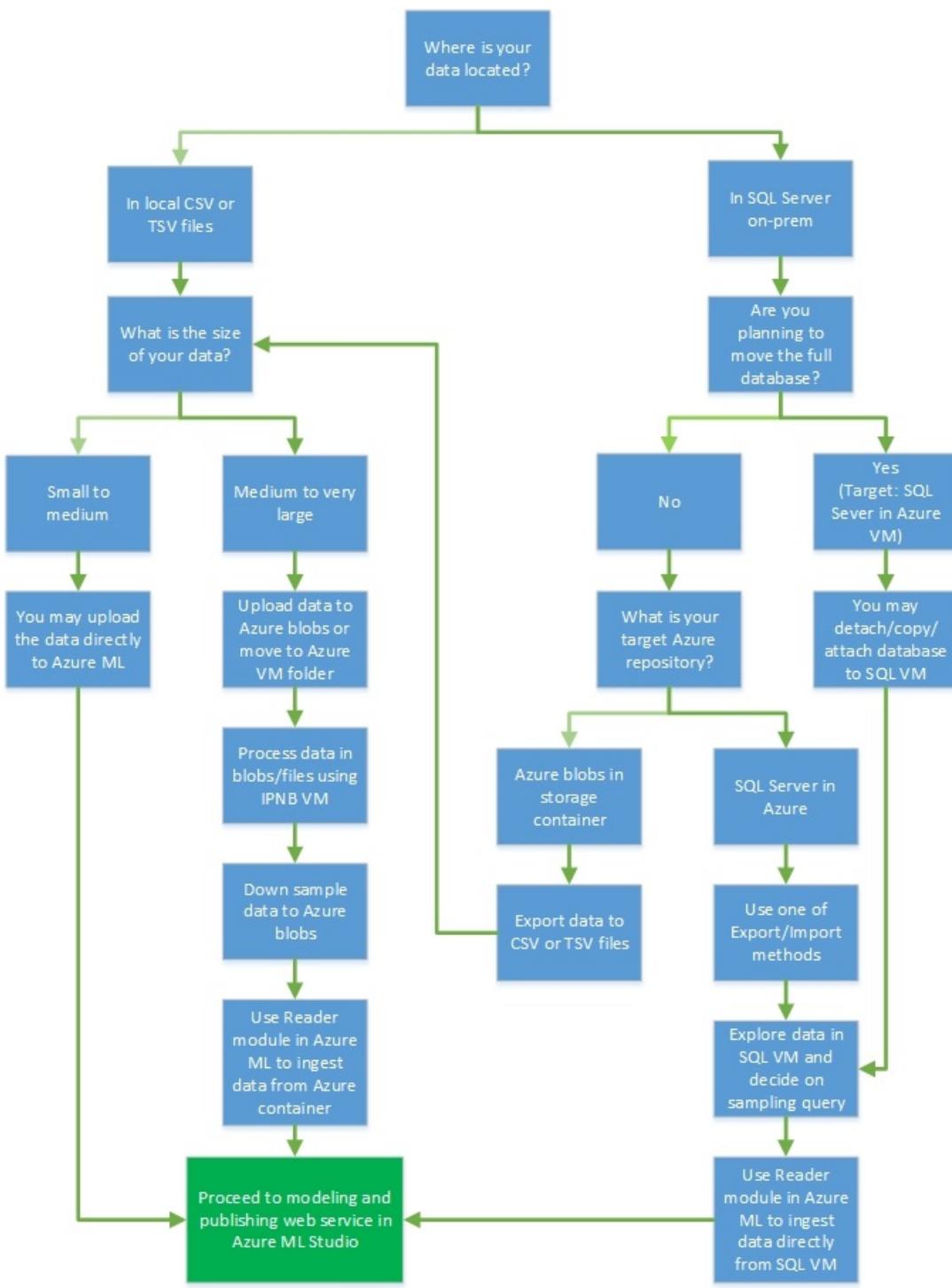
NOTE

If the data is big, users can create the Hive table with partitions. Then, users can use a `for` loop in the Hadoop Command Line on the head node to load data into the Hive table partitioned by partition.

- 6. Explore data and create features as needed in Hadoop Command Line. Note that the features do not need to be materialized in the database tables. Only note the necessary query to create them.
 - a. Log in to the head node of the Hadoop cluster
 - b. Open the Hadoop Command Line.
 - c. Enter the Hive root directory by command `cd %hive_home%\bin` in Hadoop Command Line.
 - d. Run the Hive queries in Hadoop Command Line on the head node of the Hadoop cluster to explore the data and create features as needed.
- 7. If needed and/or desired, sample the data to fit in Azure Machine Learning Studio.
- 8. Sign in to the [Azure Machine Learning Studio](#).
- 9. Read the data directly from the [Hive Queries](#) using the [Import Data](#) module. Paste the necessary query which extracts fields, creates features, and samples data if needed directly in the [Import Data](#) query.
- 10. Simple Azure Machine Learning experiment flow starting with uploaded dataset.

Decision tree for scenario selection

The following diagram summarizes the scenarios described above and the Advanced Analytics Process and Technology choices made that take you to each of the itemized scenarios. Note that data processing, exploration, feature engineering, and sampling may take place in one or more method/environment -- at the source, intermediate, and/or target environments – and may proceed iteratively as needed. The diagram only serves as an illustration of some of possible flows and does not provide an exhaustive enumeration.



Advanced Analytics in action Examples

For end-to-end Azure Machine Learning walkthroughs that employ the Advanced Analytics Process and Technology using public datasets, see:

- Team Data Science Process in action: using SQL Server.
- Team Data Science Process in action: using HDInsight Hadoop clusters.

Load data into storage environments for analytics

1/17/2017 • 1 min to read • [Edit on GitHub](#)

The Team Data Science Process requires that data be ingested or loaded into a variety of different storage environments to be processed or analyzed in the most appropriate way in each stage of the process. Data destinations commonly used for processing include Azure Blob Storage, SQL Azure databases, SQL Server on Azure VM, HDInsight (Hadoop), and Azure Machine Learning.

This **menu** links to topics that describe how to ingest data into these target environments where the data is stored and processed.

Technical and business needs, as well as the initial location, format and size of your data will determine the target environments into which the data needs to be ingested to achieve the goals of your analysis. It is not uncommon for a scenario to require data to be moved between several environments to achieve the variety of tasks required to construct a predictive model. This sequence of tasks can include, for example, data exploration, pre-processing, cleaning, down-sampling, and model training.

Move data to and from Azure Blob Storage

1/17/2017 • 1 min to read • [Edit on GitHub](#)

This menu links to technologies you can use to move data to and from Azure Blob storage:

Which method is best for you depends on your scenario. The [Scenarios for advanced analytics in Azure Machine Learning](#) article helps you determine the resources you need for a variety of data science workflows used in the advanced analytics process.

NOTE

For a complete introduction to Azure blob storage, refer to [Azure Blob Basics](#) and to [Azure Blob Service](#).

As an alternative, you can use [Azure Data Factory](#) to:

- create and schedule a pipeline that downloads data from Azure blob storage,
- pass it to a published Azure Machine Learning web service,
- receive the predictive analytics results, and
- upload the results to storage.

For more information, see [Create predictive pipelines using Azure Data Factory and Azure Machine Learning](#).

Prerequisites

This document assumes that you have an Azure subscription, a storage account, and the corresponding storage key for that account. Before uploading/downloading data, you must know your Azure storage account name and account key.

- To set up an Azure subscription, see [Free one-month trial](#).
- For instructions on creating a storage account and for getting account and key information, see [About Azure storage accounts](#).

Move data to and from Azure Blob Storage using Azure Storage Explorer

1/17/2017 • 2 min to read • [Edit on GitHub](#)

Azure Storage Explorer is a free tool from Microsoft that allows you to work with Azure Storage data on Windows, macOS, and Linux. This topic describes how to use it to upload and download data from Azure blob storage. The tool can be downloaded from [Microsoft Azure Storage Explorer](#).

This menu links to technologies you can use to move data to and from Azure Blob storage:

NOTE

If you are using VM that was set up with the scripts provided by [Data Science Virtual machines in Azure](#), then Azure Storage Explorer is already installed on the VM.

NOTE

For a complete introduction to Azure blob storage, refer to [Azure Blob Basics](#) and [Azure Blob Service](#).

Prerequisites

This document assumes that you have an Azure subscription, a storage account, and the corresponding storage key for that account. Before uploading/downloading data, you must know your Azure storage account name and account key.

- To set up an Azure subscription, see [Free one-month trial](#).
- For instructions on creating a storage account and for getting account and key information, see [About Azure storage accounts](#). Make a note the access key for your storage account as you need this key to connect to the account with the Azure Storage Explorer tool.
- The Azure Storage Explorer tool can be downloaded from [Microsoft Azure Storage Explorer](#). Accept the defaults during install.

Use Azure Storage Explorer

The following steps document how to upload/download data using Azure Storage Explorer.

1. Launch Microsoft Azure Storage Explorer.
2. To bring up the **Sign in to your account...** wizard, select **Azure account settings** icon, then **Add an account** and enter you credentials.

Microsoft Azure Storage Explorer

Edit Help

Microsoft Azure



Show resources from these subscriptions:



Microsoft

@microsoft.com

Remove

All subscriptions:

- Subscription 1
- Subscription 2

Add an account...

Apply

Cancel

Upload Download Open Copy URL

← → ↑ container1

Name Last Modified

3. To bring up the **Connect to Azure Storage** wizard, select the **Connect to Azure storage** icon.

Microsoft Azure Storage Explorer

Edit Help

Microsoft Azure



Search for resources

Upload Download Open

← → ↑ container1

Name Last Modified

4. Enter the access key from your Azure storage account on the **Connect to Azure Storage** wizard and then **Next**.

Connect to Azure Storage

Enter a connection string, Shared Access Signature (SAS) URI, or an account key.

Back

Next

Connect

Cancel

5. Enter storage account name in the **Account name** box and then select **Next**.

Attach External Storage

Enter information to connect to the Microsoft Azure storage account

Account name:

Enter a storage account name

Account key:

Storage endpoints domain:

- Microsoft Azure Default
- Microsoft Azure China
- Other (specify below)

core.windows.net

- Use HTTP (Not recommended)

[Online privacy statement](#)

Back

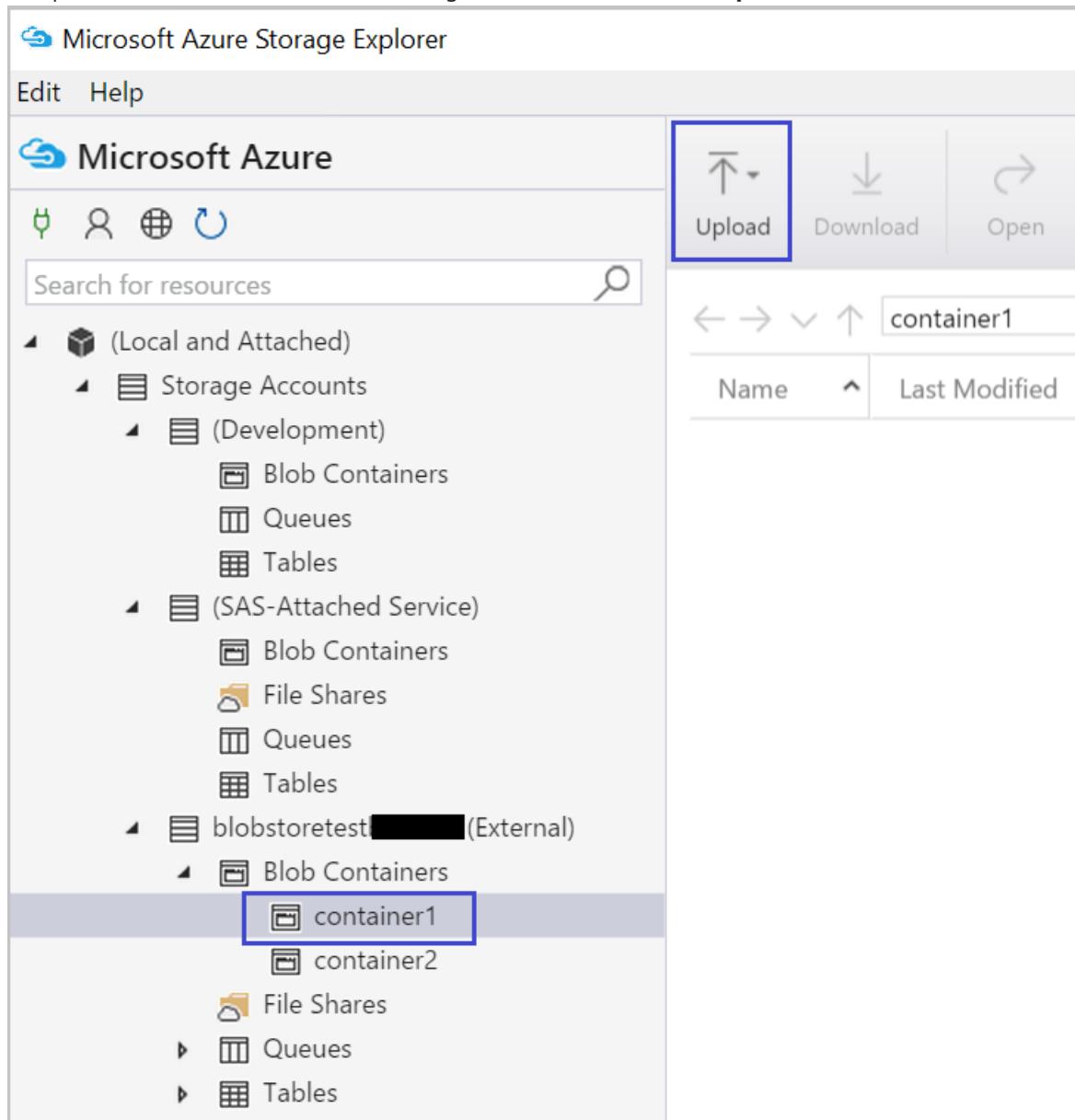
Next

Connect

Cancel

6. The storage account added should now be listed. To create a blob container in a storage account, right-click the **Blob Containers** node in that account, select **Create Blob Container**, and enter a name.

7. To upload data to a container, select the target container and click the **Upload** button.



8. Click on the ... to the right of the **Files** box, select one or multiple files to upload from the file system and click **Upload** to begin uploading the files.

Upload files

Files

No files selected ...

Blob type

Block Blob ▼

Upload .vhdx files as page blobs (recommended)

Upload to folder (optional)

Upload Cancel

9. To download data, selecting the blob in the corresponding container to download and click **Download**.

The screenshot shows the Microsoft Azure Storage Explorer interface. On the left, there is a navigation pane with sections for Local and Attached storage accounts, SAS-Attached Service, and an External account named blobstoretestbradsev. Under blobstoretestbradsev, there are Blob Containers, File Shares, Queues, and Tables. The 'container1' blob container is selected and highlighted with a blue box. On the right, the main workspace displays the contents of 'container1'. A file named 'trip_data_1.csv.zip' is listed, with its details shown: Name: trip_data_1.csv.zip, Last Modified: Wed, 31 Aug 2016 14:31:13 GMT. The 'Download' button in the toolbar above the list is also highlighted with a blue box.

Move data to and from Azure Blob Storage using AzCopy

1/17/2017 • 2 min to read • [Edit on GitHub](#)

AzCopy is a command-line utility designed for uploading, downloading, and copying data to and from Microsoft Azure blob, file, and table storage.

For instructions on installing AzCopy and additional information on using it with the Azure platform, see [Getting Started with the AzCopy Command-Line Utility](#).

This menu links to technologies you can use to move data to and from Azure Blob storage:

NOTE

If you are using VM that was set up with the scripts provided by [Data Science Virtual machines in Azure](#), then AzCopy is already installed on the VM.

NOTE

For a complete introduction to Azure blob storage, refer to [Azure Blob Basics](#) and to [Azure Blob Service](#).

Prerequisites

This document assumes that you have an Azure subscription, a storage account and the corresponding storage key for that account. Before uploading/downloading data, you must know your Azure storage account name and account key.

- To set up an Azure subscription, see [Free one-month trial](#).
- For instructions on creating a storage account and for getting account and key information, see [About Azure storage accounts](#).

Run AzCopy commands

To run AzCopy commands, open a command window and navigate to the AzCopy installation directory on your computer, where the AzCopy.exe executable is located.

The basic syntax for AzCopy commands is:

```
AzCopy /Source:<source> /Dest:<destination> [Options]
```

NOTE

You can add the AzCopy installation location to your system path and then run the commands from any directory. By default, AzCopy is installed to %ProgramFiles(x86)%\Microsoft SDKs\Azure\AzCopy or %ProgramFiles%\Microsoft SDKs\Azure\AzCopy.

Upload files to an Azure blob

To upload a file, use the following command:

```
# Upload from local file system
AzCopy /Source:<your_local_directory> /Dest: https://<your_account_name>.blob.core.windows.net/<your_container_name> /DestKey:<your_account_key> /S
```

Download files from an Azure blob

To download a file from an Azure blob, use the following command:

```
# Downloading blobs to local file system
AzCopy /Source:https://<your_account_name>.blob.core.windows.net/<your_container_name>/<your_sub_directory_at_blob> /Dest:<your_local_directory>/SourceKey:<your_account_key>/Pattern:<file_pattern>/S
```

Transfer blobs between Azure containers

To transfer blobs between Azure containers, use the following command:

```
# Transferring blobs between Azure containers
AzCopy /Source:https://<your_account_name1>.blob.core.windows.net/<your_container_name1>/<your_sub_directory_at_blob1> /Dest:https://<your_account_name2>.blob.core.windows.net/<your_container_name2>/<your_sub_directory_at_blob2> /SourceKey:<your_account_key1> /DestKey:<your_account_key2>/Pattern:<file_pattern>/S

<your_account_name>: your storage account name
<your_account_key>: your storage account key
<your_container_name>: your container name
<your_sub_directory_at_blob>: the sub directory in the container
<your_local_directory>: directory of local file system where files to be uploaded from or the directory of local file system files to be downloaded to
<file_pattern>: pattern of file names to be transferred. The standard wildcards are supported
```

Tips for using AzCopy

TIP

1. When **uploading** files, **/S** uploads files recursively. Without this parameter, files in subdirectories are not uploaded.
2. When **downloading** file, **/S** searches the container recursively until all files in the specified directory and its subdirectories, or all files that match the specified pattern in the given directory and its subdirectories, are downloaded.
3. You cannot specify a **specific blob file** to download using the **/Source** parameter. To download a specific file, specify the blob file name to download using the **/Pattern** parameter. **/S** parameter can be used to have AzCopy look for a file name pattern recursively. Without the pattern parameter, AzCopy downloads all files in that directory.

Move data to and from Azure Blob Storage using Python

1/17/2017 • 2 min to read • [Edit on GitHub](#)

This topic describes how to list, upload, and download blobs using the Python API. With the Python API provided in Azure SDK, you can:

- Create a container
- Upload a blob into a container
- Download blobs
- List the blobs in a container
- Delete a blob

For more information about using the Python API, see [How to Use the Blob Storage Service from Python](#).

This menu links to technologies you can use to move data to and from Azure Blob storage:

NOTE

If you are using VM that was set up with the scripts provided by [Data Science Virtual machines in Azure](#), then AzCopy is already installed on the VM.

NOTE

For a complete introduction to Azure blob storage, refer to [Azure Blob Basics](#) and to [Azure Blob Service](#).

Prerequisites

This document assumes that you have an Azure subscription, a storage account, and the corresponding storage key for that account. Before uploading/downloading data, you must know your Azure storage account name and account key.

- To set up an Azure subscription, see [Free one-month trial](#).
- For instructions on creating a storage account and for getting account and key information, see [About Azure storage accounts](#).

Upload Data to Blob

Add the following snippet near the top of any Python code in which you wish to programmatically access Azure Storage:

```
from azure.storage.blob import BlobService
```

The **BlobService** object lets you work with containers and blobs. The following code creates a BlobService object using the storage account name and account key. Replace account name and account key with your real account and key.

```
blob_service = BlobService(account_name="<your_account_name>", account_key="<your_account_key>")
```

Use the following methods to upload data to a blob:

1. `put_block_blob_from_path` (uploads the contents of a file from the specified path)
2. `put_block_blob_from_file` (uploads the contents from an already opened file/stream)
3. `put_block_blob_from_bytes` (uploads an array of bytes)
4. `put_block_blob_from_text` (uploads the specified text value using the specified encoding)

The following sample code uploads a local file to a container:

```
blob_service.put_block_blob_from_path("<your_container_name>", "<your_blob_name>", "<your_local_file_name>")
```

The following sample code uploads all the files (excluding directories) in a local directory to blob storage:

```
from azure.storage.blob import BlobService
from os import listdir
from os.path import isfile, join

# Set parameters here
ACCOUNT_NAME = "<your_account_name>"
ACCOUNT_KEY = "<your_account_key>"
CONTAINER_NAME = "<your_container_name>"
LOCAL_DIRECT = "<your_local_directory>"

blob_service = BlobService(account_name=ACCOUNT_NAME, account_key=ACCOUNT_KEY)
# find all files in the LOCAL_DIRECT (excluding directory)
local_file_list = [f for f in listdir(LOCAL_DIRECT) if isfile(join(LOCAL_DIRECT, f))]

file_num = len(local_file_list)
for i in range(file_num):
    local_file = join(LOCAL_DIRECT, local_file_list[i])
    blob_name = local_file_list[i]
    try:
        blob_service.put_block_blob_from_path(CONTAINER_NAME, blob_name, local_file)
    except:
        print "something wrong happened when uploading the data %s%%blob_name"
```

Download Data from Blob

Use the following methods to download data from a blob:

1. `get_blob_to_path`
2. `get_blob_to_file`
3. `get_blob_to_bytes`
4. `get_blob_to_text`

These methods that perform the necessary chunking when the size of the data exceeds 64 MB.

The following sample code downloads the contents of a blob in a container to a local file:

```
blob_service.get_blob_to_path("<your_container_name>", "<your_blob_name>", "<your_local_file_name>")
```

The following sample code downloads all blobs from a container. It uses `list_blobs` to get the list of available blobs in the container and downloads them to a local directory.

```
from azure.storage.blob import BlobService
from os.path import join

# Set parameters here
ACCOUNT_NAME = "<your_account_name>"
ACCOUNT_KEY = "<your_account_key>"
CONTAINER_NAME = "<your_container_name>"
LOCAL_DIRECT = "<your_local_directory>"

blob_service = BlobService(account_name=ACCOUNT_NAME, account_key=ACCOUNT_KEY)

# List all blobs and download them one by one
blobs = blob_service.list_blobs(CONTAINER_NAME)
for blob in blobs:
    local_file = join(LOCAL_DIRECT, blob.name)
    try:
        blob_service.get_blob_to_path(CONTAINER_NAME, blob.name, local_file)
    except:
        print "something wrong happened when downloading the data %s" % blob.name
```

Move data to or from Azure Blob Storage using SSIS connectors

1/17/2017 • 3 min to read • [Edit on GitHub](#)

The [SQL Server Integration Services Feature Pack for Azure](#) provides components to connect to Azure, transfer data between Azure and on-premises data sources, and process data stored in Azure.

This menu links to technologies you can use to move data to and from Azure Blob storage:

Once customers have moved on-premises data into the cloud, they can access it from any Azure service to leverage the full power of the suite of Azure technologies. It may be used, for example, in Azure Machine Learning or on an HDInsight cluster.

This is typically be the first step for the [SQL](#) and [HDInsight](#) walkthroughs.

For a discussion of canonical scenarios that use SSIS to accomplish business needs common in hybrid data integration scenarios, see [Doing more with SQL Server Integration Services Feature Pack for Azure](#) blog.

NOTE

For a complete introduction to Azure blob storage, refer to [Azure Blob Basics](#) and to [Azure Blob Service](#).

Prerequisites

To perform the tasks described in this article, you must have an Azure subscription and an Azure storage account set up. You must know your Azure storage account name and account key to upload or download data.

- To set up an **Azure subscription**, see [Free one-month trial](#).
- For instructions on creating a **storage account** and for getting account and key information, see [About Azure storage accounts](#).

To use the **SSIS connectors**, you must download:

- **SQL Server 2014 or 2016 Standard (or above)**: Install includes SQL Server Integration Services.
- **Microsoft SQL Server 2014 or 2016 Integration Services Feature Pack for Azure**: These can be downloaded, respectively, from the [SQL Server 2014 Integration Services](#) and [SQL Server 2016 Integration Services](#) pages.

NOTE

SSIS is installed with SQL Server, but is not included in the Express version. For information on what applications are included in various editions of SQL Server, see [SQL Server Editions](#)

For training materials on SSIS, see [Hands On Training for SSIS](#)

For information on how to get up-and-running using SSIS to build simple extraction, transformation, and load (ETL) packages, see [SSIS Tutorial: Creating a Simple ETL Package](#).

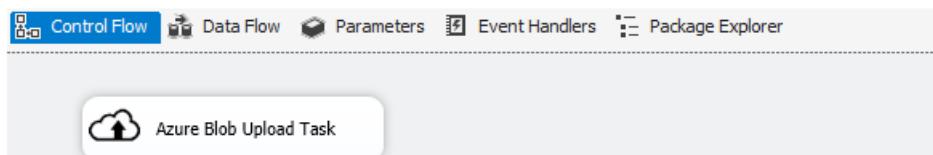
Download NYC Taxi dataset

The example described here use a publicly available dataset -- the [NYC Taxi Trips](#) dataset. The dataset consists of

about 173 million taxi rides in NYC in the year 2013. There are two types of data: trip details data and fare data. As there is a file for each month, we have 24 files in all, each of which is approximately 2GB uncompressed.

Upload data to Azure blob storage

To move data using the SSIS feature pack from on-premises to Azure blob storage, we use an instance of the [Azure Blob Upload Task](#), shown here:



The parameters that the task uses are described here:

FIELD	DESCRIPTION
AzureStorageConnection	Specifies an existing Azure Storage Connection Manager or creates a new one that refers to an Azure storage account that points to where the blob files are hosted.
BlobContainer	Specifies the name of the blob container that hold the uploaded files as blobs.
BlobDirectory	Specifies the blob directory where the uploaded file is stored as a block blob. The blob directory is a virtual hierarchical structure. If the blob already exists, it ia replaced.
LocalDirectory	Specifies the local directory that contains the files to be uploaded.
FileName	Specifies a name filter to select files with the specified name pattern. For example, MySheet*.xls* includes files such as MySheet001.xls and MySheetABC.xlsx
TimeRangeFrom/TimeRangeTo	Specifies a time range filter. Files modified after <i>TimeRangeFrom</i> and before <i>TimeRangeTo</i> are included.

NOTE

The **AzureStorageConnection** credentials need to be correct and the **BlobContainer** must exist before the transfer is attempted.

Download data from Azure blob storage

To download data from Azure blob storage to on-premise storage with SSIS, use an instance of the [Azure Blob Upload Task](#).

More advanced SSIS-Azure scenarios

The SSIS feature pack allows for more complex flows to be handled by packaging tasks together. For example, the blob data could feed directly into an HDInsight cluster, whose output could be downloaded back to a blob and then to on-premise storage. SSIS can run Hive and Pig jobs on an HDInsight cluster using additional SSIS connectors:

- To run a Hive script on an Azure HDInsight cluster with SSIS, use [Azure HDInsight Hive Task](#).
- To run a Pig script on an Azure HDInsight cluster with SSIS, use [Azure HDInsight Pig Task](#).

Move data to SQL Server on an Azure virtual machine

1/17/2017 • 8 min to read • [Edit on GitHub](#)

This topic outlines the options for moving data either from flat files (CSV or TSV formats) or from an on-premise SQL Server to SQL Server on an Azure virtual machine. These tasks for moving data to the cloud are part of the Team Data Science Process.

For a topic that outlines the options for moving data to an Azure SQL Database for Machine Learning, see [Move data to an Azure SQL Database for Azure Machine Learning](#).

The **menu** below links to topics that describe how to ingest data into other target environments where the data can be stored and processed during the Team Data Science Process (TDSP).

The following table summarizes the options for moving data to SQL Server on an Azure virtual machine.

SOURCE	DESTINATION: SQL SERVER ON AZURE VM
Flat File	<ol style="list-style-type: none">1. Command-line bulk copy utility (BCP)2. Bulk Insert SQL Query3. Graphical Built-in Utilities in SQL Server
On-Premises SQL Server	<ol style="list-style-type: none">1. Deploy a SQL Server Database to a Microsoft Azure VM wizard2. Export to a flat File3. SQL Database Migration Wizard4. Database back up and restore

Note that this document assumes that SQL commands are executed from SQL Server Management Studio or Visual Studio Database Explorer.

TIP

As an alternative, you can use [Azure Data Factory](#) to create and schedule a pipeline that will move data to a SQL Server VM on Azure. For more information, see [Copy data with Azure Data Factory \(Copy Activity\)](#).

Prerequisites

This tutorial assumes you have:

- An **Azure subscription**. If you do not have a subscription, you can sign up for a [free trial](#).
- An **Azure storage account**. You will use an Azure storage account for storing the data in this tutorial. If you don't have an Azure storage account, see the [Create a storage account](#) article. After you have created the storage account, you will need to obtain the account key used to access the storage. See [Manage your storage access keys](#).
- Provisioned **SQL Server on an Azure VM**. For instructions, see [Set up an Azure SQL Server virtual machine as an IPython Notebook server for advanced analytics](#).
- Installed and configured **Azure PowerShell** locally. For instructions, see [How to install and configure Azure PowerShell](#).

Moving data from a flat file source to SQL Server on an Azure VM

If your data is in a flat file (arranged in a row/column format), it can be moved to SQL Server VM on Azure via the following methods:

1. [Command-line bulk copy utility \(BCP\)](#)
2. [Bulk Insert SQL Query](#)
3. [Graphical Built-in Utilities in SQL Server \(Import/Export, SSIS\)](#)

Command-line bulk copy utility (BCP)

BCP is a command-line utility installed with SQL Server and is one of the quickest ways to move data. It works across all three SQL Server variants (On-premises SQL Server, SQL Azure and SQL Server VM on Azure).

NOTE

Where should my data be for BCP?

While it is not required, having files containing source data located on the same machine as the target SQL Server allows for faster transfers (network speed vs local disk IO speed). You can move the flat files containing data to the machine where SQL Server is installed using various file copying tools such as [AZCopy](#), [Azure Storage Explorer](#) or windows copy/paste via Remote Desktop Protocol (RDP).

1. Ensure that the database and the tables are created on the target SQL Server database. Here is an example of how to do that using the [Create Database](#) and [Create Table](#) commands:

```
CREATE DATABASE <database_name>

CREATE TABLE <tablename>
(
    <columnname1> <datatype> <constraint>,
    <columnname2> <datatype> <constraint>,
    <columnname3> <datatype> <constraint>
)
```

2. Generate the format file that describes the schema for the table by issuing the following command from the command-line of the machine where bcp is installed.

```
bcp dbname..tablename format nul -c -f exportformatfilename.xml -S servername\sqlinstance -T -t \t\r\n
```

3. Insert the data into the database using the bcp command as follows. This should work from the command-line assuming that the SQL Server is installed on same machine:

```
bcp dbname..tablename in datafilename.tsv -f exportformatfilename.xml -S servername\sqlinstancename -U username -P password -b block_size_to_move_in_single_attempt -t \t\r\n
```

Optimizing BCP Inserts Please refer the following article '[Guidelines for Optimizing Bulk Import](#)' to optimize such inserts.

Parallelizing Inserts for Faster Data Movement

If the data you are moving is large, you can speed things up by simultaneously executing multiple BCP commands in parallel in a PowerShell Script.

NOTE

Big data Ingestion To optimize data loading for large and very large datasets, partition your logical and physical database tables using multiple filegroups and partition tables. For more information about creating and loading data to partition tables, see [Parallel Load SQL Partition Tables](#).

The sample PowerShell script below demonstrate parallel inserts using bcp:

```
$NO_OF_PARALLEL_JOBS=2

Set-ExecutionPolicy RemoteSigned #set execution policy for the script to execute
# Define what each job does
$ScriptBlock = {
    param($partitionnumber)

    #Explicitly using SQL username password
    bcp database..tablename in datafile_path.csv -F 2 -f format_file_path.xml -U username@servername -S tcp:servername -P password -b
    block_size_to_move_in_single_attempt -t "," -r \n -o path_to_outputfile.$partitionnumber.txt

    #Trusted connection w/o username password (if you are using windows auth and are signed in with that credentials)
    #bcp database..tablename in datafile_path.csv -o path_to_outputfile.$partitionnumber.txt -h "TABLOCK" -F 2 -f format_file_path.xml -T -b
    block_size_to_move_in_single_attempt -t "," -r \n
}

# Background processing of all partitions
for ($i=1; $i -le $NO_OF_PARALLEL_JOBS; $i++)
{
    Write-Debug "Submit loading partition # $i"
    Start-Job $ScriptBlock -Arg $i
}

# Wait for it all to complete
While (Get-Job -State "Running")
{
    Start-Sleep 10
    Get-Job
}

# Getting the information back from the jobs
Get-Job | Receive-Job
Set-ExecutionPolicy Restricted #reset the execution policy
```

Bulk Insert SQL Query

[Bulk Insert SQL Query](#) can be used to import data into the database from row/column based files (the supported types are covered in the[Prepare Data for Bulk Export or Import \(SQL Server\)](#) topic).

Here are some sample commands for Bulk Insert are as below:

1. Analyze your data and set any custom options before importing to make sure that the SQL Server database assumes the same format for any special fields such as dates. Here is an example of how to set the date format as year-month-day (if your data contains the date in year-month-day format):

```
SET DATEFORMAT ymd;
```

2. Import data using bulk import statements:

```
BULK INSERT <tablename>
FROM
'<datafilename>'
WITH
(
FirstRow=2,
FIELDTERMINATOR = ',', --this should be column separator in your data
ROWTERMINATOR = '\n' --this should be the row separator in your data
)
```

You can use SQL Server Integrations Services (SSIS) to import data into SQL Server VM on Azure from a flat file. SSIS is available in two studio environments. For details, see [Integration Services \(SSIS\) and Studio Environments](#):

- For details on SQL Server Data Tools, see [Microsoft SQL Server Data Tools](#)
- For details on the Import/Export Wizard, see [SQL Server Import and Export Wizard](#)

Moving Data from on-premises SQL Server to SQL Server on an Azure VM

You can also use the following migration strategies:

1. [Deploy a SQL Server Database to a Microsoft Azure VM wizard](#)
2. [Export to Flat File](#)
3. [SQL Database Migration Wizard](#)
4. [Database back up and restore](#)

We describe each of these below:

Deploy a SQL Server Database to a Microsoft Azure VM wizard

The **Deploy a SQL Server Database to a Microsoft Azure VM wizard** is a simple and recommended way to move data from an on-premises SQL Server instance to SQL Server on an Azure VM. For detailed steps as well as a discussion of other alternatives, see [Migrate a database to SQL Server on an Azure VM](#).

Export to Flat File

Various methods can be used to bulk export data from an On-Premises SQL Server as documented in the [Bulk Import and Export of Data \(SQL Server\)](#) topic. This document will cover the Bulk Copy Program (BCP) as an example. Once data is exported into a flat file, it can be imported to another SQL server using bulk import.

1. Export the data from on-premises SQL Server to a File using the bcp utility as follows

```
bcp dbname..tablename out datafile.tsv -S servername\sqlinstancename -T -t \t -t \n -c
```

2. Create the database and the table on SQL Server VM on Azure using the `create database` and `create table` for the table schema exported in step 1.
3. Create a format file for describing the table schema of the data being exported/imported. Details of the format file are described in [Create a Format File \(SQL Server\)](#).

Format file generation when running BCP from the SQL Server machine

```
bcp dbname..tablename format nul -c -x -f exportformatfilename.xml -S servername\sqlinstance -T -t \t -r \n
```

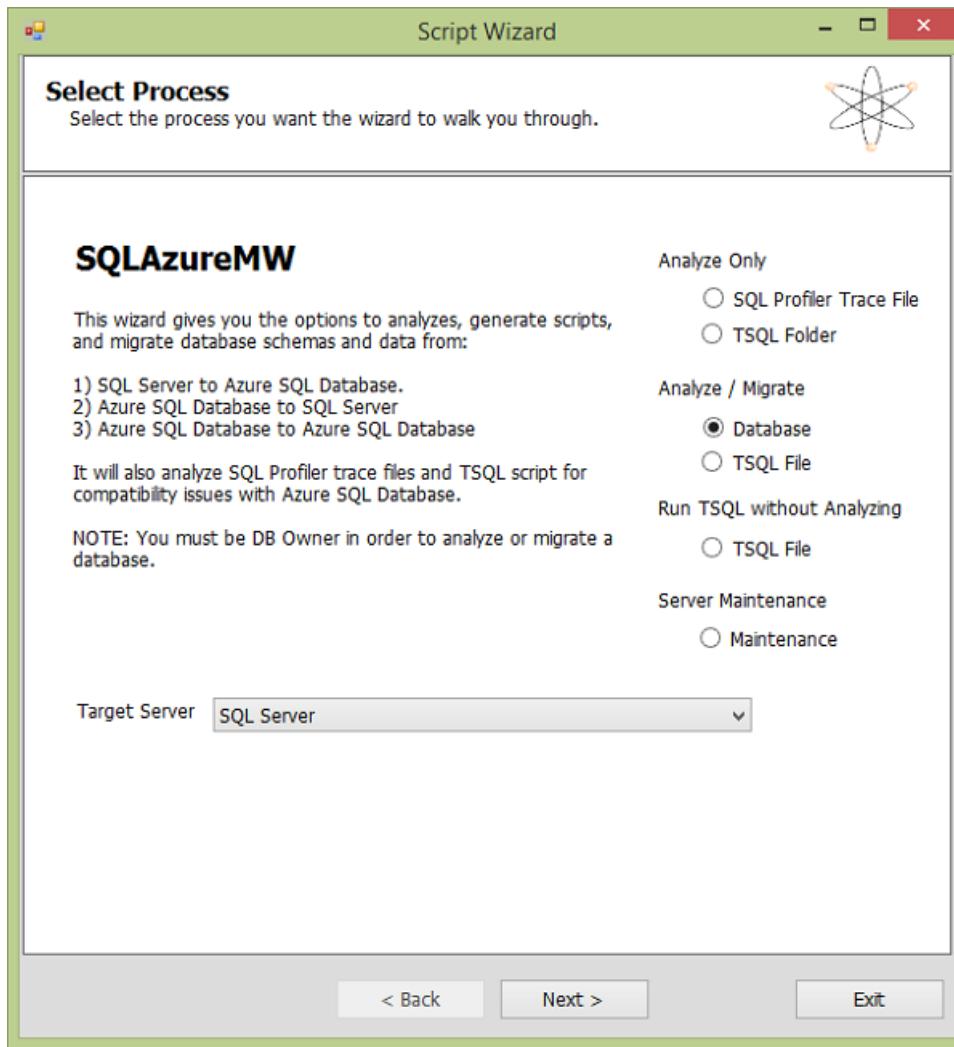
Format file generation when running BCP remotely against a SQL Server

```
bcp dbname..tablename format nul -c -x -f exportformatfilename.xml -U username@servername.database.windows.net -S tcp:servername -P password -t \t -r \n
```

4. Use any of the methods described in section [Moving Data from File Source](#) to move the data in flat files to a SQL Server.

SQL Database Migration Wizard

[SQL Server Database Migration Wizard](#) provides a user-friendly way to move data between two SQL server instances. It allows the user to map the data schema between sources and destination tables, choose column types and various other functionalities. It uses bulk copy (BCP) under the covers. A screenshot of the welcome screen for the SQL Database Migration wizard is shown below.



Database back up and restore

SQL Server supports:

1. [Database back up and restore functionality](#) (both to a local file or bacpac export to blob) and [Data Tier Applications](#) (using bacpac).
2. Ability to directly create SQL Server VMs on Azure with a copied database or copy to an existing SQL Azure database. For more details, see [Use the Copy Database Wizard](#).

A screenshot of the Database back up/restore options from SQL Server Management Studio is shown below.

```

CREATE DATABASE [disprocess]
    ON  PRIMARY 
        ( NAME = N'disprocess', FILENAME = N'C:\Program Files\Microsoft SQL Server\M12\DATA\disprocess.mdf' , SIZE = 10MB , MAXSIZE = 100MB , FILEGROWTH = 10MB )
    LOG ON ( NAME = N'disprocess_log', FILENAME = N'C:\Program Files\Microsoft SQL Server\M12\LOG\disprocess.ldf' , SIZE = 10MB , MAXSIZE = 100MB , FILEGROWTH = 10MB )
    GO
    -- [status_confidence] [varchar](max) NULL,
    -- [status_start] [varchar](max) NULL,
    -- [tempo] [float] NULL,
    -- [time_signature] [float] NULL,
    -- [title] [varchar](max) NULL,
    -- [track_digitalId] [nvarchar](max) NULL,
    -- [year] [float] NULL,
    --PRIMARY KEY CLUSTERED
    --(
    --[track_id] ASC
    --)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_SNAPSHOTLOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY],
    --(IGNORE_DUP_KEY = OFF, ALLOW_SNAPSHOTLOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
    GO

```

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, a database named 'disprocess' is selected. A context menu is open over this database, with the 'Tasks' option highlighted. Under 'Tasks', the 'Script Database As' option is also highlighted. The main pane displays the T-SQL script for creating the 'disprocess' database, including its primary and log file specifications and various options like PAD_INDEX, STATISTICS_NORECOMPUTE, etc.

Resources

Migrate a Database to SQL Server on an Azure VM

SQL Server on Azure Virtual Machines overview

Move data to an Azure SQL Database for Azure Machine Learning

1/17/2017 • 4 min to read • [Edit on GitHub](#)

This topic outlines the options for moving data either from flat files (CSV or TSV formats) or from data stored in an on-premise SQL Server to an Azure SQL database. These tasks for moving data to the cloud are part of the Team Data Science Process.

For a topic that outlines the options for moving data to an on-premise SQL Server for Machine Learning, see [Move data to SQL Server on an Azure virtual machine](#).

The following **menu** links to topics that describe how to ingest data into target environments where the data can be stored and processed during the Team Data Science Process (TDSP).

The following table summarizes the options for moving data to an Azure SQL Database.

SOURCE	DESTINATION: AZURE SQL DATABASE
Flat file (CSV or TSV formatted)	Bulk Insert SQL Query
On-premise SQL Server	<ol style="list-style-type: none">Export to Flat FileSQL Database Migration WizardDatabase back up and restoreAzure Data Factory

Prerequisites

The procedures outlined here require that you have:

- An **Azure subscription**. If you do not have a subscription, you can sign up for a [free trial](#).
- An **Azure storage account**. You use an Azure storage account for storing the data in this tutorial. If you don't have an Azure storage account, see the [Create a storage account](#) article. After you have created the storage account, you need to obtain the account key used to access the storage. See [Manage your storage access keys](#).
- Access to an **Azure SQL Database**. If you must set up an Azure SQL Database, [Getting Started with Microsoft Azure SQL Database](#) provides information on how to provision a new instance of an Azure SQL Database.
- Installed and configured **Azure PowerShell** locally. For instructions, see [How to install and configure Azure PowerShell](#).

Data: The migration processes are demonstrated using the [NYC Taxi dataset](#). The NYC Taxi dataset contains information on trip data and fares and is available on Azure blob storage: [NYC Taxi Data](#). A sample and description of these files are provided in [NYC Taxi Trips Dataset Description](#).

You can either adapt the procedures described here to a set of your own data or follow the steps as described by using the NYC Taxi dataset. To upload the NYC Taxi dataset into your on-premise SQL Server database, follow the procedure outlined in [Bulk Import Data into SQL Server Database](#). These instructions are for a SQL Server on an Azure Virtual Machine, but the procedure for uploading to the on-premise SQL Server is the same.

Moving data from a flat file source to an Azure SQL database

Data in flat files (CSV or TSV formatted) can be moved to an Azure SQL database using a Bulk Insert SQL Query.

Bulk Insert SQL Query

The steps for the procedure using the Bulk Insert SQL Query are similar to those covered in the sections for moving data from a flat file source to SQL Server on an Azure VM. For details, see [Bulk Insert SQL Query](#).

Moving Data from on-premise SQL Server to an Azure SQL database

If the source data is stored in an on-premise SQL Server, there are various possibilities for moving the data to an Azure SQL database:

1. [Export to Flat File](#)
2. [SQL Database Migration Wizard](#)
3. [Database back up and restore](#)
4. [Azure Data Factory](#)

The steps for the first three are very similar to those sections in [Move data to SQL Server on an Azure virtual machine](#) that cover these same procedures. Links to the appropriate sections in that topic are provided in the following instructions.

Export to Flat File

The steps for this exporting to a flat file are similar to those covered in [Export to Flat File](#).

SQL Database Migration Wizard

The steps for using the SQL Database Migration Wizard are similar to those covered in [SQL Database Migration Wizard](#).

Database back up and restore

The steps for using database back up and restore are similar to those covered in [Database back up and restore](#).

Azure Data Factory

The procedure for moving data to an Azure SQL database with Azure Data Factory (ADF) is provided in the topic [Move data from an on-premise SQL server to SQL Azure with Azure Data Factory](#). This topic shows how to move data from an on-premise SQL Server database to an Azure SQL database via Azure Blob Storage using ADF.

Consider using ADF when data needs to be continually migrated in a hybrid scenario that accesses both on-premise and cloud resources, and when the data is transacted or needs to be modified or have business logic added to it when being migrated. ADF allows for the scheduling and monitoring of jobs using simple JSON scripts that manage the movement of data on a periodic basis. ADF also has other capabilities such as support for complex operations.

Create and load data into Hive tables from Azure blob storage

1/17/2017 • 11 min to read • [Edit on GitHub](#)

This topic presents generic Hive queries that create Hive tables and load data from Azure blob storage. Some guidance is also provided on partitioning Hive tables and on using the Optimized Row Columnar (ORC) formatting to improve query performance.

This **menu** links to topics that describe how to ingest data into target environments where the data can be stored and processed during the Team Data Science Process (TDSP).

Prerequisites

This article assumes that you have:

- Created an Azure storage account. If you need instructions, see [About Azure storage accounts](#).
- Provisioned a customized Hadoop cluster with the HDInsight service. If you need instructions, see [Customize Azure HDInsight Hadoop clusters for advanced analytics](#).
- Enabled remote access to the cluster, logged in, and opened the Hadoop Command-Line console. If you need instructions, see [Access the Head Node of Hadoop Cluster](#).

Upload data to Azure blob storage

If you created an Azure virtual machine by following the instructions provided in [Set up an Azure virtual machine for advanced analytics](#), this script file should have been downloaded to the `C:\Users\<user name>\Documents\Data Science Scripts` directory on the virtual machine. These Hive queries only require that you plug in your own data schema and Azure blob storage configuration in the appropriate fields to be ready for submission.

We assume that the data for Hive tables is in an **uncompressed** tabular format, and that the data has been uploaded to the default (or to an additional) container of the storage account used by the Hadoop cluster.

If you want to practice on the **NYC Taxi Trip Data**, you need to:

- **download** the 24 [NYC Taxi Trip Data](#) files (12 Trip files and 12 Fare files),
- **unzip** all files into .csv files, and then
- **upload** them to the default (or appropriate container) of the Azure storage account that was created by the procedure outlined in the [Customize Azure HDInsight Hadoop clusters for Advanced Analytics Process and Technology](#) topic. The process to upload the .csv files to the default container on the storage account can be found on this [page](#).

How to submit Hive queries

Hive queries can be submitted by using:

1. [Submit Hive queries through Hadoop Command Line in headnode of Hadoop cluster](#)
2. [Submit Hive queries with the Hive Editor](#)
3. [Submit Hive queries with Azure PowerShell Commands](#)

Hive queries are SQL-like. If you are familiar with SQL, you may find the [Hive for SQL Users Cheat Sheet](#) useful.

When submitting a Hive query, you can also control the destination of the output from Hive queries, whether it be on the screen or to a local file on the head node or to an Azure blob.

1. Submit Hive queries through Hadoop Command Line in headnode of Hadoop cluster

If the Hive query is complex, submitting it directly in the head node of the Hadoop cluster typically leads to faster turn around than submitting it with a Hive Editor or Azure PowerShell scripts.

Log in to the head node of the Hadoop cluster, open the Hadoop Command Line on the desktop of the head node, and enter command `cd %hive_home%\bin`.

You have three ways to submit Hive queries in the Hadoop Command Line:

- directly
- using .hql files
- with the Hive command console

Submit Hive queries directly in Hadoop Command Line.

You can run command like `hive -e "<your hive query>"` to submit simple Hive queries directly in Hadoop Command Line. Here is an example, where the red box outlines the command that submits the Hive query, and the green box outlines the output from the Hive query.

```
C:\>cd %hive_home%\bin
C:\>hive -e "show tables;"
```

'hive' is not recognized as an internal or external command,
operable program or batch file.

```
C:\>cd %hive_home%\bin
C:\>hive -e "show tables;"
```

Logging initialized using configuration in file:/C:/apps/dist/hive-0.13.0.2.1.10.0-2290/conf/hive-log4j.properties

SLF4J: Class path contains multiple SLF4J bindings.

SLF4J: Found binding in [jar:file:/C:/apps/dist/hadoop-2.4.0.2.1.10.0-2290/share/hadoop/common/lib/slf4j-log4j12-1.7.5.jar!org/slf4j.impl.StaticLoggerBinder.class]

SLF4J: Found binding in [jar:file:/C:/apps/dist/hbase-0.98.0.2.1.10.0-2290-hadoop2/lib/slf4j-log4j12-1.6.4.jar!org/slf4j.impl.StaticLoggerBinder.class]

SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.

SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]

OK

hivesampletable

Time taken: 1.502 seconds, Fetched: 1 row(s)

```
C:\>cd %hive_home%\bin
```

Submit Hive queries in .hql files

When the Hive query is more complicated and has multiple lines, editing queries in command line or Hive command console is not practical. An alternative is to use a text editor in the head node of the Hadoop cluster to save the Hive queries in a .hql file in a local directory of the head node. Then the Hive query in the .hql file can be submitted by using the `-f` argument as follows:

```
hive -f "<path to the .hql file>"
```

```
Hadoop Command Line
```

Directory name, or volume label syntax is incorrect?

```
C:\>cd %hive_home%\bin
C:\>hive -f <path>
```

Logging initialized using configuration in file:/C:/apps/dist/hive-0.13.0.2.1.10.0-2290/conf/hive-log4j.properties

SLF4J: Class path contains multiple SLF4J bindings.

SLF4J: Found binding in [jar:file:/C:/apps/dist/hadoop-2.4.0.2.1.10.0-2290/share/hadoop/common/lib/slf4j-log4j12-1.7.5.jar!org/slf4j.impl.StaticLoggerBinder.class]

SLF4J: Found binding in [jar:file:/C:/apps/dist/hbase-0.98.0.2.1.10.0-2290-hadoop2/lib/slf4j-log4j12-1.6.4.jar!org/slf4j.impl.StaticLoggerBinder.class]

SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.

SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]

OK

```
hive -f <path>
```

hivesampletable

Time taken: 1.502 seconds, Fetched: 1 row(s)

```
C:\>cd %hive_home%\bin
```

hivequeryfile.hql - Notepad

```
select * from hivesampletable limit 10;
```

Suppress progress status screen print of Hive queries

By default, after Hive query is submitted in Hadoop Command Line, the progress of the Map/Reduce job is printed out on screen. To suppress the screen print of the Map/Reduce job progress, you can use an argument `-S` ("S" in upper case) in the command line as follows:

```
hive -S -f "<path to the .hql file>"
```

```
. hive -S -e ""
```

Submit Hive queries in Hive command console.

You can also first enter the Hive command console by running command `hive` in Hadoop Command Line, and then submit Hive queries in Hive command console. Here is an example. In this example, the two red boxes highlight the commands used to enter the Hive command console, and the Hive query submitted in Hive command console, respectively. The green box highlights the output from the Hive query.

```
OK
hivesamplatable
Time taken: 1.502 seconds. Fetched: 1 row(s)

C:\apps\dist\hive-0.13.0.2.1.10.0-2290\bin\hive

Logging initialized using configuration in file:/C:/apps/dist/hive-0.13.0.2.1.10.0-2290/conf/hive-log4j.properties
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/C:/apps/dist/hadoop-2.4.0.2.1.10.0-2290/share/hadoop/common/lib/slf4j-log4j12-1.7.5.jar!org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/C:/apps/dist/hbase-0.98.0.2.1.10.0-2290-hadoop2/lib/slf4j-log4j12-1.6.4.jar!org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.

SLF4J: Actual binding is of type org.slf4j.impl.Log4jLoggerFactory
hive> select * from hivesamplatable limit 3;
+-----+-----+-----+-----+-----+
| id   | name | country| phone | state |
+-----+-----+-----+-----+-----+
| 8    | 18:54:20 | en-US | Android | Samsung SCH-i500 | California
| United States | 13.9204007 | 0 | 0 | Incredibile | Pennsylvania
| 23   | 19:19:44 | en-US | Android | HTC | Incredibile | Pennsylvania
| United States | NULL | 0 | 0 | Incredibile | Pennsylvania
| 23   | 19:19:46 | en-US | Android | HTC | Incredibile | Pennsylvania
| United States | 1.4757422 | 0 | 1 | Incredibile | Pennsylvania
+-----+-----+-----+-----+-----+
Time taken: 2.997 seconds. Fetched: 3 rows(s)
hive> -
```

The previous examples directly output the Hive query results on screen. You can also write the output to a local file on the head node, or to an Azure blob. Then, you can use other tools to further analyze the output of Hive queries.

Output Hive query results to a local file. To output Hive query results to a local directory on the head node, you have to submit the Hive query in the Hadoop Command Line as follows:

```
hive -e "<hive query>" ><local path in the head node>
```

In the following example, the output of Hive query is written into a file `hivequeryoutput.txt` in directory `C:\apps\temp`.

```
Hadoop Command Line
Logging initialized using configuration in file:/C:/apps/dist/hive-0.13.0.2.1.10.0-2290/conf/hive-log4j.properties
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/C:/apps/dist/hadoop-2.4.0.2.1.10.0-2290/share/hadoop/common/lib/slf4j-log4j12-1.7.5.jar!org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/C:/apps/dist/hbase-0.98.0.2.1.10.0-2290-hadoop2/lib/slf4j-log4j12-1.6.4.jar!org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.

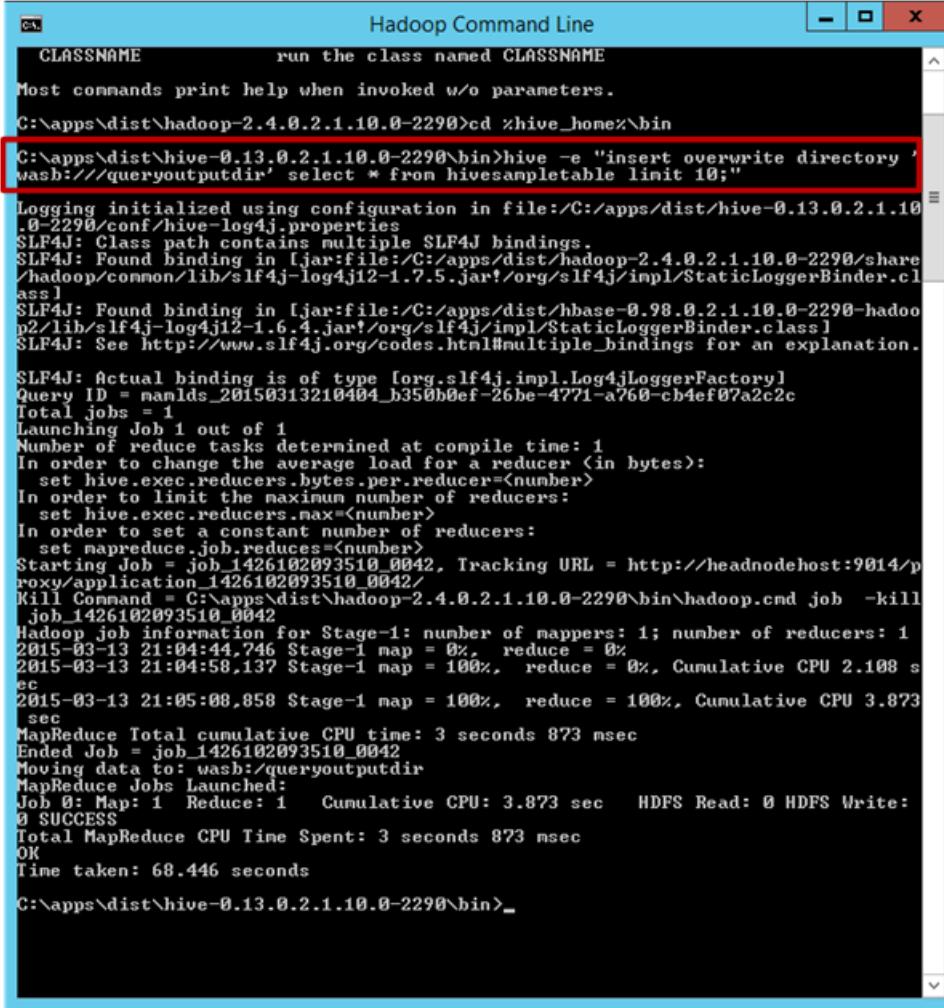
SLF4J: Actual binding is of type org.slf4j.impl.Log4jLoggerFactory
hive> select * from hivesamplatable limit 10;
+-----+-----+-----+-----+-----+
| id   | name | country| phone | state |
+-----+-----+-----+-----+-----+
| 8    | 18:54:20 | en-US | Android | Samsung SCH-i500 | California
| United States | 13.9204007 | 0 | 0 | Incredibile | Pennsylvania
| 23   | 19:19:44 | en-US | Android | HTC | Incredibile | Pennsylvania
| United States | NULL | 0 | 0 | Incredibile | Pennsylvania
| 23   | 19:19:46 | en-US | Android | HTC | Incredibile | Pennsylvania
| United States | 1.4757422 | 0 | 1 | Incredibile | Pennsylvania
| 23   | 19:19:46 | en-US | Android | HTC | Incredibile | Pennsylvania
| United States | 1.4757422 | 0 | 1 | Incredibile | Pennsylvania
| 23   | 19:19:46 | en-US | Android | HTC | Incredibile | Pennsylvania
| United States | 1.4757422 | 0 | 1 | Incredibile | Pennsylvania
+-----+-----+-----+-----+-----+
Time taken: 3.130 seconds. Fetched: 10 rows(s)
hive> -
```

Output Hive query results to an Azure blob

You can also output the Hive query results to an Azure blob, within the default container of the Hadoop cluster. The Hive query for this is as follows:

```
insert overwrite directory wasb://<directory within the default container><select clause from ...>
```

In the following example, the output of Hive query is written to a blob directory `queryoutputdir` within the default container of the Hadoop cluster. Here, you only need to provide the directory name, without the blob name. An error is thrown if you provide both directory and blob names, such as `wasb:///queryoutputdir/queryoutput.txt`.



The screenshot shows a terminal window titled "Hadoop Command Line". The command entered is:

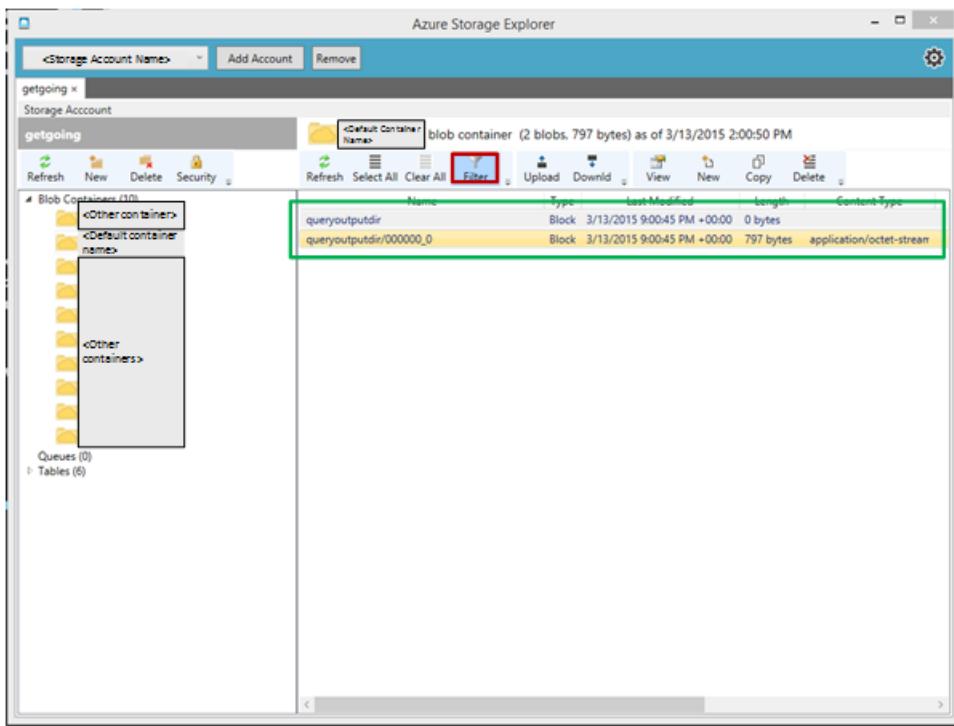
```
C:\apps\dist\hive-0.13.0.2.1.10.0-2290>hive -e "insert overwrite directory 'wasb://queryoutputdir' select * from hivesampletable limit 10;"
```

The output of the command is displayed below the command line. It shows various log messages related to SLF4J binding and a MapReduce job execution. The job starts, runs a single map task, and ends successfully. The total time taken is 68.446 seconds.

```
Logging initialized using configuration in file:/C:/apps/dist/hive-0.13.0.2.1.10.0-2290/conf/hive-log4j.properties
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/C:/apps/dist/hadoop-2.4.0.2.1.10.0-2290/share/hadoop/common/lib/slf4j-log4j12-1.7.5.jar!org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/C:/apps/dist/hbase-0.98.0.2.1.10.0-2290-hadoop2/lib/slf4j-log4j12-1.6.4.jar!org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.

SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Query ID = mamds_20150313210404_b350b0ef-26be-4771-a760-cb4ef07a2c2c
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer <in bytes>
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1426102093510_0042, Tracking URL = http://headnodehost:9014/proxy/application_1426102093510_0042/
Kill Command = C:\apps\dist\hadoop-2.4.0.2.1.10.0-2290\bin\hadoop.cmd job -kill job_1426102093510_0042
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2015-03-13 21:04:44.746 Stage-1 map = 0%,  reduce = 0%
2015-03-13 21:04:58.137 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.108 sec
2015-03-13 21:05:00.858 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 3.873 sec
MapReduce Total cumulative CPU time: 3 seconds 873 msec
Ended Job = job_1426102093510_0042
Moving data to: wasb:/queryoutputdir
MapReduce Jobs Launched:
Job 0: Map: 1 Reduce: 1  Cumulative CPU: 3.873 sec    HDFS Read: 0 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 3 seconds 873 msec
OK
Time taken: 68.446 seconds
C:\apps\dist\hive-0.13.0.2.1.10.0-2290\bin>
```

If you open the default container of the Hadoop cluster using Azure Storage Explorer, you can see the output of the Hive query as shown in the following figure. You can apply the filter (highlighted by red box) to only retrieve the blob with specified letters in names.



2. Submit Hive queries with the Hive Editor

You can also use the Query Console (Hive Editor) by entering a URL of the form `https://<Hadoop cluster name>.azurehdinsight.net/Home/HiveEditor` into a web browser. You must be logged in to see this console and so you need your Hadoop cluster credentials here.

3. Submit Hive queries with Azure PowerShell Commands

You can also use PowerShell to submit Hive queries. For instructions, see [Submit Hive jobs using PowerShell](#).

Create Hive database and tables

The Hive queries are shared in the [Github repository](#) and can be downloaded from there.

Here is the Hive query that creates a Hive table.

```
create database if not exists <database name>;
CREATE EXTERNAL TABLE if not exists <database name>.<table name>
(
  field1 string,
  field2 int,
  field3 float,
  field4 double,
  ...,
  fieldN string
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '<field separator>' lines terminated by '<line separator>'
STORED AS TEXTFILE LOCATION '<storage location>' TBLPROPERTIES("skip.header.line.count"="1");
```

Here are the descriptions of the fields that you need to plug in and other configurations:

- **<database name>**: the name of the database that you want to create. If you just want to use the default database, the query `create database...` can be omitted.
- **<table name>**: the name of the table that you want to create within the specified database. If you want to use the default database, the table can be directly referred by `<table name>` without `<database name>`.
- **<field separator>**: the separator that delimits fields in the data file to be uploaded to the Hive table.
- **<line separator>**: the separator that delimits lines in the data file.
- **<storage location>**: the Azure storage location to save the data of Hive tables. If you do not specify

LOCATION <storage location>, the database and the tables are stored in *hive/warehouse/* directory in the default container of the Hive cluster by default. If you want to specify the storage location, the storage location has to be within the default container for the database and tables. This location has to be referred as location relative to the default container of the cluster in the format of '*wasb:///<directory 1>/*' or '*wasb:///<directory 1>/<directory 2>/*', etc. After the query is executed, the relative directories are created within the default container.

- **TBLPROPERTIES("skip.header.line.count" = "1")**: If the data file has a header line, you have to add this property **at the end** of the *create table* query. Otherwise, the header line is loaded as a record to the table. If the data file does not have a header line, this configuration can be omitted in the query.

Load data to Hive tables

Here is the Hive query that loads data into a Hive table.

```
LOAD DATA INPATH '<path to blob data>' INTO TABLE <database name>.<table name>;
```

- **<path to blob data>**: If the blob file to be uploaded to the Hive table is in the default container of the HDInsight Hadoop cluster, the *<path to blob data>* should be in the format '*wasb:///<directory in this container>/<blob file name>*'. The blob file can also be in an additional container of the HDInsight Hadoop cluster. In this case, *<path to blob data>* should be in the format '*wasb://<container name>@<storage account name>.blob.core.windows.net/<blob file name>*'.

NOTE

The blob data to be uploaded to Hive table has to be in the default or additional container of the storage account for the Hadoop cluster. Otherwise, the *LOAD DATA* query fails complaining that it cannot access the data.

Advanced topics: partitioned table and store Hive data in ORC format

If the data is large, partitioning the table is beneficial for queries that only need to scan a few partitions of the table. For instance, it is reasonable to partition the log data of a web site by dates.

In addition to partitioning Hive tables, it is also beneficial to store the Hive data in the Optimized Row Columnar (ORC) format. For more information on ORC formatting, see [Using ORC files improves performance when Hive is reading, writing, and processing data](#).

Partitioned table

Here is the Hive query that creates a partitioned table and loads data into it.

```
CREATE EXTERNAL TABLE IF NOT EXISTS <database name>.<table name>
  (field1 string,
  ...
  fieldN string
)
PARTITIONED BY (<partitionfieldname> vartype) ROW FORMAT DELIMITED FIELDS TERMINATED BY '<field separator>'
  lines terminated by '<line separator>' TBLPROPERTIES("skip.header.line.count"="1");
LOAD DATA INPATH '<path to the source file>' INTO TABLE <database name>.<partitioned table name>
  PARTITION (<partitionfieldname>=<partitionfieldvalue>);
```

When querying partitioned tables, it is recommended to add the partition condition in the **beginning** of the **where** clause as this improves the efficacy of searching significantly.

```
select
  field1, field2, ..., fieldN
from <database name>.<partitioned table name>
where <partitionfieldname>=<partitionfieldvalue> and ...;
```

Store Hive data in ORC format

You cannot directly load data from blob storage into Hive tables that is stored in the ORC format. Here are the steps that you need to take to load data from Azure blobs to Hive tables stored in ORC format.

Create an external table **STORED AS TEXTFILE** and load data from blob storage to the table.

```
CREATE EXTERNAL TABLE IF NOT EXISTS <database name>.<external textfile table name>
(
  field1 string,
  field2 int,
  ...
  fieldN date
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '<field separator>'
  lines terminated by '<line separator>' STORED AS TEXTFILE
  LOCATION 'wasb:///<directory in Azure blob>' TBLPROPERTIES("skip.header.line.count"="1");

LOAD DATA INPATH '<path to the source file>' INTO TABLE <database name>.<table name>;
```

Create an internal table with the same schema as the external table in step 1, with the same field delimiter, and store the Hive data in the ORC format.

```
CREATE TABLE IF NOT EXISTS <database name>.<ORC table name>
(
  field1 string,
  field2 int,
  ...
  fieldN date
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '<field separator>' STORED AS ORC;
```

Select data from the external table in step 1 and insert into the ORC table

```
INSERT OVERWRITE TABLE <database name>.<ORC table name>
  SELECT * FROM <database name>.<external textfile table name>;
```

NOTE

If the TEXTFILE table *<database name>.<external textfile table name>* has partitions, in STEP 3, the `SELECT * FROM <database name>.<external textfile table name>` command selects the partition variable as a field in the returned data set. Inserting it into the *<database name>.<ORC table name>* fails since *<database name>.<ORC table name>* does not have the partition variable as a field in the table schema. In this case, you need to specifically select the fields to be inserted to *<database name>.<ORC table name>* as follows:

```
INSERT OVERWRITE TABLE <database name>.<ORC table name> PARTITION (<partition variable>=<partition value>)
  SELECT field1, field2, ..., fieldN
  FROM <database name>.<external textfile table name>
  WHERE <partition variable>=<partition value>;
```

It is safe to drop the *<external textfile table name>* when using the following query after all data has been inserted into *<database name>.<ORC table name>*:

```
DROP TABLE IF EXISTS <database name>.<external textfile table name>;
```

After following this procedure, you should have a table with data in the ORC format ready to use.

Move data from an on-premise SQL server to SQL Azure with Azure Data Factory

1/17/2017 • 9 min to read • [Edit on GitHub](#)

This topic shows how to move data from an on-premise SQL Server Database to a SQL Azure Database via Azure Blob Storage using the Azure Data Factory (ADF).

The following **menu** links to topics that describe how to ingest data into target environments where the data can be stored and processed during the Team Data Science Process.

Introduction: What is ADF and when should it be used to migrate data?

Azure Data Factory is a fully managed cloud-based data integration service that orchestrates and automates the movement and transformation of data. The key concept in the ADF model is pipeline. A pipeline is a logical grouping of Activities, each of which defines the actions to perform on the data contained in Datasets. Linked services are used to define the information needed for Data Factory to connect to the data resources.

With ADF, existing data processing services can be composed into data pipelines that are highly available and managed in the cloud. These data pipelines can be scheduled to ingest, prepare, transform, analyze, and publish data, and ADF manages and orchestrates the complex data and processing dependencies. Solutions can be quickly built and deployed in the cloud, connecting a growing number of on-premises and cloud data sources.

Consider using ADF:

- when data needs to be continually migrated in a hybrid scenario that accesses both on-premise and cloud resources
- when the data is transacted or needs to be modified or have business logic added to it when being migrated.

ADF allows for the scheduling and monitoring of jobs using simple JSON scripts that manage the movement of data on a periodic basis. ADF also has other capabilities such as support for complex operations. For more information on ADF, see the documentation at [Azure Data Factory \(ADF\)](#).

The Scenario

We set up an ADF pipeline that composes two data migration activities. Together they move data on a daily basis between an on-premise SQL database and an Azure SQL Database in the cloud. The two activities are:

- copy data from an on-premise SQL Server database to an Azure Blob Storage account
- copy data from the Azure Blob Storage account to an Azure SQL Database.

NOTE

The steps shown here have been adapted from the more detailed tutorial provided by the ADF team: [Move data between on-premises sources and cloud with Data Management Gateway](#) References to the relevant sections of that topic are provided when appropriate.

Prerequisites

This tutorial assumes you have:

- An **Azure subscription**. If you do not have a subscription, you can sign up for a [free trial](#).

- An **Azure storage account**. You use an Azure storage account for storing the data in this tutorial. If you don't have an Azure storage account, see the [Create a storage account](#) article. After you have created the storage account, you need to obtain the account key used to access the storage. See [Manage your storage access keys](#).
- Access to an **Azure SQL Database**. If you must set up an Azure SQL Database, the topic [Getting Started with Microsoft Azure SQL Database](#) provides information on how to provision a new instance of an Azure SQL Database.
- Installed and configured **Azure PowerShell** locally. For instructions, see [How to install and configure Azure PowerShell](#).

NOTE

This procedure uses the [Azure portal](#).

Upload the data to your on-premise SQL Server

We use the [NYC Taxi dataset](#) to demonstrate the migration process. The NYC Taxi dataset is available, as noted in that post, on Azure blob storage [NYC Taxi Data](#). The data has two files, the trip_data.csv file, which contains trip details, and the trip_fare.csv file, which contains details of the fare paid for each trip. A sample and description of these files are provided in [NYC Taxi Trips Dataset Description](#).

You can either adapt the procedure provided here to a set of your own data or follow the steps as described by using the NYC Taxi dataset. To upload the NYC Taxi dataset into your on-premise SQL Server database, follow the procedure outlined in [Bulk Import Data into SQL Server Database](#). These instructions are for a SQL Server on an Azure Virtual Machine, but the procedure for uploading to the on-premise SQL Server is the same.

Create an Azure Data Factory

The instructions for creating a new Azure Data Factory and a resource group in the [Azure portal](#) are provided [Create an Azure Data Factory](#). Name the new ADF instance *adfdsp* and name the resource group created *adfdsp**rg*.

Install and configure up the Data Management Gateway

To enable your pipelines in an Azure data factory to work with an on-premise SQL Server, you need to add it as a Linked Service to the data factory. To create a Linked Service for an on-premise SQL Server, you must:

- download and install Microsoft Data Management Gateway onto the on-premise computer.
- configure the linked service for the on-premises data source to use the gateway.

The Data Management Gateway serializes and deserializes the source and sink data on the computer where it is hosted.

For set-up instructions and details on Data Management Gateway, see [Move data between on-premises sources and cloud with Data Management Gateway](#)

Create linked services to connect to the data resources

A linked service defines the information needed for Azure Data Factory to connect to a data resource. The step-by-step procedure for creating linked services is provided in [Create linked services](#).

We have three resources in this scenario for which linked services are needed.

1. [Linked service for on-premise SQL Server](#)
2. [Linked service for Azure Blob Storage](#)
3. [Linked service for Azure SQL database](#)

Linked service for on-premise SQL Server database

To create the linked service for the on-premise SQL Server:

- click the **Data Store** in the ADF landing page on Azure Classic Portal
- select **SQL** and enter the *username* and *password* credentials for the on-premise SQL Server. You need to enter the servername as a **fully qualified servername backslash instance name (servername\instancename)**. Name the linked service *adfonpremssql*.

Linked service for Blob

To create the linked service for the Azure Blob Storage account:

- click the **Data Store** in the ADF landing page on Azure Classic Portal
- select **Azure Storage Account**
- enter the Azure Blob Storage account key and container name. Name the Linked Service *adfds*.

Linked service for Azure SQL database

To create the linked service for the Azure SQL Database:

- click the **Data Store** in the ADF landing page on Azure Classic Portal
- select **Azure SQL** and enter the *username* and *password* credentials for the Azure SQL Database. The *username* must be specified as *user@servername*.

Define and create tables to specify how to access the datasets

Create tables that specify the structure, location, and availability of the datasets with the following script-based procedures. JSON files are used to define the tables. For more information on the structure of these files, see [Datasets](#).

NOTE

You should execute the `Add-AzureAccount` cmdlet before executing the `New-AzureDataFactoryTable` cmdlet to confirm that the right Azure subscription is selected for the command execution. For documentation of this cmdlet, see [Add-AzureAccount](#).

The JSON-based definitions in the tables use the following names:

- the **table name** in the on-premise SQL server is *nytaxi_data*
- the **container name** in the Azure Blob Storage account is *containername*

Three table definitions are needed for this ADF pipeline:

1. [SQL on-premise Table](#)
2. [Blob Table](#)
3. [SQL Azure Table](#)

NOTE

These procedures use Azure PowerShell to define and create the ADF activities. But these tasks can also be accomplished using the Azure portal. For details, see [Create datasets](#).

SQL on-premise Table

The table definition for the on-premise SQL Server is specified in the following JSON file:

```
{
  "name": "OnPremSQLTable",
  "properties":
  {
    "location":
    {
      "type": "OnPremisesSqlServerTableLocation",
      "tableName": "nyctaxi_data",
      "linkedServiceName": "adfonpremsql"
    },
    "availability":
    {
      "frequency": "Day",
      "interval": 1,
      "waitOnExternal":
      {
        "retryInterval": "00:01:00",
        "retryTimeout": "00:10:00",
        "maximumRetry": 3
      }
    }
  }
}
```

The column names were not included here. You can sub-select on the column names by including them here (for details check the [ADF documentation](#) topic).

Copy the JSON definition of the table into a file called *onpremtabledef.json* file and save it to a known location (here assumed to be C:\temp\onpremtabledef.json). Create the table in ADF with the following Azure PowerShell cmdlet:

```
New-AzureDataFactoryTable -ResourceGroupName ADFdsprg -DataFactoryName ADFdsp –File C:\temp\onpremtabledef.json
```

Blob Table

Definition for the table for the output blob location is in the following (this maps the ingested data from on-premise to Azure blob):

```
{
  "name": "OutputBlobTable",
  "properties":
  {
    "location":
    {
      "type": "AzureBlobLocation",
      "folderPath": "containername",
      "format":
      {
        "type": "TextFormat",
        "columnDelimiter": "\t"
      },
      "linkedServiceName": "adfds"
    },
    "availability":
    {
      "frequency": "Day",
      "interval": 1
    }
  }
}
```

Copy the JSON definition of the table into a file called *bloboutputtabledef.json* file and save it to a known location

(here assumed to be C:\temp\bloboutputtabledef.json). Create the table in ADF with the following Azure PowerShell cmdlet:

```
New-AzureDataFactoryTable -ResourceGroupName adfdsp -DataFactoryName adfdsp -File C:\temp\bloboutputtabledef.json
```

SQL Azure Table

Definition for the table for the SQL Azure output is in the following (this schema maps the data coming from the blob):

```
{  
  "name": "OutputSQLAzureTable",  
  "properties":  
  {  
    "structure":  
    [  
      { "name": "column1", type": "String"},  
      { "name": "column2", type": "String"}  
    ],  
    "location":  
    {  
      "type": "AzureSqlTableLocation",  
      "tableName": "your_db_name",  
      "linkedServiceName": "adfdssqlazure_linked_servicename"  
    },  
    "availability":  
    {  
      "frequency": "Day",  
      "interval": 1  
    }  
  }  
}
```

Copy the JSON definition of the table into a file called *AzureSqlTable.json* file and save it to a known location (here assumed to be C:\temp\AzureSqlTable.json). Create the table in ADF with the following Azure PowerShell cmdlet:

```
New-AzureDataFactoryTable -ResourceGroupName adfdsp -DataFactoryName adfdsp -File C:\temp\AzureSqlTable.json
```

Define and create the pipeline

Specify the activities that belong to the pipeline and create the pipeline with the following script-based procedures. A JSON file is used to define the pipeline properties.

- The script assumes that the **pipeline name** is *AMLDSPipeline*.
- Also note that we set the periodicity of the pipeline to be executed on daily basis and use the default execution time for the job (12 am UTC).

NOTE

The following procedures use Azure PowerShell to define and create the ADF pipeline. But this task can also be accomplished using the Azure portal. For details, see [Create pipeline](#).

Using the table definitions provided previously, the pipeline definition for the ADF is specified as follows:

```
{
  "name": "AMLDSPProcessPipeline",
  "properties":
  {
    "description" : "This pipeline has one Copy activity that copies data from an on-premise SQL to Azure blob",
    "activities":
    [
      {
        "name": "CopyFromSQLtoBlob",
        "description": "Copy data from on-premise SQL server to blob",
        "type": "CopyActivity",
        "inputs": [ {"name": "OnPremSQLTable"} ],
        "outputs": [ {"name": "OutputBlobTable"} ],
        "transformation":
        {
          "source":
          {
            "type": "SqlSource",
            "sqlReaderQuery": "select * from nyctaxi_data"
          },
          "sink":
          {
            "type": "BlobSink"
          }
        },
        "Policy":
        {
          "concurrency": 3,
          "executionPriorityOrder": "NewestFirst",
          "style": "StartOfInterval",
          "retry": 0,
          "timeout": "01:00:00"
        }
      },
      {
        "name": "CopyFromBlobtoSQLAzure",
        "description": "Push data to Sql Azure",
        "type": "CopyActivity",
        "inputs": [ {"name": "OutputBlobTable"} ],
        "outputs": [ {"name": "OutputSQLAzureTable"} ],
        "transformation":
        {
          "source":
          {
            "type": "BlobSource"
          },
          "sink":
          {
            "type": "SqlSink",
            "WriteBatchTimeout": "00:5:00",
          }
        },
        "Policy":
        {
          "concurrency": 3,
          "executionPriorityOrder": "NewestFirst",
          "style": "StartOfInterval",
          "retry": 2,
          "timeout": "02:00:00"
        }
      }
    ]
  }
}
```

Copy this JSON definition of the pipeline into a file called *pipelinedef.json* file and save it to a known location (here assumed to be C:\temp\pipelinedef.json). Create the pipeline in ADF with the following Azure PowerShell cmdlet:

```
New-AzureDataFactoryPipeline -ResourceGroupName adfdsp -DataFactoryName adfdsp -File C:\temp\pipelinedef.json
```

Confirm that you can see the pipeline on the ADF in the Azure Classic Portal show up as following (when you click the diagram)



Start the Pipeline

The pipeline can now be run using the following command:

```
Set-AzureDataFactoryPipelineActivePeriod -ResourceGroupName ADFdsprg -DataFactoryName ADFdsp -StartTime startdateZ -EndTime enddateZ -Name AMLDSProcessPipeline
```

The *startdate* and *enddate* parameter values need to be replaced with the actual dates between which you want the pipeline to run.

Once the pipeline executes, you should be able to see the data show up in the container selected for the blob, one file per day.

Note that we have not leveraged the functionality provided by ADF to pipe data incrementally. For more information on how to do this and other capabilities provided by ADF, see the [ADF documentation](#).

Parallel Bulk Data Import Using SQL Partition Tables

1/17/2017 • 5 min to read • [Edit on GitHub](#)

This document describes how to build partitioned tables for fast parallel bulk importing of data to a SQL Server database. For big data loading/transfer to a SQL database, importing data to the SQL DB and subsequent queries can be improved by using *Partitioned Tables and Views*.

Create a new database and a set of filegroups

- [Create a new database](#) (if it doesn't exist)
- Add database filegroups to the database which will hold the partitioned physical files

This can be done with `CREATE DATABASE` if new or `ALTER DATABASE` if the database exists already

- Add one or more files (as needed) to each database filegroup

NOTE

Specify the target filegroup which holds data for this partition and the physical database file name(s) where the filegroup data will be stored.

The following example creates a new database with three filegroups other than the primary and log groups, containing one physical file in each. The database files are created in the default SQL Server Data folder, as configured in the SQL Server instance. For more information about the default file locations, see [File Locations for Default and Named Instances of SQL Server](#).

```
DECLARE @data_path nvarchar(256);
SET @data_path =(SELECT SUBSTRING(physical_name, 1, CHARINDEX(N'master.mdf', LOWER(physical_name))- 1)
FROM master.sys.master_files
WHERE database_id = 1 AND file_id = 1);

EXECUTE (
    CREATE DATABASE <database_name>
    ON PRIMARY
    ( NAME = "Primary", FILENAME = "" + @data_path + '<primary_file_name>.mdf', SIZE = 4096KB , FILEGROWTH = 1024KB ),
    FILEGROUP [filegroup_1]
    ( NAME = "FileGroup1", FILENAME = "" + @data_path + '<file_name_1>.ndf' , SIZE = 4096KB , FILEGROWTH = 1024KB ),
    FILEGROUP [filegroup_2]
    ( NAME = "FileGroup1", FILENAME = "" + @data_path + '<file_name_2>.ndf' , SIZE = 4096KB , FILEGROWTH = 1024KB ),
    FILEGROUP [filegroup_3]
    ( NAME = "FileGroup1", FILENAME = "" + @data_path + '<file_name>.ndf' , SIZE = 102400KB , FILEGROWTH = 10240KB ),
    LOG ON
    ( NAME = "LogFileGroup", FILENAME = "" + @data_path + '<log_file_name>.ldf' , SIZE = 1024KB , FILEGROWTH = 10%)
')
```

Create a partitioned table

Create partitioned table(s) according to the data schema, mapped to the database filegroups created in the previous step. When data is bulk imported to the partitioned table(s), records will be distributed among the filegroups according to a partition scheme, as described below.

To create a partition table, you need to:

- [Create a partition function](#) which defines the range of values/boundaries to be included in each individual

partition table, e.g., to limit partitions by month(some_datetime_field) in the year 2013:

```
CREATE PARTITION FUNCTION <DatetimeFieldPFN>(<datetime_field>)
AS RANGE RIGHT FOR VALUES (
    '20130201', '20130301', '20130401',
    '20130501', '20130601', '20130701', '20130801',
    '20130901', '20131001', '20131101', '20131201')
```

- [Create a partition scheme](#) which maps each partition range in the partition function to a physical filegroup, e.g.:

```
CREATE PARTITION SCHEME <DatetimeFieldPScheme> AS
PARTITION <DatetimeFieldPFN> TO (
    <filegroup_1>, <filegroup_2>, <filegroup_3>, <filegroup_4>,
    <filegroup_5>, <filegroup_6>, <filegroup_7>, <filegroup_8>,
    <filegroup_9>, <filegroup_10>, <filegroup_11>, <filegroup_12>)
```

To verify the ranges in effect in each partition according to the function/scheme, run the following query:

```
SELECT psch.name as PartitionScheme,
    pmng.value AS ParitionValue,
    pmng.boundary_id AS BoundaryID
FROM sys.partition_functions AS pfun
INNER JOIN sys.partition_schemes psch ON pfun.function_id = psch.function_id
INNER JOIN sys.partition_range_values pmng ON pmng.function_id=pfun.function_id
WHERE pfun.name = <DatetimeFieldPFN>
```

- [Create partitioned table\(s\)](#) according to your data schema, and specify the partition scheme and constraint field used to partition the table, e.g.:

```
CREATE TABLE <table_name> ( [include schema definition here] )
ON <TablePScheme>(<partition_field>)
```

For more information, see [Create Partitioned Tables and Indexes](#).

Bulk import the data for each individual partition table

- You may use BCP, BULK INSERT, or other methods such as [SQL Server Migration Wizard](#). The example provided uses the BCP method.
- [Alter the database](#) to change transaction logging scheme to BULK_LOGGED to minimize overhead of logging, e.g.:

```
ALTER DATABASE<database_name> SET RECOVERY BULK_LOGGED
```

- To expedite data loading, launch the bulk import operations in parallel. For tips on expediting bulk importing of big data into SQL Server databases, see [Load 1TB in less than 1 hour](#).

The following PowerShell script is an example of parallel data loading using BCP.

```

# Set database name, input data directory, and output log directory
# This example loads comma-separated input data files
# The example assumes the partitioned data files are named as <base_file_name>_<partition_number>.csv
# Assumes the input data files include a header line. Loading starts at line number 2.

$dbname = "<database_name>"
$indir = "<path_to_data_files>"
$logdir = "<path_to_log_directory>"

# Select authentication mode
$sqlauth = 0

# For SQL authentication, set the server and user credentials
$sqlusr = "<user@server>"
$server = "<tcp:serverdns>"
$pass = "<password>"

# Set number of partitions per table - Should match the number of input data files per table
$numofparts = <number_of_partitions>

# Set table name to be loaded, basename of input data files, input format file, and number of partitions
$tname = "<table_name>"
$basename = "<base_input_data_filename_no_extension>"
$fmtfile = "<full_path_to_format_file>"

# Create log directory if it does not exist
New-Item -ErrorAction Ignore -ItemType directory -Path $logdir

# BCP example using Windows authentication
$ScriptBlock1 = {
    param($dbname, $tname, $basename, $fmtfile, $indir, $logdir, $num)
    bcp ($dbname + "..." + $tname) in ($indir + "\" + $basename + "_" + $num + ".csv") -o ($logdir + "\" + $tname + "_" + $num + ".txt") -h "TABLOCK" -F 2 -C "RAW" -f ($fmtfile) -T -b 2500 -t "," -r \n
}

# BCP example using SQL authentication
$ScriptBlock2 = {
    param($dbname, $tname, $basename, $fmtfile, $indir, $logdir, $num, $sqlusr, $server, $pass)
    bcp ($dbname + "..." + $tname) in ($indir + "\" + $basename + "_" + $num + ".csv") -o ($logdir + "\" + $tname + "_" + $num + ".txt") -h "TABLOCK" -F 2 -C "RAW" -f ($fmtfile) -U $sqlusr -S $server -P $pass -b 2500 -t "," -r \n
}

# Background processing of all partitions
for ($i=1; $i -le $numofparts; $i++) {
    Write-Output "Submit loading trip and fare partitions # $i"
    if ($sqlauth -eq 0) {
        # Use Windows authentication
        Start-Job -ScriptBlock $ScriptBlock1 -Arg ($dbname, $tname, $basename, $fmtfile, $indir, $logdir, $i)
    }
    else {
        # Use SQL authentication
        Start-Job -ScriptBlock $ScriptBlock2 -Arg ($dbname, $tname, $basename, $fmtfile, $indir, $logdir, $i, $sqlusr, $server, $pass)
    }
}

Get-Job

# Optional - Wait till all jobs complete and report date and time
date
While (Get-Job -State "Running") { Start-Sleep 10 }
date

```

Create indexes to optimize joins and query performance

- If you will extract data for modeling from multiple tables, create indexes on the join keys to improve the join

performance.

- [Create indexes](#) (clustered or non-clustered) targeting the same filegroup for each partition, for e.g.:

```
CREATE CLUSTERED INDEX <table_idx> ON <table_name>([include index columns here])
ON <TablePScheme>(<partition>field>)
```

or,

```
CREATE INDEX <table_idx> ON <table_name>([include index columns here])
ON <TablePScheme>(<partition>field>)
```

NOTE

You may choose to create the indexes before bulk importing the data. Index creation before bulk importing will slow down the data loading.

Advanced Analytics Process and Technology in Action Example

For an end-to-end walkthrough example using the Cortana Analytics Process with a public dataset, see [Cortana Analytics Process in Action: using SQL Server](#).

Import your training data into Azure Machine Learning Studio from various data sources

1/17/2017 • 3 min to read • [Edit on GitHub](#)

To use your own data in Machine Learning Studio to develop and train a predictive analytics solution, you can:

- upload data from a **local file** ahead of time from your hard drive to create a dataset module in your workspace
- access data from one of several **online data sources** while your experiment is running using the [Import Data](#) module
- use data from another Azure Machine learning **experiment** saved as a dataset
- use data from an on-premises **SQL Server database**

Each of these options is described in one of the topics on the menu below. These topics show you how to import data from these various data sources to use in Machine Learning Studio.

NOTE

There are a number of sample datasets available in Machine Learning Studio that you can use for training data. For information on these, see [Use the sample datasets in Azure Machine Learning Studio](#).

This introductory topic also discusses how to get data ready for use in Machine Learning Studio and describes which data formats and data types are supported.

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

Get data ready for use in Azure Machine Learning Studio

Machine Learning Studio is designed to work with rectangular or tabular data, such as text data that's delimited or structured data from a database, though in some circumstances non-rectangular data may be used.

It's best if your data is relatively clean. That is, you'll want to take care of issues such as unquoted strings before you upload the data into your experiment.

However, there are modules available in Machine Learning Studio that enable some manipulation of data within your experiment. Depending on the machine learning algorithms you'll be using, you may need to decide how you'll handle data structural issues such as missing values and sparse data, and there are modules that can help with that. Look in the **Data Transformation** section of the module palette for modules that perform these functions.

At any point in your experiment you can view or download the data that's produced by a module by clicking the output port. Depending on the module, there may be different download options available, or you may be able to visualize the data within your web browser in Machine Learning Studio.

Data formats and data types supported

You can import a number of data types into your experiment, depending on what mechanism you use to import data and where it's coming from:

- Plain text (.txt)
- Comma-separated values (CSV) with a header (.csv) or without (.nh.csv)
- Tab-separated values (TSV) with a header (.tsv) or without (.nh.tsv)
- Excel file
- Azure table
- Hive table
- SQL database table
- OData values
- SVMLight data (.svmlight) (see the [SVMLight definition](#) for format information)
- Attribute Relation File Format (ARFF) data (.arff) (see the [ARFF definition](#) for format information)
- Zip file (.zip)
- R object or workspace file (.RData)

If you import data in a format such as ARFF that includes metadata, Machine Learning Studio uses this metadata to define the heading and data type of each column.

If you import data such as TSV or CSV format that doesn't include this metadata, Machine Learning Studio infers the data type for each column by sampling the data. If the data also doesn't have column headings, Machine Learning Studio provides default names.

You can explicitly specify or change the headings and data types for columns using the [Edit Metadata](#).

The following **data types** are recognized by Machine Learning Studio:

- String
- Integer
- Double
- Boolean
- DateTime
- TimeSpan

Machine Learning Studio uses an internal data type called **Data Table** to pass data between modules. You can explicitly convert your data into Data Table format using the [Convert to Dataset](#) module.

Any module that accepts formats other than Data Table will convert the data to Data Table silently before passing it to the next module.

If necessary, you can convert Data Table format back into CSV, TSV, ARFF, or SVMLight format using other conversion modules. Look in the **Data Format Conversions** section of the module palette for modules that perform these functions.

Import your training data into Azure Machine Learning Studio from a local file

1/17/2017 • 1 min to read • [Edit on GitHub](#)

To use your own data in Machine Learning Studio, you can upload a data file ahead of time from your local hard drive to create a dataset module in your workspace.

Import data from a local file

You can import data from a local hard drive by doing the following:

1. Click **+NEW** at the bottom of the Machine Learning Studio window.
2. Select **DATASET** and **FROM LOCAL FILE**.
3. In the **Upload a new dataset** dialog, browse to the file you want to upload
4. Enter a name, identify the data type, and optionally enter a description. A description is recommended - it allows you to record any characteristics about the data that you want to remember when using the data in the future.
5. The checkbox **This is the new version of an existing dataset** allows you to update an existing dataset with new data. Click this checkbox and then enter the name of an existing dataset.

During upload, you'll see a message that your file is being uploaded. Upload time depends on the size of your data and the speed of your connection to the service. If you know the file will take a long time, you can do other things inside Machine Learning Studio while you wait. However, closing the browser causes the data upload to fail.

Once your data is uploaded, it's stored in a dataset module and is available to any experiment in your workspace. When you're editing an experiment, you can find the datasets you've created in the **My Datasets** list under the **Saved Datasets** list in the module palette. You can drag and drop the dataset onto the experiment canvas when you want to use the dataset for further analytics and machine learning.

Import data into Azure Machine Learning Studio from various online data sources with the Import Data module

1/17/2017 • 8 min to read • [Edit on GitHub](#)

This article describes the support for importing online data from various sources and the information needed to move data from these sources into an Azure Machine Learning experiment.

NOTE

This article provides general information about the [Import Data](#) module. For more detailed information about the types of data you can access, formats, parameters, and answers to common questions, see the module reference topic for the [Import Data](#) module.

Introduction

By using the [Import Data](#) module, you can access data from one of several online data sources while your experiment is running in [Azure Machine Learning Studio](#):

- A Web URL using HTTP
- Hadoop using HiveQL
- Azure blob storage
- Azure table
- Azure SQL database or SQL Server on Azure VM
- On-premises SQL Server database
- A data feed provider, OData currently

To access online data sources in your Studio experiment, add the [Import Data](#) module to your, select the **Data source**, and then provide the parameters needed to access the data. The online data sources that are supported are itemized in the table below. This table also summarizes the file formats that are supported and parameters that are used to access the data.

Note that because this training data is accessed while your experiment is running, it's only available in that experiment. By comparison, data that has been stored in a dataset module are available to any experiment in your workspace.

IMPORTANT

Currently, the [Import Data](#) and [Export Data](#) modules can read and write data only from Azure storage created using the Classic deployment model. In other words, the new Azure Blob Storage account type that offers a hot storage access tier or cool storage access tier is not yet supported.

Generally, any Azure storage accounts that you might have created before this service option became available should not be affected. If you need to create a new account, select **Classic** for the Deployment model, or use Resource manager and select **General purpose** rather than **Blob storage** for **Account kind**.

For more information, see [Azure Blob Storage: Hot and Cool Storage Tiers](#).

Supported online data sources

Azure Machine Learning **Import Data** module supports the following data sources:

DATA SOURCE	DESCRIPTION	PARAMETERS
Web URL via HTTP	Reads data in comma-separated values (CSV), tab-separated values (TSV), attribute-relation file format (ARFF), and Support Vector Machines (SVM-light) formats, from any web URL that uses HTTP	<p>URL: Specifies the full name of the file, including the site URL and the file name, with any extension.</p> <p>Data format: Specifies one of the supported data formats: CSV, TSV, ARFF, or SVM-light. If the data has a header row, it is used to assign column names.</p>
Hadoop/HDFS	Reads data from distributed storage in Hadoop. You specify the data you want by using HiveQL, a SQL-like query language. HiveQL can also be used to aggregate data and perform data filtering before you add the data to Machine Learning Studio.	<p>Hive database query: Specifies the Hive query used to generate the data.</p> <p>HCatalog server URI : Specified the name of your cluster using the format <<i>your cluster name</i>>.azurehdinsight.net.</p> <p>Hadoop user account name: Specifies the Hadoop user account name used to provision the cluster.</p> <p>Hadoop user account password : Specifies the credentials used when provisioning the cluster. For more information, see Create Hadoop clusters in HDInsight.</p> <p>Location of output data: Specifies whether the data is stored in a Hadoop distributed file system (HDFS) or in Azure. If you store output data in HDFS, specify the HDFS server URI. (Be sure to use the HDInsight cluster name without the HTTPS:// prefix).</p> <p>If you store your output data in Azure, you must specify the Azure storage account name, Storage access key and Storage container name.</p>

DATA SOURCE	DESCRIPTION	PARAMETERS
SQL database	<p>Reads data that is stored in an Azure SQL database or in a SQL Server database running on an Azure virtual machine.</p>	<p>Database server name: Specifies the name of the server on which the database is running. In case of Azure SQL Database enter the server name that is generated. Typically it has the form <i><generated_identifier>.database.windows.net</i>.</p> <p>In case of a SQL server hosted on a Azure Virtual machine enter <i>tcp:<Virtual Machine DNS Name>,1433</i></p> <p>Database name : Specifies the name of the database on the server.</p> <p>Server user account name: Specifies a user name for an account that has access permissions for the database.</p> <p>Server user account password: Specifies the password for the user account.</p> <p>Accept any server certificate: Use this option (less secure) if you want to skip reviewing the site certificate before you read your data.</p> <p>Database query: Enter a SQL statement that describes the data you want to read.</p>

DATA SOURCE	DESCRIPTION	PARAMETERS
On-premises SQL database	<p>Reads data that is stored in an on-premises SQL database.</p>	<p>Data gateway: Specifies the name of the Data Management Gateway installed on a computer where it can access your SQL Server database. For information about setting up the gateway, see Perform advanced analytics with Azure Machine Learning using data from an on-premises SQL server.</p> <p>Database server name: Specifies the name of the server on which the database is running.</p> <p>Database name : Specifies the name of the database on the server.</p> <p>Server user account name: Specifies a user name for an account that has access permissions for the database.</p> <p>User name and password: Click Enter values to enter your database credentials. You can use Windows Integrated Authentication or SQL Server Authentication depending upon how your on-premises SQL Server is configured.</p> <p>Database query: Enter a SQL statement that describes the data you want to read.</p>
Azure Table	<p>Reads data from the Table service in Azure Storage.</p> <p>If you read large amounts of data infrequently, use the Azure Table Service. It provides a flexible, non-relational (NoSQL), massively scalable, inexpensive, and highly available storage solution.</p>	<p>The options in the Import Data change depending on whether you are accessing public information or a private storage account that requires login credentials. This is determined by the Authentication Type which can have value of "PublicOrSAS" or "Account", each of which has its own set of parameters.</p> <p>Public or Shared Access Signature (SAS) URI: The parameters are:</p> <p>Table URI: Specifies the Public or SAS URL for the table.</p> <p>Specifies the rows to scan for property names: The values are <i>TopN</i> to scan the specified number of rows, or <i>ScanAll</i> to get all rows in the table.</p> <p>If the data is homogeneous and predictable, it is recommended that you select <i>TopN</i> and enter a number for N. For large tables, this can result in quicker reading times.</p> <p>If the data is structured with sets of</p>

DATA SOURCE	DESCRIPTION	
	<p>properties that vary based on the PARAMETERS depth and position of the table, choose the <i>ScanAll</i> option to scan all rows. This ensures the integrity of your resulting property and metadata conversion.</p> <p>Private Storage Account: The parameters are:</p> <p>Account name: Specifies the name of the account that contains the table to read.</p> <p>Account key: Specifies the storage key associated with the account.</p> <p>Table name : Specifies the name of the table that contains the data to read.</p> <p>Rows to scan for property names: The values are <i>TopN</i> to scan the specified number of rows, or <i>ScanAll</i> to get all rows in the table.</p> <p>If the data is homogeneous and predictable, we recommend that you select <i>TopN</i> and enter a number for N. For large tables, this can result in quicker reading times.</p> <p>If the data is structured with sets of properties that vary based on the depth and position of the table, choose the <i>ScanAll</i> option to scan all rows. This ensures the integrity of your resulting property and metadata conversion.</p>	

DATA SOURCE	DESCRIPTION	PARAMETERS
Azure Blob Storage	<p>Reads data stored in the Blob service in Azure Storage, including images, unstructured text, or binary data.</p> <p>You can use the Blob service to publicly expose data, or to privately store application data. You can access your data from anywhere by using HTTP or HTTPS connections.</p>	<p>The options in the Import Data module change depending on whether you are accessing public information or a private storage account that requires login credentials. This is determined by the Authentication Type which can have a value either of "PublicOrSAS" or of "Account".</p> <p>Public or Shared Access Signature (SAS) URI: The parameters are:</p> <ul style="list-style-type: none"> URI: Specifies the Public or SAS URL for the storage blob. File Format: Specifies the format of the data in the Blob service. The supported formats are CSV, TSV, and ARFF. <p>Private Storage Account: The parameters are:</p> <ul style="list-style-type: none"> Account name: Specifies the name of the account that contains the blob you want to read. Account key: Specifies the storage key associated with the account. Path to container, directory, or blob : Specifies the name of the blob that contains the data to read. Blob file format: Specifies the format of the data in the blob service. The supported data formats are CSV, TSV, ARFF, CSV with a specified encoding, and Excel. If the format is CSV or TSV, be sure to indicate whether the file contains a header row. You can use the Excel option to read data from Excel workbooks. In the <i>Excel data format</i> option, indicate whether the data is in an Excel worksheet range, or in an Excel table. In the <i>Excel sheet or embedded table</i> option, specify the name of the sheet or table that you want to read from.

DATA SOURCE	DESCRIPTION	PARAMETERS
Data Feed Provider	Reads data from a supported feed provider. Currently only the Open Data Protocol (OData) format is supported.	<p>Data content type: Specifies the OData format.</p> <p>Source URL: Specifies the full URL for the data feed. For example, the following URL reads from the Northwind sample database: http://services.odata.org/northwind/northwind.svc/</p>

Next steps

[Deploying Azure ML web services that use Data Import and Data Export modules](#)

Import your data into Azure Machine Learning Studio from another experiment

1/17/2017 • 1 min to read • [Edit on GitHub](#)

There will be times when you'll want to take an intermediate result from one experiment and use it as part of another experiment. To do this, you save the module as a dataset:

1. Click the output of the module that you want to save as a dataset.
2. Click **Save as Dataset**.
3. When prompted, enter a name and a description that would allow you to identify the dataset easily.
4. Click the **OK** checkmark.

When the save finishes, the dataset will be available for use within any experiment in your workspace. You can find it in the **Saved Datasets** list in the module palette.

Perform advanced analytics with Azure Machine Learning using data from an on-premises SQL Server database

1/17/2017 • 9 min to read • [Edit on GitHub](#)

Often enterprises that work with on-premises data would like to take advantage of the scale and agility of the cloud for their machine learning workloads. But they don't want to disrupt their current business processes and workflows by moving their on-premises data to the cloud. Azure Machine Learning now supports reading your data from an on-premises SQL Server database and then training and scoring a model with this data. You no longer have to manually copy and sync the data between the cloud and your on-premises server. Instead, the **Import Data** module in Azure Machine Learning Studio can now read directly from your on-premises SQL Server database for your training and scoring jobs.

This article provides an overview of how to ingress on-premises SQL server data into Azure Machine Learning. It assumes that you're familiar with Azure Machine Learning concepts like workspaces, modules, datasets, experiments, etc..

NOTE

This feature is not available for free workspaces. For more information about Machine Learning pricing and tiers, see [Azure Machine Learning Pricing](#).

NOTE

[Try Azure Machine Learning for free](#)

No credit card or Azure subscription needed. [Get started now >](#)

Install the Microsoft Data Management Gateway

To access an on-premises SQL Server database in Azure Machine Learning, you need to download and install the Microsoft Data Management Gateway. When you configure the gateway connection in Machine Learning Studio, you have the opportunity to download and install the gateway using the **Download and register data gateway** dialog described below.

You also can install the Data Management Gateway ahead of time by downloading and running the MSI setup package from the [Microsoft Download Center](#). Choose the latest version, selecting either 32-bit or 64-bit as appropriate for your computer. The MSI can also be used to upgrade an existing Data Management Gateway to the latest version, with all settings preserved.

The gateway has the following prerequisites:

- The supported Windows operating system versions are Windows 7, Windows 8/8.1, Windows 10, Windows Server 2008 R2, Windows Server 2012, and Windows Server 2012 R2.
- The recommended configuration for the gateway machine is at least 2 GHz, 4 cores, 8GB RAM, and 80GB disk.
- If the host machine hibernates, the gateway won't respond to data requests. Therefore, configure an appropriate power plan on the computer before installing the gateway. If the machine is configured to hibernate, the gateway installation displays a message.

- Because copy activity occurs at a specific frequency, the resource usage (CPU, memory) on the machine also follows the same pattern with peak and idle times. Resource utilization also depends heavily on the amount of data being moved. When multiple copy jobs are in progress, you'll observe resource usage go up during peak times. While the minimum configuration listed above is technically sufficient, you may want to have a configuration with more resources than the minimum configuration depending on your specific load for data movement.

Consider the following when setting up and using a Data Management Gateway:

- You can install only one instance of Data Management Gateway on a single computer.
- You can use a single gateway for multiple on-premises data sources.
- You can connect multiple gateways on different computers to the same on-premises data source.
- You configure a gateway for only one workspace at a time. Currently, gateways can't be shared across workspaces.
- You can configure multiple gateways for a single workspace. For example, you may want to use a gateway that's connected to your test data sources during development and a production gateway when you're ready to operationalize.
- The gateway does not need to be on the same machine as the data source. But staying closer to the data source reduces the time for the gateway to connect to the data source. We recommend that you install the gateway on a machine that's different from the one that hosts the on-premises data source so that the gateway and data source don't compete for resources.
- If you already have a gateway installed on your computer serving Power BI or Azure Data Factory scenarios, install a separate gateway for Azure Machine Learning on another computer.

NOTE

You can't run Data Management Gateway and Power BI Gateway on the same computer.

- You need to use the Data Management Gateway for Azure Machine Learning even if you are using Azure ExpressRoute for other data. You should treat your data source as an on-premises data source (that's behind a firewall) even when you use ExpressRoute. Use the Data Management Gateway to establish connectivity between Machine Learning and the data source.

You can find detailed information on installation prerequisites, installation steps, and troubleshooting tips in the article [Data Management Gateway](#).

Ingress data from your on-premises SQL Server database into Azure Machine Learning

In this walkthrough, you will set up a Data Management Gateway in an Azure Machine Learning workspace, configure it, and then read data from an on-premises SQL Server database.

TIP

Before you start, disable your browser's pop-up blocker for studio.azureml.net. If you're using the Google Chrome browser, download and install one of the several plug-ins available at Google Chrome WebStore [Click Once App Extension](#).

Step 1: Create a gateway

The first step is to create and set up the gateway to access your on-premises SQL database.

- Log in to [Azure Machine Learning Studio](#) and select the workspace that you want to work in.
- Click the **SETTINGS** blade on the left, and then click the **DATA GATEWAYS** tab at the top.

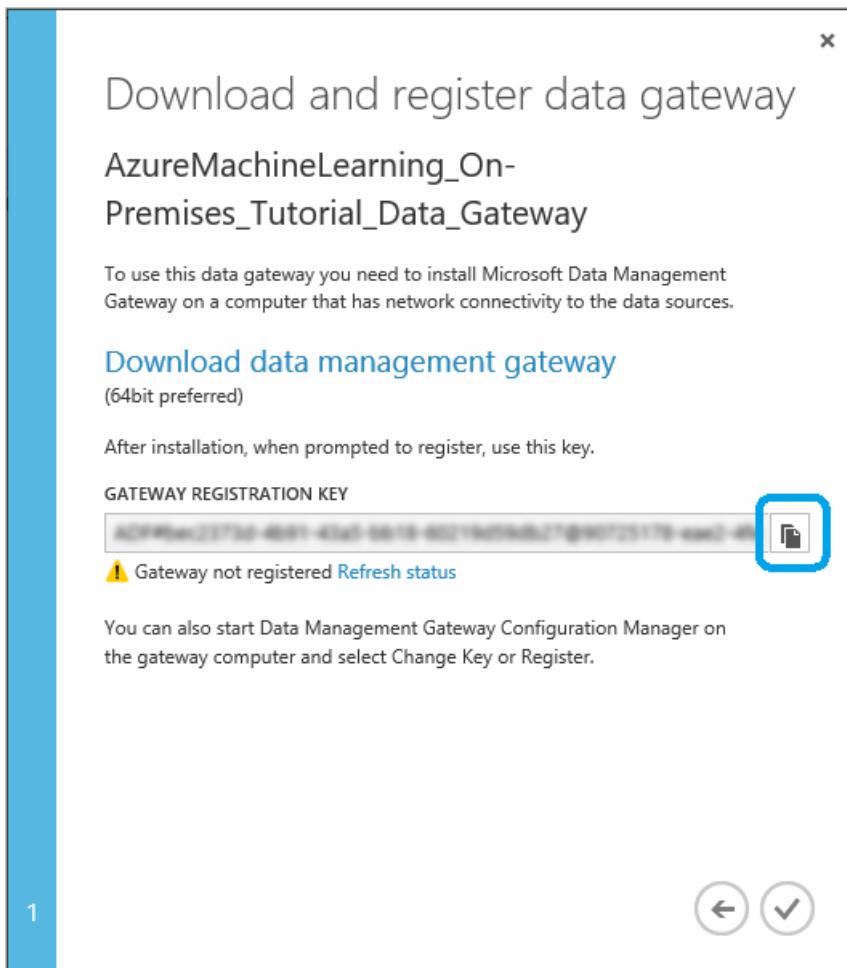
3. Click **NEW DATA GATEWAY** at the bottom of the screen.

The screenshot shows the Azure Machine Learning studio interface. On the left, there is a vertical sidebar with icons and labels: PROJECTS, EXPERIMENTS, WEB SERVICES, NOTEBOOKS, DATASETS, TRAINED MODELS, and SETTINGS. Below this is a dark bar with a '+ NEW' button and a 'NEW DATA GATEWAY' button, which is highlighted with a blue box. The main area is titled 'settings' and contains tabs: NAME, AUTHORIZATION TOKENS, USERS, and DATA GATEWAYS. The 'DATA GATEWAYS' tab is selected and highlighted with a red box. Below the tabs is a table header with columns: NAME, HOST, and STATUS, followed by a search icon. At the bottom of the interface are buttons for DELETE, EDIT DESCRIPTION, DOWNLOAD AND REGISTER, and REFRESH.

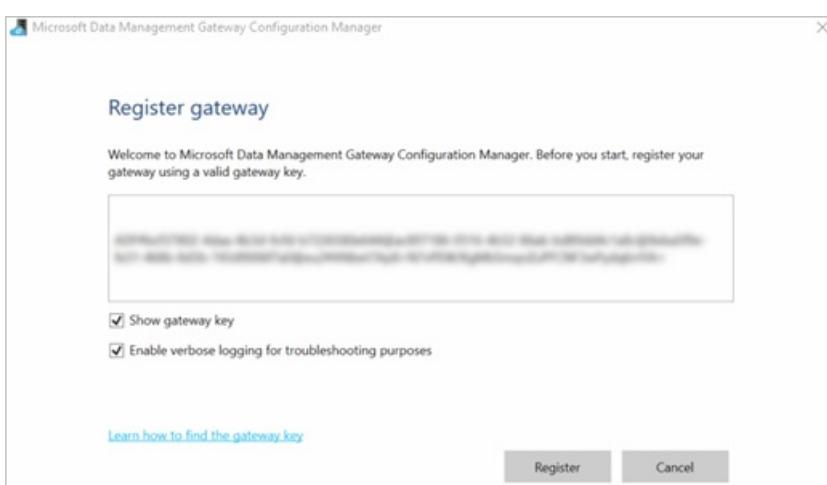
4. In the **New data gateway** dialog, enter the **Gateway Name** and optionally add a **Description**. Click the arrow on the bottom right-hand corner to go to the next step of the configuration.

The screenshot shows the 'New data gateway' configuration dialog. The title is 'New data gateway'. A descriptive text states: 'A data gateway provides access to on-premise data sources in your corporate environment.' There are two input fields: 'GATEWAY NAME' containing 'AzureMachineLearning_On-Premises_Tutorial_Data_Gateway' and 'DESCRIPTION' containing 'This gateway is used to access data from my on-premises SQL server.' At the bottom right is a large blue button with a white arrow pointing right, labeled '2' in the bottom right corner.

5. In the Download and register data gateway dialog, copy the GATEWAY REGISTRATION KEY to the clipboard.

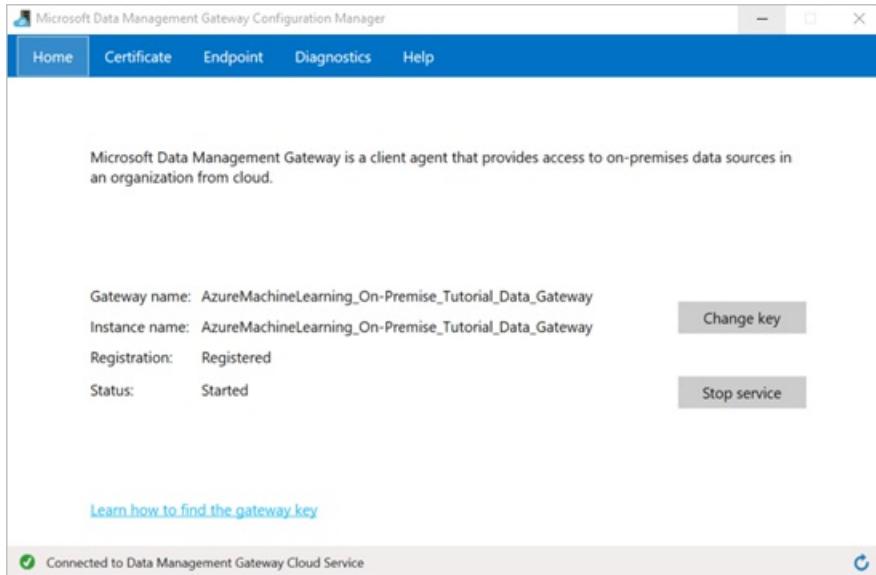


6. If you have not yet downloaded and installed the Microsoft Data Management Gateway, then click **Download data management gateway**. This takes you to the Microsoft Download Center where you can select the gateway version you need, download it, and install it. You can find detailed information on installation prerequisites, installation steps, and troubleshooting tips in the beginning sections of the article [Move data between on-premises sources and cloud with Data Management Gateway](#).
7. After the gateway is installed, the Data Management Gateway Configuration Manager will open and the **Register gateway** dialog is displayed. Paste the **Gateway Registration Key** that you copied to the clipboard and click **Register**.
8. If you already have a gateway installed, run the Data Management Gateway Configuration Manager. Click **Change key**, paste the **Gateway Registration Key** that you copied to the clipboard in the previous step, and click **OK**.
9. When the installation is complete, the **Register gateway** dialog for Microsoft Data Management Gateway Configuration Manager is displayed. Paste the GATEWAY REGISTRATION KEY that you copied to the clipboard in a previous step and click **Register**.

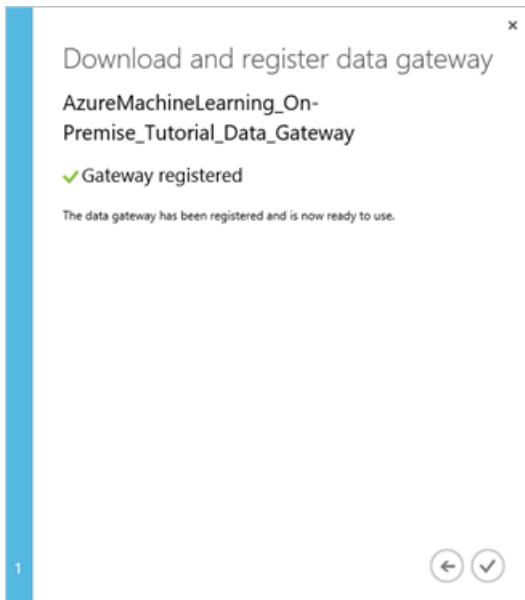


10. The gateway configuration is complete when the following values are set on the **Home** tab in Microsoft Data Management Gateway Configuration Manager:

- **Gateway name** and **Instance name** are set to the name of the gateway.
- **Registration** is set to **Registered**.
- **Status** is set to **Started**.
- The status bar at the bottom displays **Connected to Data Management Gateway Cloud Service** along with a green check mark.



Azure Machine Learning Studio also gets updated when the registration is successful.



11. In the **Download and register data gateway** dialog, click the check mark to complete the setup. The **Settings** page displays the gateway status as "Online". In the right-hand pane, you'll find status and other useful information.

AzureMachineLearning_On-Premise_Tuto...
Description
This Gateway is used to access data from my on-premise sql server
host
krishnacarbon3r.redmond.corp.microsoft.com
Status
✓ Online
Last connected
11/17/2015 12:58:52 PM
Version
1.7.5764.1
✓ Up to date
Created
11/17/2015 11:28:10 AM
Registered
11/17/2015 12:45:06 PM

12. In the Microsoft Data Management Gateway Configuration Manager switch to the **Certificate** tab. The certificate specified on this tab is used to encrypt/decrypt credentials for the on-premises data store that you specify in the portal. This certificate is the default certificate. Microsoft recommends changing this to your own certificate that you back up in your certificate management system. Click **Change** to use your own certificate instead.

! Export and backup the certificate that you must use to restore the gateway in case of a failure.
The certificate will be used to encrypt data source credentials before they are saved into the cloud credential store.

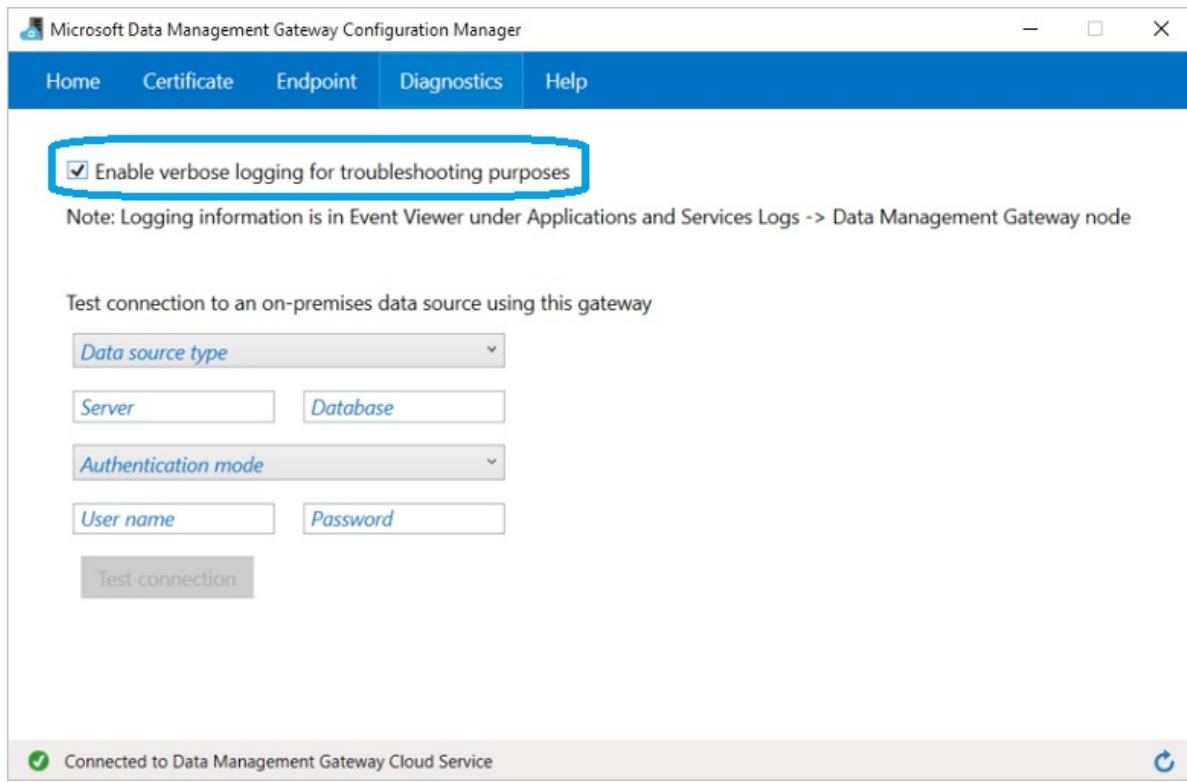
Current certificate:

Data Management Gateway Credential Certificate

View... Change... Export...

Connected to Data Management Gateway Cloud Service

13. (optional) If you want to enable verbose logging in order to troubleshoot issues with the gateway, in the Microsoft Data Management Gateway Configuration Manager switch to the **Diagnostics** tab and check the **Enable verbose logging for troubleshooting purposes** option. The logging information can be found in the Windows Event Viewer under the **Applications and Services Logs -> Data Management Gateway** node. You can also use the **Diagnostics** tab to test the connection to an on-premises data source using the gateway.



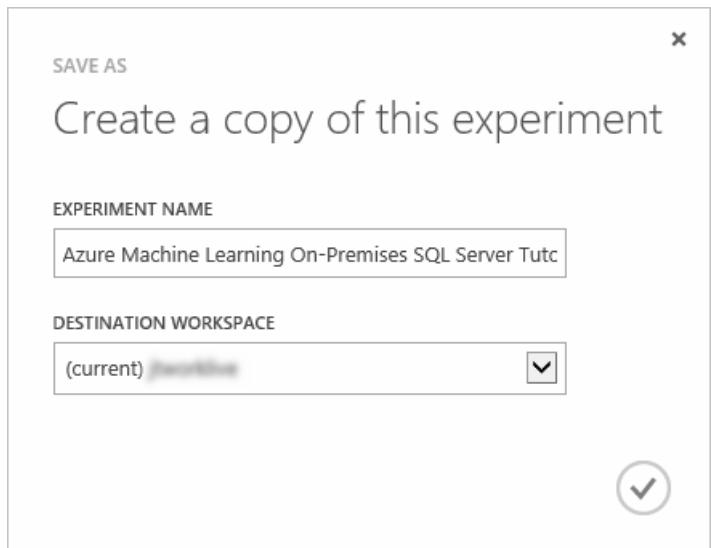
This completes the gateway setup process in Azure Machine Learning. You're now ready to use your on-premises data.

You can create and set up multiple gateways in Studio for each workspace. For example, you may have a gateway that you want to connect to your test data sources during development, and a different gateway for your production data sources. Azure Machine Learning gives you the flexibility to set up multiple gateways depending upon your corporate environment. Currently you can't share a gateway between workspaces and only one gateway can be installed on a single computer. For more information, see [Move data between on-premises sources and cloud with Data Management Gateway](#).

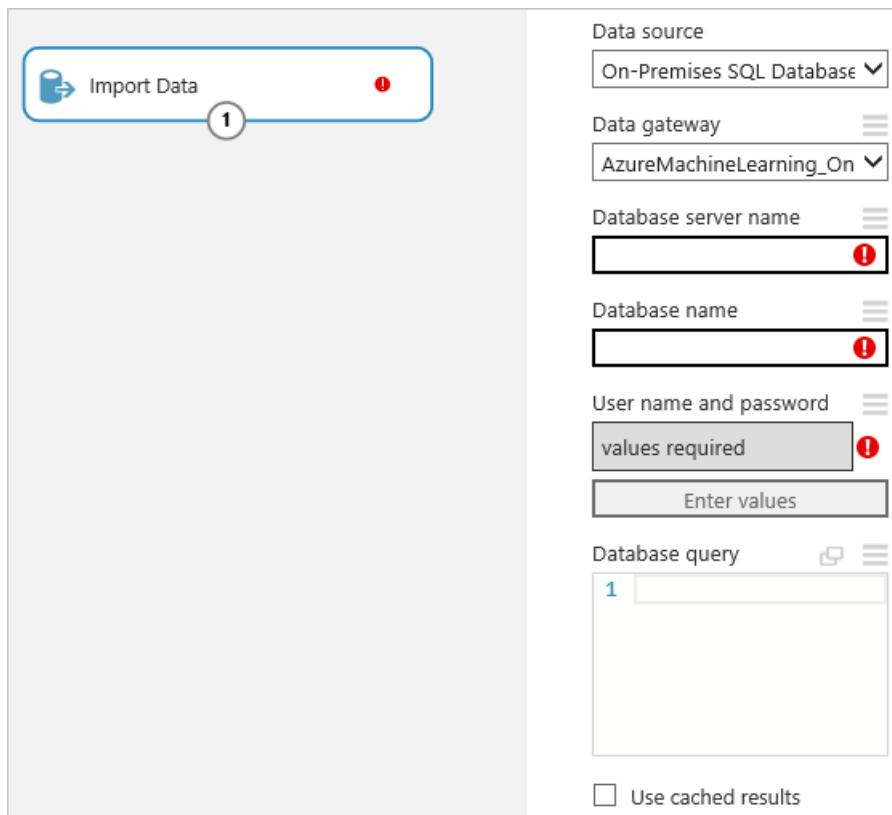
Step 2: Use the gateway to read data from an on-premises data source

After you set up the gateway, you can add an **Import Data** module to an experiment that inputs the data from the on-premises SQL Server database.

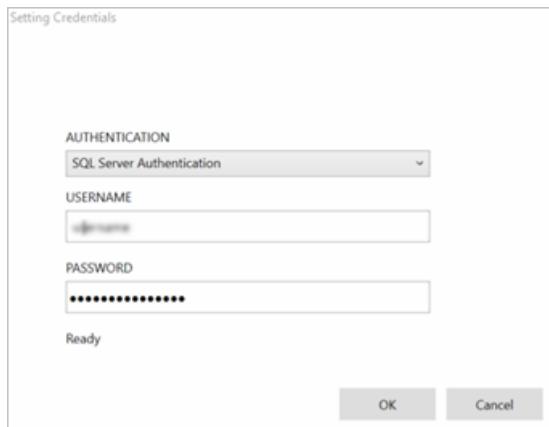
1. In Machine Learning Studio, select the **EXPERIMENTS** tab, click **+NEW** in the lower-left corner, and select **Blank Experiment** (or select one of several sample experiments available).
2. Find and drag the **Import Data** module to the experiment canvas.
3. Click **Save as** below the canvas. Enter "Azure Machine Learning On-Premises SQL Server Tutorial" for the experiment name, select the workspace, and click the **OK** check mark.



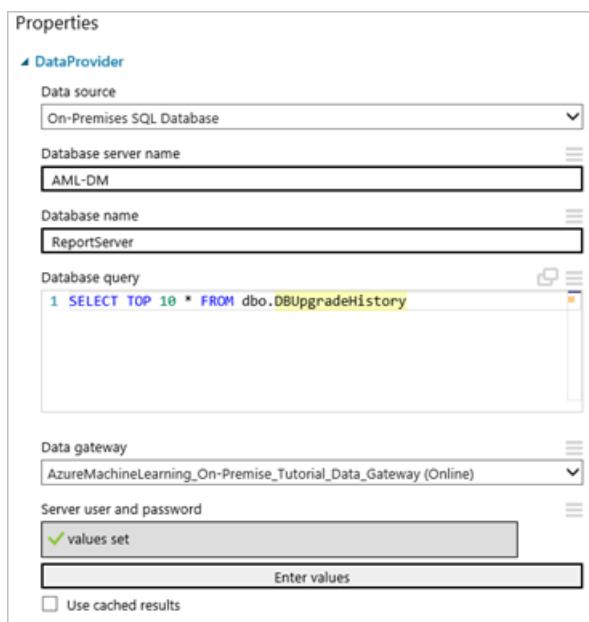
4. Click the **Import Data** module to select it, then in the **Properties** pane to the right of the canvas, select "On-Premises SQL Database" in the **Data source** dropdown list.
5. Select the **Data gateway** you installed and registered. You can set up another gateway by selecting "(add new Data Gateway...)".



6. Enter the SQL **Database server name** and **Database name**, along with the SQL **Database query** you want to execute.
7. Click **Enter values** under **User name and password** and enter your database credentials. You can use Windows Integrated Authentication or SQL Server Authentication depending upon how your on-premises SQL Server is configured.



The message "values required" changes to "values set" with a green check mark. You only need to enter the credentials once unless the database information or password changes. Azure Machine Learning uses the certificate you provided when you installed the gateway to encrypt the credentials in the cloud. Azure never stores on-premises credentials without encryption.



8. Click **RUN** to run the experiment.

Once the experiment finishes running, you can visualize the data you imported from the database by clicking the output port of the **Import Data** module and selecting **Visualize**.

Once you finish developing your experiment, you can deploy and operationalize your model. Using the Batch Execution Service, data from the on-premises SQL Server database configured in the **Import Data** module will be read and used for scoring. While you can use the Request Response Service for scoring on-premises data, Microsoft recommends using the [Excel Add-in](#) instead. Currently, writing to an on-premises SQL Server database through **Export Data** is not supported either in your experiments or published web services.

Tasks to prepare data for enhanced machine learning

1/17/2017 • 6 min to read • [Edit on GitHub](#)

Pre-processing and cleaning data are important tasks that typically must be conducted before dataset can be used effectively for machine learning. Raw data is often noisy and unreliable, and may be missing values. Using such data for modeling can produce misleading results. These tasks are part of the Team Data Science Process (TDSP) and typically follow an initial exploration of a dataset used to discover and plan the pre-processing required. For more detailed instructions on the TDSP process, see the steps outlined in the [Team Data Science Process](#).

Pre-processing and cleaning tasks, like the data exploration task, can be carried out in a wide variety of environments, such as SQL or Hive or Azure Machine Learning Studio, and with various tools and languages, such as R or Python, depending where your data is stored and how it is formatted. Since TDSP is iterative in nature, these tasks can take place at various steps in the workflow of the process.

This article introduces various data processing concepts and tasks that can be undertaken either before or after ingesting data into Azure Machine Learning.

For an example of data exploration and pre-processing done inside Azure Machine Learning studio, see the [Pre-processing data in Azure Machine Learning Studio](#) video.

Why pre-process and clean data?

Real world data is gathered from various sources and processes and it may contain irregularities or corrupt data compromising the quality of the dataset. The typical data quality issues that arise are:

- **Incomplete:** Data lacks attributes or containing missing values.
- **Noisy:** Data contains erroneous records or outliers.
- **Inconsistent:** Data contains conflicting records or discrepancies.

Quality data is a prerequisite for quality predictive models. To avoid "garbage in, garbage out" and improve data quality and therefore model performance, it is imperative to conduct a data health screen to spot data issues early and decide on the corresponding data processing and cleaning steps.

What are some typical data health screens that are employed?

We can check the general quality of data by checking:

- The number of **records**.
- The number of **attributes** (or **features**).
- The attribute **data types** (nominal, ordinal, or continuous).
- The number of **missing values**.
- **Well-formedness** of the data.
 - If the data is in TSV or CSV, check that the column separators and line separators always correctly separate columns and lines.
 - If the data is in HTML or XML format, check whether the data is well formed based on their respective standards.
 - Parsing may also be necessary in order to extract structured information from semi-structured or unstructured data.
- **Inconsistent data records.** Check the range of values are allowed. e.g. If the data contains student GPA, check if the GPA is in the designated range, say 0~4.

When you find issues with data, **processing steps** are necessary which often involves cleaning missing values, data normalization, discretization, text processing to remove and/or replace embedded characters which may affect data alignment, mixed data types in common fields, and others.

Azure Machine Learning consumes well-formed tabular data. If the data is already in tabular form, data pre-processing can be performed directly with Azure Machine Learning in the Machine Learning Studio. If data is not in tabular form, say it is in XML, parsing may be required in order to convert the data to tabular form.

What are some of the major tasks in data pre-processing?

- **Data cleaning:** Fill in or missing values, detect and remove noisy data and outliers.
- **Data transformation:** Normalize data to reduce dimensions and noise.
- **Data reduction:** Sample data records or attributes for easier data handling.
- **Data discretization:** Convert continuous attributes to categorical attributes for ease of use with certain machine learning methods.
- **Text cleaning:** remove embedded characters which may cause data misalignment, for e.g., embedded tabs in a tab-separated data file, embedded new lines which may break records, etc.

The sections below detail some of these data processing steps.

How to deal with missing values?

To deal with missing values, it is best to first identify the reason for the missing values to better handle the problem. Typical missing value handling methods are:

- **Deletion:** Remove records with missing values
- **Dummy substitution:** Replace missing values with a dummy value: e.g, *unknown* for categorical or 0 for numerical values.
- **Mean substitution:** If the missing data is numerical, replace the missing values with the mean.
- **Frequent substitution:** If the missing data is categorical, replace the missing values with the most frequent item
- **Regression substitution:** Use a regression method to replace missing values with regressed values.

How to normalize data?

Data normalization re-scales numerical values to a specified range. Popular data normalization methods include:

- **Min-Max Normalization:** Linearly transform the data to a range, say between 0 and 1, where the min value is scaled to 0 and max value to 1.
- **Z-score Normalization:** Scale data based on mean and standard deviation: divide the difference between the data and the mean by the standard deviation.
- **Decimal scaling:** Scale the data by moving the decimal point of the attribute value.

How to discretize data?

Data can be discretized by converting continuous values to nominal attributes or intervals. Some ways of doing this are:

- **Equal-Width Binning:** Divide the range of all possible values of an attribute into N groups of the same size, and assign the values that fall in a bin with the bin number.
- **Equal-Height Binning:** Divide the range of all possible values of an attribute into N groups, each containing the same number of instances, then assign the values that fall in a bin with the bin number.

How to reduce data?

There are various methods to reduce data size for easier data handling. Depending on data size and the domain, the following methods can be applied:

- **Record Sampling:** Sample the data records and only choose the representative subset from the data.
- **Attribute Sampling:** Select only a subset of the most important attributes from the data.
- **Aggregation:** Divide the data into groups and store the numbers for each group. For example, the daily revenue numbers of a restaurant chain over the past 20 years can be aggregated to monthly revenue to reduce the size of the data.

How to clean text data?

Text fields in tabular data may include characters which affect columns alignment and/or record boundaries. For e.g., embedded tabs in a tab-separated file cause column misalignment, and embedded new line characters break record lines. Improper text encoding handling while writing/reading text leads to information loss, inadvertent introduction of unreadable characters, e.g., nulls, and may also affect text parsing. Careful parsing and editing may be required in order to clean text fields for proper alignment and/or to extract structured data from unstructured or semi-structured text data.

Data exploration offers an early view into the data. A number of data issues can be uncovered during this step and corresponding methods can be applied to address those issues. It is important to ask questions such as what is the source of the issue and how the issue may have been introduced. This also helps you decide on the data processing steps that need to be taken to resolve them. The kind of insights one intends to derive from the data can also be used to prioritize the data processing effort.

References

Data Mining: Concepts and Techniques, Third Edition, Morgan Kaufmann, 2011, Jiawei Han, Micheline Kamber, and Jian Pei

Explore data in the Team Data Science Process

1/17/2017 • 1 min to read • [Edit on GitHub](#)

This document covers how to explore data in four different storage environments that are typically used in the Data Science Process:

- **Azure blob container** data is explored using the [Pandas](#) Python package.
- **SQL Server** data is explored by using SQL and by using a programming language like Python.
- **Hive table** data is explored using Hive queries.
- **Azure Machine Learning (AML) Studio** data is explored using AML modules.

The following **menu** links to the topics that describe how to use these tools to explore data from various storage environments.

Explore data in Azure blob storage with Pandas

1/17/2017 • 2 min to read • [Edit on GitHub](#)

This document covers how to explore data that is stored in Azure blob container using [Pandas](#) Python package.

The following **menu** links to topics that describe how to use tools to explore data from various storage environments. This task is a step in the [Data Science Process]().

Prerequisites

This article assumes that you have:

- Created an Azure storage account. If you need instructions, see [Create an Azure Storage account](#)
- Stored your data in an Azure blob storage account. If you need instructions, see [Moving data to and from Azure Storage](#)

Load the data into a Pandas DataFrame

To explore and manipulate a dataset, it must first be downloaded from the blob source to a local file, which can then be loaded in a Pandas DataFrame. Here are the steps to follow for this procedure:

1. Download the data from Azure blob with the following Python code sample using blob service. Replace the variable in the following code with your specific values:

```
from azure.storage.blob import BlobService
import tables

STORAGEACCOUNTNAME=<storage_account_name>
STORAGEACCOUNTKEY=<storage_account_key>
LOCALFILENAME=<local_file_name>
CONTAINERNAME=<container_name>
BLOBNAME=<blob_name>

#download from blob
t1=time.time()
blob_service=BlobService(account_name=STORAGEACCOUNTNAME,account_key=STORAGEACCOUNTKEY)
blob_service.get_blob_to_path(CONTAINERNAME,BLOBNAME,LOCALFILENAME)
t2=time.time()
print("It takes %s seconds to download "+blobname) % (t2 - t1))
```

2. Read the data into a Pandas data-frame from the downloaded file.

```
#LOCALFILE is the file path
dataframe_blobdata = pd.read_csv(LOCALFILE)
```

Now you are ready to explore the data and generate features on this dataset.

Examples of data exploration using Pandas

Here are a few examples of ways to explore data using Pandas:

1. Inspect the **number of rows and columns**

```
print 'the size of the data is: %d rows and %d columns' % dataframe_blobdata.shape
```

2. **Inspect** the first or last few **rows** in the following dataset:

```
dataframe_blobdata.head(10)  
  
dataframe_blobdata.tail(10)
```

3. Check the **data type** each column was imported as using the following sample code

```
for col in dataframe_blobdata.columns:  
    print dataframe_blobdata[col].name, '\t', dataframe_blobdata[col].dtype
```

4. Check the **basic stats** for the columns in the data set as follows

```
dataframe_blobdata.describe()
```

5. Look at the number of entries for each column value as follows

```
dataframe_blobdata['<column_name>'].value_counts()
```

6. **Count missing values** versus the actual number of entries in each column using the following sample code

```
miss_num = dataframe_blobdata.shape[0] - dataframe_blobdata.count()  
print miss_num
```

7. If you have **missing values** for a specific column in the data, you can drop them as follows:

```
dataframe_blobdata_noNA = dataframe_blobdata.dropna() dataframe_blobdata_noNA.shape
```

Another way to replace missing values is with the mode function:

```
dataframe_blobdata_mode = dataframe_blobdata.fillna({'<column_name>':dataframe_blobdata['<column_name>'].mode()[0]})
```

8. Create a **histogram** plot using variable number of bins to plot the distribution of a variable

```
dataframe_blobdata['<column_name>'].value_counts().plot(kind='bar')  
  
np.log(dataframe_blobdata['<column_name>']+1).hist(bins=50)
```

9. Look at **correlations** between variables using a scatterplot or using the built-in correlation function

```
#relationship between column_a and column_b using scatter plot  
plt.scatter(dataframe_blobdata['<column_a>'], dataframe_blobdata['<column_b>'])  
  
#correlation between column_a and column_b  
dataframe_blobdata[['<column_a>', '<column_b>']].corr()
```

Explore data in SQL Server Virtual Machine on Azure

1/17/2017 • 2 min to read • [Edit on GitHub](#)

This document covers how to explore data that is stored in a SQL Server VM on Azure. This can be done by data wrangling using SQL or by using a programming language like Python.

The following **menu** links to topics that describe how to use tools to explore data from various storage environments. This task is a step in the Cortana Analytics Process (CAP).

NOTE

The sample SQL statements in this document assume that data is in SQL Server. If it isn't, refer to the cloud data science process map to learn how to move your data to SQL Server.

Explore SQL data with SQL scripts

Here are a few sample SQL scripts that can be used to explore data stores in SQL Server.

1. Get the count of observations per day

```
SELECT CONVERT(date,<date_columnname>) as date, count(*) as c from<tablename> group by CONVERT(date,<date_columnname>)
```

2. Get the levels in a categorical column

```
select distinct <column_name> from<tablename>
```

3. Get the number of levels in combination of two categorical columns

```
select <column_a>, <column_b>, count(*) from<tablename> group by <column_a>, <column_b>
```

4. Get the distribution for numerical columns

```
select <column_name>, count(*) from<tablename> group by <column_name>
```

NOTE

For a practical example, you can use the [NYC Taxi dataset](#) and refer to the IPNB titled [NYC Data wrangling using IPython Notebook and SQL Server](#) for an end-to-end walk-through.

Explore SQL data with Python

Using Python to explore data and generate features when the data is in SQL Server is similar to processing data in Azure blob using Python, as documented in [Process Azure Blob data in your data science environment](#). The data needs to be loaded from the database into a pandas DataFrame and then can be processed further. We document the process of connecting to the database and loading the data into the DataFrame in this section.

The following connection string format can be used to connect to a SQL Server database from Python using pyodbc (replaceservername, dbname, username, and password with your specific values):

```
#Set up the SQL Azure connection
import pyodbc
conn = pyodbc.connect('DRIVER={SQL Server};SERVER=<servername>;DATABASE=<dbname>;UID=<username>;PWD=<password>')
```

The [Pandas library](#) in Python provides a rich set of data structures and data analysis tools for data manipulation for Python programming. The following code reads the results returned from a SQL Server database into a Pandas data frame:

```
# Query database and load the returned results in pandas data frame
data_frame = pd.read_sql("select <columnname1>, <columnname2>... from <tablename>", conn)
```

Now you can work with the Pandas DataFrame as covered in the topic [Process Azure Blob data in your data science environment](#).

Cortana Analytics Process in Action Example

For an end-to-end walkthrough example of the Cortana Analytics Process using a public dataset, see [The Team Data Science Process in action: using SQL Server](#).

Explore data in Hive tables with Hive queries

1/17/2017 • 1 min to read • [Edit on GitHub](#)

This document provides sample Hive scripts that are used to explore data in Hive tables in an HDInsight Hadoop cluster.

The following **menu** links to topics that describe how to use tools to explore data from various storage environments.

Prerequisites

This article assumes that you have:

- Created an Azure storage account. If you need instructions, see [Create an Azure Storage account](#)
- Provisioned a customized Hadoop cluster with the HDInsight service. If you need instructions, see [Customize Azure HDInsight Hadoop Clusters for Advanced Analytics](#).
- The data has been uploaded to Hive tables in Azure HDInsight Hadoop clusters. If it has not, follow the instructions in [Create and load data to Hive tables](#) to upload data to Hive tables first.
- Enabled remote access to the cluster. If you need instructions, see [Access the Head Node of Hadoop Cluster](#).
- If you need instructions on how to submit Hive queries, see [How to Submit Hive Queries](#)

Example Hive query scripts for data exploration

1. Get the count of observations per partition

```
SELECT <partitionfieldname>, count(*) from <databasename>.<tablename> group by <partitionfieldname>;
```

2. Get the count of observations per day

```
SELECT to_date(<date_columnname>), count(*) from <databasename>.<tablename> group by to_date(<date_columnname>);
```

3. Get the levels in a categorical column

```
SELECT distinct <column_name> from <databasename>.<tablename>
```

4. Get the number of levels in combination of two categorical columns

```
SELECT <column_a>, <column_b>, count(*) from <databasename>.<tablename> group by <column_a>, <column_b>
```

5. Get the distribution for numerical columns

```
SELECT <column_name>, count(*) from <databasename>.<tablename> group by <column_name>
```

6. Extract records from joining two tables

```

SELECT
  a.<common_columnname1> as <new_name1>,
  a.<common_columnname2> as <new_name2>,
  a.<a_column_name1> as <new_name3>,
  a.<a_column_name2> as <new_name4>,
  b.<b_column_name1> as <new_name5>,
  b.<b_column_name2> as <new_name6>
FROM
  (
    SELECT <common_columnname1>,
           <common_columnname2>,
           <a_column_name1>,
           <a_column_name2>,
      FROM <databasename>.<tablename1>
    ) a
  join
  (
    SELECT <common_columnname1>,
           <common_columnname2>,
           <b_column_name1>,
           <b_column_name2>,
      FROM <databasename>.<tablename2>
    ) b
  ON a.<common_columnname1>=b.<common_columnname1> and a.<common_columnname2>=b.<common_columnname2>

```

Additional query scripts for taxi trip data scenarios

Examples of queries that are specific to [NYC Taxi Trip Data](#) scenarios are also provided in [Github repository](#). These queries already have data schema specified and are ready to be submitted to run.

Sample data in Azure blob containers, SQL Server, and Hive tables

1/17/2017 • 1 min to read • [Edit on GitHub](#)

This document links to topics that covers how to sample data that is stored in one of three different Azure locations:

- **Azure blob container data** is sampled by downloading it programmatically and then sampling it with sample Python code.
- **SQL Server data** is sampled using both SQL and the Python Programming Language.
- **Hive table data** is sampled using Hive queries.

The following **menu** links to the topics that describe how to sample data from each of these Azure storage environments.

This sampling task is a step in the [Team Data Science Process \(TDSP\)](#).

Why sample data?

If the dataset you plan to analyze is large, it's usually a good idea to down-sample the data to reduce it to a smaller but representative and more manageable size. This facilitates data understanding, exploration, and feature engineering. Its role in the Cortana Analytics Process is to enable fast prototyping of the data processing functions and machine learning models.

Sample data in Azure blob storage

1/17/2017 • 1 min to read • [Edit on GitHub](#)

This document covers sampling data stored in Azure blob storage by downloading it programmatically and then sampling it using procedures written in Python.

The following **menu** links to topics that describe how to sample data from various storage environments.

Why sample your data? If the dataset you plan to analyze is large, it's usually a good idea to down-sample the data to reduce it to a smaller but representative and more manageable size. This facilitates data understanding, exploration, and feature engineering. Its role in the Cortana Analytics Process is to enable fast prototyping of the data processing functions and machine learning models.

This sampling task is a step in the [Team Data Science Process \(TDSP\)](#).

Download and down-sample data

1. Download the data from Azure blob storage using the blob service from the following sample Python code:

```
from azure.storage.blob import BlobService  
import tables  
  
STORAGEACCOUNTNAME=<storage_account_name>  
STORAGEACCOUNTKEY=<storage_account_key>  
LOCALFILENAME=<local_file_name>  
CONTAINERNAME=<container_name>  
BLOBNAME=<blob_name>  
  
#download from blob  
t1=time.time()  
blob_service=BlobService(account_name=STORAGEACCOUNTNAME,account_key=STORAGEACCOUNTKEY)  
blob_service.get_blob_to_path(CONTAINERNAME,BLOBNAME,LOCALFILENAME)  
t2=time.time()  
print("It takes %s seconds to download "+blobname) % (t2 - t1)
```

2. Read data into a Pandas data-frame from the file downloaded above.

```
import pandas as pd  
  
#directly ready from file on disk  
dataframe_blobdata = pd.read_csv(LOCALFILE)
```

3. Down-sample the data using the `numpy`'s `randomchoice` as follows:

```
# A 1 percent sample  
sample_ratio = 0.01  
sample_size = np.round(dataframe_blobdata.shape[0] * sample_ratio)  
sample_rows = np.random.choice(dataframe_blobdata.index.values, sample_size)  
dataframe_blobdata_sample = dataframe_blobdata.ix[sample_rows]
```

Now you can work with the above data frame with the 1 Percent sample for further exploration and feature generation.

Upload data and read it into Azure Machine Learning

You can use the following sample code to down-sample the data and use it directly in Azure Machine Learning:

1. Write the data frame to a local file

```
dataframe.to_csv(os.path.join(os.getcwd(),LOCALFILENAME), sep='\t', encoding='utf-8', index=False)
```

2. Upload the local file to an Azure blob using the following sample code:

```
from azure.storage.blob import BlobService
import tables

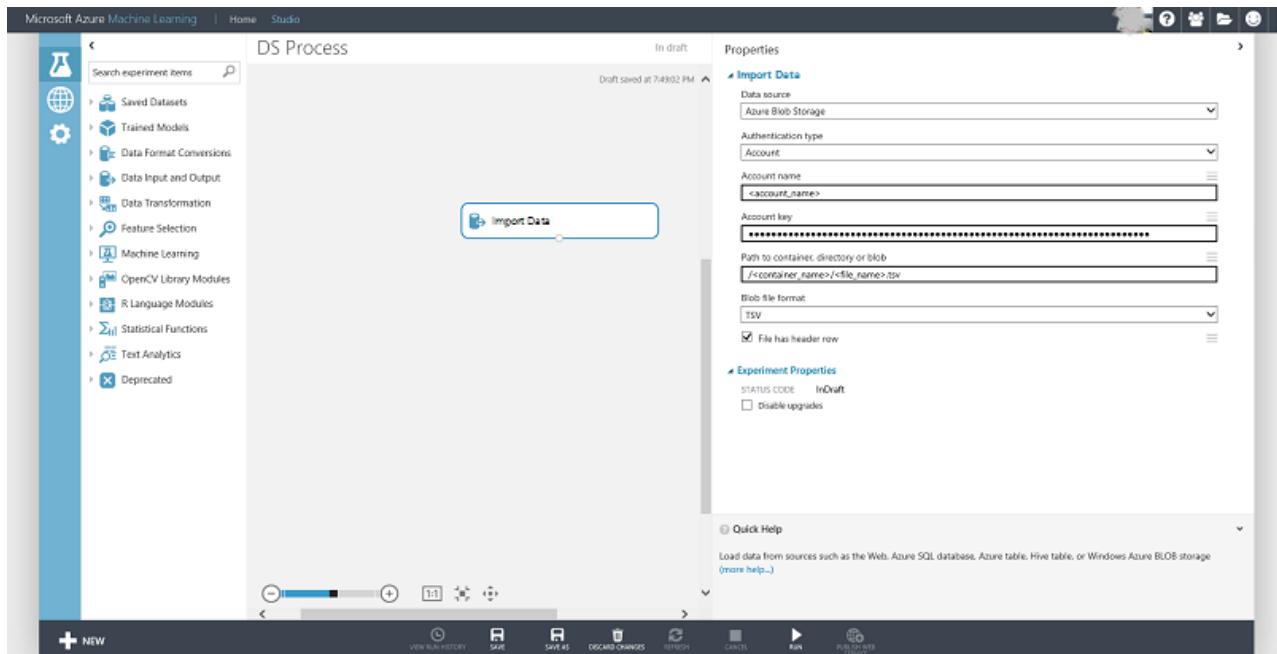
STORAGEACCOUNTNAME=<storage_account_name>
LOCALFILENAME=<local_file_name>
STORAGEACCOUNTKEY=<storage_account_key>
CONTAINERNAME=<container_name>
BLOBNAME=<blob_name>

output_blob_service=BlobService(account_name=STORAGEACCOUNTNAME,account_key=STORAGEACCOUNTKEY)
localfileprocessed = os.path.join(os.getcwd(),LOCALFILENAME) #assuming file is in current working directory

try:
    #perform upload
    output_blob_service.put_block_blob_from_path(CONTAINERNAME,BLOBNAME,localfileprocessed)

except:
    print ("Something went wrong with uploading to the blob:"+ BLOBNAME)
```

3. Read the data from the Azure blob using Azure Machine Learning [Import Data](#) as shown in the image below:



Sample data in SQL Server on Azure

1/17/2017 • 3 min to read • [Edit on GitHub](#)

This document shows how to sample data stored in SQL Server on Azure using either SQL or the Python programming language. It also shows how to move sampled data into Azure Machine Learning by saving it to a file, uploading it to an Azure blob, and then reading it into Azure Machine Learning Studio.

The Python sampling uses the [pyodbc](#) ODBC library to connect to SQL Server on Azure and the [Pandas](#) library to do the sampling.

NOTE

The sample SQL code in this document assumes that the data is in a SQL Server on Azure. If it is not, please refer to [Move data to SQL Server on Azure](#) topic for instructions on how to move your data to SQL Server on Azure.

The following **menu** links to topics that describe how to sample data from various storage environments.

Why sample your data? If the dataset you plan to analyze is large, it's usually a good idea to down-sample the data to reduce it to a smaller but representative and more manageable size. This facilitates data understanding, exploration, and feature engineering. Its role in the [Team Data Science Process \(TDSP\)](#) is to enable fast prototyping of the data processing functions and machine learning models.

This sampling task is a step in the [Team Data Science Process \(TDSP\)](#).

Using SQL

This section describes several methods using SQL to perform simple random sampling against the data in the database. Please choose a method based on your data size and its distribution.

The two items below show how to use newid in SQL Server to perform the sampling. The method you choose depends on how random you want the sample to be (pk_id in the sample code below is assumed to be an auto-generated primary key).

1. Less strict random sample

```
select * from <table_name> where <primary_key> in  
(select top 10 percent <primary_key> from <table_name> order by newid())
```

2. More random sample

```
SELECT * FROM <table_name>  
WHERE 0.1 >= CAST(CHECKSUM(NEWID(), <primary_key>) & 0xffffffff AS float) / CAST (0xffffffff AS int)
```

Tablesample can be used for sampling as well as demonstrated below. This may be a better approach if your data size is large (assuming that data on different pages is not correlated) and for the query to complete in a reasonable time.

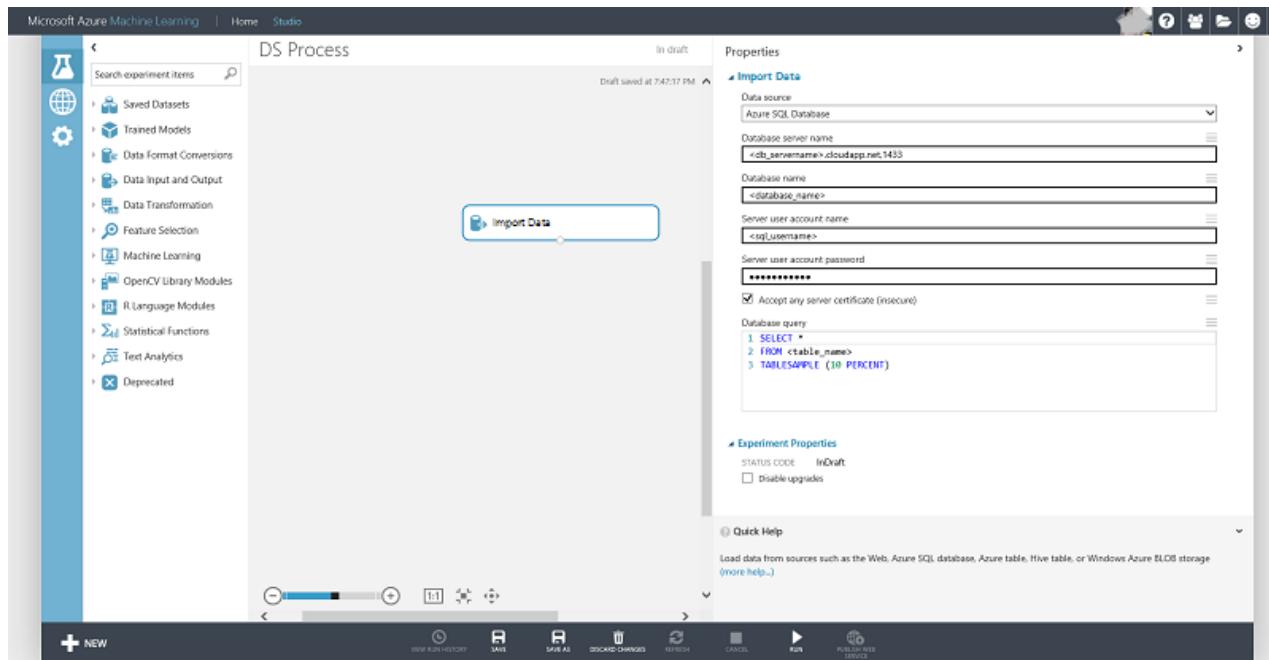
```
SELECT *  
FROM <table_name>  
TABLESAMPLE (10 PERCENT)
```

NOTE

You can explore and generate features from this sampled data by storing it in a new table

Connecting to Azure Machine Learning

You can directly use the sample queries above in the Azure Machine Learning [Import Data](#) module to down-sample the data on the fly and bring it into an Azure Machine Learning experiment. A screen shot of using the reader module to read the sampled data is shown below:



Using the Python programming language

This section demonstrates using the [pyodbc library](#) to establish an ODBC connect to a SQL server database in Python. The database connection string is as follows: (replace servername, dbname, username and password with your configuration):

```
#Set up the SQL Azure connection
import pyodbc
conn = pyodbc.connect('DRIVER={SQL Server};SERVER=<servername>;DATABASE=<dbname>;UID=<username>;PWD=<password>')
```

The [Pandas](#) library in Python provides a rich set of data structures and data analysis tools for data manipulation for Python programming. The code below reads a 0.1% sample of the data from a table in Azure SQL database into a Pandas data :

```
import pandas as pd

# Query database and load the returned results in pandas data frame
data_frame = pd.read_sql("select column1, column2... from <table_name> tablesample (0.1 percent)", conn)
```

You can now work with the sampled data in the Pandas data frame.

Connecting to Azure Machine Learning

You can use the following sample code to save the down-sampled data to a file and upload it to an Azure blob. The data in the blob can be directly read into an Azure Machine Learning Experiment using the [Import Data](#) module. The steps are as follows:

1. Write the pandas data frame to a local file

```
dataframe.to_csv(os.path.join(os.getcwd(),LOCALFILENAME), sep='\t', encoding='utf-8', index=False)
```

2. Upload local file to Azure blob

```
from azure.storage import BlobService
import tables

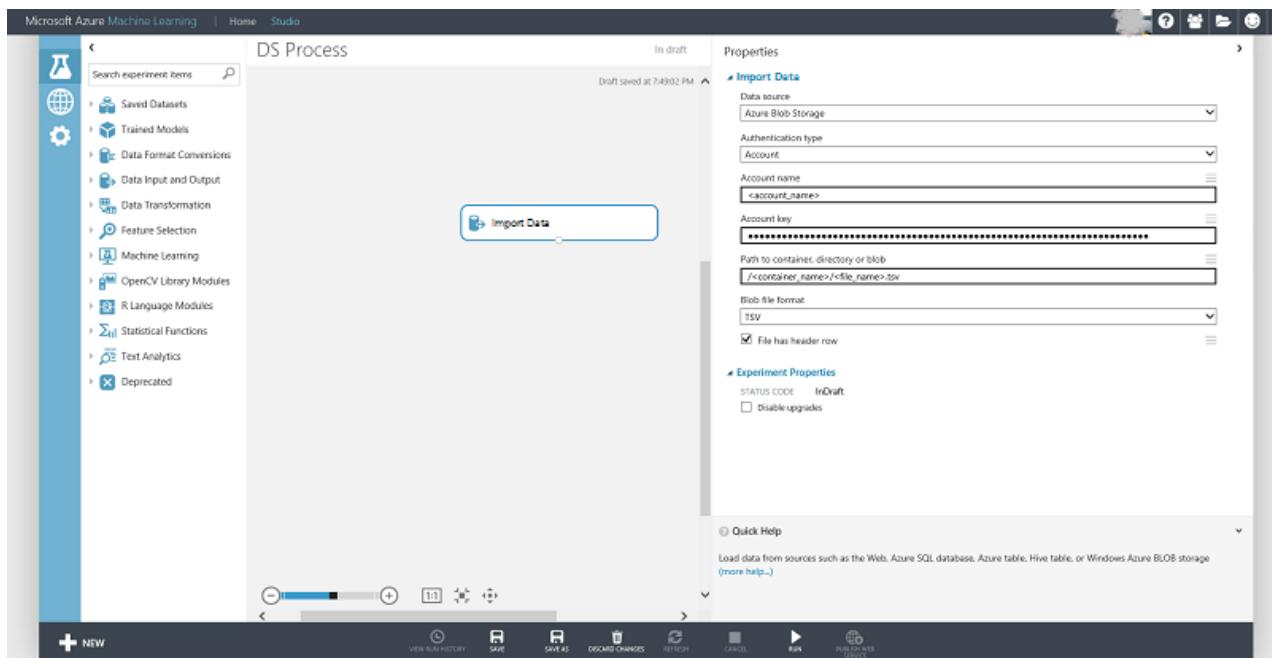
STORAGEACCOUNTNAME=<storage_account_name>
LOCALFILENAME=<local_file_name>
STORAGEACCOUNTKEY=<storage_account_key>
CONTAINERNAME=<container_name>
BLOBNAME=<blob_name>

output_blob_service=BlobService(account_name=STORAGEACCOUNTNAME,account_key=STORAGEACCOUNTKEY)
localfileprocessed = os.path.join(os.getcwd(),LOCALFILENAME) #assuming file is in current working directory

try:
    #perform upload
    output_blob_service.put_block_blob_from_path(CONTAINERNAME,BLOBNAME,localfileprocessed)

except:
    print ("Something went wrong with uploading blob."+BLOBNAME)
```

3. Read data from Azure blob using Azure Machine Learning Import Data module as shown in the screen grab below:



The Team Data Science Process in Action example

For an end-to-end walkthrough example of the Team Data Science Process a using a public dataset, see [Team Data Science Process in Action: using SQL Sever](#).

Sample data in Azure HDInsight Hive tables

1/17/2017 • 3 min to read • [Edit on GitHub](#)

In this article, we describe how to down-sample data stored in Azure HDInsight Hive tables using Hive queries. We cover three popularly used sampling methods:

- Uniform random sampling
- Random sampling by groups
- Stratified sampling

The following **menu** links to topics that describe how to sample data from various storage environments.

Why sample your data? If the dataset you plan to analyze is large, it's usually a good idea to down-sample the data to reduce it to a smaller but representative and more manageable size. This facilitates data understanding, exploration, and feature engineering. Its role in the Team Data Science Process is to enable fast prototyping of the data processing functions and machine learning models.

This sampling task is a step in the [Team Data Science Process \(TDSP\)](#).

How to submit Hive queries

Hive queries can be submitted from the Hadoop Command Line console on the head node of the Hadoop cluster. To do this, log into the head node of the Hadoop cluster, open the Hadoop Command Line console, and submit the Hive queries from there. For instructions on submitting Hive queries in the Hadoop Command Line console, see [How to Submit Hive Queries](#).

Uniform random sampling

Uniform random sampling means that each row in the data set has an equal chance of being sampled. This can be implemented by adding an extra field rand() to the data set in the inner "select" query, and in the outer "select" query that condition on that random field.

Here is an example query:

```
SET sampleRate=<sample rate, 0-1>;
select
    field1, field2, ..., fieldN
from
(
    select
        field1, field2, ..., fieldN, rand() as samplekey
    from <hive table name>
) a
where samplekey<=${hiveconf:sampleRate}'
```

Here, `<sample rate, 0-1>` specifies the proportion of records that the users want to sample.

Random sampling by groups

When sampling categorical data, you may want to either include or exclude all of the instances of some particular value of a categorical variable. This is what is meant by "sampling by group". For example, if you have a categorical variable "State", which has values NY, MA, CA, NJ, PA, etc, you want records of the same state be always together, whether they are sampled or not.

Here is an example query that samples by group:

```
SET sampleRate=<sample rate, 0-1>;
select
  b.field1, b.field2, ..., b.catfield, ..., b.fieldN
from
(
  (
    select
      field1, field2, ..., catfield, ..., fieldN
    from <table name>
  )b
join
(
  (
    select
      catfield
    from
    (
      (
        select
          catfield, rand() as samplekey
        from <table name>
        group by catfield
      )a
      where samplekey<=${hiveconf:sampleRate}
    )c
  on b.catfield=c.catfield
```

Stratified sampling

Random sampling is stratified with respect to a categorical variable when the samples obtained have values of that categorical that are in the same ratio as in the parent population from which the samples were obtained. Using the same example as above, suppose your data has sub-populations by states, say NJ has 100 observations, NY has 60 observations, and WA has 300 observations. If you specify the rate of stratified sampling to be 0.5, then the sample obtained should have approximately 50, 30, and 150 observations of NJ, NY, and WA respectively.

Here is an example query:

```
SET sampleRate=<sample rate, 0-1>;
select
  field1, field2, field3, ..., fieldN, state
from
(
  (
    select
      field1, field2, field3, ..., fieldN, state,
      count(*) over (partition by state) as state_cnt,
      rank() over (partition by state order by rand()) as state_rank
    from <table name>
  )a
  where state_rank <= state_cnt*${hiveconf:sampleRate}'
```

For information on more advanced sampling methods that are available in Hive, see [LanguageManual Sampling](#).

Access datasets with Python using the Azure Machine Learning Python client library

1/17/2017 • 8 min to read • [Edit on GitHub](#)

The preview of Microsoft Azure Machine Learning Python client library can enable secure access to your Azure Machine Learning datasets from a local Python environment and enables the creation and management of datasets in a workspace.

This topic provides instructions on how to:

- install the Machine Learning Python client library
- access and upload datasets, including instructions on how to get authorization to access Azure Machine Learning datasets from your local Python environment
- access intermediate datasets from experiments
- use the Python client library to enumerate datasets, access metadata, read the contents of a dataset, create new datasets and update existing datasets

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

Prerequisites

The Python client library has been tested under the following environments:

- Windows, Mac and Linux
- Python 2.7, 3.3 and 3.4

It has a dependency on the following packages:

- requests
- python-dateutil
- pandas

We recommend using a Python distribution such as [Anaconda](#) or [Canopy](#), which come with Python, IPython and the three packages listed above installed. Although IPython is not strictly required, it is a great environment for manipulating and visualizing data interactively.

How to install the Azure Machine Learning Python client library

The Azure Machine Learning Python client library must also be installed to complete the tasks outlined in this topic. It is available from the [Python Package Index](#). To install it in your Python environment, run the following command from your local Python environment:

```
pip install azureml
```

Alternatively, you can download and install from the sources on [github](#).

```
python setup.py install
```

If you have git installed on your machine, you can use pip to install directly from the git repository:

```
pip install git+https://github.com/Azure/Azure-MachineLearning-ClientLibrary-Python.git
```

Use Studio Code snippets to access datasets

The Python client library gives you programmatic access to your existing datasets from experiments that have been run.

From the Studio web interface, you can generate code snippets that include all the necessary information to download and deserialize datasets as Pandas DataFrame objects on your location machine.

Security for data access

The code snippets provided by Studio for use with the Python client library includes your workspace id and authorization token. These provide full access to your workspace and must be protected, like a password.

For security reasons, the code snippet functionality is only available to users that have their role set as **Owner** for the workspace. Your role is displayed in Azure Machine Learning Studio on the **USERS** page under **Settings**.

The screenshot shows the 'settings' page in the Azure Machine Learning Studio. On the left, there's a sidebar with icons for EXPERIMENTS, WEB SERVICES, MODULES, DATASETS, TRAINED MODELS, and SETTINGS. The SETTINGS icon is highlighted with a blue bar at the bottom. The main area has a title 'settings' and tabs for NAME, AUTHORIZATION TOKENS, and USERS. The USERS tab is selected, showing a table with columns: NAME, EMAIL, ROLE, and STATUS. One row is visible, showing 'Owner' in the ROLE column and 'Active' in the STATUS column. There's also a small gear icon in the top right corner of the table.

If your role is not set as **Owner**, you can either request to be reinvited as an owner, or ask the owner of the workspace to provide you with the code snippet.

To obtain the authorization token, you can do one of the following:

- Ask for a token from an owner. Owners can access their authorization tokens from the Settings page of their workspace in Studio. Select **Settings** from the left pane and click **AUTHORIZATION TOKENS** to see the primary and secondary tokens. Although either the primary or the secondary authorization tokens can be used in the code snippet, it is recommended that owners only share the secondary authorization tokens.

The screenshot shows the 'settings' page in the Azure Machine Learning Studio with the 'SETTINGS' icon in the sidebar highlighted. The 'AUTHORIZATION TOKENS' tab is selected. It displays two fields: 'PRIMARY AUTHORIZATION TOKEN' and 'SECONDARY AUTHORIZATION TOKEN'. Both fields contain blacked-out text, and each has a 'Regenerate' button to its right. The 'SECONDARY AUTHORIZATION TOKEN' field is circled in red.

- Ask to be promoted to role of owner. To do this, a current owner of the workspace needs to first remove you from the workspace then re-invite you to it as an owner.

Once developers have obtained the workspace id and authorization token, they are able to access the workspace

using the code snippet regardless of their role.

Authorization tokens are managed on the **AUTHORIZATION TOKENS** page under **SETTINGS**. You can regenerate them, but this procedure revokes access to the previous token.

Access datasets from a local Python application

1. In Machine Learning Studio, click **DATASETS** in the navigation bar on the left.
2. Select the dataset you would like to access. You can select any of the datasets from the **MY DATASETS** list or from the **SAMPLES** list.
3. From the bottom toolbar, click **Generate Data Access Code**. If the data is in a format incompatible with the Python client library, this button is disabled.

The screenshot shows the Azure Machine Learning Studio interface. On the left, there is a vertical navigation bar with icons for EXPERIMENTS, WEB SERVICES, MODULES, DATASETS (which is selected and highlighted in blue), TRAINED MODELS, and SETTINGS. Below this is a 'NEW' button. The main area is titled 'datasets' and contains two tabs: 'MY DATASETS' and 'SAMPLES'. Under 'MY DATASETS', there is a table with one row:

NAME	SUBMITTED BY	DESCRIPTION	DATA TYPE	CREA... ↓	🔍
My Data.tsv	ptvsazure	My Test Data	GenericTSV	1/20/2015 12:3...	

At the bottom of the screen, there is a toolbar with icons for DOWNLOAD, DELETE, and GENERATE DATA ACCESS CODE, with the latter being highlighted with a red box.

4. Select the code snippet from the window that appears and copy it to your clipboard.

The screenshot shows a modal dialog box titled 'GENERATE DATA ACCESS CODE'. It contains the following text:
Use this code to access your data
To programmatically access this dataset, copy the code snippet into your favorite development environment. [Learn More](#).
Note: this code includes your workspace access token, which provides full access to your workspace. It should be treated like a password.
CODE SNIPPET
Python
from azureml import Workspace
ws = Workspace(
 workspace_id='[REDACTED]',
 authorization_token='[REDACTED]',
)
ds = ws.datasets['My Data.tsv']
frame = ds.to_dataframe()

 USE SECONDARY TOKEN
A checkmark icon is located at the bottom right of the dialog.

5. Paste the code into the notebook of your local Python application.

The screenshot shows an IPython Notebook interface. In the top bar, it says "IP[y]: Notebook My Test Notebook Last Checkpoint: Jan 20 12:58 (autosaved)". Below the toolbar, there are two code cells:

```
In [3]: from azureml import Workspace
ws = Workspace(
    workspace_id='[REDACTED]',
    authorization_token='[REDACTED]')
ds = ws.datasets['My Data.tsv']
frame = ds.to_dataframe()
```

```
In [4]: frame
```

The output for cell [4] is a table titled "Out[4]:" showing 10 rows of data. The columns are labeled fLength, fWidth, fSize, fConc, fConcl, fAsym, fM3Long, fM3Trans, fAlpha, fDist, and Class. The data is as follows:

	fLength	fWidth	fSize	fConc	fConcl	fAsym	fM3Long	fM3Trans	fAlpha	fDist	Class
0	29.4491	12.7271	2.6637	0.3536	0.1876	-19.4070	-18.1295	7.1258	8.1501	252.1500	g
1	51.5830	10.7969	2.6222	0.5227	0.2733	-64.0583	-30.1280	4.3855	15.0428	269.4810	g
2	36.3558	10.3843	2.8531	0.5309	0.3599	15.4179	51.9328	16.2848	9.1263	208.7935	h
3	37.2577	12.0793	2.4354	0.3560	0.2037	5.1882	-17.8545	6.0370	30.0150	61.1727	h
4	34.8906	15.7072	2.7147	0.3587	0.1938	-8.5682	-12.6514	-14.3676	89.7920	222.4690	h
5	60.4957	17.6753	2.9380	0.1753	0.0894	31.3257	-15.9801	14.4117	15.6390	113.1820	g
6	18.5731	16.2365	2.6758	0.4895	0.3091	1.1158	8.1104	-11.7988	50.9791	224.8780	h
7	21.5929	12.4539	2.3512	0.4900	0.3096	0.1172	-4.3583	2.5525	58.3290	52.0772	g
8	45.5590	9.9957	2.5809	0.3885	0.1955	17.0284	34.9608	-7.8856	14.4173	177.6820	g
9	62.3597	21.6946	3.1739	0.3087	0.2041	-45.3412	52.1023	22.5905	36.9876	274.7409	h

Access intermediate datasets from Machine Learning experiments

After an experiment is run in the Machine Learning Studio, it is possible to access the intermediate datasets from the output nodes of modules. Intermediate datasets are data that has been created and used for intermediate steps when a model tool has been run.

Intermediate datasets can be accessed as long as the data format is compatible with the Python client library.

The following formats are supported (constants for these are in the `azureml.DataTypeIds` class):

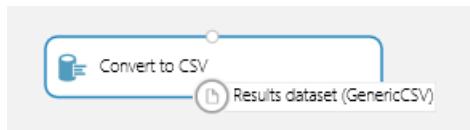
- PlainText
- GenericCSV
- GenericTSV
- GenericCSVNoHeader
- GenericTSVNoHeader

You can determine the format by hovering over a module output node. It is displayed along with the node name, in a tooltip.

Some of the modules, such as the [Split](#) module, output to a format named `[Dataset]`, which is not supported by the Python client library.

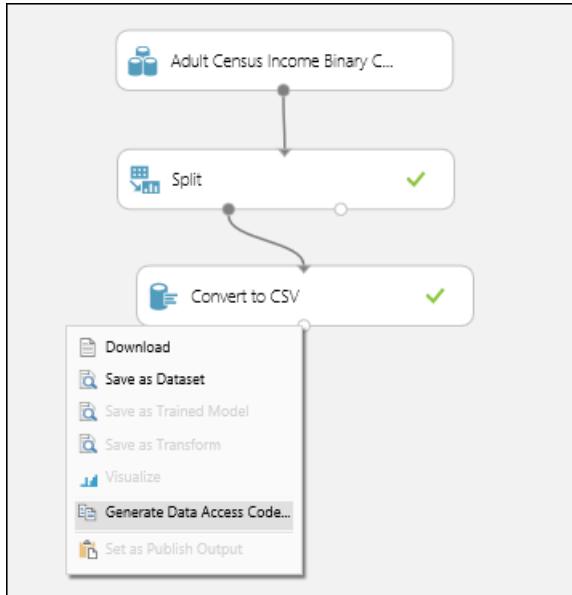


You need to use a conversion module, such as [Convert to CSV](#), to get an output into a supported format.

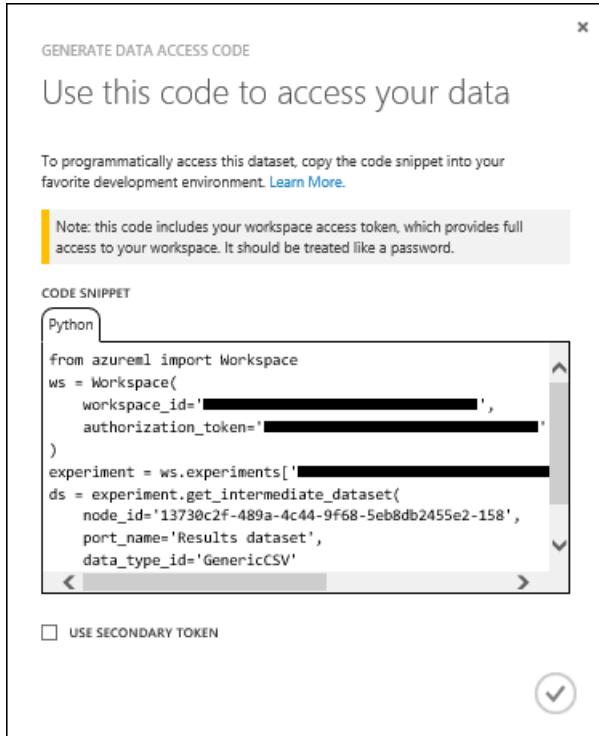


The following steps show an example that creates an experiment, runs it and accesses the intermediate dataset.

1. Create a new experiment.
2. Insert an **Adult Census Income Binary Classification dataset** module.
3. Insert a **Split** module, and connect its input to the dataset module output.
4. Insert a **Convert to CSV** module and connect its input to one of the **Split** module outputs.
5. Save the experiment, run it, and wait for it to finish running.
6. Click the output node on the **Convert to CSV** module.
7. When the context menu appears, select **Generate Data Access Code**.



8. Select the code snippet and copy it to your clipboard from the window that appears.



9. Paste the code in your notebook.

IP[y]: Notebook My Test Notebook Last Checkpoint: Jan 20 12:58 (unsaved changes)

File Edit View Insert Cell Kernel Help

Cell Toolbar: None

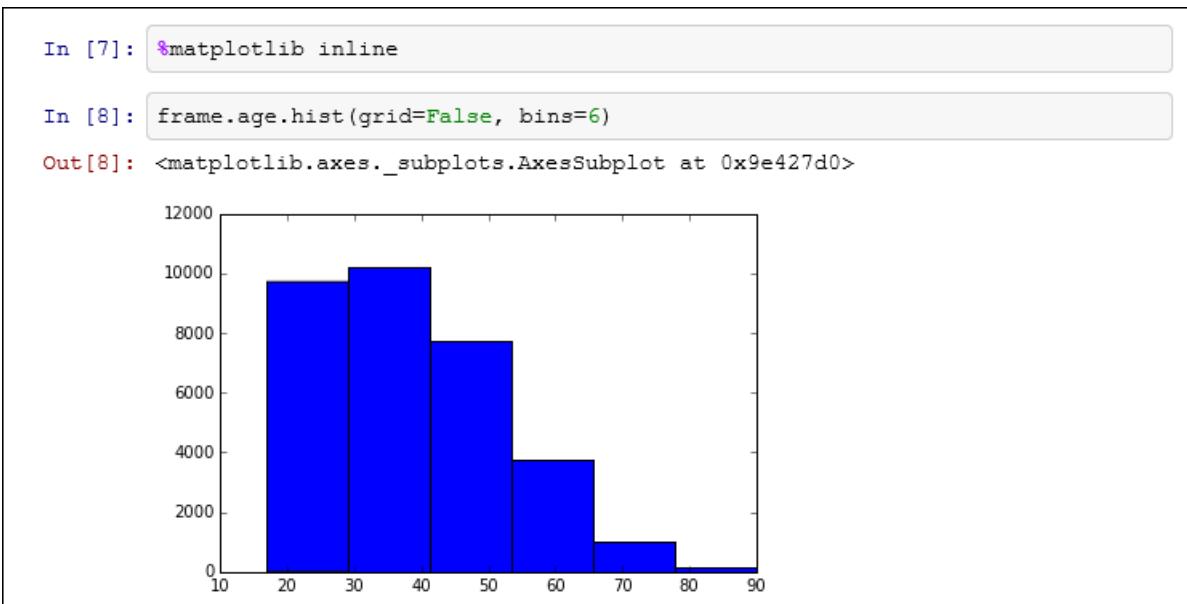
```
In [6]: from azureml import Workspace
ws = Workspace(
    workspace_id='[REDACTED]',
    authorization_token='[REDACTED]'
)
experiment = ws.experiments['[REDACTED]']
ds = experiment.get_intermediate_dataset(
    node_id='13730c2f-4b9a-4c44-9f68-5eb8db2455e2-158',
    port_name='Results dataset',
    data_type_id='GenericCSV'
)
frame = ds.to_dataframe()
```

```
In [7]: frame
```

```
Out[7]:
```

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss
0	52	Private	225317	5th-6th	3	Married-civ-spouse	Craft-repair	Husband	White	Male	0	0
1	29	Private	154017	HS-grad	9	Never-married	Sales	Not-in-family	White	Female	0	0
2	25	Private	203570	HS-grad	9	Separated	Other-service	Unmarried	Black	Male	0	0

10. You can visualize the data using matplotlib. This displays in a histogram for the age column:



Use the Machine Learning Python client library to access, read, create, and manage datasets

Workspace

The workspace is the entry point for the Python client library. Provide the `Workspace` class with your workspace id and authorization token to create an instance:

```
ws = Workspace(workspace_id='4c29e1adeba2e5a7cbeb0e4f4adfb4df',
               authorization_token='f4f3ade2c6aefdb1afb043cd8bcf3daf')
```

Enumerate datasets

To enumerate all datasets in a given workspace:

```
for ds in ws.datasets:  
    print(ds.name)
```

To enumerate just the user-created datasets:

```
for ds in ws.user_datasets:  
    print(ds.name)
```

To enumerate just the example datasets:

```
for ds in ws.example_datasets:  
    print(ds.name)
```

You can access a dataset by name (which is case-sensitive):

```
ds = ws.datasets['my dataset name']
```

Or you can access it by index:

```
ds = ws.datasets[0]
```

Metadata

Datasets have metadata, in addition to content. (Intermediate datasets are an exception to this rule and do not have any metadata.)

Some metadata values are assigned by the user at creation time:

```
print(ds.name)  
print(ds.description)  
print(ds.family_id)  
print(ds.data_type_id)
```

Others are values assigned by Azure ML:

```
print(ds.id)  
print(ds.created_date)  
print(ds.size)
```

See the `SourceDataset` class for more on the available metadata.

Read contents

The code snippets provided by Machine Learning Studio automatically download and deserialize the dataset to a Pandas DataFrame object. This is done with the `to_dataframe` method:

```
frame = ds.to_dataframe()
```

If you prefer to download the raw data, and perform the deserialization yourself, that is an option. At the moment, this is the only option for formats such as 'ARFF', which the Python client library cannot deserialize.

To read the contents as text:

```
text_data = ds.read_as_text()
```

To read the contents as binary:

```
binary_data = ds.read_as_binary()
```

You can also just open a stream to the contents:

```
with ds.open() as file:  
    binary_data_chunk = file.read(1000)
```

Create a new dataset

The Python client library allows you to upload datasets from your Python program. These datasets are then available for use in your workspace.

If you have your data in a Pandas DataFrame, use the following code:

```
from azureml import DataTypeIds  
  
dataset = ws.datasets.add_from_dataframe(  
    dataframe=frame,  
    data_type_id=DataTypeIds.GenericCSV,  
    name='my new dataset',  
    description='my description'  
)
```

If your data is already serialized, you can use:

```
from azureml import DataTypeIds  
  
dataset = ws.datasets.add_from_raw_data(  
    raw_data=raw_data,  
    data_type_id=DataTypeIds.GenericCSV,  
    name='my new dataset',  
    description='my description'  
)
```

The Python client library is able to serialize a Pandas DataFrame to the following formats (constants for these are in the `azureml.DataTypeIds` class):

- PlainText
- GenericCSV
- GenericTSV
- GenericCSVNoHeader
- GenericTSVNoHeader

Update an existing dataset

If you try to upload a new dataset with a name that matches an existing dataset, you should get a conflict error.

To update an existing dataset, you first need to get a reference to the existing dataset:

```
dataset = ws.datasets['existing dataset']  
  
print(dataset.data_type_id) # 'GenericCSV'  
print(dataset.name) # 'existing dataset'  
print(dataset.description) # 'data up to jan 2015'
```

Then use `update_from_dataframe` to serialize and replace the contents of the dataset on Azure:

```
dataset = ws.datasets['existing dataset']

dataset.update_from_dataframe(frame2)

print(dataset.data_type_id) # 'GenericCSV'
print(dataset.name)      # 'existing dataset'
print(dataset.description) # 'data up to jan 2015'
```

If you want to serialize the data to a different format, specify a value for the optional `data_type_id` parameter.

```
from azureml import DataTypeIds

dataset = ws.datasets['existing dataset']

dataset.update_from_dataframe(
    dataframe=frame2,
    data_type_id=DataTypeIds.GenericTSV,
)

print(dataset.data_type_id) # 'GenericTSV'
print(dataset.name)      # 'existing dataset'
print(dataset.description) # 'data up to jan 2015'
```

You can optionally set a new description by specifying a value for the `description` parameter.

```
dataset = ws.datasets['existing dataset']

dataset.update_from_dataframe(
    dataframe=frame2,
    description='data up to feb 2015',
)

print(dataset.data_type_id) # 'GenericCSV'
print(dataset.name)      # 'existing dataset'
print(dataset.description) # 'data up to feb 2015'
```

You can optionally set a new name by specifying a value for the `name` parameter. From now on, you'll retrieve the dataset using the new name only. The following code updates the data, name and description.

```
dataset = ws.datasets['existing dataset']

dataset.update_from_dataframe(
    dataframe=frame2,
    name='existing dataset v2',
    description='data up to feb 2015',
)

print(dataset.data_type_id)      # 'GenericCSV'
print(dataset.name)            # 'existing dataset v2'
print(dataset.description)     # 'data up to feb 2015'

print(ws.datasets['existing dataset v2'].name) # 'existing dataset v2'
print(ws.datasets['existing dataset'].name) # IndexError
```

The `data_type_id`, `name` and `description` parameters are optional and default to their previous value. The `dataframe` parameter is always required.

If your data is already serialized, use `update_from_raw_data` instead of `update_from_dataframe`. If you just pass in `raw_data` instead of `dataframe`, it works in a similar way.

Process Azure blob data with advanced analytics

1/17/2017 • 3 min to read • [Edit on GitHub](#)

This document covers exploring data and generating features from data stored in Azure Blob storage.

Load the data into a Pandas data frame

In order to explore and manipulate a dataset, it must be downloaded from the blob source to a local file which can then be loaded in a Pandas data frame. Here are the steps to follow for this procedure:

1. Download the data from Azure blob with the following sample Python code using blob service. Replace the variable in the code below with your specific values:

```
from azure.storage.blob import BlobService  
import tables  
  
STORAGEACCOUNTNAME=<storage_account_name>  
STORAGEACCOUNTKEY=<storage_account_key>  
LOCALFILENAME=<local_file_name>  
CONTAINERNAME=<container_name>  
BLOBNAME=<blob_name>  
  
#download from blob  
t1=time.time()  
blob_service=BlobService(account_name=STORAGEACCOUNTNAME,account_key=STORAGEACCOUNTKEY)  
blob_service.get_blob_to_path(CONTAINERNAME,BLOBNAME,LOCALFILENAME)  
t2=time.time()  
print("It takes %s seconds to download "+blobname) % (t2 - t1)
```

2. Read the data into a Pandas data-frame from the downloaded file.

```
#LOCALFILE is the file path  
dataframe_blobdata = pd.read_csv(LOCALFILE)
```

Now you are ready to explore the data and generate features on this dataset.

Data Exploration

Here are a few examples of ways to explore data using Pandas:

1. Inspect the number of rows and columns

```
print 'the size of the data is: %d rows and %d columns' % dataframe_blobdata.shape
```

2. Inspect the first or last few rows in the dataset as below:

```
dataframe_blobdata.head(10)  
dataframe_blobdata.tail(10)
```

3. Check the data type each column was imported as using the following sample code

```
for col in dataframe_blobdata.columns:  
    print dataframe_blobdata[col].name, '\t', dataframe_blobdata[col].dtype
```

4. Check the basic stats for the columns in the data set as follows

```
dataframe_blobdata.describe()
```

5. Look at the number of entries for each column value as follows

```
dataframe_blobdata['<column_name>'].value_counts()
```

6. Count missing values versus the actual number of entries in each column using the following sample code

```
miss_num = dataframe_blobdata.shape[0] - dataframe_blobdata.count()  
print miss_num
```

7. If you have missing values for a specific column in the data, you can drop them as follows:

```
dataframe_blobdata_noNA = dataframe_blobdata.dropna() dataframe_blobdata_noNA.shape
```

Another way to replace missing values is with the mode function:

```
dataframe_blobdata_mode = dataframe_blobdata.fillna({'<column_name>':dataframe_blobdata['<column_name>'].mode()[0]})
```

8. Create a histogram plot using variable number of bins to plot the distribution of a variable

```
dataframe_blobdata['<column_name>'].value_counts().plot(kind='bar')  
  
np.log(dataframe_blobdata['<column_name>']+1).hist(bins=50)
```

9. Look at correlations between variables using a scatterplot or using the built-in correlation function

```
#relationship between column_a and column_b using scatter plot  
plt.scatter(dataframe_blobdata['<column_a>'], dataframe_blobdata['<column_b>'])  
  
#correlation between column_a and column_b  
dataframe_blobdata[['<column_a>', '<column_b>']].corr()
```

Feature Generation

We can generate features using Python as follows:

Indicator value based Feature Generation

Categorical features can be created as follows:

1. Inspect the distribution of the categorical column:

```
dataframe_blobdata['<categorical_column>'].value_counts()
```

2. Generate indicator values for each of the column values

```
#generate the indicator column  
dataframe_blobdata_identity = pd.get_dummies(dataframe_blobdata['<categorical_column>'], prefix='<categorical_column>_identity')
```

3. Join the indicator column with the original data frame

```
#Join the dummy variables back to the original data frame  
dataframe_blobdata_with_identity = dataframe_blobdata.join(dataframe_blobdata_identity)
```

4. Remove the original variable itself:

```
#Remove the original column rate_code in df1_with_dummy  
dataframe_blobdata_with_identity.drop('<categorical_column>', axis=1, inplace=True)
```

Binning Feature Generation

For generating binned features, we proceed as follows:

1. Add a sequence of columns to bin a numeric column

```
bins =[0, 1, 2, 4, 10, 40]  
dataframe_blobdata_bin_id = pd.cut(dataframe_blobdata['<numeric_column>'], bins)
```

2. Convert binning to a sequence of boolean variables

```
dataframe_blobdata_bin_bool = pd.get_dummies(dataframe_blobdata_bin_id, prefix='<numeric_column>')
```

3. Finally, Join the dummy variables back to the original data frame

```
dataframe_blobdata_with_bin_bool = dataframe_blobdata.join(dataframe_blobdata_bin_bool)
```

Writing data back to Azure blob and consuming in Azure Machine Learning

After you have explored the data and created the necessary features, you can upload the data (sampled or featurized) to an Azure blob and consume it in Azure Machine Learning using the following steps: Note that additional features can be created in the Azure Machine Learning Studio as well.

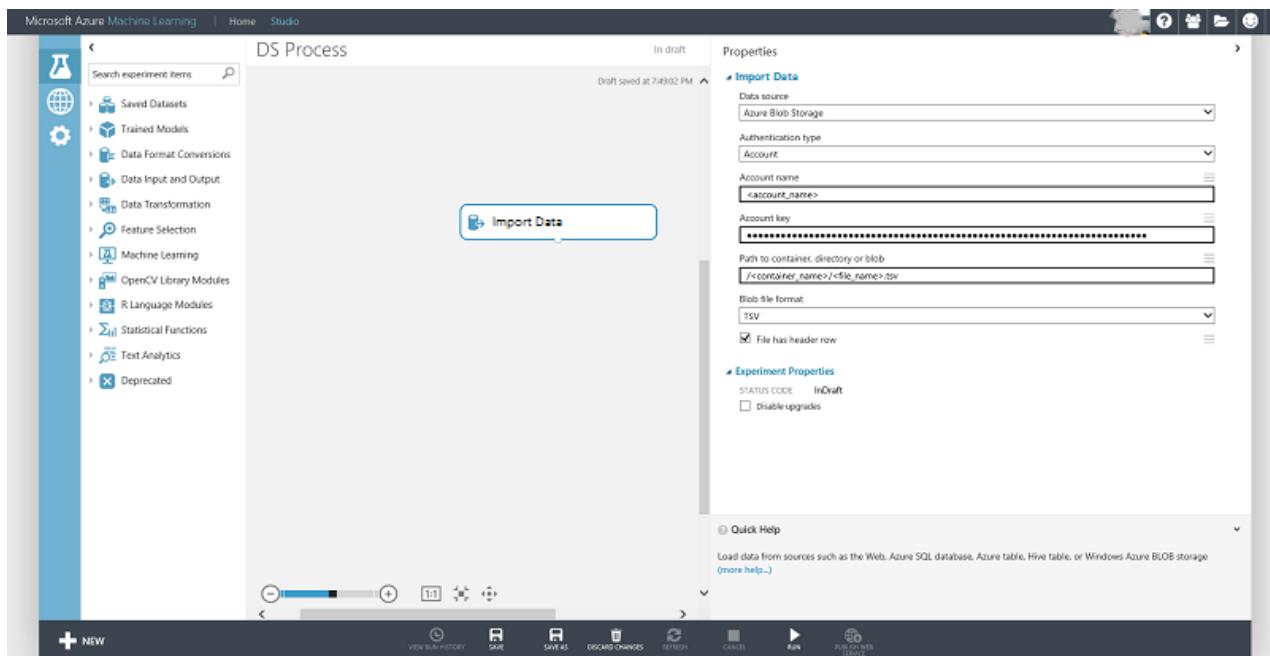
1. Write the data frame to local file

```
dataframe.to_csv(os.path.join(os.getcwd(),LOCALFILENAME), sep='\t', encoding='utf-8', index=False)
```

2. Upload the data to Azure blob as follows:

```
from azure.storage.blob import BlobService  
import tables  
  
STORAGEACCOUNTNAME=<storage_account_name>  
LOCALFILENAME=<local_file_name>  
STORAGEACCOUNTKEY=<storage_account_key>  
CONTAINERNAME=<container_name>  
BLOBNAME=<blob_name>  
  
output_blob_service=BlobService(account_name=STORAGEACCOUNTNAME,account_key=STORAGEACCOUNTKEY)  
localfileprocessed = os.path.join(os.getcwd(),LOCALFILENAME) #assuming file is in current working directory  
  
try:  
  
    #perform upload  
    output_blob_service.put_block_blob_from_path(CONTAINERNAME,BLOBNAME,localfileprocessed)  
  
except:  
    print ("Something went wrong with uploading blob:"+BLOBNAME)
```

3. Now the data can be read from the blob using the Azure Machine Learning Import Data module as shown in the screen below:



Scalable Data Science in Azure Data Lake: An end-to-end Walkthrough

1/17/2017 • 20 min to read • [Edit on GitHub](#)

This walkthrough shows how to use Azure Data Lake to do data exploration and binary classification tasks on a sample of the NYC taxi trip and fare dataset to predict whether or not a tip will be paid by a fare. It walks you through the steps of the [Team Data Science Process](#), end-to-end, from data acquisition to model training, and then to the deployment of a web service that publishes the model.

Azure Data Lake Analytics

The [Microsoft Azure Data Lake](#) has all the capabilities required to make it easy for data scientists to store data of any size, shape and speed, and to conduct data processing, advanced analytics, and machine learning modeling with high scalability in a cost-effective way. You pay on a per-job basis, only when data is actually being processed. Azure Data Lake Analytics includes U-SQL, a language that blends the declarative nature of SQL with the expressive power of C# to provide scalable distributed query capability. It enables you to process unstructured data by applying schema on read, insert custom logic and user defined functions (UDFs), and includes extensibility to enable fine grained control over how to execute at scale. To learn more about the design philosophy behind U-SQL, see [Visual Studio blog post](#).

Data Lake Analytics is also a key part of Cortana Analytics Suite and works with Azure SQL Data Warehouse, Power BI, and Data Factory. This gives you a complete cloud big data and advanced analytics platform.

This walkthrough begins by describing the prerequisites and resources that are needed to complete the tasks with Data Lake Analytics that form the data science process and how to install them. Then it outlines the data processing steps using U-SQL and concludes by showing how to use Python and Hive with Azure Machine Learning Studio to build and deploy the predictive models.

U-SQL and Visual Studio

This walkthrough recommends using Visual Studio to edit U-SQL scripts to process the dataset. The U-SQL scripts are described here and provided in a separate file. The process includes ingesting, exploring, and sampling the data. It also shows how to run a U-SQL scripted job from the Azure portal. Hive tables are created for the data in an associated HDInsight cluster to facilitate the building and deployment of a binary classification model in Azure Machine Learning Studio.

Python

This walkthrough also contains a section that shows how to build and deploy a predictive model using Python with Azure Machine Learning Studio. We provide a Jupyter notebook with the Python scripts for these steps in this process. The notebook includes code for some additional feature engineering steps and models construction such as multiclass classification and regression modeling in addition to the binary classification model outlined here. The regression task is to predict the amount of the tip based on other tip features.

Azure Machine Learning

Azure Machine Learning Studio is used to build and deploy the predictive models. This is done using two approaches: first with Python scripts and then with Hive tables on an HDInsight (Hadoop) cluster.

Scripts

Only the principal steps are outlined in this walkthrough. You can download the full **U-SQL script** and **Jupyter Notebook** from [GitHub](#).

Prerequisites

Before you begin these topics, you must have the following:

- An Azure subscription. If you do not already have one, see [Get Azure free trial](#).
- [Recommended] Visual Studio 2013, or 2015. If you do not already have one of these versions installed, you can download a free Community edition from [here](#). Click on the **Download Community 2015** button under the Visual Studio section.

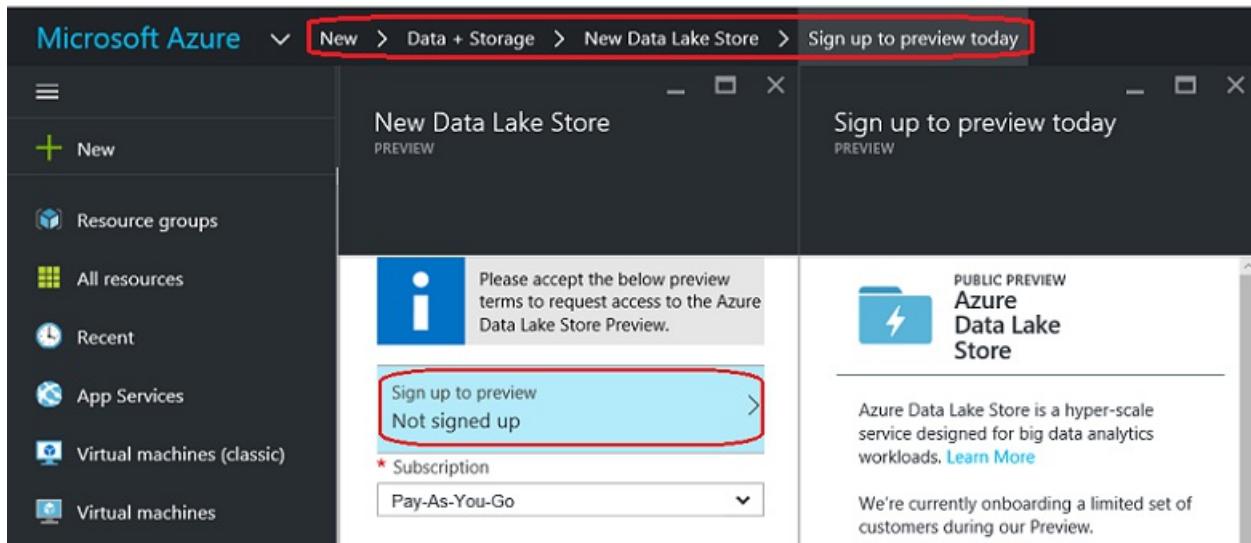
NOTE

Instead of Visual Studio, you can also use the Azure Portal to submit Azure Data Lake queries. We will provide instructions on how to do so both with Visual Studio and on the portal in the section titled **Process data with U-SQL**.

- Signup for Azure Data Lake Preview

NOTE

You need to get approval to use Azure Data Lake Store (ADLS) and Azure Data Lake Analytics (ADLA) as these services are in preview. You will be prompted to sign up when you create your first ADLS or ADLA. To sign up, click on **Sign up to preview**, read the agreement, and click **OK**. Here, for example, is the ADLS sign up page:



Prepare data science environment for Azure Data Lake

To prepare the data science environment for this walkthrough, create the following resources:

- Azure Data Lake Store (ADLS)
- Azure Data Lake Analytics (ADLA)
- Azure Blob storage account
- Azure Machine Learning Studio account
- Azure Data Lake Tools for Visual Studio (Recommended)

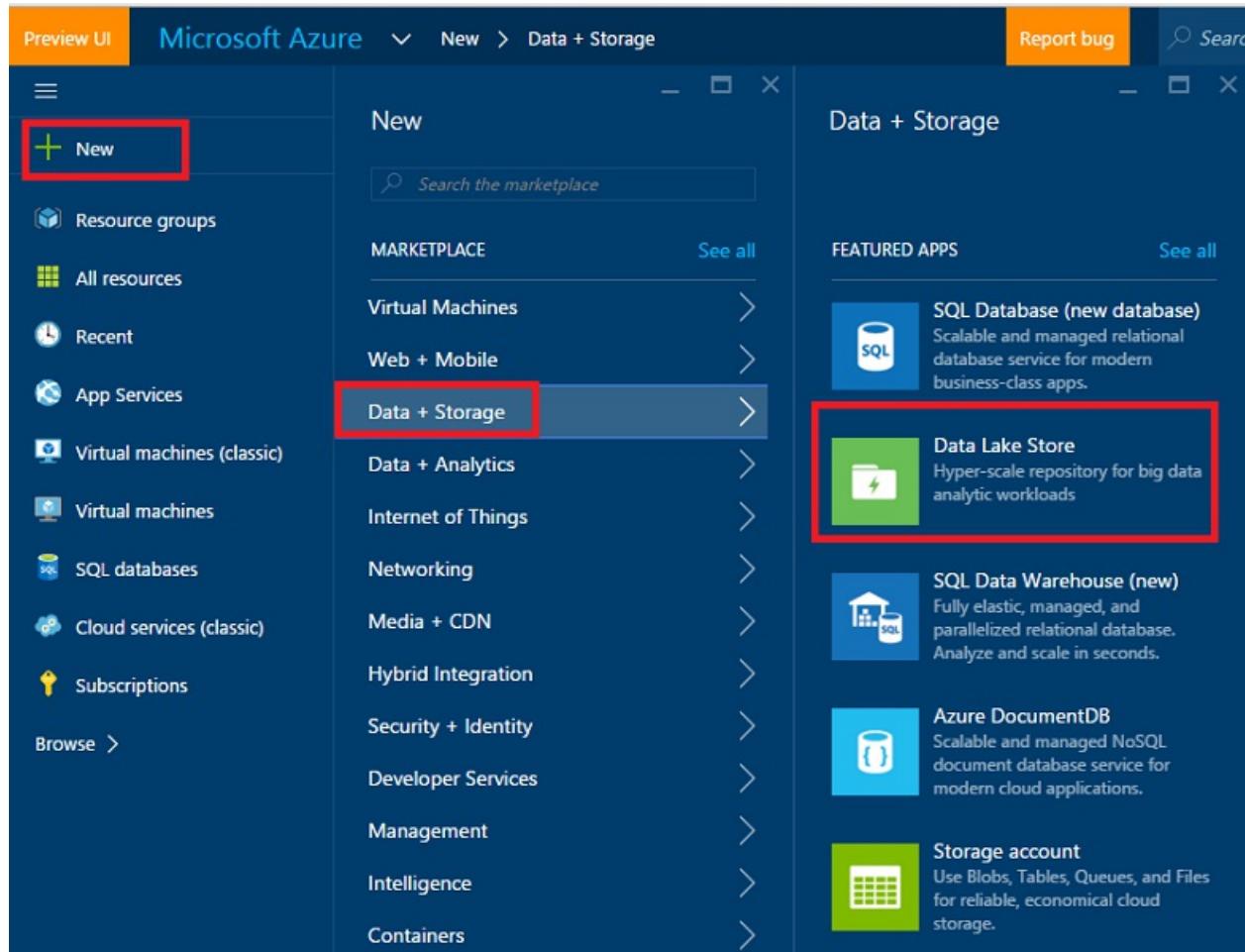
This section provides instructions on how to create each of these resources. If you choose to use Hive tables with Azure Machine Learning, instead of Python, to build a model, you will also need to provision an HDInsight (Hadoop) cluster. This alternative procedure is described in the appropriate section below.

AZURE.NOTE The **Azure Data Lake Store** can be created either separately or when you create the **Azure Data Lake Analytics** as the default storage. Instructions are referenced for creating each of these resources

separately below, but the Data Lake storage account need not be created separately.

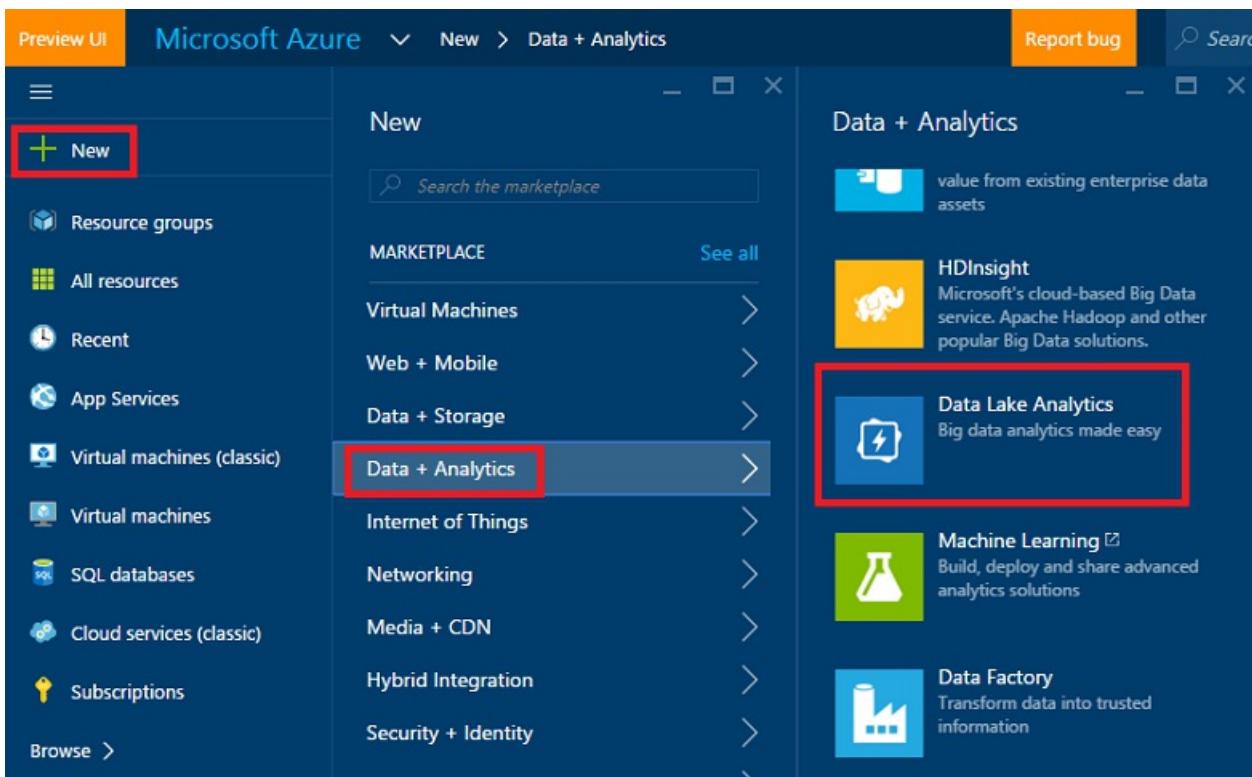
Create an Azure Data Lake Store

Create an ADLS from the [Azure Portal](#). For details, see [Create an HDInsight cluster with Data Lake Store using Azure Portal](#). Be sure to set up the Cluster AAD Identity in the **DataSource** blade of the **Optional Configuration** blade described there.



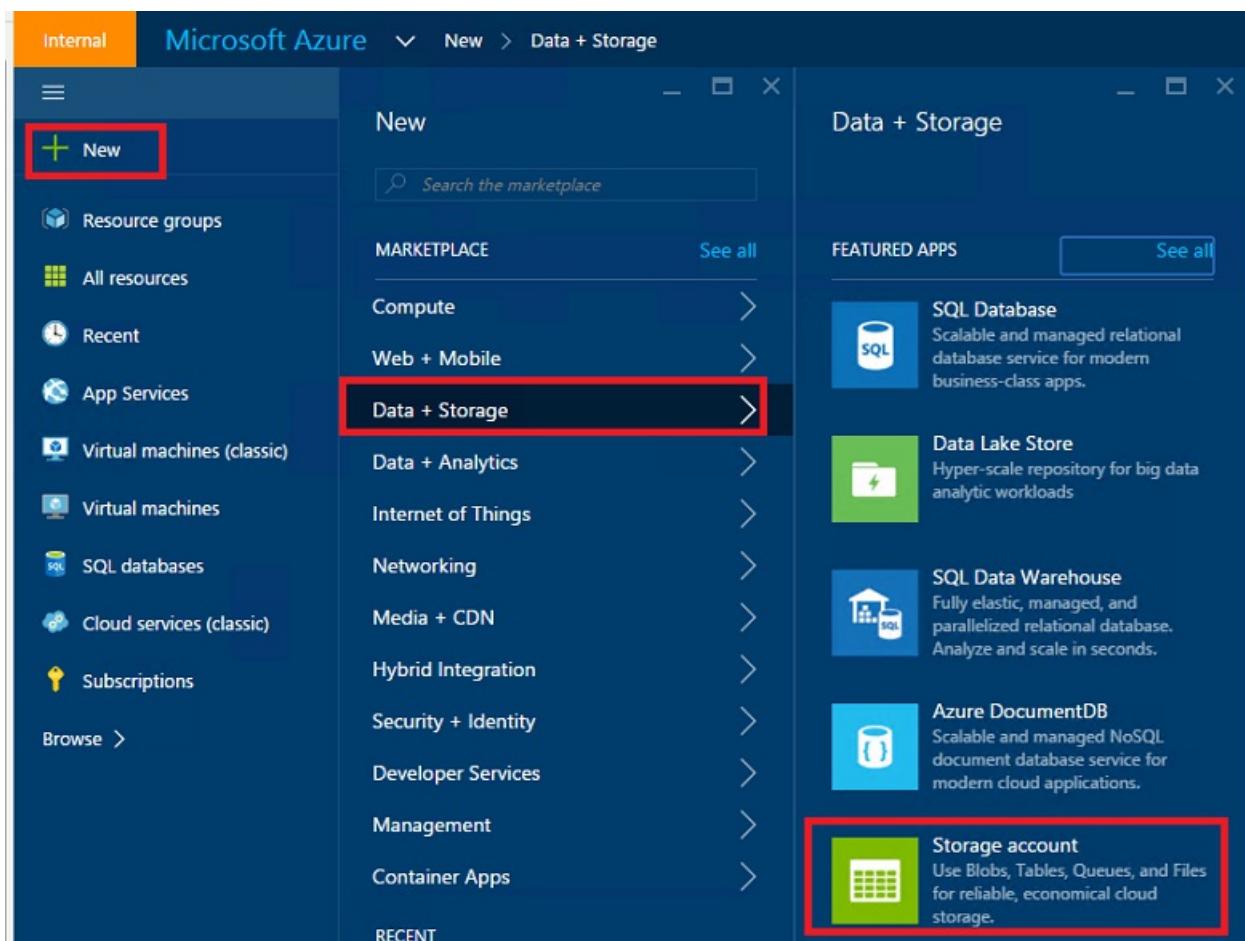
Create an Azure Data Lake Analytics account

Create an ADLA account from the [Azure Portal](#). For details, see [Tutorial: get started with Azure Data Lake Analytics using Azure Portal](#).



Create an Azure Blob storage account

Create an Azure Blob storage account from the [Azure Portal](#). For details, see the Create a storage account section in [About Azure storage accounts](#).

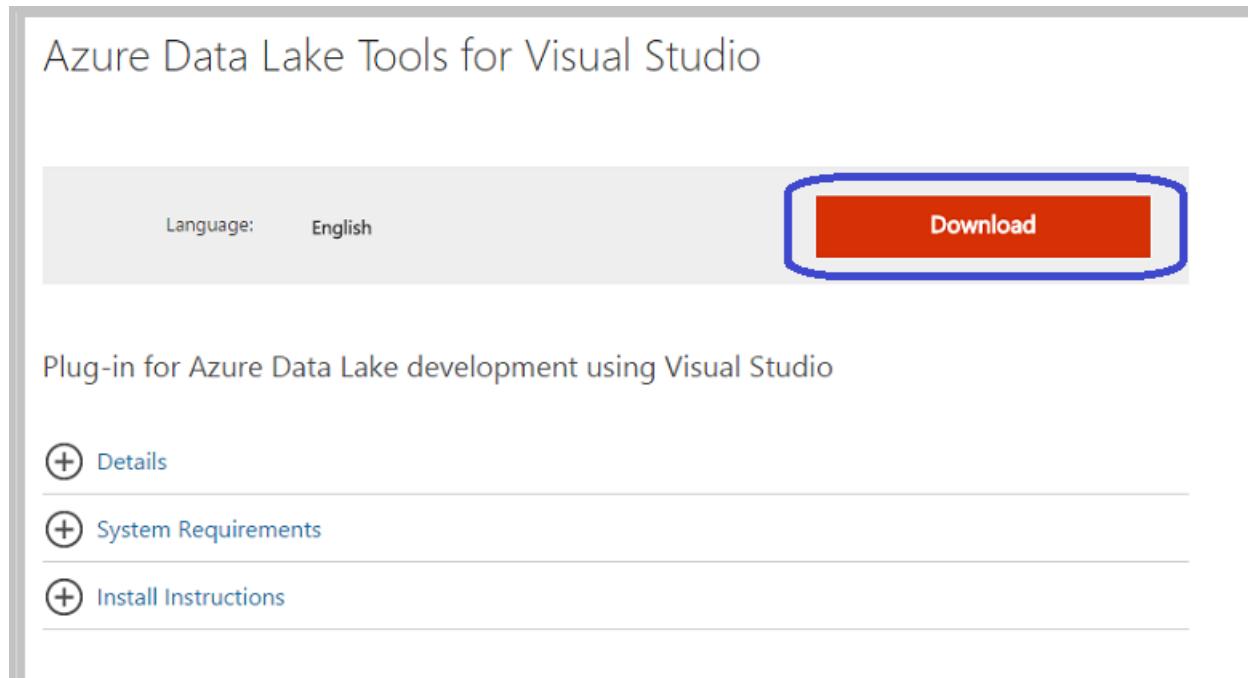


Set up an Azure Machine Learning Studio account

Sign up/into Azure Machine Learning Studio from the [Azure Machine Learning](#) page. Click on the **Get started now** button and then choose a "Free Workspace" or "Standard Workspace". After this you will be able to create experiments in Azure ML Studio.

Install Azure Data Lake Tools [Recommended]

Install Azure Data Lake Tools for your version of Visual Studio from [Azure Data Lake Tools for Visual Studio](#).



Azure Data Lake Tools for Visual Studio

Language: English

Download

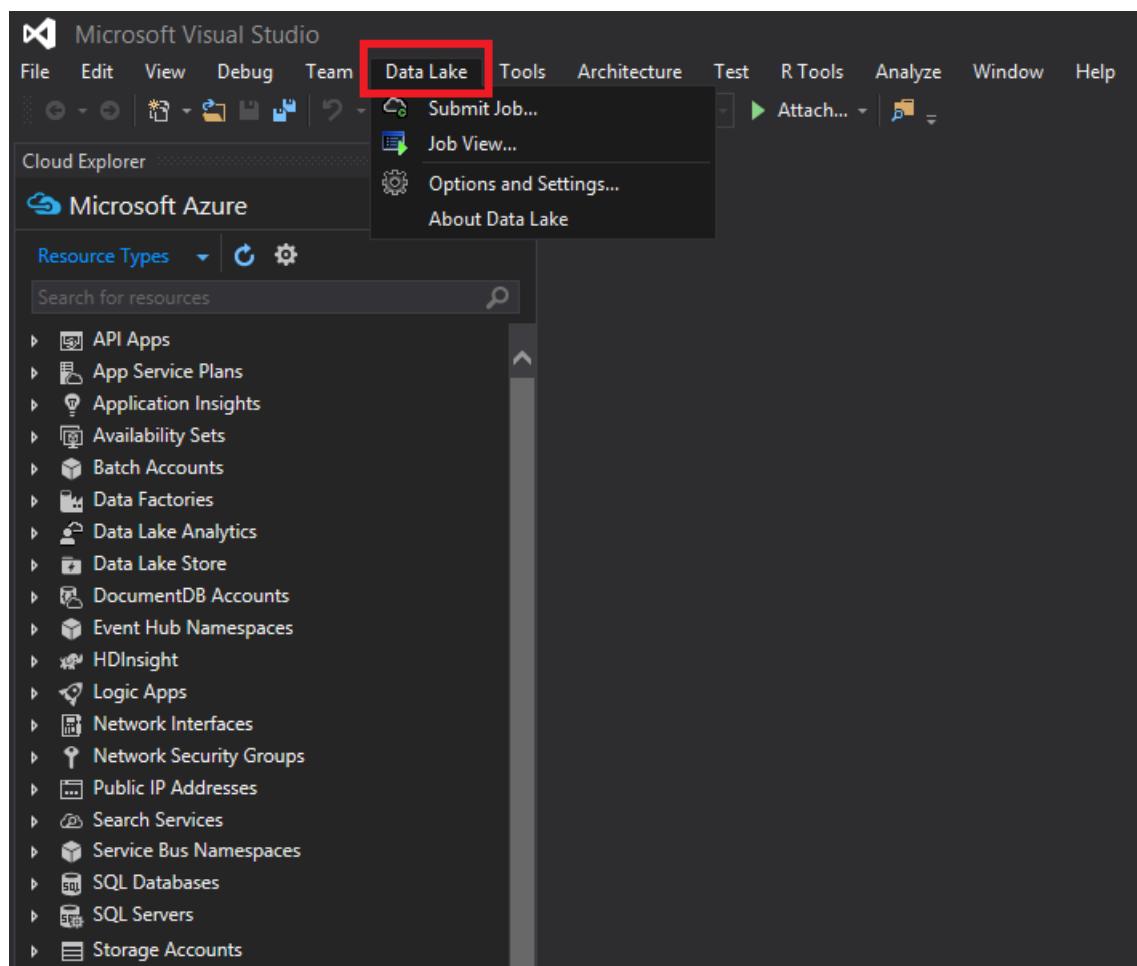
Plug-in for Azure Data Lake development using Visual Studio

⊕ Details

⊕ System Requirements

⊕ Install Instructions

After the installation finishes successfully, open up Visual Studio. You should see the Data Lake tab the menu at the top. Your Azure resources should appear in the left panel when you sign into your Azure account.



The NYC Taxi Trips dataset

The data set we used here is a publicly available dataset -- the [NYC Taxi Trips dataset](#). The NYC Taxi Trip data consists of about 20GB of compressed CSV files (~48GB uncompressed), recording more than 173 million

individual trips and the fares paid for each trip. Each trip record includes the pickup and drop-off locations and times, anonymized hack (driver's) license number, and the medallion (taxi's unique id) number. The data covers all trips in the year 2013 and is provided in the following two datasets for each month:

- The 'trip_data' CSV contains trip details, such as number of passengers, pickup and dropoff points, trip duration, and trip length. Here are a few sample records:

```
medallion,hack_license,vendor_id,rate_code,store_and_fwd_flag,pickup_datetime,dropoff_datetime,passenger_count,  
trip_time_in_secs,trip_distance,pickup_longitude,pickup_latitude,dropoff_longitude,dropoff_latitude  
89D227B655E5C82AECF13C3F540D4CF4,BA96DE419E711691B9445D6A6307C170,CMT,1,N,2013-01-01 15:11:48,2013-01-01  
15:18:10,4,382,1.00,-73.978165,40.757977,-73.989838,40.751171  
0BD7C8F5BA12B88E0B67BED28BEA73D8,9FD8F69F0804BDB5549F40E9DA1BE472,CMT,1,N,2013-01-06 00:18:35,2013-01-06  
00:22:54,1,259,1.50,-74.006683,40.731781,-73.994499,40.75066  
0BD7C8F5BA12B88E0B67BED28BEA73D8,9FD8F69F0804BDB5549F40E9DA1BE472,CMT,1,N,2013-01-05 18:49:41,2013-01-05  
18:54:23,1,282,1.10,-74.004707,40.73777,-74.009834,40.726002  
DFD2202EE08F7A8DC9A57B02ACB81FE2,51EE87E3205C985EF8431D850C786310,CMT,1,N,2013-01-07 23:54:15,2013-01-07  
23:58:20,2,244,.70,-73.974602,40.759945,-73.984734,40.759388  
DFD2202EE08F7A8DC9A57B02ACB81FE2,51EE87E3205C985EF8431D850C786310,CMT,1,N,2013-01-07 23:25:03,2013-01-07  
23:34:24,1,560,2.10,-73.97625,40.748528,-74.002586,40.747868
```

- The 'trip_fare' CSV contains details of the fare paid for each trip, such as payment type, fare amount, surcharge and taxes, tips and tolls, and the total amount paid. Here are a few sample records:

```
medallion,hack_license,vendor_id,pickup_datetime,payment_type,fare_amount,surcharge,mta_tax,tip_amount,tolls_amount,  
total_amount  
89D227B655E5C82AECF13C3F540D4CF4,BA96DE419E711691B9445D6A6307C170,CMT,2013-01-01 15:11:48,CSH,6.5,0,0.5,0,0.7  
0BD7C8F5BA12B88E0B67BED28BEA73D8,9FD8F69F0804BDB5549F40E9DA1BE472,CMT,2013-01-06 00:18:35,CSH,6,0.5,0.5,0,0.7  
0BD7C8F5BA12B88E0B67BED28BEA73D8,9FD8F69F0804BDB5549F40E9DA1BE472,CMT,2013-01-05 18:49:41,CSH,5.5,1,0.5,0,0.7  
DFD2202EE08F7A8DC9A57B02ACB81FE2,51EE87E3205C985EF8431D850C786310,CMT,2013-01-07 23:54:15,CSH,5,0.5,0.5,0,0.6  
DFD2202EE08F7A8DC9A57B02ACB81FE2,51EE87E3205C985EF8431D850C786310,CMT,2013-01-07 23:25:03,CSH,9.5,0.5,0.5,0,10.5
```

The unique key to join trip_data and trip_fare is composed of the following three fields: medallion, hack_license and pickup_datetime. The raw CSV files can be accessed from a public Azure storage blob. The U-SQL script for this join is in the [Join trip and fare tables](#) section.

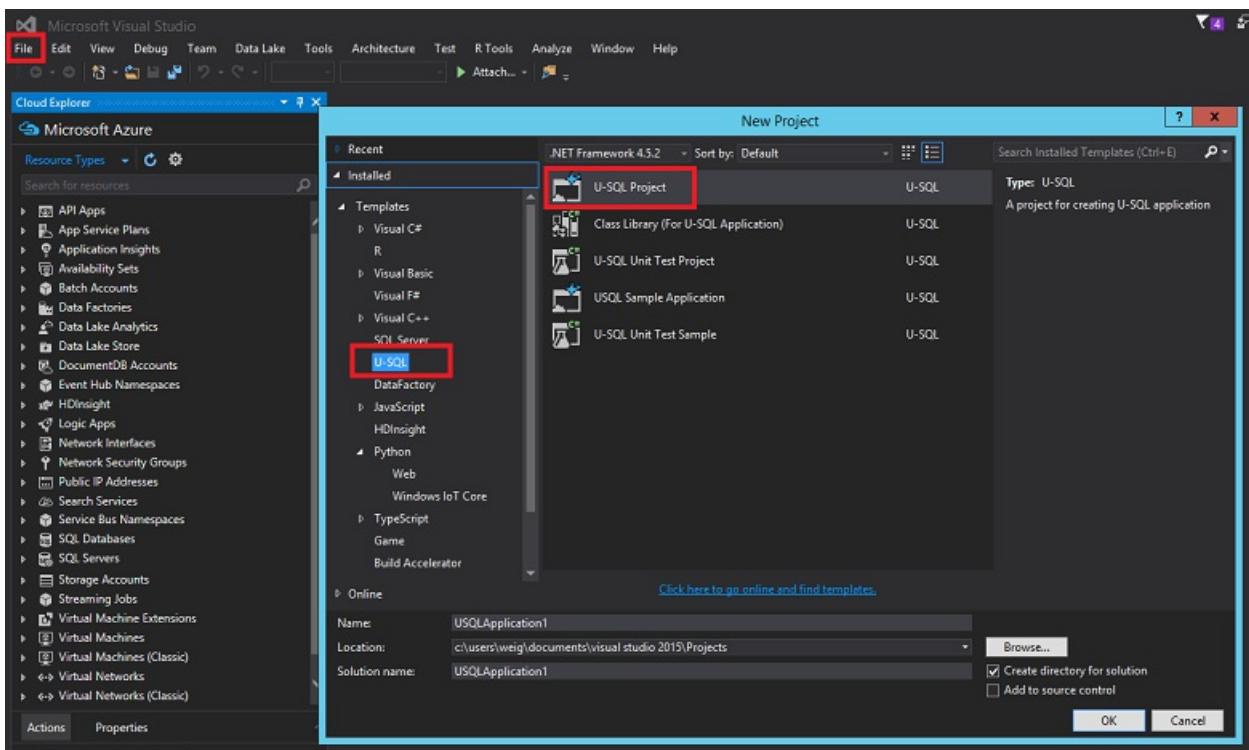
Process data with U-SQL

The data processing tasks illustrated in this section include ingesting, checking quality, exploring, and sampling the data. We also show how to join trip and fare tables. The final section shows run a U-SQL scripted job from the Azure portal. Here are links to each subsection:

- [Data ingestion: read in data from public blob](#)
- [Data quality checks](#)
- [Data exploration](#)
- [Join trip and fare tables](#)
- [Data sampling](#)
- [Run U-SQL jobs](#)

The U-SQL scripts are described here and provided in a separate file. You can download the full **U-SQL scripts** from [GitHub](#).

To execute U-SQL, Open Visual Studio, click **File --> New --> Project**, choose **U-SQL Project**, name and save it to a folder.



NOTE

It is possible to use the Azure Portal to execute U-SQL instead of Visual Studio. You can navigate to the Azure Data Lake Analytics resource on the portal and submit queries directly as illustrated in the following figure.

```

1 // Replace weig with real values
2 DECLARE @par1 string = "weig";
3 DECLARE @outpath1 string = "/MLADS_Users/" + @par1 + "/task_SS.csv";
4
5 ///tipped vs. not tipped distribution
6 @tip_or_not =
7   SELECT *
8     (tip_amount > 0 ? 1: 0) AS tipped
9   FROM weig.dbo.fare;
10
11 @task_5 =
12   SELECT tipped,
13         COUNT(*) AS tip_freq
14   FROM @tip_or_not
15   GROUP BY tipped;
16   OUTPUT @task_5
17   TO @outpath1
18   USING Outputters.Csv();
  
```

Data Ingestion: Read in data from public blob

The location of the data in the Azure blob is referenced as

wasb://container_name@blob_storage_account_name.blob.core.windows.net/blob_name and can be extracted using **Extractors.Csv()**. Substitute your own container name and storage account name in following scripts for *containername@blob_storage_account_name* in the wasb address. Since the file names are in same format, we can use ****trip_data{*}.csv**** to read in all 12 trip files.

```

///Read in Trip data
@trip0 =
    EXTRACT
        medallion string,
        hack_license string,
        vendor_id string,
        rate_code string,
        store_and_fwd_flag string,
        pickup_datetime string,
        dropoff_datetime string,
        passenger_count string,
        trip_time_in_secs string,
        trip_distance string,
        pickup_longitude string,
        pickup_latitude string,
        dropoff_longitude string,
        dropoff_latitude string
    // This is reading 12 trip data from blob
    FROM "wasb://container_name@blob_storage_account_name.blob.core.windows.net/nyctaxitrip/trip_data_{*}.csv"
    USING Extractors.Csv();

```

Since there are headers in the first row, we need to remove the headers and change column types into appropriate ones. We can either save the processed data to Azure Data Lake Storage using

swebhdfs://data_lake_storage_name.azuredatalakestorage.net/folder_name/file_name_ or to Azure Blob storage account using

wasb://container_name@blob_storage_account_name.blob.core.windows.net/blob_name.

```

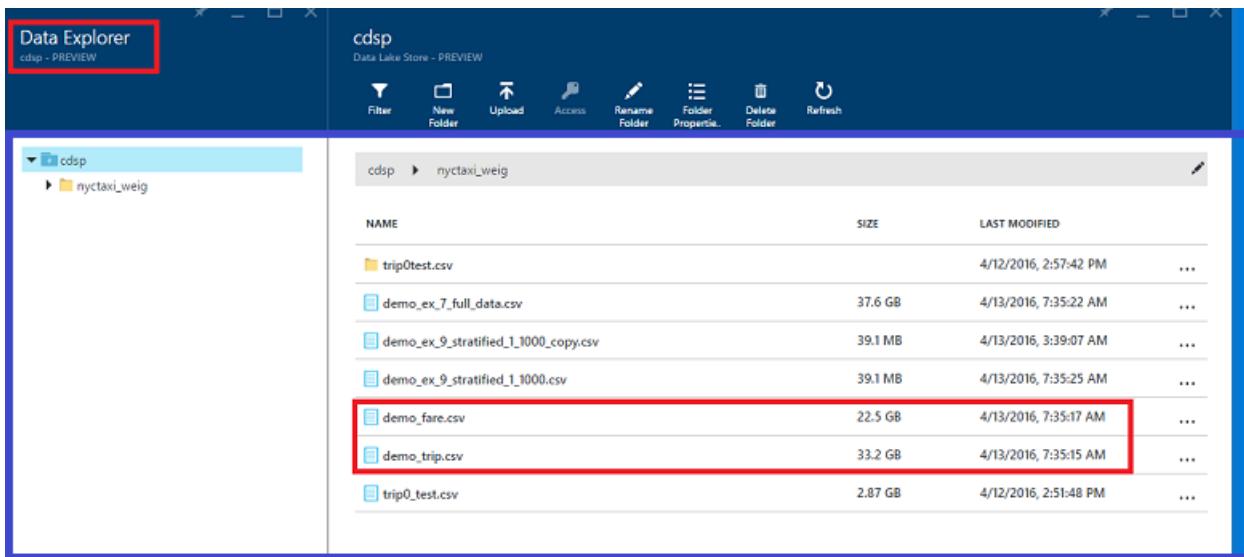
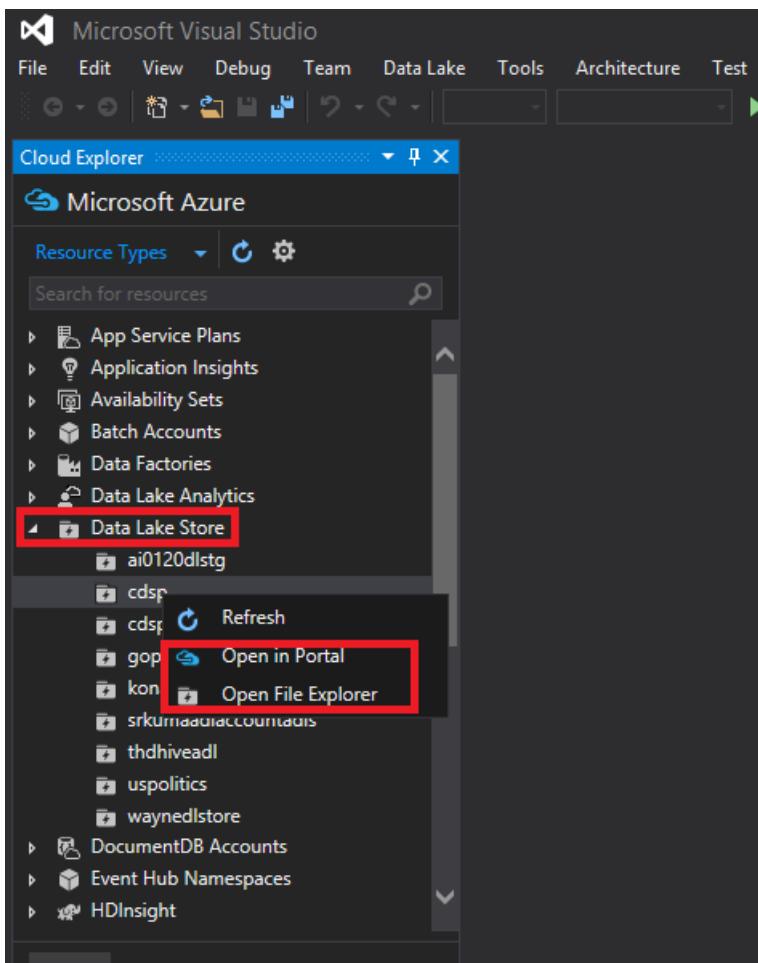
// change data types
@trip =
    SELECT
        medallion,
        hack_license,
        vendor_id,
        rate_code,
        store_and_fwd_flag,
        DateTime.Parse(pickup_datetime) AS pickup_datetime,
        DateTime.Parse(dropoff_datetime) AS dropoff_datetime,
        Int32.Parse(passenger_count) AS passenger_count,
        Double.Parse(trip_time_in_secs) AS trip_time_in_secs,
        Double.Parse(trip_distance) AS trip_distance,
        (pickup_longitude==string.Empty ? 0: float.Parse(pickup_longitude)) AS pickup_longitude,
        (pickup_latitude==string.Empty ? 0: float.Parse(pickup_latitude)) AS pickup_latitude,
        (dropoff_longitude==string.Empty ? 0: float.Parse(dropoff_longitude)) AS dropoff_longitude,
        (dropoff_latitude==string.Empty ? 0: float.Parse(dropoff_latitude)) AS dropoff_latitude
    FROM @trip0
    WHERE medallion != "medallion";

////output data to ADL
OUTPUT @trip
TO "swebhdfs://data_lake_storage_name.azuredatalakestore.net/nyctaxi_folder/demo_trip.csv"
USING Outputters.Csv();

////Output data to blob
OUTPUT @trip
TO "wasb://container_name@blob_storage_account_name.blob.core.windows.net/demo_trip.csv"
USING Outputters.Csv();

```

Similarly we can read in the fare data sets. Right click Azure Data Lake Store, you can choose to look at your data in **Azure Portal --> Data Explorer** or **File Explorer** within Visual Studio.



Data quality checks

After trip and fare tables have been read in, data quality checks can be done in the following way. The resulting CSV files can be output to Azure Blob storage or Azure Data Lake Store.

Find the number of medallions and unique number of medallions:

```

///check the number of medallions and unique number of medallions
@trip2 =
    SELECT
        medallion,
        vendor_id,
        pickup_datetime.Month AS pickup_month
    FROM @trip;

@ex_1 =
    SELECT
        pickup_month,
        COUNT(medallion) AS cnt_medallion,
        COUNT(DISTINCT(medallion)) AS unique_medallion
    FROM @trip2
    GROUP BY pickup_month;
    OUTPUT @ex_1
    TO "wasb://container_name@blob_storage_account_name.blob.core.windows.net/demo_ex_1.csv"
    USING Outputters.Csv();

```

Find those medallions that had more than 100 trips:

```

///find those medallions that had more than 100 trips
@ex_2 =
    SELECT medallion,
        COUNT(medallion) AS cnt_medallion
    FROM @trip2
    //where pickup_datetime >= "2013-01-01t00:00:00.0000000" and pickup_datetime <= "2013-04-01t00:00:00.0000000"
    GROUP BY medallion
    HAVING COUNT(medallion) > 100;
    OUTPUT @ex_2
    TO "wasb://container_name@blob_storage_account_name.blob.core.windows.net/demo_ex_2.csv"
    USING Outputters.Csv();

```

Find those invalid records in terms of pickup_longitude:

```

///find those invalid records in terms of pickup_longitude
@ex_3 =
    SELECT COUNT(medallion) AS cnt_invalid_pickup_longitude
    FROM @trip
    WHERE
        pickup_longitude < 90 OR pickup_longitude > 90;
    OUTPUT @ex_3
    TO "wasb://container_name@blob_storage_account_name.blob.core.windows.net/demo_ex_3.csv"
    USING Outputters.Csv();

```

Find missing values for some variables:

```

//check missing values
@res =
    SELECT *,
        (medallion == null? 1 : 0) AS missing_medallion
    FROM @trip;

@trip_summary6 =
    SELECT
        vendor_id,
        SUM(missing_medallion) AS medallion_empty,
        COUNT(medallion) AS medallion_total,
        COUNT(DISTINCT(medallion)) AS medallion_total_unique
    FROM @res
    GROUP BY vendor_id;
OUTPUT @trip_summary6
TO "wasb://container_name@blob_storage_account_name.blob.core.windows.net/demo_ex_16.csv"
USING Outputters.Csv();

```

Data exploration

We can do some data exploration to get a better understanding of the data.

Find the distribution of tipped and non-tipped trips:

```

///tipped vs. not tipped distribution
@tip_or_not =
    SELECT *,
        (tip_amount > 0 ? 1: 0) AS tipped
    FROM @fare;

@ex_4 =
    SELECT tipped,
        COUNT(*) AS tip_freq
    FROM @tip_or_not
    GROUP BY tipped;
    OUTPUT @ex_4
TO "wasb://container_name@blob_storage_account_name.blob.core.windows.net/demo_ex_4.csv"
USING Outputters.Csv();

```

Find the distribution of tip amount with cut-off values: 0,5,10, and 20 dollars.

```

//tip class/range distribution
@tip_class =
    SELECT *,
        (tip_amount >20? 4: (tip_amount >10? 3:(tip_amount >5 ? 2:(tip_amount > 0 ? 1: 0))) AS tip_class
    FROM @fare;

@ex_5 =
    SELECT tip_class,
        COUNT(*) AS tip_freq
    FROM @tip_class
    GROUP BY tip_class;
    OUTPUT @ex_5
TO "wasb://container_name@blob_storage_account_name.blob.core.windows.net/demo_ex_5.csv"
USING Outputters.Csv();

```

Find basic statistics of trip distance:

```

// find basic statistics for trip_distance
@trip_summary4=
SELECT
    vendor_id,
    COUNT(*) AS cnt_row,
    MIN(trip_distance) AS min_trip_distance,
    MAX(trip_distance) AS max_trip_distance,
    AVG(trip_distance) AS avg_trip_distance
FROM @trip
GROUP BY vendor_id;
OUTPUT @trip_summary4
TO "wasb://container_name@blob_storage_account_name.blob.core.windows.net/demo_ex_14.csv"
USING Outputters.Csv();

```

Find the percentiles of trip distance:

```

// find percentiles of trip_distance
@trip_summary3=
SELECT DISTINCT vendor_id AS vendor,
    PERCENTILE_DISC(0.25) WITHIN GROUP(ORDER BY trip_distance) OVER(PARTITION BY vendor_id) AS median_trip_distance_disc,
    PERCENTILE_DISC(0.5) WITHIN GROUP(ORDER BY trip_distance) OVER(PARTITION BY vendor_id) AS median_trip_distance_disc,
    PERCENTILE_DISC(0.75) WITHIN GROUP(ORDER BY trip_distance) OVER(PARTITION BY vendor_id) AS median_trip_distance_disc
FROM @trip;
// group by vendor_id;
OUTPUT @trip_summary3
TO "wasb://container_name@blob_storage_account_name.blob.core.windows.net/demo_ex_13.csv"
USING Outputters.Csv();

```

Join trip and fare tables

Trip and fare tables can be joined by medallion, hack_license, and pickup_time.

```

//join trip and fare table

@model_data_full=
SELECT t.*,
f.payment_type, f.fare_amount, f.surcharge, f.mta_tax, f.tolls_amount, f.total_amount, f.tip_amount,
(f.tip_amount > 0 ? 1: 0) AS tipped,
(f.tip_amount >20? 4: (f.tip_amount >10? 3:(f.tip_amount >5 ? 2:(f.tip_amount >0 ? 1: 0)))) AS tip_class
FROM @trip AS t JOIN @fare AS f
ON (t.medallion == f.medallion AND t.hack_license == f.hack_license AND t.pickup_datetime == f.pickup_datetime)
WHERE (pickup_longitude != 0 AND dropoff_longitude != 0);

//// output to blob
OUTPUT @model_data_full
TO "wasb://container_name@blob_storage_account_name.blob.core.windows.net/demo_ex_7_full_data.csv"
USING Outputters.Csv();

////output data to ADL
OUTPUT @model_data_full
TO "swebhdfs://data_lake_storage_name.azuredatalakestore.net/nyctaxi_folder/demo_ex_7_full_data.csv"
USING Outputters.Csv();

```

For each level of passenger count, calculate the number of records, average tip amount, variance of tip amount, percentage of tipped trips.

```

// contingency table
@trip_summary8 =
    SELECT passenger_count,
        COUNT(*) AS cnt,
        AVG(tip_amount) AS avg_tip_amount,
        VAR(tip_amount) AS var_tip_amount,
        SUM(tipped) AS cnt_tipped,
        (float)SUM(tipped)/COUNT(*) AS pct_tipped
    FROM @model_data_full
    GROUP BY passenger_count;
    OUTPUT @trip_summary8
    TO "wasb://container_name@blob_storage_account_name.blob.core.windows.net/demo_ex_17.csv"
    USING Outputters.Csv();

```

Data sampling

First we randomly select 0.1% of the data from the joined table:

```

//random select 1/1000 data for modeling purpose
@addrownumeres_randomsample =
SELECT *,
    ROW_NUMBER() OVER() AS rounum
FROM @model_data_full;

@model_data_random_sample_1_1000 =
SELECT *
FROM @addrownumeres_randomsample
WHERE rounum % 1000 == 0;

OUTPUT @model_data_random_sample_1_1000
TO "wasb://container_name@blob_storage_account_name.blob.core.windows.net/demo_ex_7_random_1_1000.csv"
USING Outputters.Csv();

```

Then we do stratified sampling by binary variable tip_class:

```

//stratified random select 1/1000 data for modeling purpose
@addrownumeres_stratifiedsample =
SELECT *,
    ROW_NUMBER() OVER(PARTITION BY tip_class) AS rounum
FROM @model_data_full;

@model_data_stratified_sample_1_1000 =
SELECT *
FROM @addrownumeres_stratifiedsample
WHERE rounum % 1000 == 0;
/// output to blob
OUTPUT @model_data_stratified_sample_1_1000
TO "wasb://container_name@blob_storage_account_name.blob.core.windows.net/demo_ex_9_stratified_1_1000.csv"
USING Outputters.Csv();
///output data to ADL
OUTPUT @model_data_stratified_sample_1_1000
TO "swebhdfs://data_lake_storage_name.azuredatalakestore.net/nytaxi_folder/demo_ex_9_stratified_1_1000.csv"
USING Outputters.Csv();

```

Run U-SQL jobs

When you finish editing U-SQL scripts, you can submit them to the server using your Azure Data Lake Analytics account. Click **Data Lake, Submit Job**, select your **Analytics Account**, choose **Parallelism**, and click **Submit** button.

```

1 ///////////////////////////////////////////////////////////////////
2 ///Read in Trip data
3
4 @trip0 =
5     EXTRACT
6         medallion string,
7         hack_license string,
8         vendor_id string,
9         rate_code string,
10        store_and_fwd_flag string,
11        pickup_datetime string,
12        dropoff_datetime string,
13        passenger_count string,
14        trip_time_in_secs string,
15        trip_distance string,
16        pickup_longitude string,
17        pickup_latitude string,
18        dropoff_longitude string,
19        dropoff_latitude string
20    // This is reading 12 trip data from blob

```

When the job is compiled successfully, the status of your job will be displayed in Visual Studio for monitoring. After the job finishes running, you can even replay the job execution process and find out the bottleneck steps to improve your job efficiency. You can also go to Azure Portal to check the status of your U-SQL jobs.

weig_first_USQL - Microsoft Visual Studio

File Edit View Project Build Debug Team Data Lake Tools Architecture Test R Tools Analyze Window Help

Cloud Explorer Microsoft Azure Resource Types Search for resources

Data Lake Store DocumentDB Accounts Event Hub Namespaces HDInsight Logic Apps Network Interfaces Network Security Groups Public IP Addresses Search Services Service Bus Namespaces SQL Databases SQL Servers

Actions Properties Refresh

Job Summary

Preparing Queued Running Finalizing

48 seconds 0 seconds 3.8 hours

Job Completed Successfully No errors to report

Job Result Succeeded Total Duration 3.8 hours

Submit Time 4/13/2016 3:49:59 AM Start Time 4/13/2016 3:51:07 AM End Time 4/13/2016 7:37:02 AM

Compilation 48 seconds Queued 0 seconds Running 3.8 hours

Account cdspk Author weig@microsoft.com

Priority 0

Output Show output from: General

..... Submit started: Script: AzureDataLake_U_SQL_Demo.usql
..... preparing job Information entity...
..... Submit: 1 succeeded

cdspk Data Lake Analytics - PREVIEW

Settings New Job Add Data Source View All Jobs Data Explorer Delete

Essentials

Resource group cdspkdatalake

Status Running

Location East US 2

Subscription name Microsoft

Subscription ID

Pricing tier Pay-As-You-Go

Default Data Lake Store

Learn Explore sample jobs

Getting Started Explore interactive tutorials

All Jobs cdspkdatalake - PREVIEW

TOTAL: 238

STATUS	JOB NAME	LANGUAGE
✓ Succeeded	AzureDataLake_U_SQL_Demo	U-SQL
		U-SQL
		U-SQL
		U-SQL

Now you can check the output files in either Azure Blob storage or Azure Portal. We will use the stratified sample data for our modeling in the next step.

Storage Account
weigstorageforsvm

test1 blob container (98 blobs, 188.94G) as of 4/27/2016 12:20:15 AM

	Name	Type	Last Modified	Length	Content Type
	demo_ex_1.csv	Block	4/13/2016 2:35:23 PM +0:00	219 bytes	text/plain; charset=utf-8
	demo_ex_10.csv	Block	4/13/2016 2:35:26 PM +0:00	40 bytes	text/plain; charset=utf-8
	demo_ex_11.csv	Block	4/13/2016 2:35:18 PM +0:00	32 bytes	text/plain; charset=utf-8
	demo_ex_12.csv	Block	4/13/2016 2:35:18 PM +0:00	118 bytes	text/plain; charset=utf-8
	demo_ex_13.csv	Block	4/13/2016 2:35:20 PM +0:00	30 bytes	text/plain; charset=utf-8
	demo_ex_14.csv	Block	4/13/2016 2:35:19 PM +0:00	89 bytes	text/plain; charset=utf-8
	demo_ex_15.csv	Block	4/13/2016 2:35:20 PM +0:00	22 bytes	text/plain; charset=utf-8
	demo_ex_16.csv	Block	4/13/2016 2:35:20 PM +0:00	46 bytes	text/plain; charset=utf-8
	demo_ex_17.csv	Block	4/13/2016 2:35:24 PM +0:00	695 bytes	text/plain; charset=utf-8
	demo_ex_2.csv	Block	4/13/2016 2:35:21 PM +0:00	550.96K	text/plain; charset=utf-8
	demo_ex_3.csv	Block	4/13/2016 2:35:19 PM +0:00	6 bytes	text/plain; charset=utf-8
	demo_ex_4.csv	Block	4/13/2016 2:35:19 PM +0:00	24 bytes	text/plain; charset=utf-8
	demo_ex_5.csv	Block	4/13/2016 2:35:19 PM +0:00	55 bytes	text/plain; charset=utf-8
	demo_ex_6.csv	Block	4/13/2016 2:35:19 PM +0:00	187.02K	text/plain; charset=utf-8
	demo_ex_7_full_data.csv	Block	4/13/2016 2:35:22 PM +0:00	35.05G	text/plain; charset=utf-8
	demo_ex_7_random_1_1000.csv	Block	4/13/2016 2:35:24 PM +0:00	37.41M	text/plain; charset=utf-8
	demo_ex_8.csv	Block	4/13/2016 2:35:27 PM +0:00	40 bytes	text/plain; charset=utf-8
	demo_ex_9_stratified_1_1000.csv	Block	4/13/2016 2:35:26 PM +0:00	37.32M	text/plain; charset=utf-8
	demo_ex_9_stratified_1_1000_copy.csv	Block	4/13/2016 10:39:08 AM +0:00	37.32M	text/plain; charset=utf-8
	demo_fare.csv	Block	4/13/2016 2:35:18 PM +0:00	20.97G	text/plain; charset=utf-8
	demo_trip.csv	Block	4/13/2016 2:35:16 PM +0:00	30.88G	text/plain; charset=utf-8

cdsp

Data Lake Store - PREVIEW

Filter New Folder Upload Access Rename Folder Folder Properties Delete Folder Refresh

cdsp > nyctaxi_weig

NAME	SIZE	LAST MODIFIED
trip0test.csv		4/12/2016, 2:57:42 PM
demo_ex_7_full_data.csv	37.6 GB	4/13/2016, 7:35:22 AM
demo_ex_9_stratified_1_1000_copy.csv	39.1 MB	4/13/2016, 3:39:07 AM
demo_ex_9_stratified_1_1000.csv	39.1 MB	4/13/2016, 7:35:25 AM
demo_fare.csv	22.5 GB	4/13/2016, 7:35:17 AM
demo_trip.csv	33.2 GB	4/13/2016, 7:35:15 AM
trip0_test.csv	2.87 GB	4/12/2016, 2:51:48 PM

Build and deploy models in Azure Machine Learning

We demonstrate two options available for you to pull data into Azure Machine Learning to build and

- In the first option, you use the sampled data that has been written to an Azure Blob (in the **Data sampling** step above) and use Python to build and deploy models from Azure Machine Learning.
- In the second option, you query the data in Azure Data Lake directly using a Hive query. This option requires that you create a new HDInsight cluster or use an existing HDInsight cluster where the Hive tables point to the NY Taxi data in Azure Data Lake Storage. We discuss both these options below.

Option 1: Use Python to build and deploy machine learning models

To build and deploy machine learning models using Python, create a Jupyter Notebook on your local machine or in Azure Machine Learning Studio. The Jupyter Notebook provided on [GitHub](#) contains the full code to explore, visualize data, feature engineering, modeling and deployment. In this article, we show just the modeling and deployment.

Import Python libraries

In order to run the sample Jupyter Notebook or the Python script file, the following Python packages are needed. If you are using the AzureML Notebook service, these packages have been pre-installed.

```
import pandas as pd
from pandas import Series, DataFrame
import numpy as np
import matplotlib.pyplot as plt
from time import time
import pyodbc
import os
from azure.storage.blob import BlobService
import tables
import time
import zipfile
import random
import sklearn
from sklearn.linear_model import LogisticRegression
from sklearn.cross_validation import train_test_split
from sklearn import metrics
from __future__ import division
from sklearn import linear_model
from azureml import services
```

Read in the data from blob

- Connection String

```
CONTAINERNAME='test1'
STORAGEACCOUNTNAME='XXXXXXXXXX'
STORAGEACCOUNTKEY='YYYYYYYYYYYYYYYYYYYYYYYYYYYY'
BLOBNAME='demo_ex_9_stratified_1_1000_copy.csv'
blob_service = BlobService(account_name=STORAGEACCOUNTNAME,account_key=STORAGEACCOUNTKEY)
```

- Read in as text

```
t1 = time.time()
data = blob_service.get_blob_to_text(CONTAINERNAME,BLOBNAME).split("\n")
t2 = time.time()
print("It takes %s seconds to read in "+BLOBNAME) % (t2 - t1)
```

It takes 1.61118912697 seconds to read in demo_ex_9_stratified_1_1000_copy.csv

- Add column names and separate columns

```
colnames =['medallion','hack_license','vendor_id','rate_code','store_and_fwd_flag','pickup_datetime','dropoff_datetime',
'passenger_count','trip_time_in_secs','trip_distance','pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude',
'payment_type','fare_amount','surcharge','mta_tax','tolls_amount','total_amount','tip_amount','tipped','tip_class','rownum']
df1 = pd.DataFrame([sub.split(",") for sub in data], columns = colnames)
```

- Change some columns to numeric

```
cols_2_float=['trip_time_in_secs','pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude',
'fare_amount','surcharge','mta_tax','tolls_amount','total_amount','tip_amount','passenger_count','trip_distance',
'tipped','tip_class','rownum']
for col in cols_2_float:
    df1[col] = df1[col].astype(float)
```

Build machine learning models

Here we build a binary classification model to predict whether a trip is tipped or not. In the Jupyter Notebook you

can find other two models: multiclass classification, and regression models.

- First we need to create dummy variables that can be used in scikit-learn models

```
dfl_payment_type_dummy = pd.get_dummies(dfl['payment_type'], prefix='payment_type_dummy')
dfl_vendor_id_dummy = pd.get_dummies(dfl['vendor_id'], prefix='vendor_id_dummy')
```

- Create data frame for the modeling

```
cols_to_keep = ['tipped', 'trip_distance', 'passenger_count']
data = dfl[cols_to_keep].join([dfl_payment_type_dummy, dfl_vendor_id_dummy])

X = data.iloc[:, 1:]
Y = data.tipped
```

- Training and testing 60-40 split

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.4, random_state=0)
```

- Logistic Regression in training set

```
model = LogisticRegression()
logit_fit = model.fit(X_train, Y_train)
print('Coefficients: \n', logit_fit.coef_)
Y_train_pred = logit_fit.predict(X_train)

![c1](./media/machine-learning-data-science-process-data-lake-walkthrough/c1-py-logit-coefficient.PNG)
```

- Score testing data set

```
Y_test_pred = logit_fit.predict(X_test)
```

- Calculate Evaluation metrics

```
fpr_train, tpr_train, thresholds_train = metrics.roc_curve(Y_train, Y_train_pred)
print fpr_train, tpr_train, thresholds_train

fpr_test, tpr_test, thresholds_test = metrics.roc_curve(Y_test, Y_test_pred)
print fpr_test, tpr_test, thresholds_test

#AUC
print metrics.auc(fpr_train, tpr_train)
print metrics.auc(fpr_test, tpr_test)

#Confusion Matrix
print metrics.confusion_matrix(Y_train, Y_train_pred)
print metrics.confusion_matrix(Y_test, Y_test_pred)

![c2](./media/machine-learning-data-science-process-data-lake-walkthrough/c2-py-logit-evaluation.PNG)
```

Build Web Service API and consume it in Python

We want to operationalize the machine learning model after it has been built. Here we use the binary logistic model as an example. Make sure the scikit-learn version in your local machine is 0.15.1. You don't have to worry about this if you use Azure ML studio service.

- Find your workspace credentials from Azure ML studio settings. In Azure Machine Learning Studio, click **Settings --> Name --> Authorization Tokens**.

```
workspaceid = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx'
auth_token = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx'
```

- Create Web Service

```
@services.publish(workspaceid, auth_token)
@services.types(trip_distance = float, passenger_count = float, payment_type_dummy_CRD = float, payment_type_dummy_CSH=float,
payment_type_dummy_DIS = float, payment_type_dummy_NOC= float, payment_type_dummy_UNK = float, vendor_id_dummy_CMT =
float, vendor_id_dummy_VTS = float)
@services.returns(int) #0, or 1
def predictNYCTAXI(trip_distance, passenger_count, payment_type_dummy_CRD,
payment_type_dummy_CSH,payment_type_dummy_DIS, payment_type_dummy_NOC, payment_type_dummy_UNK,
vendor_id_dummy_CMT, vendor_id_dummy_VTS):
    inputArray = [trip_distance, passenger_count, payment_type_dummy_CRD, payment_type_dummy_CSH, payment_type_dummy_DIS,
    payment_type_dummy_NOC, payment_type_dummy_UNK, vendor_id_dummy_CMT, vendor_id_dummy_VTS]
    return logit_.predict(inputArray)
```

- Get web service credentials

```
url = predictNYCTAXI.service.url
api_key = predictNYCTAXI.service.api_key

print url
print api_key

@services.service(url, api_key)
@services.types(trip_distance = float, passenger_count = float, payment_type_dummy_CRD = float,
payment_type_dummy_CSH=float,payment_type_dummy_DIS = float, payment_type_dummy_NOC= float, payment_type_dummy_UNK =
float, vendor_id_dummy_CMT = float, vendor_id_dummy_VTS = float)
@services.returns(float)
def NYCTAXIPredictor(trip_distance, passenger_count, payment_type_dummy_CRD,
payment_type_dummy_CSH,payment_type_dummy_DIS, payment_type_dummy_NOC, payment_type_dummy_UNK,
vendor_id_dummy_CMT, vendor_id_dummy_VTS):
    pass
```

- Call Web service API. You have to wait 5-10 seconds after the previous step.

NYCTAXIPredictor(1,2,1,0,0,0,0,0,1)

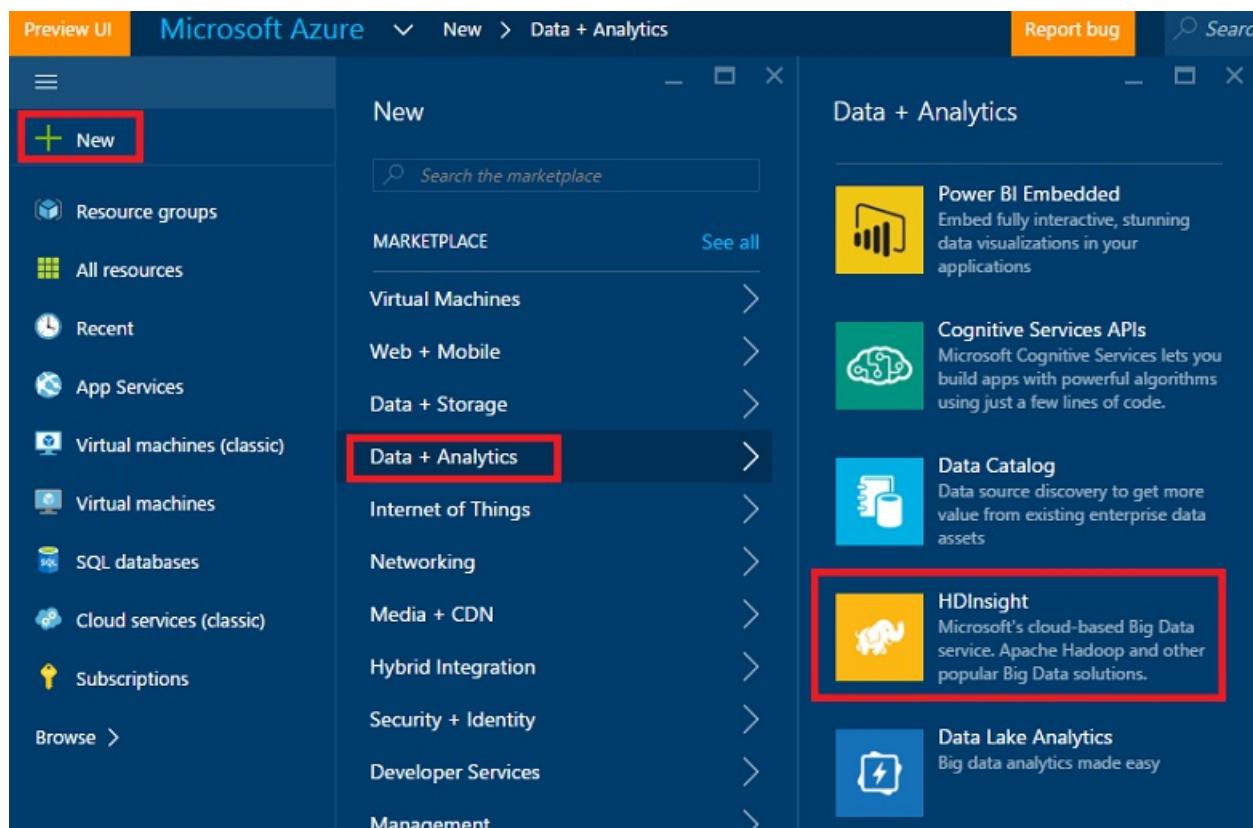
![c4](./media/machine-learning-data-science-process-data-lake-walkthrough/c4-call-API.PNG)

Option 2: Create and deploy models directly in Azure Machine Learning

Azure Machine Learning Studio can read data directly from Azure Data Lake Store and then be used to create and deploy models. This approach uses a Hive table that points at the Azure Data Lake Store. This requires that a separate Azure HDInsight cluster be provisioned, on which the Hive table is created. The following sections show how to do this.

Create an HDInsight Linux Cluster

Create an HDInsight Cluster (Linux) from the [Azure Portal](#). For details, see the **Create an HDInsight cluster with access to Azure Data Lake Store** section in [Create an HDInsight cluster with Data Lake Store using Azure Portal](#).



Create Hive table in HDInsight

Now we create Hive tables to be used in Azure Machine Learning Studio in the HDInsight cluster using the data stored in Azure Data Lake Store in the previous step. Go to the HDInsight cluster just created. Click **Settings** --> **Properties** --> **Cluster AAD Identity** --> **ADLS Access**, make sure your Azure Data Lake Store account is added in the list with read, write and execute rights.

The screenshot shows the Azure HDInsight Settings page. In the top navigation bar, the 'Dashboard' button is highlighted with a red box. The main content area shows the 'Cluster AAD Identity' configuration. The 'Add Access' button is also highlighted with a red box. Below it, the 'Other ADLS accounts' section lists several accounts, with one account's row highlighted with a red box.

Then click **Dashboard** next to the **Settings** button and a window will pop up. Click **Hive View** in the upper right corner of the page and you will see the **Query Editor**.

The screenshot shows the Ambari Metrics dashboard. On the left, there is a sidebar with service icons. In the center, there are several performance metrics displayed in cards. The top right corner features a dropdown menu with three options: 'YARN Queue Manager', 'Hive View', and 'Tez View'. The 'Hive View' option is highlighted with a red box.

The screenshot shows the Ambari Hive Query Editor. At the top, there are tabs for 'Hive', 'Query', 'Saved Queries', 'History', and 'UDFs'. The 'Query' tab is selected. In the center, there is a 'Database Explorer' panel and a 'Query Editor' panel. The 'Query Editor' panel contains a 'Worksheet' tab with the following SQL code:

```

1 CREATE EXTERNAL TABLE nyc_stratified_sample_weighted (
2   medallion string,
3   hack_license string,
4   vendor_id string,
5   rate_code string,
6   store_and_fwd_flag string,
7   pickup_datetime string,
8   dropoff_datetime string,
9   passenger_count string,
10  trip_time_in_secs string,
11  trip_distance string,
12  pickup_longitude string,
13  pickup_latitude string,
14  dropoff_longitude string,
15  dropoff_latitude string,
16  payment_type string,
17  fare_amount string,
18  surcharge string,
19  mta_tax string,
20
  
```

At the bottom of the 'Query Editor' panel, there are buttons for 'Execute', 'Explain', and 'Save as...'. The 'Execute' button is highlighted with a red box.

Paste in the following Hive scripts to create a table. The location of data source is in Azure Data Lake Store reference in this way: **adl://data_lake_store_name.azuredatalakestore.net:443/folder_name/file_name**.

```
CREATE EXTERNAL TABLE nyc_stratified_sample
(
    medallion string,
    hack_license string,
    vendor_id string,
    rate_code string,
    store_and_fwd_flag string,
    pickup_datetime string,
    dropoff_datetime string,
    passenger_count string,
    trip_time_in_secs string,
    trip_distance string,
    pickup_longitude string,
    pickup_latitude string,
    dropoff_longitude string,
    dropoff_latitude string,
    payment_type string,
    fare_amount string,
    surcharge string,
    mta_tax string,
    tolls_amount string,
    total_amount string,
    tip_amount string,
    tipped string,
    tip_class string,
    rownum string
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' lines terminated by '\n'
LOCATION 'adl://data_lake_storage_name.azuredatalakestore.net:443/nyctaxi_folder/demo_ex_9_stratified_1_1000_copy.csv';
```

When the query finishes running, you will see the results like this:

Query Process Results (Status: Succeeded)			Save results... ▾
Logs	Results		
Filter columns...		previous	next
nyc_stratified_sample_weig.medallion	nyc_stratified_sample_weig.hack_license	nyc_stratified_sample_weig.vendor_id	
"0053334C798EC6C8E637657962030F99"	"9EF690115D60940E7A8039A67542642E"	"VTS"	
"00B99071EE4DC8266384113B91E6AC13"	"081B28EDA7F73E5E2C97573F0DBAC25D"	"CMT"	
"011E4CBA1987A8553F8EA5681FFBF7F1"	"F53B26859F6C46C24E39EEAEE8096268"	"CMT"	
"1109955CCAABC BCE1A22BCED5F1DBFF5"	"EAA69F43239E5C6609CD8FF4D6725836"	"CMT"	
"0A415B814A6479EE706972D2FD6DA08A"	"12A32F655B8C9CE3AC7872477A641974"	"VTS"	
"02B29197FB7470B583ED12167D19E998"	"C1EEBC8298A619F86637D7E97F6BDD5C"	"CMT"	
"03055B956C21B1F915DCDB118AA79F21"	"E0C7A67293DE535DB04F8AFD8BF28F73"	"VTS"	
"037673EEAE0DCB912D06BED04E89D89D"	"3B247BD1230E95A0D9FED3E47FECB8D9"	"CMT"	
"03BF54085C92C385889B957D804780D1"	"776D5633A2041CEBDE03092E401D61DB"	"CMT"	
"0437660BC3704C2F185301D539434A64"	"AE9AB8C79A2A0EB6DDA76B7024FF6029"	"CMT"	

We are now ready to build and deploy a model that predicts whether or not a tip is paid with Azure Machine Learning. The stratified sample data is ready to be used in this binary classification (tip or not) problem. The predictive models using multiclass classification (tip_class) and regression (tip_amount) can also be built and deployed with Azure Machine Learning Studio, but here we only show how to handle the case using the binary classification model.

1. Get the data into Azure ML using the **Import Data** module, available in the **Data Input and Output** section. For more information, see the [Import Data module](#) reference page.
2. Select **Hive Query** as the **Data source** in the **Properties** panel.
3. Paste the following Hive script in the **Hive database query** editor

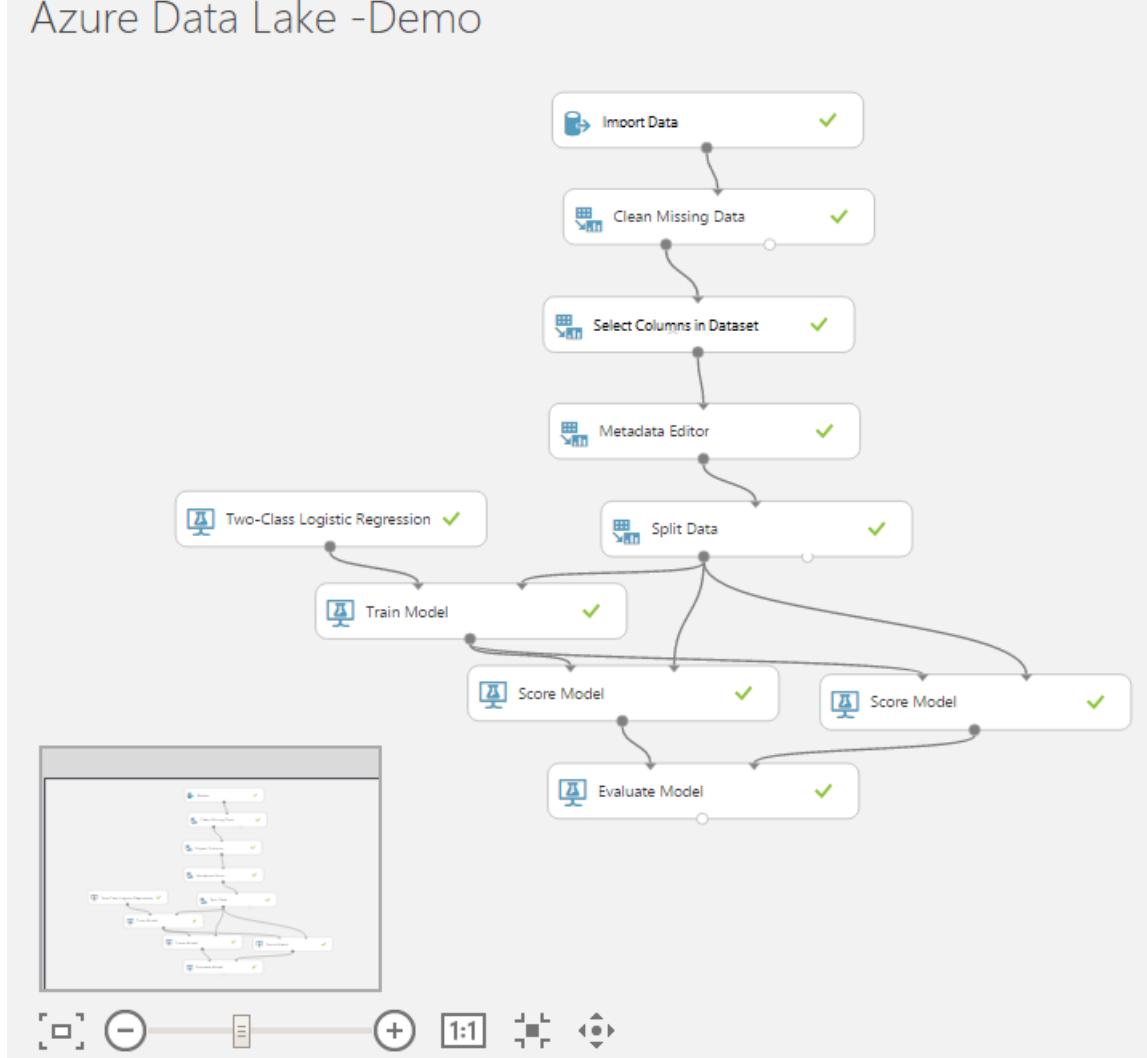
```
select * from nyc_stratified_sample;
```

4. Enter the URI of HDInsight cluster (this can be found in Azure Portal), Hadoop credentials, location of output data, and Azure storage account name/key/container name.



An example of a binary classification experiment reading data from Hive table is shown in the figure below.

Azure Data Lake -Demo

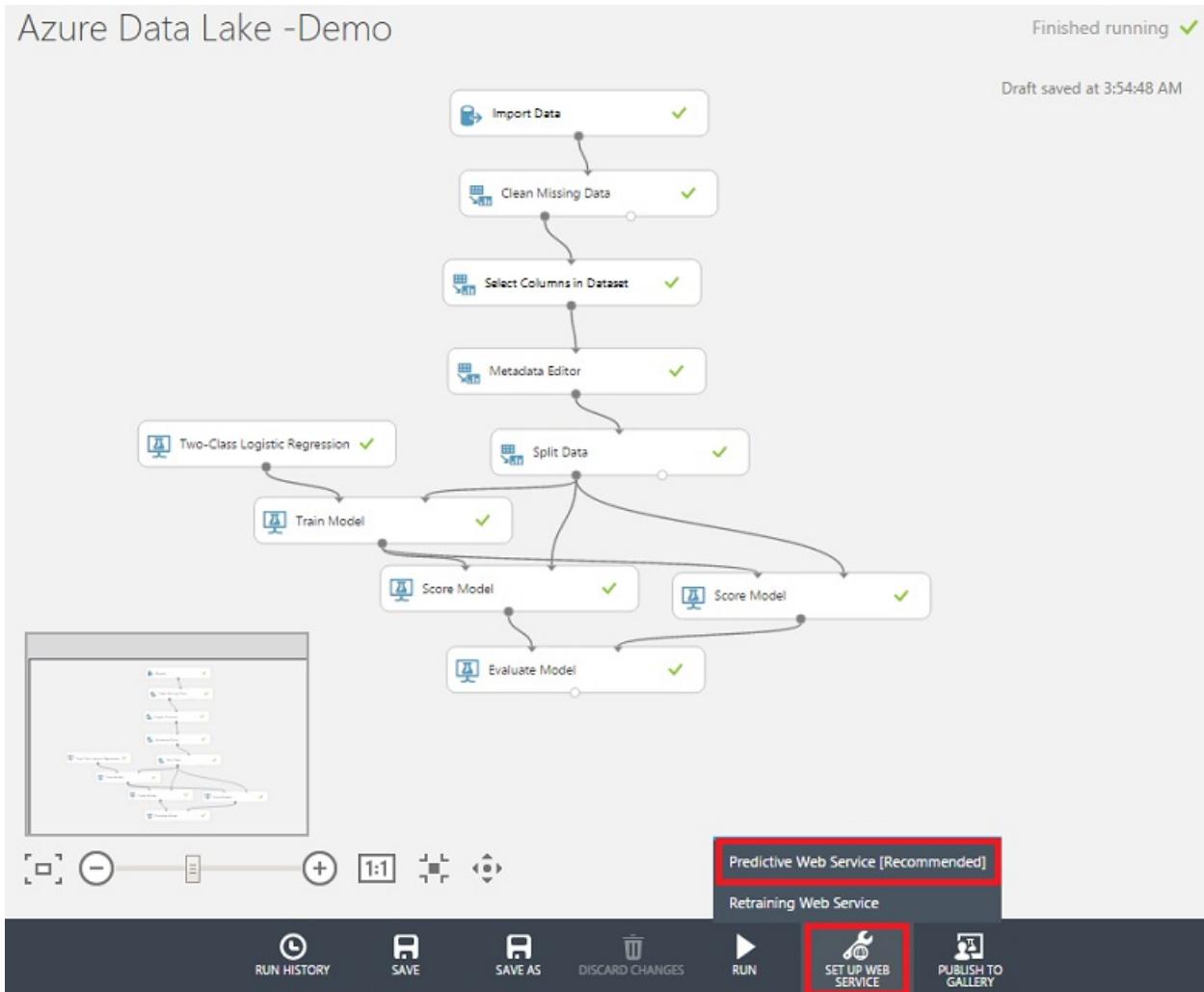


After the experiment is created, click **Set Up Web Service --> Predictive Web Service**

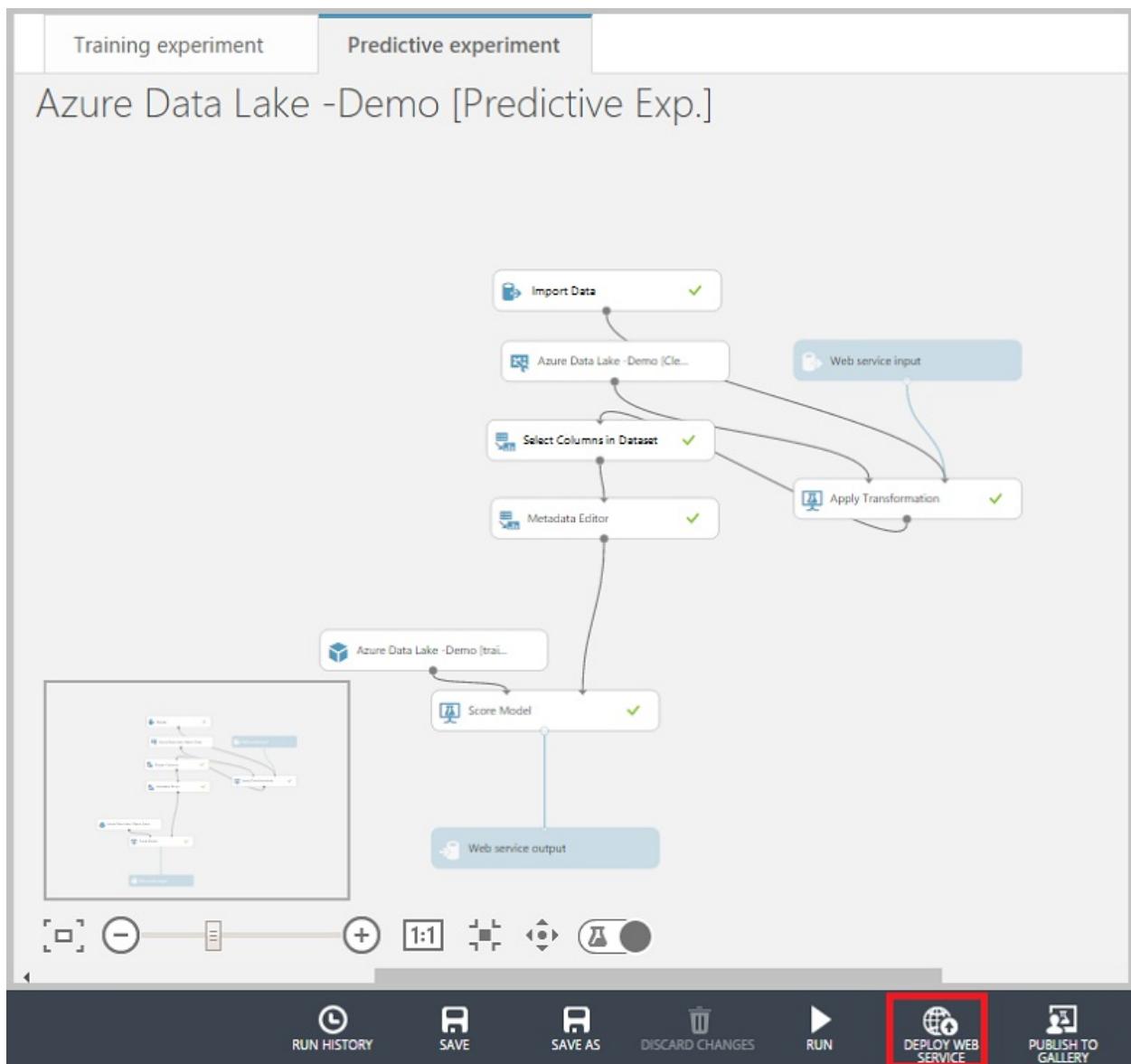
Azure Data Lake - Demo

Finished running ✓

Draft saved at 3:54:48 AM



Run the automatically created scoring experiment, when it finishes, click **Deploy Web Service**



The web service dashboard will be displayed shortly:

Microsoft Azure Machine Learning

azure data lake -demo [predictive exp.]

DASHBOARD **CONFIGURATION**

General

Published experiment
[View snapshot](#) [View latest](#)

Description
 No description provided for this web service.

API key
 h86e ... 51fw==

Default Endpoint

API HELP PAGE	TEST	APPS
REQUEST/RESPONSE	Test	Excel 2013 or later Excel 2010 or earlier workbook
BATCH EXECUTION		Excel 2013 or later workbook

Additional endpoints
 Number of additional endpoints created for this web service: 0
[Manage endpoints in Azure management portal](#)

Summary

By completing this walkthrough you have created a data science environment for building scalable end-to-end

solutions in Azure Data Lake. This environment was used to analyze a large public dataset, taking it through the canonical steps of the Data Science Process, from data acquisition through model training, and then to the deployment of the model as a web service. U-SQL was used to process, explore and sample the data. Python and Hive were used with Azure Machine Learning Studio to build and deploy predictive models.

What's next?

The learning path for the [Team Data Science Process \(TDSP\)](#) provides links to topics describing each step in the advanced analytics process. There are a series of walkthroughs itemized on the [Team Data Science Process walkthroughs](#) page that showcase how to use resources and services in various predictive analytics scenarios:

- [The Team Data Science Process in action: using SQL Data Warehouse](#)
- [The Team Data Science Process in action: using HDInsight Hadoop clusters](#)
- [The Team Data Science Process: using SQL Server](#)
- [Overview of the Data Science Process using Spark on Azure HDInsight](#)

Process Data in SQL Server Virtual Machine on Azure

1/17/2017 • 6 min to read • [Edit on GitHub](#)

This document covers how to explore data and generate features for data stored in a SQL Server VM on Azure. This can be done by data wrangling using SQL or by using a programming language like Python.

NOTE

The sample SQL statements in this document assume that data is in SQL Server. If it isn't, refer to the cloud data science process map to learn how to move your data to SQL Server.

Using SQL

We describe the following data wrangling tasks in this section using SQL:

1. [Data Exploration](#)
2. [Feature Generation](#)

Data Exploration

Here are a few sample SQL scripts that can be used to explore data stores in SQL Server.

NOTE

For a practical example, you can use the [NYC Taxi dataset](#) and refer to the IPNB titled [NYC Data wrangling using IPython Notebook and SQL Server](#) for an end-to-end walk-through.

1. Get the count of observations per day

```
SELECT CONVERT(date,<date_columnname>) as date, count(*) as c from <tablename> group by CONVERT(date,<date_columnname>)
```

2. Get the levels in a categorical column

```
select distinct <column_name> from <databasename>
```

3. Get the number of levels in combination of two categorical columns

```
select <column_a>,<column_b>,count(*) from <tablename> group by <column_a>,<column_b>
```

4. Get the distribution for numerical columns

```
select <column_name>, count(*) from <tablename> group by <column_name>
```

Feature Generation

In this section, we describe ways of generating features using SQL:

1. [Count based Feature Generation](#)
2. [Binning Feature Generation](#)
3. [Rolling out the features from a single column](#)

NOTE

Once you generate additional features, you can either add them as columns to the existing table or create a new table with the additional features and primary key, that can be joined with the original table.

Count based Feature Generation

The following examples demonstrate two ways of generating count features. The first method uses conditional sum and the second method uses the 'where' clause. These can then be joined with the original table (using primary key columns) to have count features alongside the original data.

```
select <column_name1>,<column_name2>,<column_name3>, COUNT(*) as Count_Features from <tablename> group by <column_name1>,<column_name2>,<column_name3>
```

```
select <column_name1>,<column_name2>, sum(1) as Count_Features from <tablename>
where <column_name3>='<some_value>' group by <column_name1>,<column_name2>
```

Binning Feature Generation

The following example shows how to generate binned features by binning (using five bins) a numerical column that can be used as a feature instead:

```
'SELECT <column_name>, NTILE(5) OVER (ORDER BY <column_name>) AS BinNumber from <tablename>'
```

Rolling out the features from a single column

In this section, we demonstrate how to roll out a single column in a table to generate additional features. The example assumes that there is a latitude or longitude column in the table from which you are trying to generate features.

Here is a brief primer on latitude/longitude location data (resourced from stackoverflow [How to measure the accuracy of latitude and longitude?](#)). This is useful to understand before featurizing the location field:

- The sign tells us whether we are north or south, east or west on the globe.
- A nonzero hundreds digit tells us that we're using longitude, not latitude!
- The tens digit gives a position to about 1,000 kilometers. It gives us useful information about what continent or ocean we are on.
- The units digit (one decimal degree) gives a position up to 111 kilometers (60 nautical miles, about 69 miles). It can tell us roughly what large state or country we are in.
- The first decimal place is worth up to 11.1 km: it can distinguish the position of one large city from a neighboring large city.
- The second decimal place is worth up to 1.1 km: it can separate one village from the next.
- The third decimal place is worth up to 110 m: it can identify a large agricultural field or institutional campus.
- The fourth decimal place is worth up to 11 m: it can identify a parcel of land. It is comparable to the typical accuracy of an uncorrected GPS unit with no interference.
- The fifth decimal place is worth up to 1.1 m: it distinguishes trees from each other. Accuracy to this level with commercial GPS units can only be achieved with differential correction.
- The sixth decimal place is worth up to 0.11 m: you can use this for laying out structures in detail, for designing landscapes, building roads. It should be more than good enough for tracking movements of glaciers and rivers. This can be achieved by taking painstaking measures with GPS, such as differentially corrected GPS.

The location information can be featurized as follows, separating out region, location, and city information. Note that you can also call a REST end point such as Bing Maps API available at [Find a Location by Point](#) to get the region/district information.

```

select
<location_columnname>
,round(<location_columnname>,0) as l1
,l2=case when LEN(PARSENAME(round(ABS(<location_columnname>) - FLOOR(ABS(<location_columnname>))),6,1)) >= 1 then
substring(PARSENAME(round(ABS(<location_columnname>) - FLOOR(ABS(<location_columnname>))),6,1),1,1) else '0' end
,l3=case when LEN(PARSENAME(round(ABS(<location_columnname>) - FLOOR(ABS(<location_columnname>))),6,1)) >= 2 then
substring(PARSENAME(round(ABS(<location_columnname>) - FLOOR(ABS(<location_columnname>))),6,1),2,1) else '0' end
,l4=case when LEN(PARSENAME(round(ABS(<location_columnname>) - FLOOR(ABS(<location_columnname>))),6,1)) >= 3 then
substring(PARSENAME(round(ABS(<location_columnname>) - FLOOR(ABS(<location_columnname>))),6,1),3,1) else '0' end
,l5=case when LEN(PARSENAME(round(ABS(<location_columnname>) - FLOOR(ABS(<location_columnname>))),6,1)) >= 4 then
substring(PARSENAME(round(ABS(<location_columnname>) - FLOOR(ABS(<location_columnname>))),6,1),4,1) else '0' end
,l6=case when LEN(PARSENAME(round(ABS(<location_columnname>) - FLOOR(ABS(<location_columnname>))),6,1)) >= 5 then
substring(PARSENAME(round(ABS(<location_columnname>) - FLOOR(ABS(<location_columnname>))),6,1),5,1) else '0' end
,l7=case when LEN(PARSENAME(round(ABS(<location_columnname>) - FLOOR(ABS(<location_columnname>))),6,1)) >= 6 then
substring(PARSENAME(round(ABS(<location_columnname>) - FLOOR(ABS(<location_columnname>))),6,1),6,1) else '0' end
from<tablename>

```

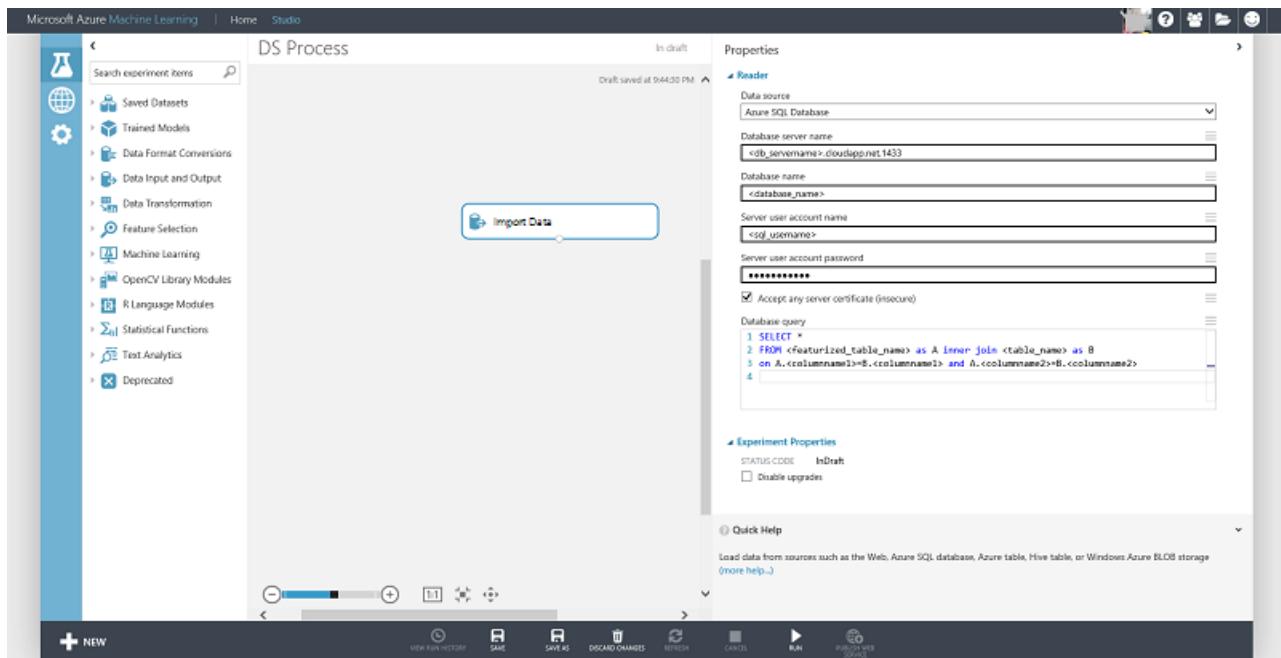
These location-based features can be further used to generate additional count features as described earlier.

TIP

You can programmatically insert the records using your language of choice. You may need to insert the data in chunks to improve write efficiency (for an example of how to do this using pyodbc, see [A HelloWorld sample to access SQLServer with python](#)). Another alternative is to insert data in the database using the [BCP utility](#).

Connecting to Azure Machine Learning

The newly generated feature can be added as a column to an existing table or stored in a new table and joined with the original table for machine learning. Features can be generated or accessed if already created, using the [Import Data](#) module in Azure Machine Learning as shown below:



Using a programming language like Python

Using Python to explore data and generate features when the data is in SQL Server is similar to processing data in Azure blob using Python as documented in [Process Azure Blob data in your data science environment](#). The data needs to be loaded from the database into a pandas data frame and then can be processed further. We document the process of connecting to the database and loading the data into the data frame in this section.

The following connection string format can be used to connect to a SQL Server database from Python using pyodbc (replace servername, dbname, username, and password with your specific values):

```
#Set up the SQL Azure connection
import pyodbc
conn = pyodbc.connect('DRIVER={SQL Server};SERVER=<servername>;DATABASE=<dbname>;UID=<username>;PWD=<password>')
```

The [Pandas library](#) in Python provides a rich set of data structures and data analysis tools for data manipulation for Python programming. The code below reads the results returned from a SQL Server database into a Pandas data frame:

```
# Query database and load the returned results in pandas data frame
data_frame = pd.read_sql("select <columnname1>, <columnname2>... from <tablename>", conn)
```

Now you can work with the Pandas data frame as covered in the article [Process Azure Blob data in your data science environment](#).

Azure Data Science in Action Example

For an end-to-end walkthrough example of the Azure Data Science Process using a public dataset, see [Azure Data Science Process in Action](#).

Cheat sheet for an automated data pipeline for Azure Machine Learning predictions

1/17/2017 • 1 min to read • [Edit on GitHub](#)

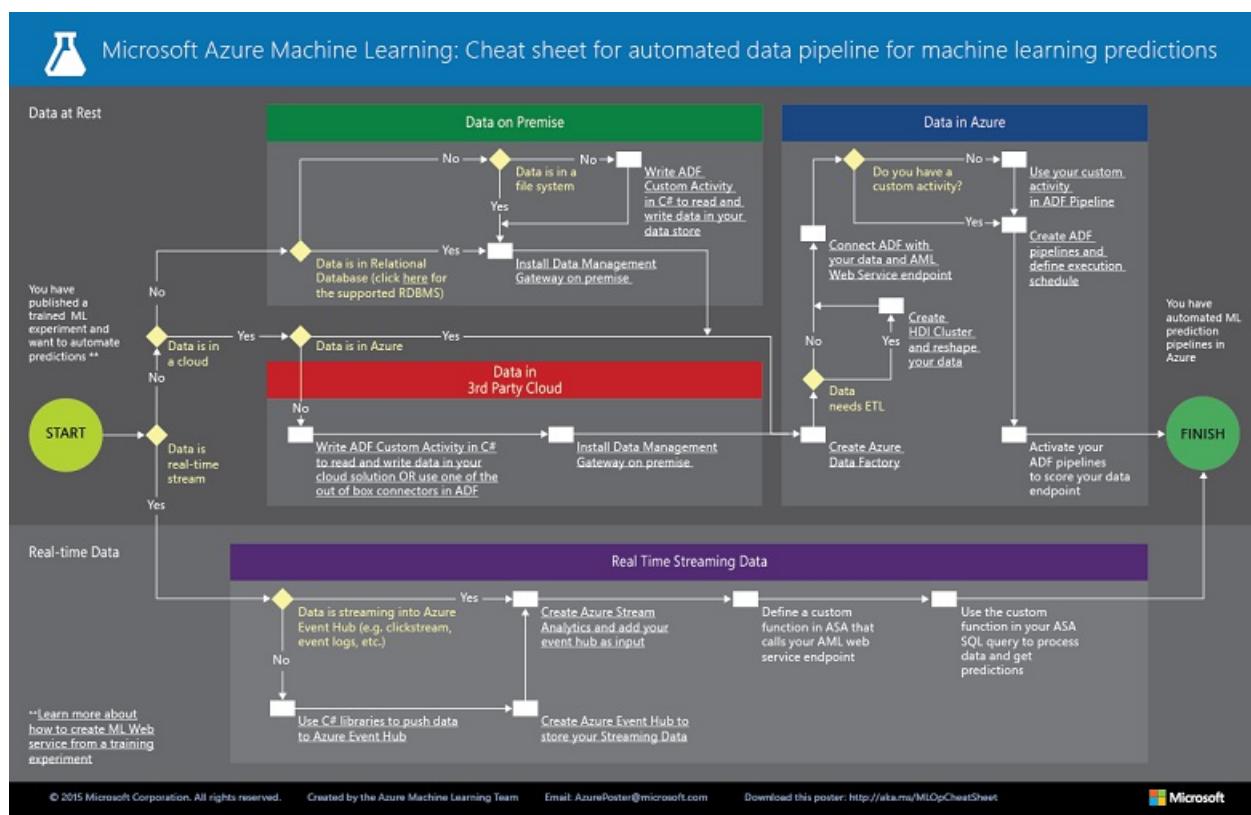
The **Microsoft Azure Machine Learning automated data pipeline cheat sheet** helps you navigate through the technology you can use to get your data to your Machine Learning web service where it can be scored by your predictive analytics model.

Depending on whether your data is on-premises, in the cloud, or streaming real-time, there are different mechanisms available to move the data to your web service endpoint for scoring. This cheat sheet walks you through the decisions you'll need to make and offers links to articles that will help you develop your solution.

Download the Machine Learning automated data pipeline cheat sheet

Once you download the cheat sheet, you can print it in tabloid size (11 x 17 in.).

Download the cheat sheet here: [Microsoft Azure Machine Learning automated data pipeline cheat sheet](#)



More help with Machine Learning Studio

- For an overview of Microsoft Azure Machine Learning, see [Introduction to machine learning on Microsoft Azure](#).
- For an explanation of how to deploy a scoring web service, see [Deploy an Azure Machine Learning web service](#).
- For a discussion of how to consume a scoring web service, see [How to consume an Azure Machine Learning web service that has been deployed from a Machine Learning experiment](#).

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

Overview of data science using Spark on Azure HDInsight

1/17/2017 • 7 min to read • [Edit on GitHub](#)

This suite of topics shows how to use HDInsight Spark to complete common data science tasks such as data ingestion, feature engineering, modeling, and model evaluation. The data used is a sample of the 2013 NYC taxi trip and fare dataset. The models built include logistic and linear regression, random forests, and gradient boosted trees. The topics also show how to store these models in Azure blob storage (WASB) and how to score and evaluate their predictive performance. More advanced topics cover how models can be trained using cross-validation and hyper-parameter sweeping. This overview topic also describes how to set up the Spark cluster that you need to complete the steps in the three walkthroughs provided.

[Spark](#) is an open-source parallel processing framework that supports in-memory processing to boost the performance of big-data analytic applications. Spark processing engine is built for speed, ease of use, and sophisticated analytics. Spark's in-memory distributed computation capabilities make it a good choice for iterative algorithms in machine learning and graph computations. [MLlib](#) is Spark's scalable machine learning library that brings modeling capabilities to this distributed environment.

[HDInsight Spark](#) is the Azure hosted offering of open-source Spark. It also includes support for [Jupyter PySpark notebooks](#) on the Spark cluster that can run Spark SQL interactive queries for transforming, filtering, and visualizing data stored in Azure Blobs (WASB). PySpark is the Python API for Spark. The code snippets that provide the solutions and show the relevant plots to visualize the data here run in Jupyter notebooks installed on the Spark clusters. The modeling steps in these topics contain code that shows how to train, evaluate, save, and consume each type of model.

The setup steps and code provided in this walkthrough is for HDInsight 3.4 Spark 1.6. However, the code here and in the notebooks is generic and should work on any Spark cluster. If you are not using HDInsight Spark, the cluster setup and management steps may be slightly different from what is shown here.

Prerequisites

1. You must have an Azure subscription. If you do not already have one, see [Get Azure free trial](#).
2. You need an HDInsight 3.4 Spark 1.6 cluster to complete this walkthrough. To create one, see the instructions provided in [Get started: create Apache Spark on Azure HDInsight](#). The cluster type and version is specified from the **Select Cluster Type** menu.

The screenshot shows the 'New HDInsight Cluster' configuration page. On the left, there's a sidebar with fields for 'Cluster Name' (with placeholder 'Enter new cluster name'), 'Subscription' (Azure-Irregulars_563702), 'Select Cluster Type' (with a note 'Configure required settings'), 'Credentials', 'Data Source', and 'Node Pricing Tiers'. On the right, the 'Cluster Type configuration' section shows 'Cluster Type' set to 'Spark (Preview)', 'Operating System' set to 'Linux', and 'Version' set to 'Spark 1.6.0 (HDI 3.4)'. Below this, there are two sections: 'STANDARD' and 'PREMIUM (PREVIEW)'. The STANDARD tier includes 'Administration', 'Scalability', '99.9% Uptime SLA', and 'Automatic patching', with a cost of '+ 0.00 USD/CORE/HOUR'. The PREMIUM tier includes 'Administration', 'Scalability', '99.9% Uptime SLA', 'Automatic patching', and 'Microsoft R Server for HDInsight', with a cost of '+ 0.02 USD/CORE/HOUR'. A red circle highlights the 'Spark (Preview)' dropdown, the 'Linux' button, and the 'Spark 1.6.0 (HDI 3.4)' button.

NOTE

For a topic that shows how to use Scala rather than Python to complete tasks for an end-to-end data science process, see the [Data Science using Scala with Spark on Azure](#).

WARNING

HDInsight clusters billing is pro-rated per minute, whether you are using them or not. Please be sure to delete your cluster after you have finished using it. For information on deleting a cluster, see [How to delete an HDInsight cluster](#).

The NYC 2013 Taxi data

The NYC Taxi Trip data is about 20 GB of compressed comma-separated values (CSV) files (~48 GB uncompressed), comprising more than 173 million individual trips and the fares paid for each trip. Each trip record includes the pick up and drop-off location and time, anonymized hack (driver's) license number and medallion (taxi's unique id) number. The data covers all trips in the year 2013 and is provided in the following two datasets for each month:

1. The 'trip_data' CSV files contain trip details, such as number of passengers, pick up and dropoff points, trip duration, and trip length. Here are a few sample records:

```
medallion,hack_license,vendor_id,rate_code,store_and_fwd_flag,pickup_datetime,dropoff_datetime,passenger_count,trip_time_in_secs,trip_distance,pickup_longitude,pickup_latitude,dropoff_longitude,dropoff_latitude
89D227B655E5C82AECF13C3F540D4CF4,BA96DE419E711691B9445D6A6307C170,CMT,1,N,2013-01-01 15:11:48,2013-01-01:18:10,4,382,1.00,-73.978165,40.757977,-73.989838,40.751171
0BD7C8F5BA12B88E0B67BED28BEA73D8,9FD8F69F0804BDB5549F40E9DA1BE472,CMT,1,N,2013-01-06 00:18:35,2013-01-06:02:54,1,259,1.50,-74.006683,40.731781,-73.994499,40.75066
0BD7C8F5BA12B88E0B67BED28BEA73D8,9FD8F69F0804BDB5549F40E9DA1BE472,CMT,1,N,2013-01-05 18:49:41,2013-01-05:18:54,1,282,1.10,-74.004707,40.73777,-74.009834,40.726002
DFD2202EE08F7A8DC9A57B02ACB81FE2,51EE87E3205C985EF8431D850C786310,CMT,1,N,2013-01-07 23:54:15,2013-01-07:23:58:20,2,244,.70,-73.974602,40.759945,-73.984734,40.759388
DFD2202EE08F7A8DC9A57B02ACB81FE2,51EE87E3205C985EF8431D850C786310,CMT,1,N,2013-01-07 23:25:03,2013-01-07:23:34:24,1,560,2.10,-73.97625,40.748528,-74.002586,40.747868
```

2. The 'trip_fare' CSV files contain details of the fare paid for each trip, such as payment type, fare amount,

surcharge and taxes, tips and tolls, and the total amount paid. Here are a few sample records:

```
medallion, hack_license, vendor_id, pickup_datetime, payment_type, fare_amount, surcharge, mta_tax, tip_amount, tolls_amount,
total_amount
89D227B655E5C82AECF13C3F540D4CF4,BA96DE419E711691B9445D6A6307C170,CMT,2013-01-01 15:11:48,CSH,6.5,0,0.5,0,0.7
0BD7C8F5BA12B88E0B67BED28BEA73D8,9FD8F69F0804BDB5549F40E9DA1BE472,CMT,2013-01-06 00:18:35,CSH,6,0.5,0.5,0,0.7
0BD7C8F5BA12B88E0B67BED28BEA73D8,9FD8F69F0804BDB5549F40E9DA1BE472,CMT,2013-01-05 18:49:41,CSH,5.5,1,0.5,0,0.7
DFD2202EE08F7A8DC9A57B02ACB81FE2,51EE87E3205C985EF8431D850C786310,CMT,2013-01-07 23:54:15,CSH,5,0.5,0.5,0,0.6
DFD2202EE08F7A8DC9A57B02ACB81FE2,51EE87E3205C985EF8431D850C786310,CMT,2013-01-07 23:25:03,CSH,9.5,0.5,0.5,0,0.10.5
```

We have taken a 0.1% sample of these files and joined the trip_data and trip_fare CVS files into a single dataset to use as the input dataset for this walkthrough. The unique key to join trip_data and trip_fare is composed of the fields: medallion, hack_licence and pickup_datetime. Each record of the dataset contains the following attributes representing a NYC Taxi trip:

FIELD	BRIEF DESCRIPTION
medallion	Anonymized taxi medallion (unique taxi id)
hack_license	Anonymized Hackney Carriage License number
vendor_id	Taxi vendor id
rate_code	NYC taxi rate of fare
store_and_fwd_flag	Store and forward flag
pickup_datetime	Pick up date & time
dropoff_datetime	Dropoff date & time
pickup_hour	Pick up hour
pickup_week	Pick up week of the year
weekday	Weekday (range 1-7)
passenger_count	Number of passengers in a taxi trip
trip_time_in_secs	Trip time in seconds
trip_distance	Trip distance traveled in miles
pickup_longitude	Pick up longitude
pickup_latitude	Pick up latitude
dropoff_longitude	Dropoff longitude
dropoff_latitude	Dropoff latitude
direct_distance	Direct distance between pick up and dropoff locations
payment_type	Payment type (cas, credit-card etc.)

FIELD	BRIEF DESCRIPTION
fare_amount	Fare amount in
surcharge	Surcharge
mta_tax	Mta tax
tip_amount	Tip amount
tolls_amount	Tolls amount
total_amount	Total amount
tipped	Tipped (0/1 for no or yes)
tip_class	Tip class (0: \$0, 1: \$0-5, 2: \$6-10, 3: \$11-20, 4: > \$20)

Execute code from a Jupyter notebook on the Spark cluster

You can launch the Jupyter Notebook from the Azure portal. Find your Spark cluster on your dashboard and click it to enter management page for your cluster. To open the notebook associated with the Spark cluster, click

Cluster Dashboards -> Jupyter Notebook .

The screenshot shows two windows side-by-side. The left window is titled 'HDInsight Cluster' and displays cluster details: Resource group [REDACTED], Status Running, Location South Central US, Subscription name [REDACTED], and Subscription ID [REDACTED]. It also shows 'Quick Links' for Cluster Dashboards, Ambari Views, and Scale Cluster. A large donut chart under 'Usage' shows cores: THIS CLUSTER 12, OTHER CLUSTERS 756, AVAILABLE 1332, and TOTAL 2100. The right window is titled 'Cluster Dashboards' and lists four tiles: 'HDInsight Cluster Dashboard' (with a yellow elephant icon), 'Jupyter Notebook' (with an orange Jupyter logo), 'Spark History Server' (with a blue square icon), and 'Yarn' (with a blue square icon). The 'Jupyter Notebook' tile is circled in red.

You can also browse to <https://CLUSTERNAME.azurehdinsight.net/jupyter> to access the Jupyter Notebooks. Replace the CLUSTERNAME part of this URL with the name of your own cluster. You need the password for your admin account to access the notebooks.

The screenshot shows the Jupyter interface with a file explorer on the left. The 'PySpark' directory is selected and highlighted with a red circle. On the right, there is a toolbar with 'Upload' (circled in red), 'New', and other options.

Select PySpark to see a directory that contains a few examples of pre-packaged notebooks that use the PySpark API. The notebooks that contain the code samples for this suite of Spark topic are available at [Github](#)

You can upload the notebooks directly from Github to the Jupyter notebook server on your Spark cluster. On the home page of your Jupyter, click the **Upload** button on the right part of the screen. It opens a file explorer. Here you can paste the Github (raw content) URL of the Notebook and click **Open**. The PySpark notebooks are available at the following URLs:

1. [pySpark-machine-learning-data-science-spark-data-exploration-modeling.ipynb](#)
2. [pySpark-machine-learning-data-science-spark-model-consumption.ipynb](#)
3. [pySpark-machine-learning-data-science-spark-advanced-data-exploration-modeling.ipynb](#)

You see the file name on your Jupyter file list with an **Upload** button again. Click this **Upload** button. Now you have imported the notebook. Repeat these steps to upload the other notebooks from this walkthrough.

TIP

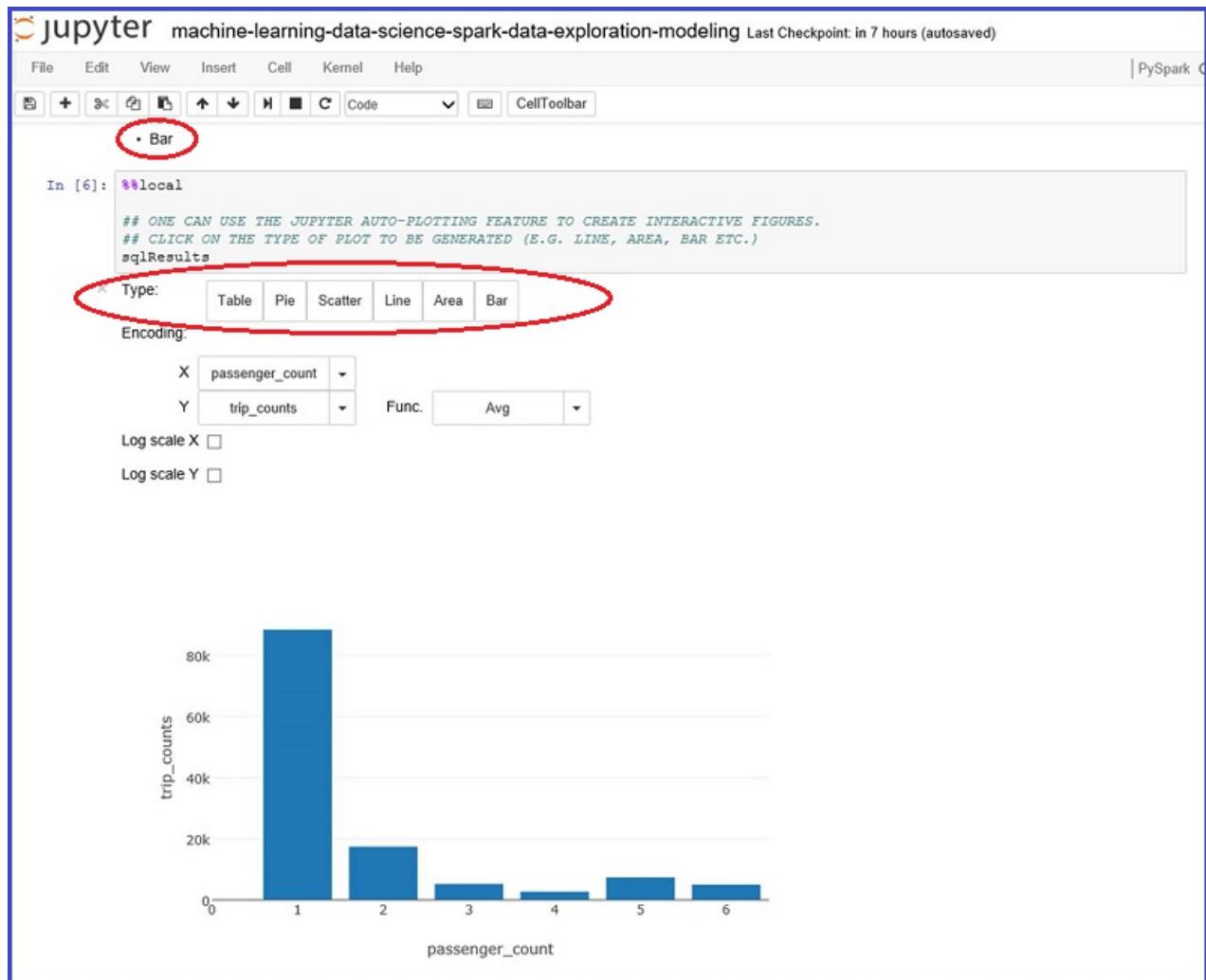
You can right-click the links on your browser and select **Copy Link** to get the github raw content URL. You can paste this URL into the Jupyter Upload file explorer dialog box.

Now you can:

- See the code by clicking the notebook.
- Execute each cell by pressing **SHIFT-ENTER**.
- Run the entire notebook by clicking on **Cell -> Run**.
- Use the automatic visualization of queries.

TIP

The PySpark kernel automatically visualizes the output of SQL (HiveQL) queries. You are given the option to select among several different types of visualizations (Table, Pie, Line, Area, or Bar) by using the **Type** menu buttons in the notebook:



What's next?

Now that you are set up with an HDInsight Spark cluster and have uploaded the Jupyter notebooks, you are ready to work through the topics that correspond to the three PySpark notebooks. They show how to explore your data and then how to create and consume models. The advanced data exploration and modeling notebook shows how

to include cross-validation, hyper-parameter sweeping, and model evaluation.

Data Exploration and modeling with Spark: Explore the dataset and create, score, and evaluate the machine learning models by working through the [Create binary classification and regression models for data with the Spark MLlib toolkit](#) topic.

Model consumption: To learn how to score the classification and regression models created in this topic, see [Score and evaluate Spark-built machine learning models](#).

Cross-validation and hyperparameter sweeping: See [Advanced data exploration and modeling with Spark](#) on how models can be trained using cross-validation and hyper-parameter sweeping

Data exploration and modeling with Spark

1/17/2017 • 29 min to read • [Edit on GitHub](#)

This walkthrough uses HDInsight Spark to do data exploration and binary classification and regression modeling tasks on a sample of the NYC taxi trip and fare 2013 dataset. It walks you through the steps of the [Data Science Process](#), end-to-end, using an HDInsight Spark cluster for processing and Azure blobs to store the data and the models. The process explores and visualizes data brought in from an Azure Storage Blob and then prepares the data to build predictive models. These models are built using the Spark MLlib toolkit to do binary classification and regression modeling tasks.

- The **binary classification** task is to predict whether or not a tip is paid for the trip.
- The **regression** task is to predict the amount of the tip based on other tip features.

The models we use include logistic and linear regression, random forests, and gradient boosted trees:

- [Linear regression with SGD](#) is a linear regression model that uses a Stochastic Gradient Descent (SGD) method and feature scaling to predict the tip amounts paid.
- [Logistic regression with LBFGS](#) or "logit" regression, is a regression model that can be used when the dependent variable is categorical to do data classification. LBFGS is a quasi-Newton optimization algorithm that approximates the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm using a limited amount of computer memory and that is widely used in machine learning.
- [Random forests](#) are ensembles of decision trees. They combine many decision trees to reduce the risk of overfitting. Random forests are used for regression and classification and can handle categorical features and can be extended to the multiclass classification setting. They do not require feature scaling and are able to capture non-linearities and feature interactions. Random forests are one of the most successful machine learning models for classification and regression.
- [Gradient boosted trees](#) (GBTs) are ensembles of decision trees. GBTs train decision trees iteratively to minimize a loss function. GBTs are used for regression and classification and can handle categorical features, do not require feature scaling, and are able to capture non-linearities and feature interactions. They can also be used in a multiclass-classification setting.

The modeling steps also contain code showing how to train, evaluate, and save each type of model. Python has been used to code the solution and to show the relevant plots.

NOTE

Although the Spark MLlib toolkit is designed to work on large datasets, a relatively small sample (~30 Mb using 170K rows, about 0.1% of the original NYC dataset) is used here for convenience. The exercise given here runs efficiently (in about 10 minutes) on an HDInsight cluster with 2 worker nodes. The same code, with minor modifications, can be used to process larger data-sets, with appropriate modifications for caching data in memory and changing the cluster size.

Prerequisites

You need an Azure account and an HDInsight Spark. You need an HDInsight 3.4 Spark 1.6 cluster to complete this walkthrough. See the [Overview of Data Science using Spark on Azure HDInsight](#) for instructions on how to satisfy these requirements. That topic also contains a description of the NYC 2013 Taxi data used here and instructions on how to execute code from a Jupyter notebook on the Spark cluster. The [pySpark-machine-learning-data-science-spark-data-exploration-modeling.ipynb](#) notebook that contains the code samples in this topic is available in [Github](#).

WARNING

HDInsight clusters billing is pro-rated per minute, whether you are using them or not. Please be sure to delete your cluster after you have finished using it. For information on deleting a cluster, see [How to delete an HDInsight cluster](#).

Setup: storage locations, libraries, and the preset Spark context

Spark is able to read and write to Azure Storage Blob (also known as WASB). So any of your existing data stored there can be processed using Spark and the results stored again in WASB.

To save models or files in WASB, the path needs to be specified properly. The default container attached to the Spark cluster can be referenced using a path beginning with: "wasb:///". Other locations are referenced by "wasb://".

Set directory paths for storage locations in WASB

The following code sample specifies the location of the data to be read and the path for the model storage directory to which the model output is saved:

```
# SET PATHS TO FILE LOCATIONS: DATA AND MODEL STORAGE

# LOCATION OF TRAINING DATA
taxi_train_file_loc = "wasb://mllibwalkthroughs@cdsparksamples.blob.core.windows.net/Data/NYCTaxi/JoinedTaxiTripFare.Point1Pct.Train.tsv";

# SET THE MODEL STORAGE DIRECTORY PATH
# NOTE THAT THE FINAL BACKSLASH IN THE PATH IS NEEDED.
modelDir = "wasb://user/remoteuser/NYCTaxi/Models/"
```

Import libraries

Set up also requires importing necessary libraries. Set spark context and import necessary libraries with the following code:

```
# IMPORT LIBRARIES
import pyspark
from pyspark import SparkConf
from pyspark import SparkContext
from pyspark.sql import SQLContext
import matplotlib
import matplotlib.pyplot as plt
from pyspark.sql import Row
from pyspark.sql.functions import UserDefinedFunction
from pyspark.sql.types import *
import atexit
from numpy import array
import numpy as np
import datetime
```

Preset Spark context and PySpark magics

The PySpark kernels that are provided with Jupyter notebooks have a preset context. So you do not need to set the Spark or Hive contexts explicitly before you start working with the application you are developing. These contexts are available for you by default. These contexts are:

- sc - for Spark
- sqlContext - for Hive

The PySpark kernel provides some predefined "magics", which are special commands that you can call with %. There are two such commands that are used in these code samples.

- **%%local** Specifies that the code in subsequent lines is to be executed locally. Code must be valid Python code.
- **%%sql -o** Executes a Hive query against the sqlContext. If the -o parameter is passed, the result of the query is persisted in the %%local Python context as a Pandas DataFrame.

For more information on the kernels for Jupyter notebooks and the predefined "magics" that they provide, see [Kernels available for Jupyter notebooks with HDInsight Spark Linux clusters on HDInsight](#).

Data ingestion from public blob

The first step in the data science process is to ingest the data to be analyzed from sources where it resides into your data exploration and modeling environment. The environment is Spark in this walkthrough. This section contains the code to complete a series of tasks:

- ingest the data sample to be modeled
- read in the input dataset (stored as a .tsv file)
- format and clean the data
- create and cache objects (RDDs or data-frames) in memory
- register it as a temp-table in SQL-context.

Here is the code for data ingestion.

```

# INGEST DATA

# RECORD START TIME
timestart = datetime.datetime.now()

# IMPORT FILE FROM PUBLIC BLOB
taxi_train_file = sc.textFile(taxi_train_file_loc)

# GET SCHEMA OF THE FILE FROM HEADER
schema_string = taxi_train_file.first()
fields = [StructField(field_name, StringType(), True) for field_name in schema_string.split("\t")]
fields[7].dataType = IntegerType() # Pickup hour
fields[8].dataType = IntegerType() # Pickup week
fields[9].dataType = IntegerType() # Weekday
fields[10].dataType = IntegerType() # Passenger count
fields[11].dataType = FloatType() # Trip time in secs
fields[12].dataType = FloatType() # Trip distance
fields[19].dataType = FloatType() # Fare amount
fields[20].dataType = FloatType() # Surcharge
fields[21].dataType = FloatType() # Mta_tax
fields[22].dataType = FloatType() # Tip amount
fields[23].dataType = FloatType() # Tolls amount
fields[24].dataType = FloatType() # Total amount
fields[25].dataType = IntegerType() # Tipped or not
fields[26].dataType = IntegerType() # Tip class
taxi_schema = StructType(fields)

# PARSE FIELDS AND CONVERT DATA TYPE FOR SOME FIELDS
taxi_header = taxi_train_file.filter(lambda l: "medallion" in l)
taxi_temp = taxi_train_file.subtract(taxi_header).map(lambda k: k.split("\t"))
.map(lambda p: (p[0],p[1],p[2],p[3],p[4],p[5],p[6],int(p[7]),int(p[8]),int(p[9]),int(p[10]),
float(p[11]),float(p[12]),p[13],p[14],p[15],p[16],p[17],p[18],float(p[19]),
float(p[20]),float(p[21]),float(p[22]),float(p[23]),float(p[24]),int(p[25]),int(p[26])))

# CREATE DATA FRAME
taxi_train_df = sqlContext.createDataFrame(taxi_temp, taxi_schema)

# CREATE A CLEANED DATA-FRAME BY DROPPING SOME UN-NECESSARY COLUMNS & FILTERING FOR UNDESIRED VALUES OR
OUTLIERS
taxi_df_train_cleaned = taxi_train_df.drop('medallion').drop('hack_license').drop('store_and_fwd_flag').drop('pickup_datetime')\
.drop('dropoff_datetime').drop('pickup_longitude').drop('pickup_latitude').drop('dropoff_latitude')\
.drop('dropoff_longitude').drop('tip_class').drop('total_amount').drop('tolls_amount').drop('mta_tax')\
.drop('direct_distance').drop('surcharge')\
.filter("passenger_count > 0 and passenger_count < 8 AND payment_type in ('CSH', 'CRD') AND tip_amount >= 0 AND tip_amount < 30 AND
fare_amount >= 1 AND fare_amount < 150 AND trip_distance > 0 AND trip_distance < 100 AND trip_time_in_secs > 30 AND trip_time_in_secs <
7200" )

# CACHE DATA-FRAME IN MEMORY & MATERIALIZE DF IN MEMORY
taxi_df_train_cleaned.cache()
taxi_df_train_cleaned.count()

# REGISTER DATA-FRAME AS A TEMP-TABLE IN SQL-CONTEXT
taxi_df_train_cleaned.registerTempTable("taxi_train")

# PRINT HOW MUCH TIME IT TOOK TO RUN THE CELL
timeend = datetime.datetime.now()
timedelta = round((timeend-timestart).total_seconds(), 2)
print "Time taken to execute above cell: " + str(timedelta) + " seconds";

```

OUTPUT:

Time taken to execute above cell: 51.72 seconds

Data exploration & visualization

Once the data has been brought into Spark, the next step in the data science process is to gain deeper understanding of the data through exploration and visualization. In this section, we examine the taxi data using SQL queries and plot the target variables and prospective features for visual inspection. Specifically, we plot the frequency of passenger counts in taxi trips, the frequency of tip amounts, and how tips vary by payment amount and type.

Plot a histogram of passenger count frequencies in the sample of taxi trips

This code and subsequent snippets use SQL magic to query the sample and local magic to plot the data.

- **SQL magic (`%%sql`)** The HDInsight PySpark kernel supports easy inline HiveQL queries against the `sqlContext`. The `-o VARIABLE_NAME` argument persists the output of the SQL query as a Pandas DataFrame on the Jupyter server. This means it is available in the local mode.
- The `%%local` **magic** is used to run code locally on the Jupyter server, which is the headnode of the HDInsight cluster. Typically, you use `%%local` magic in conjunction with the `%%sql` magic with `-o` parameter. The `-o` parameter would persist the output of the SQL query locally and then `%%local` magic would trigger the next set of code snippet to run locally against the output of the SQL queries that is persisted locally

The output is automatically visualized after you run the code.

This query retrieves the trips by passenger count.

```
# PLOT FREQUENCY OF PASSENGER COUNTS IN TAXI TRIPS

# HIVEQL QUERY AGAINST THE sqlContext
%%sql -q -o sqlResults
SELECT passenger_count, COUNT(*) as trip_counts
FROM taxi_train
WHERE passenger_count > 0 and passenger_count < 7
GROUP BY passenger_count
```

This code creates a local data-frame from the query output and plots the data. The `%%local` magic creates a local data-frame, `sqlResults`, which can be used for plotting with matplotlib.

NOTE

This PySpark magic is used multiple times in this walkthrough. If the amount of data is large, you should sample to create a data-frame that can fit in local memory.

```
#CREATE LOCAL DATA-FRAME AND USE FOR MATPLOTLIB PLOTTING

# RUN THE CODE LOCALLY ON THE JUPYTER SERVER
%%local

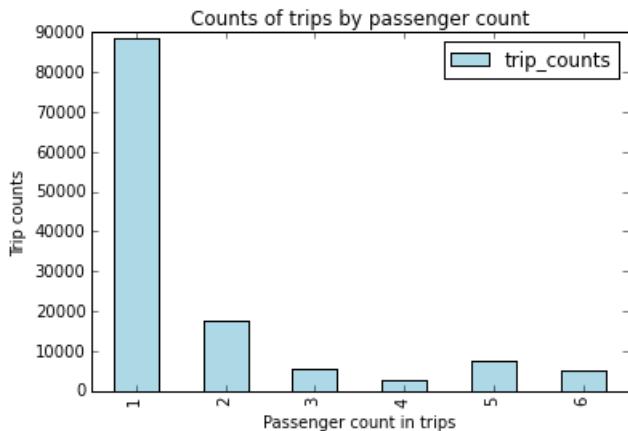
# USE THE JUPYTER AUTO-PLOTTING FEATURE TO CREATE INTERACTIVE FIGURES.
# CLICK ON THE TYPE OF PLOT TO BE GENERATED (E.G. LINE, AREA, BAR ETC.)
sqlResults
```

Here is the code to plot the trips by passenger counts

```
# PLOT PASSENGER NUMBER VS. TRIP COUNTS
%%local
import matplotlib.pyplot as plt
%matplotlib inline

x_labels = sqlResults['passenger_count'].values
fig = sqlResults[['trip_counts']].plot(kind='bar', facecolor='lightblue')
fig.set_xicklabels(x_labels)
fig.set_title('Counts of trips by passenger count')
fig.set_xlabel('Passenger count in trips')
fig.set_ylabel('Trip counts')
plt.show()
```

OUTPUT:



You can select among several different types of visualizations (Table, Pie, Line, Area, or Bar) by using the **Type** menu buttons in the notebook. The Bar plot is shown here.

Plot a histogram of tip amounts and how tip amount varies by passenger count and fare amounts.

Use a SQL query to sample data.

```
#PLOT HISTOGRAM OF TIP AMOUNTS AND VARIATION BY PASSENGER COUNT AND PAYMENT TYPE

# HIVEQL QUERY AGAINST THE sqlContext
%%sql -q -o sqlResults
SELECT fare_amount, passenger_count, tip_amount, tipped
FROM taxi_train
WHERE passenger_count > 0
AND passenger_count < 7
AND fare_amount > 0
AND fare_amount < 200
AND payment_type in ('CSH', 'CRD')
AND tip_amount > 0
AND tip_amount < 25
```

This code cell uses the SQL query to create three plots the data.

```

# RUN THE CODE LOCALLY ON THE JUPYTER SERVER
%%local

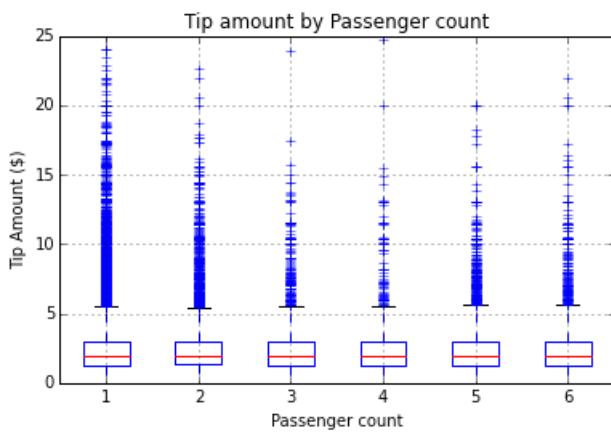
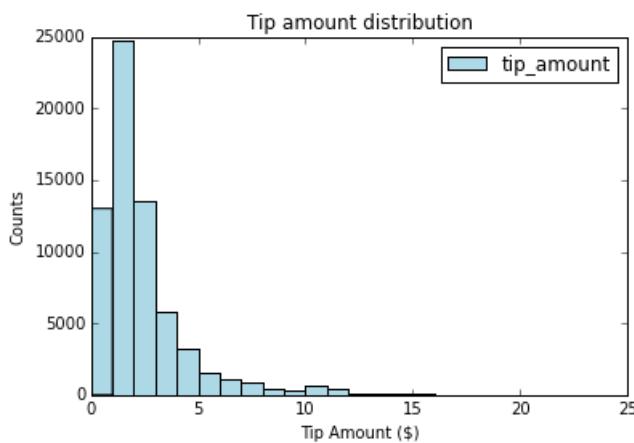
# HISTOGRAM OF TIP AMOUNTS AND PASSENGER COUNT
ax1 = sqlResults[['tip_amount']].plot(kind='hist', bins=25, facecolor='lightblue')
ax1.set_title('Tip amount distribution')
ax1.set_xlabel('Tip Amount ($)')
ax1.set_ylabel('Counts')
plt.suptitle('')
plt.show()

# TIP BY PASSENGER COUNT
ax2 = sqlResults.boxplot(column=['tip_amount'], by=[passenger_count'])
ax2.set_title('Tip amount by Passenger count')
ax2.set_xlabel('Passenger count')
ax2.set_ylabel('Tip Amount ($)')
plt.suptitle('')
plt.show()

# TIP AMOUNT BY FARE AMOUNT, POINTS ARE SCALED BY PASSENGER COUNT
ax = sqlResults.plot(kind='scatter', x='fare_amount', y='tip_amount', c='blue', alpha = 0.10, s=5*(sqlResults.passenger_count))
ax.set_title('Tip amount by Fare amount')
ax.set_xlabel('Fare Amount ($)')
ax.set_ylabel('Tip Amount ($)')
plt.axis([-2, 100, -2, 20])
plt.show()

```

OUTPUT:





Feature engineering, transformation and data preparation for modeling

This section describes and provides the code for procedures used to prepare data for use in ML modeling. It shows how to do the following tasks:

- Create a new feature by binning hours into traffic time buckets
- Index and encode categorical features
- Create labeled point objects for input into ML functions
- Create a random sub-sampling of the data and split it into training and testing sets
- Feature scaling
- Cache objects in memory

Create a new feature by binning hours into traffic time buckets

This code shows how to create a new feature by binning hours into traffic time buckets and then how to cache the resulting data frame in memory. Where Resilient Distributed Datasets (RDDs) and data-frames are used repeatedly, caching leads to improved execution times. Accordingly, we cache RDDs and data-frames at several stages in the walkthrough.

```
# CREATE FOUR BUCKETS FOR TRAFFIC TIMES
sqlStatement = """
SELECT *,
CASE
WHEN (pickup_hour <= 6 OR pickup_hour >= 20) THEN "Night"
WHEN (pickup_hour >= 7 AND pickup_hour <= 10) THEN "AMRush"
WHEN (pickup_hour >= 11 AND pickup_hour <= 15) THEN "Afternoon"
WHEN (pickup_hour >= 16 AND pickup_hour <= 19) THEN "PMRush"
END as TrafficTimeBins
FROM taxi_train
"""

taxi_df_train_with_newFeatures = sqlContext.sql(sqlStatement)

# CACHE DATA-FRAME IN MEMORY & MATERIALIZE DF IN MEMORY
# THE .COUNT() GOES THROUGH THE ENTIRE DATA-FRAME,
# MATERIALIZES IT IN MEMORY, AND GIVES THE COUNT OF ROWS.
taxi_df_train_with_newFeatures.cache()
taxi_df_train_with_newFeatures.count()
```

OUTPUT:

126050

Index and encode categorical features for input into modeling functions

This section shows how to index or encode categorical features for input into the modeling functions. The modeling and predict functions of MLlib require features with categorical input data to be indexed or encoded prior to use.

Depending on the model, you need to index or encode them in different ways:

- **Tree-based modeling** requires categories to be encoded as numerical values (for example, a feature with three categories may be encoded with 0, 1, 2). This is provided by MLlib's [StringIndexer](#) function. This function encodes a string column of labels to a column of label indices that are ordered by label frequencies. Although indexed with numerical values for input and data handling, the tree-based algorithms can be specified to treat them appropriately as categories.
- **Logistic and Linear Regression models** require one-hot encoding, where, for example, a feature with three categories can be expanded into three feature columns, with each containing 0 or 1 depending on the category of an observation. MLlib provides [OneHotEncoder](#) function to do one-hot encoding. This encoder maps a column of label indices to a column of binary vectors, with at most a single one-value. This encoding allows algorithms that expect numerical valued features, such as logistic regression, to be applied to categorical features.

Here is the code to index and encode categorical features:

```
# INDEX AND ENCODE CATEGORICAL FEATURES

# RECORD START TIME
timestart = datetime.datetime.now()

# LOAD PYSPARK LIBRARIES
from pyspark.ml.feature import OneHotEncoder, StringIndexer, VectorAssembler, VectorIndexer

# INDEX AND ENCODE VENDOR_ID
stringIndexer = StringIndexer(inputCol="vendor_id", outputCol="vendorIndex")
model = stringIndexer.fit(taxi_df_train_with_newFeatures) # Input data-frame is the cleaned one from above
indexed = model.transform(taxi_df_train_with_newFeatures)
encoder = OneHotEncoder(dropLast=False, inputCol="vendorIndex", outputCol="vendorVec")
encoded1 = encoder.transform(indexed)

# INDEX AND ENCODE RATE_CODE
stringIndexer = StringIndexer(inputCol="rate_code", outputCol="rateIndex")
model = stringIndexer.fit(encoded1)
indexed = model.transform(encoded1)
encoder = OneHotEncoder(dropLast=False, inputCol="rateIndex", outputCol="rateVec")
encoded2 = encoder.transform(indexed)

# INDEX AND ENCODE PAYMENT_TYPE
stringIndexer = StringIndexer(inputCol="payment_type", outputCol="paymentIndex")
model = stringIndexer.fit(encoded2)
indexed = model.transform(encoded2)
encoder = OneHotEncoder(dropLast=False, inputCol="paymentIndex", outputCol="paymentVec")
encoded3 = encoder.transform(indexed)

# INDEX AND TRAFFIC TIME BINS
stringIndexer = StringIndexer(inputCol="TrafficTimeBins", outputCol="TrafficTimeBinsIndex")
model = stringIndexer.fit(encoded3)
indexed = model.transform(encoded3)
encoder = OneHotEncoder(dropLast=False, inputCol="TrafficTimeBinsIndex", outputCol="TrafficTimeBinsVec")
encodedFinal = encoder.transform(indexed)

# PRINT ELAPSED TIME
timeend = datetime.datetime.now()
timedelta = round((timeend-timestart).total_seconds(), 2)
print "Time taken to execute above cell: " + str(timedelta) + " seconds";
```

OUTPUT:

Time taken to execute above cell: 1.28 seconds

Create labeled point objects for input into ML functions

This section contains code that shows how to index categorical text data as a labeled point data type and encode it

so that it can be used to train and test MLlib logistic regression and other classification models. Labeled point objects are Resilient Distributed Datasets (RDD) formatted in a way that is needed as input data by most of ML algorithms in MLlib. A [labeled point](#) is a local vector, either dense or sparse, associated with a label/response.

This section contains code that shows how to index categorical text data as a [labeled point](#) data type and encode it so that it can be used to train and test MLlib logistic regression and other classification models. Labeled point objects are Resilient Distributed Datasets (RDD) consisting of a label (target/response variable) and feature vector. This format is needed as input by many ML algorithms in MLlib.

Here is the code to index and encode text features for binary classification.

```
# FUNCTIONS FOR BINARY CLASSIFICATION

# LOAD LIBRARIES
from pyspark.mllib.regression import LabeledPoint
from numpy import array

# INDEXING CATEGORICAL TEXT FEATURES FOR INPUT INTO TREE-BASED MODELS
def parseRowIndexingBinary(line):
    features = np.array([line.paymentIndex, line.vendorIndex, line.rateIndex, line.TrafficTimeBinsIndex,
                        line.pickup_hour, line.weekday, line.passenger_count, line.trip_time_in_secs,
                        line.trip_distance, line.fare_amount])
    labPt = LabeledPoint(line.tipped, features)
    return labPt

# ONE-HOT ENCODING OF CATEGORICAL TEXT FEATURES FOR INPUT INTO LOGISTIC REGRESSION MODELS
def parseRowOneHotBinary(line):
    features = np.concatenate((np.array([line.pickup_hour, line.weekday, line.passenger_count,
                                         line.trip_time_in_secs, line.trip_distance, line.fare_amount]),
                               line.vendorVec.toArray(), line.rateVec.toArray(),
                               line.paymentVec.toArray(), line.TrafficTimeBinsVec.toArray()), axis=0)
    labPt = LabeledPoint(line.tipped, features)
    return labPt
```

Here is the code to encode and index categorical text features for linear regression analysis.

```
# FUNCTIONS FOR REGRESSION WITH TIP AMOUNT AS TARGET VARIABLE

# ONE-HOT ENCODING OF CATEGORICAL TEXT FEATURES FOR INPUT INTO TREE-BASED MODELS
def parseRowIndexingRegression(line):
    features = np.array([line.paymentIndex, line.vendorIndex, line.rateIndex, line.TrafficTimeBinsIndex,
                        line.pickup_hour, line.weekday, line.passenger_count, line.trip_time_in_secs,
                        line.trip_distance, line.fare_amount])

    labPt = LabeledPoint(line.tip_amount, features)
    return labPt

# INDEXING CATEGORICAL TEXT FEATURES FOR INPUT INTO LINEAR REGRESSION MODELS
def parseRowOneHotRegression(line):
    features = np.concatenate((np.array([line.pickup_hour, line.weekday, line.passenger_count,
                                         line.trip_time_in_secs, line.trip_distance, line.fare_amount]),
                               line.vendorVec.toArray(), line.rateVec.toArray(),
                               line.paymentVec.toArray(), line.TrafficTimeBinsVec.toArray()), axis=0)
    labPt = LabeledPoint(line.tip_amount, features)
    return labPt
```

Create a random sub-sampling of the data and split it into training and testing sets

This code creates a random sampling of the data (25% is used here). Although it is not required for this example due to the size of the dataset, we demonstrate how you can sample here so you know how to use it for your own problem when needed. When samples are large, this can save significant time while training models. Next we split the sample into a training part (75% here) and a testing part (25% here) to use in classification and regression modeling.

```

# RECORD START TIME
timestart = datetime.datetime.now()

# LOAD PYSPARK LIBRARIES
from pyspark.sql.functions import rand

# SPECIFY SAMPLING AND SPLITTING FRACTIONS
samplingFraction = 0.25;
trainingFraction = 0.75; testingFraction = (1-trainingFraction);
seed = 1234;
encodedFinalSampled = encodedFinal.sample(False, samplingFraction, seed=seed)

# SPLIT SAMPLED DATA-FRAME INTO TRAIN/TEST
# INCLUDE RAND COLUMN FOR CREATING CROSS-VALIDATION FOLDS (FOR USE LATER IN AN ADVANCED TOPIC)
dfTmpRand = encodedFinalSampled.select("*", rand(0).alias("rand"));
trainData, testData = dfTmpRand.randomSplit([trainingFraction, testingFraction], seed=seed);

# FOR BINARY CLASSIFICATION TRAINING AND TESTING
indexedTRAINbinary = trainData.map(parseRowIndexingBinary)
indexedTESTbinary = testData.map(parseRowIndexingBinary)
oneHotTRAINbinary = trainData.map(parseRowOneHotBinary)
oneHotTESTbinary = testData.map(parseRowOneHotBinary)

# FOR REGRESSION TRAINING AND TESTING
indexedTRAINreg = trainData.map(parseRowIndexingRegression)
indexedTESTreg = testData.map(parseRowIndexingRegression)
oneHotTRAINreg = trainData.map(parseRowOneHotRegression)
oneHotTESTreg = testData.map(parseRowOneHotRegression)

# PRINT ELAPSED TIME
timeend = datetime.datetime.now()
timedelta = round((timeend-timestart).total_seconds(), 2)
print "Time taken to execute above cell: " + str(timedelta) + " seconds";

```

OUTPUT:

Time taken to execute above cell: 0.24 seconds

Feature scaling

Feature scaling, also known as data normalization, insures that features with widely disbursed values are not given excessive weigh in the objective function. The code for feature scaling uses the [StandardScaler](#) to scale the features to unit variance. It is provided by MLLib for use in linear regression with Stochastic Gradient Descent (SGD), a popular algorithm for training a wide range of other machine learning models such as regularized regressions or support vector machines (SVM).

NOTE

We have found the `LinearRegressionWithSGD` algorithm to be sensitive to feature scaling.

Here is the code to scale variables for use with the regularized linear SGD algorithm.

```

# FEATURE SCALING

# RECORD START TIME
timestart = datetime.datetime.now()

# LOAD PYSPARK LIBRARIES
from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.linalg import Vectors
from pyspark.mllib.feature import StandardScaler, StandardScalerModel
from pyspark.mllib.util import MLUtils

# SCALE VARIABLES FOR REGULARIZED LINEAR SGD ALGORITHM
label = oneHotTRAINreg.map(lambda x: x.label)
features = oneHotTRAINreg.map(lambda x: x.features)
scaler = StandardScaler(withMean=False, withStd=True).fit(features)
dataTMP = label.zip(scaler.transform(features.map(lambda x: Vectors.dense(x.toArray()))))
oneHotTRAINregScaled = dataTMP.map(lambda x: LabeledPoint(x[0], x[1]))

label = oneHotTESTreg.map(lambda x: x.label)
features = oneHotTESTreg.map(lambda x: x.features)
scaler = StandardScaler(withMean=False, withStd=True).fit(features)
dataTMP = label.zip(scaler.transform(features.map(lambda x: Vectors.dense(x.toArray()))))
oneHotTESTregScaled = dataTMP.map(lambda x: LabeledPoint(x[0], x[1]))

# PRINT ELAPSED TIME
timeend = datetime.datetime.now()
timedelta = round((timeend-timestart).total_seconds(), 2)
print "Time taken to execute above cell: " + str(timedelta) + " seconds";

```

OUTPUT:

Time taken to execute above cell: 13.17 seconds

Cache objects in memory

The time taken for training and testing of ML algorithms can be reduced by caching the input data frame objects used for classification, regression, and scaled features.

```

# RECORD START TIME
timestart = datetime.datetime.now()

# FOR BINARY CLASSIFICATION TRAINING AND TESTING
indexedTRAINbinary.cache()
indexedTESTbinary.cache()
oneHotTRAINbinary.cache()
oneHotTESTbinary.cache()

# FOR REGRESSION TRAINING AND TESTING
indexedTRAINreg.cache()
indexedTESTreg.cache()
oneHotTRAINreg.cache()
oneHotTESTreg.cache()

# SCALED FEATURES
oneHotTRAINregScaled.cache()
oneHotTESTregScaled.cache()

# PRINT ELAPSED TIME
timeend = datetime.datetime.now()
timedelta = round((timeend-timestart).total_seconds(), 2)
print "Time taken to execute above cell: " + str(timedelta) + " seconds";

```

OUTPUT:

Time taken to execute above cell: 0.15 seconds

Predict whether or not a tip is paid with binary classification models

This section shows how use three models for the binary classification task of predicting whether or not a tip is paid for a taxi trip. The models presented are:

- Regularized logistic regression
- Random forest model
- Gradient Boosting Trees

Each model building code section is split into steps:

1. **Model training** data with one parameter set
2. **Model evaluation** on a test data set with metrics
3. **Saving model** in blob for future consumption

Classification using logistic regression

The code in this section shows how to train, evaluate, and save a logistic regression model with [LBFGS](#) that predicts whether or not a tip is paid for a trip in the NYC taxi trip and fare dataset.

Train the logistic regression model using CV and hyperparameter sweeping

```
# LOGISTIC REGRESSION CLASSIFICATION WITH CV AND HYPERPARAMETER SWEEPING

# GET ACCURACY FOR HYPERPARAMETERS BASED ON CROSS-VALIDATION IN TRAINING DATA-SET

# RECORD START TIME
timestart = datetime.datetime.now()

# LOAD LIBRARIES
from pyspark.mllib.classification import LogisticRegressionWithLBFGS
from sklearn.metrics import roc_curve, auc
from pyspark.mllib.evaluation import BinaryClassificationMetrics
from pyspark.mllib.evaluation import MulticlassMetrics

# CREATE MODEL WITH ONE SET OF PARAMETERS
logitModel = LogisticRegressionWithLBFGS.train(oneHotTRAINbinary, iterations=20, initialWeights=None,
                                              regParam=0.01, regType='l2', intercept=True, corrections=10,
                                              tolerance=0.0001, validateData=True, numClasses=2)

# PRINT COEFFICIENTS AND INTERCEPT OF THE MODEL
# NOTE: There are 20 coefficient terms for the 10 features,
#       and the different categories for features: vendorVec (2), rateVec, paymentVec (6), TrafficTimeBinsVec (4)
print("Coefficients: " + str(logitModel.weights))
print("Intercept: " + str(logitModel.intercept))

# PRINT ELAPSED TIME
timeend = datetime.datetime.now()
timedelta = round((timeend-timestart).total_seconds(), 2)
print "Time taken to execute above cell: " + str(timedelta) + " seconds";
```

OUTPUT:

```
Coefficients: [0.0082065285375, -0.0223675576104, -0.0183812028036, -3.48124578069e-05, -
0.00247646947233, -0.00165897881503, 0.0675394837328, -0.111823113101, -0.324609912762, -
0.204549780032, -1.36499216354, 0.591088507921, -0.664263411392, -1.00439726852, 3.46567827545, -
3.51025855172, -0.0471341112232, -0.043521833294, 0.000243375810385, 0.054518719222]
```

```
Intercept: -0.0111216486893
```

```
Time taken to execute above cell: 14.43 seconds
```

Evaluate the binary classification model with standard metrics

```
#EVALUATE LOGISTIC REGRESSION MODEL WITH LBFGS

# RECORD START TIME
timestart = datetime.datetime.now()

# PREDICT ON TEST DATA WITH MODEL
predictionAndLabels = oneHotTESTbinary.map(lambda lp: (float(logitModel.predict(lp.features)), lp.label))

# INSTANTIATE METRICS OBJECT
metrics = BinaryClassificationMetrics(predictionAndLabels)

# AREA UNDER PRECISION-RECALL CURVE
print("Area under PR =%s" % metrics.areaUnderPR)

# AREA UNDER ROC CURVE
print("Area under ROC =%s" % metrics.areaUnderROC)
metrics = MulticlassMetrics(predictionAndLabels)

# OVERALL STATISTICS
precision = metrics.precision()
recall = metrics.recall()
f1Score = metrics.fMeasure()
print("Summary Stats")
print("Precision =%s" % precision)
print("Recall =%s" % recall)
print("F1 Score =%s" % f1Score)

## SAVE MODEL WITH DATE-STAMP
datestamp = unicode(datetime.datetime.now()).replace(',') replace(':' , '_');
logisticregressionfilename = "LogisticRegressionWithLBFGS_" + datestamp;
dirfilename = modelDir + logisticregressionfilename;
logitModel.save(sc, dirfilename);

# OUTPUT PROBABILITIES AND REGISTER TEMP TABLE
logitModel.clearThreshold(); # This clears threshold for classification (0.5) and outputs probabilities
predictionAndLabelsDF = predictionAndLabels.toDF()
predictionAndLabelsDF.registerTempTable("tmp_results");

# PRINT ELAPSED TIME
timeend = datetime.datetime.now()
timedelta = round((timeend-timestart).total_seconds(), 2)
print "Time taken to execute above cell: " + str(timedelta) + " seconds";
```

OUTPUT:

Area under PR = 0.985297691373

Area under ROC = 0.983714670256

Summary Stats

Precision = 0.984304060189

Recall = 0.984304060189

F1 Score = 0.984304060189

Time taken to execute above cell: 57.61 seconds

Plot the ROC curve.

The `predictionAndLabelsDF` is registered as a table, `tmp_results`, in the previous cell. `tmp_results` can be used to do queries and output results into the `sqlResults` data-frame for plotting. Here is the code.

```
# QUERY RESULTS
%%sql -q -o sqlResults
SELECT * from tmp_results
```

Here is the code to make predictions and plot the ROC-curve.

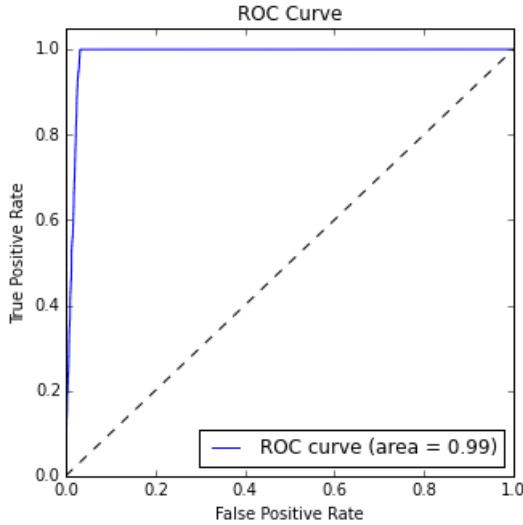
```
# MAKE PREDICTIONS AND PLOT ROC-CURVE

# RUN THE CODE LOCALLY ON THE JUPYTER SERVER AND IMPORT LIBRARIES
%%local
%matplotlib inline
from sklearn.metrics import roc_curve,auc

# MAKE PREDICTIONS
predictions_pddf = test_predictions.rename(columns={'_1':'probability','_2':'label'})
prob = predictions_pddf["probability"]
fpr, tpr, thresholds = roc_curve(predictions_pddf["label"], prob, pos_label=1);
roc_auc = auc(fpr, tpr)

# PLOT ROC CURVE
plt.figure(figsize=(5,5))
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()
```

OUTPUT:



Random forest classification

The code in this section shows how to train, evaluate, and save a random forest model that predicts whether or not a tip is paid for a trip in the NYC taxi trip and fare dataset.

```

#PREDICT WHETHER A TIP IS PAID OR NOT USING RANDOM FOREST

# RECORD START TIME
timestart = datetime.datetime.now()

# LOAD PYSPARK LIBRARIES
from pyspark.mllib.tree import RandomForest, RandomForestModel
from pyspark.mllib.util import MLUtils
from pyspark.mllib.evaluation import BinaryClassificationMetrics
from pyspark.mllib.evaluation import MulticlassMetrics

# SPECIFY NUMBER OF CATEGORIES FOR CATEGORICAL FEATURES. FEATURE #0 HAS 2 CATEGORIES, FEATURE #2 HAS 2 CATEGORIES,
# AND SO ON
categoricalFeaturesInfo={0:2, 1:2, 2:6, 3:4}

# TRAIN RANDOMFOREST MODEL
rfModel = RandomForest.trainClassifier(indexedTRAINbinary, numClasses=2,
                                       categoricalFeaturesInfo=categoricalFeaturesInfo,
                                       numTrees=25, featureSubsetStrategy="auto",
                                       impurity='gini', maxDepth=5, maxBins=32)

## UN-COMMENT IF YOU WANT TO PRINT TREES
#print('Learned classification forest model')
#print(rfModel.toDebugString())

# PREDICT ON TEST DATA AND EVALUATE
predictions = rfModel.predict(indexedTESTbinary.map(lambda x: x.features))
predictionAndLabels = indexedTESTbinary.map(lambda lp: lp.label.zip(predictions))

# AREA UNDER ROC CURVE
metrics = BinaryClassificationMetrics(predictionAndLabels)
print("Area under ROC = %s" % metrics.areaUnderROC)

# PERSIST MODEL IN BLOB
datestamp = unicode(datetime.datetime.now()).replace('!').replace(':', '_');
rfclassificationfilename = "RandomForestClassification_" + datestamp;
dirfilename = modelDir + rfclassificationfilename;

rfModel.save(sc, dirfilename);

# PRINT ELAPSED TIME
timeend = datetime.datetime.now()
timedelta = round((timeend-timestart).total_seconds(), 2)
print "Time taken to execute above cell: " + str(timedelta) + " seconds";

```

OUTPUT:

Area under ROC = 0.985297691373

Time taken to execute above cell: 31.09 seconds

Gradient boosting trees classification

The code in this section shows how to train, evaluate, and save a gradient boosting trees model that predicts whether or not a tip is paid for a trip in the NYC taxi trip and fare dataset.

```

#PREDICT WHETHER A TIP IS PAID OR NOT USING GRADIENT BOOSTING TREES

# RECORD START TIME
timestart = datetime.datetime.now()

# LOAD PYSPARK LIBRARIES
from pyspark.mllib.tree import GradientBoostedTrees, GradientBoostedTreesModel

# SPECIFY NUMBER OF CATEGORIES FOR CATEGORICAL FEATURES. FEATURE #0 HAS 2 CATEGORIES, FEATURE #2 HAS 2 CATEGORIES,
AND SO ON
categoricalFeaturesInfo={0:2, 1:2, 2:6, 3:4}

gbtModel = GradientBoostedTrees.trainClassifier(indexedTRAINbinary, categoricalFeaturesInfo=categoricalFeaturesInfo, numIterations=5)
## UNCOMMENT IF YOU WANT TO PRINT TREE DETAILS
#print('Learned classification GBT model:')
#print(bgtModel.toDebugString())

# PREDICT ON TEST DATA AND EVALUATE
predictions = gbtModel.predict(indexedTESTbinary.map(lambda x: x.features))
predictionAndLabels = indexedTESTbinary.map(lambda lp: lp.label).zip(predictions)

# AREA UNDER ROC CURVE
metrics = BinaryClassificationMetrics(predictionAndLabels)
print("Area under ROC = %s" % metrics.areaUnderROC)

# PERSIST MODEL IN A BLOB
datestamp = unicode(datetime.datetime.now()).replace('!', '_').replace(':', '_');
btcclassificationfilename = "GradientBoostingTreeClassification_" + datestamp;
dirfilename = modelDir + btcclassificationfilename;

gbtModel.save(sc, dirfilename)

# PRINT ELAPSED TIME
timeend = datetime.datetime.now()
timedelta = round((timeend-timestart).total_seconds(), 2)
print "Time taken to execute above cell: " + str(timedelta) + " seconds";

```

OUTPUT:

Area under ROC = 0.985297691373

Time taken to execute above cell: 19.76 seconds

Predict tip amounts for taxi trips with regression models

This section shows how use three models for the regression task of predicting the amount of the tip paid for a taxi trip based on other tip features. The models presented are:

- Regularized linear regression
- Random forest
- Gradient Boosting Trees

These models were described in the introduction. Each model building code section is split into steps:

1. **Model training** data with one parameter set
2. **Model evaluation** on a test data set with metrics
3. **Saving model** in blob for future consumption

Linear regression with SGD

The code in this section shows how to use scaled features to train a linear regression that uses stochastic gradient descent (SGD) for optimization, and how to score, evaluate, and save the model in Azure Blob Storage (WASB).

TIP

In our experience, there can be issues with the convergence of LinearRegressionWithSGD models, and parameters need to be changed/optimized carefully for obtaining a valid model. Scaling of variables significantly helps with convergence.

```
#PREDICT TIP AMOUNTS USING LINEAR REGRESSION WITH SGD

# RECORD START TIME
timestart = datetime.datetime.now()

# LOAD LIBRARIES
from pyspark.mllib.regression import LabeledPoint, LinearRegressionWithSGD, LinearRegressionModel
from pyspark.mllib.evaluation import RegressionMetrics
from scipy import stats

# USE SCALED FEATURES TO TRAIN MODEL
linearModel = LinearRegressionWithSGD.train(oneHotTRAINregScaled, iterations=100, step = 0.1, regType='l2', regParam=0.1, intercept = True)

# PRINT COEFFICIENTS AND INTERCEPT OF THE MODEL
# NOTE: There are 20 coefficient terms for the 10 features,
# and the different categories for features: vendorVec (2), rateVec, paymentVec (6), TrafficTimeBinsVec (4)
print("Coefficients: " + str(linearModel.weights))
print("Intercept: " + str(linearModel.intercept))

# SCORE ON SCALED TEST DATA-SET & EVALUATE
predictionAndLabels = oneHotTESTregScaled.map(lambda lp: (float(linearModel.predict(lp.features)), lp.label))
testMetrics = RegressionMetrics(predictionAndLabels)

# PRINT TEST METRICS
print("RMSE=%s" % testMetrics.rootMeanSquaredError)
print("R-sqr=%s" % testMetrics.r2)

# SAVE MODEL WITH DATE-STAMP IN THE DEFAULT BLOB FOR THE CLUSTER
datestamp = unicode(datetime.datetime.now()).replace('!').replace(':','_');
linearregressionfilename = "LinearRegressionWithSGD_" + datestamp;
dirfilename = modelDir + linearregressionfilename;

linearModel.save(sc, dirfilename)

# PRINT ELAPSED TIME
timeend = datetime.datetime.now()
timedelta = round((timeend-timestart).total_seconds(), 2)
print "Time taken to execute above cell: " + str(timedelta) + " seconds";
```

OUTPUT:

Coefficients: [0.00457675809917, -0.0226314167349, -0.0191910355236, 0.246793409578, 0.312047890459, 0.359634405999, 0.00928692253981, -0.000987181489428, -0.0888306617845, 0.0569376211553, 0.115519551711, 0.149250164995, -0.00990211159703, -0.00637410344522, 0.545083566179, -0.536756072402, 0.0105762393099, -0.0130117577055, 0.0129304737772, -0.00171065945959]

Intercept: 0.853872718283

RMSE = 1.24190115863

R-sqr = 0.608017146081

Time taken to execute above cell: 58.42 seconds

Random Forest regression

The code in this section shows how to train, evaluate, and save a random forest regression that predicts tip amount for the NYC taxi trip data.

```

#PREDICT TIP AMOUNTS USING RANDOM FOREST

# RECORD START TIME
timestart= datetime.datetime.now()

# LOAD PYSPARK LIBRARIES
from pyspark.mllib.tree import RandomForest, RandomForestModel
from pyspark.mllib.util import MLUtils
from pyspark.mllib.evaluation import RegressionMetrics

## TRAIN MODEL
categoricalFeaturesInfo={0:2, 1:2, 2:6, 3:4}
rfModel = RandomForest.trainRegressor(indexedTRAINreg, categoricalFeaturesInfo=categoricalFeaturesInfo,
                                       numTrees=25, featureSubsetStrategy="auto",
                                       impurity='variance', maxDepth=10, maxBins=32)
## UN-COMMENT IF YOU WANT TO PRING TREES
#print('Learned classification forest model!')
#print(rfModel.toDebugString())

## PREDICT AND EVALUATE ON TEST DATA-SET
predictions = rfModel.predict(indexedTESTreg.map(lambda x: x.features))
predictionAndLabels = oneHotTESTreg.map(lambda lp: lp.label.zip(predictions))

# TEST METRICS
testMetrics = RegressionMetrics(predictionAndLabels)
print("RMSE=%s" % testMetrics.rootMeanSquaredError)
print("R-sqr=%s" % testMetrics.r2)

# SAVE MODEL IN BLOB
datestamp = unicode(datetime.datetime.now()).replace(',',).replace(':', '_');
rfregressionfilename = "RandomForestRegression_" + datestamp;
dirfilename = modelDir + rfregressionfilename;

rfModel.save(sc, dirfilename);

# PRINT ELAPSED TIME
timeend = datetime.datetime.now()
timedelta = round((timeend-timestart).total_seconds(), 2)
print "Time taken to execute above cell: " + str(timedelta) + " seconds";

```

OUTPUT:

RMSE = 0.891209218139

R-sqr = 0.759661334921

Time taken to execute above cell: 49.21 seconds

Gradient boosting trees regression

The code in this section shows how to train, evaluate, and save a gradient boosting trees model that predicts tip amount for the NYC taxi trip data.

****Train and evaluate ****

```

#PREDICT TIP AMOUNTS USING GRADIENT BOOSTING TREES

# RECORD START TIME
timestart= datetime.datetime.now()

# LOAD PYSPARK LIBRARIES
from pyspark.mllib.tree import GradientBoostedTrees, GradientBoostedTreesModel
from pyspark.mllib.util import MLUtils

## TRAIN MODEL
categoricalFeaturesInfo={0:2, 1:2, 2:6, 3:4}
gbtModel = GradientBoostedTrees.trainRegressor(indexedTRAINreg, categoricalFeaturesInfo=categoricalFeaturesInfo,
                                              numIterations=10, maxBins=32, maxDepth = 4, learningRate=0.1)

## EVALUATE A TEST DATA-SET
predictions = gbtModel.predict(indexedTESTreg.map(lambda x: x.features))
predictionAndLabels = indexedTESTreg.map(lambda lp: lp.label).zip(predictions)

# TEST METRICS
testMetrics = RegressionMetrics(predictionAndLabels)
print("RMSE = %s" % testMetrics.rootMeanSquaredError)
print("R-sqr = %s" % testMetrics.r2)

# SAVE MODEL IN BLOB
datestamp = unicode(datetime.datetime.now()).replace('!').replace(':', '_');
btregressionfilename = "GradientBoostingTreeRegression_" + datestamp;
dirfilename = modelDir + btregressionfilename;
gbtModel.save(sc, dirfilename)

# CONVER RESULTS TO DF AND REGISER TEMP TABLE
test_predictions = sqlContext.createDataFrame(predictionAndLabels)
test_predictions.registerTempTable("tmp_results");

# PRINT ELAPSED TIME
timeend = datetime.datetime.now()
timedelta = round((timeend-timestart).total_seconds(), 2)
print "Time taken to execute above cell: " + str(timedelta) + " seconds";

```

OUTPUT:

RMSE = 0.908473148639

R-sqr = 0.753835096681

Time taken to execute above cell: 34.52 seconds

Plot

tmp_results is registered as a Hive table in the previous cell. Results from the table are output into the *sqlResults* data-frame for plotting. Here is the code

```

# PLOT SCATTER-PLOT BETWEEN ACTUAL AND PREDICTED TIP VALUES

# SELECT RESULTS
%%sql -q -o sqlResults
SELECT * from tmp_results

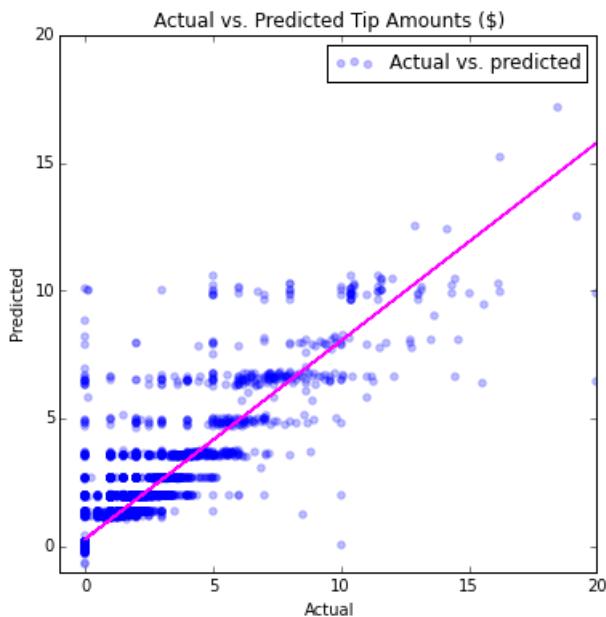
```

Here is the code to plot the data using the Jupyter server.

```
# RUN THE CODE LOCALLY ON THE JUPYTER SERVER AND IMPORT LIBRARIES
%%local
%matplotlib inline
import numpy as np

# PLOT
ax = test_predictions_pddf.plot(kind='scatter', figsize=(6,6), x='_1', y='_2', color='blue', alpha=0.25, label='Actual vs. predicted');
fit = np.polyfit(test_predictions_pddf['_1'], test_predictions_pddf['_2'], deg=1)
ax.set_title('Actual vs. Predicted Tip Amounts ($)')
ax.set_xlabel("Actual")
ax.set_ylabel("Predicted")
ax.plot(test_predictions_pddf['_1'], fit[0] * test_predictions_pddf['_1'] + fit[1], color='magenta')
plt.axis([-1, 20, -1, 20])
plt.show(ax)
```

OUTPUT:



Clean up objects from memory

Use `unpersist()` to delete objects cached in memory.

```
# REMOVE ORIGINAL DFs
taxi_df_train_cleaned.unpersist()
taxi_df_train_with_newFeatures.unpersist()

# FOR BINARY CLASSIFICATION TRAINING AND TESTING
indexedTRAINbinary.unpersist()
indexedTESTbinary.unpersist()
oneHotTRAINbinary.unpersist()
oneHotTESTbinary.unpersist()

# FOR REGRESSION TRAINING AND TESTING
indexedTRAINreg.unpersist()
indexedTESTreg.unpersist()
oneHotTRAINreg.unpersist()
oneHotTESTreg.unpersist()

# SCALED FEATURES
oneHotTRAINregScaled.unpersist()
oneHotTESTregScaled.unpersist()
```

Record storage locations of the models for consumption and scoring

To consume and score an independent dataset described in the [Score and evaluate Spark-built machine learning models](#) topic, you need to copy and paste these file names containing the saved models created here into the Consumption Jupyter notebook. Here is the code to print out the paths to model files you need there.

```
# MODEL FILE LOCATIONS FOR CONSUMPTION
print "logisticRegFileLoc = modelDir + \\" + logisticregressionfilename + \\"";
print "linearRegFileLoc = modelDir + \\" + linearregressionfilename + \\"";
print "randomForestClassificationFileLoc = modelDir + \\" + rfclassificationfilename + \\"";
print "randomForestRegFileLoc = modelDir + \\" + rfregressionfilename + \\"";
print "BoostedTreeClassificationFileLoc = modelDir + \\" + btclassificationfilename + \\"";
print "BoostedTreeRegressionFileLoc = modelDir + \\" + btregressionfilename + \\";
```

OUTPUT

```
logisticRegFileLoc = modelDir + "LogisticRegressionWithLBFGS_2016-05-0317_03_23.516568"
linearRegFileLoc = modelDir + "LinearRegressionWithSGD_2016-05-0317_05_21.577773"
randomForestClassificationFileLoc = modelDir + "RandomForestClassification_2016-05-0317_04_11.950206"
randomForestRegFileLoc = modelDir + "RandomForestRegression_2016-05-0317_06_08.723736"
BoostedTreeClassificationFileLoc = modelDir + "GradientBoostingTreeClassification_2016-05-0317_04_36.346583"
BoostedTreeRegressionFileLoc = modelDir + "GradientBoostingTreeRegression_2016-05-0317_06_51.737282"
```

What's next?

Now that you have created regression and classification models with the Spark MLlib, you are ready to learn how to score and evaluate these models. The advanced data exploration and modeling notebook dives deeper into including cross-validation, hyper-parameter sweeping, and model evaluation.

Model consumption: To learn how to score and evaluate the classification and regression models created in this topic, see [Score and evaluate Spark-built machine learning models](#).

Cross-validation and hyperparameter sweeping: See [Advanced data exploration and modeling with Spark](#) on how models can be trained using cross-validation and hyper-parameter sweeping

Score Spark-built machine learning models

1/17/2017 • 17 min to read • [Edit on GitHub](#)

This topic describes how to load machine learning (ML) models that have been built using Spark MLlib and stored in Azure Blob Storage (WASB), and how to score them with datasets that have also been stored in WASB. It shows how to pre-process the input data, transform features using the indexing and encoding functions in the MLlib toolkit, and how to create a labeled point data object that can be used as input for scoring with the ML models. The models used for scoring include Linear Regression, Logistic Regression, Random Forest Models, and Gradient Boosting Tree Models.

Prerequisites

1. You need an Azure account and an HDInsight Spark 3.4 cluster to complete this walkthrough. See the [Overview of Data Science using Spark on Azure HDInsight](#) for instructions on how to satisfy these requirements. That topic also contains a description of the NYC 2013 Taxi data used here and instructions on how to execute code from a Jupyter notebook on the Spark cluster. The **pySpark-machine-learning-data-science-spark-model-consumption.ipynb** notebook that contains the code samples in this topic is available in [Github](#).
2. You must also create the machine learning models to be scored here by working through the [Data exploration and modeling with Spark](#) topic.

WARNING

HDInsight clusters billing is pro-rated per minute, whether you are using them or not. Please be sure to delete your cluster after you have finished using it. For information on deleting a cluster, see [How to delete an HDInsight cluster](#).

Setup: storage locations, libraries, and the preset Spark context

Spark is able to read and write to an Azure Storage Blob (WASB). So any of your existing data stored there can be processed using Spark and the results stored again in WASB.

To save models or files in WASB, the path needs to be specified properly. The default container attached to the Spark cluster can be referenced using a path beginning with: "wasb//". The following code sample specifies the location of the data to be read and the path for the model storage directory to which the model output is saved.

Set directory paths for storage locations in WASB

Models are saved in: "wasb://user/remoteuser/NYCTaxi/Models". If this path is not set properly, models are not loaded for scoring.

The scored results have been saved in: "wasb://user/remoteuser/NYCTaxi/ScoredResults". If the path to folder is incorrect, results are not saved in that folder.

NOTE

The file path locations can be copied and pasted into the placeholders in this code from the output of the last cell of the **machine-learning-data-science-spark-data-exploration-modeling.ipynb** notebook.

Here is the code to set directory paths:

```

# LOCATION OF DATA TO BE SCORED (TEST DATA)
taxi_test_file_loc = "wasb://mllibwalkthroughs@cdsparksamples.blob.core.windows.net/Data/NYCTaxi/JoinedTaxiTripFare.Point1Pct.Test.tsv";

# SET THE MODEL STORAGE DIRECTORY PATH
# NOTE THE LAST BACKSLASH IN THIS PATH IS NEEDED
modelDir = "wasb:///user/remoteuser/NYCTaxi/Models/"

# SET SCORDED RESULT DIRECTORY PATH
# NOTE THE LAST BACKSLASH IN THIS PATH IS NEEDED
scoredResultDir = "wasb:///user/remoteuser/NYCTaxi/ScoredResults/";

# FILE LOCATIONS FOR THE MODELS TO BE SCORED
logisticRegFileLoc = modelDir + "LogisticRegressionWithLBFGS_2016-04-1817_40_35.796789"
linearRegFileLoc = modelDir + "LinearRegressionWithSGD_2016-04-1817_44_00.993832"
randomForestClassificationFileLoc = modelDir + "RandomForestClassification_2016-04-1817_42_58.899412"
randomForestRegFileLoc = modelDir + "RandomForestRegression_2016-04-1817_44_27.204734"
BoostedTreeClassificationFileLoc = modelDir + "GradientBoostingTreeClassification_2016-04-1817_43_16.354770"
BoostedTreeRegressionFileLoc = modelDir + "GradientBoostingTreeRegression_2016-04-1817_44_46.206262"

# RECORD START TIME
import datetime
datetime.datetime.now()

```

OUTPUT:

`datetime.datetime(2016, 4, 25, 23, 56, 19, 229403)`

Import libraries

Set spark context and import necessary libraries with the following code

```

#IMPORT LIBRARIES
import pyspark
from pyspark import SparkConf
from pyspark import SparkContext
from pyspark.sql import SQLContext
import matplotlib
import matplotlib.pyplot as plt
from pyspark.sql import Row
from pyspark.sql.functions import UserDefinedFunction
from pyspark.sql.types import *
import atexit
from numpy import array
import numpy as np
import datetime

```

Preset Spark context and PySpark magics

The PySpark kernels that are provided with Jupyter notebooks have a preset context. So you do not need to set the Spark or Hive contexts explicitly before you start working with the application you are developing. These are available for you by default. These contexts are:

- sc - for Spark
- sqlContext - for Hive

The PySpark kernel provides some predefined “magics”, which are special commands that you can call with `%%`. There are two such commands that are used in these code samples.

- **%%local** Specified that the code in subsequent lines is executed locally. Code must be valid Python code.
- **%%sql -o**
- Executes a Hive query against the `sqlContext`. If the `-o` parameter is passed, the result of the query is persisted in the `%%local` Python context as a Pandas dataframe.

For more information on the kernels for Jupyter notebooks and the predefined "magics" that they provide, see [Kernels available for Jupyter notebooks with HDInsight Spark Linux clusters on HDInsight](#).

Ingest data and create a cleaned data frame

This section contains the code for a series of tasks required to ingest the data to be scored. Read in a joined 0.1% sample of the taxi trip and fare file (stored as a .tsv file), format the data, and then creates a clean data frame.

The taxi trip and fare files were joined based on the procedure provided in the: [The Team Data Science Process in action: using HDInsight Hadoop clusters](#) topic.

```

# INGEST DATA AND CREATE A CLEANED DATA FRAME

# RECORD START TIME
timestart = datetime.datetime.now()

# IMPORT FILE FROM PUBLIC BLOB
taxi_test_file = sc.textFile(taxi_test_file_loc)

# GET SCHEMA OF THE FILE FROM HEADER
taxi_header = taxi_test_file.filter(lambda l: "medallion" in l)

# PARSE FIELDS AND CONVERT DATA TYPE FOR SOME FIELDS
taxi_temp = taxi_test_file.subtract(taxi_header).map(lambda k: k.split("\t"))
.map(lambda p: (p[0],p[1],p[2],p[3],p[4],p[5],p[6],int(p[7]),int(p[8]),int(p[9]),int(p[10]),
float(p[11]),float(p[12]),p[13],p[14],p[15],p[16],p[17],p[18],float(p[19]),
float(p[20]),float(p[21]),float(p[22]),float(p[23]),float(p[24]),int(p[25]),int(p[26])))

# GET SCHEMA OF THE FILE FROM HEADER
schema_string = taxi_test_file.first()
fields = [StructField(field_name, StringType(), True) for field_name in schema_string.split("\t")]
fields[7].dataType = IntegerType() # Pickup hour
fields[8].dataType = IntegerType() # Pickup week
fields[9].dataType = IntegerType() # Weekday
fields[10].dataType = IntegerType() # Passenger count
fields[11].dataType = FloatType() # Trip time in secs
fields[12].dataType = FloatType() # Trip distance
fields[19].dataType = FloatType() # Fare amount
fields[20].dataType = FloatType() # Surcharge
fields[21].dataType = FloatType() # Mta_tax
fields[22].dataType = FloatType() # Tip amount
fields[23].dataType = FloatType() # Tolls amount
fields[24].dataType = FloatType() # Total amount
fields[25].dataType = IntegerType() # Tipped or not
fields[26].dataType = IntegerType() # Tip class
taxi_schema = StructType(fields)

# CREATE DATA FRAME
taxi_df_test = sqlContext.createDataFrame(taxi_temp, taxi_schema)

# CREATE A CLEANED DATA-FRAME BY DROPPING SOME UN-NECESSARY COLUMNS & FILTERING FOR UNDESIRED VALUES OR
OUTLIERS
taxi_df_test_cleaned = taxi_df_test.drop('medallion').drop('hack_license').drop('store_and_fwd_flag').drop('pickup_datetime')\
.drop('dropoff_datetime').drop('pickup_longitude').drop('pickup_latitude').drop('dropoff_latitude')\
.drop('dropoff_longitude').drop('tip_class').drop('total_amount').drop('tolls_amount').drop('mta_tax')\
.drop('direct_distance').drop('surcharge')\
.filter("passenger_count > 0 and passenger_count < 8 AND payment_type in ('CSH', 'CRD') AND tip_amount >= 0 AND tip_amount < 30 AND
fare_amount >= 1 AND fare_amount < 150 AND trip_distance > 0 AND trip_distance < 100 AND trip_time_in_secs > 30 AND trip_time_in_secs <
7200" )

# CACHE DATA-FRAME IN MEMORY & MATERIALIZE DF IN MEMORY
taxi_df_test_cleaned.cache()
taxi_df_test_cleaned.count()

# REGISTER DATA-FRAME AS A TEMP-TABLE IN SQL-CONTEXT
taxi_df_test_cleaned.registerTempTable("taxi_test")

# PRINT HOW MUCH TIME IT TOOK TO RUN THE CELL
timeend = datetime.datetime.now()
timedelta = round(((timeend-timestart).total_seconds(), 2)
print "Time taken to execute above cell: " + str(timedelta) + " seconds";

```

OUTPUT:

Time taken to execute above cell: 46.37 seconds

Prepare data for scoring in Spark

This section shows how to index, encode and scale categorical features to prepare them for use in MLlib supervised learning algorithms for classification and regression.

Feature transformation: index and encode categorical features for input into models for scoring

This section shows how to index categorical data using a `StringIndexer` and encode features with `OneHotEncoder` input into the models.

The `StringIndexer` encodes a string column of labels to a column of label indices. The indices are ordered by label frequencies.

The `OneHotEncoder` maps a column of label indices to a column of binary vectors, with at most a single one-value. This encoding allows algorithms that expect continuous valued features, such as logistic regression, to be applied to categorical features.

```

#INDEX AND ONE-HOT ENCODE CATEGORICAL FEATURES

# RECORD START TIME
timestart = datetime.datetime.now()

# LOAD PYSPARK LIBRARIES
from pyspark.ml.feature import OneHotEncoder, StringIndexer, VectorAssembler, VectorIndexer

# CREATE FOUR BUCKETS FOR TRAFFIC TIMES
sqlStatement = """
    SELECT *,
    CASE
        WHEN (pickup_hour <= 6 OR pickup_hour >= 20) THEN "Night"
        WHEN (pickup_hour >= 7 AND pickup_hour <= 10) THEN "AMRush"
        WHEN (pickup_hour >= 11 AND pickup_hour <= 15) THEN "Afternoon"
        WHEN (pickup_hour >= 16 AND pickup_hour <= 19) THEN "PMRush"
    END as TrafficTimeBins
    FROM taxi_test
"""
taxi_df_test_with_newFeatures = sqlContext.sql(sqlStatement)

# CACHE DATA-FRAME IN MEMORY & MATERIALIZE DF IN MEMORY
taxi_df_test_with_newFeatures.cache()
taxi_df_test_with_newFeatures.count()

# INDEX AND ONE-HOT ENCODING
stringIndexer = StringIndexer(inputCol="vendor_id", outputCol="vendorIndex")
model = stringIndexer.fit(taxi_df_test_with_newFeatures) # Input data-frame is the cleaned one from above
indexed = model.transform(taxi_df_test_with_newFeatures)
encoder = OneHotEncoder(dropLast=False, inputCol="vendorIndex", outputCol="vendorVec")
encoded1 = encoder.transform(indexed)

# INDEX AND ENCODE RATE_CODE
stringIndexer = StringIndexer(inputCol="rate_code", outputCol="rateIndex")
model = stringIndexer.fit(encoded1)
indexed = model.transform(encoded1)
encoder = OneHotEncoder(dropLast=False, inputCol="rateIndex", outputCol="rateVec")
encoded2 = encoder.transform(indexed)

# INDEX AND ENCODE PAYMENT_TYPE
stringIndexer = StringIndexer(inputCol="payment_type", outputCol="paymentIndex")
model = stringIndexer.fit(encoded2)
indexed = model.transform(encoded2)
encoder = OneHotEncoder(dropLast=False, inputCol="paymentIndex", outputCol="paymentVec")
encoded3 = encoder.transform(indexed)

# INDEX AND ENCODE TRAFFIC TIME BINS
stringIndexer = StringIndexer(inputCol="TrafficTimeBins", outputCol="TrafficTimeBinsIndex")
model = stringIndexer.fit(encoded3)
indexed = model.transform(encoded3)
encoder = OneHotEncoder(dropLast=False, inputCol="TrafficTimeBinsIndex", outputCol="TrafficTimeBinsVec")
encodedFinal = encoder.transform(indexed)

# PRINT HOW MUCH TIME IT TOOK TO RUN THE CELL
timeend = datetime.datetime.now()
timedelta = round((timeend-timestart).total_seconds(), 2)
print "Time taken to execute above cell: " + str(timedelta) + " seconds";

```

OUTPUT:

Time taken to execute above cell: 5.37 seconds

Create RDD objects with feature arrays for input into models

This section contains code that shows how to index categorical text data as an RDD object and one-hot encode it so it can be used to train and test MLlib logistic regression and tree-based models. The indexed data is stored in [Resilient Distributed Dataset \(RDD\)](#) objects. These are the basic abstraction in Spark. An RDD object represents an

immutable, partitioned collection of elements that can be operated on in parallel with Spark.

It also contains code that shows how to scale data with the `StandardScaler` provided by MLlib for use in linear regression with Stochastic Gradient Descent (SGD), a popular algorithm for training a wide range of machine learning models. The `StandardScaler` is used to scale the features to unit variance. Feature scaling, also known as data normalization, insures that features with widely disbursed values are not given excessive weigh in the objective function.

```

# CREATE RDD OBJECTS WITH FEATURE ARRAYS FOR INPUT INTO MODELS

# RECORD START TIME
timestart = datetime.datetime.now()

# IMPORT LIBRARIES
from pyspark.mllib.linalg import Vectors
from pyspark.mllib.feature import StandardScaler, StandardScalerModel
from pyspark.mllib.util import MLUtils
from numpy import array

# INDEXING CATEGORICAL TEXT FEATURES FOR INPUT INTO TREE-BASED MODELS
def parseRowIndexingBinary(line):
    features = np.array([line.paymentIndex, line.vendorIndex, line.rateIndex, line.TrafficTimeBinsIndex,
                        line.pickup_hour, line.weekday, line.passenger_count, line.trip_time_in_secs,
                        line.trip_distance, line.fare_amount])
    return features

# ONE-HOT ENCODING OF CATEGORICAL TEXT FEATURES FOR INPUT INTO LOGISTIC REGRESSION MODELS
def parseRowOneHotBinary(line):
    features = np.concatenate((np.array([line.pickup_hour, line.weekday, line.passenger_count,
                                         line.trip_time_in_secs, line.trip_distance, line.fare_amount]),
                               line.vendorVec.toArray(), line.rateVec.toArray(),
                               line.paymentVec.toArray(), line.TrafficTimeBinsVec.toArray()), axis=0)
    return features

# ONE-HOT ENCODING OF CATEGORICAL TEXT FEATURES FOR INPUT INTO TREE-BASED MODELS
def parseRowIndexingRegression(line):
    features = np.array([line.paymentIndex, line.vendorIndex, line.rateIndex, line.TrafficTimeBinsIndex,
                        line.pickup_hour, line.weekday, line.passenger_count, line.trip_time_in_secs,
                        line.trip_distance, line.fare_amount])
    return features

# INDEXING CATEGORICAL TEXT FEATURES FOR INPUT INTO LINEAR REGRESSION MODELS
def parseRowOneHotRegression(line):
    features = np.concatenate((np.array([line.pickup_hour, line.weekday, line.passenger_count,
                                         line.trip_time_in_secs, line.trip_distance, line.fare_amount]),
                               line.vendorVec.toArray(), line.rateVec.toArray(),
                               line.paymentVec.toArray(), line.TrafficTimeBinsVec.toArray()), axis=0)
    return features

# FOR BINARY CLASSIFICATION TRAINING AND TESTING
indexedTESTbinary = encodedFinal.map(parseRowIndexingBinary)
oneHotTESTbinary = encodedFinal.map(parseRowOneHotBinary)

# FOR REGRESSION CLASSIFICATION TRAINING AND TESTING
indexedTESTreg = encodedFinal.map(parseRowIndexingRegression)
oneHotTESTreg = encodedFinal.map(parseRowOneHotRegression)

# SCALING FEATURES FOR LINEAR REGRESSION WITH SGD MODEL
scaler = StandardScaler(withMean=False, withStd=True).fit(oneHotTESTreg)
oneHotTESTregScaled = scaler.transform(oneHotTESTreg)

# CACHE RDDS IN MEMORY
indexedTESTbinary.cache();
oneHotTESTbinary.cache();
indexedTESTreg.cache();
oneHotTESTreg.cache();
oneHotTESTregScaled.cache();

# PRINT HOW MUCH TIME IT TOOK TO RUN THE CELL
timeend = datetime.datetime.now()
timedelta = round((timeend-timestart).total_seconds(), 2)
print "Time taken to execute above cell: " + str(timedelta) + " seconds";

```

OUTPUT:

Time taken to execute above cell: 11.72 seconds

Score with the Logistic Regression Model and save output to blob

The code in this section shows how to load a Logistic Regression Model that has been saved in Azure blob storage and use it to predict whether or not a tip is paid on a taxi trip, score it with standard classification metrics, and then save and plot the results to blob storage. The scored results are stored in RDD objects.

```
# SCORE AND EVALUATE LOGISTIC REGRESSION MODEL

# RECORD START TIME
timestart = datetime.datetime.now()

# IMPORT LIBRARIES
from pyspark.mllib.classification import LogisticRegressionModel

## LOAD SAVED MODEL
savedModel = LogisticRegressionModel.load(sc, logisticRegFileLoc)
predictions = oneHotTESTbinary.map(lambda features: (float(savedModel.predict(features)))))

## SAVE SCORED RESULTS (RDD) TO BLOB
datestamp = unicode(datetime.datetime.now()).replace('!','').replace('!','_');
logisticregressionfilename = "LogisticRegressionWithLBFGS_" + datestamp + ".txt";
dirfilename = scoredResultDir + logisticregressionfilename;
predictions.saveAsTextFile(dirfilename)

# PRINT HOW MUCH TIME IT TOOK TO RUN THE CELL
timeend = datetime.datetime.now()
timedelta = round(((timeend-timestart).total_seconds(), 2)
print "Time taken to execute above cell: " + str(timedelta) + " seconds";
```

OUTPUT:

Time taken to execute above cell: 19.22 seconds

Score a Linear Regression Model

We used [LinearRegressionWithSGD](#) to train a linear regression model using Stochastic Gradient Descent (SGD) for optimization to predict the amount of tip paid.

The code in this section shows how to load a Linear Regression Model from Azure blob storage, score using scaled variables, and then save the results back to the blob.

```

#SCORE LINEAR REGRESSION MODEL

# RECORD START TIME
timestart = datetime.datetime.now()

#LOAD LIBRARIES
from pyspark.mllib.regression import LinearRegressionWithSGD, LinearRegressionModel

# LOAD MODEL AND SCORE USING ** SCALED VARIABLES **
savedModel = LinearRegressionModel.load(sc, linearRegFileLoc)
predictions = oneHotTESTregScaled.map(lambda features: (float(savedModel.predict(features)))))

# SAVE RESULTS
datestamp = unicode(datetime.datetime.now()).replace(' ','').replace(':', '_');
linearregressionfilename = "LinearRegressionWithSGD_" + datestamp;
dirfilename = scoredResultDir + linearregressionfilename;
predictions.saveAsTextFile(dirfilename)

# PRINT HOW MUCH TIME IT TOOK TO RUN THE CELL
timeend = datetime.datetime.now()
timedelta = round((timeend-timestart).total_seconds(), 2)
print "Time taken to execute above cell: " + str(timedelta) + " seconds";

```

OUTPUT:

Time taken to execute above cell: 16.63 seconds

Score classification and regression Random Forest Models

The code in this section shows how to load the saved classification and regression Random Forest Models saved in Azure blob storage, score their performance with standard classifier and regression measures, and then save the results back to blob storage.

[Random forests](#) are ensembles of decision trees. They combine many decision trees to reduce the risk of overfitting. Random forests can handle categorical features, extend to the multiclass classification setting, do not require feature scaling, and are able to capture non-linearities and feature interactions. Random forests are one of the most successful machine learning models for classification and regression.

[spark.mllib](#) supports random forests for binary and multiclass classification and for regression, using both continuous and categorical features.

```

# SCORE RANDOM FOREST MODELS FOR CLASSIFICATION AND REGRESSION

# RECORD START TIME
timestart = datetime.datetime.now()

#IMPORT MLLIB LIBRARIES
from pyspark.mllib.tree import RandomForest, RandomForestModel

# CLASSIFICATION: LOAD SAVED MODEL, SCORE AND SAVE RESULTS BACK TO BLOB
savedModel = RandomForestModel.load(sc, randomForestClassificationFileLoc)
predictions = savedModel.predict(indexedTESTbinary)

# SAVE RESULTS
datestamp = unicode(datetime.datetime.now()).replace(' ','').replace(':','_');
rfclassificationfilename = "RandomForestClassification_" + datestamp + ".txt";
dirfilename = scoredResultDir + rfclassificationfilename;
predictions.saveAsTextFile(dirfilename)

# REGRESSION: LOAD SAVED MODEL, SCORE AND SAVE RESULTS BACK TO BLOB
savedModel = RandomForestModel.load(sc, randomForestRegFileLoc)
predictions = savedModel.predict(indexedTESTreg)

# SAVE RESULTS
datestamp = unicode(datetime.datetime.now()).replace(' ','').replace(':','_');
rfregressionfilename = "RandomForestRegression_" + datestamp + ".txt";
dirfilename = scoredResultDir + rfregressionfilename;
predictions.saveAsTextFile(dirfilename)

# PRINT HOW MUCH TIME IT TOOK TO RUN THE CELL
timeend = datetime.datetime.now()
timedelta = round((timeend-timestart).total_seconds(), 2)
print "Time taken to execute above cell: " + str(timedelta) + " seconds";

```

OUTPUT:

Time taken to execute above cell: 31.07 seconds

Score classification and regression Gradient Boosting Tree Models

The code in this section shows how to load classification and regression Gradient Boosting Tree Models from Azure blob storage, score their performance with standard classifier and regression measures, and then save the results back to blob storage.

spark.mllib supports GBTs for binary classification and for regression, using both continuous and categorical features.

[Gradient Boosting Trees](#) (GBTs) are ensembles of decision trees. GBTs train decision trees iteratively to minimize a loss function. GBTs can handle categorical features, do not require feature scaling, and are able to capture non-linearities and feature interactions. They can also be used in a multiclass-classification setting.

```

# SCORE GRADIENT BOOSTING TREE MODELS FOR CLASSIFICATION AND REGRESSION

# RECORD START TIME
timestart = datetime.datetime.now()

#IMPORT MLLIB LIBRARIES
from pyspark.mllib.tree import GradientBoostedTrees, GradientBoostedTreesModel

# CLASSIFICATION: LOAD SAVED MODEL, SCORE AND SAVE RESULTS BACK TO BLOB

#LOAD AND SCORE THE MODEL
savedModel = GradientBoostedTreesModel.load(sc, BoostedTreeClassificationFileLoc)
predictions = savedModel.predict(indexedTESTbinary)

# SAVE RESULTS
datestamp = unicode(datetime.datetime.now()).replace(' ', '_').replace(':', '_');
btclassificationfilename = "GradientBoostingTreeClassification_" + datestamp + ".txt";
dirfilename = scoredResultDir + btclassificationfilename;
predictions.saveAsTextFile(dirfilename)

# REGRESSION: LOAD SAVED MODEL, SCORE AND SAVE RESULTS BACK TO BLOB

# LOAD AND SCORE MODEL
savedModel = GradientBoostedTreesModel.load(sc, BoostedTreeRegressionFileLoc)
predictions = savedModel.predict(indexedTESTreg)

# SAVE RESULTS
datestamp = unicode(datetime.datetime.now()).replace(' ', '_').replace(':', '_');
btregressionfilename = "GradientBoostingTreeRegression_" + datestamp + ".txt";
dirfilename = scoredResultDir + btregressionfilename;
predictions.saveAsTextFile(dirfilename)

# PRINT HOW MUCH TIME IT TOOK TO RUN THE CELL
timeend = datetime.datetime.now()
timedelta = round((timeend-timestart).total_seconds(), 2)
print "Time taken to execute above cell: " + str(timedelta) + " seconds";

```

OUTPUT:

Time taken to execute above cell: 14.6 seconds

Clean up objects from memory and print scored file locations

```

# UNPERSIST OBJECTS CACHED IN MEMORY
taxi_df_test_cleaned.unpersist()
indexedTESTbinary.unpersist();
oneHotTESTbinary.unpersist();
indexedTESTreg.unpersist();
oneHotTESTreg.unpersist();
oneHotTESTregScaled.unpersist();

# PRINT OUT PATH TO SCORED OUTPUT FILES
print "logisticRegFileLoc: " + logisticregressionfilename;
print "linearRegFileLoc: " + linearregressionfilename;
print "randomForestClassificationFileLoc: " + rfclassificationfilename;
print "randomForestRegFileLoc: " + rfregressionfilename;
print "BoostedTreeClassificationFileLoc: " + btclassificationfilename;
print "BoostedTreeRegressionFileLoc: " + btregressionfilename;

```

OUTPUT:

logisticRegFileLoc: LogisticRegressionWithLBFGS_2016-05-0317_22_38.953814.txt

linearRegFileLoc: LinearRegressionWithSGD_2016-05-0317_22_58.878949
randomForestClassificationFileLoc: RandomForestClassification_2016-05-0317_23_15.939247.txt
randomForestRegFileLoc: RandomForestRegression_2016-05-0317_23_31.459140.txt
BoostedTreeClassificationFileLoc: GradientBoostingTreeClassification_2016-05-0317_23_49.648334.txt
BoostedTreeRegressionFileLoc: GradientBoostingTreeRegression_2016-05-0317_23_56.860740.txt

Consume Spark Models through a web interface

Spark provides a mechanism to remotely submit batch jobs or interactive queries through a REST interface with a component called Livy. Livy is enabled by default on your HDInsight Spark cluster. For more information on Livy see: [Submit Spark jobs remotely using Livy](#).

You can use Livy to remotely submit a job that batch scores a file that is stored in an Azure blob and then writes the results to another blob. To do this, you upload the Python script from [Github](#) to the blob of the Spark cluster. You can use a tool like **Microsoft Azure Storage Explorer** or **AzCopy** to copy the script to the cluster blob. In our case we uploaded the script to ***wasb://example/python/ConsumeGBNYCReg.py***.

NOTE

The access keys that you need can be found on the portal for the storage account associated with the Spark cluster.

Once uploaded to this location, this script runs within the Spark cluster in a distributed context. It loads the model and run predictions on input files based on the model.

You can invoke this script remotely by making a simple HTTPS/REST request on Livy. Here is a curl command to construct the HTTP request to invoke the Python script remotely. Replace CLUSTERLOGIN, CLUSTERPASSWORD, CLUSTERNAME with the appropriate values for your Spark cluster.

```
# CURL COMMAND TO INVOKE PYTHON SCRIPT WITH HTTP REQUEST
curl -k --user "CLUSTERLOGIN:CLUSTERPASSWORD" -X POST --data "{\"file\": \"wasb://example/python/ConsumeGBNYCReg.py\"}" -H "Content-Type: application/json" https://CLUSTERNAME.azurehdinsight.net/livy/batches
```

You can use any language on the remote system to invoke the Spark job through Livy by making a simple HTTPS call with Basic Authentication.

NOTE

It would be convenient to use the Python Requests library when making this HTTP call, but it is not currently installed by default in Azure Functions. So older HTTP libraries are used instead.

Here is the Python code for the HTTP call:

```

#MAKE AN HTTPS CALL ON LIVY.

import os

# OLDER HTTP LIBRARIES USED HERE INSTEAD OF THE REQUEST LIBRARY AS THEY ARE A VAILBLE BY DEFAULT
import httplib, urllib, base64

# REPLACE VALUE WITH ONES FOR YOUR SPARK CLUSTER
host = '<spark cluster name>.azurehdinsight.net:443'
username='<username>'
password='<password>'

#AUTHORIZATION
conn = httplib.HTTPSConnection(host)
auth = base64.encodestring("%s:%s" % (username, password)).replace("\n, ")
headers = {'Content-Type': 'application/json', 'Authorization': 'Basic %s' % auth}

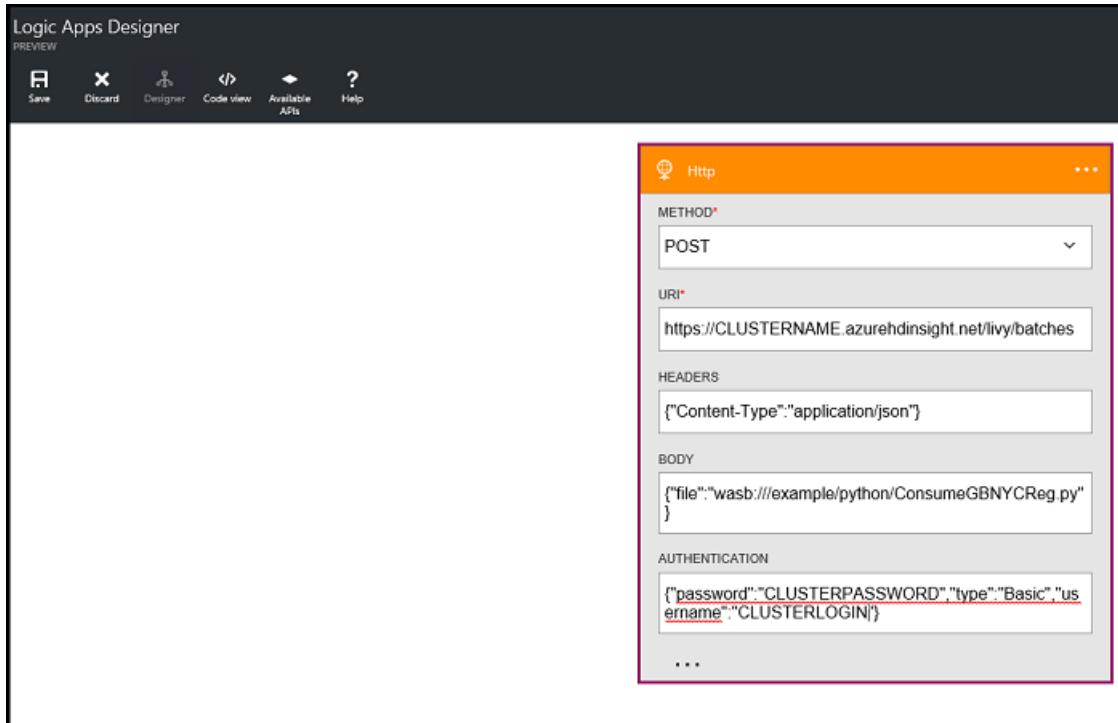
# SPECIFY THE PYTHON SCRIPT TO RUN ON THE SPARK CLUSTER
# IN THE FILE PARAMETER OF THE JSON POST REQUEST BODY
r=conn.request("POST", '/livy/batches', '{"file": "wasb://example/python/ConsumeGBNYCReg.py"}', headers )
response = conn.getresponse().read()
print(response)
conn.close()

```

You can also add this Python code to [Azure Functions](#) to trigger a Spark job submission that scores a blob based on various events like a timer, creation, or update of a blob.

If you prefer a code free client experience, use the [Azure Logic Apps](#) to invoke the Spark batch scoring by defining an HTTP action on the **Logic Apps Designer** and setting its parameters.

- From Azure portal, create a new Logic App by selecting **+New -> Web + Mobile -> Logic App**.
- To bring up the **Logic Apps Designer**, enter the name of the Logic App and App Service Plan .
- Select an HTTP action and enter the parameters shown in the following figure:



What's next?

Cross-validation and hyperparameter sweeping: See [Advanced data exploration and modeling with Spark](#) on how models can be trained using cross-validation and hyper-parameter sweeping.

Advanced data exploration and modeling with Spark

1/17/2017 • 37 min to read • [Edit on GitHub](#)

This walkthrough uses HDInsight Spark to do data exploration and train binary classification and regression models using cross-validation and hyperparameter optimization on a sample of the NYC taxi trip and fare 2013 dataset. It walks you through the steps of the [Data Science Process](#), end-to-end, using an HDInsight Spark cluster for processing and Azure blobs to store the data and the models. The process explores and visualizes data brought in from an Azure Storage Blob and then prepares the data to build predictive models. Python has been used to code the solution and to show the relevant plots. These models are built using the Spark MLlib toolkit to do binary classification and regression modeling tasks.

- The **binary classification** task is to predict whether or not a tip is paid for the trip.
- The **regression** task is to predict the amount of the tip based on other tip features.

The modeling steps also contain code showing how to train, evaluate, and save each type of model. The topic covers some of the same ground as the [Data exploration and modeling with Spark](#) topic. But it is more "advanced" in that it also uses cross-validation in conjunction with hyperparameter sweeping to train optimally accurate classification and regression models.

Cross-validation (CV) is a technique that assesses how well a model trained on a known set of data generalizes to predicting the features of datasets on which it has not been trained. The general idea behind this technique is that a model is trained on a dataset of known data and then the accuracy of its predictions is tested against an independent dataset. A common implementation used here is to divide a dataset into K folds and then train the model in a round-robin fashion on all but one of the folds.

Hyperparameter optimization is the problem of choosing a set of hyperparameters for a learning algorithm, usually with the goal of optimizing a measure of the algorithm's performance on an independent data set.

Hyperparameters are values that must be specified outside of the model training procedure. Assumptions about these values can impact the flexibility and accuracy of the models. Decision trees have hyperparameters, for example, such as the desired depth and number of leaves in the tree. Support Vector Machines (SVMs) require setting a misclassification penalty term.

A common way to perform hyperparameter optimization used here is a grid search, or a **parameter sweep**. This consists of performing an exhaustive search through the values a specified subset of the hyperparameter space for a learning algorithm. Cross validation can supply a performance metric to sort out the optimal results produced by the grid search algorithm. CV used with hyperparameter sweeping helps limit problems like overfitting a model to training data so that the model retains the capacity to apply to the general set of data from which the training data was extracted.

The models we use include logistic and linear regression, random forests, and gradient boosted trees:

- [Linear regression with SGD](#) is a linear regression model that uses a Stochastic Gradient Descent (SGD) method and for optimization and feature scaling to predict the tip amounts paid.
- [Logistic regression with LBFGS](#) or "logit" regression, is a regression model that can be used when the dependent variable is categorical to do data classification. LBFGS is a quasi-Newton optimization algorithm that approximates the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm using a limited amount of computer memory and that is widely used in machine learning.
- [Random forests](#) are ensembles of decision trees. They combine many decision trees to reduce the risk of overfitting. Random forests are used for regression and classification and can handle categorical features and can be extended to the multiclass classification setting. They do not require feature scaling and are able to capture non-linearities and feature interactions. Random forests are one of the most successful machine

learning models for classification and regression.

- **Gradient boosted trees** (GBTs) are ensembles of decision trees. GBTs train decision trees iteratively to minimize a loss function. GBTs are used for regression and classification and can handle categorical features, do not require feature scaling, and are able to capture non-linearities and feature interactions. They can also be used in a multiclass-classification setting.

Modeling examples using CV and Hyperparameter sweep are shown for the binary classification problem. Simpler examples (without parameter sweeps) are presented in the main topic for regression tasks. But in the appendix, validation using elastic net for linear regression and CV with parameter sweep using for random forest regression are also presented. The **elastic net** is a regularized regression method for fitting linear regression models that linearly combines the L1 and L2 metrics as penalties of the **lasso** and **ridge** methods.

NOTE

Although the Spark MLlib toolkit is designed to work on large datasets, a relatively small sample (~30 Mb using 170K rows, about 0.1% of the original NYC dataset) is used here for convenience. The exercise given here runs efficiently (in about 10 minutes) on an HDInsight cluster with 2 worker nodes. The same code, with minor modifications, can be used to process larger data-sets, with appropriate modifications for caching data in memory and changing the cluster size.

Prerequisites

You need an Azure account and an HDInsight Spark You need an HDInsight 3.4 Spark 1.6 cluster to complete this walkthrough. See the [Overview of Data Science using Spark on Azure HDInsight](#) for instructions on how to satisfy these requirements. That topic also contains a description of the NYC 2013 Taxi data used here and instructions on how to execute code from a Jupyter notebook on the Spark cluster. The **pySpark-machine-learning-data-science-spark-advanced-data-exploration-modeling.ipynb** notebook that contains the code samples in this topic is available in [Github](#).

WARNING

HDInsight clusters billing is pro-rated per minute, whether you are using them or not. Please be sure to delete your cluster after you have finished using it. For information on deleting a cluster, see [How to delete an HDInsight cluster](#).

Setup: storage locations, libraries, and the preset Spark context

Spark is able to read and write to Azure Storage Blob (also known as WASB). So any of your existing data stored there can be processed using Spark and the results stored again in WASB.

To save models or files in WASB, the path needs to be specified properly. The default container attached to the Spark cluster can be referenced using a path beginning with: "wasb://". Other locations are referenced by "wasb://" .

Set directory paths for storage locations in WASB

The following code sample specifies the location of the data to be read and the path for the model storage directory to which the model output is saved:

```

# SET PATHS TO FILE LOCATIONS: DATA AND MODEL STORAGE

# LOCATION OF TRAINING DATA
taxi_train_file_loc = "wasb://mlibwalkthroughs@cdsparksamples.blob.core.windows.net/Data/NYCTaxi/JoinedTaxiTripFare.Point1Pct.Train.tsv";

# SET THE MODEL STORAGE DIRECTORY PATH
# NOTE THAT THE FINAL BACKSLASH IN THE PATH IS NEEDED.
modelDir = "wasb://user/remoteuser/NYCTaxi/Models/";

# PRINT START TIME
import datetime
datetime.datetime.now()

```

OUTPUT

`datetime.datetime(2016, 4, 18, 17, 36, 27, 832799)`

Import libraries

Import necessary libraries with the following code:

```

# LOAD PYSPARK LIBRARIES
import pyspark
from pyspark import SparkConf
from pyspark import SparkContext
from pyspark.sql import SQLContext
import matplotlib
import matplotlib.pyplot as plt
from pyspark.sql import Row
from pyspark.sql.functions import UserDefinedFunction
from pyspark.sql.types import *
import atexit
from numpy import array
import numpy as np
import datetime

```

Preset Spark context and PySpark magics

The PySpark kernels that are provided with Jupyter notebooks have a preset context. So you do not need to set the Spark or Hive contexts explicitly before you start working with the application you are developing. These contexts are available for you by default. These contexts are:

- `sc` - for Spark
- `sqlContext` - for Hive

The PySpark kernel provides some predefined "magics", which are special commands that you can call with `%%`. There are two such commands that are used in these code samples.

- **`%%local`** Specifies that the code in subsequent lines is to be executed locally. Code must be valid Python code.
- **`%%sql -o`** Executes a Hive query against the `sqlContext`. If the `-o` parameter is passed, the result of the query is persisted in the `%%local` Python context as a Pandas DataFrame.

For more information on the kernels for Jupyter notebooks and the predefined "magics" that they provide, see [Kernels available for Jupyter notebooks with HDInsight Spark Linux clusters on HDInsight](#).

Data ingestion from public blob:

The first step in the data science process is to ingest the data to be analyzed from sources where it resides into your data exploration and modeling environment. This environment is Spark in this walkthrough. This section contains the code to complete a series of tasks:

- ingest the data sample to be modeled
- read in the input dataset (stored as a .tsv file)
- format and clean the data
- create and cache objects (RDDs or data-frames) in memory
- register it as a temp-table in SQL-context.

Here is the code for data ingestion.

```

# RECORD START TIME
timestart = datetime.datetime.now()

# IMPORT FILE FROM PUBLIC BLOB
taxi_train_file = sc.textFile(taxi_train_file_loc)

# GET SCHEMA OF THE FILE FROM HEADER
schema_string = taxi_train_file.first()
fields = [StructField(field_name, StringType(), True) for field_name in schema_string.split("\t")]
fields[7].dataType = IntegerType() # Pickup hour
fields[8].dataType = IntegerType() # Pickup week
fields[9].dataType = IntegerType() # Weekday
fields[10].dataType = IntegerType() # Passenger count
fields[11].dataType = FloatType() # Trip time in secs
fields[12].dataType = FloatType() # Trip distance
fields[19].dataType = FloatType() # Fare amount
fields[20].dataType = FloatType() # Surcharge
fields[21].dataType = FloatType() # Mta_tax
fields[22].dataType = FloatType() # Tip amount
fields[23].dataType = FloatType() # Tolls amount
fields[24].dataType = FloatType() # Total amount
fields[25].dataType = IntegerType() # Tipped or not
fields[26].dataType = IntegerType() # Tip class
taxi_schema = StructType(fields)

# PARSE FIELDS AND CONVERT DATA TYPE FOR SOME FIELDS
taxi_header = taxi_train_file.filter(lambda l: "medallion" in l)
taxi_temp = taxi_train_file.subtract(taxi_header).map(lambda k: k.split("\t"))\
    .map(lambda p: (p[0],p[1],p[2],p[3],p[4],p[5],p[6],int(p[7]),int(p[8]),int(p[9]),int(p[10]),\
        float(p[11]),float(p[12]),p[13],p[14],p[15],p[16],p[17],p[18],float(p[19]),\
        float(p[20]),float(p[21]),float(p[22]),float(p[23]),float(p[24]),int(p[25]),int(p[26])))

# CREATE DATA FRAME
taxi_train_df = sqlContext.createDataFrame(taxi_temp, taxi_schema)

# CREATE A CLEANED DATA-FRAME BY DROPPING SOME UN-NECESSARY COLUMNS & FILTERING FOR UNDESIRED VALUES OR OUTLIERS
taxi_df_train_cleaned = taxi_train_df.drop('medallion').drop('hack_license').drop('store_and_fwd_flag').drop('pickup_datetime')\
    .drop('dropoff_datetime').drop('pickup_longitude').drop('pickup_latitude').drop('dropoff_latitude')\
    .drop('dropoff_longitude').drop('tip_class').drop('total_amount').drop('tolls_amount').drop('mta_tax')\
    .drop('direct_distance').drop('surcharge')\
    .filter("passenger_count > 0 and passenger_count < 8 AND payment_type in ('CSH', 'CRD') AND tip_amount >= 0 AND tip_amount < 30 AND fare_amount >= 1 AND fare_amount < 150 AND trip_distance > 0 AND trip_distance < 100 AND trip_time_in_secs > 30 AND trip_time_in_secs < 7200")

# CACHE & MATERIALIZE DATA-FRAME IN MEMORY. GOING THROUGH AND COUNTING NUMBER OF ROWS MATERIALIZES THE DATA-FRAME IN MEMORY
taxi_df_train_cleaned.cache()
taxi_df_train_cleaned.count()

# REGISTER DATA-FRAME AS A TEMP-TABLE IN SQL-CONTEXT
taxi_df_train_cleaned.registerTempTable("taxi_train")

# PRINT HOW MUCH TIME IT TOOK TO RUN THE CELL
timeend = datetime.datetime.now()
timedelta = round((timeend-timestart).total_seconds(), 2)
print "Time taken to execute above cell: " + str(timedelta) + " seconds";

```

OUTPUT

Time taken to execute above cell: 276.62 seconds

Data exploration & visualization

Once the data has been brought into Spark, the next step in the data science process is to gain deeper

understanding of the data through exploration and visualization. In this section, we examine the taxi data using SQL queries and plot the target variables and prospective features for visual inspection. Specifically, we plot the frequency of passenger counts in taxi trips, the frequency of tip amounts, and how tips vary by payment amount and type.

Plot a histogram of passenger count frequencies in the sample of taxi trips

This code and subsequent snippets use SQL magic to query the sample and local magic to plot the data.

- **SQL magic (`%%sql`)** The HDInsight PySpark kernel supports easy inline HiveQL queries against the sqlContext. The (-o VARIABLE_NAME) argument persists the output of the SQL query as a Pandas DataFrame on the Jupyter server. This means it is available in the local mode.
- The `%%local` **magic** is used to run code locally on the Jupyter server, which is the headnode of the HDInsight cluster. Typically, you use `%%local` magic in conjunction with the `%%sql` magic with -o parameter. The -o parameter would persist the output of the SQL query locally and then `%%local` magic would trigger the next set of code snippet to run locally against the output of the SQL queries that is persisted locally

The output is automatically visualized after you run the code.

This query retrieves the trips by passenger count.

```
# PLOT FREQUENCY OF PASSENGER COUNTS IN TAXI TRIPS

# SQL QUERY
%%sql -o sqlResults
SELECT passenger_count, COUNT(*) as trip_counts FROM taxi_train WHERE passenger_count > 0 and passenger_count < 7 GROUP BY
passenger_count
```

This code creates a local data-frame from the query output and plots the data. The `%%local` magic creates a local data-frame, `sqlResults`, which can be used for plotting with matplotlib.

NOTE

This PySpark magic is used multiple times in this walkthrough. If the amount of data is large, you should sample to create a data-frame that can fit in local memory.

```
# RUN THE CODE LOCALLY ON THE JUPYTER SERVER
%%local

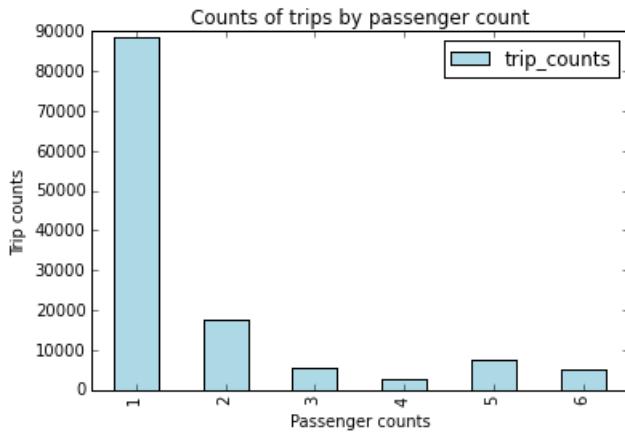
# USE THE JUPYTER AUTO-PLOTTING FEATURE TO CREATE INTERACTIVE FIGURES.
# CLICK ON THE TYPE OF PLOT TO BE GENERATED (E.G. LINE, AREA, BAR ETC.)
sqlResults
```

Here is the code to plot the trips by passenger counts

```
# RUN THE CODE LOCALLY ON THE JUPYTER SERVER AND IMPORT LIBRARIES
%%local
import matplotlib.pyplot as plt
%matplotlib inline

# PLOT PASSENGER NUMBER VS TRIP COUNTS
x_labels = sqlResults['passenger_count'].values
fig = sqlResults[['trip_counts']].plot(kind='bar', facecolor='lightblue')
fig.set_xticklabels(x_labels)
fig.set_title('Counts of trips by passenger count')
fig.set_xlabel('Passenger count in trips')
fig.set_ylabel('Trip counts')
plt.show()
```

OUTPUT



You can select among several different types of visualizations (Table, Pie, Line, Area, or Bar) by using the **Type** menu buttons in the notebook. The Bar plot is shown here.

Plot a histogram of tip amounts and how tip amount varies by passenger count and fare amounts.

Use a SQL query to sample data..

```
# SQL SQUERY
%%sql -q -o sqlResults
SELECT fare_amount, passenger_count, tip_amount, tipped
FROM taxi_train
WHERE passenger_count > 0
AND passenger_count < 7
AND fare_amount > 0
AND fare_amount < 200
AND payment_type in ('CSH', 'CRD')
AND tip_amount > 0
AND tip_amount < 25
```

This code cell uses the SQL query to create three plots the data.

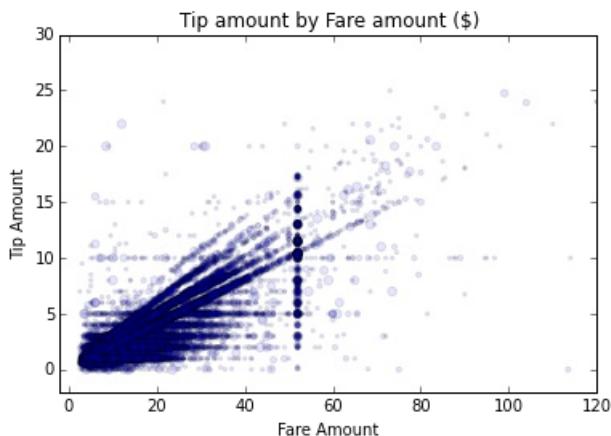
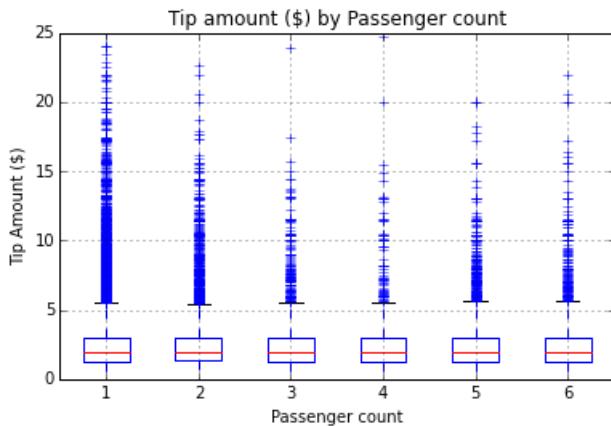
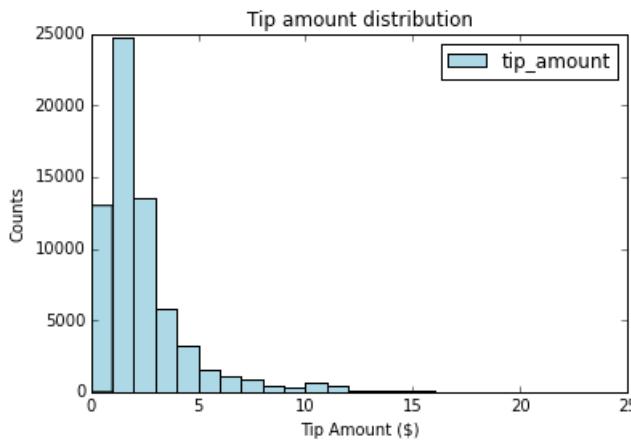
```
# RUN THE CODE LOCALLY ON THE JUPYTER SERVER AND IMPORT LIBRARIES
%%local
%matplotlib inline

# TIP BY PAYMENT TYPE AND PASSENGER COUNT
ax1 = resultsPDDF[['tip_amount']].plot(kind='hist', bins=25, facecolor='lightblue')
ax1.set_title('Tip amount distribution')
ax1.set_xlabel('Tip Amount ($)')
ax1.set_ylabel('Counts')
plt.suptitle('')
plt.show()

# TIP BY PASSENGER COUNT
ax2 = resultsPDDF.boxplot(column=['tip_amount'], by=['passenger_count'])
ax2.set_title('Tip amount ($) by Passenger count')
ax2.set_xlabel('Passenger count')
ax2.set_ylabel('Tip Amount ($)')
plt.suptitle('')
plt.show()

# TIP AMOUNT BY FARE AMOUNT, POINTS ARE SCALED BY PASSENGER COUNT
ax = resultsPDDF.plot(kind='scatter', x='fare_amount', y='tip_amount', c='blue', alpha=0.10, s=5*(resultsPDDF.passenger_count))
ax.set_title('Tip amount by Fare amount ($)')
ax.set_xlabel('Fare Amount')
ax.set_ylabel('Tip Amount')
plt.axis([-2, 120, -2, 30])
plt.show()
```

OUTPUT:



Feature engineering, transformation and data preparation for modeling

This section describes and provides the code for procedures used to prepare data for use in ML modeling. It shows how to do the following tasks:

- Create a new feature by binning hours into traffic time buckets
- Index and on-hot encode categorical features
- Create labeled point objects for input into ML functions
- Create a random sub-sampling of the data and split it into training and testing sets
- Feature scaling
- Cache objects in memory

Create a new feature by binning hours into traffic time buckets

This code shows how to create a new feature by binning hours into traffic time buckets and then how to cache the

resulting data frame in memory. Where Resilient Distributed Datasets (RDDs) and data-frames are used repeatedly, caching leads to improved execution times. Accordingly, we cache RDDs and data-frames at several stages in the walkthrough.

```
# CREATE FOUR BUCKETS FOR TRAFFIC TIMES
sqlStatement = """
    SELECT *,
    CASE
        WHEN (pickup_hour <= 6 OR pickup_hour >= 20) THEN "Night"
        WHEN (pickup_hour >= 7 AND pickup_hour <= 10) THEN "AMRush"
        WHEN (pickup_hour >= 11 AND pickup_hour <= 15) THEN "Afternoon"
        WHEN (pickup_hour >= 16 AND pickup_hour <= 19) THEN "PMRush"
    END as TrafficTimeBins
    FROM taxi_train
"""
taxi_df_train_with_newFeatures = sqlContext.sql(sqlStatement)

# CACHE DATA-FRAME IN MEMORY & MATERIALIZE DF IN MEMORY
# THE .COUNT() GOES THROUGH THE ENTIRE DATA-FRAME,
# MATERIALIZES IT IN MEMORY, AND GIVES THE COUNT OF ROWS.
taxi_df_train_with_newFeatures.cache()
taxi_df_train_with_newFeatures.count()
```

OUTPUT

126050

Index and one-hot encode categorical features

This section shows how to index or encode categorical features for input into the modeling functions. The modeling and predict functions of MLlib require features with categorical input data to be indexed or encoded prior to use.

Depending on the model, you need to index or encode them in different ways. For example, Logistic and Linear Regression models require one-hot encoding, where, for example, a feature with three categories can be expanded into three feature columns, with each containing 0 or 1 depending on the category of an observation. MLlib provides [OneHotEncoder](#) function to do one-hot encoding. This encoder maps a column of label indices to a column of binary vectors, with at most a single one-value. This encoding allows algorithms that expect numerical valued features, such as logistic regression, to be applied to categorical features.

Here is the code to index and encode categorical features:

```

# RECORD START TIME
timestart = datetime.datetime.now()

# LOAD PYSPARK LIBRARIES
from pyspark.ml.feature import OneHotEncoder, StringIndexer, VectorAssembler, OneHotEncoder, VectorIndexer

# INDEX AND ENCODE VENDOR_ID
stringIndexer = StringIndexer(inputCol="vendor_id", outputCol="vendorIndex")
model = stringIndexer.fit(taxi_df_train_with_newFeatures) # Input data-frame is the cleaned one from above
indexed = model.transform(taxi_df_train_with_newFeatures)
encoder = OneHotEncoder(dropLast=False, inputCol="vendorIndex", outputCol="vendorVec")
encoded1 = encoder.transform(indexed)

# INDEX AND ENCODE RATE_CODE
stringIndexer = StringIndexer(inputCol="rate_code", outputCol="rateIndex")
model = stringIndexer.fit(encoded1)
indexed = model.transform(encoded1)
encoder = OneHotEncoder(dropLast=False, inputCol="rateIndex", outputCol="rateVec")
encoded2 = encoder.transform(indexed)

# INDEX AND ENCODE PAYMENT_TYPE
stringIndexer = StringIndexer(inputCol="payment_type", outputCol="paymentIndex")
model = stringIndexer.fit(encoded2)
indexed = model.transform(encoded2)
encoder = OneHotEncoder(dropLast=False, inputCol="paymentIndex", outputCol="paymentVec")
encoded3 = encoder.transform(indexed)

# INDEX AND TRAFFIC TIME BINS
stringIndexer = StringIndexer(inputCol="TrafficTimeBins", outputCol="TrafficTimeBinsIndex")
model = stringIndexer.fit(encoded3)
indexed = model.transform(encoded3)
encoder = OneHotEncoder(dropLast=False, inputCol="TrafficTimeBinsIndex", outputCol="TrafficTimeBinsVec")
encodedFinal = encoder.transform(indexed)

# PRINT ELAPSED TIME
timeend = datetime.datetime.now()
timedelta = round((timeend-timestart).total_seconds(), 2)
print "Time taken to execute above cell: " + str(timedelta) + " seconds";

```

OUTPUT

Time taken to execute above cell: 3.14 seconds

Create labeled point objects for input into ML functions

This section contains code that shows how to index categorical text data as a labeled point data type and encode it so that it can be used to train and test MLlib logistic regression and other classification models. Labeled point objects are Resilient Distributed Datasets (RDD) formatted in a way that is needed as input data by most of ML algorithms in MLlib. A [labeled point](#) is a local vector, either dense or sparse, associated with a label/response.

Here is the code to index and encode text features for binary classification.

```

# FUNCTIONS FOR BINARY CLASSIFICATION

# LOAD LIBRARIES
from pyspark.mllib.regression import LabeledPoint
from numpy import array

# INDEXING CATEGORICAL TEXT FEATURES FOR INPUT INTO TREE-BASED MODELS
def parseRowIndexingBinary(line):
    features = np.array([line.paymentIndex, line.vendorIndex, line.rateIndex, line.pickup_hour, line.weekday,
                        line.passenger_count, line.trip_time_in_secs, line.trip_distance, line.fare_amount])
    labPt = LabeledPoint(line.tipped, features)
    return labPt

# ONE-HOT ENCODING OF CATEGORICAL TEXT FEATURES FOR INPUT INTO LOGISTIC REGRESSION MODELS
def parseRowOneHotBinary(line):
    features = np.concatenate((np.array([line.pickup_hour, line.weekday, line.passenger_count,
                                         line.trip_time_in_secs, line.trip_distance, line.fare_amount]),
                               line.vendorVec.toArray(), line.rateVec.toArray(), line.paymentVec.toArray()), axis=0)
    labPt = LabeledPoint(line.tipped, features)
    return labPt

```

Here is the code to encode and index categorical text features for linear regression analysis.

```

# FUNCTIONS FOR REGRESSION WITH TIP AMOUNT AS TARGET VARIABLE

# ONE-HOT ENCODING OF CATEGORICAL TEXT FEATURES FOR INPUT INTO TREE-BASED MODELS
def parseRowIndexingRegression(line):
    features = np.array([line.paymentIndex, line.vendorIndex, line.rateIndex, line.TrafficTimeBinsIndex,
                        line.pickup_hour, line.weekday, line.passenger_count, line.trip_time_in_secs,
                        line.trip_distance, line.fare_amount])
    labPt = LabeledPoint(line.tip_amount, features)
    return labPt

# INDEXING CATEGORICAL TEXT FEATURES FOR INPUT INTO LINEAR REGRESSION MODELS
def parseRowOneHotRegression(line):
    features = np.concatenate((np.array([line.pickup_hour, line.weekday, line.passenger_count,
                                         line.trip_time_in_secs, line.trip_distance, line.fare_amount]),
                               line.vendorVec.toArray(), line.rateVec.toArray(),
                               line.paymentVec.toArray(), line.TrafficTimeBinsVec.toArray()), axis=0)
    labPt = LabeledPoint(line.tip_amount, features)
    return labPt

```

Create a random sub-sampling of the data and split it into training and testing sets

This code creates a random sampling of the data (25% is used here). Although it is not required for this example due to the size of the dataset, we demonstrate how you can sample here so you know how to use it for your own problem when needed. When samples are large, this can save significant time while training models. Next we split the sample into a training part (75% here) and a testing part (25% here) to use in classification and regression modeling.

```

# RECORD START TIME
timestart = datetime.datetime.now()

# SPECIFY SAMPLING AND SPLITTING FRACTIONS
from pyspark.sql.functions import rand

samplingFraction = 0.25;
trainingFraction = 0.75; testingFraction = (1-trainingFraction);
seed = 1234;
encodedFinalSampled = encodedFinal.sample(False, samplingFraction, seed=seed)

# SPLIT SAMPLED DATA-FRAME INTO TRAIN/TEST, WITH A RANDOM COLUMN ADDED FOR DOING CV (SHOWN LATER)
# INCLUDE RAND COLUMN FOR CREATING CROSS-VALIDATION FOLDS
dfTmpRand = encodedFinalSampled.select("*", rand(0).alias("rand"));
trainData, testData = dfTmpRand.randomSplit([trainingFraction, testingFraction], seed=seed);

# CACHE TRAIN AND TEST DATA
trainData.cache()
testData.cache()

# FOR BINARY CLASSIFICATION TRAINING AND TESTING
indexedTRAINbinary = trainData.map(parseRowIndexingBinary)
indexedTESTbinary = testData.map(parseRowIndexingBinary)
oneHotTRAINbinary = trainData.map(parseRowOneHotBinary)
oneHotTESTbinary = testData.map(parseRowOneHotBinary)

# FOR REGRESSION TRAINING AND TESTING
indexedTRAINreg = trainData.map(parseRowIndexingRegression)
indexedTESTreg = testData.map(parseRowIndexingRegression)
oneHotTRAINreg = trainData.map(parseRowOneHotRegression)
oneHotTESTreg = testData.map(parseRowOneHotRegression)

# PRINT ELAPSED TIME
timeend = datetime.datetime.now()
timedelta = round((timeend-timestart).total_seconds(), 2)
print "Time taken to execute above cell: " + str(timedelta) + " seconds";

```

OUTPUT

Time taken to execute above cell: 0.31 seconds

Feature scaling

Feature scaling, also known as data normalization, insures that features with widely disbursed values are not given excessive weigh in the objective function. The code for feature scaling uses the [StandardScaler](#) to scale the features to unit variance. It is provided by MLlib for use in linear regression with Stochastic Gradient Descent (SGD), a popular algorithm for training a wide range of other machine learning models such as regularized regressions or support vector machines (SVM).

TIP

We have found the `LinearRegressionWithSGD` algorithm to be sensitive to feature scaling.

Here is the code to scale variables for use with the regularized linear SGD algorithm.

```

# RECORD START TIME
timestart = datetime.datetime.now()

# LOAD PYSPARK LIBRARIES
from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.linalg import Vectors
from pyspark.mllib.feature import StandardScaler, StandardScalerModel
from pyspark.mllib.util import MLUtils

# SCALE VARIABLES FOR REGULARIZED LINEAR SGD ALGORITHM
label = oneHotTRAINreg.map(lambda x: x.label)
features = oneHotTRAINreg.map(lambda x: x.features)
scaler = StandardScaler(withMean=False, withStd=True).fit(features)
dataTMP = label.zip(scaler.transform(features.map(lambda x: Vectors.dense(x.toArray()))))
oneHotTRAINregScaled = dataTMP.map(lambda x: LabeledPoint(x[0], x[1]))

label = oneHotTESTreg.map(lambda x: x.label)
features = oneHotTESTreg.map(lambda x: x.features)
scaler = StandardScaler(withMean=False, withStd=True).fit(features)
dataTMP = label.zip(scaler.transform(features.map(lambda x: Vectors.dense(x.toArray()))))
oneHotTESTregScaled = dataTMP.map(lambda x: LabeledPoint(x[0], x[1]))

# PRINT ELAPSED TIME
timeend = datetime.datetime.now()
timedelta = round((timeend-timestart).total_seconds(), 2)
print "Time taken to execute above cell: " + str(timedelta) + " seconds";

```

OUTPUT

Time taken to execute above cell: 11.67 seconds

Cache objects in memory

The time taken for training and testing of ML algorithms can be reduced by caching the input data frame objects used for classification, regression and, scaled features.

```

# RECORD START TIME
timestart = datetime.datetime.now()

# FOR BINARY CLASSIFICATION TRAINING AND TESTING
indexedTRAINbinary.cache()
indexedTESTbinary.cache()
oneHotTRAINbinary.cache()
oneHotTESTbinary.cache()

# FOR REGRESSION TRAINING AND TESTING
indexedTRAINreg.cache()
indexedTESTreg.cache()
oneHotTRAINreg.cache()
oneHotTESTreg.cache()

# SCALED FEATURES
oneHotTRAINregScaled.cache()
oneHotTESTregScaled.cache()

# PRINT ELAPSED TIME
timeend = datetime.datetime.now()
timedelta = round((timeend-timestart).total_seconds(), 2)
print "Time taken to execute above cell: " + str(timedelta) + " seconds";

```

OUTPUT

Time taken to execute above cell: 0.13 seconds

Predict whether or not a tip is paid with binary classification models

This section shows how use three models for the binary classification task of predicting whether or not a tip is paid for a taxi trip. The models presented are:

- Logistic regression
- Random forest
- Gradient Boosting Trees

Each model building code section is split into steps:

1. **Model training** data with one parameter set
2. **Model evaluation** on a test data set with metrics
3. **Saving model** in blob for future consumption

We show how to do cross-validation (CV) with parameter sweeping in two ways:

1. Using **generic** custom code which can be applied to any algorithm in MLlib and to any parameter sets in an algorithm.
2. Using the **pySpark CrossValidator pipeline function**. Note that although convenient, based on our experience, CrossValidator has a few limitations for Spark 1.5.0:
 - Pipeline models cannot be saved/persisted for future consumption.
 - Cannot be used for every parameter in a model.
 - Cannot be used for every MLlib algorithm.

Generic cross validation and hyperparameter sweeping used with the logistic regression algorithm for binary classification

The code in this section shows how to train, evaluate, and save a logistic regression model with [LBFGS](#) that predicts whether or not a tip is paid for a trip in the NYC taxi trip and fare dataset. The model is trained using cross validation (CV) and hyperparameter sweeping implemented with custom code that can be applied to any of the learning algorithms in MLlib.

NOTE

The execution of this custom CV code can take several minutes.

Train the logistic regression model using CV and hyperparameter sweeping

```
# LOGISTIC REGRESSION CLASSIFICATION WITH CV AND HYPERPARAMETER SWEEPING

# GET ACCURACY FOR HYPERPARAMETERS BASED ON CROSS-VALIDATION IN TRAINING DATA-SET

# RECORD START TIME
timestart = datetime.datetime.now()

# LOAD LIBRARIES
from pyspark.mllib.classification import LogisticRegressionWithLBFGS
from pyspark.mllib.evaluation import BinaryClassificationMetrics

# CREATE PARAMETER GRID FOR LOGISTIC REGRESSION PARAMETER SWEEP
from sklearn.grid_search import ParameterGrid
grid = [{"regParam": [0.01, 0.1], "iterations": [5, 10], "regType": ["l1", "l2"], "tolerance": [1e-3, 1e-4]}]
paramGrid = list(ParameterGrid(grid))
numModels = len(paramGrid)

# SET NUM FOLDS AND NUM PARAMETER SETS TO SWEEP ON
nFolds = 3;
h = 1.0 / nFolds;
```

```

metricSum = np.zeros(numModels);

# BEGIN CV WITH PARAMETER SWEEP
for i in range(nFolds):
    # Create training and x-validation sets
    validateLB = i * h
    validateUB = (i + 1) * h
    condition = (trainData["rand"] >= validateLB) & (trainData["rand"] < validateUB)
    validation = trainData.filter(condition)
    # Create LabeledPoints from data-frames
    if i > 0:
        trainCVLabPt.unpersist()
        validationLabPt.unpersist()
    trainCV = trainData.filter(~condition)
    trainCVLabPt = trainCV.map(parseRowOneHotBinary)
    trainCVLabPt.cache()
    validationLabPt = validation.map(parseRowOneHotBinary)
    validationLabPt.cache()
    # For parameter sets compute metrics from x-validation
    for j in range(numModels):
        regt = paramGrid[j]['regType']
        regp = paramGrid[j]['regParam']
        iters = paramGrid[j]['iterations']
        tol = paramGrid[j]['tolerance']
        # Train logistic regression model with hyperparameter set
        model = LogisticRegressionWithLBFGS.train(trainCVLabPt, regType=regt, iterations=iters,
                                                regParam=regp, tolerance=tol, intercept=True)
        predictionAndLabels = validationLabPt.map(lambda lp: (float(model.predict(lp.features)), lp.label))
        # Use ROC-AUC as accuracy metrics
        validMetrics = BinaryClassificationMetrics(predictionAndLabels)
        metric = validMetrics.areaUnderROC
        metricSum[j] += metric

avgAcc = metricSum / nFolds;
bestParam = paramGrid[np.argmax(avgAcc)];

# UNPERSIST OBJECTS
trainCVLabPt.unpersist()
validationLabPt.unpersist()

# TRAIN ON FULL TRAINING SET USING BEST PARAMETERS FROM CV/PARAMETER SWEEP
logitBest = LogisticRegressionWithLBFGS.train(oneHotTRAINbinary, regType=bestParam['regType'],
                                              iterations=bestParam['iterations'],
                                              regParam=bestParam['regParam'], tolerance=bestParam['tolerance'],
                                              intercept=True)

# PRINT COEFFICIENTS AND INTERCEPT OF THE MODEL
# NOTE: There are 20 coefficient terms for the 10 features,
#       and the different categories for features: vendorVec (2), rateVec, paymentVec (6), TrafficTimeBins Vec (4)
print("Coefficients: " + str(logitBest.weights))
print("Intercept: " + str(logitBest.intercept))

# PRINT ELAPSED TIME
timeend = datetime.datetime.now()
timedelta = round((timeend - timestamp).total_seconds(), 2)
print("Time taken to execute above cell: " + str(timedelta) + " seconds");

```

OUTPUT

Coefficients: [0.0082065285375, -0.0223675576104, -0.0183812028036, -3.48124578069e-05, -0.00247646947233, -0.00165897881503, 0.0675394837328, -0.111823113101, -0.324609912762, -0.204549780032, -1.36499216354, 0.591088507921, -0.664263411392, -1.00439726852, 3.46567827545, -3.51025855172, -0.0471341112232, -0.043521833294, 0.000243375810385, 0.054518719222]

Intercept: -0.0111216486893

Time taken to execute above cell: 14.43 seconds

Evaluate the binary classification model with standard metrics

The code in this section shows how to evaluate a logistic regression model against a test data-set, including a plot of the ROC curve.

```
# RECORD START TIME
timestart = datetime.datetime.now()

#IMPORT LIBRARIES
from sklearn.metrics import roc_curve, auc
from pyspark.mllib.evaluation import BinaryClassificationMetrics
from pyspark.mllib.evaluation import MulticlassMetrics

# PREDICT ON TEST DATA WITH BEST/FINAL MODEL
predictionAndLabels = oneHotTESTbinary.map(lambda lp: (float(logitBest.predict(lp.features))), lp.label)

# INSTANTIATE METRICS OBJECT
metrics = BinaryClassificationMetrics(predictionAndLabels)

# AREA UNDER PRECISION-RECALL CURVE
print("Area under PR = %s" % metrics.areaUnderPR)

# AREA UNDER ROC CURVE
print("Area under ROC = %s" % metrics.areaUnderROC)
metrics = MulticlassMetrics(predictionAndLabels)

# OVERALL STATISTICS
precision = metrics.precision()
recall = metrics.recall()
f1Score = metrics.fMeasure()
print("Summary Stats")
print("Precision = %s" % precision)
print("Recall = %s" % recall)
print("F1 Score = %s" % f1Score)

# OUTPUT PROBABILITIES AND REGISTER TEMP TABLE
logitBest.clearThreshold(); # This clears threshold for classification (0.5) and outputs probabilities
predictionAndLabelsDF = predictionAndLabels.toDF()
predictionAndLabelsDF.registerTempTable("tmp_results");

# PRINT ELAPSED TIME
timeend = datetime.datetime.now()
timedelta = round((timeend-timestart).total_seconds(), 2)
print "Time taken to execute above cell: " + str(timedelta) + " seconds";
```

OUTPUT

Area under PR = 0.985336538462

Area under ROC = 0.983383274312

Summary Stats

Precision = 0.984174341679

Recall = 0.984174341679

F1 Score = 0.984174341679

Time taken to execute above cell: 2.67 seconds

Plot the ROC curve.

The `predictionAndLabelsDF` is registered as a table, `tmp_results`, in the previous cell. `tmp_results` can be used to do

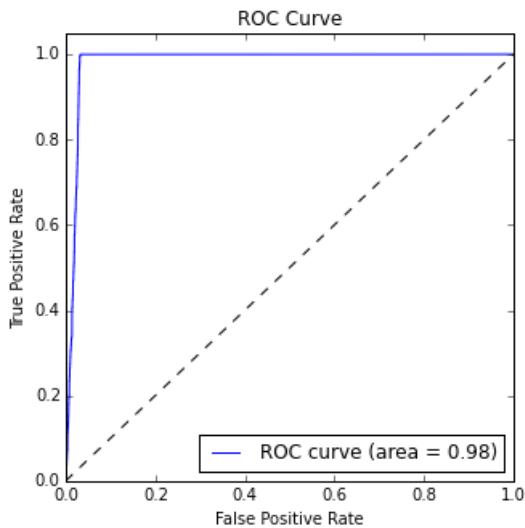
queries and output results into the sqlResults data-frame for plotting. Here is the code.

```
# QUERY RESULTS  
%%sql -q -o sqlResults  
SELECT * from tmp_results
```

Here is the code to make predictions and plot the ROC-curve.

```
# MAKE PREDICTIONS AND PLOT ROC-CURVE  
  
# RUN THE CODE LOCALLY ON THE JUPYTER SERVER AND IMPORT LIBRARIES  
%%local  
%matplotlib inline  
from sklearn.metrics import roc_curve,auc  
  
#PREDICTIONS  
predictions_pddf=sqlResults.rename(columns={'_1':'probability','_2':'label'})  
prob = predictions_pddf["probability"]  
fpr,tpr,thresholds = roc_curve(predictions_pddf['label'], prob, pos_label=1);  
roc_auc = auc(fpr, tpr)  
  
# PLOT ROC CURVES  
plt.figure(figsize=(5,5))  
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)  
plt.plot([0, 1], [0, 1], 'k--')  
plt.xlim([0.0, 1.0])  
plt.ylim([0.0, 1.05])  
plt.xlabel('False Positive Rate')  
plt.ylabel('True Positive Rate')  
plt.title('ROC Curve')  
plt.legend(loc="lower right")  
plt.show()
```

OUTPUT



Persist model in a blob for future consumption

The code in this section shows how to save the logistic regression model for consumption.

```
# RECORD START TIME
timestart = datetime.datetime.now()

# LOAD PYSPARK LIBRARIES
from pyspark.mllib.classification import LogisticRegressionModel

# PERSIST MODEL
datestamp = unicode(datetime.datetime.now()).replace(' ','_').replace(':','_');
logisticregressionfilename = "LogisticRegressionWithLBFGS_" + datestamp;
dirfilename = modelDir + logisticregressionfilename;

logitBest.save(sc, dirfilename);

# PRINT ELAPSED TIME
timeend = datetime.datetime.now()
timedelta = round((timeend-timestart).total_seconds(), 2)
print "Time taken to execute above cell: " + str(timedelta) + " seconds";
```

OUTPUT

Time taken to execute above cell: 34.57 seconds

Use MLlib's CrossValidator pipeline function with logistic regression (Elastic regression) model

The code in this section shows how to train, evaluate, and save a logistic regression model with [LBFGS](#) that predicts whether or not a tip is paid for a trip in the NYC taxi trip and fare dataset. The model is trained using cross validation (CV) and hyperparameter sweeping implemented with the MLlib CrossValidator pipeline function for CV with parameter sweep.

NOTE

The execution of this MLlib CV code can take several minutes.

```

# RECORD START TIME
timestart = datetime.datetime.now()

# LOAD PYSPARK LIBRARIES
from pyspark.ml.classification import LogisticRegression
from pyspark.ml import Pipeline
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from sklearn.metrics import roc_curve, auc

# DEFINE ALGORITHM / MODEL
lr = LogisticRegression()

# DEFINE GRID PARAMETERS
paramGrid = ParamGridBuilder().addGrid(lr.regParam, (0.01, 0.1))\
    .addGrid(lr.maxIter, (5, 10))\
    .addGrid(lr.tol, (1e-4, 1e-5))\
    .addGrid(lr.elasticNetParam, (0.25, 0.75))\
    .build()

# DEFINE CV WITH PARAMETER SWEEP
cv = CrossValidator(estimator=lr,
                     estimatorParamMaps=paramGrid,
                     evaluator=BinaryClassificationEvaluator(),
                     numFolds=3)

# CONVERT TO DATA-FRAME: THIS DOES NOT RUN ON RDDS
trainDataFrame = sqlContext.createDataFrame(oneHotTRAINbinary, ["features", "label"])

# TRAIN WITH CROSS-VALIDATION
cv_model = cv.fit(trainDataFrame)

## PREDICT AND EVALUATE ON TEST DATA-SET

# USE TEST DATASET FOR PREDICTION
testDataFrame = sqlContext.createDataFrame(oneHotTESTbinary, ["features", "label"])
test_predictions = cv_model.transform(testDataFrame)

# PRINT ELAPSED TIME
timeend = datetime.datetime.now()
timedelta = round((timeend-timestart).total_seconds(), 2)
print "Time taken to execute above cell: " + str(timedelta) + " seconds";

```

OUTPUT

Time taken to execute above cell: 107.98 seconds

Plot the ROC curve.

The *predictionAndLabelsDF* is registered as a table, *tmp_results*, in the previous cell. *tmp_results* can be used to do queries and output results into the *sqlResults* data-frame for plotting. Here is the code.

```

# QUERY RESULTS
%%sql -q -o sqlResults
SELECT label, prediction, probability from tmp_results

```

Here is the code to plot the ROC curve.

```

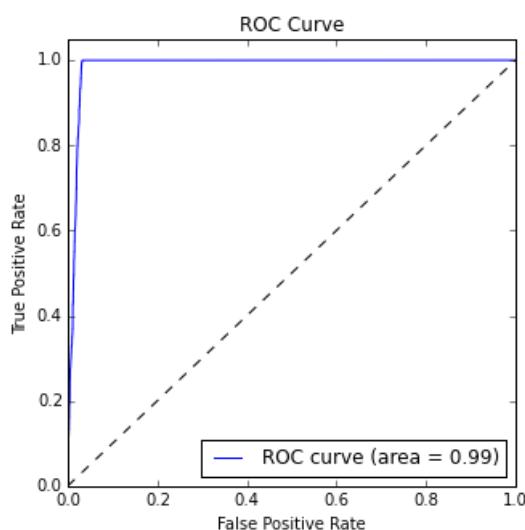
# RUN THE CODE LOCALLY ON THE JUPYTER SERVER AND IMPORT LIBRARIES
%%local
from sklearn.metrics import roc_curve,auc

# ROC CURVE
prob = [x["values"][1] for x in sqlResults[["probability"]]]
fpr, tpr, thresholds = roc_curve(sqlResults[["label"]], prob, pos_label=1);
roc_auc = auc(fpr, tpr)

#PLOT
plt.figure(figsize=(5,5))
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()

```

OUTPUT



Random forest classification

The code in this section shows how to train, evaluate, and save a random forest regression that predicts whether or not a tip is paid for a trip in the NYC taxi trip and fare dataset.

```

# RECORD START TIME
timestart = datetime.datetime.now()

# LOAD PYSPARK LIBRARIES
from pyspark.mllib.tree import RandomForest, RandomForestModel
from pyspark.mllib.util import MLUtils
from pyspark.mllib.evaluation import BinaryClassificationMetrics
from pyspark.mllib.evaluation import MulticlassMetrics

# SPECIFY NUMBER OF CATEGORIES FOR CATEGORICAL FEATURES. FEATURE #0 HAS 2 CATEGORIES, FEATURE #2 HAS 2 CATEGORIES,
# AND SO ON
categoricalFeaturesInfo={0:2, 1:2, 2:6, 3:4}

# TRAIN RANDOMFOREST MODEL
rfModel = RandomForest.trainClassifier(indexedTRAINbinary, numClasses=2,
                                       categoricalFeaturesInfo=categoricalFeaturesInfo,
                                       numTress=25, featureSubsetStrategy="auto",
                                       impurity='gini', maxDepth=5, maxBins=32)
## UN-COMMENT IF YOU WANT TO PRING TREES
#print("Learned classification forest model:")
#print(rfModel.toDebugString())

# PREDICT ON TEST DATA AND EVALUATE
predictions = rfModel.predict(indexedTESTbinary.map(lambda x: x.features))
predictionAndLabels = indexedTESTbinary.map(lambda lp: lp.label).zip(predictions)

# AREA UNDER ROC CURVE
metrics = BinaryClassificationMetrics(predictionAndLabels)
print("Area under ROC = %s" % metrics.areaUnderROC)

# PERSIST MODEL IN BLOB
datestamp = unicode(datetime.datetime.now()).replace(' ', '_').replace(':', '_');
rfclassificationfilename = "RandomForestClassification_" + datestamp;
dirfilename = modelDir + rfclassificationfilename;

rfModel.save(sc, dirfilename);

# PRINT ELAPSED TIME
timeend = datetime.datetime.now()
timedelta = round((timeend-timestart).total_seconds(), 2)
print "Time taken to execute above cell: " + str(timedelta) + " seconds";

```

OUTPUT

Area under ROC = 0.985336538462

Time taken to execute above cell: 26.72 seconds

Gradient boosting trees classification

The code in this section shows how to train, evaluate, and save a gradient boosting trees model that predicts whether or not a tip is paid for a trip in the NYC taxi trip and fare dataset.

```

# RECORD START TIME
timestart = datetime.datetime.now()

# LOAD PYSPARK LIBRARIES
from pyspark.mllib.tree import GradientBoostedTrees, GradientBoostedTreesModel

# SPECIFY NUMBER OF CATEGORIES FOR CATEGORICAL FEATURES. FEATURE #0 HAS 2 CATEGORIES, FEATURE #2 HAS 2 CATEGORIES,
# AND SO ON
categoricalFeaturesInfo={0:2, 1:2, 2:6, 3:4}

gbtModel = GradientBoostedTrees.trainClassifier(indexedTRAINbinary, categoricalFeaturesInfo=categoricalFeaturesInfo,
                                                numIterations=10)
## UNCOMMENT IF YOU WANT TO PRINT TREE DETAILS
#print("Learned classification GBT model:")
#print(bgtModel.toDebugString())

# PREDICT ON TEST DATA AND EVALUATE
predictions = gbtModel.predict(indexedTESTbinary.map(lambda x: x.features))
predictionAndLabels = indexedTESTbinary.map(lambda lp: lp.label).zip(predictions)

# Area under ROC curve
metrics = BinaryClassificationMetrics(predictionAndLabels)
print("Area under ROC = %s" % metrics.areaUnderROC)

# PERSIST MODEL IN A BLOB
datestamp = unicode(datetime.datetime.now()).replace(' ', '_').replace(':', '_');
btclassificationfilename = "GradientBoostingTreeClassification_" + datestamp;
dirfilename = modelDir + btclassificationfilename;

gbtModel.save(sc, dirfilename)

# PRINT ELAPSED TIME
timeend = datetime.datetime.now()
timedelta = round((timeend-timestart).total_seconds(), 2)
print "Time taken to execute above cell: " + str(timedelta) + " seconds";

```

OUTPUT

Area under ROC = 0.985336538462

Time taken to execute above cell: 28.13 seconds

Predict tip amount with regression models (not using CV)

This section shows how use three models for the regression task of predicting the amount of the tip paid for a taxi trip based on other tip features. The models presented are:

- Regularized linear regression
- Random forest
- Gradient Boosting Trees

These models were described in the introduction. Each model building code section is split into steps:

1. **Model training** data with one parameter set
2. **Model evaluation** on a test data set with metrics
3. **Saving model** in blob for future consumption

AZURE NOTE: Cross-validation is not used with the three regression models in this section, since this was shown in detail for the logistic regression models. An example showing how to use CV with Elastic Net for linear regression is provided in the Appendix of this topic.

AZURE NOTE: In our experience, there can be issues with convergence of LinearRegressionWithSGD models,

and parameters need to be changed/optimized carefully for obtaining a valid model. Scaling of variables significantly helps with convergence. Elastic net regression, shown in the Appendix to this topic, can also be used instead of LinearRegressionWithSGD.

Linear regression with SGD

The code in this section shows how to use scaled features to train a linear regression that uses stochastic gradient descent (SGD) for optimization, and how to score, evaluate, and save the model in Azure Blob Storage (WASB).

TIP

In our experience, there can be issues with the convergence of LinearRegressionWithSGD models, and parameters need to be changed/optimized carefully for obtaining a valid model. Scaling of variables significantly helps with convergence.

```
# LINEAR REGRESSION WITH SGD

# RECORD START TIME
timestart = datetime.datetime.now()

# LOAD LIBRARIES
from pyspark.mllib.regression import LabeledPoint, LinearRegressionWithSGD, LinearRegressionModel
from pyspark.mllib.evaluation import RegressionMetrics
from scipy import stats

# USE SCALED FEATURES TO TRAIN MODEL
linearModel = LinearRegressionWithSGD.train(oneHotTRAINregScaled, iterations=100, step = 0.1, regType='l2', regParam=0.1, intercept = True)

# PRINT COEFFICIENTS AND INTERCEPT OF THE MODEL
# NOTE: There are 20 coefficient terms for the 10 features,
#       and the different categories for features: vendorVec (2), rateVec, paymentVec (6), TrafficTimeBins Vec (4)
print("Coefficients: " + str(linearModel.weights))
print("Intercept: " + str(linearModel.intercept))

# SCORE ON SCALED TEST DATA-SET & EVALUATE
predictionAndLabels = oneHotTESTregScaled.map(lambda lp: (float(linearModel.predict(lp.features)), lp.label))
testMetrics = RegressionMetrics(predictionAndLabels)

print("RMSE=%s" % testMetrics.rootMeanSquaredError)
print("R-sqr=%s" % testMetrics.r2)

# SAVE MODEL IN BLOB
datestamp = unicode(datetime.datetime.now()).replace(' ','').replace(':', '_');
linearregressionfilename = "LinearRegressionWithSGD_" + datestamp;
dirfilename = modelDir + linearregressionfilename;

linearModel.save(sc, dirfilename)

# PRINT ELAPSED TIME
timeend = datetime.datetime.now()
timedelta = round((timeend-timestart).total_seconds(), 2)
print "Time taken to execute above cell: " + str(timedelta) + " seconds";
```

OUTPUT

Coefficients: [0.0141707753435, -0.0252930927087, -0.0231442517137, 0.247070902996, 0.312544147152, 0.360296120645, 0.0122079566092, -0.00456498588241, -0.0898228505177, 0.0714046248793, 0.102171263868, 0.100022455632, -0.00289545676449, -0.00791124681938, 0.54396316518, -0.536293513569, 0.0119076553369, -0.0173039244582, 0.0119632796147, 0.00146764882502]

Intercept: 0.854507624459

RMSE = 1.23485131376

R-sqr = 0.597963951127

Time taken to execute above cell: 38.62 seconds

Random Forest regression

The code in this section shows how to train, evaluate, and save a random forest model that predicts tip amount for the NYC taxi trip data.

NOTE

Cross-validation with parameter sweeping using custom code is provided in the appendix.

```
#PREDICT TIP AMOUNTS USING RANDOM FOREST

# RECORD START TIME
timestart= datetime.datetime.now()

# LOAD PYSPARK LIBRARIES
from pyspark.mllib.tree import RandomForest, RandomForestModel
from pyspark.mllib.util import MLUtils
from pyspark.mllib.evaluation import RegressionMetrics

# TRAIN MODEL
categoricalFeaturesInfo={0:2, 1:2, 2:6, 3:4}
rfModel = RandomForest.trainRegressor(indexedTRAINreg, categoricalFeaturesInfo=categoricalFeaturesInfo,
                                      numTrees=25, featureSubsetStrategy="auto",
                                      impurity='variance', maxDepth=10, maxBins=32)
# UN-COMMENT IF YOU WANT TO PRING TREES
#print('Learned classification forest model:')
#print(rfModel.toDebugString())

# PREDICT AND EVALUATE ON TEST DATA-SET
predictions = rfModel.predict(indexedTESTreg.map(lambda x: x.features))
predictionAndLabels = oneHotTESTreg.map(lambda lp: lp.label.zip(predictions))

testMetrics = RegressionMetrics(predictionAndLabels)
print("RMSE=%s" % testMetrics.rootMeanSquaredError)
print("R-sqr=%s" % testMetrics.r2)

# SAVE MODEL IN BLOB
datestamp = unicode(datetime.datetime.now()).replace(' ', '_').replace(':', '_');
rfregressionfilename = "RandomForestRegression_" + datestamp;
dirfilename = modelDir + rfregressionfilename;

rfModel.save(sc, dirfilename);

# PRINT ELAPSED TIME
timeend = datetime.datetime.now()
timedelta = round((timeend-timestart).total_seconds(), 2)
print "Time taken to execute above cell: " + str(timedelta) + " seconds";
```

OUTPUT

RMSE = 0.931981967875

R-sqr = 0.733445485802

Time taken to execute above cell: 25.98 seconds

Gradient boosting trees regression

The code in this section shows how to train, evaluate, and save a gradient boosting trees model that predicts tip amount for the NYC taxi trip data.

**Train and evaluate **

```
#PREDICT TIP AMOUNTS USING GRADIENT BOOSTING TREES

# RECORD START TIME
timestart= datetime.datetime.now()

# LOAD PYSPARK LIBRARIES
from pyspark.mllib.tree import GradientBoostedTrees, GradientBoostedTreesModel
from pyspark.mllib.util import MLUtils

# TRAIN MODEL
categoricalFeaturesInfo={0:2, 1:2, 2:6, 3:4}
gbtModel = GradientBoostedTrees.trainRegressor(indexedTRAINreg, categoricalFeaturesInfo=categoricalFeaturesInfo,
                                               numIterations=10, maxBins=32, maxDepth = 4, learningRate=0.1)

# EVALUATE A TEST DATA-SET
predictions = gbtModel.predict(indexedTESTreg.map(lambda x: x.features))
predictionAndLabels = indexedTESTreg.map(lambda lp: lp.label).zip(predictions)

testMetrics = RegressionMetrics(predictionAndLabels)
print("RMSE = %s" % testMetrics.rootMeanSquaredError)
print("R-sqr = %s" % testMetrics.r2)

# PLOT SCATTER-PLOT BETWEEN ACTUAL AND PREDICTED TIP VALUES
test_predictions=sqlContext.createDataFrame(predictionAndLabels)
test_predictions_pddf=test_predictions.toPandas()

# SAVE MODEL IN BLOB
datestamp = unicode(datetime.datetime.now()).replace(' ','').replace(':','_');
btregressionfilename = "GradientBoostingTreeRegression_" + datestamp;
dirfilename = modelDir + btregressionfilename;
gbtModel.save(sc, dirfilename)

# PRINT ELAPSED TIME
timeend = datetime.datetime.now()
timedelta = round((timeend-timestart).total_seconds(), 2)
print "Time taken to execute above cell: " + str(timedelta) + " seconds";
```

OUTPUT

RMSE = 0.928172197114

R-sqr = 0.732680354389

Time taken to execute above cell: 20.9 seconds

Plot

tmp_results is registered as a Hive table in the previous cell. Results from the table are output into the *sqlResults* data-frame for plotting. Here is the code

```
# PLOT SCATTER-PLOT BETWEEN ACTUAL AND PREDICTED TIP VALUES

# SELECT RESULTS
%%sql -o sqlResults
SELECT * from tmp_results
```

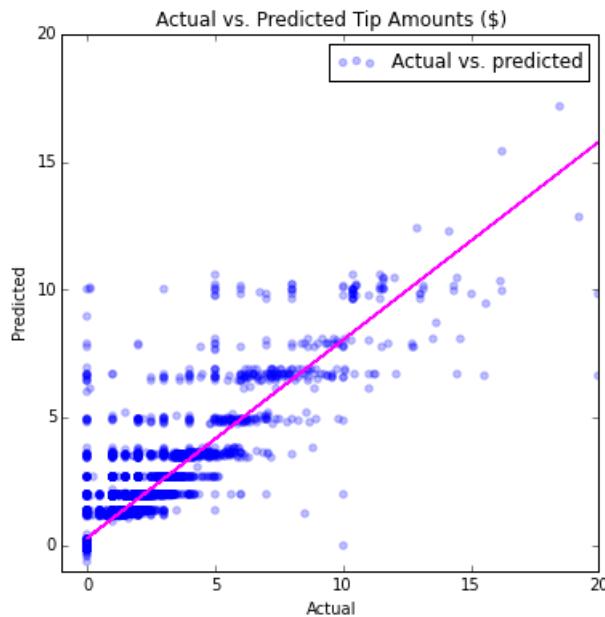
Here is the code to plot the data using the Jupyter server.

```

# RUN THE CODE LOCALLY ON THE JUPYTER SERVER AND IMPORT LIBRARIES
%%local
import numpy as np

# PLOT
ax=plt.scatter(sqlResults['_1'], sqlResults['_2'], color='blue', alpha=0.25, label='Actual vs. predicted');
fit = np.polyfit(sqlResults['_1'], sqlResults['_2'], deg=1)
ax.set_title('Actual vs. Predicted Tip Amounts ($)')
ax.set_xlabel("Actual")
ax.set_ylabel("Predicted")
ax.plot(sqlResults['_1'], fit[0] * sqlResults['_1'] + fit[1], color='magenta')
plt.axis([-1, 15, -1, 15])
plt.show(ax)

```



Appendix: Additional regression tasks using cross validation with parameter sweeps

This appendix contains code showing how to do CV using Elastic net for linear regression and how to do CV with parameter sweep using custom code for random forest regression.

Cross validation using Elastic net for linear regression

The code in this section shows how to do cross validation using Elastic net for linear regression and how to evaluate the model against test data.

```

### CV USING ELASTIC NET FOR LINEAR REGRESSION

# RECORD START TIME
timestart = datetime.datetime.now()

# LOAD PYSPARK LIBRARIES
from pyspark.ml.regression import LinearRegression
from pyspark.ml import Pipeline
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder

# DEFINE ALGORITHM/MODEL
lr = LinearRegression()

# DEFINE GRID PARAMETERS
paramGrid = ParamGridBuilder().addGrid(lr.regParam, (0.01, 0.1))\
    .addGrid(lr.maxIter, (5, 10))\
    .addGrid(lr.tol, (1e-4, 1e-5))\
    .addGrid(lr.elasticNetParam, (0.25, 0.75))\
    .build()

# DEFINE PIPELINE
# SIMPLY THE MODEL HERE, WITHOUT TRANSFORMATIONS
pipeline = Pipeline(stages=[lr])

# DEFINE CV WITH PARAMETER SWEEP
cv = CrossValidator(estimator= lr,
                     estimatorParamMaps=paramGrid,
                     evaluator=RegressionEvaluator(),
                     numFolds=3)

# CONVERT TO DATA FRAME, AS CROSSVALIDATOR WON'T RUN ON RDDS
trainDataFrame = sqlContext.createDataFrame(oneHotTRAINreg, ["features", "label"])

# TRAIN WITH CROSS-VALIDATION
cv_model = cv.fit(trainDataFrame)

# EVALUATE MODEL ON TEST SET
testDataFrame = sqlContext.createDataFrame(oneHotTESTreg, ["features", "label"])

# MAKE PREDICTIONS ON TEST DOCUMENTS
# cvModel uses the best model found (lrModel).
predictionAndLabels = cv_model.transform(testDataFrame)

# CONVERT TO DF AND SAVE REGISER DF AS TABLE
predictionAndLabels.registerTempTable("tmp_results");

# PRINT ELAPSED TIME
timeend = datetime.datetime.now()
timedelta = round((timeend-timestart).total_seconds(), 2)
print "Time taken to execute above cell: " + str(timedelta) + " seconds";

```

OUTPUT

Time taken to execute above cell: 161.21 seconds

Evaluate with R-SQR metric

tmp_results is registered as a Hive table in the previous cell. Results from the table are output into the *sqlResults* data-frame for plotting. Here is the code

```

# SELECT RESULTS
%%sql -q -o sqlResults
SELECT label,prediction from tmp_results

```

Here is the code to calculate R-sqr.

```
# RUN THE CODE LOCALLY ON THE JUPYTER SERVER AND IMPORT LIBRARIES
%%local
from scipy import stats

#R-SQR TEST METRIC
corstats = stats.linregress(sqlResults['label'],sqlResults['prediction'])
r2 = (corstats[2]*corstats[2])
print("R-sqr=%s" % r2)
```

OUTPUT

R-sqr = 0.619184907088

Cross validation with parameter sweep using custom code for random forest regression

The code in this section shows how to do cross validation with parameter sweep using custom code for random forest regression and how to evaluate the model against test data.

```
# RECORD START TIME
timestart= datetime.datetime.now()

# LOAD PYSPARK LIBRARIES
# GET ACCURACY FOR HYPERPARAMETERS BASED ON CROSS-VALIDATION IN TRAINING DATA-SET
from pyspark.mllib.tree import RandomForest, RandomForestModel
from pyspark.mllib.util import MLUtils
from pyspark.mllib.evaluation import RegressionMetrics
from sklearn.grid_search import ParameterGrid

## CREATE PARAMETER GRID
grid = [{'maxDepth': [5,10], 'numTrees': [25,50]}]
paramGrid = list(ParameterGrid(grid))

## SPECIFY LEVELS OF CATEGORICAL VARIABLES
categoricalFeaturesInfo={0:2, 1:2, 2:6, 3:4}

# SPECIFY NUMFOLDS AND ARRAY TO HOLD METRICS
nFolds = 3;
numModels = len(paramGrid)
h = 1.0 / nFolds;
metricSum = np.zeros(numModels);

for i in range(nFolds):
    # Create training and x-validation sets
    validateLB = i * h
    validateUB = (i + 1) * h
    condition = (trainData["rand"] >= validateLB) & (trainData["rand"] < validateUB)
    validation = trainData.filter(condition)
    # Create labeled points from data-frames
    if i > 0:
        trainCVLabPt.unpersist()
        validationLabPt.unpersist()
    trainCV = trainData.filter(~condition)
    trainCVLabPt = trainCV.map(parseRowIndexingRegression)
    trainCVLabPt.cache()
    validationLabPt = validation.map(parseRowIndexingRegression)
    validationLabPt.cache()
    # For parameter sets compute metrics from x-validation
    for j in range(numModels):
        maxD = paramGrid[j]['maxDepth']
        numT = paramGrid[j]['numTrees']
        # Train logistic regression model with hyperparameter set
        rfModel = RandomForest.trainRegressor(trainCVLabPt, categoricalFeaturesInfo=categoricalFeaturesInfo,
                                              numTrees=numT, featureSubsetStrategy="auto",
                                              impurity='variance', maxDepth=maxD, maxBins=32)
        predictions = rfModel.predict(validationLabPt.map(lambda x: x.features))
```

```

predictionAndLabels = validationLabPt.map(lambda lp: lp.label).zip(predictions)
# Use ROC-AUC as accuracy metrics
validMetrics = RegressionMetrics(predictionAndLabels)
metric = validMetrics.rootMeanSquaredError
metricSum[j] += metric

avgAcc = metricSum/nFolds;
bestParam = paramGrid[np.argmin(avgAcc)];

# UNPERSIST OBJECTS
trainCVLabPt.unpersist()
validationLabPt.unpersist()

## TRAIN FINAL MODEL WITH BEST PARAMETERS
rfModel = RandomForest.trainRegressor(indexedTRAINreg, categoricalFeaturesInfo=categoricalFeaturesInfo,
                                       numTrees=bestParam['numTrees'], featureSubsetStrategy="auto",
                                       impurity='variance', maxDepth=bestParam['maxDepth'], maxBins=32)

# EVALUATE MODEL ON TEST DATA
predictions = rfModel.predict(indexedTESTreg.map(lambda x: x.features))
predictionAndLabels = indexedTESTreg.map(lambda lp: lp.label).zip(predictions)

#PRINT TEST METRICS
testMetrics = RegressionMetrics(predictionAndLabels)
print("RMSE=%s" % testMetrics.rootMeanSquaredError)
print("R-sqr=%s" % testMetrics.r2)

# PRINT ELAPSED TIME
timeend = datetime.datetime.now()
timedelta = round((timeend-timestart).total_seconds(), 2)
print "Time taken to execute above cell: " + str(timedelta) + " seconds";

```

OUTPUT

RMSE = 0.906972198262

R-sqr = 0.740751197012

Time taken to execute above cell: 69.17 seconds

Clean up objects from memory and print model locations

Use `unpersist()` to delete objects cached in memory.

```

# UNPERSIST OBJECTS CACHED IN MEMORY

# REMOVE ORIGINAL DFs
taxi_df_train_cleaned.unpersist()
taxi_df_train_with_newFeatures.unpersist()
trainData.unpersist()
trainData.unpersist()

# FOR BINARY CLASSIFICATION TRAINING AND TESTING
indexedTRAINbinary.unpersist()
indexedTESTbinary.unpersist()
oneHotTRAINbinary.unpersist()
oneHotTESTbinary.unpersist()

# FOR REGRESSION TRAINING AND TESTING
indexedTRAINreg.unpersist()
indexedTESTreg.unpersist()
oneHotTRAINreg.unpersist()
oneHotTESTreg.unpersist()

# SCALED FEATURES
oneHotTRAINregScaled.unpersist()
oneHotTESTregScaled.unpersist()

```

OUTPUT

PythonRDD[122] at RDD at PythonRDD.scala: 43

**Printout path to model files to be used in the consumption notebook. ** To consume and score an independent data-set, you need to copy and paste these file names in the "Consumption notebook".

```

# PRINT MODEL FILE LOCATIONS FOR CONSUMPTION
print "logisticRegFileLoc = modelDir + \\" + logisticregressionfilename + "\\";
print "linearRegFileLoc = modelDir + \\" + linearregressionfilename + "\\";
print "randomForestClassificationFileLoc = modelDir + \\" + rfclassificationfilename + "\\";
print "randomForestRegFileLoc = modelDir + \\" + rfregressionfilename + "\\";
print "BoostedTreeClassificationFileLoc = modelDir + \\" + btclassificationfilename + "\\";
print "BoostedTreeRegressionFileLoc = modelDir + \\" + btregressionfilename + "\\";

```

OUTPUT

```

logisticRegFileLoc = modelDir + "LogisticRegressionWithLBFGS_2016-05-0316_47_30.096528"

linearRegFileLoc = modelDir + "LinearRegressionWithSGD_2016-05-0316_51_28.433670"

randomForestClassificationFileLoc = modelDir + "RandomForestClassification_2016-05-0316_50_17.454440"

randomForestRegFileLoc = modelDir + "RandomForestRegression_2016-05-0316_51_57.331730"

BoostedTreeClassificationFileLoc = modelDir + "GradientBoostingTreeClassification_2016-05-
0316_50_40.138809"

BoostedTreeRegressionFileLoc = modelDir + "GradientBoostingTreeRegression_2016-05-0316_52_18.827237"

```

What's next?

Now that you have created regression and classification models with the Spark MLlib, you are ready to learn how to score and evaluate these models.

Model consumption: To learn how to score and evaluate the classification and regression models created in this topic, see [Score and evaluate Spark-built machine learning models](#).

Data Science using Scala and Spark on Azure

1/17/2017 • 34 min to read • [Edit on GitHub](#)

This article shows you how to use Scala for supervised machine learning tasks with the Spark scalable MLlib and Spark ML packages on an Azure HDInsight Spark cluster. It walks you through the tasks that constitute the [Data Science process](#): data ingestion and exploration, visualization, feature engineering, modeling, and model consumption. The models in the article include logistic and linear regression, random forests, and gradient-boosted trees (GBTs), in addition to two common supervised machine learning tasks:

- Regression problem: Prediction of the tip amount (\$) for a taxi trip
- Binary classification: Prediction of tip or no tip (1/0) for a taxi trip

The modeling process requires training and evaluation on a test data set and relevant accuracy metrics. In this article, you can learn how to store these models in Azure Blob storage and how to score and evaluate their predictive performance. This article also covers the more advanced topics of how to optimize models by using cross-validation and hyper-parameter sweeping. The data used is a sample of the 2013 NYC taxi trip and fare data set available on GitHub.

[Scala](#), a language based on the Java virtual machine, integrates object-oriented and functional language concepts. It's a scalable language that is well suited to distributed processing in the cloud, and runs on Azure Spark clusters.

[Spark](#) is an open-source parallel-processing framework that supports in-memory processing to boost the performance of big data analytics applications. The Spark processing engine is built for speed, ease of use, and sophisticated analytics. Spark's in-memory distributed computation capabilities make it a good choice for iterative algorithms in machine learning and graph computations. The [spark.ml](#) package provides a uniform set of high-level APIs built on top of data frames that can help you create and tune practical machine learning pipelines. [MLlib](#) is Spark's scalable machine learning library, which brings modeling capabilities to this distributed environment.

[HDInsight Spark](#) is the Azure-hosted offering of open-source Spark. It also includes support for Jupyter Scala notebooks on the Spark cluster, and can run Spark SQL interactive queries to transform, filter, and visualize data stored in Azure Blob storage. The Scala code snippets in this article that provide the solutions and show the relevant plots to visualize the data run in Jupyter notebooks installed on the Spark clusters. The modeling steps in these topics have code that shows you how to train, evaluate, save, and consume each type of model.

The setup steps and code in this article are for Azure HDInsight 3.4 Spark 1.6. However, the code in this article and in the [Scala Jupyter Notebook](#) are generic and should work on any Spark cluster. The cluster setup and management steps might be slightly different from what is shown in this article if you are not using HDInsight Spark.

NOTE

For a topic that shows you how to use Python rather than Scala to complete tasks for an end-to-end Data Science process, see [Data Science using Spark on Azure HDInsight](#).

Prerequisites

- You must have an Azure subscription. If you do not already have one, [get an Azure free trial](#).
- You need an Azure HDInsight 3.4 Spark 1.6 cluster to complete the following procedures. To create a cluster, see the instructions in [Get started: Create Apache Spark on Azure HDInsight](#). Set the cluster type and version on the **Select Cluster Type** menu.

New HDInsight Cluster

Cluster Type configuration

Learn about HDInsight and cluster versions. [Learn more](#)

* Cluster Name
Enter new cluster name .azurehdinsight.net

* Subscription
Azure-Irregulars_563702

Select Cluster Type ⓘ Configure required settings

* Credentials
Configure required settings

* Data Source ⓘ
Configure required settings

* Node Pricing Tiers

Pin to dashboard

Cluster Type ⓘ Spark (Preview)

Operating System Linux

Spark 1.5.2 (HDI 3.3)
Spark 1.6.0 (HDI 3.4)

Cluster Tier ([more info](#))

STANDARD	PREMIUM (PREVIEW) ★
Administration Manage, monitor, connect	Administration Manage, monitor, connect
Scalability On-demand node scaling	Scalability On-demand node scaling
99.9% Uptime SLA	99.9% Uptime SLA
Automatic patching	Automatic patching
+ 0.00 USD/CORE/HOUR	+ 0.02 USD/CORE/HOUR

WARNING

HDInsight clusters billing is pro-rated per minute, whether you are using them or not. Please be sure to delete your cluster after you have finished using it. For information on deleting a cluster, see [How to delete an HDInsight cluster](#).

For a description of the NYC taxi trip data and instructions on how to execute code from a Jupyter notebook on the Spark cluster, see the relevant sections in [Overview of Data Science using Spark on Azure HDInsight](#).

Execute Scala code from a Jupyter notebook on the Spark cluster

You can launch a Jupyter notebook from the Azure portal. Find the Spark cluster on your dashboard, and then click it to enter the management page for your cluster. Next, click **Cluster Dashboards**, and then click **Jupyter Notebook** to open the notebook associated with the Spark cluster.

The screenshot shows the Azure HDInsight Cluster blade. In the top navigation bar, there are icons for Settings, Dashboard, Secure Shell, Scale Cluster, and Delete. Below this, the 'Essentials' section displays cluster details: Resource group (redacted), Status (Running), Location (South Central US), Subscription name (redacted), and Subscription ID (redacted). To the right, it shows Cluster Type (Standard Spark on Linux), URL (<https://<clustername>.azurehdinsight.net>), Documentation, Getting Started, and Quickstart. It also lists Head Nodes, Worker Nodes (A3 (x2), D3 (x1)). An 'All settings' button is at the bottom right of this section.

In the 'Quick Links' area, there are three buttons: 'Cluster Dashboards' (circled in red), 'Ambari Views', and 'Scale Cluster'. Below this is a 'Usage' section titled 'Cores in South Central US for subscription' featuring a donut chart and some statistics:

- THIS CLUSTER: 12 cores
- OTHER CLUSTERS: 756 cores
- AVAILABLE: 1332 cores
- TOTAL: 2100 cores

The 'Cluster nodes' section shows a list of nodes: PySpark and Scala (Scala is circled in red).

To the right, under 'Cluster Dashboards', there is a grid of tiles with 'Add tiles' buttons. One tile, 'Jupyter Notebook', is circled in red.

You also can access Jupyter notebooks at <https://<clustername>.azurehdinsight.net/jupyter>. Replace *clustername* with the name of your cluster. You need the password for your administrator account to access the Jupyter notebooks.

The screenshot shows the Jupyter notebook interface. At the top, there are tabs for 'Files', 'Running', and 'Clusters'. Below this is a message: 'Select items to perform actions on them.' A file explorer sidebar shows two items: 'PySpark' and 'Scala' (which is circled in red). In the top right corner, there are buttons for 'Upload', 'New', and a refresh icon (circled in red).

Select **Scala** to see a directory that has a few examples of prepackaged notebooks that use the PySpark API. The Exploration Modeling and Scoring using Scala.ipynb notebook that contains the code samples for this suite of Spark topics is available on [GitHub](#).

You can upload the notebook directly from GitHub to the Jupyter Notebook server on your Spark cluster. On your Jupyter home page, click the **Upload** button. In the file explorer, paste the GitHub (raw content) URL of the Scala notebook, and then click **Open**. The Scala notebook is available at the following URL:

[Exploration-Modeling-and-Scoring-using-Scala.ipynb](#)

Setup: Preset Spark and Hive contexts, Spark magics, and Spark libraries

Preset Spark and Hive contexts

```
# SET THE START TIME
import java.util.Calendar
val beginningTime = Calendar.getInstance().getTime()
```

The Spark kernels that are provided with Jupyter notebooks have preset contexts. You don't need to explicitly set the Spark or Hive contexts before you start working with the application you are developing. The preset contexts are:

- `sc` for `SparkContext`
- `sqlContext` for `HiveContext`

Spark magics

The Spark kernel provides some predefined "magics," which are special commands that you can call with `%%`.

Two of these commands are used in the following code samples.

- `%%local` specifies that the code in subsequent lines will be executed locally. The code must be valid Scala code.
- `%%sql -o <variable name>` executes a Hive query against `sqlContext`. If the `-o` parameter is passed, the result of the query is persisted in the `%%local` Scala context as a Spark data frame.

For more information about the kernels for Jupyter notebooks and their predefined "magics" that you call with `%%` (for example, `%%local`), see [Kernels available for Jupyter notebooks with HDInsight Spark Linux clusters on HDInsight](#).

Import libraries

Import the Spark, MLlib, and other libraries you'll need by using the following code.

```

# IMPORT SPARK AND JAVA LIBRARIES
import org.apache.spark.sql.SQLContext
import org.apache.spark.sql.functions._
import java.text.SimpleDateFormat
import java.util.Calendar
import sqlContext.implicits._
import org.apache.spark.sql.Row

# IMPORT SPARK SQL FUNCTIONS
import org.apache.spark.sql.types.{StructType, StructField, StringType, IntegerType, FloatType, DoubleType}
import org.apache.spark.sql.functions.rand

# IMPORT SPARK ML FUNCTIONS
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.feature.{StringIndexer, VectorAssembler, OneHotEncoder, VectorIndexer, Binarizer}
import org.apache.spark.ml.tuning.{ParamGridBuilder, TrainValidationSplit, CrossValidator}
import org.apache.spark.ml.regression.{LinearRegression, LinearRegressionModel, RandomForestRegressor, RandomForestRegressionModel, GBTRegressor, GBTRegressionModel}
import org.apache.spark.ml.classification.{LogisticRegression, LogisticRegressionModel, RandomForestClassifier, RandomForestClassificationModel, GBTClassifier, GBTClassificationModel}
import org.apache.spark.ml.evaluation.{BinaryClassificationEvaluator, RegressionEvaluator, MulticlassClassificationEvaluator}

# IMPORT SPARK MLLIB FUNCTIONS
import org.apache.spark.mllib.linalg.{Vector, Vectors}
import org.apache.spark.mllib.util.MLUtils
import org.apache.spark.mllib.classification.{LogisticRegressionWithLBFGS, LogisticRegressionModel}
import org.apache.spark.mllib.regression.{LabeledPoint, LinearRegressionWithSGD, LinearRegressionModel}
import org.apache.spark.mllib.tree.{GradientBoostedTrees, RandomForest}
import org.apache.spark.mllib.tree.configuration.BoostingStrategy
import org.apache.spark.mllib.tree.model.{GradientBoostedTreesModel, RandomForestModel, Predict}
import org.apache.spark.mllib.evaluation.{BinaryClassificationMetrics, MulticlassMetrics, RegressionMetrics}

# SPECIFY SQLCONTEXT
val sqlContext = new SQLContext(sc)

```

Data ingestion

The first step in the Data Science process is to ingest the data that you want to analyze. You bring the data from external sources or systems where it resides into your data exploration and modeling environment. In this article, the data you ingest is a joined 0.1% sample of the taxi trip and fare file (stored as a .tsv file). The data exploration and modeling environment is Spark. This section contains the code to complete the following series of tasks:

1. Set directory paths for data and model storage.
2. Read in the input data set (stored as a .tsv file).
3. Define a schema for the data and clean the data.
4. Create a cleaned data frame and cache it in memory.
5. Register the data as a temporary table in SQLContext.
6. Query the table and import the results into a data frame.

Set directory paths for storage locations in Azure Blob storage

Spark can read and write to Azure Blob storage. You can use Spark to process any of your existing data, and then store the results again in Blob storage.

To save models or files in Blob storage, you need to properly specify the path. Reference the default container attached to the Spark cluster by using a path that begins with `wasb://`. Reference other locations by using `wasb://`.

The following code sample specifies the location of the input data to be read and the path to Blob storage that is attached to the Spark cluster where the model will be saved.

```

# SET PATHS TO DATA AND MODEL FILE LOCATIONS
# INGEST DATA AND SPECIFY HEADERS FOR COLUMNS
val taxi_train_file =
sc.textFile("wasb://mllibwalkthroughs@cdspspksamples.blob.core.windows.net/Data/NYCTaxi/JoinedTaxiTripFare.Point1Pct.Train.tsv")
val header = taxi_train_file.first;

# SET THE MODEL STORAGE DIRECTORY PATH
# NOTE THAT THE FINAL BACKSLASH IN THE PATH IS REQUIRED.
val modelDir = "wasb:///user/remoteuser/NYCTaxi/Models/";

```

Import data, create an RDD, and define a data frame according to the schema

```

# RECORD THE START TIME
val starttime = Calendar.getInstance().getTime()

# DEFINE THE SCHEMA BASED ON THE HEADER OF THE FILE
val sqlContext = new SQLContext(sc)
val taxi_schema = StructType(
  Array(
    StructField("medallion", StringType, true),
    StructField("hack_license", StringType, true),
    StructField("vendor_id", StringType, true),
    StructField("rate_code", DoubleType, true),
    StructField("store_and_fwd_flag", StringType, true),
    StructField("pickup_datetime", StringType, true),
    StructField("dropoff_datetime", StringType, true),
    StructField("pickup_hour", DoubleType, true),
    StructField("pickup_week", DoubleType, true),
    StructField("weekday", DoubleType, true),
    StructField("passenger_count", DoubleType, true),
    StructField("trip_time_in_secs", DoubleType, true),
    StructField("trip_distance", DoubleType, true),
    StructField("pickup_longitude", DoubleType, true),
    StructField("pickup_latitude", DoubleType, true),
    StructField("dropoff_longitude", DoubleType, true),
    StructField("dropoff_latitude", DoubleType, true),
    StructField("direct_distance", StringType, true),
    StructField("payment_type", StringType, true),
    StructField("fare_amount", DoubleType, true),
    StructField("surcharge", DoubleType, true),
    StructField("mta_tax", DoubleType, true),
    StructField("tip_amount", DoubleType, true),
    StructField("tolls_amount", DoubleType, true),
    StructField("total_amount", DoubleType, true),
    StructField("tipped", DoubleType, true),
    StructField("tip_class", DoubleType, true)
  )
)

# CAST VARIABLES ACCORDING TO THE SCHEMA
val taxi_temp = (taxi_train_file.map(_.split("\t")))
  .filter(r => r(0) != "medallion")
  .map(p => Row(p(0), p(1), p(2),
    p(3).toDouble, p(4), p(5), p(6), p(7).toDouble, p(8).toDouble, p(9).toDouble, p(10).toDouble,
    p(11).toDouble, p(12).toDouble, p(13).toDouble, p(14).toDouble, p(15).toDouble, p(16).toDouble,
    p(17), p(18), p(19).toDouble, p(20).toDouble, p(21).toDouble, p(22).toDouble,
    p(23).toDouble, p(24).toDouble, p(25).toDouble, p(26).toDouble)))

# CREATE AN INITIAL DATA FRAME AND DROP COLUMNS, AND THEN CREATE A CLEANED DATA FRAME BY FILTERING FOR
UNWANTED VALUES OR OUTLIERS
val taxi_train_df = sqlContext.createDataFrame(taxi_temp, taxi_schema)

val taxi_df_train_cleaned = (taxi_train_df.drop(taxi_train_df.col("medallion")))
  .drop(taxi_train_df.col("hack_license")).drop(taxi_train_df.col("store_and_fwd_flag"))
  .drop(taxi_train_df.col("pickup_datetime")).drop(taxi_train_df.col("dropoff_datetime"))
  .drop(taxi_train_df.col("pickup_longitude")).drop(taxi_train_df.col("pickup_latitude"))

```

```

.drop(taxi_train_df.col("dropoff_longitude")).drop(taxi_train_df.col("dropoff_latitude"))
.drop(taxi_train_df.col("surcharge")).drop(taxi_train_df.col("mta_tax"))
.drop(taxi_train_df.col("direct_distance")).drop(taxi_train_df.col("tolls_amount"))
.drop(taxi_train_df.col("total_amount")).drop(taxi_train_df.col("tip_class"))
.filter("passenger_count > 0 and passenger_count < 8 AND payment_type in ('CSH', 'CRD') AND tip_amount >= 0 AND tip_amount < 30 AND
fare_amount >= 1 AND fare_amount < 150 AND trip_distance > 0 AND trip_distance < 100 AND trip_time_in_secs > 30 AND trip_time_in_secs <
7200");

# CACHE AND MATERIALIZE THE CLEANED DATA FRAME IN MEMORY
taxi_df_train_cleaned.cache()
taxi_df_train_cleaned.count()

# REGISTER THE DATA FRAME AS A TEMPORARY TABLE IN SQLCONTEXT
taxi_df_train_cleaned.registerTempTable("taxi_train")

# GET THE TIME TO RUN THE CELL
val endtime = Calendar.getInstance().getTime()
val elapsedtime = ((endtime.getTime() - starttime.getTime()) / 1000).toString;
println("Time taken to run the above cell: " + elapsedtime + " seconds.");

```

Output:

Time to run the cell: 8 seconds.

Query the table and import results in a data frame

Next, query the table for fare, passenger, and tip data; filter out corrupt and outlying data; and print several rows.

```

# QUERY THE DATA
val sqlStatement = """
    SELECT fare_amount, passenger_count, tip_amount, tipped
    FROM taxi_train
    WHERE passenger_count > 0 AND passenger_count < 7
        AND fare_amount > 0 AND fare_amount < 200
        AND payment_type in ('CSH', 'CRD')
        AND tip_amount > 0 AND tip_amount < 25
"""
val sqlResultsDF = sqlContext.sql(sqlStatement)

# SHOW ONLY THE TOP THREE ROWS
sqlResultsDF.show(3)

```

Output:

FARE_AMOUNT	PASSENGER_COUNT	TIP_AMOUNT	TIPPED
13.5	1.0	2.9	1.0
16.0	2.0	3.4	1.0
10.5	2.0	1.0	1.0

Data exploration and visualization

After you bring the data into Spark, the next step in the Data Science process is to gain a deeper understanding of the data through exploration and visualization. In this section, you examine the taxi data by using SQL queries.

Then, import the results into a data frame to plot the target variables and prospective features for visual inspection by using the auto-visualization feature of Jupyter.

Use local and SQL magic to plot data

By default, the output of any code snippet that you run from a Jupyter notebook is available within the context of

the session that is persisted on the worker nodes. If you want to save a trip to the worker nodes for every computation, and if all the data that you need for your computation is available locally on the Jupyter server node (which is the head node), you can use the `%%local` magic to run the code snippet on the Jupyter server.

- **SQL magic (`%%sql`)**. The HDInsight Spark kernel supports easy inline HiveQL queries against SQLContext. The `-o VARIABLE_NAME` argument persists the output of the SQL query as a Pandas data frame on the Jupyter server. This means it'll be available in the local mode.
- **%%local magic**. The `%%local` magic runs the code locally on the Jupyter server, which is the head node of the HDInsight cluster. Typically, you use `%%local` magic in conjunction with the `%%sql` magic with the `-o` parameter. The `-o` parameter would persist the output of the SQL query locally, and then `%%local` magic would trigger the next set of code snippet to run locally against the output of the SQL queries that is persisted locally.

Query the data by using SQL

This query retrieves the taxi trips by fare amount, passenger count, and tip amount.

```
# RUN THE SQL QUERY
%%sql -q -o sqlResults
SELECT fare_amount, passenger_count, tip_amount, tipped FROM taxi_train WHERE passenger_count > 0 AND passenger_count < 7 AND
fare_amount > 0 AND fare_amount < 200 AND payment_type in ('CSH', 'CRD') AND tip_amount > 0 AND tip_amount < 25
```

In the following code, the `%%local` magic creates a local data frame, `sqlResults`. You can use `sqlResults` to plot by using `matplotlib`.

TIP

Local magic is used multiple times in this article. If your data set is large, please sample to create a data frame that can fit in local memory.

Plot the data

You can plot by using Python code after the data frame is in local context as a Pandas data frame.

```
# RUN THE CODE LOCALLY ON THE JUPYTER SERVER
%%local

# USE THE JUPYTER AUTO-PLOTTING FEATURE TO CREATE INTERACTIVE FIGURES.
# CLICK THE TYPE OF PLOT TO GENERATE (LINE, AREA, BAR, ETC.)
sqlResults
```

The Spark kernel automatically visualizes the output of SQL (HiveQL) queries after you run the code. You can choose between several types of visualizations:

- Table
- Pie
- Line
- Area
- Bar

Here's the code to plot the data:

```

# RUN THE CODE LOCALLY ON THE JUPYTER SERVER AND IMPORT LIBRARIES
%%local
import matplotlib.pyplot as plt
%matplotlib inline

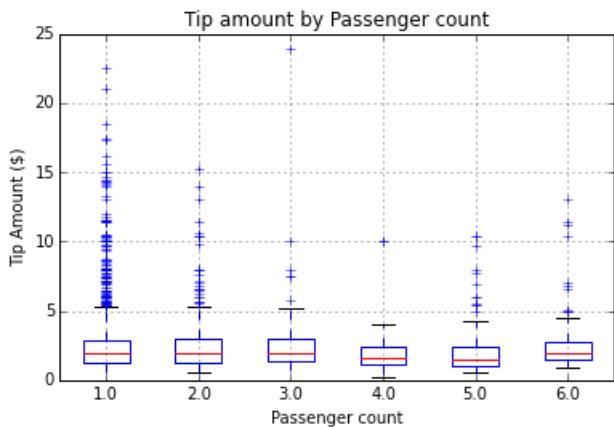
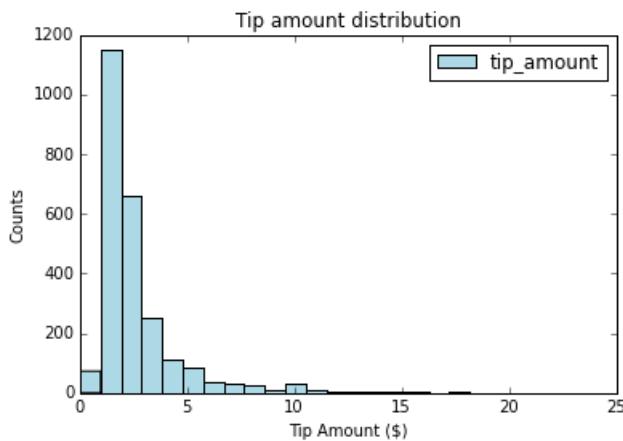
# PLOT TIP BY PAYMENT TYPE AND PASSENGER COUNT
ax1 = sqlResults[['tip_amount']].plot(kind='hist', bins=25, facecolor='lightblue')
ax1.set_title('Tip amount distribution')
ax1.set_xlabel('Tip Amount ($)')
ax1.set_ylabel('Counts')
plt.suptitle("")
plt.show()

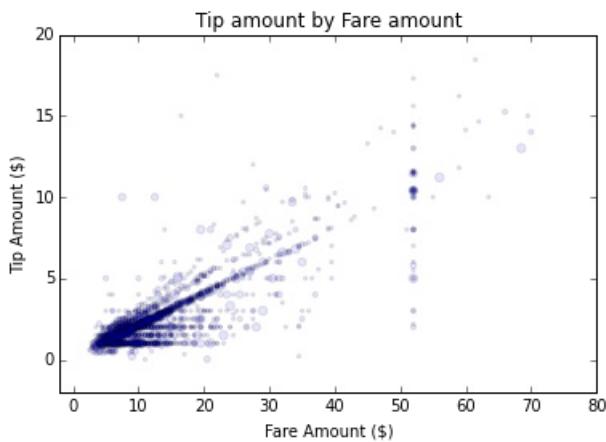
# PLOT TIP BY PASSENGER COUNT
ax2 = sqlResults.boxplot(column=['tip_amount'], by=['passenger_count'])
ax2.set_title('Tip amount by Passenger count')
ax2.set_xlabel('Passenger count')
ax2.set_ylabel('Tip Amount ($)')
plt.suptitle("")
plt.show()

# PLOT TIP AMOUNT BY FARE AMOUNT; SCALE POINTS BY PASSENGER COUNT
ax = sqlResults.plot(kind='scatter', x='fare_amount', y='tip_amount', c='blue', alpha = 0.10, s=5*(sqlResults.passenger_count))
ax.set_title('Tip amount by Fare amount')
ax.set_xlabel('Fare Amount ($)')
ax.set_ylabel('Tip Amount ($)')
plt.axis([-2, 80, -2, 20])
plt.show()

```

Output:





Create features and transform features, and then prep data for input into modeling functions

For tree-based modeling functions from Spark ML and MLLib, you have to prepare target and features by using a variety of techniques, such as binning, indexing, one-hot encoding, and vectorization. Here are the procedures to follow in this section:

1. Create a new feature by **binning** hours into traffic time buckets.
2. Apply **indexing and one-hot encoding** to categorical features.
3. **Sample and split the data set** into training and test fractions.
4. **Specify training variable and features**, and then create indexed or one-hot encoded training and testing input labeled point resilient distributed datasets (RDDs) or data frames.
5. Automatically **categorize and vectorize features and targets** to use as inputs for machine learning models.

Create a new feature by binning hours into traffic time buckets

This code shows you how to create a new feature by binning hours into traffic time buckets and how to cache the resulting data frame in memory. Where RDDs and data frames are used repeatedly, caching leads to improved execution times. Accordingly, you'll cache RDDs and data frames at several stages in the following procedures.

```
# CREATE FOUR BUCKETS FOR TRAFFIC TIMES
val sqlStatement = """
    SELECT *,
    CASE
        WHEN (pickup_hour <= 6 OR pickup_hour >= 20) THEN "Night"
        WHEN (pickup_hour >= 7 AND pickup_hour <= 10) THEN "AMRush"
        WHEN (pickup_hour >= 11 AND pickup_hour <= 15) THEN "Afternoon"
        WHEN (pickup_hour >= 16 AND pickup_hour <= 19) THEN "PMRush"
    END as TrafficTimeBins
    FROM taxi_train
"""
val taxi_df_train_with_newFeatures = sqlContext.sql(sqlStatement)

# CACHE THE DATA FRAME IN MEMORY AND MATERIALIZE THE DATA FRAME IN MEMORY
taxi_df_train_with_newFeatures.cache()
taxi_df_train_with_newFeatures.count()
```

Indexing and one-hot encoding of categorical features

The modeling and predict functions of MLLib require features with categorical input data to be indexed or encoded prior to use. This section shows you how to index or encode categorical features for input into the modeling functions.

You need to index or encode your models in different ways, depending on the model. For example, logistic and linear regression models require one-hot encoding. For example, a feature with three categories can be expanded

into three feature columns. Each column would contain 0 or 1 depending on the category of an observation. MLlib provides the [OneHotEncoder](#) function for one-hot encoding. This encoder maps a column of label indices to a column of binary vectors with at most a single one-value. With this encoding, algorithms that expect numerical valued features, such as logistic regression, can be applied to categorical features.

Here you transform only four variables to show examples, which are character strings. You also can index other variables, such as weekday, represented by numerical values, as categorical variables.

For indexing, use `StringIndexer()`, and for one-hot encoding, use `OneHotEncoder()` functions from MLlib. Here is the code to index and encode categorical features:

```
# CREATE INDEXES AND ONE-HOT ENCODED VECTORS FOR SEVERAL CATEGORICAL FEATURES

# RECORD THE START TIME
val starttime = Calendar.getInstance().getTime()

# INDEX AND ENCODE VENDOR_ID
val stringIndexer = new StringIndexer().setInputCol("vendor_id").setOutputCol("vendorIndex").fit(taxi_df_train_with_newFeatures)
val indexed = stringIndexer.transform(taxi_df_train_with_newFeatures)
val encoder = new OneHotEncoder().setInputCol("vendorIndex").setOutputCol("vendorVec")
val encoded1 = encoder.transform(indexed)

# INDEX AND ENCODE RATE_CODE
val stringIndexer = new StringIndexer().setInputCol("rate_code").setOutputCol("rateIndex").fit(encoded1)
val indexed = stringIndexer.transform(encoded1)
val encoder = new OneHotEncoder().setInputCol("rateIndex").setOutputCol("rateVec")
val encoded2 = encoder.transform(indexed)

# INDEX AND ENCODE PAYMENT_TYPE
val stringIndexer = new StringIndexer().setInputCol("payment_type").setOutputCol("paymentIndex").fit(encoded2)
val indexed = stringIndexer.transform(encoded2)
val encoder = new OneHotEncoder().setInputCol("paymentIndex").setOutputCol("paymentVec")
val encoded3 = encoder.transform(indexed)

# INDEX AND TRAFFIC TIME BINS
val stringIndexer = new StringIndexer().setInputCol("TrafficTimeBins").setOutputCol("TrafficTimeBinsIndex").fit(encoded3)
val indexed = stringIndexer.transform(encoded3)
val encoder = new OneHotEncoder().setInputCol("TrafficTimeBinsIndex").setOutputCol("TrafficTimeBinsVec")
val encodedFinal = encoder.transform(indexed)

# GET THE TIME TO RUN THE CELL
val endtime = Calendar.getInstance().getTime()
val elapsedtime = ((endtime.getTime() - starttime.getTime()) / 1000).toString;
println("Time taken to run the above cell: " + elapsedtime + " seconds.");
```

Output:

Time to run the cell: 4 seconds.

Sample and split the data set into training and test fractions

This code creates a random sampling of the data (25%, in this example). Although sampling is not required for this example due to the size of the data set, the article shows you how you can sample so that you know how to use it for your own problems when needed. When samples are large, this can save significant time while you train models. Next, split the sample into a training part (75%, in this example) and a testing part (25%, in this example) to use in classification and regression modeling.

Add a random number (between 0 and 1) to each row (in a "rand" column) that can be used to select cross-validation folds during training.

```

# RECORD THE START TIME
val starttime = Calendar.getInstance().getTime()

# SPECIFY SAMPLING AND SPLITTING FRACTIONS
val samplingFraction = 0.25;
val trainingFraction = 0.75;
val testingFraction = (1-trainingFraction);
val seed = 1234;
val encodedFinalSampledTmp = encodedFinal.sample(withReplacement = false, fraction = samplingFraction, seed = seed)
val sampledDFcount = encodedFinalSampledTmp.count().toInt

val generateRandomDouble = udf(() => {
    scala.util.Random.nextDouble
})

# ADD A RANDOM NUMBER FOR CROSS-VALIDATION
val encodedFinalSampled = encodedFinalSampledTmp.withColumn("rand", generateRandomDouble());

# SPLIT THE SAMPLED DATA FRAME INTO TRAIN AND TEST, WITH A RANDOM COLUMN ADDED FOR DOING CROSS-VALIDATION
# (SHOWN LATER)
# INCLUDE A RANDOM COLUMN FOR CREATING CROSS-VALIDATION FOLDS
val splits = encodedFinalSampled.randomSplit(Array(trainingFraction, testingFraction), seed = seed)
val trainData = splits(0)
val testData = splits(1)

# GET THE TIME TO RUN THE CELL
val endtime = Calendar.getInstance().getTime()
val elapsedtime = ((endtime.getTime() - starttime.getTime()) / 1000).toString;
println("Time taken to run the above cell: " + elapsedtime + " seconds.");

```

Output:

Time to run the cell: 2 seconds.

Specify training variable and features, and then create indexed or one-hot encoded training and testing input labeled point RDDs or data frames

This section contains code that shows you how to index categorical text data as a labeled point data type, and encode it so you can use it to train and test MLlib logistic regression and other classification models. Labeled point objects are RDDs that are formatted in a way that is needed as input data by most of machine learning algorithms in MLlib. A [labeled point](#) is a local vector, either dense or sparse, associated with a label/response.

In this code, you specify the target (dependent) variable and the features to use to train models. Then, you create indexed or one-hot encoded training and testing input labeled point RDDs or data frames.

```

# RECORD THE START TIME
val starttime = Calendar.getInstance().getTime()

# MAP NAMES OF FEATURES AND TARGETS FOR CLASSIFICATION AND REGRESSION PROBLEMS
val featuresIndOneHot = List("paymentVec", "vendorVec", "rateVec", "TrafficTimeBinsVec", "pickup_hour", "weekday", "passenger_count",
"trip_time_in_secs", "trip_distance", "fare_amount").map(encodedFinalSampled.columns.indexOf(_))
val featuresIndIndex = List("paymentIndex", "vendorIndex", "rateIndex", "TrafficTimeBinsIndex", "pickup_hour", "weekday", "passenger_count",
"trip_time_in_secs", "trip_distance", "fare_amount").map(encodedFinalSampled.columns.indexOf(_))

# SPECIFY THE TARGET FOR CLASSIFICATION ('tipped') AND REGRESSION ('tip_amount') PROBLEMS
val targetIndBinary = List("tipped").map(encodedFinalSampled.columns.indexOf(_))
val targetIndRegression = List("tip_amount").map(encodedFinalSampled.columns.indexOf(_))

# CREATE INDEXED LABELED POINT RDD OBJECTS
val indexedTRAINbinary = trainData.rdd.map(r => LabeledPoint(r.getDouble(targetIndBinary(0).toInt),
Vectors.dense(featuresIndIndex.map(r.getDouble(_).toArray))))
val indexedTESTbinary = testData.rdd.map(r => LabeledPoint(r.getDouble(targetIndBinary(0).toInt),
Vectors.dense(featuresIndIndex.map(r.getDouble(_).toArray))))
val indexedTRAINreg = trainData.rdd.map(r => LabeledPoint(r.getDouble(targetIndRegression(0).toInt),
Vectors.dense(featuresIndIndex.map(r.getDouble(_).toArray))))
val indexedTESTreg = testData.rdd.map(r => LabeledPoint(r.getDouble(targetIndRegression(0).toInt),
Vectors.dense(featuresIndIndex.map(r.getDouble(_).toArray))))

# CREATE INDEXED DATA FRAMES THAT YOU CAN USE TO TRAIN BY USING SPARK ML FUNCTIONS
val indexedTRAINbinaryDF = indexedTRAINbinary.toDF()
val indexedTESTbinaryDF = indexedTESTbinary.toDF()
val indexedTRAINregDF = indexedTRAINreg.toDF()
val indexedTESTregDF = indexedTESTreg.toDF()

# CREATE ONE-HOT ENCODED (VECTORIZED) DATA FRAMES THAT YOU CAN USE TO TRAIN BY USING SPARK ML FUNCTIONS
val assemblerOneHot = new VectorAssembler().setInputCols(Array("paymentVec", "vendorVec", "rateVec", "TrafficTimeBinsVec", "pickup_hour",
"weekday", "passenger_count", "trip_time_in_secs", "trip_distance", "fare_amount")).setOutputCol("features")
val OneHotTRAIN = assemblerOneHot.transform(trainData)
val OneHotTEST = assemblerOneHot.transform(testData)

# GET THE TIME TO RUN THE CELL
val endtime = Calendar.getInstance().getTime()
val elapsedtime = ((endtime.getTime() - starttime.getTime()) / 1000).toString;
println("Time taken to run the above cell: " + elapsedtime + " seconds.");

```

Output:

Time to run the cell: 4 seconds.

Automatically categorize and vectorize features and targets to use as inputs for machine learning models

Use Spark ML to categorize the target and features to use in tree-based modeling functions. The code completes two tasks:

- Creates a binary target for classification by assigning a value of 0 or 1 to each data point between 0 and 1 by using a threshold value of 0.5.
- Automatically categorizes features. If the number of distinct numerical values for any feature is less than 32, that feature is categorized.

Here's the code for these two tasks.

```

# CATEGORIZE FEATURES AND BINARIZE THE TARGET FOR THE BINARY CLASSIFICATION PROBLEM

# TRAIN DATA
val indexer = new VectorIndexer().setInputCol("features").setOutputCol("featuresCat").setMaxCategories(32)
val indexerModel = indexer.fit(indexedTRAINbinaryDF)
val indexedTrainwithCatFeat = indexerModel.transform(indexedTRAINbinaryDF)
val binarizer: Binarizer = new Binarizer().setInputCol("label").setOutputCol("labelBin").setThreshold(0.5)
val indexedTRAINwithCatFeatBinTarget = binarizer.transform(indexedTrainwithCatFeat)

# TEST DATA
val indexerModel = indexer.fit(indexedTESTbinaryDF)
val indexedTrainwithCatFeat = indexerModel.transform(indexedTESTbinaryDF)
val binarizer: Binarizer = new Binarizer().setInputCol("label").setOutputCol("labelBin").setThreshold(0.5)
val indexedTESTwithCatFeatBinTarget = binarizer.transform(indexedTrainwithCatFeat)

# CATEGORIZE FEATURES FOR THE REGRESSION PROBLEM
# CREATE PROPERLY INDEXED AND CATEGORIZED DATA FRAMES FOR TREE-BASED MODELS

# TRAIN DATA
val indexer = new VectorIndexer().setInputCol("features").setOutputCol("featuresCat").setMaxCategories(32)
val indexerModel = indexer.fit(indexedTRAINregDF)
val indexedTRAINwithCatFeat = indexerModel.transform(indexedTRAINregDF)

# TEST DATA
val indexerModel = indexer.fit(indexedTESTbinaryDF)
val indexedTESTwithCatFeat = indexerModel.transform(indexedTESTregDF)

```

Binary classification model: Predict whether a tip should be paid

In this section, you create three types of binary classification models to predict whether or not a tip should be paid:

- A **logistic regression model** by using the Spark ML `LogisticRegression()` function
- A **random forest classification model** by using the Spark ML `RandomForestClassifier()` function
- A **gradient boosting tree classification model** by using the MLlib `GradientBoostedTrees()` function

Create a logistic regression model

Next, create a logistic regression model by using the Spark ML `LogisticRegression()` function. You create the model building code in a series of steps:

1. **Train the model** data with one parameter set.
2. **Evaluate the model** on a test data set with metrics.
3. **Save the model** in Blob storage for future consumption.
4. **Score the model** against test data.
5. **Plot the results** with receiver operating characteristic (ROC) curves.

Here's the code for these procedures:

```

# CREATE A LOGISTIC REGRESSION MODEL
val lr = new LogisticRegression().setLabelCol("tipped").setFeaturesCol("features").setMaxIter(10).setRegParam(0.3).setElasticNetParam(0.8)
val lrModel = lr.fit(OneHotTRAIN)

# PREDICT ON THE TEST DATA SET
val predictions = lrModel.transform(OneHotTEST)

# SELECT `BinaryClassificationEvaluator()` TO COMPUTE THE TEST ERROR
val evaluator = new BinaryClassificationEvaluator().setLabelCol("tipped").setRawPredictionCol("probability").setMetricName("areaUnderROC")
val ROC = evaluator.evaluate(predictions)
println("ROC on test data = " + ROC)

# SAVE THE MODEL
val timestamp = Calendar.getInstance().getTime().toString.replaceAll(" ", ".").replaceAll(":", "_");
val modelName = "LogisticRegression_"
val filename = modelDir.concat(modelName).concat(timestamp)
lrModel.save(filename);

```

Load, score, and save the results.

```

# RECORD THE START TIME
val starttime = Calendar.getInstance().getTime()

# LOAD THE SAVED MODEL AND SCORE THE TEST DATA SET
val savedModel = org.apache.spark.ml.classification.LogisticRegressionModel.load(filename)
println(s"Coefficients: ${savedModel.coefficients} Intercept: ${savedModel.intercept}")

# SCORE THE MODEL ON THE TEST DATA
val predictions = savedModel.transform(OneHotTEST).select("tipped", "probability", "rawPrediction")
predictions.registerTempTable("testResults")

# SELECT `BinaryClassificationEvaluator()` TO COMPUTE THE TEST ERROR
val evaluator = new BinaryClassificationEvaluator().setLabelCol("tipped").setRawPredictionCol("probability").setMetricName("areaUnderROC")
val ROC = evaluator.evaluate(predictions)

# GET THE TIME TO RUN THE CELL
val endtime = Calendar.getInstance().getTime()
val elapsedtime = ((endtime.getTime() - starttime.getTime()) / 1000).toString;
println("Time taken to run the above cell: " + elapsedtime + " seconds.")

# PRINT THE ROC RESULTS
println("ROC on test data = " + ROC)

```

Output:

ROC on test data = 0.9827381497557599

Use Python on local Pandas data frames to plot the ROC curve.

```

# QUERY THE RESULTS
%%sql -q -o sqlResults
SELECT tipped, probability from testResults

# RUN THE CODE LOCALLY ON THE JUPYTER SERVER AND IMPORT LIBRARIES
%%local
%matplotlib inline
from sklearn.metrics import roc_curve,auc

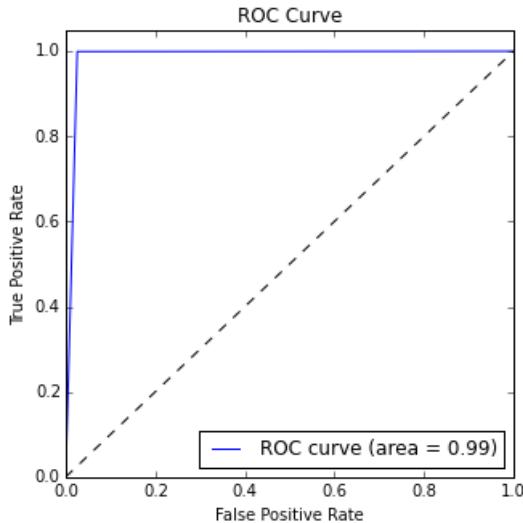
sqlResults['probFloat'] = sqlResults.apply(lambda row: row['probability'].values()[0][1], axis=1)
predictions_pddf = sqlResults[['tipped','probFloat']]

# PREDICT THE ROC CURVE
# predictions_pddf = sqlResults.rename(columns={'_1': 'probability', 'tipped': 'label'})
prob = predictions_pddf['probFloat']
fpr, tpr, thresholds = roc_curve(predictions_pddf['tipped'], prob, pos_label=1);
roc_auc = auc(fpr, tpr)

# PLOT THE ROC CURVE
plt.figure(figsize=(5,5))
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()

```

Output:



Create a random forest classification model

Next, create a random forest classification model by using the Spark ML `RandomForestClassifier()` function, and then evaluate the model on test data.

```

# RECORD THE START TIME
val starttime = Calendar.getInstance().getTime()

# CREATE THE RANDOM FOREST CLASSIFIER MODEL
val rf = new RandomForestClassifier().setLabelCol("labelBin").setFeaturesCol("featuresCat").setNumTrees(10).setSeed(1234)

# FIT THE MODEL
val rfModel = rf.fit(indexedTRAINwithCatFeatBinTarget)
val predictions = rfModel.transform(indexedTESTwithCatFeatBinTarget)

# EVALUATE THE MODEL
val evaluator = new MulticlassClassificationEvaluator().setLabelCol("label").setPredictionCol("prediction").setMetricName("f1")
val Test_f1Score = evaluator.evaluate(predictions)
println("F1 score on test data: " + Test_f1Score);

# GET THE TIME TO RUN THE CELL
val endtime = Calendar.getInstance().getTime()
val elapsedtime = ((endtime.getTime() - starttime.getTime()) / 1000).toString;
println("Time taken to run the above cell: " + elapsedtime + " seconds.");

# CALCULATE BINARY CLASSIFICATION EVALUATION METRICS
val evaluator = new BinaryClassificationEvaluator().setLabelCol("label").setRawPredictionCol("probability").setMetricName("areaUnderROC")
val ROC = evaluator.evaluate(predictions)
println("ROC on test data = " + ROC)

```

Output:

ROC on test data = 0.9847103571552683

Create a GBT classification model

Next, create a GBT classification model by using MLLib's `GradientBoostedTrees()` function, and then evaluate the model on test data.

```

# TRAIN A GBT CLASSIFICATION MODEL BY USING MLLIB AND A LABELED POINT

# RECORD THE START TIME
val starttime = Calendar.getInstance().getTime()

# DEFINE THE GBT CLASSIFICATION MODEL
val boostingStrategy = BoostingStrategy.defaultParams("Classification")
boostingStrategy.numIterations = 20
boostingStrategy.treeStrategy.numClasses = 2
boostingStrategy.treeStrategy.maxDepth = 5
boostingStrategy.treeStrategy.categoricalFeaturesInfo = Map[Int, Int]((0,2),(1,2),(2,6),(3,4))

# TRAIN THE MODEL
val gbtModel = GradientBoostedTrees.train(indexedTRAINbinary, boostingStrategy)

# SAVE THE MODEL IN BLOB STORAGE
val timestamp = Calendar.getInstance().getTime().toString.replaceAll(" ", ".").replaceAll(":", "_");
val modelName = "GBT_Classification_"
val filename = modelDir.concat(modelName).concat(timestamp)
gbtModel.save(sc, filename);

# EVALUATE THE MODEL ON TEST INSTANCES AND THE COMPUTE TEST ERROR
val labelAndPreds = indexedTESTbinary.map { point =>
    val prediction = gbtModel.predict(point.features)
    (point.label, prediction)
}
val testErr = labelAndPreds.filter(r => r._1 != r._2).count.toDouble / indexedTRAINbinary.count()
//println("Learned classification GBT model:\n" + gbtModel.toDebugString)
println("Test Error = " + testErr)

# USE BINARY AND MULTICLASS METRICS TO EVALUATE THE MODEL ON THE TEST DATA
val metrics = new MulticlassMetrics(labelAndPreds)
println(s"Precision: ${metrics.precision}")
println(s"Recall: ${metrics.recall}")
println(s"F1 Score: ${metrics.fMeasure}")

val metrics = new BinaryClassificationMetrics(labelAndPreds)
println(s"Area under PR curve: ${metrics.areaUnderPR}")
println(s"Area under ROC curve: ${metrics.areaUnderROC}")

# GET THE TIME TO RUN THE CELL
val endtime = Calendar.getInstance().getTime()
val elapsedtime = ((endtime.getTime() - starttime.getTime()) / 1000).toString;
println("Time taken to run the above cell: " + elapsedtime + " seconds.");

# PRINT THE ROC METRIC
println(s"Area under ROC curve: ${metrics.areaUnderROC}")

```

Output:

Area under ROC curve: 0.9846895479241554

Regression model: Predict tip amount

In this section, you create two types of regression models to predict the tip amount:

- A **regularized linear regression model** by using the Spark ML `LinearRegression()` function. You'll save the model and evaluate the model on test data.
- A **gradient-boosting tree regression model** by using the Spark ML `(GBT)Regressor()` function.

Create a regularized linear regression model

```

# RECORD THE START TIME
val starttime = Calendar.getInstance().getTime()

# CREATE A REGULARIZED LINEAR REGRESSION MODEL BY USING THE SPARK ML FUNCTION AND DATA FRAMES
val lr = new LinearRegression().setLabelCol("tip_amount").setFeaturesCol("features").setMaxIter(10).setRegParam(0.3).setElasticNetParam(0.8)

# FIT THE MODEL BY USING DATA FRAMES
val lrModel = lr.fit(OneHotTRAIN)
println(s"Coefficients: ${lrModel.coefficients} Intercept: ${lrModel.intercept}")

# SUMMARIZE THE MODEL OVER THE TRAINING SET AND PRINT METRICS
val trainingSummary = lrModel.summary
println(s"numIterations: ${trainingSummary.totalIterations}")
println(s"objectiveHistory: ${trainingSummary.objectiveHistory.toList}")
trainingSummary.residuals.show()
println(s"RMSE: ${trainingSummary.rootMeanSquaredError}")
println(s"r2: ${trainingSummary.r2}")

# SAVE THE MODEL IN AZURE BLOB STORAGE
val datestamp = Calendar.getInstance().getTime().toString.replaceAll(" ", ".").replaceAll(":", "_");
val modelName = "LinearRegression_"
val filename = modelDir.concat(modelName).concat(datestamp)
lrModel.save(filename);

# PRINT THE COEFFICIENTS
println(s"Coefficients: ${lrModel.coefficients} Intercept: ${lrModel.intercept}")

# SCORE THE MODEL ON TEST DATA
val predictions = lrModel.transform(OneHotTEST)

# EVALUATE THE MODEL ON TEST DATA
val evaluator = new RegressionEvaluator().setLabelCol("tip_amount").setPredictionCol("prediction").setMetricName("r2")
val r2 = evaluator.evaluate(predictions)
println("R-sqr on test data = " + r2)

# GET THE TIME TO RUN THE CELL
val endtime = Calendar.getInstance().getTime()
val elapsedtime = ((endtime.getTime() - starttime.getTime()) / 1000).toString;
println("Time taken to run the above cell: " + elapsedtime + " seconds.");

```

Output:

Time to run the cell: 13 seconds.

```

# LOAD A SAVED LINEAR REGRESSION MODEL FROM BLOB STORAGE AND SCORE A TEST DATA SET

# RECORD THE START TIME
val starttime = Calendar.getInstance().getTime()

# LOAD A SAVED LINEAR REGRESSION MODEL FROM AZURE BLOB STORAGE
val savedModel = org.apache.spark.ml.regression.LinearRegressionModel.load(filename)
println(s"Coefficients: ${savedModel.coefficients} Intercept: ${savedModel.intercept}")

# SCORE THE MODEL ON TEST DATA
val predictions = savedModel.transform(OneHotTEST).select("tip_amount", "prediction")
predictions.registerTempTable("testResults")

# EVALUATE THE MODEL ON TEST DATA
val evaluator = new RegressionEvaluator().setLabelCol("tip_amount").setPredictionCol("prediction").setMetricName("r2")
val r2 = evaluator.evaluate(predictions)
println("R-sqr on test data = " + r2)

# GET THE TIME TO RUN THE CELL
val endtime = Calendar.getInstance().getTime()
val elapsedtime = ((endtime.getTime() - starttime.getTime()) / 1000).toString;
println("Time taken to run the above cell: " + elapsedtime + " seconds.");

# PRINT THE RESULTS
println("R-sqr on test data = " + r2)

```

Output:

R-sqr on test data = 0.5960320470835743

Next, query the test results as a data frame and use AutoVizWidget and matplotlib to visualize it.

```

# RUN A SQL QUERY
%%sql -q -o sqlResults
select * from testResults

# RUN THE CODE LOCALLY ON THE JUPYTER SERVER
%%local

# USE THE JUPYTER AUTO-PLOTTING FEATURE TO CREATE INTERACTIVE FIGURES
# CLICK THE TYPE OF PLOT TO GENERATE (LINE, AREA, BAR, AND SO ON)
sqlResults

```

The code creates a local data frame from the query output and plots the data. The `%%local` magic creates a local data frame, `sqlResults`, which you can use to plot with matplotlib.

NOTE

This Spark magic is used multiple times in this article. If the amount of data is large, you should sample to create a data frame that can fit in local memory.

Create plots by using Python matplotlib.

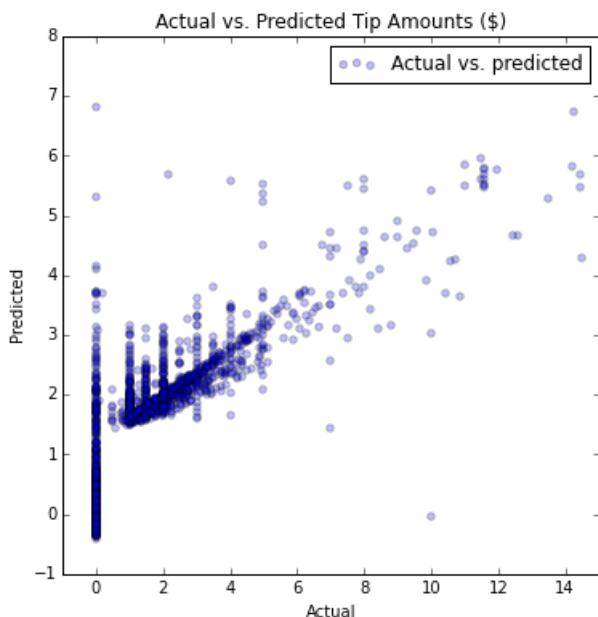
```

# RUN THE CODE LOCALLY ON THE JUPYTER SERVER AND IMPORT LIBRARIES
%%local
sqlResults
%matplotlib inline
import numpy as np

# PLOT THE RESULTS
ax=sqlResults.plot(kind='scatter', figsize=(6,6), x='tip_amount', y='prediction', color='blue', alpha = 0.25, label='Actual vs. predicted');
fit = np.polyfit(sqlResults['tip_amount'], sqlResults['prediction'], deg=1)
ax.set_title('Actual vs. Predicted Tip Amounts ($)')
ax.set_xlabel("Actual")
ax.set_ylabel("Predicted")
#ax.plot(sqlResults['tip_amount'], fit[0] * sqlResults['prediction'] + fit[1], color='magenta')
plt.axis([-1, 15, -1, 8])
plt.show(ax)

```

Output:



Create a GBT regression model

Create a GBT regression model by using the Spark ML `GBTRegressor()` function, and then evaluate the model on test data.

[Gradient-boosted trees](#) (GBTs) are ensembles of decision trees. GBTs train decision trees iteratively to minimize a loss function. You can use GBTs for regression and classification. They can handle categorical features, do not require feature scaling, and can capture nonlinearities and feature interactions. You also can use them in a multiclass-classification setting.

```

# RECORD THE START TIME
val starttime = Calendar.getInstance().getTime()

# TRAIN A GBT REGRESSION MODEL
val gbt = new GBTRegressor().setLabelCol("label").setFeaturesCol("featuresCat").setMaxIter(10)
val gbtModel = gbt.fit(indexedTRAINwithCatFeat)

# MAKE PREDICTIONS
val predictions = gbtModel.transform(indexedTESTwithCatFeat)

# COMPUTE TEST SET R2
val evaluator = new RegressionEvaluator().setLabelCol("label").setPredictionCol("prediction").setMetricName("r2")
val Test_R2 = evaluator.evaluate(predictions)

# GET THE TIME TO RUN THE CELL
val endtime = Calendar.getInstance().getTime()
val elapsedtime = ((endtime.getTime() - starttime.getTime()) / 1000).toString;
println("Time taken to run the above cell: " + elapsedtime + " seconds.");

# PRINT THE RESULTS
println("Test R-sqr is: " + Test_R2);

```

Output:

Test R-sqr is: 0.7655383534596654

Advanced modeling utilities for optimization

In this section, you use machine learning utilities that developers frequently use for model optimization. Specifically, you can optimize machine learning models three different ways by using parameter sweeping and cross-validation:

- Split the data into train and validation sets, optimize the model by using hyper-parameter sweeping on a training set, and evaluate on a validation set (linear regression)
- Optimize the model by using cross-validation and hyper-parameter sweeping by using Spark ML's CrossValidator function (binary classification)
- Optimize the model by using custom cross-validation and parameter-sweeping code to use any machine learning function and parameter set (linear regression)

Cross-validation is a technique that assesses how well a model trained on a known set of data will generalize to predict the features of data sets on which it has not been trained. The general idea behind this technique is that a model is trained on a data set of known data, and then the accuracy of its predictions is tested against an independent data set. A common implementation is to divide a data set into k -folds, and then train the model in a round-robin fashion on all but one of the folds.

Hyper-parameter optimization is the problem of choosing a set of hyper-parameters for a learning algorithm, usually with the goal of optimizing a measure of the algorithm's performance on an independent data set. A hyper-parameter is a value that you must specify outside the model training procedure. Assumptions about hyper-parameter values can affect the flexibility and accuracy of the model. Decision trees have hyper-parameters, for example, such as the desired depth and number of leaves in the tree. You must set a misclassification penalty term for a support vector machine (SVM).

A common way to perform hyper-parameter optimization is to use a grid search, also called a **parameter sweep**. In a grid search, an exhaustive search is performed through the values of a specified subset of the hyper-parameter space for a learning algorithm. Cross-validation can supply a performance metric to sort out the optimal results produced by the grid search algorithm. If you use cross-validation hyper-parameter sweeping, you can help limit problems like overfitting a model to training data. This way, the model retains the capacity to apply to the general set of data from which the training data was extracted.

Optimize a linear regression model with hyper-parameter sweeping

Next, split data into train and validation sets, use hyper-parameter sweeping on a training set to optimize the model, and evaluate on a validation set (linear regression).

```
# RECORD THE START TIME
val starttime = Calendar.getInstance().getTime()

# RENAME `tip_amount` AS A LABEL
val OneHotTRAINLabeled = OneHotTRAIN.select("tip_amount","features").withColumnRenamed(existingName="tip_amount",newName="label")
val OneHotTESTLabeled = OneHotTEST.select("tip_amount","features").withColumnRenamed(existingName="tip_amount",newName="label")
OneHotTRAINLabeled.cache()
OneHotTESTLabeled.cache()

# DEFINE THE ESTIMATOR FUNCTION: 'THE LinearRegression()' FUNCTION
val lr = new LinearRegression().setLabelCol("label").setFeaturesCol("features").setMaxIter(10)

# DEFINE THE PARAMETER GRID
val paramGrid = new ParamGridBuilder().addGrid(lr.regParam, Array(0.1, 0.01, 0.001)).addGrid(lr.fitIntercept).addGrid(lr.elasticNetParam, Array(0.1, 0.5, 0.9)).build()

# DEFINE THE PIPELINE WITH A TRAIN/TEST VALIDATION SPLIT (75% IN THE TRAINING SET), AND THEN THE SPECIFY ESTIMATOR,
# EVALUATOR, AND PARAMETER GRID
val trainPct = 0.75
val trainValidationSplit = new TrainValidationSplit().setEstimator(lr).setEvaluator(new
RegressionEvaluator).setEstimatorParamMaps(paramGrid).setTrainRatio(trainPct)

# RUN THE TRAIN VALIDATION SPLIT AND CHOOSE THE BEST SET OF PARAMETERS
val model = trainValidationSplit.fit(OneHotTRAINLabeled)

# MAKE PREDICTIONS ON THE TEST DATA BY USING THE MODEL WITH THE COMBINATION OF PARAMETERS THAT PERFORMS THE
# BEST
val testResults = model.transform(OneHotTESTLabeled).select("label", "prediction")

# COMPUTE TEST SET R2
val evaluator = new RegressionEvaluator().setLabelCol("label").setPredictionCol("prediction").setMetricName("r2")
val Test_R2 = evaluator.evaluate(testResults)

# GET THE TIME TO RUN THE CELL
val endtime = Calendar.getInstance().getTime()
val elapsedtime = ((endtime.getTime() - starttime.getTime())/1000).toString;
println("Time taken to run the above cell: " + elapsedtime + " seconds.");

println("Test R-sqr is: " + Test_R2);
```

Output:

Test R-sqr is: 0.6226484708501209

Optimize the binary classification model by using cross-validation and hyper-parameter sweeping

This section shows you how to optimize a binary classification model by using cross-validation and hyper-parameter sweeping. This uses the Spark ML `CrossValidator` function.

```

# RECORD THE START TIME
val starttime = Calendar.getInstance().getTime()

# CREATE DATA FRAMES WITH PROPERLY LABELED COLUMNS TO USE WITH THE TRAIN AND TEST SPLIT
val indexedTRAINwithCatFeatBinTargetRF =
indexedTRAINwithCatFeatBinTarget.select("labelBin", "featuresCat").withColumnRenamed(existingName="labelBin",newName="label").withColumnRenamed(existingName="featuresCat",newName="features")
val indexedTESTwithCatFeatBinTargetRF =
indexedTESTwithCatFeatBinTarget.select("labelBin", "featuresCat").withColumnRenamed(existingName="labelBin",newName="label").withColumnRenamed(existingName="featuresCat",newName="features")
indexedTRAINwithCatFeatBinTargetRF.cache()
indexedTESTwithCatFeatBinTargetRF.cache()

# DEFINE THE ESTIMATOR FUNCTION
val rf = new RandomForestClassifier().setLabelCol("label").setFeaturesCol("features").setImpurity("gini").setSeed(1234).setFeatureSubsetStrategy("auto").setMaxBins(32)

# DEFINE THE PARAMETER GRID
val paramGrid = new ParamGridBuilder().addGrid(rf.maxDepth, Array(4,8)).addGrid(rf.numTrees, Array(5,10)).addGrid(rf.minInstancesPerNode, Array(100,300)).build()

# SPECIFY THE NUMBER OF FOLDS
val numFolds = 3

# DEFINE THE TRAIN/TEST VALIDATION SPLIT (75% IN THE TRAINING SET)
val CrossValidator = new CrossValidator().setEstimator(rf).setEvaluator(new BinaryClassificationEvaluator).setEstimatorParamMaps(paramGrid).setNumFolds(numFolds)

# RUN THE TRAIN VALIDATION SPLIT AND CHOOSE THE BEST SET OF PARAMETERS
val model = CrossValidator.fit(indexedTRAINwithCatFeatBinTargetRF)

# MAKE PREDICTIONS ON THE TEST DATA BY USING THE MODEL WITH THE COMBINATION OF PARAMETERS THAT PERFORMS THE BEST
val testResults = model.transform(indexedTESTwithCatFeatBinTargetRF).select("label", "prediction")

# COMPUTE THE TEST F1 SCORE
val evaluator = new MulticlassClassificationEvaluator().setLabelCol("label").setPredictionCol("prediction").setMetricName("f1")
val Test_f1Score = evaluator.evaluate(testResults)

# GET THE TIME TO RUN THE CELL
val endtime = Calendar.getInstance().getTime()
val elapsedtime = ((endtime.getTime() - starttime.getTime()) / 1000).toString;
println("Time taken to run the above cell: " + elapsedtime + " seconds.");

```

Output:

Time to run the cell: 33 seconds.

Optimize the linear regression model by using custom cross-validation and parameter-sweeping code

Next, optimize the model by using custom code, and identify the best model parameters by using the criterion of highest accuracy. Then, create the final model, evaluate the model on test data, and save the model in Blob storage. Finally, load the model, score test data, and evaluate accuracy.

```

# RECORD THE START TIME
val starttime = Calendar.getInstance().getTime()

# DEFINE THE PARAMETER GRID AND THE NUMBER OF FOLDS
val paramGrid = new ParamGridBuilder().addGrid(rf.maxDepth, Array(5,10)).addGrid(rf.numTrees, Array(10,25,50)).build()

val nFolds = 3
val numModels = paramGrid.size
val numParamsInGrid = 2

# SPECIFY THE NUMBER OF CATEGORIES FOR CATEGORICAL VARIABLES
val categoricalFeaturesInfo = Map[Int, Int](0 -> 2, 1 -> 2, 2 -> 2, 3 -> 2)

```

```

var categoricalFeaturesInfo = Map[Int, Int]((0,2),(1,2),(2,0),(3,4))

var maxDepth = -1
var numTrees = -1
var param = ""
var paramval = -1
var validateLB = -1.0
var validateUB = -1.0
val h = 1.0 / nFolds;
val RMSE = Array.fill(numModels)(0.0)

# CREATE K-FOLDS
val splits = MLUtils.kFold(indexedTRAINbinary, numFolds = nFolds, seed=1234)

# LOOP THROUGH K-FOLDS AND THE PARAMETER GRID TO GET AND IDENTIFY THE BEST PARAMETER SET BY LEVEL OF ACCURACY
for (i < 0 to (nFolds-1)) {
    validateLB = i * h
    validateUB = (i + 1) * h
    val validationCV = trainData.filter($"rand" >= validateLB && $"rand" < validateUB)
    val trainCV = trainData.filter($"rand" < validateLB || $"rand" >= validateUB)
    val validationLabPt = validationCV.rdd.map(r => LabeledPoint(r.getDouble(targetIndRegression(0).toInt),
    Vectors.dense(featuresIndIndex.map(r.getDouble(_).toArray))));
    val trainCVLabPt = trainCV.rdd.map(r => LabeledPoint(r.getDouble(targetIndRegression(0).toInt),
    Vectors.dense(featuresIndIndex.map(r.getDouble(_).toArray))));
    validationLabPt.cache()
    trainCVLabPt.cache()

    for (nParamSets < 0 to (numModels-1)) {
        for (nParams < 0 to (numParamsInGrid-1)) {
            param = paramGrid(nParamSets).toSeq(nParams).param.toString.split("__")(1)
            paramval = paramGrid(nParamSets).toSeq(nParams).value.toInt
            if (param == "maxDepth") {maxDepth = paramval}
            if (param == "numTrees") {numTrees = paramval}
        }
        val rfModel = RandomForest.trainRegressor(trainCVLabPt, categoricalFeaturesInfo=categoricalFeaturesInfo,
            numTrees=numTrees, maxDepth=maxDepth,
            featureSubsetStrategy="auto", impurity="variance", maxBins=32)
        val labelAndPreds = validationLabPt.map { point =>
            val prediction = rfModel.predict(point.features)
            (prediction, point.label)
        }
        val validMetrics = new RegressionMetrics(labelAndPreds)
        val rmse = validMetrics.rootMeanSquaredError
        RMSE(nParamSets) += rmse
    }
    validationLabPt.unpersist();
    trainCVLabPt.unpersist();
}
val minRMSEindex = RMSE.indexOf(RMSE.min)

# GET THE BEST PARAMETERS FROM A CROSS-VALIDATION AND PARAMETER SWEEP
var best_maxDepth = -1
var best_numTrees = -1
for (nParams < 0 to (numParamsInGrid-1)) {
    param = paramGrid(minRMSEindex).toSeq(nParams).param.toString.split("__")(1)
    paramval = paramGrid(minRMSEindex).toSeq(nParams).value.toInt
    if (param == "maxDepth") {best_maxDepth = paramval}
    if (param == "numTrees") {best_numTrees = paramval}
}

# CREATE THE BEST MODEL WITH THE BEST PARAMETERS AND A FULL TRAINING DATA SET
val best_rfModel = RandomForest.trainRegressor(indexedTRAINreg, categoricalFeaturesInfo=categoricalFeaturesInfo,
    numTrees=best_numTrees, maxDepth=best_maxDepth,
    featureSubsetStrategy="auto", impurity="variance", maxBins=32)

# SAVE THE BEST RANDOM FOREST MODEL IN BLOB STORAGE
val datestamp = Calendar.getInstance().getTime().toString.replaceAll(" ", ".").replaceAll(":", "_");
val modelName = "BestCV_RF_Regression__"

```

```

val filename = modelDir.concat(modelName).concat(timestamp)
best_rfModel.save(sc, filename);

# PREDICT ON THE TRAINING SET WITH THE BEST MODEL AND THEN EVALUATE
val labelAndPreds = indexedTESTreg.map { point =>
    val prediction = best_rfModel.predict(point.features)
    (prediction, point.label)
}

val test_rmse = new RegressionMetrics(labelAndPreds).rootMeanSquaredError
val test_rsqr = new RegressionMetrics(labelAndPreds).r2

# GET THE TIME TO RUN THE CELL
val endtime = Calendar.getInstance().getTime()
val elapsedtime = ((endtime.getTime() - starttime.getTime()) / 1000).toString;
println("Time taken to run the above cell: " + elapsedtime + " seconds.");

# LOAD THE MODEL
val savedRFModel = RandomForestModel.load(sc, filename)

val labelAndPreds = indexedTESTreg.map { point =>
    val prediction = savedRFModel.predict(point.features)
    (prediction, point.label)
}

# TEST THE MODEL
val test_rmse = new RegressionMetrics(labelAndPreds).rootMeanSquaredError
val test_rsqr = new RegressionMetrics(labelAndPreds).r2

```

Output:

Time to run the cell: 61 seconds.

Consume Spark-built machine learning models automatically with Scala

For an overview of topics that walk you through the tasks that comprise the Data Science process in Azure, see [Team Data Science Process](#).

[Team Data Science Process walkthroughs](#) describes other end-to-end walkthroughs that demonstrate the steps in the Team Data Science Process for specific scenarios. The walkthroughs also illustrate how to combine cloud and on-premises tools and services into a workflow or pipeline to create an intelligent application.

[Score Spark-built machine learning models](#) shows you how to use Scala code to automatically load and score new data sets with machine learning models built in Spark and saved in Azure Blob storage. You can follow the instructions provided there, and simply replace the Python code with Scala code in this article for automated consumption.

Feature engineering in data science

1/17/2017 • 7 min to read • [Edit on GitHub](#)

This topic explains the purposes of feature engineering and provides examples of its role in the data enhancement process of machine learning. The examples used to illustrate this process are drawn from Azure Machine Learning Studio.

This **menu** links to topics that describe how to create features for data in various environments. This task is a step in the [Team Data Science Process \(TDSP\)](#).

Feature engineering attempts to increase the predictive power of learning algorithms by creating features from raw data that help facilitate the learning process. The engineering and selection of features is one part of the TDSP outlined in the [What is the Team Data Science Process lifecycle?](#) Feature engineering and selection are parts of the **Develop features** step of the TDSP.

- **feature engineering:** This process attempts to create additional relevant features from the existing raw features in the data, and to increase the predictive power of the learning algorithm.
- **feature selection:** This process selects the key subset of original data features in an attempt to reduce the dimensionality of the training problem.

Normally **feature engineering** is applied first to generate additional features, and then the **feature selection** step is performed to eliminate irrelevant, redundant, or highly correlated features.

The training data used in machine learning can often be enhanced by extraction of features from the raw data collected. An example of an engineered feature in the context of learning how to classify the images of handwritten characters is creation of a bit density map constructed from the raw bit distribution data. This map can help locate the edges of the characters more efficiently than simply using the raw distribution directly.

NOTE

[Try Azure Machine Learning for free](#)

No credit card or Azure subscription needed. [Get started now >](#)

Creating Features from Your Data - Feature Engineering

The training data consists of a matrix composed of examples (records or observations stored in rows), each of which has a set of features (variables or fields stored in columns). The features specified in the experimental design are expected to characterize the patterns in the data. Although many of the raw data fields can be directly included in the selected feature set used to train a model, it is often the case that additional (engineered) features need to be constructed from the features in the raw data to generate an enhanced training dataset.

What kind of features should be created to enhance the dataset when training a model? Engineered features that enhance the training provide information that better differentiates the patterns in the data. We expect the new features to provide additional information that is not clearly captured or easily apparent in the original or existing feature set. But this process is something of an art. Sound and productive decisions often require some domain expertise.

When starting with Azure Machine Learning, it is easiest to grasp this process concretely using samples provided in the Studio. Two examples are presented here:

- A regression example [Prediction of the number of bike rentals](#) in a supervised experiment where the target

values are known

- A text mining classification example using [Feature Hashing](#)

Example 1: Adding Temporal Features for Regression Model

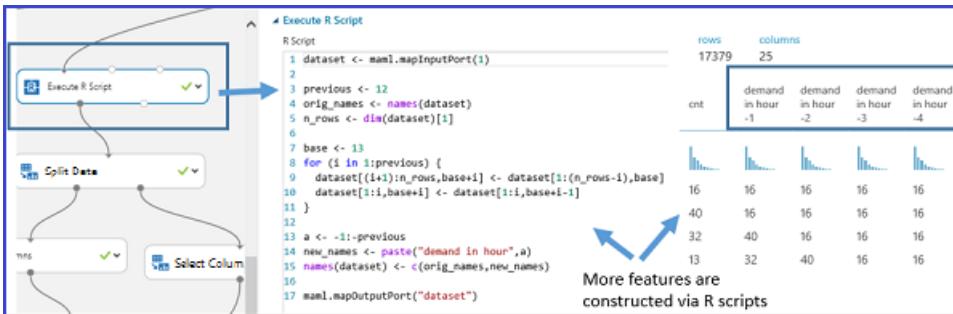
Let's use the experiment "Demand forecasting of bikes" in Azure Machine Learning Studio to demonstrate how to engineer features for a regression task. The objective of this experiment is to predict the demand for the bikes, that is, the number of bike rentals within a specific month/day/hour. The dataset "Bike Rental UCI dataset" is used as the raw input data. This dataset is based on real data from the Capital Bikeshare company that maintains a bike rental network in Washington DC in the United States. The dataset represents the number of bike rentals within a specific hour of a day in the years 2011 and year 2012 and contains 17379 rows and 17 columns. The raw feature set contains weather conditions (temperature/humidity/wind speed) and the type of the day (holiday/weekday). The field to predict is "cnt", a count which represents the bike rentals within a specific hour and which ranges from 1 to 977.

With the goal of constructing effective features in the training data, four regression models are built using the same algorithm but with four different training datasets. The four datasets represent the same raw input data, but with an increasing number of features set. These features are grouped into four categories:

1. A = weather + holiday + weekday + weekend features for the predicted day
2. B = number of bikes that were rented in each of the previous 12 hours
3. C = number of bikes that were rented in each of the previous 12 days at the same hour
4. D = number of bikes that were rented in each of the previous 12 weeks at the same hour and the same day

Besides feature set A, which already exist in the original raw data, the other three sets of features are created through the feature engineering process. Feature set B captures very recent demand for the bikes. Feature set C captures the demand for bikes at a particular hour. Feature set D captures demand for bikes at particular hour and particular day of the week. The four training datasets each includes feature set A, A+B, A+B+C, and A+B+C+D, respectively.

In the Azure Machine Learning experiment, these four training datasets are formed via four branches from the pre-processed input dataset. Except the left most branch, each of these branches contains an [Execute R Script](#) module, in which a set of derived features (feature set B, C, and D) are respectively constructed and appended to the imported dataset. The following figure demonstrates the R script used to create feature set B in the second left branch.



The comparison of the performance results of the four models are summarized in the following table. The best results are shown by features A+B+C. Note that the error rate decreases when additional feature set are included in the training data. It verifies our presumption that the feature set B, C provide additional relevant information for the regression task. But adding the D feature does not seem to provide any additional reduction in the error rate.

Features	Mean Absolute Error	Root Mean Square Error
A	89.7	124.9
A + B	51.7	88.3
A + B + C	47.6	81.1
A + B + C + D	48.3	82.1

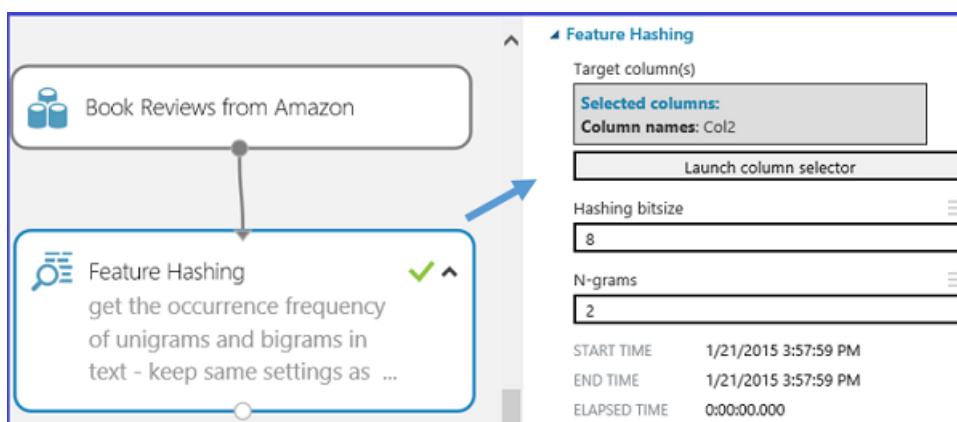
Example 2: Creating Features in Text Mining

Feature engineering is widely applied in tasks related to text mining, such as document classification and sentiment analysis. For example, when we want to classify documents into several categories, a typical assumption is that the word/phrases included in one doc category are less likely to occur in another doc category. In other words, the frequency of the words/phrases distribution is able to characterize different document categories. In text mining applications, because individual pieces of text-contents usually serve as the input data, the feature engineering process is needed to create the features involving word/phrase frequencies.

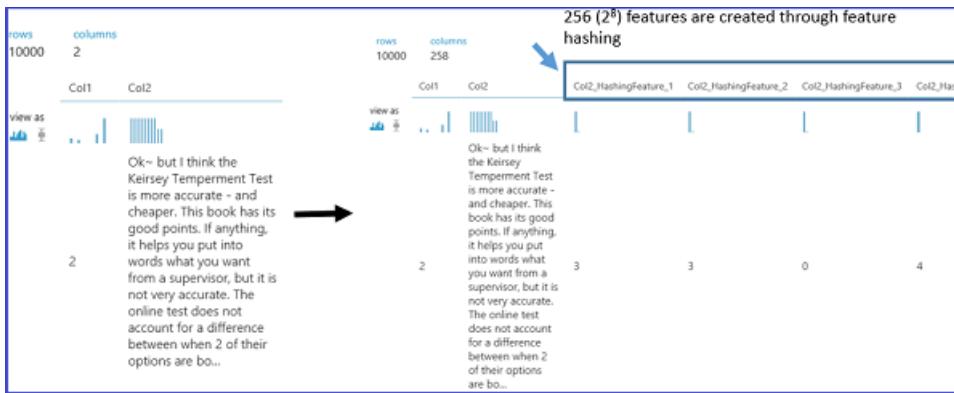
To achieve this task, a technique called **feature hashing** is applied to efficiently turn arbitrary text features into indices. Instead of associating each text feature (words/phrases) to a particular index, this method functions by applying a hash function to the features and using their hash values as indices directly.

In Azure Machine Learning, there is a [Feature Hashing](#) module that creates these word/phrase features conveniently. Following figure shows an example of using this module. The input dataset contains two columns: the book rating ranging from 1 to 5, and the actual review content. The goal of this [Feature Hashing](#) module is to retrieve a bunch of new features that show the occurrence frequency of the corresponding word(s)/phrase(s) within the particular book review. To use this module, we need to complete the following steps:

- First, select the column that contains the input text ("Col2" in this example).
- Second, set the "Hashing bitsize" to 8, which means $2^8=256$ features will be created. The word/phase in all the text will be hashed to 256 indices. The parameter "Hashing bitsize" ranges from 1 to 31. The word(s)/phrase(s) are less likely to be hashed into the same index if setting it to be a larger number.
- Third, set the parameter "N-grams" to 2. This gets the occurrence frequency of unigrams (a feature for every single word) and bigrams (a feature for every pair of adjacent words) from the input text. The parameter "N-grams" ranges from 0 to 10, which indicates the maximum number of sequential words to be included in a feature.



The following figure shows what these new features look like.



Conclusion

Engineered and selected features increase the efficiency of the training process which attempts to extract the key information contained in the data. They also improve the power of these models to classify the input data accurately and to predict outcomes of interest more robustly. Feature engineering and selection can also combine to make the learning more computationally tractable. It does so by enhancing and then reducing the number of features needed to calibrate or train a model. Mathematically speaking, the features selected to train the model are a minimal set of independent variables that explain the patterns in the data and then predict outcomes successfully.

Note that it is not always necessarily to perform feature engineering or feature selection. Whether it is needed or not depends on the data we have or collect, the algorithm we pick, and the objective of the experiment.

Create features for Azure blob storage data using Panda

1/17/2017 • 2 min to read • [Edit on GitHub](#)

This document shows how to create features for data that is stored in Azure blob container using the [Pandas](#) Python package. After outlining how to load the data into a Panda data frame, it shows how to generate categorical features using Python scripts with indicator values and binning features.

This **menu** links to topics that describe how to create features for data in various environments. This task is a step in the [Team Data Science Process \(TDSP\)](#).

Prerequisites

This article assumes that you have created an Azure blob storage account and have stored your data there. If you need instructions to set up an account, see [Create an Azure Storage account](#)

Load the data into a Pandas data frame

In order to do explore and manipulate a dataset, it must be downloaded from the blob source to a local file which can then be loaded in a Pandas data frame. Here are the steps to follow for this procedure:

1. Download the data from Azure blob with the following sample Python code using blob service. Replace the variable in the code below with your specific values:

```
from azure.storage.blob import BlobService  
import tables  
  
STORAGEACCOUNTNAME=<storage_account_name>  
STORAGEACCOUNTKEY=<storage_account_key>  
LOCALFILENAME=<local_file_name>  
CONTAINERNAME=<container_name>  
BLOBNAME=<blob_name>  
  
#download from blob  
t1=time.time()  
blob_service=BlobService(account_name=STORAGEACCOUNTNAME,account_key=STORAGEACCOUNTKEY)  
blob_service.get_blob_to_path(CONTAINERNAME,BLOBNAME,LOCALFILENAME)  
t2=time.time()  
print("It takes %s seconds to download "+blobname) % (t2 - t1)
```

2. Read the data into a Pandas data-frame from the downloaded file.

```
#LOCALFILE is the file path  
dataframe_blobdata = pd.read_csv(LOCALFILE)
```

Now you are ready to explore the data and generate features on this dataset.

Feature Generation

The next two sections show how to generate categorical features with indicator values and binning features using Python scripts.

Indicator value based Feature Generation

Categorical features can be created as follows:

1. Inspect the distribution of the categorical column:

```
dataframe_blobdata['<categorical_column>'].value_counts()
```

2. Generate indicator values for each of the column values

```
#generate the indicator column
dataframe_blobdata_identity = pd.get_dummies(dataframe_blobdata['<categorical_column>'], prefix='<categorical_column>_identity')
```

3. Join the indicator column with the original data frame

```
#Join the dummy variables back to the original data frame
dataframe_blobdata_with_identity = dataframe_blobdata.join(dataframe_blobdata_identity)
```

4. Remove the original variable itself:

```
#Remove the original column rate_code in df1_with_dummy
dataframe_blobdata_with_identity.drop('<categorical_column>', axis=1, inplace=True)
```

Binning Feature Generation

For generating binned features, we proceed as follows:

1. Add a sequence of columns to bin a numeric column

```
bins = [0, 1, 2, 4, 10, 40]
dataframe_blobdata_bin_id = pd.cut(dataframe_blobdata['<numeric_column>'], bins)
```

2. Convert binning to a sequence of boolean variables

```
dataframe_blobdata_bin_bool = pd.get_dummies(dataframe_blobdata_bin_id, prefix='<numeric_column>')
```

3. Finally, Join the dummy variables back to the original data frame

```
dataframe_blobdata_with_bin_bool = dataframe_blobdata.join(dataframe_blobdata_bin_bool)
```

Writing data back to Azure blob and consuming in Azure Machine Learning

After you have explored the data and created the necessary features, you can upload the data (sampled or featurized) to an Azure blob and consume it in Azure Machine Learning using the following steps: Note that additional features can be created in the Azure Machine Learning Studio as well.

1. Write the data frame to local file

```
dataframe.to_csv(os.path.join(os.getcwd(), LOCALFILENAME), sep='\t', encoding='utf-8', index=False)
```

2. Upload the data to Azure blob as follows:

```

from azure.storage.blob import BlobService
import tables

STORAGEACCOUNTNAME=<storage_account_name>
LOCALFILENAME=<local_file_name>
STORAGEACCOUNTKEY=<storage_account_key>
CONTAINERNAME=<container_name>
BLOBNAME=<blob_name>

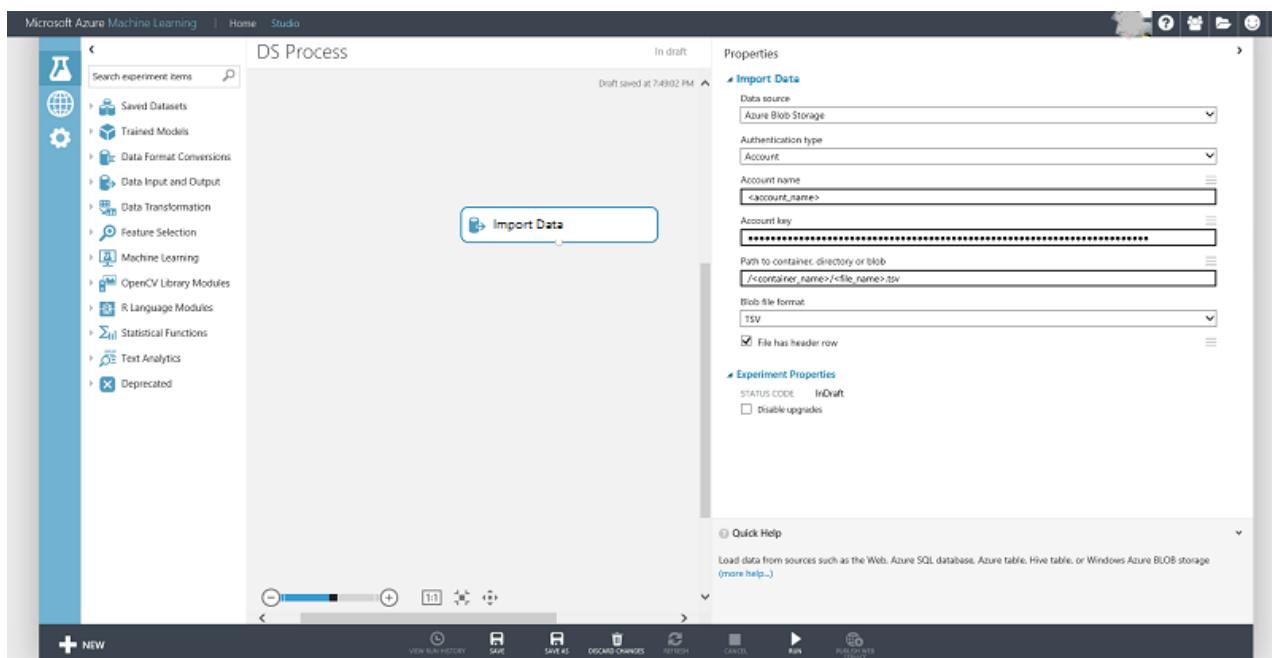
output_blob_service=BlobService(account_name=STORAGEACCOUNTNAME,account_key=STORAGEACCOUNTKEY)
localfileprocessed = os.path.join(os.getcwd(),LOCALFILENAME) #assuming file is in current working directory

try:
    #perform upload
    output_blob_service.put_block_blob_from_path(CONTAINERNAME,BLOBNAME,localfileprocessed)

except:
    print ("Something went wrong with uploading blob:"+BLOBNAME)

```

- Now the data can be read from the blob using the Azure Machine Learning Import Data module as shown in the screen below:



Create features for data in SQL Server using SQL and Python

1/17/2017 • 5 min to read • [Edit on GitHub](#)

This document shows how to generate features for data stored in a SQL Server VM on Azure that help algorithms learn more efficiently from the data. This can be done by using SQL or by using a programming language like Python, both of which are demonstrated here.

This **menu** links to topics that describe how to create features for data in various environments. This task is a step in the [Team Data Science Process \(TDSP\)](#).

NOTE

For a practical example, you can consult the [NYC Taxi dataset](#) and refer to the IPNB titled [NYC Data wrangling using IPython Notebook and SQL Server](#) for an end-to-end walk-through.

Prerequisites

This article assumes that you have:

- Created an Azure storage account. If you need instructions, see [Create an Azure Storage account](#)
- Stored your data in SQL Server. If you have not, see [Move data to an Azure SQL Database for Azure Machine Learning](#) for instructions on how to move the data there.

Feature Generation with SQL

In this section, we describe ways of generating features using SQL:

1. [Count based Feature Generation](#)
2. [Binning Feature Generation](#)
3. [Rolling out the features from a single column](#)

NOTE

Once you generate additional features, you can either add them as columns to the existing table or create a new table with the additional features and primary key, that can be joined with the original table.

Count based Feature Generation

This document demonstrates two ways of generating count features. The first method uses conditional sum and the second method uses the 'where' clause. These can then be joined with the original table (using primary key columns) to have count features alongside the original data.

```
select <column_name1>,<column_name2>,<column_name3>, COUNT(*) as Count_Features from <tablename> group by <column_name1>,<column_name2>,<column_name3>
```

```
select <column_name1>,<column_name2> , sum(1) as Count_Features from <tablename>
where <column_name3>='<some_value>' group by <column_name1>,<column_name2>
```

Binning Feature Generation

The following example shows how to generate binned features by binning (using 5 bins) a numerical column that can be used as a feature instead:

```
'SELECT <column_name>, NTILE(5) OVER (ORDER BY <column_name>) AS BinNumber from <tablename>'
```

Rolling out the features from a single column

In this section, we demonstrate how to roll-out a single column in a table to generate additional features. The example assumes that there is a latitude or longitude column in the table from which you are trying to generate features.

Here is a brief primer on latitude/longitude location data (resourced from stackoverflow

<http://gis.stackexchange.com/questions/8650/how-to-measure-the-accuracy-of-latitude-and-longitude>). This is useful to understand before featurizing the location field:

- The sign tells us whether we are north or south, east or west on the globe.
- A nonzero hundreds digit tells us we're using longitude, not latitude!
- The tens digit gives a position to about 1,000 kilometers. It gives us useful information about what continent or ocean we are on.
- The units digit (one decimal degree) gives a position up to 111 kilometers (60 nautical miles, about 69 miles). It can tell us roughly what large state or country we are in.
- The first decimal place is worth up to 11.1 km: it can distinguish the position of one large city from a neighboring large city.
- The second decimal place is worth up to 1.1 km: it can separate one village from the next.
- The third decimal place is worth up to 110 m: it can identify a large agricultural field or institutional campus.
- The fourth decimal place is worth up to 11 m: it can identify a parcel of land. It is comparable to the typical accuracy of an uncorrected GPS unit with no interference.
- The fifth decimal place is worth up to 1.1 m: it distinguishes trees from each other. Accuracy to this level with commercial GPS units can only be achieved with differential correction.
- The sixth decimal place is worth up to 0.11 m: you can use this for laying out structures in detail, for designing landscapes, building roads. It should be more than good enough for tracking movements of glaciers and rivers. This can be achieved by taking painstaking measures with GPS, such as differentially corrected GPS.

The location information can be featurized as follows, separating out region, location and city information.

Note that once can also call a REST end point such as Bing Maps API available at

<https://msdn.microsoft.com/library/h701710.aspx> to get the region/district information.

```
select
<location_columnname>
,round(<location_columnname>,0) as l1
,12=case when LEN(PARSENAME(round(ABS(<location_columnname>) - FLOOR(ABS(<location_columnname>)),6),1)) >= 1 then
substring(PARSENAME(round(ABS(<location_columnname>) - FLOOR(ABS(<location_columnname>)),6),1),1,1) else '0' end
,13=case when LEN(PARSENAME(round(ABS(<location_columnname>) - FLOOR(ABS(<location_columnname>)),6),1)) >= 2 then
substring(PARSENAME(round(ABS(<location_columnname>) - FLOOR(ABS(<location_columnname>)),6),1),2,1) else '0' end
,14=case when LEN(PARSENAME(round(ABS(<location_columnname>) - FLOOR(ABS(<location_columnname>)),6),1)) >= 3 then
substring(PARSENAME(round(ABS(<location_columnname>) - FLOOR(ABS(<location_columnname>)),6),1),3,1) else '0' end
,15=case when LEN(PARSENAME(round(ABS(<location_columnname>) - FLOOR(ABS(<location_columnname>)),6),1)) >= 4 then
substring(PARSENAME(round(ABS(<location_columnname>) - FLOOR(ABS(<location_columnname>)),6),1),4,1) else '0' end
,16=case when LEN(PARSENAME(round(ABS(<location_columnname>) - FLOOR(ABS(<location_columnname>)),6),1)) >= 5 then
substring(PARSENAME(round(ABS(<location_columnname>) - FLOOR(ABS(<location_columnname>)),6),1),5,1) else '0' end
,17=case when LEN(PARSENAME(round(ABS(<location_columnname>) - FLOOR(ABS(<location_columnname>)),6),1)) >= 6 then
substring(PARSENAME(round(ABS(<location_columnname>) - FLOOR(ABS(<location_columnname>)),6),1),6,1) else '0' end
from <tablename>
```

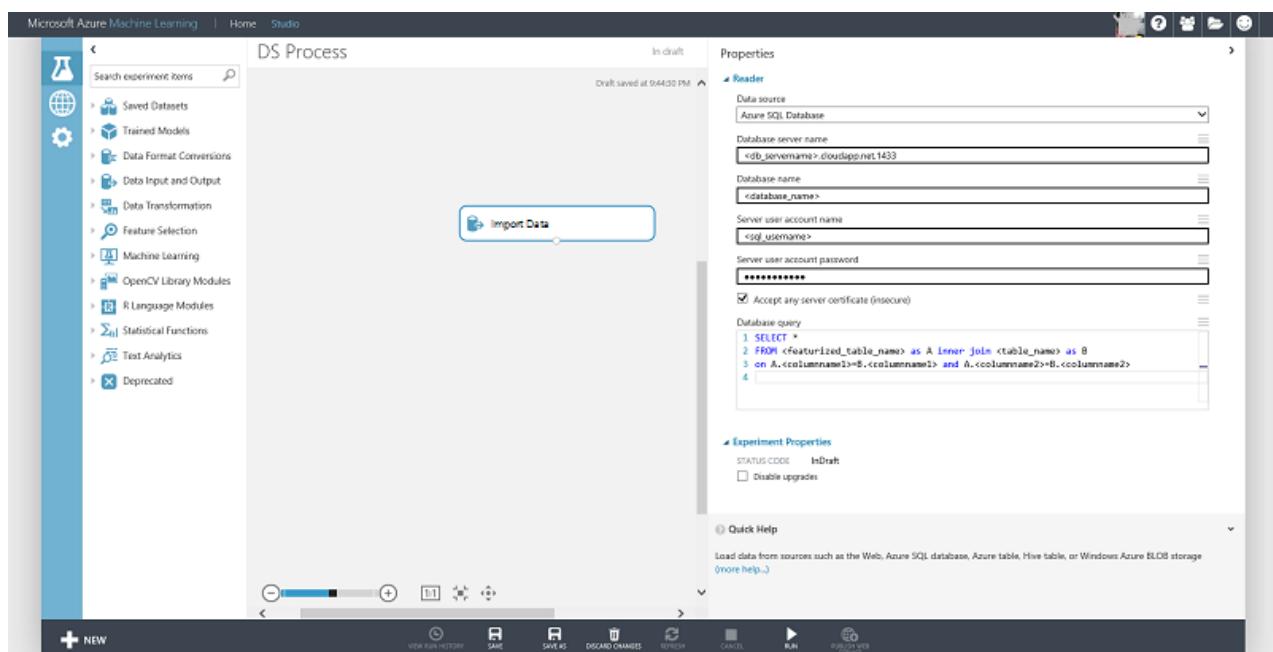
The above location based features can be further used to generate additional count features as described earlier.

TIP

You can programmatically insert the records using your language of choice. You may need to insert the data in chunks to improve write efficiency [Check out the example of how to do this using pyodbc here](#). Another alternative is to insert data in the database using [BCP utility](#)

Connecting to Azure Machine Learning

The newly generated feature can be added as a column to an existing table or stored in a new table and joined with the original table for machine learning. Features can be generated or accessed if already created, using the [Import Data](#) module in Azure ML as shown below:



Using a programming language like Python

Using Python to generate features when the data is in SQL Server is similar to processing data in Azure blob using Python as documented in [Process Azure Blob data in you data science environment](#). The data needs to be loaded from the database into a pandas data frame and then can be processed further. We document the process of connecting to the database and loading the data into the data frame in this section.

The following connection string format can be used to connect to a SQL Server database from Python using pyodbc (replace servername, dbname, username and password with your specific values):

```
#Set up the SQL Azure connection
import pyodbc
conn = pyodbc.connect('DRIVER={SQL Server};SERVER=<servername>;DATABASE=<dbname>;UID=<username>;PWD=<password>')
```

The [Pandas library](#) in Python provides a rich set of data structures and data analysis tools for data manipulation for Python programming. The code below reads the results returned from a SQL Server database into a Pandas data frame:

```
# Query database and load the returned results in pandas data frame
data_frame = pd.read_sql("select <columnname1>,<columnname2>... from <tablename>", conn)
```

Now you can work with the Pandas data frame as covered in topics [Create features for Azure blob storage data using Panda](#).

Create features for data in an Hadoop cluster using Hive queries

1/17/2017 • 7 min to read • [Edit on GitHub](#)

This document shows how to create features for data stored in an Azure HDInsight Hadoop cluster using Hive queries. These Hive queries use embedded Hive User Defined Functions (UDFs), the scripts for which are provided.

The operations needed to create features can be memory intensive. The performance of Hive queries becomes more critical in such cases and can be improved by tuning certain parameters. The tuning of these parameters is discussed in the final section.

Examples of the queries that are presented are specific to the [NYC Taxi Trip Data](#) scenarios are also provided in [Github repository](#). These queries already have data schema specified and are ready to be submitted to run. In the final section, parameters that users can tune so that the performance of Hive queries can be improved are also discussed.

This **menu** links to topics that describe how to create features for data in various environments. This task is a step in the [Team Data Science Process \(TDSP\)](#).

Prerequisites

This article assumes that you have:

- Created an Azure storage account. If you need instructions, see [Create an Azure Storage account](#)
- Provisioned a customized Hadoop cluster with the HDInsight service. If you need instructions, see [Customize Azure HDInsight Hadoop Clusters for Advanced Analytics](#).
- The data has been uploaded to Hive tables in Azure HDInsight Hadoop clusters. If it has not, please follow [Create and load data to Hive tables](#) to upload data to Hive tables first.
- Enabled remote access to the cluster. If you need instructions, see [Access the Head Node of Hadoop Cluster](#).

Feature Generation

In this section, several examples of the ways in which features can be generating using Hive queries are described. Once you have generated additional features, you can either add them as columns to the existing table or create a new table with the additional features and primary key, which can then be joined with the original table. Here are the examples presented:

1. [Frequency based Feature Generation](#)
2. [Risks of Categorical Variables in Binary Classification](#)
3. [Extract features from Datetime Field](#)
4. [Extract features from Text Field](#)
5. [Calculate distance between GPS coordinates](#)

Frequency based Feature Generation

It is often useful to calculate the frequencies of the levels of a categorical variable, or the frequencies of certain combinations of levels from multiple categorical variables. Users can use the following script to calculate these frequencies:

```

select
  a.<column_name1>, a.<column_name2>, a.sub_count/sum(a.sub_count) over () as frequency
from
(
  select
    <column_name1>,<column_name2>, count(*) as sub_count
    from <databasename>.<tablename> group by <column_name1>,<column_name2>
)a
order by frequency desc;

```

Risks of Categorical Variables in Binary Classification

In binary classification, we need to convert non-numeric categorical variables into numeric features when the models being used only take numeric features. This is done by replacing each non-numeric level with a numeric risk. In this section, we show some generic Hive queries that calculate the risk values (log odds) of a categorical variable.

```

set smooth_param1=1;
set smooth_param2=20;
select
  <column_name1>,<column_name2>,
  ln((sum_target+${hiveconf:smooth_param1})/(record_count-sum_target+${hiveconf:smooth_param2}-${hiveconf:smooth_param1})) as risk
from
(
  select
    <column_name1>,<column_name2>, sum(binary_target) as sum_target, sum(1) as record_count
  from
  (
    select
      <column_name1>,<column_name2>, if(target_column>0,1,0) as binary_target
      from <databasename>.<tablename>
  )a
  group by <column_name1>,<column_name2>
)b

```

In this example, variables `smooth_param1` and `smooth_param2` are set to smooth the risk values calculated from the data. Risks have a range between -Inf and Inf. A risks > 0 indicates that the probability that the target is equal to 1 is greater than 0.5.

After the risk table is calculated, users can assign risk values to a table by joining it with the risk table. The Hive joining query was provided in previous section.

Extract features from Datetime Fields

Hive comes with a set of UDFs for processing datetime fields. In Hive, the default datetime format is 'yyyy-MM-dd 00:00:00' ('1970-01-01 12:21:32' for example). In this section, we show examples that extract the day of a month, the month from a datetime field, and other examples that convert a datetime string in a format other than the default format to a datetime string in default format.

```

select day(<datetime field>), month(<datetime field>)
  from <databasename>.<tablename>;

```

This Hive query assumes that the `<datetime field>` is in the default datetime format.

If a datetime field is not in the default format, you need to convert the datetime field into Unix time stamp first, and then convert the Unix time stamp to a datetime string that is in the default format. When the datetime is in default format, users can apply the embedded datetime UDFs to extract features.

```

select from_unixtime(unix_timestamp(<datetime field>,'<pattern of the datetime field>'))
  from <databasename>.<tablename>;

```

In this query, if the <datetime field> has the pattern like 03/26/2015 12:04:39, the '<pattern of the datetime field>' should be 'MM/dd/yyyy HH:mm:ss'. To test it, users can run

```
select from_unixtime(unix_timestamp('05/15/2015 09:32:10','MM/dd/yyyy HH:mm:ss'))  
from hivesamplable limit 1;
```

The *hivesamplable* in this query comes preinstalled on all Azure HDInsight Hadoop clusters by default when the clusters are provisioned.

Extract features from Text Fields

When the Hive table has a text field that contains a string of words that are delimited by spaces, the following query extracts the length of the string, and the number of words in the string.

```
select length(<text field>) as str_len, size(split(<text field>,' ')) as word_num  
from <databasename>. <tablename>;
```

Calculate distances between sets of GPS coordinates

The query given in this section can be directly applied to the NYC Taxi Trip Data. The purpose of this query is to show how to apply an embedded mathematical functions in Hive to generate features.

The fields that are used in this query are the GPS coordinates of pickup and dropoff locations, named *pickup_longitude*, *pickup_latitude*, *dropoff_longitude*, and *dropoff_latitude*. The queries that calculate the direct distance between the pickup and dropoff coordinates are:

```
set R=3959;  
set pi=radians(180);  
select pickup_longitude, pickup_latitude, dropoff_longitude, dropoff_latitude,  
    ${hiveconf:R}*2*atan((1-sqrt(1-pow(sin((dropoff_latitude-pickup_latitude)  
        *${hiveconf:pi}/180/2),2)-cos(pickup_latitude*${hiveconf:pi}/180)  
        *cos(dropoff_latitude*${hiveconf:pi}/180)*pow(sin((dropoff_longitude-pickup_longitude)*${hiveconf:pi}/180/2),2)))  
        /sqrt(pow(sin((dropoff_latitude-pickup_latitude)*${hiveconf:pi}/180/2),2)  
        +cos(pickup_latitude*${hiveconf:pi}/180)*cos(dropoff_latitude*${hiveconf:pi}/180)*  
        pow(sin((dropoff_longitude-pickup_longitude)*${hiveconf:pi}/180/2),2))) as direct_distance  
from nyctaxi.trip  
where pickup_longitude between -90 and 0  
and pickup_latitude between 30 and 90  
and dropoff_longitude between -90 and 0  
and dropoff_latitude between 30 and 90  
limit 10;
```

The mathematical equations that calculate the distance between two GPS coordinates can be found on the [Movable Type Scripts](#) site, authored by Peter Lapisu. In his Javascript, the function `toRad()` is just `lat_or_lon*pi/180`, which converts degrees to radians. Here, `*lat_or_lon` is the latitude or longitude. Since Hive does not provide the function `atan2`, but provides the function `atan`, the `atan2` function is implemented by `atan` function in the above Hive query using the definition provided in [Wikipedia](#).

$$\text{atan2}(y, x) = 2 \arctan \frac{\sqrt{x^2 + y^2} - x}{y}.$$

A full list of Hive embedded UDFs can be found in the [Built-in Functions](#) section on the [Apache Hive wiki](#)).

Advanced topics: Tune Hive Parameters to Improve Query Speed

The default parameter settings of Hive cluster might not be suitable for the Hive queries and the data that the queries are processing. In this section, we discuss some parameters that users can tune that improve the performance of Hive queries. Users need to add the parameter tuning queries before the queries of processing

data.

1. **Java heap space**: For queries involving joining large datasets, or processing long records, **running out of heap space** is one of the common error. This can be tuned by setting parameters *mapreduce.map.java.opts* and *mapreduce.task.io.sort.mb* to desired values. Here is an example:

```
set mapreduce.map.java.opts=-Xmx4096m;
set mapreduce.task.io.sort.mb=-Xmx1024m;
```

This parameter allocates 4GB memory to Java heap space and also makes sorting more efficient by allocating more memory for it. It is a good idea to play with these allocations if there are any job failure errors related to heap space.

2. **DFS block size** : This parameter sets the smallest unit of data that the file system stores. As an example, if the DFS block size is 128MB, then any data of size less than and up to 128MB is stored in a single block, while data that is larger than 128MB is allotted extra blocks. Choosing a very small block size causes large overheads in Hadoop since the name node has to process many more requests to find the relevant block pertaining to the file. A recommended setting when dealing with gigabytes (or larger) data is :

```
set dfs.block.size=128m;
```

3. **Optimizing join operation in Hive** : While join operations in the map/reduce framework typically take place in the reduce phase, sometimes, enormous gains can be achieved by scheduling joins in the map phase (also called "mapjoins"). To direct Hive to do this whenever possible, we can set :

```
set hive.auto.convert.join=true;
```

4. **Specifying the number of mappers to Hive** : While Hadoop allows the user to set the number of reducers, the number of mappers is typically not be set by the user. A trick that allows some degree of control on this number is to choose the Hadoop variables, *mapred.min.split.size* and *mapred.max.split.size* as the size of each map task is determined by :

```
num_maps = max(mapred.min.split.size, min(mapred.max.split.size, dfs.block.size))
```

Typically, the default value of *mapred.min.split.size* is 0, that of *mapred.max.split.size* is **Long.MAX** and that of *dfs.block.size* is 64MB. As we can see, given the data size, tuning these parameters by "setting" them allows us to tune the number of mappers used.

5. A few other more **advanced options** for optimizing Hive performance are mentioned below. These allow you to set the memory allocated to map and reduce tasks, and can be useful in tweaking performance. Please keep in mind that the *mapreduce.reduce.memory.mb* cannot be greater than the physical memory size of each worker node in the Hadoop cluster.

```
set mapreduce.map.memory.mb = 2048;
set mapreduce.reduce.memory.mb=6144;
set mapreduce.reduce.java.opts=-Xmx8192m;
set mapred.reduce.tasks=128;
set mapred.tasktracker.reduce.tasks.maximum=128;
```

Feature selection in the Team Data Science Process (TDSP)

1/17/2017 • 4 min to read • [Edit on GitHub](#)

This article explains the purposes of feature selection and provides examples of its role in the data enhancement process of machine learning. These examples are drawn from Azure Machine Learning Studio.

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

The engineering and selection of features is one part of the Team Data Science Process (TDSP) outlined in [What is the Team Data Science Process?](#). Feature engineering and selection are parts of the **Develop features** step of the TDSP.

- **feature engineering:** This process attempts to create additional relevant features from the existing raw features in the data, and to increase predictive power to the learning algorithm.
- **feature selection:** This process selects the key subset of original data features in an attempt to reduce the dimensionality of the training problem.

Normally **feature engineering** is applied first to generate additional features, and then the **feature selection** step is performed to eliminate irrelevant, redundant, or highly correlated features.

Filtering Features from Your Data - Feature Selection

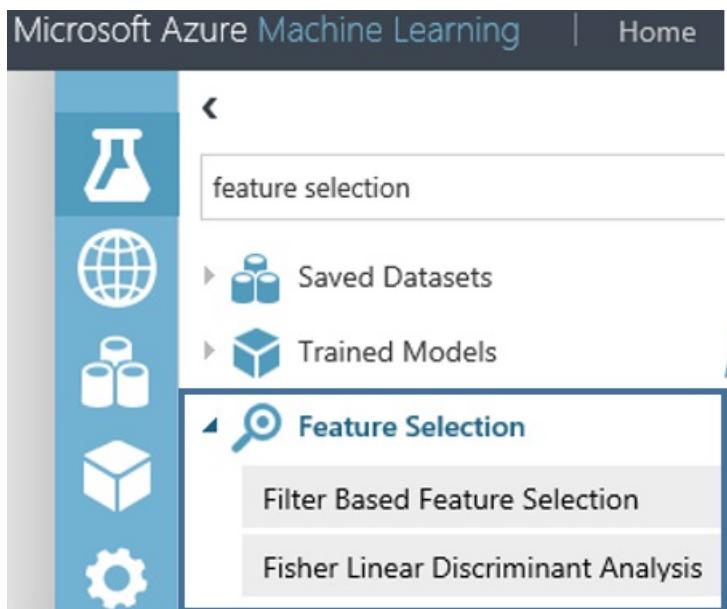
Feature selection is a process that is commonly applied for the construction of training datasets for predictive modeling tasks such as classification or regression tasks. The goal is to select a subset of the features from the original dataset that reduce its dimensions by using a minimal set of features to represent the maximum amount of variance in the data. This subset of features are, then, the only features to be included to train the model. Feature selection serves two main purposes.

- First, feature selection often increases classification accuracy by eliminating irrelevant, redundant, or highly correlated features.
- Second, it decreases the number of features which makes model training process more efficient. This is particularly important for learners that are expensive to train such as support vector machines.

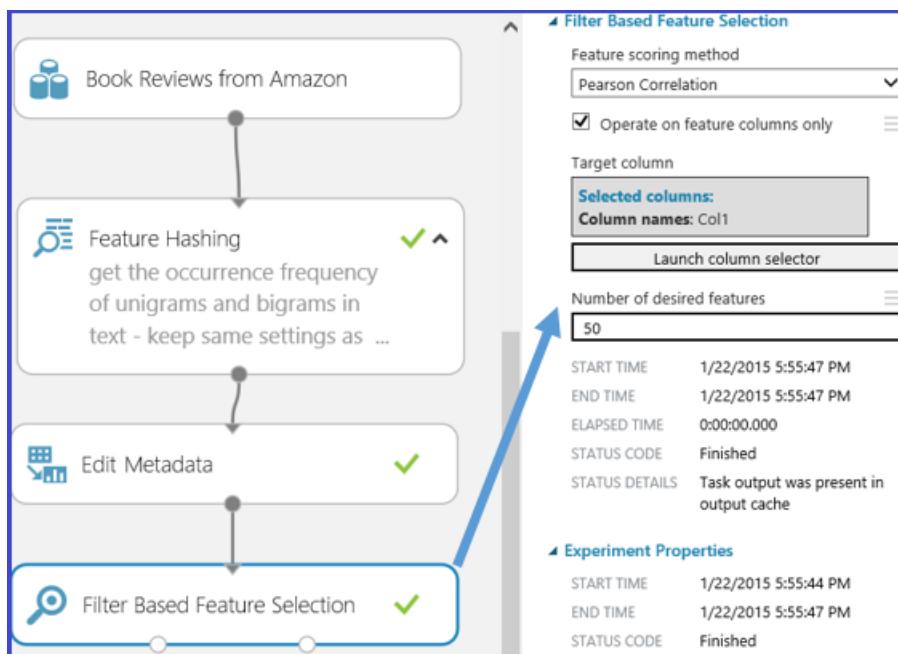
Although feature selection does seek to reduce the number of features in the dataset used to train the model, it is not usually referred to by the term "dimensionality reduction". Feature selection methods extract a subset of original features in the data without changing them. Dimensionality reduction methods employ engineered features that can transform the original features and thus modify them. Examples of dimensionality reduction methods include Principal Component Analysis, canonical correlation analysis, and Singular Value Decomposition.

Among others, one widely applied category of feature selection methods in a supervised context is called "filter based feature selection". By evaluating the correlation between each feature and the target attribute, these methods apply a statistical measure to assign a score to each feature. The features are then ranked by the score, which may be used to help set the threshold for keeping or eliminating a specific feature. Examples of the statistical measures used in these methods include Person correlation, mutual information, and the Chi squared test.

In Azure Machine Learning Studio, there are modules provided for feature selection. As shown in the following figure, these modules include [Filter-Based Feature Selection](#) and [Fisher Linear Discriminant Analysis](#).



Consider, for example, the use of the [Filter-Based Feature Selection](#) module. For the purpose of convenience, we continue to use the text mining example outlined above. Assume that we want to build a regression model after a set of 256 features are created through the [Feature Hashing](#) module, and that the response variable is the "Col1" and represents a book review ratings ranging from 1 to 5. By setting "Feature scoring method" to be "Pearson Correlation", the "Target column" to be "Col1", and the "Number of desired features" to 50. Then the module [Filter-Based Feature Selection](#) will produce a dataset containing 50 features together with the target attribute "Col1". The following figure shows the flow of this experiment and the input parameters we just described.



The following figure shows the resulting datasets. Each feature is scored based on the Pearson Correlation between itself and the target attribute "Col1". The features with top scores are kept.

ROWS 10000 COLUMNS 51

view as

Filter Based Feature Selection ✓

Right click the 1st port

Col1 Col2_HashingFeature_203 Col2_HashingFeature_146 Col2_HashingFeature_122 Col2

Col1	Col2_HashingFeature_203	Col2_HashingFeature_146	Col2_HashingFeature_122	Col2
2	6	1	2	6
2	6	4	7	5
1	9	2	1	3
2	5	2	2	3
2	3	1	1	0
2	11	6	5	5
1	2	2	3	1
2	4	3	2	1
2	2	2	6	3
2	2	0	1	0
1	11	4	3	5
2	1	0	5	0
2	2	1	2	2
+	+	+	+	+
2	2	1	2	2

The corresponding scores of the selected features are shown in the following figure.

rows 1 columns 51

view as

Filter Based Feature Selection ✓

Right click the 2nd port

Col1 Col2_HashingFeature_203 Col2_HashingFeature_146 Col2_HashingFeature_122 Col2_Hash

Col1	Col2_HashingFeature_203	Col2_HashingFeature_146	Col2_HashingFeature_122	Col2_Hash
1	0.083607	0.060681	0.05716	0.056381

By applying this [Filter-Based Feature Selection](#) module, 50 out of 256 features are selected because they have the most correlated features with the target variable "Col1", based on the scoring method "Pearson Correlation".

Conclusion

Feature engineering and feature selection are two commonly Engineered and selected features increase the efficiency of the training process which attempts to extract the key information contained in the data. They also improve the power of these models to classify the input data accurately and to predict outcomes of interest more robustly. Feature engineering and selection can also combine to make the learning more computationally tractable. It does so by enhancing and then reducing the number of features needed to calibrate or train a model.

Mathematically speaking, the features selected to train the model are a minimal set of independent variables that explain the patterns in the data and then predict outcomes successfully.

Note that it is not always necessarily to perform feature engineering or feature selection. Whether it is needed or not depends on the data we have or collect, the algorithm we pick, and the objective of the experiment.

Convert a Machine Learning training experiment to a predictive experiment

1/17/2017 • 7 min to read • [Edit on GitHub](#)

Azure Machine Learning enables you to build, test, and deploy predictive analytics solutions.

Once you've created and iterated on a *training experiment* to train your predictive analytics model, and you're ready to use it to score new data, you need to prepare and streamline your experiment for scoring. You can then deploy this *predictive experiment* as an Azure web service so that users can send data to your model and receive your model's predictions.

By converting to a predictive experiment, you're getting your trained model ready to be deployed as a web service. Users of the web service will send input data to your model and your model will send back the prediction results. So as you convert to a predictive experiment you will want to keep in mind how you expect your model to be used by others.

NOTE

[Try Azure Machine Learning for free](#)

No credit card or Azure subscription needed. [Get started now >](#)

The process of converting a training experiment to a predictive experiment involves three steps:

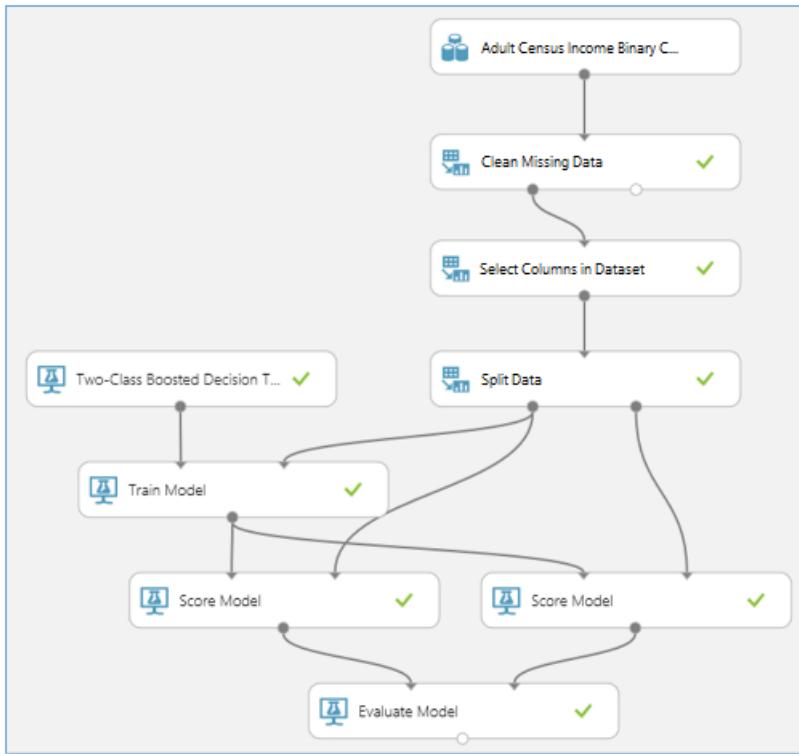
1. Save the machine learning model that you've trained, and then replace the machine learning algorithm and [Train Model](#) modules with your saved trained model.
2. Trim the experiment to only those modules that are needed for scoring. A training experiment includes a number of modules that are necessary for training but are not needed once the model is trained and ready to use for scoring.
3. Define where in your experiment you will accept data from the web service user, and what data will be returned.

Set Up Web Service button

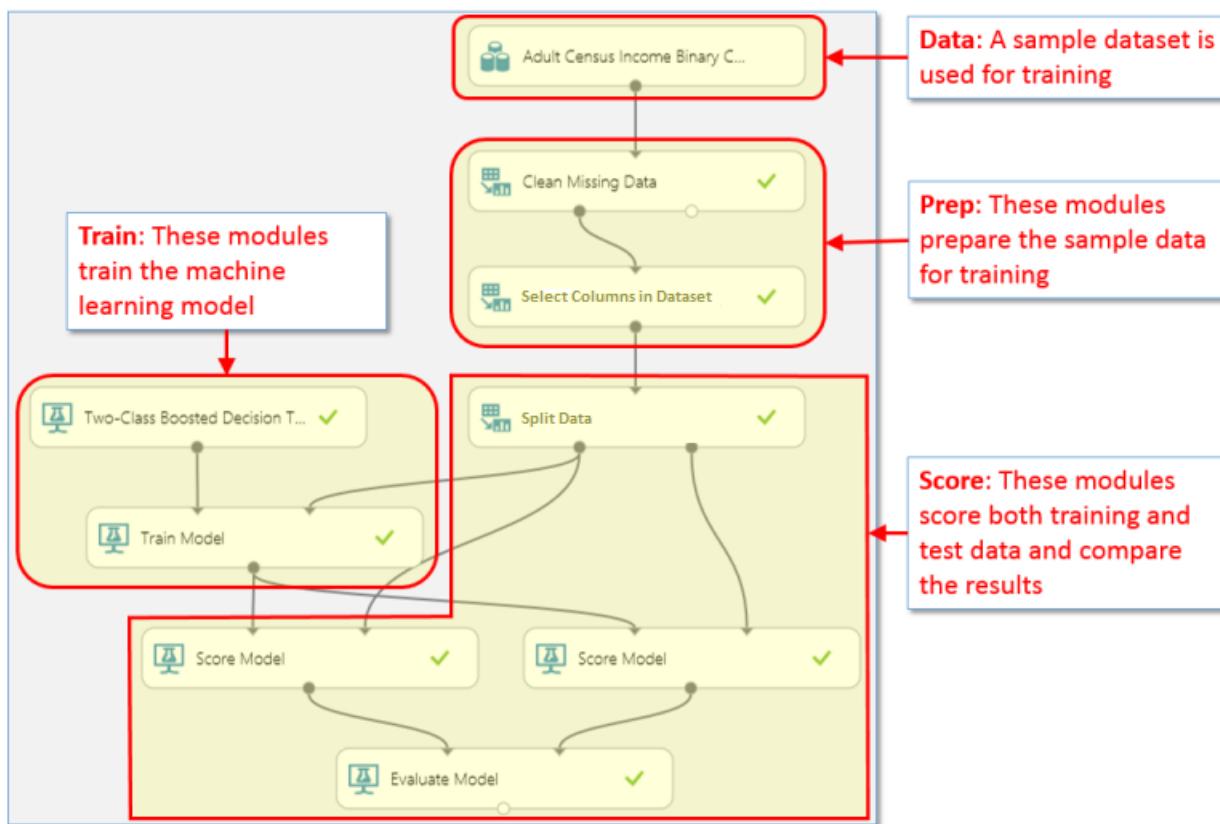
After you have run your experiment (**RUN** button at the bottom of the experiment canvas), the **Set Up Web Service** button (select the **Predictive Web Service** option) will perform for you the three steps of converting your training experiment to a predictive experiment:

1. It saves your trained model as a module in the **Trained Models** section of the module palette (to the left of the experiment canvas), then replaces the machine learning algorithm and [Train Model](#) modules with the saved trained model.
2. It removes modules that are clearly not needed. In our example, this includes the [Split Data](#), 2nd [Score Model](#), and [Evaluate Model](#) modules.
3. It creates Web service input and output modules and adds them in default locations in your experiment.

For example, the following experiment trains a two-class boosted decision tree model using sample census data:



The modules in this experiment perform basically four different functions:



When you convert this training experiment to a predictive experiment, some of these modules are no longer needed or they have a different purpose:

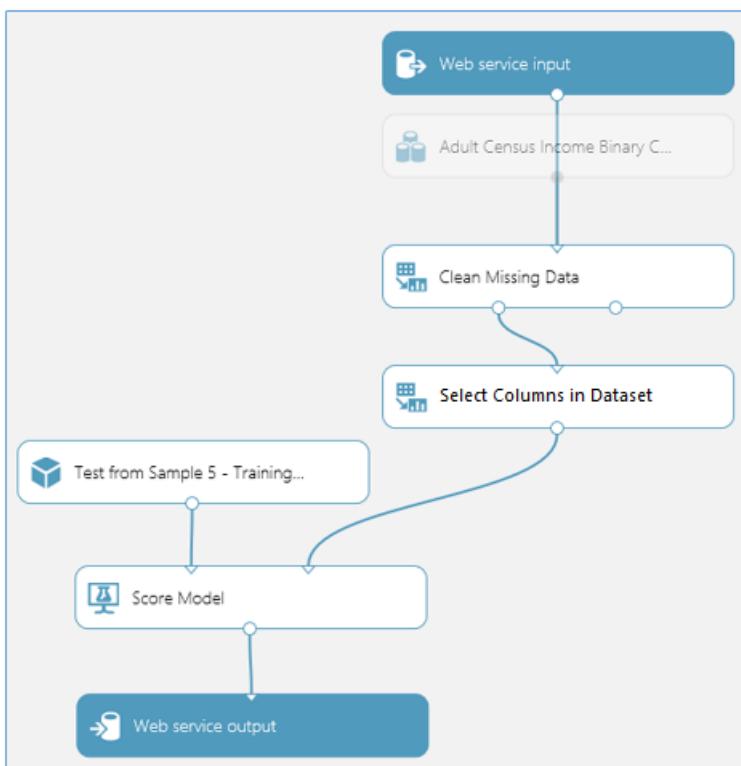
- **Data** - The data in this sample dataset is not used during scoring - the user of the web service will supply the data to be scored. However, the metadata from this dataset, such as data types, is used by the trained model. So you need to keep the dataset in the predictive experiment so that it can provide this metadata.
- **Prep** - Depending on the data that will be submitted for scoring, these modules may or may not be necessary to process the incoming data.

For instance, in this example the sample dataset may have missing values and it includes columns that are

not needed to train the model. So a [Clean Missing Data](#) module was included to deal with missing values, and a [Select Columns in Dataset](#) module was included to exclude those extra columns from the data flow. If you know that the data that will be submitted for scoring through the web service will not have missing values, then you can remove the [Clean Missing Data](#) module. However, since the [Select Columns in Dataset](#) module helps define the set of features being scored, that module needs to remain.

- **Train** - These modules are used to train the model. When you click **Set Up Web Service**, these modules are replaced with a single trained model module. This new module is saved in the **Trained Models** section of the module palette.
- **Score** - In this example, the Split module is used to divide the data stream into a set of test data and training data. In the predictive experiment this isn't needed and can be removed. Similarly, the 2nd [Score Model](#) module and the [Evaluate Model](#) module are used to compare results from the test data, so these modules are also not needed in the predictive experiment. The remaining [Score Model](#) module, however, is needed to return a score result through the web service.

Here is how our example looks after clicking **Set Up Web Service**:

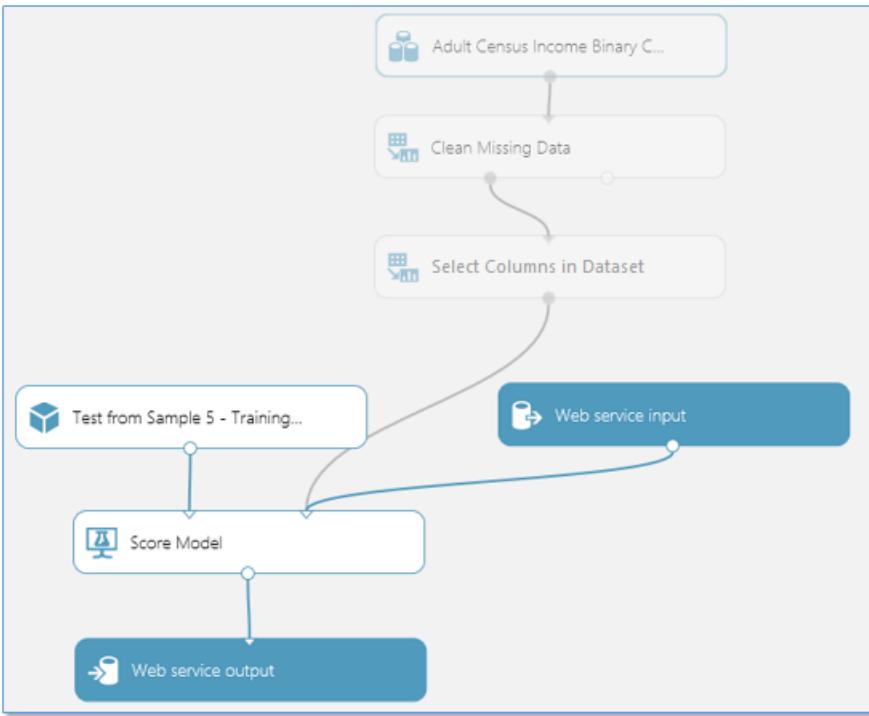


This may be sufficient to prepare your experiment to be deployed as a web service. However, you may want to do some additional work specific to your experiment.

Adjust input and output modules

In your training experiment, you used a set of training data and then did some processing to get the data in a form that the machine learning algorithm needed. If the data you expect to receive through the web service will not need this processing, you can move the **Web service input module** to a different node in your experiment (click the output of the **Web service input module** and drag it to the input port of the appropriate module).

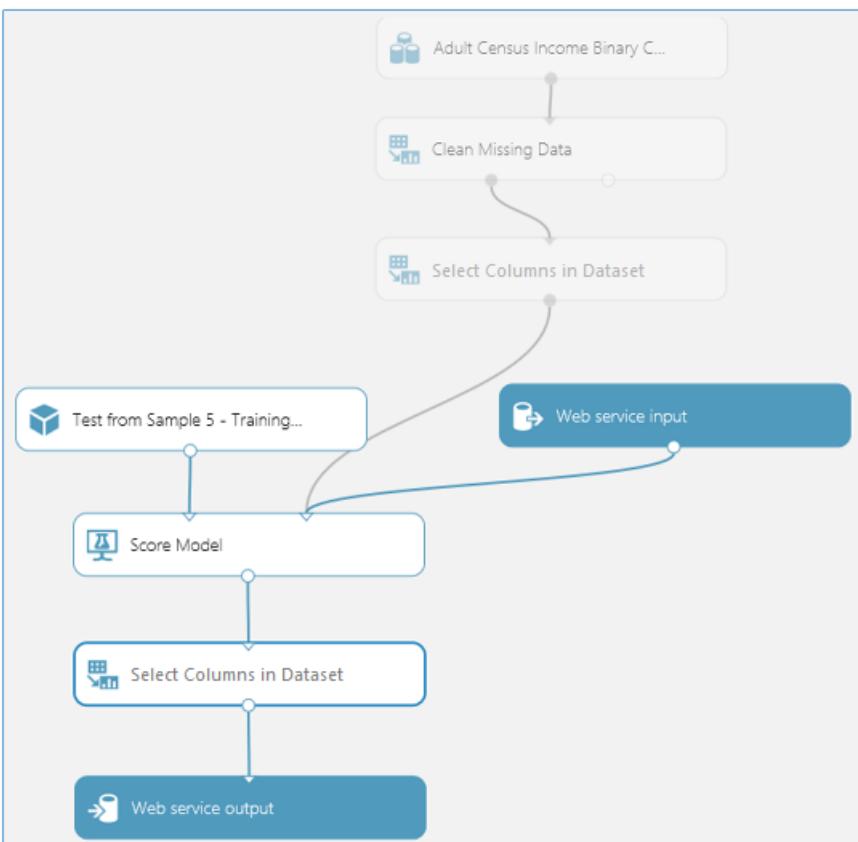
For example, by default **Set Up Web Service** puts the **Web service input** module at the top of your data flow, as in the figure above. However, if the input data will not need this processing, then you can manually position the **Web service input** past the data processing modules:



The input data provided through the web service will now pass directly into the Score Model module without any preprocessing.

Similarly, by default **Set Up Web Service** puts the Web service output module at the bottom of your data flow. In this example, the web service will return to the user the output of the **Score Model** module which includes the complete input data vector plus the scoring results.

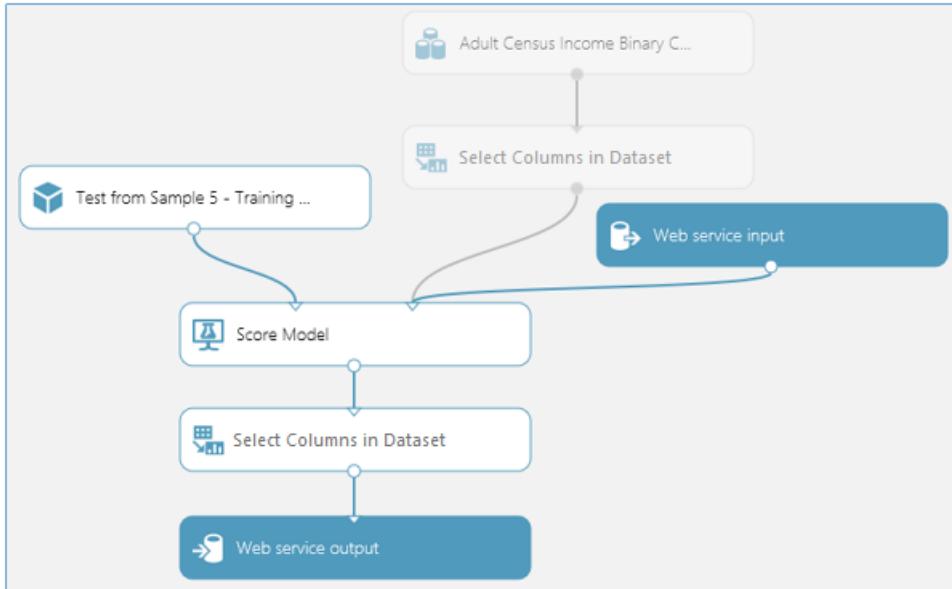
However, if you would prefer to return something different, then you can add additional modules before the **Web service output** module. For example, if you want to return only the scoring results and not the entire vector of input data, then you can add a **Select Columns in Dataset** module to exclude all columns except the scoring results. You then move the **Web service output** module to the output of the **Select Columns in Dataset** module. The experiment would then look like this:



Add or remove additional data processing modules

If there are more modules in your experiment that you know will not be needed during scoring, these can be removed. For example, because we have moved the **Web service input** module to a point after the data processing modules, we can remove the [Clean Missing Data](#) module from the predictive experiment.

Our predictive experiment now looks like this:



TIP

Notice that we didn't remove the dataset or the [Select Columns in Dataset](#) module. The model in the web service won't use the data in the original dataset, but it does use the metadata defined in the dataset, such as the data type of each column. Similarly, the [Select Columns in Dataset](#) module identifies what columns of data will be used by the model. So both of these modules need to remain in the predictive experiment.

Add optional Web Service Parameters

In some cases, you may want to allow the user of your web service to change the behavior of modules when the service is accessed. *Web Service Parameters* allow you to do this.

A common example is setting up an [Import Data](#) module so that the user of the deployed web service can specify a different data source when the web service is accessed. Or configuring an [Export Data](#) module so that a different destination can be specified.

You can define Web Service Parameters and associate them with one or more module parameters, and you can specify whether they are required or optional. The user of the web service can then provide values for these parameters when the service is accessed, and the module actions will be modified accordingly.

For more information about Web Service Parameters, see [Using Azure Machine Learning Web Service Parameters](#).

Deploy the predictive experiment as a web service

Now that the predictive experiment has been sufficiently prepared, you can deploy it as an Azure web service. Using the web service, users can send data to your model and the model will return its predictions.

For more information on the complete deployment process, see [Deploy an Azure Machine Learning web service](#)

Manage experiment iterations in Azure Machine Learning Studio

1/17/2017 • 4 min to read • [Edit on GitHub](#)

Developing a predictive analysis model is an iterative process - as you modify the various functions and parameters of your experiment, your results converge until you are satisfied that you have a trained, effective model. Key to this process is tracking the various iterations of your experiment parameters and configurations.

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

You can review previous runs of your experiments at any time in order to challenge, revisit, and ultimately either confirm or refine previous assumptions. When you run an experiment, Machine Learning Studio keeps a history of the run, including dataset, module, and port connections and parameters. This history also captures results, runtime information such as start and stop times, log messages, and execution status. You can look back at any of these runs at any time to review the chronology of your experiment and intermediate results. You can even use a previous run of your experiment to launch into a new phase of inquiry and discovery on your path to creating simple, complex, or even ensemble modeling solutions.

NOTE

When you view a previous run of an experiment, that version of the experiment is locked and can't be edited. You can, however, save a copy of it by clicking **SAVE AS** and providing a new name for the copy. Machine Learning Studio opens the new copy, which you can then edit and run. This copy of your experiment is available in the **EXPERIMENTS** list along with all your other experiments.

Viewing the Prior Run

When you have an experiment open that you have run at least once, you can view the preceding run of the experiment by clicking **Prior Run** in the properties pane.

For example, suppose you create an experiment and run versions of it at 11:23, 11:42, and 11:55. If you open the last run of the experiment (11:55) and click **Prior Run**, the version ran at 11:42 is opened.

Viewing the Run History

You can view all the previous runs of an experiment by clicking **View Run History** in an open experiment.

For example, suppose you create an experiment with the [Linear Regression](#) module and you want to observe the effect of changing the value of **Learning rate** on your experiment results. You run the experiment multiple times with different values for this parameter, as follows:

LEARNING RATE VALUE	RUN START TIME
0.1	9/11/2014 4:18:58 pm

LEARNING RATE VALUE	RUN START TIME
0.2	9/11/2014 4:24:33 pm
0.4	9/11/2014 4:28:36 pm
0.5	9/11/2014 4:33:31 pm

If you click **VIEW RUN HISTORY**, you see a list of all these runs:

example experiment					
NAME	STATE	STATUS	START TIME	END TIME	
Example Experiment	Editable	Finished	9/11/2014 4:32:58 PM	9/11/2014 4:33:31 PM	
Example Experiment	Locked	Finished	9/11/2014 4:32:58 PM	9/11/2014 4:33:31 PM	
Example Experiment	Locked	Finished	9/11/2014 4:28:04 PM	9/11/2014 4:28:36 PM	
Example Experiment	Locked	Finished	9/11/2014 4:24:05 PM	9/11/2014 4:24:33 PM	
Example Experiment	Locked	Finished	9/11/2014 4:18:10 PM	9/11/2014 4:18:58 PM	

Click any of these runs to view a snapshot of the experiment at the time you ran it. The configuration, parameter values, comments, and results are all preserved to give you a complete record of that run of your experiment.

TIP

To document your iterations of the experiment, you can modify the title each time you run it, you can update the **Summary** of the experiment in the properties pane, and you can add or update comments on individual modules to record your changes. The title, summary, and module comments are saved with each run of the experiment.

The list of experiments in the **EXPERIMENTS** tab in Machine Learning Studio always displays the latest version of an experiment. If you open a previous run of the experiment (using **Prior Run** or **VIEW RUN HISTORY**), you can return to the draft version by clicking **VIEW RUN HISTORY** and selecting the iteration that has a **STATE** of **Editable**.

Iterating on a Previous Run

When you click **Prior Run** or **VIEW RUN HISTORY** and open a previous run, you can view a finished experiment in read-only mode.

If you want to begin an iteration of your experiment starting with the way you configured it for a previous run, you can do this by opening the run and clicking **SAVE AS**. This creates a new experiment, with a new title, an empty run history, and all the components and parameter values of the previous run. This new experiment is listed in the **EXPERIMENTS** tab in the Machine Learning Studio home page, and you can modify and run it, initiating a new run history for this iteration of your experiment.

For example, suppose you have the experiment run history shown in the previous section. You want to observe what happens when you set the **Learning rate** parameter to 0.4, and try different values for the **Number of training epochs** parameter.

1. Click **VIEW RUN HISTORY** and open the iteration of the experiment that you ran at 4:28:36 pm (in which you set the parameter value to 0.4).
2. Click **SAVE AS**.

3. Enter a new title and click the **OK** checkmark. A new copy of the experiment is created.
4. Modify the **Number of training epochs** parameter.
5. Click **RUN**.

You can now continue to modify and run this version of your experiment, building a new run history to record your work.

Create many Machine Learning models and web service endpoints from one experiment using PowerShell

1/17/2017 • 9 min to read • [Edit on GitHub](#)

Here's a common machine learning problem: You want to create many models that have the same training workflow and use the same algorithm, but have different training datasets as input. This article shows you how to do this at scale in Azure Machine Learning Studio using just a single experiment.

For example, let's say you own a global bike rental franchise business. You want to build a regression model to predict the rental demand based on historic data. You have 1,000 rental locations across the world and you've collected a dataset for each location that includes important features such as date, time, weather, and traffic that are specific to each location.

You could train your model once using a merged version of all the datasets across all locations. But because each of your locations has a unique environment, a better approach would be to train your regression model separately using the dataset for each location. That way, each trained model could take into account the different store sizes, volume, geography, population, bike-friendly traffic environment, etc..

That may be the best approach, but you don't want to create 1,000 training experiments in Azure Machine Learning with each one representing a unique location. Besides being an overwhelming task, it's also seems pretty inefficient since each experiment would have all the same components except for the training dataset.

Fortunately, we can accomplish this by using the [Azure Machine Learning retraining API](#) and automating the task with [Azure Machine Learning PowerShell](#).

NOTE

To make our sample run faster, we'll reduce the number of locations from 1,000 to 10. But the same principles and procedures apply to 1,000 locations. The only difference is that if you want to train from 1,000 datasets you probably want to think of running the following PowerShell scripts in parallel. How to do that is beyond the scope of this article, but you can find examples of PowerShell multi-threading on the Internet.

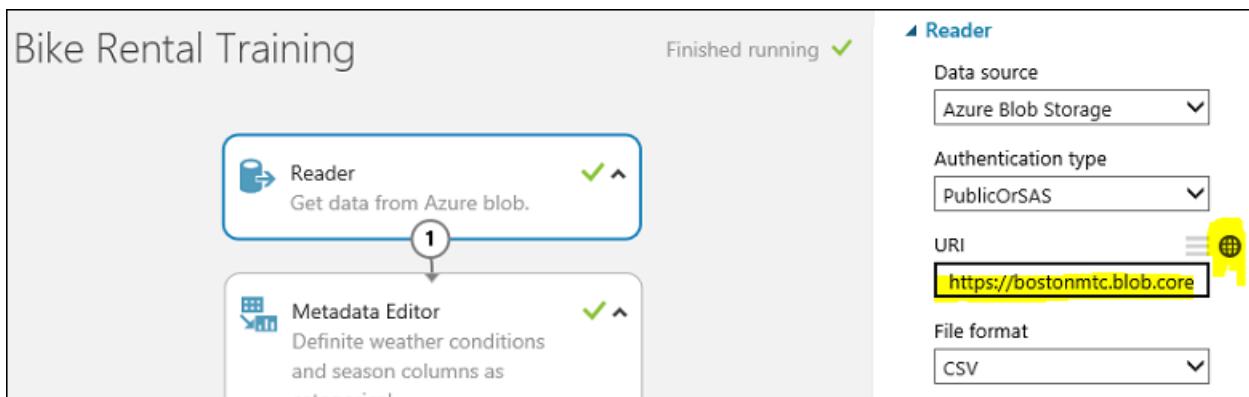
Set up the training experiment

We're going to use an example [training experiment](#) that we've already created in the [Cortana Intelligence Gallery](#). Open this experiment in your [Azure Machine Learning Studio](#) workspace.

NOTE

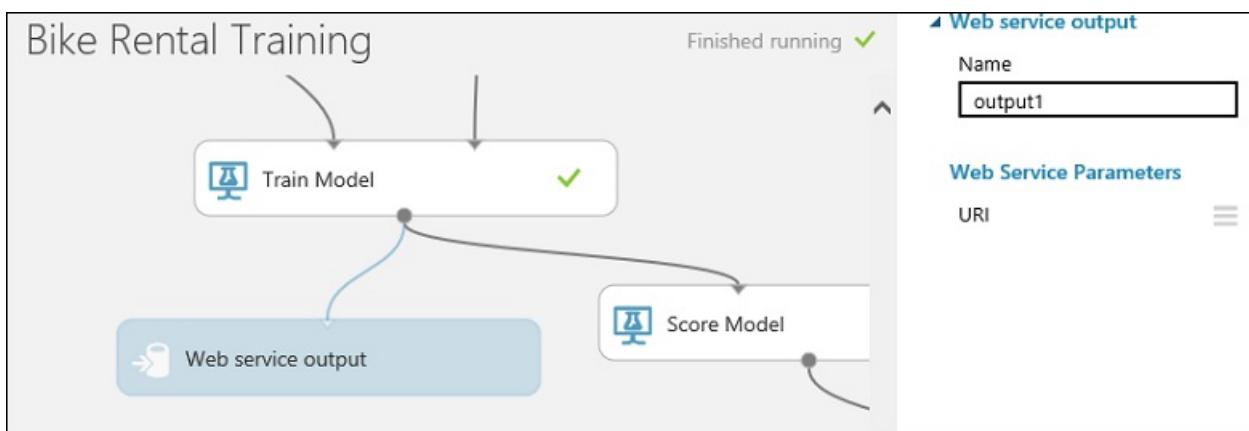
In order to follow along with this example, you may want to use a standard workspace rather than a free workspace. We'll be creating one endpoint for each customer - for a total of 10 endpoints - and that will require a standard workspace since a free workspace is limited to 3 endpoints. If you only have a free workspace, just modify the scripts below to allow for only 3 locations.

The experiment uses an **Import Data** module to import the training dataset *customer001.csv* from an Azure storage account. Let's assume we have collected training datasets from all bike rental locations and stored them in the same blob storage location with file names ranging from *rentalloc001.csv* to *rentalloc10.csv*.



Note that a **Web Service Output** module has been added to the **Train Model** module. When this experiment is deployed as a web service, the endpoint associated with that output will return the trained model in the format of a .ilearn file.

Also note that we set up a web service parameter for the URL that the **Import Data** module uses. This allows us to use the parameter to specify individual training datasets to train the model for each location. There are other ways we could have done this, such as using a SQL query with a web service parameter to get data from a SQL Azure database, or simply using a **Web Service Input** module to pass in a dataset to the web service.



Now, let's run this training experiment using the default value *rental001.csv* as the training dataset. If you view the output of the **Evaluate** module (click the output and select **Visualize**), you can see we get a decent performance of $AUC = 0.91$. At this point, we're ready to deploy a web service out of this training experiment.

Deploy the training and scoring web services

To deploy the training web service, click the **Set Up Web Service** button below the experiment canvas and select **Deploy Web Service**. Call this web service "'Bike Rental Training'".

Now we need to deploy the scoring web service. To do this, we can click **Set Up Web Service** below the canvas and select **Predictive Web Service**. This creates a scoring experiment. We'll need to make a few minor adjustments to make it work as a web service, such as removing the label column "cnt" from the input data and limiting the output to only the instance id and the corresponding predicted value.

To save yourself that work, you can simply open the [predictive experiment](#) in the Gallery that's already been prepared.

To deploy the web service, run the predictive experiment, then click the **Deploy Web Service** button below the canvas. Name the scoring web service "Bike Rental Scoring"".

Create 10 identical web service endpoints with PowerShell

This web service comes with a default endpoint. But we're not as interested in the default endpoint since it can't be updated. What we need to do is to create 10 additional endpoints, one for each location. We'll do this with

PowerShell.

First, we set up our PowerShell environment:

```
Import-Module .\AzureMLPS.dll
# Assume the default configuration file exists and is properly set to point to the valid Workspace.
$scoringSvc = Get-AmlWebService | where Name -eq 'Bike Rental Scoring'
$trainingSvc = Get-AmlWebService | where Name -eq 'Bike Rental Training'
```

Then, run the following PowerShell command:

```
# Create 10 endpoints on the scoring web service.
For ($i = 1; $i -le 10; $i++){
    $seq = $i.ToString().PadLeft(3, '0');
    $endpointName = 'rentalloc' + $seq;
    Write-Host ('adding endpoint ' + $endpointName + '...')
    Add-AmlWebServiceEndpoint -WebServiceId $scoringSvc.Id -EndpointName $endpointName -Description $endpointName
}
```

Now we've created 10 endpoints and they all contain the same trained model trained on *customer001.csv*. You can view them in the Azure Management Portal.

endpoints				
Name	Version	Update Available	Max Con...	🔍
default ➔	2/11/2016 12:34:...	false	20	
rentalloc002	2/11/2016 12:34:...	false	4	
rentalloc001	2/11/2016 12:34:...	false	4	
rentalloc003	2/11/2016 12:34:...	false	4	
rentalloc004	2/11/2016 12:34:...	false	4	
rentalloc005	2/11/2016 12:34:...	false	4	
rentalloc006	2/11/2016 12:34:...	false	4	
rentalloc007	2/11/2016 12:34:...	false	4	
rentalloc008	2/11/2016 12:34:...	false	4	
rentalloc009	2/11/2016 12:34:...	false	4	
rentalloc010	2/11/2016 12:34:...	false	4	

Update the endpoints to use separate training datasets using PowerShell

The next step is to update the endpoints with models uniquely trained on each customer's individual data. But first we need to produce these models from the **Bike Rental Training** web service. Let's go back to the **Bike Rental Training** web service. We need to call its BES endpoint 10 times with 10 different training datasets in order to produce 10 different models. We'll use the **InvokeAmlWebServiceBESEndpoint** PowerShell cmdlet to do this.

You will also need to provide credentials for your blob storage account into `$configContent`, namely, at the fields `AccountName`, `AccountKey` and `RelativeLocation`. The `AccountName` can be one of your account names, as seen in the **Classic Azure Management Portal** (*Storage* tab). Once you click on a storage account, its `AccountKey` can be found by pressing the **Manage Access Keys** button at the bottom and copying the *Primary Access Key*. The `RelativeLocation` is the path relative to your storage where a new model will be stored. For instance, the path `hai/retrain/bike_rental/` in the script below points to a container named `hai/`, and `/retrain/bike_rental/` are subfolders.

Currently, you cannot create subfolders through the portal UI, but there are [several Azure Storage Explorers](#) that allow you to do so. It is recommended that you create a new container in your storage to store the new trained models (.ilearner files) as follows: from your storage page, click on the **Add** button at the bottom and name it `retrain`. In summary, the necessary changes to the script below pertain to `AccountName`, `AccountKey` and `RelativeLocation` (`['retrain/model'+$seq+'.ilearner']`).

```
# Invoke the retraining API 10 times
# This is the default (and the only) endpoint on the training web service
$trainingSvcEp = (Get-AmlWebServiceEndpoint -WebServiceId $trainingSvc.Id)[0];
$submitJobRequestUrl = $trainingSvcEp.ApiLocation + '/jobs?api-version=2.0';
$apiKey = $trainingSvcEp.PrimaryKey;
For ($i = 1; $i -le 10; $i++){
    $seq = $i.ToString().PadLeft(3, '0');
    $inputFileName = "https://bostonmtc.blob.core.windows.net/ai/retrain/bike_rental/BikeRental'$seq'.csv";
    $configContent = '{ "GlobalParameters": { "URI": "' + $inputFileName + '" }, "Outputs": { "output1": { "ConnectionString": "DefaultEndpointsProtocol=https;AccountName=<myaccount>;AccountKey=<mykey>", "RelativeLocation": "ai/retrain/bike_rental/model'$seq'.ilearner" } } }';
    Write-Host ('Training regression model on ' + $inputFileName + ' for rental location ' + $seq + '... ');
    Invoke-AmlWebServiceBESEndpoint -JobConfigString $configContent -SubmitJobRequestUrl $submitJobRequestUrl -ApiKey $apiKey
}
```

NOTE

The BES endpoint is the only supported mode for this operation. RRS cannot be used for producing trained models.

As you can see above, instead of constructing 10 different BES job configuration json files, we dynamically create the config string instead and feed it to the `jobConfigString` parameter of the **InvokeAmlWebServiceBESEndpoint** cmdlet, since there is really no need to keep a copy on disk.

If everything goes well, after a while you should see 10 .ilearner files, from `model001.ilearner` to `model010.ilearner`, in your Azure storage account. Now we're ready to update our 10 scoring web service endpoints with these models using the **Patch-AmlWebServiceEndpoint** PowerShell cmdlet. Remember again that we can only patch the non-default endpoints we programmatically created earlier.

```
# Patch the 10 endpoints with respective .ilearner models
$baseLoc = 'http://bostonmtc.blob.core.windows.net/'
$sasToken = '<my_blob_sas_token>'
For ($i = 1; $i -le 10; $i++){
    $seq = $i.ToString().PadLeft(3, '0');
    $endpointName = 'rentalloc' + $seq;
    $relativeLoc = 'ai/retrain/bike_rental/model'$seq'.ilearner';
    Write-Host ('Patching endpoint ' + $endpointName + '... ');
    Patch-AmlWebServiceEndpoint -WebServiceId $scoringSvc.Id -EndpointName $endpointName -ResourceName 'Bike Rental [trained model]' -BaseLocation $baseLoc -RelativeLocation $relativeLoc -SasBlobToken $sasToken
}
```

This should run fairly quickly. When the execution finishes, we'll have successfully created 10 predictive web service endpoints, each containing a trained model uniquely trained on the dataset specific to a rental location, all from a single training experiment. To verify this, you can try calling these endpoints using the **InvokeAmlWebServiceRRSEndpoint** cmdlet, providing them with the same input data, and you should expect to see different prediction results since the models are trained with different training sets.

Full PowerShell script

Here's the listing of the full source code:

```

Import-Module .\AzureMLPS.dll
# Assume the default configuration file exists and properly set to point to the valid workspace.
$scoringSvc = Get-AmlWebService | where Name -eq 'Bike Rental Scoring'
$trainingSvc = Get-AmlWebService | where Name -eq 'Bike Rental Training'

# Create 10 endpoints on the scoring web service
For ($i = 1; $i -le 10; $i++) {
    $seq = $i.ToString().PadLeft(3, '0');
    $endpointName = 'rentalloc' + $seq;
    Write-Host ('adding endpoint ' + $endpointName + '...')
    Add-AmlWebServiceEndpoint -WebServiceId $scoringSvc.Id -EndpointName $endpointName -Description $endpointName
}

# Invoke the retraining API 10 times to produce 10 regression models in .ilearnr format
$trainingSvcEp = (Get-AmlWebServiceEndpoint -WebServiceId $trainingSvc.Id)[0];
$submitJobRequestUrl = $trainingSvcEp.ApiLocation + '/jobs?api-version=2.0';
$apiKey = $trainingSvcEp.PrimaryKey;
For ($i = 1; $i -le 10; $i++) {
    $seq = $i.ToString().PadLeft(3, '0');
    $inputFileName = "https://bostonmtc.blob.core.windows.net/hai/retrain/bike_rental/BikeRental" + $seq + ".csv";
    $configContent = "{ \"GlobalParameters\": { \"URI\": \"" + $inputFileName + "\" }, \"Outputs\": { \"output1\": { \"ConnectionString\": \"DefaultEndpointsProtocol=https;AccountName=<myaccount>;AccountKey=<mykey>\", \"RelativeLocation\": \"hai/retrain/bike_rental/model\" + $seq + '.ilearnr' } } }";
    Write-Host ('training regression model on ' + $inputFileName + ' for rental location ' + $seq + '...');
    Invoke-AmlWebServiceBESEndpoint -JobConfigString $configContent -SubmitJobRequestUrl $submitJobRequestUrl -ApiKey $apiKey
}

# Patch the 10 endpoints with respective .ilearnr models
$baseLoc = 'http://bostonmtc.blob.core.windows.net/'
$sasToken = '?test'
For ($i = 1; $i -le 10; $i++) {
    $seq = $i.ToString().PadLeft(3, '0');
    $endpointName = 'rentalloc' + $seq;
    $relativeLoc = 'hai/retrain/bike_rental/model' + $seq + '.ilearnr';
    Write-Host ('Patching endpoint ' + $endpointName + '...');
    Patch-AmlWebServiceEndpoint -WebServiceId $scoringSvc.Id -EndpointName $endpointName -ResourceName 'Bike Rental [trained model]' -BaseLocation $baseLoc -RelativeLocation $relativeLoc -SasBlobToken $sasToken
}

```

How to choose algorithms for Microsoft Azure Machine Learning

1/17/2017 • 15 min to read • [Edit on GitHub](#)

The answer to the question "What machine learning algorithm should I use?" is always "It depends." It depends on the size, quality, and nature of the data. It depends on what you want to do with the answer. It depends on how the math of the algorithm was translated into instructions for the computer you are using. And it depends on how much time you have. Even the most experienced data scientists can't tell which algorithm will perform best before trying them.

The Machine Learning Algorithm Cheat Sheet

The **Microsoft Azure Machine Learning Algorithm Cheat Sheet** helps you choose the right machine learning algorithm for your predictive analytics solutions from the Microsoft Azure Machine Learning library of algorithms. This article walks you through how to use it.

NOTE

To download the cheat sheet and follow along with this article, go to [Machine learning algorithm cheat sheet for Microsoft Azure Machine Learning Studio](#).

This cheat sheet has a very specific audience in mind: a beginning data scientist with undergraduate-level machine learning, trying to choose an algorithm to start with in Azure Machine Learning Studio. That means that it makes some generalizations and oversimplifications, but it will point you in a safe direction. It also means that there are lots of algorithms not listed here. As Azure Machine Learning grows to encompass a more complete set of available methods, we'll add them.

These recommendations are compiled feedback and tips from a lot of data scientists and machine learning experts. We didn't agree on everything, but I've tried to harmonize our opinions into a rough consensus. Most of the statements of disagreement begin with "It depends..."

How to use the cheat sheet

Read the path and algorithm labels on the chart as "For *<path label>* use *<algorithm>*." For example, "For *speed* use *two class logistic regression*." Sometimes more than one branch will apply. Sometimes none of them will be a perfect fit. They're intended to be rule-of-thumb recommendations, so don't worry about it being exact. Several data scientists I talked with said that the only sure way to find the very best algorithm is to try all of them.

Here's an example from the [Cortana Intelligence Gallery](#) of an experiment that tries several algorithms against the same data and compares the results: [Compare Multi-class Classifiers: Letter recognition](#).

TIP

To download and print a diagram that gives an overview of the capabilities of Machine Learning Studio, see [Overview diagram of Azure Machine Learning Studio capabilities](#).

Flavors of machine learning

Supervised

Supervised learning algorithms make predictions based on a set of examples. For instance, historical stock prices can be used to hazard guesses at future prices. Each example used for training is labeled with the value of interest—in this case the stock price. A supervised learning algorithm looks for patterns in those value labels. It can use any information that might be relevant—the day of the week, the season, the company's financial data, the type of industry, the presence of disruptive geopolitical events—and each algorithm looks for different types of patterns. After the algorithm has found the best pattern it can, it uses that pattern to make predictions for unlabeled testing data—tomorrow's prices.

This is a popular and useful type of machine learning. With one exception, all of the modules in Azure Machine Learning are supervised learning algorithms. There are several specific types of supervised learning that are represented within Azure Machine Learning: classification, regression, and anomaly detection.

- **Classification.** When the data are being used to predict a category, supervised learning is also called classification. This is the case when assigning an image as a picture of either a 'cat' or a 'dog'. When there are only two choices, this is called **two-class** or **binomial classification**. When there are more categories, as when predicting the winner of the NCAA March Madness tournament, this problem is known as **multi-class classification**.
- **Regression.** When a value is being predicted, as with stock prices, supervised learning is called regression.
- **Anomaly detection.** Sometimes the goal is to identify data points that are simply unusual. In fraud detection, for example, any highly unusual credit card spending patterns are suspect. The possible variations are so numerous and the training examples so few, that it's not feasible to learn what fraudulent activity looks like. The approach that anomaly detection takes is to simply learn what normal activity looks like (using a history of non-fraudulent transactions) and identify anything that is significantly different.

Unsupervised

In unsupervised learning, data points have no labels associated with them. Instead, the goal of an unsupervised learning algorithm is to organize the data in some way or to describe its structure. This can mean grouping it into clusters or finding different ways of looking at complex data so that it appears simpler or more organized.

Reinforcement learning

In reinforcement learning, the algorithm gets to choose an action in response to each data point. The learning algorithm also receives a reward signal a short time later, indicating how good the decision was. Based on this, the algorithm modifies its strategy in order to achieve the highest reward. Currently there are no reinforcement learning algorithm modules in Azure Machine Learning. Reinforcement learning is common in robotics, where the set of sensor readings at one point in time is a data point, and the algorithm must choose the robot's next action. It is also a natural fit for Internet of Things applications.

Considerations when choosing an algorithm

Accuracy

Getting the most accurate answer possible isn't always necessary. Sometimes an approximation is adequate, depending on what you want to use it for. If that's the case, you may be able to cut your processing time dramatically by sticking with more approximate methods. Another advantage of more approximate methods is that they naturally tend to avoid [overfitting](#).

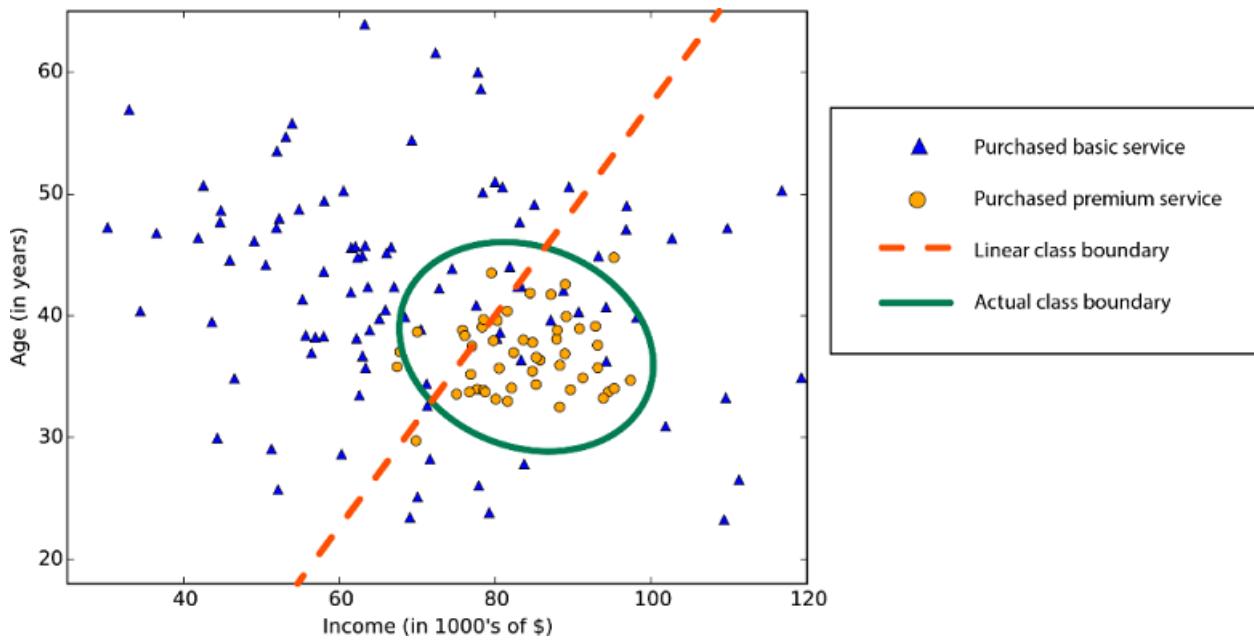
Training time

The number of minutes or hours necessary to train a model varies a great deal between algorithms. Training time is often closely tied to accuracy—one typically accompanies the other. In addition, some algorithms are more sensitive to the number of data points than others. When time is limited it can drive the choice of algorithm, especially when the data set is large.

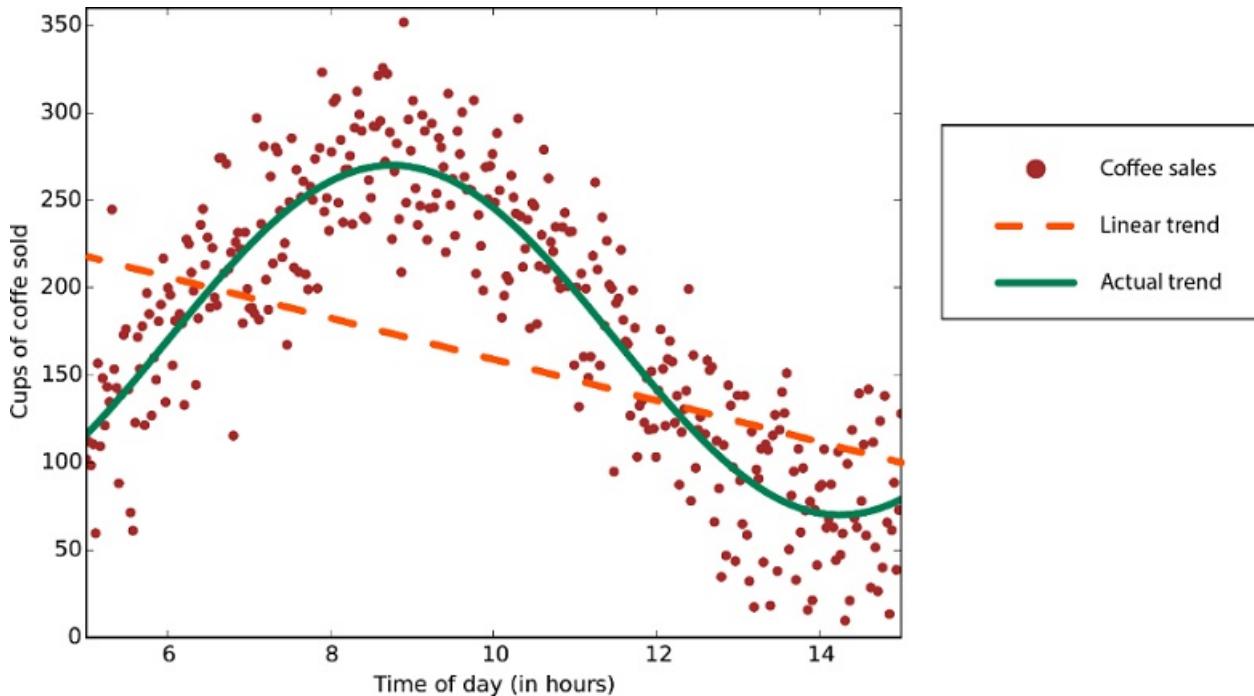
Linearity

Lots of machine learning algorithms make use of linearity. Linear classification algorithms assume that classes

can be separated by a straight line (or its higher-dimensional analog). These include logistic regression and support vector machines (as implemented in Azure Machine Learning). Linear regression algorithms assume that data trends follow a straight line. These assumptions aren't bad for some problems, but on others they bring accuracy down.



Non-linear class boundary - relying on a linear classification algorithm would result in low accuracy



Data with a nonlinear trend - using a linear regression method would generate much larger errors than necessary

Despite their dangers, linear algorithms are very popular as a first line of attack. They tend to be algorithmically simple and fast to train.

Number of parameters

Parameters are the knobs a data scientist gets to turn when setting up an algorithm. They are numbers that affect the algorithm's behavior, such as error tolerance or number of iterations, or options between variants of how the algorithm behaves. The training time and accuracy of the algorithm can sometimes be quite sensitive to getting just the right settings. Typically, algorithms with large numbers of parameters require the most trial and error to find a good combination.

Alternatively, there is a [parameter sweeping](#) module block in Azure Machine Learning that automatically tries all parameter combinations at whatever granularity you choose. While this is a great way to make sure you've spanned the parameter space, the time required to train a model increases exponentially with the number of parameters.

The upside is that having many parameters typically indicates that an algorithm has greater flexibility. It can often achieve very good accuracy. Provided you can find the right combination of parameter settings.

Number of features

For certain types of data, the number of features can be very large compared to the number of data points. This is often the case with genetics or textual data. The large number of features can bog down some learning algorithms, making training time unfeasibly long. Support Vector Machines are particularly well suited to this case (see below).

Special cases

Some learning algorithms make particular assumptions about the structure of the data or the desired results. If you can find one that fits your needs, it can give you more useful results, more accurate predictions, or faster training times.

ALGORITHM	ACCURACY	TRAINING TIME	LINEARITY	PARAMETERS	NOTES
Two-class classification					
logistic regression		●	●	5	
decision forest	●	○		6	
decision jungle	●	○		6	Low memory footprint
boosted decision tree	●	○		6	Large memory footprint
neural network	●			9	Additional customization is possible
averaged perceptron	○	○	●	4	
support vector machine		○	●	5	Good for large feature sets
locally deep support vector machine	○			8	Good for large feature sets
Bayes' point machine		○	●	3	
Multi-class classification					

ALGORITHM	ACCURACY	TRAINING TIME	LINEARITY	PARAMETERS	NOTES
logistic regression		●	●	5	
decision forest	●	○		6	
decision jungle	●	○		6	Low memory footprint
neural network	●			9	Additional customization is possible
one-v-all	-	-	-	-	See properties of the two-class method selected
Regression					
linear		●	●	4	
Bayesian linear		○	●	2	
decision forest	●	○		6	
boosted decision tree	●	○		5	Large memory footprint
fast forest quantile	●	○		9	Distributions rather than point predictions
neural network	●			9	Additional customization is possible
Poisson			●	5	Technically log-linear. For predicting counts
ordinal				0	For predicting rank-ordering
Anomaly detection					
support vector machine	○	○		2	Especially good for large feature sets
PCA-based anomaly detection		○	●	3	

ALGORITHM	ACCURACY	TRAINING TIME	LINEARITY	PARAMETERS	NOTES
K-means	○	○	●	4	A clustering algorithm

Algorithm properties:

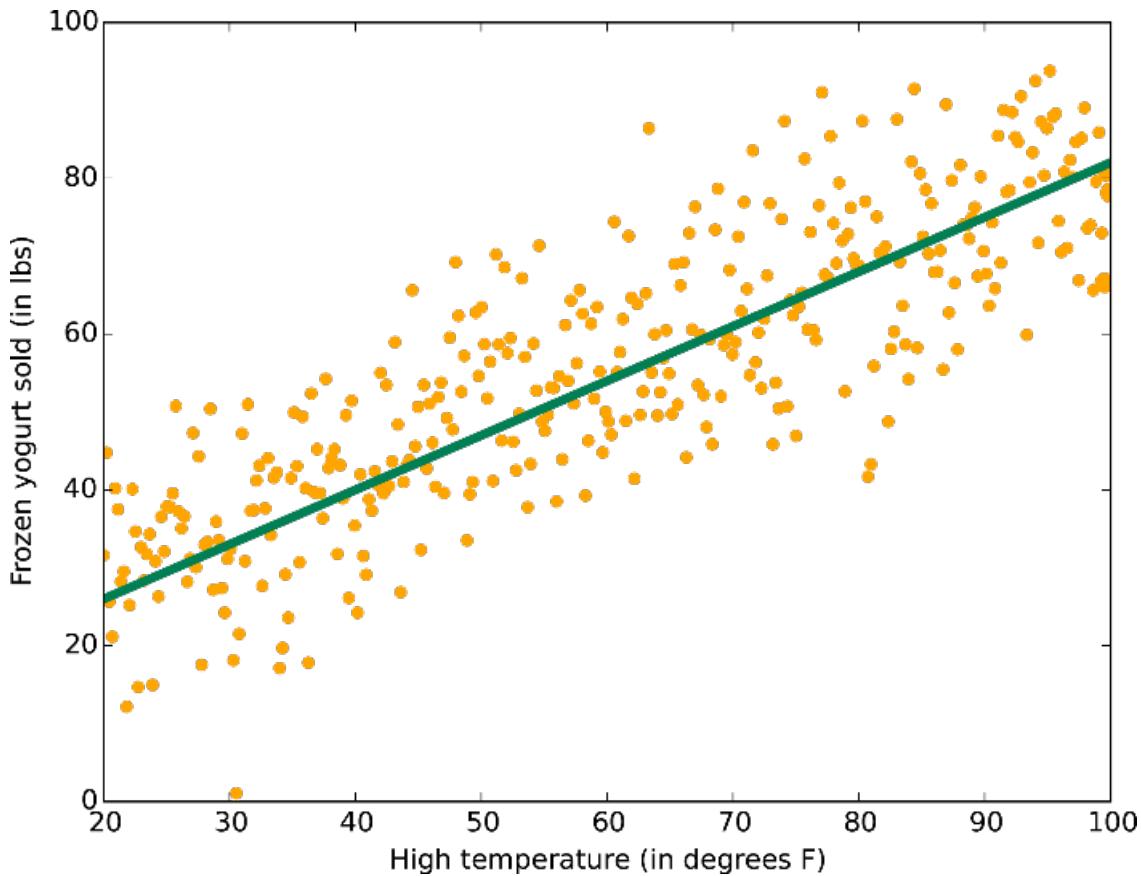
● - shows excellent accuracy, fast training times, and the use of linearity

○ - shows good accuracy and moderate training times

Algorithm notes

Linear regression

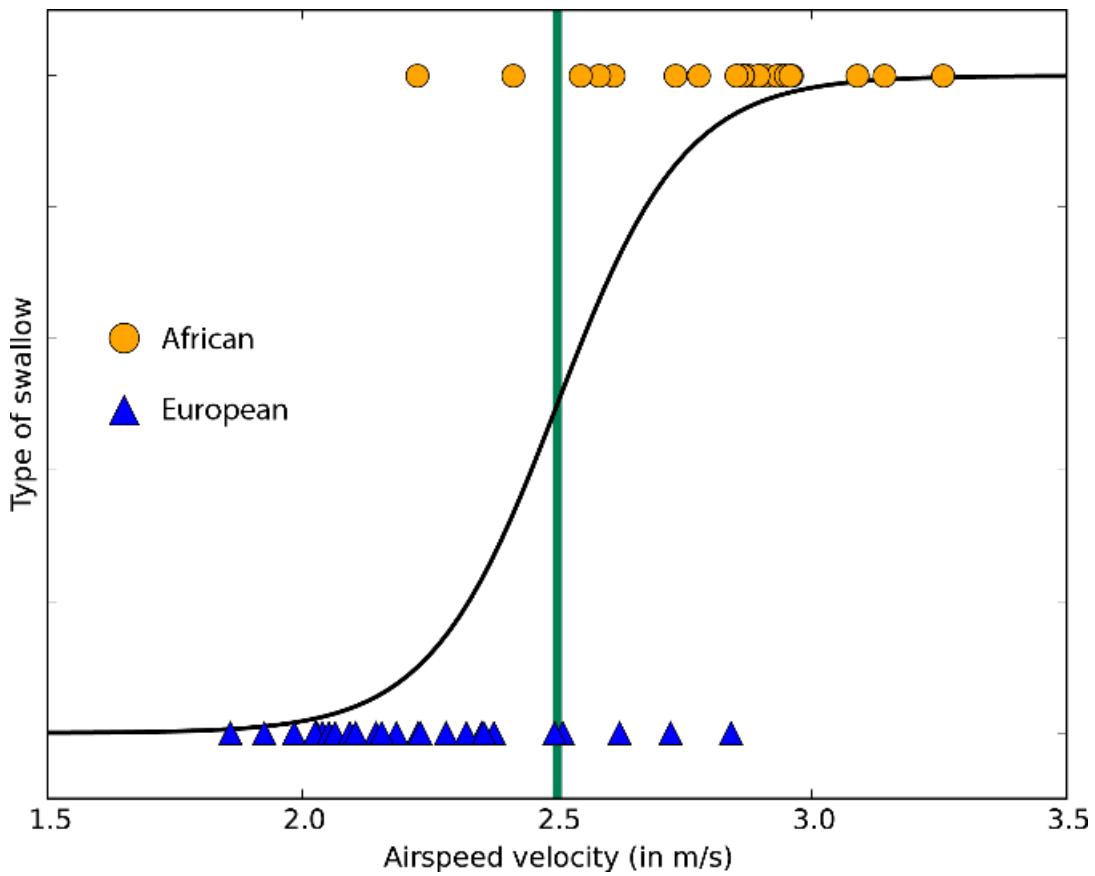
As mentioned previously, [linear regression](#) fits a line (or plane, or hyperplane) to the data set. It's a workhorse, simple and fast, but it may be overly simplistic for some problems. Check here for a [linear regression tutorial](#).



Data with a linear trend

Logistic regression

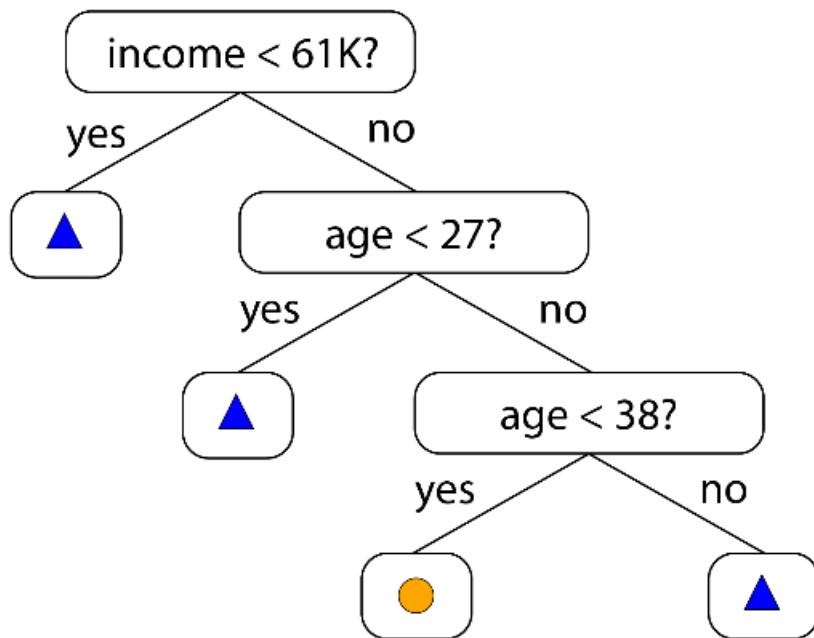
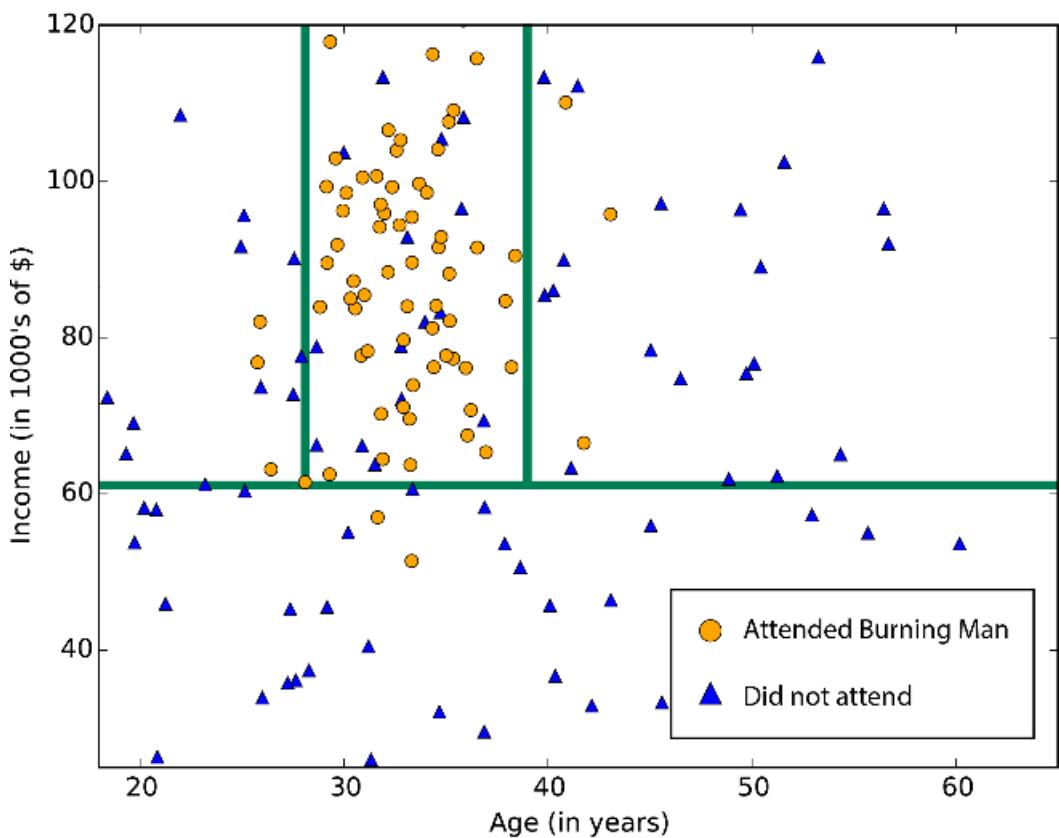
Although it confusingly includes 'regression' in the name, logistic regression is actually a powerful tool for [two-class](#) and [multiclass](#) classification. It's fast and simple. The fact that it uses an 'S'-shaped curve instead of a straight line makes it a natural fit for dividing data into groups. Logistic regression gives linear class boundaries, so when you use it, make sure a linear approximation is something you can live with.



A logistic regression to two-class data with just one feature - the class boundary is the point at which the logistic curve is just as close to both classes

Trees, forests, and jungles

Decision forests ([regression](#), [two-class](#), and [multiclass](#)), decision jungles ([two-class](#) and [multiclass](#)), and boosted decision trees ([regression](#) and [two-class](#)) are all based on decision trees, a foundational machine learning concept. There are many variants of decision trees, but they all do the same thing—subdivide the feature space into regions with mostly the same label. These can be regions of consistent category or of constant value, depending on whether you are doing classification or regression.



A decision tree subdivides a feature space into regions of roughly uniform values

Because a feature space can be subdivided into arbitrarily small regions, it's easy to imagine dividing it finely enough to have one data point per region—an extreme example of overfitting. In order to avoid this, a large set of trees are constructed with special mathematical care taken that the trees are not correlated. The average of this "decision forest" is a tree that avoids overfitting. Decision forests can use a lot of memory. Decision jungles are a variant that consumes less memory at the expense of a slightly longer training time.

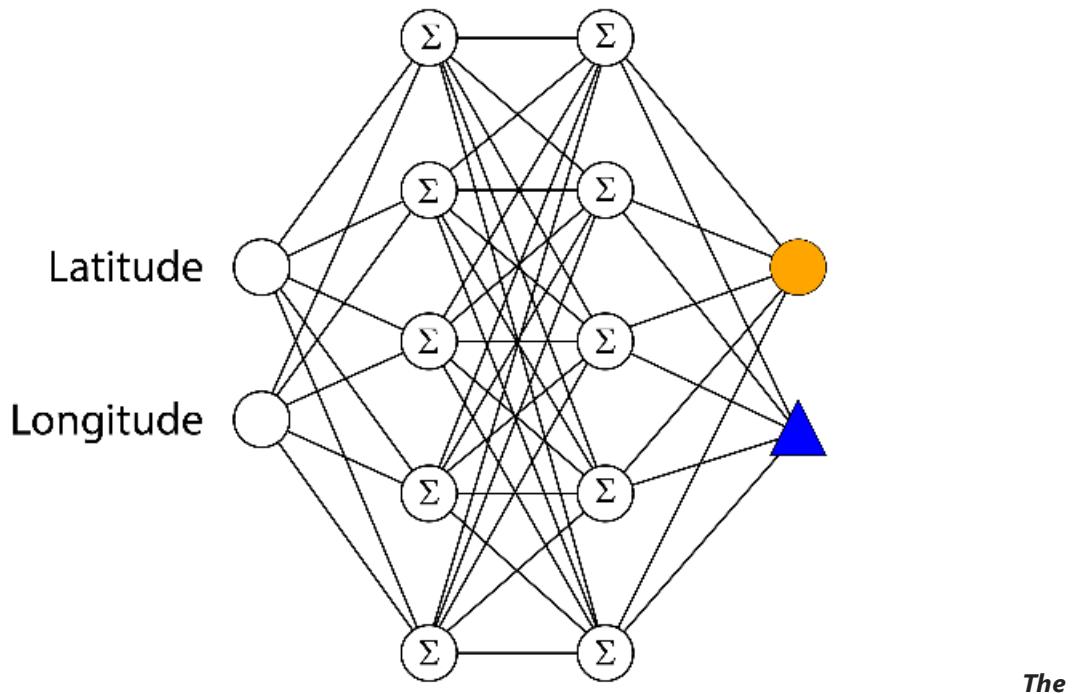
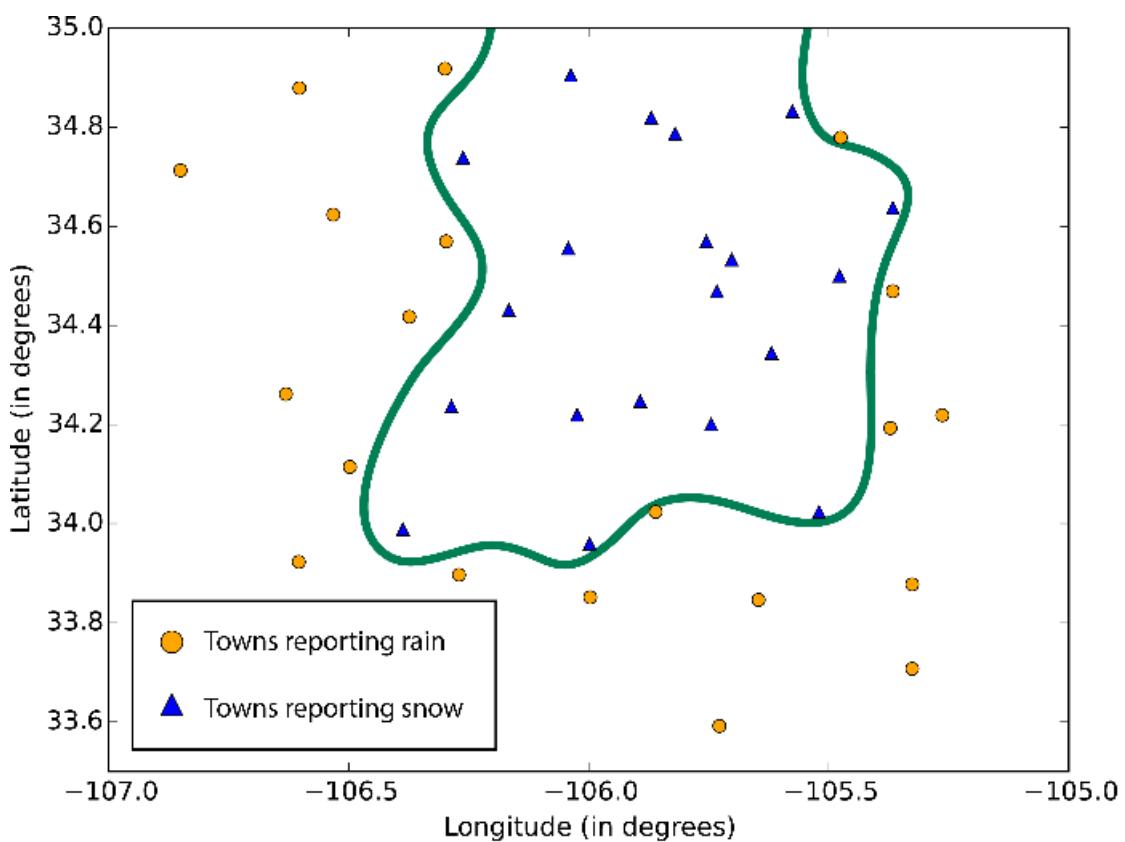
Boosted decision trees avoid overfitting by limiting how many times they can subdivide and how few data points are allowed in each region. The algorithm constructs a sequence of trees, each of which learns to compensate for the error left by the tree before. The result is a very accurate learner that tends to use a lot of memory. For the full technical description, check out [Friedman's original paper](#).

[Fast forest quantile regression](#) is a variation of decision trees for the special case where you want to know not only the typical (median) value of the data within a region, but also its distribution in the form of quantiles.

Neural networks and perceptrons

Neural networks are brain-inspired learning algorithms covering [multiclass](#), [two-class](#), and [regression](#) problems. They come in an infinite variety, but the neural networks within Azure Machine Learning are all of the form of directed acyclic graphs. That means that input features are passed forward (never backward) through a sequence of layers before being turned into outputs. In each layer, inputs are weighted in various combinations, summed, and passed on to the next layer. This combination of simple calculations results in the ability to learn sophisticated class boundaries and data trends, seemingly by magic. Many-layered networks of this sort perform the "deep learning" that fuels so much tech reporting and science fiction.

This high performance doesn't come for free, though. Neural networks can take a long time to train, particularly for large data sets with lots of features. They also have more parameters than most algorithms, which means that parameter sweeping expands the training time a great deal. And for those overachievers who wish to [specify their own network structure](#), the possibilities are inexhaustible.

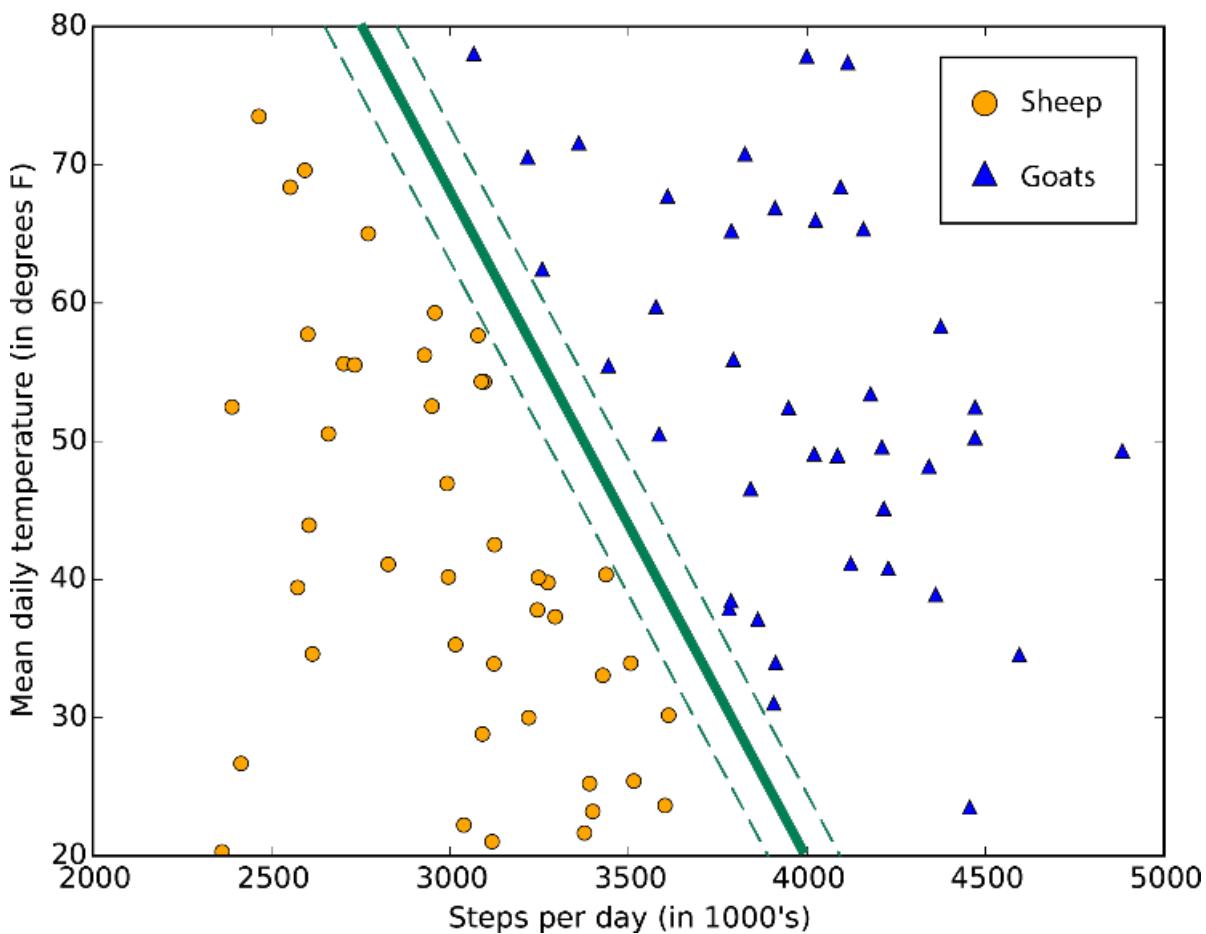


boundaries learned by neural networks can be complex and irregular

The [two-class averaged perceptron](#) is neural networks' answer to skyrocketing training times. It uses a network structure that gives linear class boundaries. It is almost primitive by today's standards, but it has a long history of working robustly and is small enough to learn quickly.

SVMs

Support vector machines (SVMs) find the boundary that separates classes by as wide a margin as possible. When the two classes can't be clearly separated, the algorithms find the best boundary they can. As written in Azure Machine Learning, the [two-class SVM](#) does this with a straight line only. (In SVM-speak, it uses a linear kernel.) Because it makes this linear approximation, it is able to run fairly quickly. Where it really shines is with feature-intense data, like text or genomic. In these cases SVMs are able to separate classes more quickly and with less overfitting than most other algorithms, in addition to requiring only a modest amount of memory.



A typical support vector machine class boundary maximizes the margin separating two classes

Another product of Microsoft Research, the [two-class locally deep SVM](#) is a non-linear variant of SVM that retains most of the speed and memory efficiency of the linear version. It is ideal for cases where the linear approach doesn't give accurate enough answers. The developers kept it fast by breaking the problem down into a bunch of small linear SVM problems. Read the [full description](#) for the details on how they pulled off this trick.

Using a clever extension of nonlinear SVMs, the [one-class SVM](#) draws a boundary that tightly outlines the entire data set. It is useful for anomaly detection. Any new data points that fall far outside that boundary are unusual enough to be noteworthy.

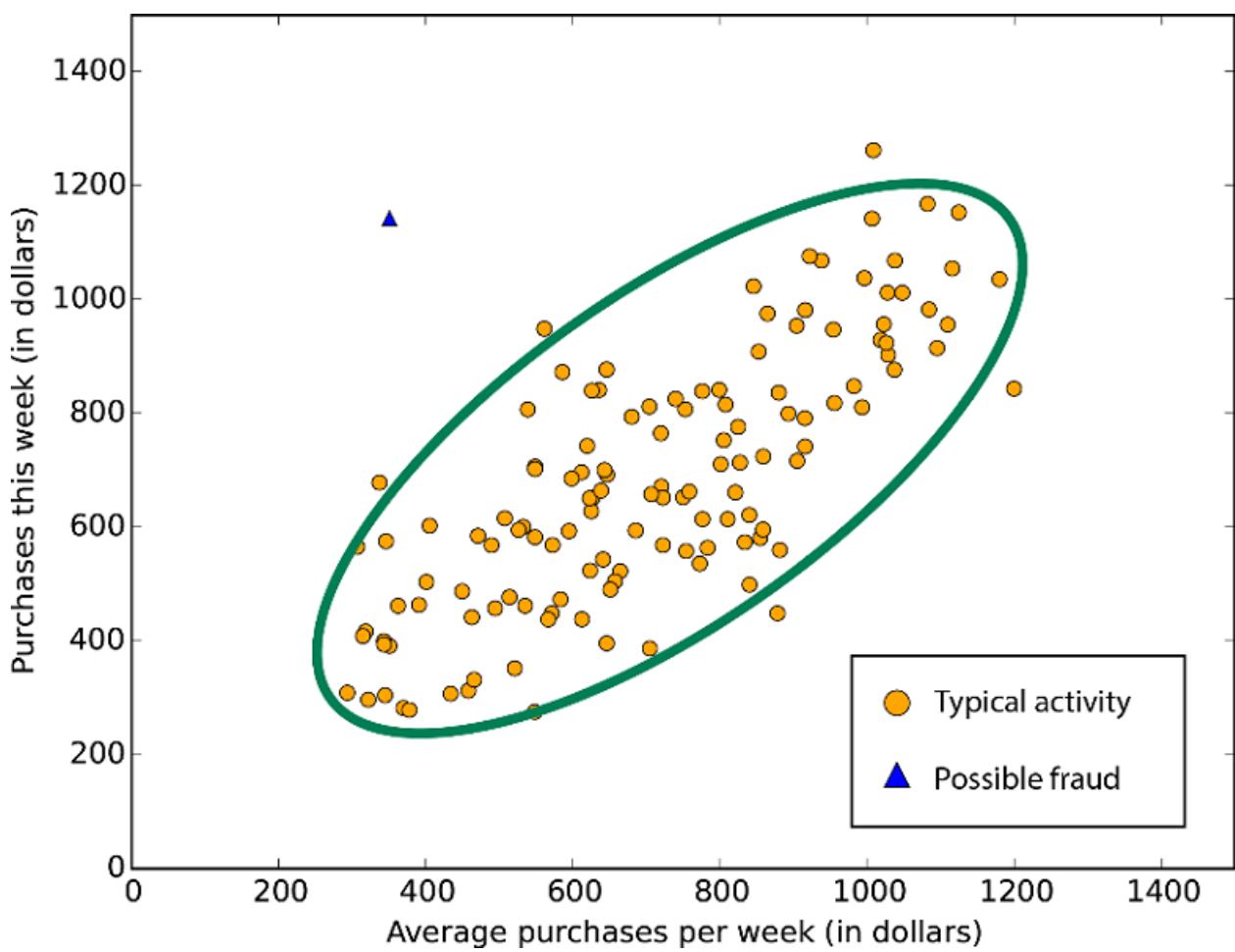
Bayesian methods

Bayesian methods have a highly desirable quality: they avoid overfitting. They do this by making some assumptions beforehand about the likely distribution of the answer. Another byproduct of this approach is that they have very few parameters. Azure Machine Learning has both Bayesian algorithms for both classification ([Two-class Bayes' point machine](#)) and regression ([Bayesian linear regression](#)). Note that these assume that the data can be split or fit with a straight line.

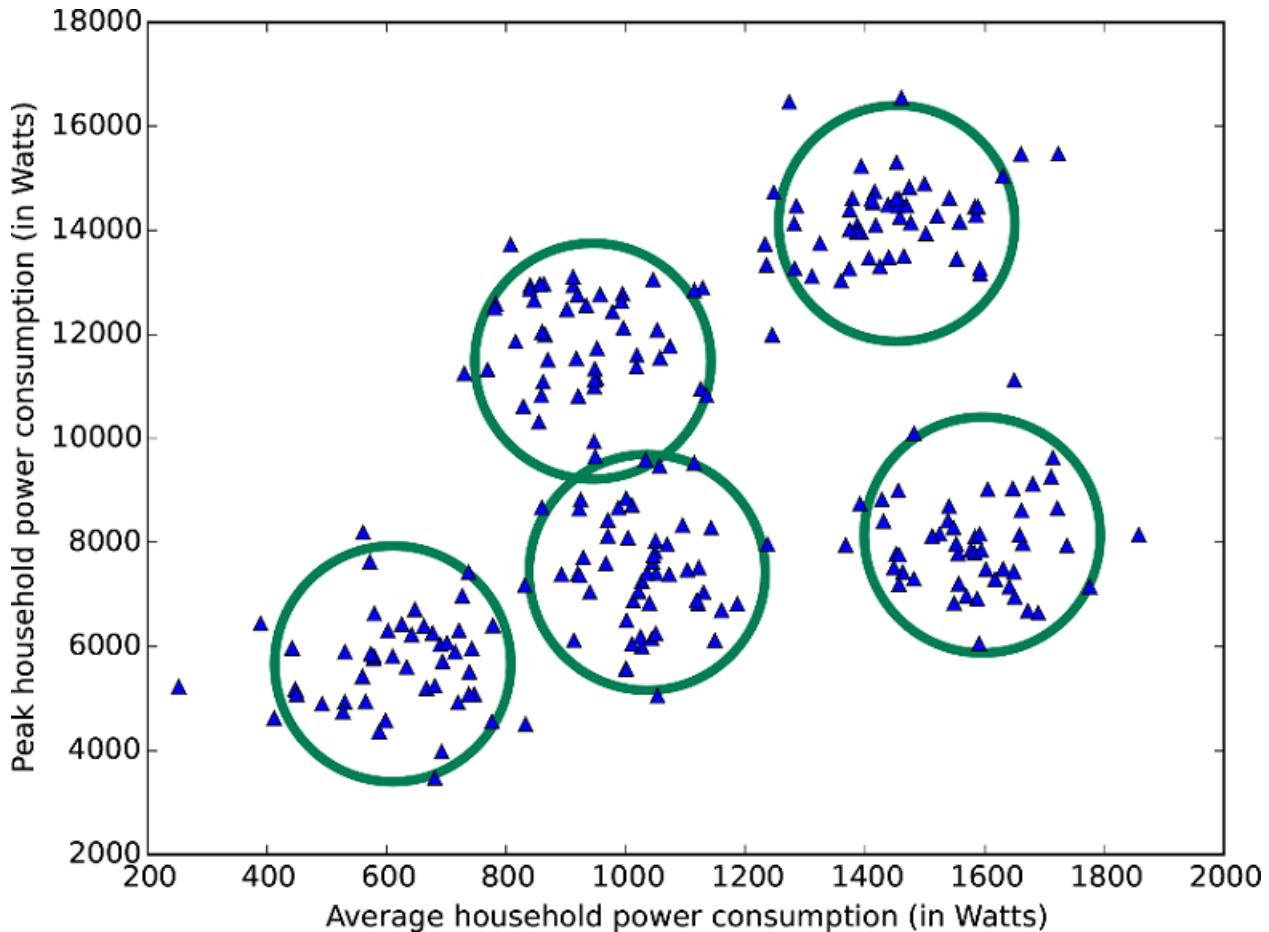
On an historical note, Bayes' point machines were developed at Microsoft Research. They have some exceptionally beautiful theoretical work behind them. The interested student is directed to the [original article in JMLR](#) and an [insightful blog by Chris Bishop](#).

Specialized algorithms

If you have a very specific goal you may be in luck. Within the Azure Machine Learning collection there are algorithms that specialize in rank prediction ([ordinal regression](#)), count prediction ([Poisson regression](#)), and anomaly detection (one based on [principal components analysis](#) and one based on [support vector machines](#)). And there is a lone clustering algorithm as well ([K-means](#)).

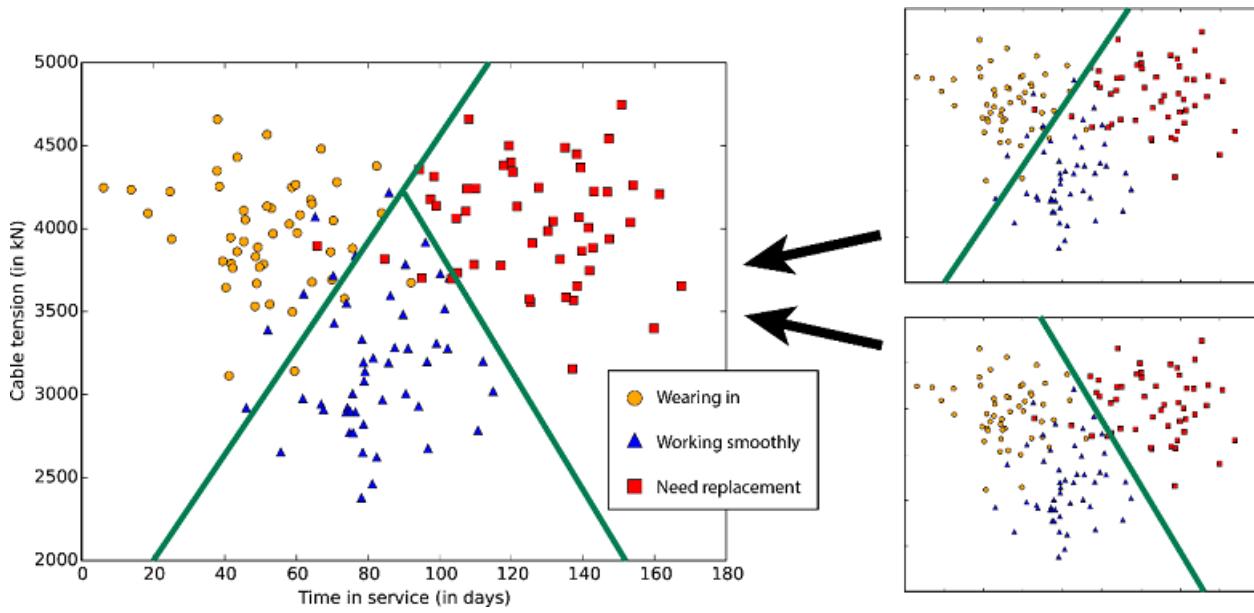


PCA-based anomaly detection - the vast majority of the data falls into a stereotypical distribution; points deviating dramatically from that distribution are suspect



A data set is grouped into 5 clusters using K-means

There is also an ensemble [one-v-all multiclass classifier](#), which breaks the N-class classification problem into N-1 two-class classification problems. The accuracy, training time, and linearity properties are determined by the two-class classifiers used.



A pair of two-class classifiers combine to form a three-class classifier

Azure Machine Learning also includes access to a powerful machine learning framework under the title of [Vowpal Wabbit](#). VW defies categorization here, since it can learn both classification and regression problems and can even learn from partially unlabeled data. You can configure it to use any one of a number of learning algorithms, loss functions, and optimization algorithms. It was designed from the ground up to be efficient, parallel, and extremely fast. It handles ridiculously large feature sets with little apparent effort. Started and led by Microsoft Research's own John Langford, VW is a Formula One entry in a field of stock car algorithms. Not every problem fits VW, but if yours does, it may be worth your while to climb the learning curve on its interface. It's also available as [stand-alone open source code](#) in several languages.

More help with algorithms

- For a downloadable infographic that describes algorithms and provides examples, see [Downloadable Infographic: Machine learning basics with algorithm examples](#).
- For a list by category of all the machine learning algorithms available in Azure Machine Learning Studio, see [Initialize Model][initialize-model] in the Machine Learning Studio Algorithm and Module Help.
- For a complete alphabetical list of algorithms and modules in Azure Machine Learning Studio, see [A-Z list of Machine Learning Studio modules][a-z-list] in Machine Learning Studio Algorithm and Module Help.
- To download and print a diagram that gives an overview of the capabilities of Azure Machine Learning Studio, see [Overview diagram of Azure Machine Learning Studio capabilities](#).

Machine learning algorithm cheat sheet for Microsoft Azure Machine Learning Studio

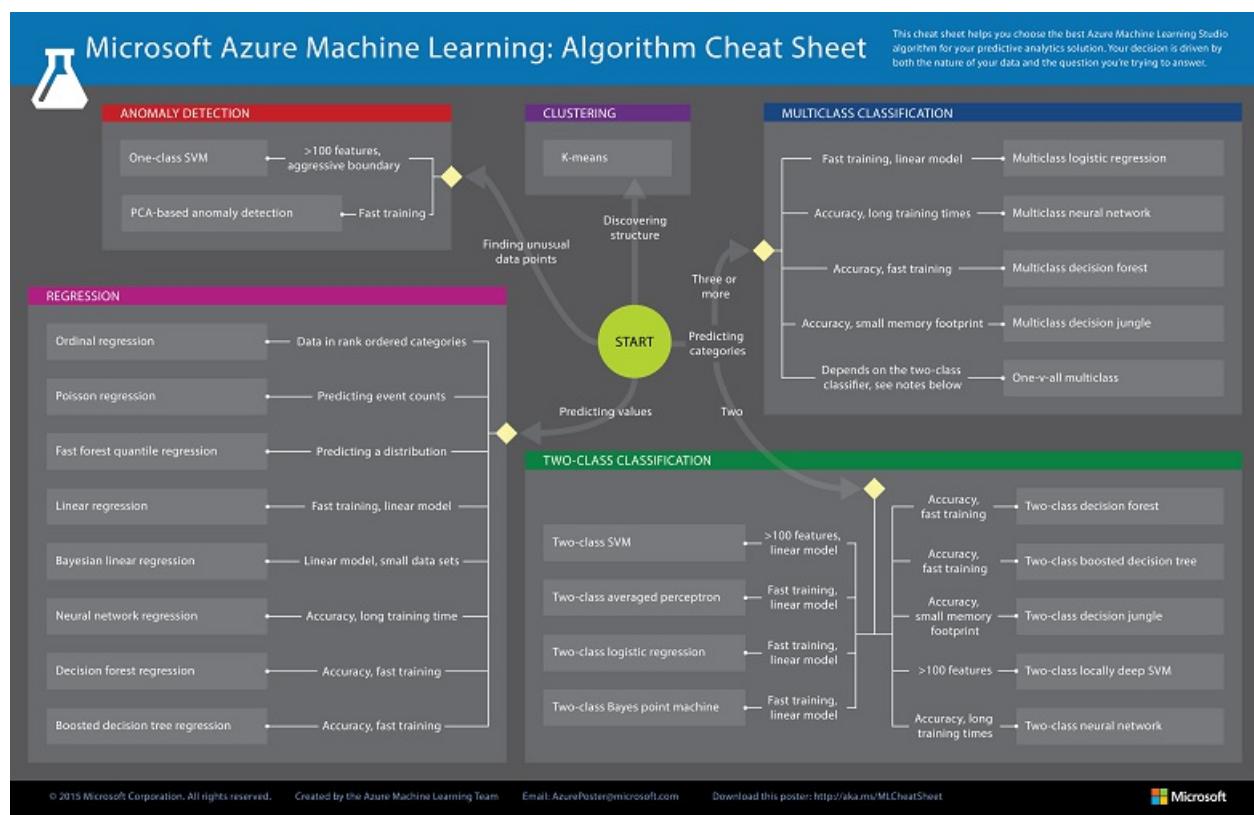
1/17/2017 • 5 min to read • [Edit on GitHub](#)

The **Microsoft Azure Machine Learning Algorithm Cheat Sheet** helps you choose the right algorithm for a predictive analytics model.

Azure Machine Learning Studio has a large library of algorithms from the **regression**, **classification**, **clustering**, and **anomaly detection** families. Each is designed to address a different type of machine learning problem.

Download: Machine learning algorithm cheat sheet

Download the cheat sheet here: [Machine Learning Algorithm Cheat Sheet \(11x17 in.\)](#)



Download and print the Machine Learning Algorithm Cheat Sheet in tabloid size to keep it handy and get help choosing an algorithm.

NOTE

See the article [How to choose algorithms for Microsoft Azure Machine Learning](#) for a detailed guide to using this cheat sheet.

More help with algorithms

- For help in using this cheat sheet for choosing the right algorithm, plus a deeper discussion of the different types of machine learning algorithms and how they're used, see [How to choose algorithms for Microsoft Azure Machine Learning](#).

- For a downloadable infographic that describes algorithms and provides examples, see [Downloadable Infographic: Machine learning basics with algorithm examples](#).
- For a list by category of all the machine learning algorithms available in Machine Learning Studio, see [Initialize Model](#) in the Machine Learning Studio Algorithm and Module Help.
- For a complete alphabetical list of algorithms and modules in Machine Learning Studio, see [A-Z list of Machine Learning Studio modules](#) in Machine Learning Studio Algorithm and Module Help.
- To download and print a diagram that gives an overview of the capabilities of Machine Learning Studio, see [Overview diagram of Azure Machine Learning Studio capabilities](#).

NOTE

[Try Azure Machine Learning for free](#)

No credit card or Azure subscription needed. [Get started now >](#)

Notes and terminology definitions for the machine learning algorithm cheat sheet

- The suggestions offered in this algorithm cheat sheet are approximate rules-of-thumb. Some can be bent, and some can be flagrantly violated. This is intended to suggest a starting point. Don't be afraid to run a head-to-head competition between several algorithms on your data. There is simply no substitute for understanding the principles of each algorithm and understanding the system that generated your data.
- Every machine learning algorithm has its own style or *inductive bias*. For a specific problem, several algorithms may be appropriate and one algorithm may be a better fit than others. But knowing which will be the best fit beforehand is not always possible. In cases like these, several algorithms are listed together in the cheat sheet. An appropriate strategy would be to try one algorithm, and if the results are not yet satisfactory, try the others. Here's an example from the [Cortana Intelligence Gallery](#) of an experiment that tries several algorithms against the same data and compares the results: [Compare Multi-class Classifiers: Letter recognition](#).
- There are three main categories of machine learning: **supervised learning**, **unsupervised learning**, and **reinforcement learning**.
 - In **supervised learning**, each data point is labeled or associated with a category or value of interest. An example of a categorical label is assigning an image as either a 'cat' or a 'dog'. An example of a value label is the sale price associated with a used car. The goal of supervised learning is to study many labeled examples like these, and then to be able to make predictions about future data points - for example, to identify new photos with the correct animal or to assign accurate sale prices to other used cars. This is a popular and useful type of machine learning. All of the modules in Azure Machine Learning are supervised learning algorithms except for [K-Means Clustering](#).
 - In **unsupervised learning**, data points have no labels associated with them. Instead, the goal of an unsupervised learning algorithm is to organize the data in some way or to describe its structure. This can mean grouping it into clusters, as K-means does, or finding different ways of looking at complex data so that it appears simpler.
 - In **reinforcement learning**, the algorithm gets to choose an action in response to each data point. It is a common approach in robotics, where the set of sensor readings at one point in time is a data point, and the algorithm must choose the robot's next action. It's also a natural fit for Internet of Things applications. The learning algorithm also receives a reward signal a short time later, indicating how good the decision was. Based on this, the algorithm modifies its strategy in order to achieve the highest reward. Currently there are no reinforcement learning algorithm modules in Azure ML.

- **Bayesian methods** make the assumption of statistically independent data points. This means that the unmodeled variability in one data point is uncorrelated with others, that is, it can't be predicted. For example, if the data being recorded is the number of minutes until the next subway train arrives, two measurements taken a day apart are statistically independent. However, two measurements taken a minute apart are not statistically independent - the value of one is highly predictive of the value of the other.
- **Boosted decision tree regression** takes advantage of feature overlap or interaction among features. That means that, in any given data point, the value of one feature is somewhat predictive of the value of another. For example, in daily high/low temperature data, knowing the low temperature for the day allows you to make a reasonable guess at the high. The information contained in the two features is somewhat redundant.
- Classifying data into more than two categories can be done by either using an inherently multi-class classifier, or by combining a set of two-class classifiers into an **ensemble**. In the ensemble approach, there is a separate two-class classifier for each class - each one separates the data into two categories: "this class" and "not this class." Then these classifiers vote on the correct assignment of the data point. This is the operational principle behind [One-vs-All Multiclass](#).
- Several methods, including logistic regression and the Bayes point machine, assume **linear class boundaries**, that is, that the boundaries between classes are approximately straight lines (or hyperplanes in the more general case). Often this is a characteristic of the data that you don't know until after you've tried to separate it, but it's something that typically can be learned by visualizing beforehand. If the class boundaries look very irregular, stick with decision trees, decision jungles, support vector machines, or neural networks.
- Neural networks can be used with categorical variables by creating a **dummy variable** for each category and setting it to 1 in cases where the category applies, 0 where it doesn't.

Using linear regression in Azure Machine Learning

1/17/2017 • 7 min to read • [Edit on GitHub](#)

Kate Baroni and *Ben Boatman* are enterprise solution architects in Microsoft's Data Insights Center of Excellence. In this article, they describe their experience migrating an existing regression analysis suite to a cloud-based solution using Azure Machine Learning.

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

Goal

Our project started with two goals in mind:

1. Use predictive analytics to improve the accuracy of our organization's monthly revenue projections
2. Use Azure Machine Learning to confirm, optimize, increase velocity, and scale of our results.

Like many businesses, our organization goes through a monthly revenue forecasting process. Our small team of business analysts was tasked with using Azure Machine Learning to support the process and improve forecast accuracy. The team spent several months collecting data from multiple sources and running the data attributes through statistical analysis identifying key attributes relevant to services sales forecasting. The next step was to begin prototyping statistical regression models on the data in Excel. Within a few weeks, we had an Excel regression model that was outperforming the current field and finance forecasting processes. This became the baseline prediction result.

We then took the next step to moving our predictive analytics over to Azure Machine Learning to find out how Machine Learning could improve on predictive performance.

Achieving predictive performance parity

Our first priority was to achieve parity between Machine Learning and Excel regression models. Given the same data, and the same split for training and testing data, we wanted to achieve predictive performance parity between Excel and Machine Learning. Initially we failed. The Excel model outperformed the Machine Learning model. The failure was due to a lack of understanding of the base tool setting in Machine Learning. After a sync with the Machine Learning product team, we gained a better understanding of the base setting required for our data sets, and achieved parity between the two models.

Create regression model in Excel

Our Excel Regression used the standard linear regression model found in the Excel Analysis ToolPak.

We calculated *Mean Absolute % Error* and used it as the performance measure for the model. It took 3 months to arrive at a working model using Excel. We brought much of the learning into the Machine Learning Studio experiment which ultimately was beneficial in understanding requirements.

Create comparable experiment in Azure Machine Learning

We followed these steps to create our experiment in Machine Learning Studio:

1. Uploaded the dataset as a csv file to Machine Learning Studio (very small file)
2. Created a new experiment and used the [Select Columns in Dataset](#) module to select the same data features used in Excel
3. Used the [Split Data](#) module (with *Relative Expression* mode) to divide the data into the same training datasets as had been done in Excel
4. Experimented with the [Linear Regression](#) module (default options only), documented, and compared the results to our Excel regression model

Review initial results

At first, the Excel model clearly outperformed the Machine Learning Studio model:

	EXCEL	STUDIO
Performance		
Adjusted R Square	0.96	N/A
Coefficient of Determination	N/A	0.78 (low accuracy)
Mean Absolute Error	\$9.5M	\$ 19.4M
Mean Absolute Error (%)	6.03%	12.2%

When we ran our process and results by the developers and data scientists on the Machine Learning team, they quickly provided some useful tips.

- When you use the [Linear Regression](#) module in Machine Learning Studio, two methods are provided:
 - Online Gradient Descent: May be more suitable for larger-scale problems
 - Ordinary Least Squares: This is the method most people think of when they hear linear regression. For small datasets, Ordinary Least Squares can be a more optimal choice.
- Consider tweaking the L2 Regularization Weight parameter to improve performance. It is set to 0.001 by default, but for our small data set we set it to 0.005 to improve performance.

Mystery solved!

When we applied the recommendations, we achieved the same baseline performance in Machine Learning Studio as with Excel:

	EXCEL	STUDIO (INITIAL)	STUDIO W/ LEAST SQUARES
Labeled value	Actuals (numeric)	same	same
Learner	Excel -> Data Analysis -> Regression	Linear Regression.	Linear Regression
Learner options	N/A	Defaults	ordinary least squares L2 = 0.005
Data Set	26 rows, 3 features, 1 label. All numeric.	same	same

	EXCEL	STUDIO (INITIAL)	STUDIO W/ LEAST SQUARES
Split: Train	Excel trained on the first 18 rows, tested on the last 8 rows.	same	same
Split: Test	Excel regression formula applied to the last 8 rows	same	same
Performance			
Adjusted R Square	0.96	N/A	
Coefficient of Determination	N/A	0.78	0.952049
Mean Absolute Error	\$9.5M	\$ 19.4M	\$9.5M
Mean Absolute Error (%)	6.03%	12.2%	6.03%

In addition, the Excel coefficients compared well to the feature weights in the Azure trained model:

	EXCEL COEFFICIENTS	AZURE FEATURE WEIGHTS
Intercept/Bias	19470209.88	19328500
Feature A	0.832653063	0.834156
Feature B	11071967.08	11007300
Feature C	25383318.09	25140800

Next Steps

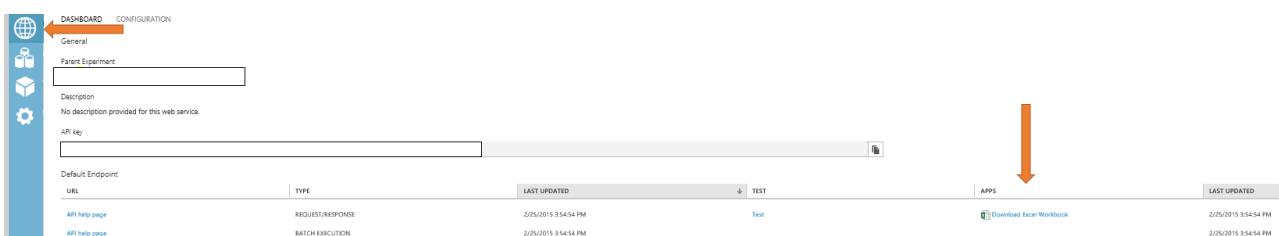
We wanted to consume the Machine Learning web service within Excel. Our business analysts rely on Excel and we needed a way to call the Machine Learning web service with a row of Excel data and have it return the predicted value to Excel.

We also wanted to optimize our model, using the options and algorithms available in Machine Learning Studio.

Integration with Excel

Our solution was to operationalize our Machine Learning regression model by creating a web service from the trained model. Within a few minutes, the web service was created and we could call it directly from Excel to return a predicted revenue value.

The *Web Services Dashboard* section includes a downloadable Excel workbook. The workbook comes pre-formatted with the web service API and schema information embedded. When you click *Download Excel Workbook*, the workbook opens and you can save it to your local computer.



With the workbook open, copy your predefined parameters into the blue Parameter section as shown below. Once the parameters are entered, Excel calls out to the Machine Learning web service and the predicted scored labels will display in the green Predicted Values section. The workbook will continue to create predictions for parameters based on your trained model for all row items entered under Parameters. For more information on how to use this feature, see [Consuming an Azure Machine Learning Web Service from Excel](#).

Optimization and further experiments

Now that we had a baseline with our Excel model, we moved ahead to optimize our Machine Learning Linear Regression Model. We used the module [Filter-Based Feature Selection](#) to improve on our selection of initial data elements and it helped us achieve a performance improvement of 4.6% Mean Absolute Error. For future projects we will use this feature which could save us weeks in iterating through data attributes to find the right set of features to use for modelling.

Next we plan to include additional algorithms like Bayesian or [Boosted Decision Trees](#) in our experiment to compare performance.

If you want to experiment with regression, a good dataset to try is the Energy Efficiency Regression sample dataset, which has lots of numerical attributes. The dataset is provided as part of the sample datasets in Machine Learning Studio. You can use a variety of learning modules to predict either Heating Load or Cooling Load. The chart below is a performance comparison of different regression learners against the Energy Efficiency dataset predicting for the target variable Cooling Load:

Model	Mean Absolute Error	Root Mean Squared Error	Relative Absolute Error	Relative Squared Error	Coefficient of Determination
Boosted Decision Tree	0.930113	1.4239	0.106647	0.021662	0.978338
Linear Regression (Gradient Descent)	2.035693	2.98006	0.233414	0.094881	0.905119
Neural Network Regression	1.548195	2.114617	0.177517	0.047774	0.952226
Linear Regression (Ordinary Least Squares)	1.428273	1.984461	0.163767	0.042074	0.957926

Key Takeaways

We learned a lot by running Excel regression and Azure Machine Learning experiments in parallel. Creating the baseline model in Excel and comparing it to models using Machine Learning [Linear Regression](#) helped us learn Azure Machine Learning, and we discovered opportunities to improve data selection and model performance.

We also found that it is advisable to use [Filter-Based Feature Selection](#) to accelerate future prediction projects. By applying feature selection to your data, you can create an improved model in Machine Learning with better overall performance.

The ability to transfer the predictive analytic forecasting from Machine Learning to Excel systemically allows a significant increase in the ability to successfully provide results to a broad business user audience.

Resources

Here are some resources for helping you work with regression:

- Regression in Excel. If you've never tried regression in Excel, this tutorial makes it easy: <http://www.excel-easy.com/examples/regression.html>
- Regression vs forecasting. Tyler Chessman wrote a blog article explaining how to do time series forecasting in Excel, which contains a good beginner's description of linear regression. <http://sqlmag.com/sql-server-analysis-services/understanding-time-series-forecasting-concepts>
- Ordinary Least Squares Linear Regression: Flaws, Problems and Pitfalls. For an introduction and discussion of Regression: <http://www.clockbackward.com/2009/06/18/ordinary-least-squares-linear-regression-flaws-problems-and-pitfalls/>

Create text analytics models in Azure Machine Learning Studio

1/17/2017 • 5 min to read • [Edit on GitHub](#)

You can use Azure Machine Learning to build and operationalize text analytics models. These models can help you solve, for example, document classification or sentiment analysis problems.

In a text analytics experiment, you would typically:

1. Clean and preprocess text dataset
2. Extract numeric feature vectors from pre-processed text
3. Train classification or regression model
4. Score and validate the model
5. Deploy the model to production

In this tutorial, you learn these steps as we walk through a sentiment analysis model using Amazon Book Reviews dataset (see this research paper “Biographies, Bollywood, Boom-boxes and Blenders: Domain Adaptation for Sentiment Classification” by John Blitzer, Mark Dredze, and Fernando Pereira; Association of Computational Linguistics (ACL), 2007.) This dataset consists of review scores (1-2 or 4-5) and a free-form text. The goal is to predict the review score: low (1-2) or high (4-5).

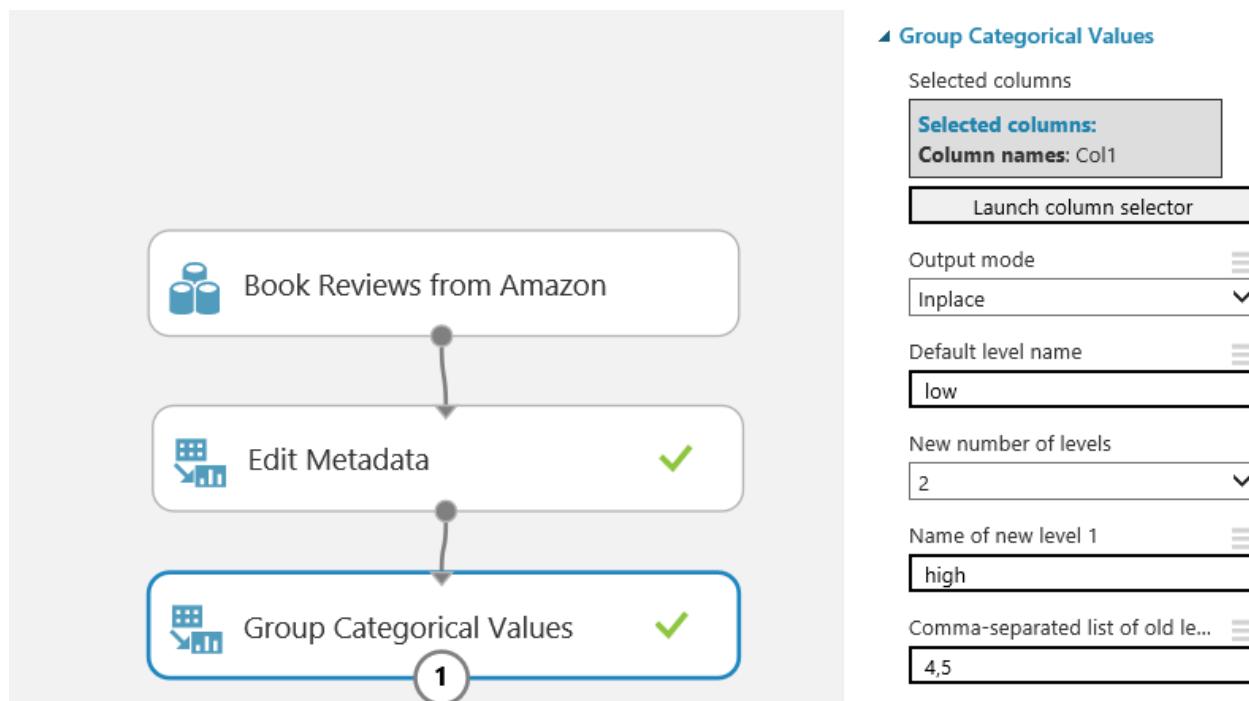
You can find experiments covered in this tutorial at Cortana Intelligence Gallery:

[Predict Book Reviews](#)

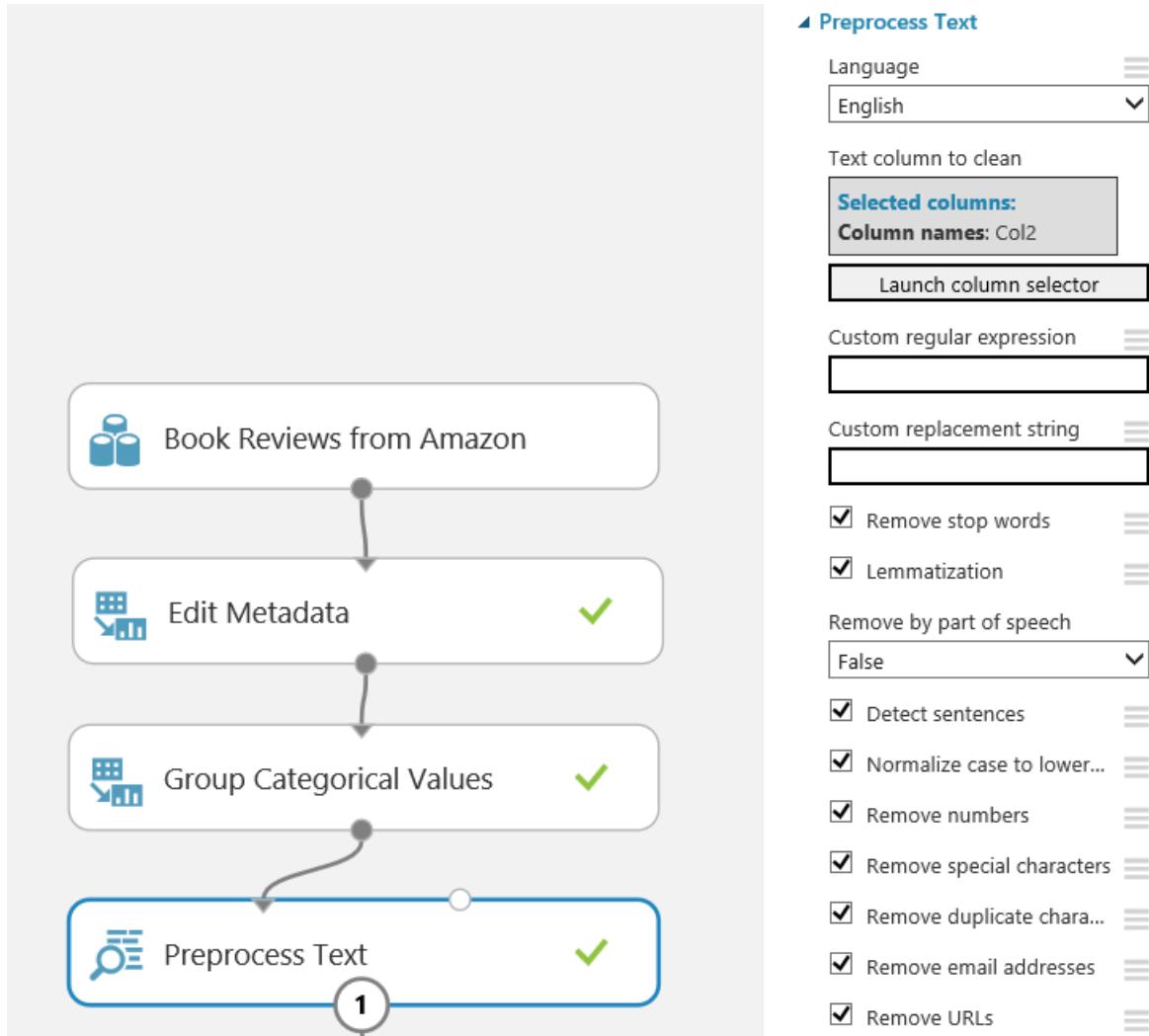
[Predict Book Reviews - Predictive Experiment](#)

Step 1: Clean and preprocess text dataset

We begin the experiment by dividing the review scores into categorical low and high buckets to formulate the problem as two-class classification. We use [Edit Metadata](#) and [Group Categorical Values](#) modules.



Then, we clean the text using [Preprocess Text](#) module. The cleaning reduces the noise in the dataset, help you find the most important features, and improve the accuracy of the final model. We remove stopwords - common words such as "the" or "a" - and numbers, special characters, duplicated characters, email addresses, and URLs. We also convert the text to lowercase, lemmatize the words, and detect sentence boundaries that are then indicated by "|||" symbol in pre-processed text.

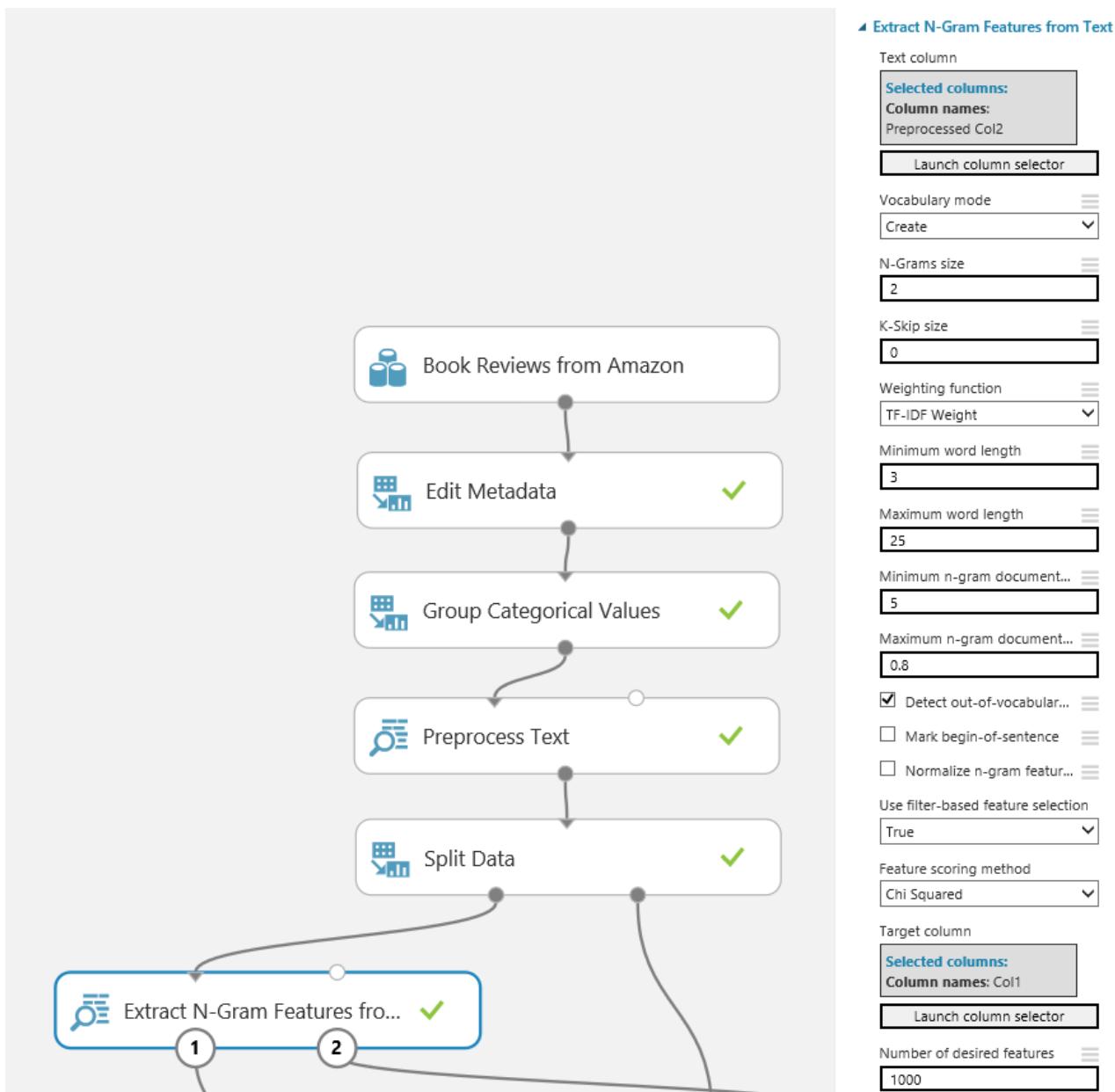


What if you want to use a custom list of stopwords? You can pass it in as optional input. You can also use custom C# syntax regular expression to replace substrings, and remove words by part of speech: nouns, verbs, or adjectives.

After the preprocessing is complete, we split the data into train and test sets.

Step 2: Extract numeric feature vectors from pre-processed text

To build a model for text data, you typically have to convert free-form text into numeric feature vectors. In this example, we use [Extract N-Gram Features from Text](#) module to transform the text data to such format. This module takes a column of whitespace-separated words and computes a dictionary of words, or N-grams of words, that appear in your dataset. Then, it counts how many times each word, or N-gram, appears in each record, and creates feature vectors from those counts. In this tutorial, we set N-gram size to 2, so our feature vectors include single words and combinations of two subsequent words.



We apply TF*IDF (Term Frequency Inverse Document Frequency) weighting to N-gram counts. This approach adds weight of words that appear frequently in a single record but are rare across the entire dataset. Other options include binary, TF, and graph weighing.

Such text features often have high dimensionality. For example, if your corpus has 100,000 unique words, your feature space would have 100,000 dimensions, or more if N-grams are used. The Extract N-Gram Features module gives you a set of options to reduce the dimensionality. You can choose to exclude words that are short or long, or too uncommon or too frequent to have significant predictive value. In this tutorial, we exclude N-grams that appear in fewer than 5 records or in more than 80% of records.

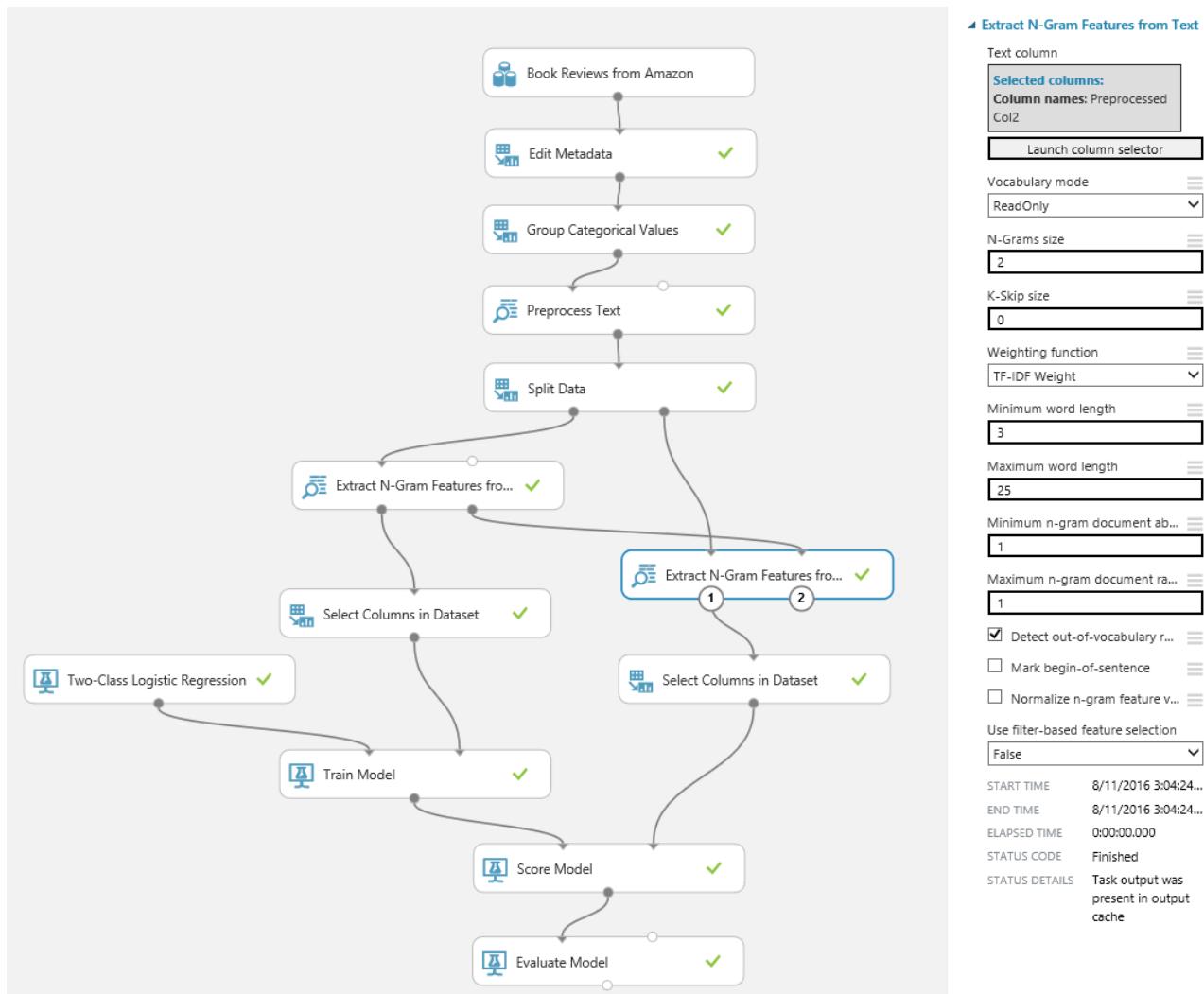
Also, you can use feature selection to select only those features that are the most correlated with your prediction target. We use Chi-Squared feature selection to select 1000 features. You can view the vocabulary of selected words or N-grams by clicking the right output of Extract N-grams module.

As an alternative approach to using Extract N-Gram Features, you can use Feature Hashing module. Note though that [Feature Hashing](#) does not have build-in feature selection capabilities, or TF*IDF weighing.

Step 3: Train classification or regression model

Now the text has been transformed to numeric feature columns. The dataset still contains string columns from previous stages, so we use Select Columns in Dataset to exclude them.

We then use [Two-Class Logistic Regression](#) to predict our target: high or low review score. At this point, the text analytics problem has been transformed into a regular classification problem. You can use the tools available in Azure Machine Learning to improve the model. For example, you can experiment with different classifiers to find out how accurate results they give, or use hyperparameter tuning to improve the accuracy.



Step 4: Score and validate the model

How would you validate the trained model? We score it against the test dataset and evaluate the accuracy. However, the model learned the vocabulary of N-grams and their weights from the training dataset. Therefore, we should use that vocabulary and those weights when extracting features from test data, as opposed to creating the vocabulary anew. Therefore, we add Extract N-Gram Features module to the scoring branch of the experiment, connect the output vocabulary from training branch, and set the vocabulary mode to read-only. We also disable the filtering of N-grams by frequency by setting the minimum to 1 instance and maximum to 100%, and turn off the feature selection.

After the text column in test data has been transformed to numeric feature columns, we exclude the string columns from previous stages like in training branch. We then use Score Model module to make predictions and Evaluate Model module to evaluate the accuracy.

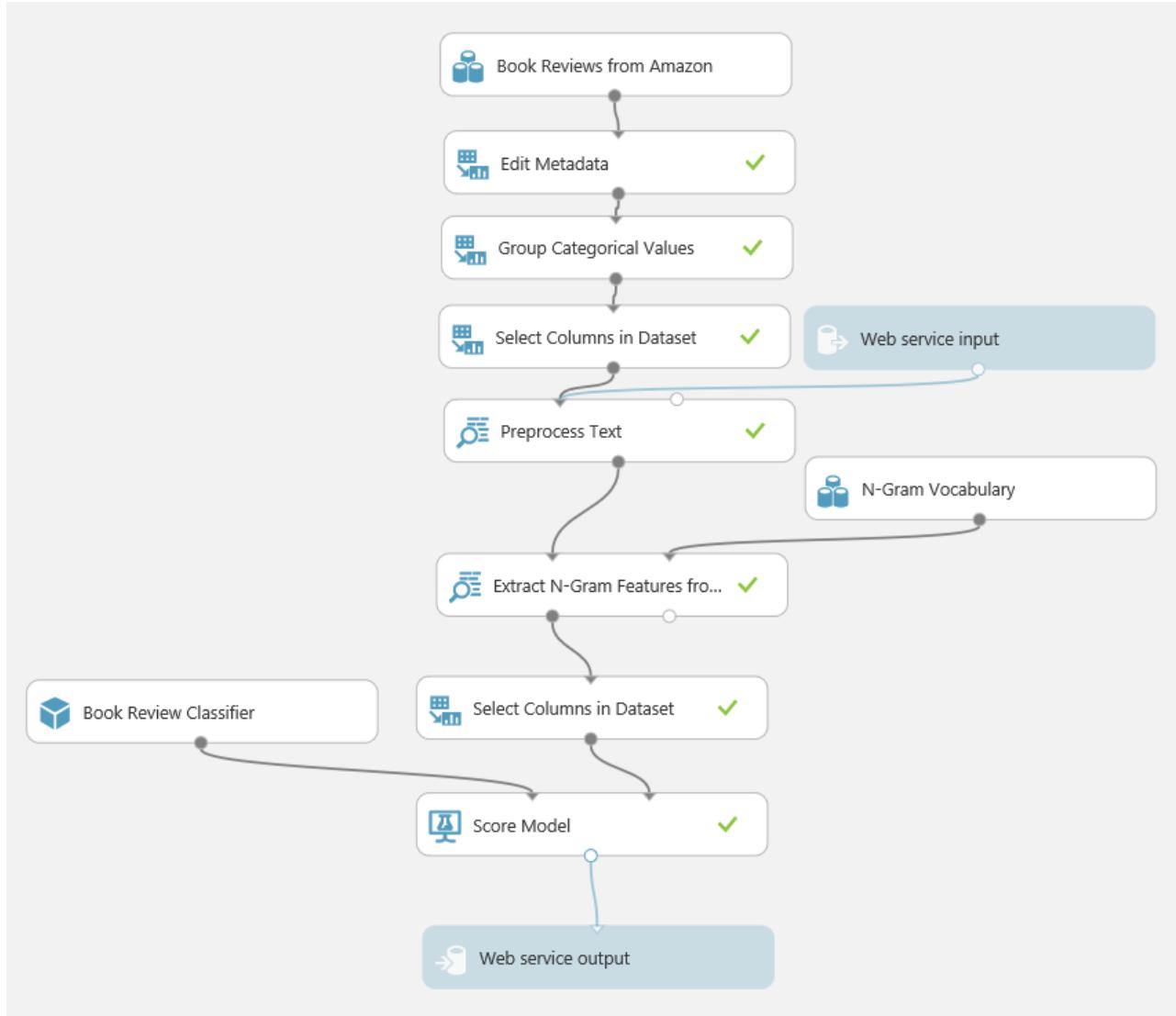
Step 5: Deploy the model to production

The model is almost ready to be deployed to production. When deployed as web service, it takes free-form text string as input, and return a prediction "high" or "low." It uses the learned N-gram vocabulary to transform the text to features, and trained logistic regression model to make a prediction from those features.

To set up the predictive experiment, we first save the N-gram vocabulary as dataset, and the trained logistic

regression model from the training branch of the experiment. Then, we save the experiment using "Save As" to create an experiment graph for predictive experiment. We remove the Split Data module and the training branch from the experiment. We then connect the previously saved N-gram vocabulary and model to Extract N-Gram Features and Score Model modules, respectively. We also remove the Evaluate Model module.

We insert Select Columns in Dataset module before Preprocess Text module to remove the label column, and unselect "Append score column to dataset" option in Score Module. That way, the web service does not request the label it is trying to predict, and does not echo the input features in response.



Now we have an experiment that can be published as a web service and called using request-response or batch execution APIs.

Next Steps

Learn about text analytics modules from [MSDN documentation](#).

How to evaluate model performance in Azure Machine Learning

1/17/2017 • 12 min to read • [Edit on GitHub](#)

This article demonstrates how to evaluate the performance of a model in Azure Machine Learning Studio and provides a brief explanation of the metrics available for this task. Three common supervised learning scenarios are presented:

- regression
- binary classification
- multiclass classification

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

Evaluating the performance of a model is one of the core stages in the data science process. It indicates how successful the scoring (predictions) of a dataset has been by a trained model.

Azure Machine Learning supports model evaluation through two of its main machine learning modules: [Evaluate Model](#) and [Cross-Validate Model](#). These modules allow you to see how your model performs in terms of a number of metrics that are commonly used in machine learning and statistics.

Evaluation vs. Cross Validation

Evaluation and cross validation are standard ways to measure the performance of your model. They both generate evaluation metrics that you can inspect or compare against those of other models.

[Evaluate Model](#) expects a scored dataset as input (or 2 in case you would like to compare the performance of 2 different models). This means that you need to train your model using the [Train Model](#) module and make predictions on some dataset using the [Score Model](#) module, before you can evaluate the results. The evaluation is based on the scored labels/probabilities along with the true labels, all of which are output by the [Score Model](#) module.

Alternatively, you can use cross validation to perform a number of train-score-evaluate operations (10 folds) automatically on different subsets of the input data. The input data is split into 10 parts, where one is reserved for testing, and the other 9 for training. This process is repeated 10 times and the evaluation metrics are averaged. This helps in determining how well a model would generalize to new datasets. The [Cross-Validate Model](#) module takes in an untrained model and some labeled dataset and outputs the evaluation results of each of the 10 folds, in addition to the averaged results.

In the following sections, we will build simple regression and classification models and evaluate their performance, using both the [Evaluate Model](#) and the [Cross-Validate Model](#) modules.

Evaluating a Regression Model

Assume we want to predict a car's price using some features such as dimensions, horsepower, engine specs, and so on. This is a typical regression problem, where the target variable (*price*) is a continuous numeric value. We can fit a

simple linear regression model that, given the feature values of a certain car, can predict the price of that car. This regression model can be used to score the same dataset we trained on. Once we have the predicted prices for all of the cars, we can evaluate the performance of the model by looking at how much the predictions deviate from the actual prices on average. To illustrate this, we use the *Automobile price data (Raw)* dataset available in the **Saved Datasets** section in Azure Machine Learning Studio.

Creating the Experiment

Add the following modules to your workspace in Azure Machine Learning Studio:

- Automobile price data (Raw)
- Linear Regression
- Train Model
- Score Model
- Evaluate Model

Connect the ports as shown below in Figure 1 and set the Label column of the **Train Model** module to *price*.

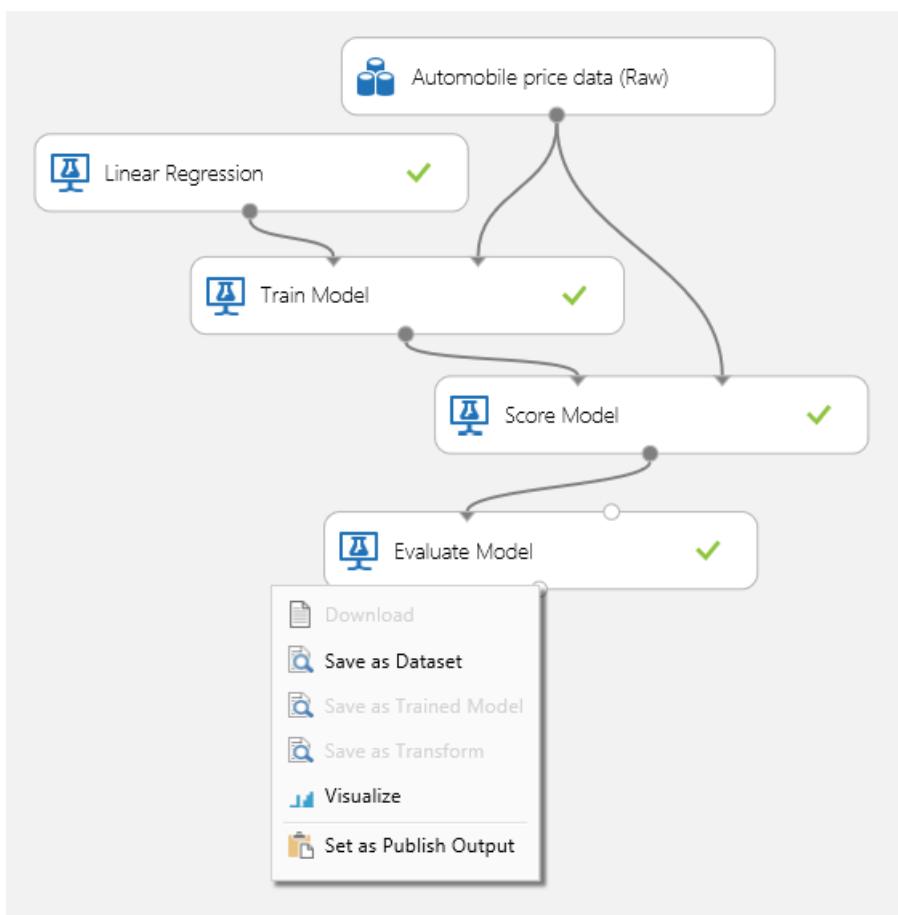


Figure 1. Evaluating a Regression Model.

Inspecting the Evaluation Results

After running the experiment, you can click on the output port of the **Evaluate Model** module and select *Visualize* to see the evaluation results. The evaluation metrics available for regression models are: *Mean Absolute Error*, *Root Mean Absolute Error*, *Relative Absolute Error*, *Relative Squared Error*, and the *Coefficient of Determination*.

The term "error" here represents the difference between the predicted value and the true value. The absolute value or the square of this difference are usually computed to capture the total magnitude of error across all instances, as the difference between the predicted and true value could be negative in some cases. The error metrics measure the predictive performance of a regression model in terms of the mean deviation of its predictions from the true values. Lower error values mean the model is more accurate in making predictions. An overall error metric of 0 means that the model fits the data perfectly.

The coefficient of determination, which is also known as R squared, is also a standard way of measuring how well the model fits the data. It can be interpreted as the proportion of variation explained by the model. A higher proportion is better in this case, where 1 indicates a perfect fit.

rows	columns
1	5
Mean Absolute Error	
947.975254	Root Mean Squared Error
955.587783	Relative Absolute Error
0.101295	Relative Squared Error
0.011495	Coefficient of Determination
view as	
1	2
3	4
5	6
7	8
9	10

Figure 2. Linear Regression Evaluation Metrics.

Using Cross Validation

As mentioned earlier, you can perform repeated training, scoring and evaluations automatically using the [Cross-Validate Model](#) module. All you need in this case is a dataset, an untrained model, and a [Cross-Validate Model](#) module (see figure below). Note that you need to set the label column to *price* in the [Cross-Validate Model](#) module's properties.

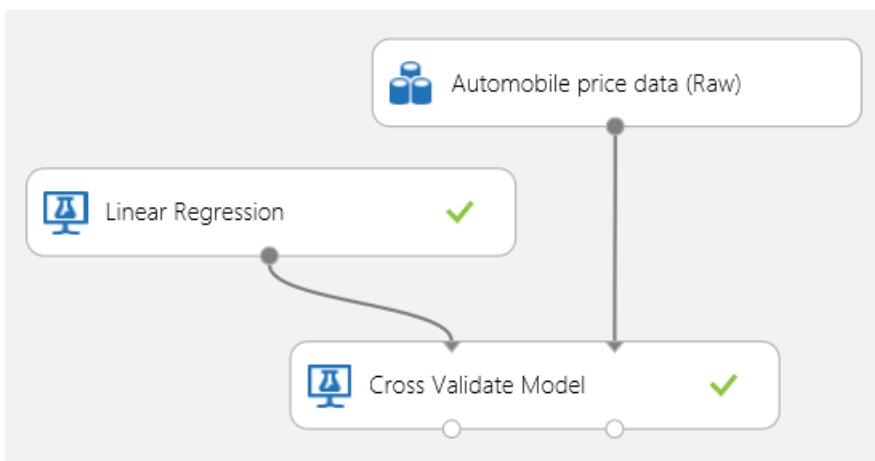


Figure 3. Cross-Validating a Regression Model.

After running the experiment, you can inspect the evaluation results by clicking on the right output port of the [Cross-Validate Model](#) module. This will provide a detailed view of the metrics for each iteration (fold), and the averaged results of each of the metrics (Figure 4).

rows	columns						
	8						
view as							
Fold Number	Number of examples in fold	Model	Mean Absolute Error	Root Mean Squared Error	Relative Absolute Error	Relative Squared Error	Coefficient of Determination
0	20	Microsoft.Analytics.Machi neLearning.LocalBatchLin earRegressor	1975.690002	2868.853475	0.318462	0.261852	0.738148
1	21	Microsoft.Analytics.Machi neLearning.LocalBatchLin earRegressor	1204.367667	1736.369322	0.259077	0.101072	0.898228
2	20	Microsoft.Analytics.Machi neLearning.LocalBatchLin earRegressor	1275.525323	1565.606411	0.146945	0.021472	0.978528
3	21	Microsoft.Analytics.Machi neLearning.LocalBatchLin earRegressor	1180.785629	1479.619128	0.150116	0.024304	0.975696
4	20	Microsoft.Analytics.Machi neLearning.LocalBatchLin earRegressor	672.068144	902.187494	0.091132	0.010176	0.989824
5	20	Microsoft.Analytics.Machi neLearning.LocalBatchLin earRegressor	1166.036215	1492.147079	0.227148	0.055168	0.944832
6	21	Microsoft.Analytics.Machi neLearning.LocalBatchLin earRegressor	1544.169782	1956.495628	0.272313	0.07162	0.92838
7	21	Microsoft.Analytics.Machi neLearning.LocalBatchLin earRegressor	1435.049593	2055.843829	0.17445	0.042546	0.957454
8	20	Microsoft.Analytics.Machi neLearning.LocalBatchLin earRegressor	1583.894079	2192.621829	0.186784	0.042628	0.957372
9	21	Microsoft.Analytics.Machi neLearning.LocalBatchLin earRegressor	1069.820402	1464.410289	0.113144	0.020426	0.979574
Mean	205	Microsoft.Analytics.Machi neLearning.LocalBatchLin earRegressor	1310.740583	1771.443708	0.199557	0.065197	0.934803
Standard Deviation	205	Microsoft.Analytics.Machi neLearning.LocalBatchLin earRegressor	350.085776	532.636933	0.073523	0.07438	0.07438

Figure 4. Cross-Validation Results of a Regression Model.

Evaluating a Binary Classification Model

In a binary classification scenario, the target variable has only two possible outcomes, for example: {0, 1} or {false, true}, {negative, positive}. Assume you are given a dataset of adult employees with some demographic and employment variables, and that you are asked to predict the income level, a binary variable with the values {"<=50K", ">50K"}. In other words, the negative class represents the employees who make less than or equal to 50K per year, and the positive class represents all other employees. As in the regression scenario, we would train a model, score some data, and evaluate the results. The main difference here is the choice of metrics Azure Machine Learning computes and outputs. To illustrate the income level prediction scenario, we will use the [Adult](#) dataset to create an Azure Machine Learning experiment and evaluate the performance of a two-class logistic regression model, a commonly used binary classifier.

Creating the Experiment

Add the following modules to your workspace in Azure Machine Learning Studio:

- Adult Census Income Binary Classification dataset
- [Two-Class Logistic Regression](#)
- [Train Model](#)
- [Score Model](#)
- [Evaluate Model](#)

Connect the ports as shown below in Figure 5 and set the Label column of the [Train Model](#) module to *income*.

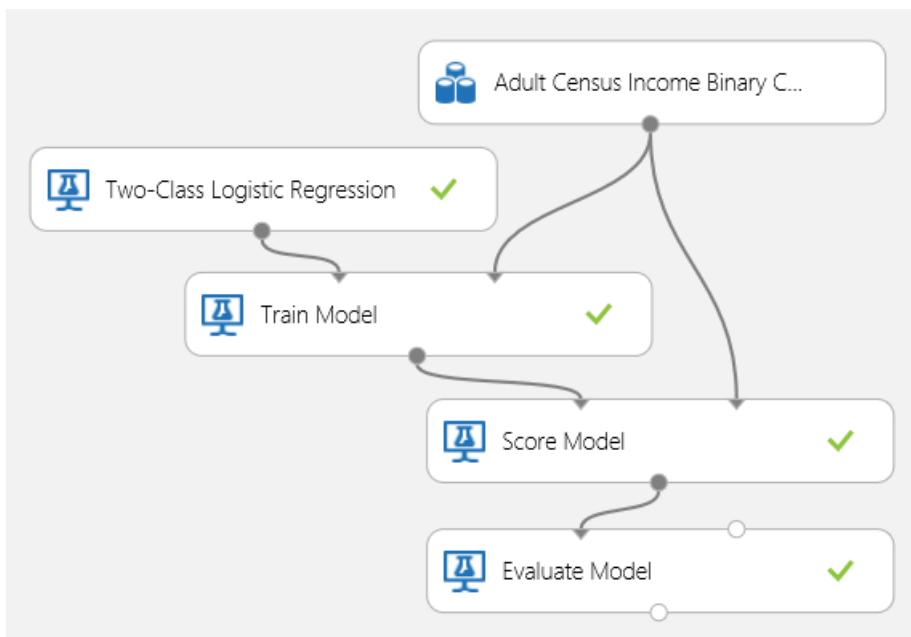


Figure 5. Evaluating a Binary Classification Model.

Inspecting the Evaluation Results

After running the experiment, you can click on the output port of the [Evaluate Model](#) module and select *Visualize* to see the evaluation results (Figure 7). The evaluation metrics available for binary classification models are: *Accuracy*, *Precision*, *Recall*, *F1 Score*, and *AUC*. In addition, the module outputs a confusion matrix showing the number of true positives, false negatives, false positives, and true negatives, as well as *ROC*, *Precision/Recall*, and *Lift* curves.

Accuracy is simply the proportion of correctly classified instances. It is usually the first metric you look at when evaluating a classifier. However, when the test data is unbalanced (where most of the instances belong to one of the classes), or you are more interested in the performance on either one of the classes, accuracy doesn't really capture the effectiveness of a classifier. In the income level classification scenario, assume you are testing on some data where 99% of the instances represent people who earn less than or equal to 50K per year. It is possible to achieve a 0.99 accuracy by predicting the class "<=50K" for all instances. The classifier in this case appears to be

doing a good job overall, but in reality, it fails to classify any of the high-income individuals (the 1%) correctly.

For that reason, it is helpful to compute additional metrics that capture more specific aspects of the evaluation. Before going into the details of such metrics, it is important to understand the confusion matrix of a binary classification evaluation. The class labels in the training set can take on only 2 possible values, which we usually refer to as positive or negative. The positive and negative instances that a classifier predicts correctly are called true positives (TP) and true negatives (TN), respectively. Similarly, the incorrectly classified instances are called false positives (FP) and false negatives (FN). The confusion matrix is simply a table showing the number of instances that fall under each of these 4 categories. Azure Machine Learning automatically decides which of the two classes in the dataset is the positive class. If the class labels are Boolean or integers, then the 'true' or '1' labeled instances are assigned the positive class. If the labels are strings, as in the case of the income dataset, the labels are sorted alphabetically and the first level is chosen to be the negative class while the second level is the positive class.

		Predicted	
		Positive	Negative
Actual True	TP	FN	
	FP	TN	

Figure 6. Binary Classification Confusion Matrix.

Going back to the income classification problem, we would want to ask several evaluation questions that help us understand the performance of the classifier used. A very natural question is: 'Out of the individuals whom the model predicted to be earning >50K (TP+FP), how many were classified correctly (TP)?' This question can be answered by looking at the **Precision** of the model, which is the proportion of positives that are classified correctly: $TP/(TP+FP)$. Another common question is "Out of all the high earning employees with income >50k (TP+FN), how many did the classifier classify correctly (TP)". This is actually the **Recall**, or the true positive rate: $TP/(TP+FN)$ of the classifier. You might notice that there is an obvious trade-off between precision and recall. For example, given a relatively balanced dataset, a classifier that predicts mostly positive instances, would have a high recall, but a rather low precision as many of the negative instances would be misclassified resulting in a large number of false positives. To see a plot of how these two metrics vary, you can click on the 'PRECISION/RECALL' curve in the evaluation result output page (top left part of Figure 7).

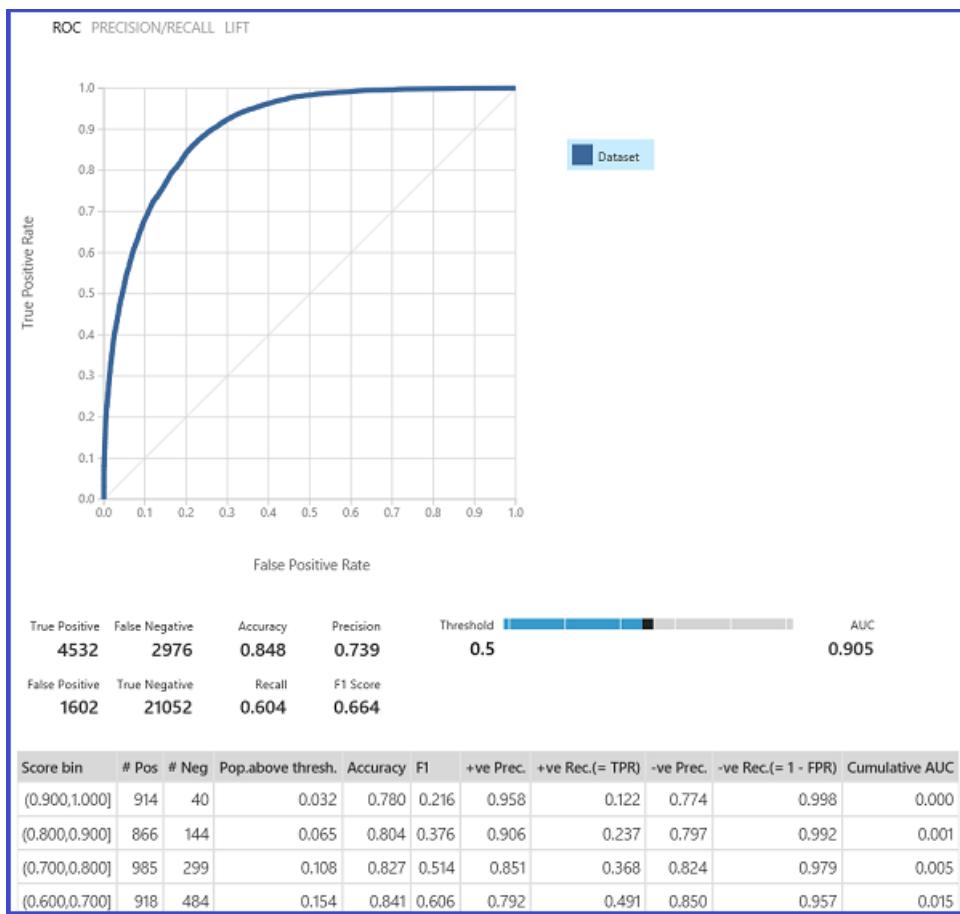


Figure 7. Binary Classification Evaluation Results.

Another related metric that is often used is the **F1 Score**, which takes both precision and recall into consideration. It is the harmonic mean of these 2 metrics and is computed as such: $F1 = 2 \times (\text{precision} \times \text{recall}) / (\text{precision} + \text{recall})$. The F1 score is a good way to summarize the evaluation in a single number, but it's always a good practice to look at both precision and recall together to better understand how a classifier behaves.

In addition, one can inspect the true positive rate vs. the false positive rate in the **Receiver Operating Characteristic (ROC)** curve and the corresponding **Area Under the Curve (AUC)** value. The closer this curve is to the upper left corner, the better the classifier's performance is (that is maximizing the true positive rate while minimizing the false positive rate). Curves that are close to the diagonal of the plot, result from classifiers that tend to make predictions that are close to random guessing.

Using Cross Validation

As in the regression example, we can perform cross validation to repeatedly train, score and evaluate different subsets of the data automatically. Similarly, we can use the **Cross-Validate Model** module, an untrained logistic regression model, and a dataset. The label column must be set to *income* in the **Cross-Validate Model** module's properties. After running the experiment and clicking on the right output port of the **Cross-Validate Model** module, we can see the binary classification metric values for each fold, in addition to the mean and standard deviation of each.

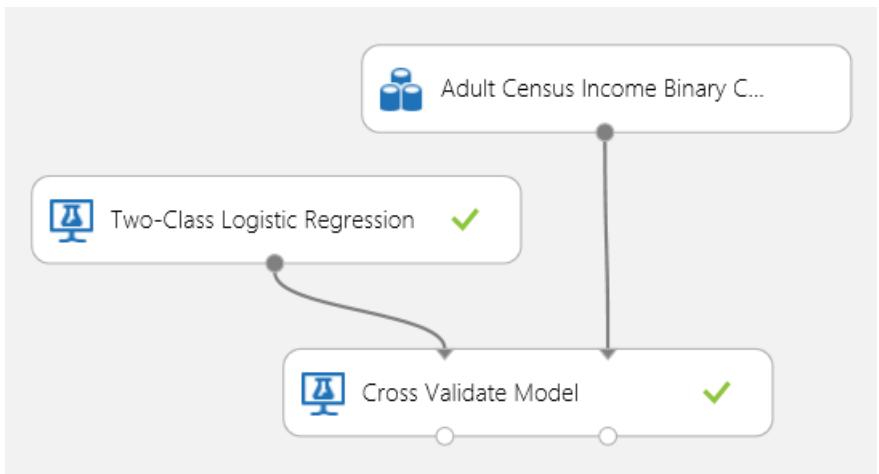


Figure 8. Cross-Validating a Binary Classification Model.

rows	columns								
12	10								
Fold Number	Number of examples in fold	Model	Accuracy	Precision	Recall	F-Score	AUC	Average Log Loss	Training Log Loss
0	3256	Logistic Regression	0.850399	0.737342	0.621333	0.674385	0.905809	0.325788	41.989626
1	3256	Logistic Regression	0.842245	0.707358	0.585062	0.640424	0.901871	0.32527	40.990733
2	3256	Logistic Regression	0.852583	0.767007	0.591864	0.668148	0.905203	0.32936	41.51439
3	3256	Logistic Regression	0.847333	0.729685	0.598639	0.657698	0.898848	0.333642	40.075913
4	3256	Logistic Regression	0.844305	0.741194	0.614213	0.671756	0.90568	0.332448	41.921163
5	3256	Logistic Regression	0.834601	0.716393	0.57199	0.636099	0.897829	0.339869	39.879498
6	3256	Logistic Regression	0.85	0.728188	0.598621	0.657078	0.902695	0.327404	40.596835
7	3257	Logistic Regression	0.846128	0.743464	0.599473	0.663749	0.90208	0.333052	41.153979
8	3256	Logistic Regression	0.846973	0.734861	0.601071	0.661267	0.902736	0.329349	41.259562
9	3256	Logistic Regression	0.849388	0.742718	0.607947	0.668609	0.903525	0.328219	41.623392
Mean	32561	Logistic Regression	0.846396	0.734821	0.599021	0.659921	0.902629	0.33044	41.100509
Standard Deviation	32561	Logistic Regression	0.005145	0.016276	0.014102	0.012778	0.00269	0.004408	0.725968

Figure 9. Cross-Validation Results of a Binary Classifier.

Evaluating a Multiclass Classification Model

In this experiment we will use the popular [Iris](#) dataset which contains instances of 3 different types (classes) of the iris plant. There are 4 feature values (sepal length/width and petal length/width) for each instance. In the previous experiments we trained and tested the models using the same datasets. Here, we will use the [Split Data](#) module to create 2 subsets of the data, train on the first, and score and evaluate on the second. The Iris dataset is publicly available on the [UCI Machine Learning Repository](#), and can be downloaded using an [Import Data](#) module.

Creating the Experiment

Add the following modules to your workspace in Azure Machine Learning Studio:

- [Import Data](#)
- [Multiclass Decision Forest](#)
- [Split Data](#)
- [Train Model](#)
- [Score Model](#)
- [Evaluate Model](#)

Connect the ports as shown below in Figure 10.

Set the Label column index of the [Train Model](#) module to 5. The dataset has no header row but we know that the class labels are in the fifth column.

Click on the [Import Data](#) module and set the *Data source* property to *Web URL via HTTP*, and the *URL* to <http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data>.

Set the fraction of instances to be used for training in the [Split Data](#) module (0.7 for example).

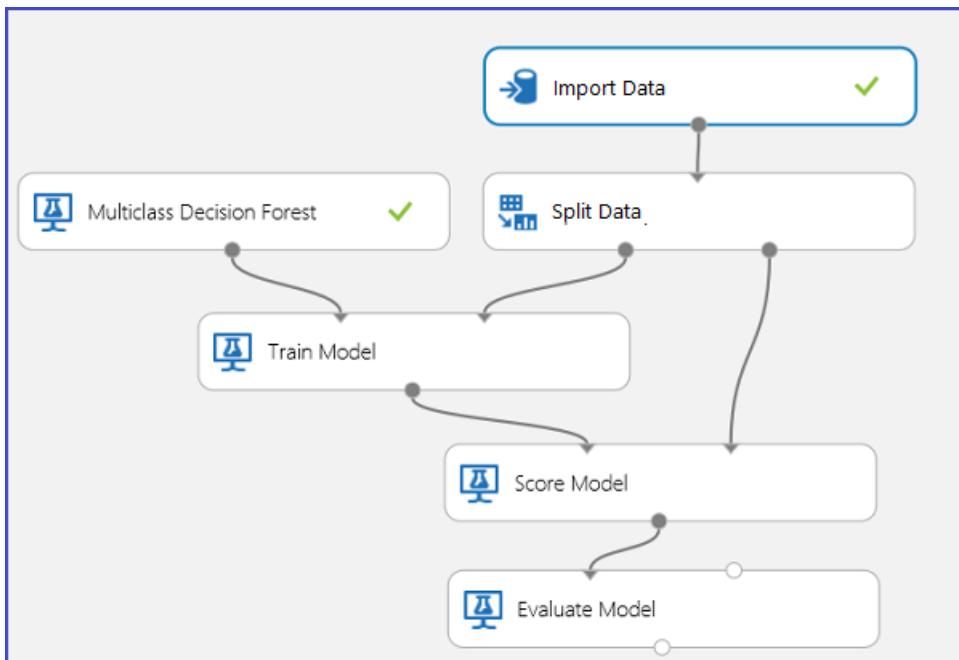


Figure 10. Evaluating a Multiclass Classifier

Inspecting the Evaluation Results

Run the experiment and click on the output port of [Evaluate Model](#). The evaluation results are presented in the form of a confusion matrix, in this case. The matrix shows the actual vs. predicted instances for all 3 classes.

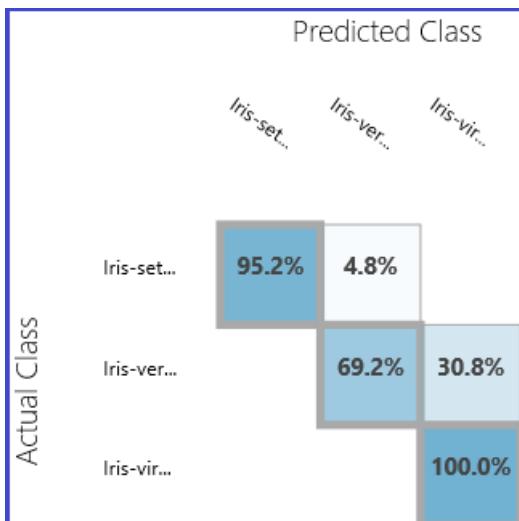


Figure 11. Multiclass Classification Evaluation Results.

Using Cross Validation

As mentioned earlier, you can perform repeated training, scoring and evaluations automatically using the [Cross-Validate Model](#) module. You would need a dataset, an untrained model, and a [Cross-Validate Model](#) module (see figure below). Again you need to set the label column of the [Cross-Validate Model](#) module (column index 5 in this case). After running the experiment and clicking the right output port of the [Cross-Validate Model](#), you can inspect the metric values for each fold as well as the mean and standard deviation. The metrics displayed here are the similar to the ones discussed in the binary classification case. However, note that in multiclass classification, computing the true positives/negatives and false positives/negatives is done by counting on a per-class basis, as there is no overall positive or negative class. For example, when computing the precision or recall of the 'Iris-setosa' class, it is assumed that this is the positive class and all others as negative.

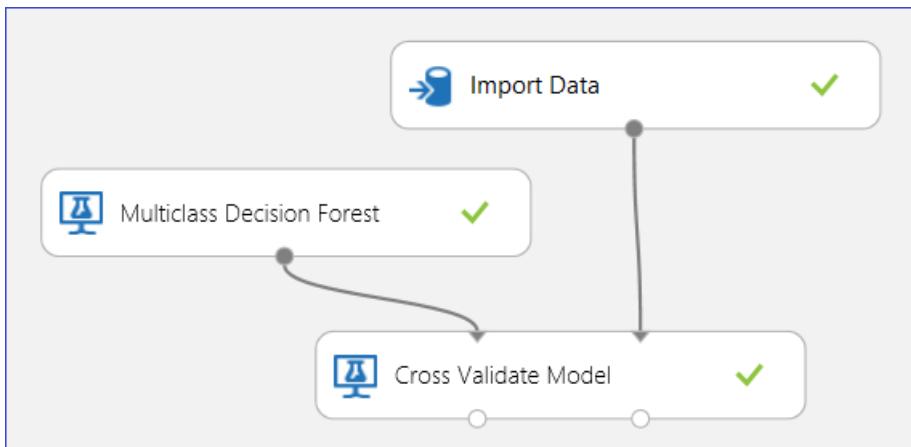


Figure 12. Cross-Validating a Multiclass Classification Model.

rows	columns												
		Fold Number	Number of examples in fold	Model	Average Log Loss for Class "Iris-setosa"	Precision for Class "Iris-setosa"	Recall for Class "Iris-setosa"	Average Log Loss for Class "Iris-versicolor"	Precision for Class "Iris-versicolor"	Recall for Class "Iris-versicolor"	Average Log Loss for Class "Iris-virginica"	Precision for Class "Iris-virginica"	Recall for Class "Iris-virginica"
view as				Microsoft.Analytics.Modules.Gemin.III.MulticlassGbm.miniDecisionForestClassifier	0	1	1	0.415888	1	0.8	0	0.857143	1
0	15			Microsoft.Analytics.Modules.Gemin.III.MulticlassGbm.miniDecisionForestClassifier	0	1	1	0.026706	0.833333	1	0.122604	1	0.875
1	15			Microsoft.Analytics.Modules.Gemin.III.MulticlassGbm.miniDecisionForestClassifier	0.057536	1	1	0.026706	1	1	0.057536	1	1
2	15			Microsoft.Analytics.Modules.Gemin.III.MulticlassGbm.miniDecisionForestClassifier	0.057536	1	1	0.026706	1	1	0.057536	1	1

Figure 13. Cross-Validation Results of a Multiclass Classification Model.

Choose parameters to optimize your algorithms in Azure Machine Learning

1/17/2017 • 3 min to read • [Edit on GitHub](#)

This topic describes how to choose the right hyperparameter set for an algorithm in Azure Machine Learning. Most machine learning algorithms have parameters to set. When you train a model, you need to provide values for those parameters. The efficacy of the trained model depends on the model parameters that you choose. The process of finding the optimal set of parameters is known as *model selection*.

NOTE

Try Azure Machine Learning for free

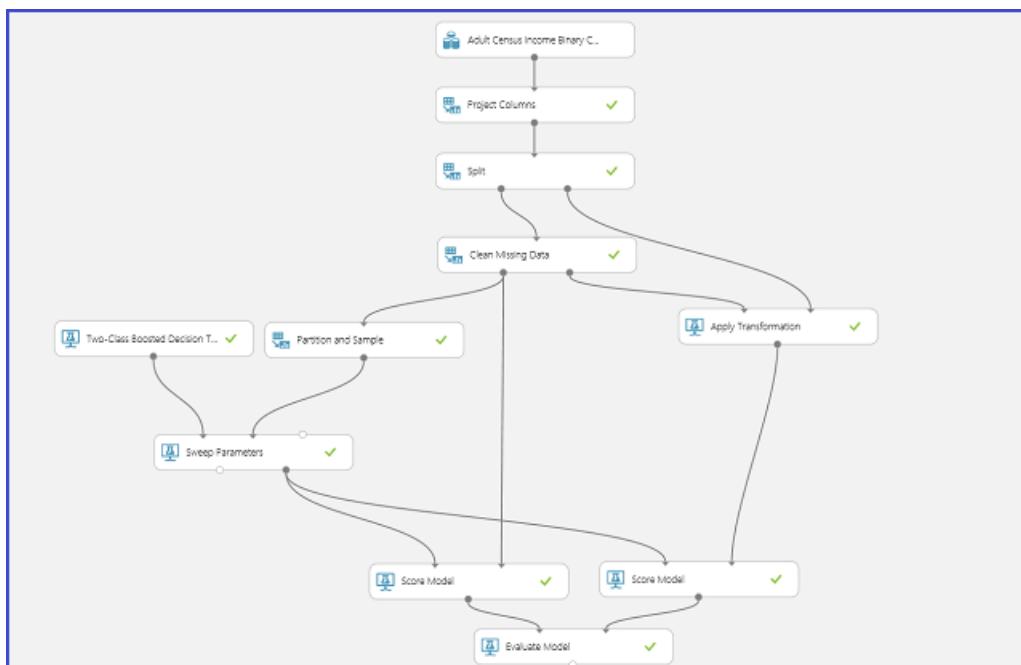
No credit card or Azure subscription needed. [Get started now >](#)

There are various ways to do model selection. In machine learning, cross-validation is one of the most widely used methods for model selection, and it is the default model selection mechanism in Azure Machine Learning. Because Azure Machine Learning supports both R and Python, you can always implement their own model selection mechanisms by using either R or Python.

There are four steps in the process of finding the best parameter set:

1. **Define the parameter space:** For the algorithm, first decide the exact parameter values you want to consider.
2. **Define the cross-validation settings:** Decide how to choose cross-validation folds for the dataset.
3. **Define the metric:** Decide what metric to use for determining the best set of parameters, such as accuracy, root mean squared error, precision, recall, or f-score.
4. **Train, evaluate, and compare:** For each unique combination of the parameter values, cross-validation is carried out by and based on the error metric you define. After evaluation and comparison, you can choose the best-performing model.

The following image illustrates shows how this can be achieved in Azure Machine Learning.



Define the parameter space

You can define the parameter set at the model initialization step. The parameter pane of all machine learning algorithms has two trainer modes: *Single Parameter* and *Parameter Range*. Choose Parameter Range mode. In Parameter Range mode, you can enter multiple values for each parameter. You can enter comma-separated values in the text box.

▲ Two-Class Boosted Decision Tree

Create trainer mode

Single Parameter

Maximum number of leaves per tree
20

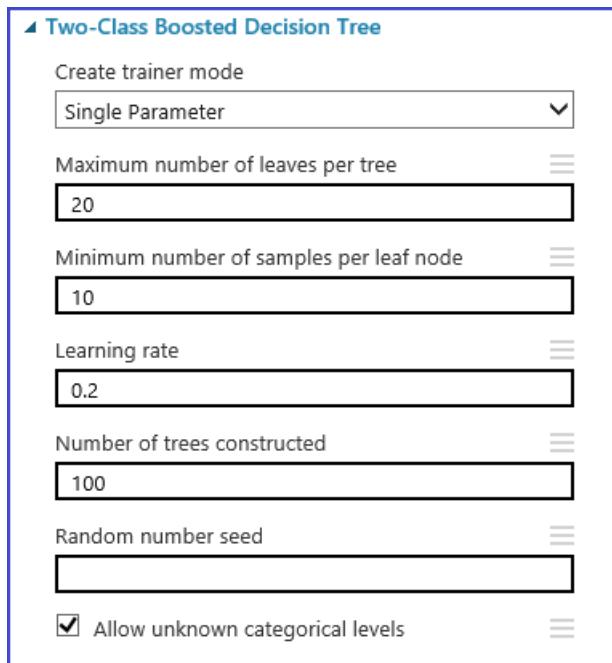
Minimum number of samples per leaf node
10

Learning rate
0.2

Number of trees constructed
100

Random number seed

Allow unknown categorical levels



Alternately, you can define the maximum and minimum points of the grid and the total number of points to be generated with **Use Range Builder**. By default, the parameter values are generated on a linear scale. But if **Log Scale** is checked, the values are generated in the log scale (that is, the ratio of the adjacent points is constant instead of their difference). For integer parameters, you can define a range by using a hyphen. For example, "1-10" means that all integers between 1 and 10 (both inclusive) form the parameter set. A mixed mode is also supported. For example, the parameter set "1-10, 20, 50" would include integers 1-10, 20, and 50.

▲ Two-Class Boosted Decision Tree

Create trainer mode

Parameter Range

Maximum number of leaves per tree
Use Range Builder
Parameter Range : 101 - 900


Number of points : 3

Log Scale

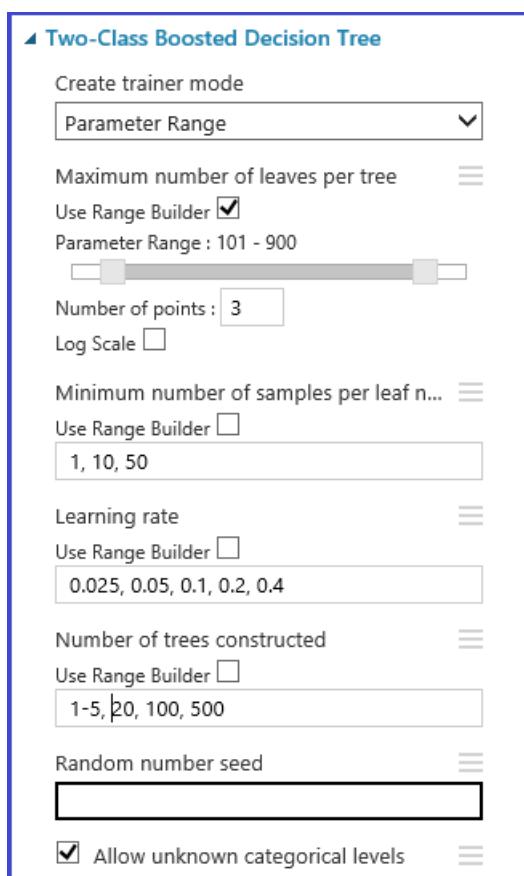
Minimum number of samples per leaf n...
Use Range Builder
1, 10, 50

Learning rate
Use Range Builder
0.025, 0.05, 0.1, 0.2, 0.4

Number of trees constructed
Use Range Builder
1-5, 20, 100, 500

Random number seed

Allow unknown categorical levels



Define cross-validation folds

The [Partition and Sample](#) module can be used to randomly assign folds to the data. In the following sample configuration for the module, we define five folds and randomly assign a fold number to the sample instances.

The screenshot shows the configuration interface for the Partition and Sample module. It includes fields for partition mode (set to 'Assign to Folds'), random seed (set to 0), partitioner method (set to 'Partition evenly'), and the number of folds (set to 5). The 'Randomized split' checkbox is checked. A 'Stratified split' dropdown is set to 'False'.

Define the metric

The [Tune Model Hyperparameters](#) module provides support for empirically choosing the best set of parameters for a given algorithm and dataset. In addition to other information regarding training the model, the **Properties** pane of this module includes the metric for determining the best parameter set. It has two different drop-down list boxes for classification and regression algorithms, respectively. If the algorithm under consideration is a classification algorithm, the regression metric is ignored and vice versa. In this specific example, the metric is **Accuracy**.

The screenshot shows the 'Sweep Parameters' section of the Tune Model Hyperparameters module. It includes a dropdown for 'Specify parameter sweeping mode' (set to 'Entire grid'), a 'Label column' field containing 'Selected columns: Column names: income' (with a 'Launch column selector' button), and dropdowns for 'Metric for measuring performance for classification' (set to 'Accuracy') and 'Metric for measuring performance for regression' (set to 'Mean absolute error').

Train, evaluate, and compare

The same [Tune Model Hyperparameters](#) module trains all the models that correspond to the parameter set, evaluates various metrics, and then creates the best-trained model based on the metric you choose. This module has two mandatory inputs:

- The untrained learner
- The dataset

The module also has an optional dataset input. Connect the dataset with fold information to the mandatory dataset input. If the dataset is not assigned any fold information, then a 10-fold cross-validation is automatically executed by default. If the fold assignment is not done and a validation dataset is provided at the optional dataset port, then a train-test mode is chosen and the first dataset is used to train the model for each parameter combination.

Boosted Decision Tree Classifier

Settings

Setting	Value
Number Of Leaves	8
Minimum Leaf Instances	10
Learning Rate	0.1
Number Of Trees	500
Allow Unknown Levels	true
Random Number Seed	

The model is then evaluated on the validation dataset. The left output port of the module shows different metrics as functions of parameter values. The right output port gives the trained model that corresponds to the best-performing model according to the chosen metric (**Accuracy** in this case).

rows	columns										
180	11										
view as	Number of leaves	Minimum leaf instances	Learning rate	Number of trees	Accuracy	Precision	Recall	F-Score	AUC	Average Log Loss	Training Log Loss
grid	8	10	0.1	500	0.866128	0.75785	0.652595	0.701295	0.922837	0.290561	47.363286
grid	8	1	0.05	500	0.866097	0.762796	0.644306	0.698562	0.922672	0.289389	47.575474
grid	8	1	0.2	100	0.865944	0.763813	0.641755	0.697484	0.921469	0.291591	47.176649
grid	32	1	0.05	100	0.865913	0.759367	0.648769	0.699725	0.921503	0.29273	46.970281
grid	8	10	0.2	100	0.865882	0.763821	0.641372	0.697262	0.922052	0.290024	47.46048
grid	32	1	0.025	500	0.865668	0.749604	0.663946	0.70418	0.923187	0.292484	47.014813
grid	8	10	0.05	500	0.865514	0.761322	0.643158	0.697269	0.922803	0.288929	47.658898
grid	32	1	0.1	100	0.865453	0.751234	0.659737	0.702519	0.922703	0.292779	46.961499
grid	8	1	0.025	500	0.865391	0.764778	0.636909	0.695011	0.921386	0.29142	47.207519
grid	8	50	0.05	500	0.865115	0.760932	0.641372	0.696055	0.92261	0.288991	47.647701
grid	32	10	0.025	500	0.865115	0.748308	0.662798	0.702962	0.923435	0.29145	47.202173
grid	8	50	0.025	500	0.865084	0.764336	0.635761	0.694145	0.921226	0.291554	47.183315
grid	32	10	0.05	100	0.865084	0.756395	0.648642	0.698387	0.92155	0.29239	47.031913
grid	8	10	0.025	500	0.864992	0.764066	0.635633	0.693957	0.921674	0.290844	47.312033
grid	32	10	0.2	20	0.864961	0.757939	0.645326	0.697114	0.920488	0.294732	46.607674
grid	8	1	0.1	500	0.864808	0.752756	0.653105	0.699399	0.922545	0.291764	47.145278
grid	32	10	0.1	100	0.864777	0.748626	0.66012	0.701593	0.92295	0.291274	47.234005
grid	8	1	0.1	100	0.864715	0.763255	0.635251	0.693395	0.920514	0.293092	46.904731
grid	8	10	0.1	100	0.864715	0.763416	0.634996	0.693309	0.920639	0.292815	46.954929
grid	8	10	0.4	100	0.864623	0.754863	0.648387	0.697585	0.921302	0.292915	46.936722
grid	32	1	0.2	20	0.864439	0.757089	0.64354	0.695712	0.920124	0.295428	46.481506

You can see the exact parameters chosen by visualizing the right output port. This model can be used in scoring a test set or in an operationalized web service after saving as a trained model.

Interpret model results in Azure Machine Learning

1/17/2017 • 13 min to read • [Edit on GitHub](#)

This topic explains how to visualize and interpret prediction results in Azure Machine Learning Studio. After you have trained a model and done predictions on top of it ("scored the model"), you need to understand and interpret the prediction result.

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

There are four major kinds of machine learning models in Azure Machine Learning:

- Classification
- Clustering
- Regression
- Recommender systems

The modules used for prediction on top of these models are:

- [Score Model](#) module for classification and regression
- [Assign to Clusters](#) module for clustering
- [Score Matchbox Recommender](#) for recommendation systems

This document explains how to interpret prediction results for each of these modules. For an overview of these modules, see [How to choose parameters to optimize your algorithms in Azure Machine Learning](#).

This topic addresses prediction interpretation but not model evaluation. For more information about how to evaluate your model, see [How to evaluate model performance in Azure Machine Learning](#).

If you are new to Azure Machine Learning and need help creating a simple experiment to get started, see [Create a simple experiment in Azure Machine Learning Studio](#) in Azure Machine Learning Studio.

Classification

There are two subcategories of classification problems:

- Problems with only two classes (two-class or binary classification)
- Problems with more than two classes (multi-class classification)

Azure Machine Learning has different modules to deal with each of these types of classification, but the methods for interpreting their prediction results are similar.

Two-class classification

Example experiment

An example of a two-class classification problem is the classification of iris flowers. The task is to classify iris flowers based on their features. The Iris data set provided in Azure Machine Learning is a subset of the popular [Iris data set](#) containing instances of only two flower species (classes 0 and 1). There are four features for each flower (sepal length, sepal width, petal length, and petal width).

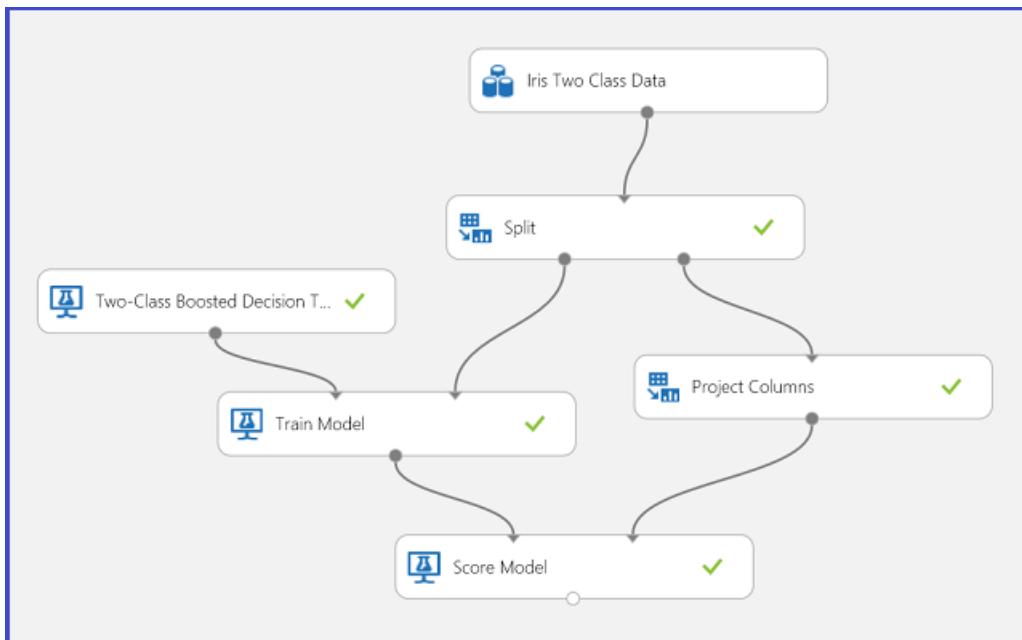
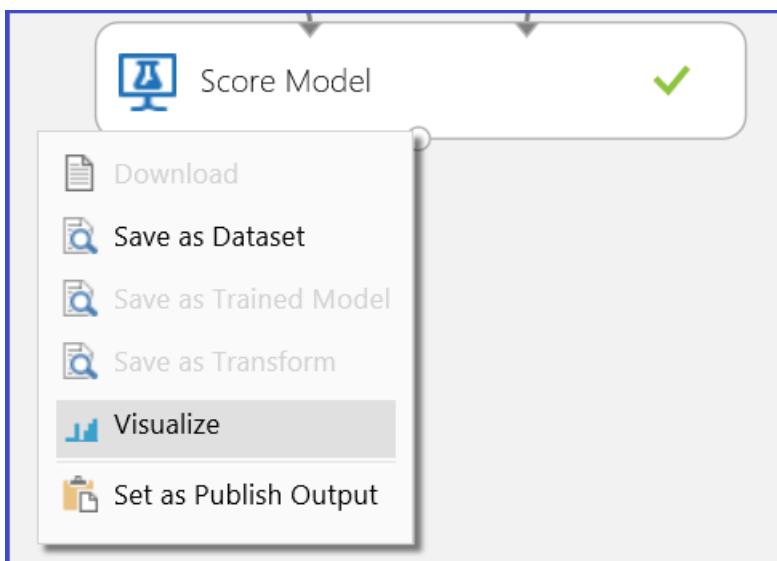


Figure 1. Iris two-class classification problem experiment

An experiment has been performed to solve this problem, as shown in Figure 1. A two-class boosted decision tree model has been trained and scored. Now you can visualize the prediction results from the **Score Model** module by clicking the output port of the **Score Model** module and then clicking **Visualize**.



This brings up the scoring results as shown in Figure 2.

Interpreting Score Module - Two-class > Score Model > Scored dataset							
row	column	sepal-length	sepal-width	petal-length	petal-width	Scored Labels	Scored Probabilities
40	sepal-length	4.4	3	1.3	0.2	0	0.028571
5.8	sepal-width	2.7	5.1	1.9	1	1	0.965517
6.4	petal-length	3.2	5.3	2.3	1	1	0.965517
5	petal-width	3	1.6	0.2	0	0	0.028571
4.7	Scored Labels	3.2	1.6	0.2	0	0	0.028571
5	Scored Probabilities	3.2	1.2	0.2	0	0	0.028571
6.3	sepal-length	2.9	5.6	1.8	1	1	0.965517
5.2	sepal-width	3.5	1.5	0.2	0	0	0.028571
6.1	petal-length	2.6	5.6	1.4	1	1	0.965517
6.3	petal-width	2.5	5	1.9	1	1	0.965517
5.8	Scored Labels	2.8	5.1	2.4	1	1	0.965517
6.4	Scored Probabilities	2.7	5.3	1.9	1	1	0.965517
6.5	sepal-length	3	5.2	2	1	1	0.965517
7.2	sepal-width	3.6	6.1	2.5	1	1	0.965517
6.3	petal-length	2.8	5.1	1.5	1	1	0.965517
6.4	petal-width	2.8	5.6	2.2	1	1	0.965517
5.1	Scored Labels	3.8	5.1	0.3	0	0	0.028571
6.9	Scored Probabilities	3.1	5.1	2.3	1	1	0.965517
4.4	sepal-length	3.2	1.3	0.2	0	0	0.028571
5.5	sepal-width	4.2	1.4	0.2	0	0	0.028571

To create a graph, select a column in the table

- [Statistics](#)
- [Visualizations](#)

Figure 2. Visualize a score model result in two-class classification

Result interpretation

There are six columns in the results table. The left four columns are the four features. The right two columns, Scored Labels and Scored Probabilities, are the prediction results. The Scored Probabilities column shows the probability that a flower belongs to the positive class (Class 1). For example, the first number in the column (0.028571) means there is 0.028571 probability that the first flower belongs to Class 1. The Scored Labels column shows the predicted class for each flower. This is based on the Scored Probabilities column. If the scored probability of a flower is larger than 0.5, it is predicted as Class 1. Otherwise, it is predicted as Class 0.

Web service publication

After the prediction results have been understood and judged sound, the experiment can be published as a web service so that you can deploy it in various applications and call it to obtain class predictions on any new iris flower. To learn how to change a training experiment into a scoring experiment and publish it as a web service, see [Publish the Azure Machine Learning web service](#). This procedure provides you with a scoring experiment as shown in Figure 3.

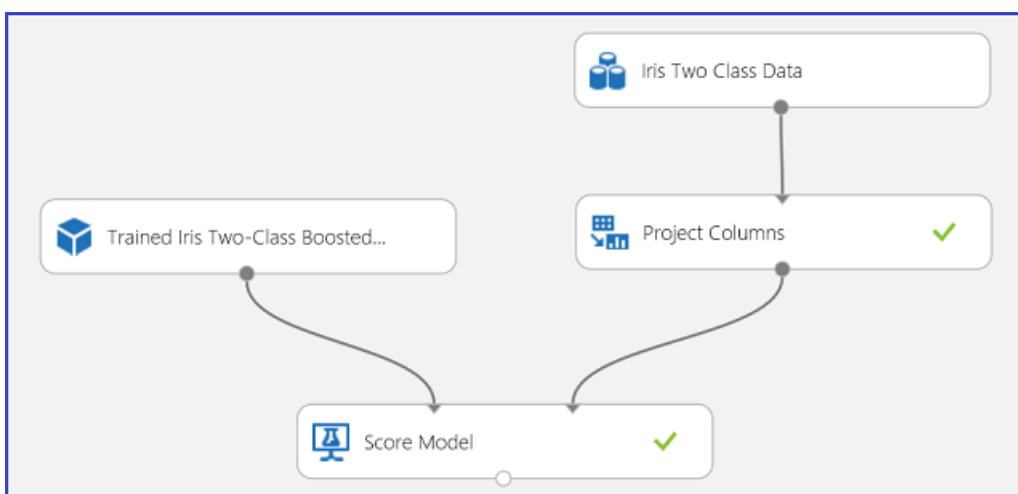


Figure 3. Scoring the iris two-class classification problem experiment

Now you need to set the input and output for the web service. The input is the right input port of [Score Model](#), which is the Iris flower features input. The choice of the output depends on whether you are interested in the predicted class (scored label), the scored probability, or both. In this example, it is assumed that you are interested in both. To select the desired output columns, use a [Select Columns in Data set](#) module. Click [Select Columns in Data set](#), click **Launch column selector**, and select **Scored Labels** and **Scored Probabilities**. After setting the output port of [Select Columns in Data set](#) and running it again, you should be ready to publish the scoring experiment as a web service by clicking **PUBLISH WEB SERVICE**. The final experiment looks like Figure 4.

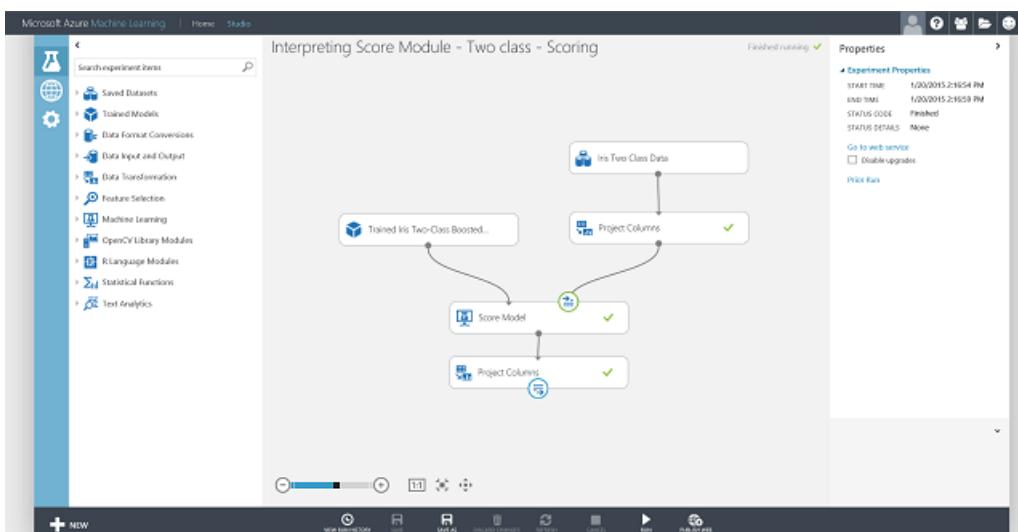


Figure 4. Final scoring experiment of an iris two-class classification problem

After you run the web service and enter some feature values of a test instance, the result returns two numbers. The first number is the scored label, and the second is the scored probability. This flower is predicted as Class 1 with 0.9655 probability.

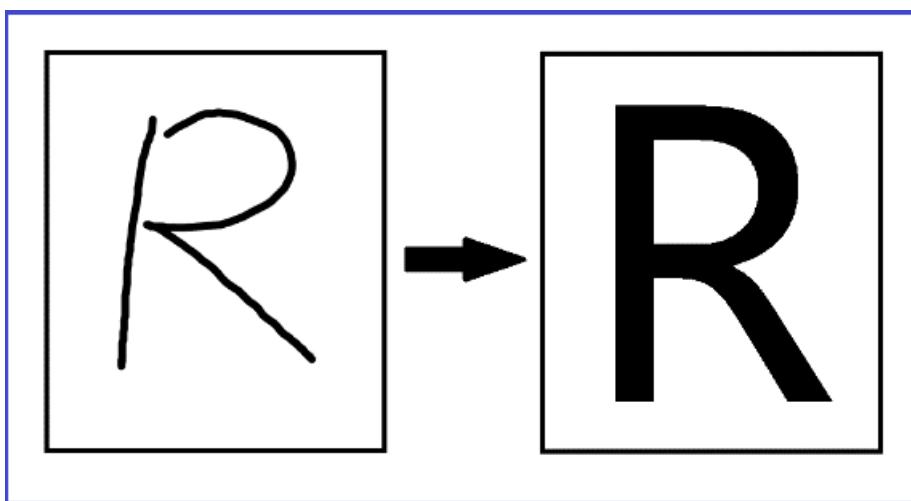
The screenshot shows a user interface for a 'Test Interpreting Score Module - Two class - Scoring Service'. At the top, it says 'Enter data to predict'. Below are four input fields for 'SEPAL-LENGTH' (value 6), 'SEPAL-WIDTH' (value 3), 'PETAL-LENGTH' (value 5), and 'PETAL-WIDTH' (value 2). A large green checkmark icon is located at the bottom right of the input area. Below the input form, a dark grey box displays the results: a left arrow followed by the text "'Interpreting Score Module - Two class - Scoring' test finished successfully", and a green checkmark followed by the text 'Result: 1,0.965517282485962'.

Figure 5. Web service result of iris two-class classification

Multi-class classification

Example experiment

In this experiment, you perform a letter-recognition task as an example of multiclass classification. The classifier attempts to predict a certain letter (class) based on some hand-written attribute values extracted from the hand-written images.



In the training data, there are 16 features extracted from hand-written letter images. The 26 letters form our 26 classes. Figure 6 shows an experiment that will train a multiclass classification model for letter recognition and predict on the same feature set on a test data set.

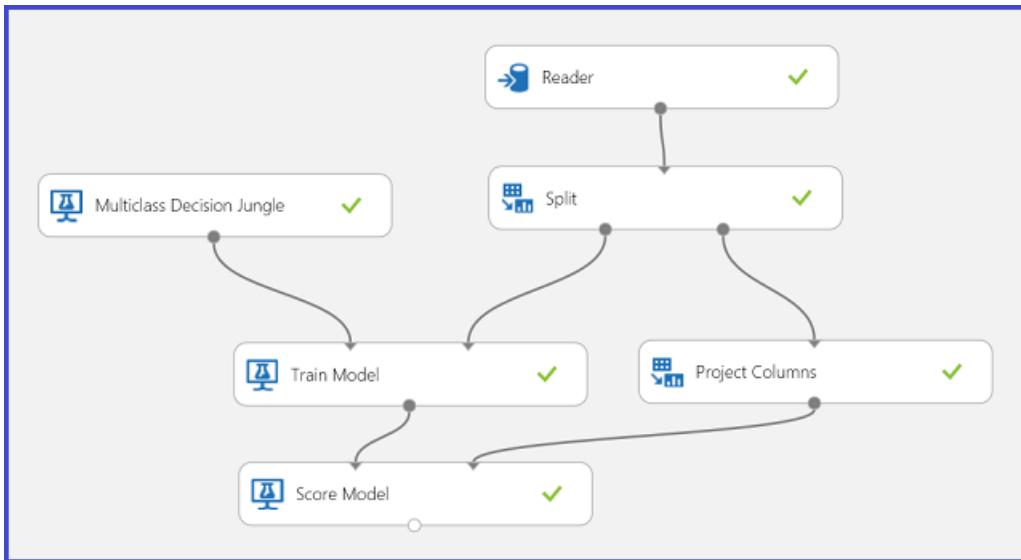


Figure 6. Letter recognition multiclass classification problem experiment

Visualizing the results from the [Score Model](#) module by clicking the output port of [Score Model](#) module and then clicking **Visualize**, you should see content as shown in Figure 7.

Col16	Col17	Scored Probabilities for Class "A"	Scored Probabilities for Class "B"	Scored Probabilities for Class "C"	Scored Probabilities for Class "D"	Scored Probabilities for Class "E"	Scored Probabilities for Class "F"	Scored Probabilities for Class "G"	Scored Probabilities for Class "H"	Scored Probabilities for Class "I"	Scored Probabilities for Class "J"
2	6	0.003571	0.000451	0	0.00119	0	0.916995	0	0	0.01228	0.00169
8	5	0.077476	0	0	0	0.04205	0.035784	0	0.027099	0.029174	0.01563
6	12	0	0.236149	0.006445	0.0731	0.005016	0.01138	0.038975	0.06729	0.008711	0.01713
3	7	0.000984	0	0	0.000434	0	0.001416	0	0.02798	0.00623	0.0025
0	8	0	0	0	0	0	0	0	0.000718	0.997776	0.00113
8	6	0.031685	0.035017	0.002778	0.021153	0.008333	0.093765	0.002778	0.001265	0.005515	0.014991
1	5	0.001012	0	0	0.000817	0	0.00458	0.001634	0	0	0
4	8	0.000984	0	0	0.000434	0	0.000868	0	0.08663	0.00125	0
8	9	0.004996	0.000273	0	0.003841	0	0.063475	0.009544	0.012195	0.090871	0.566711
4	7	0	0	0	0.002725	0	0.005265	0	0.000883	0.00433	0
2	5	0	0	0	0	0	0	0	0	0	0
11	6	0.012967	0.072156	0.096037	0.240164	0	0.134389	0.010402	0.020935	0.061644	0.007225
4	8	0	0	0	0.013801	0	0.009321	0.001582	0.002466	0.00433	0
9	9	0.013273	0.084488	0	0.018118	0.006148	0.023945	0.026386	0.034539	0.06094	0.069495
3	11	0.005247	0	0.124714	0.000012	0.012192	0	0.002435	0.053353	0	0.001046
6	9	0.009623	0.447126	0.000534	0.123377	0.003	0.007448	0.005077	0.115194	0.004	0.003
7	7	0.001147	0.024564	0.022777	0.004237	0.049034	0	0.546592	0.045465	0	0
n	n	n	n	0.000411	0.000191	n	0.003282	0.000319	n	n	n

Figure 7. Visualize score model results in a multi-class classification

Result interpretation

The left 16 columns represent the feature values of the test set. The columns with names like Scored Probabilities for Class "XX" are just like the Scored Probabilities column in the two-class case. They show the probability that the corresponding entry falls into a certain class. For example, for the first entry, there is 0.003571 probability that it is an "A," 0.000451 probability that it is a "B," and so forth. The last column (Scored Labels) is the same as Scored Labels in the two-class case. It selects the class with the largest scored probability as the predicted class of the corresponding entry. For example, for the first entry, the scored label is "F" since it has the largest probability to be an "F" (0.916995).

Web service publication

You can also get the scored label for each entry and the probability of the scored label. The basic logic is to find the largest probability among all the scored probabilities. To do this, you need to use the [Execute R Script](#) module. The R code is shown in Figure 8, and the result of the experiment is shown in Figure 9.

```

1 # Map 1-based optional input ports to variables
2 dataset <- maml.mapInputPort(1) # class: data.frame
3
4 # Get the Scored Labels and the corresponding probabilities
5 data.set = data.frame('Scored Labels'=dataset['Scored Labels'], 'Label Probability'=apply(dataset[,17:42], 1, max))
6
7 # Select data.frame to be sent to the output Dataset port
8 maml.mapOutputPort("data.set");

```

Figure 8. R code for extracting Scored Labels and the associated probabilities of the labels

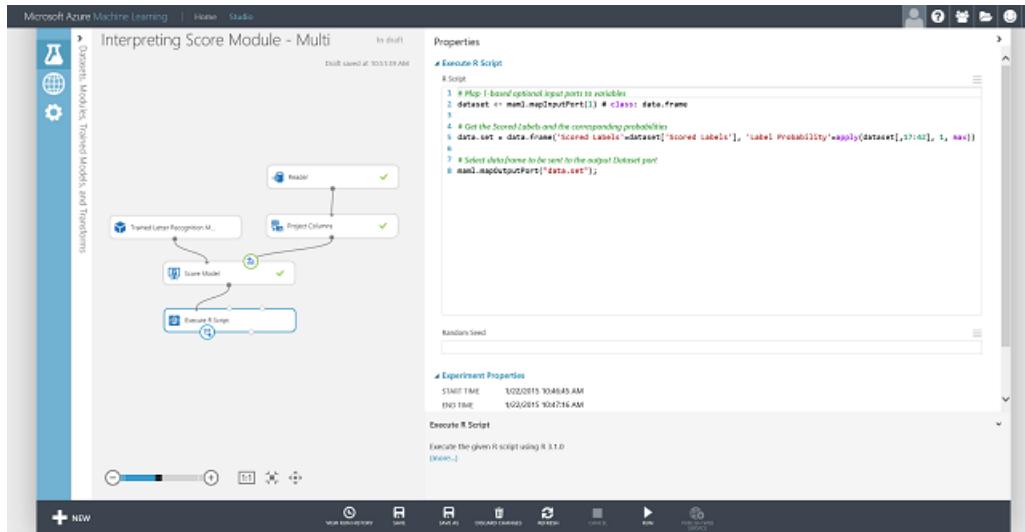


Figure 9. Final scoring experiment of the letter-recognition multiclass classification problem

After you publish and run the web service and enter some input feature values, the returned result looks like Figure 10. This hand-written letter, with its extracted 16 features, is predicted to be a "T" with 0.9715 probability.

Figure 10. Web service result of multiclass classification

Regression

Regression problems are different from classification problems. In a classification problem, you're trying to predict discrete classes, such as which class an iris flower belongs to. But as you can see in the following example of a

regression problem, you're trying to predict a continuous variable, such as the price of a car.

Example experiment

Use automobile price prediction as your example for regression. You are trying to predict the price of a car based on its features, including make, fuel type, body type, and drive wheel. The experiment is shown in Figure 11.

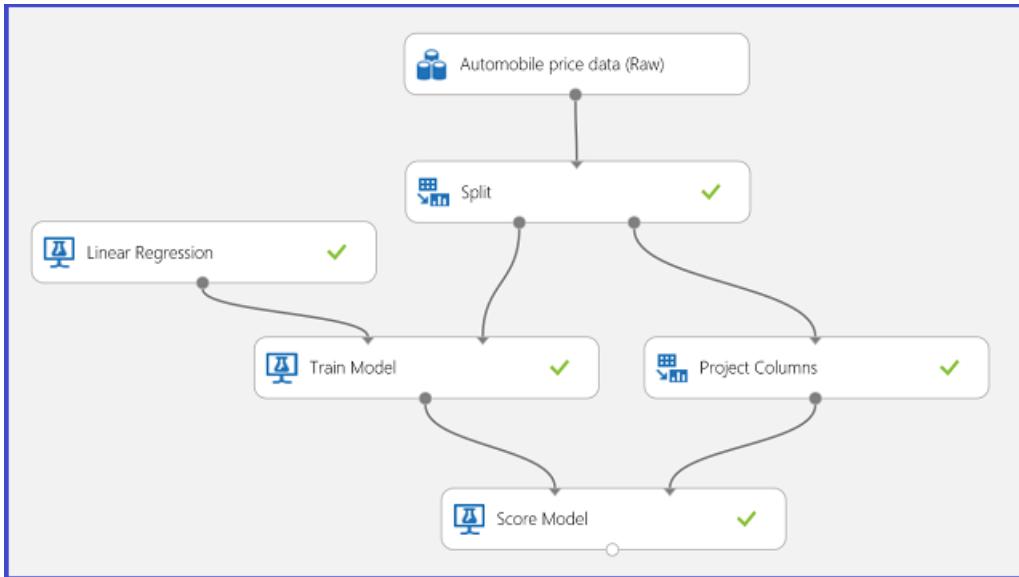


Figure 11. Automobile price regression problem experiment

Visualizing the [Score Model](#) module, the result looks like Figure 12.

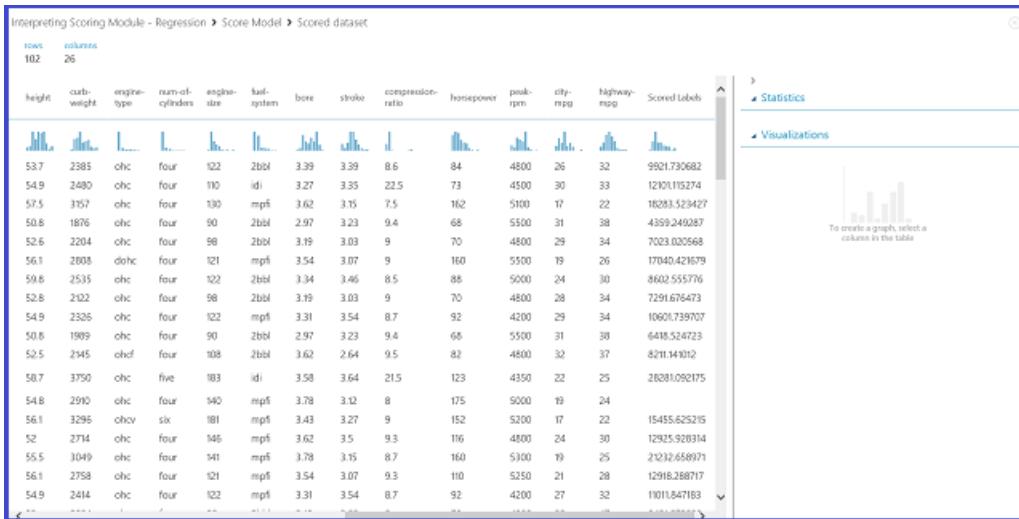


Figure 12. Scoring result for the automobile price prediction problem

Result interpretation

Scored Labels is the result column in this scoring result. The numbers are the predicted price for each car.

Web service publication

You can publish the regression experiment into a web service and call it for automobile price prediction in the same way as in the two-class classification use case.

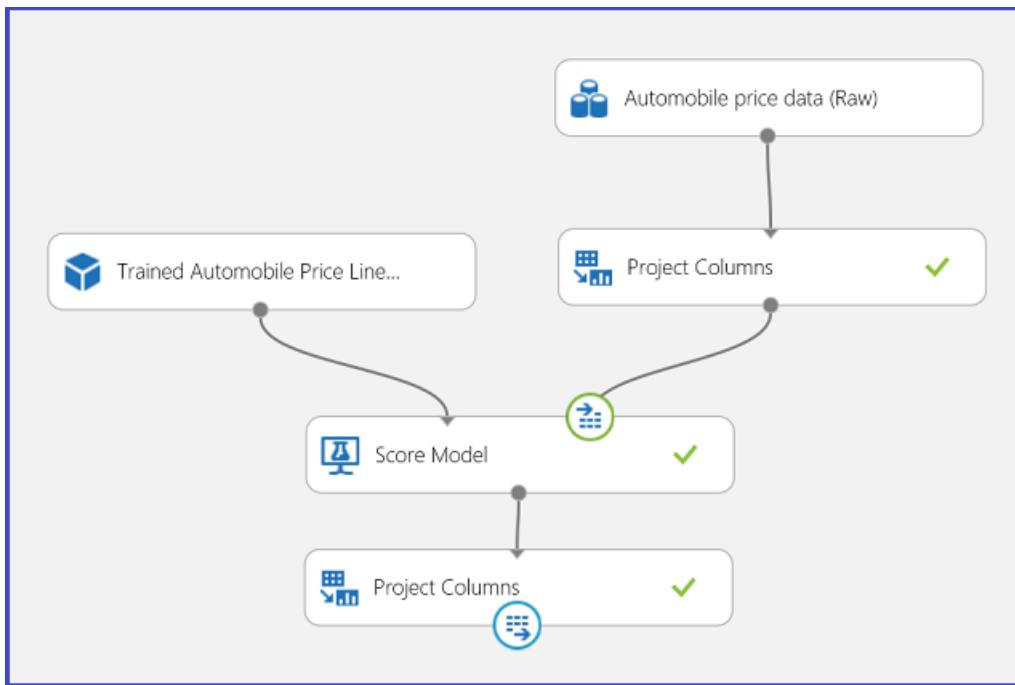


Figure 13. Scoring experiment of an automobile price regression problem

Running the web service, the returned result looks like Figure 14. The predicted price for this car is \$15,085.52.

The figure shows the results of a web service test for an automobile price regression problem. It consists of two main parts:

- Test Interpreting Scoring Module - Regression - Scoring Service**: This is the title of the test window.
- Enter data to predict**: A form with the following fields:
 - SYMBOLING**: Value: 2
 - NORMALIZED-LOSSES**: Value: 164
 - MAKE**: Value: audi
 - FUEL-TYPE**: Value: gas
 - ASPIRATION**: Value: std
- Result**: The predicted price is 15085.5173663723.
- Status**: The test finished successfully.

Figure 14. Web service result of an automobile price regression problem

Clustering

Example experiment

Let's use the Iris data set again to build a clustering experiment. Here you can filter out the class labels in the data set so that it only has features and can be used for clustering. In this iris use case, specify the number of clusters to be two during the training process, which means you would cluster the flowers into two classes. The experiment is shown in Figure 15.

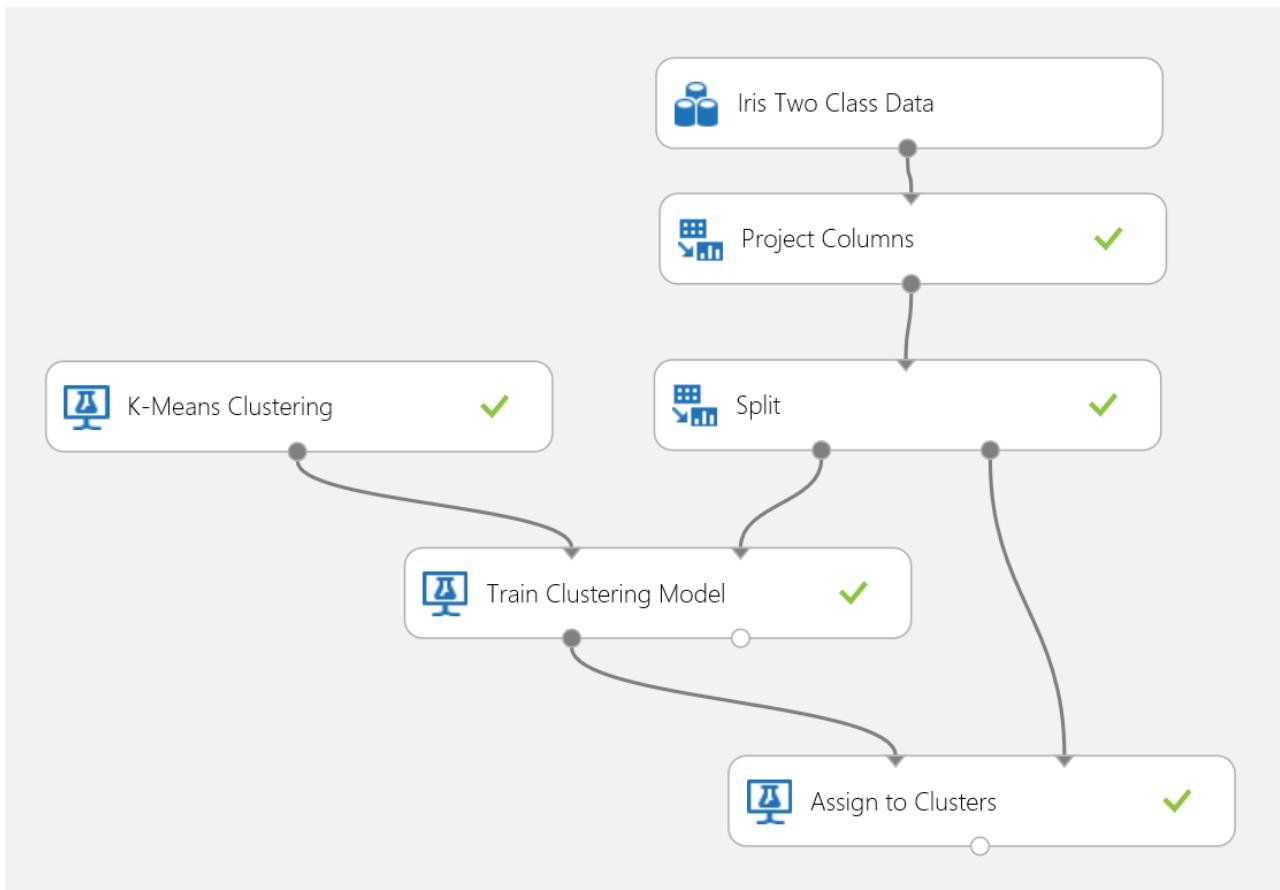


Figure 15. Iris clustering problem experiment

Clustering differs from classification in that the training data set doesn't have ground-truth labels by itself. Clustering groups the training data set instances into distinct clusters. During the training process, the model labels the entries by learning the differences between their features. After that, the trained model can be used to further classify future entries. There are two parts of the result we are interested in within a clustering problem. The first part is labeling the training data set, and the second is classifying a new data set with the trained model.

The first part of the result can be visualized by clicking the left output port of **Train Clustering Model** and then clicking **Visualize**. The visualization is shown in Figure 16.

Interpret Scoring Module - Clustering > Train Clustering Model > Results dataset					
rows	columns				
60	5				
view as	sepal-length	sepal-width	petal-length	petal-width	Assignments
histogram	4.9	3.1	1.5	0.1	1
histogram	5.7	4.4	1.5	0.4	1
histogram	4.8	3.4	1.6	0.2	1
histogram	6.1	3	4.9	1.8	0
histogram	6.4	2.8	5.6	2.1	0
histogram	6.2	3.4	5.4	2.3	0
histogram	5	3.5	1.6	0.6	1
histogram	5.4	3.7	1.5	0.2	1
histogram	6.3	2.7	4.9	1.8	0
histogram	7.6	3	6.6	2.1	0
histogram	7.9	3.8	6.4	2	0
histogram	6.7	3.3	5.7	2.5	0
histogram	4.9	3.6	1.4	0.1	1
histogram	4.9	3	1.4	0.2	1
histogram	7.7	3.8	6.7	2.2	0
histogram	4.5	2.3	1.3	0.3	1
histogram	5.1	3.5	1.4	0.2	1
histogram	5	3.4	1.6	0.4	1
histogram	4.3	3	1.1	0.1	1
histogram	4.9	3.1	1.5	0.2	1

Figure 16. Visualize clustering result for the training data set

The result of the second part, clustering new entries with the trained clustering model, is shown in Figure 17.

Interpret Scoring Module - Clustering > Assign to Clusters > Results dataset

rows	columns				
40	5				
view as	sepal-length	sepal-width	petal-length	petal-width	Assignments
4.4	3	1.3	0.2	1	
5.8	2.7	5.1	1.9	0	
6.4	3.2	5.3	2.3	0	
5	3	1.6	0.2	1	
4.7	3.2	1.6	0.2	1	
5	3.2	1.2	0.2	1	
6.3	2.9	5.6	1.8	0	
5.2	3.5	1.5	0.2	1	
6.1	2.6	5.6	1.4	0	
6.3	2.5	5	1.9	0	
5.8	2.8	5.1	2.4	0	
6.4	2.7	5.3	1.9	0	
6.5	3	5.2	2	0	
7.2	3.6	6.1	2.5	0	
6.3	2.8	5.1	1.5	0	
6.4	2.8	5.6	2.2	0	
5.1	3.8	1.5	0.3	1	
6.9	3.1	5.1	2.3	0	
4.4	3.2	1.3	0.2	1	
5.5	4.2	1.4	0.2	1	

To create a graph, select a column in the table

Figure 17. Visualize clustering result on a new data set

Result interpretation

Although the results of the two parts stem from different experiment stages, they look the same and are interpreted in the same way. The first four columns are features. The last column, Assignments, is the prediction result. The entries assigned the same number are predicted to be in the same cluster, that is, they share similarities in some way (this experiment uses the default Euclidean distance metric). Because you specified the number of clusters to be 2, the entries in Assignments are labeled either 0 or 1.

Web service publication

You can publish the clustering experiment into a web service and call it for clustering predictions the same way as in the two-class classification use case.

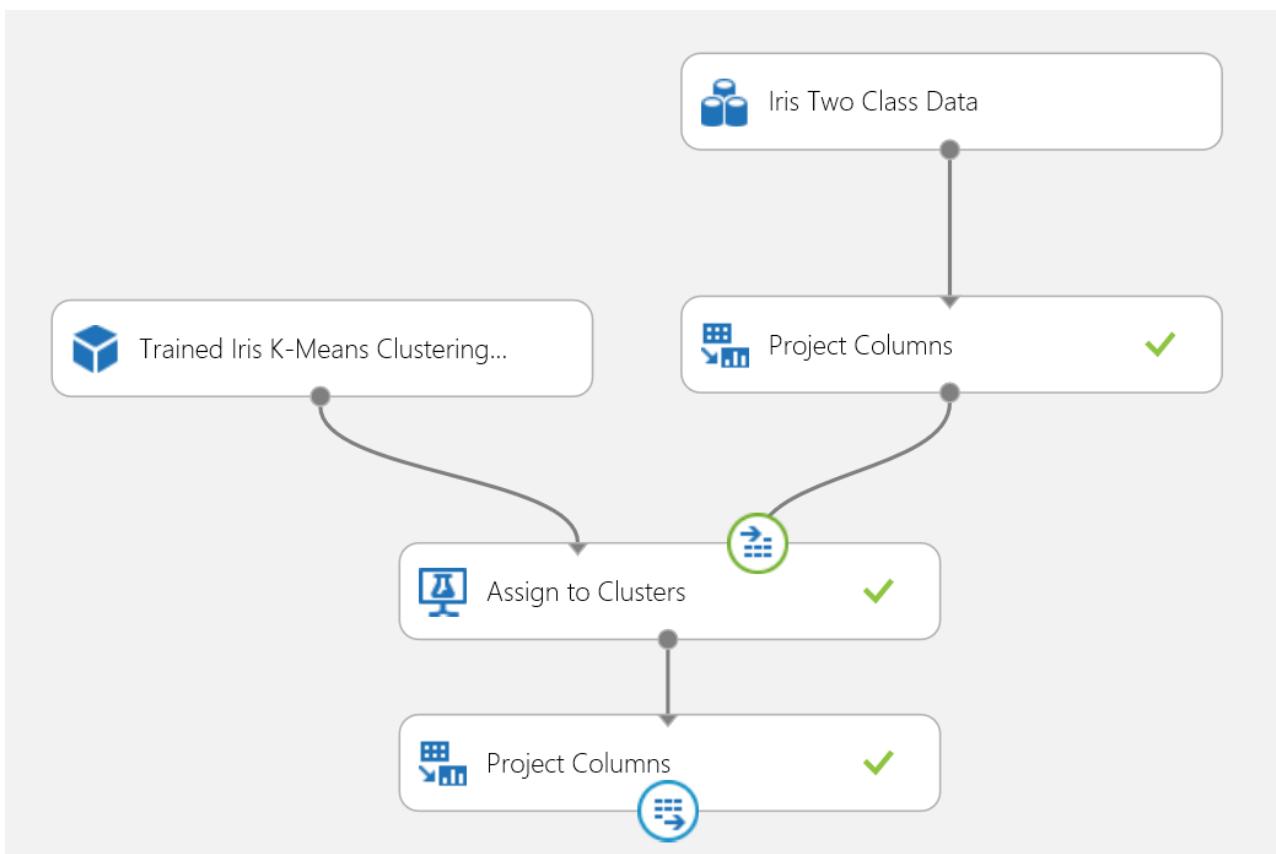


Figure 18. Scoring experiment of an iris clustering problem

After you run the web service, the returned result looks like Figure 19. This flower is predicted to be in cluster 0.

The screenshot shows a web application window titled "Test Interpret Scoring Module - Clustering - Scoring Service". At the top, there is a header "Enter data to predict". Below it, four input fields are displayed, each labeled with a feature name and a numerical value:

- SEPAL-LENGTH: 6
- SEPAL-WIDTH: 3
- PETAL-LENGTH: 5
- PETAL-WIDTH: 2

Below the input fields is a large green button with a checkmark icon. The main content area has a dark grey background and displays two messages:

- ← 'Interpret Scoring Module - Clustering - Scoring' test finished successfully
- ✓ Result: 0

Figure 19. Web service result of iris two-class classification

Recommender system

Example experiment

For recommender systems, you can use the restaurant recommendation problem as an example: you can recommend restaurants for customers based on their rating history. The input data consists of three parts:

- Restaurant ratings from customers
- Customer feature data
- Restaurant feature data

There are several things we can do with the [Train Matchbox Recommender](#) module in Azure Machine Learning:

- Predict ratings for a given user and item
- Recommend items to a given user
- Find users related to a given user
- Find items related to a given item

You can choose what you want to do by selecting from the four options in the **Recommender prediction kind** menu. Here you can walk through all four scenarios.

Properties

Score Matchbox Recommender

Recommender prediction kind

Rating Prediction

Item Recommendation

Related Users

Related Items

ELAPSED TIME 0:00:00.000

STATUS CODE Finished

STATUS DETAILS Task output was present in output cache

A typical Azure Machine Learning experiment for a recommender system looks like Figure 20. For information about how to use those recommender system modules, see [Train matchbox recommender](#) and [Score matchbox recommender](#).

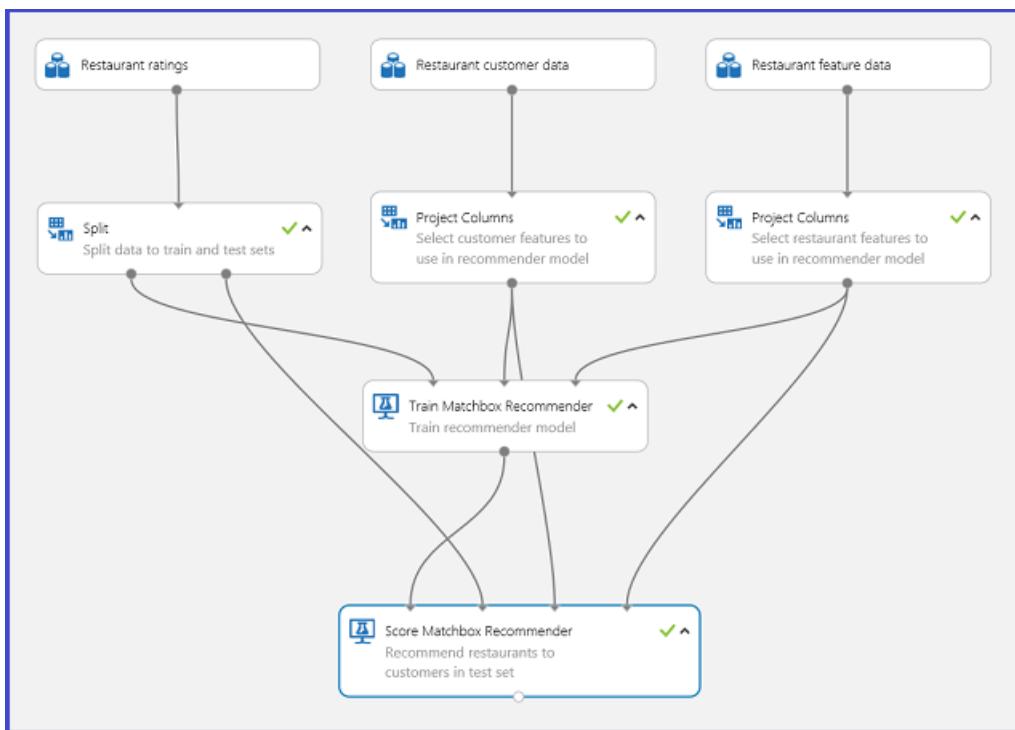


Figure 20. Recommender system experiment

Result interpretation

Predict ratings for a given user and item

By selecting **Rating Prediction** under **Recommender prediction kind**, you are asking the recommender system to predict the rating for a given user and item. The visualization of the **Score Matchbox Recommender** output looks like Figure 21.

Copy of - Sample Experiment: Recommender System > Score Matchbox Recommender > Scored dataset

rows	columns	
418	3	
User	Item	Rating
U1048	135026	2
U1048	132723	2
U1048	135065	2
U1048	135049	0
U1048	135034	2
U1117	135086	2
U1117	135018	2
U1049	132862	0
U1049	135042	0
U1049	135052	0
U1049	135032	0
U1049	135085	0
U1049	132821	0
U1049	135051	0
U1088	132830	1
U1088	135070	2
U1088	135069	2

Figure 21. Visualize the score result of the recommender system--rating prediction

The first two columns are the user-item pairs provided by the input data. The third column is the predicted rating of a user for a certain item. For example, in the first row, customer U1048 is predicted to rate restaurant 135026 as 2.

Recommend items to a given user

By selecting **Item Recommendation** under **Recommender prediction kind**, you're asking the recommender system to recommend items to a given user. The last parameter to choose in this scenario is *Recommended item selection*. The option **From Rated Items (for model evaluation)** is primarily for model evaluation during the training process. For this prediction stage, we choose **From All Items**. The visualization of the [Score Matchbox Recommender](#) output looks like Figure 22.

Copy of - Sample Experiment: Recommender System > Score Matchbox Recommender > Scored dataset

rows	columns				
User	Item 1	Item 2	Item 3	Item 4	Item 5
U1048	134986	135018	134975	135021	132862
U1117	134986	135018	132768	135075	132955
U1049	134986	135018	134976	135021	134975
U1088	135075	135057	132768	134996	132754
U1062	135018	134986	135057	134976	135075
U1035	132768	134986	135075	135018	134996
U1125	134986	135018	132768	135075	132955
U1013	134986	135018	134975	132955	132755
U1042	134986	135018	132768	135075	132922
U1123	134986	135018	132768	135075	132955
U1086	134986	135018	134975	132955	132768
U1006	134986	135018	132768	135075	135057
U1038	134986	135018	135025	132862	134975
U1053	134986	132768	135075	135018	135025
U1031	134986	135018	132768	135075	135057
U1005	134986	135018	132922	132755	132768
U1060	134986	135018	135052	134975	135025

Figure 22. Visualize score result of the recommender system--item recommendation

The first of the six columns represents the given user IDs to recommend items for, as provided by the input data. The other five columns represent the items recommended to the user in descending order of relevance. For example, in the first row, the most recommended restaurant for customer U1048 is 134986, followed by 135018, 134975, 135021, and 132862.

Find users related to a given user

By selecting **Related Users** under **Recommender prediction kind**, you're asking the recommender system to find related users to a given user. Related users are the users who have similar preferences. The last parameter to choose in this scenario is *Related user selection*. The option **From Users That Rated Items (for model evaluation)** is primarily for model evaluation during the training process. Choose **From All Users** for this prediction stage. The visualization of the [Score Matchbox Recommender](#) output looks like Figure 23.

Copy of - Sample Experiment: Recommender System > Score Matchbox Recommender > Scored dataset

rows	columns				
69	6				
User	Related User 1	Related User 2	Related User 3	Related User 4	Related User 5
U1048	U1051	U1066	U1044	U1017	U1072
U1117	U1079	U1136	U1102	U1103	U1001
U1049	U1069	U1070	U1025	U1089	U1013
U1088	U1110	U1058	U1033	U1036	U1093
U1062	U1114	U1015	U1082	U1027	U1129
U1035	U1121	U1020	U1042	U1034	U1103
U1125	U1123	U1043	U1057	U1067	U1060
U1013	U1064	U1086	U1089	U1057	U1043
U1042	U1079	U1083	U1117	U1035	U1102
U1123	U1125	U1043	U1067	U1057	U1060
U1086	U1064	U1089	U1013	U1070	U1057
U1006	U1080	U1107	U1028	U1024	U1031
U1038	U1066	U1087	U1044	U1051	U1092
U1053	U1011	U1099	U1107	U1090	U1092
U1031	U1023	U1029	U1028	U1006	U1089
U1005	U1040	U1087	U1066	U1044	U1052
U1060	U1057	U1057	U1043	U1123	U1125

view as:

Statistics

Visualizations

To create a graph, select a column in the table.

Figure 23. Visualize score results of the recommender system--related users

The first of the six columns shows the given user IDs needed to find related users, as provided by input data. The other five columns store the predicted related users of the user in descending order of relevance. For example, in the first row, the most relevant customer for customer U1048 is U1051, followed by U1066, U1044, U1017, and U1072.

Find items related to a given item

By selecting **Related Items** under **Recommender prediction kind**, you are asking the recommender system to find related items to a given item. Related items are the items most likely to be liked by the same user. The last parameter to choose in this scenario is *Related item selection*. The option **From Rated Items (for model evaluation)** is primarily for model evaluation during the training process. We choose **From All Items** for this prediction stage. The visualization of the **Score Matchbox Recommender** output looks like Figure 24.

Copy of - Sample Experiment: Recommender System > Score Matchbox Recommender > Scored dataset

rows	columns				
109	6				
User	Related Item 1	Related Item 2	Related Item 3	Related Item 4	Related Item 5
135026	135074	135035	132875	135055	134992
132723	135072	132861	135073	135000	132958
135065	132834	135054	135066	135051	132572
135049	135060	132717	135033	132951	135044
135034	134987	134999	132768	134996	135075
135088	132937	135011	135062	132958	135028
135010	132755	134986	134975	132955	134976
132862	134975	135088	135011	132955	135025
135042	132869	135039	135063	132717	132925
135052	135035	132854	135074	135032	135045
135032	135047	135026	135080	135052	135074
135085	135011	132937	132825	135088	132613
132921	135046	135058	132846	132847	135028
135051	135038	135066	135065	135085	132825
132830	135027	135108	132851	132937	135066
135070	135043	132584	132706	132613	135044
135069	132668	132715	132858	132732	132877

view as:

Statistics

Visualizations

To create a graph, select a column in the table.

Figure 24. Visualize score results of the recommender system--related items

The first of the six columns represents the given item IDs needed to find related items, as provided by the input data. The other five columns store the predicted related items of the item in descending order in terms of relevance. For example, in the first row, the most relevant item for item 135026 is 135074, followed by 135035, 132875, 135055, and 134992.

Web service publication

The process of publishing these experiments as web services to get predictions is similar for each of the four scenarios. Here we take the second scenario (recommend items to a given user) as an example. You can follow the same procedure with the other three.

Saving the trained recommender system as a trained model and filtering the input data to a single user ID column as requested, you can hook up the experiment as in Figure 25 and publish it as a web service.

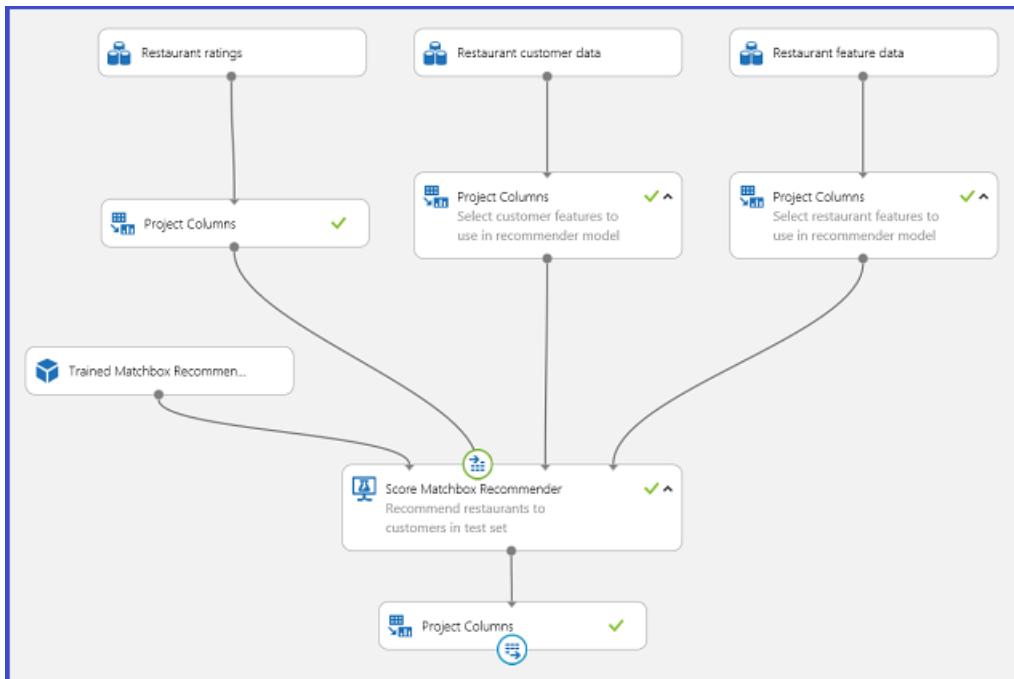


Figure 25. Scoring experiment of the restaurant recommendation problem

Running the web service, the returned result looks like Figure 26. The five recommended restaurants for user U1048 are 134986, 135018, 134975, 135021, and 132862.

The screenshot shows a web service interface:

Test Copy of - Sample Experiment: Recommender System Service

Enter data to predict

USERID: U1048

Result: ↗

Log:

- ← 'Copy of - Sample Experiment: Recommender System' test finished successfully
- ✓ Result: 134986,135018,134975,135021,132862

Figure 26. Web service result of restaurant recommendation problem

Debug your Model in Azure Machine Learning

1/17/2017 • 2 min to read • [Edit on GitHub](#)

This article explains of how to debug your models in Microsoft Azure Machine Learning. Specifically, it covers the potential reasons why either of the following two failures might be encountered when running a model:

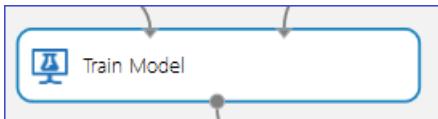
- the [Train Model](#) module produces an error
- the [Score Model](#) module produces incorrect results

NOTE

[Try Azure Machine Learning for free](#)

No credit card or Azure subscription needed. [Get started now >](#)

Train Model Module produces an error



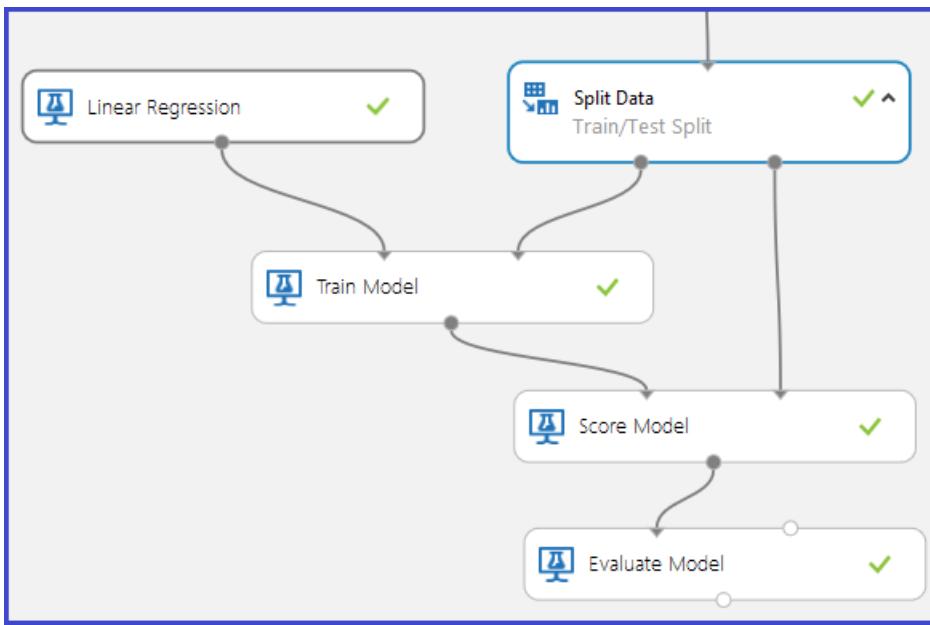
The [Train Model](#) Module expects the following two inputs:

1. The type of Classification/Regression Model from the collection of models provided by Azure Machine Learning
2. The training data with a specified Label column. The Label column specifies the variable to predict. The rest of the columns included are assumed to be Features.

This module produces an error in the following cases:

1. The Label column is specified incorrectly because either more than one column is selected as the Label or an incorrect column index is selected. For example, the second case would apply if a column index of 30 was used with an input dataset which had only 25 columns.
2. The dataset does not contain any Feature columns. For example, if the input dataset has only one column, which is marked as the Label column, there would be no features with which to build the model. In this case, the [Train Model](#) module produces an error.
3. The input dataset (Features or Label) contains Infinity as a value.

Score Model Module does not produce correct results



In a typical training/testing graph for supervised learning, the **Split Data** module divides the original dataset into two parts: the part that is used to train the model and the part that is reserved to score how well the trained model performs on data it did not train on. The trained model is then used to score the test data, after which the results are evaluated to determine the accuracy of the model.

The **Score Model** module requires two inputs:

1. A trained model output from **Train Model** module
2. A scoring dataset that is different from the dataset used to train the model

It may happen that even though the experiment succeeds, the **Score Model** module produces incorrect results.

Several scenarios may cause this to happen:

1. If the specified Label is categorical and a regression model is trained on the data, an incorrect output would be produced by the **Score Model** module. This is because regression requires a continuous response variable. In this case, it would be more suitable to use a classification model.
2. Similarly, if a classification model is trained on a dataset having floating point numbers in the Label column, it may produce undesirable results. This is because classification requires a discrete response variable that only allows values that range over a finite, and usually somewhat small, set of classes.
3. If the scoring dataset does not contain all the features used to train the model, the **Score Model** produces an error.
4. If a row in the scoring dataset contains a missing value or an infinite value for any of its features, the **Score Model** will not produce any output corresponding to that row.
5. The **Score Model** may produce identical outputs for all rows in the scoring dataset. This could occur, for example, when attempting classification using Decision Forests if the minimum number of samples per leaf node is chosen to be more than the number of training examples available.

Extend your experiment with R

1/17/2017 • 22 min to read • [Edit on GitHub](#)

You can extend the functionality of Azure Machine Learning Studio through the R language by using the [Execute R Script](#) module.

This module accepts multiple input datasets and yields a single dataset as output. You can type an R script into the **R Script** parameter of the [Execute R Script](#) module.

You access each input port of the module by using code similar to the following:

```
dataset1 <- maml.mapInputPort(1)
```

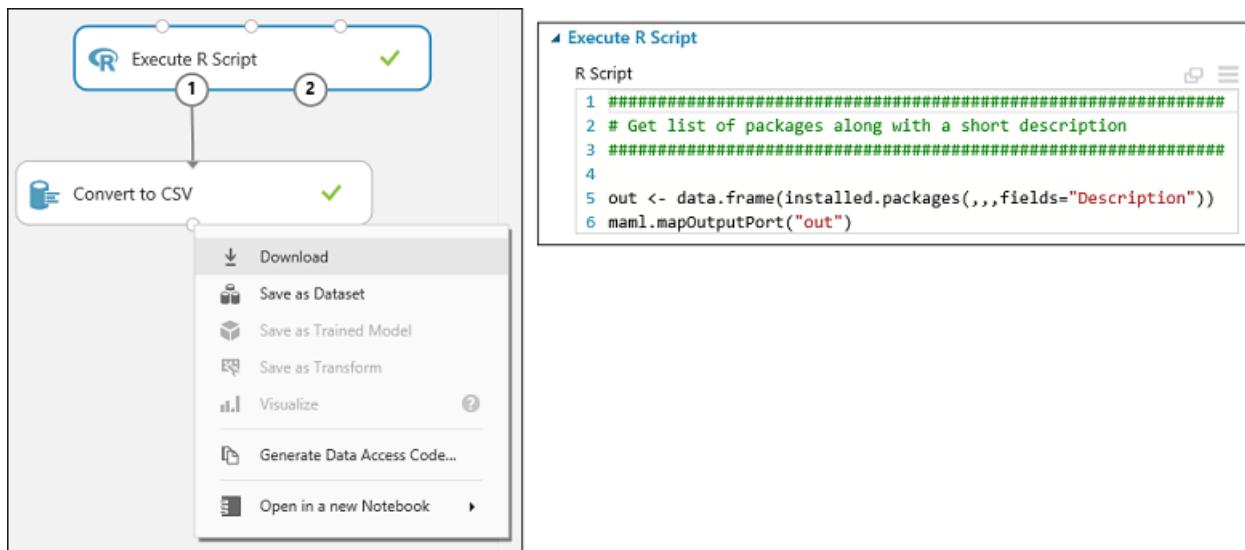
Listing all currently-installed packages

The list of installed packages can change. A list of currently installed packages can be found in [R Packages Supported by Azure Machine Learning](#).

You also can get the complete, current list of installed packages by entering the following code into the [Execute R Script](#) module:

```
out <- data.frame(installed.packages(,,fields="Description"))
maml.mapOutputPort("out")
```

This sends the list of packages to the output port of the [Execute R Script](#) module. To view the package list, connect a conversion module such as [Convert to CSV](#) to the left output of the [Execute R Script](#) module, run the experiment, then click the output of the conversion module and select **Download**.



Importing packages

You can import packages that are not already installed by using the following commands in the [Execute R Script](#) module:

```
install.packages("src/my_favorite_package.zip", lib = ".", repos = NULL, verbose = TRUE)
success <- library("my_favorite_package", lib.loc = ".", logical.return = TRUE, verbose = TRUE)
```

where the `my_favorite_package.zip` file contains your package.

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

Author custom R modules in Azure Machine Learning

1/17/2017 • 16 min to read • [Edit on GitHub](#)

This topic describes how to author and deploy a custom R module in Azure Machine Learning. It explains what custom R modules are and what files are used to define them. It illustrates how to construct the files that define a module and how to register the module for deployment in a Machine Learning workspace. The elements and attributes used in the definition of the custom module are then described in more detail. How to use auxiliary functionality and files and multiple outputs is also discussed.

NOTE

[Try Azure Machine Learning for free](#)

No credit card or Azure subscription needed. [Get started now >](#)

What is a custom R module?

A **custom module** is a user-defined module that can be uploaded to your workspace and executed as part of an Azure Machine Learning experiment. A **custom R module** is a custom module that executes a user-defined R function. R is a programming language for statistical computing and graphics that is widely used by statisticians and data scientists for implementing algorithms. Currently, R is the only language supported in custom modules, but support for additional languages is scheduled for future releases.

Custom modules have **first-class status** in Azure Machine Learning in the sense that they can be used just like any other module. They can be executed with other modules, included in published experiments or in visualizations. You have control over the algorithm implemented by the module, the input and output ports to be used, the modeling parameters, and other various runtime behaviors. An experiment that contains custom modules can also be published into the Cortana Intelligence Gallery for easy sharing.

Files in a custom R module

A custom R module is defined by a .zip file that contains, at a minimum, two files:

- A **source file** that implements the R function exposed by the module
- An **XML definition file** that describes the custom module interface

Additional auxiliary files can also be included in the .zip file that provides functionality that can be accessed from the custom module. This option is discussed in the **Arguments** part of the reference section **Elements in the XML definition file** following the quickstart example.

Quickstart example: define, package, and register a custom R module

This example illustrates how to construct the files required by a custom R module, package them into a zip file, and then register the module in your Machine Learning workspace. The example zip package and sample files can be downloaded from [Download CustomAddRows.zip file](#).

The source file

Consider the example of a **Custom Add Rows** module that modifies the standard implementation of the **Add Rows** module used to concatenate rows (observations) from two datasets (data frames). The standard **Add Rows**

module appends the rows of the second input dataset to the end of the first input dataset using the `rbind` algorithm. The customized `CustomAddRows` function similarly accepts two datasets, but also accepts a Boolean swap parameter as an additional input. If the swap parameter is set to **FALSE**, it returns the same data set as the standard implementation. But if the swap parameter is **TRUE**, the function appends rows of first input dataset to the end of the second dataset instead. The `CustomAddRows.R` file that contains the implementation of the R `CustomAddRows` function exposed by the **Custom Add Rows** module has the following R code.

```
CustomAddRows <- function(dataset1, dataset2, swap=FALSE)
{
  if(swap)
  {
    return (rbind(dataset2, dataset1));
  }
  else
  {
    return (rbind(dataset1, dataset2));
  }
}
```

The XML definition file

To expose this `CustomAddRows` function as an Azure Machine Learning module, an XML definition file must be created to specify how the **Custom Add Rows** module should look and behave.

```
<!-- Defined a module using an R Script -->
<Module name="CustomAddRows">
  <Owner>Microsoft Corporation</Owner>
  <Description>Appends one dataset to another. Dataset 2 is concatenated to Dataset 1 when Swap is FALSE, and vice versa when Swap is TRUE.</Description>

  <!-- Specify the base language, script file and R function to use for this module. -->
  <Language name="R"
    sourceFile="CustomAddRows.R"
    entryPoint="CustomAddRows" />

  <!-- Define module input and output ports -->
  <!-- Note: The values of the id attributes in the Input and Arg elements must match the parameter names in the R Function CustomAddRows defined in CustomAddRows.R. -->
  <Ports>
    <Input id="dataset1" name="Dataset 1" type="DataTable">
      <Description>First input dataset</Description>
    </Input>
    <Input id="dataset2" name="Dataset 2" type="DataTable">
      <Description>Second input dataset</Description>
    </Input>
    <Output id="dataset" name="Dataset" type="DataTable">
      <Description>The combined dataset</Description>
    </Output>
  </Ports>

  <!-- Define module parameters -->
  <Arguments>
    <Arg id="swap" name="Swap" type="bool" >
      <Description>Swap input datasets.</Description>
    </Arg>
  </Arguments>
</Module>
```

It is critical to note that the value of the **id** attributes of the **Input** and **Arg** elements in the XML file must match the function parameter names of the R code in the `CustomAddRows.R` file EXACTLY: (`dataset1`, `dataset2`, and `swap` in the example). Similarly, the value of the **entryPoint** attribute of the **Language** element must match the name of the function in the R script EXACTLY: (`CustomAddRows` in the example).

In contrast, the **id** attribute for the **Output** element does not correspond to any variables in the R script. When more than one output is required, simply return a list from the R function with results placed *in the same order as Outputs* elements are declared in the XML file.

Package and register the module

Save these two files as *CustomAddRows.R* and *CustomAddRows.xml* and then zip the two files together into a *CustomAddRows.zip* file.

To register them in your Machine Learning workspace, go to your workspace in the Machine Learning Studio, click the **+NEW** button on the bottom and choose **MODULE -> FROM ZIP PACKAGE** to upload the new **Custom Add Rows** module.



The **Custom Add Rows** module is now ready to be accessed by your Machine Learning experiments.

Elements in the XML definition file

Module elements

The **Module** element is used to define a custom module in the XML file. Multiple modules can be defined in one XML file using multiple **module** elements. Each module in your workspace must have a unique name. Register a custom module with the same name as an existing custom module and it replaces the existing module with the new one. Custom modules can, however, be registered with the same name as an existing Azure Machine Learning module. If so, they appear in the **Custom** category of the module palette.

```
<Module name="Custom Add Rows" isDeterministic="false">
  <Owner>Microsoft Corporation</Owner>
  <Description>Appends one dataset to another...</Description>/>
```

Within the **Module** element, you can specify two additional optional elements:

- an **Owner** element that is embedded into the module
- a **Description** element that contains text that is displayed in quick help for the module and when you hover over the module in the Machine Learning UI.

Rules for characters limits in the Module elements:

- The value of the **name** attribute in the **Module** element must not exceed 64 characters in length.
- The content of the **Description** element must not exceed 128 characters in length.
- The content of the **Owner** element must not exceed 32 characters in length.

A module's results can be deterministic or nondeterministic.* By default, all modules are considered to be deterministic. That is, given an unchanging set of input parameters and data, the module should return the same results each time it is run. Given this behavior, Azure Machine Learning Studio only reruns modules marked as deterministic if a parameter or the input data has changed. Returning the cached results also provides much faster execution of experiments.

There are functions that are nondeterministic, such as RAND or a function that returns the current date or time. If your module uses a nondeterministic function, you can specify that the module is non-deterministic by setting the optional **isDeterministic** attribute to **FALSE**. This insures that the module is rerun whenever the experiment is run,

even if the module input and parameters have not changed.

Language Definition

The **Language** element in your XML definition file is used to specify the custom module language. Currently, R is the only supported language. The value of the **sourceFile** attribute must be the name of the R file that contains the function to call when the module is run. This file must be part of the zip package. The value of the **entryPoint** attribute is the name of the function being called and must match a valid function defined with in the source file.

```
<Language name="R" sourceFile="CustomAddRows.R" entryPoint="CustomAddRows" />
```

Ports

The input and output ports for a custom module are specified in child elements of the **Ports** section of the XML definition file. The order of these elements determines the layout experienced (UX) by users. The first child **input** or **output** listed in the **Ports** element of the XML file becomes the left-most input port in the Machine Learning UX. Each input and output port may have an optional **Description** child element that specifies the text shown when you hover the mouse cursor over the port in the Machine Learning UI.

Ports Rules:

- Maximum number of **input and output ports** is 8 for each.

Input elements

Input ports allow you to pass data to your R function and workspace. The **data types** that are supported for input ports are as follows:

DataTable: This type is passed to your R function as a data.frame. In fact, any types (for example, CSV files or ARFF files) that are supported by Machine Learning and that are compatible with **DataTable** are converted to a data.frame automatically.

```
<Input id="dataset1" name="Input 1" type="DataTable" isOptional="false">
  <Description>Input Dataset 1</Description>
</Input>
```

The **id** attribute associated with each **DataTable** input port must have a unique value and this value must match its corresponding named parameter in your R function. Optional **DataTable** ports that are not passed as input in an experiment have the value **NULL** passed to the R function and optional zip ports are ignored if the input is not connected. The **isOptional** attribute is optional for both the **DataTable** and **Zip** types and is *false* by default.

Zip: Custom modules can accept a zip file as input. This input is unpacked into the R working directory of your function

```
<Input id="zippedData" name="Zip Input" type="Zip" IsOptional="false">
  <Description>Zip files to be extracted to the R working directory.</Description>
</Input>
```

For custom R modules, the id for a Zip port does not have to match any parameters of the R function. This is because the zip file is automatically extracted to the R working directory.

Input Rules:

- The value of the **id** attribute of the **Input** element must be a valid R variable name.
- The value of the **id** attribute of the **Input** element must not be longer than 64 characters.
- The value of the **name** attribute of the **Input** element must not be longer than 64 characters.
- The content of the **Description** element must not be longer than 128 characters
- The value of the **type** attribute of the **Input** element must be *Zip* or *DataTable*.

- The value of the **isOptional** attribute of the **Input** element is not required (and is *false* by default when not specified); but if it is specified, it must be *true* or *false*.

Output elements

Standard output ports: Output ports are mapped to the return values from your R function, which can then be used by subsequent modules. *DataTable* is the only standard output port type supported currently. (Support for *Learners* and *Transforms* is forthcoming.) A *DataTable* output is defined as:

```
<Output id="dataset" name="Dataset" type="DataTable">
  <Description>Combined dataset</Description>
</Output>
```

For outputs in custom R modules, the value of the **id** attribute does not have to correspond with anything in the R script, but it must be unique. For a single module output, the return value from the R function must be a *data.frame*. In order to output more than one object of a supported data type, the appropriate output ports need to be specified in the XML definition file and the objects need to be returned as a list. The output objects are assigned to output ports from left to right, reflecting the order in which the objects are placed in the returned list.

For example, if you want to modify the **Custom Add Rows** module to output the original two datasets, *dataset1* and *dataset2*, in addition to the new joined dataset, *dataset*, (in an order, from left to right, as: *dataset*, *dataset1*, *dataset2*), then define the output ports in the CustomAddRows.xml file as follows:

```
<Ports>
  <Output id="dataset" name="Dataset Out" type="DataTable">
    <Description>New Dataset</Description>
  </Output>
  <Output id="dataset1_out" name="Dataset 1 Out" type="DataTable">
    <Description>First Dataset</Description>
  </Output>
  <Output id="dataset2_out" name="Dataset 2 Out" type="DataTable">
    <Description>Second Dataset</Description>
  </Output>
  <Input id="dataset1" name="Dataset 1" type="DataTable">
    <Description>First Input Table</Description>
  </Input>
  <Input id="dataset2" name="Dataset 2" type="DataTable">
    <Description>Second Input Table</Description>
  </Input>
</Ports>
```

And return the list of objects in a list in the correct order in 'CustomAddRows.R':

```
CustomAddRows <- function(dataset1, dataset2, swap=FALSE) {
  if(swap) { dataset <- rbind(dataset2, dataset1) }
  else { dataset <- rbind(dataset1, dataset2) }
  return (list(dataset, dataset1, dataset2))
}
```

Visualization output: You can also specify an output port of type *Visualization*, which displays the output from the R graphics device and console output. This port is not part of the R function output and does not interfere with the order of the other output port types. To add a visualization port to the custom modules, add an **Output** element with a value of *Visualization* for its **type** attribute:

```
<Output id="deviceOutput" name="View Port" type="Visualization">
  <Description>View the R console graphics device output.</Description>
</Output>
```

Output Rules:

- The value of the **id** attribute of the **Output** element must be a valid R variable name.
- The value of the **id** attribute of the **Output** element must not be longer than 32 characters.
- The value of the **name** attribute of the **Output** element must not be longer than 64 characters.
- The value of the **type** attribute of the **Output** element must be *Visualization*.

Arguments

Additional data can be passed to the R function via module parameters which are defined in the **Arguments** element. These parameters appear in the rightmost properties pane of the Machine Learning UI when the module is selected. Arguments can be any of the supported types or you can create a custom enumeration when needed.

Similar to the **Ports** elements, **Arguments** elements can have an optional **Description** element that specifies the text that appears when you hover the mouse over the parameter name. Optional properties for a module, such as **defaultValue**, **minValue**, and **maxValue** can be added to any argument as attributes to a **Properties** element. Valid properties for the **Properties** element depend on the argument type and are described with the supported argument types in the next section. Arguments with the **isOptional** property set to "true" do not require the user to enter a value. If a value is not provided to the argument, then the argument is not passed to the entry point function. Arguments of the entry point function that are optional need to be explicitly handled by the function, e.g. assigned a default value of NULL in the entry point function definition. An optional argument will only enforce the other argument constraints, i.e. min or max, if a value is provided by the user. As with inputs and outputs, it is critical that each of the parameters have unique id values associated with them. In our quick start example the associated id/parameter was *swap*.

Arg element

A module parameter is defined using the **Arg** child element of the **Arguments** section of the XML definition file. As with the child elements in the **Ports** section, the ordering of parameters in the **Arguments** section defines the layout encountered in the UX. The parameters appear from top down in the UI in the same order in which they are defined in the XML file. The types supported by Machine Learning for parameters are listed here.

int – an Integer (32-bit) type parameter.

```
<Arg id="intValue1" name="Int Param" type="int">
  <Properties min="0" max="100" default="0" />
  <Description>Integer Parameter</Description>
</Arg>
```

- *Optional Properties:* **min**, **max**, **default** and **isOptional**

double – a double type parameter.

```
<Arg id="doubleValue1" name="Double Param" type="double">
  <Properties min="0.000" max="0.999" default="0.3" />
  <Description>Double Parameter</Description>
</Arg>
```

- *Optional Properties:* **min**, **max**, **default** and **isOptional**

bool – a Boolean parameter that is represented by a check-box in UX.

```
<Arg id="boolValue1" name="Boolean Param" type="bool">
  <Properties default="true" />
  <Description>Boolean Parameter</Description>
</Arg>
```

- *Optional Properties:* **default** - false if not set

string: a standard string

```
<Arg id="stringValue1" name="My string Param" type="string">
  <Properties isOptional="true" />
  <Description>String Parameter 1</Description>
</Arg>
```

- *Optional Properties:* **default** and **isOptional**

ColumnPicker: a column selection parameter. This type renders in the UX as a column chooser. The **Property** element is used here to specify the id of the port from which columns are selected, where the target port type must be *DataTable*. The result of the column selection is passed to the R function as a list of strings containing the selected column names.

```
<Arg id="colset" name="Column set" type="ColumnPicker">
  <Properties portId="datasetIn1" allowedTypes="Numeric" default="NumericAll"/>
  <Description>Column set</Description>
</Arg>
```

- *Required Properties:* **portId** - matches the id of an Input element with type *DataTable*.
- *Optional Properties:*
 - **allowedTypes** - Filters the column types from which you can pick. Valid values include:
 - Numeric
 - Boolean
 - Categorical
 - String
 - Label
 - Feature
 - Score
 - All
 - **default** - Valid default selections for the column picker include:
 - None
 - NumericFeature
 - NumericLabel
 - NumericScore
 - NumericAll
 - BooleanFeature
 - BooleanLabel
 - BooleanScore
 - BooleanAll
 - CategoricalFeature
 - CategoricalLabel
 - CategoricalScore
 - CategoricalAll
 - StringFeature
 - StringLabel
 - StringScore
 - StringAll
 - AllLabel
 - AllFeature

- AllScore
- All

DropDown: a user-specified enumerated (dropdown) list. The dropdown items are specified within the **Properties** element using an **Item** element. The **id** for each **Item** must be unique and a valid R variable. The value of the **name** of an **Item** serves as both the text that you see and the value that is passed to the R function.

```
<Arg id="color" name="Color" type="DropDown">
<Properties default="red">
<Item id="red" name="Red Value"/>
<Item id="green" name="Green Value"/>
<Item id="blue" name="Blue Value"/>
</Properties>
<Description>Select a color.</Description>
</Arg>
```

- *Optional Properties:*
 - **default** - The value for the default property must correspond with an id value from one of the **Item** elements.

Auxiliary Files

Any file that is placed in your custom module ZIP file is going to be available for use during execution time. Any directory structures present are preserved. This means that file sourcing works the same locally and in Azure Machine Learning execution.

NOTE

Notice that all files are extracted to 'src' directory so all paths should have 'src/' prefix.

For example, say you want to remove any rows with NAs from the dataset, and also remove any duplicate rows, before outputting it into CustomAddRows, and you've already written an R function that does that in a file RemoveDupNARows.R:

```
RemoveDupNARows <- function(dataFrame) {
  #Remove Duplicate Rows:
  dataFrame <- unique(dataFrame)
  #Remove Rows with NAs:
  finalDataFrame <- dataFrame[complete.cases(dataFrame),]
  return(finalDataFrame)
}
```

You can source the auxiliary file RemoveDupNARows.R in the CustomAddRows function:

```
CustomAddRows <- function(dataset1, dataset2, swap=FALSE) {
  source("src/RemoveDupNARows.R")
  if(swap) {
    dataset <- rbind(dataset2, dataset1)
  } else {
    dataset <- rbind(dataset1, dataset2)
  }
  dataset <- removeDupNARows(dataset)
  return (dataset)
}
```

Next, upload a zip file containing 'CustomAddRows.R', 'CustomAddRows.xml', and 'RemoveDupNARows.R' as a custom R module.

Execution Environment

The execution environment for the R script uses the same version of R as the **Execute R Script** module and can use the same default packages. You can also add additional R packages to your custom module by including them in the custom module zip package. Just load them in your R script as you would in your own R environment.

Limitations of the execution environment include:

- Non-persistent file system: Files written when the custom module is run are not persisted across multiple runs of the same module.
- No network access

Execute Python machine learning scripts in Azure Machine Learning Studio

1/17/2017 • 9 min to read • [Edit on GitHub](#)

This topic describes the design principles underlying the current support for Python scripts in Azure Machine Learning. The main capabilities are also outlined, including support for importing existing code, exporting visualizations and, finally, some of the limitations and ongoing work are discussed.

Python is an indispensable tool in the tool chest of many data scientists. It has:

- an elegant and concise syntax,
- cross-platform support,
- a vast collection of powerful libraries, and
- mature development tools.

Python is being used in all phases of the workflow typically used in machine learning modeling, from data ingest and processing, to feature construction and model training, and then validation and deployment of the models.

Azure Machine Learning Studio supports embedding Python scripts into various parts of a machine learning experiment and also seamlessly publishing them as scalable, operationalized web services on Microsoft Azure.

NOTE

[Try Azure Machine Learning for free](#)

No credit card or Azure subscription needed. [Get started now >](#)

Design principles of Python scripts in Machine Learning

The primary interface to Python in Azure Machine Learning Studio is via the [Execute Python Script](#) module shown in Figure 1.

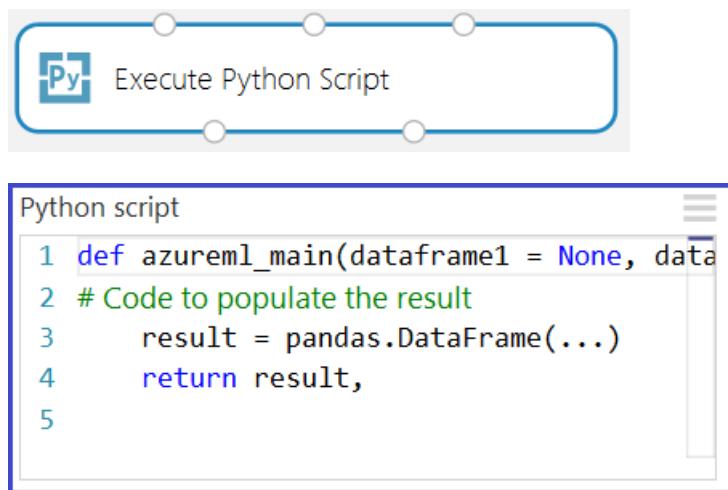


Figure 1. The **Execute Python Script** module.

The [Execute Python Script](#) module accepts up to three inputs and produces up to two outputs (discussed below), just like its R analog, the [Execute R Script](#) module. The Python code to be executed is entered into the parameter box as a specially named entry-point function called `azureml_main`. Here are the key design principles used to

implement this module:

1. *Must be idiomatic for Python users.* Most Python users factor their code as functions inside modules, so putting a lot of executable statements in a top-level module is relatively rare. As a result, the script box also takes a specially named Python function as opposed to just a sequence of statements. The objects exposed in the function are standard Python library types such as [Pandas](#) data frames and [NumPy](#) arrays.
2. *Must have high-fidelity between local and cloud executions.* The backend used to execute the Python code is based on [Anaconda](#) 2.1, a widely used cross-platform scientific Python distribution. It comes with close to 200 of the most common Python packages. Therefore, data scientists can debug and qualify their code on their local Azure Machine Learning-compatible Anaconda environment. Then use existing development environments such as [IPython](#) notebook or [Python Tools for Visual Studio](#) to run it as part of an Azure Machine Learning experiment with high confidence. Further, the `azureml_main` entry point is a vanilla Python function and can be authored without Azure Machine Learning specific code or the SDK installed.
3. *Must be seamlessly composable with other Azure Machine Learning modules.* The [Execute Python Script](#) module accepts, as inputs and outputs, standard Azure Machine Learning datasets. The underlying framework transparently and efficiently bridges the Azure Machine Learning and Python runtimes (supporting features such as missing values). Python can therefore be used in conjunction with existing Azure Machine Learning workflows, including those that call into R and SQLite. One can therefore envisage workflows that:
 - use Python and Pandas for data pre-processing and cleaning,
 - feed the data to a SQL transformation, joining multiple datasets to form features,
 - train models using the extensive collection of algorithms in Azure Machine Learning, and
 - evaluate and post-process the results using R.

Basic usage scenarios in Machine Learning for Python scripts

In this section, we survey some of the basic uses of the [Execute Python Script](#) module. As mentioned earlier, any inputs to the Python module are exposed as Pandas data frames. More information on Python Pandas and how it can be used to manipulate data effectively and efficiently can be found in *Python for Data Analysis* (O'Reilly, 2012) by W. McKinney. The function must return a single Pandas data frame packaged inside of a Python [sequence](#) such as a tuple, list, or NumPy array. The first element of this sequence is then returned in the first output port of the module. This scheme is shown in Figure 2.

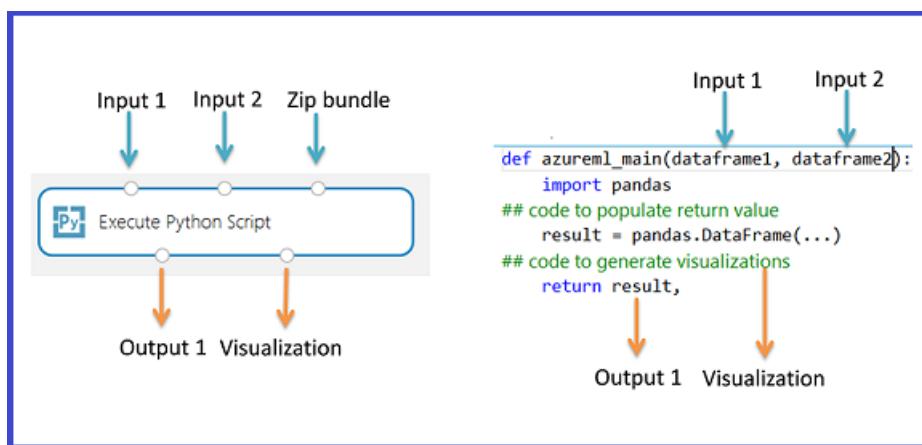


Figure 2. Mapping of input ports to parameters and return value to output port.

More detailed semantics of how the input ports get mapped to parameters of the `azureml_main` function are shown in Table 1:

Input port configuration	Python signature	Remarks
Execute Python Script	<code>def azureml_main(): pass</code>	Can also be any function with <i>all optional parameters</i>
Execute Python Script	<code>def azureml_main(dataframe1): pass</code>	Can also be any function with the <i>first parameter</i> being a data frame and all other parameters being <i>optional</i> .
Execute Python Script	<code>def azureml_main(dataframe1): pass</code>	Can also be any function with the <i>first parameter</i> being a data frame and all other parameters being <i>optional</i> .
Execute Python Script	<code>def azureml_main(dataframe1, dataframe2): pass</code>	Can also be any function with the <i>first two parameters</i> being data frames and all other parameters being <i>optional</i> .

Table 1. Mapping of input ports to function parameters.

The mapping between input ports and function parameters is positional. The first connected input port is mapped to the first parameter of the function and the second input (if connected) is mapped to the second parameter of the function.

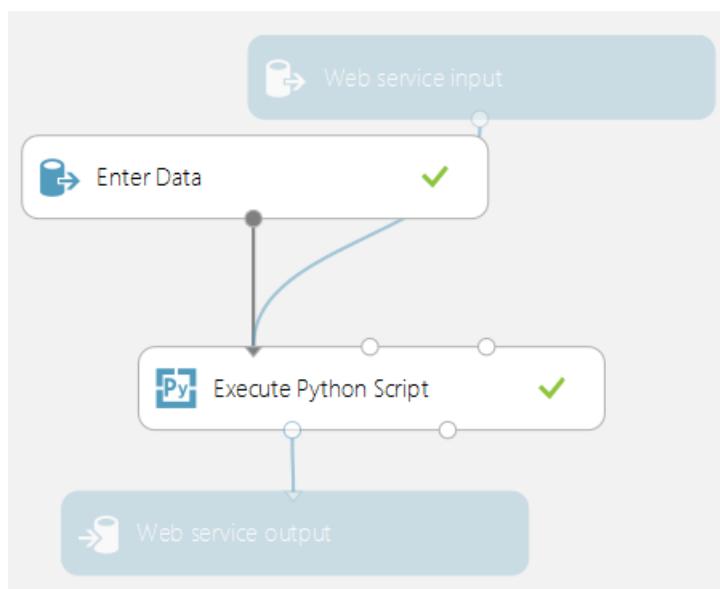
Translation of input and output types

As explained earlier, input datasets in Azure Machine Learning are converted to data frames in Pandas and output data frames are converted back to Azure Machine Learning datasets. The following conversions are performed:

1. String and numeric columns are converted as-is and missing values in a dataset are converted to 'NA' values in Pandas. The same conversion happens on the way back (NA values in Pandas are converted to missing values in Azure Machine Learning).
2. Index vectors in Pandas are not supported in Azure Machine Learning. All input data frames in the Python function always have a 64-bit numerical index from 0 through the number of rows minus 1.
3. Azure Machine Learning datasets cannot have duplicate column names and column names that are not strings. If an output data frame contains non-numeric columns, the framework calls `str` on the column names. Likewise, any duplicate column names are automatically mangled to insure the names are unique. The suffix (2) is added to the first duplicate, (3) to the second duplicate, etc.

Operationalizing Python scripts

Any [Execute Python Script](#) modules used in a scoring experiment are called when published as a web service. For example, Figure 3 shows a scoring experiment containing the code to evaluate a single Python expression.



```

Python script
1 def azureml_main(expr_as_frame):
2     import pandas as pd
3     expr = expr_as_frame.iat[0,0]
4     result = pd.DataFrame({'Expr': [expr], \
5                            'Result': [eval(expr)]})
6     return result,

```

Figure 3. Web service for evaluating a Python expression.

A web service created from this experiment takes as input a Python expression (as a string), sends it to the Python interpreter and returns a table containing both the expression and the evaluated result.

Importing existing Python script modules

A common use-case for many data scientists is to incorporate existing Python scripts into Azure Machine Learning experiments. Instead of concatenating and pasting all the code into a single script box, the [Execute Python Script](#) module accepts a third input port to which a zip file that contains the Python modules can be connected. The file is then unzipped by the execution framework at runtime and the contents are added to the library path of the Python interpreter. The `azureml_main` entry point function can then import these modules directly.

As an example, consider the file Hello.py containing a simple "Hello, World" function.

```

def print_hello(name):
    print "Hello ", name

```

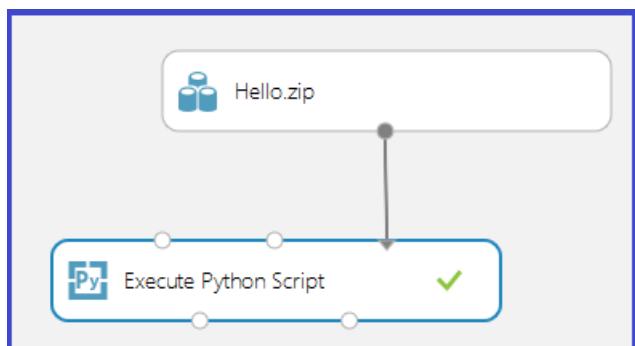
Figure 4. User-defined function.

Next, we create a file Hello.zip that contains Hello.py:

PythonCode						Hello.zip
Name	Type	Compressed size	Password ...	Size	Ratio	
 Hello.py	PY File	1 KB	No	1 KB	15%	

Figure 5. Zip file containing user-defined Python code.

Then, upload this as a dataset into Azure Machine Learning Studio. Create and run a simple experiment that uses the Python code in the Hello.zip file by attaching it to the third input port of the Execute Python Script, as shown in this figure.



Python script

```
1 # The script MUST include the following function,
2 # which is the entry point for this module:
3 # Param<dataframe1>: a pandas.DataFrame
4 # Param<dataframe2>: a pandas.DataFrame
5 def azureml_main():
6     import pandas as pd
7     import Hello
8     Hello.print_hello("World")
9     return pd.DataFrame(["Output"]),
10
11
```

Figure 6. Sample experiment with user-defined Python code uploaded as a zip file.

The module output shows that the zip file has been unpackaged and the function `print_hello` has indeed been run.

[ModuleOutput] Extracting Script Bundle.zip to .\Script Bundle	Modified	Size
[ModuleOutput] File Name		
[ModuleOutput] Hello.py	2015-02-09 14:26:26	48
[ModuleOutput] Hello World		

Figure 7. User-defined function in use inside the [Execute Python Script](#) module.

Working with visualizations

Plots created using Matplotlib that can be visualized on the browser can be returned by the [Execute Python Script](#). But the plots are not automatically redirected to images as they are when using R. So the user must explicitly save any plots to PNG files if they are to be returned back to Azure Machine Learning.

In order to generate images from Matplotlib, you must complete the following procedure:

- switch the backend to "AGG" from the default Qt-based renderer
- create a new figure object
- get the axis and generate all plots into it
- save the figure to a PNG file

This process is illustrated in the following Figure 8 that creates a scatter plot matrix using the `scatter_matrix` function in Pandas.

```
def azureml_main(dataframe1):
    import matplotlib
    matplotlib.use("agg") ← Change backend

    from pandas.tools.plotting import scatter_matrix
    import matplotlib.pyplot as plt

    fig = plt.figure(); ← Create new figure
    ax = fig.gca()
    scatter_matrix(dataframe1, ax=ax) ← Plot into specified axis

    fig.savefig("scatter.png") ← Save figure to image
    return dataframe1,
```

Figure 8. Saving Matplotlib figures to images.

Figure 9 shows an experiment that uses the script shown previously to return plots via the second output port.

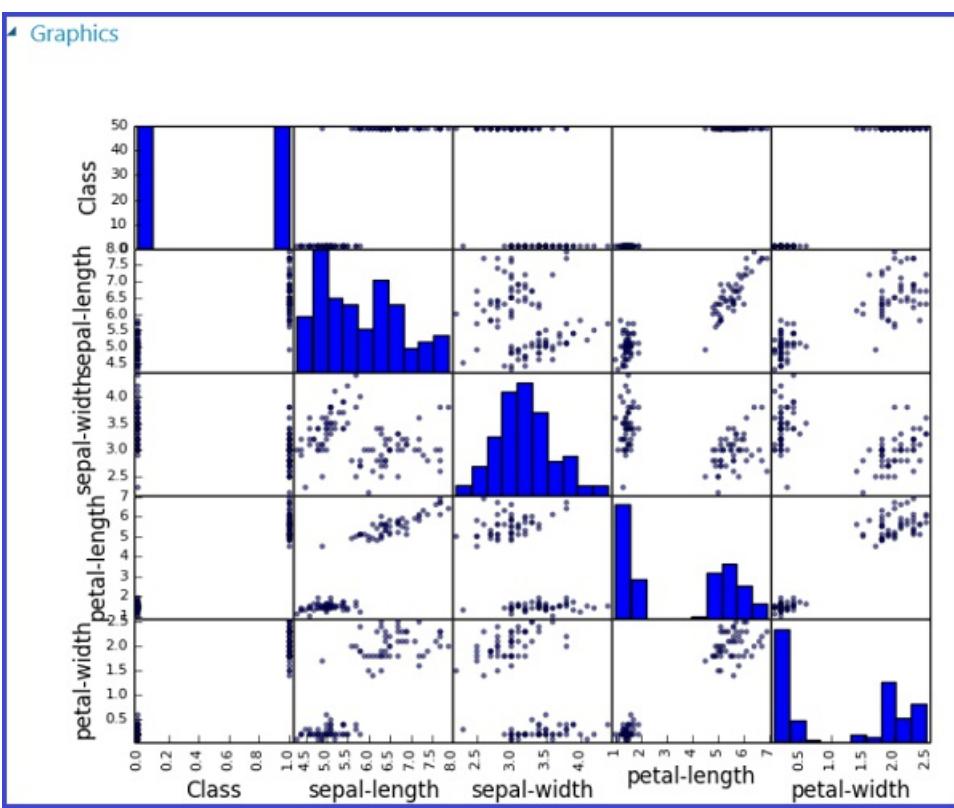
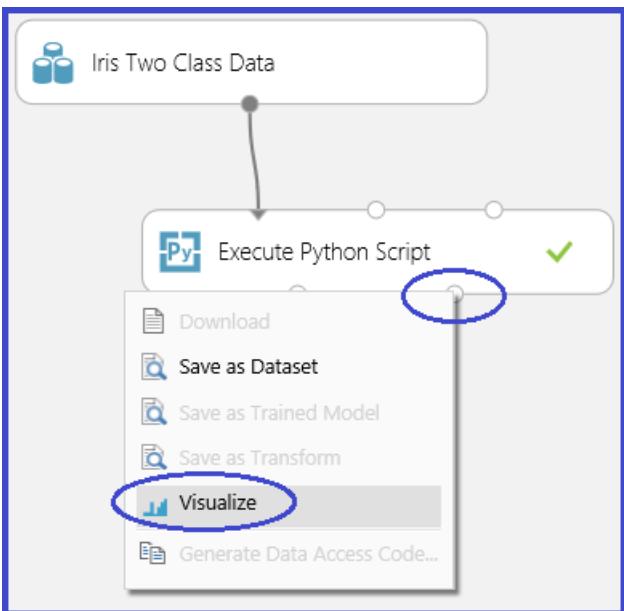


Figure 9. Visualizing plots generated from Python code.

It is possible to return multiple figures by saving them into different images, the Azure Machine Learning runtime picks up all images and concatenates them for visualization.

Advanced examples

The Anaconda environment installed in Azure Machine Learning contains common packages such as NumPy, SciPy, and Scikits-Learn and these can be effectively used for various data processing tasks in a typical machine learning pipeline. As an example, the following experiment and script illustrates the use of ensemble learners in Scikits-Learn to compute feature importance scores for a dataset. The scores can then be used to perform supervised feature selection before feeding into another machine learning model.

The Python function to compute the importance scores and order the features based on it is shown below:

```

from sklearn.ensemble import GradientBoostingClassifier
import numpy as np, pandas as pd

def azureml_main(dataframe1):
    colnames = dataframe1.columns
    y = np.array(dataframe1[colnames[-1]])
    X = np.array(dataframe1.ix[:, :len(colnames)-1])
    clf = GradientBoostingClassifier(n_estimators=100, \
        learning_rate=1.0, max_depth=1, random_state=0).\
        fit(X, y)
    fint = clf.feature_importances_
    fnames = np.array(colnames[:-1])

    perm = fint.argsort()

    ret = pd.DataFrame()
    ret["Feature"] = fnames[perm[::-1]]
    ret["Score"] = fint[perm[::-1]]
    return ret,

```

Figure 10. Function to rank features by scores. The following experiment then computes and returns the importance scores of features in the "Pima Indian Diabetes" dataset in Azure Machine Learning:

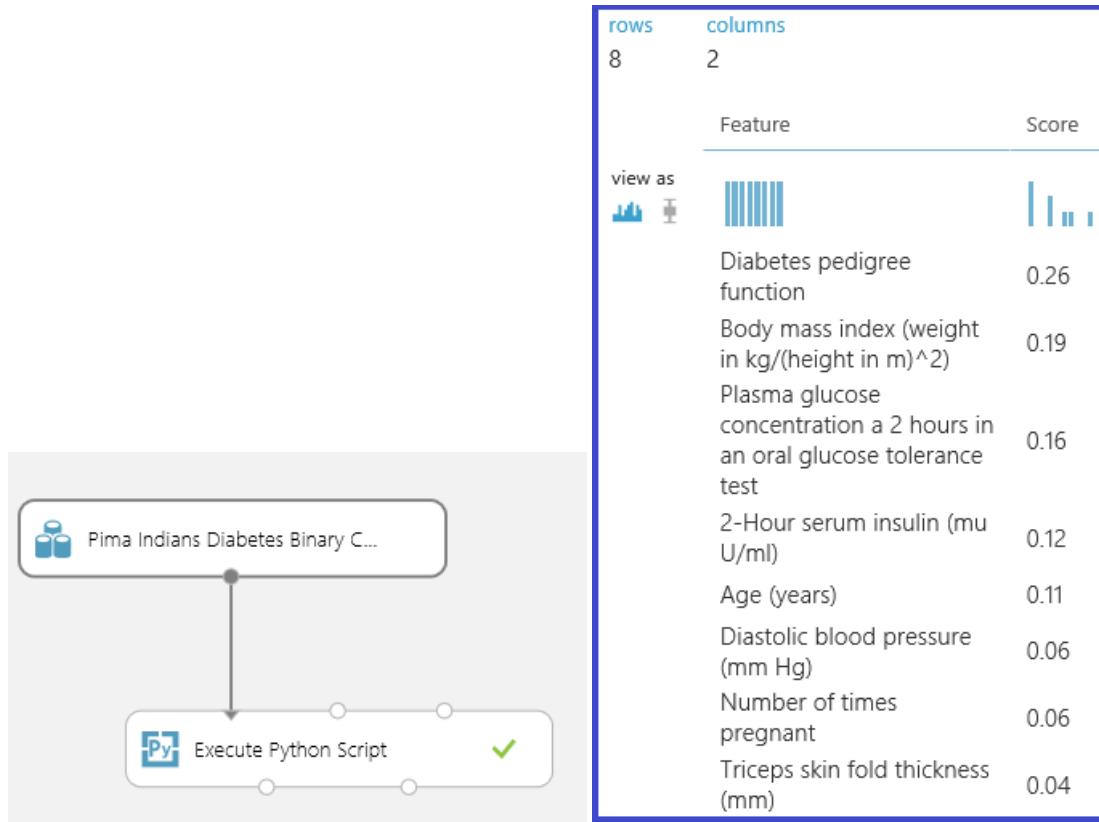


Figure 11. Experiment to rank features in the Pima Indian Diabetes dataset.

Limitations

The [Execute Python Script](#) currently has the following limitations:

1. *Sandboxed execution.* The Python runtime is currently sandboxed and, as a result, does not allow access to the network or to the local file system in a persistent manner. All files saved locally are isolated and deleted once the module finishes. The Python code cannot access most directories on the machine it runs on, the exception being the current directory and its subdirectories.
2. *Lack of sophisticated development and debugging support.* The Python module currently does not support IDE features such as intellisense and debugging. Also, if the module fails at runtime, the full Python stack trace is available, but must be viewed in the output log for the module. We currently recommend that you develop and debug their Python scripts in an environment such as IPython and then import the code into the module.

3. *Single data frame output.* The Python entry point is only permitted to return a single data frame as output. It is not currently possible to return arbitrary Python objects such as trained models directly back to the Azure Machine Learning runtime. Like [Execute R Script](#), which has the same limitation, it is however possible in many cases to pickle objects into a byte array and then return that inside of a data frame.
4. *Inability to customize Python installation.* Currently, the only way to add custom Python modules is via the zip file mechanism described earlier. While this is feasible for small modules, it is cumbersome for large modules (especially those with native DLLs) or a large number of modules.

Conclusions

The [Execute Python Script](#) module allows a data scientist to incorporate existing Python code into cloud-hosted machine learning workflows in Azure Machine Learning and to seamlessly operationalize them as part of a web service. The Python script module interoperates naturally with other modules in Azure Machine Learning and can be used for a range of tasks from data exploration to pre-processing, to feature extraction, to evaluation and post-processing of the results. The backend runtime used for execution is based on Anaconda, a well-tested and widely used Python distribution. This makes it simple for you to on-board existing code assets into the cloud.

We expect to provide additional functionality to the [Execute Python Script](#) module such as the ability to train and operationalize models in Python and to add better support for the development and debugging code in Azure Machine Learning Studio.

Next steps

For more information, see the [Python Developer Center](#).

Azure Machine Learning Web Services: Deployment and consumption

1/17/2017 • 3 min to read • [Edit on GitHub](#)

You can use Azure Machine Learning to deploy machine-learning workflows and models as web services. These web services can then be used to call the machine-learning models from applications over the Internet to do predictions in real time or in batch mode. Because the web services are RESTful, you can call them from various programming languages and platforms, such as .NET and Java, and from applications, such as Excel.

The next sections provide links to walkthroughs, code, and documentation to help get you started.

Deploy a web service

With Azure Machine Learning Studio

Machine Learning Studio and the Microsoft Azure Machine Learning Web Services portal help you deploy and manage a web service without writing code.

The following links provide general information about how to deploy a new web service:

- For an overview about how to deploy a new web service that's based on Azure Resource Manager, see [Deploy a new web service](#).
- For a walkthrough about how to deploy a web service, see [Deploy an Azure Machine Learning web service](#).
- For a full walkthrough about how to create and deploy a web service, see [Walkthrough Step 1: Create a Machine Learning workspace](#).
- For specific examples that deploy a web service, see:
 - [Walkthrough Step 5: Deploy the Azure Machine Learning web service](#)
 - [How to deploy a web service to multiple regions](#)

With web services resource provider APIs (Azure Resource Manager APIs)

The Azure Machine Learning resource provider for web services enables deployment and management of web services by using REST API calls. For additional details, see the [Machine Learning Web Service \(REST\)](#) reference.

With PowerShell cmdlets

Azure Machine Learning resource provider for web services enables deployment and management of web services by using PowerShell cmdlets.

To use the cmdlets, you must first sign in to your Azure account from within the PowerShell environment by using the [Add-AzureRmAccount](#) cmdlet. If you are unfamiliar with how to call PowerShell commands that are based on Resource Manager, see [Using Azure PowerShell with Azure Resource Manager](#).

To export your predictive experiment, use [this sample code](#). After you create the .exe file from the code, you can type:

```
C:\<folder>\GetWSD <experiment-url> <workspace-auth-token>
```

Running the application creates a web service JSON template. To use the template to deploy a web service, you must add the following information:

- Storage account name and key

You can get the storage account name and key from either the [Azure portal](#) or the [Azure classic portal](#).

- Commitment plan ID

You can get the plan ID from the [Azure Machine Learning Web Services](#) portal by signing in and clicking a plan name.

Add them to the JSON template as children of the *Properties* node at the same level as the *MachineLearningWorkspace* node.

Here's an example:

```
"StorageAccount": {  
    "name": "YourStorageAccountName",  
    "key": "YourStorageAccountKey"  
},  
"CommitmentPlan": {  
    "id":  
    "subscriptions/YouSubscriptionID/resourceGroups/YourResourceGroupID/providers/Microsoft.MachineLearning/commitmentPlans/YourPlanName  
"  
}
```

See the following articles and sample code for additional details:

- [Azure Machine Learning Cmdlets](#) reference on MSDN
- Sample [walkthrough](#) on GitHub

Consume the web services

From the Azure Machine Learning Web Services UI (Testing)

You can test your web service from the Azure Machine Learning Web Services portal. This includes testing the Request-Response service (RRS) and Batch Execution service (BES) interfaces.

- [Deploy a new web service](#)
- [Deploy an Azure Machine Learning web service](#)
- [Walkthrough Step 5: Deploy the Azure Machine Learning web service](#)

From Excel

You can download an Excel template that consumes the web service:

- [Consuming an Azure Machine Learning web service from Excel](#)
- [Excel add-in for Azure Machine Learning Web Services](#)

From a REST-based client

Azure Machine Learning Web Services are RESTful APIs. You can consume these APIs from various platforms, such as .NET, Python, R, Java, etc. The **Consume** page for your web service on the [Microsoft Azure Machine Learning Web Services portal](#) has sample code that can help you get started. For more information, see [How to consume an Azure Machine Learning web service that has been deployed from a Machine Learning experiment](#).

How a Machine Learning model progresses from an experiment to an operationalized Web service

1/17/2017 • 7 min to read • [Edit on GitHub](#)

Azure Machine Learning Studio provides an interactive canvas that allows you to develop, run, test, and iterate an **experiment** representing a predictive analysis model. There are a wide variety of modules available that can:

- Input data into your experiment
- Manipulate the data
- Train a model using machine learning algorithms
- Score the model
- Evaluate the results
- Output final values

Once you're satisfied with your experiment, you can deploy it as a **Classic Azure Machine Learning Web service** or a **New Azure Machine Learning Web service** so that users can send it new data and receive back results.

In this article, we give an overview of the mechanics of how your Machine Learning model progresses from a development experiment to an operationalized Web service.

NOTE

There are other ways to develop and deploy machine learning models, but this article is focused on how you use Machine Learning Studio. For example, to read a description of how to create a classic predictive Web service with R, see the blog post [Build & Deploy Predictive Web Apps Using RStudio and Azure ML](#).

While Azure Machine Learning Studio is designed to help you develop and deploy a *predictive analysis model*, it's possible to use Studio to develop an experiment that doesn't include a predictive analysis model. For example, an experiment might just input data, manipulate it, and then output the results. Just like a predictive analysis experiment, you can deploy this non-predictive experiment as a Web service, but it's a simpler process because the experiment isn't training or scoring a machine learning model. While it's not the typical to use Studio in this way, we'll include it in the discussion so that we can give a complete explanation of how Studio works.

Developing and deploying a predictive Web service

Here are the stages that a typical solution follows as you develop and deploy it using Machine Learning Studio:

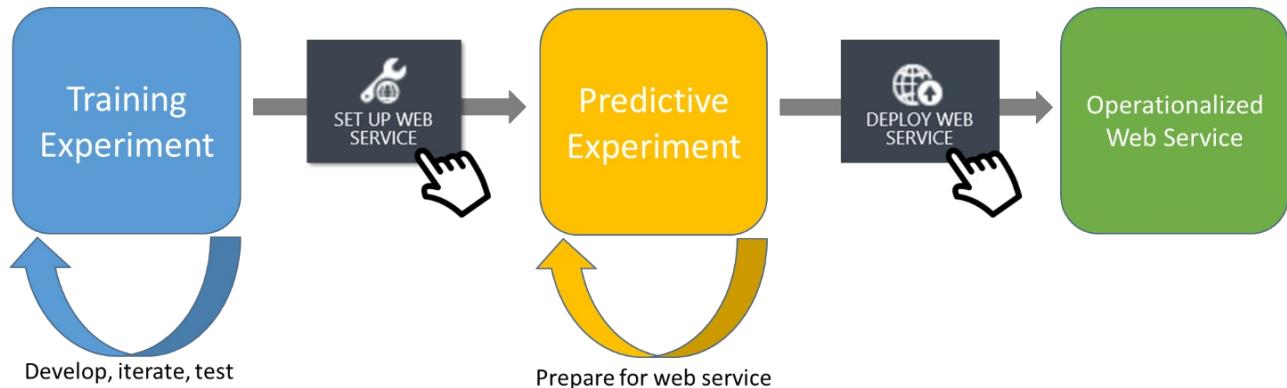


Figure 1 - Stages of a typical predictive analysis model

The training experiment

The **training experiment** is the initial phase of developing your Web service in Machine Learning Studio. The purpose of the training experiment is to give you a place to develop, test, iterate, and eventually train a machine learning model. You can even train multiple models simultaneously as you look for the best solution, but once you're done experimenting you'll select a single trained model and eliminate the rest from the experiment. For an example of developing a predictive analysis experiment, see [Develop a predictive analytics solution for credit risk assessment in Azure Machine Learning](#).

The predictive experiment

Once you have a trained model in your training experiment, click **Set Up Web Service** and select **Predictive Web Service** in Machine Learning Studio to initiate the process of converting your training experiment to a **predictive experiment**. The purpose of the predictive experiment is to use your trained model to score new data, with the goal of eventually becoming operationalized as an Azure Web service.

This conversion is done for you through the following steps:

- Convert the set of modules used for training into a single module and save it as a trained model
- Eliminate any extraneous modules not related to scoring
- Add input and output ports that the eventual Web service will use

There may be more changes you want to make to get your predictive experiment ready to deploy as a Web service. For example, if you want the Web service to output only a subset of results, you can add a filtering module before the output port.

In this conversion process, the training experiment is not discarded. When the process is complete, you have two tabs in Studio: one for the training experiment and one for the predictive experiment. This way you can make changes to the training experiment before you deploy your Web service and rebuild the predictive experiment. Or you can save a copy of the training experiment to start another line of experimentation.

NOTE

When you click **Predictive Web Service** you start an automatic process to convert your training experiment to a predictive experiment, and this works well in most cases. If your training experiment is complex (for example, you have multiple paths for training that you join together), you might prefer to do this conversion manually. For more information, see [Convert a Machine Learning training experiment to a predictive experiment](#).

The Web service

Once you're satisfied that your predictive experiment is ready, you can deploy your service as either a Classic Web service or a New Web service based on Azure Resource Manager. To operationalize your model by deploying it as a *Classic Machine Learning Web service*, click **Deploy Web Service** and select **Deploy Web Service [Classic]**. To deploy as *New Machine Learning Web service*, click **Deploy Web Service** and select **Deploy Web Service [New]**. Users can now send data to your model using the Web service REST API and receive back the results. For more information, see [How to consume an Azure Machine Learning Web service that has been deployed from a Machine Learning experiment](#).

The non-typical case: creating a non-predictive Web service

If your experiment does not train a predictive analysis model, then you don't need to create both a training experiment and a scoring experiment - there's just one experiment, and you can deploy it as a Web service. Machine Learning Studio detects whether your experiment contains a predictive model by analyzing the modules you've used.

After you've iterated on your experiment and are satisfied with it:

1. Click **Set Up Web Service** and select **Retraining Web Service** - input and output nodes are added

automatically

2. Click **Run**
3. Click **Deploy Web Service** and select **Deploy Web Service [Classic]** or **Deploy Web Service [New]** depending on the environment to which you want to deploy.

Your Web service is now deployed, and you can access and manage it just like a predictive Web service.

Updating your Web service

Now that you've deployed your experiment as a Web service, what if you need to update it?

That depends on what you need to update:

You want to change the input or output, or you want to modify how the Web service manipulates data

If you're not changing the model, but are just changing how the Web service handles data, you can edit the predictive experiment and then click **Deploy Web Service** and select **Deploy Web Service [Classic]** or **Deploy Web Service [New]** again. The Web service is stopped, the updated predictive experiment is deployed, and the Web service is restarted.

Here's an example: Suppose your predictive experiment returns the entire row of input data with the predicted result. You may decide that you want the Web service to just return the result. So you can add a **Project Columns** module in the predictive experiment, right before the output port, to exclude columns other than the result. When you click **Deploy Web Service** and select **Deploy Web Service [Classic]** or **Deploy Web Service [New]** again, the Web service is updated.

You want to retrain the model with new data

If you want to keep your machine learning model, but you would like to retrain it with new data, you have two choices:

1. **Retrain the model while the Web service is running** - If you want to retrain your model while the predictive Web service is running, you can do this by making a couple modifications to the training experiment to make it a **retraining experiment**, then you can deploy it as a **retraining web service**. For instructions on how to do this, see [Retrain Machine Learning models programmatically](#).
2. **Go back to the original training experiment and use different training data to develop your model**
 - Your predictive experiment is linked to the Web service, but the training experiment is not directly linked in this way. If you modify the original training experiment and click **Set Up Web Service**, it will create a *new* predictive experiment which, when deployed, will create a *new* Web service. It doesn't just update the original Web service.

If you need to modify the training experiment, open it and click **Save As** to make a copy. This will leave intact the original training experiment, predictive experiment, and Web service. You can now create a new Web service with your changes. Once you've deployed the new Web service you can then decide whether to stop the previous Web service or keep it running alongside the new one.

You want to train a different model

If you want to make changes to your original predictive experiment, such as selecting a different machine learning algorithm, trying a different training method, etc., then you need to follow the second procedure described above for retraining your model: open the training experiment, click **Save As** to make a copy, and then start down the new path of developing your model, creating the predictive experiment, and deploying the web service. This will create a new Web service unrelated to the original one - you can decide which one, or both, to keep running.

Next Steps

For more details on the process of developing and experiment, see the following articles:

- converting the experiment - [Convert a Machine Learning training experiment to a predictive experiment](#)
- deploying the Web service - [Deploy an Azure Machine Learning web service](#)
- retraining the model - [Retrain Machine Learning models programmatically](#)

For examples of the whole process, see:

- [Machine learning tutorial: Create your first experiment in Azure Machine Learning Studio](#)
- [Walkthrough: Develop a predictive analytics solution for credit risk assessment in Azure Machine Learning](#)

Deploy a new web service

1/17/2017 • 2 min to read • [Edit on GitHub](#)

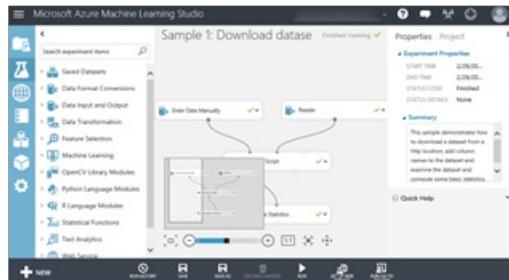
Microsoft Azure Machine learning now provides web services that are based on [Azure Resource Manager](#) allowing for new billing plan options and deploying your web service to multiple regions.

The general workflow to deploy a web service using Microsoft Azure Machine Learning Web Services is:

- Create a predictive experiment
- deploy it
- configure its name
- billing plan
- test it
- consume it.

The following graphic illustrates the workflow.

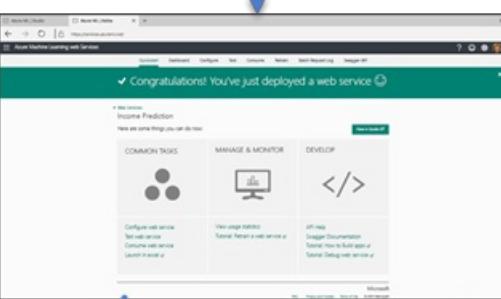
Create a predictive experiment in Machine Learning Studio and deploy it as a Web Service.



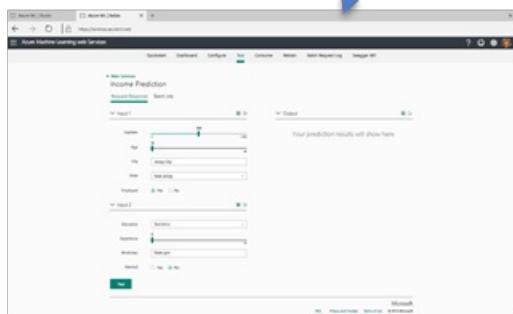
In Machine Learning Web Services deployment, name your service and select a billing plan.



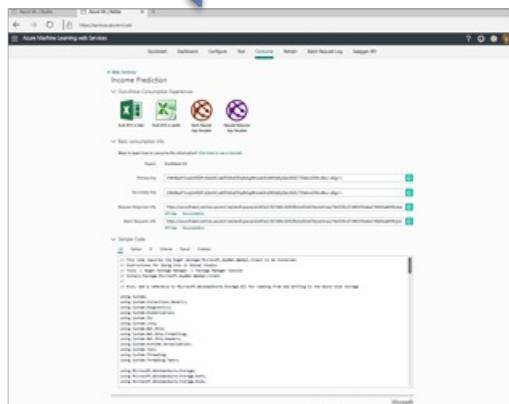
From the Web Service Quickstart, you can test your service and get the tools to consume it.



Test



Consume



Deploy web service from Studio

To deploy an experiment as a new web service. Sign into the Machine Learning Studio and create a new predictive web service.

Note: If you have already deployed an experiment as a classic web service you cannot deploy it as a new web service.

Click **Run** at the bottom of the experiment canvas and then click **Deploy Web Service** and **Deploy Web Service [New]**. The deployment page of the Machine Learning Web Service manager will open.

Machine Learning Web Service Manager Deploy Experiment Page

On the Deploy Experiment page, enter a name for the web service. Select a pricing plan. If you have an existing pricing plan you can select it, otherwise you must create a new price plan for the service.

1. In the **Price Plan** drop down, select an existing plan or select the **Select new plan** option.
2. In **Plan Name**, type a name that will identify the plan on your bill.
3. Select one of the **Monthly Plan Tiers**. Note that the plan tiers default to the plans for your default region and your web service is deployed to that region.

Click **Deploy** and the Quickstart page for your web service opens.

Quickstart page

The web service Quickstart page gives you access and guidance on the most common tasks you will perform after creating a new web service. From here you can easily access both the **Test** page and **Consume** page.

Testing your web service

From the Quickstart page, click Test web service under common tasks.

To test the web service as a Request-Response Service (RRS):

- Click **Test** on the menu bar.
- Click **Request-Response**.
- Enter appropriate values for the input columns of your experiment.
- Click Test **Request-Response**.

Your results will display on the right hand side of the page.

To test a Batch Execution Service (BES) web service, you will use a CSV file:

- Click **Test** on the menu bar.
- Click **Batch**.
- Under your input, click Browse and navigate to your sample data file.
- Click **Test**.

The status of your test is displayed under **Test Batch Jobs**.

Consuming your Web Service

When deployed as a web service, Azure Machine Learning experiments provide a REST API that can be consumed by a wide range of devices and platforms. This is because the simple REST API accepts and responds with JSON formatted messages. The Azure Machine Learning portal provides code that can be used to call the web service in R, C#, and Python.

On the Consuming page you can find:

- The API key and URI's for consuming web service in apps.
- Excel and web app templates to kick start your consumption process.
- Sample code in C#, python, and R to get you started.

For more information on consuming web services, see [How to consume an Azure Machine Learning web service that has been deployed from a Machine Learning experiment](#).

Next Steps

For more information on consuming web services, see:

[How to consume an Azure Machine Learning web service that has been deployed from a Machine Learning experiment](#)

[Azure Machine Learning Web Services: Deployment and consumption](#)

Deploy an Azure Machine Learning web service

1/17/2017 • 9 min to read • [Edit on GitHub](#)

Azure Machine Learning enables you to build, test, and deploy predictive analytic solutions.

From a high-level point-of-view, this is done in three steps:

- **Create a training experiment** - Azure Machine Learning Studio is a collaborative visual development environment that you use to train and test a predictive analytics model using training data that you supply.
- **Convert it to a predictive experiment** - Once your model has been trained with existing data and you're ready to use it to score new data, you prepare and streamline your experiment for predictions.
- **Deploy it as a web service** - You can deploy your predictive experiment as a [new](#) or [classic](#) Azure web service. Users can send data to your model and receive your model's predictions.

NOTE

[Try Azure Machine Learning for free](#)

No credit card or Azure subscription needed. [Get started now >](#)

Create a training experiment

To train a predictive analytics model, you use Azure Machine Learning Studio to create a training experiment where you include various modules to load training data, prepare the data as necessary, apply machine learning algorithms, and evaluate the results. You can iterate on an experiment and try different machine learning algorithms to compare and evaluate the results.

The process of creating and managing training experiments is covered more thoroughly elsewhere. For more information, see these articles:

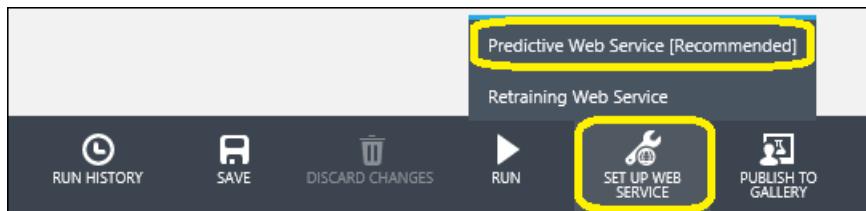
- [Create a simple experiment in Azure Machine Learning Studio](#)
- [Develop a predictive solution with Azure Machine Learning](#)
- [Import your training data into Azure Machine Learning Studio](#)
- [Manage experiment iterations in Azure Machine Learning Studio](#)

Convert the training experiment to a predictive experiment

Once you've trained your model, you're ready to convert your training experiment into a predictive experiment to score new data.

By converting to a predictive experiment, you're getting your trained model ready to be deployed as a scoring web service. Users of the web service can send input data to your model and your model will send back the prediction results. As you convert to a predictive experiment, keep in mind how you expect your model to be used by others.

To convert your training experiment to a predictive experiment, click **Run** at the bottom of the experiment canvas, click **Set Up Web Service**, then select **Predictive Web Service**.



For more information on how to perform this conversion, see [Convert a Machine Learning training experiment to a predictive experiment](#).

The following steps describe deploying a predictive experiment as a New web service. You can also deploy the experiment as Classic web service.

Deploy it as a web service

You can deploy the predictive experiment as a New web service or as a Classic web service.

Deploy the predictive experiment as a New web service

Now that the predictive experiment has been prepared, you can deploy it as a new Azure web service. Using the web service, users can send data to your model and the model will return its predictions.

To deploy your predictive experiment, click **Run** at the bottom of the experiment canvas. Once the experiment has finished running, click **Deploy Web Service** and select **Deploy Web Service New**. The deployment page of the Machine Learning Web Service portal opens.

Machine Learning Web Service portal Deploy Experiment Page

On the Deploy Experiment page, enter a name for the web service. Select a pricing plan. If you have an existing pricing plan you can select it, otherwise you must create a new price plan for the service.

1. In the **Price Plan** drop down, select an existing plan or select the **Select new plan** option.
2. In **Plan Name**, type a name that will identify the plan on your bill.
3. Select one of the **Monthly Plan Tiers**. The plan tiers default to the plans for your default region and your web service is deployed to that region.

Click **Deploy** and the **Quickstart** page for your web service opens.

The web service Quickstart page gives you access and guidance on the most common tasks you will perform after creating a web service. From here, you can easily access both the Test page and Consume page.

Test your New web service

To test your new web service, click **Test web service** under common tasks. On the Test page, you can test your web service as a Request-Response Service (RRS) or a Batch Execution service (BES).

The RRS test page displays the inputs, outputs, and any global parameters that you have defined for the experiment. To test the web service, you can manually enter appropriate values for the inputs or supply a comma separated value (CSV) formatted file containing the test values.

To test using RRS, from the list view mode, enter appropriate values for the inputs and click **Test Request-Response**. Your prediction results display in the output column to the left.

Request-Response Batch

input1 List View CSV File Input

output1

Your prediction results will display here.

Class	1
sepal-length	1
sepal-width	1
petal-length	1
petal-width	1

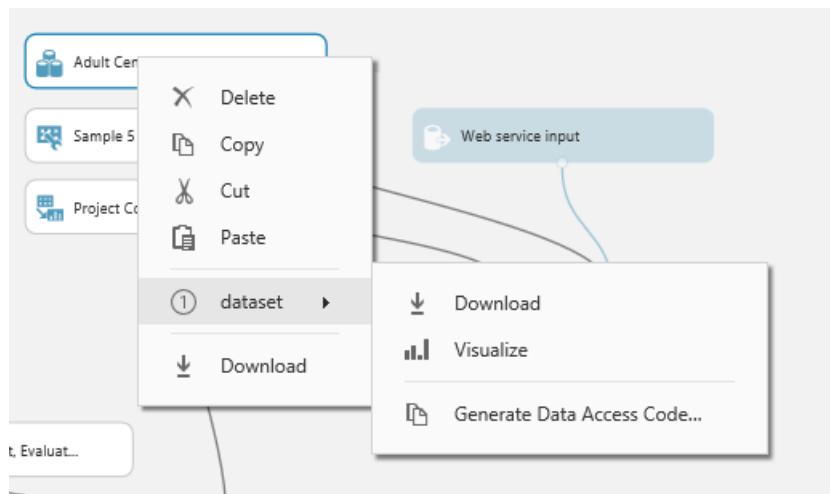
Global Parameters

Category	Trigonometric
Trigonometric Function	Sin
Column Set	{"isFilter":true,"rules":[{"ruleType":"ColumnTy
Output mode	ResultOnly

Test Request-Response

To test your BES, click **Batch**. On the Batch test page, click Browse under your input and select a CSV file containing appropriate sample values. If you don't have a CSV file, and you created your predictive experiment using Machine Learning Studio, you can download the data set for your predictive experiment and use it.

To download the data set, open Machine Learning Studio. Open your predictive experiment and right click the input for your experiment. From the context menu, select **dataset** and then select **Download**.



Click **Test**. The status of your Batch Execution job displays to the right under **Test Batch Jobs**.

The screenshot shows the 'Batch' configuration page in Azure Machine Learning Studio. At the top, there's a 'Request-Response' section with a 'Batch' tab selected. Below it, there's a 'input1' section with a 'Browse...' button and a 'Test Batch Jobs' section with a note: 'Your batch job status will display here.' Under 'Global Parameters', there are four fields: 'Category' (Trigonometric), 'Trigonometric Function' (Sin), 'Column Set' ({"isFilter":true,"rules":[{"ruleType":"ColumnTy}], and 'Output mode' (ResultOnly). A note at the bottom says 'Note: We will enable CORS on your storage account to upload this file'. A 'Test' button is also present.

On the **CONFIGURATION** page, you can change the description, title, update the storage account key, and enable sample data for your web service.

The screenshot shows the 'Configure' page for a 'ratings' web service in the Microsoft Azure Machine Learning Web Services portal. The page has tabs for Quickstart, Dashboard, Batch Request Log, Configure (which is selected), Consume, Test, and Swagger API. The 'Configure' section includes fields for 'Description' (Recommender: Restaurant ratings [Predictive Exp.]), 'Title' (Restaurant Ratings Sample), 'Primary Key' (empty), 'Secondary Key' (empty), 'Storage Account Name' (checkbox for 'Update Key for mitestaccount'), and 'Sample Data Enabled?' (radio buttons for 'Yes' and 'No'). At the bottom are 'Cancel' and 'Save' buttons. The footer includes links for Microsoft, FAQ, Privacy and Cookies, Terms of Use, and © Microsoft.

Once you've deployed the web service, you can:

- **Access** it through the web service API.
- **Manage** it through Azure Machine Learning web services portal or the Azure classic portal.
- **Update** it if your model changes.

Access your New web service

Once you deploy your web service from Machine Learning Studio, you can send data to the service and receive responses programmatically.

The **Consume** page provides all the information you need to access your web service. For example, the API key is provided to allow authorized access to the service.

For more information about accessing a Machine Learning web service, see [How to consume a deployed](#)

Azure Machine Learning web service.

Manage your New web service

You can manage your classic web services Machine Learning Web Services portal. From the [main portal page](#), click **Web Services**. From the web services page, you can delete or copy a service. To monitor a specific service, click the service and then click **Dashboard**. To monitor batch jobs associated with the web service, click **Batch Request Log**.

Deploy the predictive experiment as a Classic web service

Now that the predictive experiment has been sufficiently prepared, you can deploy it as a Classic Azure web service. Using the web service, users can send data to your model and the model will return its predictions.

To deploy your predictive experiment, click **Run** at the bottom of the experiment canvas and then click **Deploy Web Service**. The web service is set up and you are placed in the web service dashboard.



Test your Classic web service

You can test the web service in either the Machine Learning Web Services portal or Machine Learning Studio.

To test the Request Response web service, click the **Test** button in the web service dashboard. A dialog pops up to ask you for the input data for the service. These are the columns expected by the scoring experiment. Enter a set of data and then click **OK**. The results generated by the web service are displayed at the bottom of the dashboard.

You can click the **Test** preview link to test your service in the Azure Machine Learning Web Services portal as shown previously in the New web service section.

To test the Batch Execution Service, click **Test** preview link . On the Batch test page, click Browse under your input and select a CSV file containing appropriate sample values. If you don't have a CSV file, and you created your predictive experiment using Machine Learning Studio, you can download the data set for your predictive experiment and use it.

Default Endpoint		TEST	APPS	LAST UPDATED	
API HELP PAGE		Test	Test preview	Excel 2013 or later Excel 2010 or earlier	10/12/2016 9:26:08 AM
REQUEST/RESPONSE					
BATCH EXECUTION		Test	Test preview	Excel 2013 or later workbook	10/12/2016 9:26:08 AM

On the **CONFIGURATION** page, you can change the display name of the service and give it a description. The name and description is displayed in the [Azure classic portal](#) where you manage your web services.

You can provide a description for your input data, output data, and web service parameters by entering a string for each column under **INPUT SCHEMA**, **OUTPUT SCHEMA**, and **Web SERVICE PARAMETER**. These descriptions are used in the sample code documentation provided for the web service.

You can enable logging to diagnose any failures that you're seeing when your web service is accessed. For more information, see [Enable logging for Machine Learning web services](#).

The screenshot shows the 'income level prediction' experiment in the Azure Machine Learning Studio. The left sidebar has icons for Dashboard, Configuration, Datasets, Models, and Experiments. The main area shows the experiment details:

- GENERAL**
 - Display Name: Income level prediction
 - Description: Predicts income level based on user data
- ENABLE LOGGING**
 - YES
 - NO**
 - [Learn more](#)
- INPUT SCHEMA**
 - age (Numeric)
 - fnlwgt (Numeric)
 - education (String)

At the bottom are buttons for **NEW**, **SAVE**, and **DISCARD**.

You can also configure the endpoints for the web service in the Azure Machine Learning Web Services portal similar to the procedure shown previously in the New web service section. The options are different, you can add or change the service description, enable logging, and enable sample data for testing.

Access your Classic web service

Once you deploy your web service from Machine Learning Studio, you can send data to the service and receive responses programmatically.

The dashboard provides all the information you need to access your web service. For example, the API key is provided to allow authorized access to the service, and API help pages are provided to help you get started writing your code.

For more information about accessing a Machine Learning web service, see [How to consume a deployed Azure Machine Learning web service](#).

Manage your Classic web service

There are various of actions you can perform to monitor a web service. You can update it, and delete it. You can also add additional endpoints to a Classic web service in addition to the default endpoint that is created when you deploy it.

For more information, see [Manage an Azure Machine Learning workspace](#) and [Manage a web service using the Azure Machine Learning Web Services portal](#).

Update the web service

You can make changes to your web service, such as updating the model with additional training data, and deploy it again, overwriting the original web service.

To update the web service, open the original predictive experiment you used to deploy the web service and make an editable copy by clicking **SAVE AS**. Make your changes and then click **Deploy Web Service**.

Because you've deployed this experiment before, you are asked if you want to overwrite (Classic Web Service)

or update (New web service) the existing service. Clicking **YES** or **Update** stops the existing web service and deploys the new predictive experiment in its place.

NOTE

If you made configuration changes in the original web service, for example, entering a new display name or description, you will need to enter those values again.

One option for updating your web service is to retrain the model programmatically. For more information, see [Retrain Machine Learning models programmatically](#).

Deploying Azure ML web services that use Data Import and Data Export modules

1/17/2017 • 5 min to read • [Edit on GitHub](#)

When you create a predictive experiment, you typically add a web service input and output. When you deploy the experiment, consumers can send and receive data from the web service through the inputs and outputs. For some applications, a consumer's data may be available from a data feed or already reside in an external data source such as Azure Blob storage. In these cases, they do not need read and write data using web service inputs and outputs. They can, instead, use the Batch Execution Service (BES) to read data from the data source using an Import Data module and write the scoring results to a different data location using an Export Data module.

The Import Data and Export data modules, can read from and write to a number of data locations such as a Web URL via HTTP, a Hive Query, an Azure SQL database, Azure Table storage, Azure Blob storage, a Data Feed provider, or an on-premise SQL database.

This topic uses the "Sample 5: Train, Test, Evaluate for Binary Classification: Adult Dataset" sample and assumes the dataset has already been loaded into an Azure SQL table named `censusdata`.

Create the training experiment

When you open the "Sample 5: Train, Test, Evaluate for Binary Classification: Adult Dataset" sample it uses the sample Adult Census Income Binary Classification dataset. And the experiment in the canvas will look similar to the following image.



To read the data from the Azure SQL table:

1. Delete the dataset module.
2. In the components search box, type import.
3. From the results list, add an *Import Data* module to the experiment canvas.
4. Connect output of the *Import Data* module the input of the *Clean Missing Data* module.
5. In properties pane, select **Azure SQL Database** in the **Data Source** dropdown.
6. In the **Database server name**, **Database name**, **User name**, and **Password** fields, enter the appropriate information for your database.
7. In the Database query field, enter the following query.

```
select [age],
```

```
[workclass],  
[fnlwgt],  
[education],  
[education-num],  
[marital-status],  
[occupation],  
[relationship],  
[race],  
[sex],  
[capital-gain],  
[capital-loss],  
[hours-per-week],  
[native-country],  
[income]
```

```
from dbo.censusdata;
```

8. At the bottom of the experiment canvas, click **Run**.

Create the predictive experiment

Next you set up the predictive experiment from which you will deploy your web service.

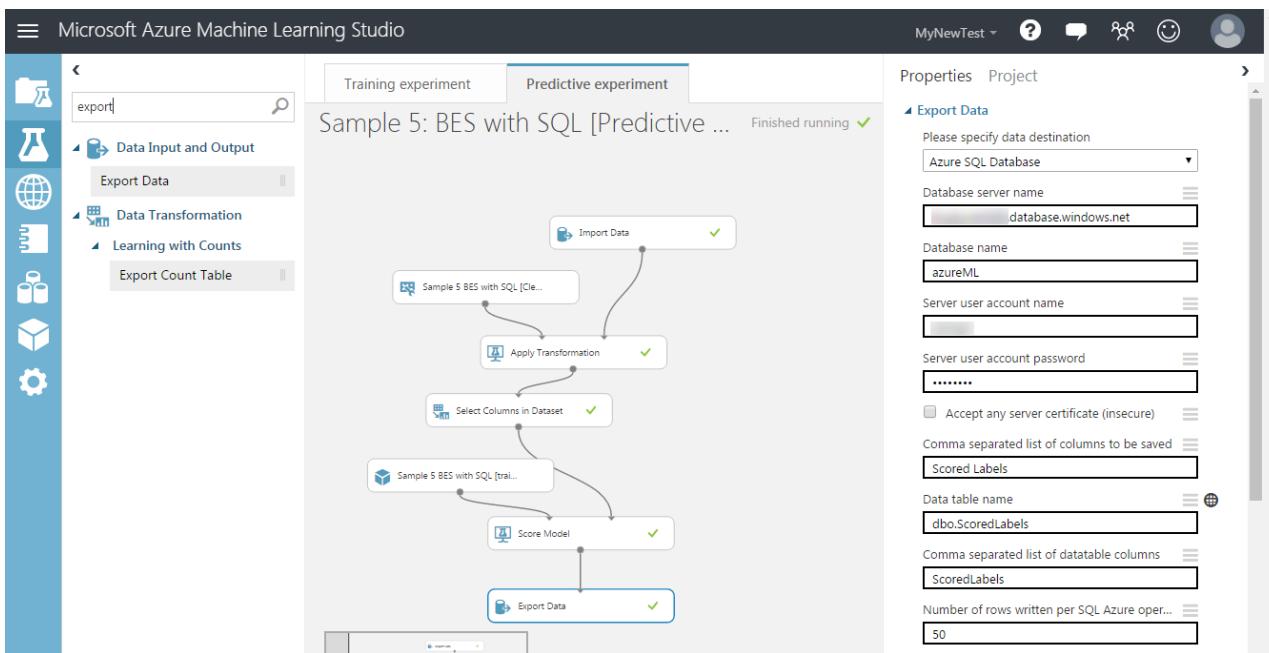
1. At the bottom of the experiment canvas, click **Set Up Web Service** and select **Predictive Web Service [Recommended]**.
2. Remove the *Web Service Input* and *Web Service Output modules* from the predictive experiment.
3. In the components search box, type export.
4. From the results list, add an *Export Data* module to the experiment canvas.
5. Connect output of the *Score Model* module the input of the *Export Data* module.
6. In properties pane, select **Azure SQL Database** in the data destination dropdown.
7. In the **Database server name**, **Database name**, **Server user account name**, and **Server user account password** fields, enter the appropriate information for your database.
8. In the **Comma separated list of columns to be saved** field, type Scored Labels.
9. In the **Data table name field**, type dbo.ScoredLabels. If the table does not exist, it is created when the experiment is run or the web service is called.
10. In the **Comma separated list of datatable columns** field, type ScoredLabels.

When you write an application that calls the final web service, you may want to specify a different input query or destination table at run time. To configure these inputs and outputs, you can use the Web Service Parameters feature to set the *Import Data* module *Data source* property and the *Export Data* mode data destination property. For more information on Web Service Parameters, see the [AzureML Web Service Parameters entry](#) on the Cortana Intelligence and Machine Learning Blog.

To configure the Web Service Parameters for the import query and the destination table:

1. In the properties pane for the *Import Data* module, click the icon at the top right of the **Database query** field and select **Set as web service parameter**.
2. In the properties pane for the *Export Data* module, click the icon at the top right of the **Data table name** field and select **Set as web service parameter**.
3. At the bottom of the *Export Data* module properties pane, in the **Web Service Parameters** section, click Database query and rename it Query.
4. Click **Data table name** and rename it **Table**.

When you are done, your experiment should look similar to the following image.



Now you can deploy the experiment as a web service.

Deploy the web service

You can deploy to either a Classic or New web service.

Deploy a Classic Web Service

To deploy as a Classic Web Service and create an application to consume it:

1. At the bottom of the experiment canvas, click Run.
2. When the run has completed, click **Deploy Web Service** and select **Deploy Web Service [Classic]**.
3. On the web service dashboard, locate your API key. Copy and save it to use later.
4. In the **Default Endpoint** table, click the **Batch Execution** link to open the API Help Page.
5. In Visual Studio, create a C# console application.
6. On the API Help Page, find the **Sample Code** section at the bottom of the page.
7. Copy and paste the C# sample code into your Program.cs file, and remove all references to the blob storage.
8. Update the value of the *apiKey* variable with the API key saved earlier.
9. Locate the request declaration and update the values of Web Service Parameters that are passed to the *Import Data* and *Export Data* modules. In this case, you will use the original query, but define a new table name.

```
var request = new BatchExecutionRequest()
{
    GlobalParameters = new Dictionary<string, string>()
    {
        {"Query", @"select [age], [workclass], [fnlwgt], [education], [education-num], [marital-status], [occupation], [relationship], [race], [sex], [capital-gain], [capital-loss], [hours-per-week], [native-country], [income] from dbo.censusdata" },
        {"Table", "dbo.ScoredTable2" }
    }
};
```

10. Run the application.

On completion of the run, a new table is added to the database containing the scoring results.

Deploy a New Web Service

To deploy as a New Web Service and create an application to consume it:

1. At the bottom of the experiment canvas, click **Run**.

2. When the run has completed, click **Deploy Web Service** and select **Deploy Web Service [New]**.
3. On the Deploy Experiment page, enter a name for your web service and select a pricing plan, then click **Deploy**.
4. On the **Quickstart** page, click **Consume**.
5. In the **Sample Code** section, click **Batch**.
6. In Visual Studio, create a C# console application.
7. Copy and paste the C# sample code into your Program.cs file.
8. Update the value of the *apiKey* variable with the **Primary Key** located in the **Basic consumption info** section.
9. Locate the *scoreRequest* declaration and update the values of Web Service Parameters that are passed to the *Import Data* and *Export Data* modules. In this case, you will use the original query, but define a new table name.

```
var scoreRequest = new
{
    Inputs = new Dictionary<string, StringTable>()
    {
        },
    GlobalParameters = new Dictionary<string, string>() {
        {"Query", @"select [age], [workclass], [fnlwgt], [education], [education-num], [marital-status], [occupation], [relationship], [race],
        [sex], [capital-gain], [capital-loss], [hours-per-week], [native-country], [income] from dbo.censusdata" },
        {"Table", "dbo.ScoredTable3" },
    }
};
```

10. Run the application.

How to deploy a Web Service to multiple regions

1/17/2017 • 1 min to read • [Edit on GitHub](#)

The New Azure Web Services allow you to easily deploy a web service to multiple regions without needing multiple subscriptions or workspaces.

Pricing is region specific, therefore you must define a billing plan for each region in which you will deploy the web service.

To create a plan in another region

1. Sign into [Microsoft Azure Machine Learning Web Services](#).
2. Click the **Plans** menu option.
3. On the Plans over view page, click **New**.
4. From the **Subscription** dropdown, select the subscription in which the new plan will reside.
5. From the **Region** dropdown, select a region for the new plan. The Plan Options for the selected region will display in the **Plan Options** section of the page.
6. From the **Resource Group** dropdown, select a resource group for the plan. From more information on resource groups, see [Azure Resource Manager overview](#).
7. In **Plan Name** type the name of the plan.
8. Under **Plan Options**, click the billing level for the new plan.
9. Click **Create**.

Deploying the web service to another region

1. Click the **Web Services** menu option.
2. Select the Web Service you are deploying to a new region.
3. Click **Copy**.
4. In **Web Service Name**, type a new name for the web service.
5. In **Web service description**, type a description for the web service.
6. From the **Subscription** dropdown, select the subscription in which the new web service will reside.
7. From the **Resource Group** dropdown, select a resource group for the web service. From more information on resource groups, see [Azure Resource Manager overview](#).
8. From the **Region** dropdown, select the region in which to deploy the web service.
9. From the **Storage account** dropdown, select a storage account in which to store the web service.
10. From the **Price Plan** dropdown, select a plan in the region you selected in step 8.
11. Click **Copy**.

(deprecated) Publish Azure Machine Learning Web Service to the Azure Marketplace

1/17/2017 • 3 min to read • [Edit on GitHub](#)

NOTE

DataMarket and Data Services are being retired, and existing subscriptions will be retired and cancelled starting 3/31/2017. As a result, this article is being deprecated.

As an alternative, you can publish your Machine Learning experiments to the [Cortana Intelligence Gallery](#) for the benefit of the data science community. For more information, see [Share and discover resources in the Cortana Intelligence Gallery](#).

The Azure Marketplace provides the ability to publish Azure Machine Learning web services as paid or free services for consumption by external customers. This article provides an overview of that process with links to guidelines to get you started. By using this process, you can make your web services available for other developers to consume in their applications.

NOTE

[Try Azure Machine Learning for free](#)

No credit card or Azure subscription needed. [Get started now >](#)

Overview of the publishing process

The following are the steps for publishing an Azure Machine Learning web service to Azure Marketplace:

1. Create and publish a Machine Learning Request-Response service (RRS)
2. Deploy the service to production, and obtain the API Key and OData endpoint information.
3. Use the URL of the published web service to publish to [Azure Marketplace \(Data Market\)](#)
4. Once submitted, your offer is reviewed and needs to be approved before your customers can start purchasing it.
The publishing process can take a few business days.

Walk through

Step 1: Create and publish a Machine Learning Request-Response service (RRS)

If you have not done this already, please take a look at this [walk through](#).

Step 2: Deploy the service to production, and obtain the API Key and OData endpoint information

1. From the [Azure Classic Portal](#), select the **MACHINE LEARNING** option from the left navigation bar, and select your workspace.
2. Click on the **WEB SERVICES** tab, and select the web service you would like to publish to the marketplace.

3. Select the endpoint you would like to have the marketplace consume. If you have not created any additional endpoints, you can select the **Default** endpoint.
4. Once you have clicked on the endpoint, you will be able to see the **API KEY**. You will need this piece of information later on in Step 3, so make a copy of it.

5. Click on the **REQUEST/RESPONSE** method, at this point we do not support publishing batch execution services to the marketplace. That will take you to the API help page for the Request/Response method.
6. Copy the **OData Endpoint Address**, you will need this information later on in Step 3.

Request Response API Documentation for Experiment created on 2/19/2015

Updated: 02/19/2015 23:12

No description provided for this web service.

- [Previous version of this API](#)
- [Submit a request](#)
- [Input Parameters](#)
- [Output Parameters](#)
- [Sample Code](#)

OData Endpoint Address

`https://ussouthcentral.services.azureml.net/odata/workspaces/290db9c376af4c2ca10030d161eafafa/services/520f9230300743ed98bff834f62e8eb8`

Request

Method	Request URI	HTTP Version
POST	<code>https://ussouthcentral.services.azureml.net/workspaces/290db9c376af4c2ca10030d161eafafa/services/520f9230300743ed98bff834f62e8eb8/execute?</code> <code>api-version=2.0&details=true</code>	HTTP/1.1

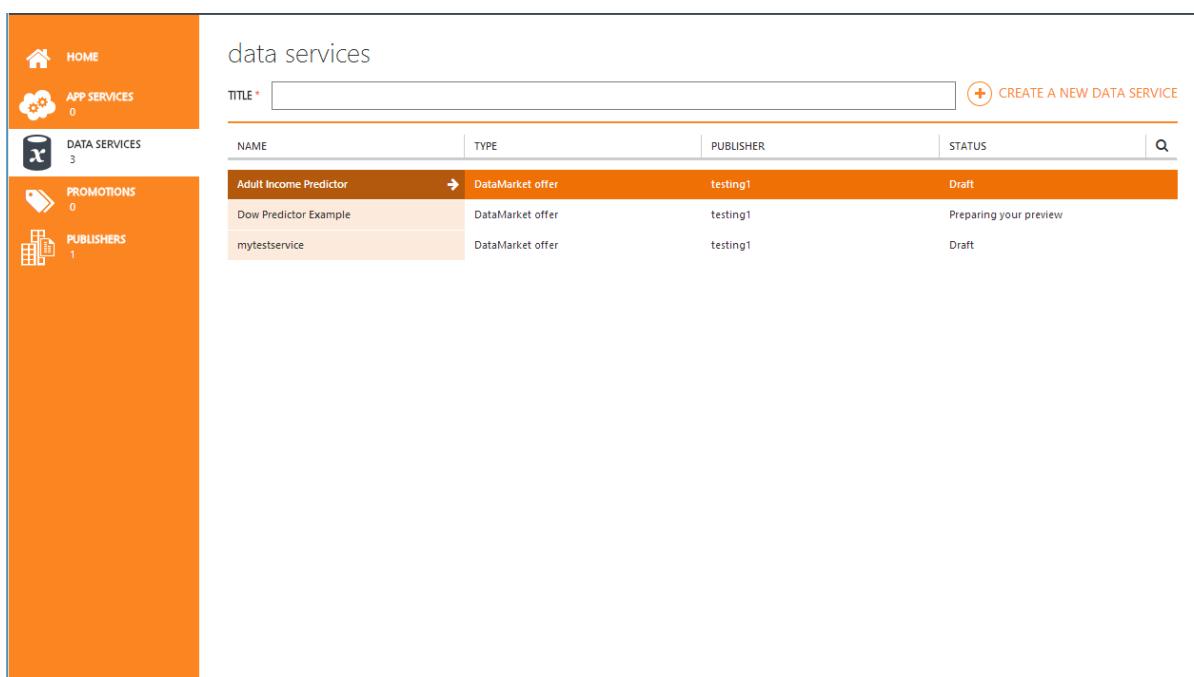
Note: You may omit the `details` parameter from the query string. This would cause `ColumnTypes` to be omitted from the output

Request Headers

deploy the service into production.

Step 3: Use the URL of the published web service to publish to Azure Marketplace (DataMarket)

1. Navigate to [Azure Marketplace \(Data Market\)](#)
2. Click on the **Publish** link at the top of the page. This will take you to the [Microsoft Azure Publishing Portal](#)
3. Click on the **publishers** section to register as a publisher.
4. When creating a new offer, select **Data Services**, then click **Create a New Data Service**.



The screenshot shows the Microsoft Azure Publishing Portal interface. On the left, there's a sidebar with icons for HOME, APP SERVICES (0), DATA SERVICES (3), PROMOTIONS (0), and PUBLISHERS (1). The main area is titled "data services". It has a search bar labeled "TITLE *". Below it is a table with columns: NAME, TYPE, PUBLISHER, and STATUS. The table contains three rows:

NAME	TYPE	PUBLISHER	STATUS
Adult Income Predictor	DataMarket offer	testing1	Draft
Dow Predictor Example	DataMarket offer	testing1	Preparing your preview
mytestservice	DataMarket offer	testing1	Draft

5. Under **Plans** provide information on your offering, including a pricing plan. Decide if you will offer a free or paid service. To get paid, provide payment information such as your bank and tax information.
6. Under **Marketing** provide information about your offer, such as the title and description for your offer.
7. Under **Pricing** you can set the price for your plans for specific countries, or let the system "autoprice" your offer.
8. On the **Data Service** tab, click **Web Service** as the **Data Source**.

The screenshot shows the 'Data Service' configuration page for a service named 'mytestservice'. The left sidebar lists other services like 'Adult Income Pred...' and 'Dow Predictor Exam...'. The main area has tabs for 'Data Service', 'Data Sources', and 'Annotations'. Under 'Data Service', there's a 'NAMESPACE' field, a 'PUBLIC API URL' field containing 'https://api.datamarket.azure.com/testing1/<ServiceNamespace>', and a 'Version 1' section. In the 'Version 1' section, there's a 'CONNECTION INFO' group with 'Service Url *' (empty), 'Authentication Scheme' (set to 'Header'), 'Header Name *' (empty), and 'Header Value *' (empty). Below this is a 'TEST CONNECTION' button. There's also a 'CSDL MAPPING' section with 'No Mapping' and a checked 'This service is OData' checkbox. Under 'Annotations', there's a 'Namespace' field with an empty input and a '+' icon.

9. Get the web service URL and API key from the Azure Classic Portal, as explained in step 2 above.
10. In the Marketplace Data Service setup dialog box, paste the OData endpoint address into the **Service URL** text box.
11. For **Authentication**, choose **Header** as the **Authentication Scheme**.
 - Enter "Authorization" for the **Header Name**.
 - For the **Header Value**, enter "Bearer" (without the quotation marks), click the **Space** bar, and then paste the API key.
 - Select the **This Service is OData** check box.
 - Click **Test Connection** to test the connection.
12. Under **Categories**, ensure **Machine Learning** is selected.
13. When you are done entering all the metadata about your offer, click on **Publish**, and then **Push to Staging**. At this point, you will be notified of any remaining issues that you need to fix.
14. After you have ensured completion of all the outstanding issues, click on **Request approval to push to Production**. The publishing process can take a few business days.

Use Azure Machine Learning Web Service Parameters

1/17/2017 • 4 min to read • [Edit on GitHub](#)

An Azure Machine Learning web service is created by publishing an experiment that contains modules with configurable parameters. In some cases, you may want to change the module behavior while the web service is running. *Web Service Parameters* allow you to do this task.

A common example is setting up the [Import Data](#) module so that the user of the published web service can specify a different data source when the web service is accessed. Or configuring the [Export Data](#) module so that a different destination can be specified. Some other examples include changing the number of bits for the [Feature Hashing](#) module or the number of desired features for the [Filter-Based Feature Selection](#) module.

You can set Web Service Parameters and associate them with one or more module parameters in your experiment, and you can specify whether they are required or optional. The user of the web service can then provide values for these parameters when they call the web service.

NOTE

[Try Azure Machine Learning for free](#)

No credit card or Azure subscription needed. [Get started now >](#)

How to set and use Web Service Parameters

You define a Web Service Parameter by clicking the icon next to the parameter for a module and selecting "Set as web service parameter". This creates a new Web Service Parameter and connects it to that module parameter. Then, when the web service is accessed, the user can specify a value for the Web Service Parameter and it is applied to the module parameter.

Once you define a Web Service Parameter, it's available to any other module parameter in the experiment. If you define a Web Service Parameter associated with a parameter for one module, you can use that same Web Service Parameter for any other module, as long as the parameter expects the same type of value. For example, if the Web Service Parameter is a numeric value, then it can only be used for module parameters that expect a numeric value. When the user sets a value for the Web Service Parameter, it will be applied to all associated module parameters.

You can decide whether to provide a default value for the Web Service Parameter. If you do, then the parameter is optional for the user of the web service. If you don't provide a default value, then the user is required to enter a value when the web service is accessed.

The API documentation for the web service includes information for the web service user on how to specify the Web Service Parameter programmatically when accessing the web service.

NOTE

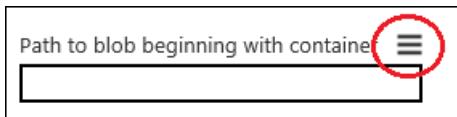
The API documentation for a classic web service is provided through the [API help page](#) link in the web service **DASHBOARD** in Machine Learning Studio. The API documentation for a new web service is provided through the [Azure Machine Learning Web Services](#) portal on the [Consume](#) and [Swagger API](#) pages for your web service.

Example

As an example, let's assume we have an experiment with an [Export Data](#) module that sends information to Azure

blob storage. We'll define a Web Service Parameter named "Blob path" that allows the web service user to change the path to the blob storage when the service is accessed.

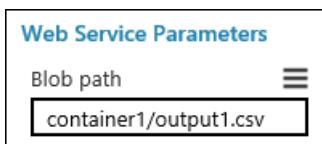
1. In Machine Learning Studio, click the [Export Data](#) module to select it. Its properties are shown in the Properties pane to the right of the experiment canvas.
2. Specify the storage type:
 - Under **Please specify data destination**, select "Azure Blob Storage".
 - Under **Please specify authentication type**, select "Account".
 - Enter the account information for the Azure blob storage.
3. Click the icon to the right of the **Path to blob beginning with container parameter**. It looks like this:



Select "Set as web service parameter".

An entry is added under **Web Service Parameters** at the bottom of the Properties pane with the name "Path to blob beginning with container". This is the Web Service Parameter that is now associated with this [Export Data](#) module parameter.

4. To rename the Web Service Parameter, click the name, enter "Blob path", and press the **Enter** key.
5. To provide a default value for the Web Service Parameter, click the icon to the right of the name, select "Provide default value", enter a value (for example, "container1/output1.csv"), and press the **Enter** key.



6. Click **Run**.
7. Click **Deploy Web Service** and select **Deploy Web Service [Classic]** or **Deploy Web Service [New]** to deploy the web service.

The user of the web service can now specify a new destination for the [Export Data](#) module when accessing the web service.

More information

For a more detailed example, see the [Web Service Parameters](#) entry in the [Machine Learning Blog](#).

For more information on accessing a Machine Learning web service, see [How to consume a published machine learning web service](#).

Enable logging for Machine Learning web services

1/17/2017 • 1 min to read • [Edit on GitHub](#)

This document provides information on the logging capability of Classic Web services. Enabling logging in Web services provides additional information, beyond just an error number and a message, that can help you troubleshoot your calls to the Machine Learning APIs.

To enable logging in Web Services in the Azure classic portal:

1. Sign in to [Azure classic portal](#)
2. In the left features column, click **MACHINE LEARNING**.
3. Click your workspace, then **WEB SERVICES**.
4. In the Web services list, click the name of the Web service.
5. In the endpoints list, click the endpoint name.
6. Click **CONFIGURE**.
7. Set **DIAGNOSTICS TRACE LEVEL** to *Error* or *All*, then click **SAVE**.

To enable logging in the Azure Machine Learning Web Services portal.

1. Sign into the [Azure Machine Learning Web Services](#) portal.
2. Click Classic Web Services.
3. In the Web services list, click the name of the Web service.
4. In the endpoints list, click the endpoint name.
5. Click **Configure**.
6. Set **Logging** to *Error* or *All*, then click **SAVE**.

The effects of enabling logging

When logging is enabled, all the diagnostics and errors from the selected endpoint are logged to the Azure Storage Account linked with the user's workspace. You can see this storage account in the Azure classic portal Dashboard view (bottom of the Quick Glance section) of their workspace.

The logs can be viewed using any of the several tools available to 'explore' an Azure Storage Account. The easiest may be to simply navigate to the Storage Account in the Azure classic portal and then click **CONTAINERS**. You would then see a Container named **ml-diagnostics**. This container holds all the diagnostics information for all the web service endpoints for all the workspaces associated with this Storage account.

Log blob detail information

Each blob in the container holds the diagnostics info for exactly one of the following:

- An execution of the Batch-Execution method
- An execution of the Request-Response method
- Initialization of a Request-Response container

The name of each blob has a prefix of the following form:

{Workspace Id}-{Web service Id}-{Endpoint Id}/{Log type}

Where Log type is one of the following values:

- batch
- score/requests
- score/init

Creating Endpoints

1/17/2017 • 2 min to read • [Edit on GitHub](#)

NOTE

This topic describes techniques applicable to a classic Web service.

When you create Web services that you sell forward to your customers, you need to provide trained models to each customer that are still linked to the experiment from which the Web service was created. In addition, any updates to the experiment should be applied selectively to an endpoint without overwriting the customizations.

To accomplish this, Azure Machine Learning allows you to create multiple endpoints for a deployed Web service. Each endpoint in the Web service is independently addressed, throttled, and managed. Each endpoint is a unique URL and authorization key that you can distribute to your customers.

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

Adding endpoints to a Web service

There are three ways to add an endpoint to a Web service.

- Programmatically
- Through the Azure Machine Learning Web Services portal
- Through the Azure classic portal

Once the endpoint is created, you can consume it through synchronous APIs, batch APIs, and excel worksheets. In addition to adding endpoints through this UI, you can also use the Endpoint Management APIs to programmatically add endpoints.

NOTE

If you have added additional endpoints to the Web service, you cannot delete the default endpoint.

Adding an endpoint programmatically

You can add an endpoint to your Web service programmatically using the [AddEndpoint sample code](#).

Adding an endpoint using the Azure Machine Learning Web Services portal

1. In Machine Learning Studio, on the left navigation column, click Web Services.
2. At the bottom of the Web service dashboard, click **Manage endpoints**. The Azure Machine Learning Web Services portal opens to the endpoints page for the Web service.
3. Click **New**.

4. Type a name and description for the new endpoint. Endpoint names must be 24 character or less in length, and must be made up of lower-case alphabets or numbers. Select the logging level and whether sample data is enabled. For more information on logging, see [Enable logging for Machine Learning Web services](#).

Adding an endpoint using the Azure classic portal

1. Sign in to the [Azure classic portal](#), click **Machine Learning** in the left column. Click the workspace which contains the Web service in which you are interested.

NAME	STORAGE	STATUS	OWNER	SUBSCRIPTION	LOCATION
mltest3	→	✓ Online		Visual Studio Ultimate with...	

2. Click **Web Services**.

NAME	ENDPOINTS
ws1	2
training	1
scoring	1
reader testing	1

3. Click the Web service you're interested in to see the list of available endpoints.

The screenshot shows the Microsoft Azure portal interface. On the left, there's a sidebar with various icons for different services like storage, databases, and machine learning. The main area is titled 'ws1' and shows a table of endpoints. The table has columns for NAME, VERSION, UPDATE AVAILABLE, and THROTTLE LEVEL. One endpoint is listed: 'second' (Version 10/24/2014 4:23:40 PM), with 'false' for update available and 'High' for throttle level. To the right of the table, there's a 'quick glance' section with details: WEB SERVICE ID (cb8cedcb4b494bdb9a81e5dc6251a08a), DESCRIPTION (No description provided for this web service), SUBSCRIPTION NAME (Visual Studio Ultimate with MSDN), and LOCATION (South Central US). At the bottom, there are buttons for ADD ENDPOINT, UPDATE ENDPOINT, and DELETE ENDPOINT.

- At the bottom of the page, click **Add Endpoint**. Type a name and description, ensure there are no other endpoints with the same name in this Web service. Leave the throttle level with its default value unless you have special requirements. To learn more about throttling, see [Scaling API Endpoints](#).

The screenshot shows a modal dialog box titled 'Add a new endpoint'. It has three input fields: 'Name' (with a placeholder 'bc9'), 'Description' (an empty text area), and 'Throttle Level' (set to 'High'). In the bottom right corner of the dialog, there's a large checkmark icon.

Next Steps

[How to consume a published Azure Machine Learning Web service.](#)

Manage a Web service using the Azure Machine Learning Web Services portal

1/17/2017 • 8 min to read • [Edit on GitHub](#)

You can manage your Machine Learning New and Classic Web services using the Microsoft Azure Machine Learning Web Services portal. Since Classic Web services and New Web services are based on different underlying technologies, you have slightly different management capabilities for each of them.

In the Machine Learning Web Services portal you can:

- Monitor how the web service is being used.
- Configure the description, update the keys for the web service (New only), update your storage account key (New only), enable logging, and enable or disable sample data.
- Delete the web service.
- Create, delete, or update billing plans (New only).
- Add and delete endpoints (Classic only)

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

Manage New Web services

To manage your New Web services:

1. Sign in to the [Microsoft Azure Machine Learning Web Services](#) portal using your Microsoft Azure account - use the account that's associated with the Azure subscription.
2. On the menu, click **Web Services**.

This displays a list of deployed Web services for your subscription.

To manage a Web service, click Web Services. From the Web Services page you can:

- Click the web service to manage it.
- Click the Billing Plan for the web service to update it.
- Delete a web service.
- Copy a web service and deploy it to another region.

When you click a web service, the web service Quickstart page opens. The web service Quickstart page has two menu options that allow you to manage your web service:

- **DASHBOARD** - Allows you to view Web service usage.
- **CONFIGURE** - Allows you to add descriptive text, update the key for the storage account associated with the Web service, and enable or disable sample data.

Monitoring how the web service is being used

Click the **DASHBOARD** tab.

From the dashboard, you can view overall usage of your Web service over a period of time. You can select the

period to view from the Period dropdown menu in the upper right of the usage charts. The dashboard shows the following information:

- **Requests Over Time** displays a step graph of the number of requests over the selected time period. It can help identify if you are experiencing spikes in usage.
- **Request-Response Requests** displays the total number of Request-Response calls the service has received over the selected time period and how many of them failed.
- **Average Request-Response Compute Time** displays an average of the time needed to execute the received requests.
- **Batch Requests** displays the total number of Batch Requests the service has received over the selected time period and how many of them failed.
- **Average Job Latency** displays an average of the time needed to execute the received requests.
- **Errors** displays the aggregate number of errors that have occurred on calls to the web service.
- **Services Costs** displays the charges for the billing plan associated with the service.

Configuring the web service

Click the **CONFIGURE** menu option.

You can update the following properties:

- **Description** allows you to enter a description for the Web service.
- **Title** allows you to enter a title for the Web service
- **Keys** allows you to rotate your primary and secondary API keys.
- **Storage account key** allows you to update the key for the storage account associated with the Web service changes.
- **Enable Sample data** allows you to provide sample data that you can use to test the Request-Response service. If you created the web service in Machine Learning Studio, the sample data is taken from the data you used to train your model. If you created the service programmatically, the data is taken from the example data you provided as part of the JSON package.

Managing billing plans

Click the **Plans** menu option from the web services Quickstart page. You can also click the plan associated with specific Web service to manage that plan.

- **New** allows you to create a new plan.
- **Add/Remove Plan instance** allows you to "scale out" an existing plan to add capacity.
- **Upgrade/DownGrade** allows you to "scale up" an existing plan to add capacity.
- **Delete** allows you to delete a plan.

Click a plan to view its dashboard. The dashboard gives you snapshot or plan usage over a selected period of time. To select the time period to view, click the **Period** dropdown at the upper right of dashboard.

The plan dashboard provides the following information:

- **Plan Description** displays information about the costs and capacity associated with the plan.
- **Plan Usage** displays the number of transactions and compute hours that have been charged against the plan.
- **Web Services** displays the number of Web services that are using this plan.
- **Top Web Service By Calls** displays the top four Web services that are making calls that are charged against the plan.
- **Top Web Services by Compute Hrs** displays the top four Web services that are using compute resources that are charged against the plan.

Manage Classic Web Services

NOTE

The procedures in this section are relevant to managing Classic web services through the Azure Machine Learning Web Services portal. For information on managing Classic Web services through the Machine Learning Studio and the Azure classic portal, see [Manage an Azure Machine Learning workspace](#).

To manage your Classic Web services:

1. Sign in to the [Microsoft Azure Machine Learning Web Services](#) portal using your Microsoft Azure account - use the account that's associated with the Azure subscription.
2. On the menu, click **Classic Web Services**.

To manage a Classic Web Service, click **Classic Web Services**. From the Classic Web Services page you can:

- Click the web service to view the associated endpoints.
- Delete a web service.

When you manage a Classic Web service, you manage each of the endpoints separately. When you click a web service in the Web Services page, the list of endpoints associated with the service opens.

On the Classic Web Service endpoint page, you can add and delete endpoints on the service. For more information on adding endpoints, see [Creating Endpoints](#).

Click one of the endpoints to open the web service Quickstart page. On the Quickstart page, there are two menu options that allow you to manage your web service:

- **DASHBOARD** - Allows you to view Web service usage.
- **CONFIGURE** - Allows you to add descriptive text, turn error logging on and off, update the key for the storage account associated with the Web service, and enable and disable sample data.

Monitoring how the web service is being used

Click the **DASHBOARD** tab.

From the dashboard, you can view overall usage of your Web service over a period of time. You can select the period to view from the Period dropdown menu in the upper right of the usage charts. The dashboard shows the following information:

- **Requests Over Time** displays a step graph of the number of requests over the selected time period. It can help identify if you are experiencing spikes in usage.
- **Request-Response Requests** displays the total number of Request-Response calls the service has received over the selected time period and how many of them failed.
- **Average Request-Response Compute Time** displays an average of the time needed to execute the received requests.
- **Batch Requests** displays the total number of Batch Requests the service has received over the selected time period and how many of them failed.
- **Average Job Latency** displays an average of the time needed to execute the received requests.
- **Errors** displays the aggregate number of errors that have occurred on calls to the web service.
- **Services Costs** displays the charges for the billing plan associated with the service.

Configuring the web service

Click the **CONFIGURE** menu option.

You can update the following properties:

- **Description** allows you to enter a description for the Web service. Description is a required field.

- **Logging** allows you to enable or disable error logging on the endpoint. For more information on Logging, see [Enable logging for Machine Learning web services](#).
- **Enable Sample data** allows you to provide sample data that you can use to test the Request-Response service. If you created the web service in Machine Learning Studio, the sample data is taken from the data you used to train your model. If you created the service programmatically, the data is taken from the example data you provided as part of the JSON package.

Grant or suspend access to Web services for users in the portal

Using the Azure classic portal, you can allow or deny access to specific users.

Access for users of New Web services

To enable other users to work with your Web services in the Azure Machine Learning Web Services portal, you must add them as co-administrators on your Azure subscription.

Sign in to the [Azure classic portal](#) using your Microsoft Azure account - use the account that's associated with the Azure subscription.

1. In the navigation pane, click **Settings**, then click **Administrators**.
2. At the bottom of the window, click **Add**.
3. In the ADD A CO-ADMINISTRATOR dialog, type the email address of the person you want to add as Co-administrator and then select the subscription that you want the Co-administrator to access.
4. Click **Save**.

Access for users of Classic Web services

To manage a workspace:

Sign in to the [Azure classic portal](#) using your Microsoft Azure account - use the account that's associated with the Azure subscription.

1. In the Microsoft Azure services panel, click **MACHINE LEARNING**.
2. Click the workspace you want to manage.
3. Click the **CONFIGURE** tab.

From the configuration tab, you can suspend access to the Machine Learning workspace by clicking **DENY**. Users will no longer be able to open the workspace in Machine Learning Studio. To restore access, click **ALLOW**.

To specific users:

To manage additional accounts who have access to the workspace in Machine Learning Studio, click **Sign-in to ML Studio** in the **DASHBOARD** tab. This opens the workspace in Machine Learning Studio. From here, click the **SETTINGS** tab and then **USERS**. You can click **INVITE MORE USERS** to give users access to the workspace, or select a user and click **REMOVE**.

NOTE

The **Sign-in to ML Studio** link opens Machine Learning Studio using the Microsoft Account you are currently signed into. The Microsoft Account you used to sign in to the Azure classic portal to create a workspace does not automatically have permission to open that workspace. To open a workspace, you must be signed in to the Microsoft Account that was defined as the owner of the workspace, or you need to receive an invitation from the owner to join the workspace.

Learn how to manage AzureML web services using API Management

1/17/2017 • 11 min to read • [Edit on GitHub](#)

Overview

This guide shows you how to quickly get started using API Management to manage your AzureML web services.

What is Azure API Management?

Azure API Management is an Azure service that lets you manage your REST API endpoints by defining user access, usage throttling, and dashboard monitoring. Click [here](#) for details on Azure API Management. Click [here](#) for a guide on how to get started with Azure API Management. This other guide, which this guide is based on, covers more topics, including notification configurations, tier pricing, response handling, user authentication, creating products, developer subscriptions, and usage dashboarding.

What is AzureML?

AzureML is an Azure service for machine learning that enables you to easily build, deploy, and share advanced analytics solutions. Click [here](#) for details on AzureML.

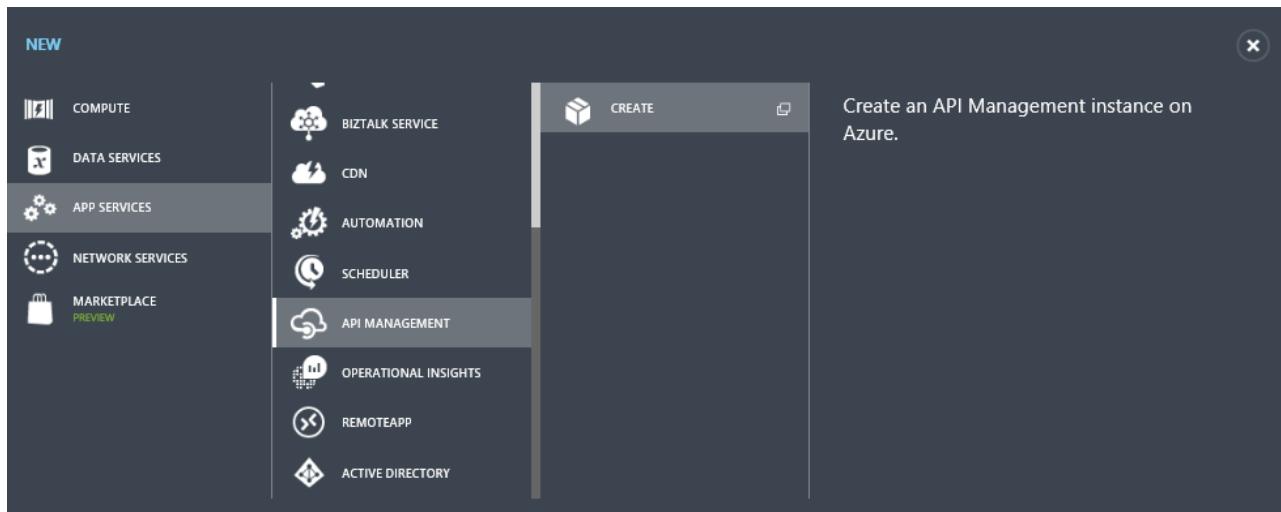
Prerequisites

To complete this guide, you need:

- An Azure account. If you don't have an Azure account, click [here](#) for details on how to create a free trial account.
- An AzureML account. If you don't have an AzureML account, click [here](#) for details on how to create a free trial account.
- The workspace, service, and api_key for an AzureML experiment deployed as a web service. Click [here](#) for details on how to create an AzureML experiment. Click [here](#) for details on how to deploy an AzureML experiment as a web service. Alternately, Appendix A has instructions for how to create and test a simple AzureML experiment and deploy it as a web service.

Create an API Management instance

Below are the steps for using API Management to manage your AzureML web service. First create a service instance. Log in to the [Classic Portal](#) and click **New > App Services > API Management > Create**.



Specify a unique **URL**. This guide uses **demoazureml** – you will need to choose something different. Choose the desired **Subscription** and **Region** for your service instance. After making your selections, click the next button.

NEW API MANAGEMENT SERVICE

Create an API Management Service

URL
 ✓

.azure-api.net

SUBSCRIPTION

REGION

Activation of a new API management service instance may take up to 30 minutes.

(1) 2

Specify a value for the **Organization Name**. This guide uses **demoazureml** – you will need to choose something different. Enter your email address in the **administrator e-mail** field. This email address is used for notifications from the API Management system.

NEW API MANAGEMENT SERVICE

Create an API Management Service

ORGANIZATION NAME
 ?

ADMINISTRATOR E-MAIL
 ?

ADVANCED SETTINGS ?

(1) ← →

Click the check box to create your service instance. *It takes up to thirty minutes for a new service to be created.*

Create the API

Once the service instance is created, the next step is to create the API. An API consists of a set of operations that can be invoked from a client application. API operations are proxied to existing web services. This guide creates APIs that proxy to the existing AzureML RRS and BES web services.

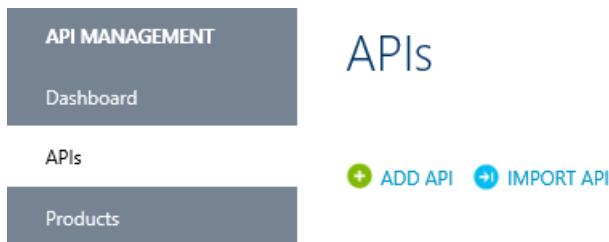
APIs are created and configured from the API publisher portal, which is accessed through the Azure Classic Portal. To reach the publisher portal, select your service instance.



Click **Manage** in the Azure Classic Portal for your API Management service.



Click **APIs** from the **API Management** menu on the left, and then click **Add API**.



Type **AzureML Demo API** as the **Web API name**. Type <https://ussouthcentral.services.azureml.net> as the **Web service URL**. Type **azureml-demo** as the **Web API URL suffix**. Check **HTTPS** as the **Web API URL scheme**. Select **Starter** as **Products**. When finished, click **Save** to create the API.

Add new API

Web API name <input type="text" value="AzureML Demo API"/>	Public name of the API as it would appear on the developer and admin portals.
Web service URL <input type="text" value="https://ussouthcentral.services.azureml.net"/>	A URL of the web service exposing the API. This URL will be used by Azure API Management only, and will not be made public.
Web API URL suffix <input type="text" value="azureml-demo"/>	Last part of the API's public URL. This URL will be used by API consumers for sending requests to the web service.
Web API URL scheme <input type="checkbox"/> HTTP <input checked="" type="checkbox"/> HTTPS	
This is what the URL is going to look like: https://demoazureml.azure-api.net/azureml-demo	
Products (optional) <input type="button" value="Starter"/>	Add this API to one or more existing products.
<input type="button" value="Save"/> <input type="button" value="Cancel"/>	

Add the operations

Click **Add operation** to add operations to this API.

APIs - AzureML Demo API

Summary Settings **Operations** Security Issues Products

Operations

Define service operations to enable service documentation, interactive API console, per operation limits, request/response validation, and operation-level statistics.

 ADD OPERATION

Search operations



The **New operation** window will be displayed and the **Signature** tab will be selected by default.

Add RRS Operation

First create an operation for the AzureML RRS service. Select **POST** as the **HTTP verb**. Type **/workspaces/{workspace}/services/{service}/execute?api-version={apiversion}&details={details}** as the **URL template**. Type **RRS Execute** as the **Display name**.

Signature

HTTP verb*	URL template*
POST	/workspaces/{workspace}/services/{service}/execute?api-version={apiversion}&details={details}
Example: GET Example: customers/{cid}/orders/{oid}/?date={date}	
REQUEST	Rewrite URL template
Parameters	
Body	When specified, rewrite template will be used to make requests to the web service. It can contain all of the template parameters specified in the original template.
RESPONSES	Display name*
 ADD	RRS Execute

Click **Responses > ADD** on the left and select **200 OK**. Click **Save** to save this operation.

Operation - RRS Execute

Code 200 OK  DELETE THIS RESPONSE

Signature	Description
Caching	Enter description
REQUEST	
Parameters	
Body	
RESPONSES	Code 200
	 ADD

+ ADD REPRESENTATION
If operation generates responses in one or more representation formats (e.g. JSON and/or XML), you may describe them in this section.

Save

Add BES Operations

Screenshots are not included for the BES operations as they are very similar to those for adding the RRS operation.

Submit (but not start) a Batch Execution job

Click **add operation** to add the AzureML BES operation to the API. Select **POST** for the **HTTP verb**. Type **/workspaces/{workspace}/services/{service}/jobs?api-version={apiversion}** for the **URL template**. Type **BES Submit** for the **Display name**. Click **Responses > ADD** on the left and select **200 OK**. Click **Save** to save this operation.

Start a Batch Execution job

Click **add operation** to add the AzureML BES operation to the API. Select **POST** for the **HTTP verb**. Type **/workspaces/{workspace}/services/{service}/jobs/{jobid}/start?api-version={apiversion}** for the **URL template**. Type **BES Start** for the **Display name**. Click **Responses > ADD** on the left and select **200 OK**. Click **Save** to save this operation.

Get the status or result of a Batch Execution job

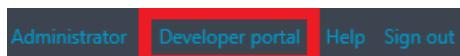
Click **add operation** to add the AzureML BES operation to the API. Select **GET** for the **HTTP verb**. Type **/workspaces/{workspace}/services/{service}/jobs/{jobid}?api-version={apiversion}** for the **URL template**. Type **BES Status** for the **Display name**. Click **Responses > ADD** on the left and select **200 OK**. Click **Save** to save this operation.

Delete a Batch Execution job

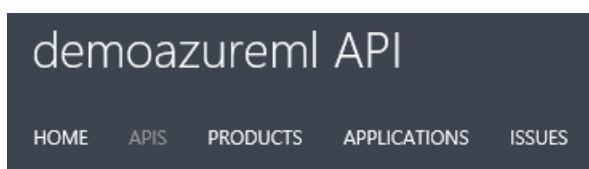
Click **add operation** to add the AzureML BES operation to the API. Select **DELETE** for the **HTTP verb**. Type **/workspaces/{workspace}/services/{service}/jobs/{jobid}?api-version={apiversion}** for the **URL template**. Type **BES Delete** for the **Display name**. Click **Responses > ADD** on the left and select **200 OK**. Click **Save** to save this operation.

Call an operation from the Developer Portal

Operations can be called directly from the Developer portal, which provides a convenient way to view and test the operations of an API. In this guide step you will call the **RRS Execute** method that was added to the **AzureML Demo API**. Click **Developer portal** from the menu at the top right of the Classic Portal.



Click **APIs** from the top menu, and then click **AzureML Demo API** to see the operations available.

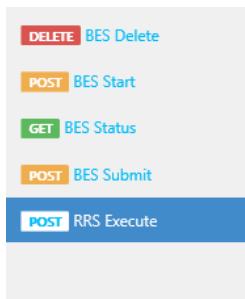


APIs

[AzureML Demo API](#)

[Echo API](#)

Select **RRS Execute** for the operation. Click **Try It**.



AzureML Demo API

RRS Execute

Try it

Request URL

<https://demoazureml.azure-api.net/azureml-demo/workspaces/{workspace}/services/{service}/execute?api-version={apiversion}&details={details}>

For Request parameters, type your **workspace**, **service**, **2.0** for the **apiversion**, and **true** for the **details**. You can find your **workspace** and **service** in the AzureML web service dashboard (see **Test the web service** in Appendix A).

For Request headers, click **Add header** and type **Content-Type** and **application/json**, then click **Add header** and type **Authorization** and **Bearer**. You can find your **api key** in the AzureML web service dashboard (see **Test the web service** in Appendix A).

Type `{"Inputs": {"input1": {"ColumnNames": ["Col2"], "Values": [["This is a good day"]]}}, "GlobalParameters": {}}` for the request body.

AzureML Demo API

</workspaces/{workspace}/services/{service}/execute?api-version={apiversion}&details={details}>

Request parameters

workspace	<input type="text" value="bf4[REDACTED]59;"/>
service	<input type="text" value="c91[REDACTED]12"/>
apiversion	<input type="text" value="2.0"/>
details	<input type="text" value="true"/>

[+ Add parameter](#)

Request headers

Ocp-Apim-Trace	<input type="text" value="true"/>
Ocp-Apim-Subscription-Key	<input type="text" value="*****"/>
Content-Type	<input type="text" value="application/json"/> ✖ Remove header
Authorization	<input type="text" value="Bearer /EHr-[REDACTED]"/> ✖ Remove header

[+ Add header](#)

Request body

```
{"Inputs": {"input1": {"ColumnNames": ["Col2"], "Values": [["This is a good day"]]}}, "GlobalParameters": {}}
```

Authorization

Subscription key	<input type="text" value="Primary-5383..."/> x ▼
------------------	--

Click **Send**.

Send

After an operation is invoked, the developer portal displays the **Requested URL** from the back-end service, the **Response status**, the **Response headers**, and any **Response content**.

Response status

200 OK

Response latency

438 ms

Response headers

1. x-ms-request-id: 3b648c32-9c35-40b6-a309-138396f59e12
2. Ocp-Apim-Trace-Location: https://apimgmtstsfa4wt270nq7f1.blob.core.windows.net/apiinspectorcontainer/k4m0GuwG51EdW1uyP4rCvA2-2?sv=2013-08-15&sr=b&sig=N%2B%2By6qu%2Fi1pDs0WI0X1zwqArAyUFS05aooy2gNPLOFA%3D&se=2015-06-03T19%3A16%3A27Z&sp=r&traceId=5be3edac41342e789b5a6c656b94730
3. Date: Tue, 02 Jun 2015 19:16:27 GMT
4. Content-Length: 428
5. Content-Type: application/json; charset=utf-8

Response content

```
[  
    "This is a good day",  
    "1",  
    "1",  
    "2",  
    "2",  
    "0",  
    "2",  
    "0",  
    "1"  
]
```

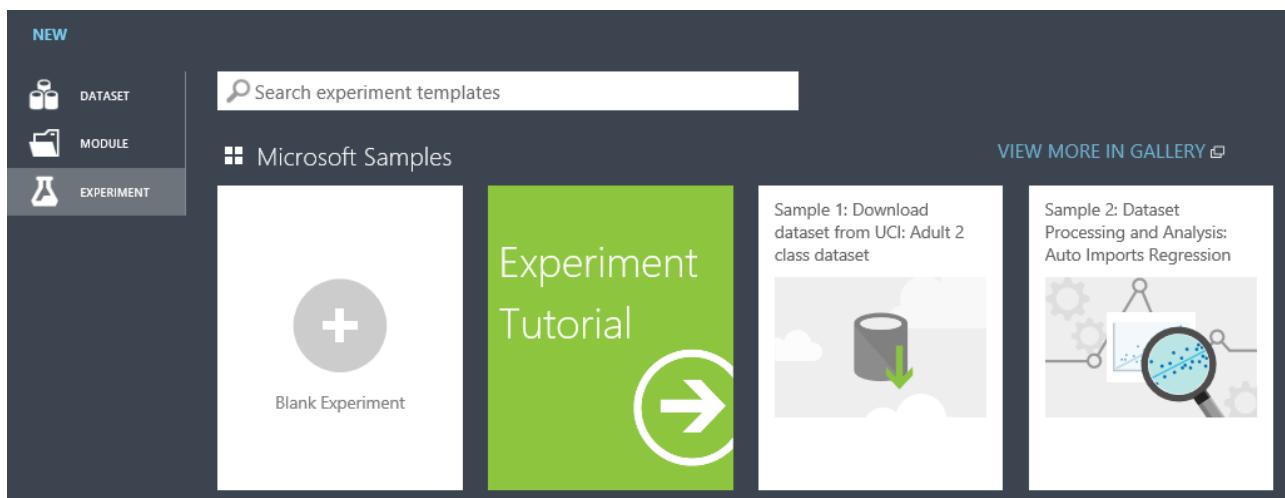
Appendix A - Creating and testing a simple AzureML web service

Creating the experiment

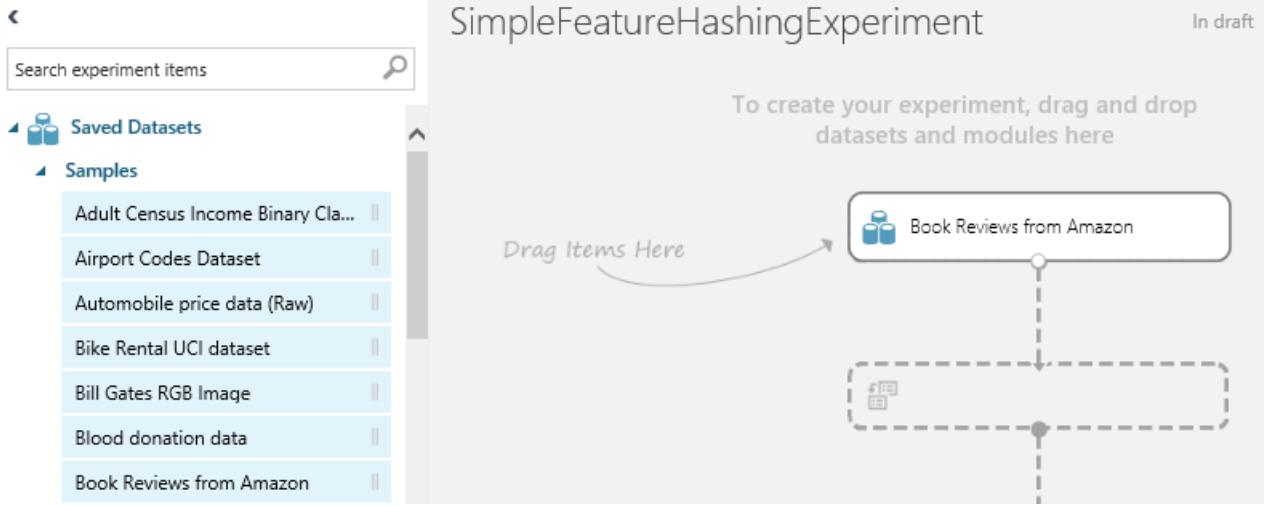
Below are the steps for creating a simple AzureML experiment and deploying it as a web service. The web service takes as input a column of arbitrary text and returns a set of features represented as integers. For example:

TEXT	HASHED TEXT
This is a good day	1 1 2 2 0 2 0 1

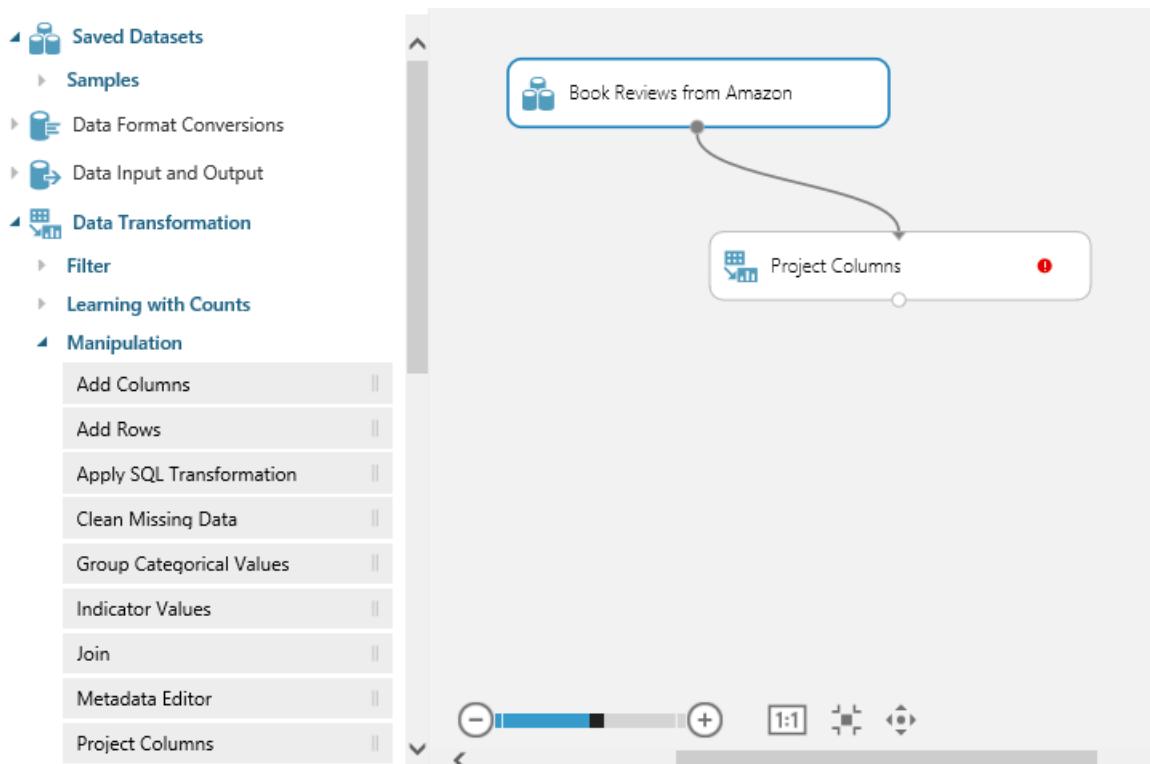
First, using a browser of your choice, navigate to: <https://studio.azureml.net/> and enter your credentials to log in. Next, create a new blank experiment.



Rename it to **SimpleFeatureHashingExperiment**. Expand **Saved Datasets** and drag **Book Reviews from Amazon** onto your experiment.



Expand **Data Transformation** and **Manipulation** and drag **Select Columns in Dataset** onto your experiment. Connect **Book Reviews from Amazon** to **Select Columns in Dataset**.



Click **Select Columns in Dataset** and then click **Launch column selector** and select **Col2**. Click the checkmark to apply these changes.

Select columns ×

Allow duplicates and preserve column order in selection

Begin With ▼

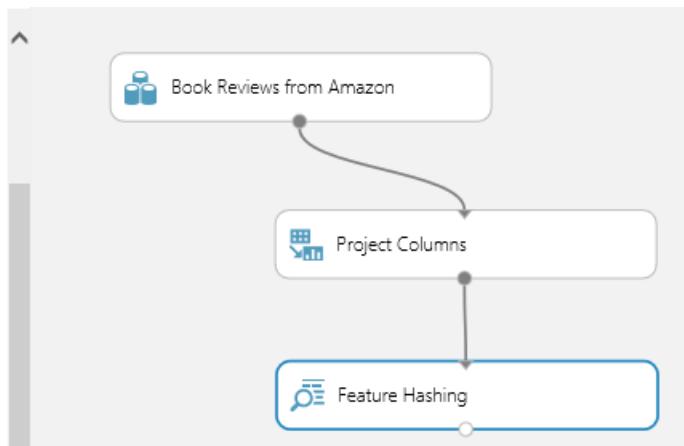
Include ▼ ▼ Col2 X + -

✓

Expand **Text Analytics** and drag **Feature Hashing** onto the experiment. Connect **Select Columns in Dataset** to

Feature Hashing.

- ▶ Data Transformation
- ▶ Feature Selection
- ▶ Machine Learning
- ▶ OpenCV Library Modules
- ▶ Python Language Modules
- ▶ R Language Modules
- ▶ Statistical Functions
- ◀ **Text Analytics**
- Feature Hashing



Type **3** for the **Hashing bitsize**. This will create 8 (23) columns.

Properties

◀ Feature Hashing

Target column(s)

Selected columns:

Column type: String,
Feature

Launch column selector

Hashing bitsize

3

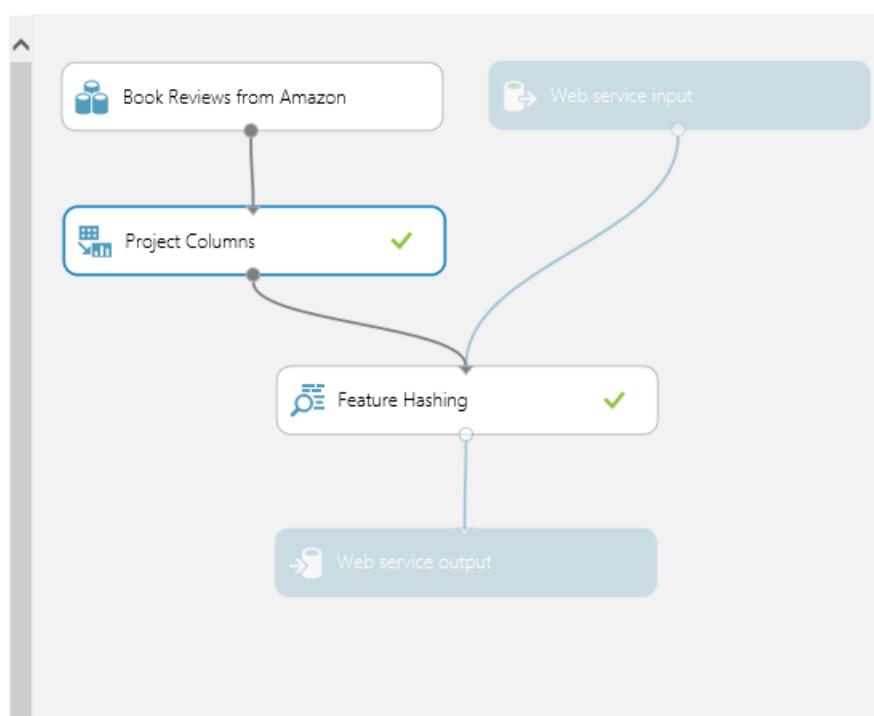
At this point, you may want to click **Run** to test the experiment.



Create a web service

Now create a web service. Expand **Web Service** and drag **Input** onto your experiment. Connect **Input** to **Feature Hashing**. Also drag **output** onto your experiment. Connect **Output** to **Feature Hashing**.

- ◀ Saved Datasets
 - ▶ Samples
- ▶ Data Format Conversions
- ▶ Data Input and Output
- ▶ Data Transformation
- ▶ Feature Selection
- ▶ Machine Learning
- ▶ OpenCV Library Modules
- ▶ Python Language Modules
- ▶ R Language Modules
- ▶ Statistical Functions
- ▶ Text Analytics
- ◀ **Web Service**
 - Input
 - Output



Click **Publish web service**.



Click **Yes** to publish the experiment.



Test the web service

An AzureML web service consists of RSS (request/response service) and BES (batch execution service) endpoints. RSS is for synchronous execution. BES is for asynchronous job execution. To test your web service with the sample Python source below, you may need to download and install the Azure SDK for Python (see: [How to install Python](#)).

You will also need the **workspace**, **service**, and **api_key** of your experiment for the sample source below. You can find the workspace and service by clicking either **Request/Response** or **Batch Execution** for your experiment in the web service dashboard.

Request

Method Request URI

POST https://ussouthcentral.services.azureml.net/workspaces/[REDACTED]/services/[REDACTED] workspace service
api-version=2.0&details=true

You can find the **api_key** by clicking your experiment in the web service dashboard.

DASHBOARD CONFIGURATION

General

Published experiment

[View snapshot](#) [View latest](#)

Description

No description provided for this web service.

API key

/EHr+Icv[REDACTED]vvdell

Default Endpoint

REQUEST/RESPONSE TEST APPS LAST UPDATED

Download Excel Workbook 5/5/2015 10:13:51 AM

BATCH EXECUTION

Test RRS endpoint

Test button

An easy way to test the RRS endpoint is to click **Test** on the web service dashboard.

DASHBOARD CONFIGURATION

General

Published experiment

[View snapshot](#) [View latest](#)

Description

No description provided for this web service.

API key

/EHr+Icv[REDACTED]vvdell

Default Endpoint

REQUEST/RESPONSE TEST APPS LAST UPDATED

Download Excel Workbook 5/5/2015 10:13:51 AM

BATCH EXECUTION

Type **This is a good day** for **col2**. Click the checkmark.



Enter data to predict

COL2

This is a good day



You will see something like

✓ 'SimpleFeatureHashingExperiment' test returned ["This is a good day","1","1","2","2","0","2","0","1"]...

Sample Code

Another way to test your RRS is from your client code. If you click **Request/response** on the dashboard and scroll to the bottom, you will see sample code for C#, Python, and R. You will also see the syntax of the RRS request, including the request URI, headers, and body.

This guide shows a working Python example. You will need to modify it with the **workspace**, **service**, and **api_key** of your experiment.

```
import urllib2
import json
workspace = "<REPLACE WITH YOUR EXPERIMENT'S WEB SERVICE WORKSPACE ID>"
service = "<REPLACE WITH YOUR EXPERIMENT'S WEB SERVICE SERVICE ID>"
api_key = "<REPLACE WITH YOUR EXPERIMENT'S WEB SERVICE API KEY>"
data = {
    "Inputs": {
        "input1": {
            "ColumnNames": ["Col2"],
            "Values": [ [ "This is a good day" ] ]
        },
    },
    "GlobalParameters": { }
}
url = "https://ussouthcentral.services.azureml.net/workspaces/" + workspace + "/services/" + service + "/execute?api-version=2.0&details=true"
headers = {'Content-Type':'application/json', 'Authorization':('Bearer ' + api_key)}
body = str.encode(json.dumps(data))
try:
    req = urllib2.Request(url, body, headers)
    response = urllib2.urlopen(req)
    result = response.read()
    print "result:" + result
except urllib2.HTTPError, error:
    print("The request failed with status code: " + str(error.code))
    print(error.info())
    print(json.loads(error.read()))
```

Test BES endpoint

Click **Batch execution** on the dashboard and scroll to the bottom. You will see sample code for C#, Python, and R. You will also see the syntax of the BES requests to submit a job, start a job, get the status or results of a job, and delete a job.

This guide shows a working Python example. You need to modify it with the **workspace**, **service**, and **api_key** of your experiment. Additionally, you need to modify the **storage account name**, **storage account key**, and **storage container name**. Lastly, you will need to modify the location of the **input file** and the location of the **output file**.

```

import urllib2
import json
import time
from azure.storage import *
workspace = "<REPLACE WITH YOUR WORKSPACE ID>"
service = "<REPLACE WITH YOUR SERVICE ID>"
api_key = "<REPLACE WITH THE API KEY FOR YOUR WEB SERVICE>"
storage_account_name = "<REPLACE WITH YOUR AZURE STORAGE ACCOUNT NAME>"
storage_account_key = "<REPLACE WITH YOUR AZURE STORAGE KEY>"
storage_container_name = "<REPLACE WITH YOUR AZURE STORAGE CONTAINER NAME>"
input_file = "<REPLACE WITH THE LOCATION OF YOUR INPUT FILE>" # Example: C:\\mydata.csv
output_file = "<REPLACE WITH THE LOCATION OF YOUR OUTPUT FILE>" # Example: C:\\myresults.csv
input_blob_name = "myblob.csv"
output_blob_name = "myresultsblob.csv"
def printHttpError(httpError):
    print("The request failed with status code: " + str(httpError.code))
    print(httpError.info())
    print(json.loads(httpError.read()))
    return
def saveBlobToFile(blobUrl, resultsLabel):
    print("Reading the result from " + blobUrl)
    try:
        response = urllib2.urlopen(blobUrl)
    except urllib2.HTTPError, error:
        printHttpError(error)
        return
    with open(output_file, "w+") as f:
        f.write(response.read())
    print(resultsLabel + " have been written to the file " + output_file)
    return
def processResults(result):
    first = True
    results = result["Results"]
    for outputName in results:
        result_blob_location = results[outputName]
        sas_token = result_blob_location["SasBlobToken"]
        base_url = result_blob_location["BaseLocation"]
        relative_url = result_blob_location["RelativeLocation"]
        print("The results for " + outputName + " are available at the following Azure Storage location:")
        print("BaseLocation: " + base_url)
        print("RelativeLocation: " + relative_url)
        print("SasBlobToken: " + sas_token)
        if (first):
            first = False
            url3 = base_url + relative_url + sas_token
            saveBlobToFile(url3, "The results for " + outputName)
    return
def invokeBatchExecutionService():
    url = "https://ussouthcentral.services.azureml.net/workspaces/" + workspace + "/services/" + service + "/jobs"
    blob_service = BlobService(account_name=storage_account_name, account_key=storage_account_key)
    print("Uploading the input to blob storage...")
    data_to_upload = open(input_file, "r").read()
    blob_service.put_blob(storage_container_name, input_blob_name, data_to_upload, x_ms_blob_type="BlockBlob")
    print "Uploaded the input to blob storage"
    input_blob_path = "/" + storage_container_name + "/" + input_blob_name
    connection_string = "DefaultEndpointsProtocol=https;AccountName=" + storage_account_name + ";AccountKey=" + storage_account_key
    payload = {
        "Input": {
            "ConnectionString": connection_string,
            "RelativeLocation": input_blob_path
        },
        "Outputs": {
            "output1": { "ConnectionString": connection_string, "RelativeLocation": "/" + storage_container_name + "/" + output_blob_name },
        },
        "GlobalParameters": {
        }
    }

```

```

body = str.encode(json.dumps(payload))
headers = { "Content-Type": "application/json", "Authorization": ("Bearer " + api_key) }
print("Submitting the job...")
# submit the job
req = urllib2.Request(url + "?api-version=2.0", body, headers)
try:
    response = urllib2.urlopen(req)
except urllib2.HTTPError, error:
    printHttpError(error)
    return
result = response.read()
job_id = result[1:-1] # remove the enclosing double-quotes
print("Job ID: " + job_id)
# start the job
print("Starting the job...")
req = urllib2.Request(url + "/" + job_id + "/start?api-version=2.0", "", headers)
try:
    response = urllib2.urlopen(req)
except urllib2.HTTPError, error:
    printHttpError(error)
    return
url2 = url + "/" + job_id + "?api-version=2.0"

while True:
    print("Checking the job status...")
    # If you are using Python 3+, replace urllib2 with urllib.request in the following code
    req = urllib2.Request(url2, headers = { "Authorization": ("Bearer " + api_key) })
    try:
        response = urllib2.urlopen(req)
    except urllib2.HTTPError, error:
        printHttpError(error)
        return
    result = json.loads(response.read())
    status = result["StatusCode"]
    print "status:" + status
    if(status == 0 or status == "NotStarted"):
        print("Job " + job_id + " not yet started...")
    elif(status == 1 or status == "Running"):
        print("Job " + job_id + " running...")
    elif(status == 2 or status == "Failed"):
        print("Job " + job_id + " failed!")
        print("Error details: " + result["Details"])
        break
    elif(status == 3 or status == "Cancelled"):
        print("Job " + job_id + " cancelled!")
        break
    elif(status == 4 or status == "Finished"):
        print("Job " + job_id + " finished!")
        processResults(result)
        break
    time.sleep(1) # wait one second
return
invokeBatchExecutionService()

```

Scaling a Web service

1/17/2017 • 1 min to read • [Edit on GitHub](#)

NOTE

This topic describes techniques applicable to a Classic Machine Learning Web service.

By default, each published Web service is configured to support 20 concurrent requests and can be as high as 200 concurrent requests. While the Azure classic portal provides a way to set this value, Azure Machine Learning automatically optimizes the setting to provide the best performance for your web service and the portal value is ignored.

If you plan to call the API with a higher load than a Max Concurrent Calls value of 200 will support, you should create multiple endpoints on the same Web service. You can then randomly distribute your load across all of them.

Add new endpoints for same web service

The scaling of a Web service is a common task. Some reasons to scale are to support more than 200 concurrent requests, increase availability through multiple endpoints, or provide separate endpoints for the web service. You can increase the scale by adding additional endpoints for the same Web service through [Azure classic portal](#) or the [Azure Machine Learning Web Service](#) portal.

For more information on adding new endpoints, see [Creating Endpoints](#).

Keep in mind that using a high concurrency count can be detrimental if you're not calling the API with a correspondingly high rate. You might see sporadic timeouts and/or spikes in the latency if you put a relatively low load on an API configured for high load.

The synchronous APIs are typically used in situations where a low latency is desired. Latency here implies the time it takes for the API to complete one request, and doesn't account for any network delays. Let's say you have an API with a 50-ms latency. To fully consume the available capacity with throttle level High and Max Concurrent Calls = 20, you need to call this API $20 * 1000 / 50 = 400$ times per second. Extending this further, a Max Concurrent Calls of 200 allows you to call the API 4000 times per second, assuming a 50-ms latency.

Retrain a Machine Learning Model

1/17/2017 • 3 min to read • [Edit on GitHub](#)

As part of the process of operationalization of machine learning models in Azure Machine Learning, your model is trained and saved. You then use it to create a predictive Web service. The Web service can then be consumed in web sites, dashboards, and mobile apps.

Models you create using Machine Learning are typically not static. As new data becomes available or when the consumer of the API has their own data the model needs to be retrained.

Retraining may occur frequently. With the Programmatic Retraining API feature, you can programmatically retrain the model using the Retraining APIs and update the Web service with the newly trained model.

This document describes the retraining process, and shows you how to use the Retraining APIs.

Why retrain: defining the problem

As part of the machine learning training process, a model is trained using a set of data. Models you create using Machine Learning are typically not static. As new data becomes available or when the consumer of the API has their own data the model needs to be retrained.

In these scenarios, a programmatic API provides a convenient way to allow you or the consumer of your APIs to create a client that can, on a one-time or regular basis, retrain the model using their own data. They can then evaluate the results of retraining, and update the Web service API to use the newly trained model.

NOTE

If you have an existing Training Experiment and New Web service, you may want to check out Retrain an existing Predictive Web service instead of following the walkthrough mentioned in the following section.

End-to-end workflow

The process involves the following components: A Training Experiment and a Predictive Experiment published as a Web service. To enable retraining of a trained model, the Training Experiment must be published as a Web service with the output of a trained model. This enables API access to the model for retraining.

The following steps apply to both New and Classic Web services:

Create the initial Predictive Web service:

- Create a training experiment
- Create a predictive web experiment
- Deploy a predictive web service

Retrain the Web service:

- Update training experiment to allow for retraining
- Deploy the retraining web service
- Use the Batch Execution Service code to retrain the model

For a walkthrough of the preceding steps, see [Retrain Machine Learning models programmatically](#).

If you deployed a Classic Web Service:

- Create a new Endpoint on the Predictive Web service
- Get the PATCH URL and code
- Use the PATCH URL to point the new Endpoint at the retrained model

For a walkthrough of the preceding steps, see [Retrain a Classic Web service](#).

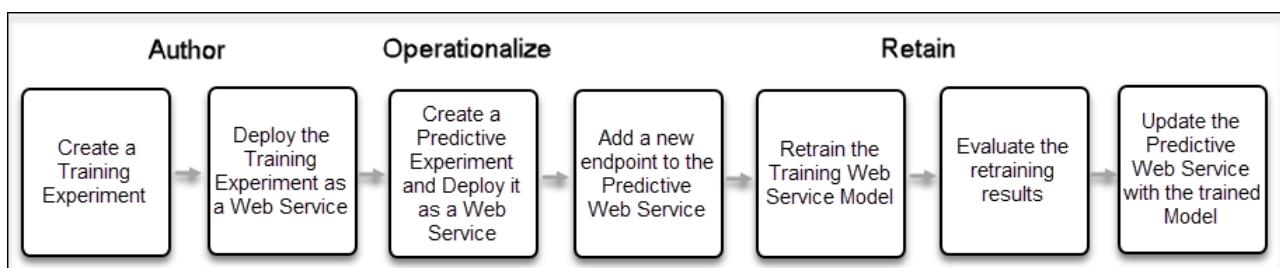
If you run into difficulties retraining a Classic Web service, see [Troubleshooting the retraining of an Azure Machine Learning Classic Web service](#).

If you deployed a New Web service:

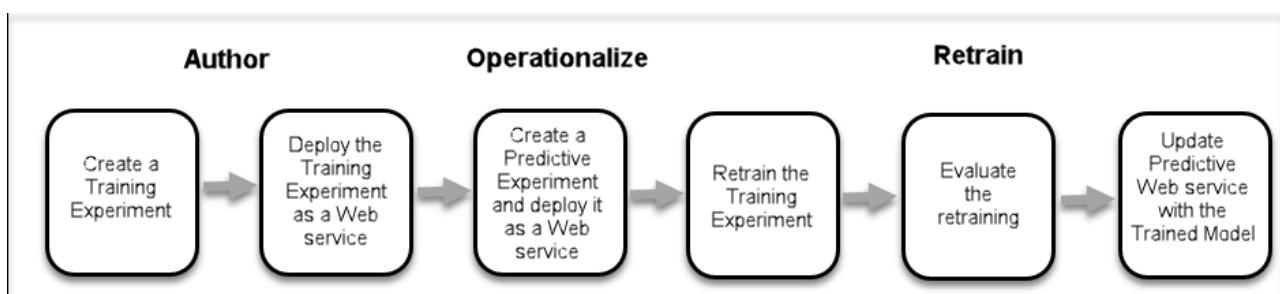
- Sign in to your Azure Resource Manager account
- Get the Web service definition
- Export the Web Service Definition as JSON
- Update the reference to the `fileame` blob in the JSON
- Import the JSON into a Web Service Definition
- Update the Web service with new Web Service Definition

For a walkthrough of the preceding steps, see [Retrain a New Web service using the Machine Learning Management PowerShell cmdlets](#).

The process for setting up retraining for a Classic Web service involves the following steps:



The process for setting up retraining for a New Web service involves the following steps:



Other Resources

- [Retraining and Updating Azure Machine Learning models with Azure Data Factory](#)
- [Create many Machine Learning models and web service endpoints from one experiment using PowerShell](#)
- The [AML Retraining Models Using APIs](#) video shows you how to retrain Machine Learning models created in Azure Machine Learning using the Retraining APIs and PowerShell.

Retrain Machine Learning models programmatically

1/17/2017 • 6 min to read • [Edit on GitHub](#)

In this walkthrough, you will learn how to programmatically retrain an Azure Machine Learning Web Service using C# and the Machine Learning Batch Execution service.

Once you have retrained the model, the following walkthroughs show how to update the model in your predictive web service:

- If you deployed a Classic web service in the Machine Learning Web Services portal, see [Retrain a Classic web service](#).
- If you deployed a New web service, see [Retrain a New web service using the Machine Learning Management cmdlets](#).

For an overview of the retraining process, see [Retrain a Machine Learning Model](#).

If you want to start with your existing New Azure Resource Manager based web service, see [Retrain an existing Predictive web service](#).

Create a training experiment

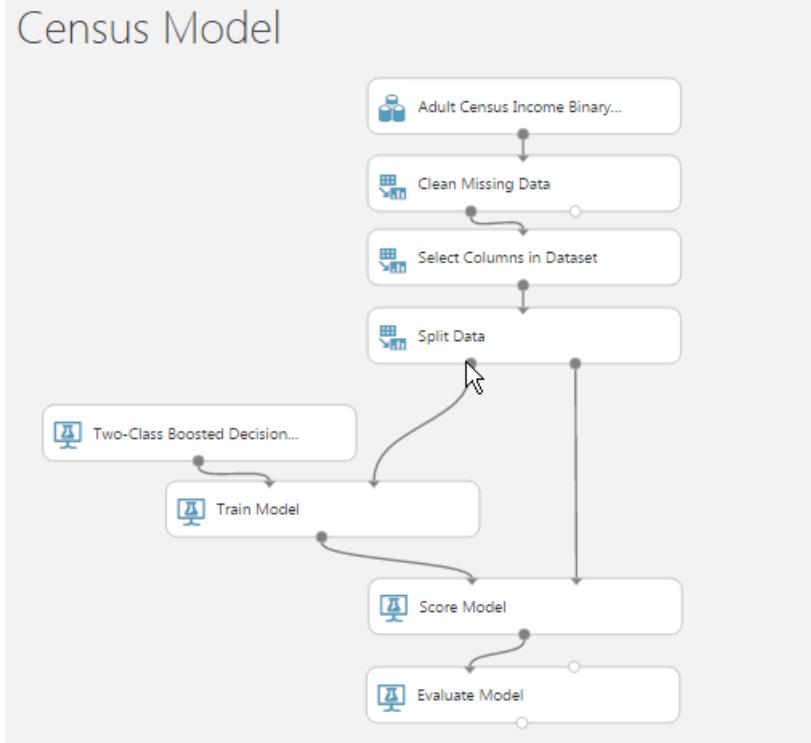
For this example, you will use "Sample 5: Train, Test, Evaluate for Binary Classification: Adult Dataset" from the Microsoft Azure Machine Learning samples.

To create the experiment:

1. Sign into Microsoft Azure Machine Learning Studio.
2. On the bottom right corner of the dashboard, click **New**.
3. From the Microsoft Samples, select Sample 5.
4. To rename the experiment, at the top of the experiment canvas, select the experiment name "Sample 5: Train, Test, Evaluate for Binary Classification: Adult Dataset".
5. Type Census Model.
6. At the bottom of the experiment canvas, click **Run**.
7. Click **Set Up web service** and select **Retraining web service**.

The following shows the initial experiment.

Census Model



Create a predictive experiment and publish as a web service

Next you create a Predictive Experiment.

1. At the bottom of the experiment canvas, click **Set Up Web Service** and select **Predictive Web Service**. This saves the model as a Trained Model and adds web service Input and Output modules.
2. Click **Run**.
3. After the experiment has finished running, click **Deploy Web Service [Classic]** or **Deploy Web Service [New]**.

Deploy the training experiment as a Training web service

To retrain the trained model, you must deploy the training experiment that you created as a Retraining web service. This web service needs a *Web Service Output* module connected to the *Train Model* module, to be able to produce new trained models.

1. To return to the training experiment, click the Experiments icon in the left pane, then click the experiment named Census Model.
2. In the Search Experiment Items search box, type web service.
3. Drag a *Web Service Input* module onto the experiment canvas and connect its output to the *Clean Missing Data* module. This ensures that your retraining data is processed the same way as your original training data.
4. Drag two *web service Output* modules onto the experiment canvas. Connect the output of the *Train Model* module to one and the output of the *Evaluate Model* module to the other. The web service output for **Train Model** gives us the new trained model. The output attached to **Evaluate Model** returns that module's output, which is the performance results.
5. Click **Run**.

Next you must deploy the training experiment as a web service that produces a trained model and model evaluation results. To accomplish this, your next set of actions are dependent on whether you are working with a Classic web service or a New web service.

Classic web service

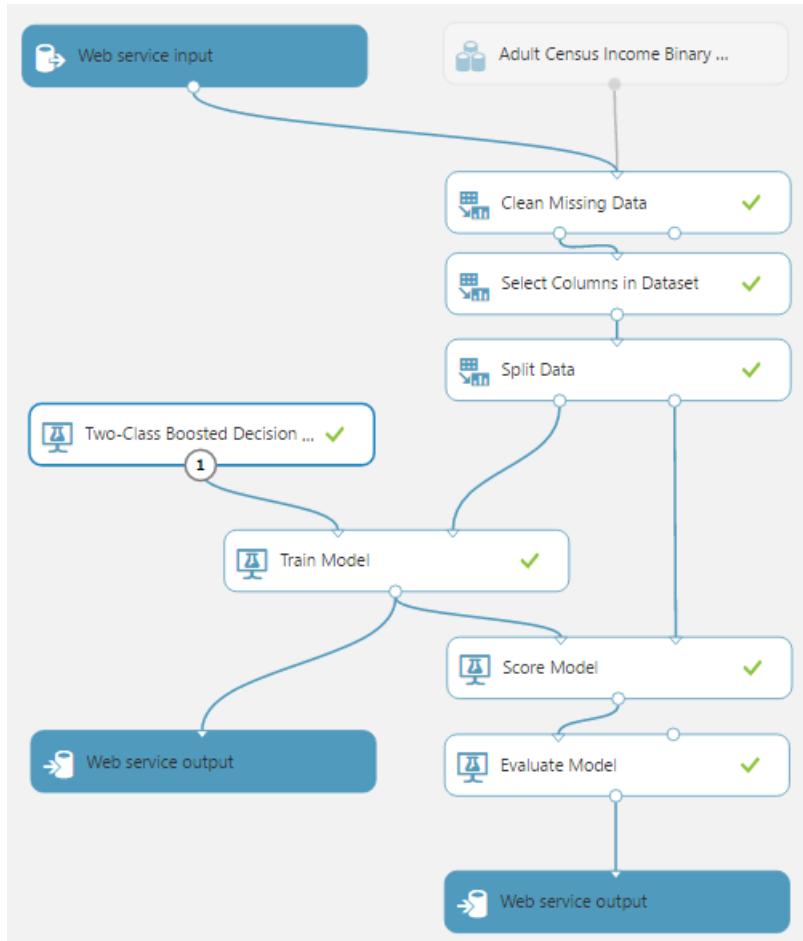
At the bottom of the experiment canvas, click **Set Up Web Service** and select **Deploy Web Service [Classic]**.

The Web Service **Dashboard** is displayed with the API Key and the API help page for Batch Execution. Only the Batch Execution method can be used for creating Trained Models.

New web service

At the bottom of the experiment canvas, click **Set Up Web Service** and select **Deploy Web Service [New]**. The Web Service Azure Machine Learning Web Services portal opens to the Deploy web service page. Type a name for your web service and choose a payment plan, then click **Deploy**. Only the Batch Execution method can be used for creating Trained Models

In either case, after experiment has completed running, the resulting workflow should look as follows:



Retrain the model with new data using BES

For this example, you are using C# to create the retraining application. You can also use the Python or R sample code to accomplish this task.

To call the Retraining APIs:

1. Create a C# Console Application in Visual Studio (New->Project->Windows Desktop->Console Application).
2. Sign in to the Machine Learning Web Service portal.
3. If you are working with a Classic web service, click **Classic Web Services**.
 - a. Click the web service you are working with.
 - b. Click the default endpoint.
 - c. Click **Consume**.
 - d. At the bottom of the **Consume** page, in the **Sample Code** section, click **Batch**.
 - e. Continue to step 5 of this procedure.
4. If you are working with a New web service, click **Web Services**.
 - a. Click the web service you are working with.

- b. Click **Consume**.
 - c. At the bottom of the Consume page, in the **Sample Code** section, click **Batch**.
5. Copy the sample C# code for batch execution and paste it into the Program.cs file, making sure the namespace remains intact.

Add the Nuget package Microsoft.AspNet.WebApi.Client as specified in the comments. To add the reference to Microsoft.WindowsAzure.Storage.dll, you might first need to install the client library for Microsoft Azure storage services. For more information, see [Windows Storage Services](#).

Update the apikey declaration

Locate the **apikey** declaration.

```
const string apiKey = "abc123"; // Replace this with the API key for the web service
```

In the **Basic consumption info** section of the **Consume** page, locate the primary key and copy it to the **apikey** declaration.

Update the Azure Storage information

The BES sample code uploads a file from a local drive (For example "C:\temp\CensusInput.csv") to Azure Storage, processes it, and writes the results back to Azure Storage.

To accomplish this task, you must retrieve the Storage account name, key, and container information for your Storage account from the classic Azure portal and the update corresponding values in the code.

1. Sign in to the classic Azure portal.
2. In the left navigation column, click **Storage**.
3. From the list of storage accounts, select one to store the retrained model.
4. At the bottom of the page, click **Manage Access Keys**.
5. Copy and save the **Primary Access Key** and close the dialog.
6. At the top of the page, click **Containers**.
7. Select an existing container or create a new one and save the name.

Locate the *StorageAccountName*, *StorageAccountKey*, and *StorageContainerName* declarations and update the values you saved from the Azure portal.

```
const string StorageAccountName = "mystorageacct"; // Replace this with your Azure Storage Account name
const string StorageAccountKey = "a_storage_account_key"; // Replace this with your Azure Storage Key
const string StorageContainerName = "mycontainer"; // Replace this with your Azure Storage Container name
```

You also must ensure the input file is available at the location you specify in the code.

Specify the output location

When specifying the output location in the Request Payload, the extension of the file specified in *RelativeLocation* must be specified as *ilearner*.

See the following example:

```

Outputs = new Dictionary<string, AzureBlobDataReference>()
{
    "output1",
    new AzureBlobDataReference()
    {
        ConnectionString = storageConnectionString,
        RelativeLocation = string.Format("{0}/output1/results.learner", StorageContainerName)/*Replace this with the location you would like to
use for your output file, and valid file extension (usually .csv for scoring results, or .learner for trained models)*/
    }
},

```

NOTE

The names of your output locations may be different from the ones in this walkthrough based on the order in which you added the web service output modules. Since you set up this training experiment with two outputs, the results include storage location information for both of them.

```

Running...
Finished!
The result 'output1' is available at the following Azure Storage location:
BaseLocation: https://esintussouthsus.blob.core.windows.net/
RelativeLocation: experimentoutput/9fcefacd-b375-45ab-bcc6-2bf69c491b0d/9fcefacd
-b375-45ab-bcc6-2bf69c491b0d.learner
SasBlobToken: ?sv=2013-08-15&sr=c&sig=%2BMREp%2Fo1%2Bbty6MpQfm9HzsZ65HQTgd4DHpwh
WoJ1UdI%3D&st=2015-02-03T18%3A41%3A13Z&se=2015-02-04T18%3A46%3A13Z&sp=r1

The result 'output2' is available at the following Azure Storage location:
BaseLocation: https://esintussouthsus.blob.core.windows.net/
RelativeLocation: experimentoutput/1cea1d59-fa6a-45d5-83f5-b3ea60a8475b/1cea1d59
-fa6a-45d5-83f5-b3ea60a8475b.csv
SasBlobToken: ?sv=2013-08-15&sr=c&sig=%2BMREp%2Fo1%2Bbty6MpQfm9HzsZ65HQTgd4DHpwh
WoJ1UdI%3D&st=2015-02-03T18%3A41%3A13Z&se=2015-02-04T18%3A46%3A13Z&sp=r1

```

Diagram 4: Retraining output.

Evaluate the Retraining Results

When you run the application, the output includes the URL and SAS token necessary to access the evaluation results.

You can see the performance results of the retrained model by combining the *BaseLocation*, *RelativeLocation*, and *SasBlobToken* from the output results for *output2* (as shown in the preceding retraining output image) and pasting the complete URL in the browser address bar.

Examine the results to determine whether the newly trained model performs well enough to replace the existing one.

Copy the *BaseLocation*, *RelativeLocation*, and *SasBlobToken* from the output results, you will use them during the retraining process.

Next steps

If you deployed the predictive web service by clicking **Deploy Web Service [Classic]**, see [Retrain a Classic web service](#).

If you deployed the predictive web service by clicking **Deploy Web Service [New]**, see [Retrain a New web service using the Machine Learning Management cmdlets](#).

Retrain a Classic web service

1/17/2017 • 4 min to read • [Edit on GitHub](#)

The Predictive Web Service you deployed is the default scoring endpoint. Default endpoints are kept in sync with the original training and scoring experiments, and therefore the trained model for the default endpoint cannot be replaced. To retrain the web service, you must add a new endpoint to the web service.

Prerequisites

You must have set up a training experiment and a predictive experiment as shown in [Retrain Machine Learning models programmatically](#).

IMPORTANT

The predictive experiment must be deployed as a Classic machine learning web service.

For additional information on Deploying web services, see [Deploy an Azure Machine Learning web service](#).

Add a new Endpoint

The Predictive Web Service that you deployed contains a default scoring endpoint that is kept in sync with the original training and scoring experiments trained model. To update your web service to with a new trained model, you must create a new scoring endpoint.

To create a new scoring endpoint, on the Predictive Web Service that can be updated with the trained model:

NOTE

Be sure you are adding the endpoint to the Predictive Web Service, not the Training Web Service. If you have correctly deployed both a Training and a Predictive Web Service, you should see two separate web services listed. The Predictive Web Service should end with "[predictive exp.]".

There are three ways in which you can add a new end point to a web service:

1. Programmatically
2. Use the Microsoft Azure Web Services portal
3. Use the Azure classic portal

Programmatically add an endpoint

You can add scoring endpoints using the sample code provided in this [github repository](#).

Use the Microsoft Azure Web Services portal to add an endpoint

1. In Machine Learning Studio, on the left navigation column, click Web Services.
2. At the bottom of the web service dashboard, click **Manage endpoints preview**.
3. Click **Add**.
4. Type a name and description for the new endpoint. Select the logging level and whether sample data is enabled.
For more information on logging, see [Enable logging for Machine Learning web services](#).

Use the Azure classic portal to add an endpoint

1. Sign in to the [classic Azure portal](#).

2. In the left menu, click **Machine Learning**.
3. Under Name, click your workspace and then click **Web Services**.
4. Under Name, click **Census Model [predictive exp.]**.
5. At the bottom of the page, click **Add Endpoint**. For more information on adding endpoints, see [Creating Endpoints](#).

Update the added endpoint's Trained Model

To complete the retraining process, you must update the trained model of the new endpoint that you added.

- If you added the new endpoint using the classic Azure portal, you can click the new endpoint's name in the portal, then the **UpdateResource** link to get the URL you would need to update the endpoint's model.
- If you added the endpoint using the sample code, this includes location of the help URL identified by the *HelpLocationURL* value in the output.

To retrieve the path URL:

1. Copy and paste the URL into your browser.
2. Click the Update Resource link.
3. Copy the POST URL of the PATCH request. For example:

PATCH URL:

<https://management.azureml.net/workspaces/00bf70534500b34rebfa1843d6/webservices/af3er32ad393852f9b30ac9a35b/endpoints/newendpoint2>

You can now use the trained model to update the scoring endpoint that you created previously.

The following sample code shows you how to use the *BaseLocation*, *RelativeLocation*, *SasBlobToken*, and PATCH URL to update the endpoint.

```

private async Task OverwriteModel()
{
    var resourceLocations = new
    {
        Resources = new[]
        {
            new
            {
                Name = "Census Model [trained model]",
                Location = new AzureBlobDataReference()
                {
                    BaseLocation = "https://esintussouthsus.blob.core.windows.net/",
                    RelativeLocation = "your endpoint relative location", //from the output, for example: "experimentoutput/8946abfd-79d6-4438-89a9-3e5d109183/8946abfd-79d6-4438-89a9-3e5d109183.ilearner"
                    SasBlobToken = "your endpoint SAS blob token" //from the output, for example: "?sv=2013-08-15&sr=c&sig=37lTTfngRwxCc94%3D&st=2015-01-30T22%3A53%3A06Z&se=2015-01-31T22%3A58%3A06Z&sp=r"
                }
            }
        };
    };

    using (var client = new HttpClient())
    {
        client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", apiKey);

        using (var request = new HttpRequestMessage(new HttpMethod("PATCH"), endpointUrl))
        {
            request.Content = new StringContent(JsonConvert.SerializeObject(resourceLocations), System.Text.Encoding.UTF8, "application/json");
            HttpResponseMessage response = await client.SendAsync(request);

            if(!response.IsSuccessStatusCode)
            {
                await WriteFailedResponse(response);
            }

            // Do what you want with a successful response here.
        }
    }
}

```

The *apiKey* and the *endpointUrl* for the call can be obtained from endpoint dashboard.

The value of the *Name* parameter in *Resources* should match the Resource Name of the saved Trained Model in the predictive experiment. To get the Resource Name:

1. Sign in to the [classic Azure portal](#).
2. In the left menu, click **Machine Learning**.
3. Under Name, click your workspace and then click **Web Services**.
4. Under Name, click **Census Model [predictive exp.]**.
5. Click the new endpoint you added.
6. On the endpoint dashboard, click **Update Resource**.
7. On the Update Resource API Documentation page for the web service, you can find the **Resource Name** under **Updatable Resources**.

If your SAS token expires before you finish updating the endpoint, you must perform a GET with the Job Id to obtain a fresh token.

When the code has successfully run, the new endpoint should start using the retrained model in approximately 30 seconds.

Summary

Using the Retraining APIs, you can update the trained model of a predictive Web Service enabling scenarios such as:

- Periodic model retraining with new data.
- Distribution of a model to customers with the goal of letting them retrain the model using their own data.

Next steps

[Troubleshooting the retraining of an Azure Machine Learning classic web service](#)

Retrain a New web service using the Machine Learning Management PowerShell cmdlets

1/17/2017 • 3 min to read • [Edit on GitHub](#)

When you retrain a New web service, you update the predictive web service definition to reference the new trained model.

Prerequisites

You must have set up a training experiment and a predictive experiment as shown in [Retrain Machine Learning models programmatically](#).

IMPORTANT

The predictive experiment must be deployed as an Azure Resource Manager (New) based machine learning web service.

For additional information on Deploying web services, see [Deploy an Azure Machine Learning web service](#).

This process requires that you have installed the Azure Machine Learning Cmdlets. For information installing the Machine Learning cmdlets, see the [Azure Machine Learning Cmdlets](#) reference on MSDN.

Copied the following information from the retraining output:

- BaseLocation
- RelativeLocation

The steps you take are:

1. Sign in to your Azure Resource Manager account.
2. Get the web service definition
3. Export the Web Service Definition as JSON
4. Update the reference to the ilearner blob in the JSON.
5. Import the JSON into a Web Service Definition
6. Update the web service with new Web Service Definition

Sign in to your Azure Resource Manager account

You must first sign in to your Azure account from within the PowerShell environment using the [Add-AzureRmAccount](#) cmdlet.

Get the Web Service Definition

Next, get the Web Service by calling the [Get-AzureRmMIWebService](#) cmdlet. The Web Service Definition is an internal representation of the trained model of the web service and is not directly modifiable. Make sure that you are retrieving the Web Service Definition for your Predictive experiment and not your training experiment.

```
$wsd = Get-AzureRmMIWebService -Name 'RetrainSamplePre.2016.8.17.0.3.51.237' -ResourceGroupName 'Default-MachineLearning-SouthCentralUS'
```

To determine the resource group name of an existing web service, run the [Get-AzureRmMIWebService](#) cmdlet

without any parameters to display the web services in your subscription. Locate the web service, and then look at its web service ID. The name of the resource group is the fourth element in the ID, just after the *resourceGroups* element. In the following example, the resource group name is Default-MachineLearning-SouthCentralUS.

```
Properties : Microsoft.Azure.Management.MachineLearning.WebServices.Models.WebServicePropertiesForGraph
Id : /subscriptions/<subscription ID>/resourceGroups/Default-MachineLearning-
SouthCentralUS/providers/Microsoft.MachineLearning/webServices/RetrainSamplePre.2016.8.17.0.3.51.237
Name : RetrainSamplePre.2016.8.17.0.3.51.237
Location : South Central US
Type : Microsoft.MachineLearning/webServices
Tags : {}
```

Alternatively, to determine the resource group name of an existing web service, log on to the Microsoft Azure Machine Learning Web Services portal. Select the web service. The resource group name is the fifth element of the URL of the web service, just after the *resourceGroups* element. In the following example, the resource group name is Default-MachineLearning-SouthCentralUS.

```
https://services.azureml.net/subscriptions/<subscription ID>/resourceGroups/Default-MachineLearning-
SouthCentralUS/providers/Microsoft.MachineLearning/webServices/RetrainSamplePre.2016.8.17.0.3.51.237
```

Export the Web Service Definition as JSON

To modify the definition to the trained model to use the newly Trained Model, you must first use the [Export-AzureRmMIWebService](#) cmdlet to export it to a JSON format file.

```
Export-AzureRmMIWebService -WebService $wsd -OutputFile "C:\temp\mlservice_export.json"
```

Update the reference to the ilearner blob in the JSON.

In the assets, locate the [trained model], update the *uri* value in the *locationInfo* node with the URI of the ilearner blob. The URI is generated by combining the *BaseLocation* and the *RelativeLocation* from the output of the BES retraining call. This updates the path to reference the new trained model.

```
"asset3": {
  "name": "Retrain Sample [trained model]",
  "type": "Resource",
  "locationInfo": {
    "uri": "https://mltestaccount.blob.core.windows.net/azuremlassetscontainer/baca7bca650f46218633552c0bcbba0e.ilearner"
  },
  "outputPorts": {
    "Results dataset": {
      "type": "Dataset"
    }
  }
},
```

Import the JSON into a Web Service Definition

You must use the [Import-AzureRmMIWebService](#) cmdlet to convert the modified JSON file back into a Web Service Definition that you can use to update the Web Service Definition.

```
$wsd = Import-AzureRmMIWebService -InputFile "C:\temp\mlservice_export.json"
```

Update the web service with new Web Service Definition

Finally, you use [Update-AzureRmMIWebService](#) cmdlet to update the Web Service Definition.

```
Update-AzureRmMIWebService -Name 'RetrainSamplePre.2016.8.17.0.3.51.237' -ResourceGroupName 'Default-MachineLearning-SouthCentralUS' -  
ServiceUpdates $wsd
```

Summary

Using the Machine Learning PowerShell management cmdlets, you can update the trained model of a predictive Web Service enabling scenarios such as:

- Periodic model retraining with new data.
- Distribution of a model to customers with the goal of letting them retrain the model using their own data.

Retrain an existing predictive web service

1/17/2017 • 7 min to read • [Edit on GitHub](#)

This document describes the retraining process for the following scenario:

- You have a training experiment and a predictive experiment that you have deployed as an operationalized web service.
- You have new data that you want your predictive web service to use to perform its scoring.

Starting with your existing web service and experiments, you need to follow these steps:

1. Update the model.
 - a. Modify your training experiment to allow for web service inputs and outputs.
 - b. Deploy the training experiment as a retraining web service.
 - c. Use the training experiment's Batch Execution Service (BES) to retrain the model.
2. Use the Azure Machine Learning PowerShell cmdlets to update the predictive experiment.
 - a. Sign in to your Azure Resource Manager account.
 - b. Get the web service definition.
 - c. Export the web service definition as JSON.
 - d. Update the reference to the ilearner blob in the JSON.
 - e. Import the JSON into a web service definition.
 - f. Update the web service with a new web service definition.

Deploy the training experiment

To deploy the training experiment as a retraining web service, you must add web service inputs and outputs to the model. By connecting a *Web Service Output* module to the experiment's *Train Model* module, you enable the training experiment to produce a new trained model that you can use in your predictive experiment. If you have an *Evaluate Model* module, you can also attach web service output to get the evaluation results as output.

To update your training experiment:

1. Connect a *Web Service Input* module to your data input (for example, a *Clean Missing Data* module). Typically, you want to ensure that your input data is processed in the same way as your original training data.
2. Connect a *Web Service Output* module to the output of your *Train Model* module.
3. If you have an *Evaluate Model* module and you want to output the evaluation results, connect a *Web Service Output* module to the output of your *Evaluate Model* module.

Run your experiment.

Next, you must deploy the training experiment as a web service that produces a trained model and model evaluation results.

At the bottom of the experiment canvas, click **Set Up Web Service**, and then select **Deploy Web Service [New]**. The Azure Machine Learning Web Services portal opens to the **Deploy Web Service** page. Type a name for your web service, choose a payment plan, and then click **Deploy**. You can only use the Batch Execution method for creating trained models.

Retrain the model with new data by using BES

For this example, we're using C# to create the retraining application. You can also use Python or R sample code to

accomplish this task.

To call the retraining APIs:

1. Create a C# console application in Visual Studio (**New > Project > Windows Desktop > Console Application**).
2. Sign in to the Machine Learning Web Services portal.
3. Click the web service that you're working with.
4. Click **Consume**.
5. At the bottom of the **Consume** page, in the **Sample Code** section, click **Batch**.
6. Copy the sample C# code for batch execution and paste it into the Program.cs file. Make sure that the namespace remains intact.

Add the NuGet package Microsoft.AspNet.WebApi.Client, as specified in the comments. To add the reference to Microsoft.WindowsAzure.Storage.dll, you might first need to install the [client library for Azure Storage services](#).

The following screenshot shows the **Consume** page in the Azure Machine Learning Web Services portal.

The screenshot shows the 'Consume' tab selected in the top navigation bar of the Azure Machine Learning Web Services portal. The main content area displays the 'Retraining Example [Predictive Exp.]' experiment details. It includes sections for 'Web service consumption options' (Excel 2013 or later and Excel 2010 or earlier), 'Basic consumption info' (Primary Key and Secondary Key fields, both circled in red), 'Request-Response' (API endpoint URL), 'Batch Requests' (API endpoint URL), and 'Sample Code' (Batch tab selected, showing C# code). A large red circle highlights the 'Batch' tab under 'Sample Code'.

Update the apikey declaration

Locate the **apikey** declaration:

```
const string apiKey = "abc123"; // Replace this with the API key for the web service
```

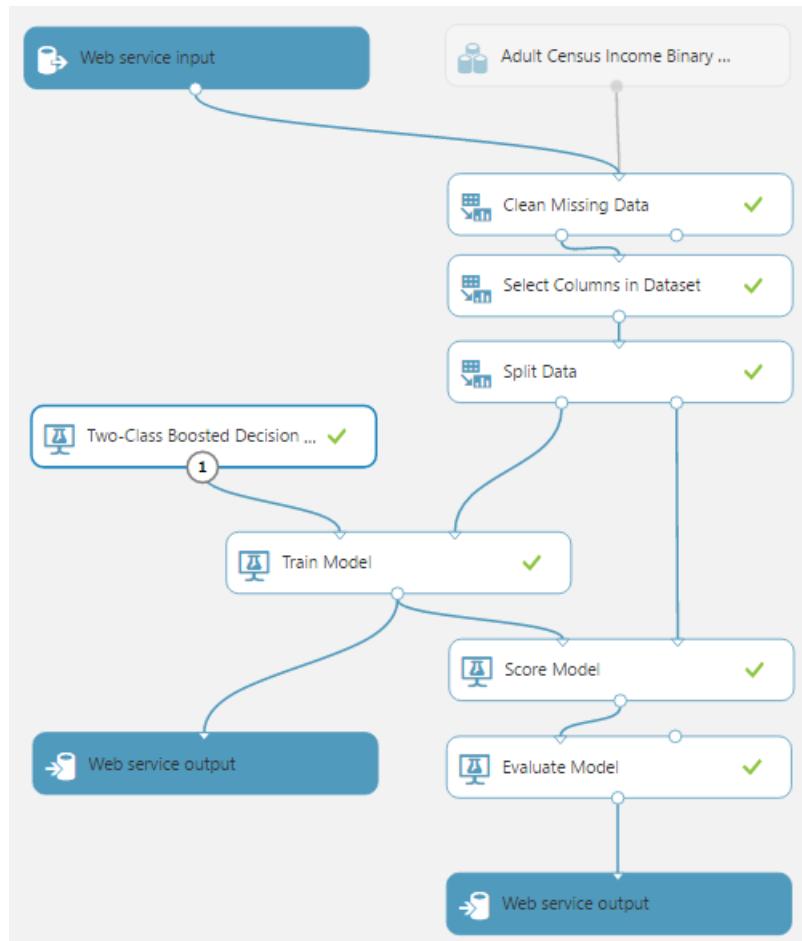
In the **Basic consumption info** section of the **Consume** page, locate the primary key and copy it to the **apikey**

declaration.

Update the Azure Storage information

The BES sample code uploads a file from a local drive (for example, "C:\temp\CensusIpnput.csv") to Azure Storage, processes it, and writes the results back to Azure Storage.

To update the Azure Storage information, you must retrieve the storage account name, key, and container information for your storage account from the Azure classic portal, and then update the corresponding values in the code. After running your experiment, the resulting workflow should be similar to the following:



ng values in the code.

1. Sign in to the Azure classic portal.
2. In the left navigation column, click **Storage**.
3. From the list of storage accounts, select one to store the retrained model.
4. At the bottom of the page, click **Manage Access Keys**.
5. Copy and save the **Primary Access Key** and close the dialog.
6. At the top of the page, click **Containers**.
7. Select an existing container, or create a new one and save the name.

Locate the *StorageAccountName*, *StorageAccountKey*, and *StorageContainerName* declarations, and update the values that you saved from the classic portal.

```
const string StorageAccountName = "mystorageacct"; // Replace this with your Azure storage account name
const string StorageAccountKey = "a_storage_account_key"; // Replace this with your Azure Storage key
const string StorageContainerName = "mycontainer"; // Replace this with your Azure Storage container name
```

You also must ensure that the input file is available at the location that you specify in the code.

Specify the output location

When you specify the output location in the Request Payload, the extension of the file that is specified in

RelativeLocation must be specified as `.ilearners`. See the following example:

```
Outputs = new Dictionary<string, AzureBlobDataReference>()
{
    "output1",
    new AzureBlobDataReference()
    {
        ConnectionString = storageConnectionString,
        RelativeLocation = string.Format("{0}/output1/results.ilearners", StorageContainerName) /*Replace this with the location you want to use for
your output file and a valid file extension (usually .csv for scoring results or .ilearners for trained models)*/
    }
},
```

The following is an example of retraining output:

```
Running...
Finished!
The result 'output1' is available at the following Azure Storage location:
BaseLocation: https://esintussouthsus.blob.core.windows.net/
RelativeLocation: experimentoutput/9fcefacd-b375-45ab-bcc6-2bf69c491b0d/9fcefacd
-b375-45ab-bcc6-2bf69c491b0d.ilearners
SasBlobToken: ?sv=2013-08-15&sr=c&sig=%2BMREpx2Fo1x2Bbty6MpQfm9HzsZ65HQTygd4DHpwH
WoJlUdl%3D&st=2015-02-03T18%3A41%3A13Z&se=2015-02-04T18%3A46%3A13Z&sp=r1

The result 'output2' is available at the following Azure Storage location:
BaseLocation: https://esintussouthsus.blob.core.windows.net/
RelativeLocation: experimentoutput/1cea1d59-fa6a-45d5-83f5-b3ea60a8475b/1cea1d59
-faba-45d5-83f5-b3ea60a8475b.csv
SasBlobToken: ?sv=2013-08-15&sr=c&sig=%2BMREpx2Fo1x2Bbty6MpQfm9HzsZ65HQTygd4DHpwH
WoJlUdl%3D&st=2015-02-03T18%3A41%3A13Z&se=2015-02-04T18%3A46%3A13Z&sp=r1
```

Evaluate the retraining results

When you run the application, the output includes the URL and shared access signatures token that are necessary to access the evaluation results.

You can see the performance results of the retrained model by combining the *BaseLocation*, *RelativeLocation*, and *SasBlobToken* from the output results for *output2* (as shown in the preceding retraining output image) and pasting the complete URL into the browser address bar.

Examine the results to determine whether the newly trained model performs well enough to replace the existing one.

Copy the *BaseLocation*, *RelativeLocation*, and *SasBlobToken* from the output results.

Retrain the web service

When you retrain a new web service, you update the predictive web service definition to reference the new trained model. The web service definition is an internal representation of the trained model of the web service and is not directly modifiable. Make sure that you are retrieving the web service definition for your predictive experiment and not your training experiment.

Sign in to Azure Resource Manager

You must first sign in to your Azure account from within the PowerShell environment by using the [Add-AzureRmAccount](#) cmdlet.

Get the Web Service Definition object

Next, get the Web Service Definition object by calling the [Get-AzureRmWebService](#) cmdlet.

```
$wsd = Get-AzureRmWebService -Name 'RetrainSamplePre.2016.8.17.0.3.51.237' -ResourceGroupName 'Default-MachineLearning-SouthCentralUS'
```

To determine the resource group name of an existing web service, run the [Get-AzureRmWebService](#) cmdlet without any parameters to display the web services in your subscription. Locate the web service, and then look at

its web service ID. The name of the resource group is the fourth element in the ID, just after the *resourceGroups* element. In the following example, the resource group name is Default-MachineLearning-SouthCentralUS.

```
Properties : Microsoft.Azure.Management.MachineLearning.WebServices.Models.WebServicePropertiesForGraph
Id : /subscriptions/<subscription ID>/resourceGroups/Default-MachineLearning-
SouthCentralUS/providers/Microsoft.MachineLearning/webServices/RetrainSamplePre.2016.8.17.0.3.51.237
Name : RetrainSamplePre.2016.8.17.0.3.51.237
Location : South Central US
Type : Microsoft.MachineLearning/webServices
Tags : {}
```

Alternatively, to determine the resource group name of an existing web service, sign in to the Azure Machine Learning Web Services portal. Select the web service. The resource group name is the fifth element of the URL of the web service, just after the *resourceGroups* element. In the following example, the resource group name is Default-MachineLearning-SouthCentralUS.

```
https://services.azureml.net/subscriptions/<subscription ID>/resourceGroups/Default-MachineLearning-
SouthCentralUS/providers/Microsoft.MachineLearning/webServices/RetrainSamplePre.2016.8.17.0.3.51.237
```

Export the Web Service Definition object as JSON

To modify the definition of the trained model to use the newly trained model, you must first use the [Export-AzureRmMIWebService](#) cmdlet to export it to a JSON-format file.

```
Export-AzureRmMIWebService -WebService $wsd -OutputFile "C:\temp\mlservice_export.json"
```

Update the reference to the iLearner blob

In the assets, locate the [trained model], update the *uri* value in the *locationInfo* node with the URI of the iLearner blob. The URI is generated by combining the *BaseLocation* and the *RelativeLocation* from the output of the BES retraining call.

```
"asset3": {
  "name": "Retrain Sample [trained model]",
  "type": "Resource",
  "locationInfo": {
    "uri": "https://mltestaccount.blob.core.windows.net/azuremlassetscontainer/baca7bca650f46218633552c0bcbba0e.iLearner"
  },
  "outputPorts": {
    "Results dataset": {
      "type": "Dataset"
    }
  }
},
```

Import the JSON into a Web Service Definition object

You must use the [Import-AzureRmMIWebService](#) cmdlet to convert the modified JSON file back into a Web Service Definition object that you can use to update the predicative experiment.

```
$wsd = Import-AzureRmMIWebService -InputFile "C:\temp\mlservice_export.json"
```

Update the web service

Finally, use the [Update-AzureRmMIWebService](#) cmdlet to update the predictive experiment.

```
Update-AzureRmMIWebService -Name 'RetrainSamplePre.2016.8.17.0.3.51.237' -ResourceGroupName 'Default-MachineLearning-SouthCentralUS'
```

Troubleshooting the retraining of an Azure Machine Learning Classic Web service

1/17/2017 • 3 min to read • [Edit on GitHub](#)

Retraining overview

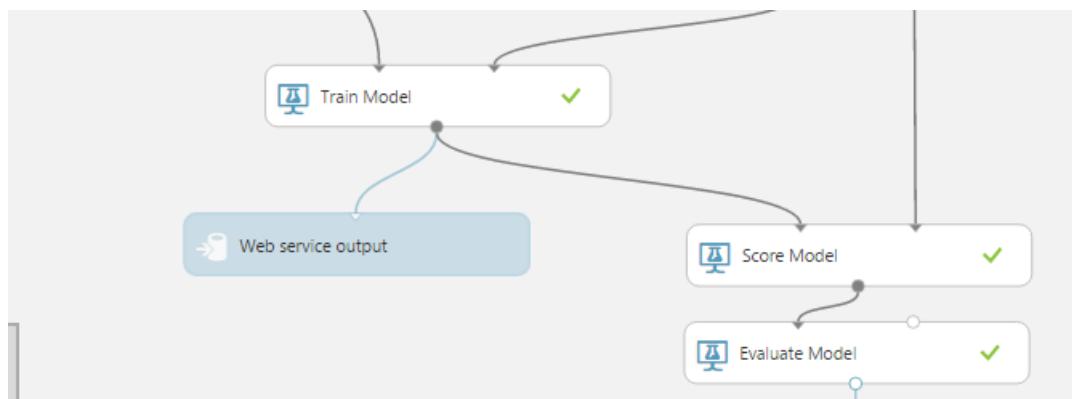
When you deploy a predictive experiment as a scoring web service it is a static model. As new data becomes available or when the consumer of the API has their own data, the model needs to be retrained.

For a complete walkthrough of the retraining process of a Classic Web service, see [Retrain Machine Learning Models Programmatically](#).

Retraining process

When you need to retrain the Web service, you must add some additional pieces:

- A Web service deployed from the Training Experiment. The experiment must have a **Web Service Output** module attached to the output of the **Train Model** module.



- A new endpoint added to your scoring Web service. You can add the endpoint programmatically using the sample code referenced in the Retrain Machine Learning models programmatically topic or through the Azure classic portal.

You can then use the sample C# code from the Training Web Service's API help page to retrain model. Once you have evaluated the results and are satisfied with them, you update the trained model scoring web service using the new endpoint that you added.

With all the pieces in place, the major steps you must take to retrain the model are as follows:

1. Call the Training Web Service: The call is to the Batch Execution Service (BES), not the Request Response Service (RRS). You can use the sample C# code on the API help page to make the call.
2. Find the values for the *BaseLocation*, *RelativeLocation*, and *SasBlobToken*: These values are returned in the output from your call to the Training Web Service.

```
Checking the job status...
Job af13598f64c34b1ea2520e0a0a28af26 running...
Checking the job status...
Job af13598f64c34b1ea2520e0a0a28af26 finished!
The result 'output2' is available at the following Azure Storage location:
BaseLocation: https://mltestaccount.blob.core.windows.net/
RelativeLocation: uploadedresources/output2results.ilearnr
SasBlobToken: ?sv=2015-02-21&sr=b&sig=JMbhu2g1ja947sJKUHxR%2FrcG86ABkManpn79418
%2Bcc%3D&st=2016-08-08T18%3A03%3A52Z&se=2016-08-09T18%3A08%3A52Z&sp=r
```

3. Update the added endpoint from the scoring web service with the new trained model: Using the sample code provided in the Retrain Machine Learning models programmatically, update the new endpoint you added to the

scoring model with the newly trained model from the Training Web Service.

Common obstacles

Check to see if you have the correct PATCH URL

The PATCH URL you are using must be the one associated with the new scoring endpoint you added to the scoring Web service. There are a number of ways to obtain the PATCH URL:

Option 1: Programmatically

To get the correct PATCH URL:

1. Run the [AddEndpoint](#) sample code.
2. From the output of AddEndpoint, find the *HelpLocation* value and copy the URL.

```
Delete endpoint? y/n
y
Starting delete endpoint.

Endpoint deleted.
Create endpoint? y/n
y
Starting create endpoint
{
    "Name": "newendpoint2",
    "Description": "New end point",
    "CreationTime": "2016-08-05T21:51:28.363Z",
    "WorkspaceId": "8aa30f5a07e74e66bb767e25cc98876b",
    "WebServiceId": "710a704638de424191f9a0d0aa8ea48a",
    "HelpLocation": "https://studio.azureml.net/api/help/worksplaces/8aa30f5a07e74e66bb767e25cc98876b/webservices/710a704638de424191f9a0d0aa8ea48a/endpoints/d8d040e7ae044968178ea6ef149d923",
    "PrimaryKey": "25vzzUHphXVkmzPr2D77twUWEIKUEN8SIs9sAZCe93P0Xrmc69+g6rX8KHzDgkpU0vI0qNPKYI9dSDLyIDXXYA==",
    "SecondaryKey": "F32EUhuvC7uwFZVDz9BXFGau8ZSb1mVxJ1V1PIoSuZi6uthhoDgI2AAAt0tCU9trLJtvZpSxZ39t51e38CmYT6g==",
    "ApiLocation": "https://ussouthcentral.services.azureml.net/worksplaces/8aa30f5a07e74e66bb767e25cc98876b/services/d8d040e7ae044968178ea6ef149d923",
    "ExperimentLocation": "https://studio.azureml.net/Home/ViewWorkspace/8aa30f5a07e74e66bb767e25cc98876b#Workspaces/Experiments/Experiment/8aa30f5a07e74e66bb767e25cc98876b.s-id.16104de92a164c70b35f67c610f9a32f.9953-06-04t06.17.00/ViewExperiment",
    "Resources": [
        {
            "Name": "Retrain Sample [trained model]",
            "Kind": "TrainedModel"
        }
    ],
    "Version": "2016-07-29T18:04:36.934Z",
    "PreventUpdate": false,
    "GlobalParameters": [],
    "MaxConcurrentCalls": 4,
    "DiagnosticsTraceLevel": "None",
    "ThrottleLevel": "Low"
}
Endpoint created.
```

3. Paste the URL into a browser to navigate to a page that provides help links for the Web service.
4. Click the **Update Resource** link to open the patch help page.

Option 2: Use the Azure classic portal

1. Sign in to the [Azure classic portal](#).
2. Open the Machine Learning tab.

NAME	STORAGE	STATUS	OWNER	SUBSCRIPTION	LOCATION	
MyNewTest	mitestaccount	✓ Online		Windows Azure MS...	South Central US	
MyTest	mitestaccount	✓ Online		Windows Azure MS...	South Central US	

3. Click your workspace name, then **Web Services**.
4. Click the scoring Web service you are working with. (If you did not modify the default name of the web service, it will end in [Scoring Exp.].)
5. Click **Add Endpoint**.

6. After the endpoint is added, click the endpoint name. Then click **Update Resource** to open the patching help page.

NOTE

If you have added the endpoint to the Training Web Service instead of the Predictive Web Service, you will receive the following error when you click the **Update Resource** link: Sorry, but this feature is not supported or available in this context. This Web Service has no updatable resources. We apologize for the inconvenience and are working on improving this workflow.

The screenshot shows the Microsoft Azure classic portal interface. At the top, there's a navigation bar with 'Microsoft Azure' on the left, a dropdown arrow in the middle, and three buttons: 'Check out the new portal' (blue), 'CREDIT STATUS' (green), and a dark grey button on the right. On the left, a vertical sidebar contains icons for various services: Grid, Network, Compute, Storage, Cloud Services, App Services, Database, Analytics, Media Services, and File Services. Below these icons, service names are listed: 'default', 'test1', and 'newendpoint1', where 'newendpoint1' is highlighted with a blue background. The main content area is titled 'newendpoint1'. It includes tabs for 'DASHBOARD' and 'CONFIGURE'. Under 'DASHBOARD', there are three radio buttons labeled 'COMPUTE', 'FAILED_PREDICTION', and 'PREDICTION', with 'COMPUTE' selected. A timeline chart shows vertical bars from 7:20 to 8:30. Below the chart, there's a section titled 'methods' with three options: 'REQUEST/RESPONSE' (with a help icon), 'BATCH EXECUTION' (with a help icon), and 'UPDATE RESOURCE' (with a help icon). The 'UPDATE RESOURCE' option is underlined, indicating it's the active link.

The PATCH help page contains the PATCH URL you must use and provides sample code you can use to call it.

Request

Method	Request URI	HTTP Version
PATCH	https://management.azureml.net/workspaces/c79283d685684d8488fc73fd3ec49a7f/webservices/0cf85bf373574e77a2918942de25beb9/endpoints/test1	HTTP/1.1

Check to see that you are updating the correct scoring endpoint

- Do not patch the Training Web Service: The patch operation must be performed on the scoring Web service.
- Do not patch the default endpoint on Web service: The patch operation must be performed on the new scoring Web service endpoint that you added.

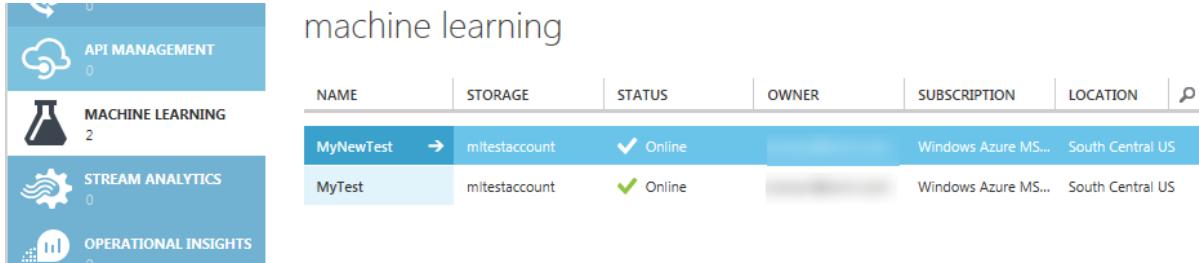
You can verify which Web service the endpoint is on by visiting the Azure classic portal.

NOTE

Be sure you are adding the endpoint to the Predictive Web Service, not the Training Web Service. If you have correctly deployed both a Training and a Predictive Web Service, you should see two separate Web services listed. The Predictive Web Service should end with "[predictive exp.]".

1. Sign in to the [Azure classic portal](#).

2. Open the Machine Learning tab.



NAME	STORAGE	STATUS	OWNER	SUBSCRIPTION	LOCATION
MyNewTest	mltestaccount	✓ Online		Windows Azure MS...	South Central US
MyTest	mltestaccount	✓ Online		Windows Azure MS...	South Central US

3. Select your workspace.

4. Click **Web Services**.

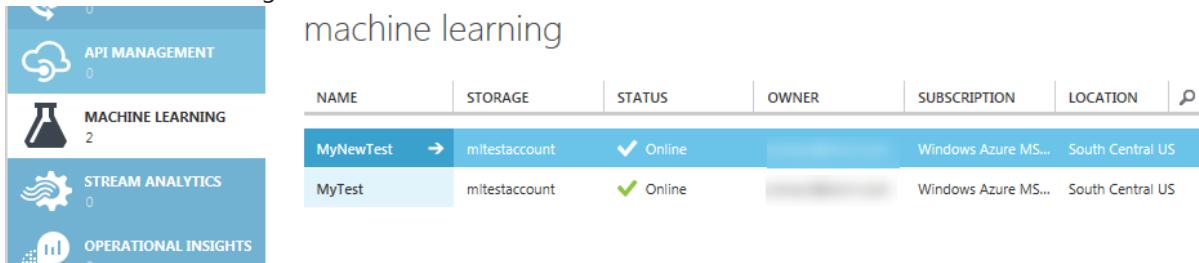
5. Select your Predictive Web Service.

6. Verify that your new endpoint was added to the Web service.

Check the workspace that your web service is in to ensure it is in the correct region

1. Sign in to the [Azure classic portal](#).

2. Select Machine Learning from the menu.



NAME	STORAGE	STATUS	OWNER	SUBSCRIPTION	LOCATION
MyNewTest	mltestaccount	✓ Online		Windows Azure MS...	South Central US
MyTest	mltestaccount	✓ Online		Windows Azure MS...	South Central US

3. Verify the location of your workspace.

Connect to an Azure Machine Learning Web Service

1/17/2017 • 4 min to read • [Edit on GitHub](#)

The Azure Machine Learning developer experience is a Web service API to make predictions from input data in real time or in batch mode. You use Azure Machine Learning Studio to create predictions and deploy an Azure Machine Learning Web service.

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

To learn about how to create and deploy a Machine Learning Web service using Machine Learning Studio:

- For a tutorial on how to create an experiment in Machine Learning Studio, see [Create your first experiment](#).
- For details on how to deploy a Web service, see [Deploy a Machine Learning Web service](#).
- For more information about Machine Learning in general, visit the [Machine Learning Documentation Center](#).

Azure Machine Learning Web service

With the Azure Machine Learning Web service, an external application communicates with a Machine Learning workflow scoring model in real time. A Machine Learning Web service call returns prediction results to an external application. To make a Machine Learning Web service call, you pass an API key that is created when you deploy a prediction. The Machine Learning Web service is based on REST, a popular architecture choice for web programming projects.

Azure Machine Learning has two types of services:

- Request-Response Service (RRS) – A low latency, highly scalable service that provides an interface to the stateless models created and deployed from the Machine Learning Studio.
- Batch Execution Service (BES) – An asynchronous service that scores a batch for data records.

For more information about Machine Learning Web services, see [Deploy a Machine Learning Web service](#).

Get an Azure Machine Learning authorization key

When you deploy your experiment, API keys are generated for the Web service. You can retrieve the keys from several locations.

From the Microsoft Azure Machine Learning Web Services portal

Sign in to the [Microsoft Azure Machine Learning Web Services](#) portal.

To retrieve the API key for a New Machine Learning Web service:

1. In the Azure Machine Learning Web Services portal, click **Web Services** the top menu.
2. Click the Web service for which you want to retrieve the key.
3. On the top menu, click **Consume**.
4. Copy and save the **Primary Key**.

To retrieve the API key for a Classic Machine Learning Web service:

1. In the Azure Machine Learning Web Services portal, click **Classic Web Services** the top menu.
2. Click the Web service with which you are working.
3. Click the endpoint for which you want to retrieve the key.
4. On the top menu, click **Consume**.
5. Copy and save the **Primary Key**.

Classic Web service

You can also retrieve a key for a Classic Web service from Machine Learning Studio or the Azure classic portal.

Machine Learning Studio

1. In Machine Learning Studio, click **WEB SERVICES** on the left.
2. Click a Web service. The **API key** is on the **DASHBOARD** tab.

Azure classic portal

1. Click **MACHINE LEARNING** on the left.
2. Click the workspace in which your Web service is located.
3. Click **WEB SERVICES**.
4. Click a Web service.
5. Click an endpoint. The "API KEY" is down at the lower-right.

Connect to a Machine Learning Web service

You can connect to a Machine Learning Web service using any programming language that supports HTTP request and response. You can view examples in C#, Python, and R from a Machine Learning Web service help page.

Machine Learning API help Machine Learning API help is created when you deploy a Web service. See [Azure Machine Learning Walkthrough- Deploy Web Service](#). The Machine Learning API help contains details about a prediction Web service.

1. Click the Web service with which you are working.
2. Click the endpoint for which you want to view the API Help Page.
3. On the top menu, click **Consume**.
4. Click **API help page** under either the Request-Response or Batch Execution endpoints.

To view Machine Learning API help for a New Web service

In the Azure Machine Learning Web Services Portal:

1. Click **WEB SERVICES** on the top menu.
2. Click the Web service for which you want to retrieve the key.

Click **Consume** to get the URIs for the Request-Reposonse and Batch Execution Services and Sample code in C#, R, and Python.

Click **Swagger API** to get Swagger based documentation for the APIs called from the supplied URIs.

C# Sample

To connect to a Machine Learning Web service, use an **HttpClient** passing ScoreData. ScoreData contains a FeatureVector, an n-dimensional vector of numerical features that represents the ScoreData. You authenticate to the Machine Learning service with an API key.

To connect to a Machine Learning Web service, the **Microsoft.AspNet.WebApi.Client** NuGet package must be installed.

Install Microsoft.AspNet.WebApi.Client NuGet in Visual Studio

1. Publish the Download dataset from UCI: Adult 2 class dataset Web Service.
2. Click **Tools > NuGet Package Manager > Package Manager Console**.
3. Choose **Install-Package Microsoft.AspNet.WebApi.Client**.

To run the code sample

1. Publish "Sample 1: Download dataset from UCI: Adult 2 class dataset" experiment, part of the Machine Learning sample collection.
2. Assign apiKey with the key from a Web service. See [Get an Azure Machine Learning authorization key](#) above.
3. Assign serviceUri with the Request URI.

Python Sample

To connect to a Machine Learning Web service, use the **urllib2** library passing ScoreData. ScoreData contains a FeatureVector, an n-dimensional vector of numerical features that represents the ScoreData. You authenticate to the Machine Learning service with an API key.

To run the code sample

1. Deploy "Sample 1: Download dataset from UCI: Adult 2 class dataset" experiment, part of the Machine Learning sample collection.
2. Assign apiKey with the key from a Web service. See the [Get an Azure Machine Learning authorization key](#) section near the beginning of this article.
3. Assign serviceUri with the Request URI.

Consuming an Azure Machine Learning Web Service from Excel

1/17/2017 • 2 min to read • [Edit on GitHub](#)

Azure Machine Learning Studio makes it easy to call web services directly from Excel without the need to write any code.

If you are using Excel 2013 (or later) or Excel Online, then we recommend that you use the Excel [Excel add-in](#).

NOTE

[Try Azure Machine Learning for free](#)

No credit card or Azure subscription needed. [Get started now >](#)

Steps

Publish a web service. [This page](#) explains how to do it. Currently the Excel workbook feature is only supported for Request/Response services that have a single output (that is, a single scoring label).

Once you have a web service, click on the **WEB SERVICES** section on the left of the studio, and then select the web service to consume from Excel.

Classic Web Service

1. On the **DASHBOARD** tab for the web service is a row for the **REQUEST/RESPONSE** service. If this service had a single output, you should see the **Download Excel Workbook** link in that row.

Default Endpoint						
URL	TYPE	LAST UPDATED	TEST	APPS	LAST UPDATED	P
API help page	REQUEST/RESPONSE	2/6/2015 12:47:31 PM	Test	Download Excel Workbook	2/6/2015 12:47:31 PM	
API help page	BATCH EXECUTION	2/6/2015 12:47:31 PM			2/6/2015 12:47:31 PM	

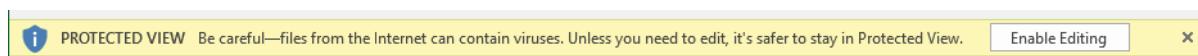
2. Click on **Download Excel Workbook**.

New Web Service

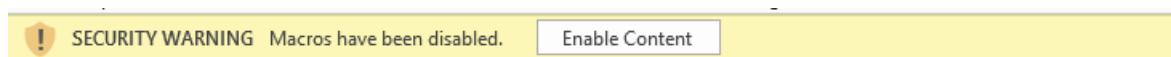
1. In the Azure Machine Learning Web Service portal, select **Consume**.
2. On the Consume page, in the **Web service consumption options** section, click the Excel icon.

Using the workbook

1. Open the workbook.
2. A Security Warning appears; click on the **Enable Editing** button.



3. A Security Warning appears. Click on the **Enable Content** button to run macros on your spreadsheet.



4. Once macros are enabled, a table is generated. Columns in blue are required as input into the RRS web

service, or **PARAMETERS**. Note the output of the RRS service, **PREDICTED VALUES** in green. When all columns for a given row are filled, the workbook automatically calls the scoring API, and displays the scored results.

PARAMETERS												PREDICTED VALUES	
age	workclass	education	education-num	marital-status	occupation	relationship	race	sex	hours-per-week	native-country	ScoredLabels	ScoredProbabilities	
0	0	0	0	0	0	0	0	0	0	0	0 <=50K	0.003267037	

5. To score more than one row, fill the second row with data and the predicted values are produced. You can even paste several rows at once.

You can use any of the Excel features (graphs, power map, conditional formatting, etc.) with the predicted values to help visualize the data.

Sharing your workbook

For the macros to work, your API Key must be part of the spreadsheet. That means that you should share the workbook only with entities/individuals you trust.

Automatic updates

An RRS call is made in these two situations:

1. The first time a row has content in all of its **PARAMETERS**
2. Any time any of the **PARAMETERS** changes in a row that had all of its **PARAMETERS** entered.

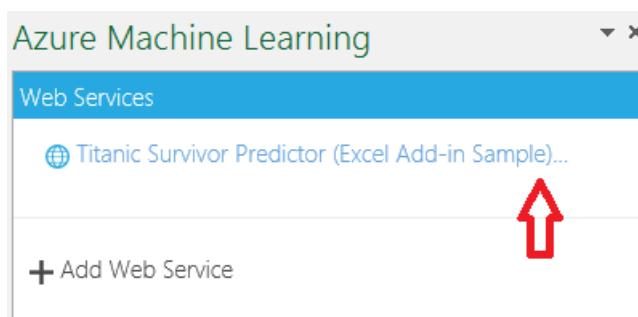
Excel Add-in for Azure Machine Learning Web services

1/17/2017 • 2 min to read • [Edit on GitHub](#)

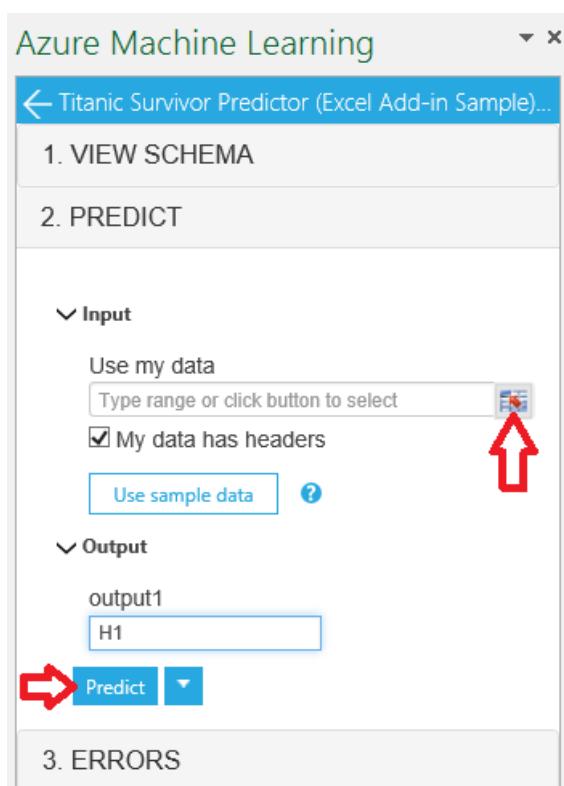
Excel makes it easy to call Web services directly without the need to write any code.

Steps to Use an Existing Web service in the Workbook

1. Open the [sample Excel file](#), which contains the Excel add-in and data about passengers on the Titanic.
2. Choose the Web service by clicking it - "Titanic Survivor Predictor (Excel Add-in Sample) [Score]" in this example.



3. This will take you to the **Predict** section. This workbook already contains sample data, but for a blank workbook you can select a cell in Excel and click **Use sample data**.
4. Select the data with headers and click the input data range icon. Make sure the "My data has headers" box is checked.
5. Under **Output**, enter the cell number where you want the output to be, for example "H1" here.
6. Click **Predict**.



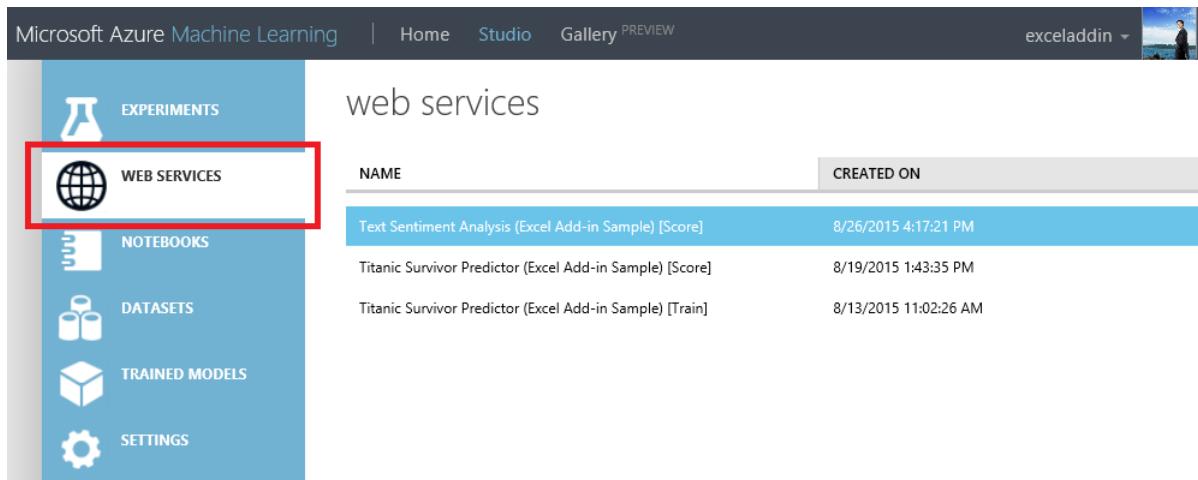
Steps to Add a New Web service

Deploy a Web service or use an existing Web service. For more information on deploying a Web service, see [Walkthrough Step 5: Deploy the Azure Machine Learning Web service](#).

Get the API key for your Web service. Where you do this depends on whether you published a Classic Machine Learning Web service or a New Machine Learning Web service.

Use a Classic Web service

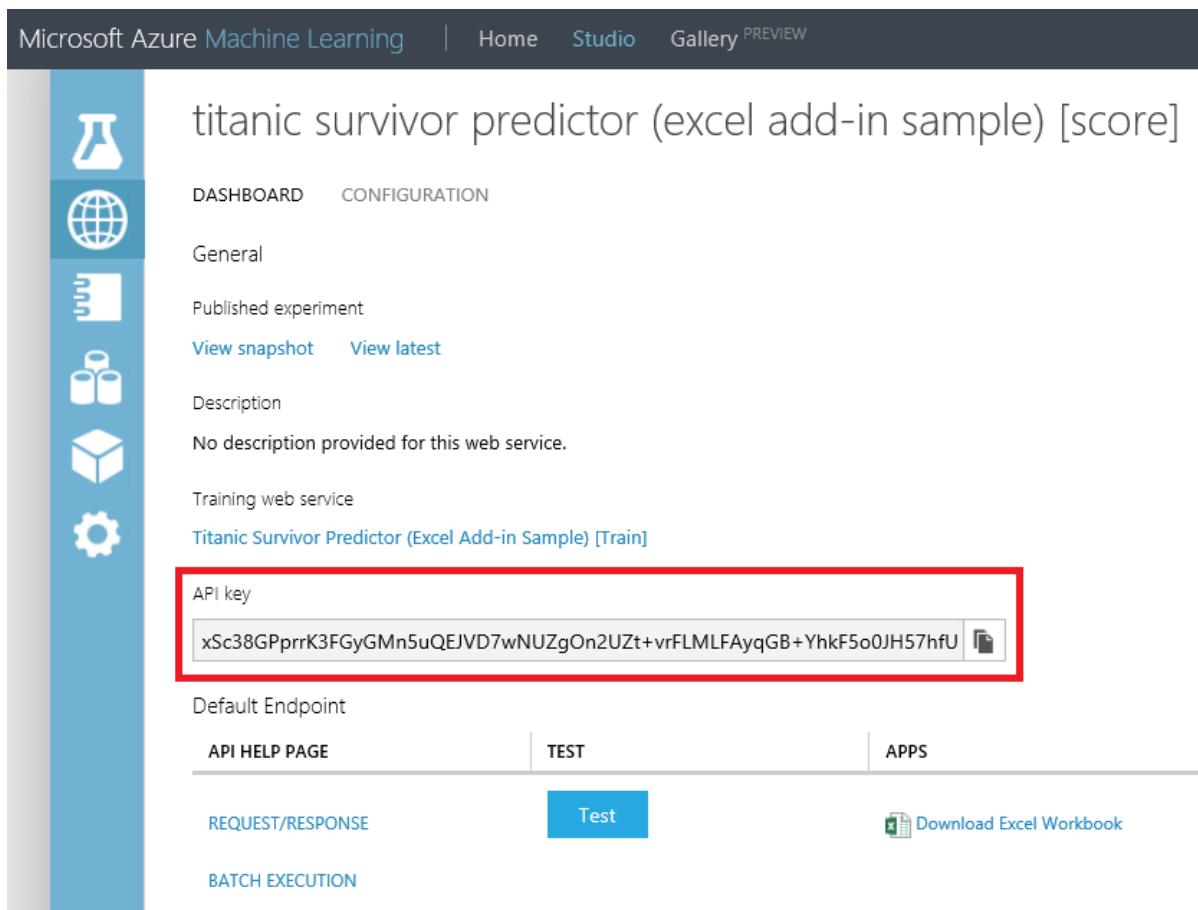
1. In Machine Learning Studio, click the **WEB SERVICES** section in the left pane, and then select the Web service.



The screenshot shows the Microsoft Azure Machine Learning Studio interface. On the left, there is a vertical sidebar with icons for EXPERIMENTS, WEB SERVICES (which is highlighted with a red box), NOTEBOOKS, DATASETS, TRAINED MODELS, and SETTINGS. The main area is titled "web services" and lists three entries:

NAME	CREATED ON
Text Sentiment Analysis (Excel Add-in Sample) [Score]	8/26/2015 4:17:21 PM
Titanic Survivor Predictor (Excel Add-in Sample) [Score]	8/19/2015 1:43:35 PM
Titanic Survivor Predictor (Excel Add-in Sample) [Train]	8/13/2015 11:02:26 AM

2. Copy the API key for the Web service.



The screenshot shows the "titanic survivor predictor (excel add-in sample) [score]" web service dashboard. On the left, there is a vertical sidebar with icons for DASHBOARD (which is selected and highlighted with a red box), CONFIGURATION, General, Published experiment, View snapshot, View latest, Description, Training web service, and Titanic Survivor Predictor (Excel Add-in Sample) [Train]. The main area displays the API key:

API key
xSc38GPrrK3FGyGMn5uQEJVD7wNUZgOn2UZt+vrFLMLFAyqGB+YhkF5o0JH57hfU

Below the API key, there are tabs for Default Endpoint, API HELP PAGE, TEST, APPS, REQUEST/RESPONSE (which is selected and highlighted with a red box), and BATCH EXECUTION. There is also a "Test" button and a link to "Download Excel Workbook".

3. On the **DASHBOARD** tab for the Web service, click the **REQUEST/RESPONSE** link.
4. Look for the **Request URI** section. Copy and save the URL.

NOTE

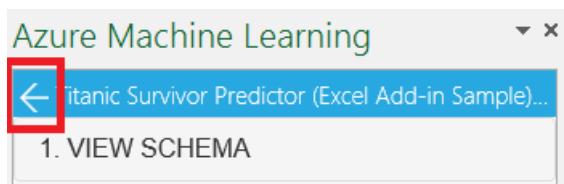
It is now possible to sign into the [Azure Machine Learning Web Services](#) portal to obtain the API key for a Classic Machine Learning Web service.

Use a New Web service

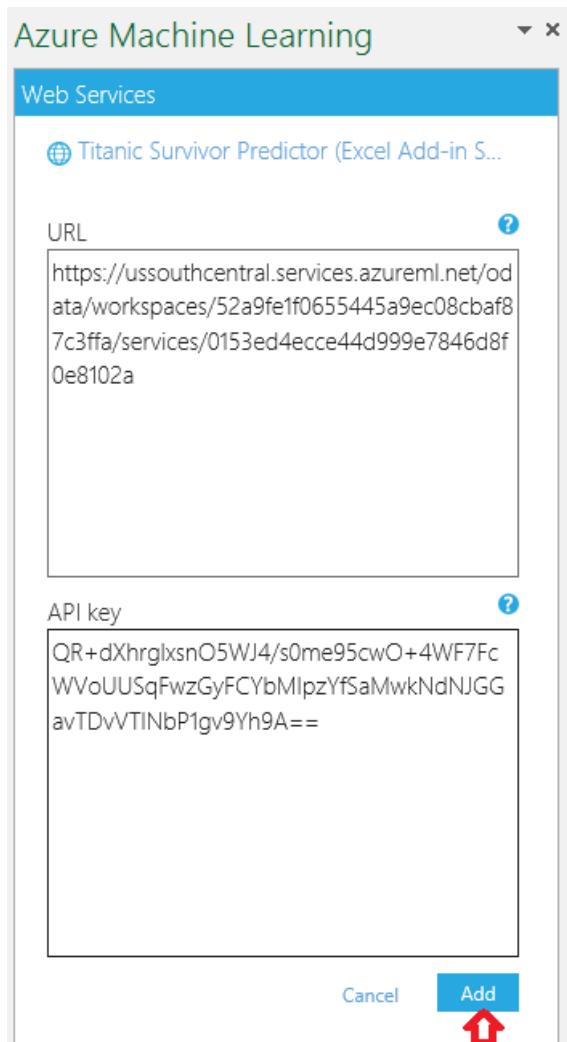
1. In the [Azure Machine Learning Web Services](#) portal, click **Web Services**, then select your Web service.
2. Click **Consume**.
3. Look for the **Basic consumption info** section. Copy and save the **Primary Key** and the **Request-Response URL**.

Steps to Add a New Web service

1. Deploy a Web service or use an existing Web service. For more information on deploying a Web service, see [Walkthrough Step 5: Deploy the Azure Machine Learning Web service](#).
2. Click **Consume**.
3. Look for the **Basic consumption info** section. Copy and save the **Primary Key** and the **Request-Response URL**.
4. In Excel, go to the **Web Services** section (if you are in the **Predict** section, click the back arrow to go to the list of Web services).



5. Click **Add Web Service**.
6. Paste the URL into the Excel add-in text box labeled **URL**.
7. Paste the API/Primary key into the text box labeled **API key**.
8. Click **Add**.



9. To use the Web service, follow the preceding directions, "Steps to Use an Existing Web Service."

Sharing Your Workbook

If you save your workbook, then the API/Primary key for the Web services you have added is also saved. That means you should only share the workbook with individuals you trust.

Ask any questions in the following comment section or on our [forum](#).

Consume an Azure Machine Learning web service with a web app template

1/17/2017 • 5 min to read • [Edit on GitHub](#)

NOTE

This topic describes techniques applicable to a classic web service.

Once you've developed your predictive model and deployed it as an Azure web service using Machine Learning Studio, or using tools such as R or Python, you can access the operationalized model using a REST API.

There are a number of ways to consume the REST API and access the web service. For example, you can write an application in C#, R, or Python using the sample code generated for you when you deployed the web service (available on the API Help Page in the web service dashboard in Machine Learning Studio). Or you can use the sample Microsoft Excel workbook created for you (also available in the web service dashboard in Studio).

But the quickest and easiest way to access your web service is through the Web App Templates available in the [Azure Web App Marketplace](#).

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

The Azure Machine Learning Web App Templates

The web app templates available in the Azure Marketplace can build a custom web app that knows your web service's input data and expected results. All you need to do is give the web app access to your web service and data, and the template does the rest.

Two templates are available:

- [Azure ML Request-Response Service Web App Template](#)
- [Azure ML Batch Execution Service Web App Template](#)

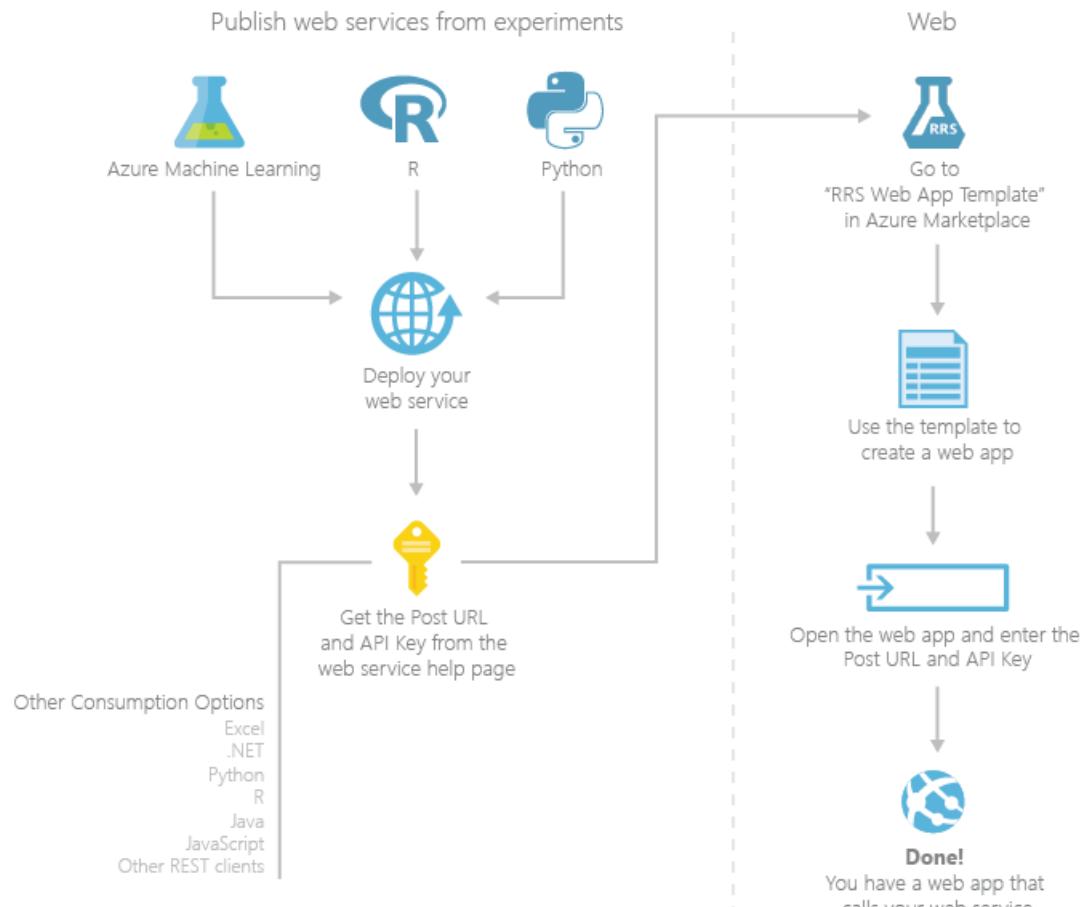
Each template creates a sample ASP.NET application, using the API URI and Key for your web service, and deploys it as a web site to Azure. The Request-Response Service (RRS) template creates a web app that allows you to send a single row of data to the web service to get a single result. The Batch Execution Service (BES) template creates a web app that allows you to send many rows of data to get multiple results.

No coding is required to use these templates. You just supply the API URI and Key and the template builds the application for you.

How to use the Request-Response Service (RRS) template

Once you've deployed your web service, you can follow the steps below to use the RRS web app template, as shown in the following diagram.

Process of publishing a predictive web app for doing real-time predictions (RRS)



1. In Machine Learning Studio, open the **Web Services** tab and then open the web service you want to access. Copy the key listed under **API key** and save it.

Screenshot of the Microsoft Azure Machine Learning Studio interface. The top navigation bar shows "Microsoft Azure Machine Learning" and tabs for "Home", "Studio", and "Gallery PREVIEW". On the left, a sidebar menu lists icons for Home, Studio, Experiment, Data, Compute, and Settings. The main content area is titled "experiment - automobile prediction". Under "DASHBOARD", there are sections for "General" (with "Published experiment" and links to "View snapshot" and "View latest"), "Description" (with the note "No description provided for this web service."), and "API key" (with a text input field containing a long API key, which is circled in red). The "CONFIGURATION" tab is also visible at the top.

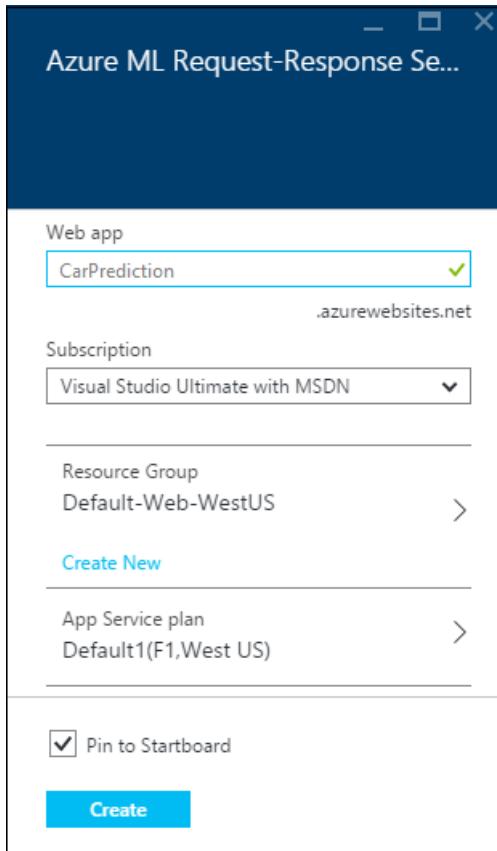
2. Open the **REQUEST/RESPONSE** API Help Page. At the top of the help page, under **Request**, copy the **Request URI** value and save it. This value will look like this:

```
https://ussouthcentral.services.azureml.net/workspaces/<workspace-id>/services/<service-id>/execute?api-version=2.0&details=true
```

Request		HTTP Version
Method	Request URI	
POST	https://ussouthcentral.services.azureml.net/workspaces/.../services/.../execute?api-version=2.0&details=true	HTTP/1.1

3. Go to the [Azure portal](#), **Login**, click **New**, Search for and select **Azure ML Request-Response Service Web App**, then click **Create**.

- Give your web app a unique name. The URL of the web app will be this name followed by `.azurewebsites.net`. For example, `http://carprediction.azurewebsites.net`.
- Select the Azure subscription and services under which your web service is running.
- Click **Create**.



4. When Azure has finished deploying the web app, click the **URL** on the web app settings page in Azure, or enter the URL in a web browser. For example, `http://carprediction.azurewebsites.net`.

5. When the web app first runs it will ask you for the **API Post URL** and **API Key**. Enter the values you saved earlier:

- **Request URI** from the API Help Page for **API Post URL**
- **API Key** from the web service dashboard for the **API Key**.

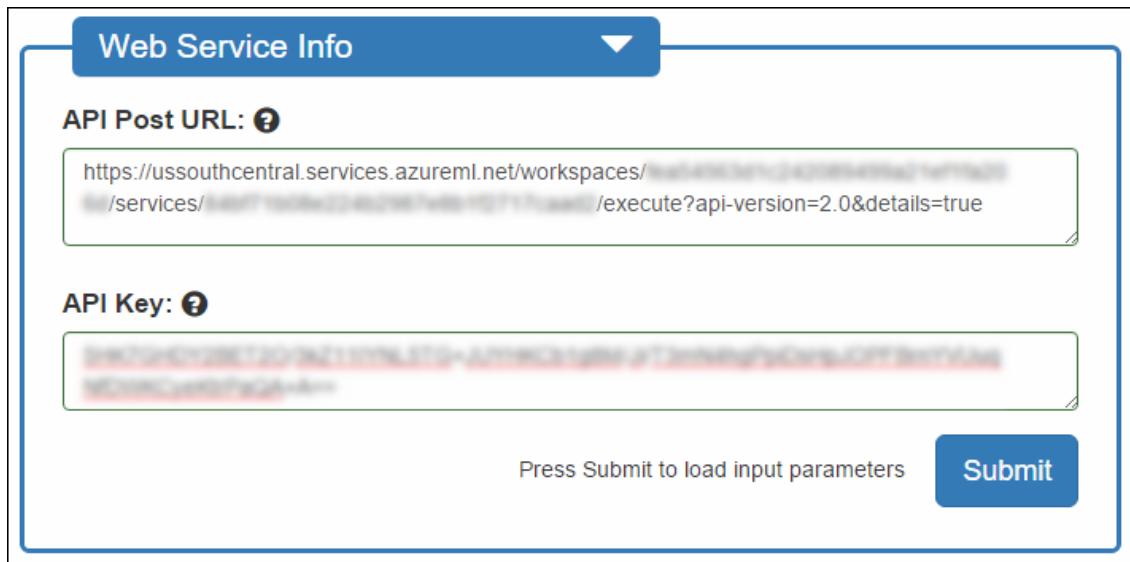
Click **Submit**.

Web Service Info

API Post URL: <https://ussouthcentral.services.azureml.net/workspaces/.../services/.../execute?api-version=2.0&details=true>

API Key: [\[REDACTED\]](#)

Press Submit to load input parameters **Submit**



6. The web app displays its **Web App Configuration** page with the current web service settings. Here you can make changes to the settings used by the web app.

NOTE

Changing the settings here only changes them for this web app. It doesn't change the default settings of your web service. For example, if you change the **Description** here it doesn't change the description shown on the web service dashboard in Machine Learning Studio.

When you're done, click **Save changes**, and then click **Go to Home Page**.

7. From the home page you can enter values to send to your web service, click **Submit**, and the result will be returned.

If you want to return to the **Configuration** page, go to the [setting.aspx](#) page of the web app. For example: <http://caprediction.azurewebsites.net/setting.aspx>. You will be prompted to enter the API key again - you need that to access the page and update the settings.

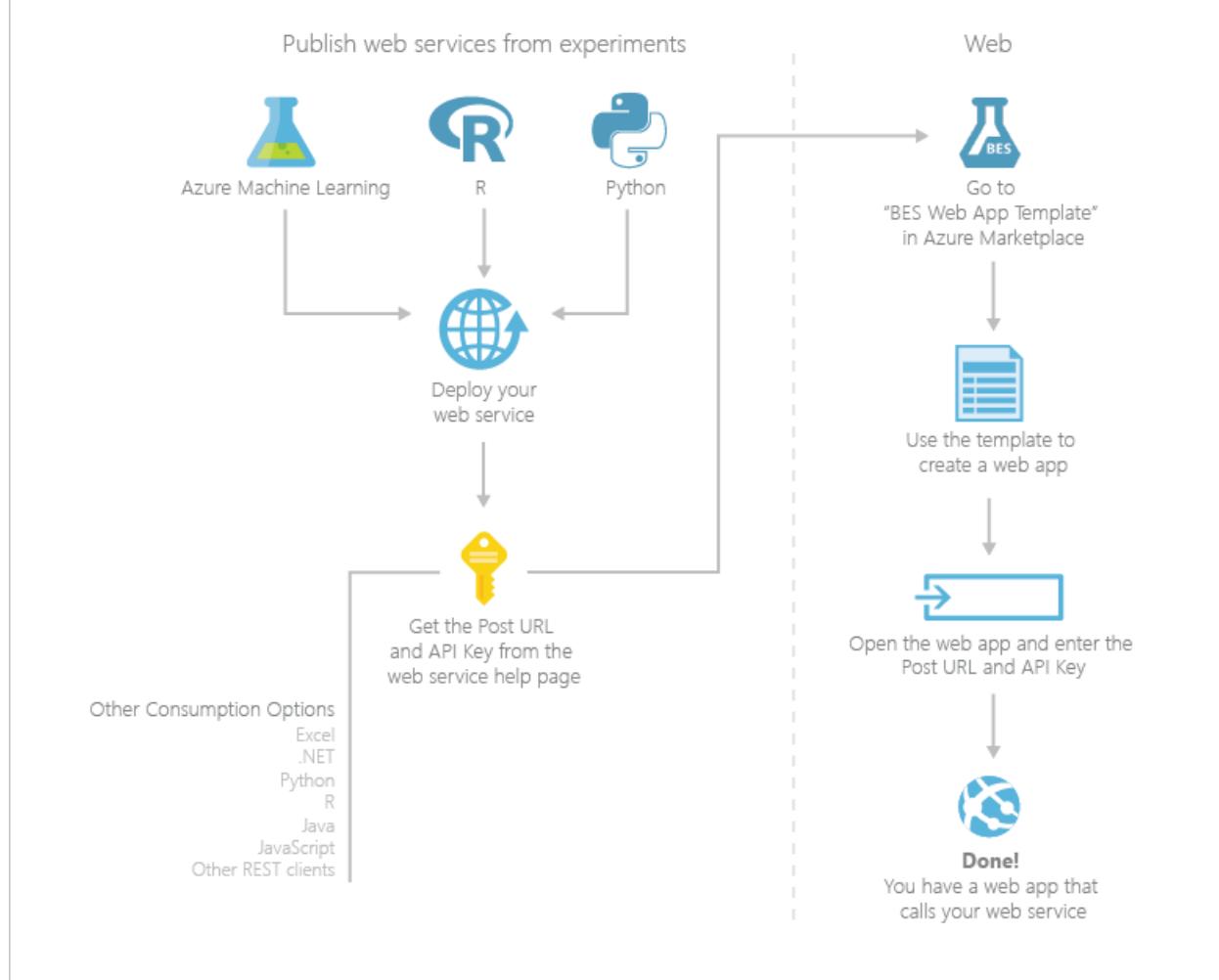
You can stop, restart, or delete the web app in the Azure portal like any other web app. As long as it is running you can browse to the home web address and enter new values.

How to use the Batch Execution Service (BES) template

You can use the BES web app template in the same way as the RRS template, except that the web app that's created will allow you to submit multiple rows of data and receive multiple results.

The results from a batch execution web service are stored in an Azure storage container; the input values can come from Azure storage or a local file. So, you'll need an Azure storage container to hold the results returned by the web app, and you'll need to get your input data ready.

Process of publishing a predictive web app for doing batch predictions (BES)



1. Follow the same procedure to create the BES web app as for the RSS template, except:
 - Get the **Request URI** from the **BATCH EXECUTION** API Help Page for the web service.
 - Go to [Azure ML Batch Execution Service Web App Template](#) to open the BES template on Azure Marketplace and click **Create Web App**.
2. To specify where you want the results stored, enter the destination container information on the web app home page. Also specify where the web app can get the input values, either in a local file or an Azure storage container. Click **Submit**.



Car price prediction [Scoring Exp.]

Azure Storage Info

Account Name
account-name

Account Key

Container Name
container-name

Input File Info

[Load from local machine](#)

Select batch file to upload for scoring
[car-models.dataset](#) [Choose file](#)

[Load from Azure Storage Blob](#) [Same as above](#)

[Job Status](#) [Submit](#)

Powered by Azure Machine Learning

The web app will display a page with job status. When the job has completed you'll be given the location of the results in Azure blob storage. You also have the option of downloading the results to a local file.

For more information

To learn more about...

- creating a machine learning experiment with Machine Learning Studio, see [Create your first experiment in Azure Machine Learning Studio](#)
- how to deploy your machine learning experiment as a web service, see [Deploy an Azure Machine Learning web service](#)
- other ways to access your web service, see [How to consume an Azure Machine Learning web service](#)

Copy sample experiments to create new machine learning experiments

1/17/2017 • 2 min to read • [Edit on GitHub](#)

Learn how to start with sample experiments from [Cortana Intelligence Gallery](#) instead of creating machine learning experiments from scratch. You can use the samples to build your own machine learning solution.

In the gallery are sample experiments by the Microsoft Azure Machine Learning team as well as samples shared by the Machine Learning community. You also can ask questions or post comments about experiments.

To see how to use the gallery, watch the 3-minute video [Copy other people's work to do data science](#) from the series [Data Science for Beginners](#).

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

Find an experiment to copy in Cortana Intelligence Gallery

To see what experiments are available, go to the [Gallery](#) and click **Experiments** at the top of the page.

Find the newest or most popular experiments

On this page, you can see **Recently added** experiments, or scroll down to look at **What's popular** or the latest **Popular Microsoft experiments**.

Look for an experiment that meets specific requirements

To browse all experiments:

1. Click **Browse all** at the top of the page.
2. Under **Refine by**, select **Experiment** to see all the experiments in the Gallery.
3. You can find experiments that meet your requirements a couple different ways:
 - **Select filters on the left.** For example, to browse experiments that use a PCA-based anomaly detection algorithm, select **Experiment** under **Categories**, and **PCA-Based Anomaly Detection** under **Algorithms Used**. (If you don't see that algorithm, click **Show all** at the bottom of the list.)

Refine by

CATEGORIES

- Solution Template
- Experiment
- Machine Learning API
- Competition
- Tutorial
- Collection
- Notebook

SHOW

- Microsoft content only

TAGS

ALGORITHMS USED

- One-Class Support Vector Machine
- PCA-Based Anomaly Detection
- K-Means Clustering
- Multiclass Neural Network
- Bayesian Linear Regression

[Show all](#)

LANGUAGE

- **Use the search box.** For example, to find experiments contributed by Microsoft related to digit recognition that use a two-class support vector machine algorithm, enter "digit recognition" in the search box. Then, select the filters **Experiment**, **Microsoft content only**, and **Two-Class Support Vector Machine**:

Machine:

The screenshot shows the Cortana Intelligence Gallery interface. At the top, there's a search bar with 'digit recognition' typed in, and a magnifying glass icon. To the right of the search bar are user profile icons and a 'Sign in' link. Below the search bar is a navigation menu with 'Browse all' (which is underlined in green), 'Solution Templates', 'Experiments', 'Machine Learning APIs', and 'More'. On the left side, there's a 'Refine by' sidebar with sections for 'CATEGORIES', 'SHOW', 'TAGS', and 'ALGORITHMS USED'. Under 'CATEGORIES', 'Experiment' is checked. Under 'SHOW', 'Microsoft content only' is checked. Under 'ALGORITHMS USED', 'Two-Class Support Vector Machine' is checked. The main content area displays search results for 'digit recognition'. It says 'We've found 2 results for digit recognition' and 'Sort by: Relevance'. There are two experiment cards shown:

- Compare Multi-class Classifiers: Letter recogn...**: This card has a small image showing a table with 'Letter' and 'Probability' columns, listing 'a' (0.7), 'q' (0.2), and 'o' (0.1). Below the image is a brief description: 'This sample demonstrates how to compare multiple multi-class classifiers using the letter recogn...'. It includes a 'One-vs-All Multiclass, Two-Class Support Vector Machine, Multicla...' tag, 3909 views, 1195 likes, and was posted 9 months ago.
- Sample 7: Train, Test, Evaluate for Multiclass ...**: This card has an image showing three boxes labeled 'A', 'B', and 'C' with a green gear icon between them. Below the image is a brief description: 'Sample experiment that uses multiclass classification to predict the letter category as one of the...'. It includes a 'Multiclass Decision Jungle, One-vs-All Multiclass, Two-Class Support ...' tag, 2108 views, 2685 likes, and was posted 9 months ago.

 Both cards have a 'Microsoft' logo at the bottom.

4. Click an experiment to learn more about it.
5. To run and/or modify the experiment, click **Open in Studio** on the experiment's page.

NOTE

To open an experiment in Machine Learning Studio, you need to sign in with your Microsoft account credentials. If you don't have a Machine Learning workspace yet, a free trial workspace is created. [Learn what's included in the Machine Learning free trial](#)

EXPERIMENT

Compare Multi-class Classifiers: Letter recognition

Microsoft • published on September 2, 2014

Summary

This sample demonstrates how to compare multiple multi-class classifiers using the letter recognition dataset.

Description

Compare Multi-class Classifiers: Letter Recognition

This sample demonstrates how to create multiclass classifiers and evaluate and compare the performance of multiple models.

Data

For this experiment, we use the letter image recognition data from the [UCI repository](#). The first column is the label, which identifies each row as one of 26 letters, A-Z. The remaining 16 columns are feature columns. The dataset contains 20000 instances.

Description and other details about the data can be found at <http://archive.ics.uci.edu/ml/machine-learning-databases/letter-recognition/letter-recognition.names>.

Letter	Probability
a	0.7
q	0.2
o	0.1

[Open in Studio](#)

+ Add to Collection

3909 views

1195 downloads

[Tweet](#)

[Share](#)



ALGORITHMS

[One-vs-All Multiclass](#) , [Two-Class Support Vector Machine](#) , [Multiclass Decision Forest](#) , [Multiclass Decision Jungle](#) , [Multiclass Logistic Regression](#) , [Multiclass Neural Network](#)

Use a template in Machine Learning Studio

You also can create a new experiment in Machine Learning Studio using a Gallery sample as a template.

1. Sign in with your Microsoft account credentials to the [Studio](#), and then click **New** to create a new experiment.
2. Browse through the sample content and click one.

A new experiment is created in your workspace using the sample experiment as a template.

Next steps

- [Prepare your data](#)
- [Try using R in your experiment](#)
- [Review sample R experiments](#)
- [Create a web service API](#)

Use the sample datasets in Azure Machine Learning Studio

1/17/2017 • 14 min to read • [Edit on GitHub](#)

When you create a new workspace in Azure Machine Learning, a number of sample datasets and experiments are included by default. Many of these sample datasets are used by the sample models in the [Azure Cortana Intelligence Gallery](#). Others are included as examples of various types of data typically used in machine learning.

Some of these datasets are available in Azure Blob storage. For these datasets, the following table provides a direct link. You can use these datasets in your experiments by using the [Import Data](#) module.

The rest of these sample datasets are available in your workspace under **Saved Datasets** in the module palette to the left of the experiment canvas when you open or create a new experiment in Machine Learning Studio. You can use any of these datasets in your own experiment by dragging it to your experiment canvas.

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

Dataset Name	Dataset Description
Adult Census Income Binary Classification dataset	A subset of the 1994 Census database, using working adults over the age of 16 with an adjusted income index of > 100. Usage: Classify people using demographics to predict whether a person earns over 50K a year. Related Research: Kohavi, R., Becker, B., (1996). UCI Machine Learning Repository http://archive.ics.uci.edu/ml . Irvine, CA: University of California, School of Information and Computer Science
Airport Codes Dataset	U.S. airport codes. This dataset contains one row for each U.S. airport, providing the airport ID number and name along with the location city and state.
Automobile price data (Raw)	Information about automobiles by make and model, including the price, features such as the number of cylinders and MPG, as well as an insurance risk score. The risk score is initially associated with auto price and then adjusted for actual risk in a process known to actuaries as symboling. A value of +3 indicates that the auto is risky, and a value of -3 that it is probably safe. Usage: Predict the risk score by features, using regression or multivariate classification. Related Research: Schlimmer, J.C. (1987). UCI Machine Learning Repository http://archive.ics.uci.edu/ml . Irvine, CA: University of California, School of Information and Computer Science

Bike Rental UCI dataset	<p>UCI Bike Rental dataset that is based on real data from Capital Bikeshare company that maintains a bike rental network in Washington DC.</p> <p>The dataset has one row for each hour of each day in 2011 and 2012, for a total of 17,379 rows. The range of hourly bike rentals is from 1 to 977.</p>
Bill Gates RGB Image	<p>Publicly available image file converted to CSV data.</p> <p>The code for converting the image is provided in the Color quantization using K-Means clustering model detail page.</p>
Blood donation data	<p>A subset of data from the blood donor database of the Blood Transfusion Service Center of Hsin-Chu City, Taiwan.</p> <p>Donor data includes the months since last donation), and frequency, or the total number of donations, time since last donation, and amount of blood donated.</p> <p>Usage: The goal is to predict via classification whether the donor donated blood in March 2007, where 1 indicates a donor during the target period, and 0 a non-donor.</p> <p>Related Research: Yeh, I.C., (2008). UCI Machine Learning Repository http://archive.ics.uci.edu/ml. Irvine, CA: University of California, School of Information and Computer Science</p> <p>Yeh, I-Cheng, Yang, King-Jang, and Ting, Tao-Ming, "Knowledge discovery on RFM model using Bernoulli sequence, "Expert Systems with Applications, 2008, http://dx.doi.org/10.1016/j.eswa.2008.07.018</p>
Book Reviews from Amazon	<p>Reviews of books in Amazon, taken from the amazon.com website by University of Pennsylvania researchers (sentiment). See the research paper, "Biographies, Bollywood, Boom-boxes and Blenders: Domain Adaptation for Sentiment Classification" by John Blitzer, Mark Dredze, and Fernando Pereira; Association of Computational Linguistics (ACL), 2007.</p> <p>The original dataset has 975K reviews with rankings 1, 2, 3, 4, or 5. The reviews were written in English and are from the time period 1997-2007. This dataset has been down-sampled to 10K reviews.</p>
Breast cancer data	<p>One of three cancer-related datasets provided by the Oncology Institute that appears frequently in machine learning literature. Combines diagnostic information with features from laboratory analysis of about 300 tissue samples.</p> <p>Usage: Classify the type of cancer, based on 9 attributes, some of which are linear and some are categorical.</p> <p>Related Research: Wohlberg, W.H., Street, W.N., & Mangasarian, O.L. (1995). UCI Machine Learning Repository http://archive.ics.uci.edu/ml. Irvine, CA: University of California, School of Information and Computer Science</p>
Breast Cancer Features	<p>The dataset contains information for 102K suspicious regions (candidates) of X-ray images, each described by 117 features. The features are proprietary and their meaning is not revealed by the dataset creators (Siemens Healthcare).</p>

Breast Cancer Info	The dataset contains additional information for each suspicious region of X-ray image. Each example provides information (e.g., label, patient ID, coordinates of patch relative to the whole image) about the corresponding row number in the Breast Cancer Features dataset. Each patient has a number of examples. For patients who have a cancer, some examples are positive and some are negative. For patients who don't have a cancer, all examples are negative. The dataset has 102K examples. The dataset is biased, 0.6% of the points are positive, the rest are negative. The dataset was made available by Siemens Healthcare.
CRM Appetency Labels Shared	Labels from the KDD Cup 2009 customer relationship prediction challenge (orange_small_train_appetency.labels).
CRM Churn Labels Shared	Labels from the KDD Cup 2009 customer relationship prediction challenge (orange_small_train_churn.labels).
CRM Dataset Shared	This data comes from the KDD Cup 2009 customer relationship prediction challenge (orange_small_train.data.zip). The dataset contains 50K customers from the French Telecom company Orange. Each customer has 230 anonymized features, 190 of which are numeric and 40 are categorical. The features are very sparse.
CRM Upselling Labels Shared	Labels from the KDD Cup 2009 customer relationship prediction challenge (orange_large_train_upselling.labels).
Energy Efficiency Regression data	A collection of simulated energy profiles, based on 12 different building shapes. The buildings are differentiated by 8 features, such as glazing area, the glazing area distribution, and orientation. Usage: Use either regression or classification to predict the energy-efficiency rating based as one of two real valued responses. For multi-class classification, is round the response variable to the nearest integer. Related Research: Xifara, A. & Tsanas, A. (2012). UCI Machine Learning Repository http://archive.ics.uci.edu/ml . Irvine, CA: University of California, School of Information and Computer Science
Flight Delays Data	Passenger flight on-time performance data taken from the TranStats data collection of the U.S. Department of Transportation (On-Time). The dataset covers the time period April–October 2013. Before uploading to Azure Machine Learning Studio, the dataset was processed as follows: <ul style="list-style-type: none">• The dataset was filtered to cover only the 70 busiest airports in the continental US• Canceled flights were labeled as delayed by more than 15 minutes• Diverted flights were filtered out• The following columns were selected: Year, Month, DayofMonth, DayOfWeek, Carrier, OriginAirportID, DestAirportID, CRSDepTime, DepDelay, DepDel15, CRSArrTime, ArrDelay, ArrDel15, Canceled

Flight on-time performance (Raw)	<p>Records of airplane flight arrivals and departures within United States from October 2011.</p> <p>Usage: Predict flight delays.</p> <p>Related Research: From US Dept. of Transportation http://www.transtats.bts.gov/DL_SelectFields.asp? Table_ID=236&DB_Short_Name=On-Time.</p>
Forest fires data	<p>Contains weather data, such as temperature and humidity indices and wind speed, from an area of northeast Portugal, combined with records of forest fires.</p> <p>Usage: This is a difficult regression task, where the aim is to predict the burned area of forest fires.</p> <p>Related Research: Cortez, P., & Morais, A. (2008). UCI Machine Learning Repository http://archive.ics.uci.edu/ml. Irvine, CA: University of California, School of Information and Computer Science</p> <p>[Cortez and Morais, 2007] P. Cortez and A. Morais. A Data Mining Approach to Predict Forest Fires using Meteorological Data. In J. Neves, M. F. Santos and J. Machado Eds., New Trends in Artificial Intelligence, Proceedings of the 13th EPIA 2007 - Portuguese Conference on Artificial Intelligence, December, Guimarães, Portugal, pp. 512-523, 2007. APPIA, ISBN-13 978-989-95618-0-9. Available at: http://www.dsi.uminho.pt/~pcortez/fires.pdf.</p>
German Credit Card UCI dataset	<p>The UCI Statlog (German Credit Card) dataset (Statlog+German+Credit+Data), using the german.data file.</p> <p>The dataset classifies people, described by a set of attributes, as low or high credit risks. Each example represents a person. There are 20 features, both numerical and categorical, and a binary label (the credit risk value). High credit risk entries have label = 2, low credit risk entries have label = 1. The cost of misclassifying a low risk example as high is 1, whereas the cost of misclassifying a high risk example as low is 5.</p>
IMDB Movie Titles	<p>The dataset contains information about movies that were rated in Twitter tweets: IMDB movie ID, movie name, genre, and production year. There are 17K movies in the dataset. The dataset was introduced in the paper "S. Dooms, T. De Pessemier and L. Martens. MovieTweetings: a Movie Rating Dataset Collected From Twitter. Workshop on Crowdsourcing and Human Computation for Recommender Systems, CrowdRec at RecSys 2013."</p>
Iris two class data	<p>This is perhaps the best known database to be found in the pattern recognition literature. The dataset is relatively small, containing 50 examples each of petal measurements from three iris varieties.</p> <p>Usage: Predict the iris type from the measurements.</p> <p>Related Research: Fisher, R.A. (1988). UCI Machine Learning Repository http://archive.ics.uci.edu/ml. Irvine, CA: University of California, School of Information and Computer Science</p>

Movie Tweets	<p>The dataset is an extended version of the Movie Tweetings dataset. The dataset has 170K ratings for movies, extracted from well-structured tweets on Twitter. Each instance represents a tweet and is a tuple: user ID, IMDB movie ID, rating, timestamp, number of favorites for this tweet, and number of retweets of this tweet. The dataset was made available by A. Said, S. Dooms, B. Loni and D. Tikk for Recommender Systems Challenge 2014.</p>
MPG data for various automobiles	<p>This dataset is a slightly modified version of the dataset provided by the StatLib library of Carnegie Mellon University. The dataset was used in the 1983 American Statistical Association Exposition.</p> <p>The data lists fuel consumption for various automobiles in miles per gallon, along with information such as the number of cylinders, engine displacement, horsepower, total weight, and acceleration.</p> <p>Usage: Predict fuel economy based on 3 multivalued discrete attributes and 5 continuous attributes.</p> <p>Related Research: StatLib, Carnegie Mellon University, (1993). UCI Machine Learning Repository http://archive.ics.uci.edu/ml. Irvine, CA: University of California, School of Information and Computer Science</p>
Pima Indians Diabetes Binary Classification dataset	<p>A subset of data from the National Institute of Diabetes and Digestive and Kidney Diseases database. The dataset was filtered to focus on female patients of Pima Indian heritage. The data includes medical data such as glucose and insulin levels, as well as lifestyle factors.</p> <p>Usage: Predict whether the subject has diabetes (binary classification).</p> <p>Related Research: Sigillito, V. (1990). UCI Machine Learning Repository http://archive.ics.uci.edu/ml. Irvine, CA: University of California, School of Information and Computer Science</p>
Restaurant customer data	<p>A set of metadata about customers, including demographics and preferences.</p> <p>Usage: Use this dataset, in combination with the other two restaurant datasets, to train and test a recommender system.</p> <p>Related Research: Bache, K. and Lichman, M. (2013). UCI Machine Learning Repository http://archive.ics.uci.edu/ml. Irvine, CA: University of California, School of Information and Computer Science.</p>
Restaurant feature data	<p>A set of metadata about restaurants and their features, such as food type, dining style, and location.</p> <p>Usage: Use this dataset, in combination with the other two restaurant datasets, to train and test a recommender system.</p> <p>Related Research: Bache, K. and Lichman, M. (2013). UCI Machine Learning Repository http://archive.ics.uci.edu/ml. Irvine, CA: University of California, School of Information and Computer Science.</p>

Restaurant ratings	<p>Contains ratings given by users to restaurants on a scale from 0 to 2.</p> <p>Usage: Use this dataset, in combination with the other two restaurant datasets, to train and test a recommender system.</p> <p>Related Research: Bache, K. and Lichman, M. (2013). UCI Machine Learning Repository http://archive.ics.uci.edu/ml. Irvine, CA: University of California, School of Information and Computer Science.</p>
Steel Annealing multi-class dataset	<p>This dataset contains a series of records from steel annealing trials with the physical attributes (width, thickness, type (coil, sheet, etc.) of the resulting steel types.</p> <p>Usage: Predict any of two numeric class attributes; hardness or strength. You might also analyze correlations among attributes.</p> <p>Steel grades follow a set standard, defined by SAE and other organizations. You are looking for a specific 'grade' (the class variable) and want to understand the values needed.</p> <p>Related Research: Sterling, D. & Buntine, W. (NA). UCI Machine Learning Repository http://archive.ics.uci.edu/ml. Irvine, CA: University of California, School of Information and Computer Science</p> <p>A useful guide to steel grades can be found here: http://www.outokumpu.com/SiteCollectionDocuments/Outokumpu-steel-grades-properties-global-standards.pdf</p>

Telescope data	<p>Records of high energy gamma particle bursts along with background noise, both simulated using a Monte Carlo process.</p> <p>The intent of the simulation was to improve the accuracy of ground-based atmospheric Cherenkov gamma telescopes, using statistical methods to differentiate between the desired signal (Cherenkov radiation showers) and background noise (hadronic showers initiated by cosmic rays in the upper atmosphere).</p> <p>The data has been pre-processed to create an elongated cluster with the long axis oriented towards the camera center. The characteristics of this ellipse (often called Hillas parameters) are among the image parameters that can be used for discrimination.</p> <p>Usage: Predict whether image of a shower represents signal or background noise.</p> <p>Notes: Simple classification accuracy is not meaningful for this data, since classifying a background event as signal is worse than classifying a signal event as background. For comparison of different classifiers the ROC graph should be used. The probability of accepting a background event as signal must be below one of the following thresholds: 0.01 , 0.02 , 0.05 , 0.1 , or 0.2.</p> <p>Also, note that the number of background events (h, for hadronic showers) is underestimated, whereas in real measurements, the h or noise class represents the majority of events.</p> <p>Related Research: Bock, R.K. (1995). UCI Machine Learning Repository http://archive.ics.uci.edu/ml. Irvine, CA: University of California, School of Information</p>
Weather Dataset	<p>Hourly land-based weather observations from NOAA (merged data from 201304 to 201310).</p> <p>The weather data covers observations made from airport weather stations, covering the time period April–October 2013. Before uploading to Azure Machine Learning Studio, the dataset was processed as follows:</p> <ul style="list-style-type: none"> • Weather station IDs were mapped to corresponding airport IDs • Weather stations not associated with the 70 busiest airports were filtered out • The Date column was split into separate Year, Month, and Day columns • The following columns were selected: AirportID, Year, Month, Day, Time, TimeZone, SkyCondition, Visibility, WeatherType, DryBulbFarenheit, DryBulbCelsius, WetBulbFarenheit, WetBulbCelsius, DewPointFarenheit, DewPointCelsius, RelativeHumidity, WindSpeed, WindDirection, ValueForWindCharacter, StationPressure, PressureTendency, PressureChange, SeaLevelPressure, RecordType, HourlyPrecip, Altimeter

Wikipedia SP 500 Dataset	<p>Data is derived from Wikipedia (http://www.wikipedia.org/) based on articles of each S&P 500 company, stored as XML data.</p> <p>Before uploading to Azure Machine Learning Studio, the dataset was processed as follows:</p> <ul style="list-style-type: none"> • Extract text content for each specific company • Remove wiki formatting • Remove non-alphanumeric characters • Convert all text to lowercase • Known company categories were added <p>Note that for some companies an article could not be found, so the number of records is less than 500.</p>
direct_marketing.csv	The dataset contains customer data and indications about their response to a direct mailing campaign. Each row represents a customer. The dataset contains 9 features about user demographics and past behavior, and 3 label columns (visit, conversion, and spend). Visit is a binary column that indicates that a customer visited after the marketing campaign, conversion indicates a customer purchased something, and spend is the amount that was spent. The dataset was made available by Kevin Hillstrom for MineThatData E-Mail Analytics And Data Mining Challenge.
lyrl2004_tokens_test.csv	Features of test examples in the RCV1-V2 Reuters news dataset. The dataset has 781K news articles along with their IDs (first column of the dataset). Each article is tokenized, stopworded, and stemmed. The dataset was made available by David. D. Lewis.
lyrl2004_tokens_train.csv	Features of training examples in the RCV1-V2 Reuters news dataset. The dataset has 23K news articles along with their IDs (first column of the dataset). Each article is tokenized, stopworded, and stemmed. The dataset was made available by David. D. Lewis.
network_intrusion_detection.csv	<p>Dataset from the KDD Cup 1999 Knowledge Discovery and Data Mining Tools Competition (kddcup99.html).</p> <p>The dataset was downloaded and stored in Azure Blob storage (network_intrusion_detection.csv) and includes both training and testing datasets. The training dataset has approximately 126K rows and 43 columns, including the labels. Three columns are part of the label information, and 40 columns, consisting of numeric and string/categorical features, are available for training the model. The test data has approximately 22.5K test examples with the same 43 columns as in the training data.</p>
rcv1-v2.topics.qrels.csv	Topic assignments for news articles in the RCV1-V2 Reuters news dataset. A news article can be assigned to several topics. The format of each row is "<topic name> <document id> 1". The dataset contains 2.6M topic assignments. The dataset was made available by David. D. Lewis.

[student_performance.txt](#)

This data comes from the KDD Cup 2010 Student performance evaluation challenge ([student performance evaluation](#)). The data used is the Algebra_2008_2009 training set (Stamper, J., Niculescu-Mizil, A., Ritter, S., Gordon, G.J., & Koedinger, K.R. (2010). Algebra I 2008-2009. Challenge dataset from KDD Cup 2010 Educational Data Mining Challenge. Find it at [downloads.jsp](#) or [algebra_2008_2009.zip](#).

The dataset was downloaded and stored in Azure Blob storage ([student_performance.txt](#)) and contains log files from a student tutoring system. The supplied features include problem ID and its brief description, student ID, timestamp, and how many attempts the student made before solving the problem in the right way. The original dataset has 8.9M records; this dataset has been down-sampled to the first 100K rows. The dataset has 23 tab-separated columns of various types: numeric, categorical, and timestamp.

Analyzing Customer Churn by using Azure Machine Learning

1/17/2017 • 12 min to read • [Edit on GitHub](#)

Overview

This article presents a reference implementation of a customer churn analysis project that is built by using Azure Machine Learning. In this article, we discuss associated generic models for holistically solving the problem of industrial customer churn. We also measure the accuracy of models that are built by using Machine Learning, and we assess directions for further development.

Acknowledgements

This experiment was developed and tested by Serge Berger, Principal Data Scientist at Microsoft, and Roger Barga, formerly Product Manager for Microsoft Azure Machine Learning. The Azure documentation team gratefully acknowledges their expertise and thanks them for sharing this white paper.

NOTE

The data used for this experiment is not publicly available. For an example of how to build a machine learning model for churn analysis, see: [Retail churn model template in Cortana Intelligence Gallery](#)

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

The problem of customer churn

Businesses in the consumer market and in all enterprise sectors have to deal with churn. Sometimes churn is excessive and influences policy decisions. The traditional solution is to predict high-propensity churners and address their needs via a concierge service, marketing campaigns, or by applying special dispensations. These approaches can vary from industry to industry and even from a particular consumer cluster to another within one industry (for example, telecommunications).

The common factor is that businesses need to minimize these special customer retention efforts. Thus, a natural methodology would be to score every customer with the probability of churn and address the top N ones. The top customers might be the most profitable ones; for example, in more sophisticated scenarios, a profit function is employed during the selection of candidates for special dispensation. However, these considerations are only a part of the holistic strategy for dealing with churn. Businesses also have to take into account risk (and associated risk tolerance), the level and cost of the intervention, and plausible customer segmentation.

Industry outlook and approaches

Sophisticated handling of churn is a sign of a mature industry. The classic example is the telecommunications industry where subscribers are known to frequently switch from one provider to another. This voluntary churn is a prime concern. Moreover, providers have accumulated significant knowledge about *churn drivers*, which are the factors that drive customers to switch.

For instance, handset or device choice is a well-known driver of churn in the mobile phone business. As a result, a popular policy is to subsidize the price of a handset for new subscribers and charging a full price to existing customers for an upgrade. Historically, this policy has led to customers hopping from one provider to another to get a new discount, which in turn, has prompted providers to refine their strategies.

High volatility in handset offerings is a factor that very quickly invalidates models of churn that are based on current handset models. Additionally, mobile phones are not only telecommunication devices; they are also fashion statements (consider the iPhone), and these social predictors are outside the scope of regular telecommunications data sets.

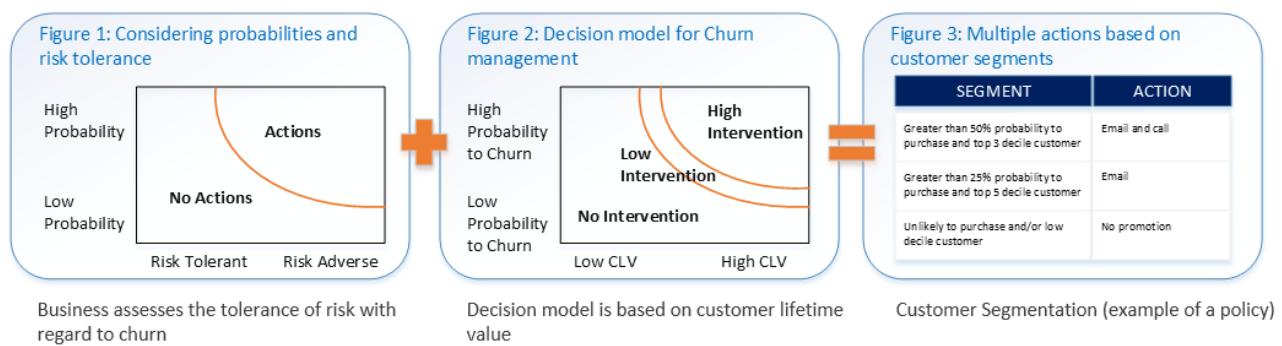
The net result for modeling is that you cannot devise a sound policy simply by eliminating known reasons for churn. In fact, a continuous modeling strategy, including classic models that quantify categorical variables (such as decision trees), is **mandatory**.

Using big data sets on their customers, organizations are performing big data analytics (in particular, churn detection based on big data) as an effective approach to the problem. You can find more about the big data approach to the problem of churn in the Recommendations on ETL section.

Methodology to model customer churn

A common problem-solving process to solve customer churn is depicted in Figures 1-3:

1. A risk model allows you to consider how actions affect probability and risk.
2. An intervention model allows you to consider how the level of intervention could affect the probability of churn and the amount of customer lifetime value (CLV).
3. This analysis lends itself to a qualitative analysis that is escalated to a proactive marketing campaign that targets customer segments to deliver the optimal offer.



This forward looking approach is the best way to treat churn, but it comes with complexity: we have to develop a multi-model archetype and trace dependencies between the models. The interaction among models can be encapsulated as shown in the following diagram:

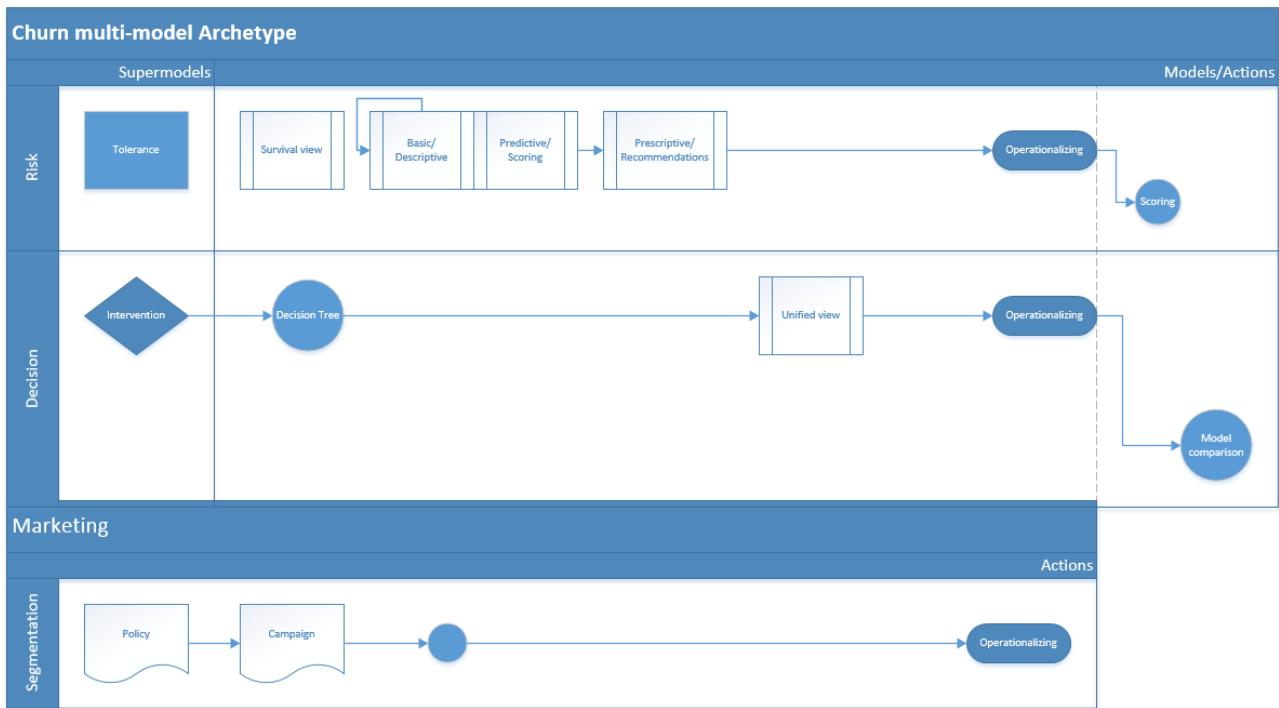


Figure 4: Unified multi-model archetype

Interaction between the models is key if we are to deliver a holistic approach to customer retention. Each model necessarily degrades over time; therefore, the architecture is an implicit loop (similar to the archetype set by the CRISP-DM data mining standard, [3]).

The overall cycle of risk-decision-marketing segmentation/decomposition is still a generalized structure, which is applicable to many business problems. Churn analysis is simply a strong representative of this group of problems because it exhibits all the traits of a complex business problem that does not allow a simplified predictive solution. The social aspects of the modern approach to churn are not particularly highlighted in the approach, but the social aspects are encapsulated in the modeling archetype, as they would be in any model.

An interesting addition here is big data analytics. Today's telecommunication and retail businesses collect exhaustive data about their customers, and we can easily foresee that the need for multi-model connectivity will become a common trend, given emerging trends such as the Internet of Things and ubiquitous devices, which allow business to employ smart solutions at multiple layers.

Implementing the modeling archetype in Machine Learning Studio

Given the problem just described, what is the best way to implement an integrated modeling and scoring approach? In this section, we will demonstrate how we accomplished this by using Azure Machine Learning Studio.

The multi-model approach is a must when designing a global archetype for churn. Even the scoring (predictive) part of the approach should be multi-model.

The following diagram shows the prototype we created, which employs four scoring algorithms in Machine Learning Studio to predict churn. The reason for using a multi-model approach is not only to create an ensemble classifier to increase accuracy, but also to protect against over-fitting and to improve prescriptive feature selection.

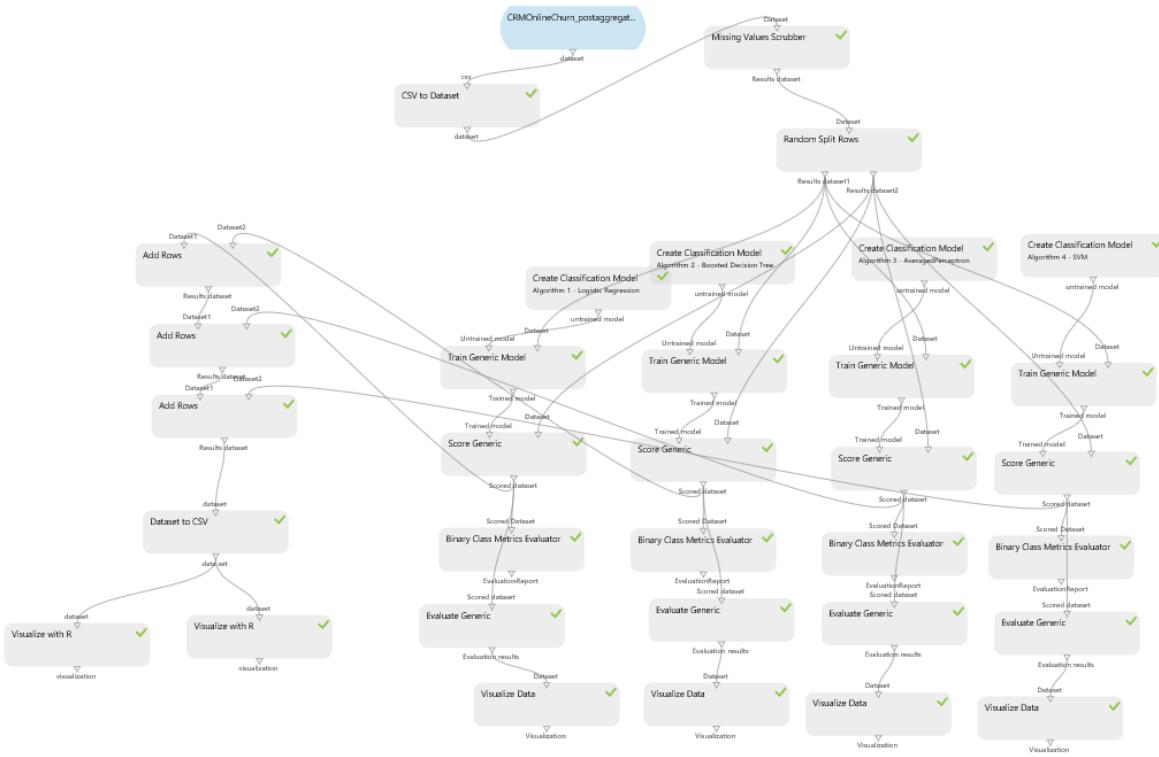


Figure 5: Prototype of a churn modeling approach

The following sections provide more details about the prototype scoring model that we implemented by using Machine Learning Studio.

Data selection and preparation

The data used to build the models and score customers was obtained from a CRM vertical solution, with the data obfuscated to protect customer privacy. The data contains information about 8,000 subscriptions in the U.S., and it combines three sources: provisioning data (subscription metadata), activity data (usage of the system), and customer support data. The data does not include any business related information about the customers; for example, it does not include loyalty metadata or credit scores.

For simplicity, ETL and data cleansing processes are out of scope because we assume that data preparation has already been done elsewhere.

Feature selection for modeling is based on preliminary significance scoring of the set of predictors, included in the process that uses the random forest module. For the implementation in Machine Learning Studio, we calculated the mean, median, and ranges for representative features. For example, we added aggregates for the qualitative data, such as minimum and maximum values for user activity.

We also captured temporal information for the most recent six months. We analyzed data for one year and we established that even if there were statistically significant trends, the effect on churn is greatly diminished after six months.

The most important point is that the entire process, including ETL, feature selection, and modeling was implemented in Machine Learning Studio, using data sources in Microsoft Azure.

The following diagrams illustrate the data that was used.

Churn Binary	GroupCustomerDBID	Mean(Ac)	Impute_I	IMP_Mean	Mean(Activ)	Impute_	IMP	Mean(Cases)	Imp	IMP_Mean(I)	Mean(Co)	Impute_
1	Commerce.3766384	2.833213	0	2.833213	7.628518	0	7.6285	0	0	0	4.430817	0
1	Commerce.3648614	2.197225	0	2.197225	5.147494	0	5.1475	0	0	0	3.401197	0
1	Commerce.1701750	4.682131	0	4.682131	7.451242	0	7.4512	2.197225	0	2.197225	5.056246	0
1	Commerce.1451565	4.787492	0	4.787492	7.960803	0	7.9608	0	0	0	5.398163	0
1	Commerce.2115489	3.433987	0	3.433987	3.7612	0	3.7612	1.791759	0	1.791759	3.091042	0
1	Commerce.6205107	3.988984	0	3.988984	7.467873	0	7.4679	0.693147	0	0.693147	4.89784	0
1	Commerce.7125701	4.356709	0	4.356709	4.077537	0	4.0775	0	0	0	6.603944	0
1	Commerce.2808747	0.693147	0	0.693147	7.427045	0	7.427	0	0	0	7.029973	0
0	Commerce.3213575	7.17012	0	7.17012	5.590987	0	5.591	0	0	0	7.284135	0

Figure 6: Excerpt of data source (obfuscated)

Columns (292/0)
Churn Binary
GroupCustomerDBID
Mean(Accounts) etc. (3/0)
Mean(Activities) etc. (3/0)
Mean(Cases) etc. (3/0)
Mean(Contacts) etc. (3/0)
Mean>EmailAttachments) etc. (3/0)
Mean(KBArticles) etc. (3/0)
Mean(Leads) etc. (3/0)
Mean(MarketingCampaigns) etc. (3/0)
Mean(MarketingLists) etc. (3/0)
Mean(Notes) etc. (3/0)
Mean(Orders) etc. (3/0)
Mean(Quotes) etc. (3/0)
Std Dev(Accounts) etc. (3/0)
Std Dev(Activities) etc. (3/0)
Std Dev(Cases) etc. (3/0)
Std Dev(Contacts) etc. (3/0)
Std Dev>EmailAttachments) etc. (3/0)
Std Dev(KBArticles) etc. (3/0)
Std Dev(Leads) etc. (3/0)
Std Dev(MarketingCampaigns) etc. (3/0)
Std Dev(MarketingLists) etc. (3/0)
Std Dev(Notes) etc. (3/0)
Std Dev(Orders) etc. (3/0)
Std Dev(Quotes) etc. (3/0)
Min(Accounts) etc. (3/0)
Min(Activities) etc. (3/0)
Min(Cases) etc. (3/0)
Min(Contacts) etc. (3/0)
Min>EmailAttachments) etc. (3/0)

Figure 7: Features extracted from data source

Note that this data is private and therefore the model and data cannot be shared. However, for a similar model using publicly available data, see this sample experiment in the [Cortana Intelligence Gallery: Telco Customer Churn](#).

To learn more about how you can implement a churn analysis model using Cortana Intelligence Suite, we also recommend [this video](#) by Senior Program Manager Wee Hyong Tok.

Algorithms used in the prototype

We used the following four machine learning algorithms to build the prototype (no customization):

1. Logistic regression (LR)
2. Boosted decision tree (BT)
3. Averaged perceptron (AP)
4. Support vector machine (SVM)

The following diagram illustrates a portion of the experiment design surface, which indicates the sequence in which the models were created:

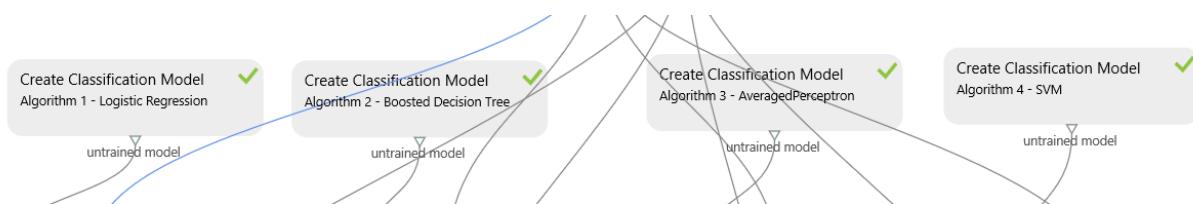


Figure 8: Creating models in Machine Learning Studio

Scoring methods

We scored the four models by using a labeled training dataset.

We also submitted the scoring dataset to a comparable model built by using the desktop edition of SAS Enterprise Miner 12. We measured the accuracy of the SAS model and all four Machine Learning Studio models.

Results

In this section, we present our findings about the accuracy of the models, based on the scoring dataset.

Accuracy and precision of scoring

Generally, the implementation in Azure Machine Learning is behind SAS in accuracy by about 10-15% (Area Under Curve or AUC).

However, the most important metric in churn is the misclassification rate: that is, of the top N churners as predicted by the classifier, which of them actually did **not** churn, and yet received special treatment? The following diagram compares this misclassification rate for all the models:

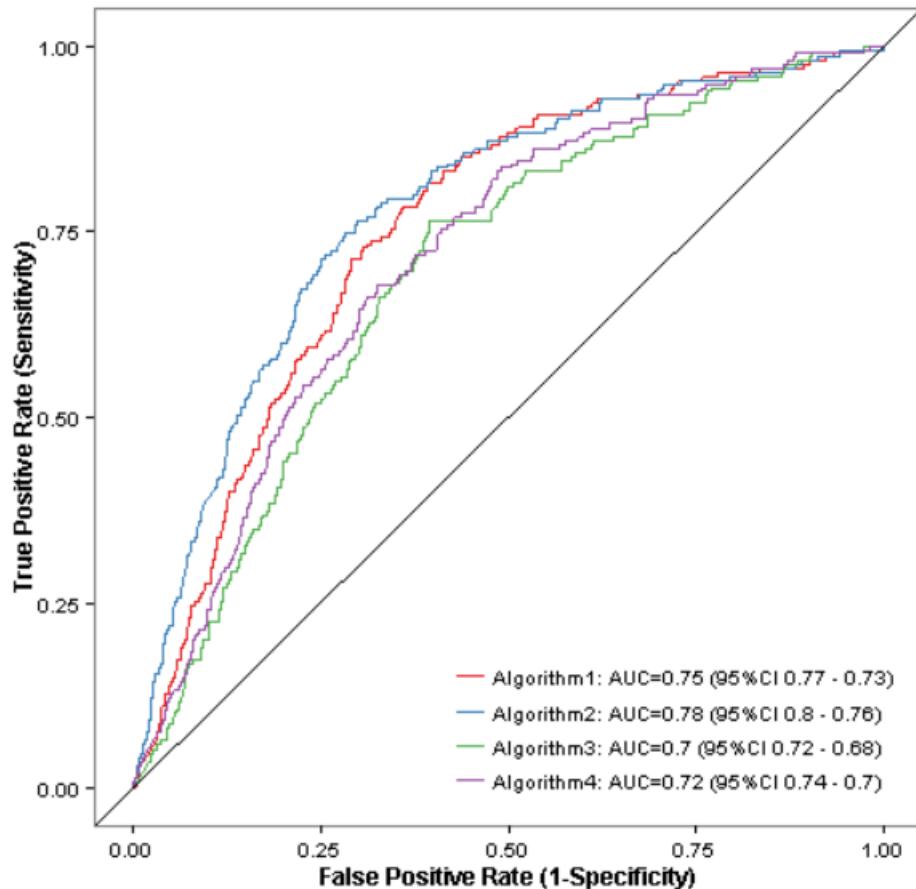


Figure 9: Passau prototype area under curve

Using AUC to compare results

Area Under Curve (AUC) is a metric that represents a global measure of *separability* between the distributions of scores for positive and negative populations. It is similar to the traditional Receiver Operator Characteristic (ROC) graph, but one important difference is that the AUC metric does not require you to choose a threshold value. Instead, it summarizes the results over **all** possible choices. In contrast, the traditional ROC graph shows the positive rate on the vertical axis and the false positive rate on the horizontal axis, and the classification threshold varies.

AUC is generally used as a measure of worth for different algorithms (or different systems) because it allows models to be compared by means of their AUC values. This is a popular approach in industries such as meteorology and biosciences. Thus, AUC represents a popular tool for assessing classifier performance.

Comparing misclassification rates

We compared the misclassification rates on the dataset in question by using the CRM data of approximately 8,000 subscriptions.

- The SAS misclassification rate was 10-15%.
- The Machine Learning Studio misclassification rate was 15-20% for the top 200-300 churners.

In the telecommunications industry, it is important to address only those customers who have the highest risk to churn by offering them a concierge service or other special treatment. In that respect, the Machine Learning Studio implementation achieves results on par with the SAS model.

By the same token, accuracy is more important than precision because we are mostly interested in correctly classifying potential churners.

The following diagram from Wikipedia depicts the relationship in a lively, easy-to-understand graphic:

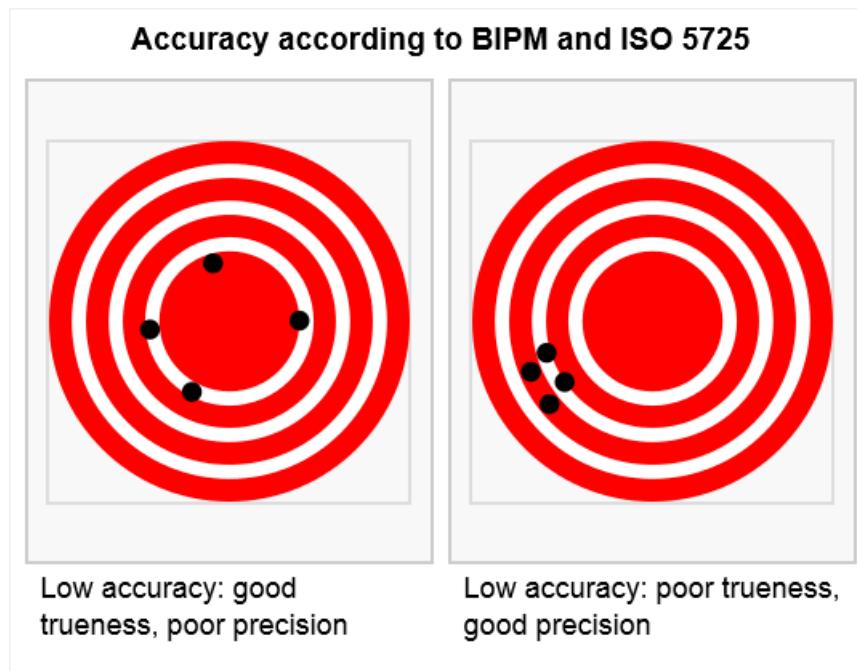


Figure 10: Tradeoff between accuracy and precision

Accuracy and precision results for boosted decision tree model

The following chart displays the raw results from scoring using the Machine Learning prototype for the boosted decision tree model, which happens to be the most accurate among the four models:

Accuracy	Precision	Recall	F-Score	AUC	Average Log Loss	Training Log Loss
0.895685	0.282258	0.179487	0.219436	0.778342	0.0306191	89.1761
0.895685	0.282258	0.179487	0.219436	0.778342	0.0306191	89.1761
0.895685	0.282258	0.179487	0.219436	0.778342	0.0306191	89.1761
0.895685	0.282258	0.179487	0.219436	0.778342	0.0306191	89.1761

Figure 11: Boosted decision tree model characteristics

Performance comparison

We compared the speed at which data was scored using the Machine Learning Studio models and a comparable model created by using the desktop edition of SAS Enterprise Miner 12.1.

The following table summarizes the performance of the algorithms:

Table 1. General performance (accuracy) of the algorithms

LR	BT	AP	SVM
Average Model	The Best Model	Underperforming	Average Model

The models hosted in Machine Learning Studio outperformed SAS by 15-25% for speed of execution, but accuracy was largely on par.

Discussion and recommendations

In the telecommunications industry, several practices have emerged to analyze churn, including:

- Derive metrics for four fundamental categories:
 - **Entity (for example, a subscription).** Provision basic information about the subscription and/or customer that is the subject of churn.
 - **Activity.** Obtain all possible usage information that is related to the entity, for example, the number of logins.
 - **Customer support.** Harvest information from customer support logs to indicate whether the subscription had issues or interactions with customer support.
 - **Competitive and business data.** Obtain any information possible about the customer (for example, can be unavailable or hard to track).
- Use importance to drive feature selection. This implies that the boosted decision tree model is always a promising approach.

The use of these four categories creates the illusion that a simple *deterministic* approach, based on indexes formed on reasonable factors per category, should suffice to identify customers at risk for churn. Unfortunately, although this notion seems plausible, it is a false understanding. The reason is that churn is a temporal effect and the factors contributing to churn are usually in transient states. What leads a customer to consider leaving today might be different tomorrow, and it certainly will be different six months from now. Therefore, a *probabilistic* model is a necessity.

This important observation is often overlooked in business, which generally prefers a business intelligence-oriented approach to analytics, mostly because it is an easier sell and admits straightforward automation.

However, the promise of self-service analytics by using Machine Learning Studio is that the four categories of information, graded by division or department, become a valuable source for machine learning about churn.

Another exciting capability coming in Azure Machine Learning is the ability to add a custom module to the repository of predefined modules that are already available. This capability, essentially, creates an opportunity to select libraries and create templates for vertical markets. It is an important differentiator of Azure Machine Learning in the market place.

We hope to continue this topic in the future, especially related to big data analytics.

Conclusion

This paper describes a sensible approach to tackling the common problem of customer churn by using a generic framework. We considered a prototype for scoring models and implemented it by using Azure Machine Learning. Finally, we assessed the accuracy and performance of the prototype solution with regard to comparable algorithms in SAS.

For more information:

Did this paper help you? Please give us your feedback. Tell us on a scale of 1 (poor) to 5 (excellent), how would you

rate this paper and why have you given it this rating? For example:

- Are you rating it high due to having good examples, excellent screen shots, clear writing, or another reason?
- Are you rating it low due to poor examples, fuzzy screen shots, or unclear writing?

This feedback will help us improve the quality of white papers we release.

[Send feedback.](#)

References

- 1 Predictive Analytics: Beyond the Predictions, W. McKnight, Information Management, July/August 2011, p.18-20.
- 2 Wikipedia article: [Accuracy and precision](#)
- [3] [CRISP-DM 1.0: Step-by-Step Data Mining Guide](#)
- [4] [Big Data Marketing: Engage Your Customers More Effectively and Drive Value](#)
- [5] [Telco churn model template in Cortana Intelligence Gallery](#)

Appendix



Microsoft Confidential | PPT Internal Use Only MICROSOFT IT | DATA & DECISION SCIENCES GROUP

Figure 12: Snapshot of a presentation on churn prototype

The Team Data Science Process in action: using HDInsight Hadoop clusters

1/17/2017 • 25 min to read • [Edit on GitHub](#)

In this walkthrough, we use the [Team Data Science Process \(TDSP\)](#) in an end-to-end scenario using an [Azure HDInsight Hadoop cluster](#) to store, explore and feature engineer data from the publicly available [NYC Taxi Trips](#) dataset, and to down sample the data. Models of the data are built with Azure Machine Learning to handle binary and multiclass classification and regression predictive tasks.

For a walkthrough that shows how to handle a larger (1 terabyte) dataset for a similar scenario using HDInsight Hadoop clusters for data processing, see [Team Data Science Process - Using Azure HDInsight Hadoop Clusters on a 1 TB dataset](#).

It is also possible to use an IPython notebook to accomplish the tasks presented the walkthrough using the 1 TB dataset. Users who would like to try this approach should consult the [Criteo walkthrough using a Hive ODBC connection](#) topic.

NYC Taxi Trips Dataset description

The NYC Taxi Trip data is about 20GB of compressed comma-separated values (CSV) files (~48GB uncompressed), comprising more than 173 million individual trips and the fares paid for each trip. Each trip record includes the pickup and drop-off location and time, anonymized hack (driver's) license number and medallion (taxi's unique id) number. The data covers all trips in the year 2013 and is provided in the following two datasets for each month:

1. The 'trip_data' CSV files contain trip details, such as number of passengers, pickup and dropoff points, trip duration, and trip length. Here are a few sample records:

```
medallion,hack_license,vendor_id,rate_code,store_and_fwd_flag,pickup_datetime,dropoff_datetime,passenger_count,trip_time_in_secs,trip_distance,pickup_longitude,pickup_latitude,dropoff_longitude,dropoff_latitude
89D227B655E5C82AECF13C3F540D4CF4,BA96DE419E711691B9445D6A6307C170,CMT,1,N,2013-01-01 15:11:48,2013-01-01 15:18:10,4.382,1.00,-73.978165,40.757977,-73.989838,40.751171
0BD7C8F5BA12B88E0B67BED28BEA73D8,9FD8F69F0804BDB5549F40E9DA1BE472,CMT,1,N,2013-01-06 00:18:35,2013-01-06 00:22:54,1,259,1.50,-74.006683,40.731781,-73.994499,40.75066
0BD7C8F5BA12B88E0B67BED28BEA73D8,9FD8F69F0804BDB5549F40E9DA1BE472,CMT,1,N,2013-01-05 18:49:41,2013-01-05 18:54:23,1,282,1.10,-74.004707,40.73777,-74.009834,40.726002
DFD2202EE08F7A8DC9A57B02ACB81FE2,51EE87E3205C985EF8431D850C786310,CMT,1,N,2013-01-07 23:54:15,2013-01-07 23:58:20,2,244,.70,-73.974602,40.759945,-73.984734,40.759388
DFD2202EE08F7A8DC9A57B02ACB81FE2,51EE87E3205C985EF8431D850C786310,CMT,1,N,2013-01-07 23:25:03,2013-01-07 23:34:24,1,560,2.10,-73.97625,40.748528,-74.002586,40.747868
```

2. The 'trip_fare' CSV files contain details of the fare paid for each trip, such as payment type, fare amount, surcharge and taxes, tips and tolls, and the total amount paid. Here are a few sample records:

```
medallion,hack_license,vendor_id,pickup_datetime,payment_type,fare_amount,surcharge,mta_tax,tip_amount,tolls_amount,total_amount
89D227B655E5C82AECF13C3F540D4CF4,BA96DE419E711691B9445D6A6307C170,CMT,2013-01-01 15:11:48,CSH,6.5,0,0.5,0,0,7
0BD7C8F5BA12B88E0B67BED28BEA73D8,9FD8F69F0804BDB5549F40E9DA1BE472,CMT,2013-01-06 00:18:35,CSH,6,0.5,0.5,0,0,7
0BD7C8F5BA12B88E0B67BED28BEA73D8,9FD8F69F0804BDB5549F40E9DA1BE472,CMT,2013-01-05 18:49:41,CSH,5.5,1,0.5,0,0,6
DFD2202EE08F7A8DC9A57B02ACB81FE2,51EE87E3205C985EF8431D850C786310,CMT,2013-01-07 23:54:15,CSH,5,0.5,0.5,0,0,6
DFD2202EE08F7A8DC9A57B02ACB81FE2,51EE87E3205C985EF8431D850C786310,CMT,2013-01-07 23:25:03,CSH,9.5,0.5,0.5,0,0,10.5
```

The unique key to join trip_data and trip_fare is composed of the fields: medallion, hack_licence and pickup_datetime.

To get all of the details relevant to a particular trip, it is sufficient to join with three keys: the "medallion", "hack_license" and "pickup_datetime".

We describe some more details of the data when we store them into Hive tables shortly.

Examples of prediction tasks

When approaching data, determining the kind of predictions you want to make based on its analysis helps clarify the tasks that you will need to include in your process. Here are three examples of prediction problems that we address in this walkthrough whose formulation is based on the *tip_amount*:

1. **Binary classification:** Predict whether or not a tip was paid for a trip, i.e. a *tip_amount* that is greater than \$0 is a positive example, while a *tip_amount* of \$0 is a negative example.

```
Class 0 : tip_amount = $0
Class 1 : tip_amount > $0
```

2. **Multiclass classification:** To predict the range of tip amounts paid for the trip. We divide the *tip_amount* into five bins or classes:

```
Class 0 : tip_amount = $0
Class 1 : tip_amount > $0 and tip_amount <= $5
Class 2 : tip_amount > $5 and tip_amount <= $10
Class 3 : tip_amount > $10 and tip_amount <= $20
Class 4 : tip_amount > $20
```

3. **Regression task:** To predict the amount of the tip paid for a trip.

Set up an HDInsight Hadoop cluster for advanced analytics

NOTE

This is typically an **Admin** task.

You can set up an Azure environment for advanced analytics that employs an HDInsight cluster in three steps:

1. [Create a storage account](#): This storage account is used for storing data in Azure Blob Storage. The data used in HDInsight clusters also resides here.
2. [Customize Azure HDInsight Hadoop clusters for the Advanced Analytics Process and Technology](#). This step creates an Azure HDInsight Hadoop cluster with 64-bit Anaconda Python 2.7 installed on all nodes. There are two important steps to remember while customizing your HDInsight cluster.
 - Remember to link the storage account created in step 1 with your HDInsight cluster when creating it. This storage account is used to access data that is processed within the cluster.
 - After the cluster is created, enable Remote Access to the head node of the cluster. Navigate to the **Configuration** tab and click **Enable Remote**. This step specifies the user credentials used for remote login.
3. [Create an Azure Machine Learning workspace](#): This Azure Machine Learning workspace is used to build machine learning models. This task is addressed after completing an initial data exploration and down sampling using the HDInsight cluster.

Get the data from a public source

NOTE

This is typically an **Admin** task.

To get the [NYC Taxi Trips](#) dataset from its public location, you may use any of the methods described in [Move Data to and from Azure Blob Storage](#) to copy the data to your machine.

Here we describe how use AzCopy to transfer the files containing data. To download and install AzCopy follow the instructions at [Getting Started with the AzCopy Command-Line Utility](#).

1. From a Command Prompt window, issue the following AzCopy commands, replacing with the desired destination:

```
"C:\Program Files (x86)\Microsoft SDKs\Azure\AzCopy\azcopy" /Source:https://nyctaxitrips.blob.core.windows.net/data /Dest:<path_to_data_folder> /S
```

2. When the copy completes, a total of 24 zipped files are in the data folder chosen. Unzip the downloaded files to the same directory on your local machine. Make a note of the folder where the uncompressed files reside. This folder will be referred to as the is what follows.

Upload the data to the default container of Azure HDInsight Hadoop cluster

NOTE

This is typically an **Admin** task.

In the following AzCopy commands, replace the following parameters with the actual values that you specified when creating the Hadoop cluster and unzipping the data files.

- **<path_to_data_folder>** the directory (along with path) on your machine that contain the unzipped data files
- **<storage account name of Hadoop cluster>** the storage account associated with your HDInsight cluster
- **<default container of Hadoop cluster>** the default container used by your cluster. Note that the name of the default container is usually the same name as the cluster itself. For example, if the cluster is called "abc123.azurehdinsight.net", the default container is abc123.
- **<storage account key>** the key for the storage account used by your cluster

From a Command Prompt or a Windows PowerShell window in your machine, run the following two AzCopy commands.

This command uploads the trip data to **nyctaxitripraw** directory in the default container of the Hadoop cluster.

```
"C:\Program Files (x86)\Microsoft SDKs\Azure\AzCopy\azcopy" /Source:<path_to_unzipped_data_files> /Dest:https://<storage account name of Hadoop cluster>.blob.core.windows.net/<default container of Hadoop cluster>/nyctaxitripraw /DestKey:<storage account key> /S /Pattern:trip_data_*.csv
```

This command uploads the fare data to **nyctaxifareraw** directory in the default container of the Hadoop cluster.

```
"C:\Program Files (x86)\Microsoft SDKs\Azure\AzCopy\azcopy" /Source:<path_to_unzipped_data_files> /Dest:https://<storage account name of Hadoop cluster>.blob.core.windows.net/<default container of Hadoop cluster>/nyctaxifareraw /DestKey:<storage account key> /S /Pattern:trip_fare_*.csv
```

The data should now in Azure Blob Storage and ready to be consumed within the HDInsight cluster.

Log into the head node of Hadoop cluster and and prepare for exploratory data analysis

NOTE

This is typically an **Admin** task.

To access the head node of the cluster for exploratory data analysis and down sampling of the data, follow the procedure outlined in [Access the Head Node of Hadoop Cluster](#).

In this walkthrough, we primarily use queries written in [Hive](#), a SQL-like query language, to perform preliminary data explorations. The Hive queries are stored in .hql files. We then down sample this data to be used within Azure Machine Learning for building models.

To prepare the cluster for exploratory data analysis, we download the .hql files containing the relevant Hive scripts from [github](#) to a local directory (C:\temp) on the head node. To do this, open the **Command Prompt** from within the head node of the cluster and issue the following two commands:

```
set script=https://raw.githubusercontent.com/Azure/Azure-MachineLearning-  
DataScience/master/Misc/DataScienceProcess/DataScienceScripts/Download_DataScience_Scripts.ps1  
  
@powershell -NoProfile -ExecutionPolicy unrestricted -Command "iex ((new-object net.webclient).DownloadString(%script%))"
```

These two commands will download all .hql files needed in this walkthrough to the local directory **C:\temp** in the head node.

Create Hive database and tables partitioned by month

NOTE

This is typically an **Admin** task.

We are now ready to create Hive tables for our NYC taxi dataset. In the head node of the Hadoop cluster, open the **Hadoop Command Line** on the desktop of the head node, and enter the Hive directory by entering the command

```
cd %hive_home%\bin
```

NOTE

Run all Hive commands in this walkthrough from the above Hive bin/ directory prompt. This will take care of any path issues automatically. We use the terms "Hive directory prompt", "Hive bin/ directory prompt", and "Hadoop Command Line" interchangeably in this walkthrough.

From the Hive directory prompt, enter the following command in Hadoop Command Line of the head node to submit the Hive query to create Hive database and tables:

```
hive -f "C:\temp\sample_hive_create_db_and_tables.hql"
```

Here is the content of the **C:\temp\sample_hive_create_db_and_tables.hql** file which creates Hive database **nyctaxidb** and tables **trip** and **fare**.

```

create database if not exists nyctaxidb;

create external table if not exists nyctaxidb.trip
(
    medallion string,
    hack_license string,
    vendor_id string,
    rate_code string,
    store_and_fwd_flag string,
    pickup_datetime string,
    dropoff_datetime string,
    passenger_count int,
    trip_time_in_secs double,
    trip_distance double,
    pickup_longitude double,
    pickup_latitude double,
    dropoff_longitude double,
    dropoff_latitude double)
PARTITIONED BY (month int)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' lines terminated by '\n'
STORED AS TEXTFILE LOCATION 'wasb://nyctaxidbdata/trip' TBLPROPERTIES('skip.header.line.count'=1');

create external table if not exists nyctaxidb.fare
(
    medallion string,
    hack_license string,
    vendor_id string,
    pickup_datetime string,
    payment_type string,
    fare_amount double,
    surcharge double,
    mta_tax double,
    tip_amount double,
    tolls_amount double,
    total_amount double)
PARTITIONED BY (month int)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' lines terminated by '\n'
STORED AS TEXTFILE LOCATION 'wasb://nyctaxidbdata/fare' TBLPROPERTIES('skip.header.line.count'=1');

```

This Hive script creates two tables:

- the "trip" table contains trip details of each ride (driver details, pickup time, trip distance and times)
- the "fare" table contains fare details (fare amount, tip amount, tolls and surcharges).

If you need any additional assistance with these procedures or want to investigate alternative ones, see the section [Submit Hive queries directly from the Hadoop Command Line](#).

Load Data to Hive tables by partitions

NOTE

This is typically an **Admin** task.

The NYC taxi dataset has a natural partitioning by month, which we use to enable faster processing and query times. The PowerShell commands below (issued from the Hive directory using the **Hadoop Command Line**) load data to the "trip" and "fare" Hive tables partitioned by month.

```
for /L %i IN (1,1,12) DO (hive -hiveconf MONTH=%i -f "C:\temp\sample_hive_load_data_by_partitions.hql")
```

The *sample_hive_load_data_by_partitions.hql* file contains the following **LOAD** commands.

```
LOAD DATA INPATH 'wasb://nyctaxitripraw/trip_data_${hiveconf:MONTH}.csv' INTO TABLE nyctaxidb.trip PARTITION  
(month=${hiveconf:MONTH});  
LOAD DATA INPATH 'wasb://nyctaxifareraw/trip_fare_${hiveconf:MONTH}.csv' INTO TABLE nyctaxidb.fare PARTITION  
(month=${hiveconf:MONTH});
```

Note that a number of Hive queries we use here in the exploration process involve looking at just a single partition or at only a couple of partitions. But these queries could be run across the entire data.

Show databases in the HDInsight Hadoop cluster

To show the databases created in HDInsight Hadoop cluster inside the Hadoop Command Line window, run the following command in Hadoop Command Line:

```
hive -e "show databases;"
```

Show the Hive tables in the nyctaxidb database

To show the tables in the nyctaxidb database, run the following command in Hadoop Command Line:

```
hive -e "show tables in nyctaxidb;"
```

We can confirm that the tables are partitioned by issuing the command below:

```
hive -e "show partitions nyctaxidb.trip;"
```

The expected output is shown below:

```
month=1  
month=10  
month=11  
month=12  
month=2  
month=3  
month=4  
month=5  
month=6  
month=7  
month=8  
month=9  
Time taken: 2.075 seconds, Fetched: 12 row(s)
```

Similarly, we can ensure that the fare table is partitioned by issuing the command below:

```
hive -e "show partitions nyctaxidb.fare;"
```

The expected output is shown below:

```
month=1
month=10
month=11
month=12
month=2
month=3
month=4
month=5
month=6
month=7
month=8
month=9
Time taken: 1.887 seconds, Fetched: 12 row(s)
```

Data exploration and feature engineering in Hive

NOTE

This is typically a **Data Scientist** task.

The data exploration and feature engineering tasks for the data loaded into the Hive tables can be accomplished using Hive queries. Here are examples of such tasks that we walk you through in this section:

- View the top 10 records in both tables.
- Explore data distributions of a few fields in varying time windows.
- Investigate data quality of the longitude and latitude fields.
- Generate binary and multiclass classification labels based on the **tip_amount**.
- Generate features by computing the direct trip distances.

Exploration: View the top 10 records in table trip

NOTE

This is typically a **Data Scientist** task.

To see what the data looks like, we examine 10 records from each table. Run the following two queries separately from the Hive directory prompt in the Hadoop Command Line console to inspect the records.

To get the top 10 records in the table "trip" from the first month:

```
hive -e "select * from nyctaxidb.trip where month=1 limit 10;"
```

To get the top 10 records in the table "fare" from the first month:

```
hive -e "select * from nyctaxidb.fare where month=1 limit 10;"
```

It is often useful to save the records to a file for convenient viewing. A small change to the above query accomplishes this:

```
hive -e "select * from nyctaxidb.fare where month=1 limit 10;" > C:\temp\testoutput
```

Exploration: View the number of records in each of the 12 partitions

NOTE

This is typically a **Data Scientist** task.

Of interest is the how the number of trips varies during the calendar year. Grouping by month allows us to see what this distribution of trips looks like.

```
hive -e "select month, count(*) from nyctaxidb.trip group by month;"
```

This gives us the output :

```
1 14776615  
2 13990176  
3 15749228  
4 15100468  
5 15285049  
6 14385456  
7 13823840  
8 12597109  
9 14107693  
10 15004556  
11 14388451  
12 13971118  
Time taken: 283.406 seconds, Fetched: 12 row(s)
```

Here, the first column is the month and the second is the number of trips for that month.

We can also count the total number of records in our trip data set by issuing the following command at the Hive directory prompt.

```
hive -e "select count(*) from nyctaxidb.trip;"
```

This yields:

```
173179759  
Time taken: 284.017 seconds, Fetched: 1 row(s)
```

Using commands similar to those shown for the trip data set, we can issue Hive queries from the Hive directory prompt for the fare data set to validate the number of records.

```
hive -e "select month, count(*) from nyctaxidb.fare group by month;"
```

This gives us the output:

```
1 14776615  
2 13990176  
3 15749228  
4 15100468  
5 15285049  
6 14385456  
7 13823840  
8 12597109  
9 14107693  
10 15004556  
11 14388451  
12 13971118  
Time taken: 253.955 seconds, Fetched: 12 row(s)
```

Note that the exact same number of trips per month is returned for both data sets. This provides the first validation that the data has been loaded correctly.

Counting the total number of records in the fare data set can be done using the command below from the Hive directory prompt:

```
hive -e "select count(*) from nyctaxidb.fare;"
```

This yields :

```
173179759  
Time taken: 186.683 seconds, Fetched: 1 row(s)
```

The total number of records in both tables is also the same. This provides a second validation that the data has been loaded correctly.

Exploration: Trip distribution by medallion

NOTE

This is typically a **Data Scientist** task.

This example identifies the medallion (taxi numbers) with more than 100 trips within a given time period. The query benefits from the partitioned table access since it is conditioned by the partition variable **month**. The query results are written to a local file queryoutput.tsv in **C:\temp** on the head node.

```
hive -f "C:\temp\sample_hive_trip_count_by_medallion.hql" > C:\temp\queryoutput.tsv
```

Here is the content of *sample_hive_trip_count_by_medallion.hql* file for inspection.

```
SELECT medallion, COUNT(*) as med_count  
FROM nyctaxidb.fare  
WHERE month<=3  
GROUP BY medallion  
HAVING med_count > 100  
ORDER BY med_count desc;
```

The medallion in the NYC taxi data set identifies a unique cab. We can identify which cabs are "busy" by asking which ones made more than a certain number of trips in a particular time period. The following example identifies cabs that made more than a hundred trips in the first three months, and saves the query results to a local file, C:\temp\queryoutput.tsv.

Here is the content of *sample_hive_trip_count_by_medallion.hql* file for inspection.

```
SELECT medallion, COUNT(*) as med_count  
FROM nyctaxidb.fare  
WHERE month<=3  
GROUP BY medallion  
HAVING med_count > 100  
ORDER BY med_count desc;
```

From the Hive directory prompt, issue the command below :

```
hive -f "C:\temp\sample_hive_trip_count_by_medallion.hql" > C:\temp\queryoutput.tsv
```

Exploration: Trip distribution by medallion and hack_license

NOTE

This is typically a **Data Scientist** task.

When exploring a dataset, we frequently want to examine the number of co-occurrences of groups of values. This section provide an example of how to do this for cabs and drivers.

The *sample_hive_trip_count_by_medallion_license.hql* file groups the fare data set on "medallion" and "hack_license" and returns counts of each combination. Below are its contents.

```
SELECT medallion, hack_license, COUNT(*) as trip_count
FROM nyctaxidb.fare
WHERE month=1
GROUP BY medallion, hack_license
HAVING trip_count > 100
ORDER BY trip_count desc;
```

This query returns cab and particular driver combinations ordered by descending number of trips.

From the Hive directory prompt, run :

```
hive -f "C:\temp\sample_hive_trip_count_by_medallion_license.hql" > C:\temp\queryoutput.tsv
```

The query results are written to a local file C:\temp\queryoutput.tsv.

Exploration: Assessing data quality by checking for invalid longitude/latitude records

NOTE

This is typically a **Data Scientist** task.

A common objective of exploratory data analysis is to weed out invalid or bad records. The example in this section determines whether either the longitude or latitude fields contain a value far outside the NYC area. Since it is likely that such records have an erroneous longitude-latitude values, we want to eliminate them from any data that is to be used for modeling.

Here is the content of *sample_hive_quality_assessment.hql* file for inspection.

```
SELECT COUNT(*) FROM nyctaxidb.trip
WHERE month=1
AND (CAST(pickup_longitude AS float) NOT BETWEEN -90 AND -30
OR CAST(pickup_latitude AS float) NOT BETWEEN 30 AND 90
OR CAST(dropoff_longitude AS float) NOT BETWEEN -90 AND -30
OR CAST(dropoff_latitude AS float) NOT BETWEEN 30 AND 90);
```

From the Hive directory prompt, run :

```
hive -S -f "C:\temp\sample_hive_quality_assessment.hql"
```

The *-S* argument included in this command suppresses the status screen printout of the Hive Map/Reduce jobs. This is useful because it makes the screen print of the Hive query output more readable.

Exploration: Binary class distributions of trip tips

NOTE

This is typically a **Data Scientist** task.

For the binary classification problem outlined in the [Examples of prediction tasks](#) section, it is useful to know whether a tip was given or not. This distribution of tips is binary:

- tip given(Class 1, tip_amount > \$0)
- no tip (Class 0, tip_amount = \$0).

The *sample_hive_tipped_frequencies.hql* file shown below does this.

```
SELECT tipped, COUNT(*) AS tip_freq
FROM
(
  SELECT if(tip_amount > 0, 1, 0) as tipped, tip_amount
  FROM nyctaxidb.fare
)tc
GROUP BY tipped;
```

From the Hive directory prompt, run:

```
hive -f "C:\temp\sample_hive_tipped_frequencies.hql"
```

Exploration: Class distributions in the multiclass setting

NOTE

This is typically a **Data Scientist** task.

For the multiclass classification problem outlined in the [Examples of prediction tasks](#) section this data set also lends itself to a natural classification where we would like to predict the amount of the tips given. We can use bins to define tip ranges in the query. To get the class distributions for the various tip ranges, we use the *sample_hive_tip_range_frequencies.hql* file. Below are its contents.

```
SELECT tip_class, COUNT(*) AS tip_freq
FROM
(
  SELECT if(tip_amount=0, 0,
           if(tip_amount>0 and tip_amount<=5, 1,
              if(tip_amount>5 and tip_amount<=10, 2,
                 if(tip_amount>10 and tip_amount<=20, 3, 4))) as tip_class, tip_amount
  FROM nyctaxidb.fare
)tc
GROUP BY tip_class;
```

Run the following command from Hadoop Command Line console:

```
hive -f "C:\temp\sample_hive_tip_range_frequencies.hql"
```

Exploration: Compute Direct Distance Between Two Longitude-Latitude Locations

NOTE

This is typically a **Data Scientist** task.

Having a measure of the direct distance allows us to find out the discrepancy between it and the actual trip distance. We motivate this feature by pointing out that a passenger might be less likely to tip if they figure out that the driver has intentionally taken them by a much longer route.

To see the comparison between actual trip distance and the [Haversine distance](#) between two longitude-latitude points (the "great circle" distance), we use the trigonometric functions available within Hive, thus :

```
set R=3959;
set pi=radians(180);

insert overwrite directory 'wasb://queryoutputdir'

select pickup_longitude, pickup_latitude, dropoff_longitude, dropoff_latitude, trip_distance, trip_time_in_secs,
${hiveconf:R}*2*2*atan((1-sqrt(1-pow(sin((dropoff_latitude-pickup_latitude)*${hiveconf:pi}/180)
*${hiveconf:pi}/180/2),2)-cos(pickup_latitude*${hiveconf:pi}/180)
*cos(dropoff_latitude*${hiveconf:pi}/180)*pow(sin((dropoff_longitude-pickup_longitude)*${hiveconf:pi}/180/2),2)))
/sqrt(pow(sin((dropoff_latitude-pickup_latitude)*${hiveconf:pi}/180/2),2)
+cos(pickup_latitude*${hiveconf:pi}/180)*cos(dropoff_latitude*${hiveconf:pi}/180)*
pow(sin((dropoff_longitude-pickup_longitude)*${hiveconf:pi}/180/2),2))) as direct_distance
from nyctaxidb.trip
where month=
and pickup_longitude between -90 and -30
and pickup_latitude between 30 and 90
and dropoff_longitude between -90 and -30
and dropoff_latitude between 30 and 90;
```

In the query above, R is the radius of the Earth in miles, and pi is converted to radians. Note that the longitude-latitude points are "filtered" to remove values that are far from the NYC area.

In this case, we write our results to a directory called "queryoutputdir". The sequence of commands shown below first creates this output directory, and then runs the Hive command.

From the Hive directory prompt, run:

```
hdfs dfs -mkdir wasb://queryoutputdir
hive -f "C:\temp\sample_hive_trip_direct_distance.hql"
```

The query results are written to 9 Azure blobs **queryoutputdir/000000_0** to **queryoutputdir/000008_0** under the default container of the Hadoop cluster.

To see the size of the individual blobs, we run the following command from the Hive directory prompt :

```
hdfs dfs -ls wasb://queryoutputdir
```

To see the contents of a given file, say 000000_0, we use Hadoop's `copyToLocal` command, thus.

```
hdfs dfs -copyToLocal wasb://queryoutputdir/000000_0 C:\temp\tempfile
```

WARNING

`copyToLocal` can be very slow for large files, and is not recommended for use with them.

A key advantage of having this data reside in an Azure blob is that we may explore the data within Azure Machine Learning using the [Import Data](#) module.

Down sample data and build models in Azure Machine Learning

NOTE

This is typically a **Data Scientist** task.

After the exploratory data analysis phase, we are now ready to down sample the data for building models in Azure Machine Learning. In this section, we show how to use a Hive query to down sample the data, which is then accessed from the [Import Data](#) module in Azure Machine Learning.

Down sampling the data

There are two steps in this procedure. First we join the **nyctaxidb.trip** and **nyctaxidb.fare** tables on three keys that are present in all records : "medallion", "hack_license", and "pickup_datetime". We then generate a binary classification label **tipped** and a multi-class classification label **tip_class**.

To be able to use the down sampled data directly from the [Import Data](#) module in Azure Machine Learning, it is necessary to store the results of the above query to an internal Hive table. In what follows, we create an internal Hive table and populate its contents with the joined and down sampled data.

The query applies standard Hive functions directly to generate the hour of day, week of year, weekday (1 stands for Monday, and 7 stands for Sunday) from the "pickup_datetime" field, and the direct distance between the pickup and dropoff locations. Users can refer to [LanguageManual UDF](#) for a complete list of such functions.

The query then down samples the data so that the query results can fit into the Azure Machine Learning Studio. Only about 1% of the original dataset is imported into the Studio.

Below are the contents of *sample_hive_prepare_for_aml_full.hql* file that prepares data for model building in Azure Machine Learning.

```
set R=3959;
set pi=radians(180);

create table if not exists nyctaxidb.nyctaxi_downsampled_dataset (
    medallion string,
    hack_license string,
    vendor_id string,
    rate_code string,
    store_and_fwd_flag string,
    pickup_datetime string,
    dropoff_datetime string,
    pickup_hour string,
    pickup_week string,
    weekday string,
    passenger_count int,
    trip_time_in_secs double,
    trip_distance double,
    pickup_longitude double,
    pickup_latitude double,
    dropoff_longitude double,
    dropoff_latitude double,
    direct_distance double,
    payment_type string,
    fare_amount double,
    surcharge double,
    mta_tax double,
    tip_amount double,
    tolls_amount double,
    total_amount double,
    tipped string,
    tip_class string
)
row format delimited fields terminated by ','
lines terminated by '\n'
```

```
stored as textfile;
```

```
-- now insert contents of the join into the above internal table
```

```
insert overwrite table nyctaxidb.nyctaxi_downsampled_dataset
select
t.medallion,
t.hack_license,
t.vendor_id,
t.rate_code,
t.store_and_fwd_flag,
t.pickup_datetime,
t.dropoff_datetime,
hour(t.pickup_datetime) as pickup_hour,
weekofyear(t.pickup_datetime) as pickup_week,
from_unixtime(unix_timestamp(t.pickup_datetime, 'yyyy-MM-dd HH:mm:ss'), 'u') as weekday,
t.passenger_count,
t.trip_time_in_secs,
t.trip_distance,
t.pickup_longitude,
t.pickup_latitude,
t.dropoff_longitude,
t.dropoff_latitude,
t.direct_distance,
f.payment_type,
f.fare_amount,
f.surcharge,
f.mta_tax,
f.tip_amount,
f.tolls_amount,
f.total_amount,
if(tip_amount>0,1,0) as tipped,
if(tip_amount=0,0,
if(tip_amount>0 and tip_amount<=5,1,
if(tip_amount>5 and tip_amount<=10,2,
if(tip_amount>10 and tip_amount<=20,3,4))) as tip_class

from
(
select
medallion,
hack_license,
vendor_id,
rate_code,
store_and_fwd_flag,
pickup_datetime,
dropoff_datetime,
passenger_count,
trip_time_in_secs,
trip_distance,
pickup_longitude,
pickup_latitude,
dropoff_longitude,
dropoff_latitude,
${hiveconf:R}*2*2*atan((1-sqrt(1-pow(sin((dropoff_latitude-pickup_latitude)*
*${hiveconf:pi}/180/2),2)-cos(pickup_latitude*${hiveconf:pi}/180)
*cos(dropoff_latitude*${hiveconf:pi}/180)*pow(sin((dropoff_longitude-pickup_longitude)*${hiveconf:pi}/180/2),2)))
/sqrt(pow(sin((dropoff_latitude-pickup_latitude)*${hiveconf:pi}/180/2),2)
+cos(pickup_latitude*${hiveconf:pi}/180)*cos(dropoff_latitude*${hiveconf:pi}/180)*pow(sin((dropoff_longitude-
pickup_longitude)*${hiveconf:pi}/180/2),2))) as direct_distance,
rand() as sample_key

from nyctaxidb.trip
where pickup_latitude between 30 and 90
and pickup_longitude between -90 and -30
and dropoff_latitude between 30 and 90
and dropoff_longitude between -90 and -30
)t
join
```

```

(
select
medallion,
hack_license,
vendor_id,
pickup_datetime,
payment_type,
fare_amount,
surcharge,
mta_tax,
tip_amount,
tolls_amount,
total_amount
from nyctaxidb.fare
)f
on t.medallion=f.medallion and t.hack_license=f.hack_license and t.pickup_datetime=f.pickup_datetime
where t.sample_key<=0.01

```

To run this query, from the Hive directory prompt :

```
hive -f "C:\temp\sample_hive_prepare_for_aml_full.hql"
```

We now have an internal table "nyctaxidb.nyctaxi_downsampled_dataset" which can be accessed using the [Import Data](#) module from Azure Machine Learning. Furthermore, we may use this dataset for building Machine Learning models.

Use the Import Data module in Azure Machine Learning to access the down sampled data

As prerequisites for issuing Hive queries in the [Import Data](#) module of Azure Machine Learning, we need access to an Azure Machine Learning workspace and access to the credentials of the cluster and its associated storage account.

Some details on the [Import Data](#) module and the parameters to input :

HCatalog server URI: If the cluster name is abc123, then this is simply : <https://abc123.azurehdinsight.net>

Hadoop user account name : The user name chosen for the cluster (**not** the remote access user name)

Hadoop ser account password : The password chosen for the cluster (**not** the remote access password)

Location of output data : This is chosen to be Azure.

Azure storage account name : Name of the default storage account associated with the cluster.

Azure container name : This is the default container name for the cluster, and is typically the same as the cluster name. For a cluster called "abc123", this is just abc123.

IMPORTANT

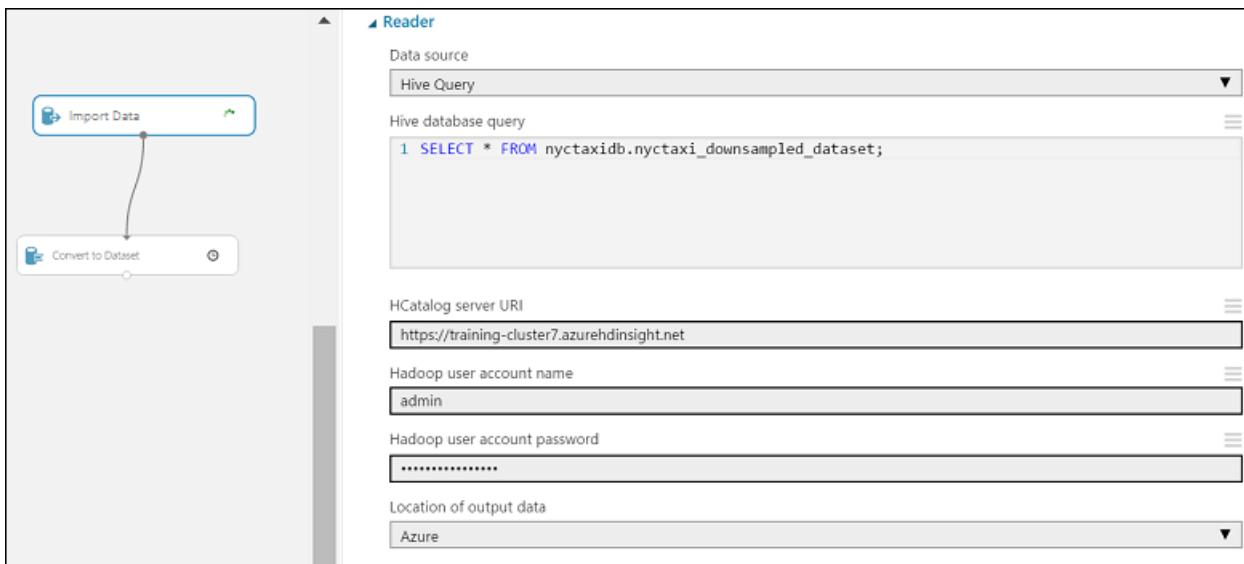
Any table we wish to query using the Import Data module in Azure Machine Learning must be an internal table. A tip for determining if a table T in a database D.db is an internal table is as follows.

From the Hive directory prompt, issue the command :

```
hdfs dfs -ls wasb:///D.db/T
```

If the table is an internal table and it is populated, its contents must show here. Another way to determine whether a table is an internal table is to use the Azure Storage Explorer. Use it to navigate to the default container name of the cluster, and then filter by the table name. If the table and its contents show up, this confirms that it is an internal table.

Here is a snapshot of the Hive query and the [Import Data](#) module:



Note that since our down sampled data resides in the default container, the resulting Hive query from Azure Machine Learning is very simple and is just a "SELECT * FROM nyctaxidb.nyctaxi_downsampled_data".

The dataset may now be used as the starting point for building Machine Learning models.

Build models in Azure Machine Learning

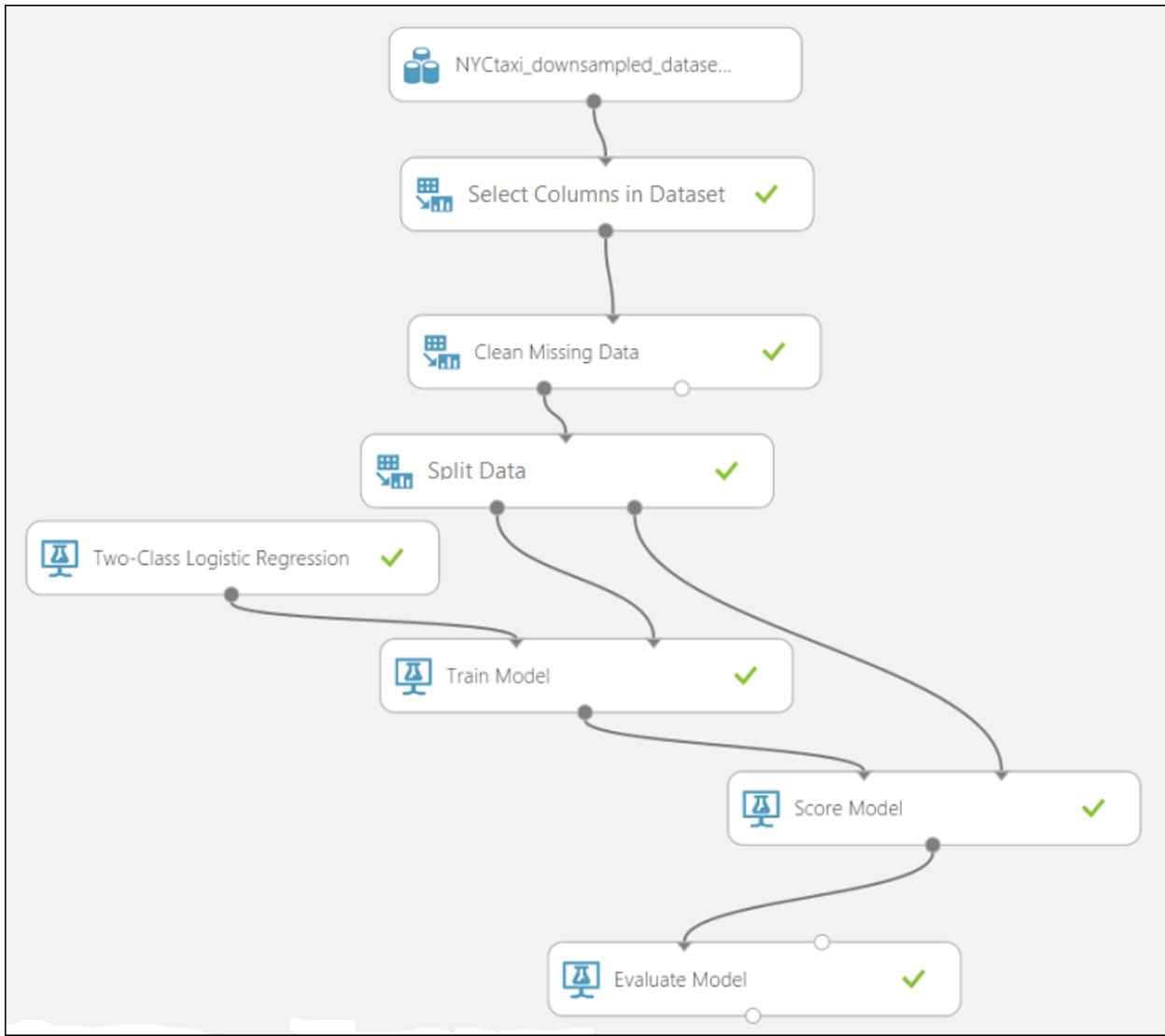
We are now able to proceed to model building and model deployment in [Azure Machine Learning](#). The data is ready for us to use in addressing the prediction problems identified above:

1. Binary classification: To predict whether or not a tip was paid for a trip.

Learner used: Two-class logistic regression

a. For this problem, our target (or class) label is "tipped". Our original down-sampled dataset has a few columns that are target leaks for this classification experiment. In particular : tip_class, tip_amount, and total_amount reveal information about the target label that is not available at testing time. We remove these columns from consideration using the [Select Columns in Dataset](#) module.

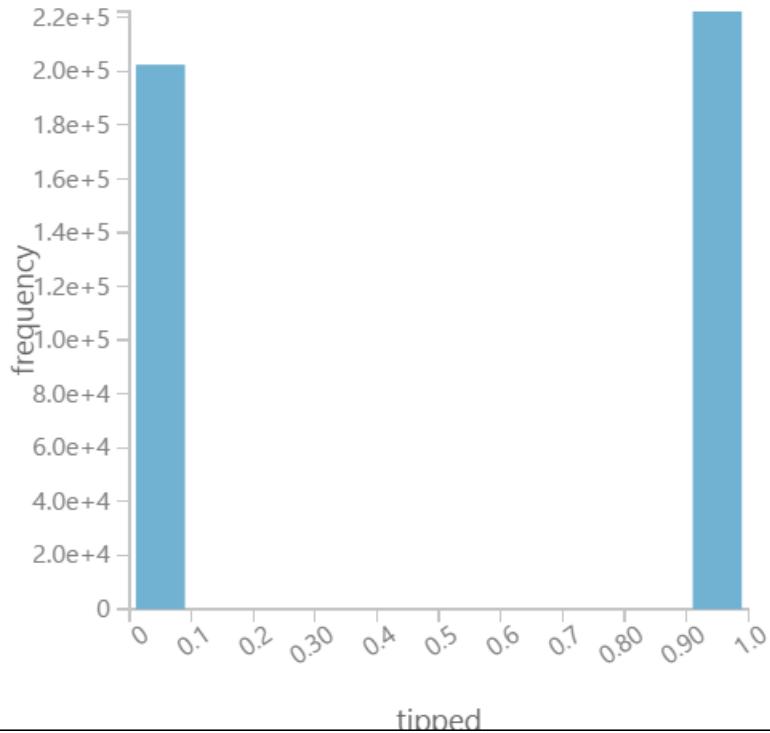
The snapshot below shows our experiment to predict whether or not a tip was paid for a given trip.



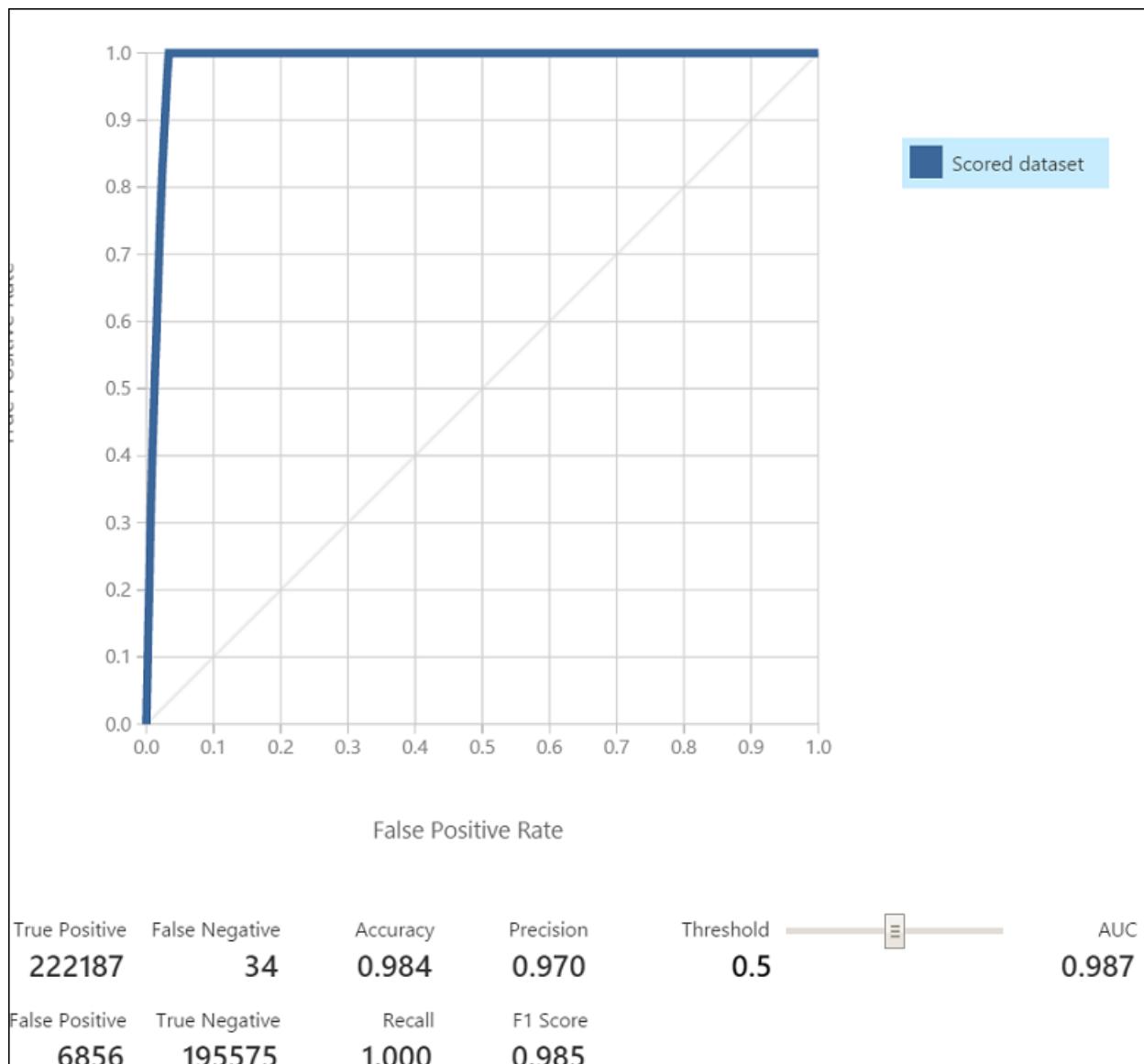
b. For this experiment, our target label distributions were roughly 1:1.

The snapshot below shows the distribution of tip class labels for the binary classification problem.

tipped Histogram



As a result, we obtain an AUC of 0.987 as shown in the figure below.

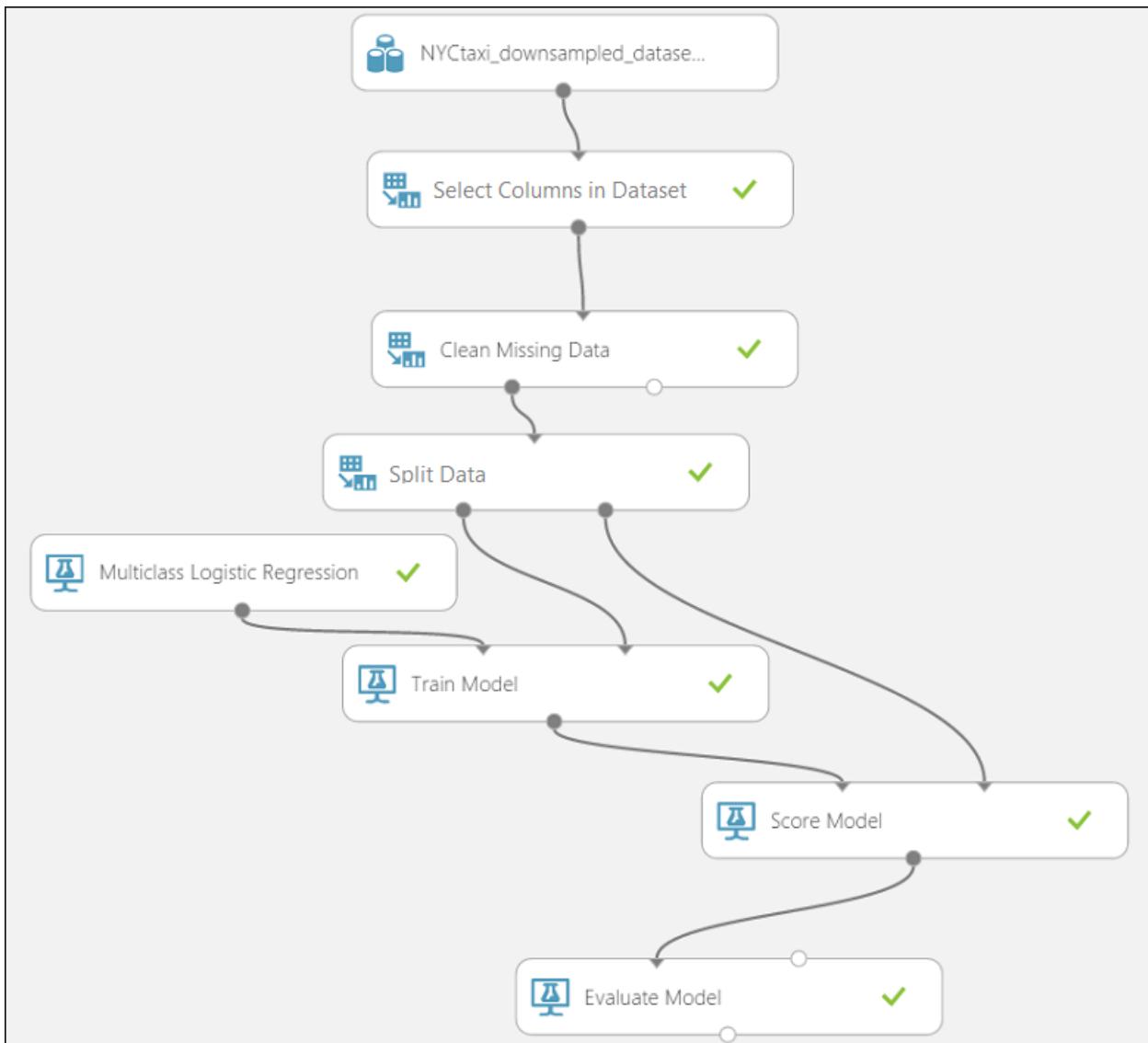


2. Multiclass classification: To predict the range of tip amounts paid for the trip, using the previously defined classes.

Learner used: Multiclass logistic regression

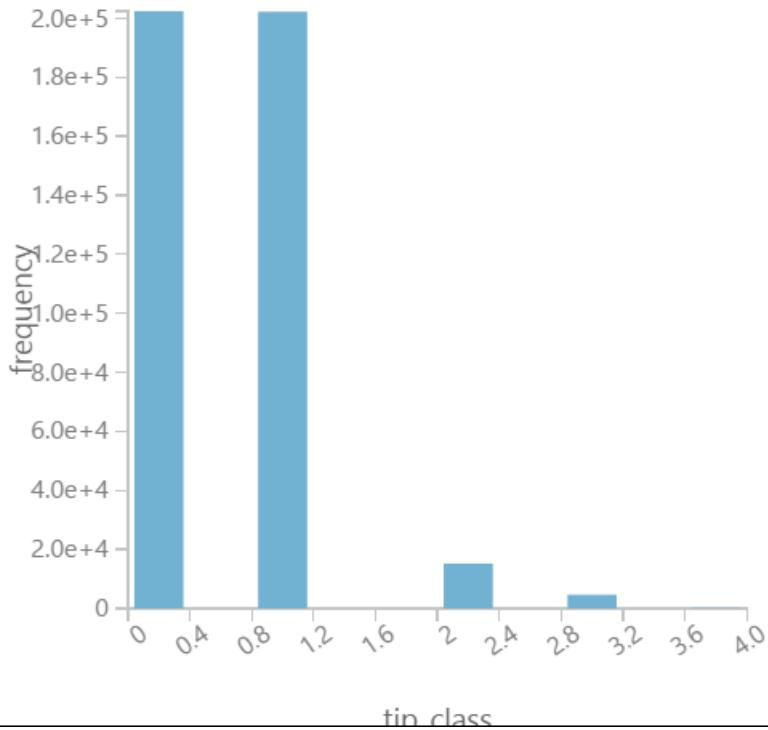
a. For this problem, our target (or class) label is "tip_class" which can take one of five values (0,1,2,3,4). As in the binary classification case, we have a few columns that are target leaks for this experiment. In particular : tipped, tip_amount, total_amount reveal information about the target label that is not available at testing time. We remove these columns using the [Select Columns in Dataset](#) module.

The snapshot below shows our experiment to predict in which bin a tip is likely to fall (Class 0: tip = \$0, class 1 : tip > \$0 and tip <= \$5, Class 2 : tip > \$5 and tip <= \$10, Class 3 : tip > \$10 and tip <= \$20, Class 4 : tip > \$20)



We now show what our actual test class distribution looks like. We see that while Class 0 and Class 1 are prevalent, the other classes are rare.

tip_class
Histogram

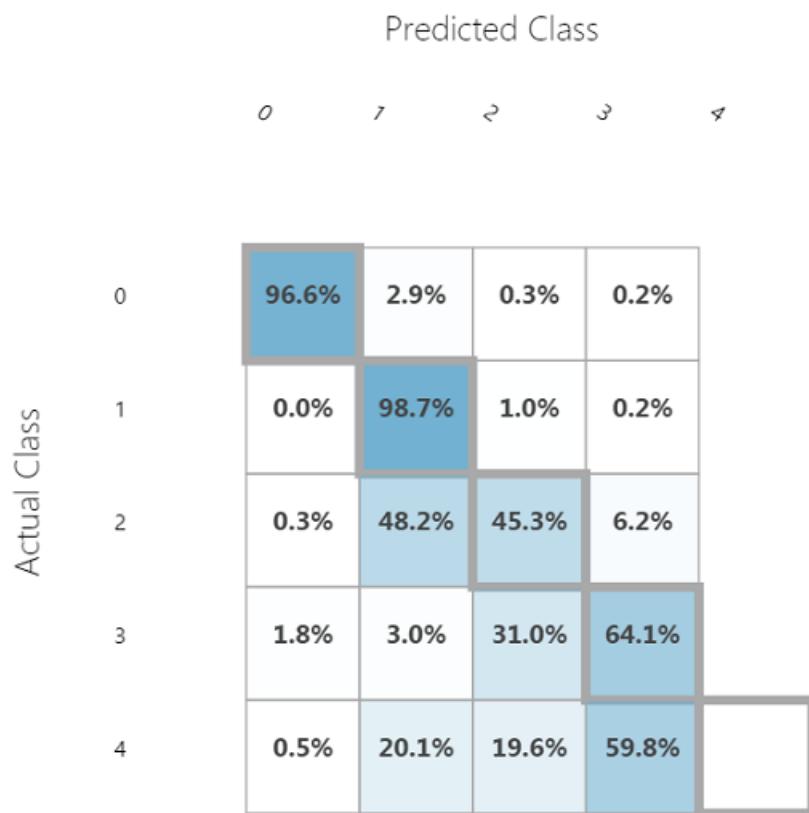


b. For this experiment, we use a confusion matrix to look at our prediction accuracies. This is shown below.

Metrics

Overall accuracy	0.953816
Average accuracy	0.981527
Micro-averaged precision	0.953816
Macro-averaged precision	NaN
Micro-averaged recall	0.953816
Macro-averaged recall	0.609453

Confusion Matrix



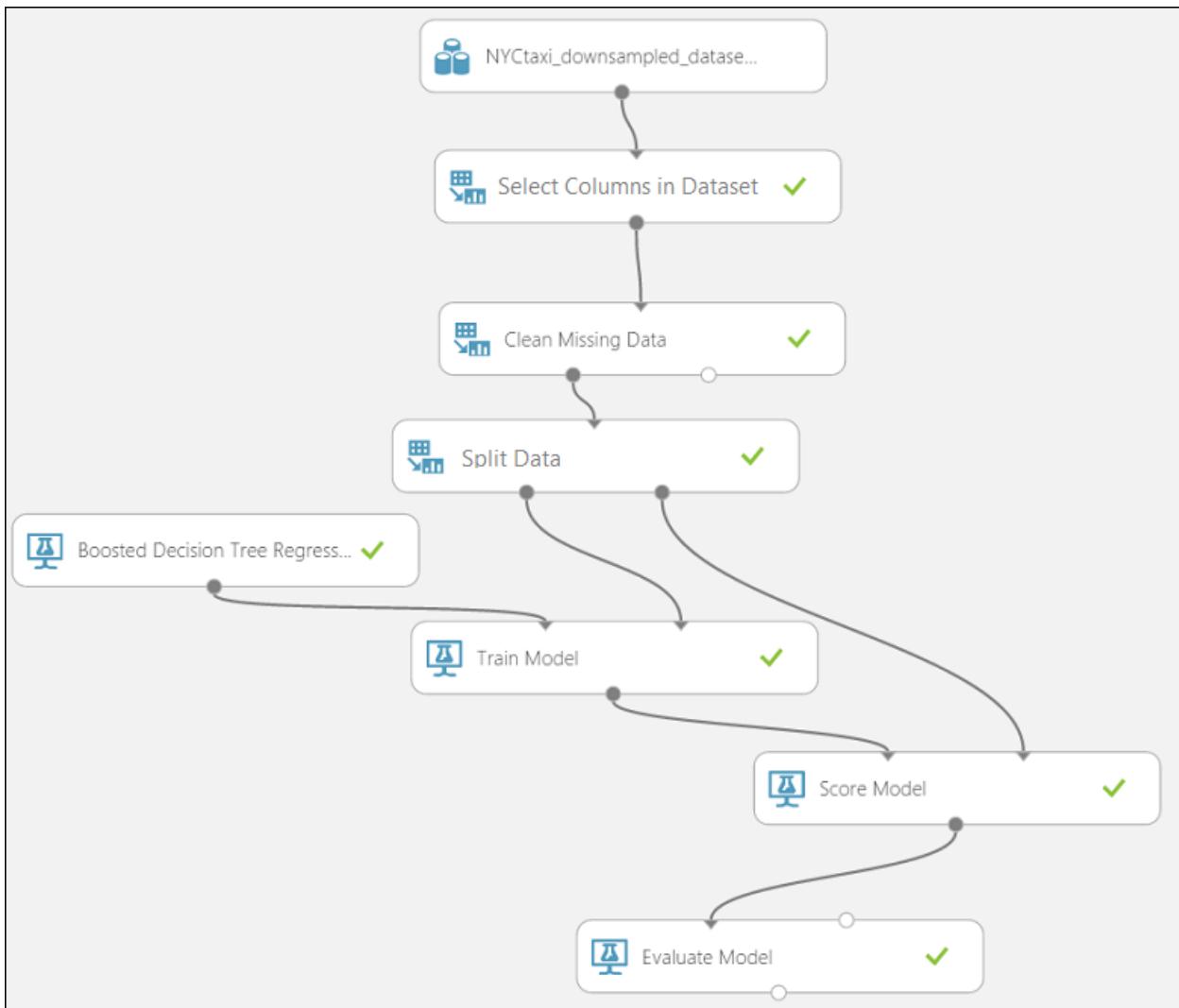
Note that while our class accuracies on the prevalent classes is quite good, the model does not do a good job of "learning" on the rarer classes.

3. Regression task: To predict the amount of tip paid for a trip.

Learner used: Boosted decision tree

a. For this problem, our target (or class) label is "tip_amount". Our target leaks in this case are : tipped, tip_class, total_amount ; all these variables reveal information about the tip amount that is typically unavailable at testing time. We remove these columns using the [Select Columns in Dataset](#) module.

The snapshot below shows our experiment to predict the amount of the given tip.



b. For regression problems, we measure the accuracies of our prediction by looking at the squared error in the predictions, the coefficient of determination, and the like. We show these below.

rows	columns	Mean Absolute Error	Root Mean Squared Error	Relative Absolute Error	Relative Squared Error	Coefficient of Determination
1	5					
view as						
		0.404674	1.181324	0.290471	0.290374	0.709626

We see that about the coefficient of determination is 0.709, implying about 71% of the variance is explained by our model coefficients.

IMPORTANT

To learn more about Azure Machine Learning and how to access and use it, please refer to [What's Machine Learning?](#). A very useful resource for playing with a bunch of Machine Learning experiments on Azure Machine Learning is the [Cortana Intelligence Gallery](#). The Gallery covers a gamut of experiments and provides a thorough introduction into the range of capabilities of Azure Machine Learning.

License Information

This sample walkthrough and its accompanying scripts are shared by Microsoft under the MIT license. Please check the LICENSE.txt file in the directory of the sample code on GitHub for more details.

References

- [Andrés Monroy NYC Taxi Trips Download Page](#)
- [FOILING NYC's Taxi Trip Data by Chris Whong](#)
- [NYC Taxi and Limousine Commission Research and Statistics](#)

The Team Data Science Process in action - Using Azure HDInsight Hadoop Clusters on a 1 TB dataset

1/17/2017 • 26 min to read • [Edit on GitHub](#)

In this walkthrough, we demonstrate using the Team Data Science Process in an end-to-end scenario with an [Azure HDInsight Hadoop cluster](#) to store, explore, feature engineer, and down sample data from one of the publicly available [Criteo](#) datasets. We use Azure Machine Learning to build a binary classification model on this data. We also show how to publish one of these models as a Web service.

It is also possible to use an IPython notebook to accomplish the tasks presented in this walkthrough. Users who would like to try this approach should consult the [Criteo walkthrough using a Hive ODBC connection](#) topic.

Criteo Dataset Description

The Criteo data is a click prediction dataset that is approximately 370GB of gzip compressed TSV files (~1.3TB uncompressed), comprising more than 4.3 billion records. It is taken from 24 days of click data made available by [Criteo](#). For the convenience of data scientists, we have unzipped data available to us to experiment with.

Each record in this dataset contains 40 columns:

- the first column is a label column that indicates whether a user clicks an **add** (value 1) or does not click one (value 0)
- next 13 columns are numeric, and
- last 26 are categorical columns

The columns are anonymized and use a series of enumerated names: "Col1" (for the label column) to 'Col40' (for the last categorical column).

Here is an excerpt of the first 20 columns of two observations (rows) from this dataset:

Col1	Col2	Col3	Col4	Col5	Col6	Col7	Col8	Col9	Col10	Col11	Col12	Col13	Col14	Col15	Col16	Col17	Col18
Col19	Col20																
0	40	42	2	54	3	0	0	2	16	0	1	4448	4	lacfe1ee	1b2ff61f	2e8b2631	6faef306
0		24		27	5		0	2	1		3	10064		9a8cb066	7a06385f	417e6103	2170fc56
1																	

There are missing values in both the numeric and categorical columns in this dataset. We describe a simple method for handling the missing values. Additional details of the data are explored when we store them into Hive tables.

Definition: *Clickthrough rate (CTR)*: This is the percentage of clicks in the data. In this Criteo dataset, the CTR is about 3.3% or 0.033.

Examples of prediction tasks

Two sample prediction problems are addressed in this walkthrough:

1. **Binary classification:** Predicts whether a user clicked an add:
 - Class 0: No Click
 - Class 1: Click
2. **Regression:** Predicts the probability of an ad click from user features.

Set Up an HDInsight Hadoop cluster for data science

Note: This is typically an **Admin** task.

Set up your Azure Data Science environment for building predictive analytics solutions with HDInsight clusters in three steps:

1. [Create a storage account](#): This storage account is used to store data in Azure Blob Storage. The data used in HDInsight clusters is stored here.
2. [Customize Azure HDInsight Hadoop Clusters for Data Science](#): This step creates an Azure HDInsight Hadoop cluster with 64-bit Anaconda Python 2.7 installed on all nodes. There are two important steps (described in this topic) to complete when customizing the HDInsight cluster.
 - You must link the storage account created in step 1 with your HDInsight cluster when it is created. This storage account is used for accessing data that can be processed within the cluster.
 - You must enable Remote Access to the head node of the cluster after it is created. Remember the remote access credentials you specify here (different from those specified for the cluster at its creation): you need them to complete the following procedures.
3. [Create an Azure ML workspace](#): This Azure Machine Learning workspace is used for building machine learning models after an initial data exploration and down sampling on the HDInsight cluster.

Get and consume data from a public source

The [Criteo](#) dataset can be accessed by clicking on the link, accepting the terms of use, and providing a name. A snapshot of what this looks like is shown here:

The screenshot shows a web browser window with the URL labs.criteo.com/downloads/download-terabyte-click-logs/. The page displays the 'CRITEOLABS DATA TERMS OF USE'. It contains a detailed description of the terms of use, followed by a section titled '1. SCOPE' which includes a sub-item '1.1.1'. At the bottom, there is a checkbox labeled 'I accept the terms of use' with a checked box, a text input field for 'Your Name' containing 'g', and a large orange button labeled 'CONTINUE TO DOWNLOAD'.

Click **Continue to Download** to read more about the dataset and its availability.

The data resides in a public [Azure blob storage](#) location:

`wasb://criteo@azuremlsampleexperiments.blob.core.windows.net/raw/`. The "wasb" refers to Azure Blob Storage location.

1. The data in this public blob storage consists of three subfolders of unzipped data.
 - a. The subfolder `raw/count/` contains the first 21 days of data - from day_00 to day_20
 - b. The subfolder `raw/train/` consists of a single day of data, day_21

- c. The subfolder `raw/test/` consists of two days of data, `day_22` and `day_23`
- 2. For those who want to start with the raw gzip data, these are also available in the main folder `raw/as day_NN.gz`, where NN goes from 00 to 23.

An alternative approach to access, explore, and model this data that does not require any local downloads is explained later in this walkthrough when we create Hive tables.

Log in to the cluster headnode

To log in to the headnode of the cluster, use the [Azure portal](#) to locate the cluster. Click the HDInsight elephant icon on the left and then double-click the name of your cluster. Navigate to the **Configuration** tab, double-click the CONNECT icon on the bottom of the page, and enter your remote access credentials when prompted. This takes you to the headnode of the cluster.

Here is what a typical first log in to the cluster headnode looks like:



On the left, we see the "Hadoop Command Line", which is our workhorse for the data exploration. We also see two useful URLs - "Hadoop Yarn Status" and "Hadoop Name Node". The yarn status URL shows job progress and the name node URL gives details on the cluster configuration.

Now we are set up and ready to begin first part of the walkthrough: data exploration using Hive and getting data ready for Azure Machine Learning.

Create Hive database and tables

To create Hive tables for our Criteo dataset, open the **Hadoop Command Line** on the desktop of the head node, and enter the Hive directory by entering the command

```
cd %hive_home%\bin
```

NOTE

Run all Hive commands in this walkthrough from the Hive bin/ directory prompt. This takes care of any path issues automatically. We use the terms "Hive directory prompt", "Hive bin/ directory prompt", and "Hadoop Command Line" interchangeably.

NOTE

To execute any Hive query, one can always use the following commands:

```
cd %hive_home%\bin  
hive
```

After the Hive REPL appears with a "hive >" sign, simply cut and paste the query to execute it.

The following code creates a database "criteo" and then generates 4 tables:

- a *table for generating counts* built on days day_00 to day_20,
- a *table for use as the train dataset* built on day_21, and
- two *tables for use as the test datasets* built on day_22 and day_23 respectively.

We split our test dataset into two different tables because one of the days is a holiday, and we want to determine if the model can detect differences between a holiday and non-holiday from the clickthrough rate.

The script [sample_hive_create_criteo_database_and_tables.hql](#) is displayed here for convenience:

```
CREATE DATABASE IF NOT EXISTS criteo;  
DROP TABLE IF EXISTS criteo.criteo_count;  
CREATE TABLE criteo.criteo_count (  
    col1 string,col2 double,col3 double,col4 double,col5 double,col6 double,col7 double,col8 double,col9 double,col10 double,col11 double,col12  
    double,col13 double,col14 double,col15 string,col16 string,col17 string,col18 string,col19 string,col20 string,col21 string,col22 string,col23  
    string,col24 string,col25 string,col26 string,col27 string,col28 string,col29 string,col30 string,col31 string,col32 string,col33 string,col34 string,col35  
    string,col36 string,col37 string,col38 string,col39 string,col40 string)  
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'  
LINES TERMINATED BY '\n'  
STORED AS TEXTFILE LOCATION 'wasb://criteo@azuremlsampleexperiments.blob.core.windows.net/raw/count';  
  
DROP TABLE IF EXISTS criteo.criteo_train;  
CREATE TABLE criteo.criteo_train (  
    col1 string,col2 double,col3 double,col4 double,col5 double,col6 double,col7 double,col8 double,col9 double,col10 double,col11 double,col12  
    double,col13 double,col14 double,col15 string,col16 string,col17 string,col18 string,col19 string,col20 string,col21 string,col22 string,col23  
    string,col24 string,col25 string,col26 string,col27 string,col28 string,col29 string,col30 string,col31 string,col32 string,col33 string,col34 string,col35  
    string,col36 string,col37 string,col38 string,col39 string,col40 string)  
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'  
LINES TERMINATED BY '\n'  
STORED AS TEXTFILE LOCATION 'wasb://criteo@azuremlsampleexperiments.blob.core.windows.net/raw/train';  
  
DROP TABLE IF EXISTS criteo.criteo_test_day_22;  
CREATE TABLE criteo.criteo_test_day_22 (  
    col1 string,col2 double,col3 double,col4 double,col5 double,col6 double,col7 double,col8 double,col9 double,col10 double,col11 double,col12  
    double,col13 double,col14 double,col15 string,col16 string,col17 string,col18 string,col19 string,col20 string,col21 string,col22 string,col23  
    string,col24 string,col25 string,col26 string,col27 string,col28 string,col29 string,col30 string,col31 string,col32 string,col33 string,col34 string,col35  
    string,col36 string,col37 string,col38 string,col39 string,col40 string)  
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'  
LINES TERMINATED BY '\n'  
STORED AS TEXTFILE LOCATION 'wasb://criteo@azuremlsampleexperiments.blob.core.windows.net/raw/test/day_22';  
  
DROP TABLE IF EXISTS criteo.criteo_test_day_23;  
CREATE TABLE criteo.criteo_test_day_23 (  
    col1 string,col2 double,col3 double,col4 double,col5 double,col6 double,col7 double,col8 double,col9 double,col10 double,col11 double,col12  
    double,col13 double,col14 double,col15 string,col16 string,col17 string,col18 string,col19 string,col20 string,col21 string,col22 string,col23  
    string,col24 string,col25 string,col26 string,col27 string,col28 string,col29 string,col30 string,col31 string,col32 string,col33 string,col34 string,col35  
    string,col36 string,col37 string,col38 string,col39 string,col40 string)  
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'  
LINES TERMINATED BY '\n'  
STORED AS TEXTFILE LOCATION 'wasb://criteo@azuremlsampleexperiments.blob.core.windows.net/raw/test/day_23';
```

We note that all these tables are external as we simply point to Azure Blob Storage (wasb) locations.

There are two ways to execute ANY Hive query that we now mention.

1. **Using the Hive REPL command-line:** The first is to issue a "hive" command and copy and paste a query at the Hive REPL command-line. To do this, do:

```
cd %hive_home%\bin  
hive
```

Now at the REPL command-line, cutting and pasting the query executes it.

2. **Saving queries to a file and executing the command:** The second is to save the queries to a .hql file ([sample_hive_create_criteo_database_and_tables.hql](#)) and then issue the following command to execute the query:

```
hive -f C:\temp\sample_hive_create_criteo_database_and_tables.hql
```

Confirm database and table creation

Next, we confirm the creation of the database with the following command from the Hive bin/ directory prompt:

```
hive -e "show databases;"
```

This gives:

```
criteo  
default  
Time taken: 1.25 seconds, Fetched: 2 row(s)
```

This confirms the creation of the new database, "criteo".

To see what tables we created, we simply issue the command here from the Hive bin/ directory prompt:

```
hive -e "show tables in criteo;"
```

We then see the following output:

```
criteo_count  
criteo_test_day_22  
criteo_test_day_23  
criteo_train  
Time taken: 1.437 seconds, Fetched: 4 row(s)
```

Data exploration in Hive

Now we are ready to do some basic data exploration in Hive. We begin by counting the number of examples in the train and test data tables.

Number of train examples

The contents of [sample_hive_count_train_table_examples.hql](#) are shown here:

```
SELECT COUNT(*) FROM criteo.criteo_train;
```

This yields:

```
192215183
```

```
Time taken: 264.154 seconds, Fetched: 1 row(s)
```

Alternatively, one may also issue the following command from the Hive bin/ directory prompt:

```
hive -f C:\temp\sample_hive_count_criteo_train_table_examples.hql
```

Number of test examples in the two test datasets

We now count the number of examples in the two test datasets. The contents of [sample_hive_count_criteo_test_day_22_table_examples.hql](#) are here:

```
SELECT COUNT(*) FROM criteo.criteo_test_day_22;
```

This yields:

```
189747893
```

```
Time taken: 267.968 seconds, Fetched: 1 row(s)
```

As usual, we may also call the script from the Hive bin/ directory prompt by issuing the command:

```
hive -f C:\temp\sample_hive_count_criteo_test_day_22_table_examples.hql
```

Finally, we examine the number of test examples in the test dataset based on day_23.

The command to do this is similar to the one just shown (refer to [sample_hive_count_criteo_test_day_23_examples.hql](#)):

```
SELECT COUNT(*) FROM criteo.criteo_test_day_23;
```

This gives:

```
178274637
```

```
Time taken: 253.089 seconds, Fetched: 1 row(s)
```

Label distribution in the train dataset

The label distribution in the train dataset is of interest. To see this, we show contents of [sample_hive_criteo_label_distribution_train_table.hql](#):

```
SELECT Coll, COUNT(*) AS CT FROM criteo.criteo_train GROUP BY Coll;
```

This yields the label distribution:

```
1 6292903
```

```
0 185922280
```

```
Time taken: 459.435 seconds, Fetched: 2 row(s)
```

Note that the percentage of positive labels is about 3.3% (consistent with the original dataset).

Histogram distributions of some numeric variables in the train dataset

We can use Hive's native "histogram_numeric" function to find out what the distribution of the numeric variables looks like. Here are the contents of [sample_hive_criteo_histogram_numeric.hql](#):

```

SELECT CAST(hist.x AS int) AS bin_center, CAST(hist.y AS bigint) AS bin_height FROM
  (SELECT
    histogram_numeric(col2, 20) AS col2_hist
   FROM
    criteo.criteo_train
  ) a
 LATERAL VIEW explode(col2_hist) exploded_table AS hist;

```

This yields the following:

```

26 155878415
2606 92753
6755 22086
11202 6922
14432 4163
17815 2488
21072 1901
24113 1283
27429 1225
30818 906
34512 723
38026 387
41007 290
43417 312
45797 571
49819 428
53505 328
56853 527
61004 160
65510 3446
Time taken: 317.851 seconds, Fetched: 20 row(s)

```

The LATERAL VIEW - explode combination in Hive serves to produce a SQL-like output instead of the usual list. Note that in this table, the first column corresponds to the bin center and the second to the bin frequency.

Approximate percentiles of some numeric variables in the train dataset

Also of interest with numeric variables is the computation of approximate percentiles. Hive's native "percentile_approx" does this for us. The contents of [sample_hive_criteo_approximate_percentiles.hql](#) are:

```

SELECT MIN(Col2) AS Col2_min, PERCENTILE_APPROX(Col2, 0.1) AS Col2_01, PERCENTILE_APPROX(Col2, 0.3) AS Col2_03,
PERCENTILE_APPROX(Col2, 0.5) AS Col2_median, PERCENTILE_APPROX(Col2, 0.8) AS Col2_08, MAX(Col2) AS Col2_max FROM
criteo.criteo_train;

```

This yields:

```

1.0 2.1418600917169246 2.1418600917169246 6.21887086390288 27.53454893115633 65535.0
Time taken: 564.953 seconds, Fetched: 1 row(s)

```

We remark that the distribution of percentiles is closely related to the histogram distribution of any numeric variable usually.

Find number of unique values for some categorical columns in the train dataset

Continuing the data exploration, we now find, for some categorical columns, the number of unique values they take. To do this, we show contents of [sample_hive_criteo_unique_values_categoricals.hql](#):

```

SELECT COUNT(DISTINCT(Col15)) AS num_uniques FROM criteo.criteo_train;

```

This yields:

```
19011825
```

```
Time taken: 448.116 seconds, Fetched: 1 row(s)
```

We note that Col15 has 19M unique values! Using naive techniques like "one-hot encoding" to encode such high-dimensional categorical variables is infeasible. In particular, we explain and demonstrate a powerful, robust technique called [Learning With Counts](#) for tackling this problem efficiently.

We end this sub-section by looking at the number of unique values for some other categorical columns as well. The contents of [sample_hive_criteo_unique_values_multiple_categoricals.hql](#) are:

```
SELECT COUNT(DISTINCT(Col16)), COUNT(DISTINCT(Col17)),
COUNT(DISTINCT(Col18)), COUNT(DISTINCT(Col19)), COUNT(DISTINCT(Col20))
FROM criteo.criteo_train;
```

This yields:

```
30935 15200 7349 20067 3
Time taken: 1933.883 seconds, Fetched: 1 row(s)
```

Again we see that except for Col20, all the other columns have many unique values.

Co-occurrence counts of pairs of categorical variables in the train dataset

The co-occurrence counts of pairs of categorical variables is also of interest. This can be determined using the code in [sample_hive_criteo_paired_categorical_counts.hql](#):

```
SELECT Col15, Col16, COUNT(*) AS paired_count FROM criteo.criteo_train GROUP BY Col15, Col16 ORDER BY paired_count DESC LIMIT 15;
```

We reverse order the counts by their occurrence and look at the top 15 in this case. This gives us:

ad98e872	cea68cd3	8964458
ad98e872	3dbb483e	8444762
ad98e872	43ced263	3082503
ad98e872	420acc05	2694489
ad98e872	ac4c5591	2559535
ad98e872	fb1e95da	2227216
ad98e872	8af1edc8	1794955
ad98e872	e56937ee	1643550
ad98e872	d1fade1c	1348719
ad98e872	977b4431	1115528
e5f3fd8d	a15d1051	959252
ad98e872	dd86c04a	872975
349b3fec	a52ef97d	821062
e5f3fd8d	a0aaaffa6	792250
265366bf	6f5c7c41	782142

```
Time taken: 560.22 seconds, Fetched: 15 row(s)
```

Down sample the datasets for Azure Machine Learning

Having explored the datasets and demonstrated how we may do this type of exploration for any variables (including combinations), we now down sample the data sets so that we can build models in Azure Machine Learning. Recall that the problem we focus on is: given a set of example attributes (feature values from Col2 - Col40), we predict if Col1 is a 0 (no click) or a 1 (click).

To down sample our train and test datasets to 1% of the original size, we use Hive's native RAND() function. The next script, [sample_hive_criteo_downsample_train_dataset.hql](#) does this for the train dataset:

```

CREATE TABLE criteo.criteo_train_downsample_1perc (
    col1 string,col2 double,col3 double,col4 double,col5 double,col6 double,col7 double,col8 double,col9 double,col10 double,col11 double,col12
double,col13 double,col14 double,col15 string,col16 string,col17 string,col18 string,col19 string,col20 string,col21 string,col22 string,col23
string,col24 string,col25 string,col26 string,col27 string,col28 string,col29 string,col30 string,col31 string,col32 string,col33 string,col34 string,col35
string,col36 string,col37 string,col38 string,col39 string,col40 string)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;

--Now downsample and store in this table

INSERT OVERWRITE TABLE criteo.criteo_train_downsample_1perc SELECT * FROM criteo.criteo_train WHERE RAND() <= 0.01;

```

This yields:

```

Time taken: 12.22 seconds
Time taken: 298.98 seconds

```

The script [sample_hive_criteo_downsample_test_day_22_dataset.hql](#) does it for test data, day_22:

```

--- Now for test data (day_22)

CREATE TABLE criteo.criteo_test_day_22_downsample_1perc (
    col1 string,col2 double,col3 double,col4 double,col5 double,col6 double,col7 double,col8 double,col9 double,col10 double,col11 double,col12
double,col13 double,col14 double,col15 string,col16 string,col17 string,col18 string,col19 string,col20 string,col21 string,col22 string,col23
string,col24 string,col25 string,col26 string,col27 string,col28 string,col29 string,col30 string,col31 string,col32 string,col33 string,col34 string,col35
string,col36 string,col37 string,col38 string,col39 string,col40 string)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;

INSERT OVERWRITE TABLE criteo.criteo_test_day_22_downsample_1perc SELECT * FROM criteo.criteo_test_day_22 WHERE RAND() <=
0.01;

```

This yields:

```

Time taken: 1.22 seconds
Time taken: 317.66 seconds

```

Finally, the script [sample_hive_criteo_downsample_test_day_23_dataset.hql](#) does it for test data, day_23:

```

--- Finally test data day_23
CREATE TABLE criteo.criteo_test_day_23_downsample_1perc (
    col1 string,col2 double,col3 double,col4 double,col5 double,col6 double,col7 double,col8 double,col9 double,col10 double,col11 double,col12
double,col13 double,col14 double,col15 string,col16 string,col17 string,col18 string,col19 string,col20 string,col21 string,col22 string,col23
string,col24 string,col25 string,col26 string,col27 string,col28 string,col29 string,col30 string,col31 string,col32 string,col33 string,col34 string,col35
string,col36 string,col37 string,col38 string,col39 string,col40 srical feature; tring)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;

INSERT OVERWRITE TABLE criteo.criteo_test_day_23_downsample_1perc SELECT * FROM criteo.criteo_test_day_23 WHERE RAND() <=
0.01;

```

This yields:

```

Time taken: 1.86 seconds
Time taken: 300.02 seconds

```

With this, we are ready to use our down sampled train and test datasets for building models in Azure Machine

Learning.

There is a final important component before we move on to Azure Machine Learning, which is concerns the count table. In the next sub-section, we discuss this in some detail.

A brief discussion on the count table

As we saw, several categorical variables have a very high dimensionality. In our walkthrough, we present a powerful technique called [Learning With Counts](#) to encode these variables in an efficient, robust manner. More information on this technique is in the link provided.

Note: In this walkthrough, we focus on using count tables to produce compact representations of high-dimensional categorical features. This is not the only way to encode categorical features; for more information on other techniques, interested users can check out [one-hot-encoding](#) and [feature hashing](#).

To build count tables on the count data, we use the data in the folder raw/count. In the modeling section, we show users how to build these count tables for categorical features from scratch, or alternatively to use a pre-built count table for their explorations. In what follows, when we refer to "pre-built count tables", we mean using the count tables that we provide. Detailed instructions on how to access these tables are provided in the next section.

Build a model with Azure Machine Learning

Our model building process in Azure Machine Learning follows these steps:

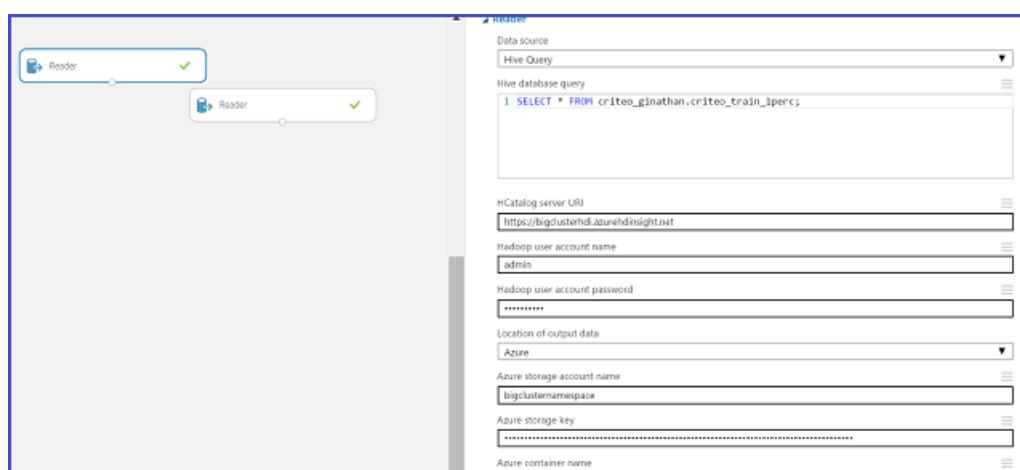
1. [Get the data from Hive tables into Azure Machine Learning](#)
2. [Create the experiment: clean the data, choose a learner, and featurize with count tables](#)
3. [Train the model](#)
4. [Score the model on test data](#)
5. [Evaluate the model](#)
6. [Publish the model as a web-service to be consumed](#)

Now we are ready to build models in Azure Machine Learning studio. Our down sampled data is saved as Hive tables in the cluster. We use the Azure Machine Learning **Import Data** module to read this data. The credentials to access the storage account of this cluster are provided in what follows.

Step 1: Get data from Hive tables into Azure Machine Learning using the Import Data module and select it for a machine learning experiment

Start by selecting a **+NEW -> EXPERIMENT -> Blank Experiment**. Then, from the **Search** box on the top left, search for "Import Data". Drag and drop the **Import Data** module on to the experiment canvas (the middle portion of the screen) to use the module for data access.

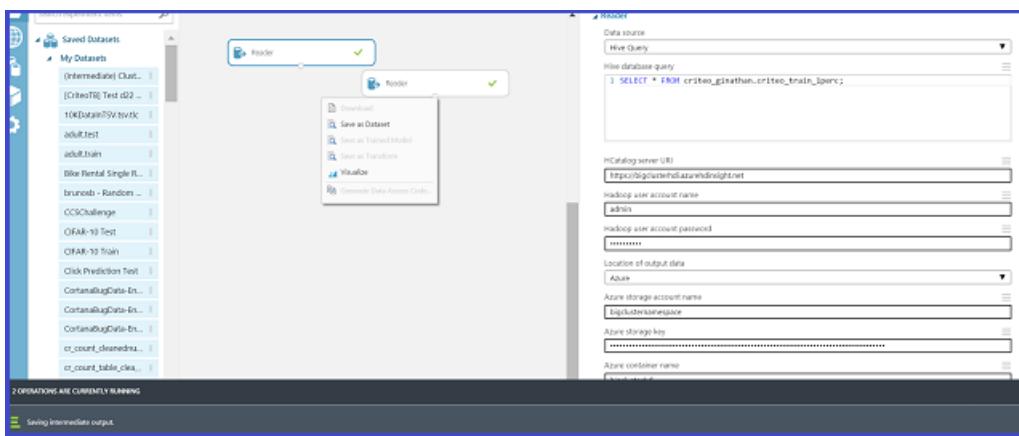
This is what the **Import Data** looks like while getting data from the Hive table:



For the **Import Data** module, the values of the parameters that are provided in the graphic are just examples of the sort of values you need to provide. Here is some general guidance on how to fill out the parameter set for the **Import Data** module.

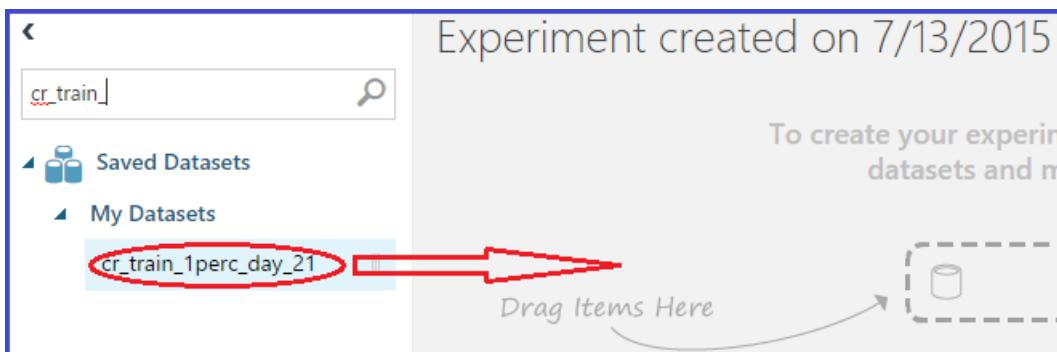
1. Choose "Hive query" for **Data Source**
2. In the **Hive database query** box, a simple SELECT * FROM - is enough.
3. **Hcatalog server URI**: If your cluster is "abc", then this is simply: <https://abc.azurehdinsight.net>
4. **Hadoop user account name**: The user name chosen at the time of commissioning the cluster. (NOT the Remote Access user name!)
5. **Hadoop user account password**: The password for the user name chosen at the time of commissioning the cluster. (NOT the Remote Access password!)
6. **Location of output data**: Choose "Azure"
7. **Azure storage account name**: The storage account associated with the cluster
8. **Azure storage account key**: The key of the storage account associated with the cluster.
9. **Azure container name**: If the cluster name is "abc", then this is simply "abc", usually.

Once the **Import Data** finishes getting data (you see the green tick on the Module), save this data as a Dataset (with a name of your choice). What this looks like:



Right-click the output port of the **Import Data** module. This reveals a **Save as dataset** option and a **Visualize** option. The **Visualize** option, if clicked, displays 100 rows of the data, along with a right panel that is useful for some summary statistics. To save data, simply select **Save as dataset** and follow instructions.

To select the saved dataset for use in a machine learning experiment, locate the datasets using the **Search** box shown in the following figure. Then simply type out the name you gave the dataset partially to access it and drag the dataset onto the main panel. Dropping it onto the main panel selects it for use in machine learning modeling.

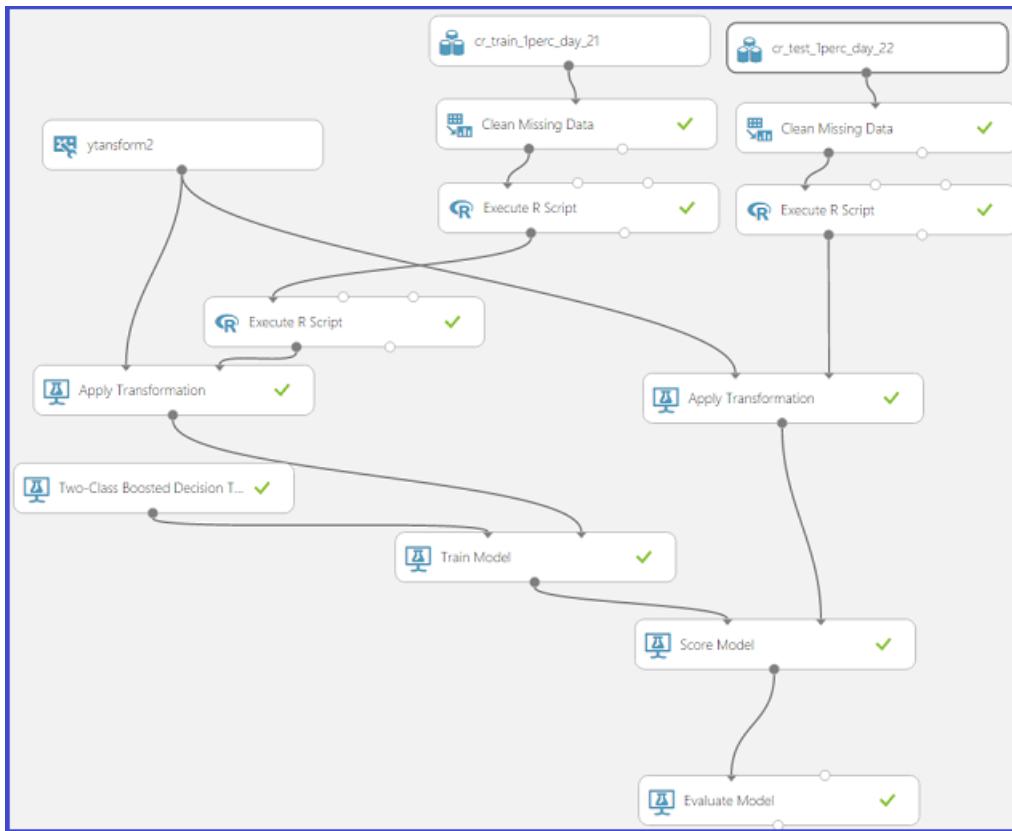


NOTE

Do this for both the train and the test datasets. Also, remember to use the database name and table names that you gave for this purpose. The values used in the figure are solely for illustration purposes.**

Step 2: Create a simple experiment in Azure Machine Learning to predict clicks / no clicks

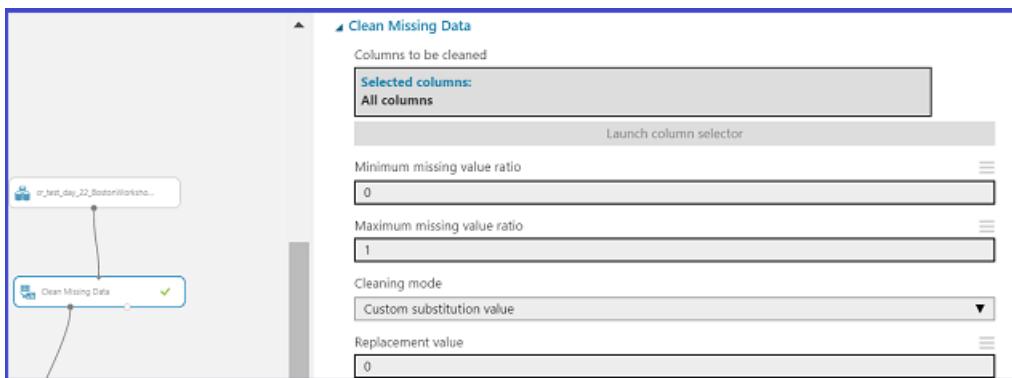
Our Azure ML experiment looks like this:



We now examine the key components of this experiment. As a reminder, we need to drag our saved train and test datasets on to our experiment canvas first.

Clean Missing Data

The **Clean Missing Data** module does what its name suggests: it cleans missing data in ways that can be user-specified. Looking into this module, we see this:



Here, we chose to replace all missing values with a 0. There are other options as well, which can be seen by looking at the dropdowns in the module.

Feature engineering on the data

There can be millions of unique values for some categorical features of large datasets. Using naive methods such as one-hot encoding for representing such high-dimensional categorical features is entirely infeasible. In this walkthrough, we demonstrate how to use count features using built-in Azure Machine Learning modules to generate compact representations of these high-dimensional categorical variables. The end-result is a smaller model size, faster training times, and performance metrics that are quite comparable to using other techniques.

Building counting transforms

To build count features, we use the **Build Counting Transform** module that is available in Azure Machine Learning. The module looks like this:

The screenshot shows the Azure HDInsight Studio interface. At the top, there is a configuration panel for the 'Build Counting Transform' module. It includes fields for 'Number of classes' (set to 2), 'The bits of hash function' (set to 20), 'The seed of hash function' (set to 20000), 'Module type' (set to MapReduce), 'Default storage account name' (set to hadoopsummitstorage), and 'Default storage account key' (redacted). Below this is a summary of the module's execution:

- Default container name: hadoopsummitstorage
- Cluster URI: https://hadoopsummit.azurehdinsight.net
- Username: admin
- Password: redacted
- Number of reducers: 10
- Input data container: DefaultContainer
- Input blob name: hive/warehouse/criteo.db/criteo_count_full_textfile/*
- Output blob path: hadoopsummitstorage
- Count columns: 15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,3
- Label column: 1
- Blob format: CSV

At the bottom, there are status details: START TIME 7/9/2015 1:35:13 PM, END TIME 7/9/2015 1:35:13 PM, ELAPSED TIME 0:00:00.000, STATUS CODE Finished, and STATUS DETAILS Task output was present in output cache.

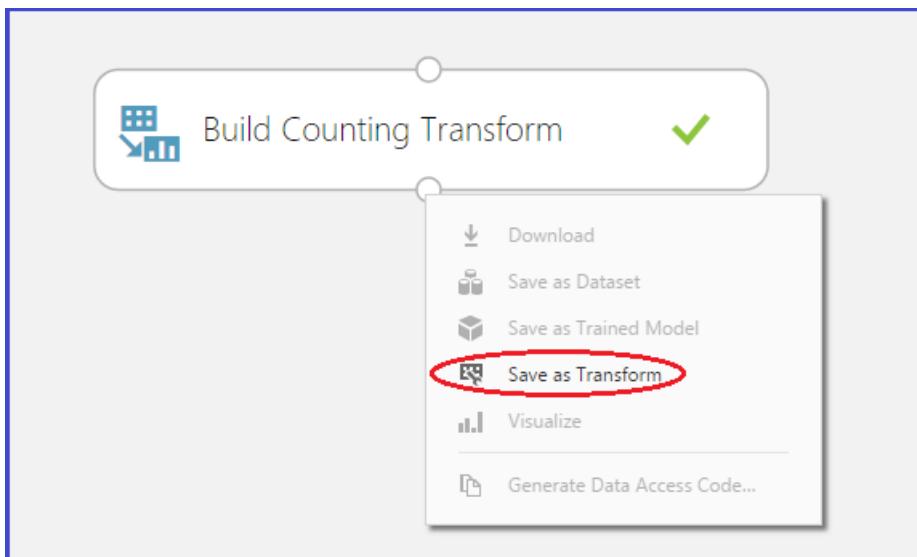
Important Note : In the **Count columns** box, we enter those columns that we wish to perform counts on. Typically, these are (as mentioned) high-dimensional categorical columns. At the start, we mentioned that the Criteo dataset has 26 categorical columns: from Col15 to Col40. Here, we count on all of them and give their indices (from 15 to 40 separated by commas as shown).

To use the module in the MapReduce mode (appropriate for large datasets), we need access to an HDInsight Hadoop cluster (the one used for feature exploration can be reused for this purpose as well) and its credentials. The previous figures illustrate what the filled-in values look like (replace the values provided for illustration with those relevant for your own use-case).

This is a detailed view of the 'Build Counting Transform' configuration dialog. The 'Input blob name' field is set to 'hive/warehouse/criteo.db/criteo_count_full_textfile/*'. Other fields visible include 'Output blob path' (hadoopsummitstorage), 'Count columns' (15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,3), 'Label column' (1), and 'Blob format' (CSV).

In the figure above, we show how to enter the input blob location. This location has the data reserved for building count tables on.

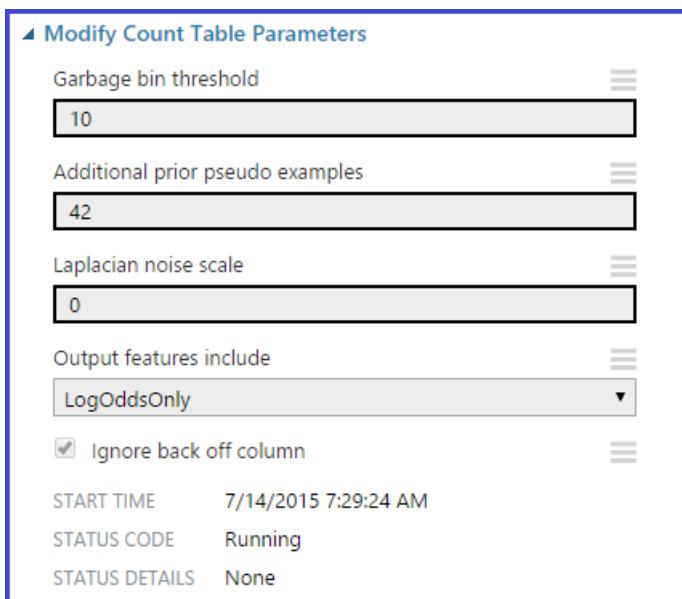
After this module finishes running, we can save the transform for later by right-clicking the module and selecting the **Save as Transform** option:



In our experiment architecture shown above, the dataset "ytransform2" corresponds precisely to a saved count transform. For the remainder of this experiment, we assume that the reader used a **Build Counting Transform** module on some data to generate counts, and can then use those counts to generate count features on the train and test datasets.

Choosing what count features to include as part of the train and test datasets

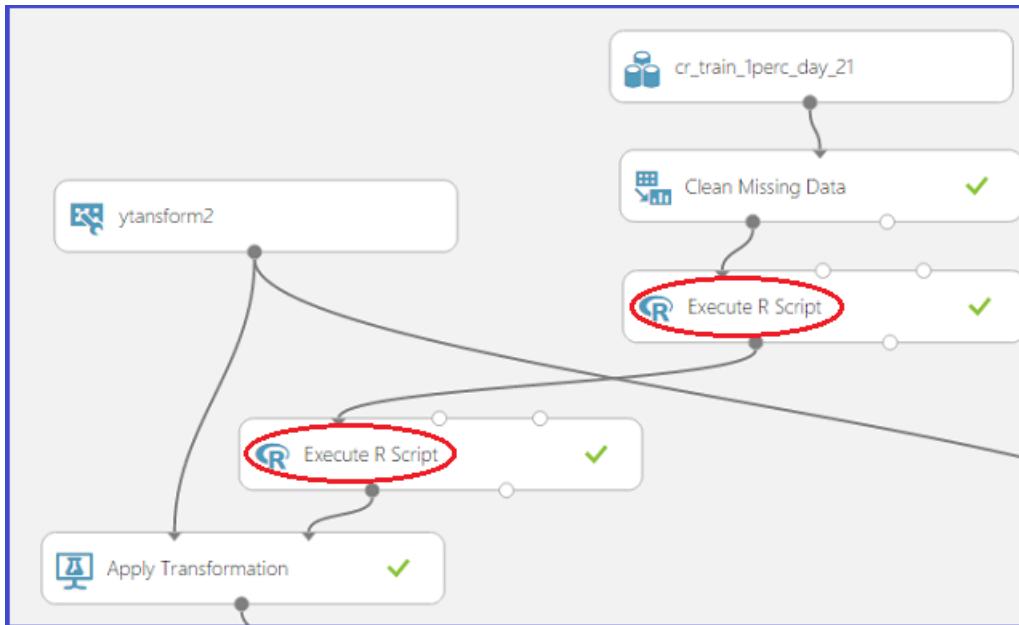
Once we have a count transform ready, the user can choose what features to include in their train and test datasets using the **Modify Count Table Parameters** module. We just show this module here for completeness, but in interests of simplicity do not actually use it in our experiment.



In this case, as can be seen, we have chosen to use just the log-odds and to ignore the back off column. We can also set parameters such as the garbage bin threshold, how many pseudo-prior examples to add for smoothing, and whether to use any Laplacian noise or not. All these are advanced features and it is to be noted that the default values are a good starting point for users who are new to this type of feature generation.

Data transformation before generating the count features

Now we focus on an important point about transforming our train and test data prior to actually generating count features. Note that there are two **Execute R Script** modules used before we apply the count transform to our data.



Here is the first R script:

Execute R Script

R Script

```

1 # Map 1-based optional input ports to variables
2 dataset1 <- maml.mapInputPort(1) # class: data.frame
3
4 colnames(dataset1) <- c("Col1","Col2","Col3","Col4","Col5","Col6","Col7","Col8","Col9","Col10",
   "Col11","Col12","Col13","Col14","Col15","Col16","Col17","Col18","Col19","Col20","Col21",
   "Col22","Col23","Col24","Col25","Col26","Col27","Col28","Col29","Col30","Col31","Col32",
   "Col33","Col34","Col35","Col36","Col37","Col38","Col39","Col40")
5
6 # Select data.frame to be sent to the output Dataset port
7 maml.mapOutputPort("dataset1");
  
```

Random Seed

START TIME 7/13/2015 7:48:31 PM
 END TIME 7/13/2015 7:48:31 PM
 ELAPSED TIME 0:00:00.000
 STATUS CODE Finished

In this R script, we rename our columns to names "Col1" to "Col40". This is because the count transform expects names of this format.

In the second R script, we balance the distribution between positive and negative classes (classes 1 and 0 respectively) by downsampling the negative class. The R script here shows how to do this:

Execute R Script

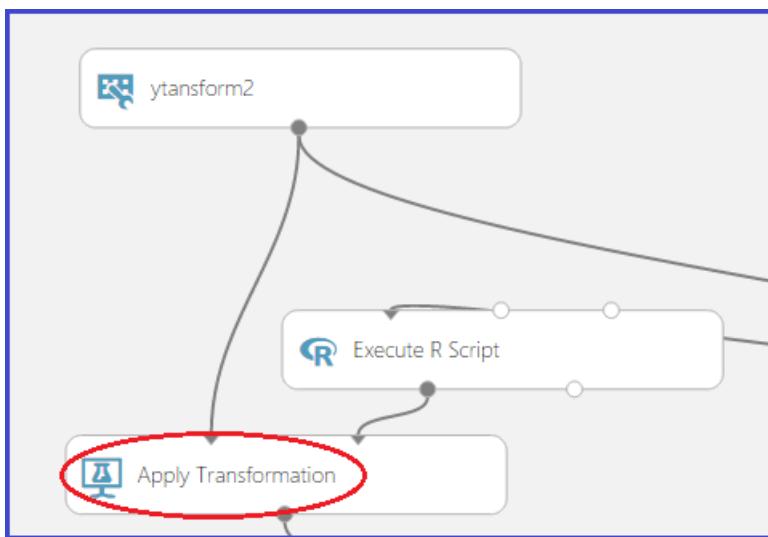
```
R Script
1 # Map 1-based optional input ports to variables
2 dataset1 <- maml.mapInputPort(1) # class: data.frame
3
4 # balance the classes so that pos-neg in a specified ratio
5
6 pos_neg_ratio <- 1
7
8 d_class0 <- subset(dataset1, Col1 == "0")
9 d_class1 <- subset(dataset1, Col1 == "1")
10
11 numRows_class0 <- nrow(d_class0)
12 numRows_class1 <- nrow(d_class1)
13
14 # downsample the negative class 0
15 numRows_class0_downsampled <- numRows_class1 * pos_neg_ratio
16 d_class0_downsampled <- d_class0[sample(numRows_class0, numRows_class0_downsampled, replace = FALSE), ]
17
18 # new output data frame containing 1:1 class ratios
19
20 data.set <- data.frame()
21
22 data.set <- rbind(data.set, d_class0_downsampled)
23 data.set <- rbind(data.set, d_class1)
24
25 # shuffle the rows
26
27 numRows_data.set <- nrow(data.set)
28 data.set <- data.set[sample(numRows_data.set, numRows_data.set, replace = FALSE), ]
29
30 # Select data.frame to be sent to the output Dataset port
31 maml.mapOutputPort("data.set");
```

Random Seed

In this simple R script, we use "pos_neg_ratio" to set the amount of balance between the positive and the negative classes. This is important to do since improving class imbalance usually has performance benefits for classification problems where the class distribution is skewed (recall that in our case, we have 3.3% positive class and 96.7% negative class).

Applying the count transformation on our data

Finally, we can use the **Apply Transformation** module to apply the count transforms on our train and test datasets. This module takes the saved count transform as one input and the train or test datasets as the other input, and returns data with count features. It is shown here:



An excerpt of what the count features look like

It is instructive to see what the count features look like in our case. Here we show an excerpt of this:

Col17 - CountForClass_1	Col17 - LogOddsForClass_0	Col17 - IsBackoff	Col18 - CountForClass_0	Col18 - CountForClass_1	Col18 - LogOddsForClass_0
20293	3.525333	0	75910130	2598584	3.374576
5499	3.902167	0	1085268350	37124760	3.375298
101725	3.730514	0	41190580	1409818	3.374733
23340	3.727357	0	5355288	183919	3.371234

In this excerpt, we show that for the columns that we counted on, we get the counts and log odds in addition to any relevant backoffs.

We are now ready to build an Azure Machine Learning model using these transformed datasets. In the next section, we show how this can be done.

Azure Machine Learning model building

Choice of learner

First, we need to choose a learner. We are going to use a two class boosted decision tree as our learner. Here are the default options for this learner:

▲ Two-Class Boosted Decision Tree

Create trainer mode

Single Parameter

Maximum number of leaves per tree
20

Minimum number of samples per leaf node
10

Learning rate
0.2

Number of trees constructed
100

Random number seed
[empty]

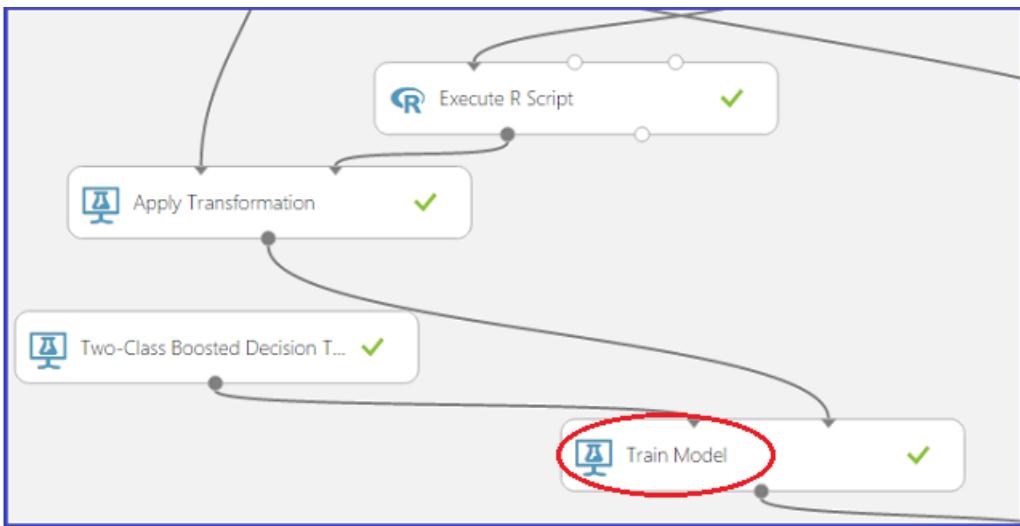
Allow unknown categorical levels

START TIME 7/9/2015 2:02:53 PM
END TIME 7/9/2015 2:02:53 PM
ELAPSED TIME 0:00:00.000
STATUS CODE Finished

For our experiment, we are going to choose the default values. We note that the defaults are usually meaningful and a good way to get quick baselines on performance. You can improve on performance by sweeping parameters if you choose to once you have a baseline.

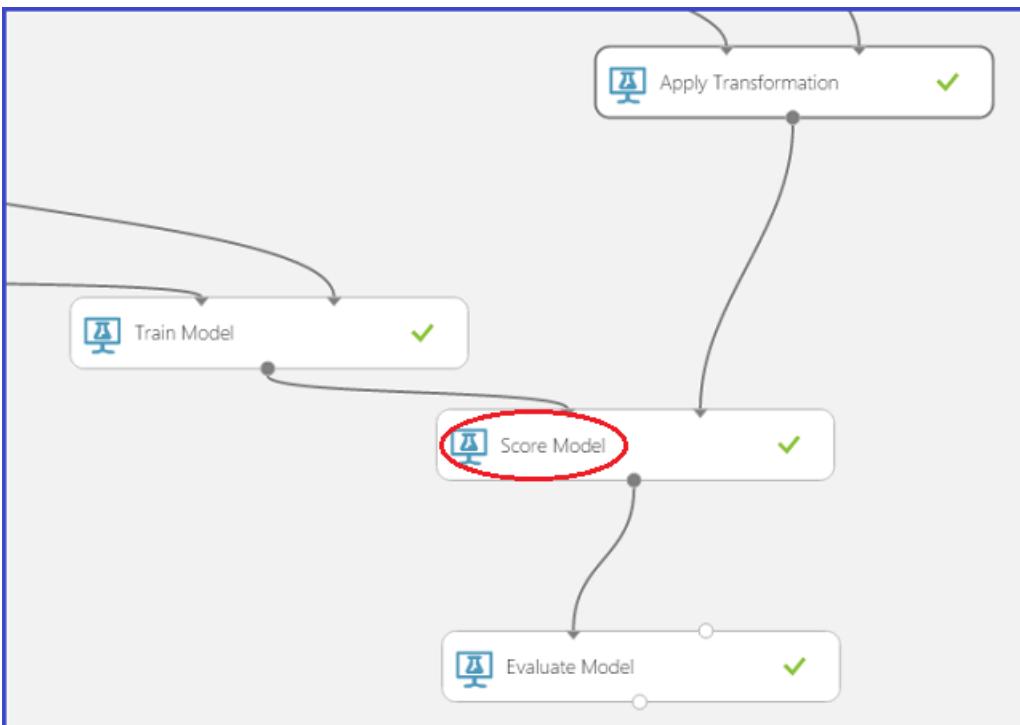
Train the model

For training, we simply invoke a **Train Model** module. The two inputs to it are the Two-Class Boosted Decision Tree learner and our train dataset. This is shown here:



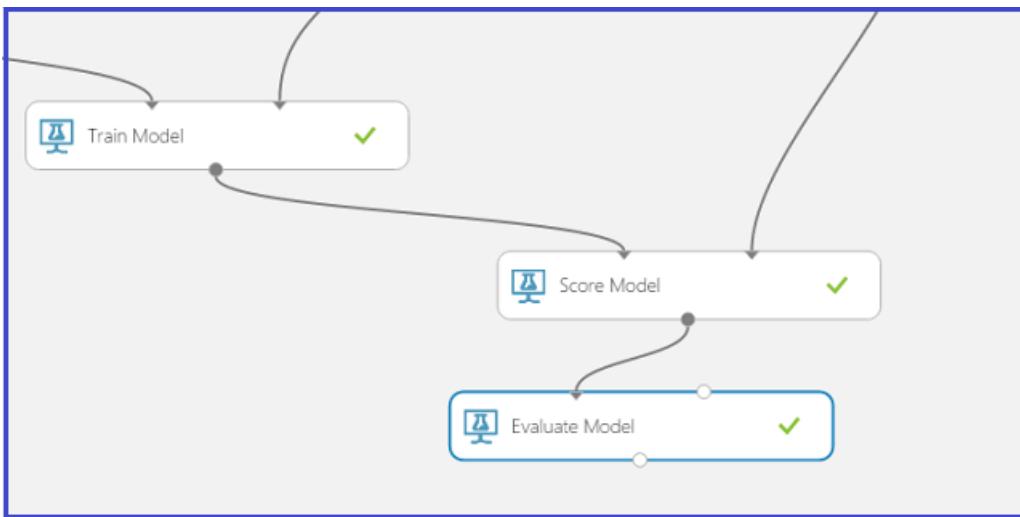
Score the model

Once we have a trained model, we are ready to score on the test dataset and to evaluate its performance. We do this by using the **Score Model** module shown in the following figure, along with an **Evaluate Model** module:

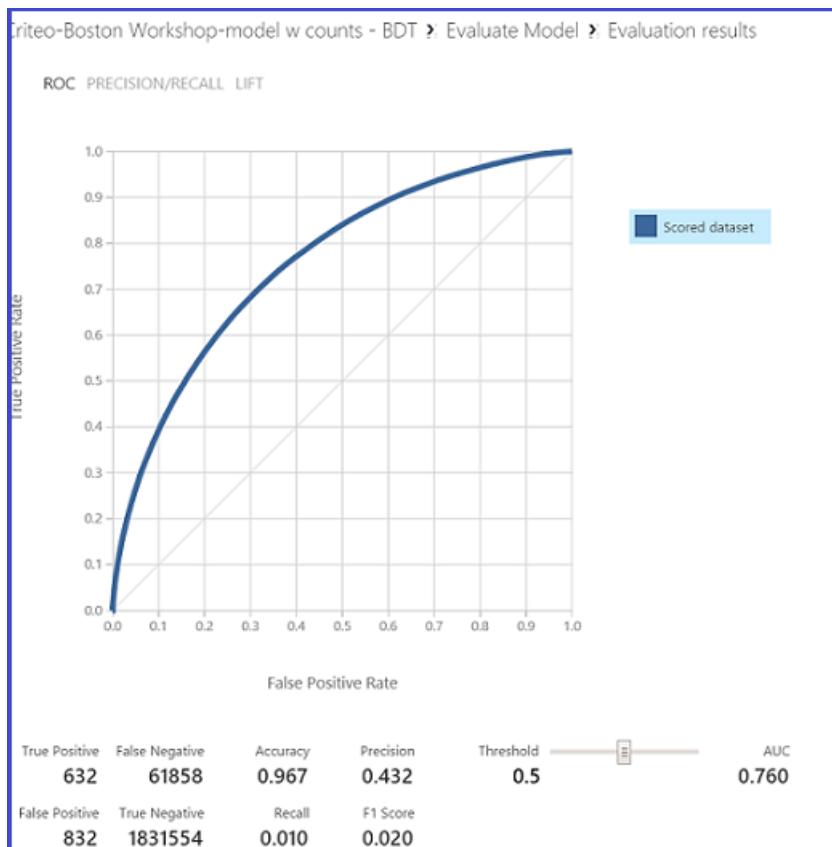


Step 5: Evaluate the model

Finally, we would like to analyze model performance. Usually, for two class (binary) classification problems, a good measure is the AUC. To visualize this, we hook up the **Score Model** module to an **Evaluate Model** module for this. Clicking **Visualize** on the **Evaluate Model** module yields a graphic like the following one:



In binary (or two class) classification problems, a good measure of prediction accuracy is the Area Under Curve (AUC). In what follows, we show our results using this model on our test dataset. To get this, right-click the output port of the **Evaluate Model** module and then **Visualize**.



Step 6: Publish the model as a Web service

The ability to publish an Azure Machine Learning model as web services with a minimum of fuss is a valuable feature for making it widely available. Once that is done, anyone can make calls to the web service with input data that they need predictions for, and the web service uses the model to return those predictions.

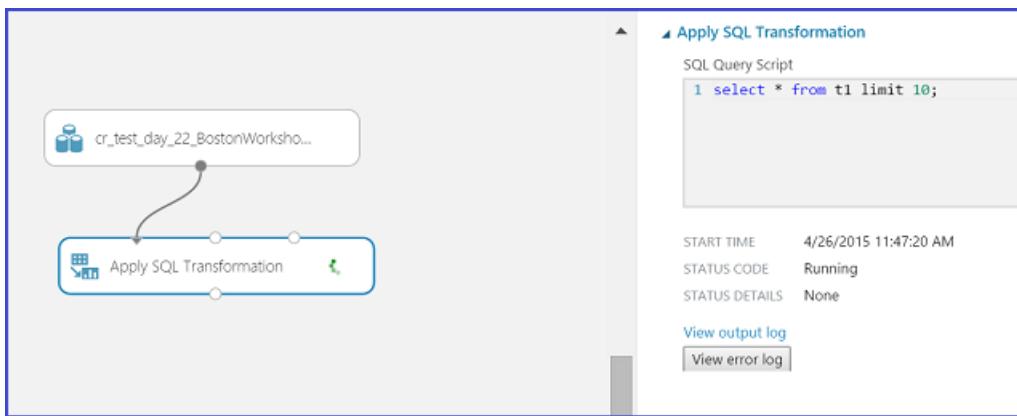
To do this, we first save our trained model as a Trained Model object. This is done by right-clicking the **Train Model** module and using the **Save as Trained Model** option.

Next, we need to create input and output ports for our web service:

- an input port takes data in the same form as the data that we need predictions for
- an output port returns the Scored Labels and the associated probabilities.

Select a few rows of data for the input port

It is convenient to use an **Apply SQL Transformation** module to select just 10 rows to serve as the input port data. Select just these rows of data for our input port using the SQL query shown here:

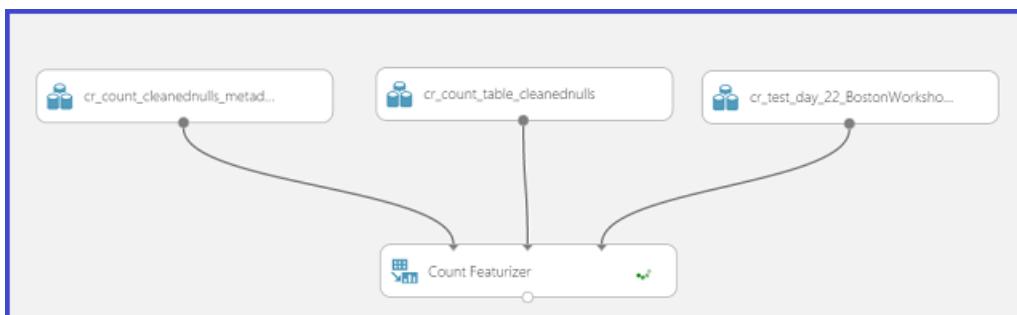


Web service

Now we are ready to run a small experiment that can be used to publish our web service.

Generate input data for webservice

As a zeroth step, since the count table is large, we take a few lines of test data and generate output data from it with count features. This can serve as the input data format for our webservice. This is shown here:



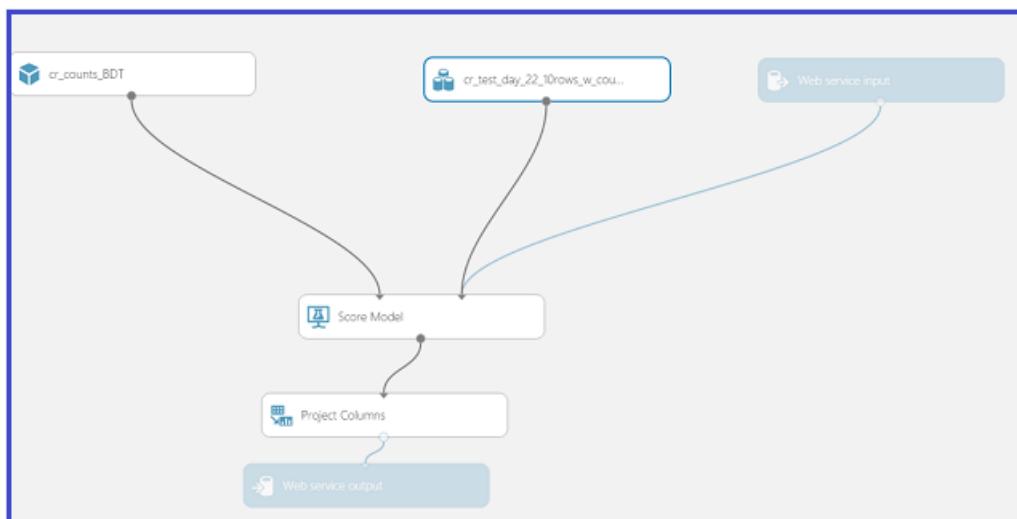
NOTE

For the input data format, we now use the OUTPUT of the **Count Featurizer** module. Once this experiment finishes running, save the output from the **Count Featurizer** module as a Dataset. This Dataset is used for the input data in the webservice.

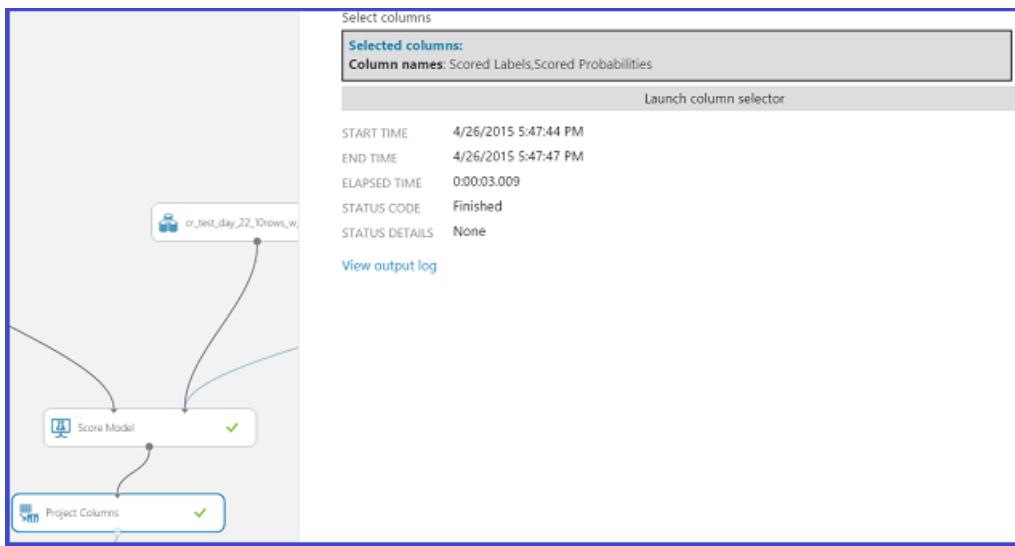
Scoring experiment for publishing webservice

First, we show what this looks like. The essential structure is a **Score Model** module that accepts our trained model object and a few lines of input data that we generated in the previous steps using the **Count Featurizer** module.

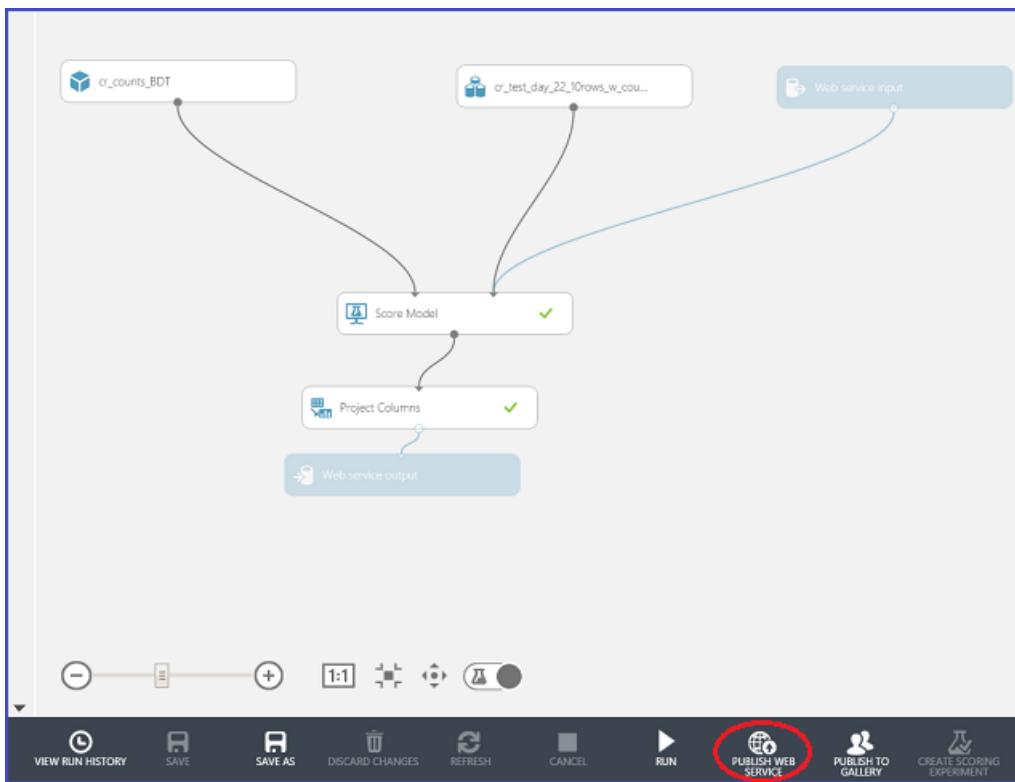
We use "Select Columns in Dataset" to project out the Scored labels and the Score probabilities.



Notice how the **Select Columns in Dataset** module can be used for 'filtering' data out from a dataset. We show the contents here:



To get the blue input and output ports, you simply click **prepare webservice** at the bottom right. Running this experiment also allows us to publish the web service: click the **PUBLISH WEB SERVICE** icon at the bottom right, shown here:



Once the webservice is published, we get redirected to a page that looks thus:

creto modeling boston - bdt - webservice final

DASHBOARD CONFIGURATION

General

Published experiment:

[View snapshot](#) [View latest](#)

Description

No description provided for this web service.

API key

```
0r3mir+6eDnkEmEkF4MDIE661ge2XgnkpCs216ltOXAS6iKzYb4uKVgs5DM3jmje8+9CjfvYzVjVKplsL2sO0Sew==
```

Default Endpoint

API HELP PAGE TEST APPS

REQUEST/RESPONSE (circled)
BATCH EXECUTION

Test

Download Excel Workbook

Additional endpoints

Number of additional endpoints created for this web service: 0
[Manage endpoints in Azure management portal](#)

We see two links for webservices on the left side:

- The **REQUEST/RESPONSE** Service (or RRS) is meant for single predictions and is what we utilize in this workshop.
 - The **BATCH EXECUTION** Service (BES) is used for batch predictions and requires that the input data used to make predictions reside in Azure Blob Storage.

Clicking on the link **REQUEST/RESPONSE** takes us to a page that gives us pre-canned code in C#, python, and R. This code can be conveniently used for making calls to the webservice. Note that the API key on this page needs to be used for authentication.

It is convenient to copy this python code over to a new cell in the IPython notebook.

Here we show a segment of python code with the correct API key.

```

data = {
    "Inputs": {
        "input1": {
            "ColumnNames": ["col1", "col2", "col3", "col4", "col5", "col6", "col7", "col8", "col9", "col10"],
            "Values": [ [ "0", "0", "0", "0", "0", "0", "0", "0", "0", "0" ] ],
            "GlobalParameters": {}
        }
    }
}

body = str.encode(json.dumps(data))

url = 'https://ussouthcentral.services.azureml.net/workspaces/28f954aef09e40608b8c58e74b879e86/services/29c1b9
api_key = '0r3m1r+6eDnkEmEkF4MD1E661ge2xgnkpcS216it0XAS61kzYb4ukVgsSDM3jmje8+9CJFYzVjVKpsL2s00SeW==' # Replace
headers = {'Content-Type':'application/json', 'Authorization':('Bearer ' + api_key)}

req = urllib2.Request(url, body, headers)

```

Note that we replaced the default API key with our webservices's API key. Clicking **Run** on this cell in an IPython notebook yields the following response:

```
{ "Results":{ "output1":{ "type": "table", "value": { "ColumnNames": [ "Scored Labels", "Scored Probabilities"], "ColumnTypes": [ "Int32", "Double"], "Values": [ [ "1", "0.966483491428746 ], [ "1", "0.966483491428746 ] ] } } }}
```

We see that for the two test examples we asked about (in the JSON framework of the python script), we get back answers in the form "Scored Labels, Scored Probabilities". Note that in this case, we chose the default values that the pre-canned code provides (0's for all numeric columns and the string "value" for all categorical columns).

This concludes our end-to-end walkthrough showing how to handle large-scale dataset using Azure Machine Learning. We started with a terabyte of data, constructed a prediction model and deployed it as a web service in the cloud.

The Team Data Science Process in action: using SQL Server

1/17/2017 • 24 min to read • [Edit on GitHub](#)

In this tutorial, you walk through the process of building and deploying a machine learning model using SQL Server and a publicly available dataset -- the [NYC Taxi Trips](#) dataset. The procedure follows a standard data science workflow: ingest and explore the data, engineer features to facilitate learning, then build and deploy a model.

NYC Taxi Trips Dataset Description

The NYC Taxi Trip data is about 20GB of compressed CSV files (~48GB uncompressed), comprising more than 173 million individual trips and the fares paid for each trip. Each trip record includes the pickup and drop-off location and time, anonymized hack (driver's) license number and medallion (taxi's unique id) number. The data covers all trips in the year 2013 and is provided in the following two datasets for each month:

1. The 'trip_data' CSV contains trip details, such as number of passengers, pickup and dropoff points, trip duration, and trip length. Here are a few sample records:

```
medallion,hack_license,vendor_id,rate_code,store_and_fwd_flag,pickup_datetime,dropoff_datetime,passenger_count,trip_time_in_secs,trip_distance,pickup_longitude,pickup_latitude,dropoff_longitude,dropoff_latitude
89D227B655E5C82AECF13C3F540D4CF4,BA96DE419E711691B9445D6A6307C170,CMT,1,N,2013-01-01 15:11:48,2013-01-01 15:18:10,4,382,1.00,-73.978165,40.757977,-73.989838,40.751171
0BD7C8F5BA12B88E0B67BED28BEA73D8,9FD8F69F0804BDB5549F40E9DA1BE472,CMT,1,N,2013-01-06 00:18:35,2013-01-06 00:22:54,1,259,1.50,-74.006683,40.731781,-73.994499,40.75066
0BD7C8F5BA12B88E0B67BED28BEA73D8,9FD8F69F0804BDB5549F40E9DA1BE472,CMT,1,N,2013-01-05 18:49:41,2013-01-05 18:54:23,1,282,1.10,-74.004707,40.737777,-74.009834,40.726002
DFD2202EE08F7A8DC9A57B02ACB81FE2,51EE87E3205C985EF8431D850C786310,CMT,1,N,2013-01-07 23:54:15,2013-01-07 23:58:20,2,244,.70,-73.974602,40.759945,-73.984734,40.759388
DFD2202EE08F7A8DC9A57B02ACB81FE2,51EE87E3205C985EF8431D850C786310,CMT,1,N,2013-01-07 23:25:03,2013-01-07 23:34:24,1,560,2.10,-73.97625,40.748528,-74.002586,40.747868
```

2. The 'trip_fare' CSV contains details of the fare paid for each trip, such as payment type, fare amount, surcharge and taxes, tips and tolls, and the total amount paid. Here are a few sample records:

```
medallion,hack_license,vendor_id,pickup_datetime,payment_type,fare_amount,surcharge,mta_tax,tip_amount,tolls_amount,total_amount
89D227B655E5C82AECF13C3F540D4CF4,BA96DE419E711691B9445D6A6307C170,CMT,2013-01-01 15:11:48,CSH,6.5,0,0.5,0,0,7
0BD7C8F5BA12B88E0B67BED28BEA73D8,9FD8F69F0804BDB5549F40E9DA1BE472,CMT,2013-01-06 00:18:35,CSH,6.0,5,0.5,0,0,7
0BD7C8F5BA12B88E0B67BED28BEA73D8,9FD8F69F0804BDB5549F40E9DA1BE472,CMT,2013-01-05 18:49:41,CSH,5.5,1,0.5,0,0,7
DFD2202EE08F7A8DC9A57B02ACB81FE2,51EE87E3205C985EF8431D850C786310,CMT,2013-01-07 23:54:15,CSH,5.0,5,0.5,0,0,6
DFD2202EE08F7A8DC9A57B02ACB81FE2,51EE87E3205C985EF8431D850C786310,CMT,2013-01-07 23:25:03,CSH,9.5,0.5,0.5,0,0,10.5
```

The unique key to join trip_data and trip_fare is composed of the fields: medallion, hack_licence and pickup_datetime.

Examples of Prediction Tasks

We will formulate three prediction problems based on the *tip_amount*, namely:

1. Binary classification: Predict whether or not a tip was paid for a trip, i.e. a *tip_amount* that is greater than \$0 is a positive example, while a *tip_amount* of \$0 is a negative example.
2. Multiclass classification: To predict the range of tip paid for the trip. We divide the *tip_amount* into five

bins or classes:

```
Class 0 : tip_amount = $0
Class 1 : tip_amount > $0 and tip_amount <= $5
Class 2 : tip_amount > $5 and tip_amount <= $10
Class 3 : tip_amount > $10 and tip_amount <= $20
Class 4 : tip_amount > $20
```

3. Regression task: To predict the amount of tip paid for a trip.

Setting Up the Azure data science environment for advanced analytics

As you can see from the [Plan Your Environment](#) guide, there are several options to work with the NYC Taxi Trips dataset in Azure:

- Work with the data in Azure blobs then model in Azure Machine Learning
- Load the data into a SQL Server database then model in Azure Machine Learning

In this tutorial we will demonstrate parallel bulk import of the data to a SQL Server, data exploration, feature engineering and down sampling using SQL Server Management Studio as well as using IPython Notebook.

[Sample scripts](#) and [IPython notebooks](#) are shared in GitHub. A sample IPython notebook to work with the data in Azure blobs is also available in the same location.

To set up your Azure Data Science environment:

1. [Create a storage account](#)
2. [Create an Azure Machine Learning workspace](#)
3. [Provision a Data Science Virtual Machine](#), which will serve as a SQL Server as well an IPython Notebook server.

NOTE

The sample scripts and IPython notebooks will be downloaded to your Data Science virtual machine during the setup process. When the VM post-installation script completes, the samples will be in your VM's Documents library:

- Sample Scripts: `C:\Users\<user_name>\Documents\Data Science Scripts`
- Sample IPython Notebooks: `C:\Users\<user_name>\Documents\IPython Notebooks\DataSetSamples`
where `<user_name>` is your VM's Windows login name. We will refer to the sample folders as **Sample Scripts** and **Sample IPython Notebooks**.

Based on the dataset size, data source location, and the selected Azure target environment, this scenario is similar to [Scenario #5: Large dataset in a local files, target SQL Server in Azure VM](#).

Get the Data from Public Source

To get the [NYC Taxi Trips](#) dataset from its public location, you may use any of the methods described in [Move Data to and from Azure Blob Storage](#) to copy the data to your new virtual machine.

To copy the data using AzCopy:

1. Log in to your virtual machine (VM)
2. Create a new directory in the VM's data disk (Note: Do not use the Temporary Disk which comes with the VM as a Data Disk).
3. In a Command Prompt window, run the following Azcopy command line, replacing `with` your data folder created in (2):

```
"C:\Program Files (x86)\Microsoft SDKs\Azure\AzCopy\azcopy" /Source:https://nyctaxitrips.blob.core.windows.net/data /Dest:  
<path_to_data_folder>/S
```

When the AzCopy completes, a total of 24 zipped CSV files (12 for trip_data and 12 for trip_fare) should be in the data folder.

4. Unzip the downloaded files. Note the folder where the uncompressed files reside. This folder will be referred to as the .

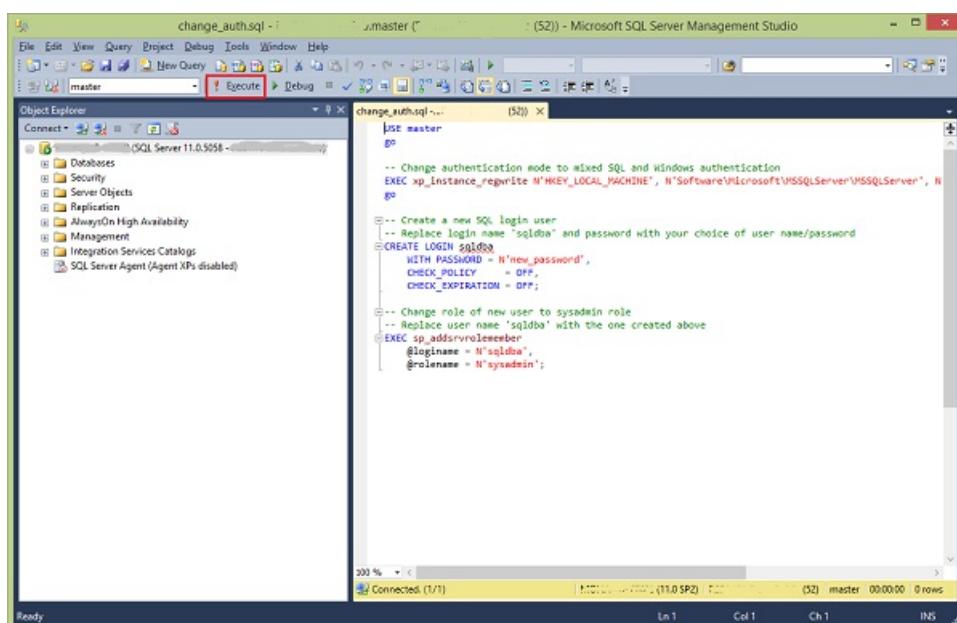
Bulk Import Data into SQL Server Database

The performance of loading/transferring large amounts of data to an SQL database and subsequent queries can be improved by using *Partitioned Tables and Views*. In this section, we will follow the instructions described in [Parallel Bulk Data Import Using SQL Partition Tables](#) to create a new database and load the data into partitioned tables in parallel.

1. While logged in to your VM, start **SQL Server Management Studio**.
2. Connect using Windows Authentication.



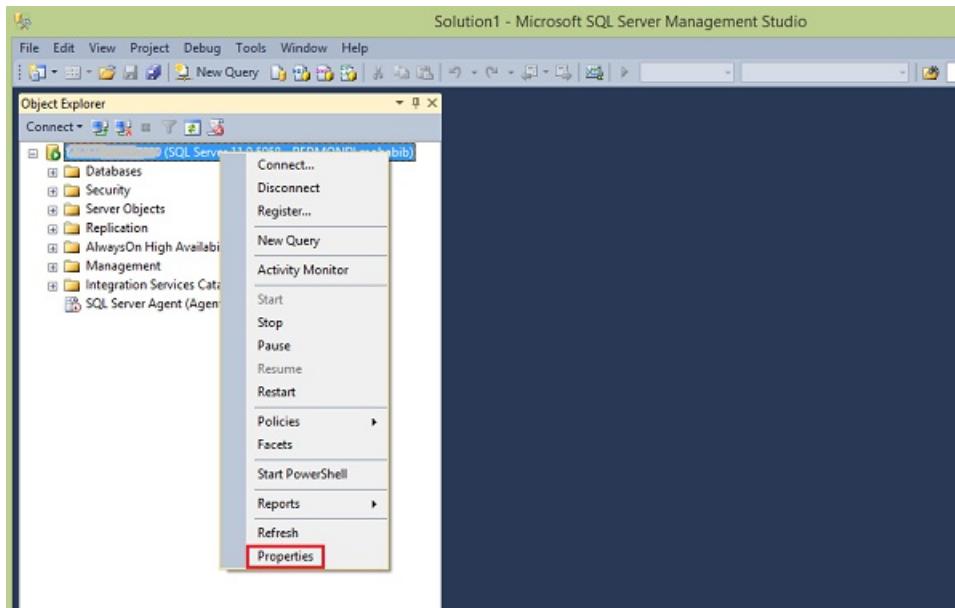
3. If you have not yet changed the SQL Server authentication mode and created a new SQL login user, open the script file named **change_auth.sql** in the **Sample Scripts** folder. Change the default user name and password. Click **!Execute** in the toolbar to run the script.



4. Verify and/or change the SQL Server default database and log folders to ensure that newly created

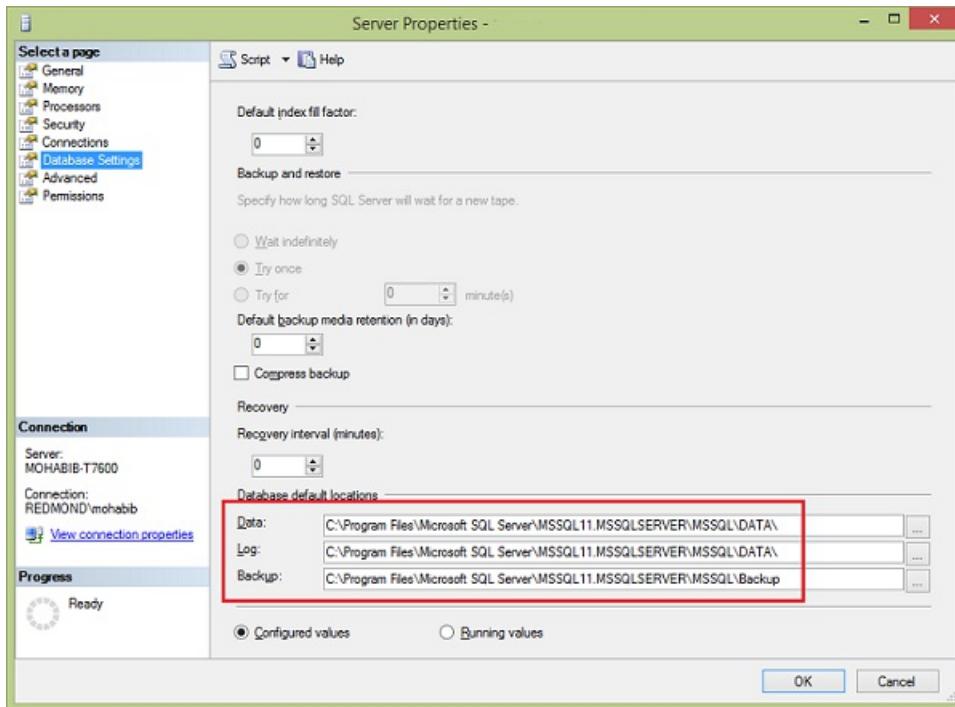
databases will be stored in a Data Disk. The SQL Server VM image that is optimized for datawarehousing loads is pre-configured with data and log disks. If your VM did not include a Data Disk and you added new virtual hard disks during the VM setup process, change the default folders as follows:

- Right-click the SQL Server name in the left panel and click **Properties**.



- Select **Database Settings** from the **Select a page** list to the left.
- Verify and/or change the **Database default locations** to the **Data Disk** locations of your choice.

This is where new databases reside if created with the default location settings.



- To create a new database and a set of filegroups to hold the partitioned tables, open the sample script **create_db_default.sql**. The script will create a new database named **TaxiNYC** and 12 filegroups in the default data location. Each filegroup will hold one month of trip_data and trip_fare data. Modify the database name, if desired. Click **!Execute** to run the script.
- Next, create two partition tables, one for the trip_data and another for the trip_fare. Open the sample script **create_partitioned_table.sql**, which will:
 - Create a partition function to split the data by month.
 - Create a partition scheme to map each month's data to a different filegroup.

- Create two partitioned tables mapped to the partition scheme: **nytaxi_trip** will hold the trip_data and **nytaxi_fare** will hold the trip_fare data.

Click **!Execute** to run the script and create the partitioned tables.

7. In the **Sample Scripts** folder, there are two sample PowerShell scripts provided to demonstrate parallel bulk imports of data to SQL Server tables.

- **bcp_parallel_generic.ps1** is a generic script to parallel bulk import data into a table. Modify this script to set the input and target variables as indicated in the comment lines in the script.
 - **bcp_parallel_nyctaxi.ps1** is a pre-configured version of the generic script and can be used to load both tables for the NYC Taxi Trips data.

- Right-click the **bcp_parallel_nyctaxi.ps1** script name and click **Edit** to open it in PowerShell. Review the preset variables and modify according to your selected database name, input data folder, target log folder, and paths to the sample format files **nyctaxi_trip.xml** and **nyctaxi_fare.xml** (provided in the **Sample Scripts** folder).

You may also select the authentication mode, default is Windows Authentication. Click the green arrow in the toolbar to run. The script will launch 24 bulk import operations in parallel, 12 for each partitioned table. You may monitor the data import progress by opening the SQL Server default data folder as set above.

9. The PowerShell script reports the starting and ending times. When all bulk imports complete, the ending time is reported. Check the target log folder to verify that the bulk imports were successful, i.e., no errors reported in the target log folder.
 10. Your database is now ready for exploration, feature engineering, and other operations as desired. Since the tables are partitioned according to the **pickup_datetime** field, queries which include **pickup_datetime** conditions in the **WHERE** clause will benefit from the partition scheme.
 11. In **SQL Server Management Studio**, explore the provided sample script **sample_queries.sql**. To run any of the sample queries, highlight the query lines then click **!Execute** in the toolbar.
 12. The NYC Taxi Trips data is loaded in two separate tables. To improve join operations, it is highly recommended to index the tables. The sample script **create_partitioned_index.sql** creates partitioned indexes on the composite join key **medallion, hack license, and pickup datetime**.

Data Exploration and Feature Engineering in SQL Server

In this section, we will perform data exploration and feature generation by running SQL queries directly in the **SQL Server Management Studio** using the SQL Server database created earlier. A sample script named **sample_queries.sql** is provided in the **Sample Scripts** folder. Modify the script to change the database name, if it is different from the default: **TaxiNYC**.

In this exercise, we will:

- Connect to **SQL Server Management Studio** using either Windows Authentication or using SQL Authentication and the SQL login name and password.
- Explore data distributions of a few fields in varying time windows.
- Investigate data quality of the longitude and latitude fields.
- Generate binary and multiclass classification labels based on the **tip_amount**.
- Generate features and compute/compare trip distances.
- Join the two tables and extract a random sample that will be used to build models.

When you are ready to proceed to Azure Machine Learning, you may either:

1. Save the final SQL query to extract and sample the data and copy-paste the query directly into a [Import Data](#) module in Azure Machine Learning, or
2. Persist the sampled and engineered data you plan to use for model building in a new database table and use the new table in the [Import Data](#) module in Azure Machine Learning.

In this section we will save the final query to extract and sample the data. The second method is demonstrated in the [Data Exploration and Feature Engineering in IPython Notebook](#) section.

For a quick verification of the number of rows and columns in the tables populated earlier using parallel bulk import,

```
-- Report number of rows in table nyctaxi_trip without table scan  
SELECT SUM(rows) FROM sys.partitions WHERE object_id = OBJECT_ID('nyctaxi_trip')  
  
-- Report number of columns in table nyctaxi_trip  
SELECT COUNT(*) FROM information_schema.columns WHERE table_name = 'nyctaxi_trip'
```

Exploration: Trip distribution by medallion

This example identifies the medallion (taxi numbers) with more than 100 trips within a given time period. The query would benefit from the partitioned table access since it is conditioned by the partition scheme of **pickup_datetime**. Querying the full dataset will also make use of the partitioned table and/or index scan.

```
SELECT medallion, COUNT(*)  
FROM nyctaxi_fare  
WHERE pickup_datetime BETWEEN '20130101' AND '20130331'  
GROUP BY medallion  
HAVING COUNT(*) > 100
```

Exploration: Trip distribution by medallion and hack_license

```
SELECT medallion, hack_license, COUNT(*)  
FROM nyctaxi_fare  
WHERE pickup_datetime BETWEEN '20130101' AND '20130131'  
GROUP BY medallion, hack_license  
HAVING COUNT(*) > 100
```

Data Quality Assessment: Verify records with incorrect longitude and/or latitude

This example investigates if any of the longitude and/or latitude fields either contain an invalid value (radian degrees should be between -90 and 90), or have (0, 0) coordinates.

```

SELECT COUNT(*) FROM nyctaxi_trip
WHERE pickup_datetime BETWEEN '20130101' AND '20130331'
AND (CAST(pickup_longitude AS float) NOT BETWEEN -90 AND 90
OR CAST(pickup_latitude AS float) NOT BETWEEN -90 AND 90
OR CAST(dropoff_longitude AS float) NOT BETWEEN -90 AND 90
OR CAST(dropoff_latitude AS float) NOT BETWEEN -90 AND 90
OR (pickup_longitude = '0' AND pickup_latitude = '0')
OR (dropoff_longitude = '0' AND dropoff_latitude = '0'))

```

Exploration: Tipped vs. Not Tipped Trips distribution

This example finds the number of trips that were tipped vs. not tipped in a given time period (or in the full dataset if covering the full year). This distribution reflects the binary label distribution to be later used for binary classification modeling.

```

SELECT tipped, COUNT(*) AS tip_freq FROM (
  SELECT CASE WHEN (tip_amount > 0) THEN 1 ELSE 0 END AS tipped, tip_amount
  FROM nyctaxi_fare
  WHERE pickup_datetime BETWEEN '20130101' AND '20131231') tc
  GROUP BY tipped

```

Exploration: Tip Class/Range Distribution

This example computes the distribution of tip ranges in a given time period (or in the full dataset if covering the full year). This is the distribution of the label classes that will be used later for multiclass classification modeling.

```

SELECT tip_class, COUNT(*) AS tip_freq FROM (
  SELECT CASE
    WHEN (tip_amount = 0) THEN 0
    WHEN (tip_amount > 0 AND tip_amount <= 5) THEN 1
    WHEN (tip_amount > 5 AND tip_amount <= 10) THEN 2
    WHEN (tip_amount > 10 AND tip_amount <= 20) THEN 3
    ELSE 4
  END AS tip_class
  FROM nyctaxi_fare
  WHERE pickup_datetime BETWEEN '20130101' AND '20131231') tc
  GROUP BY tip_class

```

Exploration: Compute and Compare Trip Distance

This example converts the pickup and drop-off longitude and latitude to SQL geography points, computes the trip distance using SQL geography points difference, and returns a random sample of the results for comparison. The example limits the results to valid coordinates only using the data quality assessment query covered earlier.

```

SELECT
  pickup_location=geography::STPointFromText('POINT(' + pickup_longitude + '' + pickup_latitude + ')', 4326)
,dropoff_location=geography::STPointFromText('POINT(' + dropoff_longitude + '' + dropoff_latitude + ')', 4326)
,trip_distance
,computedist=round(geography::STPointFromText('POINT(' + pickup_longitude + '' + pickup_latitude + ')',
4326).STDistance(geography::STPointFromText('POINT(' + dropoff_longitude + '' + dropoff_latitude + ')', 4326))/1000, 2)
FROM nyctaxi_trip
tablesample(0.01 percent)
WHERE CAST(pickup_latitude AS float) BETWEEN -90 AND 90
AND CAST(dropoff_latitude AS float) BETWEEN -90 AND 90
AND pickup_longitude != '0' AND dropoff_longitude != '0'

```

Feature Engineering in SQL Queries

The label generation and geography conversion exploration queries can also be used to generate labels/features by removing the counting part. Additional feature engineering SQL examples are provided in the [Data Exploration and Feature Engineering in IPython Notebook](#) section. It is more efficient to run the feature generation queries on the full dataset or a large subset of it using SQL queries which run directly on the SQL Server database instance. The queries may be executed in **SQL Server Management Studio**, IPython Notebook or any development

tool/environment which can access the database locally or remotely.

Preparing Data for Model Building

The following query joins the **nyctaxi_trip** and **nyctaxi_fare** tables, generates a binary classification label **tipped**, a multi-class classification label **tip_class**, and extracts a 1% random sample from the full joined dataset. This query can be copied then pasted directly in the [Azure Machine Learning Studio Import Data](#) module for direct data ingestion from the SQL Server database instance in Azure. The query excludes records with incorrect (0, 0) coordinates.

```
SELECT t.*, f.payment_type, f.fare_amount, f.surcharge, f.mta_tax, f.tolls_amount, f.total_amount, f.tip_amount,
CASE WHEN (tip_amount > 0) THEN 1 ELSE 0 END AS tipped,
CASE WHEN (tip_amount = 0) THEN 0
    WHEN (tip_amount > 0 AND tip_amount <= 5) THEN 1
    WHEN (tip_amount > 5 AND tip_amount <= 10) THEN 2
    WHEN (tip_amount > 10 AND tip_amount <= 20) THEN 3
    ELSE 4
END AS tip_class
FROM nyctaxi_trip t,nyctaxi_fare f
TABLESAMPLE(1 percent)
WHERE t.medallion = f.medallion
AND t.hack_license = f.hack_license
AND t.pickup_datetime = f.pickup_datetime
AND pickup_longitude != '0' AND dropoff_longitude != '0'
```

Data Exploration and Feature Engineering in IPython Notebook

In this section, we will perform data exploration and feature generation using both Python and SQL queries against the SQL Server database created earlier. A sample IPython notebook named **machine-Learning-data-science-process-sql-story.ipynb** is provided in the **Sample IPython Notebooks** folder. This notebook is also available on [GitHub](#).

The recommended sequence when working with big data is the following:

- Read in a small sample of the data into an in-memory data frame.
- Perform some visualizations and explorations using the sampled data.
- Experiment with feature engineering using the sampled data.
- For larger data exploration, data manipulation and feature engineering, use Python to issue SQL Queries directly against the SQL Server database in the Azure VM.
- Decide the sample size to use for Azure Machine Learning model building.

When ready to proceed to Azure Machine Learning, you may either:

1. Save the final SQL query to extract and sample the data and copy-paste the query directly into a [Import Data](#) module in Azure Machine Learning. This method is demonstrated in the [Building Models in Azure Machine Learning](#) section.
2. Persist the sampled and engineered data you plan to use for model building in a new database table, then use the new table in the [Import Data](#) module.

The following are a few data exploration, data visualization, and feature engineering examples. For more examples, see the sample SQL IPython notebook in the **Sample IPython Notebooks** folder.

Initialize Database Credentials

Initialize your database connection settings in the following variables:

```
SERVER_NAME=<server name>
DATABASE_NAME=<database name>
USERID=<user name>
PASSWORD=<password>
DB_DRIVER =<database server>
```

Create Database Connection

```
CONNECTION_STRING='DRIVER=
{+DRIVER+};SERVER='+SERVER_NAME+';DATABASE='+DATABASE_NAME+';UID='+USERID+';PWD='+PASSWORD
conn = pyodbc.connect(CONNECTION_STRING)
```

Report number of rows and columns in table nyctaxi_trip

```
nrows = pd.read_sql("""
    SELECT SUM(rows) FROM sys.partitions
    WHERE object_id = OBJECT_ID('nyctaxi_trip')
    """, conn)

print 'Total number of rows = %d' % nrows.iloc[0,0]

ncols = pd.read_sql("""
    SELECT COUNT(*) FROM information_schema.columns
    WHERE table_name = ('nyctaxi_trip')
    """, conn)

print 'Total number of columns = %d' % ncols.iloc[0,0]
```

- Total number of rows = 173179759
- Total number of columns = 14

Read-in a small data sample from the SQL Server Database

```
t0 = time.time()

query = """
SELECT t.*, f.payment_type, f.fare_amount, f.surcharge, f.mta_tax,
       f.tolls_amount, f.total_amount, f.tip_amount
  FROM nyctaxi_trip t, nyctaxi_fare f
 TABLESAMPLE (0.05 PERCENT)
 WHERE t.medallion = f.medallion
   AND t.hack_license = f.hack_license
   AND t.pickup_datetime = f.pickup_datetime
"""

df1 = pd.read_sql(query, conn)

t1 = time.time()
print 'Time to read the sample table is %f seconds' % (t1-t0)

print 'Number of rows and columns retrieved = (%d, %d)' % (df1.shape[0], df1.shape[1])
```

Time to read the sample table is 6.492000 seconds

Number of rows and columns retrieved = (84952, 21)

Descriptive Statistics

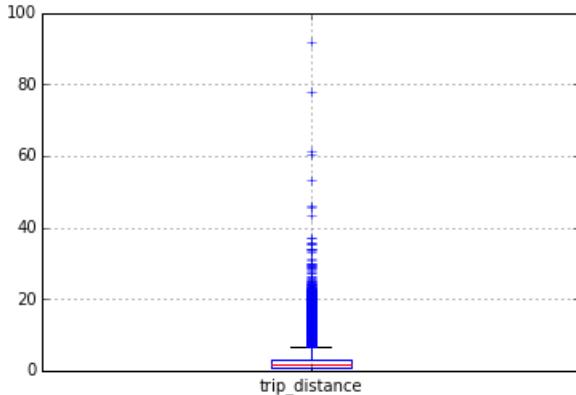
Now are ready to explore the sampled data. We start with looking at descriptive statistics for the **trip_distance** (or any other) field(s):

```
df1['trip_distance'].describe()
```

Visualization: Box Plot Example

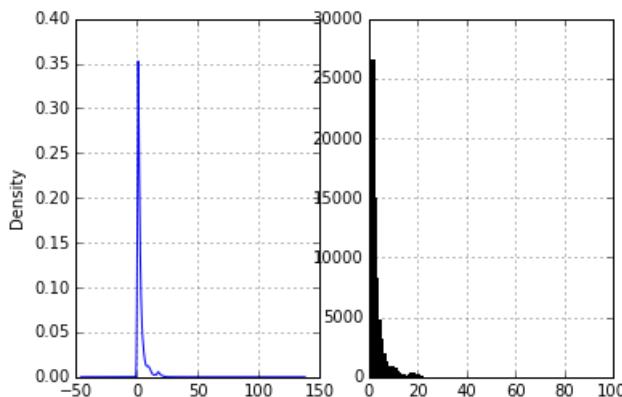
Next we look at the box plot for the trip distance to visualize the quantiles

```
df1.boxplot(column='trip_distance',return_type='dict')
```



Visualization: Distribution Plot Example

```
fig = plt.figure()
ax1 = fig.add_subplot(1,2,1)
ax2 = fig.add_subplot(1,2,2)
df1['trip_distance'].plot(ax=ax1,kind='kde', style='b-')
df1['trip_distance'].hist(ax=ax2, bins=100, color='k')
```



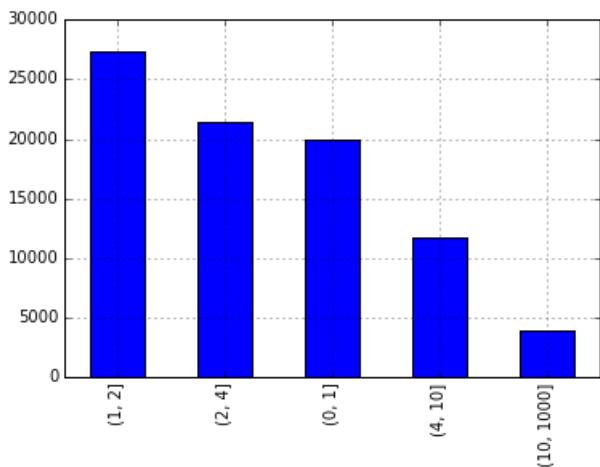
Visualization: Bar and Line Plots

In this example, we bin the trip distance into five bins and visualize the binning results.

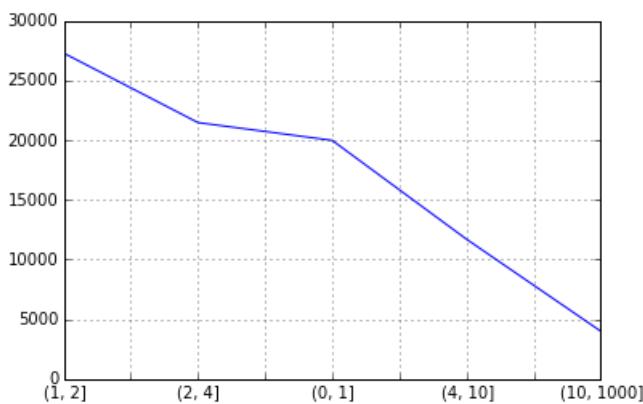
```
trip_dist_bins = [0, 1, 2, 4, 10, 1000]
df1['trip_distance']
trip_dist_bin_id = pd.cut(df1['trip_distance'], trip_dist_bins)
trip_dist_bin_id
```

We can plot the above bin distribution in a bar or line plot as below

```
pd.Series(trip_dist_bin_id.value_counts()).plot(kind='bar')
```



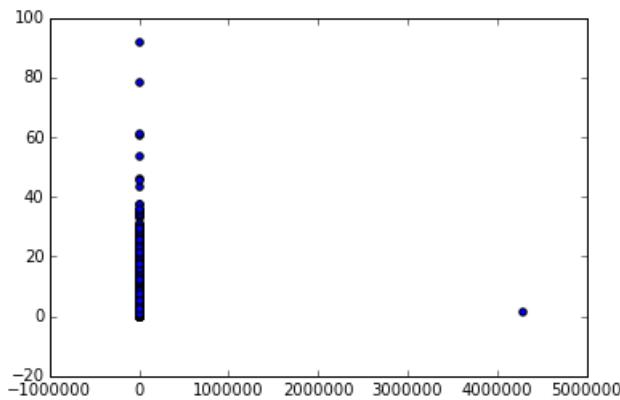
```
pd.Series(trip_dist_bin_id).value_counts().plot(kind='line')
```



Visualization: Scatterplot Example

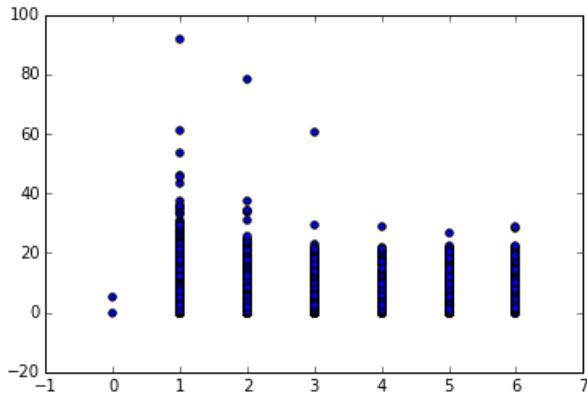
We show scatter plot between **trip_time_in_secs** and **trip_distance** to see if there is any correlation

```
plt.scatter(dfl['trip_time_in_secs'], dfl['trip_distance'])
```



Similarly we can check the relationship between **rate_code** and **trip_distance**.

```
plt.scatter(dfl['passenger_count'], dfl['trip_distance'])
```



Sub-Sampling the Data in SQL

When preparing data for model building in [Azure Machine Learning Studio](#), you may either decide on the **SQL query to use directly in the Import Data module** or persist the engineered and sampled data in a new table, which you could use in the [Import Data](#) module with a simple **SELECT * FROM**.

In this section we will create a new table to hold the sampled and engineered data. An example of a direct SQL query for model building is provided in the [Data Exploration and Feature Engineering in SQL Server](#) section.

Create a Sample Table and Populate with 1% of the Joined Tables. Drop Table First if it Exists.

In this section, we join the tables **nyctaxi_trip** and **nyctaxi_fare**, extract a 1% random sample, and persist the sampled data in a new table name **nyctaxi_one_percent**:

```
cursor = conn.cursor()

drop_table_if_exists = """
IF OBJECT_ID('nyctaxi_one_percent', 'U') IS NOT NULL DROP TABLE nyctaxi_one_percent
"""

nyctaxi_one_percent_insert = """
SELECT t.* , f.payment_type, f.fare_amount, f.surcharge, f.mta_tax, f.tolls_amount, f.total_amount, f.tip_amount
INTO nyctaxi_one_percent
FROM nyctaxi_trip t, nyctaxi_fare f
TABLESAMPLE (1 PERCENT)
WHERE t.medallion = f.medallion
AND t.hack_license = f.hack_license
AND t.pickup_datetime = f.pickup_datetime
AND pickup_longitude <> '0' AND dropoff_longitude <> '0'
"""

cursor.execute(drop_table_if_exists)
cursor.execute(nyctaxi_one_percent_insert)
cursor.commit()
```

Data Exploration using SQL Queries in IPython Notebook

In this section, we explore data distributions using the 1% sampled data which is persisted in the new table we created above. Note that similar explorations can be performed using the original tables, optionally using **TABLESAMPLE** to limit the exploration sample or by limiting the results to a given time period using the **pickup_datetime** partitions, as illustrated in the [Data Exploration and Feature Engineering in SQL Server](#) section.

Exploration: Daily distribution of trips

```
query = """
SELECT CONVERT(date, dropoff_datetime) AS date, COUNT(*) AS c
FROM nyctaxi_one_percent
GROUP BY CONVERT(date, dropoff_datetime)
"""

pd.read_sql(query, conn)
```

Exploration: Trip distribution per medallion

```
query = ""  
SELECT medallion,count(*) AS c  
FROM nyctaxi_one_percent  
GROUP BY medallion  
"  
  
pd.read_sql(query,conn)
```

Feature Generation Using SQL Queries in IPython Notebook

In this section we will generate new labels and features directly using SQL queries, operating on the 1% sample table we created in the previous section.

Label Generation: Generate Class Labels

In the following example, we generate two sets of labels to use for modeling:

1. Binary Class Labels **tipped** (predicting if a tip will be given)
2. Multiclass Labels **tip_class** (predicting the tip bin or range)

```
nyctaxi_one_percent_add_col=""  
ALTER TABLE nyctaxi_one_percent ADD tipped bit, tip_class int  
"  
  
cursor.execute(nyctaxi_one_percent_add_col)  
cursor.commit()  
  
nyctaxi_one_percent_update_col=""  
UPDATE nyctaxi_one_percent  
SET  
tipped = CASE WHEN (tip_amount > 0) THEN 1 ELSE 0 END,  
tip_class = CASE WHEN (tip_amount = 0) THEN 0  
WHEN (tip_amount > 0 AND tip_amount <= 5) THEN 1  
WHEN (tip_amount > 5 AND tip_amount <= 10) THEN 2  
WHEN (tip_amount > 10 AND tip_amount <= 20) THEN 3  
ELSE 4  
END  
"  
  
cursor.execute(nyctaxi_one_percent_update_col)  
cursor.commit()
```

Feature Engineering: Count Features for Categorical Columns

This example transforms a categorical field into a numeric field by replacing each category with the count of its occurrences in the data.

```

nyctaxi_one_percent_insert_col=""
    ALTER TABLE nyctaxi_one_percent ADD cmt_count int, vts_count int
    ""

cursor.execute(nyctaxi_one_percent_insert_col)
cursor.commit()

nyctaxi_one_percent_update_col=""
    WITH B AS
    (
        SELECT medallion, hack_license,
            SUM(CASE WHEN vendor_id = 'cmt' THEN 1 ELSE 0 END) AS cmt_count,
            SUM(CASE WHEN vendor_id = 'vts' THEN 1 ELSE 0 END) AS vts_count
        FROM nyctaxi_one_percent
        GROUP BY medallion, hack_license
    )
    UPDATE nyctaxi_one_percent
    SET nyctaxi_one_percent.cmt_count = B.cmt_count,
        nyctaxi_one_percent.vts_count = B.vts_count
    FROM nyctaxi_one_percent A INNER JOIN B
    ON A.medallion = B.medallion AND A.hack_license = B.hack_license
    ""

cursor.execute(nyctaxi_one_percent_update_col)
cursor.commit()

```

Feature Engineering: Bin features for Numerical Columns

This example transforms a continuous numeric field into preset category ranges, i.e., transform numeric field into a categorical field.

```

nyctaxi_one_percent_insert_col=""
    ALTER TABLE nyctaxi_one_percent ADD trip_time_bin int
    ""

cursor.execute(nyctaxi_one_percent_insert_col)
cursor.commit()

nyctaxi_one_percent_update_col=""
    WITH B(medallion,hack_license,pickup_datetime,trip_time_in_secs, BinNumber) AS
    (
        SELECT medallion,hack_license,pickup_datetime,trip_time_in_secs,
            NTILE(5) OVER (ORDER BY trip_time_in_secs) AS BinNumber from nyctaxi_one_percent
    )
    UPDATE nyctaxi_one_percent
    SET trip_time_bin = B.BinNumber
    FROM nyctaxi_one_percent A INNER JOIN B
    ON A.medallion = B.medallion
    AND A.hack_license = B.hack_license
    AND A.pickup_datetime = B.pickup_datetime
    ""

cursor.execute(nyctaxi_one_percent_update_col)
cursor.commit()

```

Feature Engineering: Extract Location Features from Decimal Latitude/Longitude

This example breaks down the decimal representation of a latitude and/or longitude field into multiple region fields of different granularity, such as, country, city, town, block, etc. Note that the new geo-fields are not mapped to actual locations. For information on mapping geocode locations, see [Bing Maps REST Services](#).

```

nyctaxi_one_percent_insert_col=""
    ALTER TABLE nyctaxi_one_percent
    ADD l1 varchar(6), l2 varchar(3), l3 varchar(3), l4 varchar(3),
    l5 varchar(3), l6 varchar(3), l7 varchar(3)
    ""

cursor.execute(nyctaxi_one_percent_insert_col)
cursor.commit()

nyctaxi_one_percent_update_col=""
    UPDATE nyctaxi_one_percent
    SET l1=round(pickup_longitude,0)
        , l2 = CASE WHEN LEN(PARSENNAME(ROUND(ABS(pickup_longitude) - FLOOR(ABS(pickup_longitude)),6),1)) >= 1 THEN
        SUBSTRING(PARSENNAME(ROUND(ABS(pickup_longitude) - FLOOR(ABS(pickup_longitude)),6),1),1,1) ELSE '0' END
        , l3 = CASE WHEN LEN(PARSENNAME(ROUND(ABS(pickup_longitude) - FLOOR(ABS(pickup_longitude)),6),1),2,1) >= 2 THEN
        SUBSTRING(PARSENNAME(ROUND(ABS(pickup_longitude) - FLOOR(ABS(pickup_longitude)),6),1),2,1) ELSE '0' END
        , l4 = CASE WHEN LEN(PARSENNAME(ROUND(ABS(pickup_longitude) - FLOOR(ABS(pickup_longitude)),6),1),3,1) >= 3 THEN
        SUBSTRING(PARSENNAME(ROUND(ABS(pickup_longitude) - FLOOR(ABS(pickup_longitude)),6),1),3,1) ELSE '0' END
        , l5 = CASE WHEN LEN(PARSENNAME(ROUND(ABS(pickup_longitude) - FLOOR(ABS(pickup_longitude)),6),1),4,1) >= 4 THEN
        SUBSTRING(PARSENNAME(ROUND(ABS(pickup_longitude) - FLOOR(ABS(pickup_longitude)),6),1),4,1) ELSE '0' END
        , l6 = CASE WHEN LEN(PARSENNAME(ROUND(ABS(pickup_longitude) - FLOOR(ABS(pickup_longitude)),6),1),5,1) >= 5 THEN
        SUBSTRING(PARSENNAME(ROUND(ABS(pickup_longitude) - FLOOR(ABS(pickup_longitude)),6),1),5,1) ELSE '0' END
        , l7 = CASE WHEN LEN(PARSENNAME(ROUND(ABS(pickup_longitude) - FLOOR(ABS(pickup_longitude)),6),1),6,1) >= 6 THEN
        SUBSTRING(PARSENNAME(ROUND(ABS(pickup_longitude) - FLOOR(ABS(pickup_longitude)),6),1),6,1) ELSE '0' END
    ""

cursor.execute(nyctaxi_one_percent_update_col)
cursor.commit()

```

Verify the final form of the featurized table

```

query = "SELECT TOP 100 * FROM nyctaxi_one_percent"
pd.read_sql(query,conn)

```

We are now ready to proceed to model building and model deployment in [Azure Machine Learning](#). The data is ready for any of the prediction problems identified earlier, namely:

1. Binary classification: To predict whether or not a tip was paid for a trip.
2. Multiclass classification: To predict the range of tip paid, according to the previously defined classes.
3. Regression task: To predict the amount of tip paid for a trip.

Building Models in Azure Machine Learning

To begin the modeling exercise, log in to your Azure Machine Learning workspace. If you have not yet created a machine learning workspace, see [Create an Azure Machine Learning workspace](#).

1. To get started with Azure Machine Learning, see [What is Azure Machine Learning Studio?](#)
2. Log in to [Azure Machine Learning Studio](#).
3. The Studio Home page provides a wealth of information, videos, tutorials, links to the Modules Reference, and other resources. For more information about Azure Machine Learning, consult the [Azure Machine Learning Documentation Center](#).

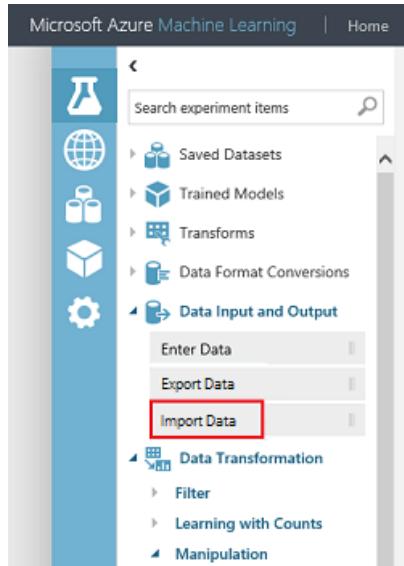
A typical training experiment consists of the following:

1. Create a **+NEW** experiment.
2. Get the data to Azure Machine Learning.
3. Pre-process, transform and manipulate the data as needed.
4. Generate features as needed.
5. Split the data into training/validation/testing datasets(or have separate datasets for each).

6. Select one or more machine learning algorithms depending on the learning problem to solve. E.g., binary classification, multiclass classification, regression.
7. Train one or more models using the training dataset.
8. Score the validation dataset using the trained model(s).
9. Evaluate the model(s) to compute the relevant metrics for the learning problem.
10. Fine tune the model(s) and select the best model to deploy.

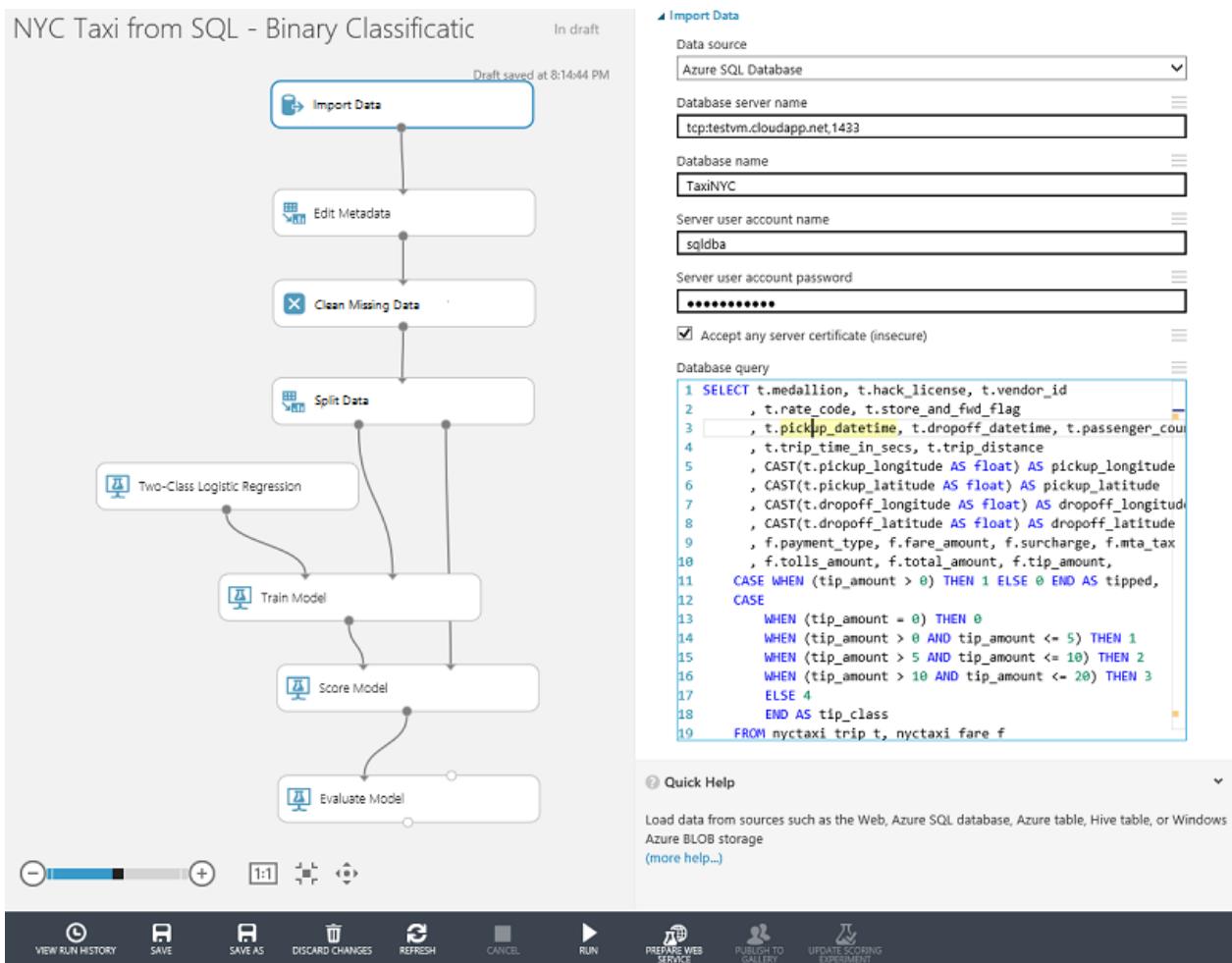
In this exercise, we have already explored and engineered the data in SQL Server, and decided on the sample size to ingest in Azure Machine Learning. To build one or more of the prediction models we decided:

1. Get the data to Azure Machine Learning using the [Import Data](#) module, available in the **Data Input and Output** section. For more information, see the [Import Data](#) module reference page.



2. Select **Azure SQL Database** as the **Data source** in the **Properties** panel.
3. Enter the database DNS name in the **Database server name** field. Format:
`tcp:<your_virtual_machine_DNS_name>.1433`
4. Enter the **Database name** in the corresponding field.
5. Enter the **SQL user name** in the **Server user account name**, and the password in the **Server user account password**.
6. Check **Accept any server certificate** option.
7. In the **Database query** edit text area, paste the query which extracts the necessary database fields (including any computed fields such as the labels) and down samples the data to the desired sample size.

An example of a binary classification experiment reading data directly from the SQL Server database is in the figure below. Similar experiments can be constructed for multiclass classification and regression problems.



IMPORTANT

In the modeling data extraction and sampling query examples provided in previous sections, **all labels for the three modeling exercises are included in the query**. An important (required) step in each of the modeling exercises is to **exclude** the unnecessary labels for the other two problems, and any other **target leaks**. For e.g., when using binary classification, use the label **tipped** and exclude the fields **tip_class**, **tip_amount**, and **total_amount**. The latter are target leaks since they imply the tip paid.

To exclude unnecessary columns and/or target leaks, you may use the [Select Columns in Dataset](#) module or the [Edit Metadata](#). For more information, see [Select Columns in Dataset](#) and [Edit Metadata](#) reference pages.

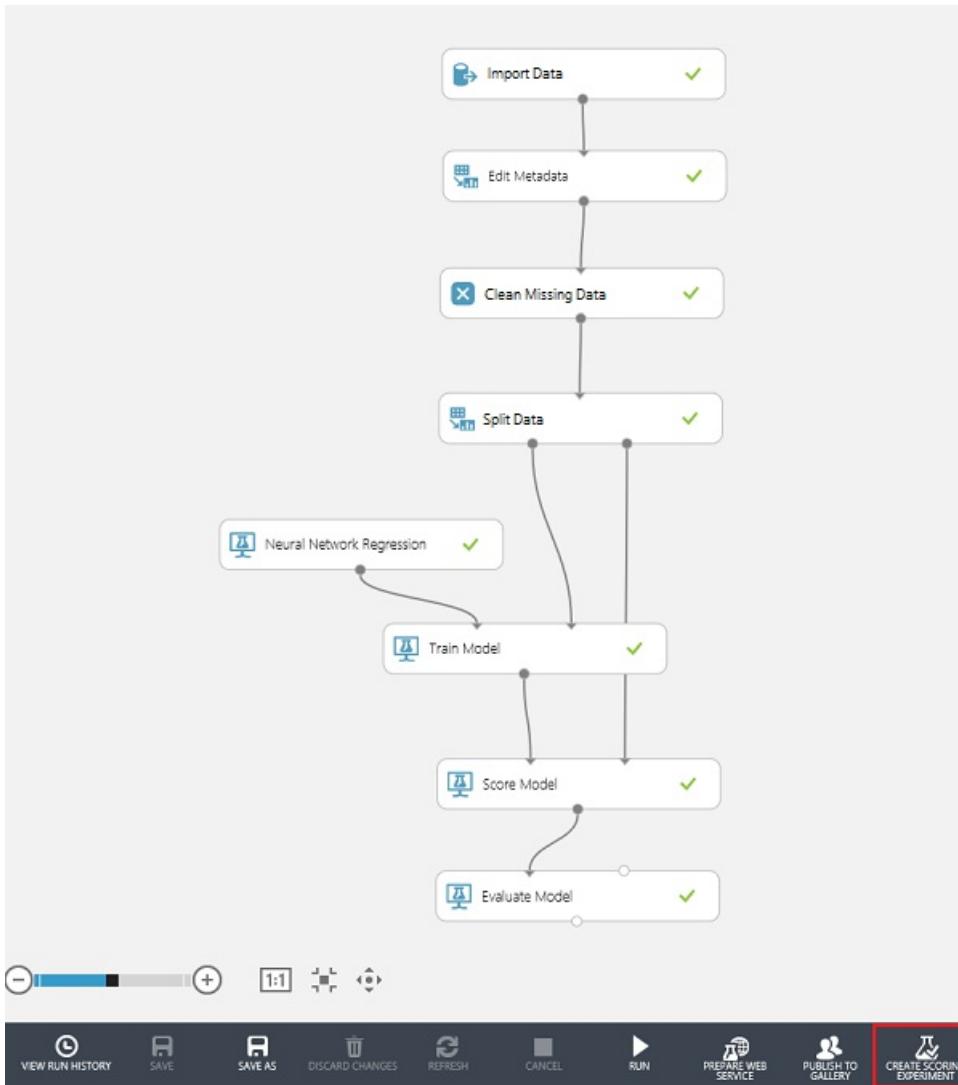
Deploying Models in Azure Machine Learning

When your model is ready, you can easily deploy it as a web service directly from the experiment. For more information about deploying Azure Machine Learning web services, see [Deploy an Azure Machine Learning web service](#).

To deploy a new web service, you need to:

1. Create a scoring experiment.
2. Deploy the web service.

To create a scoring experiment from a **Finished** training experiment, click **CREATE SCORING EXPERIMENT** in the lower action bar.

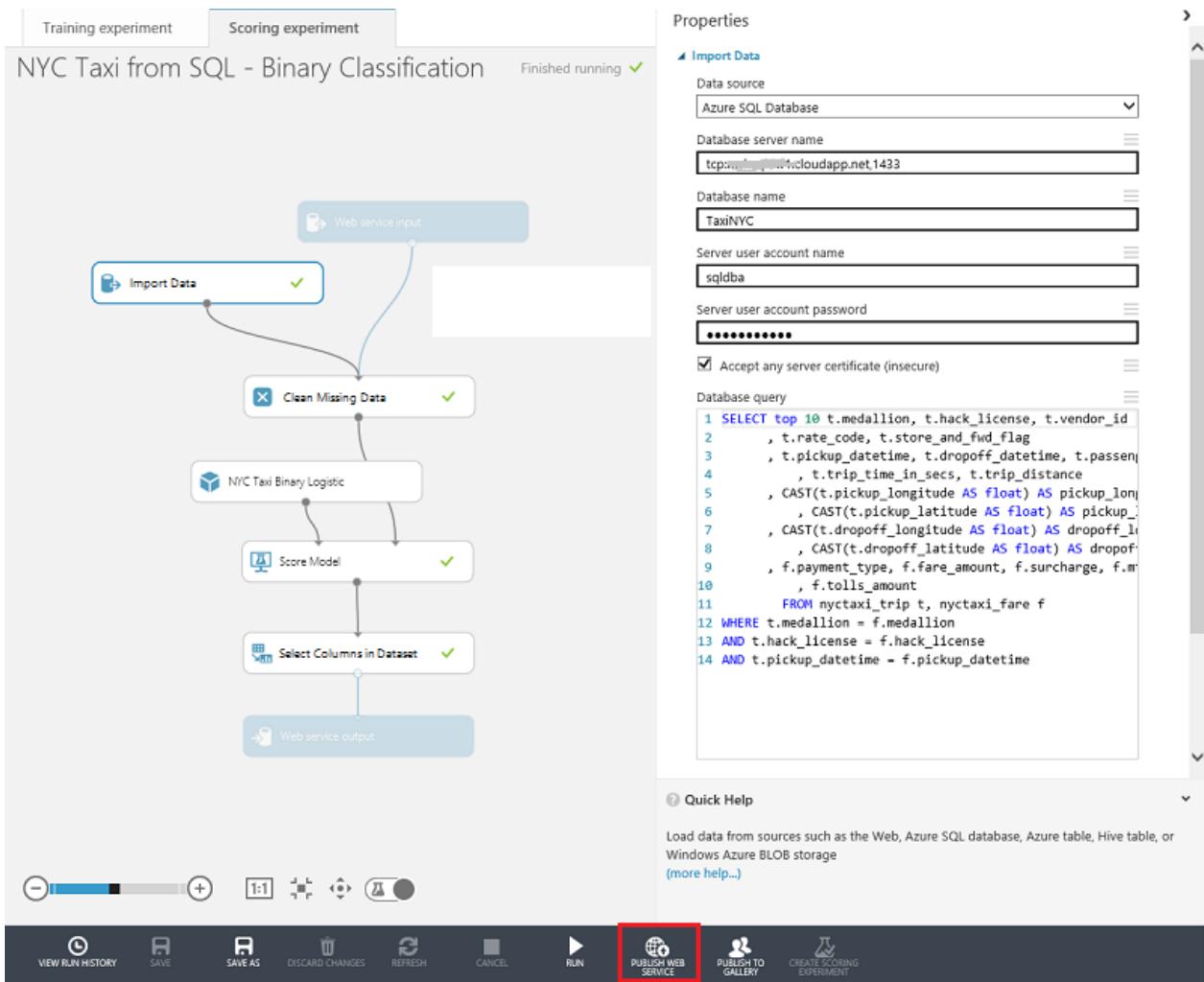


Azure Machine Learning will attempt to create a scoring experiment based on the components of the training experiment. In particular, it will:

1. Save the trained model and remove the model training modules.
2. Identify a logical **input port** to represent the expected input data schema.
3. Identify a logical **output port** to represent the expected web service output schema.

When the scoring experiment is created, review it and adjust as needed. A typical adjustment is to replace the input dataset and/or query with one which excludes label fields, as these will not be available when the service is called. It is also a good practice to reduce the size of the input dataset and/or query to a few records, just enough to indicate the input schema. For the output port, it is common to exclude all input fields and only include the **Scored Labels** and **Scored Probabilities** in the output using the [Select Columns in Dataset](#) module.

A sample scoring experiment is in the figure below. When ready to deploy, click the **PUBLISH WEB SERVICE** button in the lower action bar.



To recap, in this walkthrough tutorial, you have created an Azure data science environment, worked with a large public dataset all the way from data acquisition to model training and deploying of an Azure Machine Learning web service.

License Information

This sample walkthrough and its accompanying scripts and IPython notebook(s) are shared by Microsoft under the MIT license. Please check the LICENSE.txt file in the directory of the sample code on GitHub for more details.

References

- [Andrés Monroy NYC Taxi Trips Download Page](#)
- [FOILing NYC's Taxi Trip Data by Chris Whong](#)
- [NYC Taxi and Limousine Commission Research and Statistics](#)

The Team Data Science Process in action: using SQL Data Warehouse

1/17/2017 • 28 min to read • [Edit on GitHub](#)

In this tutorial, we walk you through building and deploying a machine learning model using SQL Data Warehouse (SQL DW) for a publicly available dataset -- the [NYC Taxi Trips](#) dataset. The binary classification model constructed predicts whether or not a tip is paid for a trip, and models for multiclass classification and regression are also discussed that predict the distribution for the tip amounts paid.

The procedure follows the [Team Data Science Process \(TDSP\)](#) workflow. We show how to setup a data science environment, how to load the data into SQL DW, and how use either SQL DW or an IPython Notebook to explore the data and engineer features to model. We then show how to build and deploy a model with Azure Machine Learning.

The NYC Taxi Trips dataset

The NYC Taxi Trip data consists of about 20GB of compressed CSV files (~48GB uncompressed), recording more than 173 million individual trips and the fares paid for each trip. Each trip record includes the pickup and drop-off locations and times, anonymized hack (driver's) license number, and the medallion (taxi's unique id) number. The data covers all trips in the year 2013 and is provided in the following two datasets for each month:

1. The **trip_data.csv** file contains trip details, such as number of passengers, pickup and dropoff points, trip duration, and trip length. Here are a few sample records:

```
medallion,hack_license,vendor_id,rate_code,store_and_fwd_flag,pickup_datetime,dropoff_datetime,passenger_count,trip_time_in_secs,trip_distance,pickup_longitude,pickup_latitude,dropoff_longitude,dropoff_latitude
89D227B655E5C82AECF13C3F540D4CF4,BA96DE419E711691B9445D6A6307C170,CMT,1,N,2013-01-01 15:11:48,2013-01-01 15:18:10,4,382,1.00,-73.978165,40.757977,-73.989838,40.751171
0BD7C8F5BA12B88E0B67BED28BEA73D8,9FD8F69F0804BDB5549F40E9DA1BE472,CMT,1,N,2013-01-06 00:18:35,2013-01-06 00:22:54,1,259,1.50,-74.006683,40.731781,-73.994499,40.75066
0BD7C8F5BA12B88E0B67BED28BEA73D8,9FD8F69F0804BDB5549F40E9DA1BE472,CMT,1,N,2013-01-05 18:49:41,2013-01-05 18:54:23,1,282,1.10,-74.004707,40.73777,-74.009834,40.726002
DFD2202EE08F7A8DC9A57B02ACB81FE2,51EE87E3205C985EF8431D850C786310,CMT,1,N,2013-01-07 23:54:15,2013-01-07 23:58:20,2,244,.70,-73.974602,40.759945,-73.984734,40.759388
DFD2202EE08F7A8DC9A57B02ACB81FE2,51EE87E3205C985EF8431D850C786310,CMT,1,N,2013-01-07 23:25:03,2013-01-07 23:34:24,1,560,2.10,-73.97625,40.748528,-74.002586,40.747868
```

2. The **trip_fare.csv** file contains details of the fare paid for each trip, such as payment type, fare amount, surcharge and taxes, tips and tolls, and the total amount paid. Here are a few sample records:

```
medallion,hack_license,vendor_id,pickup_datetime,payment_type,fare_amount,surcharge,mta_tax,tip_amount,tolls_amount,total_amount
89D227B655E5C82AECF13C3F540D4CF4,BA96DE419E711691B9445D6A6307C170,CMT,2013-01-01 15:11:48,CSH,6.5,0,0.5,0,0,0.7
0BD7C8F5BA12B88E0B67BED28BEA73D8,9FD8F69F0804BDB5549F40E9DA1BE472,CMT,2013-01-06 00:18:35,CSH,6.0,0.5,0.5,0,0.7
0BD7C8F5BA12B88E0B67BED28BEA73D8,9FD8F69F0804BDB5549F40E9DA1BE472,CMT,2013-01-05 18:49:41,CSH,5.5,1,0.5,0,0.7
DFD2202EE08F7A8DC9A57B02ACB81FE2,51EE87E3205C985EF8431D850C786310,CMT,2013-01-07 23:54:15,CSH,5.0,0.5,0.5,0,0.6
DFD2202EE08F7A8DC9A57B02ACB81FE2,51EE87E3205C985EF8431D850C786310,CMT,2013-01-07 23:25:03,CSH,9.5,0.5,0.5,0,10.5
```

The **unique key** used to join `trip_data` and `trip_fare` is composed of the following three fields:

- `medallion`,
- `hack_license` and
- `pickup_datetime`.

Address three types of prediction tasks

We formulate three prediction problems based on the *tip_amount* to illustrate three kinds of modeling tasks:

1. **Binary classification:** To predict whether or not a tip was paid for a trip, i.e. a *tip_amount* that is greater than \$0 is a positive example, while a *tip_amount* of \$0 is a negative example.
2. **Multiclass classification:** To predict the range of tip paid for the trip. We divide the *tip_amount* into five bins or classes:

```
Class 0 : tip_amount = $0  
Class 1 : tip_amount > $0 and tip_amount <= $5  
Class 2 : tip_amount > $5 and tip_amount <= $10  
Class 3 : tip_amount > $10 and tip_amount <= $20  
Class 4 : tip_amount > $20
```

3. **Regression task:** To predict the amount of tip paid for a trip.

Set up the Azure data science environment for advanced analytics

To set up your Azure Data Science environment, follow these steps.

Create your own Azure blob storage account

- When you provision your own Azure blob storage, choose a geo-location for your Azure blob storage in or as close as possible to **South Central US**, which is where the NYC Taxi data is stored. The data will be copied using AzCopy from the public blob storage container to a container in your own storage account. The closer your Azure blob storage is to South Central US, the faster this task (Step 4) will be completed.
- To create your own Azure storage account, follow the steps outlined at [About Azure storage accounts](#). Be sure to make notes on the values for following storage account credentials as they will be needed later in this walkthrough.
 - **Storage Account Name**
 - **Storage Account Key**
 - **Container Name** (which you want the data to be stored in the Azure blob storage)

Provision your Azure SQL DW instance. Follow the documentation at [Create a SQL Data Warehouse](#) to provision a SQL Data Warehouse instance. Make sure that you make notations on the following SQL Data Warehouse credentials which will be used in later steps.

- **Server Name:** .database.windows.net
- **SQLDW (Database) Name**
- **Username**
- **Password**

Install Visual Studio 2015 and SQL Server Data Tools. For instructions, see [Install Visual Studio 2015 and/or SSDT \(SQL Server Data Tools\) for SQL Data Warehouse](#).

Connect to your Azure SQL DW with Visual Studio. For instructions, see steps 1 & 2 in [Connect to Azure SQL Data Warehouse with Visual Studio](#).

NOTE

Run the following SQL query on the database you created in your SQL Data Warehouse (instead of the query provided in step 3 of the connect topic,) to **create a master key**.

```

BEGIN TRY
    --Try to create the master key
    CREATE MASTER KEY
END TRY
BEGIN CATCH
    --If the master key exists, do nothing
END CATCH;

```

Create an Azure Machine Learning workspace under your Azure subscription. For instructions, see [Create an Azure Machine Learning workspace](#).

Load the data into SQL Data Warehouse

Open a Windows PowerShell command console. Run the following PowerShell commands to download the example SQL script files that we share with you on Github to a local directory that you specify with the parameter *-DestDir*. You can change the value of parameter *-DestDir* to any local directory. If *-DestDir* does not exist, it will be created by the PowerShell script.

NOTE

You might need to **Run as Administrator** when executing the following PowerShell script if your *DestDir* directory needs Administrator privilege to create or to write to it.

```

$source = "https://raw.githubusercontent.com/Azure/Azure-MachineLearning-
DataScience/master/Misc/SQLDW/Download_Scripts_SQLDW_Walkthrough.ps1"
$ps1_dest = "$pwd\Download_Scripts_SQLDW_Walkthrough.ps1"
$wc = New-Object System.Net.WebClient
$wc.DownloadFile($source, $ps1_dest)
.\Download_Scripts_SQLDW_Walkthrough.ps1 -DestDir 'C:\tempSQLDW'

```

After successful execution, your current working directory changes to *-DestDir*. You should be able to see screen like below:

```

PS C:\> $source = "https://raw.githubusercontent.com/Azure/Azure-MachineLearning-DataScience/master/Misc/SQLDW/Download_Scripts_SQLDW_Walkthrough.ps1"
PS C:\> $ps1_dest = New-Object System.Net.WebClient
PS C:\> $wc = New-Object System.Net.WebClient
PS C:\> $wc.DownloadFile($source, $ps1_dest)
PS C:\> .\Download_Scripts_SQLDW_Walkthrough.ps1 -DestDir 'C:\tempSQLDW'
PS C:\> cd 'C:\tempSQLDW'
PS C:\tempSQLDW> dir
https://raw.githubusercontent.com/Azure/Azure-MachineLearning-DataScience/master/Misc/SQLDW/LoadDataToSQLDW.ps1
https://raw.githubusercontent.com/Azure/Azure-MachineLearning-DataScience/master/Misc/SQLDW/DeleteResourcesOnSQLDW.sql
<https://raw.githubusercontent.com/Azure/Azure-MachineLearning-DataScience/master/Misc/SQLDW/DeleteResourcesOnSQLDW.sql>
https://raw.githubusercontent.com/Azure/Azure-MachineLearning-DataScience/master/Misc/SQLDW/SQLDW_Explorations.sql
<https://raw.githubusercontent.com/Azure/Azure-MachineLearning-DataScience/master/Misc/SQLDW/SQLDW_Explorations.sql>
https://raw.githubusercontent.com/Azure/Azure-MachineLearning-DataScience/master/Misc/SQLDW/SQLDW_Explorations.ipynb
C:\tempSQLDW> ipynb
https://raw.githubusercontent.com/Azure/Azure-MachineLearning-DataScience/master/Misc/SQLDW/SQLDW_Explorations.ipynb
C:\tempSQLDW> ipynb
https://raw.githubusercontent.com/Azure/Azure-MachineLearning-DataScience/master/Misc/SQLDW/SQLDW_Explorations_Scripts.py
C:\tempSQLDW> Scripts.py
Fetching the sample script files completed.
Now entering the destination directory C:\tempSQLDW.
PS C:\tempSQLDW>

```

In your *-DestDir*, execute the following PowerShell script in administrator mode:

```
./SQLDW_Data_Import.ps1
```

When the PowerShell script runs for the first time, you will be asked to input the information from your Azure SQL DW and your Azure blob storage account. When this PowerShell script completes running for the first time, the credentials you input will have been written to a configuration file SQLDW.conf in the present working directory. The future run of this PowerShell script file has the option to read all needed parameters from this configuration file. If you need to change some parameters, you can choose to input the parameters on the screen upon prompt by deleting this configuration file and inputting the parameters values as prompted or to change the parameter values by editing the SQLDW.conf file in your *-DestDir* directory.

NOTE

In order to avoid schema name conflicts with those that already exist in your Azure SQL DW, when reading parameters directly from the SSQLDW.conf file, a 3-digit random number is added to the schema name from the SSQLDW.conf file as the default schema name for each run. The PowerShell script may prompt you for a schema name: the name may be specified at user discretion.

This **PowerShell script** file completes the following tasks:

- **Downloads and installs AzCopy**, if AzCopy is not already installed

```
$AzCopy_path = SearchAzCopy
if($AzCopy_path -eq $null){
    Write-Host "AzCopy.exe is not found in C:\Program Files*. Now, start installing AzCopy..." -ForegroundColor "Yellow"
    InstallAzCopy
    $AzCopy_path = SearchAzCopy
}
$env_path = $env:Path
for ($i=0; $i -lt $AzCopy_path.count; $i++){
    if($AzCopy_path.count -eq 1){
        $AzCopy_path_i = $AzCopy_path
    } else {
        $AzCopy_path_i = $AzCopy_path[$i]
    }
    if($env_path -notlike '*'+$AzCopy_path_i+'*'){
        Write-Host $AzCopy_path_i 'not in system path, add it...'
        [Environment]::SetEnvironmentVariable("Path", "$AzCopy_path_i;$env_path", "Machine")
        $env:Path = [System.Environment]::GetEnvironmentVariable("Path","Machine")
        $env_path = $env:Path
    }
}
```

- **Copies data to your private blob storage account** from the public blob with AzCopy

```
Write-Host "AzCopy is copying data from public blob to yo storage account. It may take a while..." -ForegroundColor "Yellow"
$start_time = Get-Date
AzCopy.exe /Source:$Source /Dest:$DestURL /DestKey:$StorageAccountKey /S
$end_time = Get-Date
$time_span = $end_time - $start_time
$total_seconds = [math]::Round($time_span.TotalSeconds,2)
Write-Host "AzCopy finished copying data. Please check your storage account to verify." -ForegroundColor "Yellow"
Write-Host "This step (copying data from public blob to your storage account) takes $total_seconds seconds." -ForegroundColor "Green"
```

- **Loads data using Polybase (by executing LoadDataToSQLDW.sql) to your Azure SQL DW** from your private blob storage account with the following commands.

- Create a schema

```
EXEC ("CREATE SCHEMA {schemaname};");
```

- Create a database scoped credential

```
CREATE DATABASE SCOPED CREDENTIAL {KeyAlias}
WITH IDENTITY="asbkey",
Secret ="{StorageAccountKey}"
```

- Create an external data source for an Azure storage blob

```
CREATE EXTERNAL DATA SOURCE {nyctaxi_trip_storage}
WITH
(
    TYPE = HADOOP,
    LOCATION = "wasbs://{ContainerName}@{StorageAccountName}.blob.core.windows.net",
    CREDENTIAL = {KeyAlias}
)
;

CREATE EXTERNAL DATA SOURCE {nyctaxi_fare_storage}
WITH
(
    TYPE = HADOOP,
    LOCATION = "wasbs://{ContainerName}@{StorageAccountName}.blob.core.windows.net",
    CREDENTIAL = {KeyAlias}
)
;
```

- o Create an external file format for a csv file. Data is uncompressed and fields are separated with the pipe character.

```
CREATE EXTERNAL FILE FORMAT {csv_file_format}
WITH
(
    FORMAT_TYPE = DELIMITEDTEXT,
    FORMAT_OPTIONS
    (
        FIELD_TERMINATOR = ",",
        USE_TYPE_DEFAULT = TRUE
    )
)
;
```

- o Create external fare and trip tables for NYC taxi dataset in Azure blob storage.

```

CREATE EXTERNAL TABLE {external_nyctaxi_fare}
(
    medallion varchar(50) not null,
    hack_license varchar(50) not null,
    vendor_id char(3),
    pickup_datetime datetime not null,
    payment_type char(3),
    fare_amount float,
    surcharge float,
    mta_tax float,
    tip_amount float,
    tolls_amount float,
    total_amount float
)
with (
    LOCATION = "/nyctaxifare/",
    DATA_SOURCE= {nyctaxi_fare_storage},
    FILE_FORMAT = {csv_file_format},
    REJECT_TYPE= VALUE,
    REJECT_VALUE= 12
)

CREATE EXTERNAL TABLE {external_nyctaxi_trip}
(
    medallion varchar(50) not null,
    hack_license varchar(50) not null,
    vendor_id char(3),
    rate_code char(3),
    store_and_fwd_flag char(3),
    pickup_datetime datetime not null,
    dropoff_datetime datetime,
    passenger_count int,
    trip_time_in_secs bigint,
    trip_distance float,
    pickup_longitude varchar(30),
    pickup_latitude varchar(30),
    dropoff_longitude varchar(30),
    dropoff_latitude varchar(30)
)
with (
    LOCATION = "/nyctaxitrip/",
    DATA_SOURCE= {nyctaxi_trip_storage},
    FILE_FORMAT = {csv_file_format},
    REJECT_TYPE= VALUE,
    REJECT_VALUE= 12
)

```

- o Load data from external tables in Azure blob storage to SQL Data Warehouse

```

CREATE TABLE {schemaname}.{nyctaxi_fare}
WITH
(
    CLUSTERED COLUMNSTORE INDEX,
    DISTRIBUTION = HASH(medallion)
)
AS
SELECT *
FROM  {external_nyctaxi_fare}
;

CREATE TABLE {schemaname}.{nyctaxi_trip}
WITH
(
    CLUSTERED COLUMNSTORE INDEX,
    DISTRIBUTION = HASH(medallion)
)
AS
SELECT *
FROM  {external_nyctaxi_trip}
;

```

- Create a sample data table (NYCTaxi_Sample) and insert data to it from selecting SQL queries on the trip and fare tables. (Some steps of this walkthrough needs to use this sample table.)

```

CREATE TABLE {schemaname}.{nyctaxi_sample}
WITH
(
    CLUSTERED COLUMNSTORE INDEX,
    DISTRIBUTION = HASH(medallion)
)
AS
(
    SELECT t.* , f.payment_type, f.fare_amount, f.surcharge, f.mta_tax, f.tolls_amount, f.total_amount, f.tip_amount,
    tipped = CASE WHEN (tip_amount > 0) THEN 1 ELSE 0 END,
    tip_class = CASE
        WHEN (tip_amount = 0) THEN 0
        WHEN (tip_amount > 0 AND tip_amount <= 5) THEN 1
        WHEN (tip_amount > 5 AND tip_amount <= 10) THEN 2
        WHEN (tip_amount > 10 AND tip_amount <= 20) THEN 3
        ELSE 4
    END
    FROM {schemaname}.{nyctaxi_trip} t, {schemaname}.{nyctaxi_fare} f
    WHERE datepart("mi",t.pickup_datetime) = 1
    AND t.medallion = f.medallion
    AND t.hack_license = f.hack_license
    AND t.pickup_datetime = f.pickup_datetime
    AND pickup_longitude <> "0"
    AND dropoff_longitude <> "0"
)
;
```

The geographic location of your storage accounts affects load times.

NOTE

Depending on the geographical location of your private blob storage account, the process of copying data from a public blob to your private storage account can take about 15 minutes, or even longer, and the process of loading data from your storage account to your Azure SQL DW could take 20 minutes or longer.

You will have to decide what do if you have duplicate source and destination files.

NOTE

If the .csv files to be copied from the public blob storage to your private blob storage account already exist in your private blob storage account, AzCopy will ask you whether you want to overwrite them. If you do not want to overwrite them, input **n** when prompted. If you want to overwrite **all** of them, input **a** when prompted. You can also input **y** to overwrite .csv files individually.

```
AzCopy is copying data from public blob to your storage account. It may take a while...
Overwrite http://weigamitess.blob.core.windows.net/dwhztest-248260597/nyctaxitrip/trip_data_7.csv with http://getgoing.b
Overwrite http://weigamitess.blob.core.windows.net/dwhztest-248260597/nyctaxitrip/trip_data_4.csv with http://getgoing.b
Overwrite http://weigamitess.blob.core.windows.net/dwhztest-248260597/nyctaxitrip/trip_data_5.csv with http://getgoing.b
Overwrite http://weigamitess.blob.core.windows.net/dwhztest-248260597/nyctaxitrip/trip_data_11.csv with http://getgoing.b
Overwrite http://weigamitess.blob.core.windows.net/dwhztest-248260597/nyctaxitrip/trip_data_12.csv with http://getgoing.b
Overwrite http://weigamitess.blob.core.windows.net/dwhztest-248260597/nyctaxitrip/trip_data_6.csv with http://getgoing.b
Overwrite http://weigamitess.blob.core.windows.net/dwhztest-248260597/nyctaxitrip/trip_data_9.csv with http://getgoing.b
Overwrite http://weigamitess.blob.core.windows.net/dwhztest-248260597/nyctaxitrip/trip_data_3.csv with http://getgoing.b
Overwrite http://weigamitess.blob.core.windows.net/dwhztest-248260597/nyctaxitrip/trip_data_10.csv with http://getgoing.b
Overwrite http://weigamitess.blob.core.windows.net/dwhztest-248260597/nyctaxitrip/trip_fare_1.csv with http://getgoing.b
Overwrite http://weigamitess.blob.core.windows.net/dwhztest-248260597/nyctaxitrip/trip_fare_12.csv with http://getgoing.b
blob.core.windows.net/public/nyctaxidataset/nyctaxifare/trip_fare_12.csv? (Yes/No/All) n
Overwrite http://weigamitess.blob.core.windows.net/dwhztest-248260597/nyctaxifare/trip_fare_2.csv with http://getgoing.b
Overwrite http://weigamitess.blob.core.windows.net/dwhztest-248260597/nyctaxifare/trip_fare_4.csv with http://getgoing.b
Overwrite http://weigamitess.blob.core.windows.net/dwhztest-248260597/nyctaxifare/trip_fare_11.csv with http://getgoing.b
Overwrite http://weigamitess.blob.core.windows.net/dwhztest-248260597/nyctaxifare/trip_fare_7.csv with http://getgoing.b
Overwrite http://weigamitess.blob.core.windows.net/dwhztest-248260597/nyctaxifare/trip_fare_3.csv with http://getgoing.b
Overwrite http://weigamitess.blob.core.windows.net/dwhztest-248260597/nyctaxifare/trip_fare_6.csv with http://getgoing.b
blob.core.windows.net/public/nyctaxidataset/nyctaxifare/trip_fare_6.csv? (Yes/No/All) n
Overwrite http://weigamitess.blob.core.windows.net/dwhztest-248260597/nyctaxifare/trip_fare_8.csv with http://getgoing.b
Overwrite http://weigamitess.blob.core.windows.net/dwhztest-248260597/nyctaxifare/trip_fare_10.csv with http://getgoing.b
Overwrite http://weigamitess.blob.core.windows.net/dwhztest-248260597/nyctaxitrip/trip_data_1.csv with http://getgoing.b
Overwrite http://weigamitess.blob.core.windows.net/dwhztest-248260597/nyctaxitrip/trip_data_2.csv with http://getgoing.b
Overwrite http://weigamitess.blob.core.windows.net/dwhztest-248260597/nyctaxitrip/trip_data_8.csv with http://getgoing.b
blob.core.windows.net/public/nyctaxidataset/nyctaxitrip/trip_data_8.csv? (Yes/No/All) n
Overwrite http://weigamitess.blob.core.windows.net/dwhztest-248260597/nyctaxifare/trip_fare_9.csv with http://getgoing.b
blob.core.windows.net/public/nyctaxidataset/nyctaxifare/trip_fare_9.csv? (Yes/No/All) n
FinishSee 0 of total 24 file(s).
[2015/12/30 20:00:40] Transfer summary:
-----
Total files transferred: 24
Transfer successfully: 0
Transfer skipped: 24
Transfer failed: 0
Elapsed time: 00:00:00:11
```

You can use your own data. If your data is in your on-premise machine in your real life application, you can still use AzCopy to upload on-premise data to your private Azure blob storage. You only need to change the **Source** location, `$Source = "http://getgoing.blob.core.windows.net/public/nyctaxidataset"`, in the AzCopy command of the PowerShell script file to the local directory that contains your data.

TIP

If your data is already in your private Azure blob storage in your real life application, you can skip the AzCopy step in the PowerShell script and directly upload the data to Azure SQL DW. This will require additional edits of the script to tailor it to the format of your data.

This Powershell script also plugs in the Azure SQL DW information into the data exploration example files `SQLDW_Explorations.sql`, `SQLDW_Explorations.ipynb`, and `SQLDW_Explorations_Scripts.py` so that these three files are ready to be tried out instantly after the PowerShell script completes.

After a successful execution, you will see screen like below:

```
AzCopy finished copying data. Please check your storage account to verify.
This step (copying data from public blob to your storage account) takes 8.11 seconds.
Executing SQL to load data into SQL DW: -----
Execution successful! -----
LoadDataToSQLDW.sql execution done
SQL script execution finished.
This step (loading data from your private blob to SQLDW) takes 1044.96 seconds.
The numbers of records from nyctaxitrip, nyctaxifare,nyctaxisample are 173179759, 173179759, and 2800275.
Please run the following command to complete the setup of the database.
Plug in the parameterized table names in SQL script file.
This step (plugging in database information) takes 0.67 seconds.
C:\temp\SQLDW\DeleteResourcesOnSQLDW.sql execution done
Deleting intermediate resources finished.
This step (deleting intermediate resources) takes 0.44 seconds.
```

Data exploration and feature engineering in Azure SQL Data Warehouse

In this section, we perform data exploration and feature generation by running SQL queries against Azure SQL DW directly using **Visual Studio Data Tools**. All SQL queries used in this section can be found in the sample script named `SQLDW_Explorations.sql`. This file has already been downloaded to your local directory by the PowerShell script. You can also retrieve it from [Github](#). But the file in Github does not have the Azure SQL DW information plugged in.

Connect to your Azure SQL DW using Visual Studio with the SQL DW login name and password and open up the

SQL Object Explorer to confirm the database and tables have been imported. Retrieve the *SQLDW_Explorations.sql* file.

NOTE

To open a Parallel Data Warehouse (PDW) query editor, use the **New Query** command while your PDW is selected in the **SQL Object Explorer**. The standard SQL query editor is not supported by PDW.

Here are the type of data exploration and feature generation tasks performed in this section:

- Explore data distributions of a few fields in varying time windows.
- Investigate data quality of the longitude and latitude fields.
- Generate binary and multiclass classification labels based on the **tip_amount**.
- Generate features and compute/compare trip distances.
- Join the two tables and extract a random sample that will be used to build models.

Data import verification

These queries provide a quick verification of the number of rows and columns in the tables populated earlier using Polybase's parallel bulk import,

```
-- Report number of rows in table <nyctaxi_trip> without table scan  
SELECT SUM(rows) FROM sys.partitions WHERE object_id = OBJECT_ID('<schemaname>.<nyctaxi_trip>')  
  
-- Report number of columns in table <nyctaxi_trip>  
SELECT COUNT(*) FROM information_schema.columns WHERE table_name = '<nyctaxi_trip>' AND table_schema = '<schemaname>'
```

Output: You should get 173,179,759 rows and 14 columns.

Exploration: Trip distribution by medallion

This example query identifies the medallions (taxi numbers) that completed more than 100 trips within a specified time period. The query would benefit from the partitioned table access since it is conditioned by the partition scheme of **pickup_datetime**. Querying the full dataset will also make use of the partitioned table and/or index scan.

```
SELECT medallion, COUNT(*)  
FROM <schemaname>.<nyctaxi_fare>  
WHERE pickup_datetime BETWEEN '20130101' AND '20130331'  
GROUP BY medallion  
HAVING COUNT(*) > 100
```

Output: The query should return a table with rows specifying the 13,369 medallions (taxis) and the number of trip completed by them in 2013. The last column contains the count of the number of trips completed.

Exploration: Trip distribution by medallion and hack_license

This example identifies the medallions (taxi numbers) and hack_license numbers (drivers) that completed more than 100 trips within a specified time period.

```
SELECT medallion, hack_license, COUNT(*)  
FROM <schemaname>.<nyctaxi_fare>  
WHERE pickup_datetime BETWEEN '20130101' AND '20130331'  
GROUP BY medallion, hack_license  
HAVING COUNT(*) > 100
```

Output: The query should return a table with 13,369 rows specifying the 13,369 car/driver IDs that have completed more than 100 trips in 2013. The last column contains the count of the number of trips completed.

Data quality assessment: Verify records with incorrect longitude and/or latitude

This example investigates if any of the longitude and/or latitude fields either contain an invalid value (radian degrees should be between -90 and 90), or have (0, 0) coordinates.

```
SELECT COUNT(*) FROM < schemaname >. < nyctaxi_trip >
WHERE pickup_datetime BETWEEN '20130101' AND '20130331'
AND (CAST(pickup_longitude AS float) NOT BETWEEN -90 AND 90
OR CAST(pickup_latitude AS float) NOT BETWEEN -90 AND 90
OR CAST(dropoff_longitude AS float) NOT BETWEEN -90 AND 90
OR CAST(dropoff_latitude AS float) NOT BETWEEN -90 AND 90
OR (pickup_longitude = '0' AND pickup_latitude = '0')
OR (dropoff_longitude = '0' AND dropoff_latitude = '0'))
```

Output: The query returns 837,467 trips that have invalid longitude and/or latitude fields.

Exploration: Tipped vs. not tipped trips distribution

This example finds the number of trips that were tipped vs. the number that were not tipped in a specified time period (or in the full dataset if covering the full year as it is set up here). This distribution reflects the binary label distribution to be later used for binary classification modeling.

```
SELECT tipped, COUNT(*) AS tip_freq FROM (
  SELECT CASE WHEN (tip_amount > 0) THEN 1 ELSE 0 END AS tipped, tip_amount
  FROM < schemaname >. < nyctaxi_fare >
  WHERE pickup_datetime BETWEEN '20130101' AND '20131231') tc
GROUP BY tipped
```

Output: The query should return the following tip frequencies for the year 2013: 90,447,622 tipped and 82,264,709 not-tipped.

Exploration: Tip class/range distribution

This example computes the distribution of tip ranges in a given time period (or in the full dataset if covering the full year). This is the distribution of the label classes that will be used later for multiclass classification modeling.

```
SELECT tip_class, COUNT(*) AS tip_freq FROM (
  SELECT CASE
    WHEN (tip_amount = 0) THEN 0
    WHEN (tip_amount > 0 AND tip_amount <= 5) THEN 1
    WHEN (tip_amount > 5 AND tip_amount <= 10) THEN 2
    WHEN (tip_amount > 10 AND tip_amount <= 20) THEN 3
    ELSE 4
  END AS tip_class
  FROM < schemaname >. < nyctaxi_fare >
  WHERE pickup_datetime BETWEEN '20130101' AND '20131231') tc
GROUP BY tip_class
```

Output:

TIP_CLASS	TIP_FREQ
1	82230915
2	6198803
3	1932223
0	82264625

TIP_CLASS	TIP_FREQ
4	85765

Exploration: Compute and compare trip distance

This example converts the pickup and drop-off longitude and latitude to SQL geography points, computes the trip distance using SQL geography points difference, and returns a random sample of the results for comparison. The example limits the results to valid coordinates only using the data quality assessment query covered earlier.

```
***** Object: UserDefinedFunction [dbo].[fnCalculateDistance] *****/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

IF EXISTS (SELECT * FROM sys.objects WHERE type IN ('FN', 'IF') AND name = 'fnCalculateDistance')
    DROP FUNCTION fnCalculateDistance
GO

-- User-defined function to calculate the direct distance in mile between two geographical coordinates.
CREATE FUNCTION [dbo].[fnCalculateDistance] (@Lat1 float, @Long1 float, @Lat2 float, @Long2 float)
RETURNS float
AS
BEGIN
    DECLARE @distance decimal(28, 10)
    -- Convert to radians
    SET @Lat1 = @Lat1 / 57.2958
    SET @Long1 = @Long1 / 57.2958
    SET @Lat2 = @Lat2 / 57.2958
    SET @Long2 = @Long2 / 57.2958
    -- Calculate distance
    SET @distance = (SIN(@Lat1) * SIN(@Lat2)) + (COS(@Lat1) * COS(@Lat2) * COS(@Long2 - @Long1))
    --Convert to miles
    IF @distance <> 0
        BEGIN
            SET @distance = 3958.75 * ATAN(SQRT(1 - POWER(@distance, 2)) / @distance);
        END
    RETURN @distance
END
GO

SELECT pickup_latitude, pickup_longitude, dropoff_latitude, dropoff_longitude,
dbo.fnCalculateDistance(pickup_latitude, pickup_longitude, dropoff_latitude, dropoff_longitude) AS DirectDistance
FROM <schemaname>.nytaxi_trip
WHERE datepart("mi",pickup_datetime)=1
AND CAST(pickup_latitude AS float) BETWEEN -90 AND 90
AND CAST(dropoff_latitude AS float) BETWEEN -90 AND 90
AND pickup_longitude != '0' AND dropoff_longitude != '0'
```

Feature engineering using SQL functions

Sometimes SQL functions can be an efficient option for feature engineering. In this walkthrough, we defined a SQL function to calculate the direct distance between the pickup and dropoff locations. You can run the following SQL scripts in **Visual Studio Data Tools**.

Here is the SQL script that defines the distance function.

```

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

IF EXISTS (SELECT * FROM sys.objects WHERE type IN ('FN', 'IF') AND name = 'fnCalculateDistance')
    DROP FUNCTION fnCalculateDistance
GO

-- User-defined function calculate the direct distance between two geographical coordinates.
CREATE FUNCTION [dbo].[fnCalculateDistance] (@Lat1 float, @Long1 float, @Lat2 float, @Long2 float)
RETURNS float
AS
BEGIN
    DECLARE @distance decimal(28, 10)
    -- Convert to radians
    SET @Lat1 = @Lat1 / 57.2958
    SET @Long1 = @Long1 / 57.2958
    SET @Lat2 = @Lat2 / 57.2958
    SET @Long2 = @Long2 / 57.2958
    -- Calculate distance
    SET @distance = (SIN(@Lat1) * SIN(@Lat2)) + (COS(@Lat1) * COS(@Lat2) * COS(@Long2 - @Long1))
    --Convert to miles
    IF @distance <> 0
        BEGIN
            SET @distance = 3958.75 * ATAN(SQRT(1 - POWER(@distance, 2)) / @distance);
        END
    RETURN @distance
END
GO

```

Here is an example to call this function to generate features in your SQL query:

```

-- Sample query to call the function to create features
SELECT pickup_latitude, pickup_longitude, dropoff_latitude, dropoff_longitude,
dbo.fnCalculateDistance(pickup_latitude, pickup_longitude, dropoff_latitude, dropoff_longitude) AS DirectDistance
FROM <schemaname>.nyctaxi_trip
WHERE datepart("mi",pickup_datetime)=1
AND CAST(pickup_latitude AS float) BETWEEN -90 AND 90
AND CAST(dropoff_latitude AS float) BETWEEN -90 AND 90
AND pickup_longitude != '0' AND dropoff_longitude != '0'

```

Output: This query generates a table (with 2,803,538 rows) with pickup and dropoff latitudes and longitudes and the corresponding direct distances in miles. Here are the results for first 3 rows:

	PICKUP_LATITUDE	PICKUP_LONGITUDE	DROPOFF_LATITUDE	DROPOFF_LONGITUDE	DIRECTDISTANCE
1	40.731804	-74.001083	40.736622	-73.988953	.7169601222
2	40.715794	-74.010635	40.725338	-74.00399	.7448343721
3	40.761456	-73.999886	40.766544	-73.988228	0.7037227967

Prepare data for model building

The following query joins the **nyctaxi_trip** and **nyctaxi_fare** tables, generates a binary classification label **tipped**, a multi-class classification label **tip_class**, and extracts a sample from the full joined dataset. The sampling is done by retrieving a subset of the trips based on pickup time. This query can be copied then pasted directly in the [Azure Machine Learning Studio Import Data](#) module for direct data ingestion from the SQL database instance in Azure.

The query excludes records with incorrect (0, 0) coordinates.

```
SELECT t.*, f.payment_type, f.fare_amount, f.surcharge, f.mta_tax, f.tolls_amount, f.total_amount, f.tip_amount,
CASE WHEN (tip_amount > 0) THEN 1 ELSE 0 END AS tipped,
CASE WHEN (tip_amount = 0) THEN
    WHEN (tip_amount > 0 AND tip_amount <= 5) THEN 1
    WHEN (tip_amount > 5 AND tip_amount <= 10) THEN 2
    WHEN (tip_amount > 10 AND tip_amount <= 20) THEN 3
    ELSE 4
END AS tip_class
FROM <schemaname>.nyctaxi_trip t, <schemaname>.nyctaxi_fare f
WHERE datepart("mi",t.pickup_datetime)=1
AND t.medallion = f.medallion
AND t.hack_license = f.hack_license
AND t.pickup_datetime = f.pickup_datetime
AND pickup_longitude != '0' AND dropoff_longitude != '0'
```

When you are ready to proceed to Azure Machine Learning, you may either:

1. Save the final SQL query to extract and sample the data and copy-paste the query directly into a [Import Data](#) module in Azure Machine Learning, or
2. Persist the sampled and engineered data you plan to use for model building in a new SQL DW table and use the new table in the [Import Data](#) module in Azure Machine Learning. The PowerShell script in earlier step has done this for you. You can read directly from this table in the Import Data module.

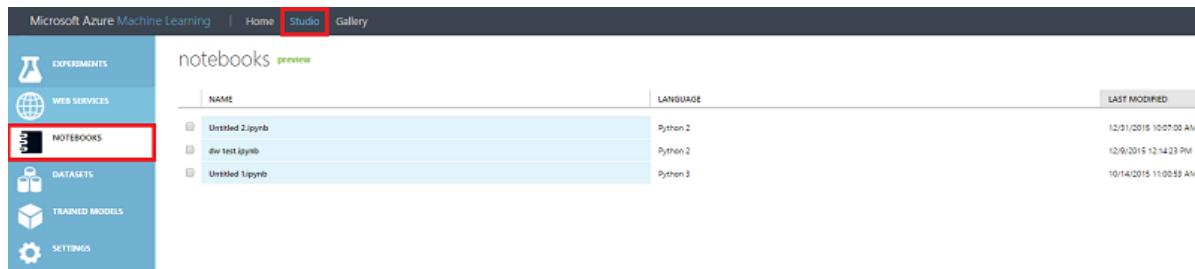
Data exploration and feature engineering in IPython notebook

In this section, we will perform data exploration and feature generation using both Python and SQL queries against the SQL DW created earlier. A sample IPython notebook named **SQLDW_Explorations.ipynb** and a Python script file **SQLDW_Explorations_Scripts.py** have been downloaded to your local directory. They are also available on [GitHub](#). These two files are identical in Python scripts. The Python script file is provided to you in case you do not have an IPython Notebook server. These two sample Python files are designed under **Python 2.7**.

The needed Azure SQL DW information in the sample IPython Notebook and the Python script file downloaded to your local machine has been plugged in by the PowerShell script previously. They are executable without any modification.

If you have already set up an AzureML workspace, you can directly upload the sample IPython Notebook to the AzureML IPython Notebook service and start running it. Here are the steps to upload to AzureML IPython Notebook service:

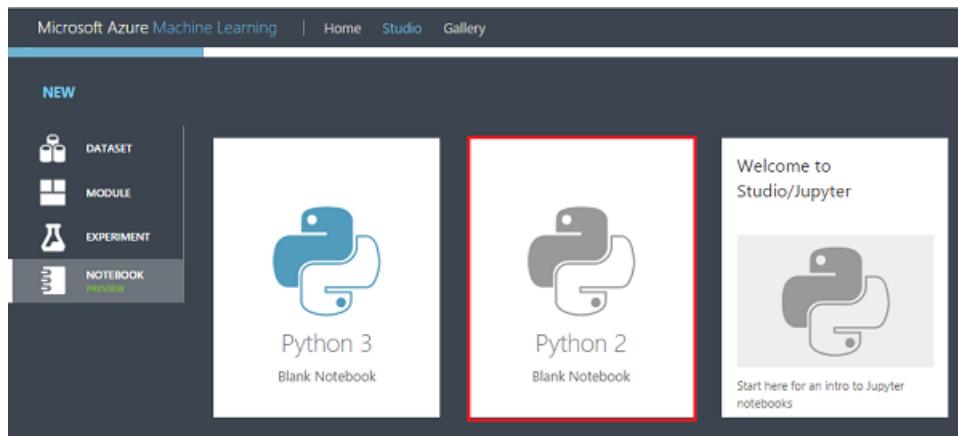
1. Log in to your AzureML workspace, click "Studio" at the top, and click "NOTEBOOKS" on the left side of the web page.



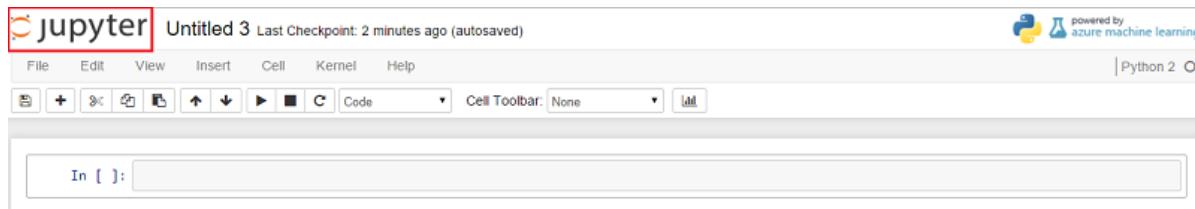
The screenshot shows the Microsoft Azure Machine Learning Studio interface. At the top, there is a navigation bar with 'Home', 'Studio' (which is highlighted with a red box), and 'Gallery'. On the left, there is a sidebar with icons for 'EXPERIMENTS', 'WEB SERVICES', 'NOTEBOOKS' (which is highlighted with a red box), 'DATASETS', 'TRAINED MODELS', and 'SETTINGS'. The main area is titled 'notebooks preview'. It contains a table with three rows of data:

NAME	LANGUAGE	LAST MODIFIED
Untitled 2.ipynb	Python 2	12/31/2015 10:07:00 AM
dw test.ipynb	Python 2	10/9/2015 12:14:23 PM
Untitled 1.ipynb	Python 3	10/14/2015 11:00:55 AM

2. Click "NEW" on the left bottom corner of the web page, and select "Python 2". Then, provide a name to the notebook and click the check mark to create the new blank IPython Notebook.



3. Click the "Jupyter" symbol on the left top corner of the new IPython Notebook.



4. Drag and drop the sample IPython Notebook to the **tree** page of your AzureML IPython Notebook service, and click **Upload**. Then, the sample IPython Notebook will be uploaded to the AzureML IPython Notebook service.



In order to run the sample IPython Notebook or the Python script file, the following Python packages are needed. If you are using the AzureML IPython Notebook service, these packages have been pre-installed.

```
- pandas
- numpy
- matplotlib
- pyodbc
- PyTables
```

The recommended sequence when building advanced analytical solutions on AzureML with large data is the following:

- Read in a small sample of the data into an in-memory data frame.
- Perform some visualizations and explorations using the sampled data.
- Experiment with feature engineering using the sampled data.
- For larger data exploration, data manipulation and feature engineering, use Python to issue SQL Queries directly against the SQL DW.
- Decide the sample size to be suitable for Azure Machine Learning model building.

The followings are a few data exploration, data visualization, and feature engineering examples. More data explorations can be found in the sample IPython Notebook and the sample Python script file.

Initialize database credentials

Initialize your database connection settings in the following variables:

```
SERVER_NAME=<server name>
DATABASE_NAME=<database name>
USERID=<user name>
PASSWORD=<password>
DB_DRIVER = <database driver>
```

Create database connection

Here is the connection string that creates the connection to the database.

```
CONNECTION_STRING='DRIVER=
{+DRIVER+};SERVER='+SERVER_NAME+';DATABASE='+DATABASE_NAME+';UID='+USERID+';PWD='+PASSWORD
conn = pyodbc.connect(CONNECTION_STRING)
```

Report number of rows and columns in table

```
nrows = pd.read_sql("""
    SELECT SUM(rows) FROM sys.partitions
    WHERE object_id = OBJECT_ID('<schemaname>.nyctaxi_trip')
    """, conn)

print 'Total number of rows = %d' % nrows.iloc[0,0]

ncols = pd.read_sql("""
    SELECT COUNT(*) FROM information_schema.columns
    WHERE table_name = ('nyctaxi_trip') AND table_schema = ('<schemaname>')
    """, conn)

print 'Total number of columns = %d' % ncols.iloc[0,0]
```

- Total number of rows = 173179759
- Total number of columns = 14

Report number of rows and columns in table

```
nrows = pd.read_sql("""
    SELECT SUM(rows) FROM sys.partitions
    WHERE object_id = OBJECT_ID('<schemaname>.nyctaxi_fare')
    """, conn)

print 'Total number of rows = %d' % nrows.iloc[0,0]

ncols = pd.read_sql("""
    SELECT COUNT(*) FROM information_schema.columns
    WHERE table_name = ('nyctaxi_fare') AND table_schema = ('<schemaname>')
    """, conn)

print 'Total number of columns = %d' % ncols.iloc[0,0]
```

- Total number of rows = 173179759
- Total number of columns = 11

Read-in a small data sample from the SQL Data Warehouse Database

```

t0 = time.time()

query = """
SELECT TOP 10000 t.* , f.payment_type, f.fare_amount, f.surcharge, f.mta_tax,
       f.tolls_amount, f.total_amount, f.tip_amount
  FROM <schemaName>.nyctaxi_trip t, <schemaName>.nyctaxi_fare f
 WHERE datepart("mi",t.pickup_datetime) = 1
   AND t.medallion = f.medallion
   AND t.hack_license = f.hack_license
   AND t.pickup_datetime = f.pickup_datetime
"""

df1 = pd.read_sql(query, conn)

t1 = time.time()
print 'Time to read the sample table is %f seconds' % (t1-t0)

print 'Number of rows and columns retrieved = (%d, %d)' % (df1.shape[0], df1.shape[1])

```

Time to read the sample table is 14.096495 seconds.

Number of rows and columns retrieved = (1000, 21).

Descriptive statistics

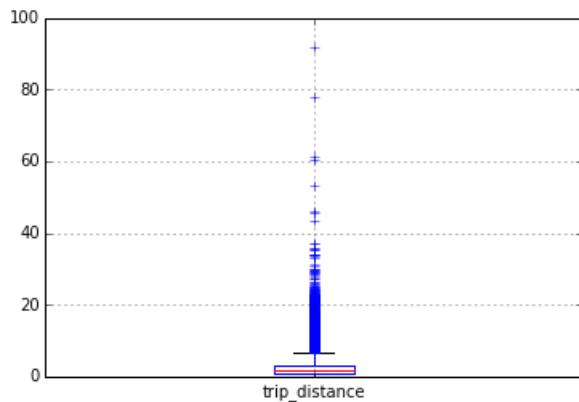
Now you are ready to explore the sampled data. We start with looking at some descriptive statistics for the **trip_distance** (or any other fields you choose to specify).

```
df1['trip_distance'].describe()
```

Visualization: Box plot example

Next we look at the box plot for the trip distance to visualize the quantiles.

```
df1.boxplot(column='trip_distance',return_type='dict')
```



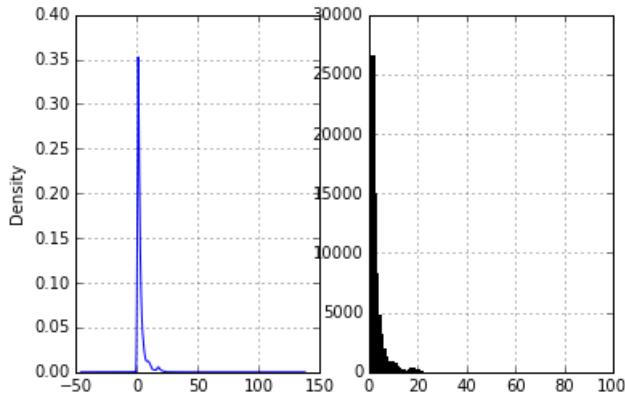
Visualization: Distribution plot example

Plots that visualize the distribution and a histogram for the sampled trip distances.

```

fig = plt.figure()
ax1 = fig.add_subplot(1,2,1)
ax2 = fig.add_subplot(1,2,2)
df1['trip_distance'].plot(ax=ax1,kind='kde', style='b-')
df1['trip_distance'].hist(ax=ax2, bins=100, color='k')

```



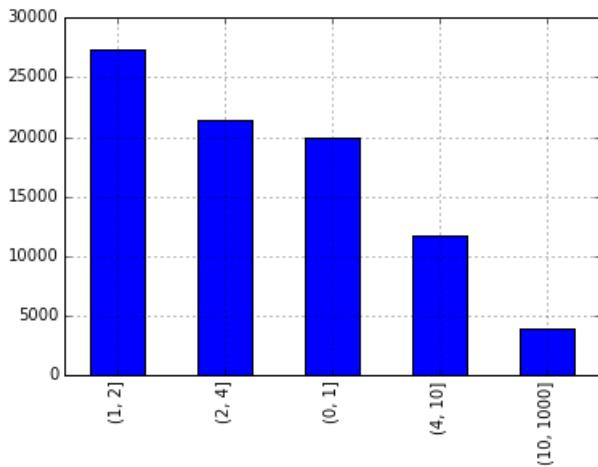
Visualization: Bar and line plots

In this example, we bin the trip distance into five bins and visualize the binning results.

```
trip_dist_bins = [0, 1, 2, 4, 10, 1000]
df1['trip_distance']
trip_dist_bin_id = pd.cut(df1['trip_distance'], trip_dist_bins)
trip_dist_bin_id
```

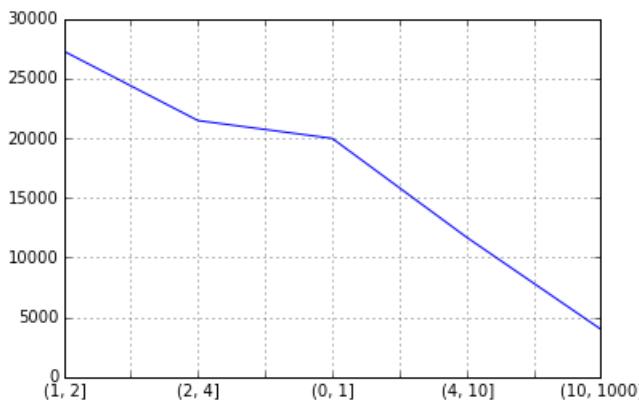
We can plot the above bin distribution in a bar or line plot with:

```
pd.Series(trip_dist_bin_id).value_counts().plot(kind='bar')
```



and

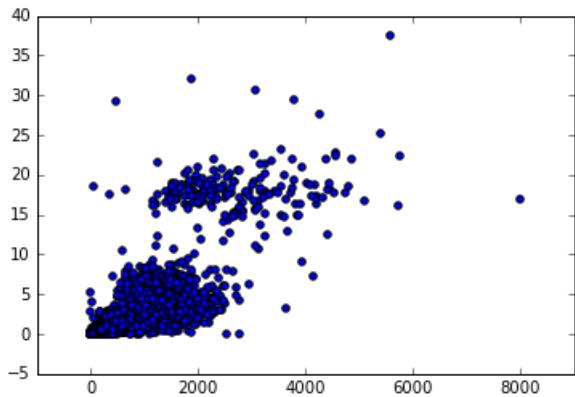
```
pd.Series(trip_dist_bin_id).value_counts().plot(kind='line')
```



Visualization: Scatterplot examples

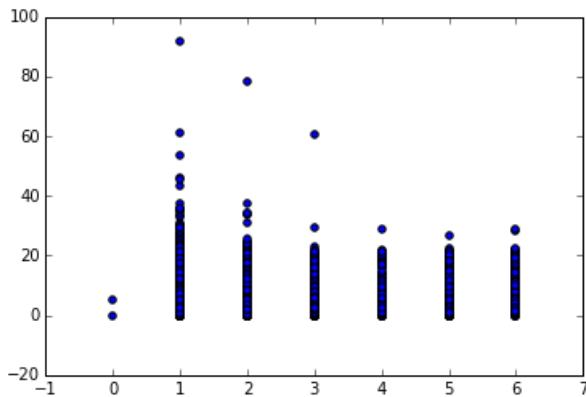
We show scatter plot between **trip_time_in_secs** and **trip_distance** to see if there is any correlation

```
plt.scatter(dfl['trip_time_in_secs'], dfl['trip_distance'])
```



Similarly we can check the relationship between **rate_code** and **trip_distance**.

```
plt.scatter(dfl['passenger_count'], dfl['trip_distance'])
```



Data exploration on sampled data using SQL queries in IPython notebook

In this section, we explore data distributions using the sampled data which is persisted in the new table we created above. Note that similar explorations can be performed using the original tables.

Exploration: Report number of rows and columns in the sampled table

```
nrows = pd.read_sql("SELECT SUM(rows) FROM sys.partitions WHERE object_id = OBJECT_ID('<schema_name>.nyctaxi_sample')", conn)
print 'Number of rows in sample = %d' % nrows.iloc[0,0]

ncols = pd.read_sql("SELECT count(*) FROM information_schema.columns WHERE table_name = ('nyctaxi_sample') AND table_schema = '<schema_name>'", conn)
print 'Number of columns in sample = %d' % ncols.iloc[0,0]
```

Exploration: Tipped/not tipped Distribution

```
query = "
SELECT tipped, count(*) AS tip_freq
FROM <schema_name>.nyctaxi_sample
GROUP BY tipped
"

pd.read_sql(query, conn)
```

Exploration: Tip class distribution

```

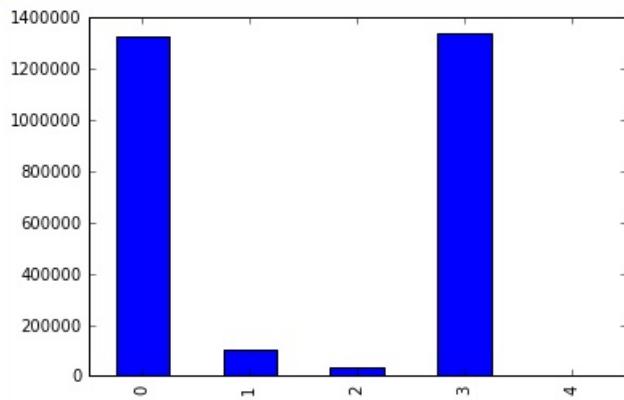
query = ""
SELECT tip_class, count(*) AS tip_freq
FROM <schemaname>.<nyctaxi_sample>
GROUP BY tip_class
"""

tip_class_dist = pd.read_sql(query, conn)

```

Exploration: Plot the tip distribution by class

```
tip_class_dist['tip_freq'].plot(kind='bar')
```



Exploration: Daily distribution of trips

```

query = ""
SELECT CONVERT(date, dropoff_datetime) AS date, COUNT(*) AS c
FROM <schemaname>.<nyctaxi_sample>
GROUP BY CONVERT(date, dropoff_datetime)
"""

pd.read_sql(query, conn)

```

Exploration: Trip distribution per medallion

```

query = ""
SELECT medallion, count(*) AS c
FROM <schemaname>.<nyctaxi_sample>
GROUP BY medallion
"""

pd.read_sql(query, conn)

```

Exploration: Trip distribution by medallion and hack license

```

query = "select medallion, hack_license, count(*) from <schemaname>.<nyctaxi_sample> group by medallion, hack_license"
pd.read_sql(query, conn)

```

Exploration: Trip time distribution

```

query = "select trip_time_in_secs, count(*) from <schemaname>.<nyctaxi_sample> group by trip_time_in_secs order by count(*) desc"
pd.read_sql(query, conn)

```

Exploration: Trip distance distribution

```

query = "select floor(trip_distance/5)*5 as tripbin, count(*) from <schemaname>.<nyctaxi_sample> group by floor(trip_distance/5)*5 order by
count(*) desc"
pd.read_sql(query, conn)

```

Exploration: Payment type distribution

```
query = "select payment_type,count(*) from <schemaname>.<nytaxi_sample> group by payment_type"
pd.read_sql(query,conn)
```

Verify the final form of the featurized table

```
query = "SELECT TOP 100 * FROM <schemaname>.<nytaxi_sample>"
pd.read_sql(query,conn)
```

Build models in Azure Machine Learning

We are now ready to proceed to model building and model deployment in [Azure Machine Learning](#). The data is ready to be used in any of the prediction problems identified earlier, namely:

1. **Binary classification:** To predict whether or not a tip was paid for a trip.
2. **Multiclass classification:** To predict the range of tip paid, according to the previously defined classes.
3. **Regression task:** To predict the amount of tip paid for a trip.

To begin the modeling exercise, log in to your **Azure Machine Learning** workspace. If you have not yet created a machine learning workspace, see [Create an Azure ML workspace](#).

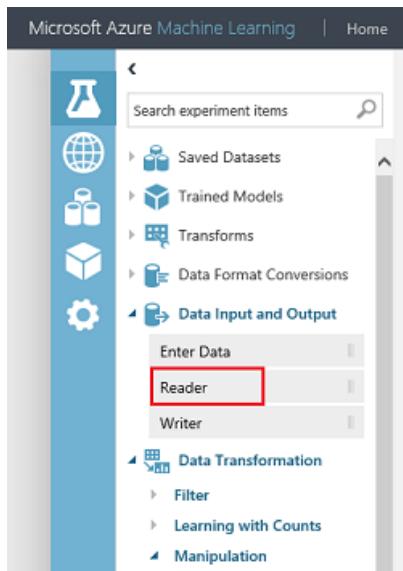
1. To get started with Azure Machine Learning, see [What is Azure Machine Learning Studio?](#)
2. Log in to [Azure Machine Learning Studio](#).
3. The Studio Home page provides a wealth of information, videos, tutorials, links to the Modules Reference, and other resources. For more information about Azure Machine Learning, consult the [Azure Machine Learning Documentation Center](#).

A typical training experiment consists of the following steps:

1. Create a **+NEW** experiment.
2. Get the data into Azure ML.
3. Pre-process, transform and manipulate the data as needed.
4. Generate features as needed.
5. Split the data into training/validation/testing datasets(or have separate datasets for each).
6. Select one or more machine learning algorithms depending on the learning problem to solve. E.g., binary classification, multiclass classification, regression.
7. Train one or more models using the training dataset.
8. Score the validation dataset using the trained model(s).
9. Evaluate the model(s) to compute the relevant metrics for the learning problem.
10. Fine tune the model(s) and select the best model to deploy.

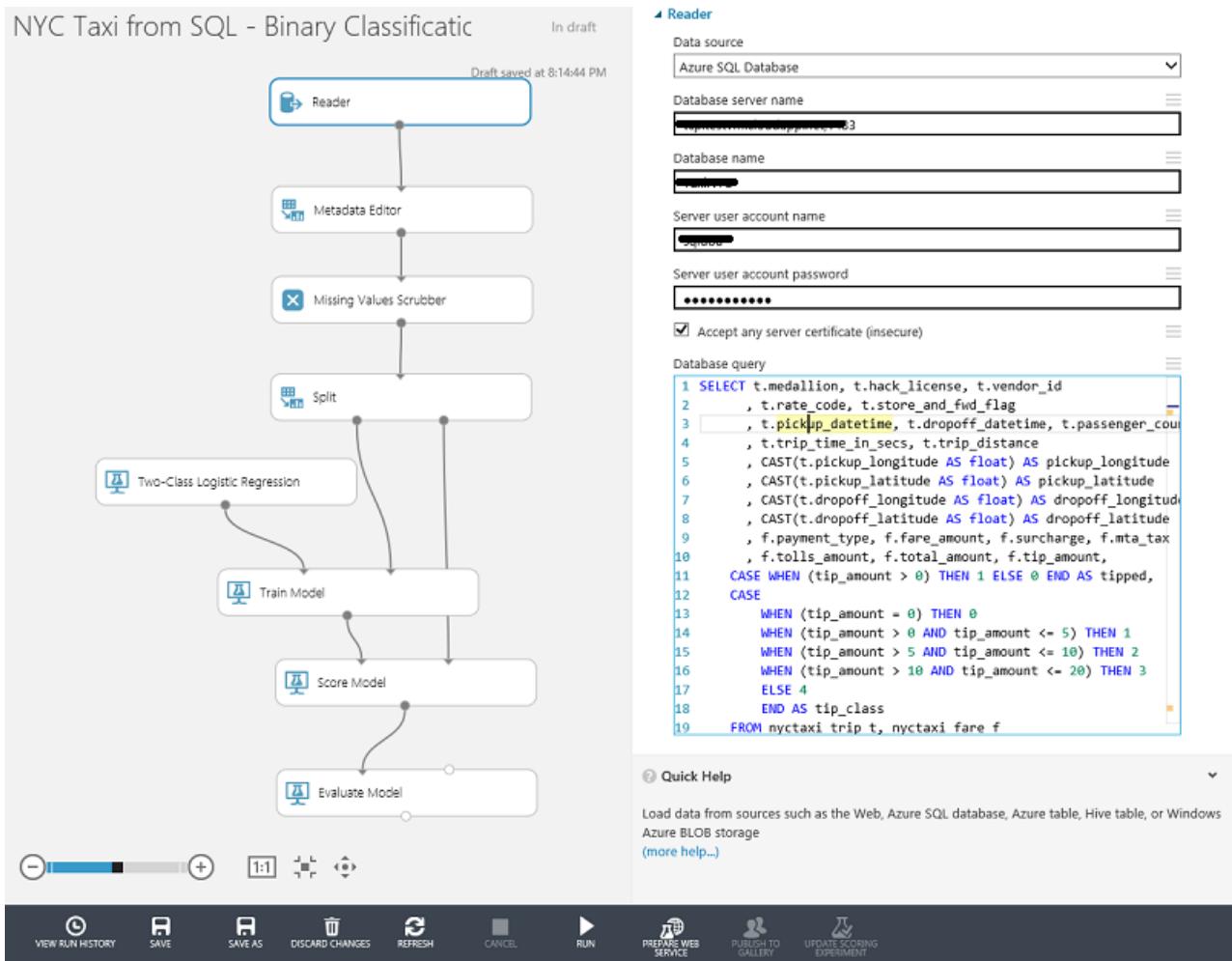
In this exercise, we have already explored and engineered the data in SQL Data Warehouse, and decided on the sample size to ingest in Azure ML. Here is the procedure to build one or more of the prediction models:

1. Get the data into Azure ML using the [Import Data](#) module, available in the **Data Input and Output** section.
For more information, see the [Import Data](#) module reference page.



2. Select **Azure SQL Database** as the **Data source** in the **Properties** panel.
3. Enter the database DNS name in the **Database server name** field. Format:
`tcp:<your virtual machine DNS name>.1433`
4. Enter the **Database name** in the corresponding field.
5. Enter the *SQL user name* in the **Server user account name**, and the *password* in the **Server user account password**.
6. Check the **Accept any server certificate** option.
7. In the **Database query** edit text area, paste the query which extracts the necessary database fields (including any computed fields such as the labels) and down samples the data to the desired sample size.

An example of a binary classification experiment reading data directly from the SQL Data Warehouse database is in the figure below (remember to replace the table names nyctaxi_trip and nyctaxi_fare by the schema name and the table names you used in your walkthrough). Similar experiments can be constructed for multiclass classification and regression problems.



IMPORTANT

In the modeling data extraction and sampling query examples provided in previous sections, **all labels for the three modeling exercises are included in the query**. An important (required) step in each of the modeling exercises is to **exclude** the unnecessary labels for the other two problems, and any other **target leaks**. For example, when using binary classification, use the label **tipped** and exclude the fields **tip_class**, **tip_amount**, and **total_amount**. The latter are target leaks since they imply the tip paid.

To exclude any unnecessary columns or target leaks, you may use the [Select Columns in Dataset](#) module or the [Edit Metadata](#). For more information, see [Select Columns in Dataset](#) and [Edit Metadata](#) reference pages.

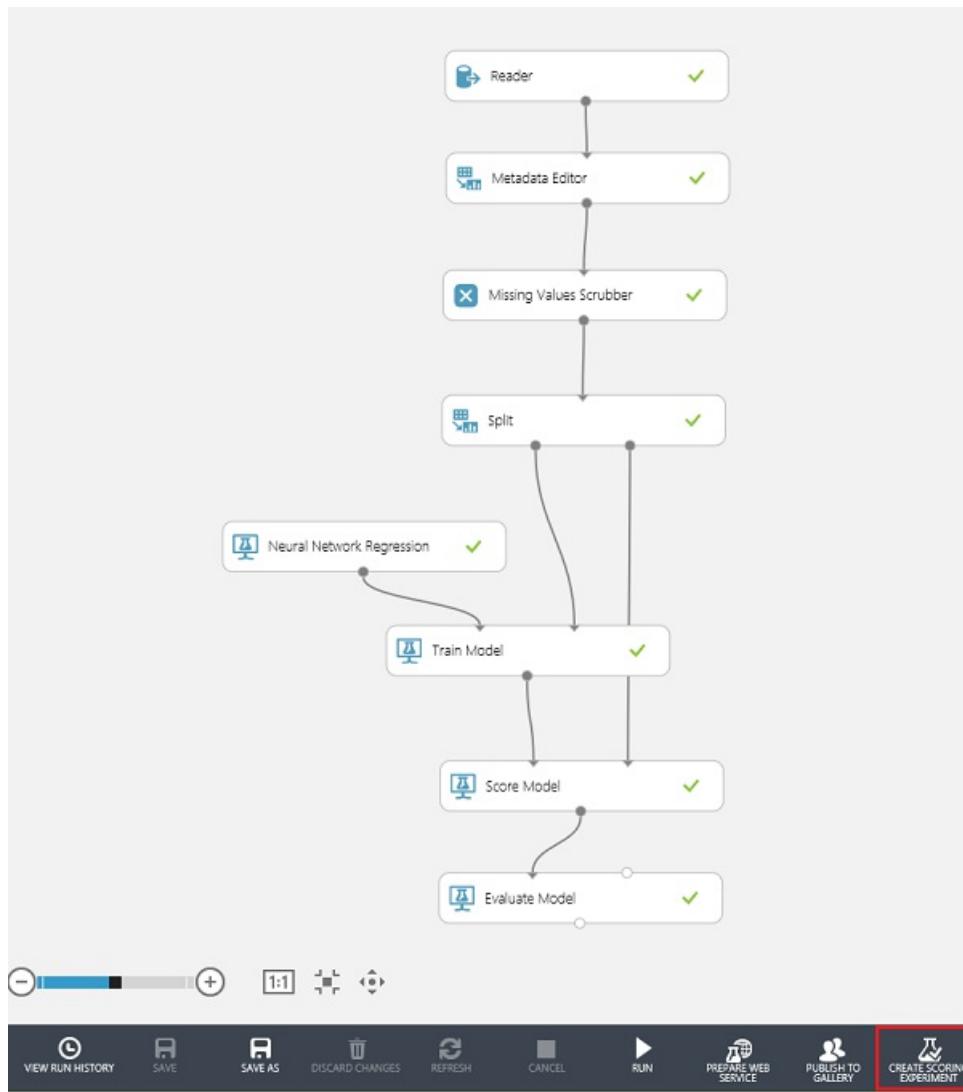
Deploy models in Azure Machine Learning

When your model is ready, you can easily deploy it as a web service directly from the experiment. For more information about deploying Azure ML web services, see [Deploy an Azure Machine Learning web service](#).

To deploy a new web service, you need to:

1. Create a scoring experiment.
2. Deploy the web service.

To create a scoring experiment from a **Finished** training experiment, click **CREATE SCORING EXPERIMENT** in the lower action bar.

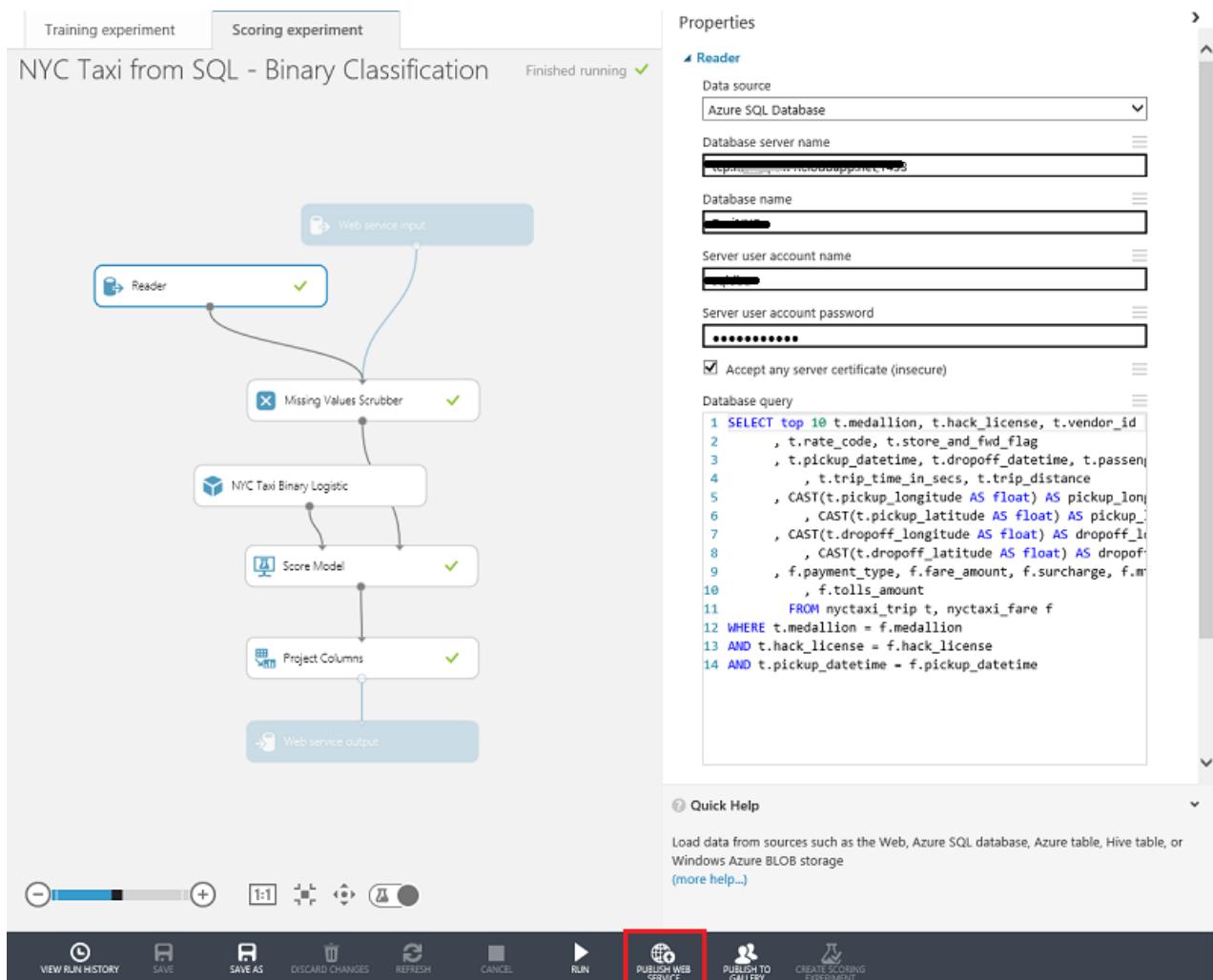


Azure Machine Learning will attempt to create a scoring experiment based on the components of the training experiment. In particular, it will:

1. Save the trained model and remove the model training modules.
2. Identify a logical **input port** to represent the expected input data schema.
3. Identify a logical **output port** to represent the expected web service output schema.

When the scoring experiment is created, review it and make adjust as needed. A typical adjustment is to replace the input dataset and/or query with one which excludes label fields, as these will not be available when the service is called. It is also a good practice to reduce the size of the input dataset and/or query to a few records, just enough to indicate the input schema. For the output port, it is common to exclude all input fields and only include the **Scored Labels** and **Scored Probabilities** in the output using the [Select Columns in Dataset](#) module.

A sample scoring experiment is provided in the figure below. When ready to deploy, click the **PUBLISH WEB SERVICE** button in the lower action bar.



Summary

To recap what we have done in this walkthrough tutorial, you have created an Azure data science environment, worked with a large public dataset, taking it through the Team Data Science Process, all the way from data acquisition to model training, and then to the deployment of an Azure Machine Learning web service.

License information

This sample walkthrough and its accompanying scripts and IPython notebook(s) are shared by Microsoft under the MIT license. Please check the LICENSE.txt file in the directory of the sample code on GitHub for more details.

References

- [Andrés Monroy NYC Taxi Trips Download Page](#)
- [FOILing NYC's Taxi Trip Data by Chris Whong](#)
- [NYC Taxi and Limousine Commission Research and Statistics](#)

(deprecated) Web services examples using R code on Azure Machine Learning and published to Microsoft Azure Marketplace

1/17/2017 • 2 min to read • [Edit on GitHub](#)

NOTE

The Microsoft DataMarket is being retired and these APIs have been deprecated.

You can find many useful example experiments and APIs in the [Cortana Intelligence Gallery](#). For more information about the Gallery, see [Share and discover resources in the Cortana Intelligence Gallery](#).

In this article are example web services were created using Azure Machine Learning and published to the Azure Marketplace. Each web service example has a comprehensive document attached, embedding sample data sets for testing the services and explaining how the user can create a similar service themselves.

In Azure Machine Learning Studio, users can write R code and with a few clicks, publish it as a web service for applications and devices to consume around the world.

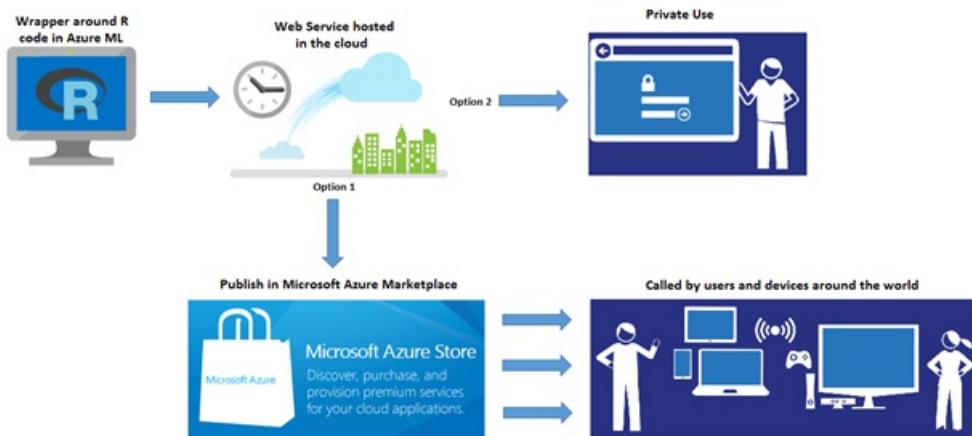
NOTE

[Try Azure Machine Learning for free](#)

No credit card or Azure subscription needed. [Get started now >](#)

From producing simple calculators that provide statistical functionality to creating a custom text-mining sentiment analysis predictor, both new and experienced R users can benefit from the ease with which users of Azure Machine Learning can operationalize R code. While these web services could be consumed by users, potentially through a mobile app or a website, the purpose of these web services examples is to show how Machine Learning can operationalize R scripts for analytical purposes and be used to create web services on top of R code.

Each example includes a C# example for web service consumption.



Consider the following scenarios.

Scenario 1: Generic model

A user works with a generic model that can be applied to a new user's data, such as a basic forecasting of time series data or a custom-built R method with advanced analytics. This user publishes the model as a web service for others to consume with their data.

- [Binary Classifier](#)
- [Cluster Model](#)
- [Multivariate Linear Regression](#)
- [Forecasting - Exponential Smoothing](#)
- [ETS + STL Forecasting](#)
- [Forecasting - Autoregressive Integrated Moving Average \(ARIMA\)](#)
- [Survival Analysis](#)

Scenario 2: Trained model – specific data

A user has data that provides useful predictions through R code, such as a large sample of personality questionnaires clustered through a k-means algorithm to predict the user's personality type, or health survey data that can be used to predict an individual's risk for lung cancer with a survival analysis R package. The user publishes the data through a web service that predicts a new user's outcome.

Scenario 3: Trained model – generic data

A user has generic data (such as a text corpus) that allows a web service to be built and applied generically across different types of use cases and scenarios.

- [Lexicon Based Sentiment Analysis](#)

Scenario 4: Advanced calculator

A user provides advanced calculations or simulations that don't require any trained model or fitting of a model to the user's data.

- [Difference in Proportions Test](#)
- [Normal Distribution Suite](#)
- [Binomial Distribution Suite](#)

FAQ

For frequently asked questions on consumption of the web service or publishing to the Marketplace, see [here](#).

(deprecated) Binary Classifier

1/17/2017 • 4 min to read • [Edit on GitHub](#)

NOTE

The Microsoft DataMarket is being retired and this API has been deprecated.

You can find many useful example experiments and APIs in the [Cortana Intelligence Gallery](#). For more information about the Gallery, see [Share and discover resources in the Cortana Intelligence Gallery](#).

Suppose you have a dataset and would like to predict a binary dependent variable based on the independent variables. ‘Logistic Regression’ is a popular statistical technique used for such predictions. Here the dependent variable is binary or dichotomous, and p is the probability of presence of the characteristic of interest.

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

A simple scenario could be where a researcher is trying to predict whether a prospective student is likely to accept an admission offer to a university based on information (GPA in high school, family income, resident state, gender). The predicted outcome is the probability of the prospective student accepting the admission offer. This [web service](#) fits the logistic regression model to the data and outputs the probability value (y) for each of the observations in the data.

This web service could be consumed by users – potentially through a mobile app, through a website, or even on a local computer, for example. But the purpose of the web service is also to serve as an example of how Azure Machine Learning can be used to create web services on top of R code. With just a few lines of R code and clicks of a button within Azure Machine Learning Studio, an experiment can be created with R code and published as a web service. The web service can then be published to the Azure Marketplace and consumed by users and devices across the world with no infrastructure setup by the author of the web service.

Consumption of web service

This web service gives the predicted values of the dependent variable based on the independent variables for all of the observations. The web service expects the end user to input data as a string where rows are separated by comma (,) and columns are separated by semicolon (;). The web service expects 1 row at a time and expects the first column to be the dependent variable. An example dataset could look like this:

Y	X1	X2
1	5	2
1	1	6
0	5.3	2.1
0	5	5
0	3	4
1	2	1
	3	4

Observations without a dependent variable should be input as "NA" for y. The data input for the above dataset would be the following string: "1;5;2;1;1;6;0;5.3;2;1;0;5;5;0;3;4;1;2;1;NA;3;4". The output is the predicted value for each of the rows based on the independent variables.

This service, as hosted on the Azure Marketplace, is an OData service; these may be called through POST or GET methods.

There are multiple ways of consuming the service in an automated fashion (an example app is [here](#)).

Starting C# code for web service consumption:

```
public class Input
{
    public string value;
}

public AuthenticationHeaderValue CreateBasicHeader(string username, string password)
{
    byte[] byteArray = System.Text.Encoding.UTF8.GetBytes(username + ":" + password);
    return new AuthenticationHeaderValue("Basic", Convert.ToBase64String(byteArray));
}

void Main()
{
    var input = new Input() { value = TextBox1.Text };
    var json = JsonConvert.SerializeObject(input);
    var actionUri = "PutAPIURLHere,e.g.https://api.datamarket.azure.com/.../v1/Score";
    var httpClient = new HttpClient();

    httpClient.DefaultRequestHeaders.Authorization = CreateBasicHeader("PutEmailAddressHere", "ChangeToAPIKey");
    httpClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

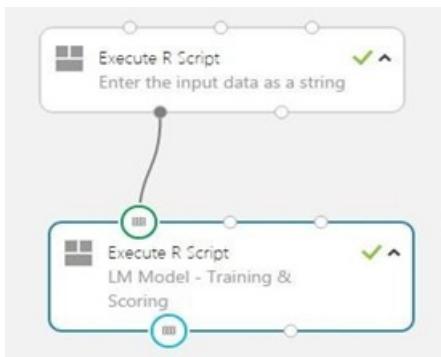
    var response = httpClient.PostAsync(actionUri, new StringContent(json));
    var result = response.Result.Content;
    var scoreResult = result.ReadAsStringAsync().Result;
}
```

Creation of web service

This web service was created using Azure Machine Learning. For a free trial, as well as introductory videos on creating experiments and [publishing web services](#), please see [azure.com/ml](#). Below is a screenshot of the experiment that created the web service and example code for each of the modules within the experiment.

From within Azure Machine Learning, a new blank experiment was created and two [Execute R Script](#) modules pulled onto the workspace. This web service runs an Azure Machine Learning experiment with an underlying R script. There are 2 parts to this experiment: schema definition, and training model + scoring. The first module

defines the expected structure of the input dataset, where the first variable is the dependent variable and the remaining variables are independent. The second module fits a generic logistic regression model for the input data.



Module 1:

```
#Schema definition
data <- data.frame(value = "1;2;3;1;5;6;0;8;9", stringsAsFactors=FALSE)
maml.mapOutputPort("data");
```

Module 2:

```
#GLM modeling
data <- maml.mapInputPort(1) # class: data.frame

data.split <- strsplit(data[1,], ",")[[1]]
data.split <- sapply(data.split, strsplit, ";", simplify = TRUE)
data.split <- sapply(data.split, strsplit, ";", simplify = TRUE)
data.split <- as.data.frame(t(data.split)) data.split <-
data.matrix(data.split)
data.split <- data.frame(data.split)

model <- glm(data.split$V1 ~., family="binomial", data=data.split)
out <- data.frame(predict(model,data.split, type="response"))
pred1 <- as.data.frame(out)
group <- array(1:nrow(pred1))
for (i in 1:nrow(pred1))
{
  if(as.numeric(pred1[i,])>0.5) {group[i]=1}
  else {group[i]=0}
}
pred2 <- as.data.frame(group)
maml.mapOutputPort("pred2");
```

Limitations

This is a very simple example of a binary classification web service. As can be seen from the example code above, no error catching is implemented and the service assumes everything is a binary/continuous variable (no categorical features allowed), as the service only inputs numeric values at the time of the creation of this web service. Also, the service currently handles limited data size, due to the request/response nature of the web service call and the fact that the model is being fit every time the web service is called.

FAQ

For frequently asked questions on consumption of the web service or publishing to the Azure Marketplace, see [here](#).

(deprecated) Cluster Model

1/17/2017 • 4 min to read • [Edit on GitHub](#)

NOTE

The Microsoft DataMarket is being retired and this API has been deprecated.

You can find many useful example experiments and APIs in the [Cortana Intelligence Gallery](#). For more information about the Gallery, see [Share and discover resources in the Cortana Intelligence Gallery](#).

How can we predict groups of credit cardholders' behaviors in order to reduce the charge-off risk of credit card issuers? How can we define groups of personality traits of employees in order to improve their performance at work? How can doctors classify patients into groups based on the characteristics of their diseases? In principle, all of these questions can be answered through cluster analysis.

NOTE

[Try Azure Machine Learning for free](#)

No credit card or Azure subscription needed. [Get started now >](#)

Cluster analysis classifies a set of observations into two or more mutually exclusive unknown groups based on combinations of variables. The purpose of cluster analysis is to discover a system of organizing observations, usually people or their characteristics, into groups, where members of the groups share properties in common. This [service](#) uses the K-Means methodology, a commonly used clustering technique, to cluster arbitrary data into groups. This web service takes the data and the number of k clusters as input, and produces predictions of which of the k groups to which each observations belongs.

This web service could be consumed by users – potentially through a mobile app, through a website, or even on a local computer, for example. But the purpose of the web service is also to serve as an example of how Azure Machine Learning can be used to create web services on top of R code. With just a few lines of R code and clicks of a button within Azure Machine Learning Studio, an experiment can be created with R code and published as a web service. The web service can then be published to the Azure Marketplace and consumed by users and devices across the world with no infrastructure setup by the author of the web service.

Consumption of web service

This web service groups the data into a set of k groups and outputs the group assignment for each row. The web service expects the end user to input data as a string where rows are separated by commas (,) and columns are separated by semicolons (;). The web service expects 1 row at a time. An example dataset could look like this:

X1	X2	X3
10	5	2
18	1	6
7	5	5
22	3	4
12	2	1
10	3	4

Suppose the user wanted to separate this data into 3 mutually exclusive groups. The data input for the above dataset would be the following: value = "10;5;2,18;1;6,7;5;5,22;3;4,12;2;1,10;3;4"; k="3". The output is the predicted group membership for each of the rows.

This service, as hosted on the Azure Marketplace, is an OData service; these may be called through POST or GET methods.

There are multiple ways of consuming the service in an automated fashion (an example app is [here](#)).

Starting C# code for web service consumption:

```
public class Input
{
    public string value;
    public string k;
}

public AuthenticationHeaderValue CreateBasicHeader(string username, string password)
{
    byte[] byteArray = System.Text.Encoding.UTF8.GetBytes(username + ":" + password);
    return new AuthenticationHeaderValue("Basic", Convert.ToBase64String(byteArray));
}

void Main()
{
    var input = new Input() { value = TextBox1.Text, k = TextBox2.Text };
    var json = JsonConvert.SerializeObject(input);
    var acitionUri = "PutAPIURLHere,e.g.https://api.datamarket.azure.com/..../v1/Score";
    var httpClient = new HttpClient();

    httpClient.DefaultRequestHeaders.Authorization = CreateBasicHeader("PutEmailAddressHere", "ChangeToAPIKey");
    httpClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

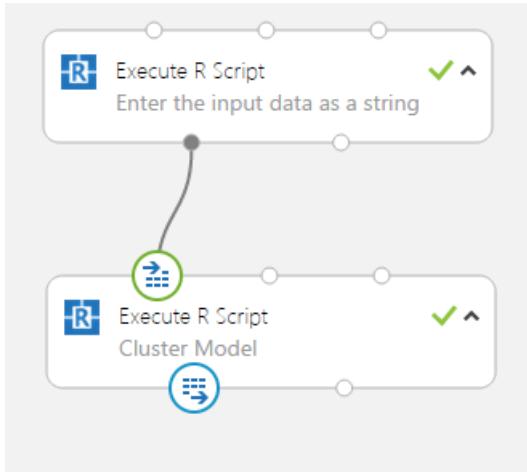
    var response = httpClient.PostAsync(acitionUri, new StringContent(json));
    var result = response.Result.Content;
    var scoreResult = result.ReadAsStringAsync().Result;
}
```

Creation of web service

This web service was created using Azure Machine Learning. For a free trial, as well as introductory videos on creating experiments and [publishing web services](#), please see [azure.com/ml](#). Below is a screenshot of the experiment that created the web service and example code for each of the modules within the experiment.

From within Azure Machine Learning, a new blank experiment was created and two [Execute R Script](#) modules pulled onto the workspace. The data schema was created with a simple [Execute R Script](#). Then, the data schema was linked to the cluster model section, again created with an [Execute R Script](#). In the [Execute R Script](#) used for the cluster model, the web service then utilizes the "k-means" function, which is prebuilt into the [Execute R Script](#) of

Azure Machine Learning.



Module 1:

```
#Enter the input data as a string
mydata<- data.frame(value = "1; 3; 5; 6; 7; 7; 5; 5; 6; 7; 2; 1; 3; 7; 2; 9; 56; 6; 1; 4; 5; 26; 4; 23, 15; 35; 6; 7; 12; 1, 32; 51; 62; 7; 21; 1", k=5,
stringsAsFactors=FALSE)

maml.mapOutputPort("mydata");
```

Module 2:

```
# Map 1-based optional input ports to variables
mydata <- maml.mapInputPort(1) # class: data.frame

data.split <- strsplit(mydata[1,], ",")[[1]]
data.split <- sapply(data.split, strsplit, ";", simplify = TRUE)
data.split <- sapply(data.split, strsplit, ";", simplify = TRUE)
data.split <- as.data.frame(t(data.split))

data.split <- data.matrix(data.split)
data.split <- data.frame(data.split)

# K-Means cluster analysis
fit <- kmeans(data.split, mydata$k) # k-cluster solution

# Get cluster means
aggregate(data.split, by=list(fit$cluster), FUN=mean)
# Append cluster assignment
mydatafinal <- data.frame(t(fit$cluster))
n_col<-ncol(mydatafinal)
colnames(mydatafinal)<- paste("V",1:n_col,sep="")

# Select data.frame to be sent to the output Dataset port
maml.mapOutputPort("mydatafinal");
```

Limitations

This is a very simple example of a clustering web service. As can be seen from the example code above, no error catching is implemented and the service assumes everything is a continuous variable (no categorical features allowed), as the service only inputs numeric values at the time of the creation of this web service. Also, the service currently handles limited data size, due to the request/response nature of the web service call and the fact that the model is being fit every time the web service is called.

FAQ

For frequently asked questions on consumption of the web service or publishing to the Azure Marketplace, see

[here.](#)

(deprecated) Multivariate Linear Regression

1/17/2017 • 4 min to read • [Edit on GitHub](#)

NOTE

The Microsoft DataMarket is being retired and this API has been deprecated.

You can find many useful example experiments and APIs in the [Cortana Intelligence Gallery](#). For more information about the Gallery, see [Share and discover resources in the Cortana Intelligence Gallery](#).

Suppose you have a dataset and would like to quickly predict a dependent variable (y) for each individual (i) based on independent variables. Linear regression is a popular statistical technique used for such predictions. Here the dependent variable y is assumed to be a continuous value.

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

A simple scenario could be where the researcher is trying to predict the weight of an individual (y) based on their height (x). A more advanced scenario could be where the researcher has additional information for the individual (such as weight, gender, race) and attempts to predict the weight of the individual. This [web service](#) fits the linear regression model to the data and outputs the predicted value (y) for each of the observations in the data.

This web service could be consumed by users – potentially through a mobile app, through a website, or even on a local computer, for example. But the purpose of the web service is also to serve as an example of how Azure Machine Learning can be used to create web services on top of R code. With just a few lines of R code and clicks of a button within Azure Machine Learning Studio, an experiment can be created with R code and published as a web service. The web service can then be published to the Azure Marketplace and consumed by users and devices across the world with no infrastructure setup by the author of the web service.

Consumption of web service

This web service gives the predicted values of the dependent variable based on the independent variables for all of the observations. The web service expects the end user to input data as a string where rows are separated by commas (,) and columns are separated by semicolons (;). The web service expects 1 row at a time and expects the first column to be the dependent variable. An example dataset could look like this:

Y	X1	X2
10	5	2
18	1	6
6	5.3	2.1
7	5	5
22	3	4
12	2	1
	3	4

Observations without a dependent variable should be input as "NA" for y. The data input for the above dataset would be the following string: "10;5;2,18;1;6,6;5.3;2.1,7;5;5,22;3;4,12;2;1,NA;3;4". The output is the predicted value for each of the rows based on the independent variables.

This service, as hosted on the Azure Marketplace, is an OData service; these may be called through POST or GET methods.

There are multiple ways of consuming the service in an automated fashion (an example app is [here](#)).

Starting C# code for web service consumption:

```
public class Input
{
    public string value;
}

public AuthenticationHeaderValue CreateBasicHeader(string username, string password)
{
    byte[] byteArray = System.Text.Encoding.UTF8.GetBytes(username + ":" + password);
    return new AuthenticationHeaderValue("Basic", Convert.ToBase64String(byteArray));
}

void Main()
{
    var input = new Input() { value = TextBox1.Text };
    var json = JsonConvert.SerializeObject(input);
    var acitionUri = "PutAPIURLHere,e.g.https://api.datamarket.azure.com/..../v1/Score";
    var httpClient = new HttpClient();

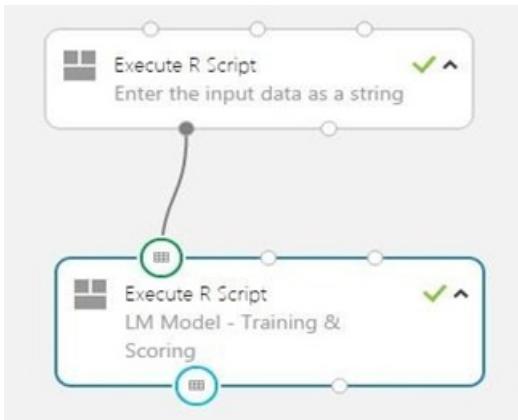
    httpClient.DefaultRequestHeaders.Authorization = CreateBasicHeader("PutEmailAddressHere", "ChangeToAPIKey");
    httpClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

    var response = httpClient.PostAsync(acitionUri, new StringContent(json));
    var result = response.Result.Content;
    var scoreResult = result.ReadAsStringAsync().Result;
}
```

Creation of web service

This web service was created using Azure Machine Learning. For a free trial, as well as introductory videos on creating experiments and [publishing web services](#), please see [azure.com/ml](#). Below is a screenshot of the experiment that created the web service and example code for each of the modules within the experiment.

From within Azure Machine Learning, a new blank experiment was created and two [Execute R Script](#) modules were pulled onto the workspace. This web service runs an Azure Machine Learning experiment with an underlying R script. There are 2 parts to this experiment: schema definition, and training model + scoring. The first module defines the expected structure of the input dataset, where the first variable is the dependent variable and the remaining variables are independent. The second module fits a generic linear regression model for the input data.



Module 1:

Schema definition

```
data <- data.frame(value = "1;2;3;4;5;6;7;8;9", stringsAsFactors=FALSE) maml.mapOutputPort("data");
```

Module 2:

LM modeling

```
data <- maml.mapInputPort(1) # class: data.frame

data.split <- strsplit(data[1,1], ",")[[1]]
data.split <- sapply(data.split, strsplit, ";", simplify = TRUE)
data.split <- sapply(data.split, strsplit, ";", simplify = TRUE)
data.split <- as.data.frame(t(data.split))
data.split <- data.matrix(data.split)
data.split <- data.frame(data.split)
model <- lm(data.split)

out = data.frame(predict(model,data.split))
out <- data.frame(t(out))

maml.mapOutputPort("out");
```

Limitations

This is a very simple example of a multiple linear regression web service. As can be seen from the example code above, no error catching is implemented and the service assumes everything is a continuous variable (no categorical features allowed), as the service only inputs numeric values at the time of the creation of this web service. Also, the service currently handles limited data size, due to the request/response nature of the web service call and the fact that the model is being fit every time the web service is called.

FAQ

For frequently asked questions on consumption of the web service or publishing to the Azure Marketplace, see [here](#).

(deprecated) Forecasting - Exponential Smoothing

1/17/2017 • 3 min to read • [Edit on GitHub](#)

NOTE

The Microsoft DataMarket is being retired and this API has been deprecated.

You can find many useful example experiments and APIs in the [Cortana Intelligence Gallery](#). For more information about the Gallery, see [Share and discover resources in the Cortana Intelligence Gallery](#).

This [web service](#) implements the Exponential Smoothing model (ETS) to produce predictions based on the historical data provided by the user. Will the demand for a specific product increase this year? Can I predict my product sales for the Christmas season, so that I can effectively plan my inventory? Forecasting models are apt to address such questions. Given the past data, these models examine hidden trends and seasonality to predict future trends.

NOTE

[Try Azure Machine Learning for free](#)

No credit card or Azure subscription needed. [Get started now >](#)

This web service could be consumed by users – potentially through a mobile app, through a website, or even on a local computer, for example. But the purpose of the web service is also to serve as an example of how Azure Machine Learning can be used to create web services on top of R code. With just a few lines of R code and clicks of a button within Azure Machine Learning Studio, an experiment can be created with R code and published as a web service. The web service can then be published to the Azure Marketplace and consumed by users and devices across the world with no infrastructure setup by the author of the web service.

Consumption of web service

This service accepts 4 arguments and calculates the ETS forecasts. The input arguments are:

- Frequency - Indicates the frequency of the raw data (daily/weekly/monthly/quarterly/yearly).
- Horizon - Future forecast time-frame.
- Date - Add in the new time series data for time.
- Value - Add in the new time series data values.

The output of the service is the calculated forecast values.

Sample input could be:

- Frequency - 12
- Horizon - 12
- Date -
1/15/2012;2/15/2012;3/15/2012;4/15/2012;5/15/2012;6/15/2012;7/15/2012;8/15/2012;9/15/2012;10/15/2012;11/15/2012;12/15/2012
1/15/2013;2/15/2013;3/15/2013;4/15/2013;5/15/2013;6/15/2013;7/15/2013;8/15/2013;9/15/2013;10/15/2013;11/15/2013;12/15/2013
1/15/2014;2/15/2014;3/15/2014;4/15/2014;5/15/2014;6/15/2014;7/15/2014;8/15/2014;9/15/2014
- Value -
3.479;3.68;3.832;3.941;3.797;3.586;3.508;3.731;3.915;3.844;3.634;3.549;3.557;3.785;3.782;3.601;3.544;3.556;3.65;3.709;3.682;3.511;
3.429;3.51;3.523;3.525;3.626;3.695;3.711;3.711;3.693;3.571;3.509

This service, as hosted on the Azure Marketplace, is an OData service; these may be called through POST or GET methods.

There are multiple ways of consuming the service in an automated fashion (an example app is [here](#)).

Starting C# code for web service consumption:

```

public class Input
{
    public string frequency;
    public string horizon;
    public string date;
    public string value;
}

public AuthenticationHeaderValue CreateBasicHeader(string username, string password)
{
    byte[] byteArray = System.Text.Encoding.UTF8.GetBytes(username + ":" + password);
    return new AuthenticationHeaderValue("Basic", Convert.ToBase64String(byteArray));
}

void Main()
{
    var input = new Input() { frequency = TextBox1.Text, horizon = TextBox2.Text, date = TextBox3.Text, value = TextBox4.Text };
    var json = JsonConvert.SerializeObject(input);
    var actionUri = "PutAPIURLHere,e.g.https://api.datamarket.azure.com/.../v1/Score";
    var httpClient = new HttpClient();

    httpClient.DefaultRequestHeaders.Authorization = CreateBasicHeader("PutEmailAddressHere", "ChangeToAPICKey");
    httpClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

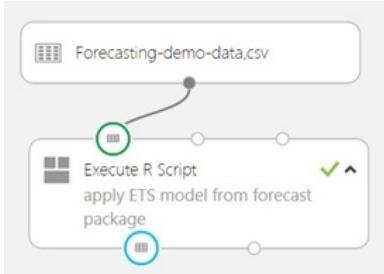
    var response = httpClient.PostAsync(actionUri, new StringContent(json));
    var result = response.Result.Content;
    var scoreResult = result.ReadAsStringAsync().Result;
}

```

Creation of web service

This web service was created using Azure Machine Learning. For a free trial, as well as introductory videos on creating experiments and [publishing web services](#), please see [azure.com/ml](#). Below is a screenshot of the experiment that created the web service and example code for each of the modules within the experiment.

From within Azure Machine Learning, a new blank experiment was created. Sample input data was uploaded with a predefined data schema. Linked to the data schema is an [Execute R Script](#) module that generates the ETS forecasting model by using 'ets' and 'forecast' functions from R.



Module 1:

Add in the CSV file with the data in the format shown below

frequency	horizon	date	value
12	24	1/1/2000;2/1/2000;3/1/2000;4/1/2000;5/1/2000;6/1/2000;7/1/2000; 8/1/2000;9/1/2000;10/1/2000;11/1/2000;12/1/2000;1/1/2001;2/1/2001; 3/1/2001;4/1/2001;5/1/2001;6/1/2001;7/1/2001;8/1/2001;9/1/2001; 10/1/2001;11/1/2001;12/1/2001;1/1/2002;2/1/2002;3/1/2002;4/1/2002; 5/1/2002;6/1/2002;7/1/2002;8/1/2002;9/1/2002;10/1/2002;11/1/2002	2403;2445;2605;2472;3047;2910;2918;3205;2868;2979;2945;3898;2693; 2631;2718;2914;3141;3178;3315;3620;2931;3322;3208;4301;2875;2800; 2851;3336;3559;3374;3735;3852;3485;3476;3317;4658;3106;3124;3304; 3434;3792;3954;4129;4142;4024;3990;3556;5204;3529;3520;4023;3967; 4028;4639;4537;4712;4503;4384;4664;6070;4755;4342;4689;4906;5516;

Module 2:

```

# Data input
data <- maml.mapInputPort(1) # class: data.frame
library(forecast)

# Preprocessing
colnames(data) <- c("frequency", "horizon", "dates", "values")
dates <- strsplit(data$dates, ";")[[1]]
values <- strsplit(data$values, ";")[[1]]

dates <- as.Date(dates, format = "%m/%d/%Y")
values <- as.numeric(values)

# Fit a time-series model
train_ts <- ts(values, frequency = data$frequency)
fit1 <- ets(train_ts)
train_model <- forecast(fit1, h = data$horizon)
plot(train_model)

# Produce forecasting
train_pred <- round(train_model$mean, 2)
data.forecast <- as.data.frame(t(train_pred))
colnames(data.forecast) <- paste("Forecast", 1:data$horizon, sep = "")

# Data output
maml.mapOutputPort("data.forecast");

```

Limitations

This is a very simple example for ETS forecasting. As can be seen from the example code above, no error catching is implemented, and the service assumes that all the variables are continuous/positive values and the frequency should be an integer greater than 1. The length of the date and value vectors should be the same. The date variable should adhere to the format 'mm/dd/yyyy'.

FAQ

For frequently asked questions on consumption of the web service or publishing to the Azure Marketplace, see [here](#).

(deprecated) Forecasting - ETS + STL

1/17/2017 • 3 min to read • [Edit on GitHub](#)

NOTE

The Microsoft DataMarket is being retired and this API has been deprecated.

You can find many useful example experiments and APIs in the [Cortana Intelligence Gallery](#). For more information about the Gallery, see [Share and discover resources in the Cortana Intelligence Gallery](#).

This [web service](#) implements Seasonal Trend Decomposition (STL) and Exponential Smoothing (ETS) models to produce predictions based on the historical data provided by the user. Will the demand for a specific product increase this year? Can I predict my product sales for the Christmas season, so that I can effectively plan my inventory? Forecasting models are apt to address such questions. Given the past data, these models examine hidden trends and seasonality to predict future trends.

NOTE

[Try Azure Machine Learning for free](#)

No credit card or Azure subscription needed. [Get started now >](#)

This web service could be consumed by users – potentially through a mobile app, through a website, or even on a local computer, for example. But the purpose of the web service is also to serve as an example of how Azure Machine Learning can be used to create web services on top of R code. With just a few lines of R code and clicks of a button within Azure Machine Learning Studio, an experiment can be created with R code and published as a web service. The web service can then be published to the Azure Marketplace and consumed by users and devices across the world with no infrastructure setup by the author of the web service.

Consumption of web service

This service accepts 4 arguments and calculates the forecasts. The input arguments are:

- Frequency - Indicates the frequency of the raw data (daily/weekly/monthly/quarterly/yearly).
- Horizon - Future forecast time-frame.
- Date - Add in the new time series data for time.
- Value - Add in the new time series data values.

The output of the service is the calculated forecast values.

Sample input could be:

- Frequency - 12
- Horizon - 12
- Date -
1/15/2012;2/15/2012;3/15/2012;4/15/2012;5/15/2012;6/15/2012;7/15/2012;8/15/2012;9/15/2012;10/15/2012;11/15/2012;12/15/2012
1/15/2013;2/15/2013;3/15/2013;4/15/2013;5/15/2013;6/15/2013;7/15/2013;8/15/2013;9/15/2013;10/15/2013;11/15/2013;12/15/2013
1/15/2014;2/15/2014;3/15/2014;4/15/2014;5/15/2014;6/15/2014;7/15/2014;8/15/2014;9/15/2014
- Value -
3.479;3.68;3.832;3.941;3.797;3.586;3.508;3.731;3.915;3.844;3.634;3.549;3.557;3.785;3.782;3.601;3.544;3.556;3.65;3.709;3.682;3.511;
3.429;3.51;3.523;3.525;3.626;3.695;3.711;3.711;3.693;3.571;3.509

This service, as hosted on the Azure Marketplace, is an OData service; these may be called through POST or GET methods.

There are multiple ways of consuming the service in an automated fashion (an example app is [here](#)).

Starting C# code for web service consumption:

```

public class Input
{
    public string frequency;
    public string horizon;
    public string date;
    public string value;
}

public AuthenticationHeaderValue CreateBasicHeader(string username, string password)
{
    byte[] byteArray = System.Text.Encoding.UTF8.GetBytes(username + ":" + password);
    return new AuthenticationHeaderValue("Basic", Convert.ToBase64String(byteArray));
}

void Main()
{
    var input = new Input() { frequency = TextBox1.Text, horizon = TextBox2.Text, date = TextBox3.Text, value = TextBox4.Text };      var json =
JsonConvert.SerializeObject(input);
    var actionUri = "PutAPIURLHere,e.g.https://api.datamarket.azure.com/.../v1/Score";
    var httpClient = new HttpClient();

    httpClient.DefaultRequestHeaders.Authorization = CreateBasicHeader("PutEmailAddressHere", "ChangeToAPIKey");
    httpClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

    var response = httpClient.PostAsync(actionUri, new StringContent(json));
    var result = response.Result.Content;
    var scoreResult = result.ReadAsStringAsync().Result;
}

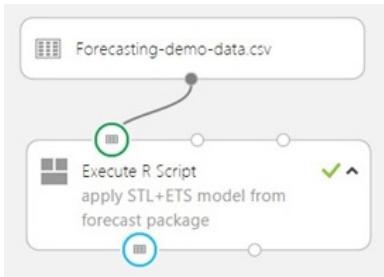
```

Creation of web service

This web service was created using Azure Machine Learning. For a free trial, as well as introductory videos on creating experiments and [publishing web services](#), please see [azure.com/ml](#). Below is a screenshot of the experiment that created the web service and example code for each of the modules within the experiment.

From within Azure Machine Learning, a new blank experiment was created. Sample input data was uploaded with a predefined data schema. Linked to the data schema is an [Execute R Script](#) module, which generates STL and ETS forecasting models by using 'stl', 'ets', and 'forecast' functions from R.

Experiment flow:



Module 1:

Add in the CSV file with the data in the format shown below

frequency	horizon	date	value
12	24	1/1/2000;2/1/2000;3/1/2000;4/1/2000;5/1/2000;6/1/2000;7/1/2000; 8/1/2000;9/1/2000;10/1/2000;11/1/2000;12/1/2000;1/1/2001;2/1/2001; 3/1/2001;4/1/2001;5/1/2001;6/1/2001;7/1/2001;8/1/2001;9/1/2001; 10/1/2001;11/1/2001;12/1/2001;1/1/2002;2/1/2002;3/1/2002;4/1/2002; 5/1/2002;6/1/2002;7/1/2002;8/1/2002;9/1/2002;10/1/2002;11/1/2002;	2403;2445;2605;2472;3047;2910;2918;3205;2868;2979;2945;3898;2693; 2631;2718;2914;3141;3178;3315;3620;2931;3322;3208;4301;2875;2800; 2851;3336;3559;3374;3735;3852;3485;3476;3317;4658;3106;3124;3304; 3434;3792;3954;4129;4142;4024;3990;3556;5204;3529;3520;4023;3967; 4028;4639;4537;4712;4503;4384;4664;6070;4755;4342;4689;4906;5516;

Module 2:

```

# Data input
data <- maml.mapInputPort(1) # class: data.frame
library(forecast)

# Preprocessing
colnames(data) <- c("frequency", "horizon", "dates", "values")
dates <- strsplit(data$dates, ";")[[1]]
values <- strsplit(data$values, ";")[[1]]

dates <- as.Date(dates, format = "%m/%d/%Y")
values <- as.numeric(values)

# Fit a time series model
train_ts <- ts(values, frequency = data$frequency)
fit1 <- stl(train_ts, s.window = "periodic")
train_model <- forecast(fit1, h = data$horizon, method = 'ets')
plot(train_model)

# Produce forecasting
train_pred <- round(train_model$mean, 2)
data.forecast <- as.data.frame(t(train_pred))
colnames(data.forecast) <- paste("Forecast", 1:data$horizon, sep = "")

# Data output
maml.mapOutputPort("data.forecast");

```

Limitations

This is a very simple example for ETS + STL forecasting. As can be seen from the example code above, no error catching is implemented, and the service assumes that all the variables are continuous/positive values and the frequency should be an integer greater than 1. The length of the date and value vectors should be the same, and the length of the time series should be greater than $2 * \text{frequency}$. The date variable should adhere to the format 'mm/dd/yyyy'.

FAQ

For frequently asked questions on consumption of the web service or publishing to the Azure Marketplace, see [here](#).

(deprecated) Forecasting - Autoregressive Integrated Moving Average (ARIMA)

1/17/2017 • 3 min to read • [Edit on GitHub](#)

NOTE

The Microsoft DataMarket is being retired and this API has been deprecated.

You can find many useful example experiments and APIs in the [Cortana Intelligence Gallery](#). For more information about the Gallery, see [Share and discover resources in the Cortana Intelligence Gallery](#).

This [service](#) implements Autoregressive Integrated Moving Average (ARIMA) to produce predictions based on the historical data provided by the user. Will the demand for a specific product increase this year? Can I predict my product sales for the Christmas season, so that I can effectively plan my inventory? Forecasting models are apt to address such questions. Given the past data, these models examine hidden trends and seasonality to predict future trends.

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

This web service could be consumed by users – potentially through a mobile app, through a website, or even on a local computer, for example. But the purpose of the web service is also to serve as an example of how Azure Machine Learning can be used to create web services on top of R code. With just a few lines of R code and clicks of a button within Azure Machine Learning Studio, an experiment can be created with R code and published as a web service. The web service can then be published to the Azure Marketplace and consumed by users and devices across the world with no infrastructure setup by the author of the web service.

Consumption of web service

This service accepts 4 arguments and calculates the ARIMA forecasts. The input arguments are:

- Frequency - Indicates the frequency of the raw data (daily/weekly/monthly/quarterly/yearly).
- Horizon - Future forecast time-frame.
- Date - Add in the new time series data for time.
- Value - Add in the new time series data values.

The output of the service is the calculated forecast values.

Sample input could be:

- Frequency - 12
- Horizon - 12
- Date -
1/15/2012;2/15/2012;3/15/2012;4/15/2012;5/15/2012;6/15/2012;7/15/2012;8/15/2012;9/15/2012;10/15/2012;11/15/2012;12/15/2012
1/15/2013;2/15/2013;3/15/2013;4/15/2013;5/15/2013;6/15/2013;7/15/2013;8/15/2013;9/15/2013;10/15/2013;11/15/2013;12/15/2013
1/15/2014;2/15/2014;3/15/2014;4/15/2014;5/15/2014;6/15/2014;7/15/2014;8/15/2014;9/15/2014
- Value -
3.479;3.68;3.832;3.941;3.797;3.586;3.508;3.731;3.915;3.844;3.634;3.549;3.557;3.785;3.782;3.601;3.544;3.556;3.65;3.709;3.682;3.511;
3.429;3.51;3.523;3.525;3.626;3.695;3.711;3.711;3.693;3.571;3.509

This service, as hosted on the Azure Marketplace, is an OData service; these may be called through POST or GET methods.

There are multiple ways of consuming the service in an automated fashion (an example app is [here](#)).

Starting C# code for web service consumption:

```

public class Input
{
    public string frequency;
    public string horizon;
    public string date;
    public string value;
}

public AuthenticationHeaderValue CreateBasicHeader(string username, string password)
{
    byte[] byteArray = System.Text.Encoding.UTF8.GetBytes(username + ":" + password);
    return new AuthenticationHeaderValue("Basic", Convert.ToBase64String(byteArray));
}

void Main()
{
    var input = new Input() { frequency = TextBox1.Text, horizon = TextBox2.Text, date = TextBox3.Text, value = TextBox4.Text };
    var json = JsonConvert.SerializeObject(input);
    var actionUri = "PutAPIURLHere,e.g.https://api.datamarket.azure.com/.../v1/Score";

    var httpClient = new HttpClient();
    httpClient.DefaultRequestHeaders.Authorization = CreateBasicHeader("PutEmailAddressHere", "ChangeToAPIKey");
    httpClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));
    var query = httpClient.PostAsync(actionUri, new StringContent(json));
    var result = query.Result.Content;
    var scoreResult = result.ReadAsStringAsync().Result;
}

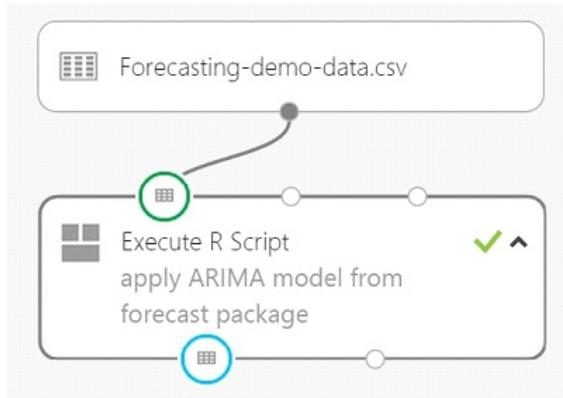
```

Creation of web service

This web service was created using Azure Machine Learning. For a free trial, as well as introductory videos on creating experiments and [publishing web services](#), please see [azure.com/ml](#). Below is a screenshot of the experiment that created the web service and example code for each of the modules within the experiment.

From within Azure Machine Learning, a new blank experiment was created. Sample input data was uploaded with a predefined data schema. Linked to the data schema is an [Execute R Script](#) module, which generates the ARIMA forecasting model by using 'auto.arima' and 'forecast' functions from R.

Experiment flow:



Module 1:

Add in the CSV file with the data in the format shown below

frequency	horizon	date	value
12	24	1/1/2000;2/1/2000;3/1/2000;4/1/2000;5/1/2000;6/1/2000;7/1/2000; 8/1/2000;9/1/2000;10/1/2000;11/1/2000;12/1/2000;1/1/2001;2/1/2001; 3/1/2001;4/1/2001;5/1/2001;6/1/2001;7/1/2001;8/1/2001;9/1/2001; 10/1/2001;11/1/2001;12/1/2001;1/1/2002;2/1/2002;3/1/2002;4/1/2002; 5/1/2002;6/1/2002;7/1/2002;8/1/2002;9/1/2002;10/1/2002;11/1/2002;	2403;2445;2605;2472;3047;2910;2918;3205;2868;2979;2945;3898;2693; 2631;2718;2914;3141;3178;3315;3620;2931;3322;3208;4301;2875;2800; 2851;3336;3559;3374;3735;3852;3485;3476;3317;4658;3106;3124;3304; 3434;3792;3954;4129;4142;4024;3990;3556;5204;3529;3520;4023;3967; 4028;4639;4537;4712;4503;4384;4664;6070;4755;4342;4689;4906;5516;

Module 2:

```

# data input
data <- maml.mapInputPort(1) # class: data.frame
library(forecast)

# preprocessing
colnames(data) <- c("frequency", "horizon", "dates", "values")
dates <- strsplit(data$dates, ";")[[1]]
values <- strsplit(data$values, ";")[[1]]

dates <- as.Date(dates, format = "%m/%d/%Y")
values <- as.numeric(values)

# fit a time-series model
train_ts <- ts(values, frequency = data$frequency)
fit1 <- auto.arima(train_ts)
train_model <- forecast(fit1, h = data$horizon)
plot(train_model)

# produce forecasting
train_pred <- round(train_model$mean, 2)
data.forecast <- as.data.frame(t(train_pred))
colnames(data.forecast) <- paste("Forecast", 1:data$horizon, sep = "")

# data output
maml.mapOutputPort("data.forecast");

```

Limitations

This is a very simple example for ARIMA forecasting. As can be seen from the example code above, no error catching is implemented, and the service assumes that all the variables are continuous/positive values and the frequency should be an integer greater than 1. The length of the date and value vectors should be the same. The date variable should adhere to the format 'mm/dd/yyyy'.

FAQ

For frequently asked questions on consumption of the web service or publishing to marketplace, see [here](#).

(deprecated) Survival Analysis

1/17/2017 • 6 min to read • [Edit on GitHub](#)

NOTE

The Microsoft DataMarket is being retired and this API has been deprecated.

You can find many useful example experiments and APIs in the [Cortana Intelligence Gallery](#). For more information about the Gallery, see [Share and discover resources in the Cortana Intelligence Gallery](#).

Under many scenarios, the main outcome under assessment is the time to an event of interest. In other words, the question "when this event will occur?" is asked. As examples, consider situations where the data describes the elapsed time (days, years, mileage, etc.) until the event of interest (disease relapse, PhD degree received, brake pad failure) occurs. Each instance in the data represents a specific object (a patient, a student, a car, etc.).

NOTE

[Try Azure Machine Learning for free](#)

No credit card or Azure subscription needed. [Get started now >](#)

This [web service](#) answers the question "what is the probability that the event of interest will occur by time n for object x?" By providing a survival analysis model, this web service enables users to supply data to train the model and test it. The main theme of the experiment is to model the length of the elapsed time until the event of interest occurs.

This web service could be consumed by users – potentially through a mobile app, through a website, or even on a local computer, for example. But the purpose of the web service is also to serve as an example of how Azure Machine Learning can be used to create web services on top of R code. With just a few lines of R code and clicks of a button within Azure Machine Learning Studio, an experiment can be created with R code and published as a web service. The web service can then be published to the Azure Marketplace and consumed by users and devices across the world with no infrastructure setup by the author of the web service.

Consumption of web service

The input data schema of the web service is shown in the following table. Six pieces of information are needed as the input: training data, testing data, time of interest, the index of "time" dimension, the index of "event" dimension, and the variable types (continuous or factor). The training data is represented with a string, where the rows are separated by comma, and the columns are separated by semicolon. The number of features of the data is flexible. All the elements in the input string must be numeric. In the training data, the "time" dimension indicates the number of time units (days, years, mileage, etc.) elapsed since the starting point of the study (a patient receiving drug treatment programs, a student starting PhD study, a car starting to be driven, etc.) until the event of interest (the patient returning to drug usage, the student obtaining the PhD degree, the car having brake pad failure, etc.) occurs. The "event" dimension indicates whether the event of interest occurs at the end of the study. A value of "event=1" means that the event of interest occurs at the time indicated by the "time" dimension; "event=0" means that the event of interest has not occurred by the time indicated by the "time" dimension.

- **trainingdata** - A character string. Rows are separated by comma, and columns are separated by semicolon. Each row includes "time" dimension, "event" dimension, and predictor variables.

- testingdata - One row of data that contains predictor variables for a particular object.
- time_of_interest - The elapsed time of interest n.
- index_time - The column index of the "time" dimension (starting from 1).
- index_event - The column index of the "event" dimension (starting from 1).
- variable_types - A character string with semicolons as separators in it. 0 represents continuous variables and 1 represents factor variables.

The output is the probability of an event occurring by a specific time.

This service, as hosted on the Azure Marketplace, is an OData service; these may be called through POST or GET methods.

There are multiple ways of consuming the service in an automated fashion (an example app is [here](#)).

Starting C# code for web service consumption:

```
public class Input
{
    public string trainingdata;
    public string testingdata;
    public string timeofinterest;
    public string indextime;
    public string indexevent;
    public string variabletypes;
}

public AuthenticationHeaderValue CreateBasicHeader(string username, string password)
{
    byte[] byteArray = System.Text.Encoding.UTF8.GetBytes(username + ":" + password);
    return new AuthenticationHeaderValue("Basic", Convert.ToBase64String(byteArray));
}

void Main()
{
    var input = new Input() { trainingdata = TextBox1.Text, testingdata = TextBox2.Text, timeofinterest = TextBox3.Text, indextime = TextBox4.Text,
    indexevent = TextBox5.Text, variabletypes = TextBox6.Text };
    var json = JsonConvert.SerializeObject(input);
    var acitionUri = "PutAPIURLHere,e.g.https://api.datamarket.azure.com/..../v1/Score";
    var httpClient = new HttpClient();

    httpClient.DefaultRequestHeaders.Authorization = CreateBasicHeader("PutEmailAddressHere", "ChangeToAPIKey");
    httpClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

    var response = httpClient.PostAsync(acitionUri, new StringContent(json));
    var result = response.Result.Content;
    var scoreResult = result.ReadAsStringAsync().Result;
}
```

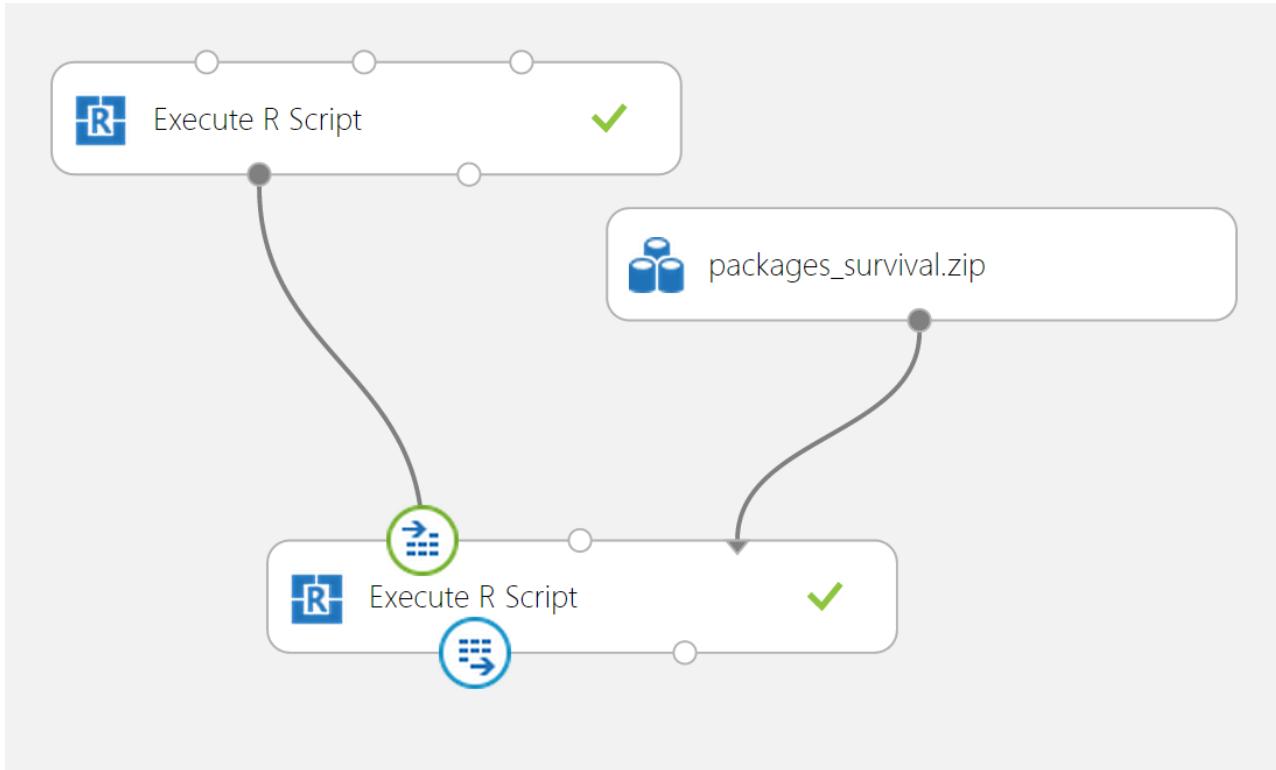
The interpretation of this test is as follows. Assuming the goal of the data is to model the elapsed time until the return to drug usage for the patients who received one of the two treatment programs. The output of the web service reads: for patients being 35 years old, having previous drug treatment 2 times, taking the long residential treatment program, and with both heroin and cocaine usage, the probability of returning to the drug usage is 95.64% by day 500.

Creation of web service

This web service was created using Azure Machine Learning. For a free trial, as well as introductory videos on creating experiments and [publishing web services](#), please see [azure.com/ml](#). Below is a screenshot of the experiment that created the web service and example code for each of the modules within the experiment.

From within Azure Machine Learning, a new blank experiment was created and two [Execute R Script](#) modules were pulled onto the workspace. The data schema was created with a simple [Execute R Script](#), which defines the input data schema for the web service. This module is then linked to the second [Execute R Script](#) module, which does major work. This module does data preprocessing, model building, and predictions. In the data preprocessing step, the input data represented by a long string is transformed and converted into a data frame. In the model building step, an external R package "survival_2.37-7.zip" is first installed for conducting survival analysis. Then the "coxph" function is executed after a series data processing tasks. The details of the "coxph" function for survival analysis can be read from the R documentation. In the prediction step, a testing instance is supplied into the trained model with the "surfit" function, and the survival curve for this testing instance is produced as "curve" variable. Finally, the probability of the time of interest is obtained.

Experiment flow:



Module 1:

```

#Data schema with example data (replaced with data from web service)
trainingdata="53;1;29;0;0;3;79;1;34;0;1;2;45;1;27;0;1;1;37;1;24;0;1;1;122;1;30;0;1;1;655;0;41;0;0;1;166;1;30;0;0;3;227;1;29;0;0;3;805;0;30;0;0;1;104;1;24
;0;0;1;90;1;32;0;0;1;373;1;26;0;0;1;70;1;36;0;0;1"
testdata="35;2;1;1"
time_of_interest="500"
index_time="1"
index_event="2"

# 0 - continuous; 1 - factor
variable_types="0;0;1;1"

sampleInput=data.frame(trainingdata,testdata,time_of_interest,index_time,index_event,variable_types)

maml.mapOutputPort("sampleInput"); #send data to output port

```

Module 2:

```

#Read data from input port
data<- maml.mapInputPort(1)
colnames(data)<- c("trainingdata","testdata","time_of_interest","index_time","index_event","variable_types")

# Preprocessing training data
trainningdata=data$trainningdata
y=strsplit(as.character(data$trainningdata),",")

```

```

n_row=length(unlist(y))
z=sapply(unlist(y),strsplit,;",",simplify = TRUE)
mydata<- data.frame(matrix(unlist(z), nrow=n_row, byrow=T), stringsAsFactors=FALSE)
n_col=ncol(mydata)

# Preprocessing testing data
testingdata=as.character(data$testingdata)
testingdata=unlist(strsplit(testingdata,";"))

# Preprocessing other input parameters
time_of_interest=data$time_of_interest
time_of_interest=as.numeric(as.character(time_of_interest))
index_time = data$index_time
index_event = data$index_event
variable_types = data$variable_types

# Necessary R packages
install.packages("src/packages_survival/survival_2.37-7.zip",lib=".",
repos=NULL,verbose=TRUE)
library(survival)

# Prepare to build model
attach(mydata)

for(i in 1:n_col){ mydata[,i]=as.numeric(mydata[,i])}
d_time=paste("X",index_time,sep = "")
d_event=paste("X",index_event,sep = "")
v_time_event <- c(d_time,d_event)
v_predictors = names(mydata)[!(names(mydata) %in% v_time_event)]

variable_types = unlist(strsplit(as.character(variable_types),";"))

len = length(v_predictors)
c="" # Construct the execution string
for (i in 1:len){
if(i==len){
if(variable_types[i]!=0){ c=paste(c, "factor(",v_predictors[i],")",sep="")}
else{ c=paste(c, v_predictors[i]);}
} else {
if(variable_types[i]!=0){c=paste(c, "factor(",v_predictors[i],") +",sep="")}
else{c=paste(c, v_predictors[i],"+"})}
}
}
f=paste("coxph(Surv(",d_time,",",d_event,") ~")
f=paste(f,c)
f=paste(f,", data=mydata )")

# Fit a Cox proportional hazards model and get the predicted survival curve for a testing instance
fit=eval(parse(text=f))

testingdata = as.data.frame(matrix(testingdata, ncol=len,byrow = TRUE),stringsAsFactors=FALSE)
names(testingdata)=v_predictors
for (i in 1:len){ testingdata[,i]=as.numeric(testingdata[,i])}

curve=survfit(fit,testingdata)

# Based on user input, find the event occurrence probability
position_closest=which.min(abs(prob_event$time - time_of_interest))

if(prob_event[position_closest,"time"]==time_of_interest){# exact match
output=prob_event[position_closest,"prob"]
} else {# not exact match
if(time_of_interest>max(prob_event$time)){
output=prob_event[position_closest,"prob"]
} else if(time_of_interest<min(prob_event$time)){
output=prob_event[position_closest,"prob"]
} else {output=(prob_event[position_closest,"prob"]+prob_event[position_closest+1,"prob"])/2}
}

#Pull out results to send to web service

```

```
output=paste(round(100*output, 2), "%")
maml.mapOutputPort("output"); #output port
```

Limitations

This web service can take only numerical values as feature variables (columns). The “event” column can take only value 0 or 1. The “time” column needs to be a positive integer.

FAQ

For frequently asked questions on consumption of the web service or publishing to the Azure Marketplace, see [here](#).

(deprecated) Lexicon Based Sentiment Analysis

1/17/2017 • 5 min to read • [Edit on GitHub](#)

NOTE

The Microsoft DataMarket is being retired and this API has been deprecated.

You can find many useful example experiments and APIs in the [Cortana Intelligence Gallery](#). For more information about the Gallery, see [Share and discover resources in the Cortana Intelligence Gallery](#).

How can you measure users' opinions and attitudes toward brands or topics in online social networks, such as Facebook posts, tweets, reviews, etc.? Sentiment analysis provides a method for analyzing such questions.

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

There are generally two methods for sentiment analysis. One is using a supervised learning algorithm, and the other can be treated as unsupervised learning. A supervised learning algorithm generally builds a classification model on a large annotated corpus. Its accuracy is mainly based on the quality of the annotation, and usually the training process will take a long time. Besides that, when we apply the algorithm to another domain, the result is usually not good. Compared to supervised learning, lexicon-based unsupervised learning uses a sentiment dictionary, which doesn't require storing a large data corpus and training - which makes the whole process much faster.

Our [service](#) is built on the MPQA Subjectivity Lexicon (http://mpqa.cs.pitt.edu/lexicons/subj_lexicon/), which is one of the most commonly used subjectivity lexicons. There are 5097 negative and 2533 positive words in MPQA. And all of these words are annotated as strong or weak polarity. The whole corpus is under GNU General Public License. The web service can be applied to any short sentences, such as tweets and Facebook posts.

This web service could be consumed by users – potentially through a mobile app, through a website, or even on a local computer for example. But the purpose of the web service is also to serve as an example of how Azure Machine Learning can be used to create web services on top of R code. With just a few lines of R code and clicks of a button within Azure Machine Learning Studio, an experiment can be created with R code and published as a web service. The web service can then be published to the Azure Marketplace and consumed by users and devices across the world with no infrastructure setup by the author of the web service.

Consumption of web service

The input data can be any text, but the web service works better with short sentences. The output is a numeric value between -1 and 1. Any value below 0 denotes that the sentiment of the text is negative; positive if above 0. The absolute value of the result denotes the strength of the associated sentiment.

This service, as hosted on the Azure Marketplace, is an OData service; these may be called through POST or GET methods.

There are multiple ways of consuming the service in an automated fashion (an example app is [here](#)).

Starting C# code for web service consumption:

```
public class ScoreResult
{
    [DataMember]
    public double result
    {
        get;
        set;
    }
}

void main()
{
    using (var wb = new WebClient())
    {
        var actionUri = new Uri("PutAPIURLHere,e.g.https://api.datamarket.azure.com/....v1/Score");
        DataServiceProvider ctx = new DataServiceProvider(actionUri);
        var cred = new NetworkCredential("PutEmailAddressHere", "ChangeToAPIKey");
        var cache = new CredentialCache();

        cache.Add(actionUri, "Basic", cred);
        ctx.Credentials = cache;
        var query = ctx.Execute<ScoreResult>(actionUri, "POST", true, new BodyOperationParameter("Text", TextBox1.Text));
        ScoreResult scoreResult = query.ElementAt(0);
        double result = scoreResult.result;
    }
}
```

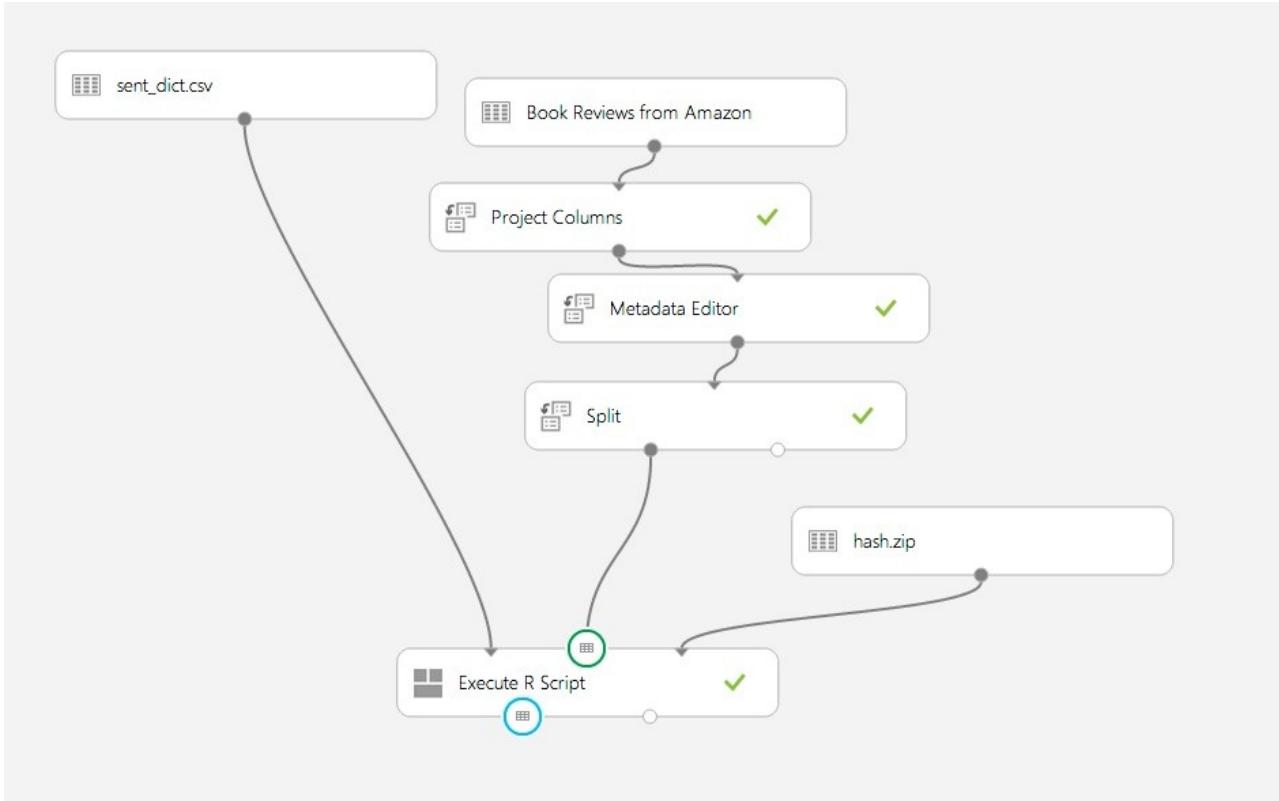
The input is "Today is a good day." The output is "1", which indicates a positive sentiment associated with the input sentence.

Creation of web service

This web service was created using Azure Machine Learning. For a free trial, as well as introductory videos on creating experiments and [publishing web services](#), please see [azure.com/ml](#). Below is a screenshot of the experiment that created the web service and example code for each of the modules within the experiment.

From within Azure Machine Learning, a new blank experiment was created. The figure below shows the experiment flow of lexicon-based sentiment analysis. The "sent_dict.csv" file is the MPQA subjectivity lexicon, and is set as one of the inputs of [Execute R Script](#). Another input is a sampled review from the Amazon review dataset for test, where we performed selection, column name modification, and split operations. We use a hash package to store the subjectivity lexicon in the memory and accelerate the score computation process. The whole text will be tokenized by "tm" package and compared with the word in the sentiment dictionary. Finally, a score will be calculated by adding the weight of each subjective word in the text.

Experiment flow:



Module 1:

```

# Map 1-based optional input ports to variables
sent_dict_data<- maml.mapInputPort(1) # class: data.frame
dataset2<- maml.mapInputPort(2) # class: data.frame
    
```

Install hash package

```

install.packages("src/hash_2.2.6.zip", lib = ".", repos = NULL, verbose = TRUE)
success <- library("hash", lib.loc = ".", logical.return = TRUE, verbose = TRUE)
library(tm)
library(stringr)

#create sentiment dictionary
negation_word <- c("not","nor", "no")
result <- c()
sent_dict <- hash()
sent_dict <- hash(sent_dict_data$word, sent_dict_data$polarity)

# Compute sentiment score for each document
for (m in 1:nrow(dataset2)){
  polarity_ratio <- 0
  polarity_total <- 0
  not <- 0
  sentence <- tolower(dataset2[m,1])
  if(nchar(sentence)>0){
    token_array <- scan_tokenizer(sentence)
    for (j in 1:length(token_array)){
      word = str_replace_all(token_array[j], "[^[:alnum:]]", "")
      for (k in 1:length(negation_word)){
        if(word == negation_word[k]){
          not <- (not+1) %% 2
        }
      }
      if(word != ""){
        if(!is.null(sent_dict[[word]])){
          polarity_ratio <- polarity_ratio + (-2*not+1)*sent_dict[[word]]
          polarity_total <- polarity_total + abs(sent_dict[[word]])
        }
      }
    }
    if(polarity_total>0){
      result <- c(result, polarity_ratio/polarity_total)
    }else{
      result<- c(result,0)
    }
  }
}

# Sample operation
data.set <- data.frame(result)

# Select data.frame to be sent to the output Dataset port
maml.mapOutputPort("data.set")

```

Limitations

From an algorithm perspective, lexicon-based sentiment analysis is a general sentiment analysis tool, which may not perform better than the classification method for specific fields. The negation problem is not well dealt with. We hardcode several negation words in our program, but a better way is using a negation dictionary and build some rules. The web service performs better on short and simple sentences, such as tweets and Facebook posts, than on long and complex sentences such as Amazon reviews.

FAQ

For frequently asked questions on consumption of the web service or publishing to the Azure Marketplace, see [here](#).

(deprecated) Difference in Proportions Test

1/17/2017 • 3 min to read • [Edit on GitHub](#)

NOTE

The Microsoft DataMarket is being retired and this API has been deprecated.

You can find many useful example experiments and APIs in the [Cortana Intelligence Gallery](#). For more information about the Gallery, see [Share and discover resources in the Cortana Intelligence Gallery](#).

Are two proportions statistically different? Suppose a user wants to compare two movies to determine if one movie has a significantly higher proportion of 'likes' when compared to the other. With a large sample, there could be a statistically significant difference between the proportions 0.50 and 0.51. With a small sample, there may not be enough data to determine if these proportions are actually different.

NOTE

[Try Azure Machine Learning for free](#)

No credit card or Azure subscription needed. [Get started now >](#)

This [web service](#) conducts a hypothesis test of the difference in two proportions based on user input of the number of successes and the total number of trials for the 2 comparison groups. In one possible scenario, this web service could be called from within a movie comparison app, telling the user whether one of the movies is really 'liked' more often than the other, based on movie ratings.

This web service could be consumed by users – potentially through a mobile app, through a website, or even on a local computer, for example. But the purpose of the web service is also to serve as an example of how Azure Machine Learning can be used to create web services on top of R code. With just a few lines of R code and clicks of a button within Azure Machine Learning Studio, an experiment can be created with R code and published as a web service. The web service can then be published to the Azure Marketplace and consumed by users and devices across the world with no infrastructure setup by the author of the web service.

Consumption of web service

This service accepts 4 arguments and does a hypothesis test of proportions.

The input arguments are:

- Successes1 - Number of success events in sample 1.
- Successes2 - Number of success events in sample 2.
- Total1 - Size of sample 1.
- Total2 - Size of sample 2.

The output of the service is the result of the hypothesis test along with the chi-square statistic, df, p-value, and proportion in sample 1/2 and confidence interval bounds.

This service, as hosted on the Azure Marketplace, is an OData service; these may be called through POST or GET methods.

There are multiple ways of consuming the service in an automated fashion (an example app is [here](#)).

Starting C# code for web service consumption:

```
public class Input
{
    public string successes1;
    public string successes2;
    public string total1;
    public string total2;
}

public AuthenticationHeaderValue CreateBasicHeader(string username, string password)
{
    byte[] byteArray = System.Text.Encoding.UTF8.GetBytes(username + ":" + password);
    return new AuthenticationHeaderValue("Basic", Convert.ToBase64String(byteArray));
}

void Main()
{
    var input = new Input() { successes1 = TextBox1.Text, successes2 = TextBox2.Text, total1 = TextBox3.Text, total2 = TextBox4.Text };
    var json = JsonConvert.SerializeObject(input);
    var actionUri = "PutAPIURLHere,e.g.https://api.datamarket.azure.com/..../v1/Score";
    var httpClient = new HttpClient();

    httpClient.DefaultRequestHeaders.Authorization = CreateBasicHeader("PutEmailAddressHere", "ChangeToAPIKey");
    httpClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

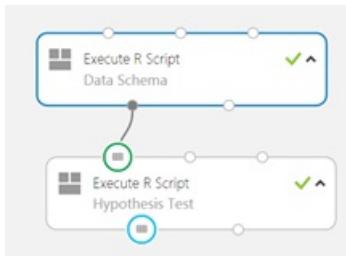
    var response = httpClient.PostAsync(actionUri, new StringContent(json));
    var result = response.Result.Content;
    var scoreResult = result.ReadAsStringAsync().Result;
}
```

Creation of web service

This web service was created using Azure Machine Learning. For a free trial, as well as introductory videos on creating experiments and [publishing web services](#), please see [azure.com/ml](#). Below is a screenshot of the experiment that created the web service and example code for each of the modules within the experiment.

From within Azure Machine Learning, a new blank experiment was created with two [Execute R Script](#) modules. In the first module the data schema is defined, while the second module uses the prop.test command within R to perform the hypothesis test for 2 proportions.

Experiment flow:



Module 1:

```
####Schema definition
data.set=data.frame(successes1=50,successes2=60,total1=100,total2=100);
maml.mapOutputPort("data.set"); #send data to output port
dataset1 = maml.mapInputPort(1) #read data from input port
```

Module 2:

```
test=prop.test(c(dataset1$successes1[1],dataset1$successes2[1]),c(dataset1$total1[1],dataset1$total2[1])) #conduct hypothesis test

result=data.frame(
  result=ifelse(test$p.value<0.05,"The proportions are different!",
               "The proportions aren't different statistically."),
  ChiSquarestatistic=round(test$statistic,2),df=test$parameter,
  pvalue=round(test$p.value,4),
  proportion1=round(test$estimate[1],4),
  proportion2=round(test$estimate[2],4),
  confintlow=round(test$conf.int[1],4),
  confinhigh=round(test$conf.int[2],4))

maml.mapOutputPort("result"); #output port
```

Limitations

This is a very simple example for a test of difference in 2 proportions. As can be seen from the example code above, no error catching is implemented and the service assumes that all the variables are continuous.

FAQ

For frequently asked questions on consumption of the web service or publishing to the Azure Marketplace, see [here](#).

(deprecated) Normal Distribution Suite

1/17/2017 • 6 min to read • [Edit on GitHub](#)

NOTE

The Microsoft DataMarket is being retired and this API has been deprecated.

You can find many useful example experiments and APIs in the [Cortana Intelligence Gallery](#). For more information about the Gallery, see [Share and discover resources in the Cortana Intelligence Gallery](#).

The Normal Distribution Suite is a set of sample web services ([Generator](#), [Quantile Calculator](#), [Probability Calculator](#)) that help in generating and handling normal distributions. The services allow generating a normal distribution sequence of any length, calculating quantiles from a given probability, and calculating probability from a given quantile. Each of the services emits different outputs based on the selected service (see description below). The Normal Distribution Suite is based on the R functions qnorm, rnorm, and pnorm, which are included in R stats package.

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

This web service could be consumed by users – potentially through a mobile app, through a website, or even on a local computer, for example. But the purpose of the web service is also to serve as an example of how Azure Machine Learning can be used to create web services on top of R code. With just a few lines of R code and clicks of a button within Azure Machine Learning Studio, an experiment can be created with R code and published as a web service. The web service can then be published to the Azure Marketplace and consumed by users and devices across the world with no infrastructure setup by the author of the web service.

Consumption of web service

The Normal Distribution Suite includes the following 3 services.

Normal Distribution Quantile Calculator

This service accepts 4 arguments of a normal distribution and calculates the associated quantile.

The input arguments are:

- p - A single probability of an event with normal distribution.
- Mean - The normal distribution mean.
- SD - The normal distribution standard deviation.
- Side - L for the lower side of the distribution and U for the upper side of the distribution.

The output of the service is the calculated quantile that is associated with the given probability.

Normal Distribution Probability Calculator

This service accepts 4 arguments of a normal distribution and calculates the associated probability.

The input arguments are:

- q - A single quantile of an event with normal distribution.
- Mean - The normal distribution mean.
- SD - The normal distribution standard deviation.
- Side - L for the lower side of the distribution and U for the upper side of the distribution.

The output of the service is the calculated probability that is associated with the given quantile.

Normal Distribution Generator

This service accepts 3 arguments of a normal distribution and generates a random sequence of numbers that are normally distributed. The following arguments should be provided to it within the request:

- n - The number of observations.
- mean - The normal distribution mean.
- sd - The normal distribution standard deviation.

The output of the service is a sequence of length n with a normal distribution based on the mean and sd arguments.

This service, as hosted on the Azure Marketplace, is an OData service; these may be called through POST or GET methods.

There are multiple ways of consuming the service in an automated fashion (example apps are here: [Generator](#), [Probability Calculator](#), [Quantile Calculator](#)).

Starting C# code for web service consumption:

Normal Distribution Quantile Calculator

```
public class Input
{
    public string p;
    public string mean;
    public string sd;
    public string side;
}

public AuthenticationHeaderValue CreateBasicHeader(string username, string password)
{
    byte[] byteArray = System.Text.Encoding.UTF8.GetBytes(username + ":" + password);
    return new AuthenticationHeaderValue("Basic", Convert.ToBase64String(byteArray));
}

void Main()
{
    var input = new Input() { p = TextBox1.Text, mean = TextBox2.Text, sd = TextBox3.Text, side = TextBox4.Text };
    var json = JsonConvert.SerializeObject(input);
    var acitionUri = "PutAPIURLHere,e.g.https://api.datamarket.azure.com/..../v1/Score";
    var httpClient = new HttpClient();

    httpClient.DefaultRequestHeaders.Authorization = CreateBasicHeader("PutEmailAddressHere", "ChangeToAPIKey");
    httpClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

    var response = httpClient.PostAsync(acitionUri, new StringContent(json));
    var result = response.Result.Content;
    var scoreResult = result.ReadAsStringAsync().Result;
}
```

Normal Distribution Probability Calculator

```

public class Input
{
    public string q;
    public string mean;
    public string sd;
    public string side;
}

public AuthenticationHeaderValue CreateBasicHeader(string username, string password)
{
    byte[] byteArray = System.Text.Encoding.UTF8.GetBytes(username + ":" + password);
    return new AuthenticationHeaderValue("Basic", Convert.ToBase64String(byteArray));
}

void Main()
{
    var input = new Input() { q = TextBox1.Text, mean = TextBox2.Text, sd = TextBox3.Text, side = TextBox4.Text };
    var json = JsonConvert.SerializeObject(input);
    var acitionUri = "PutAPIURLHere,e.g.https://api.datamarket.azure.com/....v1/Score";
    var httpClient = new HttpClient();

    httpClient.DefaultRequestHeaders.Authorization = CreateBasicHeader("PutEmailAddressHere", "ChangeToAPIKey");
    httpClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

    var response = httpClient.PostAsync(acitionUri, new StringContent(json));
    var result = response.Result.Content;
    var scoreResult = result.ReadAsStringAsync().Result;
}

```

Normal Distribution Generator

```

public class Input
{
    public string n;
    public string mean;
    public string sd;
}

public AuthenticationHeaderValue CreateBasicHeader(string username, string password)
{
    byte[] byteArray = System.Text.Encoding.UTF8.GetBytes(username + ":" + password);
    return new AuthenticationHeaderValue("Basic", Convert.ToBase64String(byteArray));
}

void Main()
{
    var input = new Input() { n = TextBox1.Text, mean = TextBox2.Text, sd = TextBox3.Text };
    var json = JsonConvert.SerializeObject(input);
    var acitionUri = "PutAPIURLHere,e.g.https://api.datamarket.azure.com/....v1/Score";
    var httpClient = new HttpClient();

    httpClient.DefaultRequestHeaders.Authorization = CreateBasicHeader("PutEmailAddressHere", "ChangeToAPIKey");
    httpClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

    var response = httpClient.PostAsync(acitionUri, new StringContent(json));
    var result = response.Result.Content;
    var scoreResult = result.ReadAsStringAsync().Result;
}

```

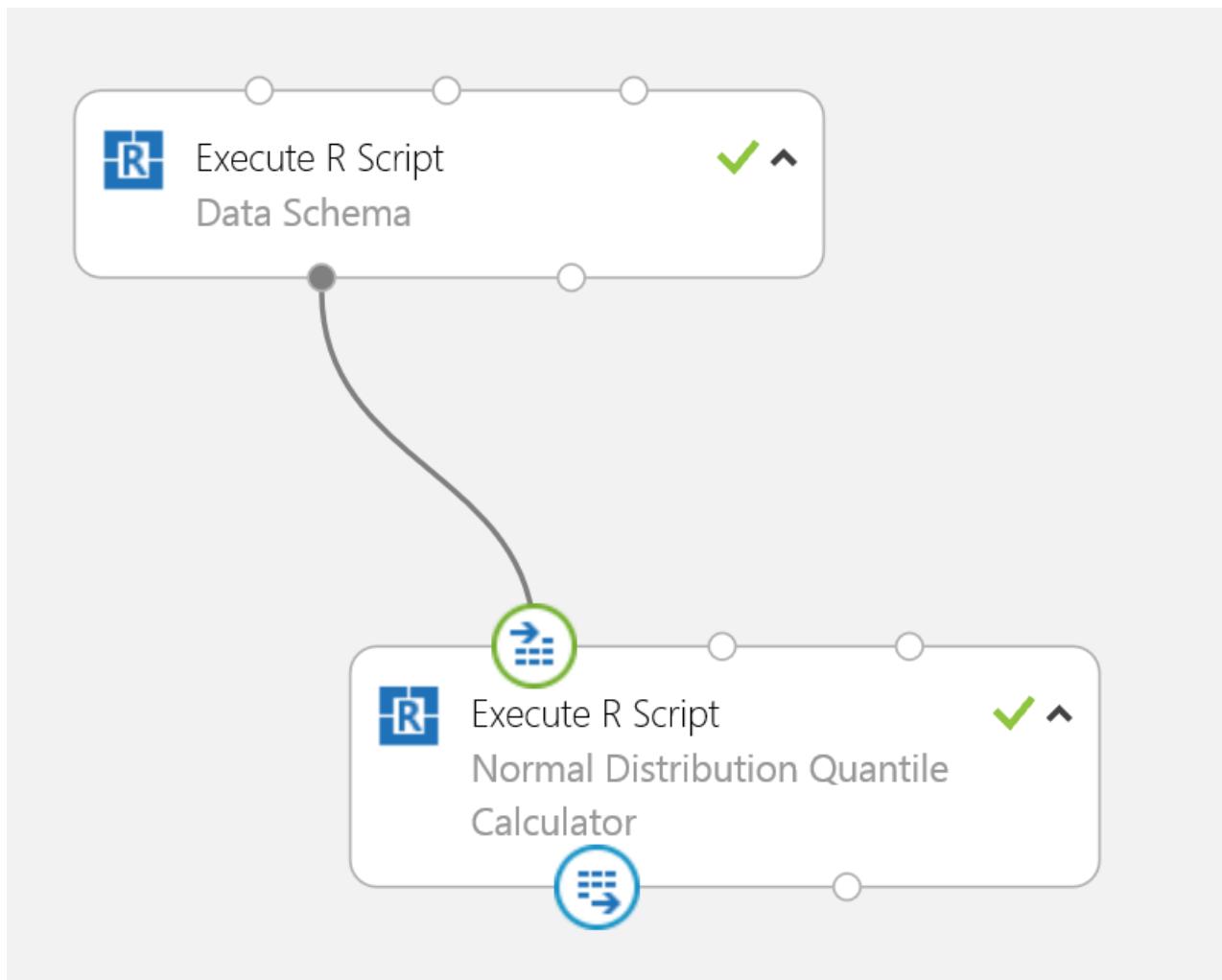
Creation of web service

This web service was created using Azure Machine Learning. For a free trial, as well as introductory videos on creating experiments and [publishing web services](#), please see [azure.com/ml](#).

Below is a screenshot of the experiment that created the web service and example code for each of the modules within the experiment.

Normal Distribution Quantile Calculator

Experiment flow:



```

#Data schema with example data (replaced with data from web service)
data.set=data.frame(p=0.1,mean=0,sd=1,side='L');
maml.mapOutputPort("data.set"); #send data to output port

# Map 1-based optional input ports to variables
dataset1<- maml.mapInputPort(1) # class: data.frame

param=dataset1
if(param$p < 0) {
  print('Bad input: p must be between 0 and 1')
  param$p = 0
} else if(param$p > 1) {
  print('Bad input: p must be between 0 and 1')
  param$p = 1
}
q = qnorm(param$p,mean=param$mean,sd=param$sd)

if(param$side == 'U'){
  q = 2* param$mean - q
} else if (param$side == 'L') {
  q = q
} else {
  print("Invalid side choice")
}

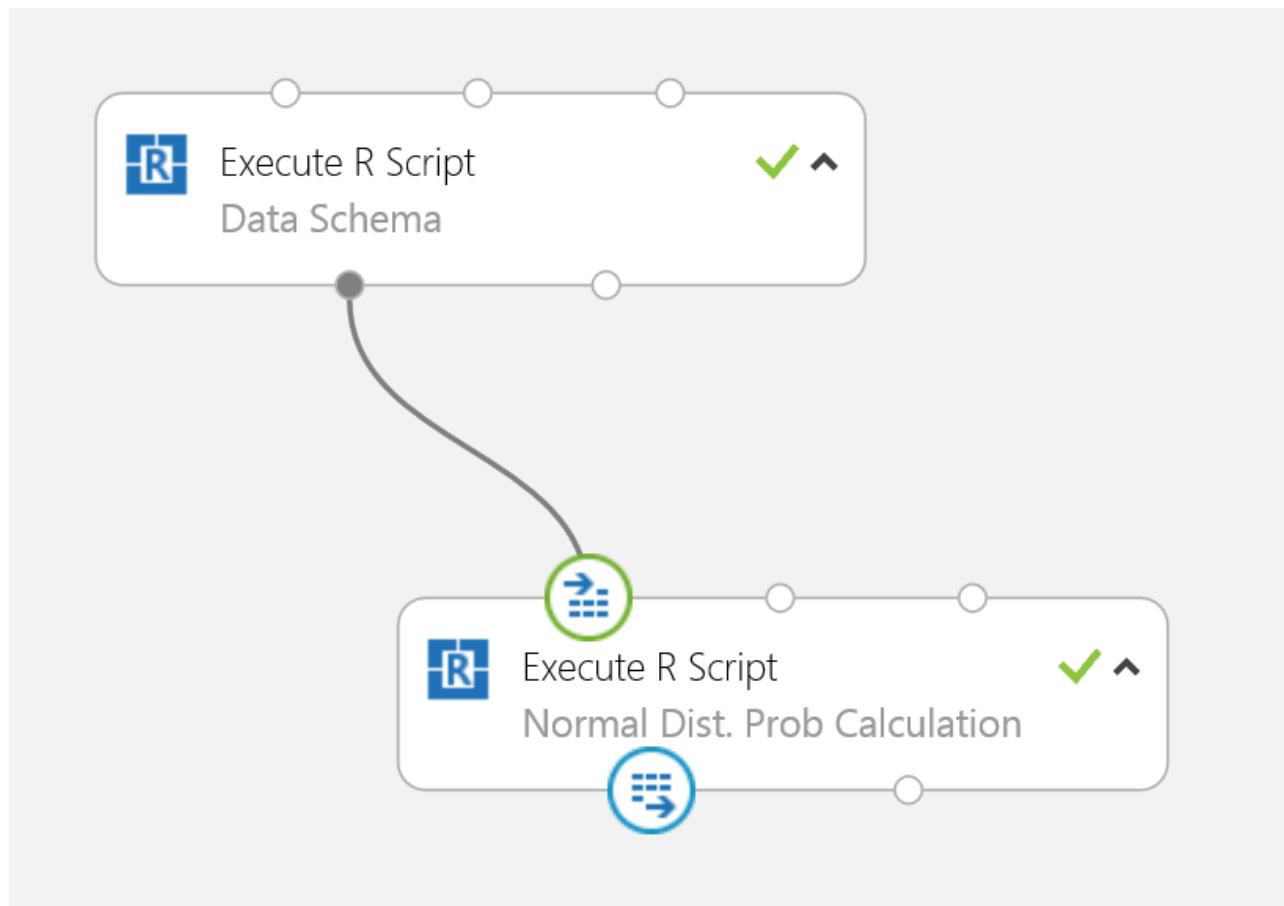
output = as.data.frame(q)

# Select data.frame to be sent to the output Dataset port
maml.mapOutputPort("output");

```

Normal Distribution Probability Calculator

Experiment flow:



```

#Data schema with example data (replaced with data from web service)
data.set=data.frame(q=-1,mean=0,sd=1,side='L');
maml.mapOutputPort("data.set"); #send data to output port

# Map 1-based optional input ports to variables
dataset1 <- maml.mapInputPort(1) # class: data.frame

param=dataset1
prob = pnorm(param$q,mean=param$mean,sd=param$sd)

if(param$side == 'U'){
prob = 1 - prob
} else if (param$side == 'B') {
prob = ifelse(prob<=0.5,prob * 2, 1)
} else if (param$side == 'L') {
prob = prob
} else {
print("Invalid side choice")
}

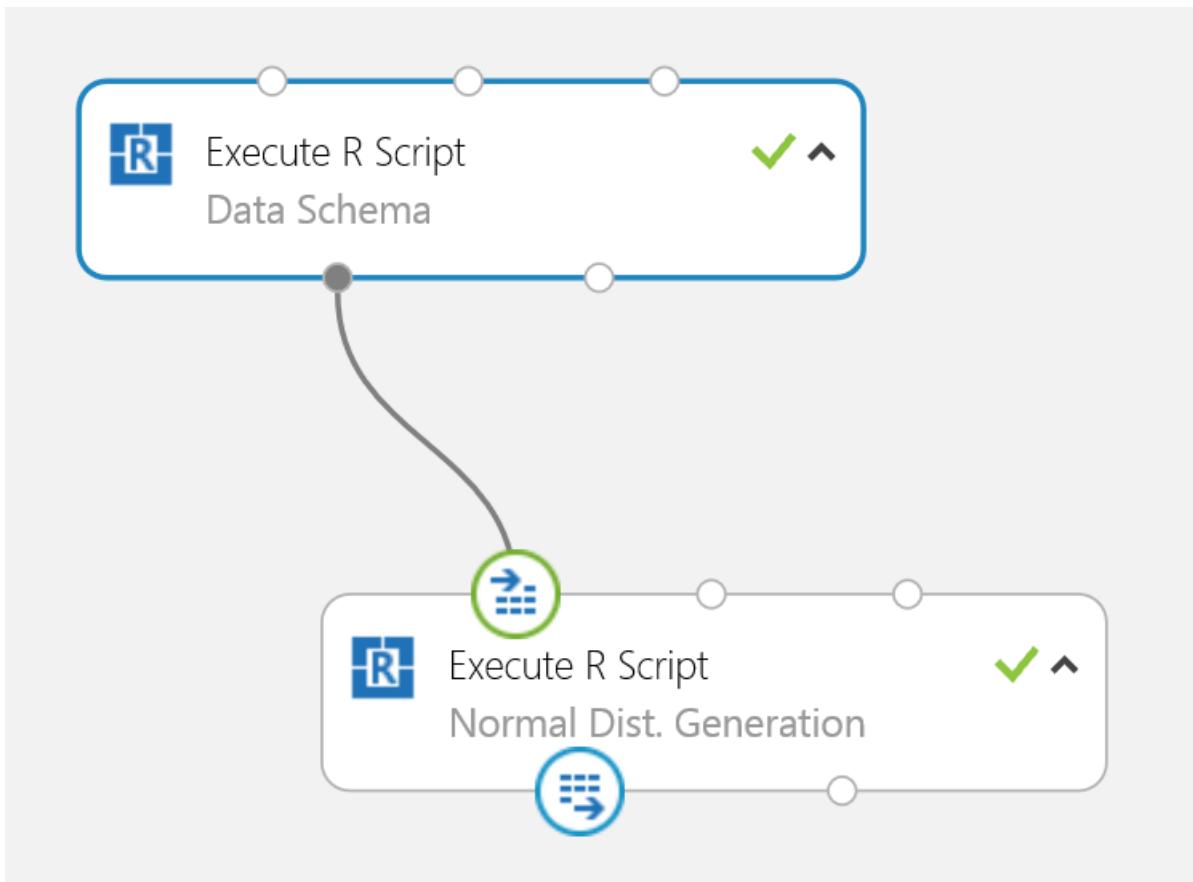
output = as.data.frame(prob)

# Select data.frame to be sent to the output Dataset port
maml.mapOutputPort("output");

```

Normal Distribution Generator

Experiment flow:



```
#Data schema with example data (replaced with data from web service)
data.set=data.frame(n=50,mean=0,sd=1);
maml.mapOutputPort("data.set"); #send data to output port

# Map 1-based optional input ports to variables
dataset1<-maml.mapInputPort(1) # class: data.frame

param=dataset1
dist=rnorm(param$n,mean=param$mean,sd=param$sd)

output=as.data.frame(t(dist))

# Select data.frame to be sent to the output Dataset port
maml.mapOutputPort("output");
```

Limitations

These are very simple examples surrounding the normal distribution. As can be seen from the example code above, little error catching is implemented.

FAQ

For frequently asked questions on consumption of the web service or publishing to the Azure Marketplace, see [here](#).

(deprecated) Binomial Distribution Suite

1/17/2017 • 6 min to read • [Edit on GitHub](#)

NOTE

The Microsoft DataMarket is being retired and this API has been deprecated.

You can find many useful example experiments and APIs in the [Cortana Intelligence Gallery](#). For more information about the Gallery, see [Share and discover resources in the Cortana Intelligence Gallery](#).

The Binomial Distribution Suite is a set of sample web services ([Binomial Generator](#), [Probability Calculator](#), [Quantile Calculator](#)) that help in generating and dealing with binomial distributions. The services allow generating a binomial distribution sequence of any length, calculating quantiles out of given probability and calculating probability from a given quantile. Each of the services emits different outputs based on the selected service (see description below). The Binomial Distribution Suite is based on the R functions `qbinom`, `rbinom`, and `pbinom`, which are included in the R stats package.

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

These web services could be consumed by users – potentially directly on the marketplace, through a mobile app, through a website, or even on a local computer, for example. But the purpose of the web service is also to serve as an example of how Azure Machine Learning can be used to create web services on top of R code. With just a few lines of R code and clicks of a button within Azure Machine Learning Studio, an experiment can be created with R code and published as a web service. The web service can then be published to the Azure Marketplace and consumed by users and devices across the world – no infrastructure setup by the author of the web service is required.

Consumption of web service

The Binomial Distribution Suite includes the following 3 services.

Binomial Distribution Quantile Calculator

This service accepts 4 arguments of a normal distribution and calculates the associated quantile. The input arguments are:

- p - A single aggregated probability of multiple trials.
- size - The number of trials.
- prob - The probability of success in a trial.
- Side - L for the lower side of the distribution, U for the upper side of the distribution.

The output of the service is the calculated quantile that is associated with the given probability.

Binomial Distribution Probability Calculator

This service accepts 4 arguments of a binomial distribution and calculates the associated probability. The input arguments are:

- q - A single quantile of an event with binomial distribution.
- size - The number of trials.
- prob - The probability of success in a trial.
- side - L for the lower side of the distribution, U for the upper side of the distribution, or E that is equal to a single number of successes.

The output of the service is the calculated probability that is associated with the given quantile.

Binomial Distribution Generator

This service accepts 3 arguments of a binomial distribution and generates a random sequence of numbers that are binomially distributed. The following arguments should be provided to it within the request:

- n - Number of observations.
- size - Number of trials.
- prob - Probability of success.

The output of the service is a sequence of length n with a binomial distribution based on the size and prob arguments.

This service, as hosted on the Azure Marketplace, is an OData service; these may be called through POST or GET methods.

There are multiple ways of consuming the service in an automated fashion (example apps are here: [Generator](#), [Probability Calculator](#), [Quantile Calculator](#)).

Starting C# code for web service consumption:

Binomial Distribution Quantile Calculator

```
public class Input
{
    public string p;
    public string size;
    public string prob;
    public string side;
}

public AuthenticationHeaderValue CreateBasicHeader(string username, string password)
{
    byte[] byteArray = System.Text.Encoding.UTF8.GetBytes(username + ":" + password);
    return new AuthenticationHeaderValue("Basic", Convert.ToBase64String(byteArray));
}

void main()
{
    var input = new Input() { p = TextBox1.Text, size = TextBox2.Text, prob = TextBox3.Text, side = TextBox4.Text };
    var json = JsonConvert.SerializeObject(input);
    var acitionUri = "PutAPIURLHere,e.g.https://api.datamarket.azure.com/..../v1/Score";
    var httpClient = new HttpClient();

    httpClient.DefaultRequestHeaders.Authorization = CreateBasicHeader("PutEmailAddressHere", "ChangeToAPIKey");
    httpClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

    var response = httpClient.PostAsync(acitionUri, new StringContent(json));
    var result = response.Result.Content;
    var scoreResult = result.ReadAsStringAsync().Result;
}
```

Binomial Distribution Probability Calculator

```

public class Input
{
    public string q;
    public string size;
    public string prob;
    public string side;
}

public AuthenticationHeaderValue CreateBasicHeader(string username, string password)
{
    byte[] byteArray = System.Text.Encoding.UTF8.GetBytes(username + ":" + password);
    return new AuthenticationHeaderValue("Basic", Convert.ToBase64String(byteArray));
}

void Main()
{
    var input = new Input() { q = TextBox1.Text, size = TextBox2.Text, prob = TextBox3.Text, side = TextBox4.Text };
    var json = JsonConvert.SerializeObject(input);
    var acitionUri = " PutAPIURLHere,e.g.https://api.datamarket.azure.com/..../v1/Score";
    var httpClient = new HttpClient();

    httpClient.DefaultRequestHeaders.Authorization = CreateBasicHeader("PutEmailAddressHere", "ChangeToAPIKey");
    httpClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

    var response = httpClient.PostAsync(acitionUri, new StringContent(json));
    var result = response.Result.Content;
    var scoreResult = result.ReadAsStringAsync().Result;
}

```

Binomial Distribution Generator

```

public class Input
{
    public string n;
    public string size;
    public string p;
}

public AuthenticationHeaderValue CreateBasicHeader(string username, string password)
{
    byte[] byteArray = System.Text.Encoding.UTF8.GetBytes(username + ":" + password);
    return new AuthenticationHeaderValue("Basic", Convert.ToBase64String(byteArray));
}

void Main()
{
    var input = new Input() { n = TextBox1.Text, size = TextBox2.Text, p = TextBox3.Text };
    var json = JsonConvert.SerializeObject(input);
    var acitionUri = "PutAPIURLHere,e.g.https://api.datamarket.azure.com/..../v1/Score";
    var httpClient = new HttpClient();

    httpClient.DefaultRequestHeaders.Authorization = CreateBasicHeader("PutEmailAddressHere", "ChangeToAPIKey");
    httpClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

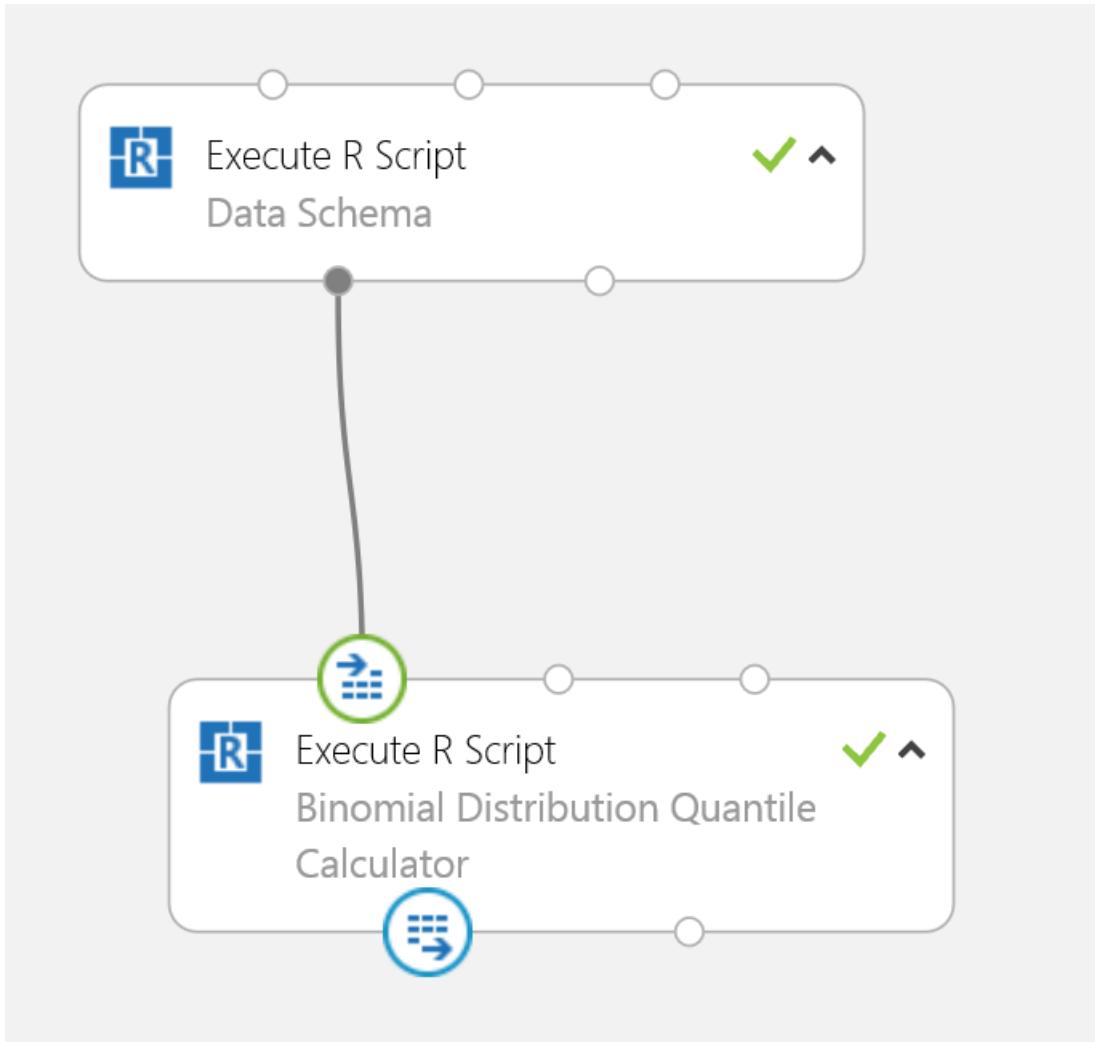
    var response = httpClient.PostAsync(acitionUri, new StringContent(json));
    var result = response.Result.Content;
    var scoreResult = result.ReadAsStringAsync().Result;
}

```

Creation of web service

This web service was created using Azure Machine Learning. For a free trial, as well as introductory videos on creating experiments and [publishing web services](#), please see [azure.com/ml](#). Below is a screenshot of the experiment that created the web service and example code for each of the modules within the experiment.

Binomial Distribution Quantile Calculator



Module 1:

```
#data schema with example data (replaced with data from web service)
data.set=data.frame(p=0.1,size=10,prob=.5,side='L');
maml.mapOutputPort("data.set"); #send data to output port
```

Module 2:

```

dataset1 <- maml.mapInputPort(1) # class: data.frame
param = dataset1
if(param$p < 0) {
  print('Bad input: p must be between 0 and 1')
  param$p = 0
} else if (param$p > 1) {
  print('Bad input: p must be between 0 and 1')
  param$p = 1
}

if(param$prob < 0) {
  print('Bad input: prob must be between 0 and 1')
  param$prob = 0
} else if (param$prob > 1) {
  print('Bad input: prob must be between 0 and 1')
  param$prob = 1
}

quantile = qbinom(param$p, size=param$size, prob=param$prob)
df = data.frame(x=1:param$size, prob=dbinom(1:param$size, param$size, prob=param$prob))
quantile

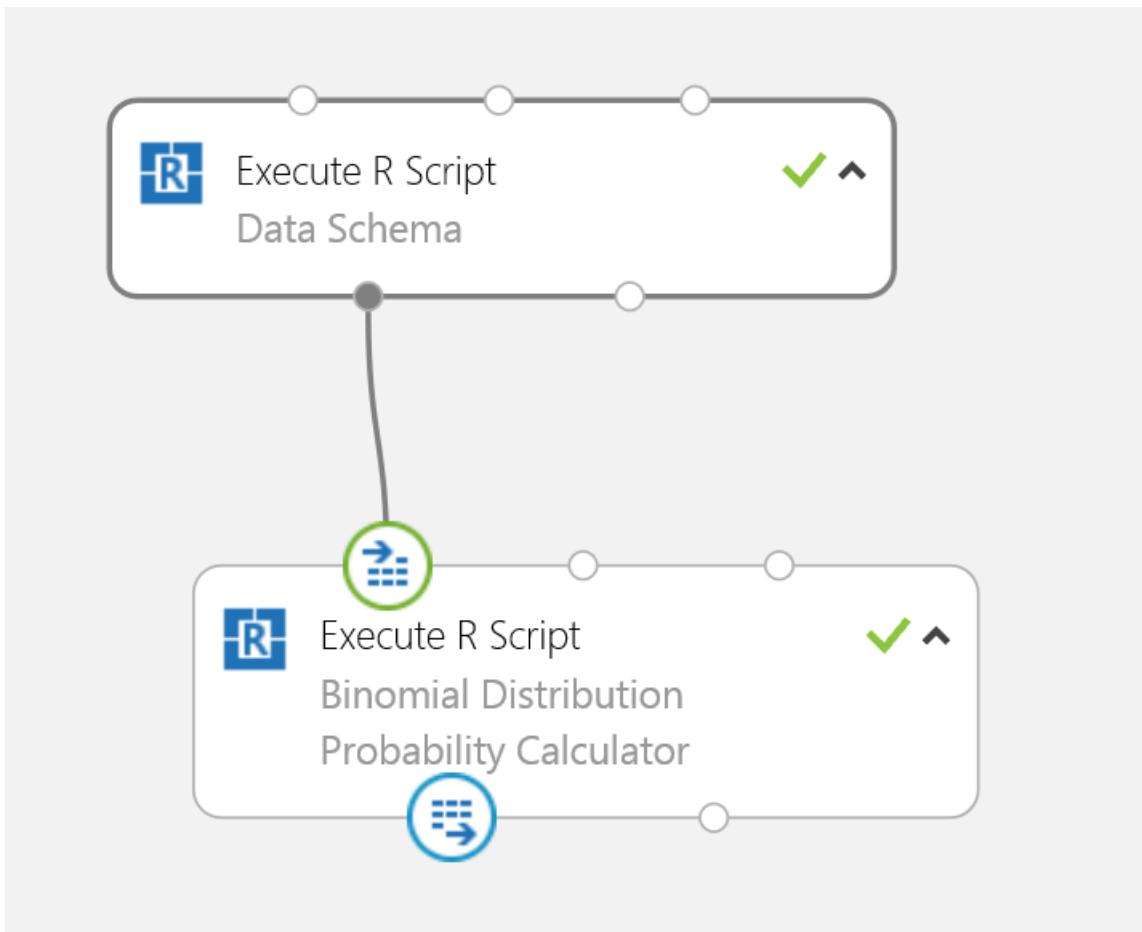
if(param$side == 'U'){
  quantile = qbinom(param$p, size=param$size, prob=param$prob, lower.tail = F)
  band=subset(df,x>quantile)
} else if (param$side == 'L') {
  quantile = qbinom(param$p, size=param$size, prob=param$prob, lower.tail = T)
  band=subset(df,x<=quantile)
} else {
  print("Invalid side choice")
}

output = as.data.frame(quantile)

# Select data.frame to be sent to the output Dataset port
maml.mapOutputPort("output");

```

Binomial Distribution Probability Calculator



Module 1:

```
#data schema with example data (replaced with data from web service)
data.set=data.frame(q=5,size=10,prob=.5,side='L');
maml.mapOutputPort("data.set"); #send data to output port
```

Module 2:

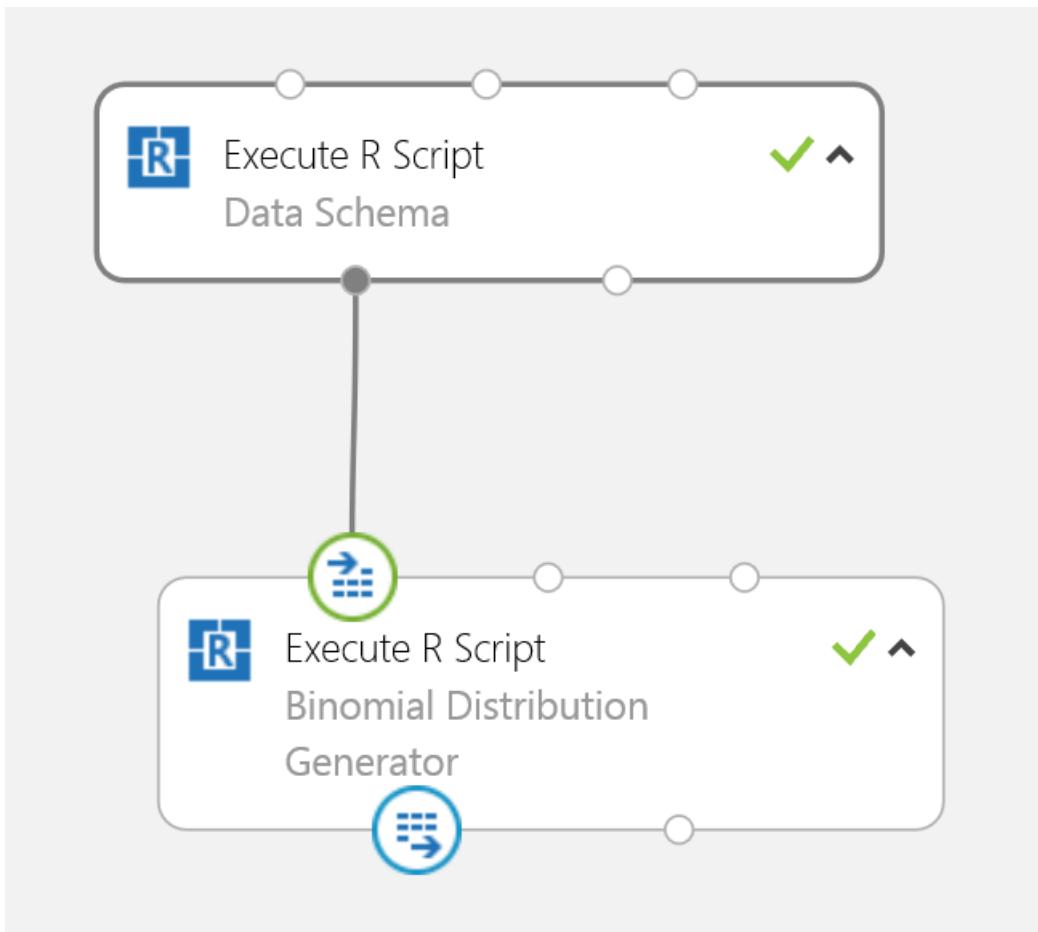
```
dataset1 <- maml.mapInputPort(1) # class: data.frame
param = dataset1
prob = pbinom(param$q,size=param$size,prob=param$prob)
prob.eq = dbinom(param$q,size=param$size,prob=param$prob)
df = data.frame(x=1:param$size, prob=dbinom(1:param$size, param$size, prob=param$prob))
prob

if (param$side == 'U'){
  prob = 1 - prob
  band=subset(df,x>param$q)
} else if (param$side == 'E') {
  prob = prob.eq
  band=subset(df,x==param$q)
} else if (param$side == 'L') {
  prob = prob
  band=subset(df,x<=param$q)
} else {
  print("Invalid side choice")
}

output = as.data.frame(prob)

# Select data.frame to be sent to the output Dataset port
maml.mapOutputPort("output");
```

Binomial Distribution Generator



Module 1:

```
#data schema with example data (replaced with data from web service)
data.set=data.frame(n=50,size=10,p=.5);
maml.mapOutputPort("data.set"); #send data to output port
```

Module 2:

```
dataset1 <- maml.mapInputPort(1) # class: data.frame
param=dataset1
dist = rbinom(param$n,param$size,param$p)

output = as.data.frame(t(dist))

# Select data.frame to be sent to the output Dataset port
maml.mapOutputPort("output");
```

Limitations

These are very simple examples surrounding the binomial distribution. As can be seen from the example code above, little error catching is implemented.

FAQ

For frequently asked questions on consumption of the web service or publishing to the Azure Marketplace, see [here](#).

(deprecated) Publishing and using Machine Learning apps in the Azure Marketplace: FAQ

1/17/2017 • 3 min to read • [Edit on GitHub](#)

NOTE

DataMarket and Data Services are being retired, and existing subscriptions will be retired and cancelled starting 3/31/2017. As a result, this article is being deprecated.

As an alternative, you can publish your Machine Learning experiments to the [Cortana Intelligence Gallery](#) for the benefit of the data science community. For more information, see [Share and discover resources in the Cortana Intelligence Gallery](#).

Questions about consuming from Marketplace

1. Why do I get the following error message after I enter input for the web service:

The request resulted in a back-end time out or back-end error. The team is investigating the issue. We are sorry for the inconvenience. (500)

Your input parameter(s) may not conform to the required format for the specific web service. Please refer to the corresponding documentation link to find the correct format for input parameters and the limitations of this web service.

NOTE

[Try Azure Machine Learning for free](#)

No credit card or Azure subscription needed. [Get started now >](#)

2. If I copy the API link for the web service that I see on the "Explore this dataset" page and paste it into another browser window, what credentials should I use to access the results, and how do I see them?

You should use your Marketplace account as the username and the primary account key as the password. The primary account key can be found on the **Explore this dataset** page under the description of the web service (click the **show** button). The result may display in the browser or it may be available to download, depending on which browser you are using.

3. Why do I get the following error message after I enter the input for the web service on the "Explore this dataset" page:

An unexpected error occurred while processing your request. Please try again.

One or more input parameters of your web service may have exceeded the length limit when consuming the web service on the marketplace **Explore this dataset** page. The services can be called with a longer input length by using HTTP POST methods. For examples, see [Sample solutions using R on Machine Learning and published to Marketplace](#).

4. Why do I not see anything in the "API EXPLORER" tab int the Store in the Azure Classic Portal?

This is a known issue with the Azure Classic Portal Marketplace. The team is working to resolve this issue.

Questions about publishing from Azure Machine Learning on Marketplace

1. Why are my transactions of logos or images not refreshing for my web service?

Logos and images are cached in the publishing portal, and it may take up to 10 days for the new logo or image to update on the portal.

2. Why is the "Detail" tab of my web service on Marketplace showing an error message?

There is a known Marketplace issue when connecting to Azure Machine Learning for service details. The team is working to resolve this issue.

3. Why does the R sample code in the Azure Machine Learning web services not work for consuming the web services in Marketplace?

The authentication systems are different when connecting to Azure Machine Learning web services directly compared to connecting to these web services through the Marketplace. The services in Marketplace are OData services, and they can be called with GET or POST methods.

4. Why are the support links of my web service offers not updating correctly for some of my offers?

The support links are global per publisher, not per offer.

5. How do I publish a web service with batch input mode in Marketplace?

The batch input mode is currently not supported in Marketplace web services.

6. Who should I contact to get help if I have questions about becoming a data publisher, or if I have issues during publishing?

Please contact the Azure Marketplace team at datamarketbd@microsoft.com for more information.

PowerShell module for Microsoft Azure Machine Learning

1/17/2017 • 2 min to read • [Edit on GitHub](#)

The PowerShell module for Azure Machine Learning is a powerful tool that allows you to use Windows PowerShell to manage workspaces, experiments, datasets, web services, and more.

You can view the documentation and download the module, along with the full source code, at <https://aka.ms/amlps>.

NOTE

The Azure Machine Learning PowerShell module is currently in preview mode. The module will continue to be improved and expanded during this preview period. Keep an eye on the [Cortana Intelligence and Machine Learning Blog](#) for news and information.

What is the Machine Learning PowerShell module?

The Machine Learning PowerShell module is a .NET-based DLL module that allows you to fully manage Azure Machine Learning workspaces, experiments, datasets, web services, and web service endpoints from Windows PowerShell. Along with the module, you can download the full source code which includes a cleanly-separated [C# API layer](#). This means you can reference this DLL from your own .NET project and manage Azure Machine Learning through .NET code. In addition, the DLL depends on underlying REST APIs that you can leverage directly from your favorite client.

What can I do with the PowerShell module?

Here are some of the tasks you can perform with this PowerShell module. Check out the [full documentation](#) for these and many more functions.

- Provision a new workspace using a management certificate ([New-AmlWorkspace](#))
- Export and import a JSON file representing an experiment graph ([Export-AmlExperimentGraph](#) and [Import-AmlExperimentGraph](#))
- Run an experiment ([Start-AmlExperiment](#))
- Create a web service out of a predictive experiment ([New-AmlWebService](#))
- Create a new endpoint on a published web service ([Add-AmlWebServiceEndpoint](#))
- Invoke an RRS and/or BES web service endpoint ([Invoke-AmlWebServiceRRSEndpoint](#) and [Invoke-AmlWebServiceBESEndpoint](#))

Here's a quick example of using PowerShell to run an existing experiment:

```
#Find the first Experiment named "xyz"
$exp = (Get-AmlExperiment | where Description -eq 'xyz')[0]
#Run the Experiment
Start-AmlExperiment -ExperimentId $exp.ExperimentId
```

For a more in-depth use case, see this article on using the PowerShell module to automate a very commonly-requested task: [Create many Machine Learning models and web service endpoints from one experiment using PowerShell](#).

How do I get started?

To get started with Machine Learning PowerShell, download the [release package](#) from GitHub and follow the [instructions for installation](#). The instructions explain how to unblock the downloaded/unzipped DLL and then import it into your PowerShell environment. Most of the cmdlets require that you supply the workspace ID, the workspace authorization token, and the Azure region that the workspace is in. The simplest way to provide these is through a default config.json file. The instructions also explain how to configure this file.

And if you want, you can clone the git tree, modify the code, and compile it locally using Visual Studio.

Next steps

You can find the full documentation for the PowerShell module at <https://aka.ms/amlps>.

For an extended example of how to use the module in a real-world scenario, check out the in-depth use case, [Create many Machine Learning models and web service endpoints from one experiment using PowerShell](#).

Share and discover resources in the Cortana Intelligence Gallery

1/17/2017 • 3 min to read • [Edit on GitHub](#)

The [Cortana Intelligence Gallery](#) is a community driven site for discovering and sharing solutions built with the Cortana Intelligence Suite. The Gallery contains a variety of resources that you can use to develop your own analytics solutions.

NOTE

[Try Azure Machine Learning for free](#)

No credit card or Azure subscription needed. [Get started now >](#)

What can I find in the Gallery?

The Cortana Intelligence Gallery contains a variety of resources that you can use to develop your own analytics solutions.

- **Experiments** - The Gallery contains a wide variety of experiments that have been developed in Azure Machine Learning Studio. These range from quick proof-of-concept experiments that demonstrate a specific machine learning technique, to fully-developed solutions for complex machine learning problems.
- **Jupyter Notebooks** - Jupyter Notebooks include code, data visualizations, and documentation in a single, interactive canvas. Notebooks in the Gallery provide tutorials and detailed explanations of advanced machine learning techniques and solutions.
- **Solutions** - Quickly build Cortana Intelligence Solutions from preconfigured solutions, reference architectures, and design patterns. Make them your own with the included instructions or with a featured partner.
- **Tutorials** - A number of tutorials are available to walk you through machine learning technologies and concepts, or to describe advanced methods for solving various machine learning problems.

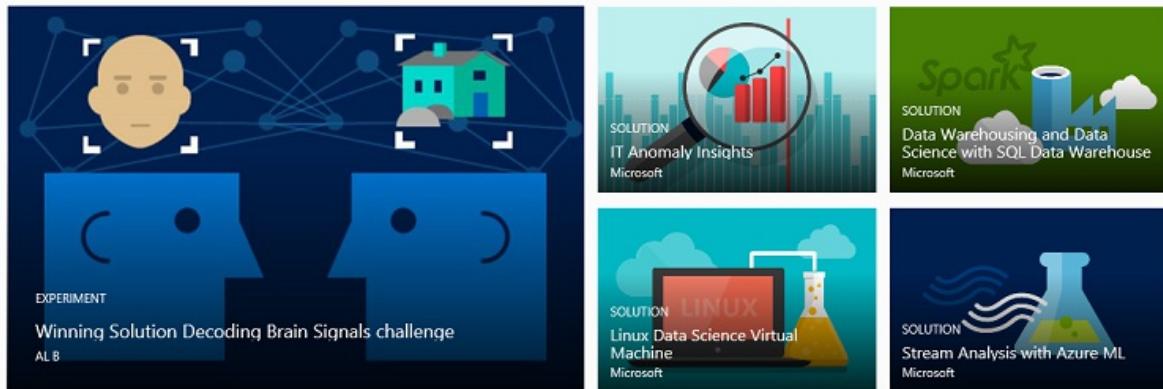
These basic Gallery resources can be grouped together logically in a couple different ways:

- **Collections** - A collection allows you to group together experiments, APIs, and other Gallery items that address a specific solution or concept.
- **Industries** - The Industries section of the Gallery brings together various resources that are specific to such industries as retail, manufacturing, banking, and healthcare.

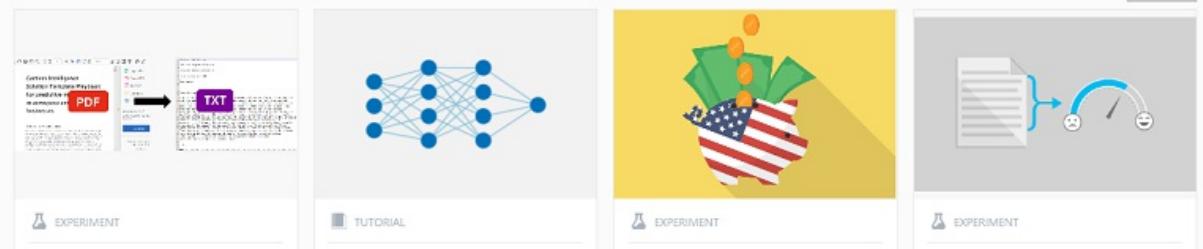
Finally, **Competitions** provide an exciting opportunity to compete with the community of data scientists to solve complex problems using Cortana Intelligence Suite.

[Browse all](#) [Industries](#) [Solutions](#) [Experiments](#) [Machine Learning APIs](#) [Competitions](#) [Notebooks](#) [More](#)

Cortana Intelligence Gallery enables our growing community of developers and data scientists to share their analytics solutions. [Learn how to contribute.](#)



Recently added

[See all](#)

Discover and learn

Anyone can browse and search the different types of resources in the Gallery that have been contributed by Microsoft and the advanced analytics community. Use these to learn more and get a head start on solving your own data analysis problems. You can also download experiments and Jupyter notebooks to your own Machine Learning Studio workspace.

You can easily find recently published and popular resources in the Gallery, or you can search by name, tags, algorithms, and other attributes. Click **Browse all** in the Gallery header, and then select search refinements on the left of the page and enter search terms at the top.

View contributions from a particular author by clicking the author name from within any of the tiles:

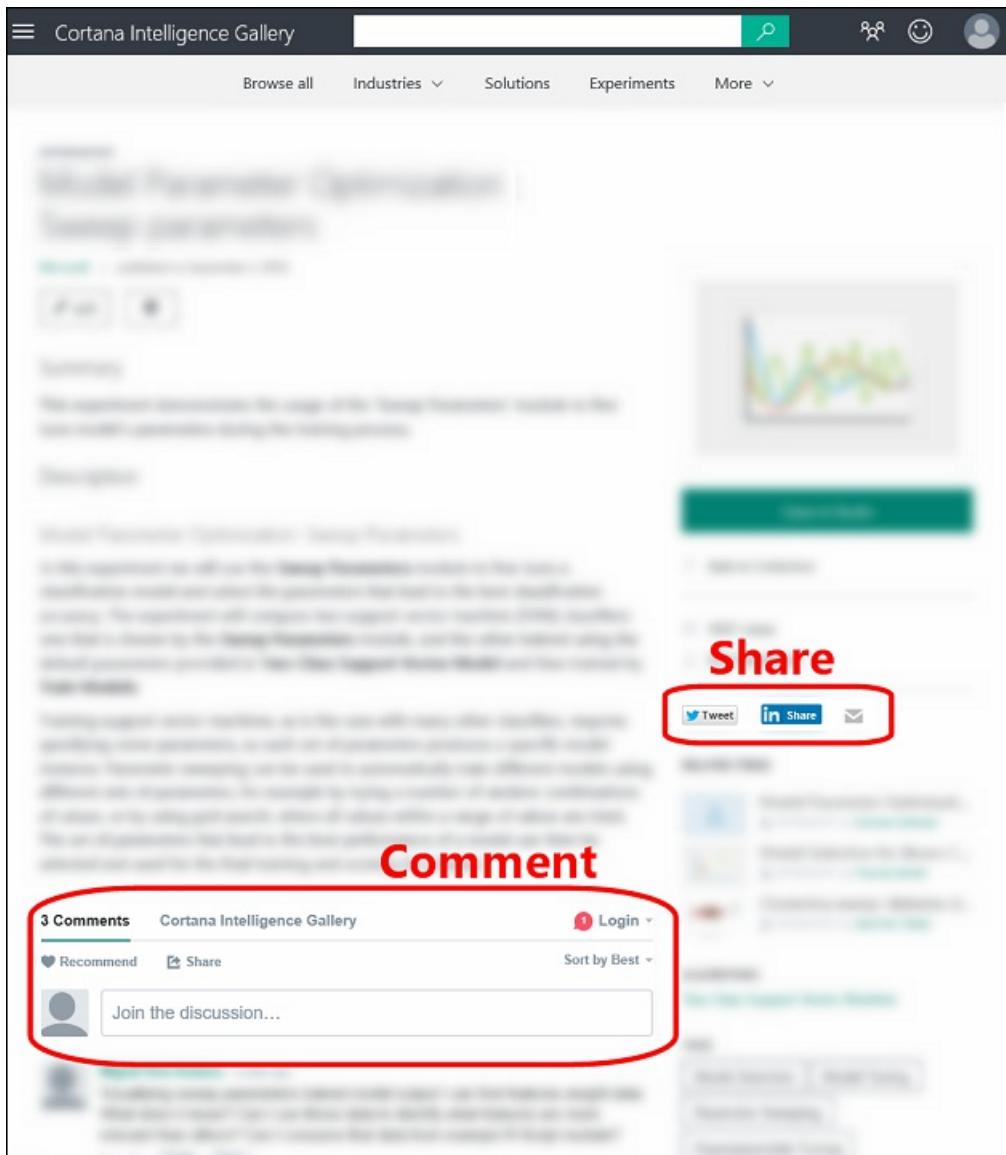


12 ITEMS 4 COLLECTIONS

Sort by: Popular

 EXPERIMENT Predictive Experiment - Mini Twitter sentiment analysis This experiment demonstrates the use of the Execute R Script, Feature Selection, Feature Hashing modules to train a text sentiment classification engine. 2604 17011 months ago	 EXPERIMENT Time Series Forecasting using Custom Modules Time Series Forecasting with Azure ML R custom modules for Arima and ETS 280 116 13 days ago	 EXPERIMENT Training Experiment Mini Twitter sentiment analysis This experiment demonstrates the use of the Execute R Script, Feature Selection, Feature Hashing modules to train a text sentiment classification engine. 674 213 11 months ago	 EXPERIMENT Demand Estimation Using Lag Features This experiment demonstrates demand estimation using regression with UCI bike rental data. 190 91 13 days ago	 EXPERIMENT Training Experiment for Twitter sentiment analysis This experiment demonstrates the use of the Execute R Script, Feature Selection, Feature Hashing modules to train a text sentiment classification engine. 249 104 5 months ago
 EXPERIMENT Compare Datasets This module compares two datasets and summarizes the comparison results. 	 EXPERIMENT Predictive Experiment for Twitter sentiment analysis This experiment demonstrates the use of the Execute R Script, Feature Selection, Feature Hashing modules to train a text sentiment classification engine. 	 EXPERIMENT Using XGBoost to build Graduation Admit Model This experiment shows a sample of using XGBoost to find a probability of graduate admission. 	 EXPERIMENT Retrieve CPI from data.gov This experiment dynamically retrieves XML data from data.gov for CPI and converts XML to JSON. 	 EXPERIMENT Retrieve data.gov datasets This experiment dynamically retrieves XML data from data.gov for any data.gov dataset with a URL.

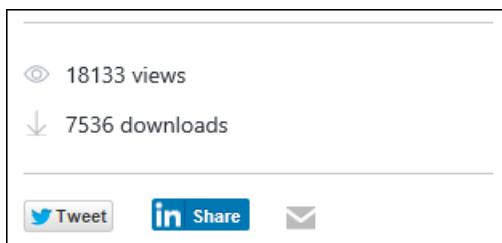
You can comment, provide feedback, or ask questions through the comments section on each resource page. You can even share a resource of interest with friends or colleagues using the share capabilities of LinkedIn or Twitter. You may also email links to these resources to invite other users to view the pages.



Contribute to the Gallery

When you sign in you become a member of the Gallery community. This allows you to contribute your own Gallery items so that others can benefit from the solutions you've discovered.

As others come across your contribution in the Gallery, you can follow the number of views and downloads of your contribution:



Users can also add comments and share your contribution with other members of the data science community. You can log in with a discussion tool such as Disqus and receive notifications for comments on your contributions.

8 Comments

Cortana Intelligence Gallery

1 Login ▾

♥ Recommend 8

Share



Join the discussion...



disqus_lolC7Ftsg · 2 months ago

After setting input output deploying web service gives error: ports selected for input and output are not valid.

^ | v · Reply · Share >



Brandon Rohrer → disqus_lolC7Ftsg · 2 months ago

Hi disqus_lolC7Ftsg,

Without seeing your workspace, it's tough to know exactly what correction to make. I recommend opening up this finalized version of the experiment (<https://gallery.cortanaintelli...>) and comparing yours. This version runs and deploys for me, so I'm guessing there is some subtle difference hiding between yours and it.

Brandon

^ | v · Reply · Share >



Charlie Louland (charlie.louland@microsoft.com) · Brandon Rohrer · 2 months ago

You can contribute the following items to the Gallery - follow these links for more information:

- [Collections](#)
- [Experiments](#)
- [Tutorials](#)

We want to hear from you!

We want the Gallery to be driven by our users and for our users. Use the smiley on the right to tell us what you love or hate about the Gallery.

Sign in

Send a smile Send a frown

What did you like?

<Provide your feedback here>...

969 character(s) left

[TAKE ME TO THE GALLERY >>](#)

Discover industry-specific solutions in the Cortana Intelligence Gallery

1/17/2017 • 1 min to read • [Edit on GitHub](#)

The [Cortana Intelligence Gallery](#) is a community driven site for discovering and sharing solutions built with the Cortana Intelligence Suite. The Gallery contains a variety of resources that you can use to develop your own analytics solutions.

Industry-specific Cortana Intelligence solutions

The [Industries](#) section of the Gallery brings together various resources that are specific to such industries as

- [Retail](#) - Find retail solutions such as sales forecasting, predicting customer churn, and developing pricing models.
- [Manufacturing](#) - Find manufacturing solutions such as anticipating equipment maintenance and forecasting energy prices.
- [Banking](#) - Find banking solutions such as predicting credit risk and monitoring for online fraud.
- [Healthcare](#) - Find healthcare solutions such as detecting disease, and predicting hospital readmissions.

These resources include experiments, custom modules, APIs, collections, and any other Gallery items that can help you develop solutions specific to the industry you're working in.

Discover

To browse through the industry-specific solutions in the Gallery, open the [Gallery](#), point your mouse at **Industries** at the top of the Gallery home page, select a specific industry segment, or select **View All** to see an overview page for all industries.

Each industry page displays a list of the most popular Gallery items associated with that industry. Click **See all** to view all the industry-specific resources in the Gallery. From this page, you can browse all the resources in the Gallery. You also can search by selecting filter criteria on the left of the page and entering search terms at the top.



Click any Gallery item to open the item's details page for more information.

[TAKE ME TO THE GALLERY >>](#)

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

Discover Solutions in the Cortana Intelligence Gallery

1/17/2017 • 1 min to read • [Edit on GitHub](#)

The [Cortana Intelligence Gallery](#) is a community driven site for discovering and sharing solutions built with the Cortana Intelligence Suite. The Gallery contains a variety of resources that you can use to develop your own analytics solutions.

Solutions

A **Solution** in the Gallery provides a jumpstart to quickly build Cortana Intelligence Solutions from preconfigured solutions, reference architectures, and design patterns. Make them your own with the included instructions or with a featured partner.

Discover

To browse for solutions in the Gallery, open the [Gallery](#) and click **Solutions** at the top of the Gallery home page.

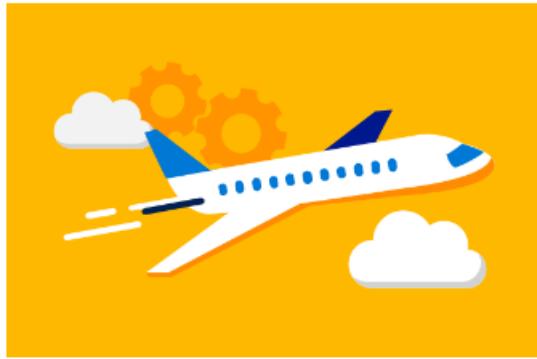
The **Solutions** page displays a list of the most recently added Solutions. Click **See all** to view all Solutions. From this page, you can browse all the Solutions in the Gallery. You also can search by selecting filter criteria on the left of the page and entering search terms at the top.

Click any Solution to open its details page and read more information. On this page you can comment, provide feedback, or ask questions through the comments section. You can even share it with friends or colleagues using the share capabilities of LinkedIn or Twitter. You can also email a link to the Solution to invite other users to view the page.



Deploy

If you want to use a Solution, click **Deploy**. Follow the steps presented to configure and deploy the Solution in your workspace.



Deploy ↗

[TAKE ME TO THE GALLERY >>](#)

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

Discover experiments in the Cortana Intelligence Gallery

1/17/2017 • 8 min to read • [Edit on GitHub](#)

The [Cortana Intelligence Gallery](#) is a community driven site for discovering and sharing solutions built with the Cortana Intelligence Suite. The Gallery contains a variety of resources that you can use to develop your own analytics solutions.

Experiments for Machine Learning Studio

The Gallery contains a wide variety of [Experiments](#) that have been developed in [Azure Machine Learning Studio](#). These range from quick proof-of-concept experiments that demonstrate a specific machine learning technique, to fully-developed solutions for complex machine learning problems.

NOTE

An **experiment** is a canvas in Azure Machine Learning Studio that lets you construct a predictive analysis model by connecting together data with various analytical modules. You can try different ideas, do trial runs, and eventually publish your model as a web service in Azure. For an example of creating a simple experiment, see [Machine learning tutorial: Create your first experiment in Azure Machine Learning Studio](#). For a more complete walkthrough of creating a predictive analytics solution, see [Walkthrough: Develop a predictive analytics solution for credit risk assessment in Azure Machine Learning](#).

Discover

To browse for experiments in the Gallery, open the [Gallery](#) and click **Experiments** at the top of the Gallery home page.

The [Experiments](#) page displays a list of the most recently added and most popular experiments. Click **See all** to view all experiments. From this page, you can browse all the experiments in the Gallery. You also can search by selecting filter criteria on the left of the page and entering search terms at the top.

Click any experiment to open the experiment's details page and read information about what the experiment does. On this page you can comment, provide feedback, or ask questions through the comments section. You can even share it with friends or colleagues using the share capabilities of LinkedIn or Twitter. You can also email a link to the experiment to invite other users to view the page.



Download

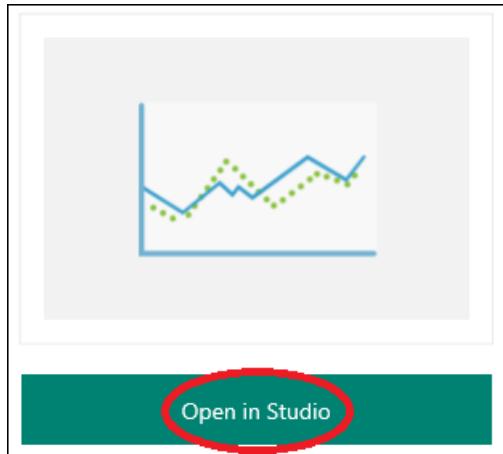
You can download a copy of any experiment from the Gallery into your Studio workspace and then modify your copy to create your own solutions. There are two ways to get a copy of the experiment:

- **From the Gallery** - If you find an experiment you like in the Gallery, you can easily download a copy and open it in your Machine Learning Studio workspace.
- **From within Machine Learning Studio** - In Studio, you can use any experiment in the Gallery as a template to create a new experiment.

From the Gallery

To download a copy of an experiment from the Gallery:

1. Open the experiment's details page in the Gallery
2. Click **Open in Studio**



When you click **Open in Studio**, the experiment is loaded into your Machine Learning Studio workspace and opened (if you're not already signed in to Studio, you will be prompted to sign in using your Microsoft account before the experiment is copied to your workspace).

From within Machine Learning Studio

You can also open the same sample experiments while you're working in Machine Learning Studio:

1. In Machine Learning Studio, click **+NEW**
2. Select **Experiment** - you can choose from a list of Gallery experiments contributed by Microsoft, or you can find a specific experiment using the search box
3. Point your mouse at an experiment and select **Open in Studio** - the experiment is copied to your workspace and opened (to see information about the experiment, select **View in Gallery** which takes you to the details page for the experiment in the Gallery)

You can now customize, iterate, and deploy this experiment like any other experiment you create in Machine Learning Studio.

Contribute

When you sign in to the Gallery you become a member of the Gallery community. This allows you to contribute your own experiments so that others can benefit from the solutions you've discovered.

Publish your experiment to the Gallery

Follow these steps to contribute an experiment to the Cortana Intelligence Gallery:

1. Sign in to Machine Learning Studio using your Microsoft account.
2. Create your experiment and run it.
3. When you're ready to publish your experiment to the Gallery, click **PUBLISH TO GALLERY** below the experiment canvas.



4. Fill out the title and tags fields. Keep them descriptive, highlighting the techniques used or the real-world problem being solved, for instance, "Binary Classification: Twitter Sentiment Analysis".

5. Write a summary of what your content covers. Briefly describe the problem being solved and how you approached it.
6. Use the detailed description box to step through the different parts of your experiment. Some useful topics to include here are:
 - Experiment graph screenshot
 - Data sources and explanation
 - Data processing

- Feature engineering
- Model description
- Results and evaluation of model performance

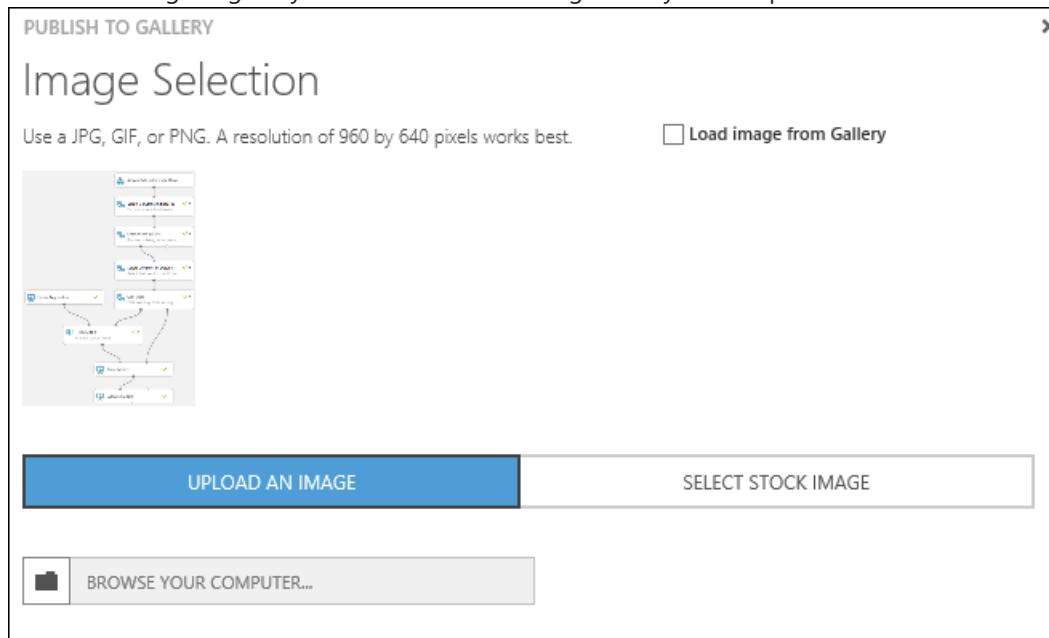
You can use Markdown to format as needed. Click the **Preview** icon to see how things will look when published.



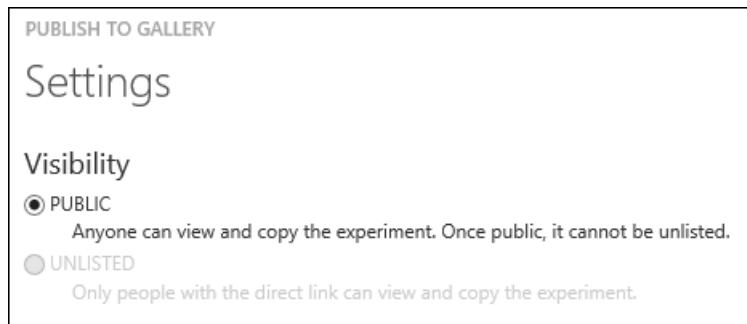
TIP

The box provided for Markdown editing and preview box is quite small. We recommend that you write your documentation in a Markdown editor and paste the completed document into the text box. After you've published your experiment, you can use standard web-based tools in Markdown for editing and preview to make necessary tweaks and corrections.

7. Upload a thumbnail image for your gallery item. This will appear at the top of the item page and in the item tile when browsing the gallery. You can choose an image from your computer or select one of the stock images.



8. Choose whether to publish your content publicly, or have it only accessible to people with the link.



TIP

If you want to make sure your documentation looks right before releasing it publicly, you can publish it as unlisted first, and then switch it to Public from the item page.

9. Click the **OK** checkmark to publish the experiment to the Gallery.

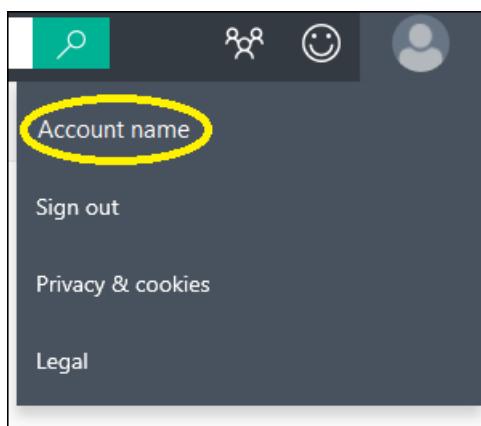


See the section below, **Suggestions for publishing and for quality documentation**, for tips on how to publish a quality Gallery experiment.

That's it – you're all done.

You can now view your experiment in the Gallery and share the link with others. If you published it publicly, your experiment will show up in browse and search results in the Gallery. You can also edit your documentation on the item page any time you're logged in.

To see the list of your contributions, click your image in the upper-right corner of any Gallery page and then click your name to open your account page.



Update your experiment

If you need to make changes to the workflow (modules, parameters, etc.) in an experiment you published to the Gallery, go back to the experiment in Machine Learning Studio, make your changes, and publish it again. Your existing published experiment will be updated with your changes.

If you just need to change any of the following information for your experiment, or you need to delete the experiment from the Gallery, you can make all of these changes in the Gallery:

- Experiment name
- Summary or description text
- Tags specified
- Image used
- Visibility setting (public or unlisted)
- Delete experiment from the Gallery

These changes can be made in the Gallery from the experiment's details page or from your profile page.

From your experiment's details page

From the experiment's details page, click "edit" to change the details for your experiment.

EXPERIMENT

Your first data science experiment - Automobile price prediction

Gary Ericson • September 29, 2016

[Summary](#)

The details page enters edit mode, and you can click "Edit" next to the experiment name, summary, tags, etc., to make changes to them. When you've finished making changes, click "Done".

EXPERIMENT

Your first data science experiment - Automobile price prediction

Gary Ericson • September 29, 2016

[Summary](#) 

This is the experiment created using the tutorial article, "Create your first data science experiment".

Description 

This is the experiment that's created when you follow the steps in the article, [Machine learning tutorial: Create your first data science experiment in Azure Machine Learning Studio](#). It's a simple linear regression model that predicts price from a set of automobile data.

Please see the article for more information on how this experiment was developed.

You can also select the settings icon to change visibility of the experiment (public or unlisted) or you can delete the experiment from the Gallery.

EXPERIMENT

Your first data science experiment - Automobile price prediction

Gary Ericson • September 29, 2016

[Summary](#)

Change Visibility
Delete

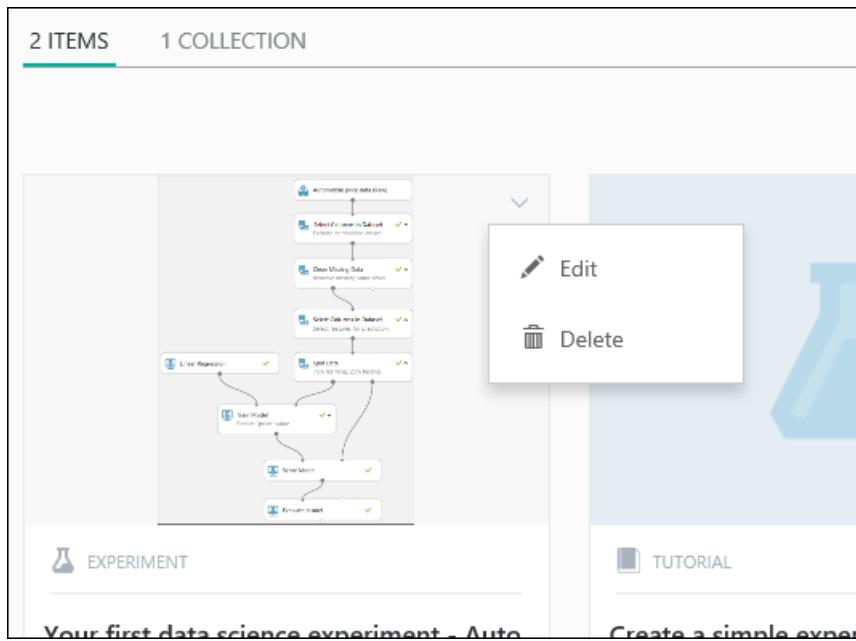
This is the experiment created using the tutorial article, "Create your first

From your profile page

From your profile page, you can click the down arrow on the experiment and select "Edit". This takes you to the

details page for your experiment in edit mode. When you have finished making changes, click "Done".

You can also click "Delete" to delete the experiment from the Gallery.



Suggestions for publishing and for quality documentation

- While you can assume that the reader has prior data science experience, it still helps to simplify your language and explain things in detail wherever possible.
- Not all readers will be familiar with the Cortana Intelligence Suite, given that it's relatively new. So provide enough information and step-by-step explanations to help readers navigate through your work.
- Visuals including experiment graphs or screenshots of data can be very helpful for readers to interpret and use your content the right way. See the [Publishing Guidelines and Examples collection](#) for more information on how to include images in your documentation.
- If you include a dataset in your experiment (it's not being imported through the Import Data module), it's part of your experiment and will get published to the Gallery. Therefore, ensure that the dataset you're publishing has appropriate licensing terms for sharing and downloading by anyone. Gallery contributions are covered under the Azure [Terms of Use](#).

Frequently Asked Questions

What are the image requirements when submitting or editing an image for my experiment?

The images you submit along with your experiment will be used to create an experiment tile for your contribution. It's recommended that the images be < 500Kb in size, with an aspect ratio of 3:2. A resolution of 960x640 is recommended.

What happens to the dataset I used in the experiment? Does the dataset get published to the Gallery as well?

If your dataset is part of your experiment and not being imported through the Import Data module, it's part of your experiment and gets published to the Gallery with your experiment. For this reason make sure that the dataset you're publishing with the experiment has the appropriate licensing terms that allow sharing and downloading by anyone.

I have an experiment that uses an Import Data module to pull data from HDInsight or SQL. It uses my credentials to retrieve the data. How can I publish such an experiment and be assured that my credentials will not be shared?

At this time we do not allow publishing of experiments that use credentials.

How do I enter multiple tags?

After you enter a tag, press the tab key to enter another tag.

[TAKE ME TO THE GALLERY >>](#)

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

Discover Jupyter Notebooks in the Cortana Intelligence Gallery

1/17/2017 • 2 min to read • [Edit on GitHub](#)

The [Cortana Intelligence Gallery](#) is a community driven site for discovering and sharing solutions built with the Cortana Intelligence Suite. The Gallery contains a variety of resources that you can use to develop your own analytics solutions.

Jupyter Notebooks

Jupyter Notebooks include code, data visualizations, and documentation in a single, interactive canvas. Notebooks in the Gallery provide tutorials and detailed explanations of advanced machine learning techniques and solutions.

Discover

To browse for notebooks in the Gallery, open the [Gallery](#) and click **Notebooks** at the top of the Gallery home page.

The [Jupyter Notebooks](#) page displays a list of the most popular notebooks. Click **See all** to view all notebooks. From this page, you can browse all the notebooks in the Gallery. You also can search by selecting filter criteria on the left of the page and entering search terms at the top.

Click any notebook to open the notebook's details page and read more information. On this page you can comment, provide feedback, or ask questions through the comments section. You can even share it with friends or colleagues using the share capabilities of LinkedIn or Twitter. You can also email a link to the notebook to invite other users to view the page.



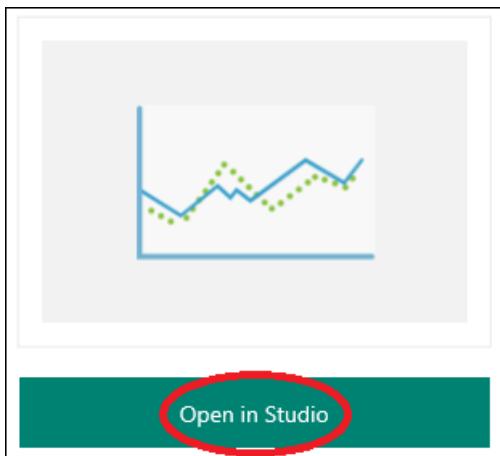
Download

You can download a copy of any notebook from the Gallery into your Machine Learning Studio workspace.

From the Gallery

To download a copy of a notebook from the Gallery:

1. Open the notebook's details page in the Gallery
2. Click **Open in Studio**
3. Select the Region and Workspace you want to use

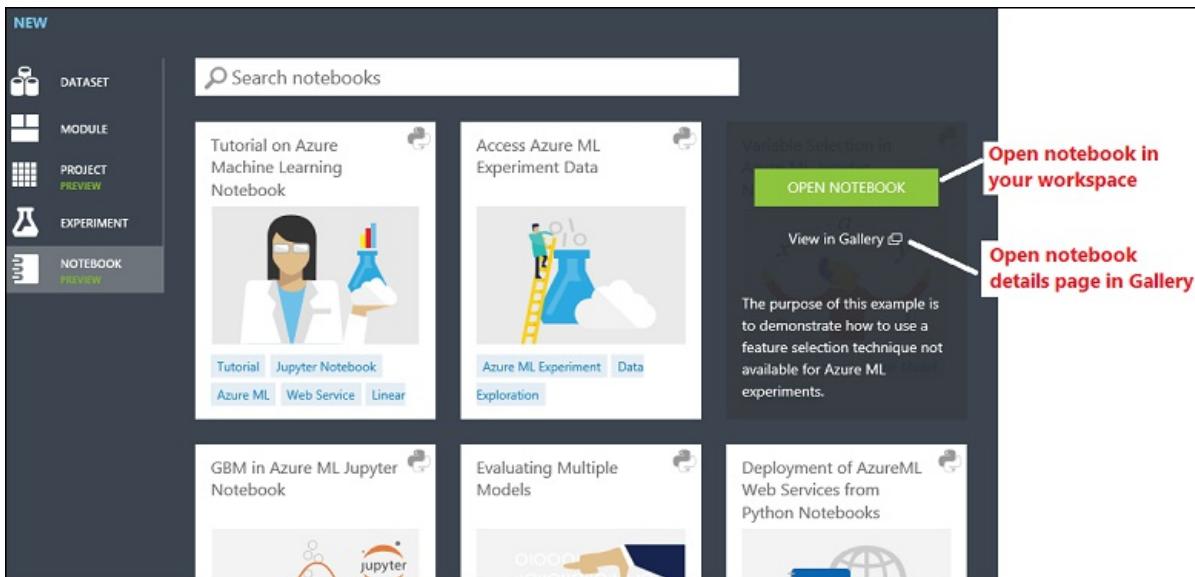


When you click **Open in Studio**, the notebook is loaded into your Machine Learning Studio workspace and opened (if you're not already signed in to Studio, you will be prompted to sign in using your Microsoft account before the notebook is copied to your workspace). You can find the notebook again later by clicking **Notebooks** on the left side of the Studio home page.

In Machine Learning Studio

You can also open any of the same Gallery notebooks while you're working in Machine Learning Studio:

1. In Machine Learning Studio, click **+NEW**
2. Select **Notebook** - a list of notebooks from the Gallery is displayed that you can choose from, or you can find a specific notebook using the search box
3. Point your mouse at a notebook and select **Open Notebook**



A copy of the notebook is downloaded and opened in your workspace in the Jupyter Notebooks section of Machine Learning Studio. The notebook will be listed, along with your other notebooks, on the **Notebooks** page of Studio (on the [Studio home page](#), click **Notebooks** on the left).

[TAKE ME TO THE GALLERY >>](#)

NOTE

[Try Azure Machine Learning for free](#)

No credit card or Azure subscription needed. [Get started now >](#)

Discover competitions in the Cortana Intelligence Gallery

1/17/2017 • 1 min to read • [Edit on GitHub](#)

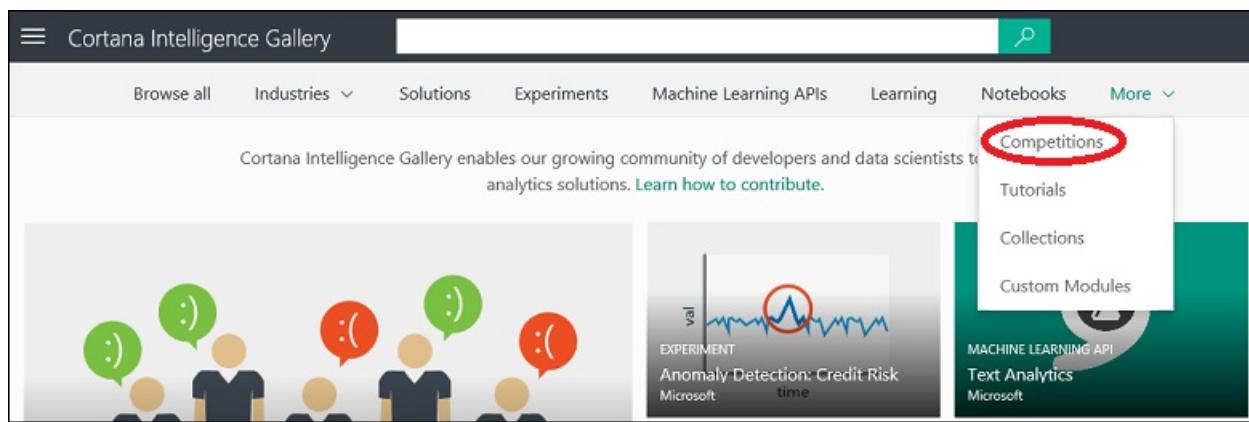
The [Cortana Intelligence Gallery](#) is a community driven site for discovering and sharing solutions built with the Cortana Intelligence Suite. The Gallery contains a variety of resources that you can use to develop your own analytics solutions.

Cortana Intelligence Competitions

Competitions provide an exciting opportunity to compete with the community of data scientists to solve complex problems using Cortana Intelligence Suite.

Discover

To browse for competitions in the Gallery, open the [Gallery](#), point your mouse to **More** at the top of the Gallery home page, and select **Competitions**.



The [Competitions](#) page displays a list of the most popular competitions. Click **See all** to view all competitions. From this page, you can browse all the competitions in the Gallery. You also can search by selecting filter criteria on the left of the page and entering search terms at the top.

Click any competition to open the competition's details page and read more information. On this page you can comment, provide feedback, or ask questions through the comments section. You can even share it with friends or colleagues using the share capabilities of LinkedIn or Twitter. You can also email a link to the competition to invite other users to view the page.

A screenshot of a competition details page. At the top, it shows '26 views' and '30 downloads'. Below that, there are three social sharing buttons: 'Tweet' (Twitter icon), 'Share' (LinkedIn icon), and 'Email' (envelope icon), all enclosed in a red circle. Below the sharing buttons, there's a section for '8 Comments' from 'Cortana Intelligence Gallery'. It includes a 'Login' button, a 'Recommend' button with a count of 8, a 'Share' button, and a 'Sort by Best' dropdown. At the bottom, there's a text input field with placeholder text 'Join the discussion...'. There's also a small user profile icon.

Enter a competition

If the competition is open, the status on the details page will be **Active**. To enter the competition, click **Enter Competition** and you will be given instructions on the resources you need and the steps you need to take.

COMPETITION

Women's Health Risk Assessment

Sponsored by Microsoft • ends 10/5/2016

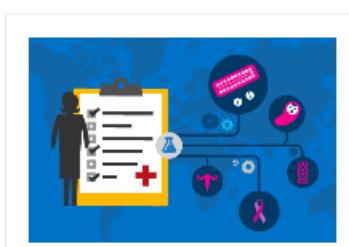
ACTIVE Microsoft Band 2 48 participants

WINNERS

1st	Farhad Ghassemi	2nd	Subhojit Som
	Xbox		worse xbox

Started 9/28/2016 12:32:50 PM (Pacific Daylight Time)
Ended 10/5/2016 12:32:50 PM (Pacific Daylight Time)

Summary



Enter Competition

2258 views
48 participants
322 submissions
[Frequently Asked Questions](#)

If the competition is no longer open, its status on the details page will be **Completed** and the **Enter Competition** link will be replaced with the word **Finished**.

COMPETITION

Xbox Bundle Purchase Prediction

Sponsored by Microsoft • ended 9/20/2015

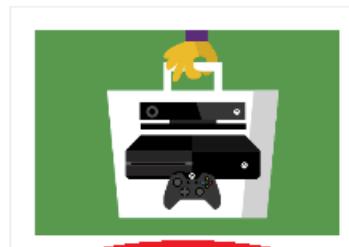
COMPLETED n/a 0 participants

Started 7/20/2015 5:00:00 PM (Pacific Daylight Time)
Ended 9/20/2015 5:00:00 PM (Pacific Daylight Time)

Summary

Predict whether a user will purchase an Xbox bundle, based on past purchase history.

Description



Finished

1213 views
0 participants
0 submissions

[TAKE ME TO THE GALLERY >>](#)

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

Microsoft Cortana Intelligence Competitions FAQ

1/17/2017 • 7 min to read • [Edit on GitHub](#)

What is Cortana Intelligence Competitions?

The Microsoft Cortana Intelligence Competitions allow us to unite a global community of data enthusiasts by collectively solving some of the world's most complex data science problems. Cortana Intelligence Competitions allow data enthusiasts from across the world to compete and build highly accurate and intelligent data science models. Our hosted competitions are based on unique data sets that have been made available publically for the first time. Participants can win rewards or get recognition via our top 10 public leaderboard. Please go [here](#) to access the Competitions home page.

How often will Microsoft release new competitions?

We will be launching 1st-party, Microsoft-owned competitions on a regular basis, approximately every 8-12 weeks.

Where can I ask general questions about data science?

Please use our [Microsoft Azure Machine Learning forum](#).

How do I enter a competition?

Access the [Competitions](#) home page in the [Cortana Intelligence Gallery](#), or go to <http://aka.ms/CIComp>. The home page lists all competitions that are currently running. Each competition has detailed instructions and participation rules, prizes, and duration on its sign-up page.

1. Find the competition you'd like to participate in, read all the instructions and watch the tutorial video, then click the **Enter Competition** button to copy the Starter Experiment into your existing Azure Machine Learning workspace. If you don't already have access to a workspace, you must create one beforehand. Run the Starter Experiment, observe the performance metric, then use your creativity to improve the performance of the model. You'll likely spend the majority of your time in this step.
2. Create a Predictive Experiment with the trained model out of your Starter Experiment. Then carefully adjust the input and output schema of the web service to ensure they conform to the requirements specified in the Competition documentation. The tutorial document generally will have detailed instructions for this. You can also watch the tutorial video, if available.
3. Deploy a web service out of your Predictive Experiment. Test your web service using the **Test** button or the Excel template automatically created for you to ensure it's working properly.
4. Submit your web service as the competition entry and see your public score in the Cortana Intelligence Gallery competition page. And celebrate if you make it onto the leaderboard!

After you successfully submit an entry, you can go back to your copied Starter Experiment, iterate, and update your Predictive Experiment, update the web service, and submit a new entry.

Can I use open source tools for participating in these Competitions?

The competition participants leverage Azure Machine Learning Studio, a cloud-based service within Cortana Intelligence Suite for development of the data science models and to create Competition entries for submission. Machine Learning Studio not only provides a GUI interface for constructing machine learning experiments, it also allows you to bring your own R and/or Python scripts for native execution. The R and Python runtimes in Studio come with a rich set of open source R/Python packages, and you can import your own packages as part of the experiment as well. Studio also has a built-in Jupyter Notebook service for you to do free style data exploration. Of

course, you can always download the datasets used in the Competition and explore it in your favorite tool outside of Machine Learning Studio.

Do I need to be a data scientist to enter?

No. In fact, we encourage data enthusiasts, those curious about data science, and other aspiring data scientists to enter our contest. We have designed supporting documents to allow everyone to compete. Our target audience is:

- **Data Developers, Data Scientists, BI and Analytics Professionals:** those who are responsible for producing data and analytics content for others to consume
- **Data Stewards:** those who have the knowledge about the data, what it means, and how it's intended to be used and for which purpose
- **Students & Researchers:** those who are learning and gaining data related skills via academic programs in universities, or participants in Massive Open Online Courses (MOOCs)

Can I enter with my colleagues as a team?

The Competition platform currently doesn't support team participation. Each competition entry is tied to a single user identity.

Do I need to pay to participate in a competition?

Competitions are free to participate in. You do, however, need access to an Azure Machine Learning workspace to participate. You can create a Free workspace without a credit card by simply logging in with a valid Microsoft account, or an Office 365 account. If you're already an Azure or Cortana Intelligence Suite customer, you can create and use a Standard workspace under the same Azure subscription. If you'd like to purchase an Azure subscription, go to the [Azure pricing](#) page. Note that the standard rates will apply when using a Standard workspace to construct experiments. See [Azure Machine Learning pricing information](#) for more information.

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

What are public and private scores?

In most competitions, you'll notice that you'll receive a public score for every submission you make, generally within 10-20 minutes. But after the competition ends, you'll receive a private score which is used for final ranking.

Here's what happens:

- The entire dataset used in the competition is randomly split with stratification into training and testing (the remaining) data. The random split is stratified to ensure that the distributions of labels in both training and testing data are consistent.
- The training data is uploaded and given to you as part of the Starter Experiment in the Import Data module configuration.
- The testing data is further split into public and private testing data, using the same stratification.
- The public testing data is used for the initial round of scoring. The result is referred to as the public score and it's what you see in your submission history when you submit your entry. This score is calculated for every entry you submit. This public score is used to rank you on the public leaderboard.
- The private testing data is used for the final round of scoring after the Competition ends. This is referred to as private score.
- For each participant, a fixed number of entries with the highest public scores are automatically selected to enter the private scoring round (this number can vary depending on the competition). The entry with the highest private score is then selected to enter the final ranking, which ultimately determines the prize winners.

Can Customers host a Competition on our platform?

We welcome 3rd-party organizations to partner with us and host both public and private competitions on our platform. We have a competition onboarding team who will be happy to discuss the onboarding process for such competitions. Please get in touch with us at compsupport@microsoft.com for more details.

What are the limitations for submissions?

A typical competition may choose to limit the number of entries you can submit within a 24-hour span (UTC time 00:00:00 to 23:59:59), and the total number of entries you can submit over the duration of the competition. You'll receive appropriate error messages when a limitation is exceeded.

What happens if I win a Competition?

Microsoft will verify the results of the private leaderboard, and then we'll contact you. Please make sure that your email address in your user profile is up to date.

How will I get the prize money if I win a Competition?

If you are a competition winner, you will need to sign a declaration of eligibility, license, and release from. This form reiterates the Competition Rules. Winners need to fill out a US Tax form W-9, or a Form W-8BEN if they are not US taxpayers. We will contact all winners as part of the rewards disbursement process by using their registration email. Please refer to our [Terms and Conditions](#) for additional details.

What if more than one entry receives the same score?

The submission time is the tie-breaker. The entry submitted earlier outranks the entry submitted later.

Can I participate using Guest workspace?

No. You must use a Free or a Standard workspace to participate. You can open the Competition starter experiment in a Guest workspace, but you'll not be able to create a valid entry for submission from that workspace.

Can I participate using a Workspace in any Azure region?

Currently, the Competition platform only supports entries submitted from a workspace in the **South Central US** Azure region. All Free workspaces reside in South Central US, so you can submit an entry from any Free workspace. If you choose to use a Standard workspace, just ensure that it resides in the South Central US Azure region, otherwise your submission won't succeed.

Do we keep Users' Competitions Solutions/Entries?

User entries are only retained for evaluation purposes to identify the winning solutions. Please refer to our [Terms and Conditions](#) for specifics.

Discover and share tutorials in the Cortana Intelligence Gallery

1/17/2017 • 2 min to read • [Edit on GitHub](#)

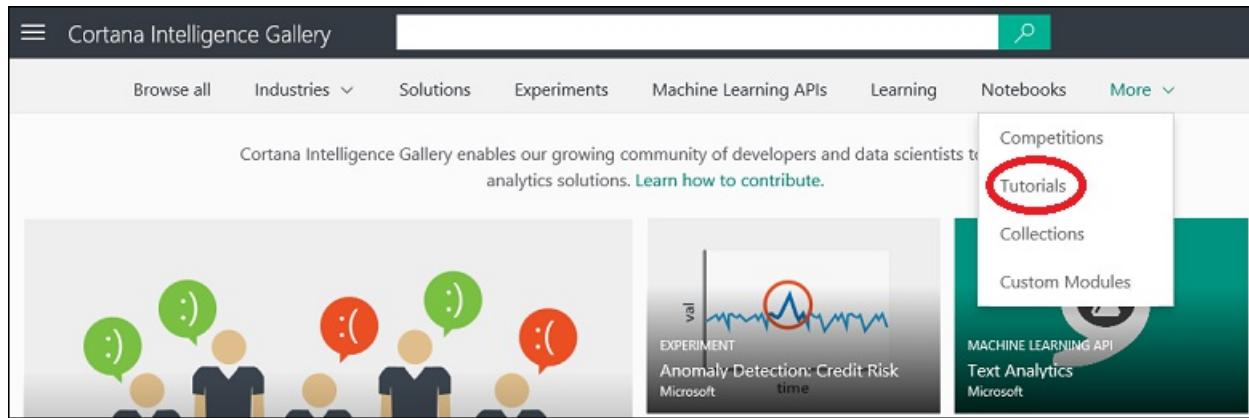
The [Cortana Intelligence Gallery](#) is a community driven site for discovering and sharing solutions built with the Cortana Intelligence Suite. The Gallery contains a variety of resources that you can use to develop your own analytics solutions.

Tutorials

A number of **Tutorials** are available to walk you through machine learning technologies and concepts, or to describe advanced methods for solving various machine learning problems.

Discover

To browse for tutorials in the Gallery, open the [Gallery](#), point your mouse at **More** at the top of the Gallery home page, and select **Tutorials**.



The [Tutorials](#) page displays a list of the most recently added and most popular tutorials. Click **See all** to view all tutorials. From this page, you can browse all the tutorials in the Gallery. You also can search by selecting filter criteria on the left of the page and entering search terms at the top.

Click any tutorial to open the tutorial's details page and read more information about it. On this page you can comment, provide feedback, or ask questions through the comments section. You can even share it with friends or colleagues using the share capabilities of LinkedIn or Twitter. You can also email a link to the tutorial to invite other users to view the page.

Two screenshots of a tutorial details page. The top screenshot shows a summary box with '26 views', '30 downloads', and social sharing buttons for Twitter, LinkedIn, and Email, with the LinkedIn button circled in red. The bottom screenshot shows the full page with a header for 'Cortana Intelligence Gallery', a login link, and a comment section with 8 comments, sorting options, and a placeholder for joining the discussion.

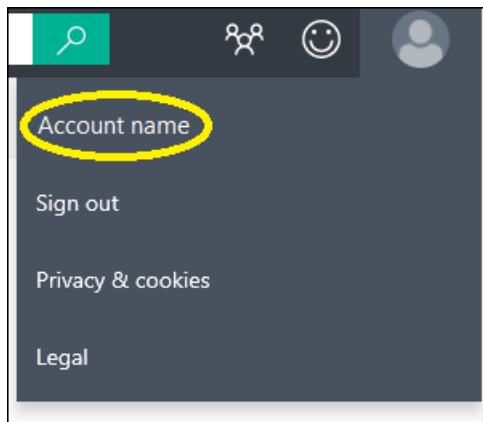
Contribute

You can contribute a tutorial to the Gallery to help other users solve a problem or learn a concept.

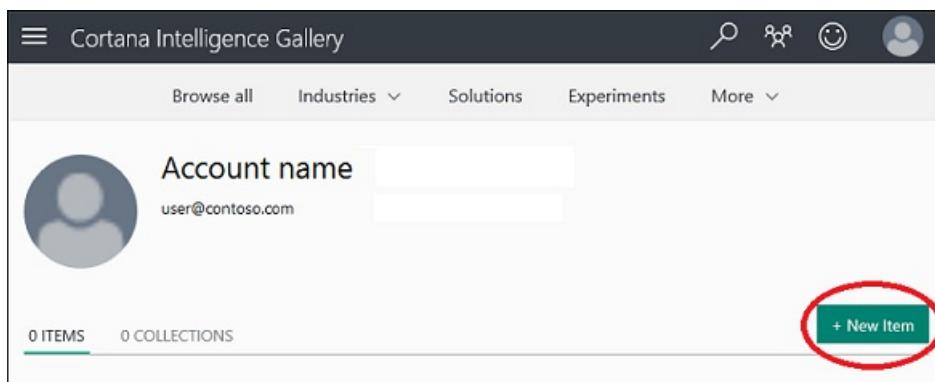
Create a tutorial

Follow these steps to create and contribute a tutorial to the Cortana Intelligence Gallery:

1. Sign in to the Gallery using your Microsoft account
2. Click your image at the top of the window and then click your name



3. Click **New Item**



4. Select **Collection** for **Item Type**, then give the collection a name, a brief summary, a description, and any tags that will help users find the collection

CREATE NEW GALLERY ITEM

Description

ITEM TYPE

Tutorial

NAME

Create a simple experiment in Azure Machine Learning Studio

SUMMARY

Follow a simple step-by-step process to create your first experiment in Azure Machine Learning Studio.

DETAILED DESCRIPTION

 Markdown  Preview

B *I* |       |     |  

TAGS

Next

5. Click **Next** - you can upload an image file, or select a stock image, that will be displayed with the collection; choose something that will help users identify the content and purpose of the collection

CREATE NEW GALLERY ITEM X

Image Selection

Use a JPG, GIF, or PNG. A resolution of 960 by 640 pixels works best.



UPLOAD AN IMAGE SELECT STOCK IMAGE

BROWSE YOUR COMPUTER...

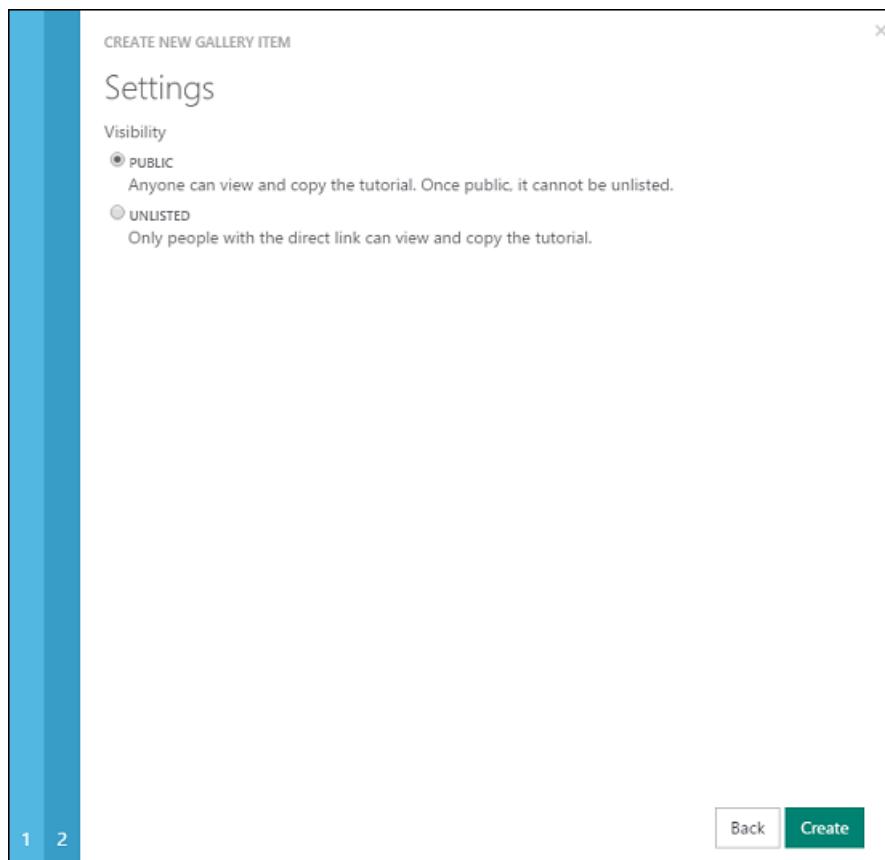
1

Back Next

6. Click **Next** - you can decide whether the tutorial is **Public** (it can be viewed by anyone) or **Unlisted** (only people with a direct link can view the tutorial)

IMPORTANT

Once you set a tutorial to **Public**, you can not set it to **Unlisted**.



7. Click **Create**

Your tutorial is now part of the Cortana Intelligence Gallery. It will be listed on your account page under the **Items** tab.

[TAKE ME TO THE GALLERY >>](#)

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

Discover and share collections in the Cortana Intelligence Gallery

1/17/2017 • 3 min to read • [Edit on GitHub](#)

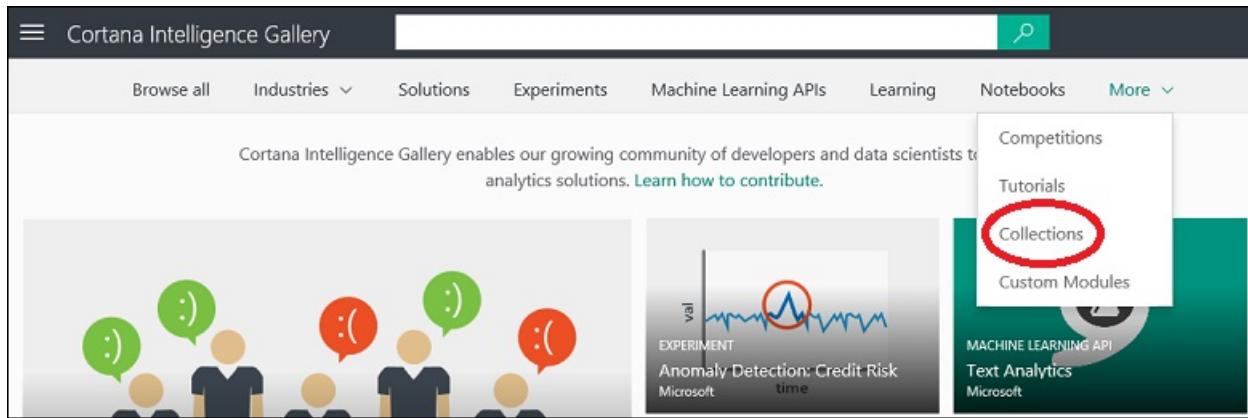
The [Cortana Intelligence Gallery](#) is a community driven site for discovering and sharing solutions built with the Cortana Intelligence Suite. The Gallery contains a variety of resources that you can use to develop your own analytics solutions.

Collections

A **Collection** allows you to group together experiments, APIs, and other Gallery items that address a specific solution or concept. These can be grouped together for later reference, use, or sharing.

Discover

To browse for collections in the Gallery, open the [Gallery](#), point your mouse at **More** at the top of the Gallery home page, and select **Collections**.



The [Collections](#) page displays a list of the most recently added and most popular collections. Click **See all** to view all collections. From this page, you can browse all the collections in the Gallery. You also can search by selecting filter criteria on the left of the page and entering search terms at the top.

Click any collection to open the collection's details page and read more information about it. On this page you can comment, provide feedback, or ask questions through the comments section. You can even share it with friends or colleagues using the share capabilities of LinkedIn or Twitter. You can also email a link to the experiment to invite other users to view the page.

Two screenshots of a collection details page. The top part shows a summary with '26 views', '30 downloads', and social sharing buttons for Twitter, LinkedIn, and Email, with the LinkedIn button circled in red. The bottom part shows a comments section with '8 Comments', a login link, and a 'Join the discussion...' input field.

Contribute

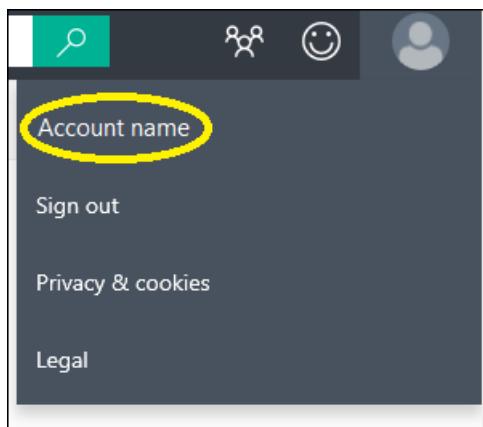
You can create a collection that contains items that you own or items that have been contributed by the community. They can be any Gallery items that address a specific solution or concept.

For example, you can use a collection to group together items on a specific topic, or you can group together a multi-step experiment that solves a complex problem. The initial collections contributed by Microsoft consist of multi-step machine learning experiment templates for solving real world problems, such as online fraud detection, text classification, retail forecasting, and predictive maintenance.

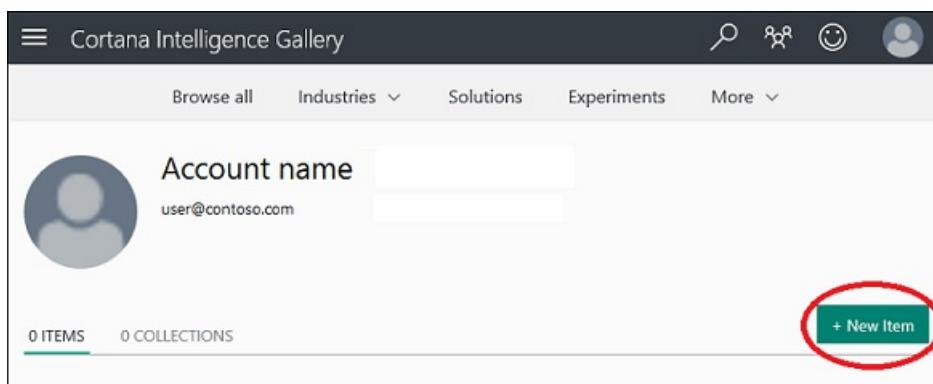
Create a collection

Follow these steps to create and contribute a collection to the Cortana Intelligence Gallery:

1. Sign in to the Gallery using your Microsoft account
2. Click your image at the top of the window and then click your name



3. Click **New Item**



4. Select **Collection** for **Item Type**, then give the collection a name, a brief summary, a description, and any tags that will help users find the collection

CREATE NEW GALLERY ITEM

Description

ITEM TYPE

Collection

NAME

Linear regression examples

SUMMARY

Collection of various experiments, tutorials, etc. related to linear regression

DETAILED DESCRIPTION

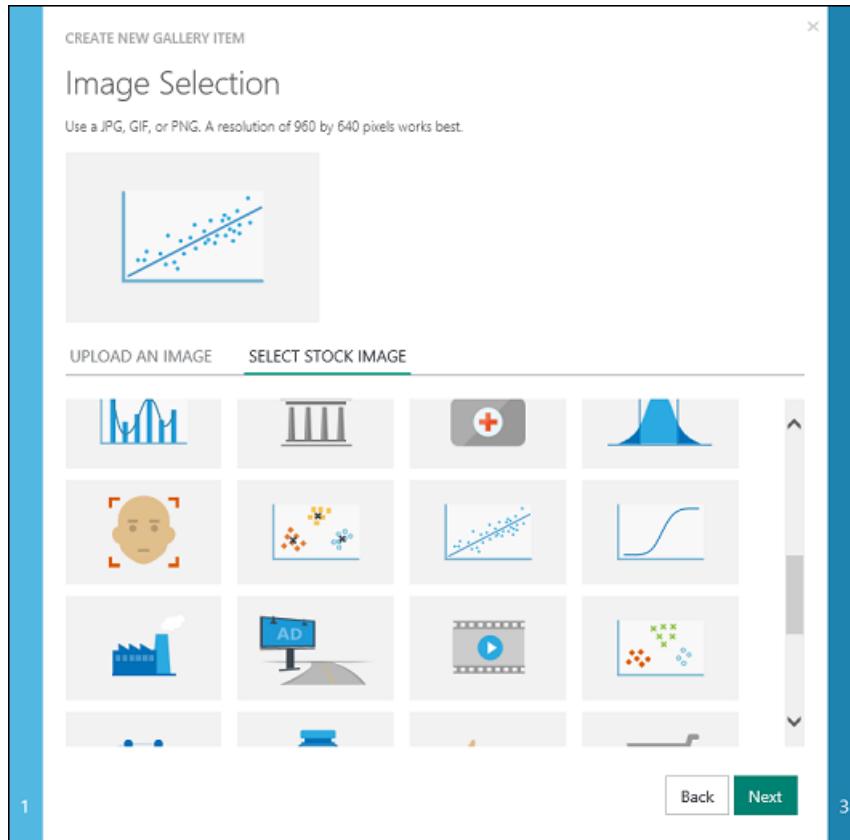
Markdown Preview

B *I* | “ “ | | ↺ ↻

TAGS

Next 2

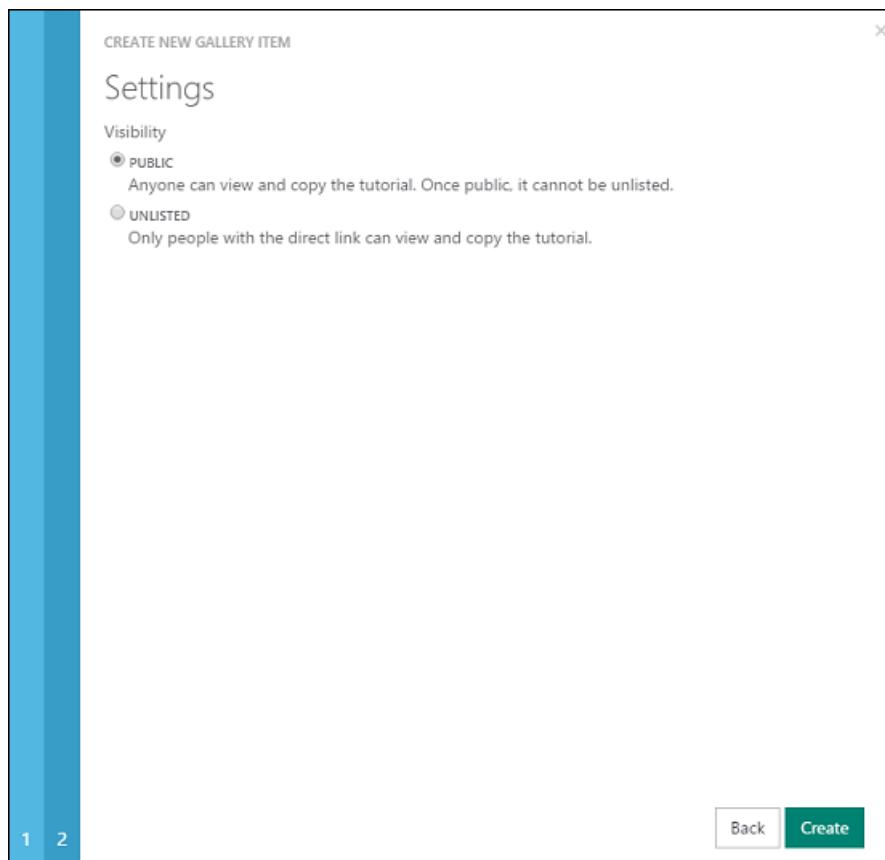
- Click **Next** - you can upload an image file, or select a stock image, that will be displayed with the collection; choose something that will help users identify the content and purpose of the collection



- Click **Next** - you can decide whether the collection is **Public** (it can be viewed by anyone) or **Unlisted** (only people with a direct link can view the collection)

IMPORTANT

Once you set a collection to **Public**, you can not set it to **Unlisted**.



7. Click **Create**

Your collection is now part of the Cortana Intelligence Gallery. It will be listed on your account page under the **Collection** tab.

Add items to a collection

You can add items to your collection by opening the collection, clicking **Edit**, and then clicking **Add Item**.

COLLECTION

A random collection of things

Gary Ericson • September 29, 2016

Done

Add Item

Summary

You'll be shown a list of items you've contributed to the Gallery, or you can search the Gallery for items to add. Click an item to select it. Every item you select is included in the set of items to add - the **Add** button indicates how many items have been selected.

Add to collection

YOUR ITEMS SEARCH GALLERY

Your first data science experiment - Automobile price prediction
EXPERIMENT by Gary Ericson

Create a simple experiment in Azure Machine Learning Studio
TUTORIAL by Gary Ericson

A random collection of things
COLLECTION by Gary Ericson

CANCEL ADD

Or, if you find an item while browsing through the Gallery that you want to include, just open the item, click **Add to collection**, and specify the collection you want to add it to.

Open in Studio

+ Add to Collection

12915 views

You can change the summary, description, or tags of your collection by opening the collection and clicking **Edit**. While you're editing your collection, you can also change the order of the items in the collection by using the arrow buttons next to an item to move it in the list. And you add notes to the items in your collection by clicking the upper-right corner of an item and selecting **Add/Edit note**. To remove an item from your collection, select

Remove.

COLLECTION

A random collection of things [Edit](#)

Gary Ericson • September 29, 2016

[Done](#) [⚙️](#) [Add Item](#)

Summary [Edit](#)

Description [Edit](#)

4 Items

Your first data science experiment - Automobile price prediction
EXPERIMENT by Gary Ericson

Womens Health Risk Assessment
COMPETITION by Microsoft

HEART DISEASE

Add/Edit note
Use as collection image
Remove

[TAKE ME TO THE GALLERY >>](#)

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

Discover custom machine learning modules in the Cortana Intelligence Gallery

1/17/2017 • 3 min to read • [Edit on GitHub](#)

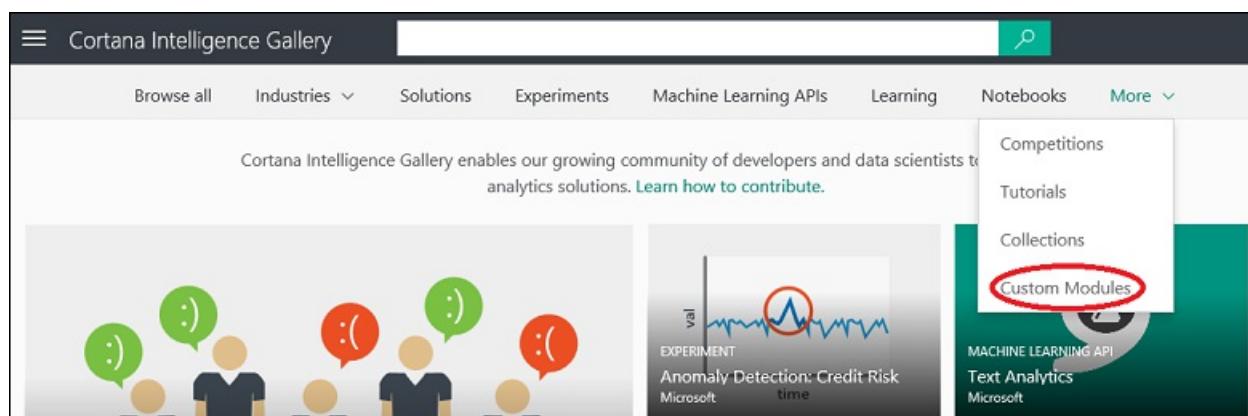
The [Cortana Intelligence Gallery](#) is a community driven site for discovering and sharing solutions built with the Cortana Intelligence Suite. The Gallery contains a variety of resources that you can use to develop your own analytics solutions.

Custom modules for Machine Learning Studio

A number of [Custom Modules](#) are available in the Cortana Intelligence Gallery that expand the capabilities of Azure Machine Learning Studio. You can download these modules for use in your experiments so that you can develop even more advanced predictive analytics solutions.

Discover

To browse for custom modules in the Gallery, open the [Gallery](#), point your mouse at **More** at the top of the Gallery home page, and select **Custom Modules**.



The [Custom Modules](#) page displays a list of the most popular modules. Click **See all** to view all custom modules. From this page, you can browse all the custom modules in the Gallery. You also can search by selecting filter criteria on the left of the page and entering search terms at the top.

Comment and share

Click any custom module to open the module's details page and read information about what the module does and how to use it. On this page you can comment, provide feedback, or ask questions through the comments section. You can even share it with friends or colleagues using the share capabilities of LinkedIn or Twitter. You can also email a link to the custom module to invite other users to view the page.



The screenshot shows a web-based interface for the Cortana Intelligence Gallery. At the top left, there are links for '8 Comments' and 'Cortana Intelligence Gallery'. On the right, there are 'Login' and 'Sort by Best' buttons. Below the header, there are 'Recommend' and 'Share' buttons. A user profile icon is shown next to a text input field that says 'Join the discussion...'. The main content area displays a module's details, which is partially visible.

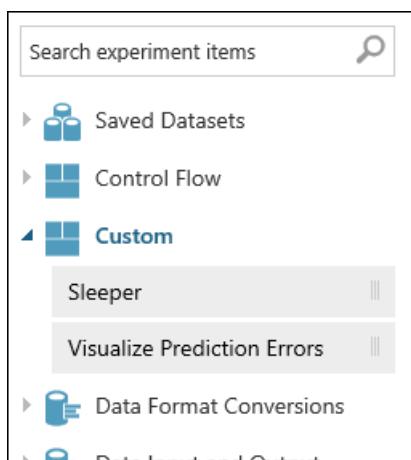
Download

You can use any custom module in the Gallery in your own experiments. There are two ways to get a copy of the module:

- **From the Gallery** - When you download a module from the Gallery, the module is added to the module palette in your workspace. You also get a sample experiment that gives you an example of how to use the module.
- **From within Machine Learning Studio** - You can import any custom module while you're working in Studio, and the module is added to the module palette in your workspace.

Once the custom module is in your module palette, it's available for you to use in any experiment in your workspace just like any other module:

1. Create a new experiment or open an existing one
2. In the module palette to the left of the experiment canvas, click **Custom** to expand the list of custom modules in your workspace

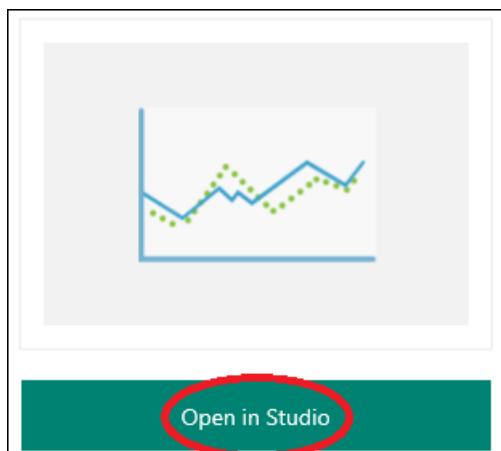


3. Select the module you imported and drag it to your experiment

From the Gallery

To download a copy of a custom module from the Gallery:

1. Open the module's details page in the Gallery
2. Click **Open in Studio**



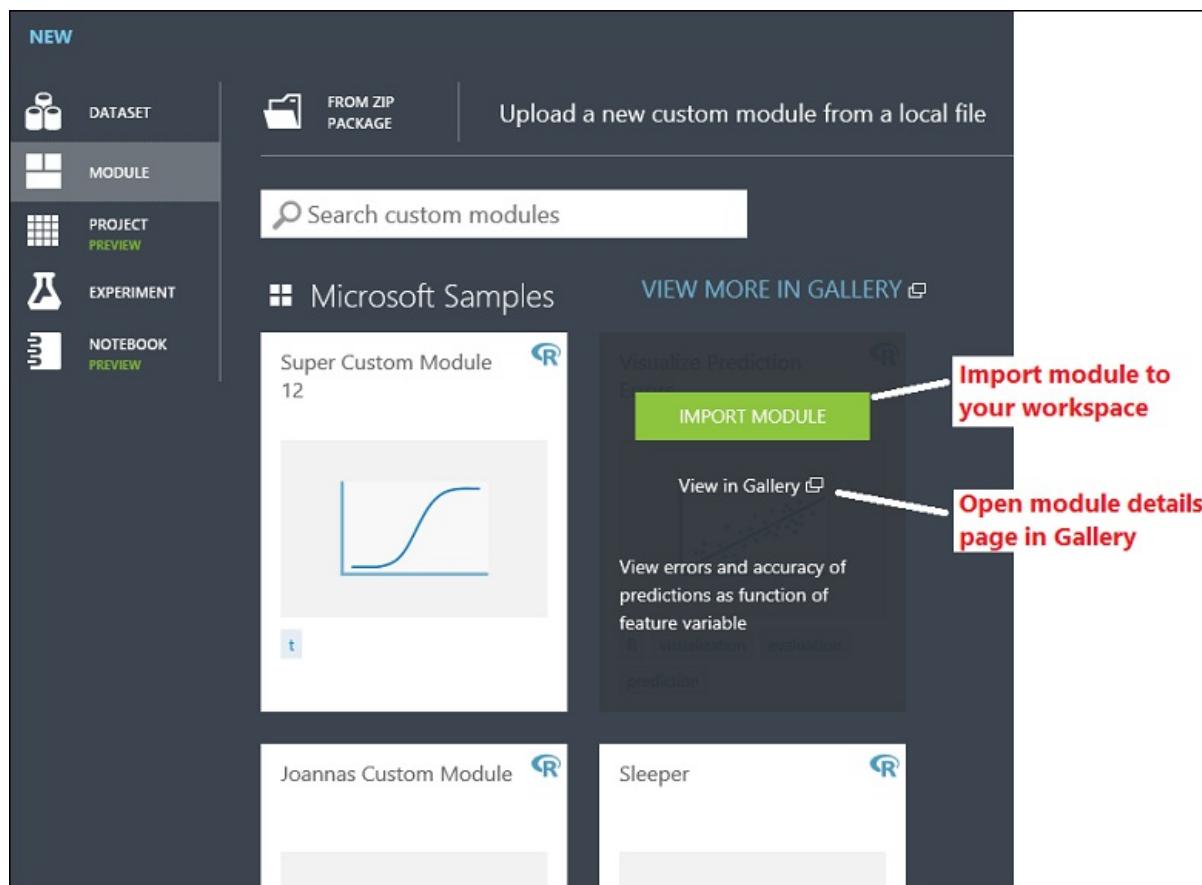
Each custom module includes a sample experiment that demonstrates how to use the module. When you click **Open in Studio**, this sample experiment is loaded into your Machine Learning Studio workspace and opened (if you're not already signed in to Studio, you will be prompted to sign in using your Microsoft account before the experiment is copied to your workspace).

The custom module is copied to your workspace and placed in your module palette alongside all the other built-in or custom Studio modules. You can now use it like any other module in your workspace.

From within Machine Learning Studio

You can also import the same custom modules from the Gallery while you're working in Machine Learning Studio:

1. In Machine Learning Studio, click **+NEW**
2. Select **Module** - a list of Gallery modules is displayed that you can choose from, or you can find a specific module using the search box
3. Point your mouse at a module and select **Import Module** (to see information about the module, select **View in Gallery** which takes you to the details page for the module in the Gallery)



The custom module is copied to your workspace and placed in your module palette alongside all the other built-in or custom Studio modules. You can now use it like any other module in your workspace.

TAKE ME TO THE GALLERY >>

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

Machine Learning Anomaly Detection API

1/17/2017 • 10 min to read • [Edit on GitHub](#)

Overview

[Anomaly Detection API](#) is an example built with Azure Machine Learning that detects anomalies in time series data with numerical values that are uniformly spaced in time.

This API can detect the following types of anomalous patterns in time series data:

- **Positive and negative trends:** For example, when monitoring memory usage in computing an upward trend may be of interest as it may be indicative of a memory leak,
- **Changes in the dynamic range of values:** For example, when monitoring the exceptions thrown by a cloud service, any changes in the dynamic range of values could indicate instability in the health of the service, and
- **Spikes and Dips:** For example, when monitoring the number of login failures in a service or number of checkouts in an e-commerce site, spikes or dips could indicate abnormal behavior.

These machine learning detectors track such changes in values over time and report ongoing changes in their values as anomaly scores. They do not require adhoc threshold tuning and their scores can be used to control false positive rate. The anomaly detection API is useful in several scenarios like service monitoring by tracking KPIs over time, usage monitoring through metrics such as number of searches, numbers of clicks, performance monitoring through counters like memory, CPU, file reads, etc. over time.

The Anomaly Detection offering comes with useful tools to get you started.

- The [web application](#) helps you evaluate and visualize the results of anomaly detection APIs on your data.

NOTE

Try [IT Anomaly Insights solution](#) powered by [this API](#)

To get this end to end solution deployed to your Azure subscription [Start here >](#)

API Deployment

In order to use the API, you must deploy it to your Azure subscription where it will be hosted as an Azure Machine Learning web service. You can do this from the [Cortana Intelligence Gallery](#). This will deploy two AzureML Web Services (and their related resources) to your Azure subscription - one for anomaly detection with seasonality detection, and one without seasonality detection. Once the deployment has completed, you will be able to manage your APIs from the [AzureML web services](#) page. From this page, you will be able to find your endpoint locations, API keys, as well as sample code for calling the API. More detailed instructions are available [here](#).

Scaling the API

By default, your deployment will have a free Dev/Test billing plan which includes 1,000 transactions/month and 2 compute hours/month. You can upgrade to another plan as per your needs. Details on the pricing of different plans are available [here](#) under "Production Web API pricing".

Managing AML Plans

You can manage your billing plan [here](#). The plan name will be based on the resource group name you chose when

deploying the API, plus a string that is unique to your subscription. Instructions on how to upgrade your plan are available [here](#) under the "Managing billing plans" section.

API Definition

The web service provides a REST-based API over HTTPS that can be consumed in different ways including a web or mobile application, R, Python, Excel, etc. You send your time series data to this service via a REST API call, and it runs a combination of the three anomaly types described below.

Calling the API

In order to call the API, you will need to know the endpoint location and API key. Both of these, along with sample code for calling the API, are available from the [AzureML web services](#) page. Navigate to the desired API, and then click the "Consume" tab to find them. Note that you can call the API as a Swagger API (i.e. with the URL parameter `format=swagger`) or as a non-Swagger API (i.e. without the `format` URL parameter). The sample code uses the Swagger format. Below is an example request and response in non-Swagger format. These examples are to the seasonality endpoint. The non-seasonality endpoint is similar.

Sample Request Body

The request contains two objects: `input1` and `GlobalParameters`. In the example request below, some parameters are sent explicitly while others are not (scroll down for a full list of parameters for each endpoint). Parameters that are not sent explicitly in the request will use the default values given below.

```
{
  "input1": {
    "ColumnNames": ["Time", "Data"],
    "Values": [
      ["5/30/2010 18:07:00", "1"],
      ["5/30/2010 18:08:00", "1.4"],
      ["5/30/2010 18:09:00", "1.1"]
    ]
  },
  "GlobalParameters": {
    "tspikedetector.sensitivity": "3",
    "zspikedetector.sensitivity": "3",
    "bileveldetector.sensitivity": "3.25",
    "detectors.spikesdips": "Both"
  }
}
```

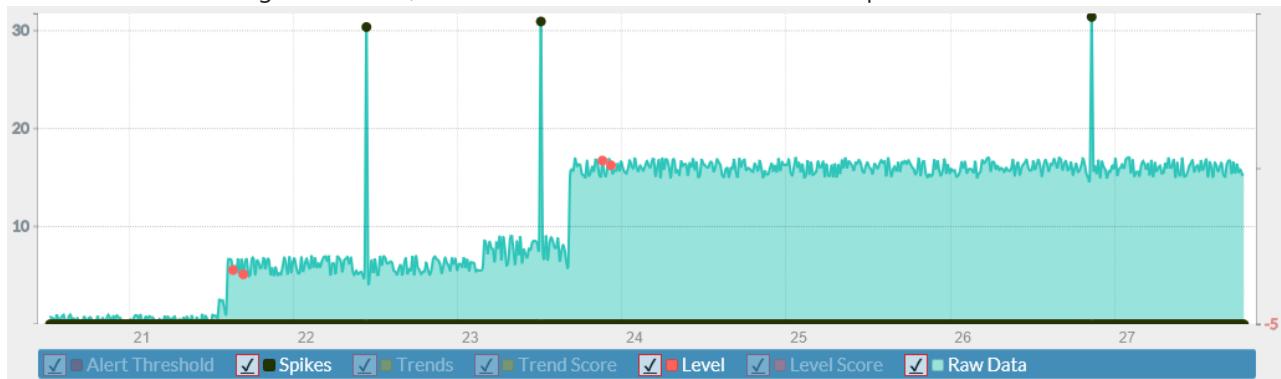
Sample Response

Note that, in order to see the `ColumnNames` field, you must include `details=true` as a URL parameter in your request. See the tables below for the meaning behind each of these fields.

```
{
  "Results": {
    "output1": {
      "type": "table",
      "value": {
        "Values": [
          ["5/30/2010 6:07:00 PM", "1", "1", "0", "0", "-0.687952590518378", "0", "-0.687952590518378", "0", "-0.687952590518378", "0"],
          ["5/30/2010 6:08:00 PM", "1.4", "1.4", "0", "0", "-1.07030497733224", "0", "-0.884548154298423", "0", "-1.07030497733224", "0"],
          ["5/30/2010 6:09:00 PM", "1.1", "1.1", "0", "0", "-1.30229513613974", "0", "-1.173800281031", "0", "-1.30229513613974", "0"]
        ],
        "ColumnNames": ["Time", "OriginalData", "ProcessedData", "TSpike", "ZSpike", "BiLevelChangeScore", "BiLevelChangeAlert", "PosTrendScore", "Pos Trend Alert", "NegTrendScore", "NegTrendAlert"],
        "ColumnTypes": ["DateTime", "Double", "Double", "Double", "Double", "Double", "Int32", "Double", "Int32", "Double", "Int32"]
      }
    }
  }
}
```

Score API

The Score API is used for running anomaly detection on non-seasonal time series data. The API runs a number of anomaly detectors on the data and returns their anomaly scores. The figure below shows an example of anomalies that the Score API can detect. This time series has 2 distinct level changes, and 3 spikes. The red dots show the time at which the level change is detected, while the black dots show the detected spikes.



Detectors

The anomaly detection API supports detectors in 3 broad categories. Details on specific input parameters and outputs for each detector can be found in the following table.

Detector Category	Detector	Description	Input Parameters	Outputs
Spike Detectors	TSpike Detector	Detect spikes and dips based on far the values are from first and third quartiles	<i>tspikedetector.sensitivity</i> : takes integer value in the range 1-10, default: 3; Higher values will catch more extreme values thus making it less sensitive	TSpike: binary values – '1' if a spike/dip is detected, '0' otherwise
Spike Detectors	ZSpike Detector	Detect spikes and dips based on how far the datapoints are from their mean	<i>zspikedetector.sensitivity</i> : take integer value in the range 1-10, default: 3; Higher values will catch more extreme values making it less sensitive	ZSpike: binary values – '1' if a spike/dip is detected, '0' otherwise

Detector Category	Detector	Description	Input Parameters	Outputs
Slow Trend Detector	Slow Trend Detector	Detect slow positive trend as per the set sensitivity	<i>trenddetector.sensitivity</i> : threshold on detector score (default: 3.25, 3.25 – 5 is a reasonable range to select this from; The higher the less sensitive)	tscore: floating number representing anomaly score on trend
Level Change Detectors	Bidirectional Level Change Detector	Detect both upward and downward level change as per the set sensitivity	<i>bileveldetector.sensitivity</i> : threshold on detector score (default: 3.25, 3.25 – 5 is a reasonable range to select this from; The higher the less sensitive)	rpscore: floating number representing anomaly score on upward and downward level change

Parameters

More detailed information on these input parameters is listed in the table below:

Input Parameters	Description	Default Setting	Type	Valid Range	Suggested Range
detectors.historyWindow	History (in # of data points) used for anomaly score computation	500	integer	10-2000	Time-series dependent
detectors.spikesdips	Whether to detect only spikes, only dips, or both	Both	enumerated	Both, Spikes, Dips	Both
bileveldetector.sensitivity	Sensitivity for bidirectional level change detector.	3.25	double	None	3.25-5 (Lesser values mean more sensitive)
trenddetector.sensitivity	Sensitivity for positive trend detector.	3.25	double	None	3.25-5 (Lesser values mean more sensitive)
tspikedetector.sensitivity	Sensitivity for TSpike Detector	3	integer	1-10	3-5 (Lesser values mean more sensitive)
zspikedetector.sensitivity	Sensitivity for ZSpike Detector	3	integer	1-10	3-5 (Lesser values mean more sensitive)
postprocess.tailRows	Number of the latest data points to be kept in the output results	0	integer	0 (keep all data points), or specify number of points to keep in results	N/A

Output

The API runs all detectors on your time series data and returns anomaly scores and binary spike indicators for each point in time. The table below lists outputs from the API.

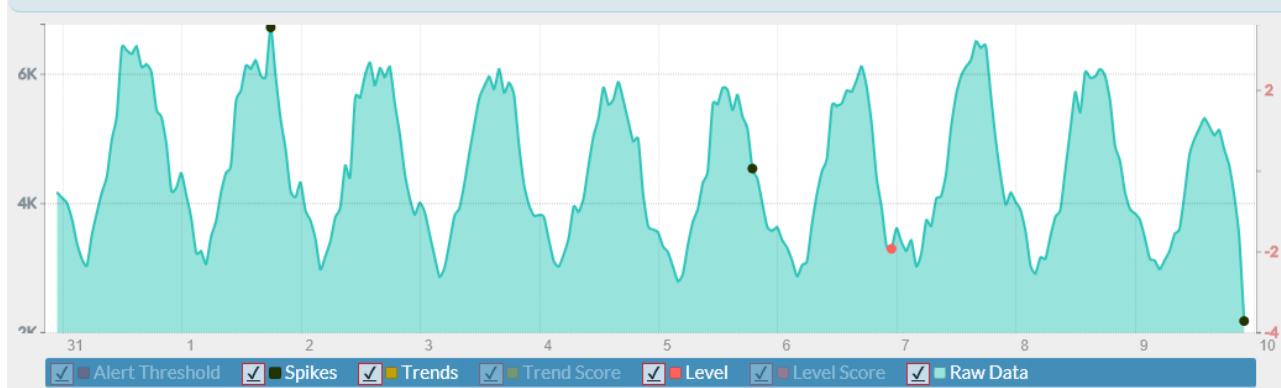
OUTPUTS	DESCRIPTION
Time	Timestamps from raw data, or aggregated (and/or) imputed data if aggregation (and/or) missing data imputation is applied
Data	Values from raw data, or aggregated (and/or) imputed data if aggregation (and/or) missing data imputation is applied
TSpike	Binary indicator to indicate whether a spike is detected by TSpike Detector
ZSpike	Binary indicator to indicate whether a spike is detected by ZSpike Detector
rpscore	A floating number representing anomaly score on bidirectional level change
rpalert	1/0 value indicating there is a bidirectional level change anomaly based on the input sensitivity
tscore	A floating number representing anomaly score on positive trend
talert	1/0 value indicating there is a positive trend anomaly based on the input sensitivity

ScoreWithSeasonality API

The ScoreWithSeasonality API is used for running anomaly detection on time series that have seasonal patterns. This API is useful to detect deviations in seasonal patterns.

The following figure shows an example of anomalies detected in a seasonal time series. The time series has one spike (the 1st black dot), two dips (the 2nd black dot and one at the end), and one level change (red dot). Note that both the dip in the middle of the time series and the level change are only discernable after seasonal components are removed from the series.

With seasonality detection



Detectors

The detectors in the seasonality endpoint are similar to the ones in the non-seasonality endpoint, but with slightly different parameter names (listed below).

Parameters

More detailed information on these input parameters is listed in the table below:

INPUT PARAMETERS	DESCRIPTION	DEFAULT SETTING	TYPE	VALID RANGE	SUGGESTED RANGE
preprocess.aggregationInterval	Aggregation interval in seconds for aggregating input time series	0 (no aggregation is performed)	integer	0: skip aggregation, > 0 otherwise	5 minutes to 1 day, time-series dependent
preprocess.aggregationFunc	Function used for aggregating data into the specified AggregationInterval	mean	enumerated	mean, sum, length	N/A
preprocess.replaceMissing	Values used to impute missing data	lkv (last known value)	enumerated	zero, lkv, mean	N/A
detectors.historyWindow	History (in # of data points) used for anomaly score computation	500	integer	10-2000	Time-series dependent
detectors.spikesdips	Whether to detect only spikes, only dips, or both	Both	enumerated	Both, Spikes, Dips	Both
bileveldetector.sensitivity	Sensitivity for bidirectional level change detector.	3.25	double	None	3.25-5 (Lesser values mean more sensitive)
postrenddetector.sensitivity	Sensitivity for positive trend detector.	3.25	double	None	3.25-5 (Lesser values mean more sensitive)
negtrenddetector.sensitivity	Sensitivity for negative trend detector.	3.25	double	None	3.25-5 (Lesser values mean more sensitive)
tspikedetector.sensitivity	Sensitivity for TSpike Detector	3	integer	1-10	3-5 (Lesser values mean more sensitive)
zspikedetector.sensitivity	Sensitivity for ZSpike Detector	3	integer	1-10	3-5 (Lesser values mean more sensitive)
seasonality.enable	Whether seasonality analysis is to be performed	true	boolean	true, false	Time-series dependent

INPUT PARAMETERS	DESCRIPTION	DEFAULT SETTING	TYPE	VALID RANGE	SUGGESTED RANGE
seasonality.numSeasonality	Maximum number of periodic cycles to be detected	1	integer	1, 2	1-2
seasonality.transform	Whether seasonal (and) trend components shall be removed before applying anomaly detection	deseason	enumerated	none, deseason, deseasontrend	N/A
postprocess.tailRows	Number of the latest data points to be kept in the output results	0	integer	0 (keep all data points), or specify number of points to keep in results	N/A

Output

The API runs all detectors on your time series data and returns anomaly scores and binary spike indicators for each point in time. The table below lists outputs from the API.

OUTPUTS	DESCRIPTION
Time	Timestamps from raw data, or aggregated (and/or) imputed data if aggregation (and/or) missing data imputation is applied
OriginalData	Values from raw data, or aggregated (and/or) imputed data if aggregation (and/or) missing data imputation is applied
ProcessedData	Either of the following: <ul style="list-style-type: none">• Seasonally adjusted time series if significant seasonality has been detected and deseason option selected;• seasonally adjusted and detrended time series if significant seasonality has been detected and deseasontrend option selected• otherwise, this is the same as OriginalData
TSpike	Binary indicator to indicate whether a spike is detected by TSpike Detector
ZSpike	Binary indicator to indicate whether a spike is detected by ZSpike Detector
BiLevelChangeScore	A floating number representing anomaly score on level change
BiLevelChangeAlert	1/0 value indicating there is a level change anomaly based on the input sensitivity
PosTrendScore	A floating number representing anomaly score on positive trend

OUTPUTS	DESCRIPTION
PosTrendAlert	1/0 value indicating there is a positive trend anomaly based on the input sensitivity
NegTrendScore	A floating number representing anomaly score on negative trend
NegTrendAlert	1/0 value indicating there is a negative trend anomaly based on the input sensitivity

Machine Learning APIs: Text Analytics for Sentiment, Key Phrase Extraction, Language Detection and Topic Detection

1/17/2017 • 6 min to read • [Edit on GitHub](#)

NOTE

This guide is for version 1 of the API. For version 2, [refer to this document](#). Version 2 is now the preferred version of this API.

Overview

The Text Analytics API is a suite of text analytics [web services](#) built with Azure Machine Learning. The API can be used to analyze unstructured text for tasks such as sentiment analysis, key phrase extraction, language detection and topic detection. No training data is needed to use this API: just bring your text data. This API uses advanced natural language processing techniques to deliver best in class predictions.

You can see text analytics in action on our [demo site](#), where you will also find [samples](#) on how to implement text analytics in C# and Python.

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

Sentiment analysis

The API returns a numeric score between 0 & 1. Scores close to 1 indicate positive sentiment, while scores close to 0 indicate negative sentiment. Sentiment score is generated using classification techniques. The input features to the classifier include n-grams, features generated from part-of-speech tags, and word embeddings. Currently, English is the only supported language.

Key phrase extraction

The API returns a list of strings denoting the key talking points in the input text. We employ techniques from Microsoft Office's sophisticated Natural Language Processing toolkit. Currently, English is the only supported language.

Language detection

The API returns the detected language and a numeric score between 0 & 1. Scores close to 1 indicate 100% certainty that the identified language is true. A total of 120 languages are supported.

Topic detection

This is a newly released API which returns the top detected topics for a list of submitted text records. A topic is

identified with a key phrase, which can be one or more related words. This API requires a minimum of 100 text records to be submitted, but is designed to detect topics across hundreds to thousands of records. Note that this API charges 1 transaction per text record submitted. The API is designed to work well for short, human written text such as reviews and user feedback.

API Definition

Headers

Ensure that you include the correct headers in your request, which should be as follows:

```
Authorization: Basic <creds>
Accept: application/json

Where <creds> = ConvertToBase64("AccountKey:" + yourActualAccountKey);
```

You can find your account key from your account in the [Azure Data Market](#). Note that currently only JSON is accepted for input and output formats. XML is not supported.

Single Response APIs

GetSentiment

URL

```
https://api.datamarket.azure.com/data.ashx/aml/text-analytics/v1/GetSentiment
```

Example request

In the call below, we are requesting sentiment analysis for the phrase "Hello World":

```
GET https://api.datamarket.azure.com/data.ashx/aml/text-analytics/v1/GetSentiment?Text=hello+world
```

This will return a response as follows:

```
{
  "odata.metadata":"https://api.datamarket.azure.com/data.ashx/aml/text-analytics/v1/$metadata",
  "Score":1.0
}
```

GetKeyPhrases

URL

```
https://api.datamarket.azure.com/data.ashx/aml/text-analytics/v1/GetKeyPhrases
```

Example request

In the call below, we are requesting the key phrases found in the text "It was a wonderful hotel to stay at, with unique decor and friendly staff":

```
GET https://api.datamarket.azure.com/data.ashx/aml/text-analytics/v1/GetKeyPhrases?
Text=It+was+a+wonderful+hotel+to+stay+at,+with+unique+decor+and+friendly+staff
```

This will return a response as follows:

```
{  
  "odata.metadata": "https://api.datamarket.azure.com/data.ashx/aml/text-analytics/v1/$metadata",  
  "KeyPhrases": [  
    "wonderful hotel",  
    "unique decor",  
    "friendly staff"  
  ]  
}
```

GetLanguage

URL

```
https://api.datamarket.azure.com/data.ashx/aml/text-analytics/v1/GetLanguage
```

Example request

In the GET call below, we are requesting for the sentiment for the key phrases in the text *Hello World*

```
GET https://api.datamarket.azure.com/data.ashx/aml/text-analytics/v1/GetLanguages?  
Text=Hello+World
```

This will return a response as follows:

```
{  
  "UnknownLanguage": false,  
  "DetectedLanguages": [  
    {  
      "Name": "English",  
      "Iso6391Name": "en",  
      "Score": 1.0  
    }  
  ]  
}
```

Optional parameters

`NumberOfLanguagesToDetect` is an optional parameter. The default is 1.

Batch APIs

The Text Analytics service allows you to do sentiment and key-phrase extractions in batch mode. Note that each of the records scored counts as one transaction. As an example, if you request sentiment for 1000 records in a single call, 1000 transactions will be deducted.

Note that the IDs entered into the system are the IDs returned by the system. The web service does not check that these IDs are unique. It is the responsibility of the caller to verify uniqueness.

GetSentimentBatch

URL

```
https://api.datamarket.azure.com/data.ashx/aml/text-analytics/v1/GetSentimentBatch
```

Example request

In the POST call below, we are requesting for the sentiments of the phrases "Hello World", "Hello Foo World" and "Hello My World" in the body of the request:

```
POST https://api.datamarket.azure.com/data.ashx/aml/text-analytics/v1/GetSentimentBatch
```

Request body:

```
{"Inputs":  
[  
    {"Id":"1","Text":"hello world"},  
    {"Id":"2","Text":"hello foo world"},  
    {"Id":"3","Text":"hello my world"},  
]}
```

In the response below, you get the list of scores associated with your text Ids:

```
{  
    "odata.metadata": "<url>",  
    "SentimentBatch":  
    [  
        {"Score": 0.9549767, "Id": "1"},  
        {"Score": 0.7767222, "Id": "2"},  
        {"Score": 0.8988889, "Id": "3"}  
    ],  
    "Errors": []  
}
```

GetKeyPhrasesBatch

URL

```
https://api.datamarket.azure.com/data.ashx/aml/text-analytics/v1/GetKeyPhrasesBatch
```

Example request

In this example, we are requesting for the list of sentiments for the key phrases in the following texts:

- "It was a wonderful hotel to stay at, with unique decor and friendly staff"
- "It was an amazing build conference, with very interesting talks"
- "The traffic was terrible, I spent three hours going to the airport"

This request is made as a POST call to the endpoint:

```
POST https://api.datamarket.azure.com/data.ashx/aml/text-analytics/v1/GetKeyPhrasesBatch
```

Request body:

```
{"Inputs":  
[  
    {"Id": "1", "Text": "It was a wonderful hotel to stay at, with unique decor and friendly staff"},  
    {"Id": "2", "Text": "It was an amazing build conference, with very interesting talks"},  
    {"Id": "3", "Text": "The traffic was terrible, I spent three hours going to the airport"}  
]}
```

In the response below, you get the list of key phrases associated with your text Ids:

```
{
  "odata.metadata": "<url>",
  "KeyPhrasesBatch":
  [
    {"KeyPhrases": ["unique decor", "friendly staff", "wonderful hotel"], "Id": "1"},  

    {"KeyPhrases": ["amazing build conference", "interesting talks"], "Id": "2"},  

    {"KeyPhrases": ["hours", "traffic", "airport"], "Id": "3" }
  ],
  "Errors": []
}
```

GetLanguageBatch

In the POST call below, we are requesting language detection for two text inputs:

```
POST https://api.datamarket.azure.com/data.ashx/aml/text-analytics/v1/GetLanguageBatch
```

Request body:

```
{
  "Inputs": [
    {"Text": "hello world", "Id": "1"},  

    {"Text": "c'est la vie", "Id": "2"}
  ]
}
```

This returns the following response, where English is detected in the first input and French in the second input:

```
{
  "LanguageBatch": [
    {
      "Id": "1",
      "DetectedLanguages": [
        {
          "Name": "English",
          "Iso6391Name": "en",
          "Score": 1.0
        }
      ],
      "UnknownLanguage": false
    },
    {
      "Id": "2",
      "DetectedLanguages": [
        {
          "Name": "French",
          "Iso6391Name": "fr",
          "Score": 1.0
        }
      ],
      "UnknownLanguage": false
    }
  ],
  "Errors": []
}
```

Topic Detection APIs

This is a newly released API which returns the top detected topics for a list of submitted text records. A topic is identified with a key phrase, which can be one or more related words. Note that this API charges 1 transaction per text record submitted.

This API requires a minimum of 100 text records to be submitted, but is designed to detect topics across hundreds to thousands of records.

Topics – Submit job

URL

```
https://api.datamarket.azure.com/data.ashx/ama/text-analytics/v1/StartTopicDetection
```

Example request

In the POST call below, we are requesting topics for a set of 100 articles, where the first and last input articles are shown, and two StopPhrases are included.

```
POST https://api.datamarket.azure.com/data.ashx/ama/text-analytics/v1/StartTopicDetection HTTP/1.1
```

Request body:

```
{"Inputs": [
    {"Id":"1","Text":"I loved the food at this restaurant"},
    ...,
    {"Id":"100","Text":"I hated the decor"}
],
"StopPhrases": [
    "restaurant", "visitor"
]}
```

In the response below, you get the JobId for the submitted job:

```
{
    "odata.metadata": "<url>",
    "JobId": "<JobId>"
}
```

A list of single word or multiple word phrases which should not be returned as topics. Can be used to filter out very generic topics. For example, in a dataset about hotel reviews, "hotel" and "hostel" may be sensible stop phrases.

Topics – Poll for job results

URL

```
https://api.datamarket.azure.com/data.ashx/ama/text-analytics/v1/GetTopicDetectionResult
```

Example request

Pass the JobId returned from the 'Submit job' step to fetch the results. We recommend that you call this endpoint every minute until Status='Complete' in the response. It will take around 10 mins for a job to complete, or longer for jobs with many thousands of records.

```
GET https://api.datamarket.azure.com/data.ashx/ama/text-analytics/v1/GetTopicDetectionResult?JobId=<JobId>
```

While it is processing, the response will be as follows:

```
{
    "odata.metadata": "<url>",
    "Status": "Running",
    "TopicInfo": [],
    "TopicAssignment": [],
    "Errors": []
}
```

The API returns output in JSON format in the following format:

```
{
  "odata.metadata": "<url>",
  "Status": "Finished",
  "TopicInfo": [
    {
      "TopicId": "ed00480e-f0a0-41b3-8fe4-07c1593f4af",
      "Score": 8.0,
      "KeyPhrase": "food"
    },
    ...
    {
      "TopicId": "a5ca3f1a-fdb1-4f02-8f1b-89f2f626d692",
      "Score": 6.0,
      "KeyPhrase": "decor"
    }
  ],
  "TopicAssignment": [
    {
      "Id": "1",
      "TopicId": "ed00480e-f0a0-41b3-8fe4-07c1593f4af",
      "Distance": 0.7809
    },
    ...
    {
      "Id": "100",
      "TopicId": "a5ca3f1a-fdb1-4f02-8f1b-89f2f626d692",
      "Distance": 0.8034
    }
  ],
  "Errors": []
}
```

The properties for each part of the response are as follows:

TopicInfo properties

KEY	DESCRIPTION
TopicId	A unique identifier for each topic.
Score	Count of records assigned to topic.
KeyPhrase	A summarizing word or phrase for the topic. Can be 1 or multiple words.

TopicAssignment properties

KEY	DESCRIPTION
Id	Identifier for the record. Equates to the ID included in the input.
TopicId	The topic ID which the record has been assigned to.
Distance	Confidence that the record belongs to the topic. Distance closer to zero indicates higher confidence.

Recommendations API Sample Application Walkthrough

1/17/2017 • 4 min to read • [Edit on GitHub](#)

NOTE

You should start using the Recommendations API Cognitive Service instead of this version. The Recommendations Cognitive Service will be replacing this service, and all the new features will be developed there. It has new capabilities like batching support, a better API Explorer, a cleaner API surface, more consistent signup/billing experience, etc. Learn more about [Migrating to the new Cognitive Service](#)

Purpose

This document shows the usage of the Azure Machine Learning Recommendations API via a [sample application](#).

This application is not intended to include full functionality, nor does it use all the APIs. It demonstrates some common operations to perform when you first want to play with the Machine Learning recommendation service.

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

Introduction to Machine Learning recommendation service

Recommendations via the Machine Learning recommendation service are enabled when you build a recommendation model based on the following data:

- A repository of the items you want to recommend, also known as a catalog
- Data representing the usage of items per user or session (this can be acquired over time via data acquisition, not as part of the sample app)

After a recommendation model is built, you can use it to predict items that a user might be interested in, according to a set of items (or a single item) the user selects.

To enable the previous scenario, do the following in the Machine Learning recommendation service:

- Create a model: This is a logical container that holds the data (catalog and usage) and the prediction model(s). Each model container is identified via a unique ID, which is allocated when it is created. This ID is called the model ID, and it is used by most of the APIs.
- Upload to catalog: When a model container is created, you can associate to it a catalog.

Note: Creating a model and uploading to a catalog are usually performed once for the model lifecycle.

- Upload usage: This adds usage data to the model container.
- Build a recommendation model: After you have enough data, you can build the recommendation model. This operation uses the top Machine Learning algorithms to create a recommendation model. Each build is associated with a unique ID. You need to keep a record of this ID because it is necessary for the functionality of some APIs.

- Monitor the building process: A recommendation model build is an asynchronous operation, and it can take from several minutes to several hours, depending on the amount of data (catalog and usage) and the build parameters. Therefore, you need to monitor the build. A recommendation model is created only if its associated build completes successfully.
- (Optional) Choose an active recommendation model build: This step is only necessary if you have more than one recommendation model built in your model container. Any request to get recommendations without indicating the active recommendation model is redirected automatically by the system to the default active build.

Note: An active recommendation model is production ready and it is built for production workload. This differs from a non-active recommendation model, which stays in a test-like environment (sometimes called staging).

- Get recommendations: After you have a recommendation model, you can trigger recommendations for a single item or a list of items that you select.

You will usually invoke Get Recommendation for a certain period of time. During that period of time, you can redirect usage data to the Machine Learning recommendation system, which adds this data to the specified model container. When you have enough usage data, you can build a new recommendation model that incorporates the additional usage data.

Prerequisites

- Visual Studio 2013
- Internet access
- Subscription to the Recommendations API (<https://datamarket.azure.com/dataset/amla/recommendations>).

Azure Machine Learning sample app solution

This solution contains the source code, sample usage, catalog file, and directives to download the packages that are required for compilation.

The APIs used

The application uses Machine Learning recommendation functionality via a subset of available APIs. The following APIs are demonstrated in the application:

- Create model: Create a logical container to hold data and recommendation models. A model is identified by a name, and you cannot create more than one model with the same name.
- Upload catalog file: Use to upload catalog data.
- Upload usage file: Use to upload usage data.
- Trigger build: Use to create a recommendation model.
- Monitor build execution: Use to monitor the status of a recommendation model build.
- Choose a built model for recommendation: Use to indicate which recommendation model to use by default for a certain model container. This step is necessary only if you have more than one recommendation model and you want to activate a non-active build as the active recommendation model.
- Get recommendation: Use to retrieve recommended items according to a given single item or a set of items.

For a complete description of the APIs, please see the Microsoft Azure Marketplace documentation.

Note: A model can have several builds over time (not simultaneously). Each build is created with the same or an updated catalog and additional usage data.

Common pitfalls

- You need to provide your user name and your Microsoft Azure Marketplace primary account key to run the

sample app.

- Running the sample app consecutively will fail. The application flow includes creating, uploading, building the monitor, and getting recommendations from a predefined model; therefore, it will fail on consecutive execution if you do not change the model name between invocations.
- Recommendations might return without data. The sample app uses a very small catalog and usage file. Therefore, some items from the catalog will have no recommended items.

Disclaimer

The sample app is not intended to be run in a production environment. The data provided in the catalog is very small, and it will not provide a meaningful recommendation model. The data is provided as a demonstration.

Quick start guide for the Machine Learning Recommendations API

1/17/2017 • 12 min to read • [Edit on GitHub](#)

NOTE

You should start using the Recommendations API Cognitive Service instead of this version. The Recommendations Cognitive Service will be replacing this service, and all the new features will be developed there. It has new capabilities like batching support, a better API Explorer, a cleaner API surface, more consistent signup/billing experience, etc. Learn more about [Migrating to the new Cognitive Service](#)

This document describes how to onboard your service or application to use Microsoft Azure Machine Learning Recommendations. You can find more details on the Recommendations API in the [gallery](#).

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

General overview

To use Azure Machine Learning Recommendations, you need to take the following steps:

- Create a model - A model is a container of your usage data, catalog data, and the recommendation model.
- Import catalog data - A catalog contains metadata information on the items.
- Import usage data - Usage data can be uploaded in one of two ways (or both):
 - By uploading a file that contains the usage data.
 - By sending data acquisition events. Usually you upload a usage file in order to be able to create an initial recommendation model (bootstrap) and use it until the system gathers enough data by using the data acquisition format.
- Build a recommendation model - This is an asynchronous operation in which the recommendation system takes all the usage data and creates a recommendation model. This operation can take several minutes or several hours depending on the size of the data and the build configuration parameters. When triggering the build, you will get a build ID. Use it to check when the build process has ended before starting to consume recommendations.
- Consume recommendations - Get recommendations for a specific item or list of items.

All the steps above are done through the Azure Machine Learning Recommendations API. You can download a sample application that implements each of these steps from the [gallery as well](#).

Limitations

- The maximum number of models per subscription is 10.
- The maximum number of items that a catalog can hold is 100,000.
- The maximum number of usage points that are kept is ~5,000,000. The oldest will be deleted if new ones will be uploaded or reported.
- The maximum size of data that can be sent in POST (for example, import catalog data, import usage data) is 200MB.
- The number of transactions per second for a recommendation model build that is not active is ~2TPS. A recommendation model build that is active can hold up to 20TPS.

Integration

Authentication

Microsoft Azure Marketplace supports either the Basic or OAuth authentication method. You can easily find the account keys by navigating to the keys in the marketplace under [your account settings](#).

Basic Authentication

Add the Authorization header:

Authorization: Basic <creds>

Where <creds> = ConvertToBase64("AccountKey:" + yourAccountKey);

Convert to Base64 (C#)

```
var bytes = Encoding.UTF8.GetBytes("AccountKey:" + yourAccountKey);
var creds = Convert.ToBase64String(bytes);
```

Convert to Base64 (JavaScript)

```
var creds = window.btoa("AccountKey" + ":" + yourAccountKey);
```

Service URI

The service root URI for the Azure Machine Learning Recommendations APIs is [here](#).

The full service URI is expressed using elements of the OData specification.

API version

Each API call will have, at the end, a query parameter called apiVersion that should be set to "1.0".

IDs are case-sensitive

IDs, returned by any of the APIs, are case-sensitive and should be used as such when passed as parameters in subsequent API calls. For instance, model IDs and catalog IDs are case-sensitive.

Create a model

Creating a "create model" request:

HTTP METHOD	URI
POST	<rootURI>/CreateModel?modelName=%27<model_name>%27&apiVersion=%271.0%27

Example:

```
<rootURI>/CreateModel?modelName=%27MyFirstModel%27&apiVersion=%271.0%27
```

PARAMETER NAME	VALID VALUES
modelName	Only letters (A-Z, a-z), numbers (0-9), hyphens (-) and underscore (_) are allowed. Max length: 20
apiVersion	1.0
Request Body	NONE

Response:

HTTP Status code: 200

- <id> - Contains the model ID. Note that the model ID is case-sensitive.

OData XML

```

<feed xmlns:base="https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/CreateModel" xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
      xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns="http://www.w3.org/2005/Atom">
  <title type="text" />
  <subtitle type="text">Create A New Model</subtitle>
  <id>https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/CreateModel?modelName=MyFirstModel&amp;apiVersion=1.0</id>
  <rights type="text" />
  <updated>2014-10-05T06:35:21Z</updated>
  <link rel="self" href="https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/CreateModel?modelName=MyFirstModel&amp;apiVersion=1.0" />
  <entry>
    <id>https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/CreateModel?modelName=MyFirstModel&amp;apiVersion=1.0&skip=0&top=1</id>
    <title type="text">Create A New Model Entity 2</title>
    <updated>2014-10-05T06:35:21Z</updated>
    <link rel="self" href="https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/CreateModel?
modelName=MyFirstModel&amp;apiVersion=1.0&skip=0&top=1" />
    <content type="application/xml">
      <mproperties>
        <d:Id mtype="Edm.String">a658c626-2baa-43a7-ac98-f6ee26120a12</d:Id>
        <d:Name mtype="Edm.String">MyFirstModel</d:Name>
        <d:Date mtype="Edm.String">10/5/2014 6:35:19 AM</d:Date>
        <d>Status mtype="Edm.String">Created</d>Status>
        <d:HasActiveBuild mtype="Edm.String">false</d:HasActiveBuild>
        <d:BuildId mtype="Edm.String">-1</d:BuildId>
        <d:Mpr mtype="Edm.String">0</d:Mpr>
        <d:UserName mtype="Edm.String">5-4058-ab36-1fe254f05102@dm.com</d:UserName>
        <d>Description mtype="Edm.String"></d>Description>
      </mproperties>
    </content>
  </entry>
</feed>

```

Import catalog data

If you upload several catalog files to the same model with several calls, we will insert only the new catalog items. Existing items will remain with the original values.

HTTP METHOD	URI
POST	<rootURI>/ImportCatalogFile? modelId=%27<modelId>%27&filename=%27<fileName>%27&apiVersion=%271.0%27
	Example: <rootURI>/ImportCatalogFile?modelId=%27a658c626-2baa-43a7-ac98-f6ee26120a12%27&filename=%27catalog10_small.txt%27&apiVersion=%271.0%27
PARAMETER NAME	VALID VALUES
modelId	Unique identifier of the model (case-sensitive)
filename	Textual identifier of the catalog. Only letters (A-Z, a-z), numbers (0-9), hyphens (-) and underscore (_) are allowed. Max length: 50
apiVersion	1.0

PARAMETER NAME	VALID VALUES																				
Request Body	<p>Catalog data. Format: <code><Item Id><Item Name><Item Category><description></code></p> <table border="1"> <thead> <tr> <th>NAME</th><th>MANDATORY</th><th>TYPE</th><th>DESCRIPTION</th></tr> </thead> <tbody> <tr> <td>Item Id</td><td>Yes</td><td>Alphanumeric, max length 50</td><td>Unique identifier of an item</td></tr> <tr> <td>Item Name</td><td>Yes</td><td>Alphanumeric, max length 255</td><td>Item name</td></tr> <tr> <td>Item Category</td><td>Yes</td><td>Alphanumeric, max length 255</td><td>Category to which this item belongs (for example, Cooking Books, Drama...)</td></tr> <tr> <td>Description</td><td>No</td><td>Alphanumeric, max length 4000</td><td>Description of this item</td></tr> </tbody> </table> <p>Maximum file size is 200MB.</p> <p>Example:</p> <pre>2406e770-769c-4189-89de-1c9283f93a96,Clara Callan,Book 21bf8088-b6c0-4509-870c-e1c7ac78304a,The Forgetting Room: A Fiction (Byzantium Book),Book 3bb5cb44-d143-4bdd-a55c-443964bf4b23,Spadework,Book 552a1940-21c4-4399-82bb-594b46d7ed54,Restraint of Beasts,Book</pre>	NAME	MANDATORY	TYPE	DESCRIPTION	Item Id	Yes	Alphanumeric, max length 50	Unique identifier of an item	Item Name	Yes	Alphanumeric, max length 255	Item name	Item Category	Yes	Alphanumeric, max length 255	Category to which this item belongs (for example, Cooking Books, Drama...)	Description	No	Alphanumeric, max length 4000	Description of this item
NAME	MANDATORY	TYPE	DESCRIPTION																		
Item Id	Yes	Alphanumeric, max length 50	Unique identifier of an item																		
Item Name	Yes	Alphanumeric, max length 255	Item name																		
Item Category	Yes	Alphanumeric, max length 255	Category to which this item belongs (for example, Cooking Books, Drama...)																		
Description	No	Alphanumeric, max length 4000	Description of this item																		

Response:

HTTP Status code: 200

- Feed\entry\content\properties\LineCount - Number of lines accepted.
- Feed\entry\content\properties\ErrorCount - Number of lines that were not inserted due to an error.

OData XML

```
<feed xmlns:base="https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/ImportCatalogFile"
  xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices" xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
  xmlns="http://www.w3.org/2005/Atom">
  <title type="text" />
  <subtitle type="text">Import catalog file</subtitle>
  <id>https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/ImportCatalogFile?modelId=a658c626-2baa-43a7-ac98-f6ee26120a12&filename=catalog10_small.txt&apiVersion='1.0'</id>
  <rights type="text" />
  <updated>2014-10-05T06:58:04Z</updated>
  <link rel="self" href="https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/ImportCatalogFile?modelId=a658c626-2baa-43a7-ac98-f6ee26120a12&filename=catalog10_small.txt&apiVersion='1.0'" />
  <entry>
    <id>https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/ImportCatalogFile?modelId=a658c626-2baa-43a7-ac98-f6ee26120a12&filename=catalog10_small.txt&apiVersion='1.0'&$skip=0&$top=1</id>
    <title type="text">ImportCatalogFileEntity2</title>
    <updated>2014-10-05T06:58:04Z</updated>
    <link rel="self" href="https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/ImportCatalogFile?modelId=a658c626-2baa-43a7-ac98-f6ee26120a12&filename=catalog10_small.txt&apiVersion='1.0'&$skip=0&$top=1" />
    <content type="application/xml">
      <m:properties>
        <d:LineCount m:type="Edm.String">4</d:LineCount>
        <d:ErrorCount m:type="Edm.String">0</d:ErrorCount>
      </m:properties>
    </content>
  </entry>
</feed>
```

Import usage data

Uploading a file

This section shows how to upload usage data by using a file. You can call this API several times with usage data. All usage data will be saved for all calls.

HTTP METHOD	URI
POST	<rootURI>/ImportUsageFile? modelId=%27<modelId>%27&filename=%27<fileName>%27&apiVersion=%271.0%27
	Example: <rootURI>/ImportUsageFile?modelId=%27a658e626-2baa-43a7-ac98- f6cec26120a1%27&filename=%27ImplicitMatrix10_Guid_small.txt%27&apiVersion=%271.0%27
PARAMETER NAME	VALID VALUES
modelId	Unique identifier of the model (case-sensitive)
filename	Textual identifier of the catalog. Only letters (A-Z, a-z), numbers (0-9), hyphens (-) and underscore (_) are allowed. Max length: 50
apiVersion	1.0

PARAMETER NAME	VALID VALUES			
Request Body	Usage data. Format: [<UserId>,<Item Id>,<Time>,<Event>]			
	NAME	MANDATORY	TYPE	DESCRIPTION
	User Id	Yes	Alphanumeric	Unique identifier of a user
	Item Id	Yes	Alphanumeric, max length 50	Unique identifier of an item
	Time	No	Date in format: YYYY/MM/DDT HH:MM:SS (for example, 2013/06/20T10:00:00)	Time of data
	Event	No, if supplied then must also put date	One of the following: • Click • RecommendationClick • AddShopCart • RemoveShopCart • Purchase	

Maximum file size is 200MB.

Example:

```
149452,1b3d95e2-84e4-414c-bb38-be9cf461c347
6360,1b3d95e2-84e4-414c-bb38-be9cf461c347
50321,1b3d95e2-84e4-414c-bb38-be9cf461c347
71285,1b3d95e2-84e4-414c-bb38-be9cf461c347
224450,1b3d95e2-84e4-414c-bb38-be9cf461c347
236645,1b3d95e2-84e4-414c-bb38-be9cf461c347
107951,1b3d95e2-84e4-414c-bb38-be9cf461c347
```

Response:

HTTP Status code: 200

- `Feed.entry.content(properties)LineCount` - Number of lines accepted.
- `Feed.entry.content(properties)ErrorCount` - Number of lines that were not inserted due to an error.
- `Feed.entry.content(properties)FileId` - File identifier.

OData XML

```

<feed xmlns:base="https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/ImportUsageFile"
      xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices" xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
      xmlns="http://www.w3.org/2005/Atom">
  <title type="text" />
  <subtitle type="text">Add bulk usage data (usage file)</subtitle>
  <id>https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/ImportUsageFile?modelId=a658c626-2baa-43a7-ac98-f6ee26120a12&filename='ImplicitMatrix10_Guid_small.txt'&apiVersion=1.0</id>
  <rights type="text" />
  <updated>2014-10-05T07:21:44Z</updated>
  <link rel="self" href="https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/ImportUsageFile?modelId=a658c626-2baa-43a7-ac98-f6ee26120a12&filename='ImplicitMatrix10_Guid_small.txt'&apiVersion=1.0" />
  <entry>
    <id>https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/ImportUsageFile?modelId=a658c626-2baa-43a7-ac98-f6ee26120a12&filename='ImplicitMatrix10_Guid_small.txt'&apiVersion=1.0&skip=0&stop=1</id>
    <title type="text">AddBulkUsageDataUsageFileEntity2</title>
    <updated>2014-10-05T07:21:44Z</updated>
    <link rel="self" href="https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/ImportUsageFile?modelId=a658c626-2baa-43a7-ac98-f6ee26120a12&filename='ImplicitMatrix10_Guid_small.txt'&apiVersion=1.0&skip=0&stop=1" />
    <content type="application/xml">
      <mproperties>
        <d:LineCount mtype="Edm.String">33</d:LineCount>
        <d:ErrorCount mtype="Edm.String">0</d:ErrorCount>
        <d:FileId mtype="Edm.String">fead7c1c-db01-46c0-872f-65bcda36025d</d:FileId>
      </mproperties>
    </content>
  </entry>
</feed>

```

Using data acquisition

This section shows how to send events in real time to Azure Machine Learning Recommendations, usually from your website.

HTTP METHOD	URI
POST	<rootURL>/AddUsageEvent?apiVersion=%271.0%27-f6ee26120a12%27&filename=%27catalog10_small.txt%27&apiVersion=%271.0%27
PARAMETER NAME	VALID VALUES
apiVersion	1.0
Request body	Event data entry for each event you want to send. You should send for the same user or browser session the same ID in the SessionId field. (See sample of event body below.)

- Example for event 'Click':

```

<Event xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xi="http://www.w3.org/2001/XMLSchema-instance">
  <ModelId>2779c063-48fb-46c1-bae3-74acddc8c1d1</ModelId>
  <SessionId>11112222</SessionId>
  <EventData>
    <EventData>
      <Name>Click</Name>
      <ItemId>21BF8088-B6C0-4509-870C-E1C7AC78304A</ItemId>
    </EventData>
  </EventData>
</Event>

```

- Example for event 'RecommendationClick':

```

<Event xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xi="http://www.w3.org/2001/XMLSchema-instance">
  <ModelId>2779c063-48fb-46c1-bae3-74acddc8c1d1</ModelId>
  <SessionId>11112222</SessionId>
  <EventData>
    <EventData>
      <Name>RecommendationClick</Name>
      <ItemId>21BF8088-B6C0-4509-870C-E1C7AC78304A</ItemId>
    </EventData>
  </EventData>
</Event>

```

- Example for event 'AddShopCart':

```
<Event xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xi="http://www.w3.org/2001/XMLSchema-instance">
<ModelId>2779c063-48fb-46c1-bae3-74acddc8c1d1</ModelId>
<SessionId>11112222</SessionId>
<EventData>
<Name>AddShopCart</Name>
<ItemId>21BF8088-B6C0-4509-870C-E1C7AC78304A</ItemId>
</EventData>
</EventData>
</Event>
```

- Example for event 'RemoveShopCart':

```
<Event xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xi="http://www.w3.org/2001/XMLSchema-instance">
<ModelId>2779c063-48fb-46c1-bae3-74acddc8c1d1</ModelId>
<SessionId>11112222</SessionId>
<EventData>
<Name>RemoveShopCart</Name>
<ItemId>21BF8088-B6C0-4509-870C-E1C7AC78304A</ItemId>
</EventData>
</EventData>
</Event>
```

- Example for event 'Purchase':

```
<Event xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xi="http://www.w3.org/2001/XMLSchema-instance">
<ModelId>2779c063-48fb-46c1-bae3-74acddc8c1d1</ModelId>
<SessionId>11112222</SessionId>
<EventData>
<EventData>
<Name>Purchase</Name>
<PurchaseItems>
<PurchaseItems>
<ItemId>21BF8088-B6C0-4509-870C-E1C7AC78304A</ItemId>
<Count>3</Count>
</PurchaseItems>
</PurchaseItems>
</EventData>
</EventData>
</Event>
```

- Example sending 2 events, 'Click' and 'AddShopCart':

```
<Event xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xi="http://www.w3.org/2001/XMLSchema-instance">
<ModelId>2779c063-48fb-46c1-bae3-74acddc8c1d1</ModelId>
<SessionId>11112222</SessionId>
<EventData>
<EventData>
<Name>Click</Name>
<ItemId>21BF8088-B6C0-4509-870C-E1C7AC78304A</ItemId>
<ItemName>itemName</ItemName>
<ItemDescription>item description</ItemDescription>
<ItemCategory>category</ItemCategory>
</EventData>
<EventData>
<Name>AddShopCart</Name>
<ItemId>552A1940-21E4-4399-82BB-594B46D7ED54</ItemId>
</EventData>
</Event>
```

Response: HTTP Status code: 200

Build a recommendation model

HTTP METHOD	URI

HTTP METHOD	URI
POST	<pre><rootURI>/BuildModel? (modelId=%27<modelId%27&userDescription=%27<description%27&apiVersion=%271.0%27</pre> <p>Example:</p> <pre><rootURI>/BuildModel?modelId=%27a658c626-2baa-43a7-ac98- f6ee26120a1%27&userDescription=%27First%20build%27&apiVersion=%271.0%27</pre>

PARAMETER NAME	VALID VALUES
modelId	Unique identifier of the model (case-sensitive)
userDescription	Textual identifier of the catalog. Note that if you use spaces you must encode it with %20 instead. See example above. Max length: 50
apiVersion	1.0
Request Body	NONE

Response:

HTTP Status code: 200

This is an asynchronous API. You will get a build ID as a response. To know when the build has ended, you should call the "Get Builds Status of a Model" API and locate this build ID in the response. Note that a build can take from minutes to hours depending on the size of the data.

You cannot consume recommendations till the build ends.

Valid build status:

- Create – Model was created.
- Queued – Model build was triggered and it is queued.
- Building – Model is being built.
- Success – Build ended successfully.
- Error – Build ended with a failure.
- Canceled – Build was canceled.
- Canceling – Build is being canceled.

Note that the build ID can be found under the following path: `/FeedEntry/content/properties/Id`

OData XML

```

<feed xmlns:base="https://api.datamarket.azure.com/Data.ashx/amla/recommendations/v2/BuildModel" xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices" 
xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns="http://www.w3.org/2005/Atom">
<title type="text" />
<subtitle type="text">Build a Model with RequestBody</subtitle>
<id>https://api.datamarket.azure.com/Data.ashx/amla/recommendations/v2/BuildModel?modelId=9559872f-7a53-4076-a3c7-19d9385c1265&amp;userDescription=First build&amp;apiVersion=1.0</id>
<rights type="text" />
<updated>2014-10-05T08:56:34Z</updated>
<link rel="self" href="https://api.datamarket.azure.com/Data.ashx/amla/recommendations/v2/BuildModel?modelId=9559872f-7a53-4076-a3c7-19d9385c1265&amp;userDescription=First%20build&amp;apiVersion=1.0" />
<entry>
<id>https://api.datamarket.azure.com/Data.ashx/amla/recommendations/v2/BuildModel?modelId=9559872f-7a53-4076-a3c7-19d9385c1265&amp;userDescription=First build&amp;apiVersion=1.0&amp;$skip=0&amp;$top=1</id>
<title type="text">BuildAModelEntity2</title>
<updated>2014-10-05T08:56:34Z</updated>
<link rel="self" href="https://api.datamarket.azure.com/Data.ashx/amla/recommendations/v2/BuildModel?modelId=9559872f-7a53-4076-a3c7-19d9385c1265&amp;userDescription=First%20build&amp;apiVersion=1.0&amp;$skip=0&amp;$top=1" />
<content type="application/xml">
<mproperties>
<d:Id mtype="Edm.String">1000272</d:Id>
<d:UserName mtype="Edm.String"></d:UserName>
<d:ModelId mtype="Edm.String">9559872f-7a53-4076-a3c7-19d9385c1265</d:ModelId>
<d:ModelName mtype="Edm.String">docTest</d:ModelName>
<d>Type mtype="Edm.String">Recommendation</d>Type>
<d:CreationTime mtype="Edm.String">2014-10-05T08:56:31.893</d:CreationTime>
<d:Progress_BuildId mtype="Edm.String">1000272</d:Progress_BuildId>
<d:Progress_ModelId mtype="Edm.String">9559872f-7a53-4076-a3c7-19d9385c1265</d:Progress_ModelId>
<d:Progress_UserName mtype="Edm.String">5-4058-ab36-1fe254f05102@dm.com</d:Progress_UserName>
<d:Progress_IsExecutionStarted mtype="Edm.String">false</d:Progress_IsExecutionStarted>
<d:Progress_IsExecutionEnded mtype="Edm.String">false</d:Progress_IsExecutionEnded>
<d:Progress_Percent mtype="Edm.String">0</d:Progress_Percent>
<d:Progress_StartTime mtype="Edm.String">2014-10-05T08:56:31.893</d:Progress_StartTime>
<d:Progress_EndTime mtype="Edm.String">2014-10-05T08:56:31.893</d:Progress_EndTime>
<d:Progress_UpdateDateUTC mtype="Edm.String"></d:Progress_UpdateDateUTC>
<d>Status mtype="Edm.String">Queued</d>Status>
<d:Key1 mtype="Edm.String">UseFeaturesInModel</d:Key1>
<d:Value1 mtype="Edm.String">False</d:Value1>
</mproperties>
<content>
</content>
</entry>
</feed>

```

Get build status of a model

HTTP METHOD	URI
GET	<root URI>/GetModelBuildsStatus?modelId=%27<modelId%27&onlyLastBuild=%27<onlyLastBuild%27&apiVersion=%271.0%27

Example:
<root URI>/GetModelBuildsStatus?modelId=9559872f-7a53-4076-a3c7-19d9385c1265&onlyLastBuild=true&apiVersion=1.0

PARAMETER NAME	VALID VALUES
modelId	Unique identifier of the model (case-sensitive)
onlyLastBuild	Indicates whether to return all the build history of the model or only the status of the most recent build.
apiVersion	1.0

Response:

HTTP Status code: 200

The response includes one entry per build. Each entry has the following data:

- feed/entry/content/properties/UserName – Name of the user.
- feed/entry/content/properties/ModelName – Name of the model.
- feed/entry/content/properties/ModelId – Model unique identifier.
- feed/entry/content/properties/isDeployed – Whether the build is deployed (a.k.a. active build).

- `feed/entry/content/properties/BuildId` – Build unique identifier.
- `feed/entry/content/properties/BuildType` – Type of the build.
- `feed/entry/content/properties/Status` – Build status. Can be one of the following: Error, Building, Queued, Canceling, Canceled, Success
- `feed/entry/content/properties/StatusMessage` – Detailed status message (applies only to specific states).
- `feed/entry/content/properties/Progress` – Build progress (%).
- `feed/entry/content/properties/StartTime` – Build start time.
- `feed/entry/content/properties/EndTime` – Build end time.
- `feed/entry/content/properties/ExecutionTime` – Build duration.
- `feed/entry/content/properties/ProgressStep` – Details about the current stage that a build in progress is in.

Valid build status:

- Created – Build request entry was created.
- Queued – Build request was triggered and it is queued.
- Building – Build is in process.
- Success – Build ended successfully.
- Error – Build ended with a failure.
- Canceled – Build was canceled.
- Canceling – Build is being canceled.

Valid values for build type:

- Rank - Rank build. (For rank build details, please refer to the "Machine Learning Recommendation API documentation" document.)
- Recommendation - Recommendation build.
- Fbt - Frequently bought together build.

OData XML

```
<feed xmlns:base="https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/GetModelBuildsStatus"
      xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices" xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
      xmlns="http://www.w3.org/2005/Atom">
  <title type="text" />
  <subtitle type="text">Get builds status of a model</subtitle>
  <id>https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/GetModelBuildsStatus?modelId='1d20c34f-dca1-4eac-8e5d-f299e4e4ad66'&onlyLastBuild=False&apiVersion=1.0'</id>
  <rights type="text" />
  <updated>2014-11-05T17:51:10Z</updated>
  <link rel="self" href="https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/GetModelBuildsStatus?modelId='1d20c34f-dca1-4eac-8e5d-f299e4e4ad66'&onlyLastBuild=False&apiVersion=1.0" />
  <entry>
    <id>https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/GetModelBuildsStatus?modelId='1d20c34f-dca1-4eac-8e5d-f299e4e4ad66'&onlyLastBuild=False&apiVersion=1.0'&skip=0&stop=1</id>
    <title type="text">GetBuilds Status Entity</title>
    <updated>2014-11-05T17:51:10Z</updated>
    <link rel="self" href="https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/GetModelBuildsStatus?modelId='1d20c34f-dca1-4eac-8e5d-f299e4e4ad66'&onlyLastBuild=False&apiVersion=1.0'&skip=0&stop=1" />
    <content type="application/xml">
      <m:properties>
        <d:UserName m:type="Edm.String">b-434e-b2c9-84935664ff20@dm.com</d:UserName>
        <d:ModelName m:type="Edm.String">modelName</d:ModelName>
        <d:ModelId m:type="Edm.String">1d20c34f-dca1-4eac-8e5d-f299e4e4ad66</d:ModelId>
        <d:IsDeployed m:type="Edm.String">true</d:IsDeployed>
        <d:BuildId m:type="Edm.String">1000272</d:BuildId>
        <d:BuildType m:type="Edm.String">Recommendation</d:BuildType>
        <d>Status m:type="Edm.String">Success</d>Status>
        <d>StatusMessage m:type="Edm.String"></d>StatusMessage>
        <d:Progress m:type="Edm.String">0</d:Progress>
        <d:StartTime m:type="Edm.String">2014-11-02T13:43:51</d:StartTime>
        <d:EndTime m:type="Edm.String">2014-11-02T13:45:10</d:EndTime>
        <d:ExecutionTime m:type="Edm.String">00:01:19</d:ExecutionTime>
        <d:IsExecutionStarted m:type="Edm.String">false</d:IsExecutionStarted>
        <d:ProgressStep m:type="Edm.String"></d:ProgressStep>
      </m:properties>
    </content>
  </entry>
</feed>
```

[Get recommendations](#)

HTTP METHOD	URI
GET	<rootURI>/ItemRecommend? modelId=%27<modelId>%27&itemIds=%27<itemId>%27&numberOfResults=% <int>&includeMetadata=<bool>&apiVersion=%271.0%27
	Example: <rootURI>/ItemRecommend?modelId=%272779c063-48fb-46c1-bae3- 74acddc8c1d1%27&itemIds=%271003%27&numberOfResults=10&includeMetadata=false&apiVersion=1.0
PARAMETER NAME	VALID VALUES
modelId	Unique identifier of the model (case-sensitive)
itemIds	Comma-separated list of the items to recommend for. Max length: 1024
numberOfResults	Number of required results
includeMetadata	Future use, always false
apiVersion	1.0

Response:

HTTP Status code: 200

The response includes one entry per recommended item. Each entry has the following data:

- Feed.entry.properties.Id - Recommended item ID.
- Feed.entry.properties.Name - Name of the item.
- Feed.entry.properties.Rating - Rating of the recommendation; higher number means higher confidence.
- Feed.entry.properties.Reasoning - Recommendation reasoning (for example, recommendation explanations).

OData XML

The example response below includes 10 recommended items:

```

<feed xmlns:base="https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/ItemRecommend"
      xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices" xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
      xmlns="http://www.w3.org/2005/Atom">
  <title type="text" />
  <subtitle type="text">Get Recommendation</subtitle>
  <id>https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/ItemRecommend?modelId=2779c063-48fb-46c1-bae3-  
74acddc8c1d1'&itemIds='1003'&numberOfResults=10&includeMetadata=false&apiVersion=1.0</id>
  <rights type="text" />
  <updated>2014-10-05T12:28:48Z</updated>
  <link rel="self" href="https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/ItemRecommend?modelId=2779c063-48fb-46c1-bae3-  
74acddc8c1d1'&itemIds='1003'&numberOfResults=10&includeMetadata=false&apiVersion=1.0" />
  <entry>
    <id>https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/ItemRecommend?modelId=2779c063-48fb-46c1-bae3-  
74acddc8c1d1'&itemIds='1003'&numberOfResults=10&includeMetadata=false&apiVersion=1.0&$skip=0&$top=1</id>
    <title type="text">GetRecommendationEntity</title>
    <updated>2014-10-05T12:28:48Z</updated>
    <link rel="self" href="https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/ItemRecommend?modelId=2779c063-48fb-46c1-bae3-  
74acddc8c1d1'&itemIds='1003'&numberOfResults=10&includeMetadata=false&apiVersion=1.0&$skip=0&$top=1" />
    <content type="application/xml">
      <mproperties>
        <d:Id mtype="Edm.String">159</d:Id>
        <d:Name mtype="Edm.String">159</d:Name>
        <d:Rating mtype="Edm.Double">0.543343480387708</d:Rating>
        <d:Reasoning mtype="Edm.String">People who like '1003' also like '159'</d:Reasoning>
      </mproperties>
    </content>
  </entry>
  <entry>
    <id>https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/ItemRecommend?modelId=2779c063-48fb-46c1-bae3-  
74acddc8c1d1'&itemIds='1003'&numberOfResults=10&includeMetadata=false&apiVersion=1.0&$skip=1&$top=1</id>
    <title type="text">GetRecommendationEntity</title>
    <updated>2014-10-05T12:28:48Z</updated>
    <link rel="self" href="https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/ItemRecommend?modelId=2779c063-48fb-46c1-bae3-  
74acddc8c1d1'&itemIds='1003'&numberOfResults=10&includeMetadata=false&apiVersion=1.0&$skip=1&$top=1" />
  </entry>
</feed>
```

```

<content type="application/xml">
<mproperties>
<d:Id mtype="Edm.String">52</d:Id>
<d:Name mtype="Edm.String">52</d:Name>
<d:Rating mtype="Edm.Double">0.539588900748721</d:Rating>
<d:Reasoning mtype="Edm.String">People who like '1003' also like '52'</d:Reasoning>
</mproperties>
</content>
</entry>
</entry>
<id>https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/ItemRecommend?modelId=2779c063-48fb-46c1-bae3-74acddc8c1d1&itemIds='1003'&numberOfResults=10&includeMetadata=false&apiVersion=1.0&$skip=2&$top=1</id>
<title type="text">GetRecommendationEntity</title>
<updated>2014-10-05T12:28:48Z</updated>
<link rel="self" href="https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/ItemRecommend?modelId=2779c063-48fb-46c1-bae3-74acddc8c1d1&itemIds='1003'&numberOfResults=10&includeMetadata=false&apiVersion=1.0&$skip=2&$top=1" />
<content type="application/xml">
<mproperties>
<d:Id mtype="Edm.String">35</d:Id>
<d:Name mtype="Edm.String">35</d:Name>
<d:Rating mtype="Edm.Double">0.53842946443853</d:Rating>
<d:Reasoning mtype="Edm.String">People who like '1003' also like '35'</d:Reasoning>
</mproperties>
</content>
</entry>
</entry>
<id>https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/ItemRecommend?modelId=2779c063-48fb-46c1-bae3-74acddc8c1d1&itemIds='1003'&numberOfResults=10&includeMetadata=false&apiVersion=1.0&$skip=3&$top=1</id>
<title type="text">GetRecommendationEntity</title>
<updated>2014-10-05T12:28:48Z</updated>
<link rel="self" href="https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/ItemRecommend?modelId=2779c063-48fb-46c1-bae3-74acddc8c1d1&itemIds='1003'&numberOfResults=10&includeMetadata=false&apiVersion=1.0&$skip=3&$top=1" />
<content type="application/xml">
<mproperties>
<d:Id mtype="Edm.String">124</d:Id>
<d:Name mtype="Edm.String">124</d:Name>
<d:Rating mtype="Edm.Double">0.536712832792886</d:Rating>
<d:Reasoning mtype="Edm.String">People who like '1003' also like '124'</d:Reasoning>
</mproperties>
</content>
</entry>
</entry>
<id>https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/ItemRecommend?modelId=2779c063-48fb-46c1-bae3-74acddc8c1d1&itemIds='1003'&numberOfResults=10&includeMetadata=false&apiVersion=1.0&$skip=4&$top=1</id>
<title type="text">GetRecommendationEntity</title>
<updated>2014-10-05T12:28:48Z</updated>
<link rel="self" href="https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/ItemRecommend?modelId=2779c063-48fb-46c1-bae3-74acddc8c1d1&itemIds='1003'&numberOfResults=10&includeMetadata=false&apiVersion=1.0&$skip=4&$top=1" />
<content type="application/xml">
<mproperties>
<d:Id mtype="Edm.String">120</d:Id>
<d:Name mtype="Edm.String">120</d:Name>
<d:Rating mtype="Edm.Double">0.533673023762878</d:Rating>
<d:Reasoning mtype="Edm.String">People who like '1003' also like '120'</d:Reasoning>
</mproperties>
</content>
</entry>
</entry>
<id>https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/ItemRecommend?modelId=2779c063-48fb-46c1-bae3-74acddc8c1d1&itemIds='1003'&numberOfResults=10&includeMetadata=false&apiVersion=1.0&$skip=5&$top=1</id>
<title type="text">GetRecommendationEntity</title>
<updated>2014-10-05T12:28:48Z</updated>
<link rel="self" href="https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/ItemRecommend?modelId=2779c063-48fb-46c1-bae3-74acddc8c1d1&itemIds='1003'&numberOfResults=10&includeMetadata=false&apiVersion=1.0&$skip=5&$top=1" />
<content type="application/xml">
<mproperties>
<d:Id mtype="Edm.String">96</d:Id>
<d:Name mtype="Edm.String">96</d:Name>
<d:Rating mtype="Edm.Double">0.532544826370521</d:Rating>
<d:Reasoning mtype="Edm.String">People who like '1003' also like '96'</d:Reasoning>
</mproperties>
</content>
</entry>
</entry>
<id>https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/ItemRecommend?modelId=2779c063-48fb-46c1-bae3-74acddc8c1d1&itemIds='1003'&numberOfResults=10&includeMetadata=false&apiVersion=1.0&$skip=6&$top=1</id>
<title type="text">GetRecommendationEntity</title>
<updated>2014-10-05T12:28:48Z</updated>
<link rel="self" href="https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/ItemRecommend?modelId=2779c063-48fb-46c1-bae3-74acddc8c1d1&itemIds='1003'&numberOfResults=10&includeMetadata=false&apiVersion=1.0&$skip=6&$top=1" />

```

```

<content type="application/xml">
<mproperties>
<d:Id mtype="Edm.String">69</d:Id>
<d:Name mtype="Edm.String">69</d:Name>
<d:Rating mtype="Edm.Double">0.531678607847759</d:Rating>
<d:Reasoning mtype="Edm.String">People who like '1003' also like '69'</d:Reasoning>
</mproperties>
</content>
</entry>
</entry>
<id>https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/ItemRecommend?modelId=2779c063-48fb-46c1-bae3-74acddc8c1d1&itemIds='1003'&numberResults=10&includeMetadata=false&apiVersion=1.0&$skip=7&$top=1</id>
<title type="text">GetRecommendationEntity</title>
<updated>2014-10-05T12:28:48Z</updated>
<link rel="self" href="https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/ItemRecommend?modelId=2779c063-48fb-46c1-bae3-74acddc8c1d1&itemIds='1003'&numberResults=10&includeMetadata=false&apiVersion=1.0&$skip=7&$top=1" />
<content type="application/xml">
<mproperties>
<d:Id mtype="Edm.String">172</d:Id>
<d:Name mtype="Edm.String">172</d:Name>
<d:Rating mtype="Edm.Double">0.530957821375951</d:Rating>
<d:Reasoning mtype="Edm.String">People who like '1003' also like '172'</d:Reasoning>
</mproperties>
</content>
</entry>
</entry>
<id>https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/ItemRecommend?modelId=2779c063-48fb-46c1-bae3-74acddc8c1d1&itemIds='1003'&numberResults=10&includeMetadata=false&apiVersion=1.0&$skip=8&$top=1</id>
<title type="text">GetRecommendationEntity</title>
<updated>2014-10-05T12:28:48Z</updated>
<link rel="self" href="https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/ItemRecommend?modelId=2779c063-48fb-46c1-bae3-74acddc8c1d1&itemIds='1003'&numberResults=10&includeMetadata=false&apiVersion=1.0&$skip=8&$top=1" />
<content type="application/xml">
<mproperties>
<d:Id mtype="Edm.String">155</d:Id>
<d:Name mtype="Edm.String">155</d:Name>
<d:Rating mtype="Edm.Double">0.529093541481333</d:Rating>
<d:Reasoning mtype="Edm.String">People who like '1003' also like '155'</d:Reasoning>
</mproperties>
</content>
</entry>
</entry>
<id>https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/ItemRecommend?modelId=2779c063-48fb-46c1-bae3-74acddc8c1d1&itemIds='1003'&numberResults=10&includeMetadata=false&apiVersion=1.0&$skip=9&$top=1</id>
<title type="text">GetRecommendationEntity</title>
<updated>2014-10-05T12:28:48Z</updated>
<link rel="self" href="https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/ItemRecommend?modelId=2779c063-48fb-46c1-bae3-74acddc8c1d1&itemIds='1003'&numberResults=10&includeMetadata=false&apiVersion=1.0&$skip=9&$top=1" />
<content type="application/xml">
<mproperties>
<d:Id mtype="Edm.String">32</d:Id>
<d:Name mtype="Edm.String">32</d:Name>
<d:Rating mtype="Edm.Double">0.528917978168322</d:Rating>
<d:Reasoning mtype="Edm.String">People who like '1003' also like '32'</d:Reasoning>
</mproperties>
</content>
</entry>
</feed>

```

Update model

You can update the model description or the active build ID. *Active build ID* - Every build for every model has a build ID. The active build ID is the first successful build of every new model. Once you have an active build ID and you do additional builds for the same model, you need to explicitly set it as the default build ID if you want to. When you consume recommendations, if you do not specify the build ID that you want to use, the default one will be used automatically.

This mechanism enables you - once you have a recommendation model in production - to build new models and test them before you promote them to production.

HTTP METHOD	URI
PUT	<rootURI>/UpdateModel?id=%27[modelId]&apiVersion=%271.0%27

Example:
 <rootURI>/UpdateModel?id=%279559872f7a53-4076-a3c7-19d9385e1265%27&apiVersion=%271.0%27

PARAMETER NAME	VALID VALUES
id	Unique identifier of the model (case-sensitive)
apiVersion	1.0
Request Body	<pre><ModelUpdateParams xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> <Description>New Description</Description> <ActiveBuildId>1</ActiveBuildId> </ModelUpdateParams></pre> <p>Note that the XML tags Description and ActiveBuildId are optional. If you do not want to set Description or ActiveBuildId, remove the entire tag.</p>

Response:

HTTP Status code: 200

OData XML

```
<feed xmlns:base="https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/UpdateModel" xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
      xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns="http://www.w3.org/2005/Atom">
    <title type="text" />
    <subtitle type="text">Update an Existing Model</subtitle>
    <id>https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/UpdateModel?id=9559872f-7a53-4076-a3c7-19d9385c1265&amp;apiVersion=1.0</id>
    <rights type="text" />
    <updated>2014-10-05T10:27:17Z</updated>
    <link rel="self" href="https://api.datamarket.azure.com/Data.ashx/aml/recommendations/v2/UpdateModel?id=9559872f-7a53-4076-a3c7-19d9385c1265&amp;apiVersion=1.0" />
</feed>
```

Legal

This document is provided "as-is". Information and views expressed in this document, including URL and other Internet website references, may change without notice. Some examples depicted herein are provided for illustration only and are fictitious. No real association or connection is intended or should be inferred. This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes. © 2014 Microsoft. All rights reserved.

Azure Machine Learning Recommendations API Documentation

1/17/2017 • 60 min to read • [Edit on GitHub](#)

NOTE

You should start using the Recommendations API Cognitive Service instead of this version. The Recommendations Cognitive Service will be replacing this service, and all the new features will be developed there. It has new capabilities like batching support, a better API Explorer, a cleaner API surface, more consistent signup/billing experience, etc. Learn more about [Migrating to the new Cognitive Service](#)

This document depicts Microsoft Azure Machine Learning Recommendations APIs.

NOTE

[Try Azure Machine Learning for free](#)

No credit card or Azure subscription needed. [Get started now >](#)

1. General overview

This document is an API reference. You should start with the "Azure Machine Learning Recommendation - Quick Start" document.

The Azure Machine Learning Recommendations API can be divided into the following logical groups:

- [Limitations](#) - Recommendations API limitations.
- [General Information](#) - Information on authentication, service URI and versioning.
- [Model Basic](#) - APIs that enable you to do the basic operations on model (e.g. create, update and delete a model).
- [Model Advanced](#) - APIs that enable you to get advanced data insights on the model.
- [Model Business Rules](#) - APIs that enable you to manage business rules on the model recommendation results.
- [Catalog](#) - APIs that enable you to do basic operations on a model catalog. A catalog contains metadata information on the items of the usage data.
- [Feature](#) - APIs that enable to get insights on item into the catalog and how to use this information to build better recommendations.
- [Usage Data](#) - APIs that enable you to do basic operations on the model usage data. Usage data in the basic form consists of rows that include pairs of <userId>,<itemId>.
- [Build](#) - APIs that enable you to trigger a model build and do basic operations that are related to this build. You can trigger a model build once you have valuable usage data.
- [Recommendation](#) - APIs that enable you to consume recommendations once the build of a model ends.
- [User Data](#) - APIs that enable you to fetch information on the user usage data.
- [Notifications](#) - APIs that enable you to receive notifications on problems related to your API operations. (For example, you are reporting usage data via Data Acquisition and most of the events processing are failing. An error notification will be raised.)

2. Limitations

- The maximum number of models per subscription is 10.
- The maximum number of builds per model is 20.
- The maximum number of items that a catalog can hold is 100,000.
- The maximum number of usage points that are kept is ~5,000,000. The oldest will be deleted if new ones will be uploaded or reported.
- The maximum size of data that can be sent in POST (e.g. import catalog data, import usage data) is 200MB.
- The maximum number of items that can be asked for when getting recommendations is 150.

3. APIs - general information

3.1. Authentication

Please follow the Microsoft Azure Marketplace guidelines regarding authentication. The marketplace supports either the Basic or OAuth authentication method.

3.2. Service URI

The service root URI for the Azure Machine Learning Recommendations APIs is [here](#).

The full service URI is expressed using elements of the OData specification.

3.3. API version

Each API call will have, at the end, a query parameter called apiVersion that should be set to 1.0.

3.4. IDs are case sensitive

IDs, returned by any of the APIs, are case sensitive and should be used as such when passed as parameters in subsequent API calls. For instance, model IDs and catalog IDs are case sensitive.

4. Recommendations Quality and Cold Items

4.1. Recommendation quality

Creating a recommendation model is usually enough to allow the system to provide recommendations. Nevertheless, recommendation quality varies based on the usage processed and the coverage of the catalog. For example if you have a lot of cold items (items without significant usage), the system will have difficulties providing a recommendation for such an item or using such an item as a recommended one. In order to overcome the cold item problem, the system allows the use of metadata of the items to enhance the recommendations. This metadata is referred to as features. Typical features are a book's author or a movie's actor. Features are provided via the catalog in the form of key/value strings. For the full format of the catalog file, please refer to the [import catalog section](#).

4.2. Rank build

Features can enhance the recommendation model, but to do so requires the use of meaningful features. For this purpose a new build was introduced - a rank build. This build will rank the usefulness of features. A meaningful feature is a feature with a rank score of 2 and up. After understanding which of the features are meaningful, trigger a recommendation build with the list (or sublist) of meaningful features. It is possible to use these features for the enhancement of both warm items and cold items. In order to use them for warm items, the `UseFeatureModel` build parameter should be set up. In order to use features for cold items, the `AllowColdItemPlacement` build parameter should be enabled. Note: It is not possible to enable `AllowColdItemPlacement` without enabling `UseFeatureModel`.

4.3. Recommendation reasoning

Recommendation reasoning is another aspect of feature usage. Indeed, the Azure Machine Learning Recommendations engine can use features to provide recommendation explanations (a.k.a. reasoning), leading to more confidence in the recommended item from the recommendation consumer. To enable reasoning, the `?showFeatureExplanations` and `?returningFeatureList` parameters should be setup prior to requesting a recommendation build.

5. Model Basic

5.1. Create Model

Creates a "create model" request.

HTTP METHOD	URI
POST	<code><rootURI>/CreateModel?modelName=%27MyFirstModel%27&apiVersion=%271.0%27</code> Example: <code><rootURI>/CreateModel?modelName=%27MyFirstModel%27&apiVersion=%271.0%27</code>
PARAMETER NAME	VALID VALUES
modelName	Only letters (A-Z, a-z), numbers (0-9), hyphens (-) and underscore (_) are allowed. Max length: 20
apiVersion	1.0
Request Body	NONE

Response:

HTTP Status code: 200

- `<entry/content/properties/id` - Contains the model ID. **Note:** model ID is case sensitive.

OData XML

```
<feed xmlns:base="https://api.datamarket.azure.com/aml/recommendations/v3/CreateModel" xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices" xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns:ns1="http://www.w3.org/2005/Atom">
<title type="text" />
<subtitle type="text">Create A New Model</subtitle>
<d:https://api.datamarket.azure.com/aml/recommendations/v3/CreateModel?modelName=MyFirstModel&apiVersion=1.0>/id>
<rights type="text" />
<updated>2014-10-05T06:35:21Z</updated>
<link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/CreateModel?modelName=MyFirstModel&apiVersion=1.0" />
<entry>
<id href="https://api.datamarket.azure.com/aml/recommendations/v3/CreateModel?modelName=MyFirstModel&apiVersion=1.0&skip=0&stop=1">/id>
<title type="text">CreateA NewModelEntity2</title>
<updated>2014-10-05T06:35:21Z</updated>
<link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/CreateModel?modelName=MyFirstModel&apiVersion=1.0&skip=0&stop=1" />
<content type="application/xml">
<m:properties>
<d:id mtype="Edm.String">a658c626-2baa-43a7-ac98-f6ee26120a12</d:id>
<d:name mtype="Edm.String">MyFirstModel</d:name>
<d:date mtype="Edm.String">10/5/2014 6:35:19 AM</d:date>
<d:status mtype="Edm.String">Created</d:status>
<d:hasActiveBuild mtype="Edm.String">false</d:hasActiveBuild>
<d:buildId mtype="Edm.String">1</d:buildId>
<d:mpr mtype="Edm.String">0</d:mpr>
<d:userName mtype="Edm.String">54-058-ab36-1fc254f05102@dm.com</d:userName>
<d:description mtype="Edm.String"></d:description>
</m:properties>
<content>
</entry>
</feed>
```

5.2. Get Model

Creates a "get model" request.

HTTP METHOD	URI
GET	<code><rootURI>/GetModel?id=%27<model_id>%27&apiVersion=%271.0%27</code> Example: <code><rootURI>/GetModel?id=%271caefb76-dcf4-4111-be81-29b806a6fb1d%27&apiVersion=%271.0%27</code>
PARAMETER NAME	VALID VALUES
id	Unique identifier of the model (case sensitive)
apiVersion	1.0
Request Body	NONE

Response:

HTTP Status code: 200

The model data can be found under the following elements:

- `<entry/content/properties/id` - Model unique ID.
- `<entry/content/properties/Name` - Model name.
- `<entry/content/properties/Date` - Model creation date.
- `<entry/content/properties>Status` - Model status. One of the following:
 - Created - Model is created and does not contain Catalog and Usage.

- ReadyForBuild - Model is created and contains Catalog and Usage.
- `/feed/entry/content/properties/IsActiveBuild` - Indicates if the model was built successfully.
- `/feed/entry/content/properties/BuilderId` - Model active build ID.
- `/feed/entry/content/properties/Mpr` - Model mean percentile ranking (MPR - see ModelInsight for more information).
- `/feed/entry/content/properties/UserName` - Model internal user name.

OData XML

```
<feed xmlns:base="https://api.datamarket.azure.com/aml/recommendations/v3/GetAllModels" xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices" xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns="http://www.w3.org/2005/Atom">
<title type="text" />
<subtitle type="text">Get A List Of All Models</subtitle>
<id>https://api.datamarket.azure.com/aml/recommendations/v3/GetAllModels?apiVersion=1.0</id>
<rights type="text" />
<updated>2014-10-28T14:35:51Z</updated>
<link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetAllModels?apiVersion=1.0" />
<entry>
<id>https://api.datamarket.azure.com/aml/recommendations/v3/GetAllModels?apiVersion=1.0&&$skip=0&&$top=1</id>
<title type="text">GetAListOfAllModelsEntity</title>
<updated>2014-10-28T14:35:51Z</updated>
<link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetAllModels?apiVersion=1.0&&$skip=0&&$top=1" />
<content type="application/xml">
<m:properties>
<d:Id m:type="Edm.String">68b232f2-1037-45f7-8f49-af822695ee63</d:Id>
<d:Name m:type="Edm.String">vah-11111</d:Name>
<d:Date m:type="Edm.String">10/28/2014 2:16:07 PM</d:Date>
<d>Status m:type="Edm.String">ReadyForBuild</d>Status>
<d:HasActiveBuild m:type="Edm.String">false</d:HasActiveBuild>
<d:BuildId m:type="Edm.String">1</d:BuildId>
<d:Mpr m:type="Edm.String">0</d:Mpr>
<d:UsageFileNames m:type="Edm.String">ImplicitMatrix0_Guid_small.txt, ImplicitMatrix1_Guid_small.txt</d:UsageFileNames>
<d:CatalogId m:type="Edm.String">626babdb-cab6-43a6-82ef-4fb86345700c</d:CatalogId>
<d:UserName m:type="Edm.String">89dc4722-03ba-4f90-8821-b1db388408b5@dm.com</d:UserName>
<d>Description m:type="Edm.String">short description</d>Description>
<d:CatalogFileName m:type="Edm.String">catalog10_small.txt</d:CatalogFileName>
</m:properties>
</content>
</entry>
</feed>
```

5.3. Get All Models

Retrieves all models of the current user.

HTTP METHOD	URI
GET	<code>/root/URI/GetAllModels?apiVersion=1.0</code> Example: <code>/root/URI/GetAllModels?apiVersion=1.0</code>
PARAMETER NAME	VALID VALUES
apiVersion	1.0
Request Body	NONE

Response:

HTTP Status code: 200

- `/feed/entry/content/properties/Id` - Model unique ID.
- `/feed/entry/content/properties/Name` - Model name.
- `/feed/entry/content/properties/Date` - Model creation date.
- `/feed/entry/content/properties/Status` - Model status. One of the following:
 - Created - Model is created and does not contain Catalog and Usage.
 - ReadyForBuild - Model is created and contains Catalog and Usage.
- `/feed/entry/content/properties/IsActiveBuild` - Indicates if the model was built successfully.
- `/feed/entry/content/properties/BuilderId` - Model active build ID.
- `/feed/entry/content/properties/Mpr` - Model MPR (see ModelInsight for more information).
- `/feed/entry/content/properties/UserName` - Model internal user name.
- `/feed/entry/content/properties/UsageFileNames` - List of model usage files separated by comma.
- `/feed/entry/content/properties/CatalogId` - Model catalog ID.
- `/feed/entry/content/properties/Description` - Model description.
- `/feed/entry/content/properties/CatalogFileName` - Model catalog file name.

OData XML

```

<feed xmlns:base="https://api.datamarket.azure.com/aml/recommendations/v3/GetAllModels" xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices" 
  xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns="http://www.w3.org/2005/Atom">
  <title type="text" />
  <subtitle type="text">Get A List Of All Models</subtitle>
  <id>https://api.datamarket.azure.com/aml/recommendations/v3/GetAllModels?apiVersion=1.0</id>
  <rights type="text" />
  <updated>2014-10-28T14:35:51Z</updated>
  <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetAllModels?apiVersion=1.0" />
  <entry>
    <id>https://api.datamarket.azure.com/aml/recommendations/v3/GetAllModels?apiVersion=1.0&skip=0&stop=1</id>
    <title type="text">Get A List Of All Models Entity</title>
    <updated>2014-10-28T14:35:51Z</updated>
    <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetAllModels?apiVersion=1.0&skip=0&stop=1" />
    <content type="application/xml">
      <m:properties>
        <d:id m:type="Edm.String">68b232f2-1037-45f7-8f49-ad822695ee63</d:id>
        <d:Name m:type="Edm.String">yah-1111</d:Name>
        <d:Date m:type="Edm.String">10/28/2014 2:16:07 PM</d:Date>
        <d>Status m:type="Edm.String">ReadyForBuild</d>Status>
        <d:HasActiveBuild m:type="Edm.String">false</d:HasActiveBuild>
        <d:BuildId m:type="Edm.String">-1</d:BuildId>
        <d:Mpr m:type="Edm.String"><0</d:Mpr>
        <d:UsageFileNames m:type="Edm.String">ImplicitMatrix0_Guid_small.txt, ImplicitMatrix1_Guid_small.txt</d:UsageFileNames>
        <d:CatalogId m:type="Edm.String">626babdb-cab6-43a6-82cf-4fb86345700c</d:CatalogId>
        <d:UserName m:type="Edm.String">89dc4722-03ba-4990-8821-b1db388408b5@dm.com</d:UserName>
        <d:Description m:type="Edm.String">short description</d:Description>
        <d:CatalogFileName m:type="Edm.String">catalog_10_small.txt</d:CatalogFileName>
      </m:properties>
    </content>
  </entry>
</feed>

```

5.4. Update Model

You can update the model description or the active build ID.

Active build ID - Every build for every model has a build ID. The active build ID is the first successful build of every new model. Once you have an active build ID and you do additional builds for the same model, you need to explicitly set it as the default build ID if you want to. When you consume recommendations, if you do not specify the build ID that you want to use, the default one will be used automatically.

This mechanism enables you - once you have a recommendation model in production - to build new models and test them before you promote them to production.

HTTP METHOD	URI
PUT	<rootURI>/UpdateModel?id=%27&apiVersion=%271.0%27 Example: <rootURI>/UpdateModel?id=%279559872f7a53-4076-a367-19d9385c1265%27&apiVersion=%271.0%27

PARAMETER NAME	VALID VALUES
id	Unique identifier of the model (case sensitive)
apiVersion	1.0

Request Body	
<ModelUpdateParams xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> <Description>New Description</Description> <ActiveBuildId>-1</ActiveBuildId> <ModelUpdateParams>	Note that the XML tags Description and ActiveBuildId are optional. If you do not want to set Description or ActiveBuildId, remove the entire tag.

Response:

HTTP Status code: 200

5.5. Delete Model

Deletes an existing model by ID.

HTTP METHOD	URI
DELETE	<rootURI>/DeleteModel?id=%27&model_id=%27&apiVersion=%271.0%27 Example: <rootURI>/DeleteModel?id=%27caefb76-de44-11be81-29b800a0b1de%27&apiVersion=%271.0%27

PARAMETER NAME	VALID VALUES
id	Unique identifier of the model (case sensitive)
apiVersion	1.0
Request Body	NONE

Response:

HTTP Status code: 200

OData XML

```

<feed xmlns:base="https://api.datamarket.azure.com/aml/recommendations/v3/DeleteModel" xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
      xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns="http://www.w3.org/2005/Atom">
  <title type="text" />
  <subtitle type="text">Delete Model by Id</subtitle>
  <id>https://api.datamarket.azure.com/aml/recommendations/v3/DeleteModel?Id=1cac7b76-def4-41f1-be81-29b806adb1de&amp;apiVersion=1.0</id>
  <rights type="text" />
  <updated>2014-10-28T10:39:33Z</updated>
  <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/DeleteModel?Id=1cac7b76-def4-41f1-be81-29b806adb1de&amp;apiVersion=1.0" />
  <entry>
    <id>https://api.datamarket.azure.com/aml/recommendations/v3/DeleteModel?Id=1cac7b76-def4-41f1-be81-29b806adb1de&amp;apiVersion=1.0&amp;$skip=0&amp;$top=1</id>
    <title type="text">DeleteModelByIdEntity</title>
    <updated>2014-10-28T10:39:33Z</updated>
    <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/DeleteModel?Id=1cac7b76-def4-41f1-be81-29b806adb1de&amp;apiVersion=1.0&amp;$skip=0&amp;$top=1" />
    <content type="application/xml">
      <m:properties>
        <d:string m:type="Edm.String"></d:string>
      </m:properties>
      <content>
        <entry>
          <id>https://api.datamarket.azure.com/aml/recommendations/v3/DeleteModel?Id=1cac7b76-def4-41f1-be81-29b806adb1de&amp;apiVersion=1.0&amp;$skip=0&amp;$top=1" />
        </content>
      </entry>
    </content>
  </entry>
</feed>

```

6. Model Advanced

6.1. Model Data Insight

Returns statistical data on the usage data that this model was built with.

Available only for Recommendation build.

HTTP METHOD	URI
GET	https://api.datamarket.azure.com/aml/DataInsight?modelId=%27-model_id-%27&apiVersion=%271.0%27 Example: https://api.datamarket.azure.com/aml/DataInsight?modelId=%271cac7b76-def4-41f1-be81-29b806adb1de%27&apiVersion=%271.0%27
PARAMETER NAME	VALID VALUES
modelId	Unique identifier of the model
apiVersion	1.0
Request Body	NONE

Response:

HTTP Status code: 200

The data is returned as a collection of properties.

- Holds the property name.
- Holds the property value.

The table below depicts the value that each key represents.

KEY	DESCRIPTION
AvgItemLength	Average number of distinct users per item.
AvgUserLength	Average number of distinct items per user.
DensificationNumberOfItems	Number of items after pruning items that cannot be modelled.
DensificationNumberOfUsers	Number of usage points after pruning users and items that can't be modelled.
DensificationNumberOfRecords	Number of usage points after pruning users and items that can't be modelled.
MaxItemLength	Number of distinct users for the most popular item.
MaxUserLength	Maximal number of distinct items for a user.
MinItemLength	Maximal number of distinct users for an item.
MinUserLength	Minimal number of distinct items for a user.
RawNumberOfItems	Number of items in the usage files.
RawNumberOfUsers	Number of usage points before any pruning.
RawNumberOfRecords	Number of usage points before any pruning.
SamplingNumberOfItems	N/A
SamplingNumberOfRecords	N/A
SamplingNumberOfUsers	N/A

OData XML

```

<feed xmlns:base="https://api.datamarket.azure.com/aml/recommendations/v3/GetDataInsight" xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
      xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns="http://www.w3.org/2005/Atom">
  <title type="text" />

```



```

</mproperties>
</content>
<entry>
<id>https://api.datamarket.azure.com/aml/a/recommendations/v3/GetDataInsight?modelId=6254b40d-0514-49cb-a298-b81256d2b3ca&amp;apiVersion=1.0&amp;$skip=9&amp;$top=1</id>
<title type="text">GetDataInsightStatisticsEntity</title>
<updated>2014-10-27T14:21:21Z</updated>
<link rel="self" href="https://api.datamarket.azure.com/aml/a/recommendations/v3/GetDataInsight?modelId=6254b40d-0514-49cb-a298-b81256d2b3ca&amp;apiVersion=1.0&amp;$skip=9&amp;$top=1" />
<content type="application/xml">
<mproperties>
<d:Key mtype="Edm.String">RawNumberOfItems</d:Key>
<d:Value mtype="Edm.String">2,000</d:Value>
</mproperties>
</content>
</entry>
<entry>
<id>https://api.datamarket.azure.com/aml/a/recommendations/v3/GetDataInsight?modelId=6254b40d-0514-49cb-a298-b81256d2b3ca&amp;apiVersion=1.0&amp;$skip=10&amp;$top=1</id>
<title type="text">GetDataInsightStatisticsEntity</title>
<updated>2014-10-27T14:21:21Z</updated>
<link rel="self" href="https://api.datamarket.azure.com/aml/a/recommendations/v3/GetDataInsight?modelId=6254b40d-0514-49cb-a298-b81256d2b3ca&amp;apiVersion=1.0&amp;$skip=10&amp;$top=1" />
<content type="application/xml">
<mproperties>
<d:Key mtype="Edm.String">RawNumberOfRecords</d:Key>
<d:Value mtype="Edm.String">72,022</d:Value>
</mproperties>
</content>
</entry>
<entry>
<id>https://api.datamarket.azure.com/aml/a/recommendations/v3/GetDataInsight?modelId=6254b40d-0514-49cb-a298-b81256d2b3ca&amp;apiVersion=1.0&amp;$skip=11&amp;$top=1</id>
<title type="text">GetDataInsightStatisticsEntity</title>
<updated>2014-10-27T14:21:21Z</updated>
<link rel="self" href="https://api.datamarket.azure.com/aml/a/recommendations/v3/GetDataInsight?modelId=6254b40d-0514-49cb-a298-b81256d2b3ca&amp;apiVersion=1.0&amp;$skip=11&amp;$top=1" />
<content type="application/xml">
<mproperties>
<d:Key mtype="Edm.String">RawNumberOfUsers</d:Key>
<d:Value mtype="Edm.String">9,044</d:Value>
</mproperties>
</content>
</entry>
<entry>
<id>https://api.datamarket.azure.com/aml/a/recommendations/v3/GetDataInsight?modelId=6254b40d-0514-49cb-a298-b81256d2b3ca&amp;apiVersion=1.0&amp;$skip=12&amp;$top=1</id>
<title type="text">GetDataInsightStatisticsEntity</title>
<updated>2014-10-27T14:21:21Z</updated>
<link rel="self" href="https://api.datamarket.azure.com/aml/a/recommendations/v3/GetDataInsight?modelId=6254b40d-0514-49cb-a298-b81256d2b3ca&amp;apiVersion=1.0&amp;$skip=12&amp;$top=1" />
<content type="application/xml">
<mproperties>
<d:Key mtype="Edm.String">SamplingNumberOfItems</d:Key>
<d:Value mtype="Edm.String">2,000</d:Value>
</mproperties>
</content>
</entry>
<entry>
<id>https://api.datamarket.azure.com/aml/a/recommendations/v3/GetDataInsight?modelId=6254b40d-0514-49cb-a298-b81256d2b3ca&amp;apiVersion=1.0&amp;$skip=13&amp;$top=1</id>
<title type="text">GetDataInsightStatisticsEntity</title>
<updated>2014-10-27T14:21:21Z</updated>
<link rel="self" href="https://api.datamarket.azure.com/aml/a/recommendations/v3/GetDataInsight?modelId=6254b40d-0514-49cb-a298-b81256d2b3ca&amp;apiVersion=1.0&amp;$skip=13&amp;$top=1" />
<content type="application/xml">
<mproperties>
<d:Key mtype="Edm.String">SamplingNumberOfRecords</d:Key>
<d:Value mtype="Edm.String">72,022</d:Value>
</mproperties>
</content>
</entry>
<entry>
<id>https://api.datamarket.azure.com/aml/a/recommendations/v3/GetDataInsight?modelId=6254b40d-0514-49cb-a298-b81256d2b3ca&amp;apiVersion=1.0&amp;$skip=14&amp;$top=1</id>
<title type="text">GetDataInsightStatisticsEntity</title>
<updated>2014-10-27T14:21:21Z</updated>
<link rel="self" href="https://api.datamarket.azure.com/aml/a/recommendations/v3/GetDataInsight?modelId=6254b40d-0514-49cb-a298-b81256d2b3ca&amp;apiVersion=1.0&amp;$skip=14&amp;$top=1" />
<content type="application/xml">
<mproperties>
<d:Key mtype="Edm.String">SamplingNumberOfUsers</d:Key>
<d:Value mtype="Edm.String">9,044</d:Value>
</mproperties>
</content>
</entry>
</feed>

```

6.2. Model Insight

Returns model insight on the active build or (if given) on a specific build.

Available only for Recommendation build.

HTTP METHOD	URI
GET	With active build ID: https://rootURI/api/GetModelInsight?modelId=%27model_id%27&apiVersion=%271.0%27 Example: https://rootURI/api/GetModelInsight?modelId=%27model_id%27&buildId=%27build_id%27&apiVersion=%271.0%27
PARAMETER NAME	VALID VALUES
modelId	Unique identifier of the model
buildId	Optional - number that identifies a successful build.
apiVersion	1.0

PARAMETER NAME	VALID VALUES
Request Body	NONE

Response:

HTTP Status code: 200

The data is returned as a collection of properties.

- `<!--> /entry /id /content /properties /key`
- `<!--> /entry /id /content /properties /value`

The table below depicts the value that each key represents.

KEY	DESCRIPTION
CatalogCoverage	What part of the catalog can be modelled with usage patterns. The rest of the items will need content-based features.
Mpr	Mean percentile ranking of the model. Lower is better.
NumberOfDimensions	Number of dimensions used by the matrix factorization algorithm.

OData XML

```

<feed xmlns:base="https://api.datamarket.azure.com/aml/recommendations/v3/GetModelInsight" xmlns:sd="http://schemas.microsoft.com/ado/2007/08/dataservices"
      xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns="http://www.w3.org/2005/Atom">
  <title type="text" />
  <subtitle type="text">Get model insight statistics</subtitle>
  <id>https://api.datamarket.azure.com/aml/recommendations/v3/GetModelInsight?modelId=6254b40d-0514-49cb-a298-b81256d2b3ca&amp;apiVersion=1.0</id>
  <right type="text" />
  <updated>2014-10-27T14:27:11Z</updated>
  <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetModelInsight?modelId=6254b40d-0514-49cb-a298-b81256d2b3ca&amp;apiVersion=1.0" />
  <entry>
    <id>https://api.datamarket.azure.com/aml/recommendations/v3/GetModelInsight?modelId=6254b40d-0514-49cb-a298-b81256d2b3ca&amp;apiVersion=1.0&amp;$skip=0&amp;$top=1</id>
    <title type="text">Get ModelInsightStatistics Entity</title>
    <updated>2014-10-27T14:27:11Z</updated>
    <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetModelInsight?modelId=6254b40d-0514-49cb-a298-b81256d2b3ca&amp;apiVersion=1.0&amp;$skip=0&amp;$top=1" />
    <content type="application/xml">
      <m:properties>
        <d:Key mtype="Edm.String">CatalogCoverage</d:Key>
        <d:Value mtype="Edm.String">1.000</d:Value>
        <m:properties>
        </m:properties>
      </content>
    </entry>
    <entry>
      <id>https://api.datamarket.azure.com/aml/recommendations/v3/GetModelInsight?modelId=6254b40d-0514-49cb-a298-b81256d2b3ca&amp;apiVersion=1.0&amp;$skip=1&amp;$top=1</id>
      <title type="text">Get ModelInsightStatistics Entity</title>
      <updated>2014-10-27T14:27:11Z</updated>
      <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetModelInsight?modelId=6254b40d-0514-49cb-a298-b81256d2b3ca&amp;apiVersion=1.0&amp;$skip=1&amp;$top=1" />
      <content type="application/xml">
        <m:properties>
          <d:Key mtype="Edm.String">Mpr</d:Key>
          <d:Value mtype="Edm.String">0.367</d:Value>
          <m:properties>
          </m:properties>
        </content>
      </entry>
      <entry>
        <id>https://api.datamarket.azure.com/aml/recommendations/v3/GetModelInsight?modelId=6254b40d-0514-49cb-a298-b81256d2b3ca&amp;apiVersion=1.0&amp;$skip=2&amp;$top=1</id>
        <title type="text">Get ModelInsightStatistics Entity</title>
        <updated>2014-10-27T14:27:11Z</updated>
        <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetModelInsight?modelId=6254b40d-0514-49cb-a298-b81256d2b3ca&amp;apiVersion=1.0&amp;$skip=2&amp;$top=1" />
        <content type="application/xml">
          <m:properties>
            <d:Key mtype="Edm.String">NumberOfDimensions</d:Key>
            <d:Value mtype="Edm.String">20</d:Value>
            <m:properties>
            </m:properties>
          </content>
        </entry>
      </feed>
    
```

6.3. Get Model Sample

Gets a sample of the recommendation model.

HTTP METHOD	URI
GET	<code><rootURL>/GetModelSample?modelId=%27model_id%27&apiVersion=%271.0%27</code> Example: <code><rootURL>/GetModelSample?modelId=%271eac7b76-de44-41f1-bx81-29b8f6cafb1.e%27&apiVersion=%271.0%27</code>

With specific build ID:

<code><rootURL>/GetModelSample?modelId=%27model_id%27&buildId=%27build_id%27&apiVersion=%271.0%27</code> Example: <code><rootURL>/GetModelSample?modelId=%271eac7b76-de44-41f1-bx81-29b8f6cafb1.e%27&buildId=%271500068%27&apiVersion=%271.0%27</code>

PARAMETER NAME	VALID VALUES
modelId	Unique identifier of the model
apiVersion	1.0
Request Body	NONE

Response:

HTTP Status code: 200

OData XML

Response is returned in raw text format:

Level 1

655fc955-a5a3-4a26-9723-3090859cb27b, Prey: A Novel
655fc955-a5a3-4a26-9723-3090859cb27b, Prey: A Novel Rating: 0.5215
3f471802-f84f-44a0-99c8-6d2e7418ecc1, Black Hawk Down: A Story of Modern War Rating: 0.5151
07b10e28-9e7c-4032-90b7-10acab7f2460, Cryptonomicon Rating: 0.5148
6afc18e4-8c2a-43d1-9021-57543d6b11d8, Imajica Rating: 0.5146
e4cc5e69-3567-43ab-b00f-f0d8d506870, Hit List Rating: 0.514
56b61441-0ecd-46cc-a8f6-112775b81892, Life and Death Shanghai
56b61441-0ecd-46cc-a8f6-112775b81892, Life and Death in Shanghai Rating: 0.5218
53156702-cc0c-443d-b718-6fb74b2491d3, Son of ' Rating: 0.5212
fb8c77a6-8719-46ce-97d4-92f931d77a3a, Smoke and Mirrors: Short Fictions and Illusions Rating: 0.5188
8f5fe006-79e4-4679-816b-950989d1db4b, A Place I've Never Been (Contemporary American Fiction) Rating: 0.5156
d8db4583-cc0f-49cc-be95-b7fa3491623f, Happiness: A Novel Rating: 0.5156
50471eecc-9aeb-4900-84d7-21567ab18546, If the Buddha Dated: A Handbook for Finding Love on a Spiritual Path
cf922aa1-7ca0-4f8d-ad9d-b7cc87bfe0ef, Divine Stories of the Ya-Ya Sisterhood: A Novel Rating: 0.5266
ff91a483-1ce5-4b37-a6fd-5ffc21f8743, The Poisonwood Bible: A Novel Rating: 0.5252
973f8cbd-0846-4f6b-9d28-4dd0d7dc3a19, Pigs in Heaven Rating: 0.5244
e2cbf7ad-0636-4117-8b30-298da0df7077, Animal Dreams Rating: 0.5227
6c818fd3-5a09-417d-9ab4-7ffe090f0fef, Confessions of an Ugly Stepsister: A Novel Rating: 0.5222
5e97148f-defb-4d74-af2d-80f4763bf531, The Deep End of the Ocean (Oprah's Book Club)
5e97148f-defb-4d74-af2d-80f4763bf531, The Deep End of the Ocean (Oprah's Book Club) Rating: 0.5277
5dcbaac37-2946-4f2a-a0b3-bbc710f9409a, Up Island: A Novel Rating: 0.5275
bc5b69db-733b-4346-adde-39275442587, Downtown Rating: 0.5275
31fce5c63-3e5a-48d0-802b-d3b0f989a634, Have a Nice Day: A Tale of Blood and Sweatsocks Rating: 0.5252
0adf981a-b65b-4c11-b36b-78aca2f948a2, The Perfect Storm: A True Story of Men Against the Sea Rating: 0.5238
6897068-ac1a-4163-9e94-396b800b743d, Modoc: The True Story of the Greatest Elephant That Ever Lived
6897068-ac1a-4163-9e94-396b800b743d, Modoc: The True Story of the Greatest Elephant That Ever Lived Rating: 0.5379
6724862e-e4e7-4022-9614-1468dbb902ff, Little House on the Prairie Rating: 0.5345
cdedb837-1620-496d-94c4-6ccfed888320, Little House in the Big Woods Rating: 0.5325
382164ba-406b-4187-b726-d7a5b9d790d, The Tao of Pooh Rating: 0.5309
6a068d6a-bb74-4ba3-b3f2-a956c4f9d1b5, On the Banks of Plum Creek Rating: 0.5285
37fe8e74-c348-44e5-aabc-1d79efcb25b, Men Are from Mars Women Are from Venus: A Practical Guide for Improving Communication and Getting What You Want in Your Relationships
37fe8e74-c348-44e5-aabc-1d79efcb25b, Men Are from Mars, Women Are from Venus: A Practical Guide for Improving Communication and Getting What You Want in Your Relationships Rating: 0.5397
f2be164a-f5af-4d32-ab33-9b4d5c292d6, Politically Correct Bedtime Stories: Modern Tales for Our Life and Times Rating: 0.5207
ef732c5c-334b-4d6b-ab82-7255eb7286d0, Honor Among Thieves Rating: 0.5195
0b209b8c-7cdd-47fd-b940-05c7ff7e60fc, The Giving Tree Rating: 0.5194
883b360f-8b42-407f-b977-2f44ad840877, Scary Stories to Tell in the Dark: Collected from American Folklore (Scary Stories) Rating: 0.5184
ff51b67e-fa8e-4c5e-8f4d-02a928dc735c, Men at Work: The Craft of Baseball
d008ade9-c73a-40a1-9a9b-96d5cf546f36, The Gulag Archipelago 1918-1956: An Experiment in Literary Investigation I-II Rating: 0.5416
ff51b67e-fa8e-4c5e-8f4d-02a928dc735d, Men at Work: The Craft of Baseball Rating: 0.5403
49dec30e-0adb-411a-b186-48eaabf6f8be, Fatherland Rating: 0.5394
cc7964fd-d30f-478e-a425-93ddbd0f94ed, Magic the Gathering: Arena Vol. 1 Rating: 0.5379
8a1e9f36-97af-4614-be99-24e3940a0f53, More Sniglets: Any Word That Doesn't Appear in the Dictionary but Should Rating: 0.5377
12a6d988-be21-4a09-8143-9d5f4261ba16, A Dream of Eagles
07b10e28-9e7c-4032-90b7-10acab7f2460, Cryptonomicon Rating: 0.5417
e4cc5e69-3567-43ab-b00f-f0d8d506870, Hit List Rating: 0.5416
1f1a34c4-9781-49f5-a3cc-accc3ae3c71d, The Family Rating: 0.5371
56daeffe-7d48-43cd-8ef8-7dff0c103d3, Kilo Class Rating: 0.5366
b2fe511e-5cb9-4a56-b283-2801e63e6a96, Legal Tender Rating: 0.5366
df87525b-e435-4bde-8701-4e60ad344e28, Finding Fish
56d33036-dfd4-46b9-8e2a-76cb03921bb0, The X-Files: Ground Zero Rating: 0.5417
0780cde8-6529-4e1d-b6c6-082c1b80e596, Twelve Red Herrings Rating: 0.5416
d87525b-e435-4bde-8701-4e60ad344e28, Finding Fish Rating: 0.5408
400fc331-2c35-490c-adbc-b284b4f73d5c, Shall We Tell the President? Rating: 0.5383
f86ad7d0-5c03-42b3-aebf-13d44aec8b30, Shades of Grace Rating: 0.5358
de1f6ad244-89e6-44d2-aace-992a4bf0931, The Map That Changed the World: William Smith and the Birth of Modern Geology
de16242a-89e6-44d2-aace-992a4bf0931, The Map That Changed the World: William Smith and the Birth of Modern Geology Rating: 0.5422
b30338f-e2c6-4a2c-b425-8d21e684fc3e, My Uncle Oswald Rating: 0.5385
34b84627-48af-44dc-96c4-b26fb3863f56, Midnight in the Garden of Good and Evil Rating: 0.5379
306cbaa7-b1a8-4142-9d55-e11b5018a7a8, The Street Lawyer Rating: 0.5376
e53b4baa-8c09-45c4-95c0-b6a26b98770b, Miss Smillas Feeling for Snow Rating: 0.5367

Level 2

352aaea1-6b12-454d-a3d5-46379d9e4eb2, The Sinister Pig (Hillerman Tony)
352aaea1-6b12-454d-a3d5-46379d9e4eb2, The Sinister Pig (Hillerman Tony) Rating: 0.5425
74c49398-bc10-4af5-a658-a996a1201254, Children of the Storm (Peters Elizabeth) Rating: 0.5387
9ba80080-196e-43fd-8025-391d963f77e7, The Floating Girl Rating: 0.5372
e6881d5-7745-4ce7-b943-fedb8fccc2ed, Killer Smile (Scottoline Lisa) Rating: 0.5353
b2fe511e-5cb9-4a56-b283-2801e63e6a96, Legal Tender Rating: 0.5332
c65c3995-ab7-4c7b-bb3c-8eb5aa9b7e5a, Lake Wobegon days
0ad981a-b65b-4c11-b36b-78aca2f948a2, The Perfect Storm: A True Story of Men Against the Sea Rating: 0.5433
c65c3995-ab7-4c7b-bb3c-8eb5aa9b7e5a, Lake Wobegon days Rating: 0.543
a00a6ad-4a7f-4211-9836-75ce8834eb11, Sniglets (Snig'lit: Any Word That Doesn't Appear in the Dictionary But Should) Rating: 0.5327
6f6e192c-0d64-49ca-9b63-f09413ea1e6, Politically Correct Holiday Stories: For an Enlightened Yuletide Season Rating: 0.5307
798051a8-147d-4d46-b6dc-e836325029e6, AGE OF INNOCENCE (MOVIE TIE-IN) Rating: 0.5301
73f3e25a-e996-4162-9e8d-f3d34075650, O Pioneers! (Penguin Twentieth-Century Classics)
c8b8163f-6536-436b-8130-47b4a43c827f, Trust No One (The Official Guide to the X-Files Vol. 2) Rating: 0.5434
5708e4cb-2492-49c0-94a8-cc413eec5d89, Small Gods (Discworld Novels (Paperback)) Rating: 0.5406
73f3e25a-e996-4162-9e8d-f3d34075650, O Pioneers! (Penguin Twentieth-Century Classics) Rating: 0.5403
d885b0bd-ae4b-452d-bdf2-faa0197db9, The Color of Magic Rating: 0.539
b133a9c4-4784-4db3-b100-d0d6dff94d2, The Truth Is Out There (The Official Guide to the X-Files Vol. 1) Rating: 0.5367
271700a5-854a-4d5a-8409-6b57a5ee4de4, Fluke: Or I Know Why the Winged Whale Sings
271700a5-854a-4d5a-8409-6b57a5ee4de4, Fluke: Or I Know Why the Winged Whale Sings Rating: 0.5445
2de1c354-90ff-47c5-a0db-1bad7d88e94, The Salaryman's Wife (Children of Violence Series) Rating: 0.5329
d279416e-19c0-43f8-9ec9-a585947879ca, Zen Attitude Rating: 0.5316
c8f854d7-3d3e-4b23-8217-f4851670d4, Revenge of the Cootie Girls: A Robin Hudson Mystery (Robin Hudson Mysteries (Paperback)) Rating: 0.5305
8ef4751c-7074-409e-3ac4-d49b222fc864, Where the Wild Things Are Rating: 0.5289
9ad1b620-0a7b-4543-8673-66d4c3bcb2f1, Their Eyes Were Watching God
9ad1b620-0a7b-4543-8673-66d4c3bcb2f1, Their Eyes Were Watching God Rating: 0.5446
da45c4d5-ab1-413b-a9bd-50d1f98b1e1d2, The Bean Trees Rating: 0.5389
65ecbdd1-131c-40c3-a3d6-d868ca281377a, The God of Small Things Rating: 0.5387

5f17d90a-2604-4fe8-8977-1a280b9098b1, One for the Money (Stephanie Plum Novels (Paperback))
5f17d90a-2604-4fe8-8977-1a280b9098b1, One for the Money (Stephanie Plum Novels (Paperback)) Rating: 0.5446
57169b2b-9a8a-486b-9aac-1ed98c5e7168, Final Appeal Rating: 0.5332
efcb1be4-7278-4a8f-91bf-0ef2070d6, Moment of Truth Rating: 0.5329
1ef9a12-993b-4c43-9f5c-3454fcf2612d, Burn Factor Rating: 0.5309
24c59962-458a-4ec8-b95d-d694e861919c, At Home in Mitford (The Mitford Years) Rating: 0.5303
4fd48c46-1a20-4c57-bc7f-0ef20f123dc52, As Nature Made Him: The Boy Who Was Raised As a Girl
4fd48c46-1a20-4c57-bc7f-a0ef2123dc52, As Nature Made Him: The Boy Who Was Raised As a Girl Rating: 0.5449
cf52d20c-43bc-43be-a1fb-3b4233e63222, Pigs in Heaven Rating: 0.5329
19985fdb-d07a-4a25-ac4a-97b9cb61e5d1, Love in the Time of Cholera (Penguin Great Books of the 20th Century) Rating: 0.5267
15689d09-c711-4844-84d8-130a90237b26, Bel Canto Rating: 0.5245
ff91a983-1ce5-4b37-af6d-5ffc2f18745, The Poisonwood Bible: A Novel Rating: 0.5235
98df28ec-41c7-4fc4-b77f-b8bd3109085d, Star Trek Memories
f874b5a3-5d40-4436-94f6-0fa1c090ddf5, The Sun Also Rises (A Scribner classic) Rating: 0.5451
98df28ec-41c7-4fc4-b77f-b8bd3109085d, Star Trek Memories Rating: 0.5442
0ce0014a-9a48-4013-a08a-7f2c11877930, H.M.S. Unseen Rating: 0.5421
15316ca6-1e38-425f-893d-6919444e7000, More Scary Stories To Tell In The Dark Rating: 0.5409
329d5682-3d3c-4206-8aa2-eef4b1032258, Letters from the Earth Rating: 0.54
5b9445d-c072-4194-8d49-f6696bb1b0a9, Daughter of Fortune: A Novel (Oprah's Book Club (Hardcover))
5b9445d-c072-4194-8d49-f6696bb1b0a9, Daughter of Fortune: A Novel (Oprah's Book Club (Hardcover)) Rating: 0.5462
ff91a843-1ce5-4b37-af6d-5ffc2f18745, The Poisonwood Bible: A Novel Rating: 0.5372
604eb3bd-6026-4f51-bffd-9fb54f180400, Family Pictures: A Novel Rating: 0.5341
8d06d01d-31cd-4678-b6b1-140a67987e9, Songs in Ordinary Time (Oprah's Book Club (Paperback)) Rating: 0.5334
da45c4d-abaa-413b-a9bd-50d98b1e1d2, The Bean Trees Rating: 0.5319
d5358189-d70f-4e35-8add-34b83b4942b3, Pigs in Heaven
d5358189-d70f-4e35-8add-34b83b4942b3, Pigs in Heaven Rating: 0.5491
ff91a843-1ce5-4b37-af6d-5ffc2f18745, The Poisonwood Bible: A Novel Rating: 0.5401
c78743bf-7947-4a0c-8db7-8a3bf69a70, The Stone Diaries Rating: 0.5393
8d06d01d-31cd-4678-b6b1-140a67987e9, Songs in Ordinary Time (Oprah's Book Club (Paperback)) Rating: 0.5382
973f8cbd-0846-4f6b-9d28-4dd0d7dc3a19, Pigs in Heaven Rating: 0.5367

7. Model Business Rules

These are the types of rules supported:

- **BlockList** - BlockList enables you to provide a list of items that you do not want to return in the recommendation results.
 - **FeatureBlockList** - Feature BlockList enables you to block items based on the values of its features.

Do not send more than 1000 items in a single blocklist rule or your call may timeout. If you need to block more than 1000 items, you can make several blocklist calls.

- **Upsale** - Upsale enables you to enforce items to return in the recommendation results.
 - **WhiteList** - White List enables you to only suggest recommendations from a list of items.
 - **FeatureWhiteList** - Feature White List enables you to only recommend items that have specific feature values.
 - **PerSeedBlockList** - Per Seed Block List enables you to provide per item a list of items that cannot be returned as recommendation results.

7.1. Get Model Rules

HTTP METHOD	URI
GET	<code>/api/v1/rd/GetModelDetails?modelId={0}&modelType={1}&apiVersion={2}&isTest={3}</code> Example: <code>/api/v1/rd/GetModelDetails?modelId=1&modelType=1&apiVersion=1&isTest=0</code>
PARAMETER NAME	VALID VALUES
modelId	Unique identifier of the model
apiVersion	1.0
Request Body	NONE

Response:

HTTP Status code: 200

- `/feed/entry/content/properties/Id` - Unique identifier of this rule.
 - `/feed/entry/content/properties/Type` - Type of the rule.
 - `/feed/entry/content/properties/Parameters` - Rule parameter.

OData XML

```

<feed xmlns:base="https://api.datamarket.azure.com/aml/recommendations/v3/GetModelRules" xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
  xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns="http://www.w3.org/2005/Atom">
  <title type="text" />
  <subtitle type="text">Get a list of rules for a model</subtitle>
  <id href="https://api.datamarket.azure.com/aml/recommendations/v3/GetModelRules?modelId='5e824626-50d3-469d-a824-564d38453103'&apiVersion='1.0'" />
  <rights type="text" />
  <updated>2014-11-05T12:58:57Z</updated>
  <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetModelRules?modelId='5e824626-50d3-469d-a824-564d38453103'&apiVersion='1.0'" />
  <entry>
    <id href="https://api.datamarket.azure.com/aml/recommendations/v3/GetModelRules?modelId='5e824626-50d3-469d-a824-564d38453103'&apiVersion='1.0'&skip=0&stop=1"/>
    <title type="text">GetAListOfRulesForAModelEntity</title>
    <updated>2014-11-05T12:58:57Z</updated>
    <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetModelRules?modelId='5e824626-50d3-469d-a824-564d38453103'&apiVersion='1.0'&skip=0&stop=1" />
    <content type="application/xml">
      <m:properties>
        <d:Id m:type="Edm.String">1000043</d:Id>
        <d>Type m:type="Edm.String">BlockList</d>Type>
        <d:Parameters m:type="Edm.String">{"ItemsToExclude":["1000"]}</d:Parameters>
      </m:properties>
    </content>
  </entry>
  <entry>
    <id href="https://api.datamarket.azure.com/aml/recommendations/v3/GetModelRules?modelId='5e824626-50d3-469d-a824-564d38453103'&apiVersion='1.0'&skip=1&stop=1"/>
    <title type="text">GetAListOfRulesForAModelEntity</title>
    <updated>2014-11-05T12:58:57Z</updated>
    <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetModelRules?modelId='5e824626-50d3-469d-a824-564d38453103'&apiVersion='1.0'&skip=1&stop=1" />
    <content type="application/xml">
      <m:properties>
        <d:Id m:type="Edm.String">1000044</d:Id>
        <d>Type m:type="Edm.String">BlockList</d>Type>
        <d:Parameters m:type="Edm.String">{"ItemsToExclude":["1001"]}</d:Parameters>
      </m:properties>
    </content>
  </entry>
</feed>

```

7.2. Add Rule

HTTP METHOD	URI
POST	<rootURL>/AddRule?apiVersion='2014-07-01'
HEADER	X-Auth-Token: [token]
PARAMETER NAME	VALID VALUES
apiVersion	1.0
Request Body	

Whenever providing Item Ids for business rules, make sure to use the External Id of the item (the same Id that you used in the catalog file).

To add a BlockList rule:

```
<Apifilter xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><ModelId>24024f7e-b45c-419e-bfa2-dfd947e0d253</ModelId><Type>BlockList</Type><Value>["ItemsToExclude": ["2406E770-7070-4189-89DE-1C9283F93A96", "3900E10-7070-4189-89DE-1C9283F98888"]]</Value></Apifilter>
```

To add a FeatureBlockList rule:

```
<Apifilter xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><ModelId>24024f7e-b45c-419e-bfa2-dfd947e0d253</ModelId><Type>FeatureBlockList</Type><Value>["Name": "Movie_category", "Values": ["Adult", "Drama"]]</Value></Apifilter>
```

To add an Upsale rule:

```
<Apifilter xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><ModelId>24024f7e-b45c-419e-bfa2-dfd947e0d253</ModelId><Type>Upsale</Type><Value>["ItemsToUpsale": ["2406E770-769C-4189-89DE-1C9283F93A96", "11164E770-769C-4189-89DE-1C9283F88888"]]</Value></Apifilter>
```

To add a WhiteList rule:

```
<Apifilter xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><ModelId>24024f7e-b45c-419e-bfa2-dfd947e0d253</ModelId><Type>WhiteList</Type><Value>["Name": "Movie_rating", "Values": ["PG13"]]</Value></Apifilter>
```

To add a FeatureWhiteList rule:

```
<Apifilter xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><ModelId>24024f7e-b45c-419e-bfa2-dfd947e0d253</ModelId><Type>FeatureWhiteList</Type><Value>["Name": "Movie_rating", "Values": ["PG13"]]</Value></Apifilter>
```

To add a PerSeedBlockList rule:

```
<Apifilter xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><ModelId>24024f7e-b45c-419e-bfa2-dfd947e0d253</ModelId><Type>PerSeedBlockList</Type><Value>["SeedItems": "9949", "ItemsToExclude": ["9862", "8158", "8244"]]</Value></Apifilter>
```

Response:

HTTP Status code: 200

The API returns the newly created rule with its details. The rules property can be retrieved from the following paths:

- readEntryContentProperties(id) - Unique identifier of this rule.
- readEntryContentProperties(Type) - Type of the rule: BlockList or Upsale.
- readEntryContentProperties(Parameters) - Rule parameter.

OData XML

```

<feed xmlns:base="https://api.datamarket.azure.com/aml/recommendations/v3/AddRule" xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
      xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns="http://www.w3.org/2005/Atom">
  <title type="text" />
  <subtitle type="text">Add A Rule</subtitle>
  <id>https://api.datamarket.azure.com/aml/recommendations/v3/AddRule?apiVersion=1.0</id>
  <rights type="text" />
  <updated>2014-11-05T11:13:28Z</updated>
  <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/AddRule?apiVersion=1.0" />
  <entry>
    <id>https://api.datamarket.azure.com/aml/recommendations/v3/AddRule?apiVersion=1.0&#038;$skip=0&#038;$top=1</id>
    <title type="text">Add A RuleEntity</title>
    <updated>2014-11-05T11:13:28Z</updated>
    <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/AddRule?apiVersion=1.0&#038;$skip=0&#038;$top=1" />
    <content type="application/xml">
      <mp:properties>
        <d:id m:type="Edm.String">1000041</d:id>
        <d>Type m:type="Edm.String">BlockList</d>Type>
        <d:Parameters m:type="Edm.String">{"ItemsToExclude":["1002"]}</d:Parameters>
      </mp:properties>
    </content>
  </entry>
</feed>

```

7.3. Delete Rule

HTTP METHOD	URI
DELETE	<rootURI>/DeleteRule/modelId=%27%model_id%27&filterId=%27%filter_Id%27&apiVersion=%271.0%27
	Example: /DeleteRule/modelId=%274024f7e-b45c-419e-bfa2-dff947e0253%27&filterId=%271000011%27&apiVersion=%271.0%27
PARAMETER NAME	VALID VALUES
modelId	Unique identifier of the model
filterId	Unique identifier of the filter
apiVersion	1.0
Request Body	NONE

Response:

HTTP Status code: 200

7.4. Delete All Rules

HTTP METHOD	URI
DELETE	<rootURI>/DeleteAllRules/modelId=%27%model_id%27&apiVersion=%271.0%27
	Example: /DeleteAllRules/modelId=%274024f7e-b45c-419e-bfa2-dff947e0253%27&apiVersion=%271.0%27
PARAMETER NAME	VALID VALUES
modelId	Unique identifier of the model
apiVersion	1.0
Request Body	NONE

Response:

HTTP Status code: 200

8. Catalog

8.1. Import Catalog Data

If you upload several catalog files to the same model with several calls, we will insert only the new catalog items. Existing items will remain with the original values. You cannot update catalog data by using this method.

The catalog data should follow the following format:

- Without features - <Item Id><Item Name><Item Category><Description>
- With features - <Item Id><Item Name><Item Category><Description><Features>

Note: The maximum file size is 200MB.

**** Format details ****

NAME	MANDATORY	TYPE	DESCRIPTION
Item Id	Yes	[A-z], [a-z], [0-9], [_] (Underscore), [-] (Dash) Max length: 50	Unique identifier of an item.
Item Name	Yes	Any alphanumeric characters Max length: 255	Item name.

NAME	MANDATORY	TYPE	DESCRIPTION
Item Category	Yes	Any alphanumeric characters Max length: 255	Category to which this item belongs (e.g. Cooking Books, Drama...); can be empty.
Description	No, unless features are present (but can be empty)	Any alphanumeric characters Max length: 4000	Description of this item.
Features list	No	Any alphanumeric characters Max length: 4000; Max number of features:20	Comma-separated list of feature name=feature value that can be used to enhance model recommendation; see Advanced topics section.
HTTP METHOD	URI		
POST	<code><rootURI>/ImportCatalogFile?modelId=%27&filename=%27&apiVersion=%271.0%27</code>		
	Example: <code><rootURI>/ImportCatalogFile?modelId=%27a658c626-2baa-43a7-ac98-ffee26120a12%27&filename=%27catalog10_small.txt%27&apiVersion=%271.0%27</code>		
HEADER	<code>Content-Type: text/xml</code>		
PARAMETER NAME	VALID VALUES		
modelId	Unique identifier of the model		
filename	Textual identifier of the catalog. Only letters (A-Z, a-z), numbers (0-9), hyphens (-) and underscore (_) are allowed. Max length: 50		
apiVersion	1.0		
Request Body	Example (with features): 2406e770-769c-4189-89de-1c9283f93a96,Clara Callan,Book,the book description,author=Richard Wright,publisher=Harper Flamingo Canada,year=2001 21bf8088-b6c0-4509-870c-e1c7ac78304a,The Forgetting Room: A Fiction (Byzantium Book),Book,author=Nick Bantock,publisher=Harpercollins,year=1997 3bb5cb44-d143-4bdd-a5c-443964bf4b23,Spadework,Book,,author=Timothy Findley, publisher=HarperFlamingo Canada, year=2001 552a1940-21e4-4399-82bb-594b46d7ed54,Restraint of Beasts,Book,the book description,author=Magnus Mills, publisher=Arcade Publishing, year=1998		

Response:

HTTP Status code: 200

The API returns a report of the import.

- `d:entry content properties LineCount` - Number of lines accepted.
- `d:entry content properties ErrorCount` - Number of lines that were not inserted due to an error.

OData XML

```
<feed xmlns:base="https://api.datamarket.azure.com/aml/recommendations/v3/ImportCatalogFile" xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices" xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns:r="http://www.w3.org/2005/Atom">
<title type="text" />
<subtitle type="text">Import catalog file</subtitle>
<id>https://api.datamarket.azure.com/aml/recommendations/v3/ImportCatalogFile?modelId=a658c626-2baa-43a7-ac98-ffee26120a12&filename=catalog10_small.txt&apiVersion=1.0</id>
<rights type="text" />
<updated>2014-10-05T06:58:04Z</updated>
<link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/ImportCatalogFile?modelId=a658c626-2baa-43a7-ac98-ffee26120a12&filename=catalog10_small.txt&apiVersion=1.0" />
<entry>
<id>https://api.datamarket.azure.com/aml/recommendations/v3/ImportCatalogFile?modelId=a658c626-2baa-43a7-ac98-ffee26120a12&filename=catalog10_small.txt&apiVersion=1.0&skip=0&stop=1</id>
<title type="text">ImportCatalogFileEntity</title>
<updated>2014-10-05T06:58:04Z</updated>
<link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/ImportCatalogFile?modelId=a658c626-2baa-43a7-ac98-ffee26120a12&filename=catalog10_small.txt&apiVersion=1.0&skip=0&stop=1" />
<content type="application/xml">
<m:properties>
<d:LineCount m:type="Edm.String">4</d:LineCount>
<d:ErrorCount m:type="Edm.String">0</d:ErrorCount>
</m:properties>
</content>
</entry>
</feed>
```

8.2. Get Catalog

Retrieves all catalog items. The catalog will be retrieved one page at a time. If you want to get items at a particular index, you can use the \$skip odata parameter. For instance if you want to get items starting at position 100, add the parameter \$skip=100 to the request.

HTTP METHOD	URI
GET	<code><rootURI>/GetCatalog/modelId=%27&model_id=%27&apiVersion=%271.0%27</code>
	Example: <code>GetCatalog?modelId=%272224024f7a445c419a-bfb2-4f97c012519%27&apiVersion=%271.0%27</code>

PARAMETER NAME	VALID VALUES
modelId	Unique identifier of the model
apiVersion	1.0
Request Body	NONE

Response:

HTTP Status code: 200

The response includes one entry per catalog item. Each entry has the following data:

- [[entry/content-properties/ExternalId]] - Catalog item external ID, the one provided by the customer.
- [[feed/entry/content-properties/InternalId]] - Catalog item internal ID, the one that Azure Machine Learning Recommendations has generated.
- [[feed/entry/content-properties/Name]] - Catalog item name.
- [[feed/entry/content-properties/Category]] - Catalog item category.
- [[feed/entry/content-properties/Description]] - Catalog item description.
- [[feed/entry/content-properties/Metadata]] - Catalog item metadata.

OData XML

```

<feed xmlns:base="https://api.datamarket.azure.com/aml/recommendations/v3/GetCatalog" xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices" xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns="http://www.w3.org/2005/Atom">
  <title type="text"/>
  <subtitle type="text">Get All Catalog Items</subtitle>
  <id>https://api.datamarket.azure.com/aml/recommendations/v3/GetCatalog?modelId=0dbb55fa-7f11-418d-8537-8ff2d9d1d9c6&amp;apiVersion=1.0</id>
  <rights type="text"/>
  <updated>2014-10-29T11:13:26Z</updated>
  <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetCatalog?modelId=0dbb55fa-7f11-418d-8537-8ff2d9d1d9c6&amp;apiVersion=1.0" />
  <entry>
    <id>https://api.datamarket.azure.com/aml/recommendations/v3/GetCatalog?modelId=0dbb55fa-7f11-418d-8537-8ff2d9d1d9c6&amp;apiVersion=1.0&amp;$skip=0&amp;$top=1</id>
    <title type="text">AllCatalogItemsEntity</title>
    <updated>2014-10-29T11:13:26Z</updated>
    <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetCatalog?modelId=0dbb55fa-7f11-418d-8537-8ff2d9d1d9c6&amp;apiVersion=1.0&amp;$skip=0&amp;$top=1" />
    <content type="application/xml">
      <m:properties>
        <d:ExternalId m:type="Edm.String">552A1940-21E4-4399-82BB-594B46D7ED54</d:ExternalId>
        <d:InternalId m:type="Edm.String">060db2f6-e6a6-464c-bb52-436d2da82a50</d:InternalId>
        <d:Name m:type="Edm.String">Restraint of Beasts</d:Name>
        <d:Category m:type="Edm.String">Book</d:Category>
        <d:Description m:type="Edm.String"></d:Description>
        <d:Metadata m:type="Edm.String"></d:Metadata>
      </m:properties>
    </content>
  </entry>
  <entry>
    <id>https://api.datamarket.azure.com/aml/recommendations/v3/GetCatalog?modelId=0dbb55fa-7f11-418d-8537-8ff2d9d1d9c6&amp;apiVersion=1.0&amp;$skip=1&amp;$top=1</id>
    <title type="text">AllCatalogItemsEntity</title>
    <updated>2014-10-29T11:13:26Z</updated>
    <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetCatalog?modelId=0dbb55fa-7f11-418d-8537-8ff2d9d1d9c6&amp;apiVersion=1.0&amp;$skip=1&amp;$top=1" />
    <content type="application/xml">
      <m:properties>
        <d:ExternalId m:type="Edm.String">2406E770-769C-4189-89DE-1C9283F93A96</d:ExternalId>
        <d:InternalId m:type="Edm.String">209d7bfc-2eb9-4455-92a3-7c867a41a74a</d:InternalId>
        <d:Name m:type="Edm.String">Clara Callan</d:Name>
        <d:Category m:type="Edm.String">Book</d:Category>
        <d:Description m:type="Edm.String"></d:Description>
        <d:Metadata m:type="Edm.String"></d:Metadata>
      </m:properties>
    </content>
  </entry>
  <entry>
    <id>https://api.datamarket.azure.com/aml/recommendations/v3/GetCatalog?modelId=0dbb55fa-7f11-418d-8537-8ff2d9d1d9c6&amp;apiVersion=1.0&amp;$skip=2&amp;$top=1</id>
    <title type="text">AllCatalogItemsEntity</title>
    <updated>2014-10-29T11:13:26Z</updated>
    <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetCatalog?modelId=0dbb55fa-7f11-418d-8537-8ff2d9d1d9c6&amp;apiVersion=1.0&amp;$skip=2&amp;$top=1" />
    <content type="application/xml">
      <m:properties>
        <d:ExternalId m:type="Edm.String">3BBS5CB44-D143-4BDD-A55C-443964BF4B23</d:ExternalId>
        <d:InternalId m:type="Edm.String">913ff79b-05b4-d4d-8042-6fa4cc1391dd</d:InternalId>
        <d:Name m:type="Edm.String">Spadeworks</d:Name>
        <d:Category m:type="Edm.String">Book</d:Category>
        <d:Description m:type="Edm.String"></d:Description>
        <d:Metadata m:type="Edm.String"></d:Metadata>
      </m:properties>
    </content>
  </entry>
  <entry>
    <id>https://api.datamarket.azure.com/aml/recommendations/v3/GetCatalog?modelId=0dbb55fa-7f11-418d-8537-8ff2d9d1d9c6&amp;apiVersion=1.0&amp;$skip=3&amp;$top=1</id>
    <title type="text">AllCatalogItemsEntity</title>
    <updated>2014-10-29T11:13:26Z</updated>
    <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetCatalog?modelId=0dbb55fa-7f11-418d-8537-8ff2d9d1d9c6&amp;apiVersion=1.0&amp;$skip=3&amp;$top=1" />
    <content type="application/xml">
      <m:properties>
        <d:ExternalId m:type="Edm.String">21BF8088-B6C0-4509-870C-E1C7AC7304A</d:ExternalId>
        <d:InternalId m:type="Edm.String">ea65e4fa-768c-40b4-92c3-69d3e8178691</d:InternalId>
        <d:Name m:type="Edm.String">The Forgetting Room: A Fiction (Byzantium Book)</d:Name>
        <d:Category m:type="Edm.String">Book</d:Category>
        <d:Description m:type="Edm.String"></d:Description>
        <d:Metadata m:type="Edm.String"></d:Metadata>
      </m:properties>
    </content>
  </entry>
</feed>
```

8.3. Get Catalog Items by Token

HTTP METHOD	URI
GET	/rootURI/GetCatalogItemsByToken?modelId=%27&token=%27&apiVersion=%271.0%27
PARAMETER NAME	VALID VALUES
modelId	Unique identifier of the model
token	Token of the catalog item's name. Should contain at least 3 characters.
apiVersion	1.0
Request Body	NONE

Response:

HTTP Status code: 200

The response includes one entry per catalog item. Each entry has the following data:

- [[entry content properties Internal]] - Catalog item internal ID, the one that Azure Machine Learning Recommendations has generated.
- [[entry content properties Name]] - Catalog item name.
- [[entry content properties Rating]] - (for future use)
- [[entry content properties Reasoning]] - (for future use)
- [[entry content properties Metadata]] - (for future use)
- [[entry content properties FormattedRating]] - (for future use)

OData XML

```
<feed xmlns:base="https://api.datamarket.azure.com/aml/recommendations/v3/GetCatalogItemsByToken" xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices" xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns="http://www.w3.org/2005/Atom">
<title type="text"/>
<subtitle type="text">Get Catalog items that contain a token</subtitle>
<id>https://api.datamarket.azure.com/aml/recommendations/v3/GetCatalogItemsByToken?modelId=0dbb55fa-7f11-418d-8537-8ff2d9d1d9c6&amp;token=Cl&amp;apiVersion=1.0</id>
<rights type="text"/>
<updated>2014-10-29T11:48:19Z</updated>
<link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetCatalogItemsByToken?modelId=0dbb55fa-7f11-418d-8537-8ff2d9d1d9c6&amp;token=Cl&amp;apiVersion=1.0" />
<entry>
<id>https://api.datamarket.azure.com/aml/recommendations/v3/GetCatalogItemsByToken?modelId=0dbb55fa-7f11-418d-8537-8ff2d9d1d9c6&amp;token=Cl&amp;apiVersion=1.0&amp;$skip=0&amp;$top=1</id>
<title type="text">CatalogItemsThatContainATokenEntity</title>
<updated>2014-10-29T11:48:19Z</updated>
<link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetCatalogItemsByToken?modelId=0dbb55fa-7f11-418d-8537-8ff2d9d1d9c6&amp;token=Cl&amp;apiVersion=1.0&amp;$skip=0&amp;$top=1" />
<content type="application/xml">
<m:properties>
<d:Internal type="Edm.String">2406E770-769C-4189-89DE-1C9283F93A96</d:Internal>
<d:Name mtype="Edm.String">Clara Callan</d:Name>
<d:Rating mtype="Edm.Double">0</d:Rating>
<d:Reasoning mtype="Edm.String"></d:Reasoning>
<d:Metadata mtype="Edm.String"></d:Metadata>
<d:FormattedRating mtype="Edm.Double" m:null="true"></d:FormattedRating>
</m:properties>
</content>
</entry>
</feed>
```

9. Usage data

9.1. Import Usage Data

9.1.1. Uploading File

This section shows how to upload usage data by using a file. You can call this API several times with usage data. All usage data will be saved for all calls.

HTTP METHOD	URI
POST	/rootURI/ImportUsageFile(modelId=%27&filename=%27&apiVersion=%271.0%27)
PARAMETER NAME	VALID VALUES
modelId	Unique identifier of the model
filename	Textual identifier of the catalog. Only letters (A-Z, a-z), numbers (0-9), hyphens (-) and underscore (_) are allowed. Max length: 50
apiVersion	1.0

PARAMETER NAME	VALID VALUES																					
Request Body	Usage data. Format: [User Id]-[Item Id]-[Time]-[Event]																					
	<table border="1"> <thead> <tr> <th>NAME</th><th>MANDATORY</th><th>TYPE</th><th>DESCRIPTION</th></tr> </thead> <tbody> <tr> <td>User Id</td><td>Yes</td><td>[A-z], [a-z], [0-9], [_] (Underscore), [-] (Dash) Max length: 255</td><td>Unique identifier of a user.</td></tr> <tr> <td>Item Id</td><td>Yes</td><td>[A-z], [a-z], [0-9], [_] (Underscore), [-] (Dash) Max length: 50</td><td>Unique identifier of an item.</td></tr> <tr> <td>Time</td><td>No</td><td>Date in format YYYY/MM/DDTHH:MM:SS (e.g. 2013/06/20T10:00:00)</td><td>Time of data.</td></tr> <tr> <td>Event</td><td>No; if supplied then must also put date</td><td>One of the following: • Click • RecommendationClick • AddShopCart • RemoveShopCart • Purchase</td><td></td></tr> </tbody> </table>	NAME	MANDATORY	TYPE	DESCRIPTION	User Id	Yes	[A-z], [a-z], [0-9], [_] (Underscore), [-] (Dash) Max length: 255	Unique identifier of a user.	Item Id	Yes	[A-z], [a-z], [0-9], [_] (Underscore), [-] (Dash) Max length: 50	Unique identifier of an item.	Time	No	Date in format YYYY/MM/DDTHH:MM:SS (e.g. 2013/06/20T10:00:00)	Time of data.	Event	No; if supplied then must also put date	One of the following: • Click • RecommendationClick • AddShopCart • RemoveShopCart • Purchase		
NAME	MANDATORY	TYPE	DESCRIPTION																			
User Id	Yes	[A-z], [a-z], [0-9], [_] (Underscore), [-] (Dash) Max length: 255	Unique identifier of a user.																			
Item Id	Yes	[A-z], [a-z], [0-9], [_] (Underscore), [-] (Dash) Max length: 50	Unique identifier of an item.																			
Time	No	Date in format YYYY/MM/DDTHH:MM:SS (e.g. 2013/06/20T10:00:00)	Time of data.																			
Event	No; if supplied then must also put date	One of the following: • Click • RecommendationClick • AddShopCart • RemoveShopCart • Purchase																				
	Maximum file size: 200MB																					
	Example:																					
	<pre>149452,lb3d95e2-84e4-414c-bb38-be9cf4f61c347 6360,lb3d95e2-84e4-414c-bb38-be9cf4f61c347 50321,lb3d95e2-84e4-414c-bb38-be9cf4f61c347 71285,lb3d95e2-84e4-414c-bb38-be9cf4f61c347 224450,lb3d95e2-84e4-414c-bb38-be9cf4f61c347 236645,lb3d95e2-84e4-414c-bb38-be9cf4f61c347 107951,lb3d95e2-84e4-414c-bb38-be9cf4f61c347</pre>																					

Response:

HTTP Status code: 200

- `<d>entry content properties LineCount` - Number of lines accepted.
- `<d>entry content properties ErrorCode` - Number of lines that were not inserted due to an error.
- `<d>entry content properties FileId` - File identifier.

OData XML

```
<feed xmlns:base="https://api.datamarket.azure.com/aml/recommendations/v3/ImportUsageFile" xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
      xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns="http://www.w3.org/2005/Atom">
  <title type="text" />
  <subtitle type="text">Add bulk usage data (usage file)</subtitle>
  <id href="https://api.datamarket.azure.com/aml/recommendations/v3/ImportUsageFile?modelId=a658c626-2baa-43a7-ac98-f8ee26120a12&filename=ImplicitMatrix10_Guid_small.txt"&apiVersion=1.0"></id>
  <rights type="text" />
  <updated>2014-10-05T07:21:44Z</updated>
  <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/ImportUsageFile?modelId=a658c626-2baa-43a7-ac98-f8ee26120a12&filename=ImplicitMatrix10_Guid_small.txt"&apiVersion=1.0" />
  <entry>
    <id href="https://api.datamarket.azure.com/aml/recommendations/v3/ImportUsageFile?modelId=a658c626-2baa-43a7-ac98-f8ee26120a12&filename=ImplicitMatrix10_Guid_small.txt"&apiVersion=1.0"></id>
    <title type="text">AddBulkUsageDataUsageFileEntity</title>
    <updated>2014-10-05T07:21:44Z</updated>
    <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/ImportUsageFile?modelId=a658c626-2baa-43a7-ac98-f8ee26120a12&filename=ImplicitMatrix10_Guid_small.txt"&apiVersion=1.0&Sskip=0&Stop=1" />
    <content type="application/xml">
      <m:properties>
        <d:LineCount mtype="Edm.String">33</d:LineCount>
        <d:ErrorCount mtype="Edm.String">0</d:ErrorCount>
        <d:FieldId mtype="Edm.String">fead7c1c-db01-4fc0-872f-65bcd36025d</d:FieldId>
      </m:properties>
    </content>
  </entry>
</feed>
```

9.1.2. Using Data Acquisition

This section shows how to send events in real time to Azure Machine Learning Recommendations, usually from your website.

HTTP METHOD	URI
POST	<code>/api/aml/v3/importusagefile</code>
HEADER	<code>["Content-Type", "text/xml"]</code>
PARAMETER NAME	VALID VALUES
apiVersion	1.0
Request body	Event data entry for each event you want to send. You should send for the same user or browser session the same ID in the SessionId field. (See sample of event body below.)

- Example for event 'Click':

```

<Event xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xi="http://www.w3.org/2001/XMLSchema-instance">
<ModelId>2779c063-48fb-46c1-bae3-74acddc8c1d1</ModelId>
<SessionId>11112222</SessionId>
<EventData>
<EventData>
<Name>Click</Name>
<ItemId>21BF8088-B6C0-4509-870C-E1C7AC78304A</ItemId>
<EventData>
<EventData>
</Event>

```

- Example for event 'RecommendationClick':

```

<Event xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xi="http://www.w3.org/2001/XMLSchema-instance">
<ModelId>2779c063-48fb-46c1-bae3-74acddc8c1d1</ModelId>
<SessionId>11112222</SessionId>
<EventData>
<EventData>
<Name>RecommendationClick</Name>
<ItemId>21BF8088-B6C0-4509-870C-E1C7AC78304A</ItemId>
<EventData>
<EventData>
</Event>

```

- Example for event 'AddShopCart':

```

<Event xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xi="http://www.w3.org/2001/XMLSchema-instance">
<ModelId>2779c063-48fb-46c1-bae3-74acddc8c1d1</ModelId>
<SessionId>11112222</SessionId>
<EventData>
<Name>AddShopCart</Name>
<ItemId>21BF8088-B6C0-4509-870C-E1C7AC78304A</ItemId>
<EventData>
<EventData>
</Event>

```

- Example for event 'RemoveShopCart':

```

<Event xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xi="http://www.w3.org/2001/XMLSchema-instance">
<ModelId>2779c063-48fb-46c1-bae3-74acddc8c1d1</ModelId>
<SessionId>11112222</SessionId>
<EventData>
<EventData>
<Name>RemoveShopCart</Name>
<ItemId>21BF8088-B6C0-4509-870C-E1C7AC78304A</ItemId>
<EventData>
<EventData>
</Event>

```

- Example for event 'Purchase':

```

<Event xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xi="http://www.w3.org/2001/XMLSchema-instance">
<ModelId>2779c063-48fb-46c1-bae3-74acddc8c1d1</ModelId>
<SessionId>11112222</SessionId>
<EventData>
<EventData>
<Name>Purchase</Name>
<PurchaseItems>
<PurchaseItem>
<ItemId>ABBF8081-C5C0-4F09-9701-E1C7AC78304A</ItemId>
<Count>1</Count>
</PurchaseItem>
<PurchaseItem>
<ItemId>21BF8088-B6C0-4509-870C-11C0AC7F304B</ItemId>
<Count>3</Count>
</PurchaseItem>
</PurchaseItems>
</EventData>
<EventData>
</Event>

```

- Example sending 2 events, 'Click' and 'AddShopCart':

```

<Event xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xi="http://www.w3.org/2001/XMLSchema-instance">
<ModelId>2779c063-48fb-46c1-bae3-74acddc8c1d1</ModelId>
<SessionId>11112222</SessionId>
<EventData>
<EventData>
<Name>Click</Name>
<ItemId>21BF8088-B6C0-4509-870C-E1C7AC78304A</ItemId>
<ItemName>itemName</ItemName>
<ItemDescription>item description</ItemDescription>
<ItemCategory>category</ItemCategory>
<EventData>
<EventData>
<Name>AddShopCart</Name>
<ItemId>552A1940-21E4-4399-82BB-594B46D7ED54</ItemId>
<EventData>
</Event>

```

Response: HTTP Status code: 200

9.2. List Model Usage Files

Retrieves metadata of all model usage files. The usage files will be retrieved one page at a time. Each page contains 100 items. If you want to get items at a particular index, you can use the \$skip odata parameter. For instance if you want to get items starting at position 100, add the parameter \$skip=100 to the request.

HTTP METHOD	URI
GET	<root URI>/ListModelUsageFiles?forModelId={model_id}&apiVersion=1.0
	Example: <root URI>/ListModelUsageFiles?forModelId=%270dbb55a-7f11-418d-8537- 91f34011b9&apiVersion=1.0
PARAMETER NAME	VALID VALUES
forModelId	Unique identifier of the model
apiVersion	1.0
Request Body	NONE

Response:

HTTP Status code: 200

The response includes one entry per usage file. Each entry has the following data:

- `entry/content/properties/Id` - Usage file ID.
- `entry/content/properties/Length` - Usage file length in MB.
- `entry/content/properties/DateModified` - Date when the usage file was created.
- `entry/content/properties/UsedInModel` - Whether the usage file is used in the model.

OData XML

```

<feed xmlns:base="https://api.datamarket.azure.com/aml/recommendations/v3>ListModelUsageFiles" xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
  xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns="http://www.w3.org/2005/Atom">
  <title type="text" />
  <subtitle type="text">Get a list of model's usage files info</subtitle>
  <id>https://api.datamarket.azure.com/aml/recommendations/v3>ListModelUsageFiles?forModelId=1c1110f8-7d9f-4c64-a807-4c9c5329993a&amp;apiVersion=1.0</id>
  <rights type="text" />
  <updated>2014-10-30T09:40:25Z</updated>
  <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3>ListModelUsageFiles?forModelId=1c1110f8-7d9f-4c64-a807-4c9c5329993a&amp;apiVersion=1.0" />
  <entry>
    <id>https://api.datamarket.azure.com/aml/recommendations/v3>ListModelUsageFiles?forModelId=1c1110f8-7d9f-4c64-a807-4c9c5329993a&amp;apiVersion=1.0&amp;$skip=0&amp;$top=1</id>
    <title type="text">GetAListOfModelsUsageFilesInfoEntity</title>
    <updated>2014-10-30T09:40:25Z</updated>
    <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3>ListModelUsageFiles?forModelId=1c1110f8-7d9f-4c64-a807-4c9c5329993a&amp;apiVersion=1.0&amp;$skip=0&amp;$top=1" />
    <content type="application/xml">
      <m:properties>
        <d:Id m:type="Edm.String">4c067b42-e975-4cb2-8c98-a6ab80ed6fd3</d:Id>
        <d:Length m:type="Edm.Double">0</d:Length>
        <d:DateModified m:type="Edm.String">10/30/2014 9:19:57 AM</d:DateModified>
        <d:UsedInModel m:type="Edm.String">true</d:UsedInModel>
      </m:properties>
    </content>
  </entry>
  <entry>
    <id>https://api.datamarket.azure.com/aml/recommendations/v3>ListModelUsageFiles?forModelId=1c1110f8-7d9f-4c64-a807-4c9c5329993a&amp;apiVersion=1.0&amp;$skip=1&amp;$top=1</id>
    <title type="text">GetAListOfModelsUsageFilesInfoEntity</title>
    <updated>2014-10-30T09:40:25Z</updated>
    <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3>ListModelUsageFiles?forModelId=1c1110f8-7d9f-4c64-a807-4c9c5329993a&amp;apiVersion=1.0&amp;$skip=1&amp;$top=1" />
    <content type="application/xml">
      <m:properties>
        <d:Id m:type="Edm.String">3126d816-4e80-4248-8339-1ebbd9d544d</d:Id>
        <d:Length m:type="Edm.Double">0.001</d:Length>
        <d:DateModified m:type="Edm.String">10/30/2014 9:21:44 AM</d:DateModified>
        <d:UsedInModel m:type="Edm.String">true</d:UsedInModel>
      </m:properties>
    </content>
  </entry>
</feed>
```

9.3. Get Usage Statistics

Gets usage statistics.

HTTP METHOD	URI
GET	<root URI>/GetUsageStatistics/modelId={modelId}&startDate={start Date}&endDate={end Date}&eventTypes={event Types}&apiVersion=1.0
	Example: <root URI>/GetUsageStatistics?modelId=%271d20c34fde1-4ea8-8e5d- 1793a6da469&27&start Date=%272014-02-10T00%3A00%27&end Date=%272014-02-11T00%3A00%27&event Types=%27&apiVersion=1.0
PARAMETER NAME	VALID VALUES
modelId	Unique identifier of the model
startDate	Start date. Format: yyyy/MM/ddTHH:mm:ss
endDate	End date. Format: yyyy/MM/ddTHH:mm:ss
eventTypes	Comma-separated string of event types or null to get all events
apiVersion	1.0
Request Body	NONE

Response:

HTTP Status code: 200

A collection of key/value elements. Each one contains the sum of events for a specific event type grouped by hour.

- `eventEntry[0].content.properties.Key` - Contains the time (grouped by hour) and the event type.
- `eventEntry[0].content.properties.Value` - Total event count.

OData XML

```
<feed xmlns:base="https://api.datamarket.azure.com/aml/recommendations/v3/GetUsageStatistics" xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
  xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns="http://www.w3.org/2005/Atom">
  <title type="text" />
  <subtitle type="text">Get usage statistics</subtitle>
  <id>https://api.datamarket.azure.com/aml/recommendations/v3/GetUsageStatistics?modelId=1d20c34f-dca1-4eac-8e5d-f299e4e4ad66&amp;startDate=2014/10/12T00:00:00&amp;endDate=2014/11/10T00:00:00&amp;eventTypes=&amp;apiVersion=1.0</id>
  <right type="text" />
  <updated>2014-11-18T11:39:16Z</updated>
  <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetUsageStatistics?modelId=1d20c34f-dca1-4eac-8e5d-f299e4e4ad66&amp;startDate=2014/10/12T00:00:00&amp;endDate=2014/11/10T00:00:00&amp;eventTypes=&amp;apiVersion=1.0" />
  <entry>
    <id>https://api.datamarket.azure.com/aml/recommendations/v3/GetUsageStatistics?modelId=1d20c34f-dca1-4eac-8e5d-f299e4e4ad66&amp;startDate=2014/10/12T00:00:00&amp;endDate=2014/11/10T00:00:00&amp;eventTypes=&amp;apiVersion=1.0&amp;$skip=0&amp;$top=1</id>
    <title type="text">GetUsageStatisticsEntity</title>
    <updated>2014-11-18T11:39:16Z</updated>
    <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetUsageStatistics?modelId=1d20c34f-dca1-4eac-8e5d-f299e4e4ad66&amp;startDate=2014/10/12T00:00:00&amp;endDate=2014/11/10T00:00:00&amp;eventTypes=&amp;apiVersion=1.0&amp;$skip=0&amp;$top=1" />
    <content type="application/xml">
      <m:properties>
        <d:Event m:type="Edm.String">11/9/2014 8:00:00 AM;Click</d:Event>
        <d:Value m:type="Edm.String">5</d:Value>
      </m:properties>
    </content>
  </entry>
  <entry>
    <id>https://api.datamarket.azure.com/aml/recommendations/v3/GetUsageStatistics?modelId=1d20c34f-dca1-4eac-8e5d-f299e4e4ad66&amp;startDate=2014/10/12T00:00:00&amp;endDate=2014/11/10T00:00:00&amp;eventTypes=&amp;apiVersion=1.0&amp;$skip=1&amp;$top=1</id>
    <title type="text">GetUsageStatisticsEntity</title>
    <updated>2014-11-18T11:39:16Z</updated>
    <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetUsageStatistics?modelId=1d20c34f-dca1-4eac-8e5d-f299e4e4ad66&amp;startDate=2014/10/12T00:00:00&amp;endDate=2014/11/10T00:00:00&amp;eventTypes=&amp;apiVersion=1.0&amp;$skip=1&amp;$top=1" />
    <content type="application/xml">
      <m:properties>
        <d:Event m:type="Edm.String">11/9/2014 8:00:00 AM;RecommendationClick</d:Event>
        <d:Value m:type="Edm.String">10</d:Value>
      </m:properties>
    </content>
  </entry>
  <entry>
    <id>https://api.datamarket.azure.com/aml/recommendations/v3/GetUsageStatistics?modelId=1d20c34f-dca1-4eac-8e5d-f299e4e4ad66&amp;startDate=2014/10/12T00:00:00&amp;endDate=2014/11/10T00:00:00&amp;eventTypes=&amp;apiVersion=1.0&amp;$skip=2&amp;$top=1</id>
    <title type="text">GetUsageStatisticsEntity</title>
    <updated>2014-11-18T11:39:16Z</updated>
    <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetUsageStatistics?modelId=1d20c34f-dca1-4eac-8e5d-f299e4e4ad66&amp;startDate=2014/10/12T00:00:00&amp;endDate=2014/11/10T00:00:00&amp;eventTypes=&amp;apiVersion=1.0&amp;$skip=2&amp;$top=1" />
    <content type="application/xml">
      <m:properties>
        <d:Event m:type="Edm.String">11/9/2014 8:00:00 AM;RemoveShopCart</d:Event>
        <d:Value m:type="Edm.String">10</d:Value>
      </m:properties>
    </content>
  </entry>
  <entry>
    <id>https://api.datamarket.azure.com/aml/recommendations/v3/GetUsageStatistics?modelId=1d20c34f-dca1-4eac-8e5d-f299e4e4ad66&amp;startDate=2014/10/12T00:00:00&amp;endDate=2014/11/10T00:00:00&amp;eventTypes=&amp;apiVersion=1.0&amp;$skip=3&amp;$top=1</id>
    <title type="text">GetUsageStatisticsEntity</title>
    <updated>2014-11-18T11:39:16Z</updated>
    <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetUsageStatistics?modelId=1d20c34f-dca1-4eac-8e5d-f299e4e4ad66&amp;startDate=2014/10/12T00:00:00&amp;endDate=2014/11/10T00:00:00&amp;eventTypes=&amp;apiVersion=1.0&amp;$skip=3&amp;$top=1" />
    <content type="application/xml">
      <m:properties>
        <d:Event m:type="Edm.String">11/5/2014 8:00:00 AM;RemoveShopCart</d:Event>
        <d:Value m:type="Edm.String">10</d:Value>
      </m:properties>
    </content>
  </entry>
</feed>
```

9.4. Get Usage File Sample

Retrieves the first 2KB of usage file content.

HTTP METHOD	URI
GET	<code>[rootURL]/GetUsageFileSample?modelId=%27&modelId=%27&fieldId=%27&fieldId=%27&fileId=%27&fileId=%27&FileVersion=%271.0%27</code>
	Example: <code>[rootURL]/GetUsageFileSample?modelId=1d20c34f-dca1-4eac-8e5d-f299e4e4ad66&fieldId=1&fileId=1&FileVersion=1.0&fileId=274c067b42-0754-4b28-99af-cd80cd4639-27&fieldId=271&FileVersion=1.0&fileId=27</code>
PARAMETER NAME	VALID VALUES
modelId	Unique identifier of the model
fieldId	Unique identifier of the model usage file
apiVersion	1.0
Request Body	NONE

Response:

HTTP Status code: 200

Response is returned in raw text format:

```
85526,2406E770-769C-4189-89DE-1C9283F93A96,2014/11/02T13:40:15,True,1
210926,2406E770-769C-4189-89DE-1C9283F93A96,2014/11/02T13:40:15,True,1
116866,2406E770-769C-4189-89DE-1C9283F93A96,2014/11/02T13:40:15,True,1
177458,2406E770-769C-4189-89DE-1C9283F93A96,2014/11/02T13:40:15,True,1
274004,21BF8088-B6C0-4509-870C-E1C7AC78304A,2014/11/02T13:40:15,True,1
123883,21BF8088-B6C0-4509-870C-E1C7AC78304A,2014/11/02T13:40:15,True,1
37712,21BF8088-B6C0-4509-870C-E1C7AC78304A,2014/11/02T13:40:15,True,1
152249,21BF8088-B6C0-4509-870C-E1C7AC78304A,2014/11/02T13:40:15,True,1
250948,21BF8088-B6C0-4509-870C-E1C7AC78304A,2014/11/02T13:40:15,True,1
235588,21BF8088-B6C0-4509-870C-E1C7AC78304A,2014/11/02T13:40:15,True,1
158254,21BF8088-B6C0-4509-870C-E1C7AC78304A,2014/11/02T13:40:15,True,1
271195,21BF8088-B6C0-4509-870C-E1C7AC78304A,2014/11/02T13:40:15,True,1
141157,21BF8088-B6C0-4509-870C-E1C7AC78304A,2014/11/02T13:40:15,True,1
171118,3BB5CB44-D143-4BDD-A55C-443964BF4B23,2014/11/02T13:40:15,True,1
225087,3BB5CB44-D143-4BDD-A55C-443964BF4B23,2014/11/02T13:40:15,True,1
```

9.5. Get Model Usage File

Retrieves the full content of the usage file.

HTTP METHOD	URI
GET	<rootURL>/GetModelUsageFile?mid=%271e1108-7d9f-4cb4-0807-4e9c5329993a%27&fdl=%273126d816-4c80-4348-8339-1abbfb9454d4%27&download=%271%27&apiVersion=%271.0%27
PARAMETER NAME	VALID VALUES
mid	Unique identifier of the model
fid	Unique identifier of the model usage file
download	1
apiVersion	1.0
Request Body	NONE

Response:

HTTP Status code: 200

Response is returned in raw text format:

```
85526,2406E770-769C-4189-89DE-1C9283F93A96,2014/11/02T13:40:15,True,1
210926,2406E770-769C-4189-89DE-1C9283F93A96,2014/11/02T13:40:15,True,1
116866,2406E770-769C-4189-89DE-1C9283F93A96,2014/11/02T13:40:15,True,1
177458,2406E770-769C-4189-89DE-1C9283F93A96,2014/11/02T13:40:15,True,1
274004,21BF8088-B6C0-4509-870C-E1C7AC78304A,2014/11/02T13:40:15,True,1
123883,21BF8088-B6C0-4509-870C-E1C7AC78304A,2014/11/02T13:40:15,True,1
37712,21BF8088-B6C0-4509-870C-E1C7AC78304A,2014/11/02T13:40:15,True,1
152249,21BF8088-B6C0-4509-870C-E1C7AC78304A,2014/11/02T13:40:15,True,1
250948,21BF8088-B6C0-4509-870C-E1C7AC78304A,2014/11/02T13:40:15,True,1
235588,21BF8088-B6C0-4509-870C-E1C7AC78304A,2014/11/02T13:40:15,True,1
158254,21BF8088-B6C0-4509-870C-E1C7AC78304A,2014/11/02T13:40:15,True,1
271195,21BF8088-B6C0-4509-870C-E1C7AC78304A,2014/11/02T13:40:15,True,1
141157,21BF8088-B6C0-4509-870C-E1C7AC78304A,2014/11/02T13:40:15,True,1
171118,3BB5CB44-D143-4BDD-A55C-443964BF4B23,2014/11/02T13:40:15,True,1
225087,3BB5CB44-D143-4BDD-A55C-443964BF4B23,2014/11/02T13:40:15,True,1
244881,3BB5CB44-D143-4BDD-A55C-443964BF4B23,2014/11/02T13:40:15,True,1
50547,3BB5CB44-D143-4BDD-A55C-443964BF4B23,2014/11/02T13:40:15,True,1
213090,3BB5CB44-D143-4BDD-A55C-443964BF4B23,2014/11/02T13:40:15,True,1
260655,3BB5CB44-D143-4BDD-A55C-443964BF4B23,2014/11/02T13:40:15,True,1
72214,3BB5CB44-D143-4BDD-A55C-443964BF4B23,2014/11/02T13:40:15,True,1
189334,3BB5CB44-D143-4BDD-A55C-443964BF4B23,2014/11/02T13:40:15,True,1
36326,3BB5CB44-D143-4BDD-A55C-443964BF4B23,2014/11/02T13:40:15,True,1
189336,3BB5CB44-D143-4BDD-A55C-443964BF4B23,2014/11/02T13:40:15,True,1
189334,552A1940-21E4-4399-82BB-594B46D7ED54,2014/11/02T13:40:15,True,1
260655,552A1940-21E4-4399-82BB-594B46D7ED54,2014/11/02T13:40:15,True,1
162100,552A1940-21E4-4399-82BB-594B46D7ED54,2014/11/02T13:40:15,True,1
54946,552A1940-21E4-4399-82BB-594B46D7ED54,2014/11/02T13:40:15,True,1
260965,552A1940-21E4-4399-82BB-594B46D7ED54,2014/11/02T13:40:15,True,1
102758,552A1940-21E4-4399-82BB-594B46D7ED54,2014/11/02T13:40:15,True,1
112602,552A1940-21E4-4399-82BB-594B46D7ED54,2014/11/02T13:40:15,True,1
163925,552A1940-21E4-4399-82BB-594B46D7ED54,2014/11/02T13:40:15,True,1
262998,552A1940-21E4-4399-82BB-594B46D7ED54,2014/11/02T13:40:15,True,1
144717,552A1940-21E4-4399-82BB-594B46D7ED54,2014/11/02T13:40:15,True,1
```

9.6. Delete Usage File

Deletes the specified model usage file.

HTTP METHOD	URI
-------------	-----

HTTP METHOD	URI
DELETE	<rootURL>/DeleteUsageFile?modelId=%@&fieldId=%@&fileId=%@&apiVersion=%@
	Example: <rootURL>/DeleteUsageFile?modelId=%270f8cd98d0f4-440e-a084-d13d22e47a73%27&fieldId=%27Qce0b09d-be5c-4e3b-0a03-ac7BC22e5a18%27&apiVersion=%271.0%27
PARAMETER NAME	VALID VALUES
modelId	Unique identifier of the model
fieldId	Unique identifier of the file to be deleted
apiVersion	1.0
Request Body	NONE

Response:

HTTP Status code: 200

9.7. Delete All Usage Files

Deletes all model usage files.

HTTP METHOD	URI
DELETE	<rootURL>/DeleteAllUsageFiles?modelId=%@&apiVersion=%@
	Example: <rootURL>/DeleteAllUsageFiles?modelId=%271c110f8-7d9f-4c64-a807-4c9e5329993a%27&apiVersion=%271.0%27
PARAMETER NAME	VALID VALUES
modelId	Unique identifier of the model
apiVersion	1.0
Request Body	NONE

Response:

HTTP Status code: 200

10. Features

This section shows how to retrieve feature information, such as the imported features and their values, their rank, and when this rank was allocated. Features are imported as part of the catalog data, and then their rank is associated when a rank build is done. Feature rank can change according to the pattern of usage data and type of items. But for consistent usage/items, the rank should have only small fluctuations. The rank of features is a non-negative number. The number 0 means that the feature was not ranked (happens if you invoke this API prior to the completion of the first rank build). The date at which the rank was attributed is called the score freshness.

10.1. Get Features Info (For Last Rank Build)

Retrieves the feature information, including ranking, for the last successful rank build.

HTTP METHOD	URI
GET	<rootURL>/GetModelFeatures?modelId=%@&samplingSize=%@&apiVersion=%@
	Example: <rootURL>/GetModelFeatures?modelId=%271c110f8-7d9f-4c64-a807-4c9e5329993a%27&samplingSize=10%27&apiVersion=%271.0%27
PARAMETER NAME	VALID VALUES
modelId	Unique identifier of the model
samplingSize	Number of values to include for each feature according to the data present in the catalog. Possible values are: -1 - All samples. 0 - No sampling. N - Return N samples for each feature name.
apiVersion	1.0
Request Body	NONE

Response:

HTTP Status code: 200

The response contains a list of feature info entries. Each entry contains:

- `entry/content/importProperties/dName` - Feature name.
- `entry/content/importProperties/dRankUpdateDate` - Date at which the rank was allocated to this feature, a.k.a. score freshness feature. A historical date ('0001-01-01T00:00:00')

means that no rank build was performed.

- `<id>/entry/content/mprProperties/d:Rank` - Feature rank (float). A rank of 2.0 and up is considered a good feature.
- `<id>/entry/content/mprProperties/d:SampleValues` - Comma-separated list of values up to the sampling size requested.

OData XML

```

<feed xmlns:base="https://api.datamarket.azure.com/aml/recommendations/v3/GetModelFeatures" xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
  xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns="http://www.w3.org/2005/Atom">
  <title type="text" />
  <subtitle type="text">Get the features of a model</subtitle>
  <id href="https://api.datamarket.azure.com/aml/recommendations/v3/GetModelFeatures?modelId=f13ab2e8-b530-4aa1-86f7-2f4a24714765&amp;samplingSize=10&amp;apiVersion=1.0"/>
  <rights type="text" />
  <updated>2015-01-08T13:15:02Z</updated>
  <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetModelFeatures?modelId=f13ab2e8-b530-4aa1-86f7-2f4a24714765&amp;samplingSize=10&amp;apiVersion=1.0" />
  <entry>
    <id href="https://api.datamarket.azure.com/aml/recommendations/v3/GetModelFeatures?modelId=f13ab2e8-b530-4aa1-86f7-2f4a24714765&amp;samplingSize=10&amp;apiVersion=1.0&amp;$skip=0&amp;$top=1"/>
    <title type="text">ModelFeaturesEntity</title>
    <updated>2015-01-08T13:15:02Z</updated>
    <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetModelFeatures?modelId=f13ab2e8-b530-4aa1-86f7-2f4a24714765&amp;samplingSize=10&amp;apiVersion=1.0&amp;$skip=0&amp;$top=1" />
    <content type="application/xml">
      <m:properties>
        <d:Name mtype="Edm.String">Author</d:Name>
        <d:Rank1UpdateDate mtype="Edm.String">0001-01-01T00:00:00</d:RankUpdateDate>
        <d:Rank mtype="Edm.String">0</d:Rank>
        <d:SampleValues mtype="Edm.String">A. A. Attanasio, A. A. Milne, A. Bates, A. C. Bhaktivedanta Swami Prabhupada et al., A. C. Crispin, A. C. Doyle, A. C. H. Smith, A. E. Parker, A. J. Holt, A. J. Matthews</d:SampleValues>
      </m:properties>
    </content>
  </entry>
  <entry>
    <id href="https://api.datamarket.azure.com/aml/recommendations/v3/GetModelFeatures?modelId=f13ab2e8-b530-4aa1-86f7-2f4a24714765&amp;samplingSize=10&amp;apiVersion=1.0&amp;$skip=1&amp;$top=1"/>
    <title type="text">ModelFeaturesEntity</title>
    <updated>2015-01-08T13:15:02Z</updated>
    <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetModelFeatures?modelId=f13ab2e8-b530-4aa1-86f7-2f4a24714765&amp;samplingSize=10&amp;apiVersion=1.0&amp;$skip=1&amp;$top=1" />
    <content type="application/xml">
      <m:properties>
        <d:Name mtype="Edm.String">Publisher</d:Name>
        <d:Rank1UpdateDate mtype="Edm.String">0001-01-01T00:00:00</d:RankUpdateDate>
        <d:Rank mtype="Edm.String">0</d:Rank>
        <d:SampleValues mtype="Edm.String">A. Mondadori, Abacus, Abacus UK, Abstract Studio, Acacia Press, Academy Chicago Publishers, Ace Books, ACE Charter, Actar</d:SampleValues>
      </m:properties>
    </content>
  </entry>
  <entry>
    <id href="https://api.datamarket.azure.com/aml/recommendations/v3/GetModelFeatures?modelId=f13ab2e8-b530-4aa1-86f7-2f4a24714765&amp;samplingSize=10&amp;apiVersion=1.0&amp;$skip=2&amp;$top=1"/>
    <title type="text">ModelFeaturesEntity</title>
    <updated>2015-01-08T13:15:02Z</updated>
    <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetModelFeatures?modelId=f13ab2e8-b530-4aa1-86f7-2f4a24714765&amp;samplingSize=10&amp;apiVersion=1.0&amp;$skip=2&amp;$top=1" />
    <content type="application/xml">
      <m:properties>
        <d:Name mtype="Edm.String">Year</d:Name>
        <d:Rank1UpdateDate mtype="Edm.String">0001-01-01T00:00:00</d:RankUpdateDate>
        <d:Rank mtype="Edm.String">0</d:Rank>
        <d:SampleValues mtype="Edm.String">0, 1920, 1926, 1927, 1929, 1930, 1932, 1942, 1943, 1946</d:SampleValues>
      </m:properties>
    </content>
  </entry>

```

10.2. Get Features Info (For Specific Rank Build)

Retrieves the feature information, including the ranking for a specific rank build.

HTTP METHOD	URI
GET	<code><rootURI>/GetModelFeatures?modelId=%271&modelId=%27&samplingSize=%27&samplingSize=%27&rankBuildId=%27&apiVersion=%271.0%27</code>
	Example: <code><rootURI>/GetModelFeatures?modelId=%271&modelId=%27&samplingSize=10&rankBuildId=1000551&apiVersion=%271.0%27</code>
PARAMETER NAME	VALID VALUES
modelId	Unique identifier of the model
samplingSize	Number of values to include for each feature according to the data present in the catalog. Possible values are: -1 - All samples. 0 - No sampling. N - Return N samples for each feature name.
rankBuildId	Unique identifier of the rank build or -1 for the last rank build
apiVersion	1.0
Request Body	NONE

Response:

HTTP Status code: 200

The response contains a list of feature info entries. Each entry contains:

- `<id>/entry/content/mprProperties/d:Name` - Feature name.
- `<id>/entry/content/mprProperties/d:Rank_updateDate` - Date at which the rank was allocated to this feature, a.k.a. score freshness feature. A historical date ('0001-01-01T00:00:00') means that no rank build was performed.

- `modelEntry/content/mpnProperties/d/Rank` - Feature rank (float). A rank of 2.0 and up is considered a good feature.
- `modelEntry/content/mpnProperties/d/SampleValues` - Comma-separated list of values up to the sampling size requested.

OData

```
<feed xmlns:base="https://api.datamarket.azure.com/aml/recommendations/v3/GetModelFeatures" xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices" 
  xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns:iis="http://www.w3.org/2005/Atom">
  <title type="text" />
  <subtitle type="text">Get the features of a model</subtitle>
  <id>https://api.datamarket.azure.com/aml/recommendations/v3/GetModelFeatures?modelId=f13ab2e8-b530-4aa1-86f7-2f4a24714765&amp;samplingSize=10&amp;rankBuildId=1000653&amp;apiVersion=1.0</id>
  <rights type="text" />
  <updated>2015-01-08T13:54:22Z</updated>
  <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetModelFeatures?modelId=f13ab2e8-b530-4aa1-86f7-2f4a24714765&amp;samplingSize=10&amp;rankBuildId=1000653&amp;apiVersion=1.0" />
  <entry>
    <id>https://api.datamarket.azure.com/aml/recommendations/v3/GetModelFeatures?modelId=f13ab2e8-b530-4aa1-86f7-2f4a24714765&amp;samplingSize=10&amp;rankBuildId=1000653&amp;apiVersion=1.0</id>
    <title type="text">ModelFeatures Entity</title>
    <updated>2015-01-08T13:54:22Z</updated>
    <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetModelFeatures?modelId=f13ab2e8-b530-4aa1-86f7-2f4a24714765&amp;samplingSize=10&amp;rankBuildId=1000653&amp;apiVersion=1.0&amp;$skip=0&amp;$top=1" />
    <content type="application/xml">
      <m:properties>
        <d:Name m:type="Edm.String">Author</d:Name>
        <d:RankUpdateDate m:type="Edm.String">2015-01-08T13:52:14.673</d:RankUpdateDate>
        <d:Rank m:type="Edm.String">3.38867426</d:Rank>
        <d:SampleValues m:type="Edm.String">A. Attanasio, A. A. Milne, A. Bates, A. C. Bhaktivedanta Swami Prabhupada et al., A. C. Crispin, A. C. Doyle, A. C. H. Smith, A. E. Parker, A. J. Holt, A. J. Matthews</d:SampleValues>
      </m:properties>
      <content type="application/xml">
        <m:properties>
          <d:Name m:type="Edm.String">Publisher</d:Name>
          <d:RankUpdateDate m:type="Edm.String">2015-01-08T13:52:14.673</d:RankUpdateDate>
          <d:Rank m:type="Edm.String">1.67839336</d:Rank>
          <d:SampleValues m:type="Edm.String">A. Mondadori, Abacus, Abacus Press, Abacus Uk, Abstract Studio, Acacia Press, Academy Chicago Publishers, Ace Books, ACE Charter, Actar</d:SampleValues>
        </m:properties>
      </content>
    </entry>
    <entry>
      <id>https://api.datamarket.azure.com/aml/recommendations/v3/GetModelFeatures?modelId=f13ab2e8-b530-4aa1-86f7-2f4a24714765&amp;samplingSize=10&amp;rankBuildId=1000653&amp;apiVersion=1.0&amp;$skip=1&amp;$top=1</id>
      <title type="text">ModelFeatures Entity</title>
      <updated>2015-01-08T13:54:22Z</updated>
      <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetModelFeatures?modelId=f13ab2e8-b530-4aa1-86f7-2f4a24714765&amp;samplingSize=10&amp;rankBuildId=1000653&amp;apiVersion=1.0&amp;$skip=1&amp;$top=1" />
      <content type="application/xml">
        <m:properties>
          <d:Name m:type="Edm.String">Year</d:Name>
          <d:RankUpdateDate m:type="Edm.String">2015-01-08T13:52:14.673</d:RankUpdateDate>
          <d:Rank m:type="Edm.String">1.12352145</d:Rank>
          <d:SampleValues m:type="Edm.String">0, 1920, 1926, 1927, 1929, 1930, 1932, 1942, 1943, 1946</d:SampleValues>
        </m:properties>
      </content>
    </entry>
  </feed>
```

11. Build

This section explains the different APIs related to builds. There are 3 types of builds: a recommendation build, a rank build and an FBT (frequently bought together) build.

The recommendation build's purpose is to generate a recommendation model used for predictions. Predictions (for this type of build) come in two flavors:

- I2I - a.k.a. Item to Item recommendations - given an item or a list of items this option will predict a list of items that are likely to be of high interest.
- U2I - a.k.a. User to Item recommendations - given a user id (and optionally a list of items) this option will predict a list of items that are likely to be of high interest for the given user (and its additional choice of items). The U2I recommendations are based on the history of items that were of interest for the user up to the time the model was built.

A rank build is a technical build that allows you to learn about the usefulness of your features. Usually, in order to get the best result for a recommendation model involving features, you should take the following steps:

- Trigger a rank build (unless the score of your features is stable) and wait till you get the feature score.
- Retrieve the rank of your features by calling the [Get Features Info API](#).
- Configure a recommendation build with the following parameters:
 - `useFeatureInModel` - Set to True.
 - `ModelFeatureList` - Set to a comma-separated list of features with a score of 2.0 or more (according to the ranks you retrieved in the previous step).
 - `ShowDetailedExplanation` - Set to True.
 - Optionally you can set `UseDetailedExplanation` to True and `ExplainingFeatureList` to the list of features you want to use for explanations (usually the same list of features used in modelling or a sublist).
- Trigger the recommendation build with the configured parameters.

Note: If you do not configure any parameters (e.g. invoke the recommendation build without parameters) or you do not explicitly disable the usage of features (e.g. `UseFeatureInModel` set to False), the system will set up the feature-related parameters to the explained values above in case a rank build exists.

There is no restriction on running a rank build and a recommendation build concurrently for the same model. Nevertheless, you cannot run two builds of the same type on the same model in parallel.

An FBT (Frequently bought together) build is yet another recommendations algorithm called sometimes "conservative" recommender, which is good for catalogs that are not homogeneous in nature (homogeneous: books, movies, some food, fashion; non-homogeneous: computer and devices, cross-domain, highly diverse).

Note: if the usage files that you uploaded contain the optional field "event type" then for FBT modelling only "Purchase" events will be used. If no event type is provided all

events will be considered as purchase.

11.1 Build parameters

Each build type can be configured via a set of parameters (depicted below). If you don't configure the parameters, the system will automatically attribute values to the parameters according to the information present at the time you trigger a build.

11.1.1. Usage condenser

Users or items with few usage points might contain more noise than information. The system attempts to predict the minimal number of usage points per user/item to be used in a model. This number will be within the range defined by the ItemCutoffLowerBound and ItemCutoffUpperBound parameters for items, and the range defined by the UserCutOffLowerBound and UserCutOffUpperBound parameters for users. The condenser effect on items or users can be minimized by setting at least one of the corresponding bounds to zero.

11.1.2. Rank build parameters

The table below depicts the build parameters for a rank build.

KEY	DESCRIPTION	TYPE	VALID VALUE
NumberOfModelIterations	The number of iterations the model performs is reflected by the overall compute time and the model accuracy. The higher the number, the better accuracy you will get, but the compute time will take longer.	Integer	10-50
NumberOfModelDimensions	The number of dimensions relates to the number of 'features' the model will try to find within your data. Increasing the number of dimensions will allow better fine-tuning of the results into smaller clusters. However, too many dimensions will prevent the model from finding correlations between items.	Integer	10-40
ItemCutOffLowerBound	Defines the item lower bound for the condenser. See usage condenser above.	Integer	2 or more (0 disable condenser)
ItemCutOffUpperBound	Defines the item upper bound for the condenser. See usage condenser above.	Integer	2 or more (0 disable condenser)
UserCutOffLowerBound	Defines the user lower bound for the condenser. See usage condenser above.	Integer	2 or more (0 disable condenser)
UserCutOffUpperBound	Defines the user upper bound for the condenser. See usage condenser above.	Integer	2 or more (0 disable condenser)

11.1.3. Recommendation build parameters

The table below depicts the build parameters for recommendation build.

KEY	DESCRIPTION	TYPE	VALID VALUE
NumberOfModelIterations	The number of iterations the model performs is reflected by the overall compute time and the model accuracy. The higher the number, the better accuracy you will get, but the compute time will take longer.	Integer	10-50
NumberOfModelDimensions	The number of dimensions relates to the number of 'features' the model will try to find within your data. Increasing the number of dimensions will allow better fine-tuning of the results into smaller clusters. However, too many dimensions will prevent the model from finding correlations between items.	Integer	10-40
ItemCutOffLowerBound	Defines the item lower bound for the condenser. See usage condenser above.	Integer	2 or more (0 disable condenser)
ItemCutOffUpperBound	Defines the item upper bound for the condenser. See usage condenser above.	Integer	2 or more (0 disable condenser)
UserCutOffLowerBound	Defines the user lower bound for the condenser. See usage condenser above.	Integer	2 or more (0 disable condenser)
UserCutOffUpperBound	Defines the user upper bound for the condenser. See usage condenser above.	Integer	2 or more (0 disable condenser)
Description	Build description.	String	Any text, maximum 512 chars
EnableModelingInsights	Allows you to compute metrics on the recommendation model.	Boolean	True/False
UseFeaturesInModel	Indicates if features can be used in order to enhance the recommendation model.	Boolean	True/False
ModelingFeatureList	Comma-separated list of feature names to be used in the recommendation build, in order to enhance the recommendation.	String	Feature names, up to 512 chars
AllowColdItemPlacement	Indicates if the recommendation should also push cold items via feature similarity.	Boolean	True/False
EnableFeatureCorrelation	Indicates if features can be used in reasoning.	Boolean	True/False
ReasoningFeatureList	Comma-separated list of feature names to be used for reasoning sentences (e.g. recommendation explanations).	String	Feature names, up to 512 chars

KEY	DESCRIPTION	TYPE	VALID VALUE
EnableU2I	Allow the personalized recommendation a.k.a. U2I (user to item recommendations).	Boolean	True/False (default true)

11.1.4. FBT build parameters

The table below depicts the build parameters for recommendation build.

KEY	DESCRIPTION	TYPE	VALID VALUE (DEFAULT)
FbtSupportThreshold	How conservative the model is. Number of co-occurrences of items to be considered for modeling.	Integer	3-50 (6)
FbtMaxItemSetSize	Bounds the number of items in a frequent set.	Integer	2-3 (2)
FbtMinimalScore	Minimal score that a frequent set should have in order to be included in the returned results. The higher the better.	Double	0 and above (0)
FbtSimilarityFunction	Defines the similarity function to be used by the build. Lift favors serendipity, Co-occurrence favors predictability, and Jaccard is a nice compromise between the two.	String	cooccurrence, lift, jaccard (lift)

11.2. Trigger a Recommendation Build

By default this API will trigger a recommendation model build. To trigger a rank build (in order to score features), the build API variant with build type parameter should be used.

HTTP METHOD	URI
POST	<root URL>/BuildModel? modelId=%27-modelId%27&userDescription=%27<description>%27&apiVersion=%271.0%27
	Example: <root URL>/BuildModel?modelId=%27a658c626-2baa-43a7-ac98-1fe2f6120612%27&userDescription=%27inst%20build%27&apiVersion=%271.0%27
HEADER	Content-Type: application/json (if sending Request Body)
PARAMETER NAME	VALID VALUES
modelId	Unique identifier of the model
userDescription	Textual identifier of the catalog. Note that if you use spaces you must encode it with %20 instead. See example above. Max length: 50
apiVersion	1.0
Request Body	If left empty then the build will be executed with the default build parameters. If you want to set the build parameters, send the parameters as XML into the body as in the following sample. (See the "Build parameters" section for an explanation of the parameters.) <NumberOfModelIterations>40</NumberOfModelIterations> <Number Of ModelDimensions>20</Number Of ModelDimensions><MinItemAppearance>5</MinItemAppearance> <MinUserAppearance>5</MinUserAppearance><EnableModelingInsights>true</EnableModelingInsights> <UseFeaturesInModel>false</UseFeaturesInModel><ModelingFeatureList>feature_name_1,feature_name_2,...</ModelingFeatureList><AllowColdItemPlacement>false</AllowColdItemPlacement> <EnabledFeatureCorrelation>false</EnabledFeatureCorrelation> <RankingFeatureList>feature_name_1,feature_name_2,...</RankingFeatureList><BuildParametersList>

Response:

HTTP Status code: 200

This is an asynchronous API. You will get a build ID as a response. To know when the build has ended, you should call the "Get Builds Status of a Model" API and locate this build ID in the response. Note that a build can take from minutes to hours depending on the size of the data.

You cannot consume recommendations till the build ends.

Valid build status:

- Create - Build request was created.
- Queued - Build request was sent and it is queued.
- Building - Build is in progress.
- Success - Build ended successfully.
- Error - Build ended with a failure.
- Cancelled - Build was cancelled.
- Cancelling - A cancel request for the build was sent.

Note that the build ID can be found under the following path: `/feedentry/content.properties[0].id`

OData XML

```

<feed xmlns:base="https://api.datamarket.azure.com/aml/recommendations/v3/BuildModel" xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices" xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns="http://www.w3.org/2005/Atom">
<title type="text" />
<subtitle type="text">Build a Model with RequestBody</subtitle>
<id href="https://api.datamarket.azure.com/aml/recommendations/v3/BuildModel?modelId=9559872f-7a53-4076-a3c7-19d9385c1265&userDescription=First build&apiVersion=1.0" />
<rights type="text" />
<updated>2014-10-05T08:56:34Z</updated>
<link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/BuildModel?modelId=9559872f-7a53-4076-a3c7-19d9385c1265&userDescription=First%20build&apiVersion=1.0" />
<entry>
<id href="https://api.datamarket.azure.com/aml/recommendations/v3/BuildModel?modelId=9559872f-7a53-4076-a3c7-19d9385c1265&userDescription=First build&apiVersion=1.0&skip=0&stop=1" />
<link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/BuildModel?modelId=9559872f-7a53-4076-a3c7-19d9385c1265&userDescription=First%20build&apiVersion=1.0&skip=0&stop=1" />
<content type="application/xml">
<nproperties>
<d:Id mtype="Edm.String">1000272</d:Id>
<d:UserName mtype="Edm.String"></d:UserName>
<d:ModelId mtype="Edm.String">9559872f-7a53-4076-a3c7-19d9385c1265</d:ModelId>
<d:ModelName mtype="Edm.String"></d:ModelName>
<d>Type mtype="Edm.String">Recommendation</d>Type>
<d:CreationTime mtype="Edm.String">2014-10-05T08:56:31.893</d:CreationTime>
<d:Progress_BuildId mtype="Edm.String">1000272</d:Progress_BuildId>
<d:Progress_ModelId mtype="Edm.String">9559872f-7a53-4076-a3c7-19d9385c1265</d:Progress_ModelId>
<d:Progress_UserName mtype="Edm.String">5-4058-ab36-1fc25405102@dm.com</d:Progress_UserName>
<d:Progress_IsExecutionStarted mtype="Edm.String">false</d:Progress_IsExecutionStarted>
<d:Progress_IsExecutionEnded mtype="Edm.String">false</d:Progress_IsExecutionEnded>
<d:Progress_Percent mtype="Edm.String"></d:Progress_Percent>
<d:Progress_StartTime mtype="Edm.String">~0001-01-01T00:00:00</d:Progress_StartTime>
<d:Progress_EndTime mtype="Edm.String">~0001-01-01T00:00:00</d:Progress_EndTime>
<d:Progress_UpdateDateUTC mtype="Edm.String"></d:Progress_UpdateDateUTC>
<d>Status mtype="Edm.String">Queued</d>Status>
<d:Key1 mtype="Edm.String">UseFeaturesInModel</d:Key1>
<d:Value1 mtype="Edm.String">False</d:Value1>
</nproperties>
</content>
</entry>
</feed>

```

11.3. Trigger Build (Recommendation, Rank or FBT)

HTTP METHOD	URI
POST	<pre> <rootURI>/BuildModel? modelId=%27[modelId]&userDescription=%27<description>%27&buildType=%27&buildType=%27&apiVersion=%271.0%27 </pre> <p>Example: <code><rootURI>/BuildModel?modelId=%279559872f-7a53-4076-a3c7-19d9385c1265&userDescription=%27First build%20and%27&buildType=%27Ranking%27&apiVersion=%271.0%27</code></p>
HEADER	<pre> "Content-Type", "text/xml" (If sending Request Body) </pre>
PARAMETER NAME	VALID VALUES
modelId	Unique identifier of the model
userDescription	Textual identifier of the catalog. Note that if you use spaces you must encode it with %20 instead. See example above. Max length: 50
buildType	Type of the build to invoke: - 'Recommendation' for recommendation build - 'Ranking' for rank build - 'Fbt' for FBT build
apiVersion	1.0
Request Body	If left empty then the build will be executed with the default build parameters. If you want to set build parameters, send them as XML into the body like in the following sample. (See the "Build parameters" section for an explanation and full list of the parameters.) <pre> <BuildParametersList><NumberOfModelIterations>40</NumberOfModelIterations> <NumberOfWorkDimensions>20</NumberOfWorkDimensions> <MinItemAppearance>5</MinItemAppearance> <MinUserAppearance>5</MinUserAppearance> </BuildParametersList> </pre>

Response:

HTTP Status code: 200

This is an asynchronous API. You will get a build ID as a response. To know when the build has ended, you should call the "Get Builds Status of a Model" API and locate this build ID in the response. Note that a build can take from minutes to hours depending on the size of the data.

You cannot consume recommendations till the build ends.

Valid build status:

- Create - Model was created.
- Queued - Model build was triggered and it is queued.
- Building - Model is being built.
- Success - Build ended successfully.
- Error - Build ended with a failure.
- Cancelled - Build was cancelled.
- Cancelling - Build is being cancelled.

Note that the build ID can be found under the following path: `/feed/entry/content/properties/1`

OData XML

```

<feed xmlns:base="https://api.datamarket.azure.com/aml/recommendations/v3/BuildModel" xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices" xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns="http://www.w3.org/2005/Atom">
<title type="text" />
<subtitle type="text">Build a Model with RequestBody</subtitle>
<id href="https://api.datamarket.azure.com/aml/recommendations/v3/BuildModel?modelId=9559872f-7a53-4076-a3c7-19d9385c1265&userDescription=First build&apiVersion=1.0" />
<rights type="text" />
<updated>2014-10-05T08:56:34Z</updated>
<link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/BuildModel?modelId=9559872f-7a53-4076-a3c7-19d9385c1265&userDescription=First%20build&apiVersion=1.0" />
<entry>
<id href="https://api.datamarket.azure.com/aml/recommendations/v3/BuildModel?modelId=9559872f-7a53-4076-a3c7-19d9385c1265&userDescription=First build&apiVersion=1.0&skip=0&stop=1" />
<link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/BuildModel?modelId=9559872f-7a53-4076-a3c7-19d9385c1265&userDescription=First%20build&apiVersion=1.0&skip=0&stop=1" />
<content type="application/xml">
<nproperties>
<d:Id mtype="Edm.String">1000272</d:Id>
<d:UserName mtype="Edm.String"></d:UserName>
<d:ModelId mtype="Edm.String">9559872f-7a53-4076-a3c7-19d9385c1265</d:ModelId>
<d:ModelName mtype="Edm.String">docTest</d:ModelName>
<d>Type mtype="Edm.String">Recommendation</d>Type>
<d:CreationTime mtype="Edm.String">2014-10-05T08:56:31.893</d:CreationTime>
<d:Progress_BuildId mtype="Edm.String">1000272</d:Progress_BuildId>
<d:Progress_ModelId mtype="Edm.String">9559872f-7a53-4076-a3c7-19d9385c1265</d:Progress_ModelId>
<d:Progress_UserName mtype="Edm.String"><!-->5-4058-ab36-1fc25405102@dm.com</d:Progress_UserName>
<d:Progress_IsExecutionStarted mtype="Edm.String">false</d:Progress_IsExecutionStarted>
<d:Progress_IsExecutionEnded mtype="Edm.String">false</d:Progress_IsExecutionEnded>
<d:Progress_Percent mtype="Edm.String"><!-->0</d:Progress_Percent>
<d:Progress_StartTime mtype="Edm.String"><!-->2001-01-01T00:00:00</d:Progress_StartTime>
<d:Progress_EndTime mtype="Edm.String"><!-->2001-01-01T00:00:00</d:Progress_EndTime>
<d:Progress_UpdateDateUTC mtype="Edm.String"><!--></d:Progress_UpdateDateUTC>
<d>Status mtype="Edm.String">Queued</d>Status>
<d:Key1 mtype="Edm.String">UseFeaturesInModel</d:Key1>
<d:Value1 mtype="Edm.String">False</d:Value1>
</nproperties>
</content>
</entry>
</feed>

```

11.4. Get Builds Status of a Model

Retrieves builds and their status for a specified model.

HTTP METHOD	URI
GET	<code>/root/URI/GetModelBuildStatus?modelId={modelId}&onlyLastBuild={onlyLastBuild}&apiVersion={apiVersion}</code>
Example:	
<code>/root/URI/GetModelBuildStatus?modelId=9559872f-7a53-4076-a3c7-19d9385c1265&onlyLastBuild=true&apiVersion=1.0</code>	
PARAMETER NAME	VALID VALUES
modelId	Unique identifier of the model
onlyLastBuild	Indicates whether to return all the build history of the model or only the status of the most recent build
apiVersion	1.0

Response:

HTTP Status code: 200

The response includes one entry per build. Each entry has the following data:

- `<!--> entry/content-properties/UserName` - Name of the user.
- `<!--> entry/content-properties/ModelName` - Name of the model.
- `<!--> entry/content-properties/ModelId` - Model unique identifier.
- `<!--> entry/content-properties/IsDeployed` - Whether the build is deployed (a.k.a. active build).
- `<!--> entry/content-properties/BuildId` - Build unique identifier.
- `<!--> entry/content-properties/BuildType` - Type of the build.
- `<!--> entry/content-properties/Status` - Build status. Can be one of the following: Error, Building, Queued, Cancelling, Cancelled, Success.
- `<!--> entry/content-properties/StatusMessage` - Detailed status message (applies only to specific states).
- `<!--> entry/content-properties/Progress` - Build progress (%).
- `<!--> entry/content-properties/StartTime` - Build start time.
- `<!--> entry/content-properties/EndTime` - Build end time.
- `<!--> entry/content-properties/Duration` - Build duration.
- `<!--> entry/content-properties/ProgressStep` - Details about the current stage of a build in progress.

Valid build status:

- Created - Build request entry was created.
- Queued - Build request was triggered and it is queued.
- Building - Build is in process.
- Success - Build ended successfully.
- Error - Build ended with a failure.
- Cancelled - Build was cancelled.
- Canceling - Build is being cancelled.

Valid values for build type:

- Rank - Rank build.
- Recommendation - Recommendation build.

OData XML

```

<feed xmlns:base="https://api.datamarket.azure.com/aml/recommendations/v3/GetModelBuildsStatus" xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices" xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns="http://www.w3.org/2005/Atom">
  <title type="text">
    <subtitle type="text">Get builds status of a model</subtitle>
  </title>
  <id href="https://api.datamarket.azure.com/aml/recommendations/v3/GetModelBuildsStatus?modelId=1d20c34f-dca1-4eac-8e5d-f299e4e4ad66&onlyLastBuild=False&apiVersion=1.0"/>
  <rights type="text">
    <updated>2014-11-05T17:51:10Z</updated>
  </rights>
  <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetModelBuildsStatus?modelId=1d20c34f-dca1-4eac-8e5d-f299e4e4ad66&onlyLastBuild=False&apiVersion=1.0" />
  <entry>
    <id href="https://api.datamarket.azure.com/aml/recommendations/v3/GetModelBuildsStatus?modelId=1d20c34f-dca1-4eac-8e5d-f299e4e4ad66&onlyLastBuild=False&apiVersion=1.0&skip=0&stop=1"/>
    <title type="text">GetBuildsStatusEntity</title>
    <updated>2014-11-05T17:51:10Z</updated>
    <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetModelBuildsStatus?modelId=1d20c34f-dca1-4eac-8e5d-f299e4e4ad66&onlyLastBuild=False&apiVersion=1.0&skip=0&stop=1" />
    <content type="application/xml">
      <m:properties>
        <d:UserName m:type="Edm.String">b-434e-b2c9-84935664ff20@dm.com</d:UserName>
        <d:ModelName m:type="Edm.String">ModelName</d:ModelName>
        <d:ModelId m:type="Edm.String">1d20c34f-dca1-4eac-8e5d-f299e4e4ad66</d:ModelId>
        <d:lsDeployed m:type="Edm.String">true</d:lsDeployed>
        <d:BuildId m:type="Edm.String">1000272</d:BuildId>
        <d:BuildType m:type="Edm.String">Recommendation</d:BuildType>
        <d>Status m:type="Edm.String">Success</d>Status>
        <d>StatusMessage m:type="Edm.String"></d>StatusMessage>
        <d:Progress m:type="Edm.String"></d:Progress>
        <d:StartTime m:type="Edm.String">2014-11-02T13:43:51</d:StartTime>
        <d:EndTime m:type="Edm.String">2014-11-02T13:45:10</d:EndTime>
        <d:ExecutionTime m:type="Edm.String">00:01:19</d:ExecutionTime>
        <d:lsExecutionStarted m:type="Edm.String">false</d:lsExecutionStarted>
        <d:ProgressStep m:type="Edm.String"></d:ProgressStep>
      </m:properties>
    </content>
  </entry>
</feed>

```

11.5. Get Builds Status

Retrieves build statuses of all models of a user.

HTTP METHOD	URI
GET	<rootURI>/GetUserBuildsStatus?onlyLastBuild=<bool>&apiVersion=<2 1.0>%37
Example: [<rootURI>/GetUserBuildsStatus?onlyLastBuild=true&apiVersion=%271.0%37]	
PARAMETER NAME	VALID VALUES
onlyLastBuild	Indicates whether to return all the build history of the model or only the status of the most recent build.
apiVersion	1.0

Response:

HTTP Status code: 200

The response includes one entry per build. Each entry has the following data:

- [<entry>/content/properties/UserName] - Name of the user.
- [<entry>/content/properties/ModelName] - Name of the model.
- [<entry>/content/properties/ModelId] - Model unique identifier.
- [<entry>/content/properties/lsDeployed] - Whether the build is deployed.
- [<entry>/content/properties/BuildId] - Build unique identifier.
- [<entry>/content/properties/BuildType] - Type of the build.
- [<entry>/content/properties>Status] - Build status. Can be one of the following: Error, Building, Queued, Cancelled, Cancelling, Success.
- [<entry>/content/properties/StatusMessage] - Detailed status message (applies only to specific states).
- [<entry>/content/properties/Progress] - Build progress (%).
- [<entry>/content/properties/StartTime] - Build start time.
- [<entry>/content/properties/EndTime] - Build end time.
- [<entry>/content/properties/ExecutionTime] - Build duration.
- [<entry>/content/properties/ProgressStep] - Details about the current stage of a build in progress.

Valid build status:

- Created - Build request entry was created.
- Queued - Build request was triggered and it is queued.
- Building - Build is in process.
- Success - Build ended successfully.
- Error - Build ended with a failure.
- Cancelled - Build was cancelled.
- Cancelling - Build is being cancelled.

Valid values for build type:

- Rank - Rank build.
- Recommendation - Recommendation build.

OData XML

```

<feed xmlns:base="https://api.datamarket.azure.com/aml/recommendations/v3/GetUserBuildsStatus" xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices" xmlns="http://www.w3.org/2005/Atom">
  <title type="text" />
  <subtitle type="text">Get builds status of a user</subtitle>
  <id>https://api.datamarket.azure.com/aml/recommendations/v3/GetUserBuildsStatus?onlyLastBuilds=False&apiVersion=1.0</id>
  <rights type="text" />
  <updated>2014-11-05T18:41:21Z</updated>
  <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetUserBuildsStatus?onlyLastBuilds=False&apiVersion=1.0" />
  <entry>
    <id>https://api.datamarket.azure.com/aml/recommendations/v3/GetUserBuildsStatus?onlyLastBuilds=False&apiVersion=1.0&skip=0&stop=1</id>
    <title type="text">GetBuilds Status Entity</title>
    <updated>2014-11-05T18:41:21Z</updated>
    <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetUserBuildsStatus?onlyLastBuilds=False&apiVersion=1.0&skip=0&stop=1" />
    <content type="application/xml">
      <mproperties>
        <d:UserName mtype="Edm.String">b-434e-b2c9-84935664ff20@dm.com</d:UserName>
        <d:ModelName mtype="Edm.String">ModelName</d:ModelName>
        <d:ModelId mtype="Edm.String">1d20c34f-dea1-4ea8-8e5d-f299e4ad66</d:ModelId>
        <d:IsDeployed mtype="Edm.String">true</d:IsDeployed>
        <d:BuildId mtype="Edm.String">1000272</d:BuildId>
        <d:BuildType mtype="Edm.String">Recommendation</d:BuildType>
        <d>Status mtype="Edm.String">Success</d>Status>
        <d>StatusMessage mtype="Edm.String"></d>StatusMessage>
        <d:Progress mtype="Edm.String"></d:Progress>
        <d:StartTime mtype="Edm.String">2014-11-02T13:43:51</d:StartTime>
        <d:EndTime mtype="Edm.String">2014-11-02T13:45:10</d:EndTime>
        <d:ExecutionTime mtype="Edm.String">00:01:19</d:ExecutionTime>
        <d:ExecutionStarted mtype="Edm.String">false</d:ExecutionStarted>
        <d:ProgressStep mtype="Edm.String"></d:ProgressStep>
      </mproperties>
    </content>
  </entry>
</feed>

```

11.6. Delete Build

Deletes a build.

NOTE:

You cannot delete an active build. The model should be updated to a different active build before you delete it.

You cannot delete an in-progress build. You should cancel the build first by calling **Cancel Build**.

HTTP METHOD	URI
DELETE	<root URI>/DeleteBuild?buildId={27}&apiVersion={27}&{27}
	Example: <root URI>/DeleteBuild?buildId=1-271590055-27&apiVersion=2710&27
PARAMETER NAME	VALID VALUES
buildId	Unique identifier of the build.
apiVersion	1.0

Response:

HTTP Status code: 200

11.7. Cancel Build

Cancels a build that is in building status.

HTTP METHOD	URI
PUT	<root URI>/CancelBuild?buildId={27}&apiVersion={2710}&{27}
	Example: <root URI>/CancelBuild?buildId=1-271590055-27&apiVersion=2710&27
PARAMETER NAME	VALID VALUES
buildId	Unique identifier of the build.
apiVersion	1.0

Response:

HTTP Status code: 200

11.8. Get Build Parameters

Retrieves build parameters.

HTTP METHOD	URI
GET	<root URI>/GetBuildParameters?buildId={27}&buildId={27}&apiVersion={2710}&{27}
	Example: <root URI>/GetBuildParameters?buildId=1-27100055-27&apiVersion=2710&27
PARAMETER NAME	VALID VALUES
buildId	Unique identifier of the build.
apiVersion	1.0

Response:

HTTP Status code: 200

This API returns a collection of key/value elements. Each element represents a parameter and its value:

- `key` - Build parameter name.
- `value` - Build parameter value.

The table below depicts the value that each key represents.

KEY	DESCRIPTION	TYPE	VALID VALUE
NumberOfModelIterations	The number of iterations the model performs is reflected by the overall compute time and the model accuracy. The higher the number, the better accuracy you will get, but the compute time will take longer.	Integer	10-50
NumberOfModelDimensions	The number of dimensions relates to the number of 'features' the model will try to find within your data. Increasing the number of dimensions will allow better fine-tuning of the results into smaller clusters. However, too many dimensions will prevent the model from finding correlations between items.	Integer	10-40
ItemCutOffLowerBound	Defines the item lower bound for the condenser. See usage condenser above.	Integer	2 or more (0 disable condenser)
ItemCutOffUpperBound	Defines the item upper bound for the condenser. See usage condenser above.	Integer	2 or more (0 disable condenser)
UserCutOffLowerBound	Defines the user lower bound for the condenser. See usage condenser above.	Integer	2 or more (0 disable condenser)
UserCutOffUpperBound	Defines the user upper bound for the condenser. See usage condenser above.	Integer	2 or more (0 disable condenser)
Description	Build description.	String	Any text, maximum 512 chars
EnableModelingInsights	Allows you to compute metrics on the recommendation model.	Boolean	True/False
UseFeaturesInModel	Indicates if features can be used in order to enhance the recommendation model.	Boolean	True/False
ModelingFeatureList	Comma-separated list of feature names to be used in the recommendation build, in order to enhance the recommendation.	String	Feature names, up to 512 chars
AllowColdItemPlacement	Indicates if the recommendation should also push cold items via feature similarity.	Boolean	True/False
EnableFeatureCorrelation	Indicates if features can be used in reasoning.	Boolean	True/False
ReasoningFeatureList	Comma-separated list of feature names to be used for reasoning sentences (e.g. recommendation explanations).	String	Feature names, up to 512 chars

OData XML

```
<feed xmlns:base="https://api.datamarket.azure.com/aml/recommendations/v3/GetBuildParameters" xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
  xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns="http://www.w3.org/2005/Atom">
  <title type="text" />
  <subtitle type="text">Get build parameters</subtitle>
  <id>https://api.datamarket.azure.com/aml/recommendations/v3/GetBuildParameters?buildId=1000653&amp;apiVersion=1.0</id>
  <rights type="text" />
  <updated>2015-01-08T13:50:57Z</updated>
  <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetBuildParameters?buildId=1000653&amp;apiVersion=1.0" />
  <entry>
    <id>https://api.datamarket.azure.com/aml/recommendations/v3/GetBuildParameters?buildId=1000653&amp;apiVersion=1.0&amp;$skip=0&amp;$top=1</id>
    <title type="text">GetBuildParametersEntity</title>
    <updated>2015-01-08T13:50:57Z</updated>
    <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetBuildParameters?buildId=1000653&amp;apiVersion=1.0&amp;$skip=0&amp;$top=1" />
    <content type="application/xml">
      <m:properties>
        <d:Key m:type="Edm.String">UseFeaturesInModel</d:Key>
        <d:Value m:type="Edm.String">False</d:Value>
      </m:properties>
    </content>
  </entry>
  <entry>
    <id>https://api.datamarket.azure.com/aml/recommendations/v3/GetBuildParameters?buildId=1000653&amp;apiVersion=1.0&amp;$skip=1&amp;$top=1</id>
    <title type="text">GetBuildParametersEntity</title>
    <updated>2015-01-08T13:50:57Z</updated>
    <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetBuildParameters?buildId=1000653&amp;apiVersion=1.0&amp;$skip=1&amp;$top=1" />
    <content type="application/xml">
      <m:properties>
        <d:Key m:type="Edm.String">AllowColdItemPlacement</d:Key>
        <d:Value m:type="Edm.String">False</d:Value>
      </m:properties>
    </content>
  </entry>
  <entry>
    <id>https://api.datamarket.azure.com/aml/recommendations/v3/GetBuildParameters?buildId=1000653&amp;apiVersion=1.0&amp;$skip=2&amp;$top=1</id>
    <title type="text">GetBuildParametersEntity</title>
    <updated>2015-01-08T13:50:57Z</updated>
    <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetBuildParameters?buildId=1000653&amp;apiVersion=1.0&amp;$skip=2&amp;$top=1" />
  </entry>
</feed>
```



```

<d:Key mtype="Edm.String">ReasoningFeatureList</d:Key>
<d:Value mtype="Edm.String"/>
</mproperties>
</content>
</entry>
<entry>
<id>https://api.datamarket.azure.com/aml/recommendations/v3/GetBuildParameters?buildId=1000653&amp;apiVersion=1.0&amp;$skip=12&amp;Stop=1</id>
<title type="text">GetBuildParametersEntity</title>
<updated>2015-01-08T13:50:57Z</updated>
<link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetBuildParameters?buildId=1000653&amp;apiVersion=1.0&amp;$skip=12&amp;Stop=1" />
<content type="application/xml">
<mproperties>
<d:Key mtype="Edm.String">Description</d:Key>
<d:Value mtype="Edm.String">rankBuild</d:Value>
</mproperties>
</content>
</entry>
</feed>

```

12. Recommendation

12.1. Get Item Recommendations (for active build)

Get recommendations of the active build of type "Recommendation" or "Fbt" based on a list of seeds (input) items.

HTTP METHOD	URI
GET	<code><rootURI>/ItemRecommend?modelId=%27&modelDb=%27&itemIds=%27&numberOfResults=10&includeMetadata=false&apiVersion=1.0%27</code>
Example:	
<code><rootURI>/ItemRecommend?modelId=%2779c063-48fb-46c1-bae3-74acddc8c1d1%27&itemIds=%271003%27&numberOfResults=10&includeMetadata=false&apiVersion=1.0%27</code>	
PARAMETER NAME	VALID VALUES
modelId	Unique identifier of the model
itemIds	Comma-separated list of the items to recommend for. If the active build is of type FBT then you can send only one item. Max length: 1024
numberOfResults	Number of required results Max: 150
includeMetadata	Future use, always false
apiVersion	1.0

Response:

HTTP Status code: 200

The response includes one entry per recommended item. Each entry has the following data:

- `<entry><content properties="0" />` - Recommended item ID.
- `<entry><content properties="Name" />` - Name of the item.
- `<entry><content properties="Rating" />` - Rating of the recommendation; higher number means higher confidence.
- `<entry><content properties="Reasoning" />` - Recommendation reasoning (e.g. recommendation explanations).

The example response below includes 10 recommended items.

OData XML

```

<feed xmlns:base="https://api.datamarket.azure.com/aml/recommendations/v3/ItemRecommend" xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
      xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns="http://www.w3.org/2005/Atom">
  <title type="text" />
  <subtitle type="text">Get Recommendation</subtitle>
  <id>https://api.datamarket.azure.com/aml/recommendations/v3/ItemRecommend?modelId=2779c063-48fb-46c1-bae3-74acddc8c1d1&itemIds=1003&amp;numberOfResults=10&amp;includeMetadata=false&amp;apiVersion=1.0</id>
  <rights type="text" />
  <updated>2014-10-05T12:28:48Z</updated>
  <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/ItemRecommend?modelId=2779c063-48fb-46c1-bae3-74acddc8c1d1&itemIds=1003&amp;numberOfResults=10&amp;includeMetadata=false&amp;apiVersion=1.0" />
  <entry>
    <id>https://api.datamarket.azure.com/aml/recommendations/v3/ItemRecommend?modelId=2779c063-48fb-46c1-bae3-74acddc8c1d1&itemIds=1003&amp;numberOfResults=10&amp;includeMetadata=false&amp;apiVersion=1.0&amp;$skip=0&amp;Stop=1</id>
    <title type="text">GetRecommendationEntity</title>
    <updated>2014-10-05T12:28:48Z</updated>
    <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/ItemRecommend?modelId=2779c063-48fb-46c1-bae3-74acddc8c1d1&itemIds=1003&amp;numberOfResults=10&amp;includeMetadata=false&amp;apiVersion=1.0&amp;$skip=0&amp;Stop=1" />
    <content type="application/xml">
      <mproperties>
        <d:Id mtype="Edm.String">159</d:Id>
        <d:Name mtype="Edm.String">159</d:Name>
        <d:Rating mtype="Edm.Double">0.54343480387708</d:Rating>
        <d:Reasoning mtype="Edm.String">People who like '1003' also like '159'</d:Reasoning>
      </mproperties>
      <content>
        <entry>
          <entry>
            <entry>
              <id>https://api.datamarket.azure.com/aml/recommendations/v3/ItemRecommend?modelId=2779c063-48fb-46c1-bae3-74acddc8c1d1&itemIds=1003&amp;numberOfResults=10&amp;includeMetadata=false&amp;apiVersion=1.0&amp;$skip=1&amp;Stop=1</id>
              <title type="text">GetRecommendationEntity</title>
              <updated>2014-10-05T12:28:48Z</updated>
              <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/ItemRecommend?modelId=2779c063-48fb-46c1-bae3-74acddc8c1d1&itemIds=1003&amp;numberOfResults=10&amp;includeMetadata=false&amp;apiVersion=1.0&amp;$skip=1&amp;Stop=1" />
              <content type="application/xml">
                <mproperties>
                  <d:Id mtype="Edm.String">52</d:Id>
                  <d:Name mtype="Edm.String">52</d:Name>
                </mproperties>
              </content>
            </entry>
          </entry>
        </entry>
      </content>
    </entry>
  </feed>

```



```

<d:Id mtype="Edm.String">155</d:Id>
<d:Name mtype="Edm.String">155</d:Name>
<d:Rating mtype="Edm.Double">0.529093541481333</d:Rating>
<d:Reasoning mtype="Edm.String">People who like '1003' also like '155'</d:Reasoning>
<properties>
<content>
</entry>
<entry>
<id>https://api.datamarket.azure.com/aml/recommendations/v3/ItemRecommend?modelId=2779c063-48fb-46c1-bae3-74acddc8&id1=&#1003&amp;itemIds=1003&amp;numberOfResults=10&amp;includeMetadata=false&amp;apiVersion=1.0&amp;$skip=9&amp;$top=1</id>
<title type="text">GetRecommendationEntity</title>
<updated>2014-10-05T12:28:48Z</updated>
<link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/ItemRecommend?modelId=2779c063-48fb-46c1-bae3-74acddc8&id1=&#1003&amp;itemIds=1003&amp;numberOfResults=10&amp;includeMetadata=false&amp;apiVersion=1.0&amp;$skip=9&amp;$top=1" />
<content type="application/xml">
<properties>
<d:Id mtype="Edm.String">32</d:Id>
<d:Name mtype="Edm.String">32</d:Name>
<d:Rating mtype="Edm.Double">0.528917978168322</d:Rating>
<d:Reasoning mtype="Edm.String">People who like '1003' also like '32'</d:Reasoning>
<properties>
<content>
</entry>
</feed>

```

12.2. Get Item Recommendations (of a specific build)

Get recommendations of a specific build of type "Recommendation" or "Fbt".

HTTP METHOD	URI
GET	<rootURL>/ItemRecommend?modelId=%27<modelId%27&itemIds=%27<itemIds%27&numberOfResults=%20'&includeMetadata=true'&buildId=%20'&apiVersion=%271.0%27
PARAMETER NAME	VALID VALUES
modelId	Unique identifier of the model
itemIds	Comma-separated list of the items to recommend for. If the active build is of type FBT then you can send only one item. Max length: 1024
numberOfResults	Number of required results Max: 150
includeMetadata	Future use, always false
buildId	the build id to use for this recommendation request
apiVersion	1.0

Response:

HTTP Status code: 200

The response includes one entry per recommended item. Each entry has the following data:

- reed entry content properties Id - Recommended item ID.
- reed entry content properties Name - Name of the item.
- reed entry content properties Rating - Rating of the recommendation; higher number means higher confidence.
- reed entry content properties Reasoning - Recommendation reasoning (e.g. recommendation explanations).

See a response example in 12.1

12.3. Get FBT Recommendations (for active build)

Get recommendations of the active build of type "Fbt" based on a seed (input) item.

HTTP METHOD	URI
GET	<rootURL>/ItemFbRecommend?modelId=%27<modelId%27&itemId=%27<itemId%27&numberOfResults=%20'&minimalScore=double'&includeMetadata=true'&version=%271.0%27
PARAMETER NAME	VALID VALUES
modelId	Unique identifier of the model
itemId	Item to recommend for. Max length: 1024
numberOfResults	Number of required results Max: 150
minimalScore	Minimal score that a frequent set should have in order to be included in the returned results
includeMetadata	Future use, always false

PARAMETER NAME	VALID VALUES
apiVersion	1.0

Response:

HTTP Status code: 200

The response includes one entry per recommended item set (a set of items which are usually bought together with the seed/input item). Each entry has the following data:

- [seed entry content properties (d1)] - Recommended item ID.
- [seed entry content properties (Name1)] - Name of the item.
- [seed entry content properties (d2)] - 2nd recommended item ID (optional).
- [seed entry content properties (Name2)] - Name of the 2nd item (optional).
- [seed entry content properties Rating] - Rating of the recommendation; higher number means higher confidence.
- [seed entry content properties Reasoning] - Recommendation reasoning (e.g. recommendation explanations).

The example response below includes 3 recommended item sets.

OData XML

```
<feed xmlns:base="https://api.datamarket.azure.com/aml/recommendations/v3/ItemFbtRecommend" xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices" xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns="http://www.w3.org/2005/Atom">
<title type="text" />
<subtitle type="text">Get Recommendation</subtitle>
<d href="https://api.datamarket.azure.com/aml/recommendations/v3/ItemFbtRecommend?modelId=2779c063-48fb-46c1-bae3-74acddc8&id1='1003'&numberOfResults=3&minimalScore=0.1&includeMetadata=false&apiVersion=1.0" type="text" />
<right type="text" />
<updated>2014-10-05T12:28:48Z</updated>
<link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/ItemFbtRecommend?modelId=2779c063-48fb-46c1-bae3-74acddc8&id1='1003'&numberOfResults=3&minimalScore=0.1&includeMetadata=false&apiVersion=1.0" />
<entry>
<id href="https://api.datamarket.azure.com/aml/recommendations/v3/ItemFbtRecommend?modelId=2779c063-48fb-46c1-bae3-74acddc8&id1='1003'&numberOfResults=3&minimalScore=0.1&includeMetadata=false&apiVersion=1.0&$skip=0&Stop=1" type="text" />
<title type="text">GetFbtRecommendationEntity</title>
<updated>2014-10-05T12:28:48Z</updated>
<link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/ItemFbtRecommend?modelId=2779c063-48fb-46c1-bae3-74acddc8&id1='1003'&numberOfResults=3&minimalScore=0.1&includeMetadata=false&apiVersion=1.0&$skip=0&Stop=1" type="text" />
<content type="application/xml">
<mproperties>
<d id1 mtype="Edm.String">159</d><d id1>
<d name1 mtype="Edm.String">159</d><d name1>
<d id2 mtype="Edm.String"></d><d id2>
<d name2 mtype="Edm.String"></d><d name2>
<d rating mtype="Edm.Double">0.543343480387708</d><d rating>
<d reasoning mtype="Edm.String">People who bought '1003' also bought '159'</d><d reasoning>
</mproperties>
<content>
<entry>
<id href="https://api.datamarket.azure.com/aml/recommendations/v3/ItemFbtRecommend?modelId=2779c063-48fb-46c1-bae3-74acddc8&id1='1003'&numberOfResults=3&minimalScore=0.1&includeMetadata=false&apiVersion=1.0&$skip=1&Stop=1" type="text" />
<title type="text">GetFbtRecommendationEntity</title>
<updated>2014-10-05T12:28:48Z</updated>
<link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/ItemFbtRecommend?modelId=2779c063-48fb-46c1-bae3-74acddc8&id1='1003'&numberOfResults=3&minimalScore=0.1&includeMetadata=false&apiVersion=1.0&$skip=1&Stop=1" type="text" />
<content type="application/xml">
<mproperties>
<d id1 mtype="Edm.String">52</d><d id1>
<d name1 mtype="Edm.String">52</d><d name1>
<d id2 mtype="Edm.String"></d><d id2>
<d name2 mtype="Edm.String"></d><d name2>
<d rating mtype="Edm.Double">0.533343480387708</d><d rating>
<d reasoning mtype="Edm.String">People who bought '1003' also bought '52'</d><d reasoning>
</mproperties>
<content>
<entry>
<id href="https://api.datamarket.azure.com/aml/recommendations/v3/ItemFbtRecommend?modelId=2779c063-48fb-46c1-bae3-74acddc8&id1='1003'&numberOfResults=3&minimalScore=0.1&includeMetadata=false&apiVersion=1.0&$skip=2&Stop=1" type="text" />
<title type="text">GetFbtRecommendationEntity</title>
<updated>2014-10-05T12:28:48Z</updated>
<link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/ItemFbtRecommend?modelId=2779c063-48fb-46c1-bae3-74acddc8&id1='1003'&numberOfResults=3&minimalScore=0.1&includeMetadata=false&apiVersion=1.0&$skip=2&Stop=1" type="text" />
<content type="application/xml">
<mproperties>
<d id1 mtype="Edm.String">35</d><d id1>
<d name1 mtype="Edm.String">35</d><d name1>
<d id2 mtype="Edm.String">102</d><d id2>
<d name2 mtype="Edm.String">102</d><d name2>
<d rating mtype="Edm.Double">0.523343480387708</d><d rating>
<d reasoning mtype="Edm.String">People who bought '1003' also bought '35' and '102'</d><d reasoning>
</mproperties>
<content>
<entry>
<id href="rootURI%2fItemFbtRecommend?modelId=%27modelId%27&id1=%27itemID%27&numberOfResults=1&minimalScore=0.1&includeMetadata=false&buildId=172779c063-48fb-46c1-bae3-74acddc8&id1='1003'&numberOfResults=1&minimalScore=0.1&includeMetadata=false&buildId=17234&apiVersion=0.5" type="text" />
<title type="text">GetFbtRecommendationEntity</title>
<updated>2014-10-05T12:28:48Z</updated>
<link rel="self" href="rootURI%2fItemFbtRecommend?modelId=%27modelId%27&id1=%27itemID%27&numberOfResults=1&minimalScore=0.1&includeMetadata=false&buildId=172779c063-48fb-46c1-bae3-74acddc8&id1='1003'&numberOfResults=1&minimalScore=0.1&includeMetadata=false&buildId=17234&apiVersion=0.5" type="text" />
<content type="application/xml">
```

12.4. Get FBT Recommendations (of a specific build)

Get recommendations of a specific build of type "Fbt".

HTTP METHOD	URI
GET	<pre>rootURI%2fItemFbtRecommend?modelId=%27modelId%27&id1=%27itemID%27&numberOfResults=1&minimalScore=0.1&includeMetadata=false&buildId=172779c063-48fb-46c1-bae3-74acddc8&id1='1003'&numberOfResults=1&minimalScore=0.1&includeMetadata=false&buildId=17234&apiVersion=0.5</pre> <p>Example:</p> <pre>rootURI%2fItemFbtRecommend?modelId=%27modelId%27&id1=%27itemID%27&numberOfResults=1&minimalScore=0.1&includeMetadata=false&buildId=172779c063-48fb-46c1-bae3-74acddc8&id1='1003'&numberOfResults=1&minimalScore=0.1&includeMetadata=false&buildId=17234&apiVersion=0.5</pre>

PARAMETER NAME	VALID VALUES
modelId	Unique identifier of the model
itemId	Item to recommend for. Max length: 1024
numberOfResults	Number of required results Max: 150
minimalScore	Minimal score that a frequent set should have in order to be included in the returned results
includeMetadata	Future use, always false
buildId	the build id to use for this recommendation request
apiVersion	1.0

Response:

HTTP Status code: 200

The response includes one entry per recommended item set (a set of items which are usually bought together with the seed/input item). Each entry has the following data:

- `[seed entry content properties Id]` - Recommended item ID.
- `[seed entry content properties Name]` - Name of the item.
- `[seed entry content properties Id2]` - 2nd recommended item ID (optional).
- `[seed entry content properties Name2]` - Name of the 2nd item (optional).
- `[seed entry content properties Rating]` - Rating of the recommendation; higher number means higher confidence.
- `[seed entry content properties Reasoning]` - Recommendation reasoning (e.g. recommendation explanations).

See a response example in 12.3

12.5. Get User Recommendations (for active build)

Get user recommendations of a build of type "Recommendation" marked as active build.

The API will return a list of predicted item according to the usage history of the user.

Notes:

1. There is no user recommendation for FBT build.
2. If the active build is FBT this method will returns an error.

HTTP METHOD	URI
GET	<code><root URI>/UserRecommendation?modelId=%27&modelType=fbt&userId=%27&userId=%27&numberOfResults=%27&includeMetadata=false&apiVersion=1.0%27</code>

Example:

```
<root URI>/UserRecommendation?modelId=%27&modelType=fbt&userId=%27&userId=%27&numberOfResults=%27&includeMetadata=false&apiVersion=1.0%27
```

PARAMETER NAME	VALID VALUES
modelId	Unique identifier of the model
userId	Unique identifier of the user
numberOfResults	Number of required results
includeMetadata	Future use, always false
apiVersion	1.0

Response:

HTTP Status code: 200

The response includes one entry per recommended item. Each entry has the following data:

- `[seed entry content properties Id]` - Recommended item ID.
- `[seed entry content properties Name]` - Name of the item.
- `[seed entry content properties Rating]` - Rating of the recommendation; higher number means higher confidence.
- `[seed entry content properties Reasoning]` - Recommendation reasoning (e.g. recommendation explanations).

See a response example in 12.1

12.6. Get User Recommendations with item list (for active build)

Get user recommendations of a build of type "Recommendation" marked as active build with an additional list of items

The API will return a list of predicted item according to the usage history of the user and the additional provided items.

Notes:

1. There is no user recommendation for FBT build.
2. If the active build is FBT this method will returns an error.

HTTP METHOD	URI
-------------	-----

HTTP METHOD	URI
GET	<rootURI>/UserRecommend?modelId=<int>&userId=<int>&itemIds=<int>&numberOfResults=<int>&includeMetadata=<bool>&apiVersion=<float>
	Example: <rootURI>/UserRecommend?modelId=1&userId=1000&itemIds=1000&numberOfResults=10&includeMetadata=false&apiVersion=1.0
PARAMETER NAME	VALID VALUES
modelId	Unique identifier of the model
userId	Unique identifier of the user
itemIds	Comma-separated list of the items to recommend for. Max length: 1024
numberOfResults	Number of required results
includeMetadata	Future use, always false
apiVersion	1.0

Response:

HTTP Status code: 200

The response includes one entry per recommended item. Each entry has the following data:

- Recommended item ID.
- Name of the item.
- Rating of the recommendation; higher number means higher confidence.
- Recommendation reasoning (e.g. recommendation explanations).

See a response example in 12.1

12.7. Get User Recommendations (of a specific build)

Get user recommendations of a specific build of type "Recommendation".

The API will return a list of predicted item according to the usage history of the user (used in the specific build).

Note: There is no user recommendation for FBT build.

HTTP METHOD	URI
GET	<rootURI>/UserRecommend?modelId=<int>&userId=<int>&numberOfResults=<int>&includeMetadata=<bool>&buildId=<int>&apiVersion=<float>
	Example: <rootURI>/UserRecommend?modelId=1&userId=1000&numberofResults=10&includeMetadata=false&buildId=5001&apiVersion=1.0
PARAMETER NAME	VALID VALUES
modelId	Unique identifier of the model
userId	Unique identifier of the user
numberOfResults	Number of required results
includeMetadata	Future use, always false
buildId	the build id to use for this recommendation request
apiVersion	1.0

Response:

HTTP Status code: 200

The response includes one entry per recommended item. Each entry has the following data:

- Recommended item ID.
- Name of the item.
- Rating of the recommendation; higher number means higher confidence.
- Recommendation reasoning (e.g. recommendation explanations).

See a response example in 12.1

12.8. Get User Recommendations with item list (of a specific build)

Get user recommendations of a specific build of type "Recommendation" and the list of additional items.

The API will return a list of predicted item according to the usage history of the user and the additional list of items.

Note: There is no user recommendation for FBT build.

HTTP METHOD	URI

HTTP METHOD	URI
GET	<root URI>/UserRecommend? modelId=<modelId>&userId=<userId>&itemIds=<itemIds>&numberOResults=<int>&includeMetadata=<bool>&buildId=<int>&apiVersion=<271.0>
	Example: <root URI>/UserRecommend?modelId=1&userId=1001001278&itemIds=1003&numberOResults=10&includeMetadata=false&buildId=50012&apiVersion=271.0
PARAMETER NAME	VALID VALUES
modelId	Unique identifier of the model
userId	Unique identifier of the user
itemIds	Comma-separated list of the items to recommend for. Max length: 1024
numberOfResults	Number of required results
includeMetadata	Future use, always false
buildId	the build id to use for this recommendation request
apiVersion	1.0

Response:

HTTP Status code: 200

The response includes one entry per recommended item. Each entry has the following data:

- `entry/content/properties/Id` - Recommended item ID.
- `entry/content/properties/Name` - Name of the item.
- `entry/content/properties/Rating` - Rating of the recommendation; higher number means higher confidence.
- `entry/content/properties/Reasoning` - Recommendation reasoning (e.g. recommendation explanations).

See a response example in 12.1

13. User Usage History

Once a recommendation model was built the system will allow to retrieve the user history (items associated to a specific user) used for the build. This API allow to retrieve the user history

Note: the user history is currently available only for recommendation builds.

13.1 Retrieve user history

Retrieve the list of item used in the active build or in the specified build for the given user id.

HTTP METHOD	URI
GET	Get the user history for the active build. <root URI>/GetUserHistory?modelId=<modelId>&userId=<userId>&apiVersion=<271.0>
	Get the user history for the given build <root URI>/GetUserHistory?modelId=<modelId>&userId=<userId>&buildId=<buildId>&apiVersion=<271.0>
	Example: <root URI>/GetUserHistory?modelId=1&userId=1001001278&buildId=1013&apiVersion=271.0
PARAMETER NAME	VALID VALUES
modelId	the unique identifier of the model.
userId	the unique identifier of the user.
buildId	optional parameter, allow to indicate from which build the user history should be fetch
apiVersion	1.0

Response:

HTTP Status code: 200

The response includes one entry per recommended item. Each entry has the following data:

- `entry/content/properties/Id` - Recommended item ID.
- `entry/content/properties/Name` - Name of the item.
- `entry/content/properties/Rating` - N/A.
- `entry/content/properties/Reasoning` - N/A.

OData XML

```

<feed xmlns:base="https://api.datamarket.azure.com/aml/recommendations/v3/GetUserHistory" xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
  xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns="http://www.w3.org/2005/Atom">
  <title type="text" />
  <subtitle type="text">Get User History</subtitle>
  <id href="https://api.datamarket.azure.com/aml/recommendations/v3/GetUserHistory?modelId=2779c063-48fb-46c1-bac3-74acddc8c1d1&userId=u_1013&apiVersion=1.0" />
  <rights type="text" />
  <updated>2015-05-26T15:32:47Z</updated>
  <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetUserHistory?modelId=2779c063-48fb-46c1-bac3-74acddc8c1d1&userId=u_1013&apiVersion=1.0" />
  <entry>
    <id href="https://api.datamarket.azure.com/aml/recommendations/v3/GetUserHistory?modelId=2779c063-48fb-46c1-bac3-74acddc8c1d1&userId=u_1013&apiVersion=1.0&skip=0&stop=1" />
    <title type="text">CatalogItemsThatContainATokenEntity</title>
    <updated>2015-05-26T15:32:47Z</updated>
    <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetUserHistory?modelId=2779c063-48fb-46c1-bac3-74acddc8c1d1&userId=u_1013&apiVersion=1.0&skip=0&stop=1" />
    <content type="application/xml">
      <m:properties>
        <d:Id mtype="Edm.String">2406E770-769C-4189-89DE-1C9283F93A96</d:Id>
        <d:Name mtype="Edm.String">Clara Callan</d:Name>
        <d:Rating mtype="Edm.Double">0</d:Rating>
        <d:Reasoning mtype="Edm.String"/>
        <d:Metadata mtype="Edm.String"/>
        <d:FormattedRating mtype="Edm.Double" m:null="true"/>
      </m:properties>
    </content>
  </entry>

```

14. Notifications

Azure Machine Learning Recommendations creates notifications when persistent errors happen in the system. There are 3 types of notifications:

1. Build failure - This notification is triggered for every build failure.
2. Data acquisition processing failure - This notification is triggered when we have more than 100 errors in the last 5 minutes in the processing of usage events per model.
3. Recommendation consumption failure - This notification is triggered when we have more than 100 errors in the last 5 minutes in the processing of recommendation requests per model.

14.1. Get Notifications

Retrieves all the notifications for all models or for a single model.

HTTP METHOD	URI
GET	<code>/rootURI/GetNotifications?modelId=%27[model_id%27]&apiVersion=%271.0%27</code> Getting all notifications for all models: <code>/rootURI/GetNotifications?modelId=&apiVersion=%271.0%27</code> Example for getting notifications for a specific model: <code>/rootURI/GetNotifications?modelId=%2796713608-1808-4258-9331-10d567f87fae%27&apiVersion=%271.0%27</code>
PARAMETER NAME	VALID VALUES
modelId	Optional parameter. When omitted, you will get all notifications for all models. Valid value: unique identifier of the model.
apiVersion	1.0
Request Body	NONE

Response:

HTTP Status code: 200

OData XML

```

The response includes one entry per notification. Each entry has the following data:
* feed\entry\content\properties\UserName - Internal user name identification.
* feed\entry\content\properties\ModelId - Model ID.
* feed\entry\content\properties\Message - Notification message.
* feed\entry\content\properties\DateCreated - Date that this notification was created in UTC format.
* feed\entry\content\properties\NotificationType - Notification types. Values are BuildFailure, RecommendationFailure, and DataAquisitionFailure.

<feed xmlns:base="https://api.datamarket.azure.com/aml/recommendations/v3/GetNotifications" xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
  xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns="http://www.w3.org/2005/Atom">
  <title type="text" />
  <subtitle type="text">Get a list of Notifications for a user</subtitle>
  <id href="https://api.datamarket.azure.com/aml/recommendations/v3/GetNotifications?modelId=967136e8-f868-4258-9331-10d567f87fae&apiVersion=1.0" />
  <rights type="text" />
  <updated>2014-11-04T13:24:23Z</updated>
  <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetNotifications?modelId=967136e8-f868-4258-9331-10d567f87fae&apiVersion=1.0" />
  <entry>
    <id href="https://api.datamarket.azure.com/aml/recommendations/v3/GetNotifications?modelId=967136e8-f868-4258-9331-10d567f87fae&apiVersion=1.0&skip=0&stop=1" />
    <title type="text">GetListOfNotificationsForAUserEntity</title>
    <updated>2014-11-04T13:24:23Z</updated>
    <link rel="self" href="https://api.datamarket.azure.com/aml/recommendations/v3/GetNotifications?modelId=967136e8-f868-4258-9331-10d567f87fae&apiVersion=1.0&skip=0&stop=1" />
    <content type="application/xml">
      <m:properties>
        <d:UserName mtype="Edm.String">515506bc-3693-4dce-a5e2-81cb3e8efb56@dm.com</d:UserName>
        <d:ModelId mtype="Edm.String">967136e8-f868-4258-9331-10d567f87fae</d:ModelId>
        <d:Message mtype="Edm.String">Build failed for user</d:Message>
        <d:DateCreated mtype="Edm.String">2014-11-04T13:23:14.383</d:DateCreated>
        <d:NotificationType mtype="Edm.String">BuildFailure</d:NotificationType>
      </m:properties>
    </content>
  </entry>
</feed>

```

14.2. Delete Model Notifications

Deletes all read notifications for a model.

HTTP METHOD	URI
DELETE	<root URI>/DeleteModelNotifications?modelId=%e-model_id%27&apiVersion=%271.0%27
Example:	<root URI>/DeleteModelNotifications?modelId=%27967136e8-0608-4258-9331-10d56787fe9b%27&apiVersion=%271.0%27
PARAMETER NAME	VALID VALUES
modelId	Unique identifier of the model
apiVersion	1.0
Request Body	NONE

Response:

HTTP Status code: 200

14.3. Delete User Notifications

Deletes all notifications for all models.

HTTP METHOD	URI
DELETE	<root URI>/DeleteUserNotifications?apiVersion=%271.0%27
PARAMETER NAME	VALID VALUES
apiVersion	1.0
Request Body	NONE

Response:

HTTP Status code: 200

15. Legal

This document is provided "as-is". Information and views expressed in this document, including URL and other Internet Web site references, may change without notice.

Some examples depicted herein are provided for illustration only and are fictitious. No real association or connection is intended or should be inferred.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

© 2015 Microsoft. All rights reserved.

Azure Machine Learning Recommendations - JavaScript Integration

1/17/2017 • 7 min to read • [Edit on GitHub](#)

NOTE

You should start using the Recommendations API Cognitive Service instead of this version. The Recommendations Cognitive Service will be replacing this service, and all the new features will be developed there. It has new capabilities like batching support, a better API Explorer, a cleaner API surface, more consistent signup/billing experience, etc. Learn more about [Migrating to the new Cognitive Service](#)

This document depicts how to integrate your site using JavaScript. The JavaScript enables you to send Data Acquisition events and to consume recommendations once you build a recommendation model. All operations done via JS can be also done from server side.

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

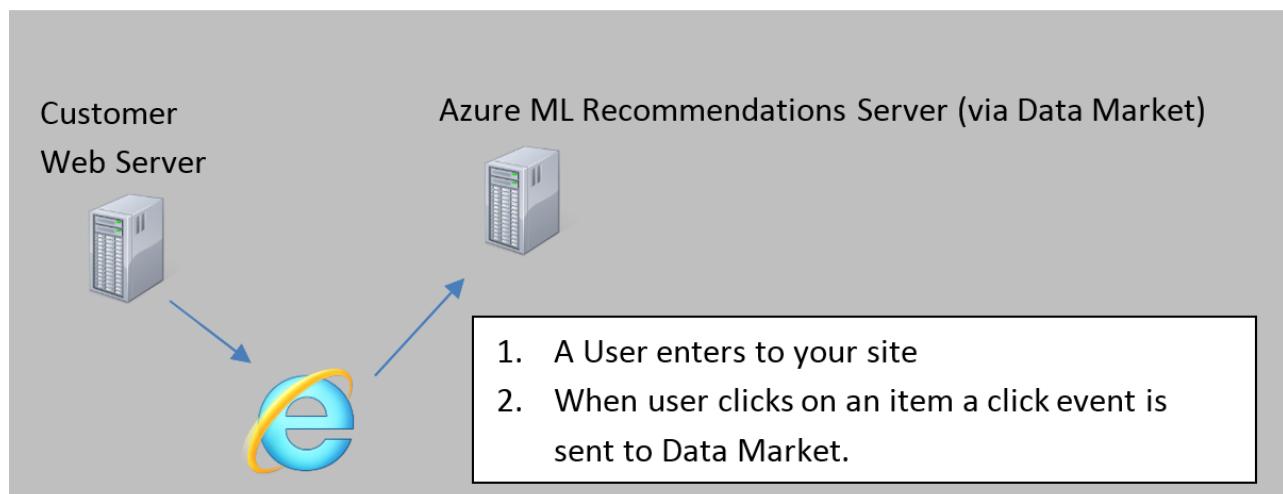
1. General Overview

Integrating your site with Azure ML Recommendations consist on 2 Phases:

1. Send Events into Azure ML Recommendations. This will enable to build a recommendation model.
2. Consume the recommendations. After the model is built you can consume the recommendations. (This document does not explain how to build a model, read the quick start guide to get more information on how).

Phase I

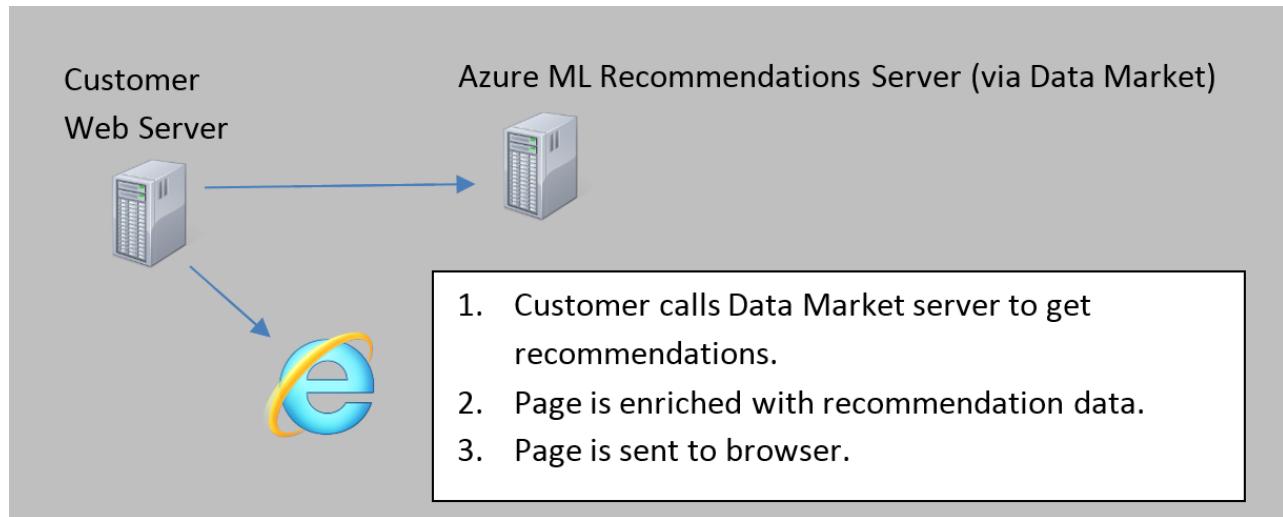
In the first phase you insert into your html pages a small JavaScript library that enables the page to send events as they occur on the html page into Azure ML Recommendations servers (via Data Market):



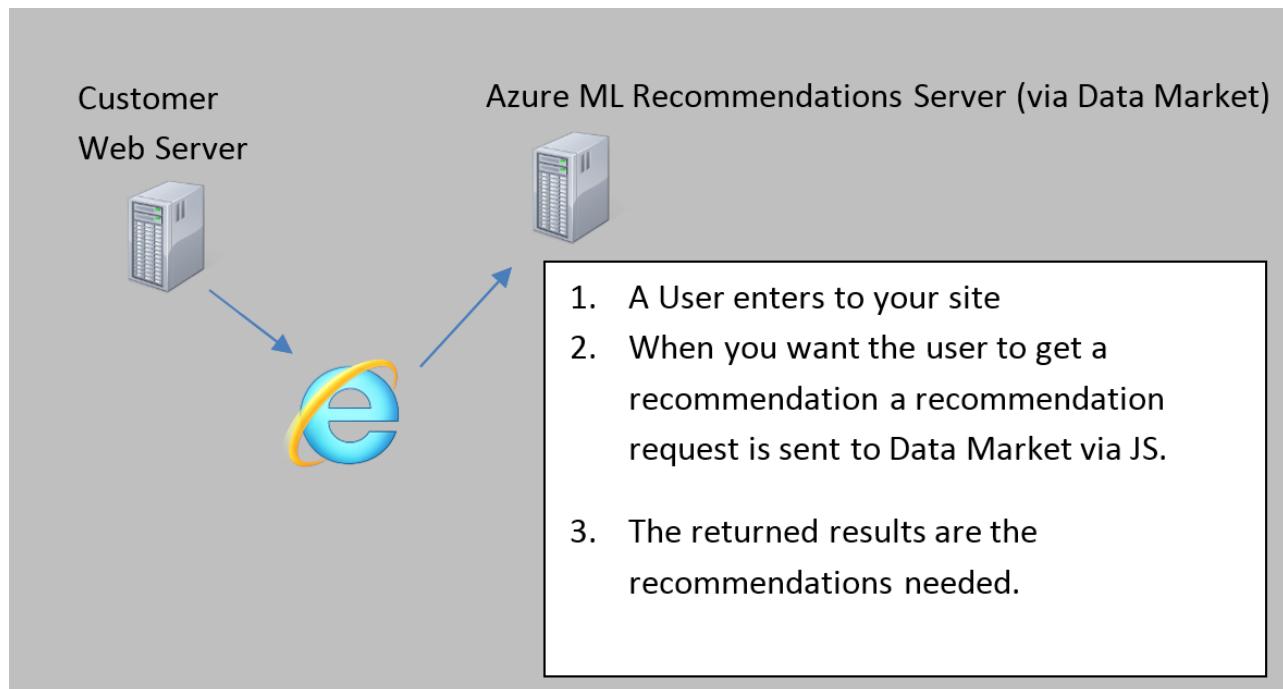
Phase II

In the second phase when you want to show the recommendations on the page you select one of the following options:

1.Your server (on the phase of page rendering) calls Azure ML Recommendations Server (via Data Market) to get recommendations. The results include a list of items id. Your server needs to enrich the results with the items Meta data (e.g. images, description) and send the created page to the browser.



2.The other option is to use the small JavaScript file from phase one to get a simple list of recommended items. The data received here is leaner than the one in the first option.



2. Prerequisites

1. Create a new model using the APIs. See the Quick start guide on how to do it.
2. Encode your <dataMarketUser>:<dataMarketKey> with base64. (This will be used for the basic authentication to enable the JS code to call the APIs).

3. Send Data Acquisition events using JavaScript

The following steps facilitate sending events:

1. Include JQuery library in your code. You can download it from nuget in the following URL.

<http://www.nuget.org/packages/jQuery/1.8.2>

2. Include the Recommendations Java Script library from the following URL: <http://aka.ms/RecoJSLib1>
3. Initialize Azure ML Recommendations library with the appropriate parameters.
4. Send the appropriate event. See detailed section below on all type of events (example of click event)

3.1. Limitations and Browser Support

This is a reference implementation and it is given as is. It should support all major browsers.

3.2. Type of Events

There are 5 types of event that the library supports: Click, Recommendation Click, Add to Shop Cart, Remove from Shop Cart and Purchase. There is an additional event that is used to set the user context called Login.

3.2.1. Click Event

This event should be used any time a user clicked on an item. Usually when user clicks on an item a new page is opened with the item details; in this page this event should be triggered.

Parameters:

- event (string, mandatory) - "click"
- item (string, mandatory) - Unique identifier of the item
- itemName (string, optional) - the name of the item
- itemDescription (string, optional) - the description of the item
- itemCategory (string, optional) - the category of the item

```
<script>
  if(typeof AzureMLRecommendationsEvent == "undefined") { AzureMLRecommendationsEvent = []; }
  AzureMLRecommendationsEvent.push({ event: "click", item: "3111718" });
</script>
```

Or with optional data:

```
<script>
  if(typeof AzureMLRecommendationsEvent === "undefined") { AzureMLRecommendationsEvent = []; }
  AzureMLRecommendationsEvent.push({event: "click", item: "3111718", itemName: "Plane", itemDescription: "It is a big plane", itemCategory: "Aviation"});
</script>
```

3.2.2. Recommendation Click Event

This event should be used any time a user clicked on an item that was received from Azure ML Recommendations as a recommended item. Usually when user clicks on an item a new page is opened with the item details; in this page this event should be triggered.

Parameters:

- event (string, mandatory) - "recommendationclick"
- item (string, mandatory) - Unique identifier of the item
- itemName (string, optional) - the name of the item
- itemDescription (string, optional) - the description of the item
- itemCategory (string, optional) - the category of the item
- seeds (string array, optional) - the seeds that generated the recommendation query.
- recoList (string array, optional) - the result of the recommendation request that generated the item that was clicked.

```
<script>
if (typeof AzureMLRecommendationsEvent == "undefined") { AzureMLRecommendationsEvent = []; }
AzureMLRecommendationsEvent.push({event: "recommendationclick", item: "18899918" });
</script>
```

Or with optional data:

```
<script>
if (typeof AzureMLRecommendationsEvent == "undefined") { AzureMLRecommendationsEvent = []; }
AzureMLRecommendationsEvent.push({ event: eventName, item: "198", itemName: "Plane2", itemDescription: "It is a big plane2", itemCategory: "Default2", seeds: ["Seed1", "Seed2"], recoList: ["199", "198", "197"] });
</script>
```

3.2.3. Add Shopping Cart Event

This event should be used when the user add an item to the shopping cart. Parameters:

- event (string, mandatory) - "addshopcart"
- item (string, mandatory) - Unique identifier of the item
- itemName (string, optional) - the name of the item
- itemDescription (string, optional) - the description of the item
- itemCategory (string, optional) - the category of the item

```
<script>
if (typeof AzureMLRecommendationsEvent == "undefined") { AzureMLRecommendationsEvent = []; }
AzureMLRecommendationsEvent.push({event: "addshopcart", item: "1322118" });
</script>
```

3.2.4. Remove Shopping Cart Event

This event should be used when the user removes an item to the shopping cart.

Parameters:

- event (string, mandatory) - "removeshopcart"
- item (string, mandatory) - Unique identifier of the item
- itemName (string, optional) - the name of the item
- itemDescription (string, optional) - the description of the item
- itemCategory (string, optional) - the category of the item

```
<script>
if (typeof AzureMLRecommendationsEvent == "undefined") { AzureMLRecommendationsEvent = []; }
AzureMLRecommendationsEvent.push({ event: "removeshopcart", item: "111118" });
</script>
```

3.2.5. Purchase Event

This event should be used when the user purchased his shopping cart.

Parameters:

- event (string) - "purchase"
- items (Purchased[]) - Array holding an entry for each item purchased.

Purchased format:

- item (string) - Unique identifier of the item.
- count (int or string) - number of items that were purchased.
- price (float or string) - optional field - the price of the item.

The example below shows purchase of 3 items (33, 34, 35), two with all fields populated (item, count, price) and one (item 34) without a price.

```
<script>
  if( typeof AzureMLRecommendationsEvent == "undefined") { AzureMLRecommendationsEvent = []; }
  AzureMLRecommendationsEvent.push({ event: "purchase", items: [{ item: "33", count: "1", price: "10" }, { item: "34", count: "2" }, { item: "35", count: "1", price: "210" }] });
</script>
```

3.2.6. User Login Event

Azure ML Recommendations Event library creates and use a cookie in order to identify events that came from the same browser. In order to improve the model results Azure ML Recommendations enables to set a user unique identification that will override the cookie usage.

This event should be used after the user login to your site.

Parameters:

- event (string) - "userlogin"
- user (string) - unique identification of the user.

```
<script>
  if(typeof AzureMLRecommendationsEvent=="undefined") { AzureMLRecommendationsEvent = []; }
  AzureMLRecommendationsEvent.push({event: "userlogin", user: "ABCD10AA"});
</script>
```

4. Consume Recommendations via JavaScript

The code that consumes the recommendation is triggered by some JavaScript event by the client's webpage. The recommendation response includes the recommended items Ids, their names and their ratings. It's best to use this option only for a list display of the recommended items - more complex handling (such as adding the item's metadata) should be done on the server side integration.

4.1 Consume Recommendations

To consume recommendations you need to include the required JavaScript libraries in your page and to call AzureMLRecommendationsStart. See section 2.

To consume recommendations for one or more items you need to call a method called:
AzureMLRecommendationsGetI2IRecommendation.

Parameters:

- items (array of strings) - One or more items to get recommendations for. If you consume an Fbt build then you can set here only one item.
- numberOfRowsInSection (int) - number of required results.
- includeMetadata (boolean, optional) - if set to 'true' indicates that the metadata field must be populated in the result.
- Processing function - a function that will handle the recommendations returned. The data is returned as an array of:
 - Item - item unique id
 - name - item name (if exist in catalog)
 - rating - recommendation rating
 - metadata - a string that represents the metadata of the item

Example: The following code requests 8 recommendations for item "64f6eb0d-947a-4c18-a16c-888da9e228ba" (and by not specifying includeMetadata - it implicitly says that no metadata is required), it then concatenate the

results into a buffer.

```
<script>
var reco = AzureMLRecommendationsGetI2IRecommendation(["64f6eb0d-947a-4c18-a16c-888da9e228ba"], 8, false, function (reco) {
    var buff="";
    for (var ii=0; ii<reco.length; ii++) {
        buff+=reco[ii].item + "," + reco[ii].name + "," + reco[ii].rating + "\n";
    }
    alert(buff);
});
</script>
```

Setting up and using Machine Learning Recommendations API FAQ

1/17/2017 • 5 min to read • [Edit on GitHub](#)

What is RECOMMENDATIONS?

NOTE

You should start using the Recommendations API Cognitive Service instead of this version. The Recommendations Cognitive Service will be replacing this service, and all the new features will be developed there. It has new capabilities like batching support, a better API Explorer, a cleaner API surface, more consistent signup/billing experience, etc. Learn more about [Migrating to the new Cognitive Service](#)

For organizations and businesses that rely on recommendations to cross-sell and up-sell products and services to their customers, RECOMMENDATIONS in Azure Machine Learning provides a self-service recommendations engine. It is an implementation of collaborative filtering that uses matrix factorization as its core algorithm. Application developers can access RECOMMENDATIONS by using REST APIs.

NOTE

Try Azure Machine Learning for free

No credit card or Azure subscription needed. [Get started now >](#)

What can I do with RECOMMENDATIONS?

RECOMMENDATIONS takes as input an item or a set of items and returns a list of relevant recommendations. For example: A customer of an online retailer clicks a product. The online retailer sends that product as input to RECOMMENDATIONS, gets a list of products in return, and decides which of these products will be shown to the customer. You may want to use RECOMMENDATIONS to optimize your online store or even to inform your inside sales department or call center.

Are there any usage limitations?

Recommendations has the following usage limitations:

- Maximum number of models per subscription: 10
- Maximum number of items that a catalog can hold: 100,000
- The maximum number of usage points that are kept is ~5,000,000. The oldest will be deleted if new ones will be uploaded or reported.
- Maximum size of data that can be sent in email (for example, import catalog data, import usage data) is 200 MB
- Number of transactions per second (TPS) for a Recommendations model build that is not active is ~2 TPS. A Recommendations model build that is active can hold up to 20 TPS.

Purchase and Billing

How much does Recommendations cost during the launch period?

Recommendations is a subscription-based service. Charging is based on volume of transactions per month. You can check the [offer page](#) in Microsoft Azure Marketplace for pricing information.

Are there any costs associated with having Recommendations track and store user activity for me?

Not at the moment.

Does Recommendations have a free trial?

There is a free trial which is restricted to 10,000 transactions per month.

When will I be billed for Recommendations?

A paid subscription is any subscription for which there is a monthly fee. When you purchase a paid subscription, you are immediately charged for the first month's use. You are charged the amount that is associated with the offer on the subscription page (plus applicable taxes). This monthly charge is made each month on the same calendar date as your original purchase until you cancel the subscription.

How do I upgrade to a higher tier service?

You can buy or update your subscription from the [offer page](#) page on Microsoft Azure Marketplace.

When you upgrade a subscription:

- Transactions that are remaining on your old subscription are not added to your new subscription.
- You pay full price for the new subscription, even though you have unused transactions on your old subscription.

Process to upgrade a subscription:

- Navigate to the [offer page](#).
- Sign in to the Marketplace if you aren't already Signed in.
- In the right pane, all the available plans are listed. Click the radio button for the plan you want to upgrade to.
- If you want to upgrade, click **OK**. If you do not want to upgrade, click **Cancel**.

Important Carefully read the dialog box before you upgrade because there are billing and use implications.

When will my subscription to Recommendations end?

Your subscription will end when you cancel it. If you would like to cancel your subscriptions, see the following instructions.

How do I cancel my Recommendations subscription?

To cancel your subscription, use the following steps. If your current subscription is a paid subscription, your subscription continues in effect until the end of the current billing period. If you need the cancellation to be effective immediately, contact us at [Microsoft Support](#).

Note No refund is given if you cancel before the end of a billing period or for unused transactions in a billing period.

- Navigate to the [offer page](#).
- Sign in to the Marketplace if you aren't already Signed in.
- Click **Cancel** to the right of the dataset name and status. You can use this subscription until the end of the current billing period or your transaction limit is reached (whichever occurs first).

If you would like to cancel your subscription immediately so you can purchase a new subscription, file a ticket at [Microsoft Support](#).

Getting started with Recommendations

Is Recommendations for me?

Recommendations in Machine Learning is for organizations and businesses that rely on recommendations to cross-

sell and up-sell products or services to their customers. If you have a customer-facing website, a sales force, an inside sales force, or a call center, and if you offer a catalog of more than a few dozen products or services, your bottom line may benefit from using Recommendations.

Experimenting with Recommendations is designed to be fairly simple. The current API-based version requires basic programming skills. If you need assistance, contact the vendor who developed your website. If you have an internal IT department or an in-house developer, they should be able to get Recommendations to work for you.

What are the prerequisites for setting up Recommendations?

Recommendations requires that you have a log of user choices as it relates to your catalog. If you don't have such a log and you do have a customer facing website, Recommendations can collect user activity for you.

Recommendations also requires a catalog of your products or services. If you don't have the catalog, Recommendations can use the actual customer usage data and distill a catalog. An implied catalog will not include items that were not reported as part of user transactions.

How do I set up Recommendations for the first time?

After [subscribing](#) to Recommendations, you should use the API documentation in the [Azure Machine Learning Recommendations - Quick Start Guide](#) to set up the service.

Where can I find API documentation?

The API documentation is [Azure Machine Learning Recommendations -Quick Start Guide](#).

What options do I have to upload catalog and usage data to Recommendations?

You have two options for uploading your catalog and usage data: You can export the data from your CRM system or other logs and upload it to Recommendations, or you can add tags to your website that will track user activities. If you use the latter method, the data will be stored in Azure.

Maintenance and support

How large can my data set be?

Each data set can contain up to 100,000 catalog items and up to 2048 MB of usage data. In addition, a subscription can contain up to 10 data sets (models).

Where can I get technical support for Recommendations?

Technical support is available on the [Microsoft Azure Support](#) site.

Where can I find the terms of use?

[Microsoft Azure Machine Learning Recommendations API Terms of Service](#).

Cortana Intelligence Solution Template Playbook for predictive maintenance in aerospace and other businesses

1/17/2017 • 48 min to read • [Edit on GitHub](#)

Executive summary

Predictive maintenance is one of the most demanded applications of predictive analytics with unarguable benefits including tremendous amount of cost savings. This playbook aims at providing a reference for predictive maintenance solutions with the emphasis on major use cases. It is prepared to give the reader an understanding of the most common business scenarios of predictive maintenance, challenges of qualifying business problems for such solutions, data required to solve these business problems, predictive modeling techniques to build solutions using such data and best practices with sample solution architectures. It also describes the specifics of the predictive models developed such as feature engineering, model development and performance evaluation. In essence, this playbook brings together the business and analytical guidelines needed for a successful development and deployment of predictive maintenance solutions. These guidelines are prepared to help the audience create an initial solution using Cortana Intelligence Suite and specifically Azure Machine Learning as a starting point in their long-term predictive maintenance strategy. The documentation regarding Cortana Intelligence Suite and Azure Machine Learning can be found in [Cortana Analytics](#) and [Azure Machine Learning](#) pages.

TIP

For a technical guide to implementing this Solution Template, see [Technical guide to the Cortana Intelligence Solution Template for predictive maintenance](#). To download a diagram that provides an architectural overview of this template, see [Architecture of the Cortana Intelligence Solution Template for predictive maintenance](#).

Playbook overview and target audience

This playbook is organized to benefit both technical and non-technical audience with varying backgrounds and interests in predictive maintenance space. The playbook covers both high-level aspects of the different types of predictive maintenance solutions and details of how to implement them. The content is balanced to cater both to the audience who are only interested in understanding the solution space and the type of applications as well as those who are looking to implement these solutions and are hence interested in the technical details.

Majority of the content in this playbook does not assume prior data science knowledge or expertise. However, some parts of the playbook will require somewhat familiarity with data science concepts to be able to follow implementation details. Introductory level data science skills are required to fully benefit from the material in those sections.

The first half of the playbook covers an introduction to predictive maintenance applications, how to qualify a predictive maintenance solution, a collection of common use cases with the details of the business problem, the data surrounding these use cases and the business benefits of implementing these predictive maintenance solutions. These sections don't require any technical knowledge in the predictive analytics domain.

In the second half of the playbook, we cover the types of predictive modeling techniques for predictive maintenance applications and how to implement these models through examples from the use cases outlined in the first half of the playbook. This is illustrated by going through the steps of data preprocessing such as data labeling and feature engineering, model selection, training/testing and performance evaluation best practices. These sections are

suitable for technical audience.

Predictive maintenance in IoT

The impact of unscheduled equipment downtime can be extremely destructive for businesses. It is critical to keep field equipment running in order to maximize utilization and performance and by minimizing costly, unscheduled downtime. Simply, waiting for the failure to occur is not affordable in today's business operations scene. To remain competitive, companies look for new ways to maximize asset performance by making use of the data collected from various channels. One important way to analyze such information is to utilize predictive analytic techniques that use historical patterns to predict future outcomes. One of the most popular of these solutions is called Predictive Maintenance which can generally be defined as but not limited to predicting possibility of failure of an asset in the near future so that the assets can be monitored to proactively identify failures and take action before the failures occur. These solutions detect failure patterns to determine assets that are at the greatest risk of failure. This early identification of issues helps deploy limited maintenance resources in a more cost-effective way and enhance quality and supply chain processes.

With the rise of the Internet of Things (IoT) applications, predictive maintenance has been gaining increasing attention in the industry as the data collection and processing technologies has matured enough to generate, transmit, store and analyze all kinds of data in batches or in real-time. Such technologies enable easy development and deployment of end-to-end solutions with advanced analytics solutions, with predictive maintenance solutions providing arguably the largest benefit.

Business problems in the predictive maintenance domain range from high operational risk due to unexpected failures and limited insight into the root cause of problems in complex business environments. The majority of these problems can be categorized to fall under the following business questions:

- What is the probability that a piece of equipment fails in the near future?
- What is the remaining useful life of the equipment?
- What are the causes of failures and what maintenance actions should be performed to fix these issues?

By utilizing predictive maintenance to answer these questions, businesses can:

- Reduce operational risk and increase rate of return on assets by spotting failures before they occurred
- Reduce unnecessary time-based maintenance operations and control cost of maintenance
- Improve overall brand image, eliminate bad publicity and resulting lost sales from customer attrition.
- Lower inventory costs by reducing inventory levels by predicting the reorder point
- Discover patterns connected to various maintenance problems

Predictive maintenance solutions can provide businesses with key performance indicators such as health scores to monitor real-time asset condition, an estimate of the remaining lifespan of assets, recommendation for proactive maintenance activities and estimated order dates for replacement of parts.

Qualification criteria for predictive maintenance

It is important to emphasize that not all use cases or business problems can be effectively solved by predictive maintenance. Important qualification criteria include whether the problem is predictive in nature, that a clear path of action exists in order to prevent failures when they are detected beforehand and most importantly, data with sufficient quality to support the use case is available. Here, we focus on the data requirements for building a successful predictive maintenance solution.

When building predictive models, we use historical data to train the model which can then recognize hidden patterns and further identify these patterns in the future data. These models are trained with examples described by their features and the target of prediction. The trained model is expected to make predictions on the target by only looking at the features of the new examples. It is crucial that the model capture the relationship between features and the target of prediction. In order to train an effective machine learning model, we need training data which

includes features that actually have predictive power towards the target of prediction meaning the data should be relevant to the prediction goal to expect accurate predictions.

For example, if the target is to predict failures of train wheels, the training data should contain wheel-related features (e.g. telemetry reflecting the health status of wheels, the mileage, car load, etc.). However, if the target is to predict train engine failures, we probably need another set of training data that has engine-related features. Before building predictive models, we expect the business expert to understand the data relevancy requirement and provide the domain knowledge that is needed to select relevant subsets of data for the analysis.

There are three essential data sources we look for when qualifying a business problem to be suitable for a predictive maintenance solution:

1. Failure History: Typically, in predictive maintenance applications, failure events are very rare. However, when building predictive models that predict failures, the algorithm needs to learn the normal operation pattern as well as the failure pattern through the training process. Hence, it is essential that the training data contains sufficient number of examples in both categories in order to learn these two different patterns. For that reason, we require that data has sufficient number of failure events. Failure events can be found in maintenance records and parts replacement history or anomalies in the training data can also be used as failures as identified by the domain experts.
2. Maintenance/Repair History: An essential source of data for predictive maintenance solutions is the detailed maintenance history of the asset containing information about the components replaced, preventive maintenance activates performed, etc. It is extremely important to capture these events as these affect the degradation patterns and absence of this information causes misleading results.
3. Machine Conditions: In order to predict how many more days (hours, miles, transactions, etc.) a machine lasts before it fails, we assume the machine's health status degrades over time during its operation. Therefore, we expect the data to contain time-varying features that capture this aging pattern and any anomalies that leads to degradation. In IoT applications, the telemetry data from different sensors represent one good example. In order to predict if a machine is going to fail within a time frame, ideally the data should capture degrading trend during this time frame before the actual failure event.

Additionally, we require data that is directly related to the operating conditions of the target asset of prediction. The decision of target is based on both business needs and data availability. Taking the train wheel failure prediction as an example, we may predict "if the wheel is going to have a failure" or "if the whole train is going have a failure". The first one targets a more specific component whereas the second one targets failure of the train. The second one is a more general question that requires a lot more dispersed data elements than the first one, making it harder to build a model. Conversely, trying to predict wheel failures just by looking at the high-level train condition data may not be feasible as it does not contain information at the component level. In general, it is more sensible to predict specific failure events than more general ones.

One common question that is usually asked about failure history data is "How many failure events are required to train a model and how many is considered as "enough"? There is no clear answer to that question as in many predictive analytics scenarios, it is usually the quality of the data that dictates what is acceptable. If the dataset does not include features that are relevant to failure prediction, then even if there are many failure events, building a good model may not be possible. However, the rule of thumb is that the more the failure events the better the model is and a rough estimate of how many failure examples are required is a very context and data-dependent measure. This issue is discussed in the section for handling imbalanced datasets where we propose methods to cope with the problem of not having enough failures.

Sample use cases

This section focuses on a collection of predictive maintenance use cases from several industries such as Aerospace, Utilities and Transportation. Each subsection drills into the use-cases collected from these areas and discusses a business problem, the data surrounding the business problem and the benefits of a predictive maintenance solution.

Aerospace

Use Case 1: Flight delay and cancellations

Business problem and data sources

One of the major business problems that airlines face is the significant costs that are associated with flights being delayed due to mechanical problems. If the mechanical failures cannot be repaired, flights may even be canceled. This is extremely costly as delays create problems in scheduling and operations, causes bad reputation and customer dissatisfaction along with many other problems. Airlines are particularly interested in predicting such mechanical failures in advance so that they can reduce flight delays or cancellations. The goal of the predictive maintenance solution for these cases is to predict the probability of an aircraft being delayed or canceled, based on relevant data sources such as maintenance history and flight route information. The two major data sources for this use case are the flight legs and page logs. Flight leg data includes data about the flight route details such as the date and time of departure and arrival, departure and arrival airports, etc. Page log data includes a series of error and maintenance codes that are recorded by the maintenance personnel.

Business value of the predictive model

Using the available historical data, a predictive model was built using a multi-classification algorithm to predict the type of mechanical issue which results in a delay or cancellation of a flight within the next 24 hours. By making this prediction, necessary maintenance actions can be taken to mitigate the risk while an aircraft is being serviced and thus prevent possible delays or cancellations. Using Azure Machine Learning web service, the predictive models can seamlessly and easily be integrated into airlines' existing operating platforms.

Use Case 2: Aircraft component failure

Business problem and data sources

Aircraft engines are very sensitive and expensive pieces of equipment and engine part replacements are among the most common maintenance tasks in the airline industry. Maintenance solutions for airlines require careful management of component stock availability, delivery and planning. Being able to gather intelligence on component reliability leads to substantial reduction on investment costs. The major data source for this use case is telemetry data collected from a number of sensors in the aircraft providing information on the condition of the aircraft. Maintenance records were also used to identify when component failures occurred and replacements were made.

Business value of the predictive model

A multi-class classification model was built that predicts the probability of a failure due to a certain component within the next month. By employing these solutions, airlines can reduce component repair costs, improve component stock availability, reduce inventory levels of related assets and improve maintenance planning.

Utilities

Use Case 1: ATM cash dispense failure

Business problem and data sources

Executives in asset intensive industries often state that primary operational risk to their businesses is unexpected failures of their assets. As an example, failure of machinery such as ATMs in banking industry is a very common problem that occurs frequently. These types of problems make predictive maintenance solutions very desirable for operators of such machinery. In this use-case, prediction problem is to calculate the probability that an ATM cash withdrawal transaction gets interrupted due to a failure in the cash dispenser such as a paper jam or a part failure. Major data sources for this case are sensor readings that collect measurements while cash notes are being dispensed and also maintenance records collected over time. Sensor data included sensor readings per each transaction completed and also sensor readings per each note dispensed. The sensor readings provided measurements such as gaps between notes, thickness, note arrival distance etc. Maintenance data included error codes and repair information. These were used to identify failure cases.

Business value of the predictive model

Two predictive models were built to predict failures in the cash withdrawal transactions and failures in the individual notes dispensed during a transaction. By being able to predict transaction failures beforehand, ATMs can be serviced proactively to prevent failures from occurring. Also, with note failure prediction, if a transaction is likely to fail before it is complete due to a note dispense failure, it may be best to stop the process and warn the customer for incomplete transaction rather than waiting for the maintenance service to arrive after the error occurs which

may lead to larger customer dissatisfaction.

Use Case 2: Wind turbine failures

Business problem and data sources

With the raise of environmental awareness, wind turbines have become one of the major sources of energy generation and they usually cost millions of dollars. One of the key components of wind turbines is the generator motor which is equipped with many sensors that helps to monitor turbine conditions and status. The sensor readings contain valuable information which can be used to build a predictive model to predict critical Key Performance Indicators (KPIs) such as mean time to failure for components of the wind turbine. Data for this use case comes from multiple wind turbines that are located in three different farm locations. Measurements from close to a hundred sensors from each turbine were recorded every 10 seconds for one year. These readings include measurements such as temperature, generator speed, turbine power and generator winding.

Business value of the predictive model

Predictive models were built to estimate remaining useful life for generators and temperature sensors. By predicting the probability of failure, maintenance technicians can focus on suspicious turbines that are likely to fail soon to complement time-based maintenance regimes. Additionally, predictive models bring insight to the level of contribution for different factors to the probability of a failure which helps business to have a better understanding of the root cause of the problems.

Use Case 3: Circuit breaker failures

Business problem and data sources

Electricity and gas operations that include generation, distribution and sale of electrical energy require significant amount of maintenance to ensure power lines are operational at all times to guarantee delivery of energy to households. Failure of such operations is critical as almost every entity is effected by power problems in the regions that they occur. Circuit breakers are critical for such operations as they are a piece of equipment that cut electrical current in case of problems and short circuits to prevent any damage to power lines from happening. The business problem for this use case is to predict circuit breaker failures given maintenance logs, command history and technical specifications.

Three major data sources for this case are maintenance logs that include corrective, preventive and systematic actions, operational data that includes automatic and manual commands send to circuit breakers such as for open and close actions and technical specification data about the properties of each circuit breaker such as year made, location, model, etc.

Business value of the predictive model

Predictive maintenance solutions help reduce repair costs and increase the lifecycle of equipment such as circuit breakers. These models also help improve the quality of the power network since models provide warnings ahead of time that reduce unexpected failures which lead to fewer interruptions to the service.

Use Case 4: Elevator door failures

Business problem and data sources

Most large elevator companies typically have millions of elevators running around the world. To gain a competitive edge, they focus on reliability which is what matters most to their customers. Drawing on the potential of the Internet of Things, by connecting their elevators to the cloud and gathering data from elevator sensors and systems, they are able to transform data into valuable business intelligence which vastly improves operations by offering predictive and preemptive maintenance that is not something that is available to the competitors yet. The business requirement for this case is to provide a knowledge base predictive application that predicts the potential causes of door failures. The required data for this implementation consists of three parts which are elevator static features (e.g. identifiers, contract maintenance frequency, building type, etc.), usage information (e.g. number of door cycles, average door close time, etc.) and failure history (i.e. historical failure records and their causes).

A multiclass logistic regression model was built with Azure Machine Learning to solve the prediction problem, with the integrated static features and usage data as features, and the causes of historical failure records as class labels. This predictive model is consumed by an app on a mobile device which is used by field technicians to help improve working efficiency. When a technician goes on site to repair an elevator, he/she can refer to this app for recommended causes and best courses of maintenance actions to fix the elevator doors as fast as possible.

Transportation and logistics

Use Case 1: Brake disc failures

Business problem and data sources

Typical maintenance policies for vehicles include corrective and preventive maintenance. Corrective maintenance implies that the vehicle is repaired after a failure has occurred which can cause a severe inconvenience to the driver as a result of an unexpected malfunction and the time wasted on a visit to mechanic. Most vehicles are also subject to a preventive maintenance policy, which requires performing certain inspections at a schedule which does not take into account the actual condition of the car subsystems. None of these approaches are successful in fully eliminating problems. The specific use case here is brake disc failure prediction based on data collected through sensors installed in the tire system of a car which keeps track of historical driving patterns and other conditions that the car is exposed to. The most important data source for this case is the sensor data that measure, for instance, accelerations, braking patterns, driving distances, velocity, etc. This information, coupled with other static information such as car features, help build a good set of predictors that can be used in a predictive model. Another set of essential information is the failure data which is inferred from the part order database (used to keep the spare part order dates and quantities as cars are being serviced in the dealerships).

Business value of the predictive model

The business value of a predictive approach here is substantial. A predictive maintenance system can schedule a visit to the dealer based on a predictive model. The model can be based on sensory information that is representing the current condition of the car and the driving history. This approach can minimize the risk of unexpected failures, which may as well occur before the next periodic maintenance. It can also reduce the amount of unnecessary preventive maintenance. Driver can proactively be informed that a change of parts might be necessary in a few weeks and supply the dealer with that information. The dealer could then prepare an individual maintenance package for the driver in advance.

Use Case 2: Subway train door failures

Business problem and data sources

One of the major reasons of delays and problems on subway operations is door failures of train cars. Predicting if a train car may have a door failure, or being able to forecast the number of days till the next door failure, is extremely important foresight. It provides the opportunity to optimize train door servicing and reduce the train's down time.

Data sources

Three sources of data in this use-case are

- **train event data**, which is the historical records of train events,
- **maintenance data** such as maintenance types, work order types, and priority codes,
- **records of failures**.

Business value of the predictive model

Two models were built to predict next day failure probability using binary classification and days till failure using regression. Similar to the earlier cases, the models create tremendous opportunity to improve quality of service and increase customer satisfaction by complementing the regular maintenance regimes.

Data preparation

Data sources

The common data elements for predictive maintenance problems can be summarized as follows:

- Failure history: The failure history of a machine or component within the machine.
- Maintenance history: The repair history of a machine, e.g. error codes, previous maintenance activities or component replacements.
- Machine conditions and usage: The operating conditions of a machine e.g. data collected from sensors.
- Machine features: The features of a machine, e.g. engine size, make and model, location.
- Operator features: The features of the operator, e.g. gender, past experience.

It is possible and usually the case that failure history is contained in maintenance history such as in the form of

special error codes or order dates for spare parts. In those cases, failures can be extracted from the maintenance data. Additionally, different business domains may have a variety of other data sources that influence failure patterns which are not listed here exhaustively. These should be identified by consulting the domain experts when building predictive models.

Some examples of above data elements from use cases are:

Failure history: flight delay dates, aircraft component failure dates and types, ATM cash withdrawal transaction failures, train/elevator door failures, brake disk replacement order dates, wind turbine failure dates and circuit breaker command failures.

Maintenance history: Flight error logs, ATM transaction error logs, train maintenance records including maintenance type, short description etc. and circuit breaker maintenance records.

Machine conditions and usage: Flight routes and times, sensor data collected from aircraft engines, sensor readings from ATM transactions, train events data, sensor readings from wind turbines, elevators and connected cars.

Machine features: Circuit breaker technical specifications such as voltage levels, geolocation or car features such as make, model, engine size, tire types, production facility etc.

Given the above data sources, the two main data types we observe in predictive maintenance domain are temporal data and static data. Failure history, machine conditions, repair history, usage history almost always come with time-stamps indicating the time of collection for each piece of data. Machine features and operator features in general are static since they usually describe the technical specifications of machines or operator's properties. It is possible for these features to change over time and if so they should be treated as time stamped data sources.

Merging data sources

Before getting into any type of feature engineering or labeling process, we need to first prepare our data in the form required to create features from. The ultimate goal is to generate a record for each time unit for each asset with its features and labels to be fed into the machine learning algorithm. In order to prepare that clean final data set, some pre-processing steps should be taken. First step is to divide the duration of data collection into time units where each record belongs to a time unit for an asset. Data collection can also be divided into other units such as actions, however for simplicity we use time units for the rest of the explanations.

The measurement unit for time can be in seconds, minutes, hours, days, months, cycles, miles or transactions depending on the efficiency of data preparation and the changes observed in the conditions of the asset from a time unit to the other or other factors specific to the domain. In other words, the time unit does not have to be the same as the frequency of data collection as in many cases data may not show any difference from one unit to the other. For example, if temperature values were being collected every 10 seconds, picking a time unit of 10 seconds for the whole analysis inflates the number of examples without providing any additional information. Better strategy would be to use average over an hour as an example.

Example generic data schemas for the possible data sources explained in the earlier section are:

Maintenance records: These are the records of maintenance actions performed. The raw maintenance data usually comes with an Asset ID and time stamp with information about what maintenance activities have been performed at that time. In case of such raw data, maintenance activities need to be translated into categorical columns with each category corresponding to a maintenance action type. The basic data schema for maintenance records would include asset ID, time and maintenance action columns.

Failure records: These are the records that belong to the target of prediction which we call failures or failure reason. These can be specific error codes or events of failures defined by specific business condition. In some cases, data includes multiple error codes some of which correspond to failures of interest. Not all errors are target of prediction so other errors are usually used to construct features that may correlate with failures. The basic data schema for failure records would include asset ID, time and failure or failure reason columns if reason is available.

Machine conditions: These are preferably real-time monitoring data about the operating conditions of the data. For

example, for door failures, door opening and closing times are good indicators about the current condition of doors. The basic data schema for machine conditions would include asset ID, time and condition value columns.

Machine and operator data: Machine and operator data can be merged into one schema to identify which asset was operated by which operator along with asset and operator properties. For example, a car is usually owned by a driver with attributes such as age, driving experience etc. If this data changes over time, this data should also include a time column and should be treated as time varying data for feature generation. The basic data schema for machine conditions would include asset ID, asset features, operator ID and operator features.

The final table before labeling and feature generation can be generated by left joining machine conditions table with failure records on Asset ID and time fields. This table can then be joined with maintenance records on Asset ID and Time fields and finally with machine and operator features on Asset ID. The first left join leaves null values for failure column when machine is in normal operation, these can be imputed by an indicator value for normal operation. This failure column is used to create labels for the predictive model.

Feature engineering

The first step in modeling is feature engineering. The idea of feature generation is to conceptually describe and abstract a machine's health condition at a given time using historical data that was collected up to that point in time. In the next section, we provide an overview of the type of techniques that can be used for predictive maintenance and how the labeling is done for each technique. The exact technique that should be used depends on the data and business problem. However, the feature engineering methods described below can be used as baseline for creating features. Below, we discuss lag features that should be constructed from data sources that come with time-stamps and also static features created from static data sources and provide examples from the use cases.

Lag features

As mentioned earlier, in predictive maintenance, historical data usually comes with timestamps indicating the time of collection for each piece of data. There are many ways of creating features from the data that comes with timestamped data. In this section, we discuss some of these methods used for predictive maintenance. However, we are not limited by these methods alone. Since feature engineering is considered to be one of the most creative areas of predictive modeling, there could be many other ways to create features. Here, we provide some general techniques.

Rolling aggregates

For each record of an asset, we pick a rolling window of size "W" which is the number of units of time that we would like to compute historical aggregates for. We then compute rolling aggregate features using the W periods before the date of that record. Some example rolling aggregates can be rolling counts, means, standard deviations, outliers based on standard deviations, CUSUM measures, minimum and maximum values for the window. Another interesting technique is to capture trend changes, spikes and level changes using algorithms that detect anomalies in data using anomaly detection algorithms.

For demonstration, see Figure 1 where we represent sensor values recorded for an asset for each unit of time with the blue lines and mark the rolling average feature calculation for W=3 for the records at t_1 and t_2 which are indicated by orange and green groupings respectively.

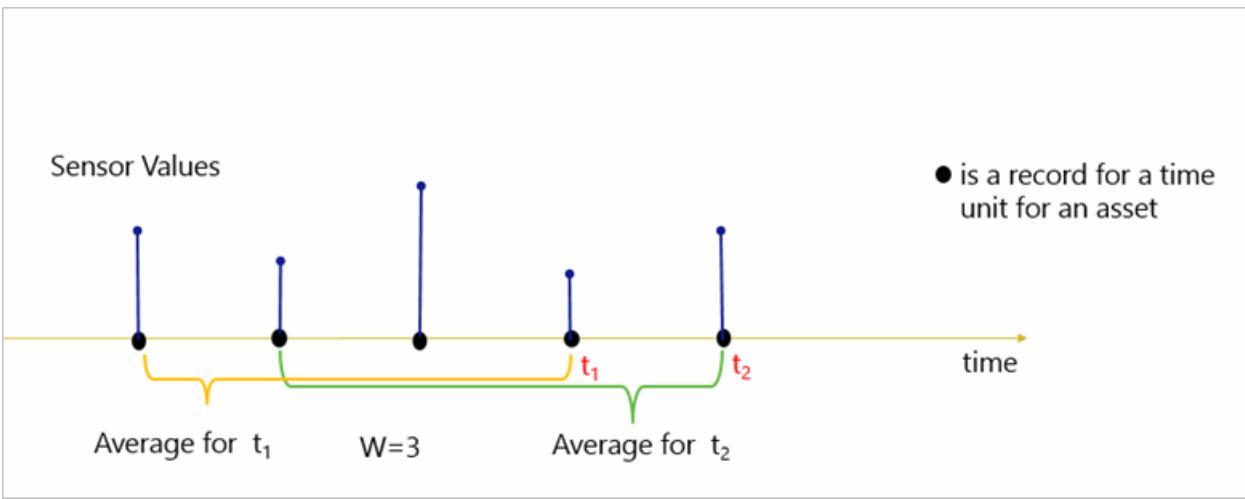


Figure 1. Rolling aggregate features

As examples, for aircraft component failure, sensor values from last week, last three days and last day were used to create rolling means, standard deviation and sum features. Similarly, for ATM failures, both raw sensor values and rolling means, median, range, standard deviations, number of outliers beyond three standard deviations, upper and lower CUMSUM features were used.

For flight delay prediction, counts of error codes from last week were used to create features. For train door failures, counts of the events on the last day, counts of events over the previous 2 weeks and variance of counts of events of the previous 15 days were used to create lag features. Same counting was used for maintenance-related events.

Additionally, by picking a W that is very large (ex. years), it is possible to look at the whole history of an asset such as counting all maintenance records, failures etc. up until the time of the record. This method was used for counting circuit breaker failures for the last three years. Also for train failures, all maintenance events were counted to create a feature to capture the long-term maintenance effects.

Tumbling aggregates

For each labeled record of an asset, we pick a window of size " $W-k$ " where k is the number of windows of size " W " that we want to create lag features for. " k " can be picked as a large number to capture long-term degradation patterns or a small number to capture short-term effects. We then use k tumbling windows $W-k, W-(k-1), \dots, W-2, W-1$ to create aggregate features for the periods before the record date and time (see Figure 2). These are also rolling windows at the record level for a time unit which is not captured in Figure 2 but the idea is the same as in Figure 1 where t_2 is also used to demonstrate the rolling effect.

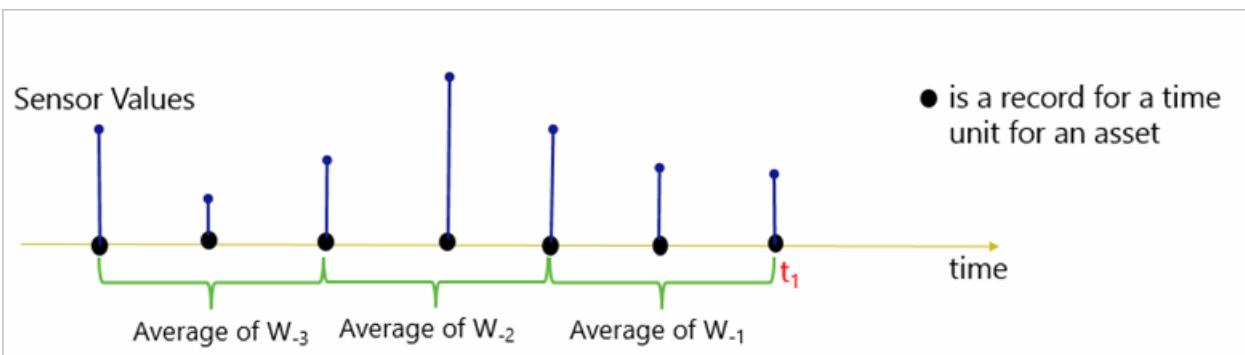


Figure 2. Tumbling aggregate features

As an example, for wind turbines, $W=1$ and $k=3$ months were used to create lag features for each of the last 3 months using top and bottom outliers.

Static features

These are technical specifications of the equipment such as manufacture date, model number, location, etc. While lag features are mostly numeric in nature, static features usually become categorical variables in the models. As an example, circuit breaker properties such as voltage, current and power specifications along with transformer types,

power sources etc. were used. For brake disc failures, the type of tire wheels such as if they are alloy or steel were used as some of the static features.

During feature generation, some other important steps such as handling missing values and normalization should be performed. There are numerous methods of missing value imputation and also data normalization which is not discussed here. However, it is beneficial to try different methods to see if an increase in prediction performance is possible.

The final feature table after feature engineering steps discussed in the earlier section should resemble the following example data schema when time unit is a day:

ASSET ID	TIME	FEATURE COLUMNS	LABEL
1	Day 1		
1	Day 2		
...	...		
2	Day 1		
2	Day 2		
...	...		

Modeling techniques

Predictive Maintenance is a very rich domain often employing business questions which may be approached from many different angles of the predictive modeling perspective. In the next sections, we provide main techniques that are used to model different business questions that can be answered with predictive maintenance solutions.

Although there are similarities, each model has its own way of constructing labels which are described in detail. As an accompanying resource, you can refer to the predictive maintenance template that is included in the sample experiments provided within Azure Machine Learning. The links to the online material for this template are provided in the resources section. You can see how some of the feature engineering techniques discussed above and the modeling technique that is described in the next sections are applied to predict aircraft engine failures using Azure Machine Learning.

Binary classification for predictive maintenance

Binary Classification for predictive maintenance is used to predict the probability that equipment fails within a future time period. The time period is determined by and based on business rules and the data at hand. Some common time periods are minimum lead time required to purchase spare parts to replace likely to damage components or time required to deploy maintenance resources to perform maintenance routines to fix the problem that is likely to occur within that time period. We call this future horizon period "X".

In order to use binary classification, we need to identify two types of examples which we call positive and negative. Each example is a record that belongs to a time unit for an asset conceptually describing and abstracting its operating conditions up to that time unit through feature engineering using historical and other data sources described earlier. In the context of binary classification for predictive maintenance, positive type denotes failures (label 1) and negative type denotes normal operations (label = 0) where labels are of type categorical. The goal is to find a model that identifies each new example as likely to fail or operate normally within the next X units of time.

Label construction

In order to create a predictive model to answer the question "What is the probability that the asset fails in the next X units of time?", labeling is done by taking X records prior to the failure of an asset and labeling them as "about to

"fail" (label = 1) while labeling all other records as "normal" (label = 0). In this method, labels are categorical variables (see Figure 3).

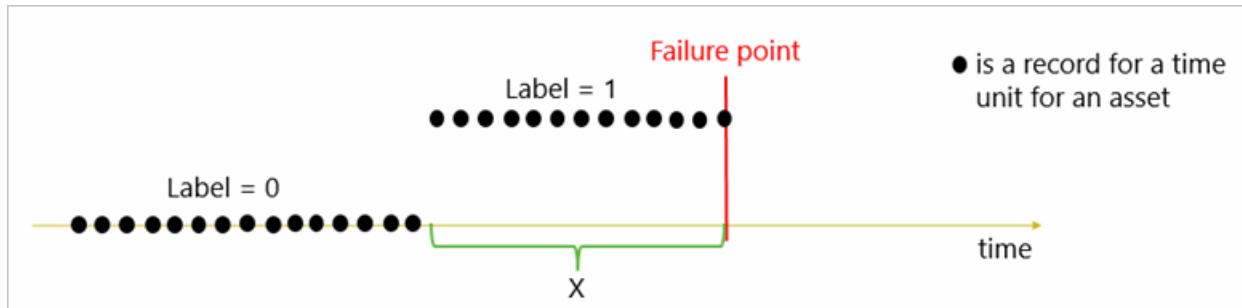


Figure 3. Labeling for binary classification

For flight delays and cancellations, X is picked as one day to predict delays in the next 24 hours. All flights that are within 24 hours before failures were labeled as 1s. For ATM cash dispense failures, two binary classification models were built to predict the failure probability of a transaction in the next 10 minutes and also to predict the probability of failure in the next 100 notes dispensed. All transactions that happened within the last 10 minutes of the failure are labeled as 1 for the first model. And all notes dispensed within the last 100 notes of a failure were labeled as 1 for the second model. For circuit breaker failures, the task is to predict the probability that the next circuit breaker command fails in which case X is chosen to be one future command. For train door failures, the binary classification model was built to predict failures within the next 7 days. For wind turbine failures, X was chosen as 3 months.

Wind turbine and train door cases are also used for regression analysis to predict remaining useful life using the same data but by utilizing a different labeling strategy which is explained in the next section.

Regression for predictive maintenance

Regression models in predictive maintenance are used to compute the remaining useful life (RUL) of an asset which is defined as the amount of time that the asset is operational before the next failure occurs. Same as binary classification, each example is a record that belongs to a time unit "Y" for an asset. However, in the context of regression, the goal is to find a model that calculates the remaining useful life of each new example as a continuous number which is the period of time remaining before the failure. We call this time period some multiple of Y. Each example also has a remaining useful life which can be calculated by measuring the amount of time remaining for that example before the next failure.

Label construction

Given the question "What is the remaining useful life of the equipment? ", labels for the regression model can be constructed by taking each record prior to the failure and labeling them by calculating how many units of time remain before the next failure. In this method, labels are continuous variables (See Figure 4).

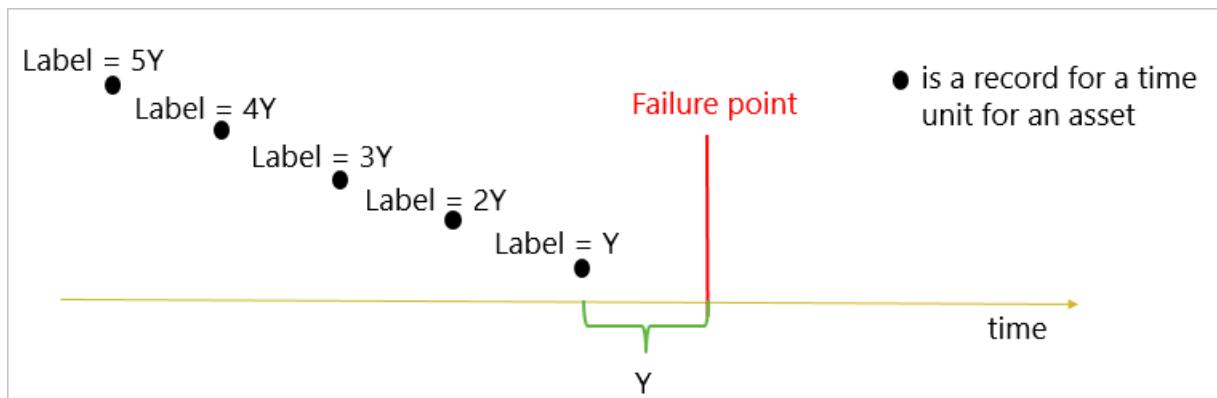


Figure 4. Labeling for regression

Different than binary classification, for regression, assets without any failures in the data cannot be used for modeling as labeling is done in reference to a failure point and its calculation is not possible without knowing how long the asset survived before failure. This issue is best addressed by another statistical technique called Survival

Analysis. We are not going to discuss Survival Analysis in this playbook because of the potential complications that may arise when applying the technique to predictive maintenance use cases that involve time-varying data with frequent intervals.

Multi-class classification for predictive maintenance

Multi-class classification for predictive maintenance can be used to predict two future outcomes. The first one is to assign an asset to one of the multiple possible periods of time to give a range of time to failure for each asset. The second one is to identify the likelihood of failure in a future period due to one of the multiple root causes. That allows maintenance personnel who are equipped with this knowledge to handle the problems in advance. Another multi-class modeling technique focuses on determining the most likely root cause of a given a failure. This allows recommendations to be given for the top maintenance actions to be taken in order to fix a failure. By having a ranked list of root causes and associated repair actions, technicians can be more effective in taking their first repair actions after failures.

Label construction

Given the two questions which are "What is the probability that an asset fails in the next " aZ " units of time where " a " is the number of periods" and "What is the probability that the asset fails in the next X units of time due to problem " P_i " where " i " is the number of possible root causes, labeling is done in the following way for these to techniques.

For the first question, labeling is done by taking aZ records prior to the failure of an asset and labeling them using buckets of time ($3Z$, $2Z$, Z) as their labels while labeling all other records as "normal" (label =0). In this method, label is categorical variable (See Figure 5).

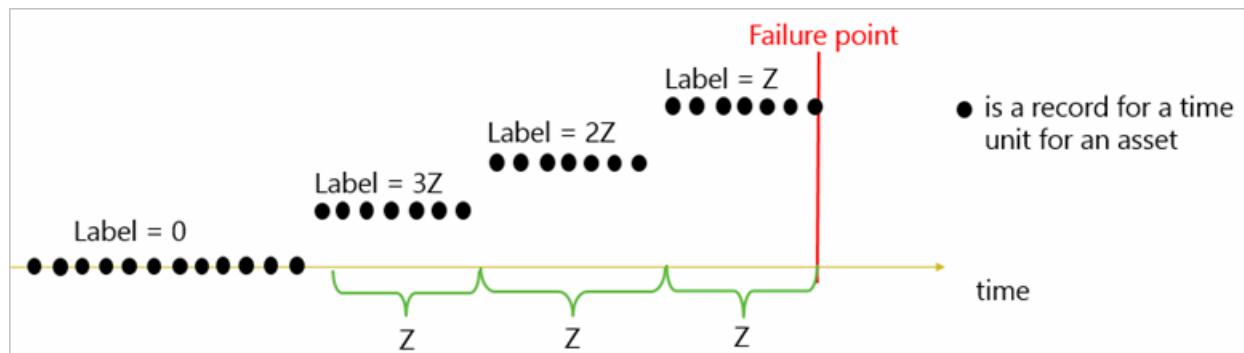


Figure 5. Labeling for multiclass classification for failure time prediction

For the second question, labeling is done by taking X records prior to the failure of an asset and labeling them as "about to fail due to problem P_i " (label = P_i) while labeling all other records as "normal" (label =0). In this method, labels are categorical variables (See Figure 6).

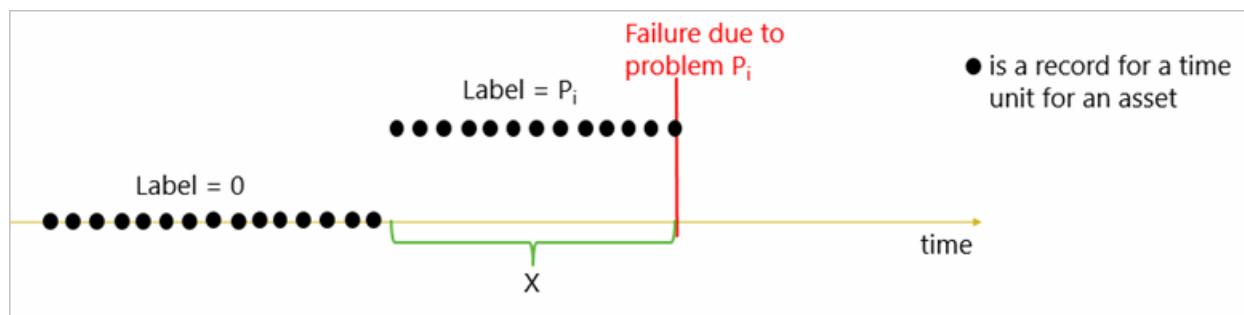


Figure 6. Labeling for multiclass classification for root cause prediction

The model assigns a failure probability due to each P_i as well as the probability of no failure. These probabilities can be ordered by magnitude to allow prediction of the problems that are most likely to occur in the future. Aircraft component failure use case was structured as a multiclass classification problem. This enables the prediction of the probabilities of failure due to two different pressure valve components occurring within the next month.

For recommending maintenance actions after failures, labeling does not require a future horizon to be picked. This

is because the model is not predicting failure in the future but it is just predicting the most likely root cause once the failure has already happened. Elevator door failures fall into the third case where the goal is to predict the cause of the failure given historical data on operating conditions. This model is then used to predict the most likely root causes after a failure has occurred. One key benefit of this model is that it helps inexperienced technicians to easily diagnose and fix problems that would otherwise need years' worth of experience.

Training, validation and testing methods in predictive maintenance

In predictive maintenance, similar to any other solution space containing timestamped data, the typical training and testing routine needs to take account the time varying aspects to better generalize on unseen future data.

Cross validation

Many machine learning algorithms depend on a number of hyperparameters that can change model performance significantly. The optimal values of these hyperparameters are not computed automatically when training the model, but should be specified by data scientist. There are several ways of finding good values of hyperparameters. The most common one is "k-fold cross-validation" which splits the examples randomly into "k" folds. For each set of hyperparameters values, learning algorithm is run k times. At each iteration, the examples in the current fold are used as a validation set, the rest of the examples are used as a training set. The algorithm trains over training examples and the performance metrics are computed over validation examples. At the end of this loop for each set of hyperparameter values, we compute average of the k performance metric values and choose hyperparameter values that have the best average performance.

As mentioned before, in predictive maintenance problems, data is recorded as a time series of events that come from several data sources. These records can be ordered according to the time of labeling a record or an example. Hence, if we split the dataset randomly into training and validation set, some of the training examples are later in time than some of validation examples. This results in estimating future performance of hyperparameter values based on the data that arrived before model was trained. These estimations might be overly optimistic, especially if time-series are not stationary and change their behavior over time. As a result, chosen hyperparameter values might be sub-optimal.

A better way of finding good values of hyperparameters is to split the examples into training and validation set in a time-dependent way, such that all validation examples are later in time than all training examples. Then, for each set of values of hyperparameters we train the algorithm over training set, measure model's performance over the same validation set and choose hyperparameter values that show the best performance. When time-series data is not stationary and evolves over time, the hyperparameter values chosen by train/validation split lead to a better future "model's performance than with the values chosen randomly by cross-validation.

The final model is generated by training a learning algorithm over entire data using the best hyperparameter values that are found by using training/validation split or cross-validation.

Testing for model performance

After building a model we need to estimate its future performance on new data. The simplest estimate could be the performance of the model over the training data. But this estimate is overly optimistic, because the model is tailored to the data that is used to estimate performance. A better estimate could be a performance metric of hyperparameter values computed over the validation set or an average performance metric computed from cross-validation. But for the same reasons as previously stated, these estimations are still overly optimistic. We need more realistic approaches for measuring model performance.

One way is to split the data randomly into training, validation and test sets. The training and validation sets are used to select values of hyperparameters and train the model with them. The performance of the model is measured over the test set.

Another way which is relevant to predictive maintenance, is to split the examples into training, validation and test sets in a time-dependent way, such that all test examples are later in time than all training and validation examples. After the split, model generation and performance measurement are done the same as described earlier.

When time-series are stationary and easy to predict both approaches generate similar estimations of future performance. But when time-series are non-stationary and/or hard to predict, the second approach will generate more realistic estimates of future performance than the first one.

Time-dependent split

As a best practice, in this section we take a closer look at how to implement time-dependent split. We describe a time-dependent two-way split between training and test sets, however exactly the same logic should be applied for time-dependent split for training and validation sets.

Suppose we have a stream of timestamped events such as measurements from various sensors. Features of training and test examples, as well as their labels, are defined over timeframes that contain multiple events. For example, for binary classification, as described in Feature Engineering and Modeling Techniques sections, features are created based on the past events and labels are created based on future events within "X" units of time in the future. Thus, the labeling timeframe of an example comes later than the timeframe of its features. For time-dependent split, we pick a point in time at which we train a model with tuned hyperparameters by using historical data up to that point. To prevent leakage of future labels that are beyond the training cut-off into training data, we choose the latest timeframe to label training examples to be X units before the training cut-off date. In Figure 7, each solid circle represents a row in the final feature data set for which the features and labels are computed according to the method described above. Given that, the figure shows the records that should go into training and testing sets when implementing time-dependent split for X=2 and W=3:



Figure 7. Time-dependent split for binary classification

The green squares represent the records belonging to the time units that can be used for training. As explained earlier, each training example in the final feature table is generated by looking at past 3 periods for feature generation and 2 future periods for labeling before the training day cut-off. We do not use examples in the training set when any part of the 2 future periods for that example is beyond the training cut-off since we assume that we do not have visibility beyond the training cut-off. Due to that constraint, black examples represent the records of the final labeled dataset that should not be used in the training data set. These records won't be used in testing data either since they are before the training cut-off and their labeling timeframes partially depend on the training timeframe which should not be the case as we would like to completely separate labeling timeframes for training and testing to prevent label information leakage.

This technique allows for overlap in the data used for feature generation between training and testing examples that are close to the training cut-off. Depending on data availability, an even more severe separation can be accomplished by not using any of the examples in the test set that are within W time units of the training cut-off.

From our work, we found that regression models used for predicting remaining useful life are more severely affected by the leakage problem and using a random split leads to extreme overfitting. Similarly, in regression problems, the split should be such that records belonging to assets with failures before training cut off should be used for the training set and assets that have failures after the cut-off should be used for testing set.

As a general method, another important best practice for splitting data for training and testing is to use a split by asset ID so that none of the assets that were used in training are used for testing since the idea of testing is to make sure that when a new asset is used to make predictions on, the model provides realistic results.

Handling imbalanced data

In classification problems, if there are more examples of one class than of the others, the data is said to be imbalanced. Ideally, we would like to have enough representatives of each class in the training data to be able to differentiate between different classes. If one class is less than 10% of the data, we can say that the data is imbalanced and we call the underrepresented dataset minority class. Drastically, in many cases we find imbalanced datasets where one class is severely underrepresented compared to others for example by only constituting 0.001% of the data points. Class imbalance is a problem in many domains including fraud detection, network intrusion and predictive maintenance where failures are usually rare occurrences in the lifetime of the assets which make up the minority class examples.

In case of class imbalance, performance of most standard learning algorithms is compromised as they aim to minimize the overall error rate. For example, for a data set with 99% negative class examples and 1% positive class examples, we can get 99% accuracy by simply labeling all instances as negative. However, this misclassifies all positive examples so the algorithm is not a useful one although the accuracy metric is very high. Consequently, conventional evaluation metrics such as overall accuracy or error rate, are not sufficient in case of imbalanced learning. Other metrics, such as precision, recall, F1 scores and cost adjusted ROC curves are used for evaluations in case of imbalanced datasets which is discussed in the Evaluation Metrics section.

However, there are some methods that help remedy class imbalance problem. The two major ones are sampling techniques and cost sensitive learning.

Sampling methods

The use of sampling methods in imbalanced learning consists of modification of the dataset by some mechanisms in order to provide a balanced dataset. Although there are a lot of different sampling techniques, most straight forward ones are random oversampling and under sampling.

Simply stated, random oversampling is selecting a random sample from minority class, replicating these examples and adding them to training data set. This increases the number of total examples in minority class and eventually balance the number of examples of different classes. One danger of oversampling is that multiple instances of certain examples can cause the classifier to become too specific leading to overfitting. This would result in high training accuracy but performance on the unseen testing data may be very poor. Conversely, random under sampling is selecting a random sample from majority class and removing those examples from training data set. However, removing examples from majority class may cause the classifier to miss important concepts pertaining to the majority class. Hybrid sampling where minority class is oversampled and majority class is under sampled at the same time is another viable approach. There are many other more sophisticated sampling techniques available and effective sampling methods for class imbalance is a popular research area receiving constant attention and contributions from many channels. Use of different techniques to decide on the most effective ones is usually left to the data scientist to research and experiment and are highly dependent on the data properties. Additionally, it is important to make sure that sampling methods are applied only to the training set but not the test set.

Cost sensitive learning

In predictive maintenance, failures which constitute the minority class are of more interest than normal examples and thus the focus is on the performance of the algorithm on failures is usually the focus. This is commonly referred as unequal loss or asymmetric costs of misclassifying elements of different classes wherein incorrectly predicting a positive as negative can cost more than vice versa. The desired classifier should be able to give high prediction accuracy over the minority class, without severely compromising on the accuracy for the majority class.

There are several ways this can be achieved. By assigning a high cost to misclassification of the minority class, and trying to minimize the overall cost, the problem of unequal losses can be effectively dealt with. Some machine learning algorithms use this idea inherently such as SVMs (Support Vector Machines) where cost of positive and negative examples can be incorporated during training time. Similarly, boosting methods are used and usually show good performance in case of imbalanced data such as boosted decision tree algorithms.

Evaluation metrics

As mentioned earlier, class imbalance causes poor performance as algorithms tend to classify majority class examples better in expense of minority class cases as the total misclassification error is much improved when majority class is labeled correctly. This causes low recall rates and becomes a larger problem when the cost of false alarms to the business is very high. Accuracy is the most popular metric used for describing a classifier's performance. However as explained above accuracy is ineffective and do not reflect the real performance of a classifier's functionality as it is very sensitive to data distributions. Instead, other evaluation metrics are used to assess imbalanced learning problems. In those cases, precision, recall and F1 scores should be the initial metrics to look at when evaluating predictive maintenance model performance. In predictive maintenance, recall rates denote how many of the failures in the test set were correctly identified by the model. Higher recall rates mean the model is successful in catching the true failures. Precision metric relates to the rate of false alarms where lower precision rates correspond to higher false alarms. F1 score considers both precision and recall rates with best value being 1 and worst being 0.

Moreover, for binary classification, decile tables and lift charts are very informative when evaluating performance. They focus only on the positive class (failures) and provide a more complex picture of the algorithm performance than what is seen by looking at just a fixed operating point on the ROC (Receiver Operating Characteristic) curve. Decile tables are obtained by ordering the test examples according to their predicted probabilities of failures computed by the model before thresholding to decide on the final label. The ordered samples are then grouped in deciles (i.e. the 10% samples with largest probability and then 20%, 30% and so on). By computing the ratio between true positive rate of each decile and its random baseline (i.e. 0.1, 0.2 ..) one can estimate how the algorithm performance changes at each decile. Lift charts are used to plot decile values by plotting decile true positive rate versus random true positive rate pairs for all deciles. Usually, the first deciles are the focus of the results since here we see the largest gains. First deciles can also be seen as representative for "at risk" when used for predictive maintenance.

Sample solution architecture

When deploying a predictive maintenance solution, we are interested in an end to end solution that provides a continuous cycle of data ingestion, data storage for model training, feature generation, prediction and visualization of the results along with an alert generating mechanism such as an asset monitoring dashboard. We want a data pipeline that provides future insights to the user in a continuous automated manner. An example predictive maintenance architecture for such an IoT data pipeline is illustrated in Figure 8 below. In the architecture, real-time telemetry is collected into an Event Hub which stores streaming data. This data is ingested by stream analytics for real-time processing of data such as feature generation. The features are then used to call the predictive model web service and results are displayed on the dashboard. At the same time, ingested data is also stored in an historical database and merged with external data sources such as on-premise data bases to create training examples for modeling. Same data warehouses can be used for batch scoring of the examples and storing of the results which can again be used to provide predictive reports on the dashboard.

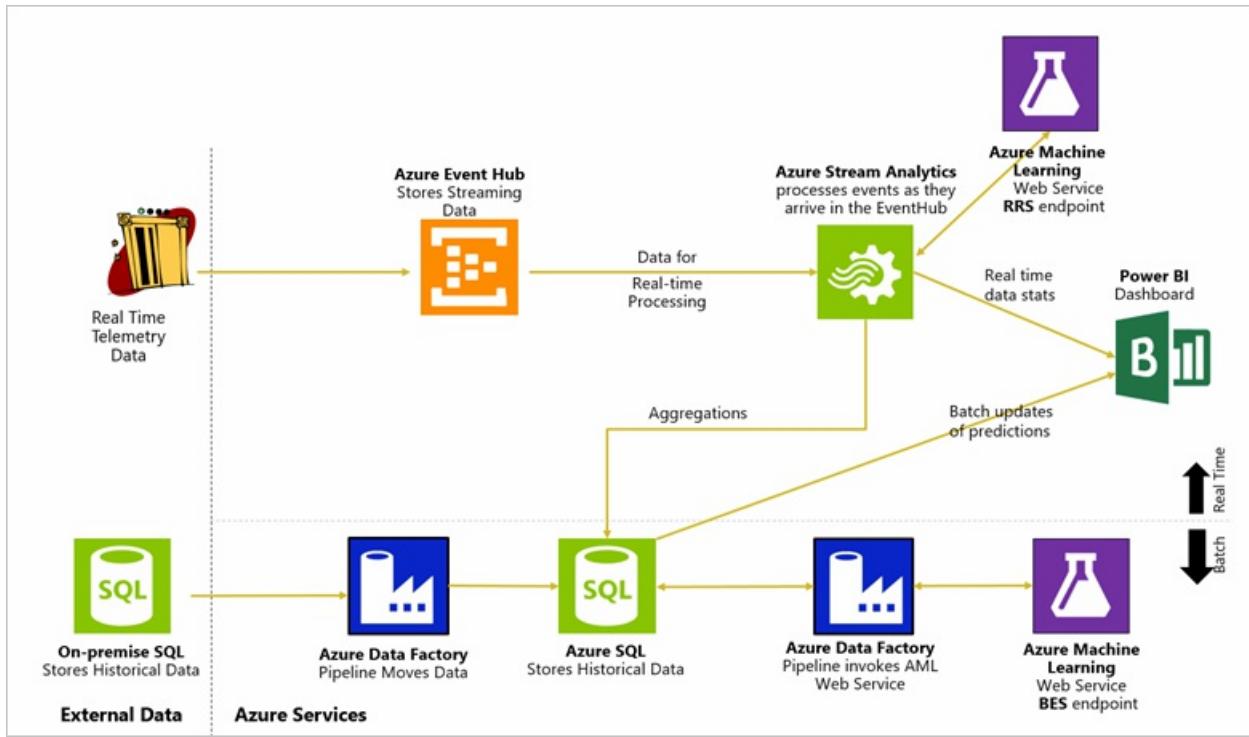


Figure 8. Example solution architecture for predictive maintenance

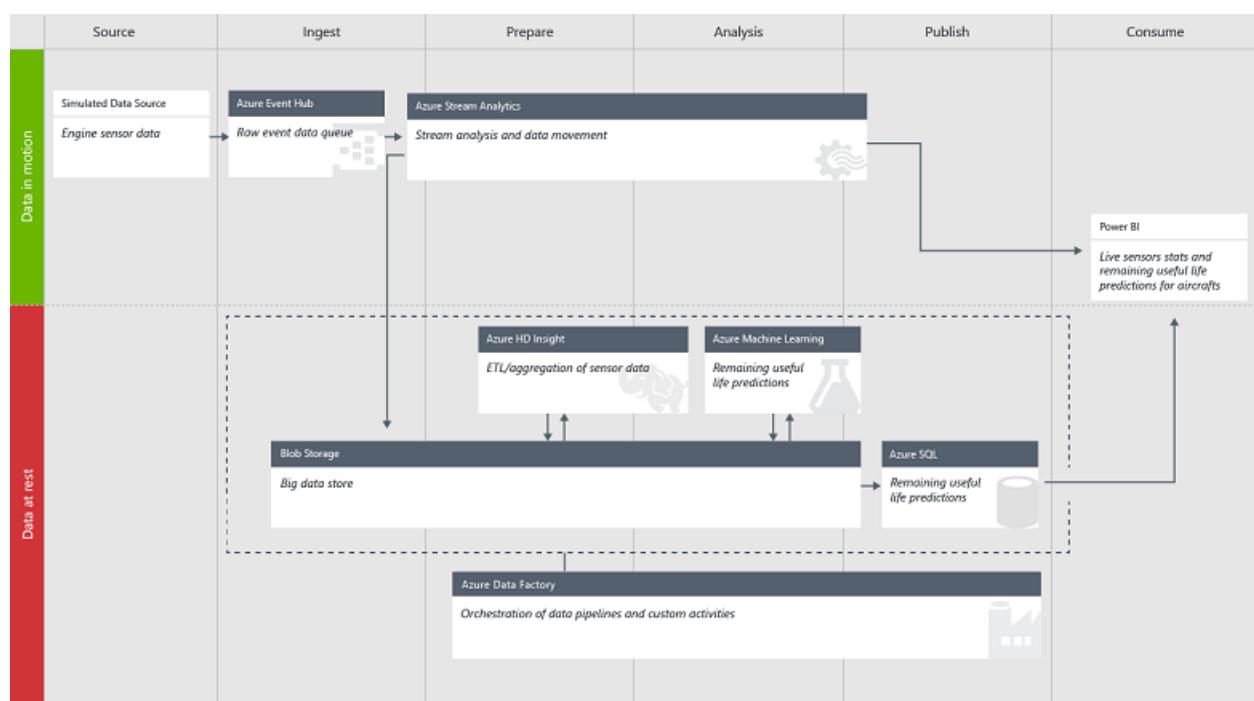
For more information about each of the components of the architecture please refer to [Azure documentation](#).

Architecture of the Cortana Intelligence Solution Template for predictive maintenance in aerospace and other businesses

1/17/2017 • 1 min to read • [Edit on GitHub](#)

The diagram below provides an architectural overview of the [Cortana Intelligence Solution Template for predictive maintenance](#).

You can download a full-size version of the diagram here: [Architecture diagram: Solution Template for predictive maintenance](#).



Technical guide to the Cortana Intelligence Solution Template for predictive maintenance in aerospace and other businesses

1/17/2017 • 17 min to read • [Edit on GitHub](#)

Important

This article has been deprecated. The information is still relevant to the problem at hand, i.e. Predictive Maintenance in Aerospace, but the latest article with the most up to date information can be found [here](#).

Acknowledgements

This article is authored by data scientists Yan Zhang, Gauher Shaheen, Fidan Boylu Uz and software engineer Dan Greco at Microsoft.

Overview

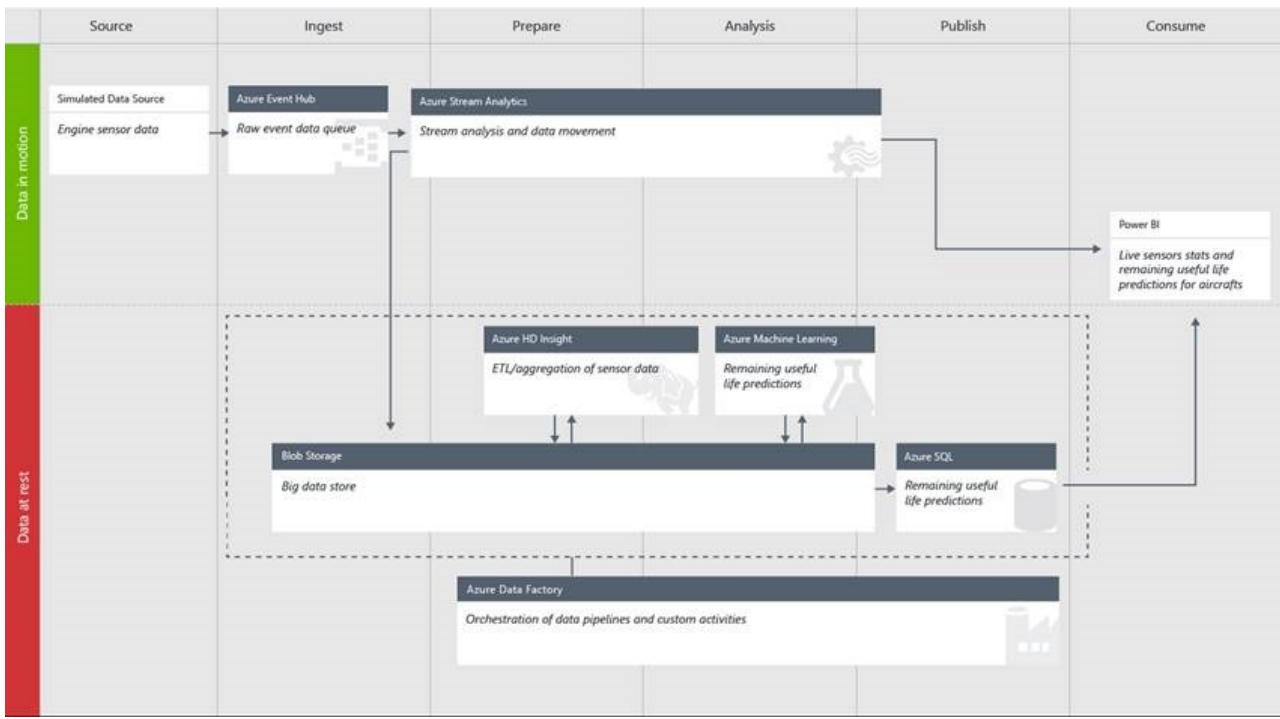
Solution Templates are designed to accelerate the process of building an E2E demo on top of Cortana Intelligence Suite. A deployed template will provision your subscription with necessary Cortana Intelligence components and build the relationships between them. It also seeds the data pipeline with sample data generated from a data generator application which you will download and install on your local machine after you deploy the solution template. The data generated from the generator will hydrate the data pipeline and start generating machine learning predictions which can then be visualized on the Power BI dashboard. The deployment process will guide you through several steps to set up your solution credentials. Make sure you record these credentials such as solution name, username, and password you provide during the deployment.

The goal of this document is to explain the reference architecture and different components provisioned in your subscription as part of this solution template. The document also talks about how to replace the sample data with real data of your own to be able to see insights and predictions from your own data. Additionally, the document discusses the parts of the Solution Template that would need to be modified if you wanted to customize the solution with your own data. Instructions on how to build the Power BI dashboard for this Solution Template are provided at the end.

TIP

You can download and print a [PDF version of this document](#).

Big picture



When the solution is deployed, various Azure services within Cortana Analytics Suite are activated (*i.e.* Event Hub, Stream Analytics, HDInsight, Data Factory, Machine Learning, *etc.*). The architecture diagram above shows, at a high level, how the Predictive Maintenance for Aerospace Solution Template is constructed from end-to-end. You will be able to investigate these services in the azure portal by clicking on them on the solution template diagram created with the deployment of the solution with the exception of HDInsight as this service is provisioned on demand when the related pipeline activities are required to run and deleted afterwards. You can download a [full-size version of the diagram](#).

The following sections describe each piece.

Data source and ingestion

Synthetic data source

For this template the data source used is generated from a desktop application that you will download and run locally after successful deployment. You will find the instructions to download and install this application in the properties bar when you select the first node called Predictive Maintenance Data Generator on the solution template diagram. This application feeds the [Azure Event Hub](#) service with data points, or events, that will be used in the rest of the solution flow. This data source is comprised of or derived from publicly available data from the [NASA data repository](#) using the [Turbofan Engine Degradation Simulation Data Set](#).

The event generation application will populate the Azure Event Hub only while it's executing on your computer.

Azure event hub

The [Azure Event Hub](#) service is the recipient of the input provided by the Synthetic Data Source described above.

Data preparation and analysis

Azure Stream Analytics

The [Azure Stream Analytics](#) service is used to provide near real-time analytics on the input stream from the [Azure Event Hub](#) service and publish results onto a [Power BI](#) dashboard as well as archiving all raw incoming events to the [Azure Storage](#) service for later processing by the [Azure Data Factory](#) service.

HD Insights custom aggregation

The Azure HD Insight service is used to run [Hive](#) scripts (orchestrated by Azure Data Factory) to provide aggregations on the raw events that were archived using the Azure Stream Analytics service.

Azure Machine Learning

The [Azure Machine Learning](#) service is used (orchestrated by Azure Data Factory) to make predictions on the remaining useful life (RUL) of a particular aircraft engine given the inputs received.

Data publishing

Azure SQL Database Service

The [Azure SQL Database](#) service is used to store (managed by Azure Data Factory) the predictions received by the Azure Machine Learning service that will be consumed in the [Power BI](#) dashboard.

Data consumption

Power BI

The [Power BI](#) service is used to show a dashboard that contains aggregations and alerts provided by the [Azure Stream Analytics](#) service as well as RUL predictions stored in [Azure SQL Database](#) that were produced using the [Azure Machine Learning](#) service. For Instructions on how to build the Power BI dashboard for this Solution Template, refer to the section below.

How to bring in your own data

This section describes how to bring your own data to Azure, and what areas would require changes for the data you bring into this architecture.

It's unlikely that any dataset you bring would match the dataset used by the [Turbofan Engine Degradation Simulation Data Set](#) used for this solution template. Understanding your data and the requirements will be crucial in how you modify this template to work with your own data. If this is your first exposure to the Azure Machine Learning service, you can get an introduction to it by using the example in [How to create your first experiment](#).

The following sections will discuss the sections of the template that will require modifications when a new dataset is introduced.

Azure Event Hub

The Azure Event Hub service is very generic, such that data can be posted to the hub in either CSV or JSON format. No special processing occurs in the Azure Event Hub, but it's important that you understand the data that's fed into it.

This document does not describe how to ingest your data, but you can easily send events or data to an Azure Event Hub using the Event Hub API's.

Azure Stream Analytics

The Azure Stream Analytics service is used to provide near real-time analytics by reading from data streams and outputting data to any number of sources.

For the Predictive Maintenance for Aerospace Solution Template, the Azure Stream Analytics query consists of four sub-queries, each consuming events from the Azure Event Hub service, and having outputs to four distinct locations. These outputs consist of three Power BI datasets and one Azure Storage location.

The Azure Stream Analytics query can be found by:

- Logging into the Azure portal
- Locating the Stream Analytics jobs  that were generated when the solution was deployed (e.g., **maintenancesa02asapbi** and **maintenancesa02asablob** for the predictive maintenance solution)
- Selecting
 - **INPUTS** to view the query input

- **QUERY** to view the query itself
- **OUTPUTS** to view the different outputs

Information about Azure Stream Analytics query construction can be found in the [Stream Analytics Query Reference](#) on MSDN.

In this solution, the queries output three datasets with near real-time analytics information about the incoming data stream to a Power BI dashboard that's provided as part of this solution template. Because there's implicit knowledge about the incoming data format, these queries would need to be altered based on your data format.

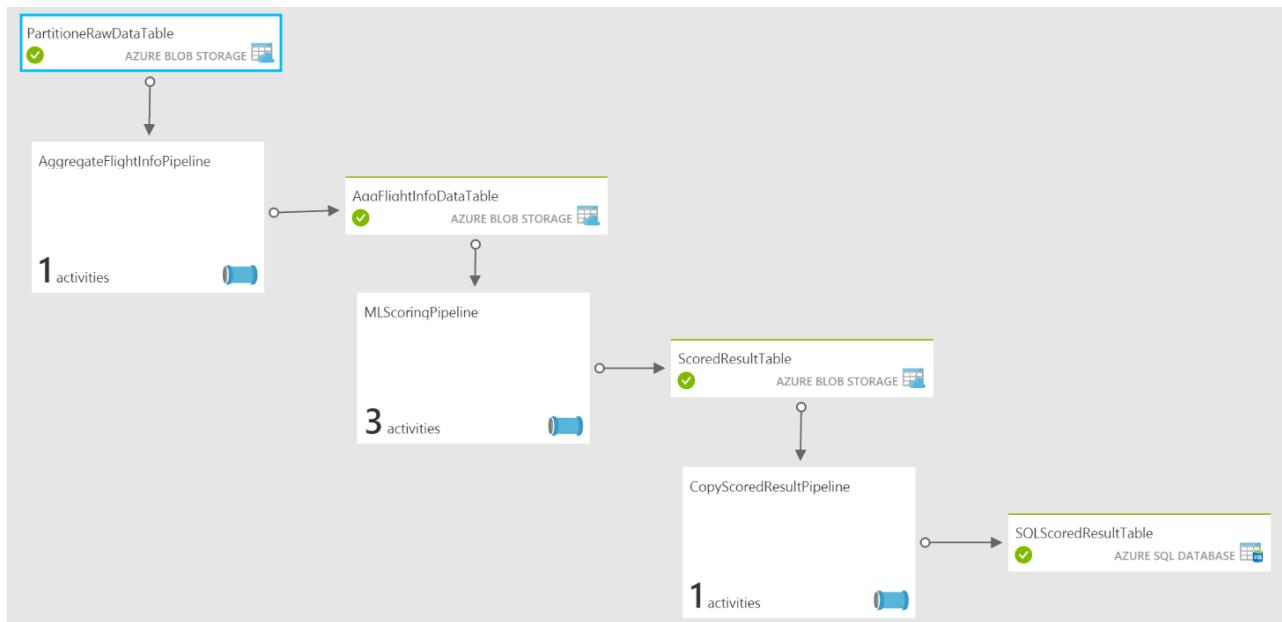
The query in the second Stream Analytics job **maintenancesa02asblob** simply outputs all [Event Hub](#) events to [Azure Storage](#) and hence requires no alteration regardless of your data format as the full event information is streamed to storage.

Azure Data Factory

The [Azure Data Factory](#) service orchestrates the movement and processing of data. In the Predictive Maintenance for Aerospace Solution Template the data factory is made up of three [pipelines](#) that move and process the data using various technologies. You can access your data factory by opening the the Data Factory node at the bottom of the solution template diagram created with the deployment of the solution. This will take you to the data factory on your Azure portal. If you see errors under your datasets, you can ignore those as they are due to data factory being deployed before the data generator was started. Those errors do not prevent your data factory from functioning.

The screenshot shows the Azure Data Factory Essentials blade. It has a summary section with tabs for 'Author and deploy', 'Diagram', and 'Sample pipelines'. Below this is a 'Contents' section with tabs for 'Datasets', 'Pipelines', and 'Linked services'. The 'Datasets' tab shows 6 datasets, 1 of which has errors. The 'Pipelines' tab shows 3 pipelines. The 'Linked services' tab shows 5 linked services. A sidebar on the right lists 'FAILED DATASETS GENERATED WITH EXCE' with one item: 'PartitionsRawDataTable'.

This section discusses the necessary [pipelines](#) and [activities](#) contained in the [Azure Data Factory](#). Below is the diagram view of the solution.



Two of the pipelines of this factory contain [Hive](#) scripts that are used to partition and aggregate the data. When noted, the scripts will be located in the [Azure Storage](#) account created during setup. Their location will be: `maintenancesascript\script\hive\` (or [https://\[Your solution name\].blob.core.windows.net/maintenancesascript](https://[Your solution name].blob.core.windows.net/maintenancesascript)).

Similar to the [Azure Stream Analytics](#) queries, the [Hive](#) scripts have implicit knowledge about the incoming data

format, these queries would need to be altered based on your data format and feature engineering requirements.

AggregateFlightInfoPipeline

This pipeline contains a single activity - an HDInsightHive activity using a HDInsightLinkedService that runs a Hive script to partition the data put in Azure Storage during the Azure Stream Analytics job.

The Hive script for this partitioning task is **AggregateFlightInfo.hql**

MLScoringPipeline

This pipeline contains several activities and whose end result is the scored predictions from the Azure Machine Learning experiment associated with this solution template.

The activities contained in this are:

- HDInsightHive activity using an HDInsightLinkedService that runs a Hive script to perform aggregations and feature engineering necessary for the Azure Machine Learning experiment. The Hive script for this partitioning task is **PrepareMLInput.hql**.
- Copy activity that moves the results from the HDInsightHive activity to a single Azure Storage blob that can be accessed by the AzureMLBatchScoring activity.
- AzureMLBatchScoring activity that calls the Azure Machine Learning experiment which results in the results being put in a single Azure Storage blob.

CopyScoredResultPipeline

This pipeline contains a single activity - a Copy activity that moves the results of the Azure Machine Learning experiment from the **MLScoringPipeline** to the Azure SQL Database that was provisioned as part of the solution template installation.

Azure Machine Learning

The Azure Machine Learning experiment used for this solution template provides the Remaining Useful Life (RUL) of an aircraft engine. The experiment is specific to the data set consumed and therefore will require modification or replacement specific to the data that's brought in.

For information about how the Azure Machine Learning experiment was created, see [Predictive Maintenance: Step 1 of 3, data preparation and feature engineering](#).

Monitor Progress

Once the Data Generator is launched, the pipeline begins to get hydrated and the different components of your solution start kicking into action following the commands issued by the Data Factory. There are two ways you can monitor the pipeline.

1. One of the Stream Analytics job writes the raw incoming data to blob storage. If you click on Blob Storage component of your solution from the screen you successfully deployed the solution and then click Open in the right panel, it will take you to the [management portal](#). Once there, click on Blobs. In the next panel, you will see a list of Containers. Click on **maintenancesadata**. In the next panel, you will see the **rawdata** folder. Inside the rawdata folder, you will see folders with names such as hour=17, hour=18 etc. If you see these folders, it indicates that the raw data is successfully being generated on your computer and stored in blob storage. You should see csv files that should have finite sizes in MB in those folders.
2. The last step of the pipeline is to write data (e.g. predictions from machine learning) into SQL Database. You might have to wait a maximum of three hours for the data to appear in SQL Database. One way to monitor how much data is available in your SQL Database is through [azure portal](#). On the left panel locate SQL DATABASES  and click it. Then locate your database **pmaintainedb** and click on it. On the next page at the bottom, click on MANAGE



Here, you can click on New Query and query for the number of rows (e.g. select count(*) from PMResult). As your database grows, the number of rows in the table should increase.

Power BI Dashboard

Overview

This section describes how to set up Power BI dashboard to visualize your real time data from Azure Stream Analytics (hot path), as well as batch prediction results from Azure machine learning (cold path).

Setup cold path dashboard

In the cold path data pipeline, the essential goal is to get the predictive RUL (remaining useful life) of each aircraft engine once it finishes a flight (cycle). The prediction result is updated every 3 hours for predicting the aircraft engines that have finished a flight during the past 3 hours.

Power BI connects to an Azure SQL database as its data source, where the prediction results are stored. Note: 1) Upon deploying your solution, a real prediction will show up in the database within 3 hours. The pbix file that came with the Generator download contains some seed data so that you may create the Power BI dashboard right away. 2) In this step, the prerequisite is to download and install the free software [Power BI desktop](#).

The following steps will guide you on how to connect the pbix file to the SQL Database that was spun up at the time of solution deployment containing data (e.g.. prediction results) for visualization.

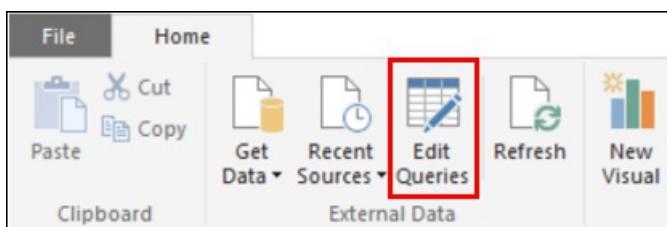
1. Get the database credentials.

You'll need **database server name, database name, user name and password** before moving to next steps. Here are the steps to guide you how to find them.

- Once '**Azure SQL Database**' on your solution template diagram turns green, click it and then click '**Open**'.
- You'll see a new browser tab/window which displays the Azure portal page. Click '**Resource groups**' on the left panel.
- Select the subscription you're using for deploying the solution, and then select '**YourSolutionName_ResourceGroup**'.
- In the new pop out panel, click the icon to access your database. Your database name is next to the this icon (e.g., '**pmaintainedb**'), and the **database server name** is listed under the Server name property and should look similar to **YourSoutionName.database.windows.net**.
- Your database **username** and **password** are the same as the username and password previously recorded during deployment of the solution.

2. Update the data source of the cold path report file with Power BI Desktop.

- In the folder on your PC where you downloaded and unzipped the Generator file, double-click the **PowerBI\PredictiveMaintenanceAerospace.pbix** file. If you see any warning messages when you open the file, ignore them. On the top of the file, click '**Edit Queries**'.



- You'll see two tables, **RemainingUsefulLife** and **PMResult**. Select the first table and click next to

'Source' under '**APPLIED STEPS**' on the right '**Query Settings**' panel. Ignore any warning messages that appear.

- In the pop out window, replace '**Server**' and '**Database**' with your own server and database names, and then click '**OK**'. For server name, make sure you specify the port 1433 (**YourSoutionName.database.windows.net, 1433**). Leave the Database field as **pmaintainededb**. Ignore the warning messages that appear on the screen.
- In the next pop out window, you'll see two options on the left pane (**Windows** and **Database**). Click '**Database**', fill in your '**Username**' and '**Password**' (this is the username and password you entered when you first deployed the solution and created an Azure SQL database). In **Select which level to apply these settings to**, check database level option. Then click '**Connect**'.
- Click on the second table **PMResult** then click next to '**Source**' under '**APPLIED STEPS**' on the right '**Query Settings**' panel, and update the server and database names as in the above steps and click OK.
- Once you're guided back to the previous page, close the window. A message will pop out - click **Apply**. Lastly, click the **Save** button to save the changes. Your Power BI file has now established connection to the server. If your visualizations are empty, make sure you clear the selections on the visualizations to visualize all the data by clicking the eraser icon on the upper right corner of the legends. Use the refresh button to reflect new data on the visualizations. Initially, you will only see the seed data on your visualizations as the data factory is scheduled to refresh every 3 hours. After 3 hours, you will see new predictions reflected in your visualizations when you refresh the data.

- (Optional) Publish the cold path dashboard to [Power BI online](#). Note that this step needs a Power BI account (or Office 365 account).

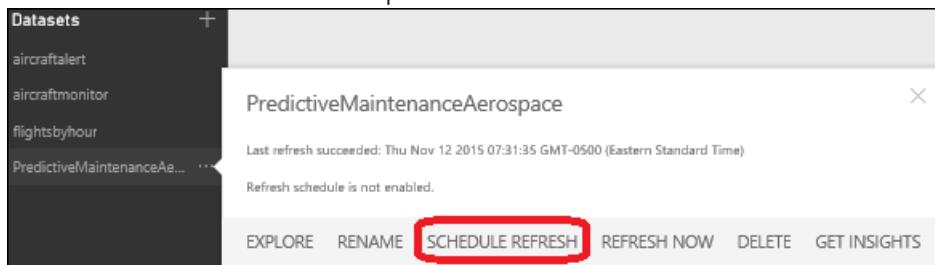
- Click '**Publish**' and few seconds later a window appears displaying "Publishing to Power BI Success!" with a green check mark. Click the link below "Open PredictiveMaintenanceAerospace.pbix in Power BI". To find detailed instructions, see [Publish from Power BI Desktop](#).
- To create a new dashboard: click the + sign next to the **Dashboards** section on the left pane. Enter the name "Predictive Maintenance Demo" for this new dashboard.
- Once you open the report, click to pin all the visualizations to your dashboard. To find detailed instructions, see [Pin a tile to a Power BI dashboard from a report](#). Go to the dashboard page and adjust the size and location of your visualizations and edit their titles. To find detailed instructions on how to edit your tiles, see [Edit a tile -- resize, move, rename, pin, delete, add hyperlink](#). Here is an example dashboard with some cold path visualizations pinned to it. Depending on how long you run your data generator, your numbers on the visualizations may be different.



- To schedule refresh of the data, hover your mouse over the **PredictiveMaintenanceAerospace** dataset,

click  and then choose **Schedule Refresh**.

Note: If you see a warning message, click **Edit Credentials** and make sure your database credentials are the same as those described in step 1.



- Expand the **Schedule Refresh** section. Turn on "keep your data up-to-date".
- Schedule the refresh based on your needs. To find more information, see [Data refresh in Power BI](#).

Setup hot path dashboard

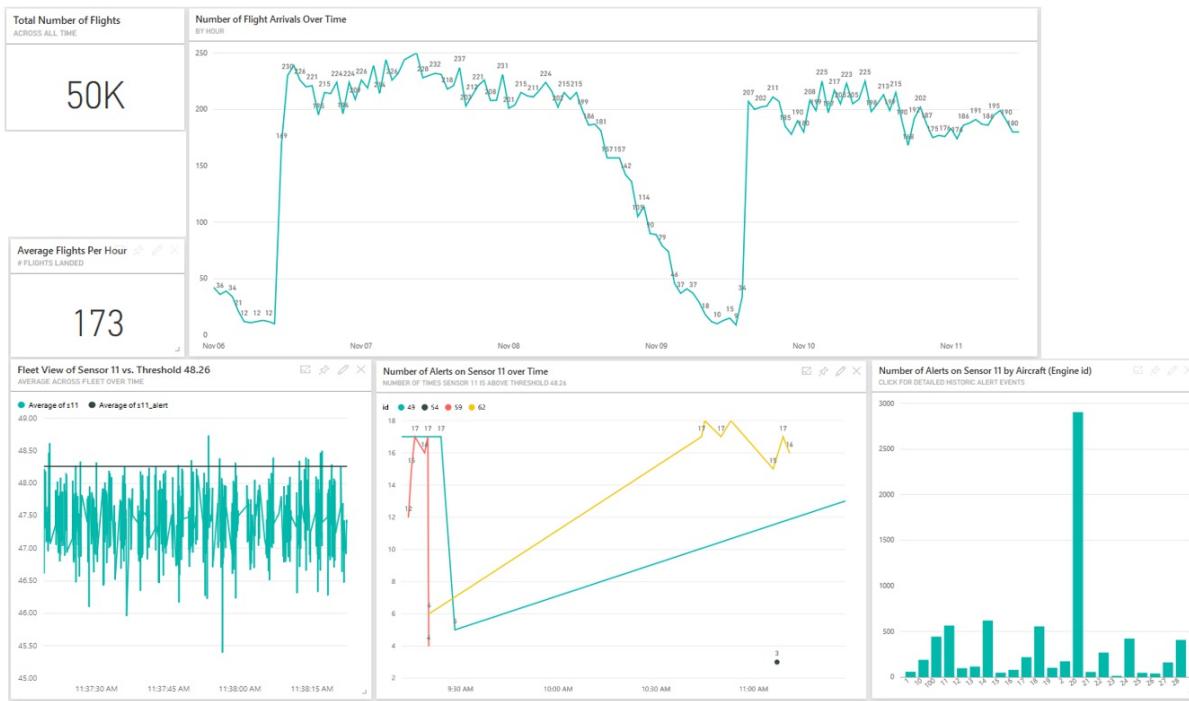
The following steps will guide you how to visualize real time data output from Stream Analytics jobs that were generated at the time of solution deployment. A [Power BI online](#) account is required to perform the following steps. If you don't have an account, you can [create one](#).

1. Add Power BI output in Azure Stream Analytics (ASA).

- You will need to follow the instructions in [Azure Stream Analytics & Power BI: A real-time analytics dashboard for real-time visibility of streaming data](#) to set up the output of your Azure Stream Analytics job as your Power BI dashboard.
- The ASA query has three outputs which are **aircraftmonitor**, **aircraftalert**, and **flightsbyhour**. You can view the query by clicking on query tab. Corresponding to each of these tables, you will need to add an output to ASA. When you add the first output (e.g. **aircraftmonitor**) make sure the **Output Alias**, **Dataset Name** and **Table Name** are the same (**aircraftmonitor**). Repeat the steps to add outputs for **aircraftalert**, and **flightsbyhour**. Once you have added all three output tables and started the ASA job, you should get a confirmation message (e.g., "Starting Stream Analytics job maintenancesa02asapbi succeeded").

2. Log in to [Power BI online](#)

- On the left panel Datasets section in My Workspace, the **DATASET** names **aircraftmonitor**, **aircraftalert**, and **flightsbyhour** should appear. This is the streaming data you pushed from Azure Stream Analytics in the previous step. The dataset **flightsbyhour** may not show up at the same time as the other two datasets due to the nature of the SQL query behind it. However, it should show up after an hour.
 - Make sure the **Visualizations** pane is open and is shown on the right side of the screen.
3. Once you have the data flowing into Power BI, you can start visualizing the streaming data. Below is an example dashboard with some hot path visualizations pinned to it. You can create other dashboard tiles based on appropriate datasets. Depending on how long you run your data generator, your numbers on the visualizations may be different.



4. Here are some steps to create one of the tiles above – the "Fleet View of Sensor 11 vs. Threshold 48.26" tile:

- Click dataset **aircraftmonitor** on the left panel Datasets section.
- Click the **Line Chart** icon.
- Click **Processed** in the **Fields** pane so that it shows under "Axis" in the **Visualizations** pane.
- Click "s11" and "s11_alert" so that they both appear under "Values". Click the small arrow next to **s11** and **s11_alert**, change "Sum" to "Average".
- Click **SAVE** on the top and name the report "aircraftmonitor". The report named "aircraftmonitor" will be shown in the **Reports** section in the **Navigator** pane on the left.
- Click the **Pin Visual** icon on the top right corner of this line chart. A "Pin to Dashboard" window may show up for you to choose a dashboard. Select "Predictive Maintenance Demo", then click "Pin".
- Hover the mouse over this tile on the dashboard, click the "edit" icon on the top right corner to change its title to "Fleet View of Sensor 11 vs. Threshold 48.26" and subtitle to "Average across fleet over time".

How to delete your solution

Please ensure that you stop the data generator when not actively using the solution as running the data generator will incur higher costs. Please delete the solution if you are not using it. Deleting your solution will delete all the components provisioned in your subscription when you deployed the solution. To delete the solution click on your solution name in the left panel of the solution template and click on delete.

Cost estimation tools

The following two tools are available to help you better understand the total costs involved in running the Predictive Maintenance for Aerospace Solution Template in your subscription:

- [Microsoft Azure Cost Estimator Tool \(online\)](#)
- [Microsoft Azure Cost Estimator Tool \(desktop\)](#)

Vehicle telemetry analytics solution playbook

1/17/2017 • 2 min to read • [Edit on GitHub](#)

This **menu** links to the chapters in this playbook.

Overview

Super computers have moved out of the lab and are now parked in our garage! These cutting-edge automobiles contain a myriad of sensors, giving them the ability to track and monitor millions of events every second. We expect that by 2020, most of these cars will have been connected to the Internet. Imagine tapping into this wealth of data to provide greater safety, reliability and a better driving experience! Microsoft has made this dream a reality with Cortana Intelligence.

Microsoft's Cortana Intelligence is a fully managed big data and advanced analytics suite that enables you to transform your data into intelligent action. We want to introduce you to the Cortana Intelligence Vehicle Telemetry Analytics Solution Template. This solution demonstrates how car dealerships, automobile manufacturers, and insurance companies can use the capabilities of Cortana Intelligence to gain real-time and predictive insights on vehicle health and driving habits.

The solution is implemented as a [Lambda architecture pattern](#) showing the full potential of the Cortana Intelligence platform for real-time and batch processing. The solution:

- provides a Vehicle Telematics simulator
- leverages Event Hubs for ingesting millions of simulated vehicle telemetry events into Azure
- uses Stream Analytics to gain real-time insights on vehicle health
- persists the data into long-term storage for richer batch analytics.
- takes advantage of Machine Learning for anomaly detection in real-time and batch processing to gain predictive insights.
- leverages HDInsight to transform data at scale and Data Factory to handle orchestration, scheduling, resource management, and monitoring of the batch processing pipeline
- gives this solution a rich dashboard for real-time data and predictive analytics visualizations using Power BI

Architecture

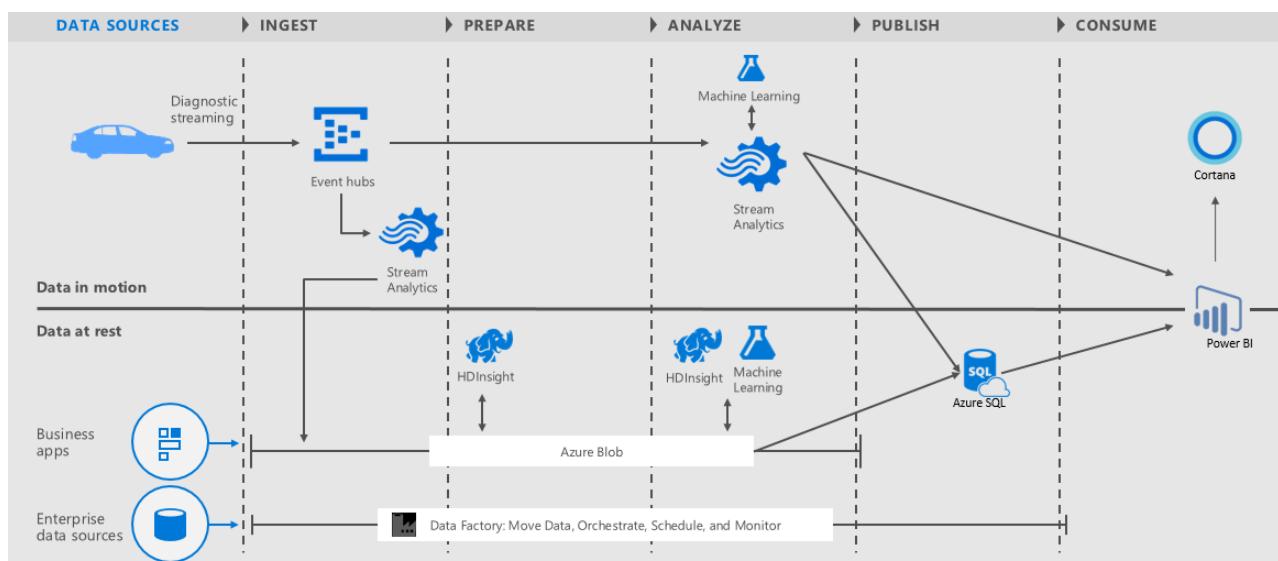


Figure 1 – Vehicle Telemetry Analytics Solution Architecture

This solution includes the following **Cortana Intelligence components** and showcases their end to end integration:

- **Event Hubs** for ingesting millions of vehicle telemetry events into Azure.
- **Stream Analytics** for gaining real-time insights on vehicle health and persists that data into long-term storage for richer batch analytics.
- **Machine Learning** for anomaly detection in real-time and batch processing to gain predictive insights.
- **HDIInsight** is leveraged to transform data at scale
- **Data Factory** handles orchestration, scheduling, resource management and monitoring of the batch processing pipeline.
- **Power BI** gives this solution a rich dashboard for real-time data and predictive analytics visualizations.

This solution accesses two different **data sources**:

- **Simulated vehicle signals and diagnostics:** A vehicle telematics simulator emits diagnostic information and signals that correspond to the state of the vehicle and the driving pattern at a given point in time.
- **Vehicle catalog:** A reference dataset containing a VIN to model mapping.

Vehicle telemetry analytics solution playbook: deep dive into the solution

1/17/2017 • 19 min to read • [Edit on GitHub](#)

This **menu** links to the sections of this playbook:

This section drills down into each of the stages depicted in the Solution Architecture with instructions and pointers for customization.

Data Sources

The solution uses two different data sources:

- **simulated vehicle signals and diagnostic dataset** and
- **vehicle catalog**

A vehicle telematics simulator is included as part of this solution. It emits diagnostic information and signals corresponding to the state of the vehicle and to the driving pattern at a given point in time. Click [Vehicle Telematics Simulator](#) to download the **Vehicle Telematics Simulator Visual Studio Solution** for customizations based on your requirements. The vehicle catalog contains a reference dataset with a VIN to model mapping.

```
*****
Virtual Car Telematics Application
*****
Press Ctrl-C to stop the sender process
11/28/2015 11:57:05 PM > Sending message: {"vin":"UUMDK8HD35HUJ6IQ4","outsideTemperature":1,"engineTemperature":167,"speed":3,"fuel":36,"engineoil":0,"tirepressure":0,"odometer":52416,"city":"Seattle","accelerator_pedal_position":99,"parking_brake_status":true,"brake_pedal_status":true,"headlamp_status":true,"transmission_gear_position":"first","ignition_status":true,"windshield_wiper_status":true,"abs":true,"timestamp":"2015-11-29T07:57:05.3256672Z"}
11/28/2015 11:57:07 PM > Sending message: {"vin":"YFHZ7NRXBR2J3F1QS","outsideTemperature":95,"engineTemperature":395,"speed":84,"fuel":0,"engineoil":48,"tirepressure":46,"odometer":9173,"city":"Redmond","accelerator_pedal_position":79,"parking_brake_status":true,"brake_pedal_status":true,"headlamp_status":true,"transmission_gear_position":"seventh","ignition_status":true,"windshield_wiper_status":true,"abs":true,"timestamp":"2015-11-29T07:57:07.4352649Z"}
11/28/2015 11:57:07 PM > Sending message: {"vin":"75I14TF2XUU5QWB8Q","outsideTemperature":48,"engineTemperature":27,"speed":11,"fuel":23,"engineoil":24,"tirepressure":15,"odometer":44239,"city":"Redmond","accelerator_pedal_position":57,"parking_brake_status":true,"brake_pedal_status":true,"headlamp_status":true,"transmission_gear_position":"first","ignition_status":true,"windshield_wiper_status":true,"abs":true,"timestamp":"2015-11-29T07:57:07.8258954Z"}
11/28/2015 11:57:08 PM > Sending message: {"vin":"6TJNA3DZ87NBG1NMG","outsideTemperature":76,"engineTemperature":161,"speed":63,"fuel":5,"engineoil":38,"tirepressure":31,"odometer":23747,"city":"Bellevue","accelerator_pedal_position":58,"pa
```

Figure 1 – Vehicle Telematics Simulator

This is a JSON-formatted dataset that contains the following schema.

COLUMN	DESCRIPTION	VALUES
--------	-------------	--------

COLUMN	DESCRIPTION	VALUES
VIN	Randomly generated Vehicle Identification Number	This is obtained from a master list of 10,000 randomly generated vehicle identification numbers.
Outside temperature	The outside temperature where the vehicle is driving	Randomly generated number from 0-100
Engine temperature	The engine temperature of the vehicle	Randomly generated number from 0-500
Speed	The engine speed at which the vehicle is driving	Randomly generated number from 0-100
Fuel	The fuel level of the vehicle	Randomly generated number from 0-100 (indicates fuel level percentage)
EngineOil	The engine oil level of the vehicle	Randomly generated number from 0-100 (indicates engine oil level percentage)
Tire pressure	The tire pressure of the vehicle	Randomly generated number from 0-50 (indicates tire pressure level percentage)
Odometer	The odometer reading of the vehicle	Randomly generated number from 0-200000
Accelerator_pedal_position	The accelerator pedal position of the vehicle	Randomly generated number from 0-100 (indicates accelerator level percentage)
Parking_brake_status	Indicates whether the vehicle is parked or not	True or False
Headlamp_status	Indicates where the headlamp is on or not	True or False
Brake_pedal_status	Indicates whether the brake pedal is pressed or not	True or False
Transmission_gear_position	The transmission gear position of the vehicle	States: first, second, third, fourth, fifth, sixth, seventh, eighth
Ignition_status	Indicates whether the vehicle is running or stopped	True or False
Windshield_wiper_status	Indicates whether the windshield wiper is turned or not	True or False
ABS	Indicates whether ABS is engaged or not	True or False
Timestamp	The timestamp when the data point is created	Date

COLUMN	DESCRIPTION	VALUES
City	The location of the vehicle	4 cities in this solution: Bellevue, Redmond, Sammamish, Seattle

The vehicle model reference dataset contains VIN to the model mapping.

VIN	MODEL
FHL3O1SA4IEHB4WU1	Sedan
8J0U8XCPRGW4Z3NQE	Hybrid
WORG68Z2PLTNZDBI7	Family Saloon
JTHMYHQTEPP4WBMRN	Sedan
W9FTHG27LZN1YWO0Y	Hybrid
MHTP9N792PHK08WJM	Family Saloon
EI4QXI2AXVQQING4I	Sedan
5KKR2VB4WHQH97PF8	Hybrid
W9NSZ423XZHAONYXB	Family Saloon
26WJSGHX4MA5ROHNL	Convertible
GHLUB6ONKMOSI7E77	Station Wagon
9C2RHVRVLMEJDBXLP	Compact Car
BRNHVMZOUJ6EOCP32	Small SUV
VCYVW0WUZNBTM594J	Sports Car
HNVCE6YFZSA5M82NY	Medium SUV
4R30FOR7NUOBL05GJ	Station Wagon
WYNIIY42VKV6OQS1J	Large SUV
8Y5QKG27QET1RBK7I	Large SUV
DF6OX2WSRA6511BVG	Coupe
Z2EOZWZBXAEW3E60T	Sedan
M4TV6IEALD5QDS3IR	Hybrid
VHRA1Y2TGTAA84F00H	Family Saloon

VIN	MODEL
ROJAUHT1L1R3BIKIO	Sedan
9230C202Z60XX84AU	Hybrid
T8DNDN5UDCWLM72H	Family Saloon
4WPYRUZII5YV7YA42	Sedan
D1ZVY26UV2BFGHZNO	Hybrid
XUF99EW9OIQOMV7Q7	Family Saloon
80MCL3LGI7XNC21U	Convertible
.....	

To generate simulated data

1. To download the data simulator package, click the arrow on the upper right of the Vehicle Telematics Simulator node . Save and extract the files locally on your machine.

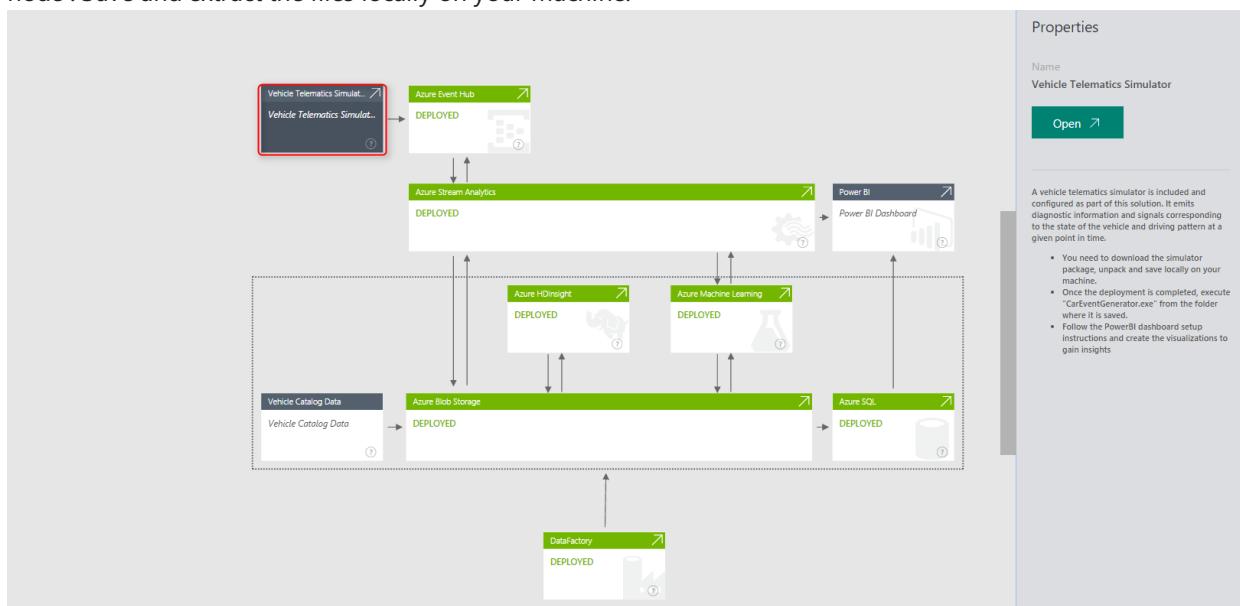


Figure 2 – Vehicle Telemetry Analytics Solution Blueprint

2. On your local machine, go to the folder where you extracted the Vehicle Telematics Simulator package.

Name	Type	Compressed size	Password ...	Size	Ratio	Date modified
ADF	File folder					8/26/2015 11:00 PM
ASA	File folder					8/26/2015 11:00 PM
bin	File folder					8/26/2015 11:00 PM
CarEventGenerator	File folder					8/26/2015 11:00 PM
RealTimeDashboardApp	File folder					8/26/2015 11:00 PM
referencedata	File folder					8/26/2015 11:00 PM
scripts	File folder					8/26/2015 11:00 PM

Figure 3 – Vehicle

Telematics Simulator folder

3. Execute the application **CarEventGenerator.exe**.

References

[Vehicle Telematics Simulator Visual Studio Solution](#)

[Azure Event Hub](#)

[Azure Data Factory](#)

Ingestion

Combinations of Azure Event Hubs, Stream Analytics, and Data Factory are leveraged to ingest the vehicle signals, the diagnostic events, and real-time and batch analytics. All these components are created and configured as part of the solution deployment.

Real-time analysis

The events generated by the Vehicle Telematics Simulator are published to the Event Hub using the Event Hub SDK. The Stream Analytics job ingests these events from the Event Hub and processes the data in real time to analyze the vehicle health.

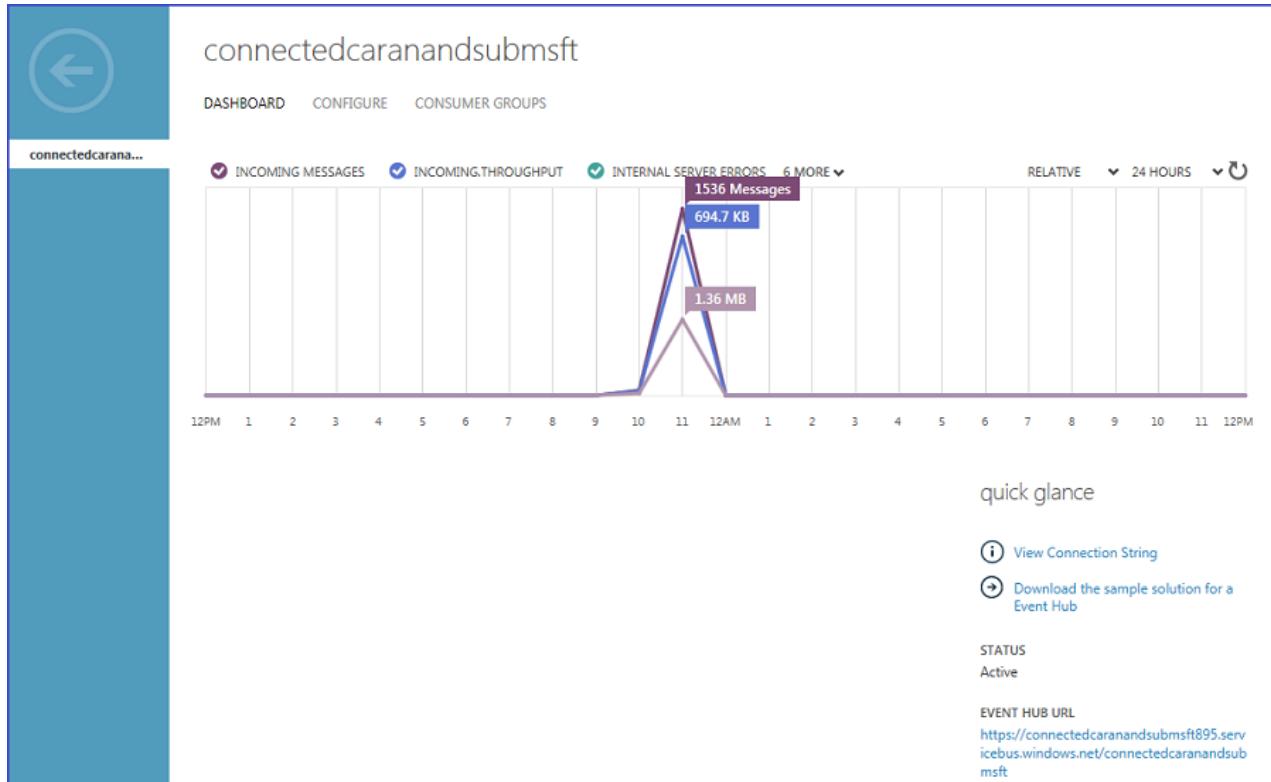


Figure 4 - Event Hub dashboard

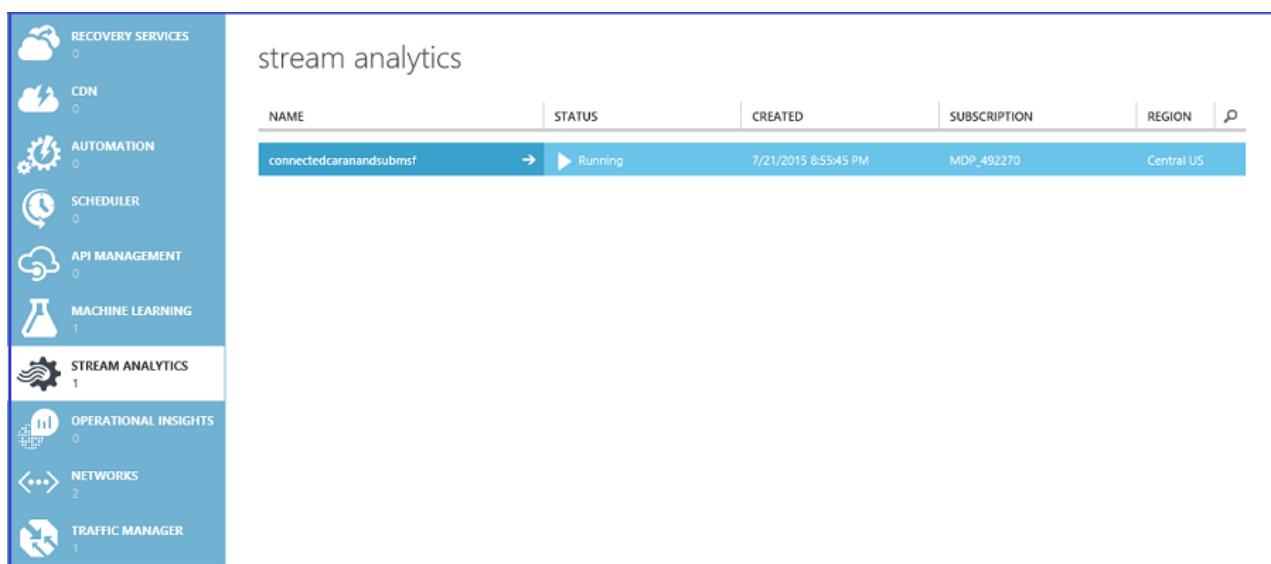


Figure 5 - Stream Analytics job processing data

The Stream Analytics job;

- ingests data from the Event Hub
- performs a join with the reference data to map the vehicle VIN to the corresponding model

- persists them into Azure blob storage for rich batch analytics.

The following Stream Analytics query is used to persist the data into Azure blob storage.

```
Select EventHubSource.vin, BlobSource.Model, EventHubSource.timestamp,
EventHubSource.outsideTemperature, EventHubSource.engineTemperature,
EventHubSource.speed, EventHubSource.fuel, EventHubSource.engineoil,
EventHubSource.tirepressure, EventHubSource.odometer, EventHubSource.city,
EventHubSource.accelerator_pedal_position, EventHubSource.parking_brake_status,
EventHubSource.headlamp_status, EventHubSource.brake_pedal_status,
EventHubSource.transmission_gear_position, EventHubSource.ignition_status,
EventHubSource.windshield_wiper_status, EventHubSource.abs into Blobsink from
EventHubSource join BlobSource on EventHubSource.vin = BlobSource.VIN
```

Figure 6 - Stream Analytics job query for data ingestion

Batch analysis

We are also generating an additional volume of simulated vehicle signals and diagnostic dataset for richer batch analytics. This is required to ensure a good representative data volume for batch processing. For this purpose, we are using a pipeline named "PrepareSampleDataPipeline" in the Azure Data Factory workflow to generate one year's worth of simulated vehicle signals and diagnostic dataset. Click [Data Factory custom activity](#) to download the Data Factory custom DotNet activity Visual Studio solution for customizations based on your requirements.

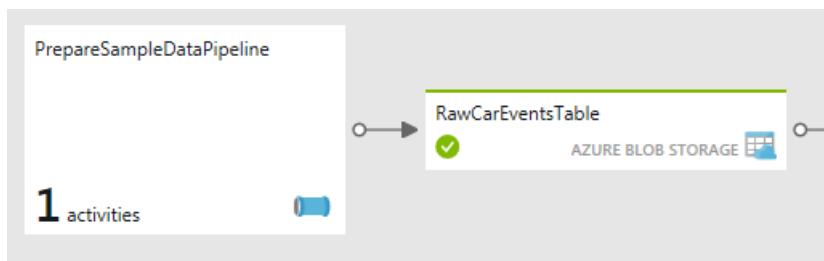


Figure 7 - Prepare Sample data for batch processing workflow

The pipeline consists of a custom ADF .Net Activity, show here:

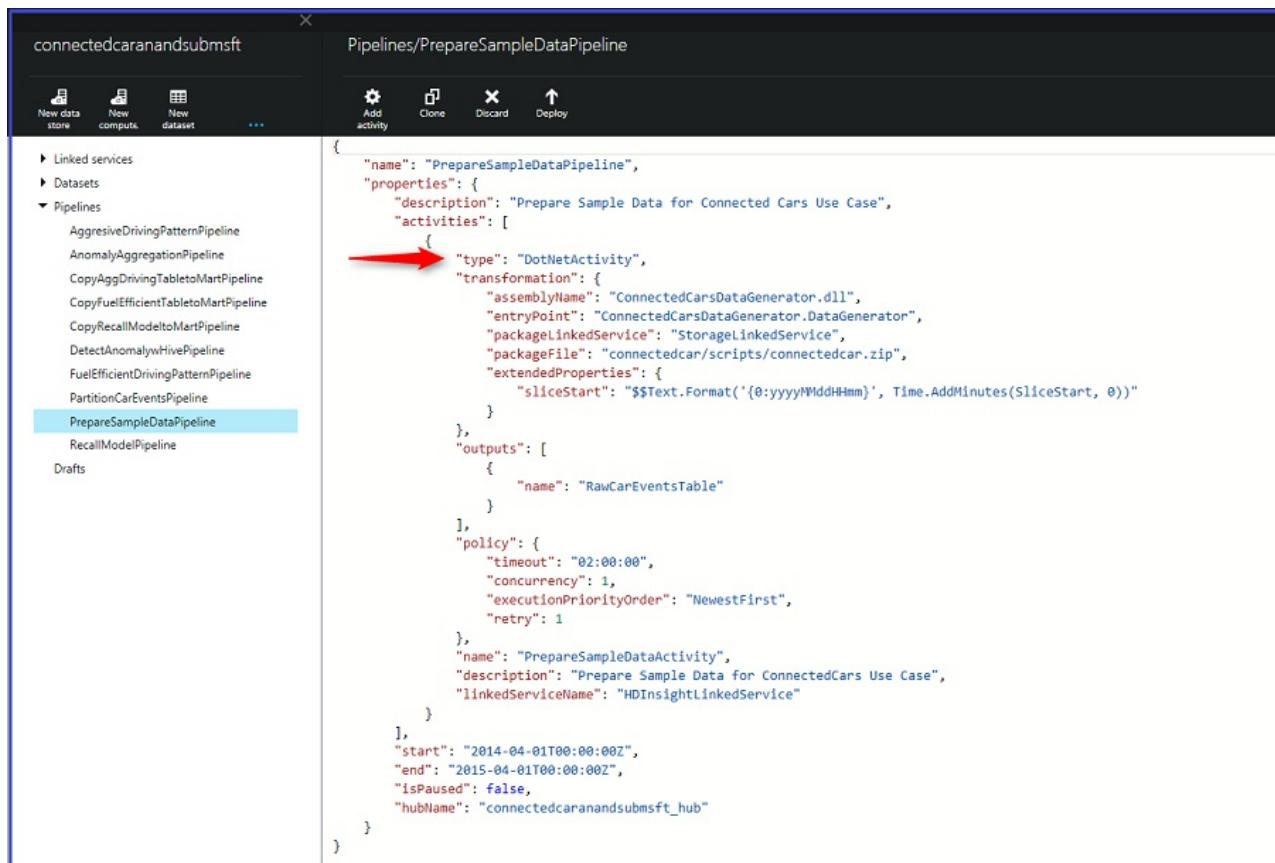


Figure 8 - PrepareSampleDataPipeline

Once the pipeline executes successfully and "RawCarEventsTable" dataset is marked "Ready", one-year worth of simulated vehicle signals and diagnostic data are produced. You see the following folder and file created in your storage account under the "connectedcar" container:

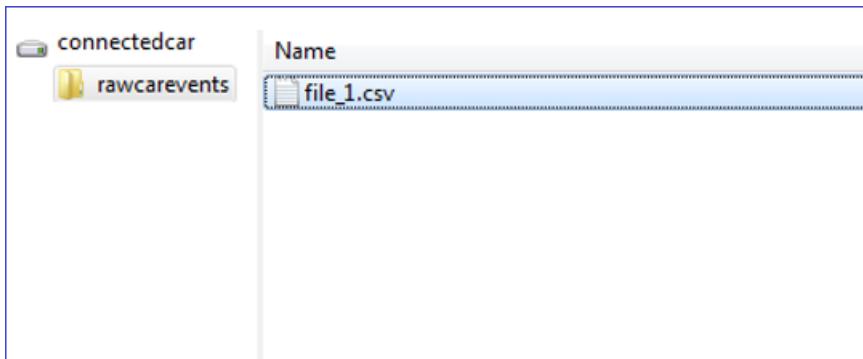


Figure 9 - PrepareSampleDataPipeline Output

References

[Azure Event Hub SDK for stream ingestion](#)

[Azure Data Factory data movement capabilities](#) [Azure Data Factory DotNet Activity](#)

[Azure Data Factory DotNet activity visual studio solution for preparing sample data](#)

Partition the dataset

The raw semi-structured vehicle signals and diagnostic dataset are partitioned in the data preparation step into a YEAR/MONTH format. This partitioning promotes more efficient querying and scalable long-term storage by enabling fault-over from one blob account to the next as the first account fills up.

NOTE

This step in the solution is applicable only to batch processing.

Input and output data management:

- The **output data** (labeled *PartitionedCarEventsTable*) is to be kept for a long period of time as the foundational/"rawest" form of data in the customer's "Data Lake".
- The **input data** to this pipeline would typically be discarded as the output data has full fidelity to the input - it's just stored (partitioned) better for subsequent use.

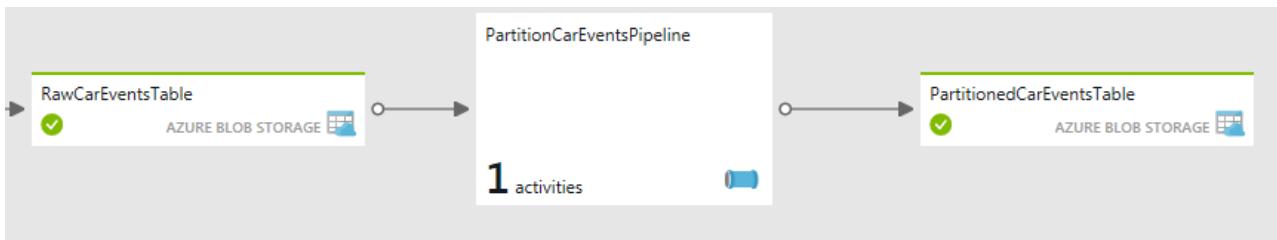


Figure 10 – Partition Car Events workflow

The raw data is partitioned using a Hive HDInsight activity in "PartitionCarEventsPipeline". The sample data generated in step 1 for a year is partitioned by YEAR/MONTH. The partitions are used to generate vehicle signals and diagnostic data for each month (total 12 partitions) of a year.

The screenshot shows the Azure Data Factory Pipeline Editor interface. On the left, there's a navigation pane with options like 'New data store', 'New compute', 'New dataset', 'Add activity', 'Clone', 'Discard', and 'Deploy'. The main area displays a pipeline named 'Pipelines/PartitionCarEventsPipeline'. Under the 'Pipelines' section, several pipelines are listed, including 'AggresiveDrivingPatternPipeline', 'AnomalyAggregationPipeline', 'CopyAggDrivingTabletoMartPipeline', 'CopyFuelEfficientTabletoMartPipeline', 'CopyRecallModeltoMartPipeline', 'DetectAnomalywithHivePipeline', 'FuelEfficientDrivingPatternPipeline', 'PartitionCarEventsPipeline' (which is selected and highlighted in blue), 'PrepareSampleDataPipeline', and 'RecallModelPipeline'. Below this is a 'Drafts' section.

The right side of the screen shows the JSON configuration for the 'PartitionCarEventsPipeline'. A red arrow points to the 'scriptPath' field in the 'activities' section, which is set to 'connectedcar\scripts\partitioncarevents.hql'.

```

{
  "name": "PartitionCarEventsPipeline",
  "properties": {
    "description": "This is a sample pipeline to prepare the car events data for further processing",
    "activities": [
      {
        "type": "HDInsightActivity",
        "transformation": {
          "scriptPath": "connectedcar\\scripts\\partitioncarevents.hql",
          "scriptlinkedService": "StorageLinkedService",
          "type": "Hive",
          "extendedProperties": {
            "RAWINPUT": "wasbs://connectedcar@connectedcarandsubmsft.blob.core.windows.net/rawcarevents/",
            "PARTITIONEDOUTPUT": "wasbs://connectedcar@connectedcarandsubmsft.blob.core.windows.net/partitionedcarevents/",
            "Year": "${$Text.Format('{0:yyyy}',SliceStart)}",
            "Month": "${$Text.Format('{0:MM}',SliceStart)}",
            "Day": "${$Text.Format('{0:Xd}',SliceStart)}"
          }
        },
        "inputs": [
          {
            "name": "RawCarEventsTable"
          }
        ],
        "outputs": [
          {
            "name": "PartitionedCarEventsTable"
          }
        ],
        "policy": {
          "timeout": "01:00:00",
          "concurrency": 1,
          "executionPriorityOrder": "NewestFirst",
          "retry": 2
        },
        "name": "BlobPartitionHiveActivity",
        "linkedServiceName": "HDInsightLinkedService"
      }
    ],
    "start": "2014-04-01T00:00:00Z",
    "end": "2015-04-01T00:00:00Z",
    "isPaused": false,
    "hubName": "connectedcarandsubmsft_hub"
  }
}

```

Figure 11 - PartitionCarEventsPipeline

PartitionConnectedCarEvents Hive Script

The following Hive script, named "partitioncarevents.hql", is used for partitioning and is located in the "\demo\src\connectedcar\scripts" folder of the downloaded zip.

```

SET hive.exec.dynamic.partition=true;
SET hive.exec.dynamic.partition.mode = nonstrict;
set hive.cli.print.header=true;

DROP TABLE IF EXISTS RawCarEvents;
CREATE EXTERNAL TABLE RawCarEvents
(
  vin          string,
  model        string,
  timestamp    string,
  outsidetemperature   string,
  enginetemperature  string,
  speed         string,
  fuel          string,
  engineoil     string,
  tirepressure   string,
  odometer      string,
  city          string,
  accelerator_pedal_position string,
  parking_brake_status  string,
  headlamp_status  string,
  brake_pedal_status string,
  transmission_gear_position string,
  ignition_status   string,
  windshield_wiper_status string,
  abs           string,
  gendate       string
) ROW FORMAT DELIMITED FIELDS TERMINATED BY ';' LINES TERMINATED BY '\0' STORED AS TEXTFILE LOCATION
`${hiveconf:RAWINPUT}`;

```

```

DROP TABLE IF EXISTS PartitionedCarEvents;
CREATE EXTERNAL TABLE PartitionedCarEvents
(
    vin          string,
    model        string,
    timestamp    string,
    outsidetemperature   string,
    enginetemperature   string,
    speed         string,
    fuel          string,
    engineoil     string,
    tirepressure   string,
    odometer      string,
    city          string,
    accelerator_pedal_position  string,
    parking_brake_status   string,
    headlamp_status     string,
    brake_pedal_status   string,
    transmission_gear_position  string,
    ignition_status     string,
    windshield_wiper_status  string,
    abs             string,
    gendate        string
) partitioned by (YearNo int, MonthNo int) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\0' STORED AS TEXTFILE LOCATION '${hiveconf:PARTITIONEDOUTPUT}';

```

```

DROP TABLE IF EXISTS Stage_RawCarEvents;
CREATE TABLE IF NOT EXISTS Stage_RawCarEvents

```

```

(
    vin          string,
    model        string,
    timestamp    string,
    outsidetemperature   string,
    enginetemperature   string,
    speed         string,
    fuel          string,
    engineoil     string,
    tirepressure   string,
    odometer      string,
    city          string,
    accelerator_pedal_position  string,
    parking_brake_status   string,
    headlamp_status     string,
    brake_pedal_status   string,
    transmission_gear_position  string,
    ignition_status     string,
    windshield_wiper_status  string,
    abs             string,
    gendate        string,
    YearNo         int,
    MonthNo        int)

```

```

ROW FORMAT delimited fields terminated by ',' LINES TERMINATED BY '\0';

```

```

INSERT OVERWRITE TABLE Stage_RawCarEvents

```

```

SELECT
    vin,
    model,
    timestamp,
    outsidetemperature,
    enginetemperature,
    speed,
    fuel,
    engineoil,
    tirepressure,
    odometer,
    city,
    accelerator_pedal_position,
    parking_brake_status,

```

```

headlamp_status,
brake_pedal_status,
transmission_gear_position,
ignition_status,
windshield_wiper_status,
abs,
gendate,
Year(gendate),
Month(gendate)

FROM RawCarEvents WHERE Year(gendate) = ${hiveconf:Year} AND Month(gendate) = ${hiveconf:Month};

INSERT OVERWRITE TABLE PartitionedCarEvents PARTITION(YearNo, MonthNo)
SELECT
  vin,
  model,
  timestamp,
  outsidetemperature,
  enginetemperature,
  speed,
  fuel,
  engineoil,
  tirepressure,
  odometer,
  city,
  accelerator_pedal_position,
  parking_brake_status,
  headlamp_status,
  brake_pedal_status,
  transmission_gear_position,
  ignition_status,
  windshield_wiper_status,
  abs,
  gendate,
  YearNo,
  MonthNo
FROM Stage_RawCarEvents WHERE YearNo = ${hiveconf:Year} AND MonthNo = ${hiveconf:Month};

```

Once the pipeline is executed successfully, you see the following partitions generated in your storage account under the "connectedcar" container.

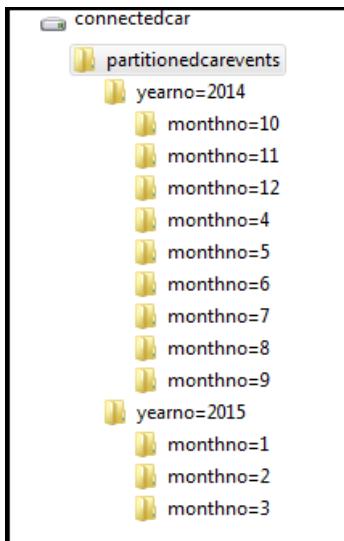


Figure 12 - Partitioned Output

The data is now optimized, is more manageable and ready for further processing to gain rich batch insights.

Data Analysis

In this section, you see how to combine Azure Stream Analytics, Azure Machine Learning, Azure Data Factory, and Azure HDInsight for rich advanced analytics on vehicle health and driving habits. There are three subsections here:

1. **Machine Learning**: This subsection contains information on the anomaly detection experiment that we have used in this solution to predict vehicles requiring servicing maintenance and vehicles requiring recalls due to safety issues.
2. **Real-time analysis**: This subsection contains information regarding the real-time analytics using the Stream Analytics Query Language and operationalizing the machine learning experiment in real time using a custom application.
3. **Batch analysis**: This subsection contains information regarding the transforming and processing of the batch data using Azure HDInsight and Azure Machine Learning operationalized by Azure Data Factory.

Machine Learning

Our goal here is to predict the vehicles that require maintenance or recall based on certain heath statistics. We make the following assumptions

- If one of the following three conditions are true, the vehicles require **servicing maintenance**:
 - Tire pressure is low
 - Engine oil level is low
 - Engine temperature is high
- If one of the following conditions are true, the vehicles may have a **safety issue** and require **recall**:
 - Engine temperature is high but outside temperature is low
 - Engine temperature is low but outside temperature is high

Based on the previous requirements, we have created two separate models to detect anomalies, one for vehicle maintenance detection, and one for vehicle recall detection. In both these models, the built-in Principal Component Analysis (PCA) algorithm is used for anomaly detection.

Maintenance detection model

If one of three indicators - tire pressure, engine oil, or engine temperature - satisfies its respective condition, the maintenance detection model reports an anomaly. As a result, we only need to consider these three variables in building the model. In our experiment in Azure Machine Learning, we first use a **Select Columns in Dataset** module to extract these three variables. Next we use the PCA-based anomaly detection module to build the anomaly detection model.

Principal Component Analysis (PCA) is an established technique in machine learning that can be applied to feature selection, classification, and anomaly detection. PCA converts a set of case containing possibly correlated variables, into a set of values called principal components. The key idea of PCA-based modeling is to project data onto a lower-dimensional space so that features and anomalies can be more easily identified.

For each new input to the detection model, the anomaly detector first computes its projection on the eigenvectors, and then computes the normalized reconstruction error. This normalized error is the anomaly score. The higher the error, the more anomalous the instance is.

In the maintenance detection problem, each record can be considered as a point in a 3-dimensional space defined by tire pressure, engine oil, and engine temperature coordinates. To capture these anomalies, we can project the original data in the 3-dimensional space onto a 2-dimensional space using PCA. Thus, we set the parameter Number of components to use in PCA to be 2. This parameter plays an important role in applying PCA-based anomaly detection. After projecting data using PCA, we can identify these anomalies more easily.

Recall anomaly detection model In the recall anomaly detection model, we use the Select Columns in Dataset and PCA-based anomaly detection modules in a similar way. Specifically, we first extract three variables - engine temperature, outside temperature, and speed - using the **Select Columns in Dataset** module. We also include the

speed variable since the engine temperature typically is correlated to the speed. Next we use PCA-based anomaly detection module to project the data from the 3-dimensional space onto a 2-dimensional space. The recall criteria are satisfied and so the vehicle requires recall when engine temperature and outside temperature are highly negatively correlated. Using PCA-based anomaly detection algorithm, we can capture the anomalies after performing PCA.

When training either model, we need to use normal data, which does not require maintenance or recall as the input data to train the PCA-based anomaly detection model. In the scoring experiment, we use the trained anomaly detection model to detect whether or not the vehicle requires maintenance or recall.

Real-time analysis

The following Stream Analytics SQL Query is used to get the average of all the important vehicle parameters like vehicle speed, fuel level, engine temperature, odometer reading, tire pressure, engine oil level, and others. The averages are used to detect anomalies, issue alerts, and determine the overall health conditions of vehicles operated in specific region and then correlate it to demographics.

```
select BlobSource.Model, EventHubSource.city, count(vin) as cars, avg(EventHubSource.engineTemperature) as engineTemperature, avg(EventHubSource.speed) as Speed, avg(EventHubSource.fuel) as Fuel, avg(EventHubSource.engineoil) as EngineOil, avg(EventHubSource.tirepressure) as TirePressure, avg(EventHubSource.odometer) as Odometer into SQLSink from EventHubSource join BlobSource on EventHubSource.vin = BlobSource.VIN group by BlobSource.model, EventHubSource.city, TumblingWindow(second, 3)
```

Figure 13 – Stream Analytics query for real-time processing

All the averages are calculated over a 3-second TumblingWindow. We are using TubmlingWindow in this case since we require non-overlapping and contiguous time intervals.

To learn more about all the "Windowing" capabilities in Azure Stream Analytics, click [Windowing \(Azure Stream Analytics\)](#).

Real-time prediction

An application is included as part of the solution to operationalize the machine learning model in real time. This application called "RealTimeDashboardApp" is created and configured as part of the solution deployment. The application performs the following:

1. Listens to an Event Hub instance where Stream Analytics is publishing the events in a pattern continuously.

```
Select EventHubSource.vin, BlobSource.Model, EventHubSource.timestamp, EventHubSource.outsideTemperature, EventHubSource.engineTemperature, EventHubSource.speed, EventHubSource.fuel, EventHubSource.engineoil, EventHubSource.tirepressure, EventHubSource.odometer, EventHubSource.city, EventHubSource.accelerator_pedal_position, EventHubSource.parking_brake_status, EventHubSource.headlamp_status, EventHubSource.brake_pedal_status, EventHubSource.transmission_gear_position, EventHubSource.ignition_status, EventHubSource.windshield_wiper_status, EventHubSource.abs into EventHubOut from EventHubSource join BlobSource on EventHubSource.vin = BlobSource.VIN
```

Figure 14 – Stream Analytics query for publishing the data to an output Event Hub instance

2. For every event that this application receives:

- Processes the data using Machine Learning Request-Response Scoring (RRS) endpoint. The RRS endpoint is automatically published as part of the deployment.
- The RRS output is published to a Power BI dataset using the push APIs.

This pattern is also applicable to scenarios in which you want to integrate a Line of Business (LoB) application with the real-time analytics flow, for scenarios such as alerts, notifications, and messaging.

Click [RealtimeDashboardApp download](#) to download the RealtimeDashboardApp Visual Studio solution for customizations.

To execute the Real-time Dashboard Application

1. Click the Power BI node on the diagram view and click the "Download Real-time Dashboard Application" link on

the properties pane.

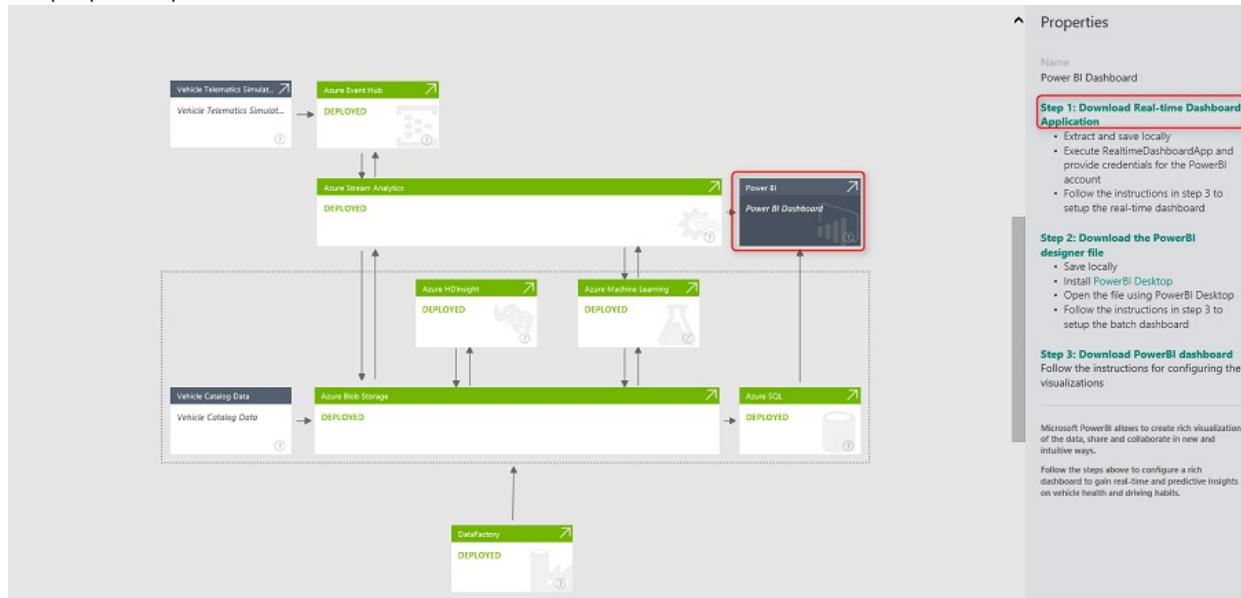


Figure 15 – Power BI dashboard setup instructions

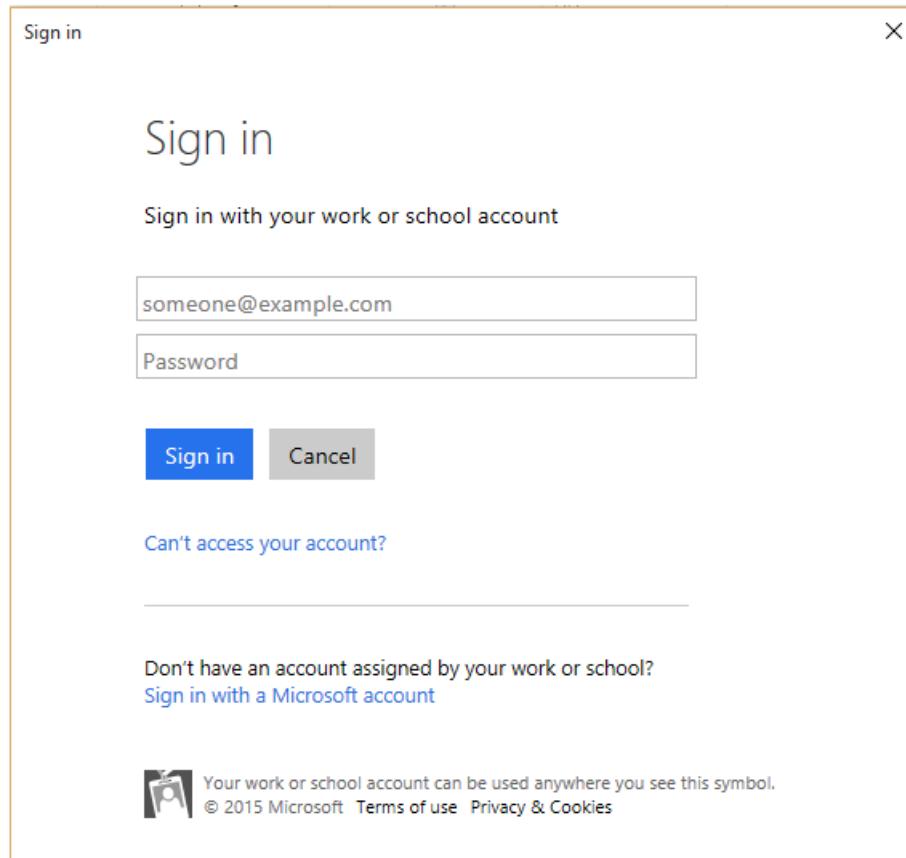
2. Extract and save locally

Name	Date modified	Type	Size
ADF	7/21/2015 6:33 PM	File folder	
ASA	7/21/2015 6:33 PM	File folder	
bin	7/21/2015 6:33 PM	File folder	
CarEventGenerator	7/27/2015 11:26 PM	File folder	
RealTimeDashboardApp	7/21/2015 6:34 PM	File folder	
referencedata	7/21/2015 6:34 PM	File folder	
scripts	7/21/2015 6:34 PM	File folder	

Figure 16 – RealtimeDashboardApp folder

3. Execute the application RealtimeDashboardApp.exe

4. Provide valid Power BI credentials, sign in and click Accept



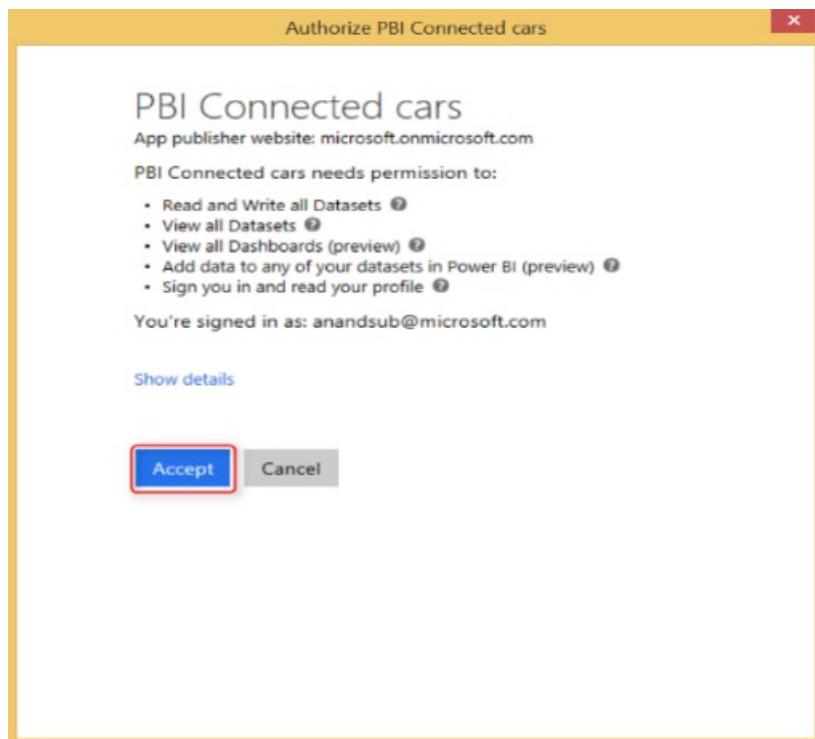


Figure 17 – RealtimeDashboardApp: Sign-in to Power BI

NOTE

If you want to flush the Power BI dataset, execute the RealtimeDashboardApp with the "flushdata" parameter:

```
RealtimeDashboardApp.exe -flushdata
```

Batch analysis

The goal here is to show how Contoso Motors utilizes the Azure compute capabilities to harness big data to gain rich insights on driving pattern, usage behavior, and vehicle health. This makes it possible to:

- Improve the customer experience and make it cheaper by providing insights on driving habits and fuel efficient driving behaviors
- Learn proactively about customers and their driving patters to govern business decisions and provide the best in class products & services

In this solution, we are targeting the following metrics:

1. **Aggressive driving behavior:** Identifies the trend of the models, locations, driving conditions, and time of the year to gain insights on aggressive driving patterns. Contoso Motors can use these insights for marketing campaigns, driving new personalized features and usage-based insurance.
2. **Fuel efficient driving behavior:** Identifies the trend of the models, locations, driving conditions, and time of the year to gain insights on fuel efficient driving patterns. Contoso Motors can use these insights for marketing campaigns, driving new features and proactive reporting to the drivers for cost effective and environment friendly driving habits.
3. **Recall models:** Identifies models requiring recalls by operationalizing the anomaly detection machine learning experiment

Let's look into the details of each of these metrics,

Aggressive driving pattern

The partitioned vehicle signals and diagnostic data are processed in the pipeline named

"AggressiveDrivingPatternPipeline" using Hive to determine the models, location, vehicle, driving conditions, and other parameters that exhibits aggressive driving pattern.

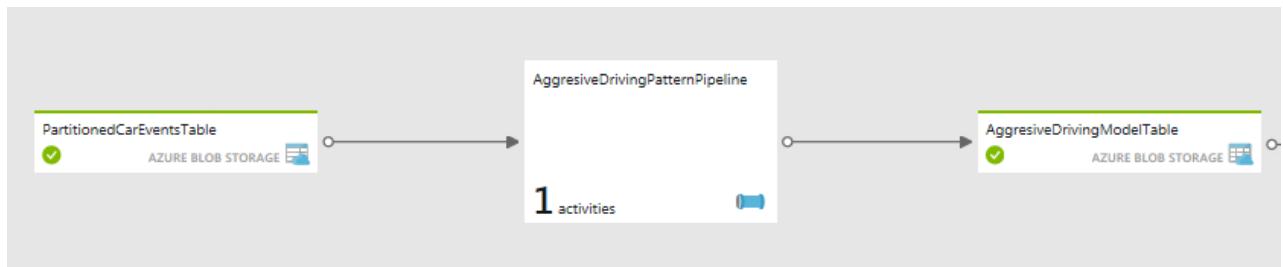


Figure 18 – Aggressive driving pattern workflow

Aggressive driving pattern Hive query

The Hive script named "aggresivedriving.hql" used for analyzing aggressive driving condition pattern is located at "\demo\src\connectedcar\scripts" folder of the downloaded zip.

```

DROP TABLE IF EXISTS PartitionedCarEvents;
CREATE EXTERNAL TABLE PartitionedCarEvents
(
    vin          string,
    model        string,
    timestamp    string,
    outsidetemperature   string,
    enginetemperature   string,
    speed         string,
    fuel          string,
    engineoil     string,
    tirepressure  string,
    odometer      string,
    city          string,
    accelerator_pedal_position  string,
    parking_brake_status   string,
    headlamp_status    string,
    brake_pedal_status  string,
    transmission_gear_position  string,
    ignition_status    string,
    windshield_wiper_status  string,
    abs            string,
    gendate        string
)

) ROW FORMAT DELIMITED FIELDS TERMINATED BY',' LINES TERMINATED BY'10' STORED AS TEXTFILE LOCATION
`${hiveconf:PARTITIONEDINPUT}`;

DROP TABLE IF EXISTS CarEventsAggresive;
CREATE EXTERNAL TABLE CarEventsAggresive
(
    vin          string,
    model        string,
    timestamp    string,
    city          string,
    speed         string,
    transmission_gear_position  string,
    brake_pedal_status  string,
    Year          string,
    Month         string
)

) ROW FORMAT DELIMITED FIELDS TERMINATED BY',' LINES TERMINATED BY'10' STORED AS TEXTFILE LOCATION
`${hiveconf:AGGRESIVEOUTPUT}`;

INSERT OVERWRITE TABLE CarEventsAggresive
select
vin,
model,
timestamp,
city,
speed,
transmission_gear_position,
brake_pedal_status,
"${hiveconf:Year}" as Year,
"${hiveconf:Month}" as Month
from PartitionedCarEvents
where transmission_gear_position IN ('fourth','fifth','sixth','seventh','eighth') AND brake_pedal_status = 'l' AND speed >= '50'

```

It uses the combination of vehicle's transmission gear position, brake pedal status, and speed to detect reckless/aggressive driving behavior based on braking pattern at high speed.

Once the pipeline is executed successfully, you see the following partitions generated in your storage account under the "connectedcar" container.

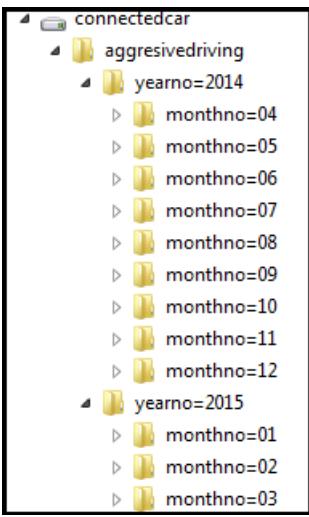


Figure 19 – AggressiveDrivingPatternPipeline output

Fuel efficient driving pattern

The partitioned vehicle signals and diagnostic data are processed in the pipeline named "FuelEfficientDrivingPatternPipeline". Hive is used to determine the models, location, vehicle, driving conditions, and other properties that exhibit fuel efficient driving pattern.

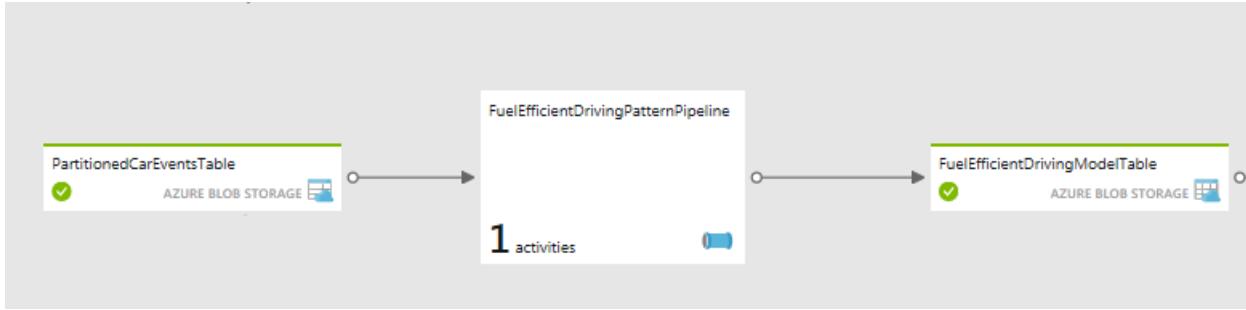


Figure 20 – Fuel-efficient driving pattern workflow

Fuel efficient driving pattern Hive query

The Hive script named "fuelefficientdriving.hql" used for analyzing aggressive driving condition pattern is located at "\demo\src\connectedcar\scripts" folder of the downloaded zip.

```

DROP TABLE IF EXISTS PartitionedCarEvents;
CREATE EXTERNAL TABLE PartitionedCarEvents
(
    vin          string,
    model        string,
    timestamp    string,
    outsidetemperature   string,
    enginetemperature   string,
    speed         string,
    fuel          string,
    engineoil     string,
    tirepressure  string,
    odometer      string,
    city          string,
    accelerator_pedal_position  string,
    parking_brake_status   string,
    headlamp_status    string,
    brake_pedal_status  string,
    transmission_gear_position  string,
    ignition_status    string,
    windshield_wiper_status  string,
    abs            string,
    gendate        string
)

) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '10' STORED AS TEXTFILE LOCATION
`${hiveconf:PARTITIONEDINPUT}`;

DROP TABLE IF EXISTS FuelEfficientDriving;
CREATE EXTERNAL TABLE FuelEfficientDriving
(
    vin          string,
    model        string,
    city          string,
    speed         string,
    transmission_gear_position  string,
    brake_pedal_status  string,
    accelerator_pedal_position  string,
    Year          string,
    Month         string
)

) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '10' STORED AS TEXTFILE LOCATION
`${hiveconf:FUELEFFICIENTOUTPUT}`;

INSERT OVERWRITE TABLE FuelEfficientDriving
select
vin,
model,
city,
speed,
transmission_gear_position,
brake_pedal_status,
accelerator_pedal_position,
"${hiveconf:Year}" as Year,
"${hiveconf:Month}" as Month
from PartitionedCarEvents
where transmission_gear_position IN ('fourth','fifth','sixth','seventh','eighth') AND parking_brake_status = '0' AND brake_pedal_status = '0' AND
speed <= '60' AND accelerator_pedal_position >= '50'

```

It uses the combination of vehicle's transmission gear position, brake pedal status, speed, and accelerator pedal position to detect fuel efficient driving behavior based on acceleration, braking, and speed patterns.

Once the pipeline is executed successfully, you see the following partitions generated in your storage account under the "connectedcar" container.

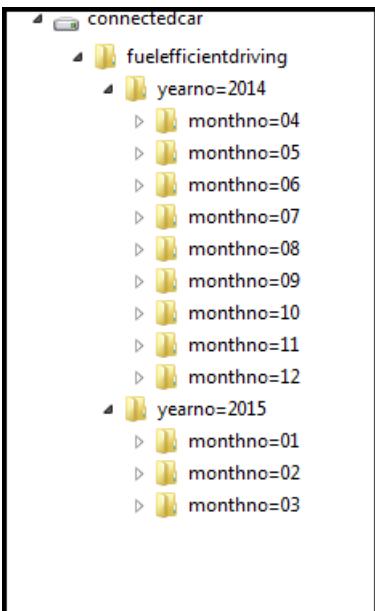


Figure 21 – FuelEfficientDrivingPatternPipeline output

Recall Predictions

The machine learning experiment is provisioned and published as a web service as part of the solution deployment. The batch scoring end point is leveraged in this workflow, registered as a data factory linked service and operationalized using data factory batch scoring activity.

```
{
  "name": "AzureMLAnomalydetectionendpoint",
  "properties": {
    "hubName": "connectedcaranandsubnsf_hub",
    "type": "AzureML",
    "typeProperties": {
      "mlEndpoint": "https://ussouthcentral.services.azureml.net/workspaces/e501a87b96754eae8c2512b6d591097a/services/99d9a2e7c480467283ea95d1e1075274/jobs",
      "apiKey": "*****"
    }
  }
}
```

Figure 22 – Machine learning endpoint registered as a linked service in data factory

The registered linked service is used in the DetectAnomalyPipeline to score the data using the anomaly detection model.

```

{
    "type": "AzureMLBatchScoring",
    "typeProperties": {},
    "inputs": [
        {
            "name": "PartitionedCarEventsTableforMLCSV"
        }
    ],
    "outputs": [
        {
            "name": "CarEventsAnomalyTable"
        }
    ],
    "policy": {
        "timeout": "01:00:00",
        "concurrency": 1,
        "executionPriorityOrder": "NewestFirst",
        "retry": 2
    },
    "scheduler": {
        "frequency": "Month",
        "interval": 1,
        "style": "StartOfInterval"
    },
    "name": "AMLBatchScoringforAnomalyActivity",
    "linkedServiceName": "AzureMLAnomalydetectionendpoint"
},
],
"start": "2014-07-01T00:00:00Z",
"end": "2015-07-01T00:00:00Z",
"isPaused": false,
"hubName": "connectedcaranandsubmsf_hub",
"pipelineMode": "Scheduled"
}
}

```

Figure 23 – Azure Machine Learning Batch Scoring activity in data factory

There are few steps performed in this pipeline for data preparation so that it can be operationalized with the batch scoring web service.

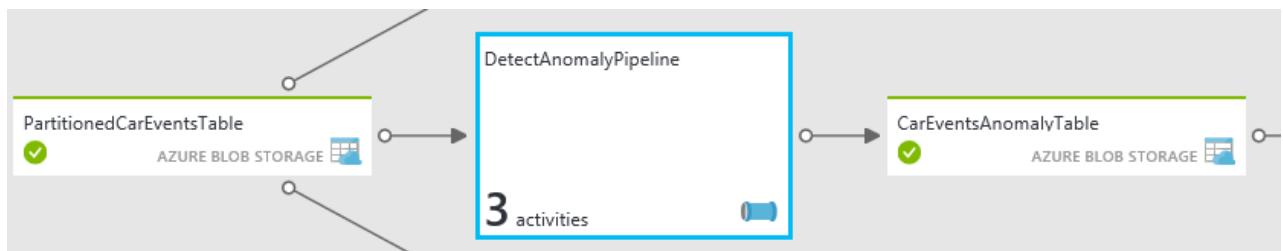


Figure 24 – DetectAnomalyPipeline for predicting vehicles requiring recalls

Anomaly detection Hive query

Once the scoring is completed, an HDInsight activity is used to process and aggregate the data that are categorized as anomalies by the model with a probability score of 0.60 or higher.

```

DROP TABLE IF EXISTS CarEventsAnomaly;
CREATE EXTERNAL TABLE CarEventsAnomaly
(
    vin          string,
    model        string,
    gendate     string,
    outsidetemperature   string,
    enginetemperature   string,
    speed         string,
    fuel          string,
    engineoil     string,
    tirepressure   string,
    odometer      string,
    city          string,
    accelerator_pedal_position  string,
    parking_brake_status   string,
    headlamp_status     string,
    brake_pedal_status   string,
    transmission_gear_position  string,
    ignition_status     string,
    windshield_wiper_status   string,
    abs            string,
    maintenanceLabel   string,
    maintenanceProbability   string,
    RecallLabel      string,
    RecallProbability   string
)

) ROW FORMAT DELIMITED FIELDS TERMINATED BY',' LINES TERMINATED BY'10' STORED AS TEXTFILE LOCATION
`${hiveconf:ANOMALYOUTPUT}`;

DROP TABLE IF EXISTS RecallModel;
CREATE EXTERNAL TABLE RecallModel
(
    vin          string,
    model        string,
    city          string,
    outsidetemperature   string,
    enginetemperature   string,
    speed         string,
    Year          string,
    Month         string
)

) ROW FORMAT DELIMITED FIELDS TERMINATED BY',' LINES TERMINATED BY'10' STORED AS TEXTFILE LOCATION
`${hiveconf:RECALLMODELOUTPUT}`;

INSERT OVERWRITE TABLE RecallModel
select
    vin,
    model,
    city,
    outsidetemperature,
    enginetemperature,
    speed,
    "${hiveconf:Year}" as Year,
    "${hiveconf:Month}" as Month
from CarEventsAnomaly
where RecallLabel ='1' AND RecallProbability >= '0.60'

```

Once the pipeline is executed successfully, you see the following partitions generated in your storage account under the "connectedcar" container.

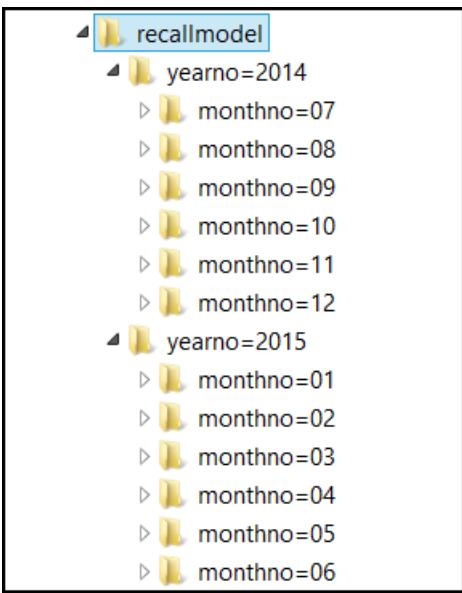


Figure 25 – DetectAnomalyPipeline output

Publish

Real-time analysis

One of the queries in the Stream Analytics job publishes the events to an output Event Hub instance.

NAME	SINK	DIAGNOSIS
BlobSink	Blob storage	✓ OK
EventHubOut	Event Hub	✓ OK
SQISink	SQL Database	✓ OK

Figure 26 – Stream Analytics job publishes to an output Event Hub instance

```
Select EventHubSource.vin, BlobSource.Model, EventHubSource.timestamp, EventHubSource.outsideTemperature,
EventHubSource.engineTemperature, EventHubSource.speed, EventHubSource.fuel, EventHubSource.engineoil,
EventHubSource.tirepressure, EventHubSource.odometer, EventHubSource.city,
EventHubSource.accelerator_pedal_position, EventHubSource.parking_brake_status,
EventHubSource.headlamp_status,
EventHubSource.brake_pedal_status, EventHubSource.transmission_gear_position,
EventHubSource.ignition_status, EventHubSource.windshield_wiper_status, EventHubSource.abs into
EventHubOut from EventHubSource join BlobSource on EventHubSource.vin = BlobSource.VIN
```

Figure 27 – Stream Analytics query to publish to the output Event Hub instance

This stream of events is consumed by the RealTimeDashboardApp included in the solution. This application leverages the Machine Learning Request-Response web service for real-time scoring and publishes the resultant data to a Power BI dataset for consumption.

Batch analysis

The results of the batch and real-time processing are published to the Azure SQL Database tables for consumption. The Azure SQL Server, Database, and the tables are created automatically as part of the setup script.



Figure 28 – Batch processing results copy to data mart workflow

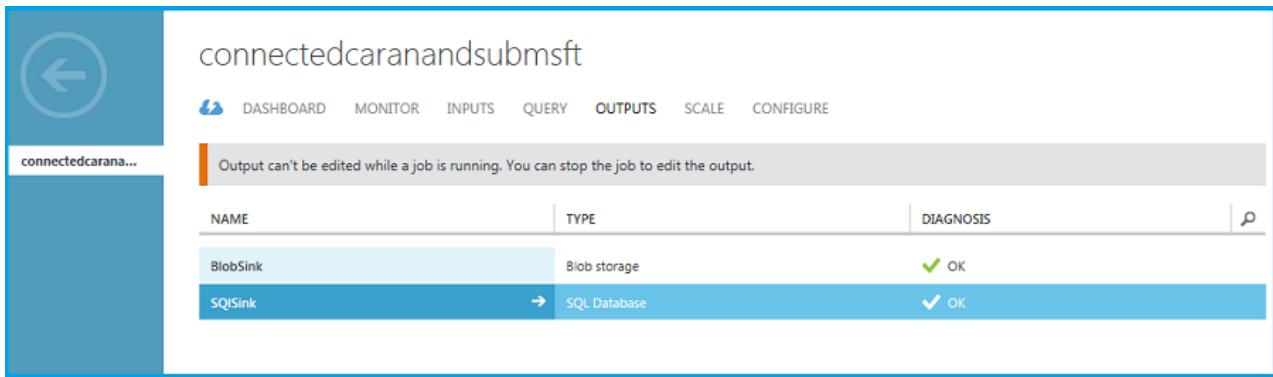


Figure 29 – Stream Analytics job publishes to data mart

The screenshot shows the configuration for a Stream Analytics job named 'general'. On the left, there are two tabs: 'BlobSink' (selected) and 'SQISink'. The main area contains the following fields:

- SUBSCRIPTION:** A dropdown menu set to 'Use SQL Database from Another Subscription'.
- SERVER NAME:** 'zrzel4v9yp'
- DATABASE:** 'connectedcar'
- USERNAME:** 'mylogin'
- PASSWORD:** (Redacted)
- TABLE:** 'NearRealTimeAveragesReport'

Figure 30 – Data mart setting in Stream Analytics job

Consume

Power BI gives this solution a rich dashboard for real-time data and predictive analytics visualizations.

Click here for detailed instructions on setting up the Power BI reports and the dashboard. The final dashboard looks like this:



Figure 31 - Power BI Dashboard

Summary

This document contains a detailed drill-down of the Vehicle Telemetry Analytics Solution. This showcases a lambda architecture pattern for real-time and batch analytics with predictions and actions. This pattern applies to a wide range of use cases that require hot path (real-time) and cold path (batch) analytics.

Vehicle telemetry analytics solution template Power BI Dashboard setup instructions

1/17/2017 • 11 min to read • [Edit on GitHub](#)

This **menu** links to the chapters in this playbook.

The Vehicle Telemetry Analytics solution showcases how car dealerships, automobile manufacturers and insurance companies can leverage the capabilities of Cortana Intelligence to gain real-time and predictive insights on vehicle health and driving habits to drive improvements in the area of customer experience, R&D and marketing campaigns. This document contains step by step instructions on how you can configure the Power BI reports and dashboard once the solution is deployed in your subscription.

Prerequisites

1. Deploy the Vehicle Telemetry Analytics solution by navigating to
<https://gallery.cortanaanalytics.com/SolutionTemplate/Vehicle-Telemetry-Analytics-3>
2. [Install Microsoft Power BI Desktop](#)
3. An [Azure subscription](#). If you don't have an Azure subscription, get started with Azure free subscription
4. Microsoft Power BI account

Cortana Intelligence Suite Components

As part of the Vehicle Telemetry Analytics solution template, the following Cortana Intelligence services are deployed in your subscription.

- **Event Hubs** for ingesting millions of vehicle telemetry events into Azure.
- **Stream Analytics** for gaining real-time insights on vehicle health and persists that data into long-term storage for richer batch analytics.
- **Machine Learning** for anomaly detection in real-time and batch processing to gain predictive insights.
- **HDInsight** is leveraged to transform data at scale
- **Data Factory** handles orchestration, scheduling, resource management and monitoring of the batch processing pipeline.

Power BI gives this solution a rich dashboard for real-time data and predictive analytics visualizations.

The solution uses two different data sources: **Simulated vehicle signals and diagnostic dataset** and **vehicle catalog**.

A vehicle telematics simulator is included as part of this solution. It emits diagnostic information and signals corresponding to the state of the vehicle and driving pattern at a given point in time.

The Vehicle Catalog is a reference dataset containing VIN to model mapping

Power BI Dashboard Preparation

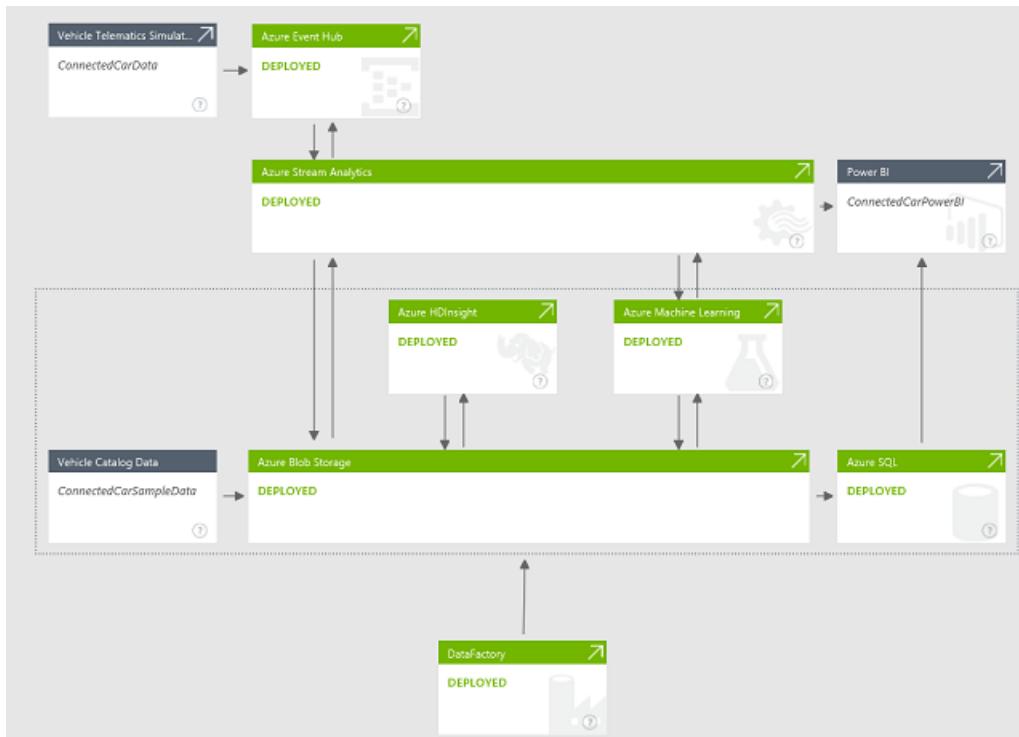
Deployment

Once the deployment is completed, you should see the following diagram with all of these components marked in GREEN.

- To navigate to the corresponding services to validate whether all of these have deployed successfully, click the

arrow on the upper right of the green nodes.

- To download the data simulator package, click the arrow on the upper right on the **Vehicle Telematics Simulator** node. Save and extract the files locally on your machine.



Now, you are ready to configure the Power BI dashboard with rich visualizations to gain real-time and predictive insights on vehicle health and driving habits. It takes about 45 minutes to an hour to create all the reports and configure the dashboard.

Setup Power BI Real-Time Dashboard

Generate simulated data

1. On your local machine, go to the folder where you extracted the Vehicle Telematics Simulator package.

Name	Type	Compressed size	Password ...	Size	Ratio	Date modified
ADF	File folder					8/26/2015 11:00 PM
ASA	File folder					8/26/2015 11:00 PM
bin	File folder					8/26/2015 11:00 PM
CarEventGenerator	File folder					8/26/2015 11:00 PM
RealTimeDashboardApp	File folder					8/26/2015 11:00 PM
referencedata	File folder					8/26/2015 11:00 PM
scripts	File folder					8/26/2015 11:00 PM

2. Execute the application **CarEventGenerator.exe**.
3. It emits diagnostic information and signals corresponding to the state of the vehicle and driving pattern at a given point in time. This is published to an Azure Event Hub instance that is configured as part of your deployment.

```

"parking_brake_status":true,"brake_pedal_status":true,"headlamp_status":true,"transmission_gear_position":"eight","ignition_status":true,"windshield_wiper_status":true,"abs":true,"timestamp":"2015-08-28T06:43:56.3979761Z"}  

8/27/2015 11:43:56 PM > Sending message: {"vin":"8XWRS39URQRAIZC8T","outsideTemperature":61,"engineTemperature":418,"speed":93,"fuel":19,"engineoil":48,"tirepressure":47,"odometer":148092,"city":"Bellevue","accelerator_pedal_position":58,"parking_brake_status":true,"brake_pedal_status":true,"headlamp_status":true,"transmission_gear_position":"seventh","ignition_status":true,"windshield_wiper_status":true,"abs":true,"timestamp":"2015-08-28T06:43:56.9206682Z"}  

8/27/2015 11:43:57 PM > Sending message: {"vin":"ZN6WJN7MM8HNU7Z0I","outsideTemperature":61,"engineTemperature":411,"speed":69,"fuel":34,"engineoil":30,"tirepressure":25,"odometer":175946,"city":"Seattle","accelerator_pedal_position":80,"parking_brake_status":false,"brake_pedal_status":false,"headlamp_status":false,"transmission_gear_position":"sixth","ignition_status":false,"windshield_wiper_status":false,"abs":false,"timestamp":"2015-08-28T06:43:57.5048278Z"}  

8/27/2015 11:43:58 PM > Sending message: {"vin":"0413HGH1MM8TKJYQGQ","outsideTemperature":72,"engineTemperature":136,"speed":58,"fuel":26,"engineoil":36,"tirepressure":29,"odometer":50559,"city":"Bellevue","accelerator_pedal_position":35,"parking_brake_status":false,"brake_pedal_status":false,"headlamp_status":false,"transmission_gear_position":"third","ignition_status":false,"windshield_wiper_status":false,"abs":false,"timestamp":"2015-08-28T06:43:58.1045707Z"}  

8/27/2015 11:43:58 PM > Sending message: {"vin":"3IG87CCLZ5BCOZPW7","outsideTemperature":81,"engineTemperature":217,"speed":71,"fuel":25,"engineoil":41,"tirepressure":35,"odometer":181962,"city":"Bellevue","accelerator_pedal_position":25,"parking_brake_status":false,"brake_pedal_status":false,"headlamp_status":false,"transmission_gear_position":fourth,"ignition_status":false,"windshield_wiper_status":false,"abs":false,"timestamp":"2015-08-28T06:43:58.3966445Z"}  

8/27/2015 11:43:58 PM > Sending message: {"vin":"U3WWHY029B6YRR7MG","outsideTemperature":81,"engineTemperature":215,"speed":71,"fuel":20,"engineoil":41,"tirepressure":35,"odometer":152388,"city":"Bellevue","accelerator_pedal_position":12,"parking_brake_status":false,"brake_pedal_status":false,"headlamp_status":false,"transmission_gear_position":fourth,"ignition_status":false,"windshield_wiper_status":false,"abs":false,"timestamp":"2015-08-28T06:43:58.704122Z"}  

8/27/2015 11:43:58 PM > Sending message: {"vin":"UNZZYMS6LN003WJJY","outsideTemperature":80,"engineTemperature":197,"speed":68,"fuel":19,"engineoil":40,"tirepressure":34,"odometer":88042,"city":"Bellevue","accelerator_pedal_position":83,"parking_brake_status":false,"brake_pedal_status":false,"headlamp_status":false,"transmission_gear_position":fourth,"ignition_status":false,"windshield_wiper_status":false,"abs":false,"timestamp":"2015-08-28T06:43:58.9655091Z"}  

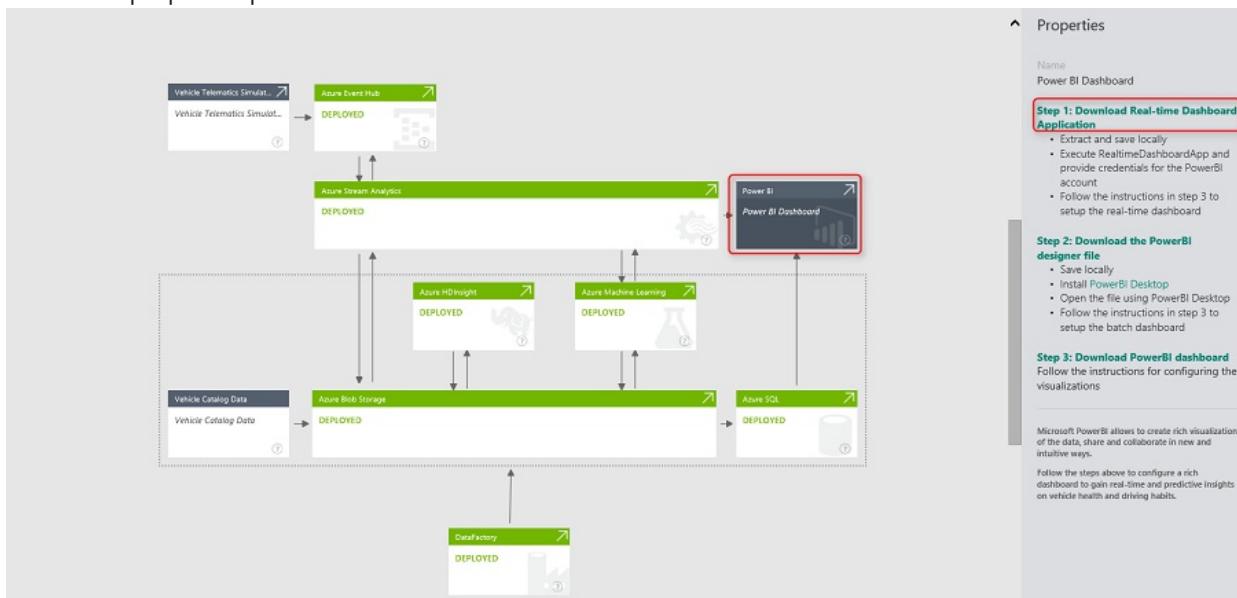

```

Start the real-time dashboard application

The solution includes an application that generates a real-time dashboard in Power BI. This application listens to an Event Hub instance, from which Stream Analytics publishes the events continuously. For every event that this application receives, it processes the data using a Machine Learning Request-Response scoring endpoint. The resultant dataset is published to the Power BI push APIs for visualization.

To download the application:

- Click the Power BI node on the diagram view and click the **Download Real-time Dashboard Application** link on the properties pane.



- Extract and save the application locally

Name	Date modified	Type	Size
ADF	7/21/2015 6:33 PM	File folder	
ASA	7/21/2015 6:33 PM	File folder	
bin	7/21/2015 6:33 PM	File folder	
CarEventGenerator	7/27/2015 11:26 PM	File folder	
RealTimeDashboardApp	7/21/2015 6:34 PM	File folder	
referencedata	7/21/2015 6:34 PM	File folder	
scripts	7/21/2015 6:34 PM	File folder	

3. Execute the application **RealtimeDashboardApp.exe**
4. Provide valid Power BI credentials, sign in and click **Accept**

Sign in X

Sign in

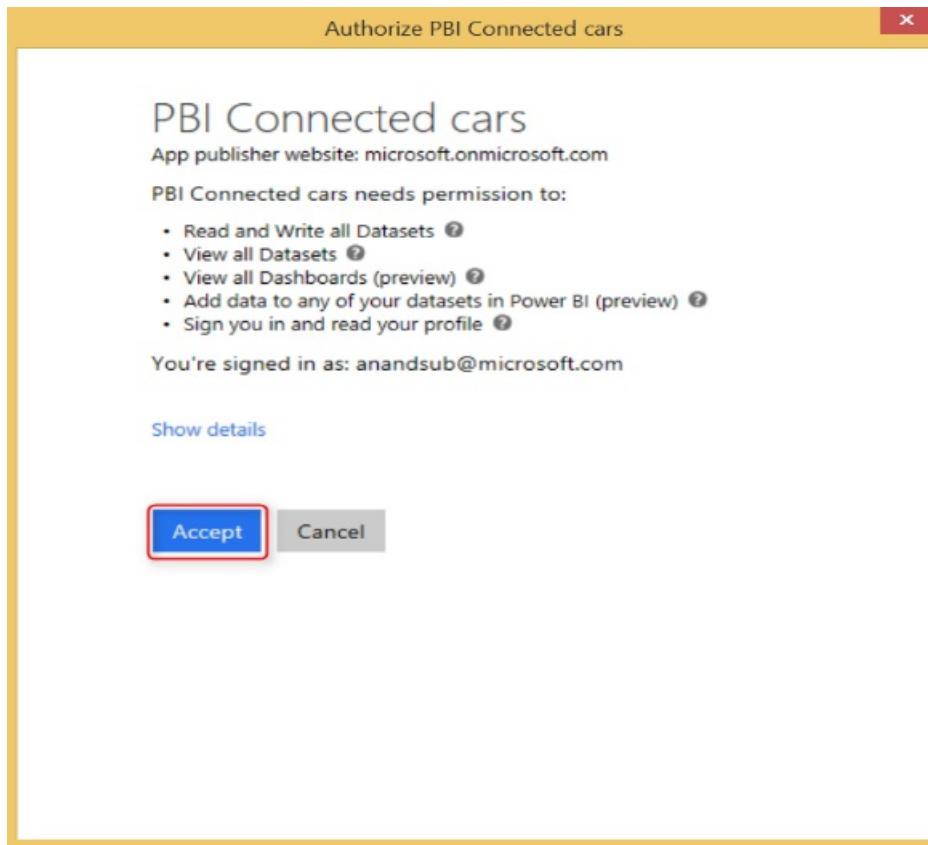
Sign in with your work or school account

Sign in Cancel

[Can't access your account?](#)

Don't have an account assigned by your work or school?
[Sign in with a Microsoft account](#)

 Your work or school account can be used anywhere you see this symbol.
 © 2015 Microsoft [Terms of use](#) [Privacy & Cookies](#)

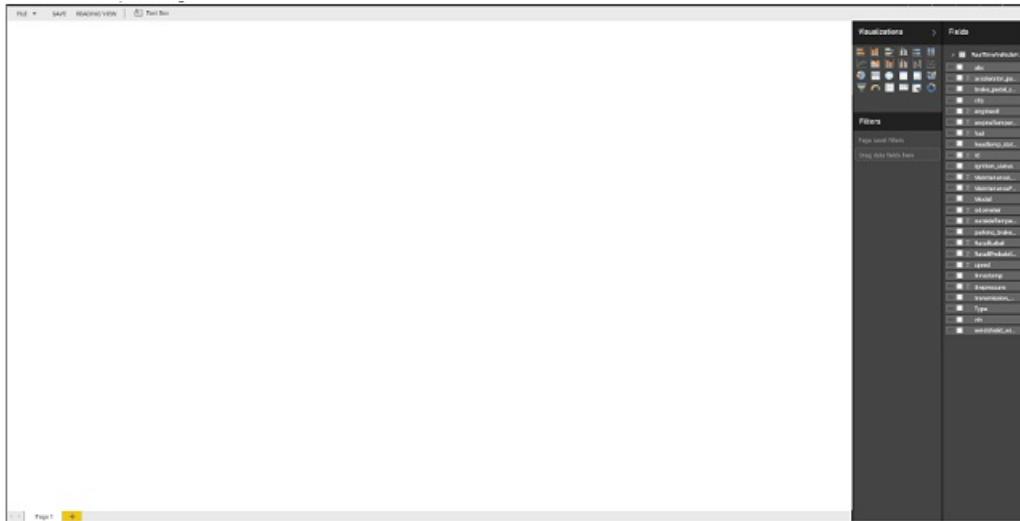


Configure Power BI reports

The real-time reports and the dashboard take about 30-45 minutes to complete. Browse to <http://powerbi.com> and login.

A new dataset is generated in Power BI. Click the **ConnectedCarsRealtime** dataset.

Save the blank report using **Ctrl + s**.



Provide report name *Vehicle Telemetry Analytics Real-time - Reports*.



Real-time reports

There are three real-time reports in this solution:

1. Vehicles in operation
2. Vehicles Requiring Maintenance
3. Vehicles Health Statistics

You can choose to configure all the three real-time reports or stop after any stage and proceed to the next section of configuring the batch reports. We recommend you to create all the three reports to visualize the full insights of the real-time path of the solution.

1. Vehicles in operation

Double-click **Page 1** and rename it to "Vehicles in operation"



Select **vin** field from **Fields** and choose visualization type as "**Card**".

Card visualization is created as shown in figure.

Power BI

FILE SAVE READING VIEW Text Box

Visualizations

Count of vin

Fields

- brake_pedal_s...
- city
- engineoil
- engineTemper...
- fuel
- headlamp_stat...
- Id
- ignition_status
- Maintenanc...
- MaintenanceP...
- Model
- odometer
- outsideTempe...
- parking_brake...
- RecallLabel
- RecallProbabil...
- speed
- timestamp
- tirepressure
- transmission_...
- Type
- vin**
- windshield_wi...

Filters

Count of vin (All)

Page Level Filters

Drag data fields here

Get Data

Vehicles in operation

Click the blank area to add new visualization.

Select **City** and **vin** from fields. Change visualization to “**Map**”. Drag **vin** in values area. Drag **city** from fields to **Legend** area.

Power BI

Connected Cars Real time - Reports*

FILE SAVE READING VIEW Text Box

Visualizations

Count of vin by city and city

Fields

- accelerator_p...
- brake_pedal_s...
- city**
- engineoil
- engineTemper...
- fuel
- headlamp_stat...
- Id
- ignition_status
- Maintenanc...
- MaintenanceP...
- Model
- odometer
- outsideTempe...
- parking_brake...
- RecallLabel
- RecallProbabil...
- speed
- timestamp
- tirepressure
- transmission_...
- Type
- vin**
- windshield_wi...

Location

city

Legend

city

Longitude

Latitude

Values

Count of vin

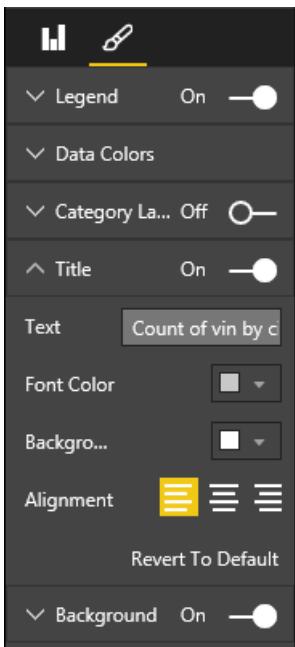
Color Saturation

Drag data fields here

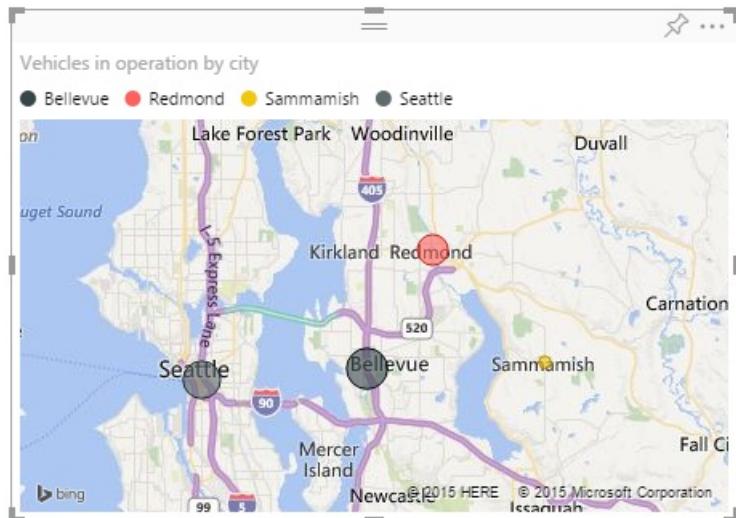
Get Data

Vehicles in operation

Select **format** section from **Visualizations**, click **Title** and change the **Text** to “**Vehicles in operation by city**”.



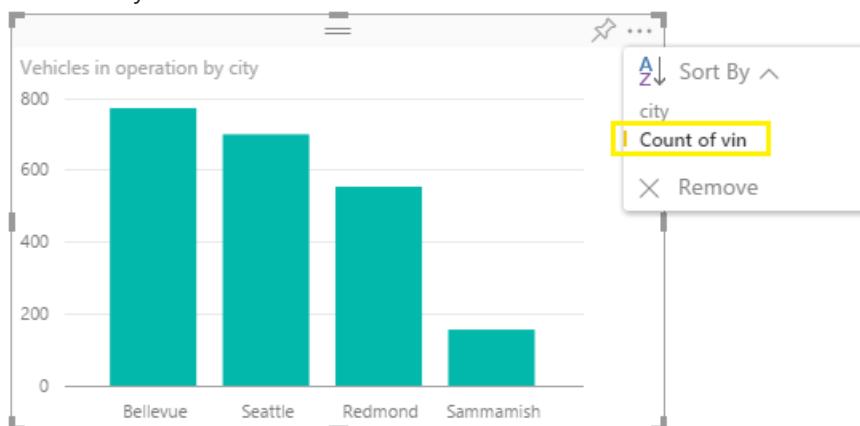
Final visualization looks as shown in figure.



Click the blank area to add new visualization.

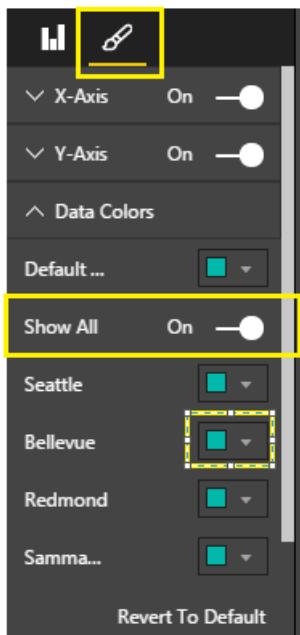
Select **City** and **vin**, change visualization type to **Clustered Column Chart**. Ensure **City** field in **Axis area** and **vin** in **Value area**

Sort chart by “**Count of vin**”

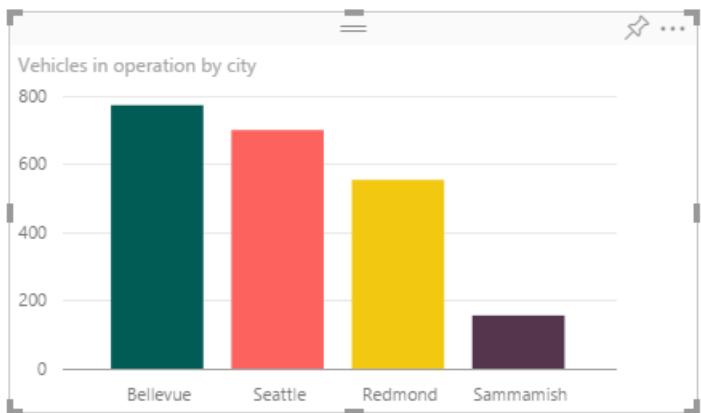


Change chart **Title** to “**Vehicles in operation by city**”

Click the **Format** section, then select **Data Colors**, Click the “**On**” to **Show All**



Change the color of individual city by clicking on color icon.



Click the blank area to add new visualization.

Select **Clustered Column Chart** visualization from visualizations, drag **city** field in **Axis** area, **Model** in **Legend** area and **vin** in **Value** area.

Visualizations > **Fields** >

Axis: city

Legend: Model

Value: Count of vin

Color Saturation: Drag data fields here

Filters: city (All)

Vehicles by city,model

Model: Compact ... Convertible Coupe Family Sal... Hybrid Large SUV Medium S... Sedan Small SUV

City	Compact ...	Convertible	Coupe	Family Sal...	Hybrid	Large SUV	Medium S...	Sedan	Small SUV
Seattle	55	60	55	210	200	140	55	210	60
Bellevue	60	75	55	75	200	100	65	190	60
Redmond	50	50	45	145	150	90	50	145	55
Sammamish	10	20	25	40	45	30	25	60	30

Rearrange all visualization on this page as shown in figure.

Power BI

Connected cars realtime reports*

FILE SAVE READING VIEW Text Box

Dashboards +

Connected cars

Connected Cars - Desktop

Connected cars backup

Connected Cars Dash...

Revenue vs Target

Sales Analysis

Wide World Importers

Reports

Connected car backup

Connected Car Report

Connected car reports

Connected Cars - Desktop

Connected Cars - Desktop

Connected Cars - Histo...

Connected Cars - Realti...

Get Data

Count of vin by city

3585

Count of vin

1K

OK

Seattle Bellevue Redmond Sammamish

Vehicles in operation by city

Bellevue Redmond Sammamish Seattle Kirkland Carnation

Vehicles by city,model

Model Compact Car Convertible Coupe Family Saloon Hybrid Large SUV Medium SUV Sedan Small SUV Sports Car Station Wagon

250

200

150

100

50

0

Seattle Bellevue Redmond Sammamish

Vehicles in operation

Visualizations Filters

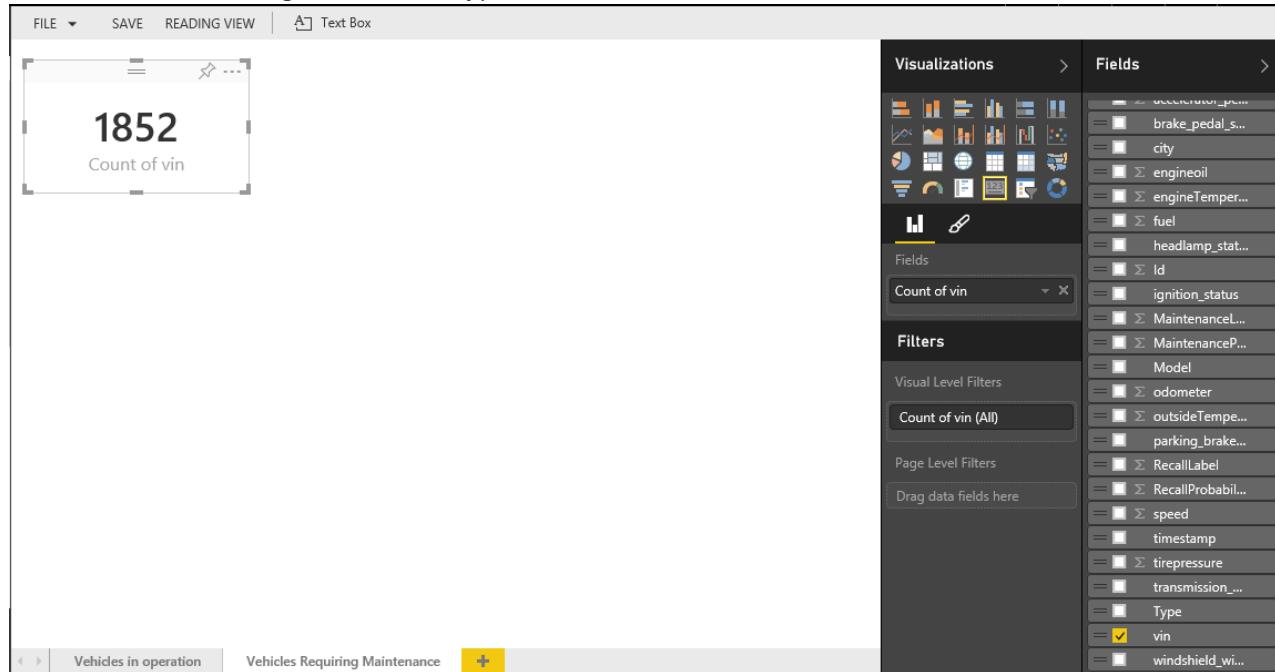
You have successfully configured the "Vehicles in operation" real-time report. You can proceed to create the next real-time report or stop here and configure the dashboard.

2. Vehicles Requiring Maintenance

Click  to add a new report, rename it to "**Vehicles Requiring Maintenance**"



Select **vin** field and change visualization type to **Card**.

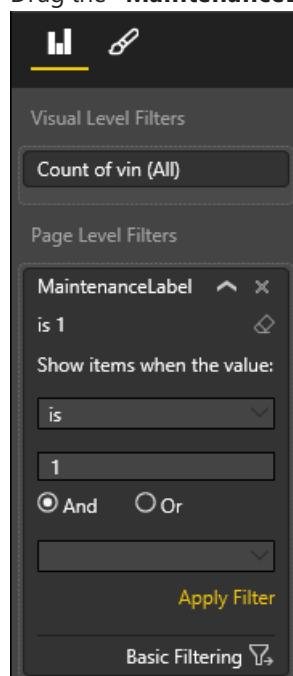


The screenshot shows the Power BI visual editor. On the left, there is a card visualization displaying the value "1852" with the subtitle "Count of vin". Above the card, there are buttons for FILE, SAVE, READING VIEW, and a Text Box. To the right of the card, there is a "Visualizations" pane with various chart icons and a "Fields" pane listing dataset fields such as "brake_pedal_s...", "city", "engineoil", "engineTemper...", "fuel", "headlamp_stat...", "Id", "ignition_status", "MaintenanceLabel", "MaintenanceP...", "Model", "odometer", "outsideTempe...", "parking_brake...", "RecallLabel", "RecallProbabil...", "speed", "timestamp", "tirepressure", "transmission ...", "Type", "vin", and "windshield_wi...". A yellow box highlights the "MaintenanceLabel" field in the Fields pane.

We have a field named "MaintenanceLabel" in the dataset. This field can have a value of "0" or "1". It is set by the Azure Machine Learning model provisioned as part of solution and integrated with the real-time path. The value "1" indicates a vehicle requires maintenance.

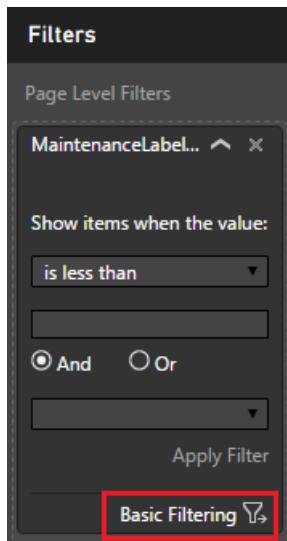
To add a **Page Level** filter for showing vehicles data, which are requiring maintenance:

1. Drag the "**MaintenanceLabel**" field into **Page Level Filters**.

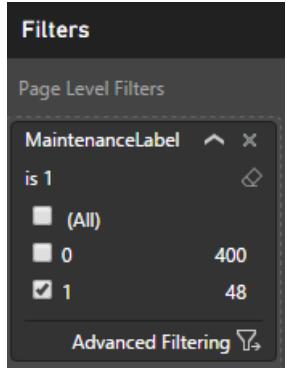


The screenshot shows the "MaintenanceLabel" Page Level Filter configuration dialog. It includes a "Visual Level Filters" section with a "Count of vin (All)" item. Below it is a "Page Level Filters" section for "MaintenanceLabel" with the condition "is 1". A dropdown menu "Show items when the value:" has "is" selected and "1" entered. There are "And" and "Or" radio buttons, and an "Apply Filter" button. At the bottom is a "Basic Filtering" menu.

2. Click **Basic Filtering** menu present at bottom of MaintenanceLabel Page Level Filter.



3. Set its filter value to "1"



Click the blank area to add new visualization.

Select **Clustered Column Chart** from visualizations

Drag field **Model** into **Axis** area, **Vin** to **Value** area. Then sort visualization by **Count of vin**. Change chart **Title** to "**Vehicles requiring maintenance by model**"

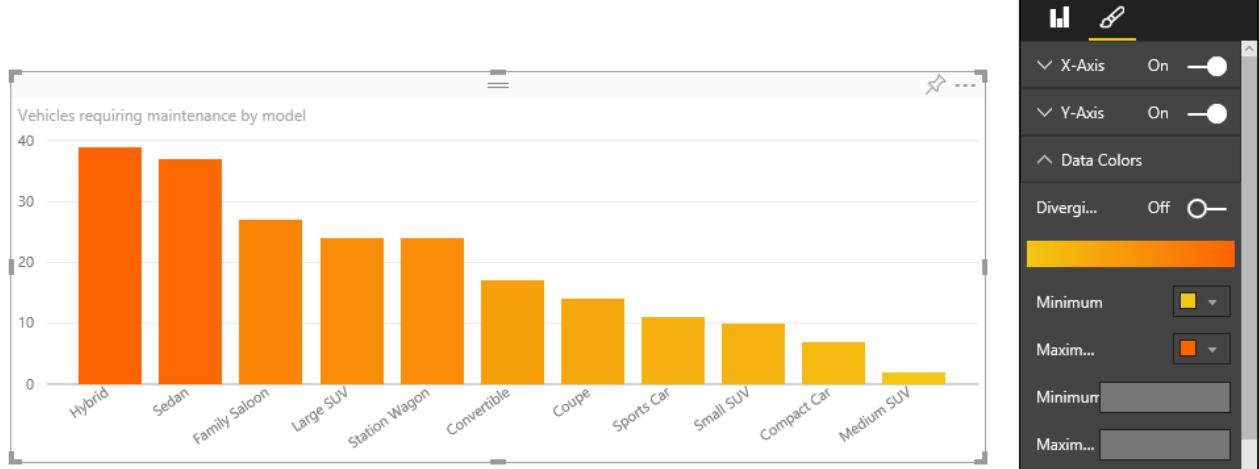
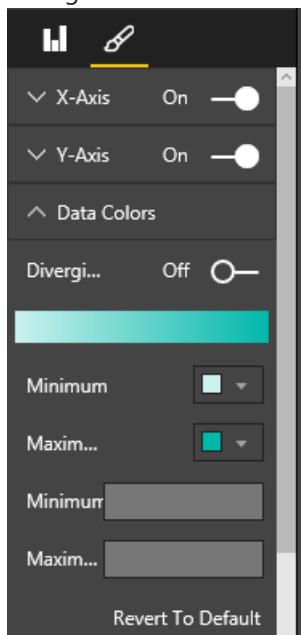
Drag vin fields into Color Saturation present at Fields  section of Visualization tab



Change Data Colors in visualizations from **Format** section

Change Minimum color to: **F2C812**

Change Maximum color to: **FF6300**



Click the blank area to add new visualization.

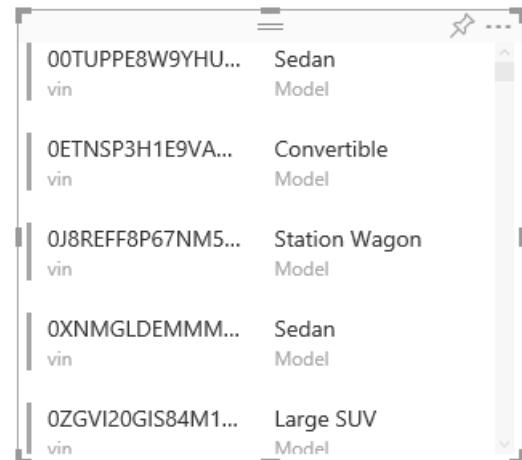
Select **Clustered column chart** from visualizations, drag **vin** field into **Value** area, drag **City** field into **Axis** area.

Sort chart by “**Count of vin**”. Change chart **Title** to “**Vehicles requiring maintenance by city**”



Click the blank area to add new visualization.

Select **Multi-Row Card** visualization from visualizations, drag **Model** and **vin** into the **Fields** area.



Rearranging all of the visualization, the final report looks as follows:

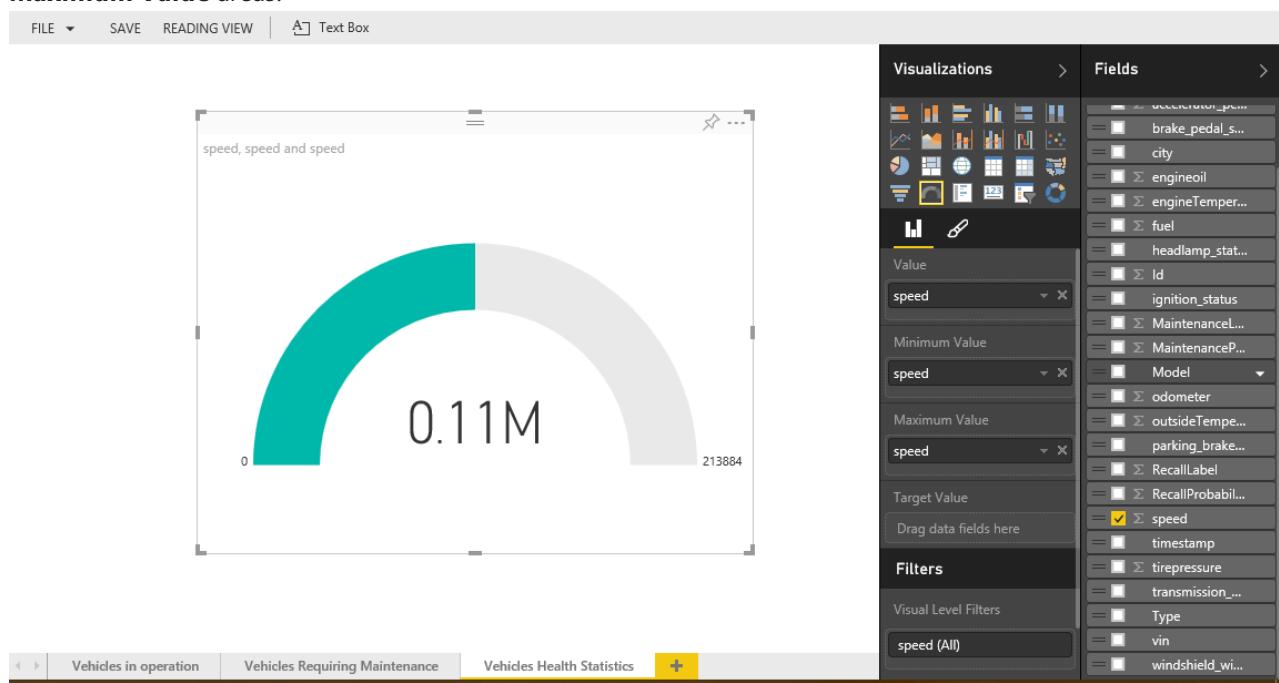
The dashboard also includes a navigation menu on the left and a Fields/Visualizations/Filters pane on the right.

You have successfully configured the “Vehicles Requiring Maintenance” real-time report. You can proceed to create the next real-time report or stop here and configure the dashboard.

3. Vehicles Health Statistics

Click to add new report, rename it to “**Vehicles Health Statistics**”

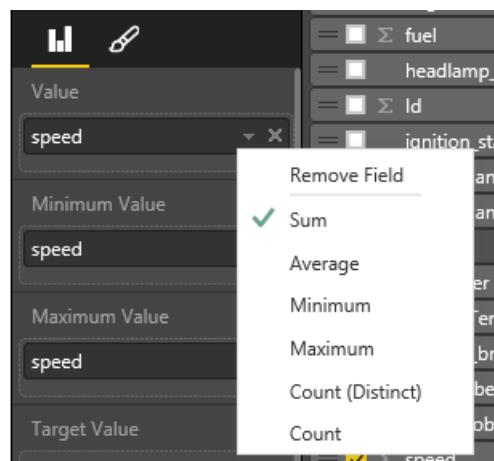
Select **Gauge** visualization from visualizations, then drag the **Speed** field into **Value**, **Minimum Value**, **Maximum Value** areas.



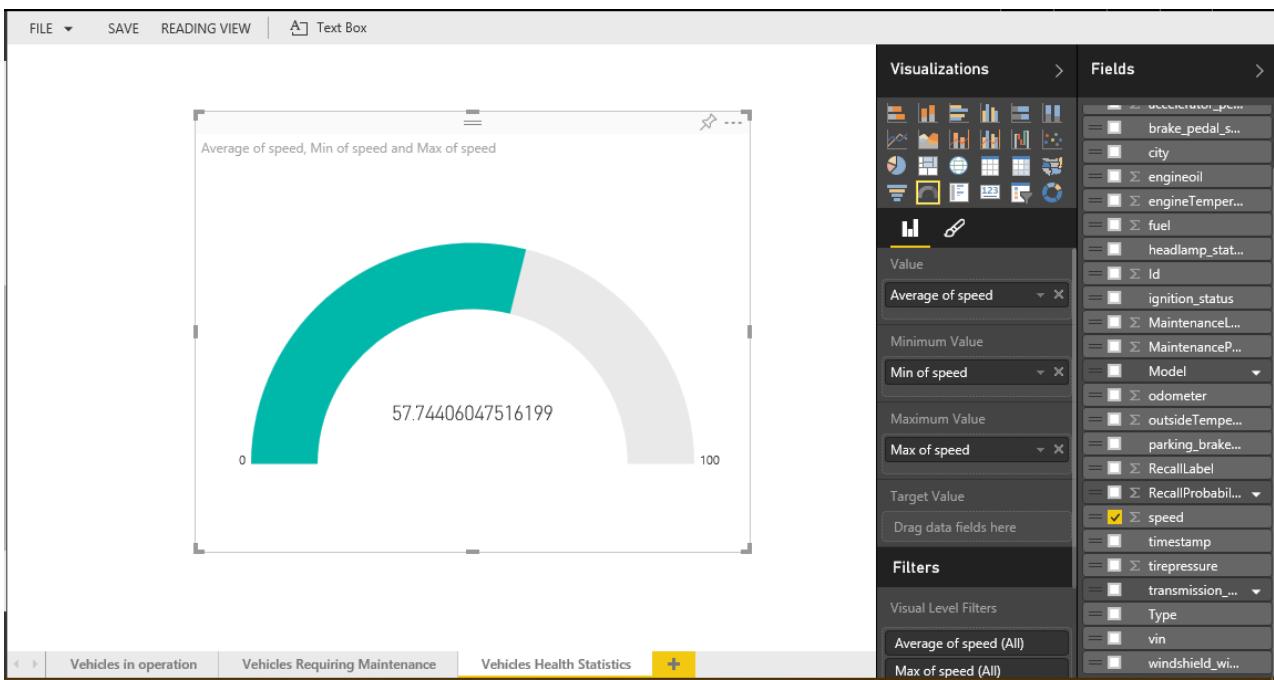
Change the default aggregation of **speed** in **Value area** to **Average**

Change the default aggregation of **speed** in **Minimum area** to **Minimum**

Change the default aggregation of **speed** in **Maximum area** to **Maximum**



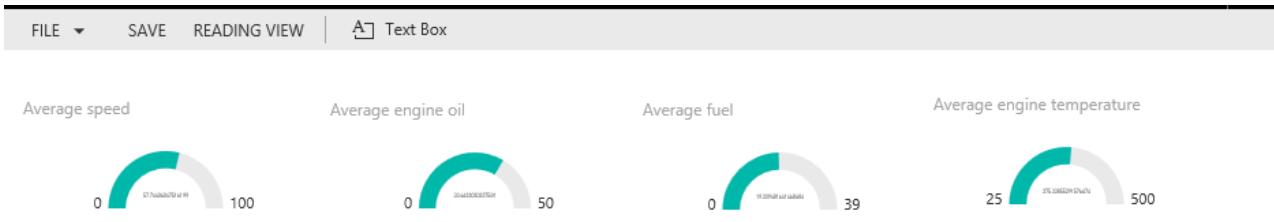
Rename the **Gauge Title** to “**Average speed**”



Click the blank area to add new visualization.

Similarly add a **Gauge** for **average engine oil**, **average fuel**, and **average engine temperate**.

Change the default aggregation of fields in each gauge as per above steps in “**Average speed**” gauge.



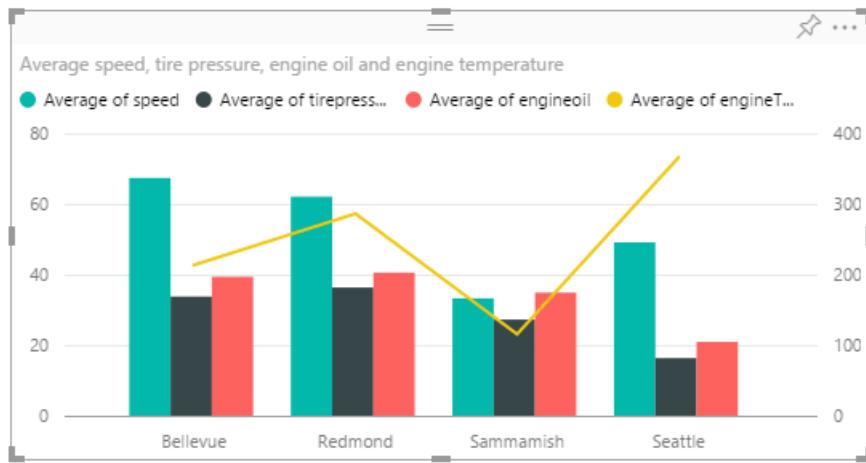
Click the blank area to add new visualization.

Select **Line and Clustered Column Chart** from visualizations, then drag **City** field into **Shared Axis**, drag **speed**, **tirepressure** and **engineoil** fields into **Column Values** area, change their aggregation type to **Average**.

Drag the **engineTemperature** field into **Line Values** area, change the aggregation type to **Average**.

The screenshot shows the Power BI Fields pane. On the left, there's a toolbar with icons for different visualization types. Below it, the 'Shared Axis' section has 'city' selected. Under 'Column Series', there are three dropdown menus: 'Average of tirepressure', 'Average of engineoil', and 'Average of speed'. Under 'Line Values', there's another dropdown menu: 'Average of engineTe...'. The 'Filters' section is collapsed. On the right, a long list of fields is shown, many of which have checkboxes next to them. The checked fields include 'city', 'engineoil', 'engineTemper...', 'speed', and 'tirepressure'.

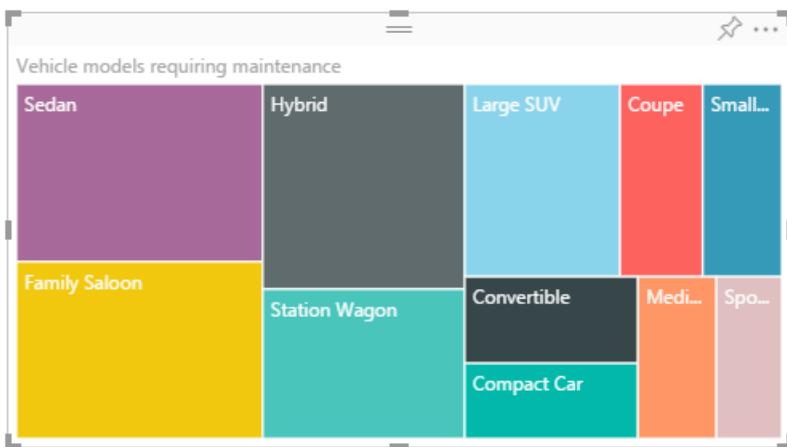
Change the chart **Title** to “**Average speed, tire pressure, engine oil and engine temperature**”.



Click the blank area to add new visualization.

Select **Treemap** visualization from visualizations, drag the **Model** field into the **Group** area, and drag the field **MaintenanceProbability** into the **Values** area.

Change the chart **Title** to “**Vehicle models requiring maintenance**”.



Click the blank area to add new visualization.

Select **100% Stacked Bar Chart** from visualization, drag the **city** field into the **Axis** area, and drag the **MaintenanceProbability**, **RecallProbability** fields into the **Value** area.

Visualizations

Fields

Axis: city

Legend: Drag data fields here

Value: MaintenanceProbability, RecallProbability

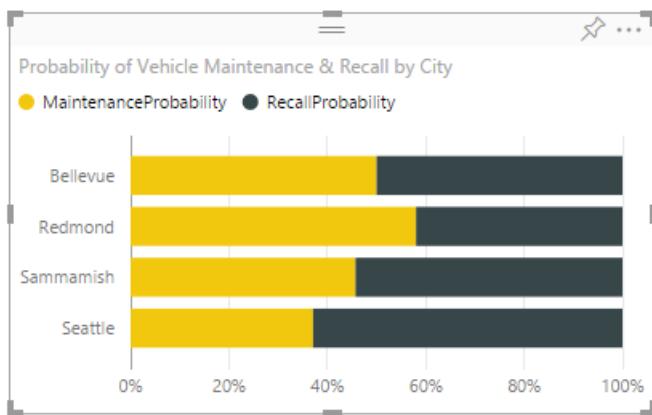
Filters: Visual Level Filters: city (All)

Fields (List):

- brake_pedal_s...
- city
- engineoil
- Σ engineTemper...
- fuel
- headlamp_stat...
- Σ Id
- ignition_status
- Σ Maintenance...
- Σ MaintenanceP...
- Model
- Σ odometer
- Σ outsideTempe...
- parking_brake...
- Σ RecallLabel
- Σ RecallProbabil...
- speed
- timestamp
- Σ tirepressure
- transmission_...
- Type
- vin
- windshield_wi...

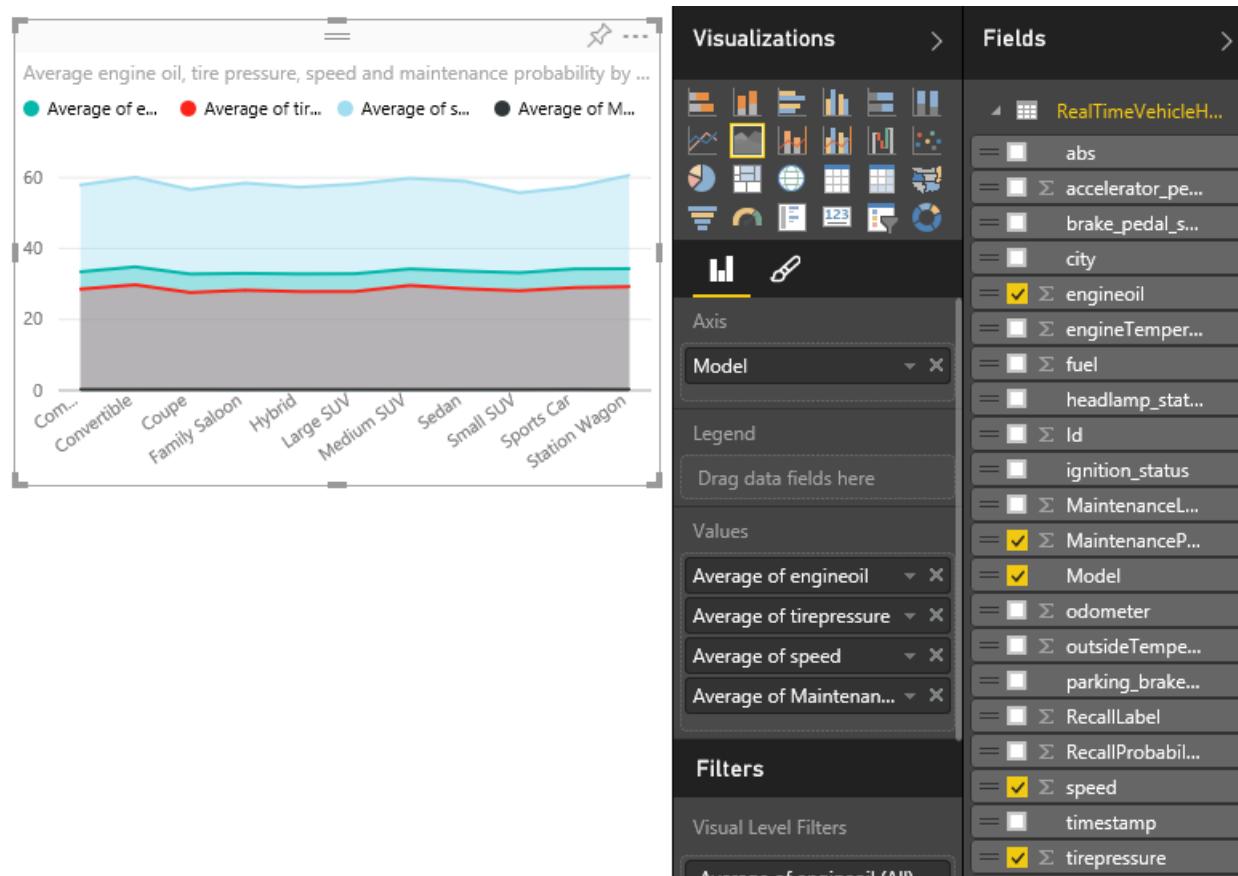
Click **Format**, select **Data Colors**, and set the **MaintenanceProbability** color to the value “**F2C80F**”.

Change the **Title** of the chart to “**Probability of Vehicle Maintenance & Recall by City**”.

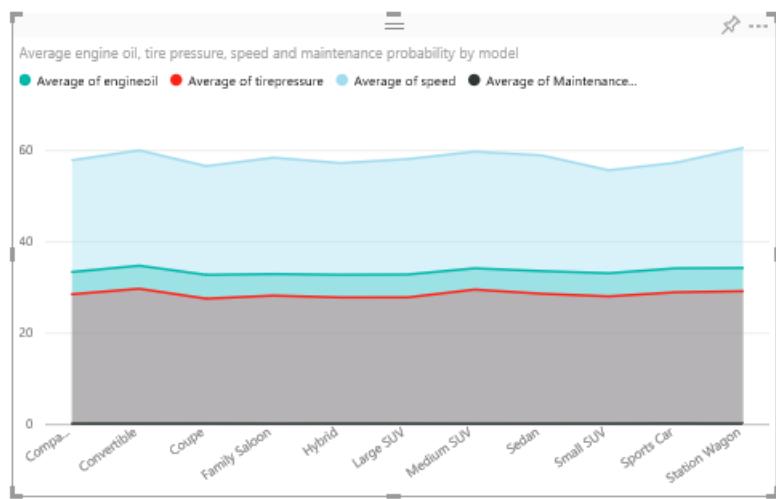


Click the blank area to add new visualization.

Select **Area Chart** from visualization from visualizations, drag the **Model** field into the **Axis** area, and drag the **engineOil, tirepressure, speed and MaintenanceProbability** fields into the **Values** area. Change their aggregation type to “**Average**”.



Change the title of the chart to “**Average engine oil, tire pressure, speed and maintenance probability by model**”.



Click the blank area to add new visualization:

1. Select **Scatter Chart** visualization from visualizations.
2. Drag the **Model** field into the **Details and Legend** area.
3. Drag the **fuel** field into the **X-Axis** area, change the aggregation to **Average**.
4. Drag **engineTemparature** into **Y-Axis area**, change the aggregation to **Average**
5. Drag the **vin** field into the **Size** area.

Visualizations > Fields >

Details: Model

Legend: Model

X Axis: Average of fuel

Y Axis: Average of engineTemp...

Size: Count of vin

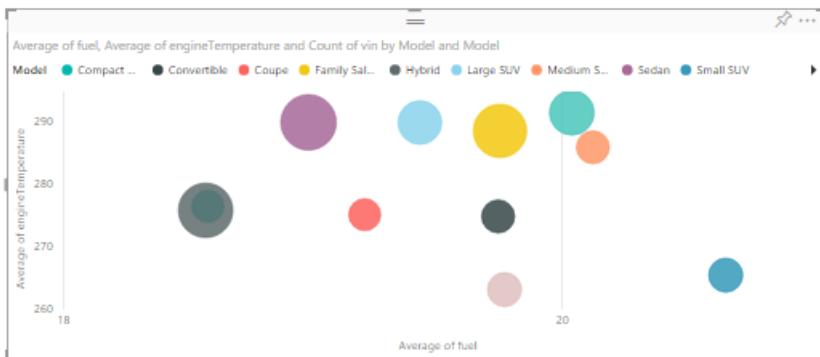
Fields pane:

- RealTimeVehicleH...
- abs
- accelerator_p...
- brake_pedal_s...
- city
- engineoil
- engineTemp...
- fuel
- headlamp_sta...
- Id
- ignition_status
- Maintenance...
- MaintenanceP...
- Model
- odometer
- outsideTempe...
- parking_brake...
- RecallLabel
- RecallProbabil...
- speed
- timestamp
- tirepressure
- transmission_...
- Type
- vin
- windshield_wi...

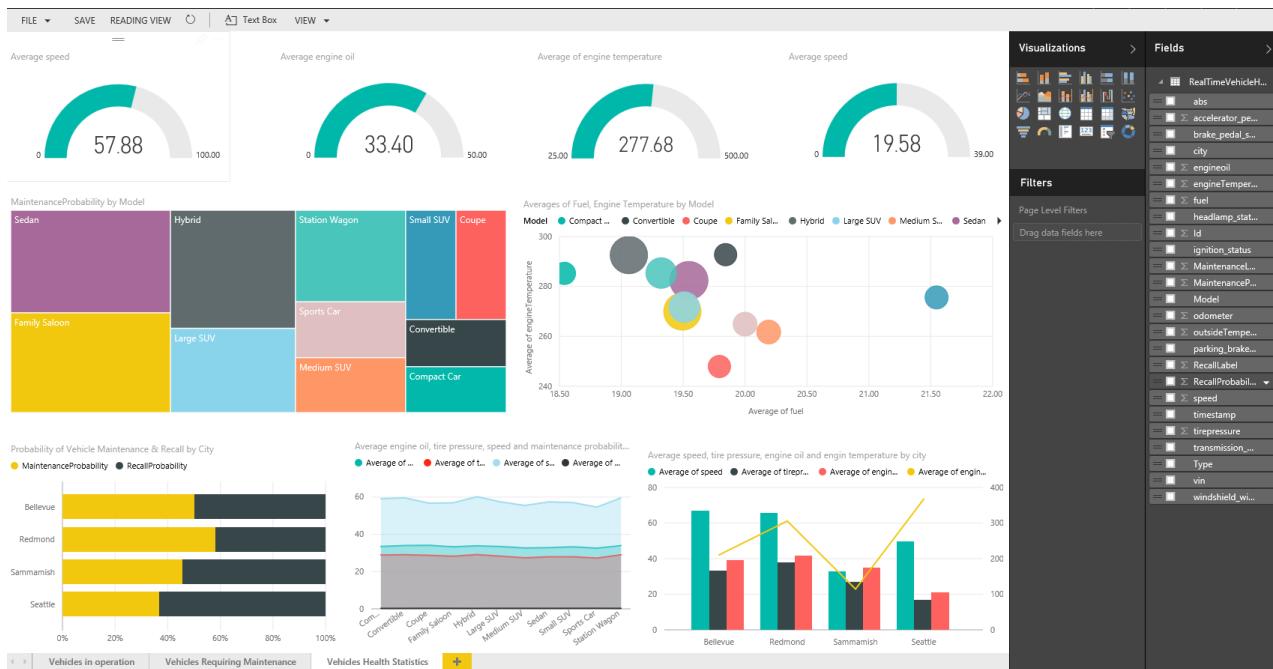
Visual Level Filters: Average of engineTemp..., Average of fuel (All), Count of vin (All), Model (All)

Page Level Filters: Drag data fields here

Change the chart **Title** to “**Averages of Fuel, Engine Temperature by Model**”.

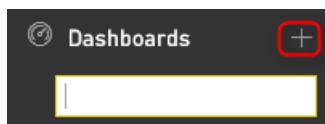


The final report will look like as shown below.



Pin visualizations from the reports to the real-time dashboard

Create a blank dashboard by clicking on the plus icon next to Dashboards. You can name it "Vehicle Telemetry Analytics Dashboard"



Pin the visualization from the above reports to the dashboard.



The dashboard should look as follows when all the three reports are created and the corresponding visualizations are pinned to the dashboard. If you have not created all the reports, your dashboard could look different.



Congratulations! You have successfully created the real-time dashboard. As you continue to execute CarEventGenerator.exe and RealtimeDashboardApp.exe, you should see live updates on the dashboard. It should take about 10 to 15 minutes to complete the following steps.

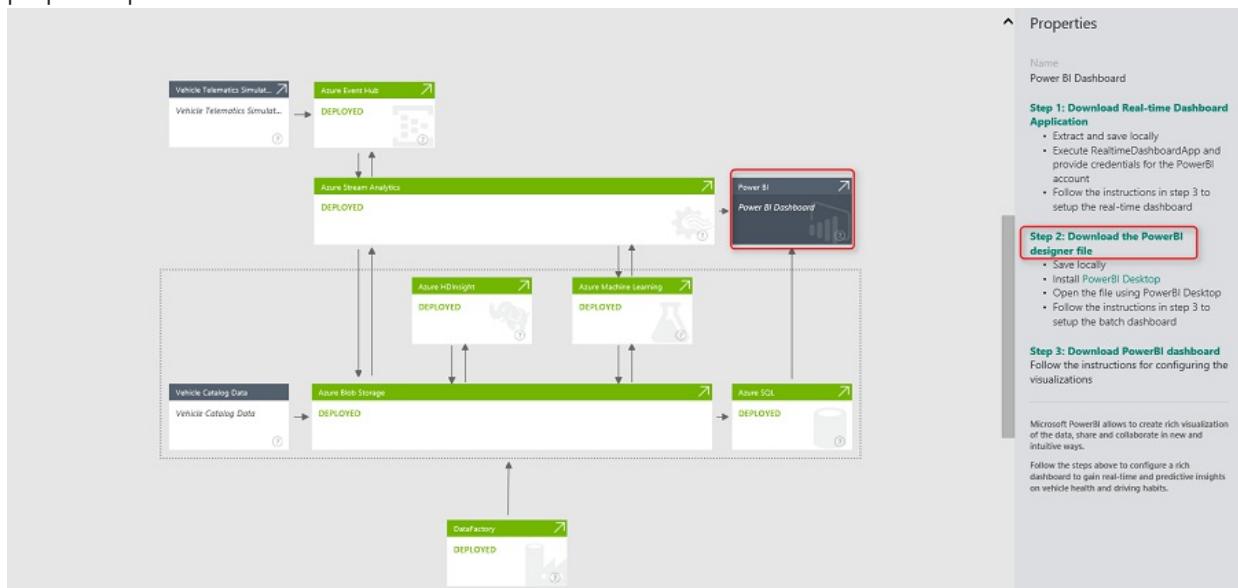
Setup Power BI batch processing dashboard

NOTE

It takes about two hours (from the successful completion of the deployment) for the end to end batch processing pipeline to finish execution and process a year worth of generated data. So wait for the processing to finish before proceeding with the next steps.

Download the Power BI designer file

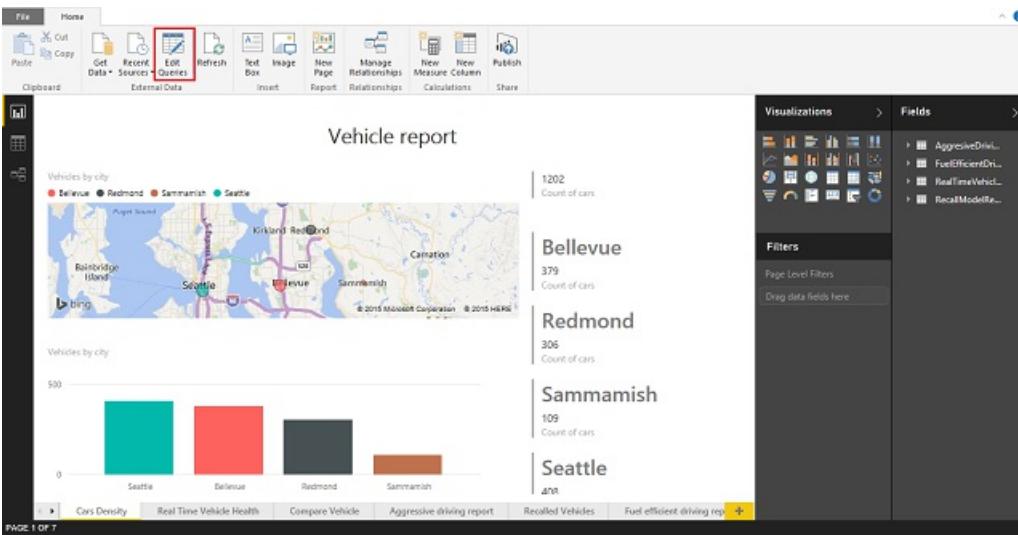
- A pre-configured Power BI designer file is included as part of the deployment
- Click the Power BI node on the diagram view and click **Download the Power BI designer file** link on the properties pane



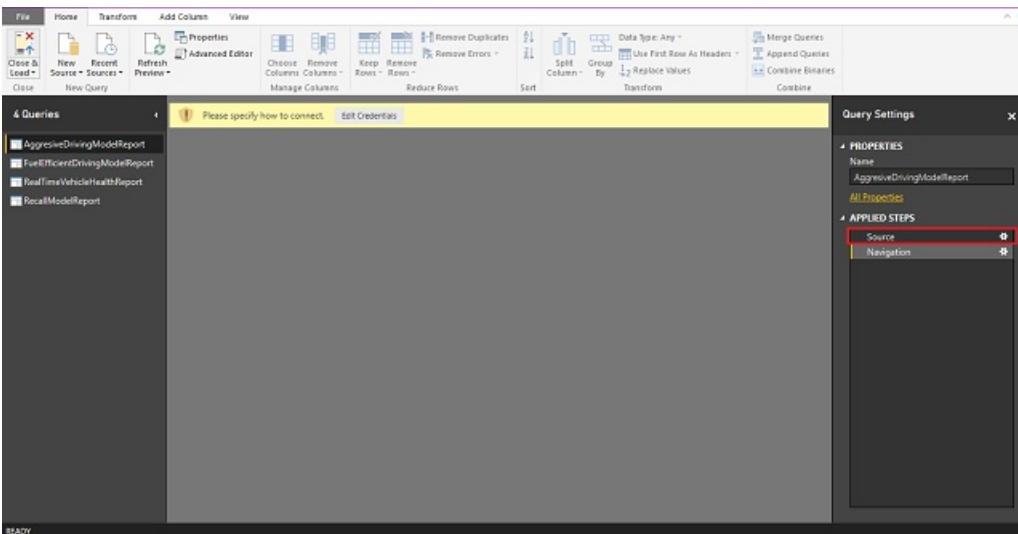
- Save locally

Configure Power BI reports

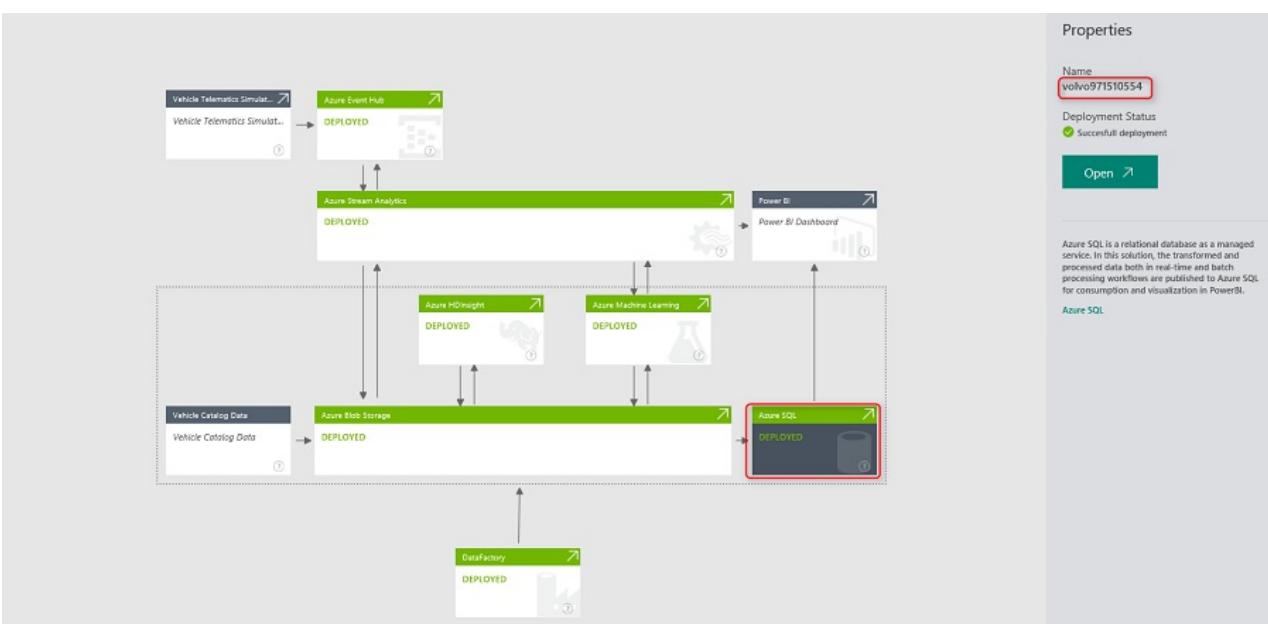
- Open the designer file 'VehicleTelemetryAnalytics - Desktop Report.pbix' using Power BI Desktop. If you do not already have, install the Power BI Desktop from [Power BI Desktop install](#).
- Click the **Edit Queries**.



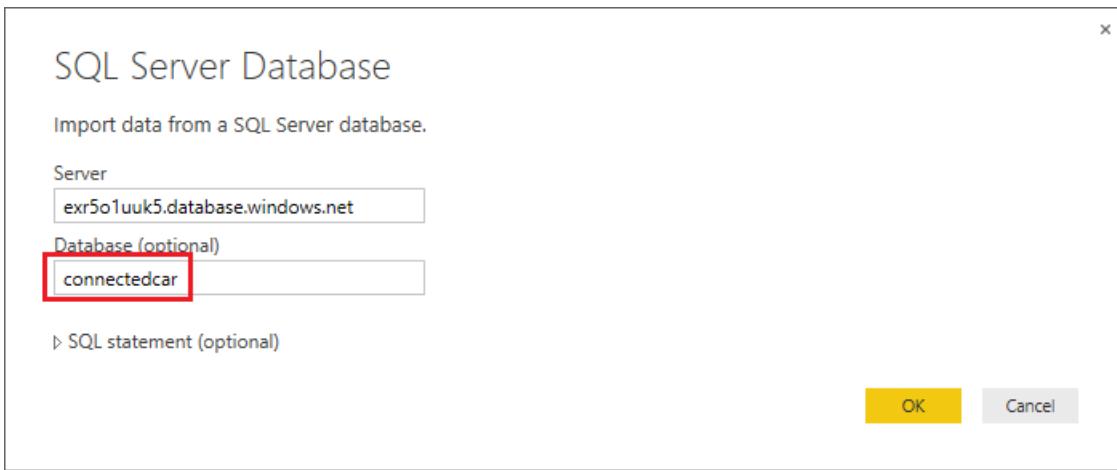
- Double-click the **Source**.



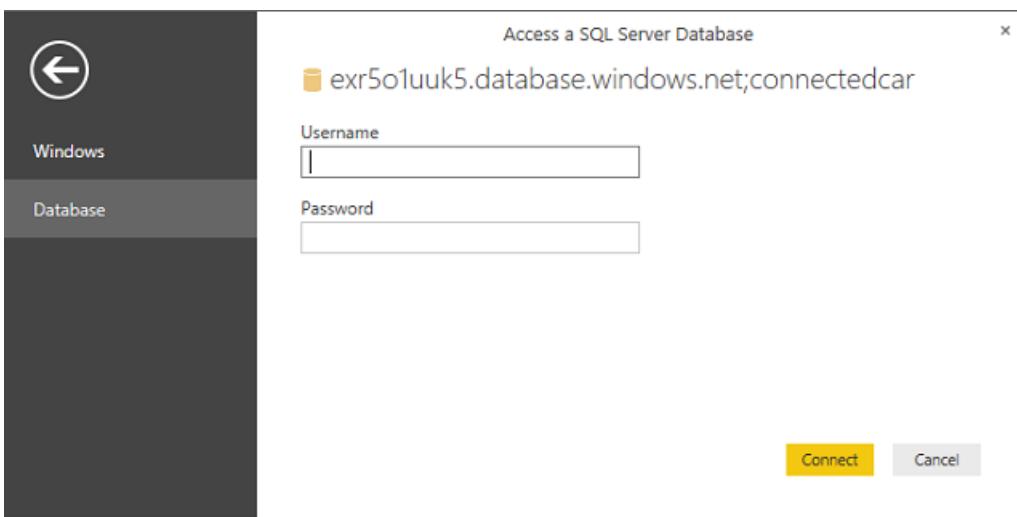
- Update Server connection string with the Azure SQL server that got provisioned as part of the deployment. Click the Azure SQL node on the diagram and view the server name of the properties pane.



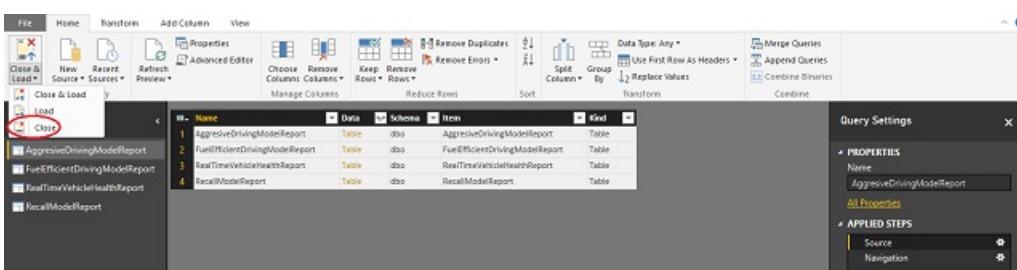
- Leave **Database** as *connectedcar*.



- Click **OK**.
- You will see **Windows credential** tab selected by default, change it to **Database credentials** by clicking on **Database** tab at right.
- Provide the **Username** and **Password** of your Azure SQL Database that was specified during its deployment setup.



- Click **Connect**
- Repeat the above steps for each of the three remaining queries present at right pane, and then update the data source connection details.
- Click **Close and Load**. Power BI Desktop file datasets are connected to SQL Azure Database tables.
- **Close** Power BI Desktop file.



- Click **Save** button to save the changes.

You have now configured all the reports corresponding to the batch processing path in the solution.

Upload to *powerbi.com*

1. Navigate to the Power BI web portal at <http://powerbi.com> and login.

2. Click **Get Data**
3. Upload the Power BI Desktop File.
4. To upload, click **Get Data -> Files Get -> Local file**
5. Navigate to the “**VehicleTelemetryAnalytics – Desktop Report.pbix**”
6. Once the file is uploaded, you will be navigated back to your Power BI work space.

A dataset, report and a blank dashboard will be created for you.

Pin charts to the existing dashboard **Vehicle Telemetry Analytics Dashboard** in **Power BI**. Click the blank dashboard created above and then navigate to the **Reports** section click the newly uploaded report.

The screenshot shows the Power BI interface with the 'Connected Cars Dashboard' selected. A report titled 'Connected Cars - Desktop Report...' is pinned to the dashboard, indicated by a yellow box around its preview icon. The left sidebar lists various datasets and reports under categories like 'Connected Cars Dashboards', 'Reports', and 'Datasets'. A yellow box highlights the 'Get Data' button at the bottom left of the interface.

Note the report has six pages:

- Page 1: Vehicle density
- Page 2: Real-time vehicle health
- Page 3: Aggressively Driven Vehicles
- Page 4: Recalled vehicles
- Page 5: Fuel Efficiently Driven Vehicles
- Page 6: Contoso Logo

The screenshot shows the 'Connected Cars - Desktop Report' with six pages of data visualizations.
 - Page 1: 'Vehicles in operation' bar chart showing vehicle counts by city: Seattle (408), Bellevue (379), Redmond (306), and Sammamish (109).
 - Page 2: 'Vehicles by city' map of Seattle and surrounding areas with callouts for Bellevue, Redmond, Sammamish, and Seattle.
 - Page 3: 'Vehicle models by city' grouped bar chart for Seattle, Bellevue, Redmond, and Sammamish, showing counts for various vehicle types like Compact Car, Convertible, Coupe, Family Saloon, Hybrid, Large SUV, Medium SUV, Sedan, Small SUV, Sports Car, and Station Wagon.
 - Page 4: Not visible in the screenshot.
 - Page 5: Not visible in the screenshot.
 - Page 6: Not visible in the screenshot.
 The left sidebar lists datasets and reports, and a yellow box highlights the 'Get Data' button at the bottom left.

From Page 3, pin the following:

1. Count of VIN



2. Aggressively driven vehicles by model – Waterfall chart

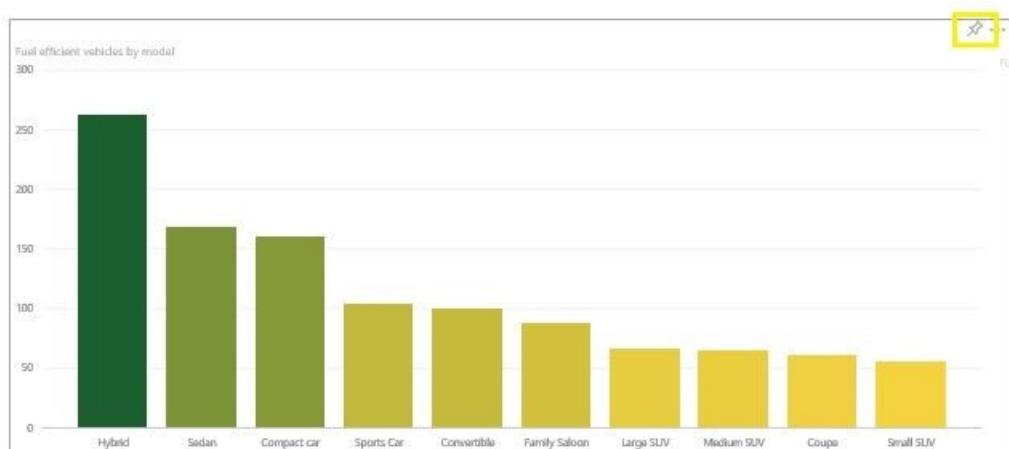


From Page 5, pin the following:

1. Count of vin

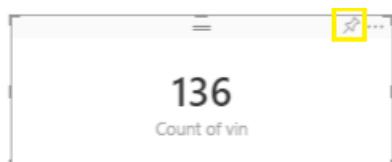


2. Fuel efficient vehicles by model: Clustered column chart



From Page 4, pin the following:

1. Count of vin



2. Recalled vehicles by city, model: Treemap



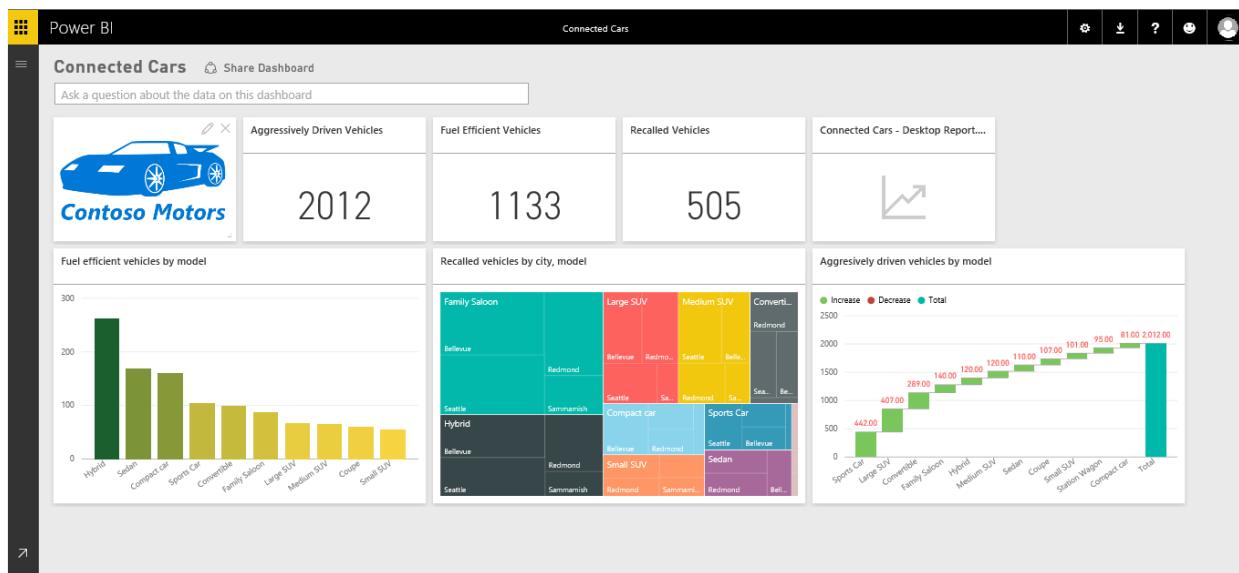
From Page 6, pin the following:

1. Contoso Motors logo

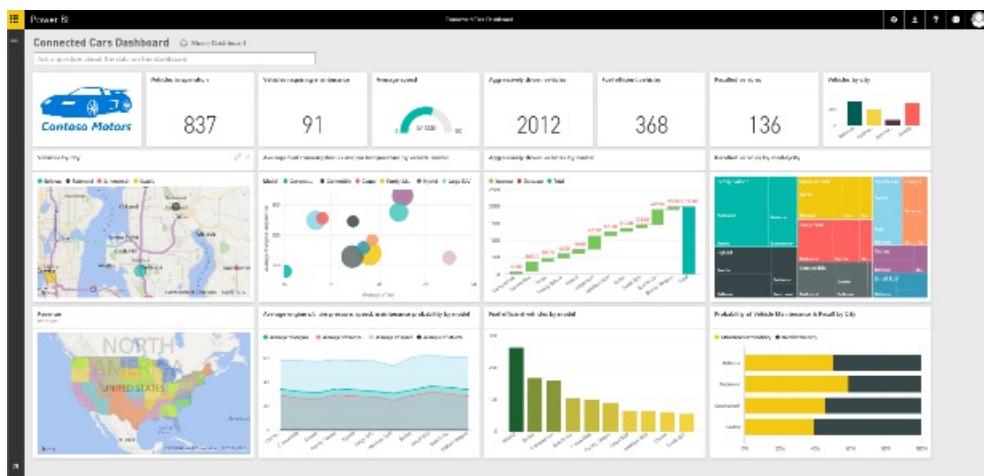


Organize the dashboard

1. Navigate to the dashboard
2. Hover over each chart and rename it based on the naming provided in the complete dashboard image below.
Also move the charts around to look like the dashboard below.



- If you have created all the reports as mentioned in this document, the final completed dashboard should look like the following figure.



Congratulations! You have successfully created the reports and the dashboard to gain real-time, predictive and batch insights on vehicle health and driving habits.

Guide to Net# neural network specification language for Azure Machine Learning

1/17/2017 • 23 min to read • [Edit on GitHub](#)

Overview

Net# is a language developed by Microsoft that is used to define neural network architectures. You can use Net# in neural network modules in Microsoft Azure Machine Learning, or in the `rxNeuralNetwork()` function in [MicrosoftML](#).

In this article, you will learn basic concepts needed to develop a custom neural network:

- Neural network requirements and how to define the primary components
- The syntax and keywords of the Net# specification language
- Examples of custom neural networks created using Net#

NOTE

[Try Azure Machine Learning for free](#)

No credit card or Azure subscription needed. [Get started now >](#)

Neural network basics

A neural network structure consists of **nodes** that are organized in **layers**, and weighted **connections** (or **edges**) between the nodes. The connections are directional, and each connection has a **source** node and a **destination** node.

Each **trainable layer** (a hidden or an output layer) has one or more **connection bundles**. A connection bundle consists of a source layer and a specification of the connections from that source layer. All the connections in a given bundle share the same **source layer** and the same **destination layer**. In Net#, a connection bundle is considered as belonging to the bundle's destination layer.

Net# supports various kinds of connection bundles, which lets you customize the way inputs are mapped to hidden layers and mapped to the outputs.

The default or standard bundle is a **full bundle**, in which each node in the source layer is connected to every node in the destination layer.

Additionally, Net# supports the following four kinds of advanced connection bundles:

- **Filtered bundles.** The user can define a predicate by using the locations of the source layer node and the destination layer node. Nodes are connected whenever the predicate is True.
- **Convolutional bundles.** The user can define small neighborhoods of nodes in the source layer. Each node in the destination layer is connected to one neighborhood of nodes in the source layer.
- **Pooling bundles and Response normalization bundles.** These are similar to convolutional bundles in that the user defines small neighborhoods of nodes in the source layer. The difference is that the weights of the edges in these bundles are not trainable. Instead, a predefined function is applied to the source node values to determine the destination node value.

Using Net# to define the structure of a neural network makes it possible to define complex structures such as deep neural networks or convolutions of arbitrary dimensions, which are known to improve learning on data such as

image, audio, or video.

Supported customizations

The architecture of neural network models that you create in Azure Machine Learning can be extensively customized by using Net#. You can:

- Create hidden layers and control the number of nodes in each layer.
- Specify how layers are to be connected to each other.
- Define special connectivity structures, such as convolutions and weight sharing bundles.
- Specify different activation functions.

For details of the specification language syntax, see [Structure Specification](#).

For examples of defining neural networks for some common machine learning tasks, from simplex to complex, see [Examples](#).

General requirements

- There must be exactly one output layer, at least one input layer, and zero or more hidden layers.
- Each layer has a fixed number of nodes, conceptually arranged in a rectangular array of arbitrary dimensions.
- Input layers have no associated trained parameters and represent the point where instance data enters the network.
- Trainable layers (the hidden and output layers) have associated trained parameters, known as weights and biases.
- The source and destination nodes must be in separate layers.
- Connections must be acyclic; in other words, there cannot be a chain of connections leading back to the initial source node.
- The output layer cannot be a source layer of a connection bundle.

Structure specifications

A neural network structure specification is composed of three sections: the **constant declaration**, the **layer declaration**, the **connection declaration**. There is also an optional **share declaration** section. The sections can be specified in any order.

Constant declaration

A constant declaration is optional. It provides a means to define values used elsewhere in the neural network definition. The declaration statement consists of an identifier followed by an equal sign and a value expression.

For example, the following statement defines a constant **x**:

```
Const X=28;
```

To define two or more constants simultaneously, enclose the identifier names and values in braces, and separate them by using semicolons. For example:

```
Const { X=28; Y=4; }
```

The right-hand side of each assignment expression can be an integer, a real number, a Boolean value (True or False), or a mathematical expression. For example:

```
Const { X=17 * 2; Y=true; }
```

Layer declaration

The layer declaration is required. It defines the size and source of the layer, including its connection bundles and attributes. The declaration statement starts with the name of the layer (input, hidden, or output), followed by the dimensions of the layer (a tuple of positive integers). For example:

```
input Data auto;  
hidden Hidden[5,20] from Data all;  
output Result[2] from Hidden all;
```

- The product of the dimensions is the number of nodes in the layer. In this example, there are two dimensions [5,20], which means there are 100 nodes in the layer.
- The layers can be declared in any order, with one exception: If more than one input layer is defined, the order in which they are declared must match the order of features in the input data.

To specify that the number of nodes in a layer be determined automatically, use the **auto** keyword. The **auto** keyword has different effects, depending on the layer:

- In an input layer declaration, the number of nodes is the number of features in the input data.
- In a hidden layer declaration, the number of nodes is the number that is specified by the parameter value for **Number of hidden nodes**.
- In an output layer declaration, the number of nodes is 2 for two-class classification, 1 for regression, and equal to the number of output nodes for multiclass classification.

For example, the following network definition allows the size of all layers to be automatically determined:

```
input Data auto;  
hidden Hidden auto from Data all;  
output Result auto from Hidden all;
```

A layer declaration for a trainable layer (the hidden or output layers) can optionally include the output function (also called an activation function), which defaults to **sigmoid** for classification models, and **linear** for regression models. (Even if you use the default, you can explicitly state the activation function, if desired for clarity.)

The following output functions are supported:

- sigmoid
- linear
- softmax
- rlinear
- square
- sqrt
- srlinear
- abs
- tanh
- brlinear

For example, the following declaration uses the **softmax** function:

```
output Result [100] softmax from Hidden all;
```

Connection declaration

Immediately after defining the trainable layer, you must declare connections among the layers you have defined. The connection bundle declaration starts with the keyword **from**, followed by the name of the bundle's source layer and the kind of connection bundle to create.

Currently, five kinds of connection bundles are supported:

- **Full** bundles, indicated by the keyword **all**
- **Filtered** bundles, indicated by the keyword **where**, followed by a predicate expression
- **Convolutional** bundles, indicated by the keyword **convolve**, followed by the convolution attributes
- **Pooling** bundles, indicated by the keywords **max pool** or **mean pool**
- **Response normalization** bundles, indicated by the keyword **response norm**

Full bundles

A full connection bundle includes a connection from each node in the source layer to each node in the destination layer. This is the default network connection type.

Filtered bundles

A filtered connection bundle specification includes a predicate, expressed syntactically, much like a C# lambda expression. The following example defines two filtered bundles:

```
input Pixels [10, 20];
hidden ByRow[10, 12] from Pixels where (s,d) => s[0] == d[0];
hidden ByCol[5, 20] from Pixels where (s,d) => abs(s[1] - d[1]) <= 1;
```

- In the predicate for *ByRow*, **s** is a parameter representing an index into the rectangular array of nodes of the input layer, *Pixels*, and **d** is a parameter representing an index into the array of nodes of the hidden layer, *ByRow*. The type of both **s** and **d** is a tuple of integers of length two. Conceptually, **s** ranges over all pairs of integers with $0 \leq s[0] < 10$ and $0 \leq s[1] < 20$, and **d** ranges over all pairs of integers, with $0 \leq d[0] < 10$ and $0 \leq d[1] < 12$.
- On the right-hand side of the predicate expression, there is a condition. In this example, for every value of **s** and **d** such that the condition is True, there is an edge from the source layer node to the destination layer node. Thus, this filter expression indicates that the bundle includes a connection from the node defined by **s** to the node defined by **d** in all cases where **s[0]** is equal to **d[0]**.

Optionally, you can specify a set of weights for a filtered bundle. The value for the **Weights** attribute must be a tuple of floating point values with a length that matches the number of connections defined by the bundle. By default, weights are randomly generated.

Weight values are grouped by the destination node index. That is, if the first destination node is connected to K source nodes, the first K elements of the **Weights** tuple are the weights for the first destination node, in source index order. The same applies for the remaining destination nodes.

It's possible to specify weights directly as constant values. For example, if you learned the weights previously, you can specify them as constants using this syntax:

```
const Weights_1 = [0.0188045055, 0.130500451, ...]
```

Convolutional bundles

When the training data has a homogeneous structure, convolutional connections are commonly used to learn high-level features of the data. For example, in image, audio, or video data, spatial or temporal dimensionality can be

fairly uniform.

Convolutional bundles employ rectangular **kernels** that are slid through the dimensions. Essentially, each kernel defines a set of weights applied in local neighborhoods, referred to as **kernel applications**. Each kernel application corresponds to a node in the source layer, which is referred to as the **central node**. The weights of a kernel are shared among many connections. In a convolutional bundle, each kernel is rectangular and all kernel applications are the same size.

Convolutional bundles support the following attributes:

InputShape defines the dimensionality of the source layer for the purposes of this convolutional bundle. The value must be a tuple of positive integers. The product of the integers must equal the number of nodes in the source layer, but otherwise, it does not need to match the dimensionality declared for the source layer. The length of this tuple becomes the **arity** value for the convolutional bundle. (Typically arity refers to the number of arguments or operands that a function can take.)

To define the shape and locations of the kernels, use the attributes **KernelShape**, **Stride**, **Padding**, **LowerPad**, and **UpperPad**:

- **KernelShape**: (required) Defines the dimensionality of each kernel for the convolutional bundle. The value must be a tuple of positive integers with a length that equals the arity of the bundle. Each component of this tuple must be no greater than the corresponding component of **InputShape**.
- **Stride**: (optional) Defines the sliding step sizes of the convolution (one step size for each dimension), that is the distance between the central nodes. The value must be a tuple of positive integers with a length that is the arity of the bundle. Each component of this tuple must be no greater than the corresponding component of **KernelShape**. The default value is a tuple with all components equal to one.
- **Sharing**: (optional) Defines the weight sharing for each dimension of the convolution. The value can be a single Boolean value or a tuple of Boolean values with a length that is the arity of the bundle. A single Boolean value is extended to be a tuple of the correct length with all components equal to the specified value. The default value is a tuple that consists of all True values.
- **MapCount**: (optional) Defines the number of feature maps for the convolutional bundle. The value can be a single positive integer or a tuple of positive integers with a length that is the arity of the bundle. A single integer value is extended to be a tuple of the correct length with the first components equal to the specified value and all the remaining components equal to one. The default value is one. The total number of feature maps is the product of the components of the tuple. The factoring of this total number across the components determines how the feature map values are grouped in the destination nodes.
- **Weights**: (optional) Defines the initial weights for the bundle. The value must be a tuple of floating point values with a length that is the number of kernels times the number of weights per kernel, as defined later in this article. The default weights are randomly generated.

There are two sets of properties that control padding, the properties being mutually exclusive:

- **Padding**: (optional) Determines whether the input should be padded by using a **default padding scheme**. The value can be a single Boolean value, or it can be a tuple of Boolean values with a length that is the arity of the bundle. A single Boolean value is extended to be a tuple of the correct length with all components equal to the specified value. If the value for a dimension is True, the source is logically padded in that dimension with zero-valued cells to support additional kernel applications, such that the central nodes of the first and last kernels in that dimension are the first and last nodes in that dimension in the source layer. Thus, the number of "dummy" nodes in each dimension is determined automatically, to fit exactly $(InputShape[d] - 1) / Stride[d] + 1$ kernels into the padded source layer. If the value for a dimension is False, the kernels are defined so that the number of nodes on each side that are left out is the same (up to a difference of 1). The default value of this attribute is a tuple with all components equal to False.
- **UpperPad** and **LowerPad**: (optional) Provide greater control over the amount of padding to use. **Important:** These attributes can be defined if and only if the **Padding** property above is **not** defined. The values should be

integer-valued tuples with lengths that are the arity of the bundle. When these attributes are specified, "dummy" nodes are added to the lower and upper ends of each dimension of the input layer. The number of nodes added to the lower and upper ends in each dimension is determined by **LowerPad[i]** and **UpperPad[i]** respectively. To ensure that kernels correspond only to "real" nodes and not to "dummy" nodes, the following conditions must be met:

- Each component of **LowerPad** must be strictly less than $\text{KernelShape}[d]/2$.
- Each component of **UpperPad** must be no greater than $\text{KernelShape}[d]/2$.
- The default value of these attributes is a tuple with all components equal to 0.

The setting **Padding** = true allows as much padding as is needed to keep the "center" of the kernel inside the "real" input. This changes the math a bit for computing the output size. Generally, the output size D is computed as $D = (I - K) / S + 1$, where I is the input size, K is the kernel size, S is the stride, and $/$ is integer division (round toward zero). If you set $\text{UpperPad} = [1, 1]$, the input size I is effectively 29, and thus $D = (29 - 5) / 2 + 1 = 13$. However, when **Padding** = true, essentially I gets bumped up by $K - 1$; hence $D = ((28 + 4) - 5) / 2 + 1 = 27 / 2 + 1 = 13 + 1 = 14$. By specifying values for **UpperPad** and **LowerPad** you get much more control over the padding than if you just set **Padding** = true.

For more information about convolutional networks and their applications, see these articles:

- <http://deeplearning.net/tutorial/lenet.html>
- <http://research.microsoft.com/pubs/68920/icdar03.pdf>
- http://people.csail.mit.edu/jvb/papers/cnn_tutorial.pdf

Pooling bundles

A **pooling bundle** applies geometry similar to convolutional connectivity, but it uses predefined functions to source node values to derive the destination node value. Hence, pooling bundles have no trainable state (weights or biases). Pooling bundles support all the convolutional attributes except **Sharing**, **MapCount**, and **Weights**.

Typically, the kernels summarized by adjacent pooling units do not overlap. If $\text{Stride}[d]$ is equal to $\text{KernelShape}[d]$ in each dimension, the layer obtained is the traditional local pooling layer, which is commonly employed in convolutional neural networks. Each destination node computes the maximum or the mean of the activities of its kernel in the source layer.

The following example illustrates a pooling bundle:

```
hidden P1 [5, 12, 12]
from C1 max pool {
  InputShape = [5, 24, 24];
  KernelShape = [1, 2, 2];
  Stride = [1, 2, 2];
}
```

- The arity of the bundle is 3 (the length of the tuples **InputShape**, **KernelShape**, and **Stride**).
- The number of nodes in the source layer is $5 * 24 * 24 = 2880$.
- This is a traditional local pooling layer because **KernelShape** and **Stride** are equal.
- The number of nodes in the destination layer is $5 * 12 * 12 = 1440$.

For more information about pooling layers, see these articles:

- <http://www.cs.toronto.edu/~hinton/absps/imagenet.pdf> (Section 3.4)
- <http://cs.nyu.edu/~koray/publis/lecun-iscas-10.pdf>
- <http://cs.nyu.edu/~koray/publis/jarrett-iccv-09.pdf>

Response normalization bundles

Response normalization is a local normalization scheme that was first introduced by Geoffrey Hinton, et al, in the paper [ImageNet Classification with Deep Convolutional Neural Networks](#). Response normalization is used to aid generalization in neural nets. When one neuron is firing at a very high activation level, a local response normalization layer suppresses the activation level of the surrounding neurons. This is done by using three parameters (α , β , and k) and a convolutional structure (or neighborhood shape). Every neuron in the destination layer y corresponds to a neuron x in the source layer. The activation level of y is given by the following formula, where f is the activation level of a neuron, and N_x is the kernel (or the set that contains the neurons in the neighborhood of x), as defined by the following convolutional structure:

$$\frac{f(x)}{\left(k + \frac{\alpha}{|N_x|} \sum_{z \in N_x} (f(z))^2 \right)^\beta}$$

Response normalization bundles support all the convolutional attributes except **Sharing**, **MapCount**, and **Weights**.

- If the kernel contains neurons in the same map as x , the normalization scheme is referred to as **same map normalization**. To define same map normalization, the first coordinate in **InputShape** must have the value 1.
- If the kernel contains neurons in the same spatial position as x , but the neurons are in other maps, the normalization scheme is called **across maps normalization**. This type of response normalization implements a form of lateral inhibition inspired by the type found in real neurons, creating competition for big activation levels amongst neuron outputs computed on different maps. To define across maps normalization, the first coordinate must be an integer greater than one and no greater than the number of maps, and the rest of the coordinates must have the value 1.

Because response normalization bundles apply a predefined function to source node values to determine the destination node value, they have no trainable state (weights or biases).

Alert: The nodes in the destination layer correspond to neurons that are the central nodes of the kernels. For example, if *KernelShape[d]* is odd, then *KernelShape[d]/2* corresponds to the central kernel node. If *KernelShape[d]* is even, the central node is at *KernelShape[d]/2 - 1*. Therefore, if **Padding**[d] is False, the first and the last *KernelShape[d]/2* nodes do not have corresponding nodes in the destination layer. To avoid this situation, define **Padding** as [true, true, ..., true].

In addition to the four attributes described earlier, response normalization bundles also support the following attributes:

- **Alpha**: (required) Specifies a floating-point value that corresponds to α in the previous formula.
- **Beta**: (required) Specifies a floating-point value that corresponds to β in the previous formula.
- **Offset**: (optional) Specifies a floating-point value that corresponds to k in the previous formula. It defaults to 1.

The following example defines a response normalization bundle using these attributes:

```
hidden RN1 [5, 10, 10]
from P1 response norm {
    InputShape = [ 5, 12, 12];
    KernelShape = [ 1, 3, 3];
    Alpha = 0.001;
    Beta = 0.75;
}
```

- The source layer includes five maps, each with aof dimension of 12x12, totaling in 1440 nodes.
- The value of **KernelShape** indicates that this is a same map normalization layer, where the neighborhood is a 3x3 rectangle.

- The default value of **Padding** is False, thus the destination layer has only 10 nodes in each dimension. To include one node in the destination layer that corresponds to every node in the source layer, add `Padding = [true, true, true]`; and change the size of RN1 to [5, 12, 12].

Share declaration

Net# optionally supports defining multiple bundles with shared weights. The weights of any two bundles can be shared if their structures are the same. The following syntax defines bundles with shared weights:

```

share-declaration:
share { layer-list }
share { bundle-list }
share { bias-list }

layer-list:
layer-name , layer-name
layer-list , layer-name

bundle-list:
bundle-spec , bundle-spec
bundle-list , bundle-spec

bundle-spec:
layer-name => layer-name

bias-list:
bias-spec , bias-spec
bias-list , bias-spec

bias-spec:
1 => layer-name

layer-name:
identifier

```

For example, the following share-declaration specifies the layer names, indicating that both weights and biases should be shared:

```

Const {
  InputSize = 37;
  HiddenSize = 50;
}
input {
  Data1 [InputSize];
  Data2 [InputSize];
}
hidden {
  H1 [HiddenSize] from Data1 all;
  H2 [HiddenSize] from Data2 all;
}
output Result [2] {
  from H1 all;
  from H2 all;
}
share { H1, H2 } // share both weights and biases

```

- The input features are partitioned into two equal sized input layers.
- The hidden layers then compute higher level features on the two input layers.
- The share-declaration specifies that *H1* and *H2* must be computed in the same way from their respective inputs.

Alternatively, this could be specified with two separate share-declarations as follows:

```
share { Data1 => H1, Data2 => H2 } // share weights
```

```
share { 1 => H1, 1 => H2 } // share biases
```

You can use the short form only when the layers contain a single bundle. In general, sharing is possible only when the relevant structure is identical, meaning that they have the same size, same convolutional geometry, and so forth.

Examples of Net# usage

This section provides some examples of how you can use Net# to add hidden layers, define the way that hidden layers interact with other layers, and build convolutional networks.

Define a simple custom neural network: "Hello World" example

This simple example demonstrates how to create a neural network model that has a single hidden layer.

```
input Data auto;
hidden H [200] from Data all;
output Out [10] sigmoid from H all;
```

The example illustrates some basic commands as follows:

- The first line defines the input layer (named *Data*). When you use the **auto** keyword, the neural network automatically includes all feature columns in the input examples.
- The second line creates the hidden layer. The name *H* is assigned to the hidden layer, which has 200 nodes. This layer is fully connected to the input layer.
- The third line defines the output layer (named *O*), which contains 10 output nodes. If the neural network is used for classification, there is one output node per class. The keyword **sigmoid** indicates that the output function is applied to the output layer.

Define multiple hidden layers: computer vision example

The following example demonstrates how to define a slightly more complex neural network, with multiple custom hidden layers.

```
// Define the input layers
input Pixels [10, 20];
input MetaData [7];

// Define the first two hidden layers, using data only from the Pixels input
hidden ByRow [10, 12] from Pixels where (s,d) => s[0] == d[0];
hidden ByCol [5, 20] from Pixels where (s,d) => abs(s[1] - d[1]) <= 1;

// Define the third hidden layer, which uses as source the hidden layers ByRow and ByCol
hidden Gather [100]
{
    from ByRow all;
    from ByCol all;
}

// Define the output layer and its sources
output Result [10]
{
    from Gather all;
    from MetaData all;
}
```

This example illustrates several features of the neural networks specification language:

- The structure has two input layers, *Pixels* and *MetaData*.
- The *Pixels* layer is a source layer for two connection bundles, with destination layers, *ByRow* and *ByCol*.
- The layers *Gather* and *Result* are destination layers in multiple connection bundles.
- The output layer, *Result*, is a destination layer in two connection bundles; one with the second level hidden (*Gather*) as a destination layer, and the other with the input layer (*MetaData*) as a destination layer.
- The hidden layers, *ByRow* and *ByCol*, specify filtered connectivity by using predicate expressions. More precisely, the node in *ByRow* at [x, y] is connected to the nodes in *Pixels* that have the first index coordinate equal to the node's first coordinate, x. Similarly, the node in *ByCol* at [x, y] is connected to the nodes in *_Pixels* that have the second index coordinate within one of the node's second coordinate, y.

Define a convolutional network for multiclass classification: digit recognition example

The definition of the following network is designed to recognize numbers, and it illustrates some advanced techniques for customizing a neural network.

```
input Image [29, 29];
hidden Conv1 [5, 13, 13] from Image convolve
{
    InputShape = [29, 29];
    KernelShape = [ 5, 5];
    Stride = [ 2, 2];
    MapCount = 5;
}
hidden Conv2 [50, 5, 5]
from Conv1 convolve
{
    InputShape = [ 5, 13, 13];
    KernelShape = [ 1, 5, 5];
    Stride = [ 1, 2, 2];
    Sharing = [false, true, true];
    MapCount = 10;
}
hidden Hid3 [100] from Conv2 all;
output Digit [10] from Hid3 all;
```

- The structure has a single input layer, *Image*.
- The keyword **convolve** indicates that the layers named *Conv1* and *Conv2* are convolutional layers. Each of these layer declarations is followed by a list of the convolution attributes.
- The net has a third hidden layer, *Hid3*, which is fully connected to the second hidden layer, *Conv2*.
- The output layer, *Digit*, is connected only to the third hidden layer, *Hid3*. The keyword **all** indicates that the output layer is fully connected to *Hid3*.
- The arity of the convolution is three (the length of the tuples **InputShape**, **KernelShape**, **Stride**, and **Sharing**).
- The number of weights per kernel is $1 + **\text{KernelShape}[0] * **\text{KernelShape}[1] * **\text{KernelShape}[2] = 1 + 1 * 5 * 5 = 26$. Or $26 * 50 = 1300$.
- You can calculate the nodes in each hidden layer as follows:
 - **NodeCount**[0] = $(5 - 1) / 1 + 1 = 5$.
 - **NodeCount**[1] = $(13 - 5) / 2 + 1 = 5$.
 - **NodeCount**[2] = $(13 - 5) / 2 + 1 = 5$.
- The total number of nodes can be calculated by using the declared dimensionality of the layer, [50, 5, 5], as follows: **MapCount*** * ****NodeCount**[0] * **NodeCount**[1] * **NodeCount**[2] = $10 * 5 * 5 * 5 * 5 = 1250$.
- Because **Sharing**[d] is False only for d == 0, the number of kernels is **MapCount*** * ****NodeCount**[0] = $10 * 5 = 50$.

Acknowledgements

The Net# language for customizing the architecture of neural networks was developed at Microsoft by Shon

Katzenberger (Architect, Machine Learning) and Alexey Kamenev (Software Engineer, Microsoft Research). It is used internally for machine learning projects and applications ranging from image detection to text analytics. For more information, see [Neural Nets in Azure ML - Introduction to Net#](#)

Get help from Machine Learning Live Chat Support

1/17/2017 • 1 min to read • [Edit on GitHub](#)

Azure Machine Learning Studio provides an intuitive interface for building machine learning models. There is a [Gallery](#) of examples contributed by the community to help you get started. And there is a [Forum](#) where you can ask questions to help keep you going.

But sometimes you just need to ask a quick question to unblock you. In [Machine Learning Studio](#) look for the chat icon in the top navigation. If you see this icon, that means a member of the product team is online to help you in real time.



Type in your question and get your answers!

A screenshot of a live chat interface titled "Chat with us". It shows a message from "olga" that reads: "Hello! I have questions related to Multiclass Decision Forest algorithm....". Below the message is a "Start Chatting" button and the Zopim logo.

Don't see the Live Chat Icon?

The Live Chat is staffed by members of the Machine Learning team. If you don't see the live chat icon it's because the team is not currently available. For example, it may be outside of normal working hours.

NOTE

[Try Azure Machine Learning for free](#)

No credit card or Azure subscription needed. [Get started now >](#)