

Table of Contents

Team Data Science Process Documentation

Overview

Lifecycle

1. Business understanding
2. Data acquisition and understanding
3. Modeling
4. Deployment
5. Customer acceptance

With Azure ML

Roles and tasks

Group manager

Team lead

Project lead

Individual contributor

Project structure

Project planning

Project execution

Agile development

Collaborative coding with Git

Execute data science tasks

Track progress

Examples

Azure Machine Learning

Sentiment classification

US income classification

Biomedical entity recognition

Spark with PySpark and Scala

Explore and model data

Advanced data exploration and modeling

[Score models](#)

[Hive with HDInsight Hadoop](#)

[U-SQL with Azure Data Lake](#)

[R, Python and T-SQL with SQL Server](#)

[T-SQL and Python with SQL DW](#)

[Training](#)

[For data scientists](#)

[For DevOps](#)

[How To](#)

[Use TDSP in Azure ML](#)

[Set up data science environments](#)

[Azure storage accounts](#)

[Platforms and tools](#)

[R and Python on HDInsight clusters](#)

[Azure Machine Learning workspace](#)

[Set up a Data Science VM](#)

[Analyze business needs](#)

[Identify your scenario](#)

[Acquire and understand data](#)

[Manage data logistics](#)

[Explore and visualize data](#)

[Develop models](#)

[Engineer features](#)

[Select features](#)

[Create and train models](#)

[Deploy models in production](#)

[Related](#)

[Microsoft Cognitive Toolkit - CNTK](#)

[Cortana Intelligence Partner Solutions](#)

[Cortana Intelligence Gallery](#)

[Cortana Intelligence publishing guide](#)

[Cortana Intelligence solution evaluation tool](#)

Cortana Analytics

APIs

Resources

[TDSP tools and utilities](#)

[Download from GitHub](#)

[Data Science Utilities v.0.11 for the TDSP](#)

[Latest utilities for the TDSP](#)

Blogs

[Recent Updates for the TDSP](#)

[Using the TDSP in Azure Machine Learning](#)

[Data Science Project Planning Template](#)

[Cortana Intelligence and Machine Learning Blog](#)

Organizations using TDSP

[New Signature](#)

[Blue Granite](#)

[Microsoft Consulting Services](#)

Related Microsoft resources

[MSDN forum](#)

[Stack Overflow](#)

[Videos](#)

What is the Team Data Science Process?

10/20/2017 • 4 min to read • [Edit Online](#)

The Team Data Science Process (TDSP) is an agile, iterative data science methodology to deliver predictive analytics solutions and intelligent applications efficiently. TDSP helps improve team collaboration and learning. It contains a distillation of the best practices and structures from Microsoft and others in the industry that facilitate the successful implementation of data science initiatives. The goal is to help companies fully realize the benefits of their analytics program.

This article provides an overview of TDSP and its main components. We provide a generic description of the process here that can be implemented with a variety of tools. A more detailed description of the project tasks and roles involved in the lifecycle of the process is provided in additional linked topics. Guidance on how to implement the TDSP using a specific set of Microsoft tools and infrastructure that we use to implement the TDSP in our teams is also provided.

Key components of the TDSP

TDSP comprises of the following key components:

- A **data science lifecycle** definition
- A **standardized project structure**
- **Infrastructure and resources** for data science projects
- **Tools and utilities** for project execution

Data science lifecycle

The Team Data Science Process (TDSP) provides a lifecycle 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. At a high level, these different methodologies have much in common.

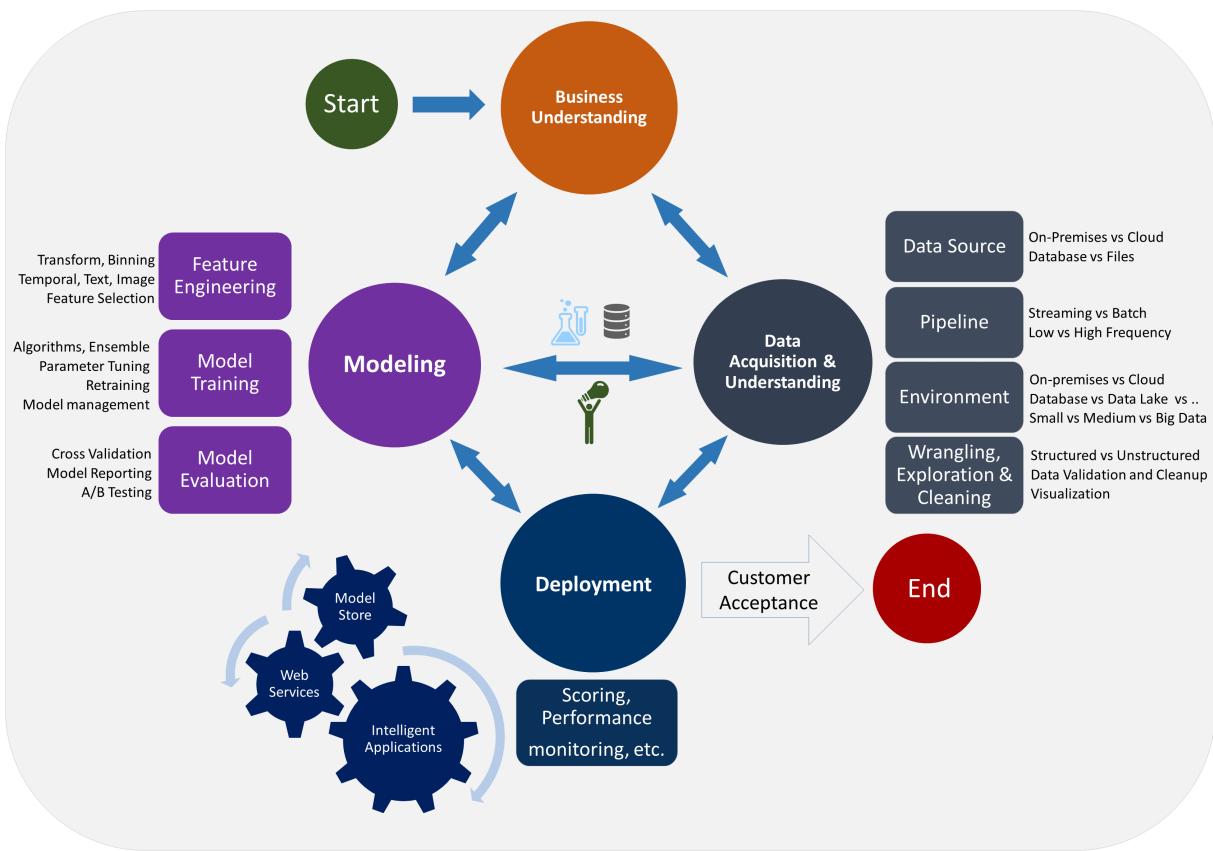
This lifecycle has been designed for data science projects that ship as part of intelligent applications. These applications deploy machine learning or artificial intelligence models for predictive analytics. Exploratory data science projects or ad hoc analytics projects can also benefit from using this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

The lifecycle outlines the major stages that projects typically execute, often iteratively:

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

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

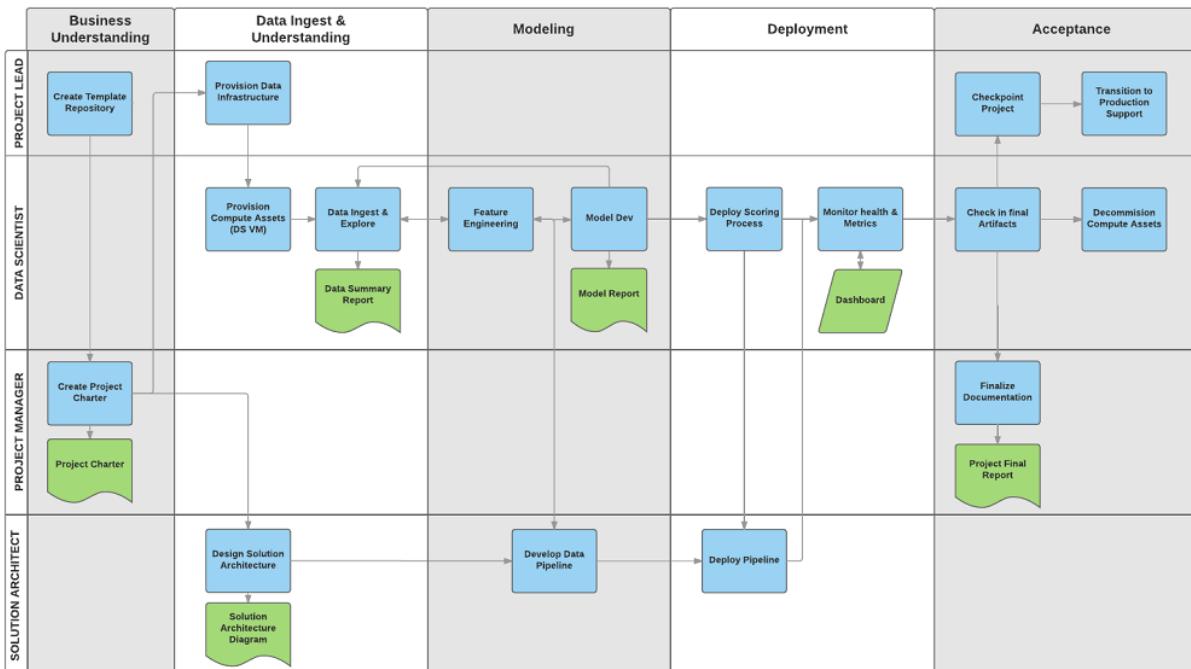
Data Science Lifecycle



The goals, tasks, and documentation artifacts for each stage of the lifecycle in TDSP are described in the [Team Data Science Process lifecycle](#) topic. These tasks and artifacts are associated with project roles:

- Solution architect
- Project manager
- Data scientist
- Project lead

The following diagram provides a grid view of the tasks (in blue) and artifacts (in green) associated with each stage of the lifecycle (on the horizontal axis) for these roles (on the vertical axis).

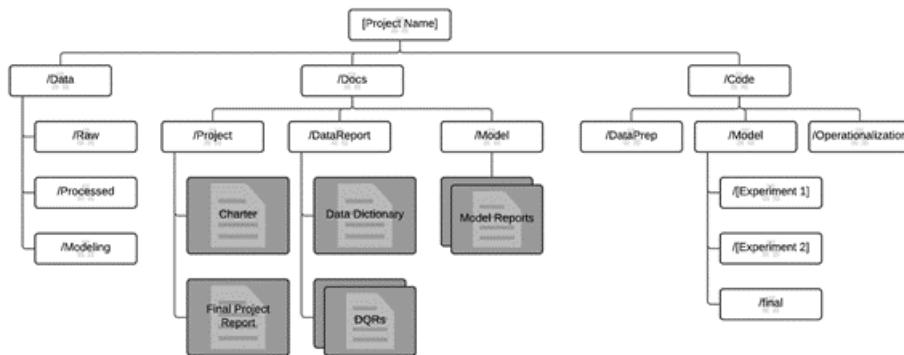


Standardized project structure

Having all projects share a directory structure and use templates for project documents makes it easy for the team members to find information about their projects. All code and documents are stored in a version control system (VCS) like Git, TFS, or Subversion to enable team collaboration. Tracking tasks and features in an agile project tracking system like Jira, Rally, Visual Studio Team Services allows closer tracking of the code for individual features. Such tracking also enables teams to obtain better cost estimates. TDSP recommends creating a separate repository for each project on the VCS for versioning, information security, and collaboration. The standardized structure for all projects helps build institutional knowledge across the organization.

We provide templates for the folder structure and required documents in standard locations. This folder structure organizes the files that contain code for data exploration and feature extraction, and that record model iterations. These templates make it easier for team members to understand work done by others and to add new members to teams. It is easy to view and update document templates in markdown format. Use templates to provide checklists with key questions for each project to insure that the problem is well-defined and that deliverables meet the quality expected. Examples include:

- a project charter to document the business problem and scope of the project
- data reports to document the structure and statistics of the raw data
- model reports to document the derived features
- model performance metrics such as ROC curves or MSE



The directory structure can be cloned from [Github](#).

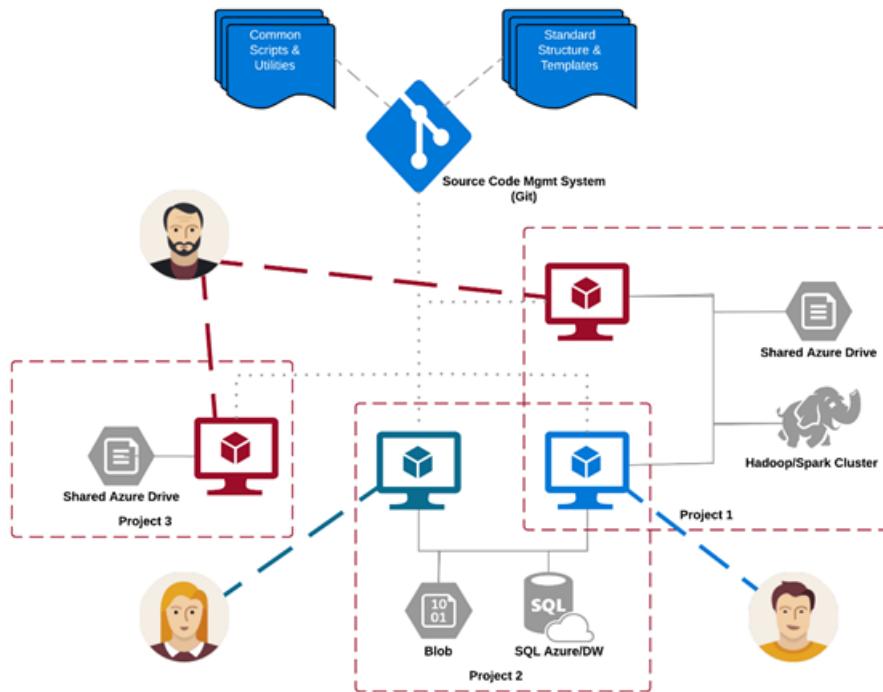
Infrastructure and resources for data science projects

TDSP provides recommendations for managing shared analytics and storage infrastructure such as:

- cloud file systems for storing datasets,
- databases
- big data (Hadoop or Spark) clusters
- machine learning services.

The analytics and storage infrastructure can be in the cloud or on-premises. This is where raw and processed datasets are stored. This infrastructure enables reproducible analysis. It also avoids duplication, which can lead to inconsistencies and unnecessary infrastructure costs. Tools are provided to provision the shared resources, track them, and allow each team member to connect to those resources securely. It is also a good practice have project members create a consistent compute environment. Different team members can then replicate and validate experiments.

Here is an example of a team working on multiple projects and sharing various cloud analytics infrastructure components.



Tools and utilities for project execution

Introducing processes in most organizations is challenging. Tools provided to implement the data science process and lifecycle help lower the barriers to and increase the consistency of their adoption. TDSP provides an initial set of tools and scripts to jump-start adoption of TDSP within a team. It also helps automate some of the common tasks in the data science lifecycle such as data exploration and baseline modeling. There is a well-defined structure provided for individuals to contribute shared tools and utilities into their team's shared code repository. These resources can then be leveraged by other projects within the team or the organization. TDSP also plans to enable the contributions of tools and utilities to the whole community. The TDSP utilities can be cloned from [Github](#).

Next steps

[Team Data Science Process: Roles and tasks](#) Outlines the key personnel roles and their associated tasks for a data science team that standardizes on this process.

The Team Data Science Process lifecycle

11/9/2017 • 1 min to read • [Edit Online](#)

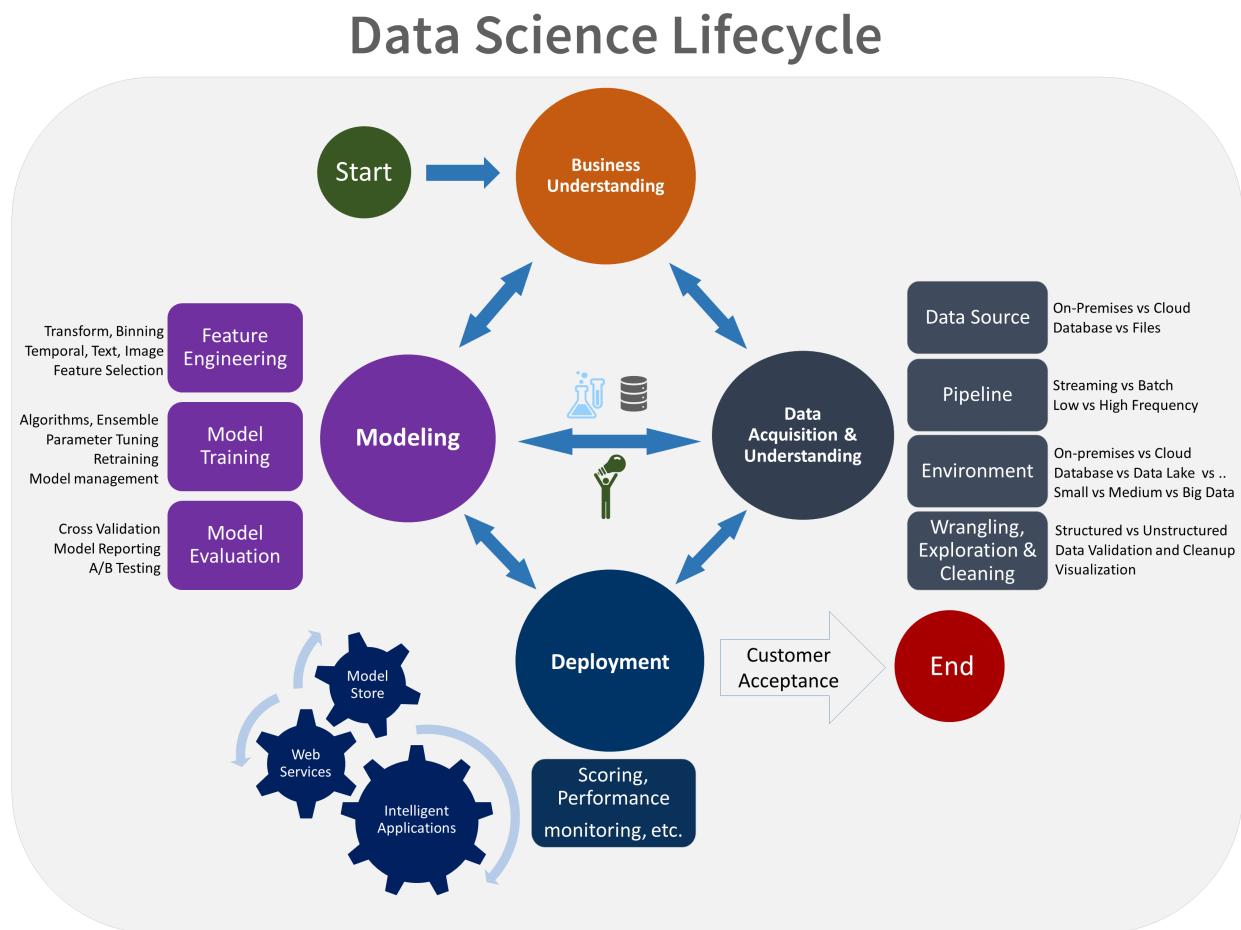
The Team Data Science Process (TDSP) provides a recommended lifecycle that you can use to structure your data-science projects. The lifecycle outlines the steps, from start to finish, that projects usually follow when they are executed. If you use another data-science lifecycle, such as the Cross Industry Standard Process for Data Mining ([CRISP-DM](#)), Knowledge Discovery in Databases ([KDD](#)), or your organization's own custom process, you can still use the task-based TDSP.

This lifecycle is 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 analytics projects can also benefit from the use of this process. But for those projects, some of the steps described here might not be needed.

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

1. [Business understanding](#)
2. [Data acquisition and understanding](#)
3. [Modeling](#)
4. [Deployment](#)
5. [Customer acceptance](#)

Here is a visual representation of the TDSP lifecycle:



The TDSP lifecycle is modeled as a sequence of iterated steps that provide guidance on the tasks needed to use predictive models. You deploy the predictive models in the production environment that you plan to use to build

the intelligent applications. The goal of this process lifecycle is to continue to move a data-science project toward a clear engagement end point. Data science is an exercise in research and discovery. The ability to communicate tasks to your team and your customers by using a well-defined set of artifacts that employ standardized templates helps to avoid misunderstandings. Using these templates also increases the chance of the successful completion of a complex data-science project.

For each stage, we provide the following information:

- **Goals:** The specific objectives.
- **How to do it:** An outline of the specific tasks and guidance on how to complete them.
- **Artifacts:** The deliverables and the support to produce them.

Next steps

We provide full end-to-end walkthroughs that demonstrate all the steps in the process for specific scenarios. The [Example walkthroughs](#) article provides a list of the scenarios with links and thumbnail descriptions. The walkthroughs illustrate how to combine cloud, on-premises tools, and services into a workflow or pipeline to create an intelligent application.

For examples of how to execute steps in TDSPs that use Azure Machine Learning Studio, see [Use the TDSP with Azure Machine Learning](#).

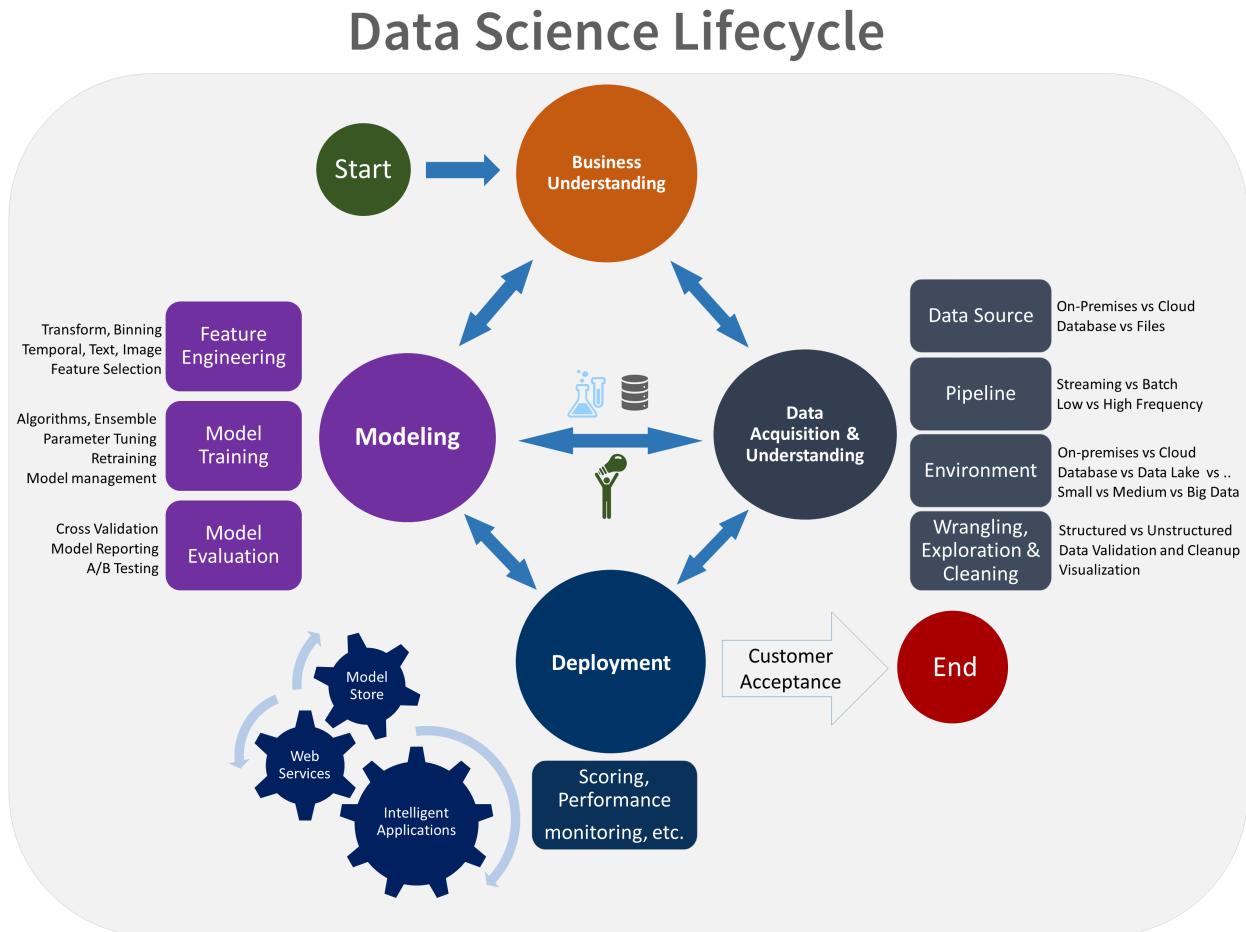
Business understanding

11/9/2017 • 3 min to read • [Edit Online](#)

This article outlines the goals, tasks, and deliverables associated with the business understanding stage of the Team Data Science Process (TDSP). This process provides a recommended lifecycle that you can use to structure your data-science projects. The lifecycle outlines the major stages that projects typically execute, often iteratively:

1. **Business understanding**
2. **Data acquisition and understanding**
3. **Modeling**
4. **Deployment**
5. **Customer acceptance**

Here is a visual representation of the TDSP lifecycle:



Goals

- Specify the key variables that are to serve as the model targets and whose related metrics are used determine the success of the project.
- Identify the relevant data sources that the business has access to or needs to obtain.

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 that the 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.

Define objectives

1. A central objective of this step is to identify the key business variables that the analysis needs to predict. We refer to these variables as the *model targets*, and we use the metrics associated with them to determine the success of the project. Two examples of such targets are sales forecasts or the probability of an order being fraudulent.

2. Define the project goals by asking and refining "sharp" questions that are relevant, specific, and unambiguous. Data science is a process that uses names and numbers to answer such questions. For more information on asking sharp questions, see the [How to do data science](#) blog. You typically use data science or machine learning 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)

Determine which of these questions you're asking and how answering it achieves your business goals.

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 you discover more information.

4. Define the success metrics. For example, you might want to achieve a customer churn prediction. You need an accuracy rate of "x" percent by the end of this three-month project. With this data, you can offer customer promotions to reduce churn. The metrics must be **SMART**:

- Specific
- Measurable
- Achievable
- Relevant
- Time-bound

Identify data sources

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

- Data that's relevant to the question. Do you have measures of the target and features that are related to the target?
- Data that's an accurate measure of your model target and the features of interest.

For example, you might find that the existing systems need to collect and log additional kinds of data to address the problem and achieve the project goals. In this situation, you might want to look for external data sources or update your systems to collect new data.

Artifacts

Here are the deliverables in this stage:

- **Charter document:** A standard template is provided in the TDSP project structure definition. The charter document is a living document. You update the template throughout the project as you make new discoveries 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:** The **Raw data sources** section of the **Data definitions** report that's found in the TDSP project **Data report** folder contains the data sources. This section specifies the original and destination locations for the raw data. In later stages, you fill in additional details like the scripts to move the data to your analytic environment.
- **Data dictionaries:** This document provides descriptions of the data that's provided by the client. These descriptions include information about the schema (the data types and information on the validation rules, if any) and the entity-relation diagrams, if available.

Next steps

Here are links to each step in the lifecycle of the TDSP:

1. [Business understanding](#)
2. [Data acquisition and understanding](#)
3. [Modeling](#)
4. [Deployment](#)
5. [Customer acceptance](#)

We provide full end-to-end walkthroughs that demonstrate all the steps in the process for specific scenarios. The [Example walkthroughs](#) article provides a list of the scenarios with links and thumbnail descriptions. The walkthroughs illustrate how to combine cloud, on-premises tools, and services into a workflow or pipeline to create an intelligent application.

For examples of how to execute steps in TDSPs that use Azure Machine Learning Studio, see [Use the TDSP with Azure Machine Learning](#).

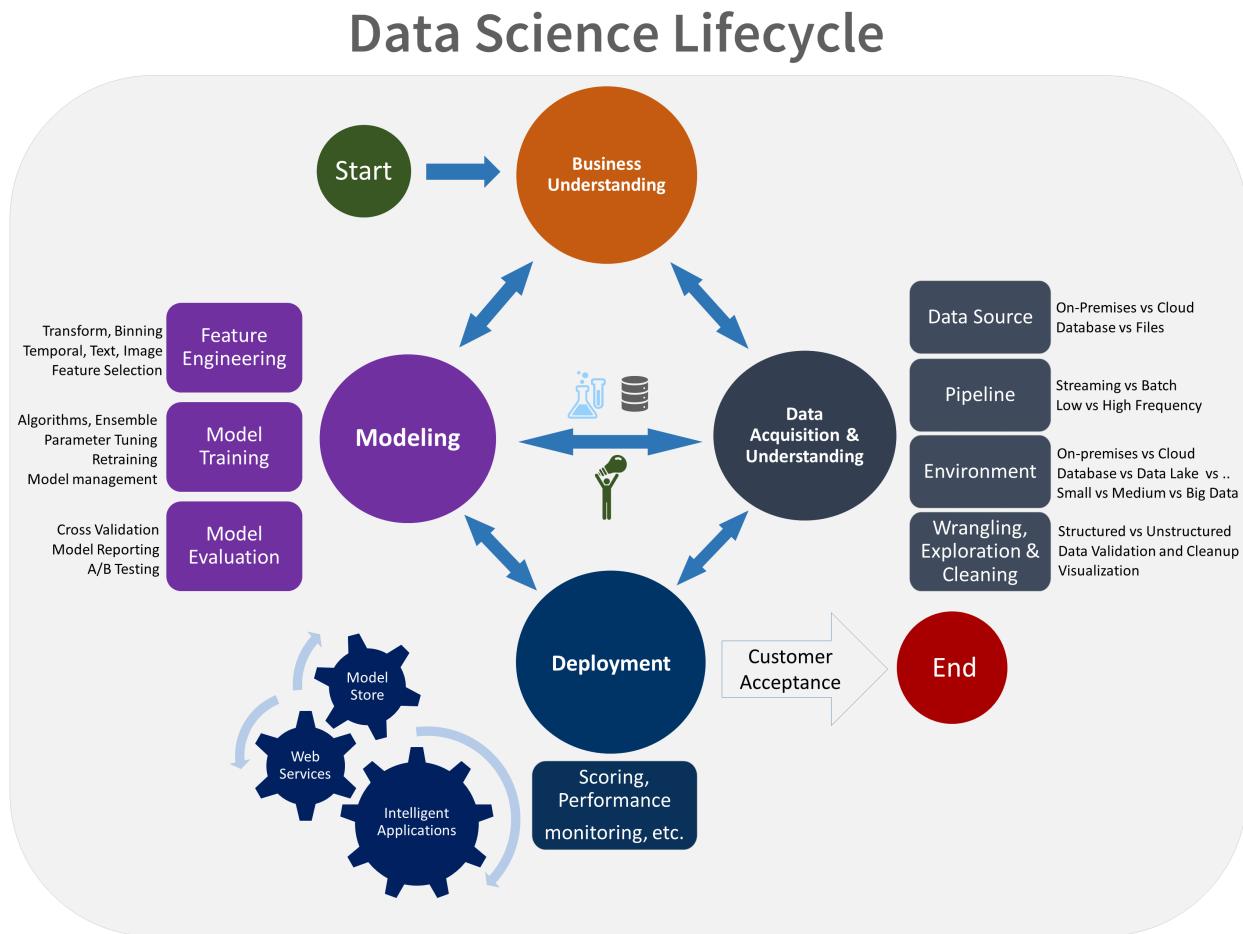
Data acquisition and understanding

11/9/2017 • 4 min to read • [Edit Online](#)

This article outlines the goals, tasks, and deliverables associated with the data acquisition and understanding stage of the Team Data Science Process (TDSP). This process provides a recommended lifecycle that you can use to structure your data-science projects. The lifecycle outlines the major stages that projects typically execute, often iteratively:

1. **Business understanding**
2. **Data acquisition and understanding**
3. **Modeling**
4. **Deployment**
5. **Customer acceptance**

Here is a visual representation of the TDSP lifecycle:



Goals

- Produce a clean, high-quality data set whose relationship to the target variables is understood. Locate the data set in the appropriate analytics environment so you are ready to model.
- Develop a solution architecture of the data pipeline that refreshes and scores the data regularly.

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.

Ingest the data

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

Explore the data

Before you train your models, you need to develop a sound understanding of the data. Real-world data sets are often noisy, are missing values, or have a host of other discrepancies. You can use data summarization and visualization to audit the quality of your data and provide the information you need to process the data before it's ready for modeling. This process is often iterative.

TDSP provides an automated utility, called [IDEAR](#), to help visualize the data and prepare data summary reports. We recommend that you start with IDEAR first to explore the data to help develop initial data understanding interactively with no coding. Then you can write custom code for data exploration and visualization. For guidance on cleaning the data, see [Tasks to prepare data for enhanced machine learning](#).

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

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. You do this by setting up a data pipeline or workflow. The [Move data from an on-premises SQL Server instance to Azure SQL Database with Azure Data Factory](#) article gives an example of how to set up a pipeline with [Azure Data Factory](#).

In this stage, you develop a solution architecture of the data pipeline. You develop the pipeline in parallel with the next stage of the data science project. Depending on your business needs and the constraints of your existing systems into which this solution is being integrated, the pipeline can be one of the following:

- Batch-based
- Streaming or real time
- A hybrid

Artifacts

The following are the deliverables in this stage:

- **Data quality report:** This report includes data summaries, the relationships between each attribute and target, variable ranking, and more. The [IDEAR](#) tool provided as part of TDSP can quickly generate this report on any tabular data set, such as a CSV file or a relational table.
- **Solution architecture:** The solution architecture can be a diagram or description of your data pipeline that you use to run scoring or predictions on new data after you have built a model. It also contains the pipeline to retrain your model based on new data. Store the document in the [Project](#) directory when you use the TDSP directory structure template.
- **Checkpoint decision:** Before you begin full-feature engineering and model building, you can reevaluate the project to determine whether the value expected is sufficient to continue pursuing it. You might, 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.

Next steps

Here are links to each step in the lifecycle of the TDSP:

1. [Business understanding](#)
2. [Data acquisition and understanding](#)
3. [Modeling](#)
4. [Deployment](#)
5. [Customer acceptance](#)

We provide full end-to-end walkthroughs that demonstrate all the steps in the process for specific scenarios. The [Example walkthroughs](#) article provides a list of the scenarios with links and thumbnail descriptions. The walkthroughs illustrate how to combine cloud, on-premises tools, and services into a workflow or pipeline to create an intelligent application.

For examples of how to execute steps in TDSPs that use Azure Machine Learning Studio, see [Use the TDSP with Azure Machine Learning](#).

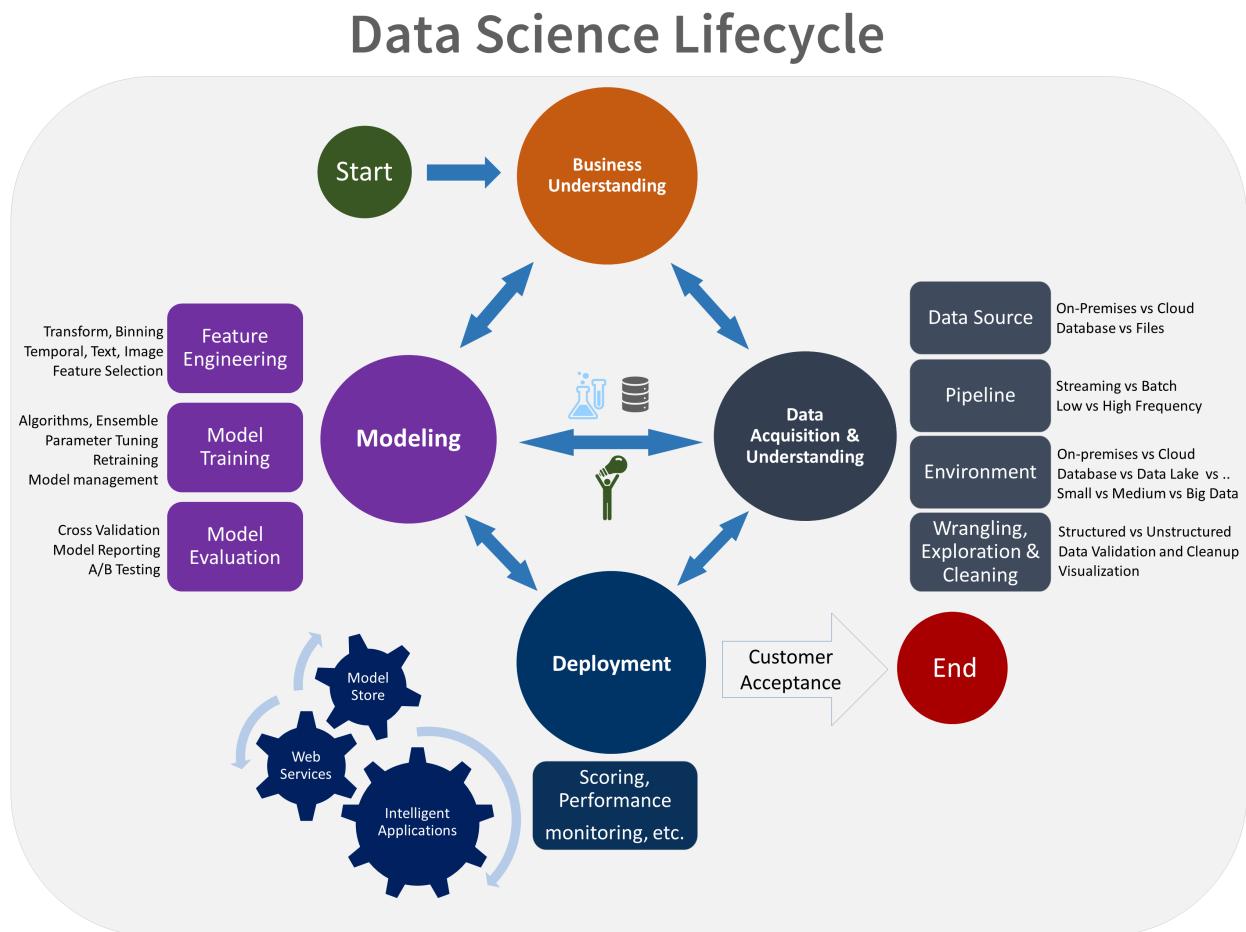
Modeling

11/9/2017 • 4 min to read • [Edit Online](#)

This article outlines the goals, tasks, and deliverables associated with the modeling stage of the Team Data Science Process (TDSP). This process provides a recommended lifecycle that you can use to structure your data-science projects. The lifecycle outlines the major stages that projects typically execute, often iteratively:

1. **Business understanding**
2. **Data acquisition and understanding**
3. **Modeling**
4. **Deployment**
5. **Customer acceptance**

Here is a visual representation of the TDSP lifecycle:



Goals

- Determine the optimal data features for the machine-learning model.
- Create an informative machine-learning model that predicts the target most accurately.
- Create a machine-learning model that's 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**.

Feature engineering

Feature engineering involves the 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 the features relate to each other and how the machine-learning algorithms are to use those features.

This step requires a creative combination of domain expertise and the insights obtained from the data exploration step. Feature engineering is a balancing act of finding and including informative variables, but at the same time trying to avoid too many unrelated variables. Informative variables improve your result; unrelated variables introduce unnecessary noise into the model. You also need to generate these features for any new data obtained during scoring. As a result, the generation of these features can only depend on data that's available at the time of scoring.

For technical guidance on feature engineering when make use of various Azure data technologies, see [Feature engineering in the data science process](#).

Model training

Depending on the type of question that you're trying to 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 uses Azure Machine Learning, the guidance it provides is useful for any machine-learning projects.

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** by using the training data set.
- **Evaluate** the training and the test data set. Use a series of competing machine-learning algorithms along with the various associated tuning parameters (known as a *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 metrics between alternative methods.

NOTE

Avoid leakage: You can cause data leakage if you include data from outside the training data set 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 leakage often requires iterating between building an analysis data set, creating a model, and evaluating the accuracy of the results.

We provide an [automated modeling and reporting tool](#) with TDSP that's able to run through multiple algorithms and parameter sweeps to produce a baseline model. It also produces a baseline modeling report that summarizes the 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 sets** section of the **Data**

definition report. It contains pointers to the code to generate the features and a description of how the feature was generated.

- **Model report:** For each model that's tried, a standard, template-based report that provides details on each experiment is produced.
- **Checkpoint decision:** Evaluate whether the model performs 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 you try any alternative approaches? Should you collect additional data, do more feature engineering, or experiment with other algorithms?

Next steps

Here are links to each step in the lifecycle of the TDSP:

1. [Business understanding](#)
2. [Data acquisition and understanding](#)
3. [Modeling](#)
4. [Deployment](#)
5. [Customer acceptance](#)

We provide full end-to-end walkthroughs that demonstrate all the steps in the process for specific scenarios. The [Example walkthroughs](#) article provides a list of the scenarios with links and thumbnail descriptions. The walkthroughs illustrate how to combine cloud, on-premises tools, and services into a workflow or pipeline to create an intelligent application.

For examples of how to execute steps in TDSPs that use Azure Machine Learning Studio, see [Use the TDSP with Azure Machine Learning](#).

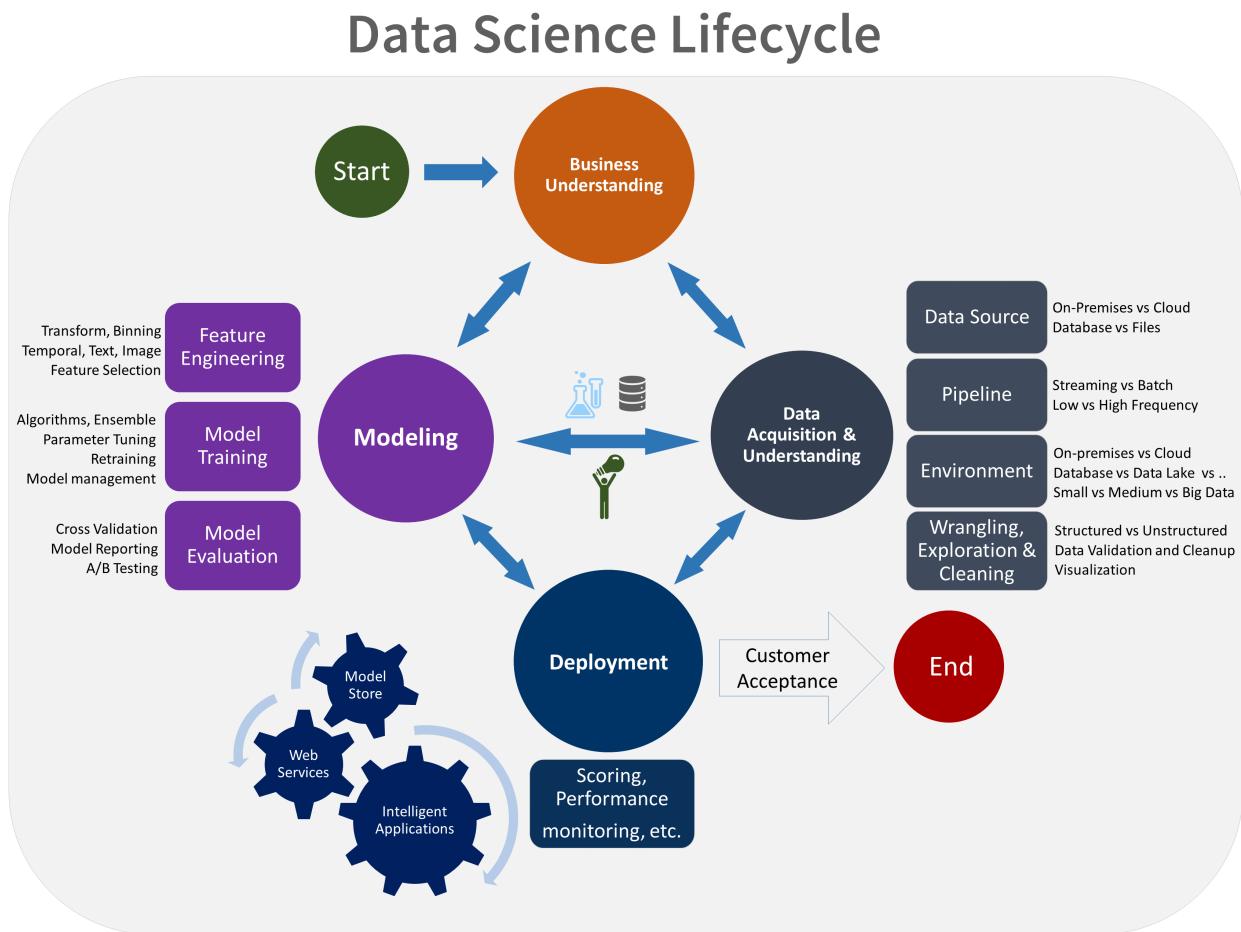
Deployment

11/9/2017 • 1 min to read • [Edit Online](#)

This article outlines the goals, tasks, and deliverables associated with the deployment of the Team Data Science Process (TDSP). This process provides a recommended lifecycle that you can use to structure your data-science projects. The lifecycle outlines the major stages that projects typically execute, often iteratively:

1. **Business understanding**
2. **Data acquisition and understanding**
3. **Modeling**
4. **Deployment**
5. **Customer acceptance**

Here is a visual representation of the TDSP lifecycle:



Goal

Deploy models with a data pipeline 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.

Operationalize a model

After you have a set of models that perform well, you can operationalize them for other applications to consume. Depending on the business requirements, predictions are made either in real time or on a batch basis. To deploy models, you expose them with an open API interface. The interface enables the model to be easily consumed from various applications, such as:

- Online websites
- Spreadsheets
- Dashboards
- Line-of-business applications
- Back-end applications

For examples of model operationalization with an Azure Machine Learning web service, see [Deploy an Azure Machine Learning web service](#). It is a best practice to build telemetry and monitoring into the production model and the data pipeline that you deploy. This practice helps with subsequent system status reporting and troubleshooting.

Artifacts

- A status dashboard that displays the system health and key metrics
- A final modeling report with deployment details
- A final solution architecture document

Next steps

Here are links to each step in the lifecycle of the TDSP:

1. [Business understanding](#)
2. [Data Acquisition and understanding](#)
3. [Modeling](#)
4. [Deployment](#)
5. [Customer acceptance](#)

We provide full end-to-end walkthroughs that demonstrate all the steps in the process for specific scenarios. The [Example walkthroughs](#) article provides a list of the scenarios with links and thumbnail descriptions. The walkthroughs illustrate how to combine cloud, on-premises tools, and services into a workflow or pipeline to create an intelligent application.

For examples of how to execute steps in TDSPs that use Azure Machine Learning Studio, see [Use the TDSP with Azure Machine Learning](#).

Customer acceptance

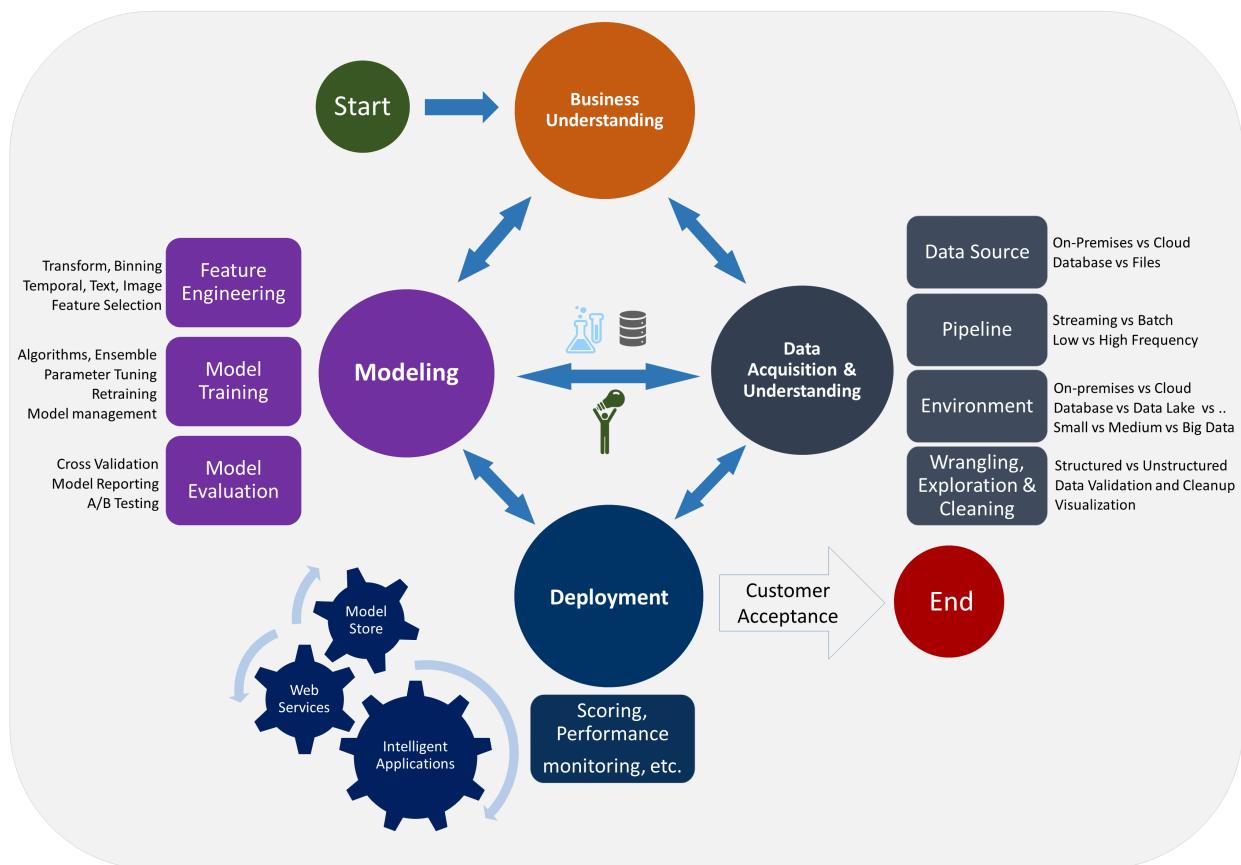
11/9/2017 • 1 min to read • [Edit Online](#)

This article outlines the goals, tasks, and deliverables associated with the customer acceptance stage of the Team Data Science Process (TDSP). This process provides a recommended lifecycle that you can use to structure your data-science projects. The lifecycle outlines the major stages that projects typically execute, often iteratively:

1. **Business understanding**
2. **Data acquisition and understanding**
3. **Modeling**
4. **Deployment**
5. **Customer acceptance**

Here is a visual representation of the TDSP lifecycle:

Data Science Lifecycle



Goal

Finalize the project deliverables: Confirm that the pipeline, the model, and their deployment in a production environment satisfy the customer's objectives.

How to do it

There are two main tasks addressed in this stage:

- **System validation:** Confirm that the deployed model and pipeline meet the customer's needs.

- **Project hand-off:** Hand the project off to the entity that's going to run the system in production.

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

Artifacts

The main artifact produced in this final stage is the **Exit report of the project for the customer**. This technical report contains all the details of the project that are useful for learning about how to operate the system. TDSP provides an [Exit report](#) template. You can use the template as is, or you can customize it for specific client needs.

Next steps

Here are links to each step in the lifecycle of the TDSP:

1. [Business understanding](#)
2. [Data acquisition and understanding](#)
3. [Modeling](#)
4. [Deployment](#)
5. [Customer acceptance](#)

We provide full end-to-end walkthroughs that demonstrate all the steps in the process for specific scenarios. The [Example walkthroughs](#) article provides a list of the scenarios with links and thumbnail descriptions. The walkthroughs illustrate how to combine cloud, on-premises tools, and services into a workflow or pipeline to create an intelligent application.

For examples of how to execute steps in TDSPs that use Azure Machine Learning Studio, see [Use the TDSP with Azure Machine Learning](#).

Team Data Science Process roles and tasks

9/25/2017 • 7 min to read • [Edit Online](#)

The Team Data Science Process is a framework developed by Microsoft that provides a structured methodology to build predictive analytics solutions and intelligent applications efficiently. This article outlines the key personnel roles, and their associated tasks that are handled by a data science team standardizing on this process.

This introduction links to tutorials that provide instructions on how to set up the TDSP environment for the entire data science group, data science teams, and projects. We provide detailed guidance using Visual Studio Team Services (VSTS) in the tutorials as our code-hosting platform and agile planning tool to manage team tasks, control access, and manage the repositories.

You will also be able to use this information to implement TDSP on your own code-hosting and agile planning tool.

Structures of data science groups and teams

Data science functions in enterprises may often be organized in the following hierarchy:

1. ***Data science group/s***
2. ***Data science team/s within group/s***

In such a structure there will be group and team leads. Typically, a data science project is done by a data science team, which may be composed of project leads (for project management and governance tasks) and data scientists or engineers (individual contributors / technical personnel) who will execute the data science and data engineering parts of the project. Prior to execution, the setup and governance is done by the group, team or project leads.

Definition of four TDSP roles

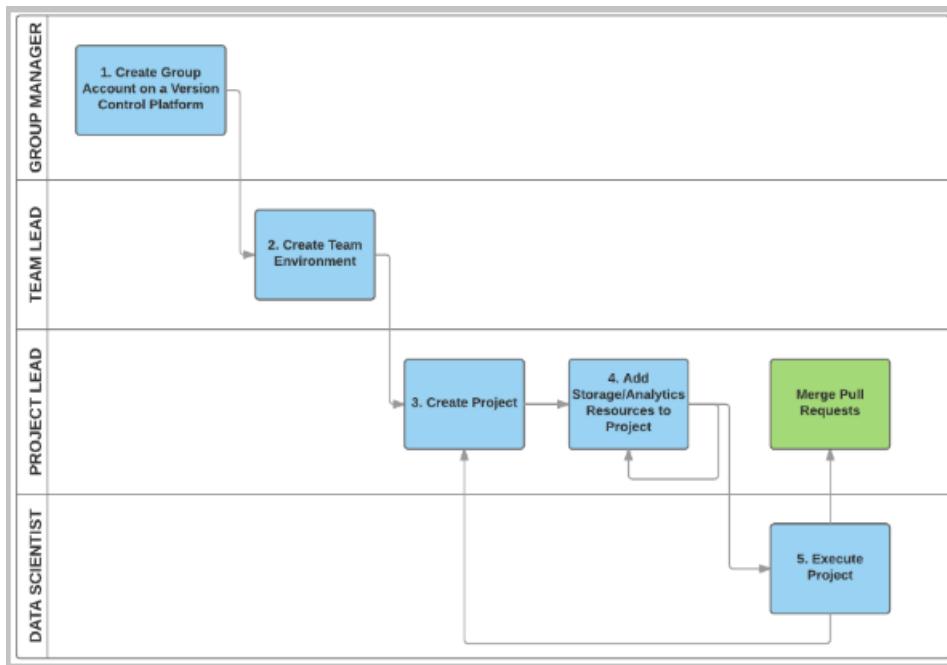
With the above assumption, we have specified four distinct roles for our team personnel:

1. ***Group Manager***. Group Manager is the manager of the entire data science unit in an enterprise. A data science unit might have multiple teams, each of which is working on multiple data science projects in distinct business verticals. A Group Manager might delegate their tasks to a surrogate, but the tasks associated with the role do not change.
2. ***Team Lead***. A team lead is managing a team in the data science unit of an enterprise. A team consists of multiple data scientists. For data science unit with only a small number of data scientists, the Group Manager and the Team Lead might be the same person.
3. ***Project Lead***. A project lead manages the daily activities of individual data scientists on a specific data science project.
4. ***Project Individual Contributor***. Data Scientist, Business Analyst, Data Engineer, Architect, etc. A project individual contributor executes a data science project.

[AZURE.NOTE]: Depending on the structure in an enterprise, a single person may play more than one roles OR there may be more than one person working on a role. This may frequently be the case in small enterprises or enterprises with a small number of personnel in their data science organization.

Tasks to be completed by four personnel

The following picture depicts the top-level tasks for personnel by role in adopting and implementing the Team Data Science Process as conceptualized by Microsoft.



This schema and the following, more detailed outline of tasks that are assigned to each role in the TDSP should help you choose the appropriate tutorial based on your responsibilities in the organization.

[AZURE.NOTE] In the following instructions, we show steps of how to set up a TDSP environment and complete other data science tasks in Visual Studio Team Services (VSTS). We specify how to accomplish these tasks with VSTS because that is what we are using to implement TDSP at Microsoft. VSTS facilitates collaboration by integrating the management of work items that track tasks and a code hosting service used to share utilities, organize versions, and provide role-based security. You are able to choose other platforms, if you prefer, to implement the tasks outlined by the TDSP. But depending on your platform, some features we leverage from VSTS may not be available.

We also use the [Data Science Virtual Machine \(DSVM\)](#) on the Azure cloud as the analytics desktop with several popular data science tools pre-configured and integrated with various Microsoft software and Azure services. You can use the DSVM or any other development environment to implement TDSP.

Group Manager tasks

The following tasks are completed by the Group Manager (or a designated TDSP system administrator) to adopt the TDSP:

- Create a **group account** on a code hosting platform (like Github, Git, VSTS, or others)
- Create a **project template repository** on the group account, and seed it from the project template repository developed by Microsoft TDSP team. The TDSP project template repository from Microsoft provides a **standardized directory structure** including directories for data, code, and documents, and provides a set of **standardized document templates** to guide an efficient data science process.
- Create a **utility repository**, and seed it from the utility repository developed by Microsoft TDSP team. The TDSP utility repository from Microsoft provides a set of useful utilities to make the work of a data scientist more efficient, including utilities for interactive data exploration, analysis, and reporting, and for baseline modeling and reporting.
- Set up the **security control policy** of these two repositories on your group account.

For detailed step-by-step instructions, see [Group Manager tasks for a data science team](#).

Team Lead tasks

The following tasks are completed by the Team Lead (or a designated team project administrator) to adopt the TDSP:

- If VSTS is selected to be the code hosting platform for versioning and collaboration, create a **team project** on the group's VSTS server. Otherwise, this task can be skipped.
- Create the **team project template repository** under the team project, and seed it from the group project template repository set up by your group manager or the delegate of the manager.
- Create the **team utility repository**, and add the team-specific utilities to the repository.
- (Optional) Create **Azure file storage** to be used to store data assets that can be useful for the entire team. Other team members can mount this shared cloud file store on their analytics desktops.
- (Optional) Mount the Azure file storage to the **Data Science Virtual Machine** (DSVM) of the team lead and add data assets on it.
- Set up the **security control** by adding team members and configure their privileges.

For detailed step-by-step instructions, see [Team Lead tasks for a data science team](#).

Project Lead tasks

The following tasks are completed by the Project Lead to adopt the TDSP:

- Create a **project repository** under the team project, and seed it from the Team project template repository.
- (Optional) Create **Azure file storage** to be used to store data assets of the project.
- (Optional) Mount the Azure file storage to the **Data Science Virtual Machine** (DSVM) of the Project Lead and add project data assets on it.
- Set up the **security control** by adding project members and configure their privileges.

For detailed step-by-step instructions, see [Project Lead tasks for a data science team](#).

Project Individual Contributor tasks

The following tasks are completed by a Project Individual Contributor (usually a Data Scientist) to conduct the data science project using the TDSP:

- Clone the **project repository** set up by the project lead.
- (Optional) Mount the shared **Azure file storage** of the team and project on their **Data Science Virtual Machine** (DSVM).
- Execute the project.

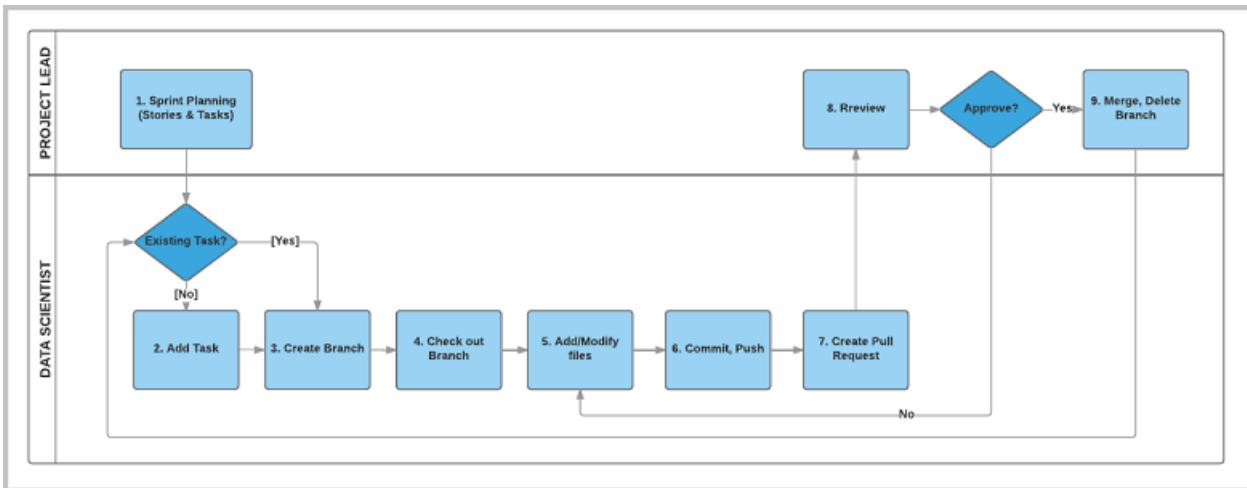
For detailed step-by-step instructions for on-boarding onto a project, see [Project Individual Contributors for a data science team](#).

Data science project execution

By following the relevant set of instructions, data scientists, project lead, and team leads can create work items to track all tasks and stages that a project needs from its beginning to its end. Using git also promotes collaboration among data scientists and ensures that the artifacts generated during project execution are version controlled and shared by all project members.

The instructions provided for project execution have been developed based on the assumption that both work items and project git repositories are on VSTS. Using VSTS for both allows you to link your work items with the Git branches of your project repositories. In this way, you can easily track what has been done for a work item.

The following figure outlines this workflow for project execution using the TDSP.



The workflow includes steps that can be grouped into three activities:

- Sprint planning (Project Lead)
- Developing artifacts on git branches to address work items (Data Scientist)
- Code review and merging branches with master branches (Project Lead or other team members)

For detailed step-by-step instructions on project execution workflow, see [Execution of data science projects](#).

Next steps

Here are links to the more detailed descriptions of the roles and tasks defined by the Team Data Science Process:

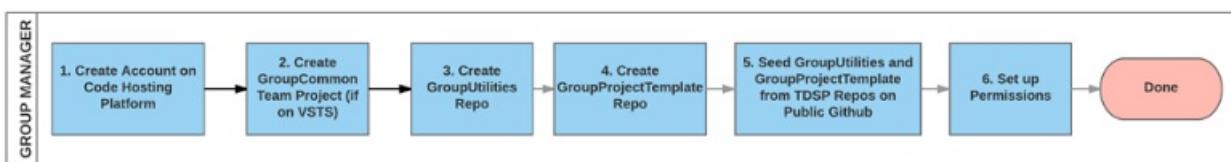
- [Group Manager tasks for a data science team](#)
- [Team Lead tasks for a data science team](#)
- [Project Lead tasks for a data science team](#)
- [Project Individual Contributors for a data science team](#)

Group Manager tasks

11/14/2017 • 11 min to read • [Edit Online](#)

This topic outlines the tasks that a Group Manager is expected to complete for his/her data science organization. The objective is to establish collaborative group environment that standardizes on the [Team Data Science Process](#) (TDSP). For an outline of the personnel roles and their associated tasks that are handled by a data science team standardizing on this process, see [Team Data Science Process roles and tasks](#).

The **Group Manager** is the manager of the entire data science unit in an enterprise. A data science unit may have multiple teams, each of which is working on multiple data science projects in distinct business verticals. A Group Manager may delegate their tasks to a surrogate, but the tasks associated with the role are the same. There are six main tasks as shown in the following diagram:



[AZURE.NOTE] We outline the steps needed to set up a TDSP group environment using VSTS in the instructions that follow. We specify how to accomplish these tasks with VSTS because that is how we implement TDSP at Microsoft. If another code hosting platform is used for your group, the tasks that need to be completed by the group manager generally do not change. But the way to complete these tasks is going to be different.

1. Set up **Visual Studio Team Services (VSTS) server** for the group.
2. Create a **group team project** on Visual Studio Team Services server (for VSTS users)
3. Create the **GroupProjectTemplate** repository
4. Create the **GroupUtilities** repository
5. Seed the **GroupProjectTemplate** and **GroupUtilities** repositories for the VSTS server with content from the TDSP repositories.
6. Set up the **security controls** for team members to access to the GroupProjectTemplate and GroupUtilities repositories.

Each of the preceding steps is described in detail. But first, we familiarize you with the abbreviations and discuss the pre-requisites for working with repositories.

Abbreviations for repositories and directories

This tutorial uses abbreviated names for repositories and directories. These definitions make it easier to follow the operations between the repositories and directories. This notation is used in the following sections:

- **G1:** The project template repository developed and managed by TDSP team of Microsoft.
- **G2:** The utilities repository developed and managed by TDSP team of Microsoft.
- **R1:** The GroupProjectTemplate repository on Git you set up on your VSTS group server.
- **R2:** The GroupUtilities repository on Git you set up on your VSTS group server.
- **LG1** and **LG2:** The local directories on your machine that you clone G1 and G2 to, respectively.
- **LR1** and **LR2:** The local directories on your machine that you clone R1 and R2 to, respectively.

Pre-requisites for cloning repositories and checking code in and out

- Git must be installed on your machine. If you are using a Data Science Virtual Machine (DSVM), Git has been

pre-installed and you are good to go. Otherwise, see the [Platforms and tools appendix](#).

- If you are using a **Windows DSVM**, you need to have [Git Credential Manager \(GCM\)](#) installed on your machine. In the README.md file, scroll down to the **Download and Install** section and click the *latest installer*. This step takes you to the latest installer page. Download the .exe installer from here and run it.
- If you are using **Linux DSVM**, create an SSH public key on your DSVM and add it to your group VSTS server. For more information about SSH, see the **Create SSH public key** section in the [Platforms and tools appendix](#).

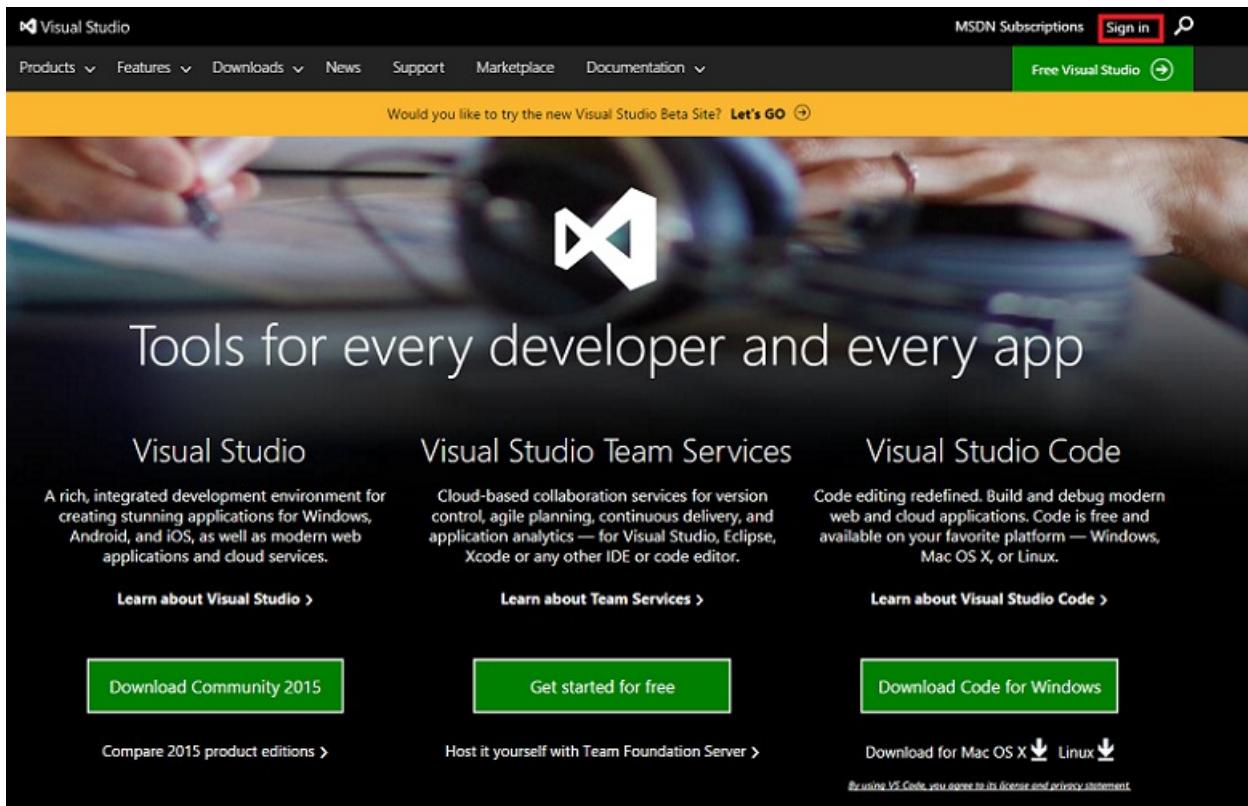
1. Create Account on VSTS server

The VSTS server hosts the following repositories:

- **group common repositories:** General-purpose repositories that can be adopted by multiple teams within a group for multiple data science projects. For example, the *GroupProjectTemplate* and *GroupUtilities* repositories.
- **team repositories:** Repositories for specific teams within a group. These repositories are specific for a team's need, and can be adopted by multiple projects executed by that team, but not general enough to be useful to multiple teams within a data science group.
- **project repositories:** Repositories available for specific projects. Such repositories may not be general enough to be useful to multiple projects performed by a team, and to multiple teams in a data science group.

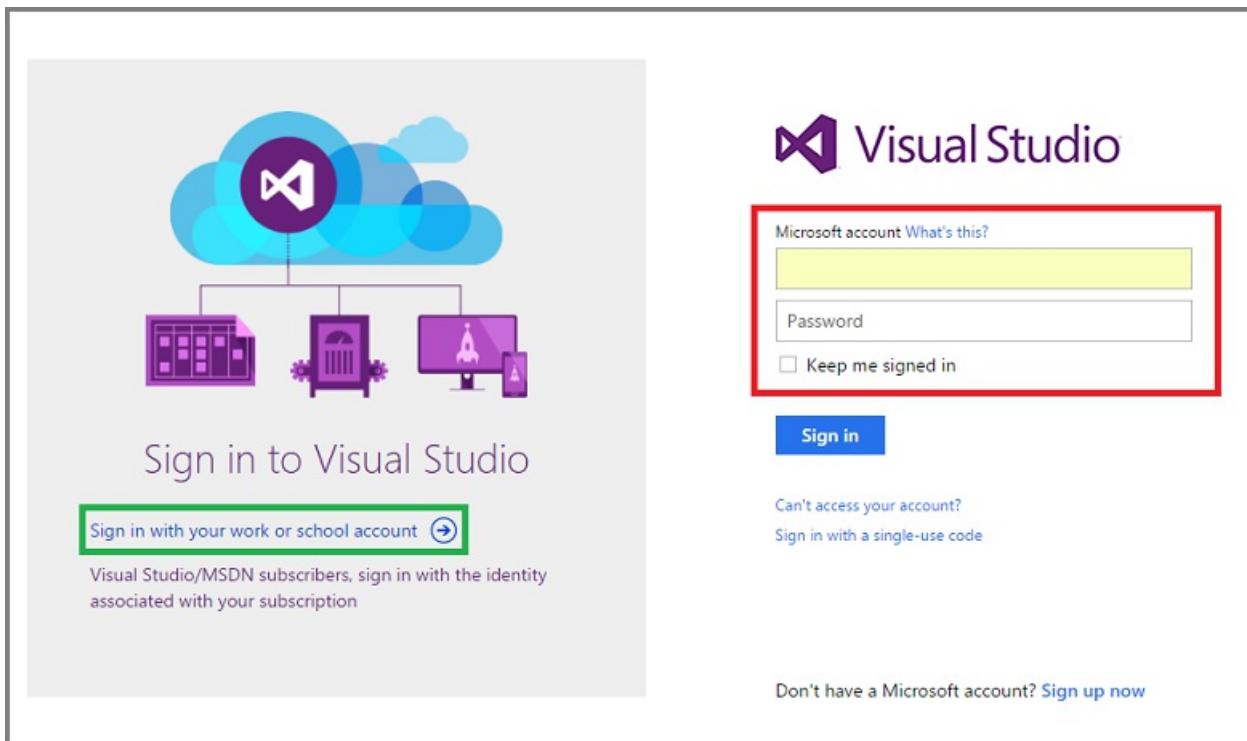
Setting up the VSTS Server Sign into your Microsoft account

Go to [Visual Studio online](#), click **Sign in** in the upper right corner and sign into your Microsoft account.

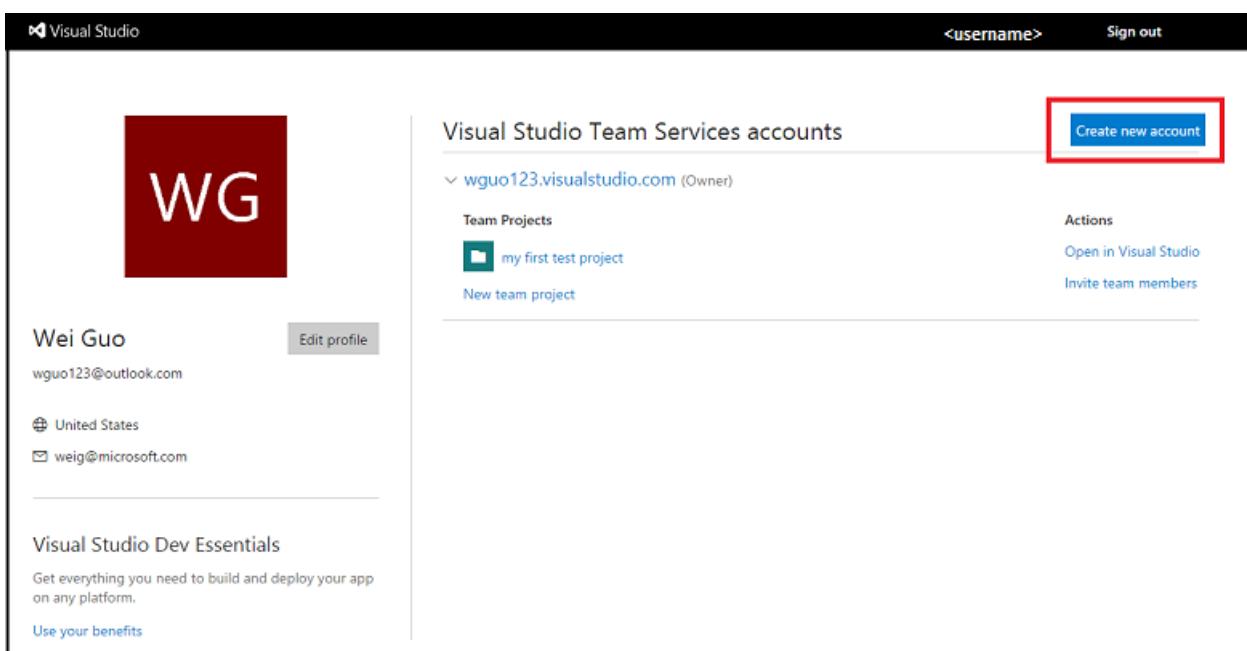


If you do not have a Microsoft account, click **Sign up now** to create a Microsoft account, and then sign in using this account.

If your organization has a Visual Studio/MSDN subscription, click the green **Sign in with your work or school account** box and sign in with the credentials associated with this subscription.



After you sign in, click **Create New Account** in the upper right corner as shown in the following image:



Fill in the information for the VSTS server that you want to create in the **Create your account** wizard with the following values:

- **Server URL:** Replace *mysamplegroup* with your own *server name*. The URL of your server is going to be: <https://<servername>.visualstudio.com>.
- **Manage code using:** Select **Git**.
- **Project name:** Enter *GroupCommon*.
- **Organize work using:** Choose *Agile*.
- **Host your projects in:** Choose a geo location. In this example, we choose *South Central US*.

You're on your way to managing projects
with Visual Studio Team Services

Create your account

mysamplegroup .visualstudio.com

Manage code using:

Git
 Team Foundation Version Control

Project name:

GroupCommon

Organize work using:

Agile

Host your projects in:

South Central US

You can share the work with other users of:

Continue

[AZURE.NOTE] If you see the following pop-up window after you click **Create new account**, then you need to click **Change details** to display all the fields itemized.

You're on your way to managing projects
with Visual Studio Team Services

Create your account

Pick a memorable name .visualstudio.com

Manage code using:

Git
 Team Foundation Version Control

We will host your projects in South Central US region.
[Change details](#)

Continue

Click **Continue**.

2. GroupCommon Team Project

The **GroupCommon** page (<https://<servername>.visualstudio.com/GroupCommon>) opens after your VSTS server is created.

The screenshot shows the 'Overview' tab of the GroupCommon page in VSTS. The left sidebar has links for 'Manage Work', 'Collaborate on code', 'Continuously integrate', and 'Visualize progress'. The main area displays a clipboard icon with a checkmark, indicating work assigned to the user. It also shows a 'New Work Item' form and a '0 Work Items' summary. On the right, there's a 'Team Members' section with a 'Work' sidebar containing 'Backlog', 'Board', 'Task board', and 'Queries' options. A 'Visual Studio' section provides links to 'Open in Visual Studio' and 'Get Visual Studio'.

3. Create the GroupUtilities (R2) repository

To create the **GroupUtilities** (R2) repository under VSTS server:

- To open the **Create a new repository** wizard, click **New repository** on the **Version Control** tab of your team project.

The screenshot shows the 'Version Control' tab in the VSTS Control Panel for the 'mysamplegroup' project. Under 'Repositories', there is a 'New repository...' button. The 'Security' section shows 'Security for all Git repositories' with a search bar and a list of identities. The 'ACCESS CONTROL SUMMARY' table lists permissions for 'Build Administrators' and other groups. Buttons for 'Remove', 'Save changes', and 'Undo changes' are at the bottom.

Identity	Permission
Administrator	Not set
Branch creation	Allow
Contribute	Allow
Exempt from policy enforcement	Not set
Note management	Allow
Read	Allow
Rewrite and destroy history (force push)	Not set
Tag creation	Allow

- Select **Git** as the **Type**, and enter *GroupUtilities* as the **Name**, and then click **Create**.

Create a new repository

Type	Name
Git	GroupUtilities
Files to include	
<input type="checkbox"/> Add a README to describe your repository <small>(?)</small>	
<input type="button" value="Create"/> <input type="button" value="Cancel"/>	

Now you should see two Git repositories **GroupProjectTemplate** and **GroupUtilities** in the left column of the **Version Control** page:

Permission	Value
Administer	Not set
Branch creation	Inherited allow
Contribute	Inherited allow
Exempt from policy enforcement	Not set
Note management	Inherited allow
Read	Inherited allow
Rewrite and destroy history (force push)	Not set
Tag creation	Inherited allow

4. Create the GroupProjectTemplate (R1) repository

The setup of the repositories for the VSTS group server consists of two tasks:

- Rename the default **GroupCommon** repository **GroupProjectTemplate**.
- Create the **GroupUtilities** repository on the VSTS server under team project **GroupCommon**.

Instructions for the first task are contained in this section after remarks on naming conventions or our repositories and directories. The instructions for the second task are contained in the following section for step 4.

Rename the default GroupCommon repository

To rename the default **GroupCommon** repository as *GroupProjectTemplate* (referred as **R1** in this tutorial):

- Click **Collaborate on code** on the **GroupCommon** team project page. This takes you to the default Git repository page of the team project **GroupCommon**. Currently, this Git repository is empty.

The screenshot shows the Visual Studio Team Services Overview dashboard for the 'GroupCommon' team project. The top navigation bar includes links for HOME, CODE, WORK, BUILD, TEST, and RELEASE. The 'Overview' tab is selected. The dashboard is divided into several sections:

- Welcome**: A section with a 'Manage Work' button (highlighted with a red box) and three other buttons: 'Collaborate on code' (highlighted with a green box), 'Continuously integrate', and 'Visualize progress'.
- Open User Stories**: A section showing a message: "Query returned no results. Create new work items or edit query to see results." It features a large '0' icon and a character holding a phone.
- Team Members**: A section with a placeholder message: "It's lonely in here..." and a "Invite a friend" button.
- Work**: A section with links for Backlog, Board, Task board, and Queries.
- Sprint Burndown**: A section with a placeholder message: "Set iteration dates to use the sprint burndown widget" and a "Set iteration dates" button.
- New Work Item**: A form to create a new work item, with fields for 'Enter title' and a dropdown menu set to 'Bug'. A 'Create' button is at the bottom.
- Open User Stories**: A section with a large '0' icon and the text 'Work items'.
- Visual Studio**: A section with two buttons: 'Open in Visual Studio' (highlighted with a red box) and 'Get Visual Studio'.

- Click **GroupCommon** on the top left corner (highlighted with a red box in the following figure) on the Git repository page of **GroupCommon** and select **Manage repositories** (highlighted with a green box in the following figure). This procedure brings up the **CONTROL PANEL**.
- Select the **Version Control** tab of your team project.

Team Services / GroupCommon

HOME CODE WORK BUILD TEST RELEASE

GroupCommon **Explorer** History Branches Pull Requests

Filter repositories

GroupCommon

+ New repository...

Manage repositories...

Empty. Add some code!

Command line or another Git client

Clone URL

HTTPS | SSH https://mysamplegroup.visualstudio.com/_git/GroupCommon

Generate Git credentials

Download Git for Windows

Plug-ins and credential managers

These provide the best experience with single sign in, multi-factor auth, and integration with your IDE.

IntelliJ IDEA Android Studio Eclipse Windows command line

Command line instructions

Clone this repository

```
git clone https://mysamplegroup.visualstudio.com/_git/GroupCommon
```

Push an existing repository

```
git remote add origin https://mysamplegroup.visualstudio.com/_git/GroupCommon
```

```
git push -u origin --all
```

- Click the ... to the right of the **GroupCommon** repository on the left panel, and select **Rename repository**.

Repositories

New repository...

Git repositories

GroupCommon

...

Security for all Git repositories

Add... Inheritance ▾

Search

VSTS Groups

- Build Administrators
- Contributors
- Project Administrators
- Readers
- Project Collection Administrators
- Project Collection Build Service Accounts
- Project Collection Service Accounts
- Users
- Project Collection Build Service (mysamplegroup)

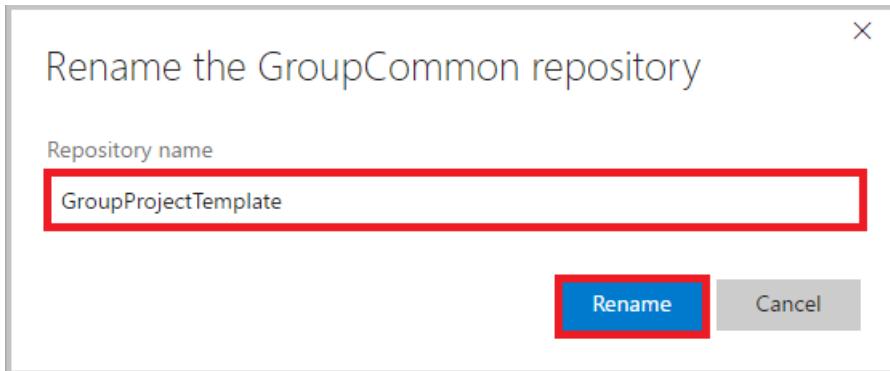
ACCESS CONTROL SUMMARY
Shows information about the permissions being granted to this identity

Role	Permission
Administrator	Not set
Branch creation	Allow
Contribute	Allow
Exempt from policy enforcement	Not set
Note management	Allow
Read	Allow
Rewrite and destroy history (force push)	Not set
Tag creation	Allow

Clear explicit permissions

Remove Save changes Undo changes

- In the **Rename the GroupCommon repository** wizard that pops up, enter *GroupProjectTemplate* in the **Repository name** box, and then click **Rename**.



5. Seed the R1 & R2 repositories on the VSTS server

In this stage of the procedure, you seed the *GroupProjectTemplate* (R1) and *GroupUtilities* (R2) repositories that you set up in the previous section. These repositories are seeded with the **ProjectTemplate (G1)** and **Utilities (G2)** repositories that are managed by Microsoft for the Team Data Science Process. When this seeding is completed:

- your R1 repository is going to have the same set of directories and document templates that the G1 does
- your R2 repository is going to contain the set of data science utilities developed by Microsoft.

The seeding procedure uses the directories on your local DSVM as intermediate staging sites. Here are the steps followed in this section:

- G1 & G2 - cloned to -> LG1 & LG2
- R1 & R2 - cloned to -> LR1 & LR2
- LG1 & LG2 - files copied into -> LR1 & LR2
- (Optional) customization of LR1 & LR2
- LR1 & LR2 - contents add to -> R1 & R2

Clone G1 & G2 repositories to your local DSVM

In this step, you clone the Team Data Science Process (TDSP) ProjectTemplate repository (G1) and Utilities (G2) from the TDSP github repositories to folders in your local DSVM as LG1 and LG2:

- Create a directory to serve as the root directory to host all your clones of the repositories.

- In the Windows DSVM, create a directory `C:\GitRepos\TDSPCommon`.
- In the Linux DSVM, create a directory `GitRepos\TDSPCommon` in your home directory.
- Run the following set of commands from the `GitRepos\TDSPCommon` directory.

```
git clone https://github.com/Azure/Azure-TDSP-ProjectTemplate
git clone https://github.com/Azure/Azure-TDSP-Utilities
```

```
PS C:\GitRepos\TDSPCommon> git clone https://github.com/Azure/Azure-TDSP-ProjectTemplate
Cloning into 'Azure-TDSP-ProjectTemplate'...
remote: Counting objects: 41, done.
remote: Total 41 (delta 0), reused 0 (delta 0), pack-reused 41
Unpacking objects: 100% (41/41), done.
PS C:\GitRepos\TDSPCommon> git clone https://github.com/Azure/Azure-TDSP-Utilities
Cloning into 'Azure-TDSP-Utilities'...
remote: Counting objects: 96, done.
remote: Compressing objects: 100% (89/89), done.
remote: Total 96 (delta 12), reused 0 (delta 0), pack-reused 6
Unpacking objects: 100% (96/96), done.
```

- Using our abbreviated repository names, this is what these scripts have achieved:
 - G1 - cloned into -> LG1
 - G2 - cloned into -> LG2
- After the cloning is completed, you should be able to see two directories, `ProjectTemplate` and `Utilities`, under `GitRepos\TDSPCommon` directory.

Clone R1 & R2 repositories to your local DSVM

In this step, you clone the GroupProjectTemplate repository (R1) and GroupUtilities repository (R2) on local directories (referred as LR1 and LR2, respectively) under `GitRepos\GroupCommon` on your DSVM.

- To get the URLs of the R1 and R2 repositories, go to your **GroupCommon** home page on VSTS. This usually has the URL `https://<Your VSTS Server Name>.visualstudio.com/GroupCommon`.
- Click **CODE**.
- Choose the **GroupProjectTemplate** and **GroupUtilities** repositories. Copy and save each of the URLs (HTTPS for Windows; SSH for Linux) from the **Clone URL** element, in turn, for use in the following scripts:

The screenshot shows the VSTS GroupCommon repository page. At the top, there's a navigation bar with links for HOME, CODE, WORK, BUILD, TEST, and RELEASE. The CODE link is highlighted. Below the navigation, there's a dropdown menu for 'GroupProjectTemplate' and tabs for Explorer, History, Branches, and Pull Requests. The 'Explorer' tab is selected. A message says 'GroupProjectTemplate is empty. Add some code!'. Below that, there's a section for 'Clone in your favorite IDE' with a 'Clone in Visual Studio' button. Under 'Command line or another Git client', there's a 'Clone URL' field containing 'HTTPS | SSH https://weig-ds.visualstudio.com/GroupCommon/_git/Group..'. There's also a 'Generate Git credentials' button and a 'Download Git for Windows' link.

- Change into the `GitRepos\GroupCommon` directory on your Windows or Linux DSVM and run one of the following sets of commands to clone R1 and R2 into that directory.

Here are the Windows and Linux scripts:

```
# Windows DSVM

git clone <the HTTPS URL of the GroupProjectTemplate repository>
git clone <the HTTPS URL of the GroupUtilities repository>
```

```
PS D:\AML_Projects\TDSP-Linux\Tutorial_testing\test2> git clone https://weig-ds.visualstudio.com/GroupCommon/_git/GroupProjectTemplate
Cloning into 'GroupProjectTemplate'...
warning: You appear to have cloned an empty repository.
Checking connectivity... done.
PS D:\AML_Projects\TDSP-Linux\Tutorial_testing\test2> git clone https://weig-ds.visualstudio.com/GroupCommon/_git/GroupUtilities
Cloning into 'GroupUtilities'...
warning: You appear to have cloned an empty repository.
Checking connectivity... done.
```

```
# Linux DSVM

git clone <the SSH URL of the GroupProjectTemplate repository>
git clone <the SSH URL of the GroupUtilities repository>
```

```
[ds1@weiglinuxdsvml test2]$ git clone ssh://weig-ds@weig-ds.visualstudio.com:22/GroupCommon/_git/GroupProjectTemplate
Cloning into 'GroupProjectTemplate'...
warning: You appear to have cloned an empty repository.
[ds1@weiglinuxdsvml test2]$ git clone ssh://weig-ds@weig-ds.visualstudio.com:22/GroupCommon/_git/GroupUtilities
Cloning into 'GroupUtilities'...
warning: You appear to have cloned an empty repository.
```

[AZURE.NOTE] Expect to receive warning messages that LR1 and LR2 are empty.

- Using our abbreviated repository names, this is what these scripts have achieved:
 - R1 - cloned into -> LR1
 - R2 - cloned into -> LR2

Seed your GroupProjectTemplate (LR1) and GroupUtilities (LR2)

Next, in your local machine, copy the content of ProjectTemplate and Utilities directories (except the metadata in the .git directories) under GitRepos\TDSPCommon to your GroupProjectTemplate and GroupUtilities directories under **GitRepos\GroupCommon**. Here are the two tasks to complete in this step:

- Copy the files in GitRepos\TDSPCommon\ProjectTemplate (**LG1**) to GitRepos\GroupCommon\GroupProjectTemplate (**LR1**)
- Copy the files in GitRepos\TDSPCommon\Utilities (**LG2**) to GitRepos\GroupCommon\Utilities (**LR2**).

To achieve these two tasks, run the following scripts in PowerShell console (Windows) or Shell script console (Linux). You are prompted to input the complete paths to LG1, LR1, LG2, and LR2. The paths that you input are validated. If you input a directory that does not exist, you are asked to input it again.

```
# Windows DSVM

wget "https://raw.githubusercontent.com/Azure/Azure-MachineLearning-
DataScience/master/Misc/TDSP/tdsp_local_copy_win.ps1" -outfile "tdsp_local_copy_win.ps1"
.\tdsp_local_copy_win.ps1 1
```

```
PS D:\AML_Projects\TDSP-Linux\test0912> wget "https://raw.githubusercontent.com/Azure/Azure-MachineLearning-DataScience/master/Misc/TDSP/tdsp_local_co
py_win.ps1" -outfile "tdsp_local_copy_win.ps1"
PS D:\AML_Projects\TDSP-Linux\test0912> .\tdsp_local_copy_win.ps1 1
Please input the full path to ProjectTemplate repository from Microsoft TDSP team (source directory): D:\AML_Projects\TDSP-Linux\test0912\ProjectTempl
ate
Please input the full path to Your GroupProjectTemplate repository (destination directory): D:\AML_Projects\TDSP-Linux\test0912\GroupProjectTemplate
Start copying files (except files in .git directory) from D:\AML_Projects\TDSP-Linux\test0912\ProjectTemplate to D:\AML_Projects\TDSP-Linux\test0912\G
roupProjectTemplate...
Please input the full path to Utilities repository from Microsoft TDSP team (source directory): D:\AML_Projects\TDSP-Linux\test0912\Utilities
Please input the full path to Your GroupUtilities repository (destination directory): D:\AML_Projects\TDSP-Linux\test0912\GroupUtilities
Start copying files (except files in .git directory) from D:\AML_Projects\TDSP-Linux\test0912\Utilities to D:\AML_Projects\TDSP-Linux\test0912\GroupU
tilities...
PS D:\AML_Projects\TDSP-Linux\test0912>
```

Now you can see that files in directories LG1 and LG1 (except files in the .git directory) have been copied to LR1

and LR2, respectively.

```
PS D:\AML_Projects\TDSP-Linux\Tutorial_testing> $src = "ProjectTemplate"
PS D:\AML_Projects\TDSP-Linux\Tutorial_testing> $dest = "GroupProjectTemplate"
PS D:\AML_Projects\TDSP-Linux\Tutorial_testing> $SourceDirectory = $PWD.Path+"\\"+$src
PS D:\AML_Projects\TDSP-Linux\Tutorial_testing> $DestinationDirectory = $PWD.Path+"\\"+$dest
PS D:\AML_Projects\TDSP-Linux\Tutorial_testing> $ExcludeSubDirectory = $SourceDirectory+'\.git'
PS D:\AML_Projects\TDSP-Linux\Tutorial_testing> $files = Get-ChildItem $SourceDirectory -Recurse | Where-Object { $ExcludeSubDirectory -notcontains $_.DirectoryName }
PS D:\AML_Projects\TDSP-Linux\Tutorial_testing>
PS D:\AML_Projects\TDSP-Linux\Tutorial_testing> foreach ($file in $files)
>> {
>>     $CopyPath = Join-Path $DestinationDirectory $file.FullName.Substring($SourceDirectory.length)
>>     Copy-Item $file.FullName -Destination $CopyPath
>> }
>>
PS D:\AML_Projects\TDSP-Linux\Tutorial_testing>
PS D:\AML_Projects\TDSP-Linux\Tutorial_testing>
PS D:\AML_Projects\TDSP-Linux\Tutorial_testing>
PS D:\AML_Projects\TDSP-Linux\Tutorial_testing> $src = "Utilities"
PS D:\AML_Projects\TDSP-Linux\Tutorial_testing> $dest = "GroupUtilities"
PS D:\AML_Projects\TDSP-Linux\Tutorial_testing> $SourceDirectory = $PWD.Path+"\\"+$src
PS D:\AML_Projects\TDSP-Linux\Tutorial_testing> $DestinationDirectory = $PWD.Path+"\\"+$dest
PS D:\AML_Projects\TDSP-Linux\Tutorial_testing> $ExcludeSubDirectory = $SourceDirectory+'\.git'
PS D:\AML_Projects\TDSP-Linux\Tutorial_testing> $files = Get-ChildItem $SourceDirectory -Recurse | Where-Object { $ExcludeSubDirectory -notcontains $_.DirectoryName }
PS D:\AML_Projects\TDSP-Linux\Tutorial_testing>
PS D:\AML_Projects\TDSP-Linux\Tutorial_testing> foreach ($file in $files)
>> {
>>     $CopyPath = Join-Path $DestinationDirectory $file.FullName.Substring($SourceDirectory.length)
>>     Copy-Item $file.FullName -Destination $CopyPath
>> }
>>
PS D:\AML_Projects\TDSP-Linux\Tutorial_testing>
```

```
# Linux DSVM

wget "https://raw.githubusercontent.com/Azure/Azure-MachineLearning-
DataScience/master/Misc/TDSP/tdsp_local_copy_linux.sh"
bash tdsp_local_copy_linux.sh 1
```

```
[ds1@linuxdsvm6 test0912]$ wget "https://raw.githubusercontent.com/Azure/Azure-MachineLearning-
>DataScience/master/Misc/TDSP/tdsp_local_copy_linux.sh"
--2016-09-12 16:32:41-- https://raw.githubusercontent.com/Azure/Azure-MachineLearning-
>DataScience/master/Misc/TDSP/tdsp_local_copy_linux.sh
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.48.133
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.48.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2057 (2.0K) [text/plain]
Saving to: 'tdsp_local_copy_linux.sh'

100%[=====] 2,057 --> 2,057      0.0K/s   in 0s

[ds1@linuxdsvm6 test0912]$ bash tdsp_local_copy_linux.sh
Please input the full path to ProjectTemplate repository from Microsoft TDSP team (source directory): /home/ds1/test0912/ProjectTemplate
Please input the full path to Your GroupProjectTemplate repository (destination directory): /home/ds1/test0912/GroupProjectTemplate
Code/
Code/DataPrep/
Code/DataPrep/dataPrep.py
Code/Model/
Code/Model/model.R
Code/Operationalization/
Code/Operationalization/operationalization.py
Data/
```

Now you see that the files in the two folders (except files in the .git directory) are copied to GroupProjectTemplate and GroupUtilities respectively.

```
[dsl@weiglinuxdsvm ~]$ src="ProjectTemplate"
[dsl@weiglinuxdsvm ~]$ dest="GroupProjectTemplate"
[dsl@weiglinuxdsvm ~]$ SourceDirectory=$PWD/$src
[dsl@weiglinuxdsvm ~]$ DestinationDirectory=$PWD/$dest
[dsl@weiglinuxdsvm ~]$ cd $SourceDirectory
[dsl@weiglinuxdsvm ProjectTemplate]$ git archive HEAD --format=tar | (cd $DestinationDirectory; tar xvf -)
Code/
Code/DataPrep/
Code/DataPrep/dataPrep.py
Code/Model/
Code/Model/model.R
Code/Operationalization/
Code/Operationalization/operationalization.py
Data/
Data/Modeling/
Data/Modeling/modelling.md
Data/Processed/
Data/Processed/processed.md
Data/Raw/
Data/Raw/rawData.md
Docs/
Docs/DataDictionaries/
Docs/DataDictionaries/ReadMe.md
Docs/DataDictionaries/dict1.csv
Docs/DataReport/
Docs/DataReport/Data Defintion.md
Docs/DataReport/DataPipeline.txt
Docs/DataReport/DataQualityReport.md
Docs/DataReport/ReadMe.md
Docs/Model/
Docs/Model/Baseline/
Docs/Model/Baseline/Baseline Models.md
Docs/Model/FinalReport.md
Docs/Model/Model 1/
Docs/Model/Model 1/Model Report.md
Docs/Project/
Docs/Project/Charter.md
Docs/Project/Exit Report.md
Docs/Project/Glossary.md
Docs/Project/System Architecture.docx
Docs/Project/UseCases.md
README.md
```

- Using our abbreviated repository names, this is what these scripts have achieved:
 - LG1 - files copied into -> LR1
 - LG2 - files copied into -> LR2

Option to customize the contents of LR1 & LR2

If you want to customize the contents of LR1 and LR2 to meet the specific needs of your group, this is the stage of the procedure where that is appropriate. You can modify the template documents, change the directory structure, and add existing utilities that your group has developed or that are helpful for your entire group.

Add the contents in LR1 & LR2 to R1 & R2 on group server

Now, you need to add the contents in LR1 and LR2 to repositories R1 and R2. Here are the git bash commands you can run in either Windows PowerShell or Linux.

Run the following commands from the GitRepos\GroupCommon\GroupProjectTemplate directory:

```
git status
git add .
git commit -m"push from DSVM"
git push
```

The -m option lets you set a message for your git commit.

```

PS D:\AML_Projects\TDSP-Linux\Tutorial_testing\test2\GroupProjectTemplate> git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    Code/
    Data/
    Docs/
    README.md

nothing added to commit but untracked files present (use "git add" to track)
PS D:\AML_Projects\TDSP-Linux\Tutorial_testing\test2\GroupProjectTemplate> git add .
PS D:\AML_Projects\TDSP-Linux\Tutorial_testing\test2\GroupProjectTemplate> git commit -m"push from local Win DSVM"
[master (root-commit) 4170390] push from local Win DSVM
 21 files changed, 233 insertions(+)
 create mode 100644 Code/DataPrep/dataPrep.py
 create mode 100644 Code/Model/model.R
 create mode 100644 Code/Operationalization/operationalization.py
 create mode 100644 Data/Modeling/modelling.md
 create mode 100644 Data/Processed/processed.md
 create mode 100644 Data/Raw/rawData.md
 create mode 100644 Docs/DataDictionaries/ReadMe.md

```

You can see that in your group's VSTS server, in the GroupProjectTemplate repository, the files are synced instantly.

Name	Last Change	Comments
Code	a minute ago	push from local Win DSVM - Wei Guo
Data	a minute ago	push from local Win DSVM - Wei Guo
Docs	a minute ago	push from local Win DSVM - Wei Guo
README.md	a minute ago	push from local Win DSVM - Wei Guo

Finally, change to the **GitRepos\GroupCommon\GroupUtilities** directory and run the same set of git bash commands:

```

git status
git add .
git commit -m"push from DSVM"
git push

```

[AZURE.NOTE] If this is the first time you commit to a Git repository, you need to configure global parameters *user.name* and *user.email* before you run the `git commit` command. Run the following two commands:

```

git config --global user.name <your name>
git config --global user.email <your email address>

```

If you are committing to multiple Git repositories, use the same name and email address when you commit to each of them. Using the same name and email address proves convenient later on when you build PowerBI dashboards to track your Git activities on multiple repositories.

- Using our abbreviated repository names, this is what these scripts have achieved:
 - LR1 - contents add to -> R1
 - LR2 - contents add to -> R2

6. Add group members to the group server

From your group VSTS server's homepage, click the **gear icon** next to your user name in the upper right corner, then select the **Security** tab. You can add members to your group here with various permissions.

The screenshot shows the VSTS Control Panel with the Security tab selected. On the left, there is a search bar labeled "Create VSTS group" and a dropdown menu showing a result for "yuso". The main area displays the "Project Collection Administrators" group with its permissions. The "Permissions" section lists numerous administrative tasks, many of which are highlighted in yellow, indicating they cannot be modified. The "Members" section shows the user "yuso" listed. The "Member of" section shows the group is part of the "Administrator group". At the bottom, there are "Save changes" and "Undo changes" buttons.

Permission	Allow
Administer build resource permissions	Allow
Administer process permissions	Allow
Administer Project Server integration	Allow
Administer shelved changes	Allow
Administer workspaces	Allow
Alter trace settings	Allow
Create a workspace	Allow
Create new projects	Allow
Create process	Allow
Delete process	Allow
Delete team project	Allow
Edit collection-level information	Allow
Edit process	Allow
Make requests on behalf of others	Not set
Manage build resources	Allow
Manage test controllers	Allow
Trigger events	Allow
Use build resources	Allow
View build resources	Allow
View collection-level information	Allow
View system synchronization information	Allow

Next steps

Here are links to the more detailed descriptions of the roles and tasks defined by the Team Data Science Process:

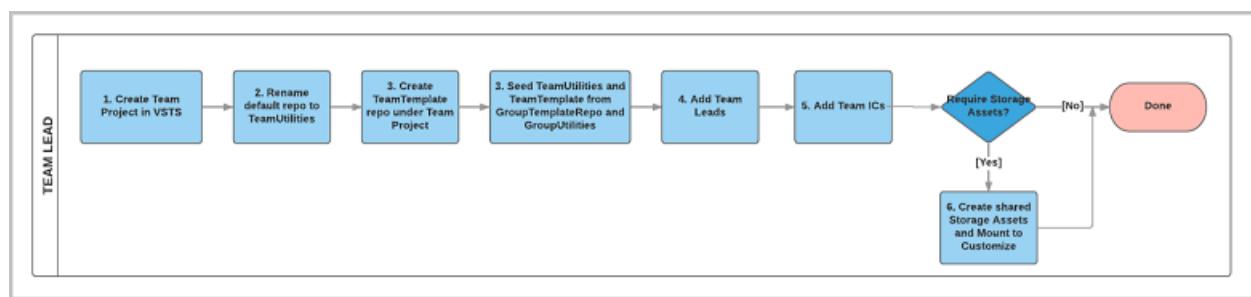
- [Group Manager tasks for a data science team](#)
- [Team Lead tasks for a data science team](#)
- [Project Lead tasks for a data science team](#)
- [Project Individual Contributors for a data science team](#)

Team Lead tasks

11/14/2017 • 16 min to read • [Edit Online](#)

This topic outlines the tasks that a team lead is expected to complete for their data science team. The objective is to establish collaborative team environment that standardizes on the [Team Data Science Process](#) (TDSP). TDSP is an agile, iterative data science methodology to deliver predictive analytics solutions and intelligent applications efficiently. It is designed to help improve collaboration and team learning. The process is a distillation of the best practices and structures from both Microsoft as well as from the industry, needed for successful implementation of data science initiatives to help companies fully realize the benefits of their analytics programs. For an outline of the personnel roles and their associated tasks that are handled by a data science team standardizing on this process, see [Team Data Science Process roles and tasks](#).

A **Team Lead** manages a team in the data science unit of an enterprise. A team consists of multiple data scientists. For data science unit with only a small number of data scientists, the **Group Manager** and the **Team Lead** might be the same person or they could delegate their task to a surrogate. But the tasks themselves do not change. The workflow for the tasks to be completed by team leads to set up this environment are depicted in the following figure:



[AZURE.NOTE] The tasks in blocks 1 and 2 of the figure are needed if you are using Visual Studio Team Services (VSTS) as the code hosting platform and you want to have a separate team project for your own team. Once these tasks are completed, all repositories of your team can be created under this team project.

After several prerequisites tasks specified in a following section are satisfied by the group manager, there are the five principal tasks (some optional) that you complete in this tutorial. These tasks correspond the main numbered sections of this topic:

1. Create a **team project** on the group's VSTS server of the group and two team repositories in the project:
 - **ProjectTemplate repository**
 - **TeamUtilities repository**
2. Seed the team **ProjectTemplate** repository from the **GroupProjectTemplate** repository which has been set up by your group manager.
3. Create team data and analytics resources:
 - Add the team-specific utilities to the **TeamUtilities** repository.
 - (Optional) Create an **Azure file storage** to be used to store data assets that can be useful for the entire team.
4. (Optional) Mount the Azure file storage to the **Data Science Virtual Machine** (DSVM) of the team lead and add data assets on it.
5. Set up the **security control** by adding team members and configure their privileges.

[AZURE.NOTE] We outline the steps needed to set up a TDSP team environment using VSTS in the following

instructions. We specify how to accomplish these tasks with VSTS because that is how we implement TDSP at Microsoft. If another code hosting platform is used for your group, the tasks that need to be completed by the team lead generally do not change. But the way to complete these tasks is going to be different.

Repositories and directories

This topic uses abbreviated names for repositories and directories. These names make it easier to follow the operations between the repositories and directories. This notation (**R** for Git repositories and **D** for local directories on your DSVM) is used in the following sections:

- **R1:** The **GroupProjectTemplate** repository on Git that your group manager set up on your VSTS group server.
- **R3:** The team **ProjectTemplate** repository on Git you set up.
- **R4:** The **TeamUtilities** repository on Git you set up.
- **D1:** The local directory cloned from R1 and copied to D3.
- **D3:** The local directory cloned from R3, customize, and copied back to R3.
- **D4:** The local directory cloned from R4, customize, and copied back to R4.

The names specified for the repositories and directories in this tutorial have been provided on the assumption that your objective is to establish a separate team project for your own team within a larger data science group. But there are other options open to you as team lead:

- The entire group can choose to create a single team project. Then all projects from all data science teams would be under this single team project. To achieve this, you can designate a git administrator to follow these instructions to create a single team project. This scenario might be valid, for example, for:
 - a small data science group that does not have multiple data science teams
 - a larger data science group with multiple data science teams that nevertheless wants to optimize inter-team collaboration with activities such as group-level sprint planning.
- Teams can choose to have team-specific project templates or team-specific utilities under the single team project for the entire group. In this case, the team leads should create team project template repositories and/or team utilities repositories under the same team project. Name these repositories *<TeamName>ProjectTemplate* and *<TeamName>Utilities*, for instance, *TeamJohnProjectTemplate* and *TeamJohnUtilities*.

In any case, team leads need to let their team members know which template and utilities repositories to adopt when they are setting up and cloning the project and utilities repositories. Project leads should follow the [Project Lead tasks for a data science team](#) to create project repositories, whether under separate team projects or under a single team project.

0. Prerequisites

The prerequisites are satisfied by completing the tasks assigned to your group manager outlined in [Group Manager tasks for a data science team](#). To summarize here, the following requirements need to meet before you begin the team lead tasks:

- Your **group VSTS server** (or group account on some other code hosting platform) has been set up by your group manager.
- Your **GroupProjectTemplate repository** (R1) has been set up on your group account by your group manager on the code hosting platform you plan to use.
- You have been **authorized** on your group account to create repositories for your team.
- Git must be installed on your machine. If you are using a Data Science Virtual Machine (DSVM), Git has been pre-installed and you are good to go. Otherwise, see the [Platforms and tools appendix](#).
- If you are using a **Windows DSVM**, you need to have [Git Credential Manager \(GCM\)](#) installed on your

machine. In the README.md file, scroll down to the **Download and Install** section and click the *latest installer*. This takes you to the latest installer page. Download the .exe installer from here and run it.

- If you are using **Linux DSVM**, create an SSH public key on your DSVM and add it to your group VSTS server. For more information about SSH, see the **Create SSH public key** section in the [Platforms and tools appendix](#).

1. Create a team project and repositories

Complete this step if you are using VSTS as your code hosting platform for versioning and collaboration. This section has you create three artifacts in the VSTS server of your group:

- **MyTeam** project in VSTS
- **MyProjectTemplate** repository (**R3**) on Git
- **MyTeamUtilities** repository (**R4**) on Git

Create the MyTeam project

- Go to your group's VSTS server homepage at URL <https://<VSTS Server Name>.visualstudio.com>.
- Click **New** to create a team project.

The screenshot shows the 'Overview' page of a VSTS group. At the top, there are sections for 'About Visual Studio Team Services' (Features, Pricing, Learn, Get Visual Studio) and 'Visualize data across your enterprise' (using Power BI). Below this, there are sections for 'Recent projects & teams' (listing 'GroupCommon' with a '21 hours ago' update), 'Recent team rooms' (with a link to the Rooms hub), and 'Enjoy free benefits' (listing 'Download software, use Azure services, view online training and more' with icons for Jenkins, Microsoft Azure, and Pluralsight). On the right side, there is a 'News' sidebar with links to 'Release, test and Git/TFVC history view i...', 'Resize charts, PR comment tracking, RM...', and 'Git & TFVC updates plus improved test tr...'. The main content area below the sidebar is currently empty.

- A Create team project window asks you to input the Project name (**MyTeam** in this example). Make sure that you select **Agile** as the **Process template** and **Git** as the **Version control**.

Create team project

Project name
MyTeam

Description

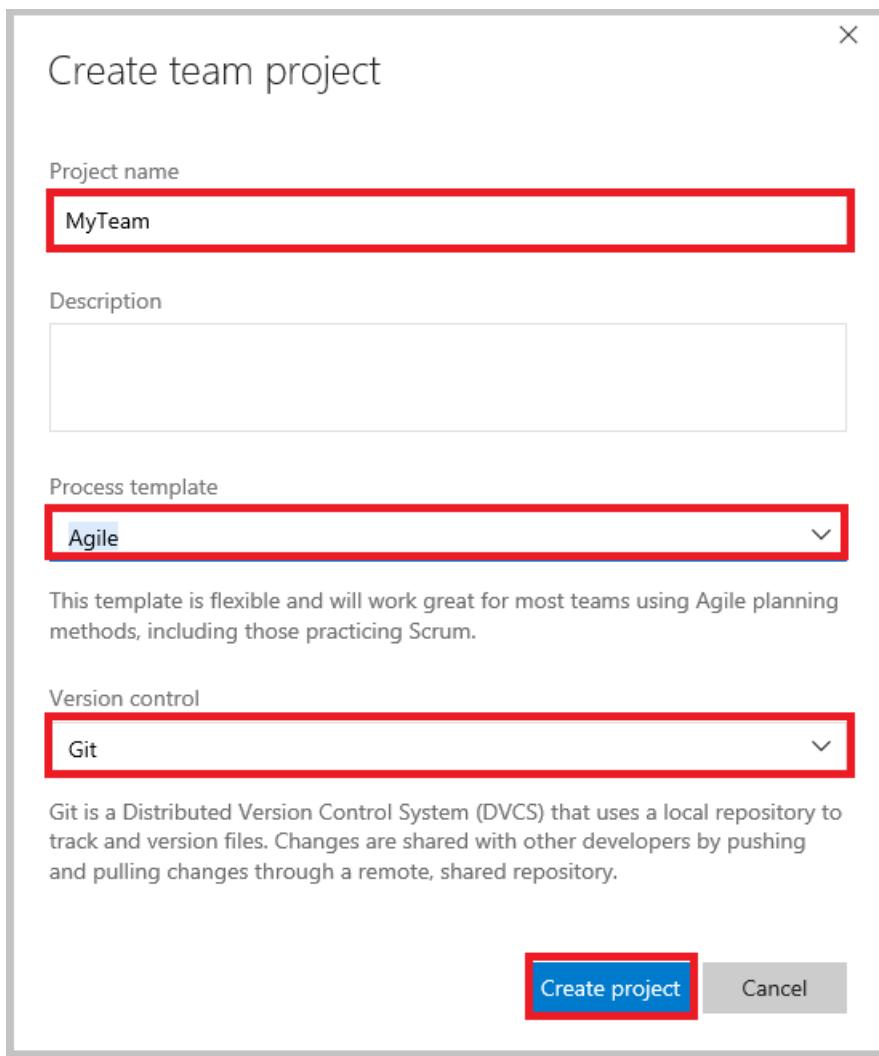
Process template
Agile

This template is flexible and will work great for most teams using Agile planning methods, including those practicing Scrum.

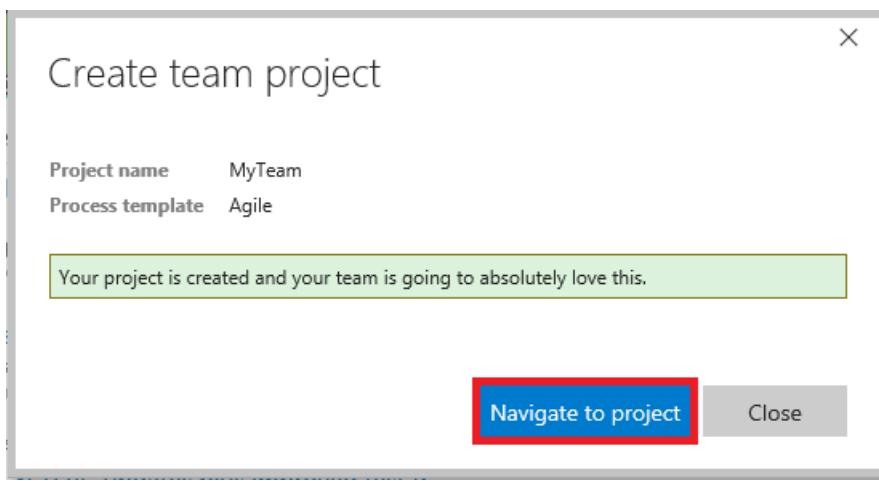
Version control
Git

Git is a Distributed Version Control System (DVCS) that uses a local repository to track and version files. Changes are shared with other developers by pushing and pulling changes through a remote, shared repository.

Create project Cancel



- Click **Create project**. Your team project **MyTeam** is created in less than 1 minute.
- After the team project **MyTeam** is created, click **Navigate to project** button, to be directed to the home page of your team project.



- If you see a **Congratulations!** popup window, click the **Add code** (button in red box). Otherwise, click **Code** (in yellow box). This directs you to the Git repository page of your team project.

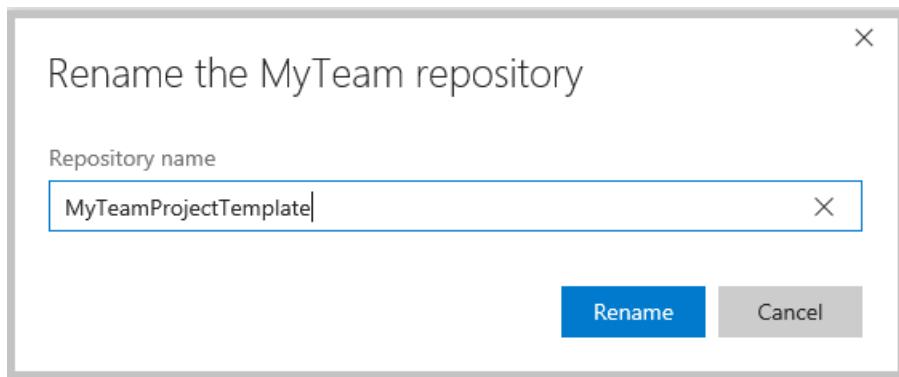
Create the MyProjectTemplate repository (R3) on Git

- On the Git repository page of your team project, click the downward arrow beside repository name **MyTeam**, and select **Manage repositories....**

- On the **Version control** tab of the control panel of your team project, click **MyTeam**, then select **Rename repository....**

The screenshot shows the 'Version Control' tab selected in the top navigation bar. On the left, under 'Repositories', there is a 'Git repositories' section with 'MyTeam' listed. The 'MyTeam' entry is highlighted with a red box. To the right, a detailed view of the access control summary is shown, listing various permissions like 'Administer', 'Branch creation', etc., with their respective inheritance status.

- Input a new name to the repository in the **Rename the MyTeam repository** window. In this example, *MyTeamProjectTemplate*. You can choose something like <*Your team name*>*ProjectTemplate*. Click **Rename** to continue.

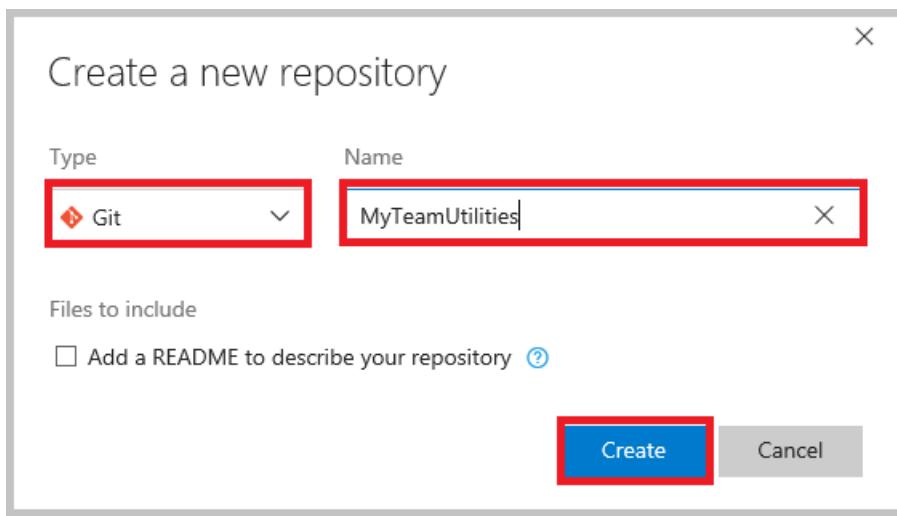


Create the MyTeamUtilities repository (R4) on Git

- To create a new repository <*your team name*>*Utilities* under your team project, click **New repository...** on the **Version control** tab of your team project's control panel.

The screenshot shows the 'Version Control' tab selected in the top navigation bar. On the left, under 'Repositories', there is a 'New repository...' button highlighted with a red box. Below it, 'Git repositories' and 'MyTeamProjectTemplate' are listed. To the right, a detailed view of the access control summary is shown, listing various permissions like 'Administer', 'Branch creation', etc., with their respective inheritance status.

- In the **Create a new repository** window that pops up, provide a name for this repository. In this example, we name it as *MyTeamUtilities*, which is **R4** in our notation. Choose something like <*your team name*>*Utilities*. Make sure that you select **Git** for **Type**. Then, click **Create** to continue.



- Confirm that you see the two new Git repositories created under your team project **MyTeam**. In this example:
- MyTeamProjectTemplate** (R3)
- MyTeamUtilities** (R4).

Access Control Summary	Description
Administer	Inherited allow
Branch creation	Inherited allow
Contribute	Inherited allow
Exempt from policy enforcement	Not set
Note management	Inherited allow
Read	Inherited allow
Rewrite and destroy history (force push)	Inherited allow
Tag creation	Inherited allow

2. Seed your team ProjectTemplate and TeamUtilities repositories

The seeding procedure uses the directories on your local DSVM as intermediate staging sites. If you need to customize your **ProjectTemplate** and **TeamUtilities** repositories to meet some specific team needs, you do so in the penultimate step of following procedure. Here is a summary of the steps used to seed the content of the **MyTeamProjectTemplate** and **MyTeamUtilities** repositories for a data science team. The individual steps correspond to the subsections in the seeding procedure:

- Clone group repository into local directory: team R1 - cloned to -> local D1
- Clone your team repositories into local directories: team R3 & R4 - cloned to -> local D3 & D4
- Copy the group project template content to the local team folder: D1 - contents copied to -> D3
- (Optional) customization of local D3 & D4
- Push local directory content to team repositories: D3 & D4 - contents add to -> team R3 & R4

Initialize the team repositories

In this step, you initialize your team project template repository from the group project template repository:

- **MyTeamProjectTemplate** repository (**R3**) from your **GroupProjectTemplate** (**R1**) repository

Clone group repositories into local directories

To begin this procedure:

- Create directories on your local machine:
 - For **Windows**: **C:\GitRepos\GroupCommon** and **C:\GitRepos\MyTeam**
 - For **Linux**: **GitRepos\GroupCommon** and **GitRepos\MyTeam** on your home directory
- Change to directory **GitRepos\GroupCommon**.
- Run the following command, as appropriate, on the operating system of your local machine.

Windows

```
git clone https://<Your VSTS Server name>.visualstudio.com/GroupCommon/_git/GroupProjectTemplate
```

```
PS D:\AML_Projects\TDSP-Linux\Tutorial_testing\test_team> git clone https://weig-ds.visualstudio.com/GroupCommon/_git/GroupProjectTemplate
Cloning into 'GroupProjectTemplate'...
remote:
remote:           vSTS
remote:           vSTSVSTSV
remote:           vSTSVSTSVST
remote: VSTS           vSTSVSTSVSTSV
remote: VSTSVS          vSTSVSTSV STSVS
remote: VTSVSTSVsTSVSTSVS   TSVST
remote: VS  tSVSTSVSTv       STSVS
remote: VS  tSVSTSVST        SVSTS
remote: VS  tSVSTSVSTv       STSVS
remote: VSTSVST          SVSTSVSTs VSTSV
remote: VSTSv            STSVSTSVSTSV
remote:           VSTSVSTSVST
remote:           VSTSVSTs
remote:           VSTS     (TM)
remote:
remote: Microsoft (R) Visual Studio (R) Team Services
remote:
Unpacking objects: 100% (34/34), done.
Checking connectivity... done.
```

Linux

```
git clone ssh://<Your VSTS Server name>@<Your VSTS Server
name>.visualstudio.com:22/GroupCommon/_git/GroupProjectTemplate
```

```
[ds1@weiglinuxdsvml test_team]$ git clone ssh://weig-ds@weig-ds.visualstudio.com:22/GroupCommon/_git/GroupProjectTemplate
Cloning into 'GroupProjectTemplate'...
remote:
remote:           vSTS
remote:           vSTSVSTSV
remote:           vSTSVSTSVST
remote: VSTS           vSTSVSTSVSTSV
remote: VSTSVS          vSTSVSTSV STSVS
remote: VTSVSTSVsTSVSTSVS   TSVST
remote: VS  tSVSTSVSTv       STSVS
remote: VS  tSVSTSVST        SVSTS
remote: VSTSVST          SVSTSVSTs VSTSV
remote: VSTSv            STSVSTSVSTSV
remote:           VSTSVSTSVST
remote:           VSTSVSTs
remote:           VSTS     (TM)
remote:
remote: Microsoft (R) Visual Studio (R) Team Services
remote:
Receiving objects: 100% (34/34), 14.65 KiB | 0 bytes/s, done.
Resolving deltas: 100% (2/2), done.
```

These commands clone your **GroupProjectTemplate** (**R1**) repository on your group VSTS server to local directory in **GitRepos\GroupCommon** on your local machine. After cloning, directory **GroupProjectTemplate** (**D1**) is created in directory **GitRepos\GroupCommon**. Here, we assume that your group manager created a team project **GroupCommon**, and the **GroupProjectTemplate** repository is under this team project.

Clone your team repositories into local directories

These commands clone your **MyTeamProjectTemplate** (**R3**) and **MyTeamUtilities** (**R4**) repositories under your team project **MyTeam** on your group VSTS server to the **MyTeamProjectTemplate** (**D3**) and

MyTeamUtilities (D4) directories in **GitRepos\MyTeam** on your local machine.

- Change to directory **GitRepos\MyTeam**
- Run the following commands, as appropriate, on the operating system of your local machine.

Windows

```
git clone https://<Your VSTS Server name>.visualstudio.com/<Your Team Name>/_git/MyTeamProjectTemplate
git clone https://<Your VSTS Server name>.visualstudio.com/<Your Team Name>/_git/MyTeamUtilities
```

```
PS D:\AML_Projects\TDSP-Linux\Tutorial_testing\test_team> git clone https://weig-ds.visualstudio.com/DS_Team_1/_git/DS_Team_1_ProjectTemplate
Cloning into 'DS_Team_1_ProjectTemplate'...
warning: You appear to have cloned an empty repository.
Checking connectivity... done.
PS D:\AML_Projects\TDSP-Linux\Tutorial_testing\test_team> git clone https://weig-ds.visualstudio.com/DS_Team_1/_git/DS_Team_1_Utils
Cloning into 'DS_Team_1_Utils'...
warning: You appear to have cloned an empty repository.
Checking connectivity... done.
```

Linux

```
git clone ssh://<Your VSTS Server name>@<Your VSTS Server name>.visualstudio.com:22/<Your Team
Name>/_git/MyTeamProjectTemplate
git clone ssh://<Your VSTS Server name>@<Your VSTS Server name>.visualstudio.com:22/<Your Team
Name>/_git/MyTeamUtilities
```

```
[dsl@weiglinuxdsvm1 test_team]$ git clone ssh://weig-ds@weig-ds.visualstudio.com:22/DS_Team_1/_git/DS_Team_1_Utils
Cloning into 'DS_Team_1_Utils'...
warning: You appear to have cloned an empty repository.
[dsl@weiglinuxdsvm1 test_team]$ git clone ssh://weig-ds@weig-ds.visualstudio.com:22/DS_Team_1/_git/DS_Team_1_ProjectTemplate
Cloning into 'DS_Team_1_ProjectTemplate'...
warning: You appear to have cloned an empty repository.
```

After cloning, two directories **MyTeamProjectTemplate** (D3) and **MyTeamUtilities** (D4) are created in directory **GitRepos\MyTeam**. We have assumed here that you named your team project template and utilities repositories **MyTeamProjectTemplate** and **MyTeamUtilities**.

Copy the group project template content to the local team project template directory

To copy the content of the local **GroupProjectTemplate** (D1) folder to the local **MyTeamProjectTemplate** (D3), run one of the following shell scripts:

From the PowerShell command-line for Windows

```
wget "https://raw.githubusercontent.com/Azure/Azure-MachineLearning-
DataScience/master/Misc/TDSP/tdsp_local_copy_win.ps1" -outfile "tdsp_local_copy_win.ps1"
.\tdsp_local_copy_win.ps1 2
```

```
PS D:\AML_Projects\TDSP-Linux\test0912> .\tdsp_local_copy_win.ps1 2
Please input the full path to Your GroupProjectTemplate repository (source directory): D:\AML_Projects\TDSP-Linux\test0912\GroupProjectTemplate
Please input the full path to Your TeamProjectTemplate repository (destination directory): D:\AML_Projects\TDSP-Linux\test0912\DS_Team_1_ProjectTemplate
Start copying files (except files in .git directory) from D:\AML_Projects\TDSP-Linux\test0912\GroupProjectTemplate to D:\AML_Projects\TDSP-Linux\test0912\DS_Team_1_ProjectTemplate...
PS D:\AML_Projects\TDSP-Linux\test0912>
```

From the Linux shell for the Linux DSVM

```
wget "https://raw.githubusercontent.com/Azure/Azure-MachineLearning-
DataScience/master/Misc/TDSP/tdsp_local_copy_linux.sh"
bash tdsp_local_copy_linux.sh 2
```

```
[dsl@linuxdsvm6 test0912]$ bash tdsp_local_copy_linux.sh 2
Please input the full path to Your GroupProjectTemplate repository (source directory): /home/dsl/test0912/GroupProjectTemplate
Please input the full path to Your TeamProjectTemplate repository (destination directory): /home/dsl/test0912/DS_Team_1_ProjectTemplate
Code/
Code/DataPrep/
Code/DataPrep/dataPrep.py
Code/Model/
Code/Model/model.R
Code/Operationalization/
Code/Operationalization/operationalization.py
Data/
Data/Modeling/
Data/Modeling/modelling.md
```

The scripts exclude the contents of the .git directory. The scripts prompt you to provide the **complete paths** to the source directory D1 and to the destination directory D3.

Customize your team project template or team utilities (optional)

Customize your **MyTeamProjectTemplate** (D3) and **MyTeamUtilities** (D4), if needed, at this stage of the setup process.

- If you want to customize the contents of D3 to meet the specific needs of your team, you can modify the template documents or change the directory structure.
- If your team has developed some utilities that you want to share with your entire team, copy and paste these utilities into directory D4.

Push local directory content to team repositories

To add the contents in the (optionally customized) local directories D3 and D4 to the team repositories R3 and R4, run the following git bash commands either from a Windows PowerShell console or from the Linux shell. Run the commands from the **GitRepos\MyTeam\MyTeamProjectTemplate** directory.

```
git status
git add .
git commit -m"push from DSVM"
git push
```

```
PS D:\AML_Projects\TDSP-Linux\Tutorial_testing\test2\GroupProjectTemplate> git status
On branch master

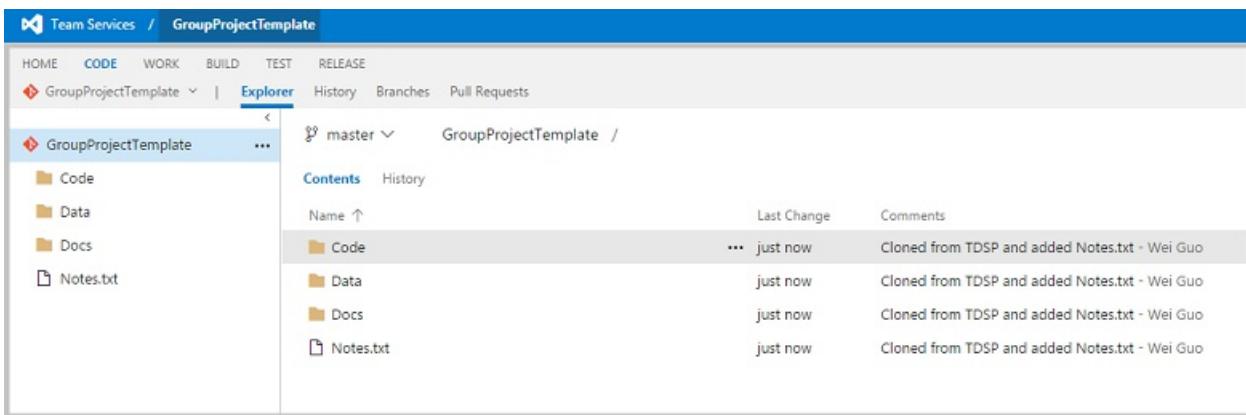
Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    Code/
    Data/
    Docs/
    README.md

nothing added to commit but untracked files present (use "git add" to track)
PS D:\AML_Projects\TDSP-Linux\Tutorial_testing\test2\GroupProjectTemplate> git add .
PS D:\AML_Projects\TDSP-Linux\Tutorial_testing\test2\GroupProjectTemplate> git commit -m"push from local Win DSVM"
[master (root-commit) 4170390] push from local Win DSVM
 21 files changed, 233 insertions(+)
 create mode 100644 Code/DataPrep/dataPrep.py
 create mode 100644 Code/Model/model.R
 create mode 100644 Code/Operationalization/operationalization.py
 create mode 100644 Data/Modeling/modelling.md
 create mode 100644 Data/Processed/processed.md
 create mode 100644 Data/Raw/rawData.md
 create mode 100644 Docs/DataDictionary/ReadMe.md
```

The files in the MyTeamProjectTemplate repository of your group's VSTS server are synced nearly instantly when this script is run.



Now run the same set of four git commands from the **GitRepos\MyTeam\MyTeamUtilities** directory.

[AZURE.NOTE] If this is the first time you commit to a Git repository, you need to configure global parameters `user.name` and `user.email` before you run the `git commit` command. Run the following two commands:

```
git config --global user.name <your name>
git config --global user.email <your email address>
```

If you are committing to multiple Git repositories, use the same name and email address when you commit to each of them. Using the same name and email address proves convenient later on when you build PowerBI dashboards to track your Git activities on multiple repositories.

```
[ds1@linuxdsvm6 ~]$ git config --global user.name "weig"
[ds1@linuxdsvm6 ~]$ git config --global user.email "weig@microsoft.com"
[ds1@linuxdsvm6 ~]$
```

3. Create team data and analytics resources (Optional)

Sharing data and analytics resources with your entire team has performance and cost benefits: team members can execute their projects on the shared resources, save on budgets, and collaborate more efficiently. In this section, we provide instructions on how to create Azure file storage. In the next section, we provide instruction on how to mount Azure file storage to your local machine. For additional information on sharing other resources, such as Azure Data Science Virtual Machines, Azure HDInsight Spark Clusters, see [Platforms and tools](#). This topic provides you guidance from a data science perspective on selecting resources that are appropriate for your needs, and links to product pages and other relevant and useful tutorials that we have published.

[AZURE.NOTE] To avoid data transmitting cross data centers, which might be slow and costly, make sure that the resource group, storage account, and the Azure VM (e.g., DSVM) are in the same Azure data center.

Run the following scripts to create Azure file storage for your team. Azure file storage for your team can be used to store data assets that are useful for your entire team. The scripts prompt you for your Azure account and subscription information, so have these credentials ready to enter.

Create Azure file storage with PowerShell from Windows

Run this script from the PowerShell command-line:

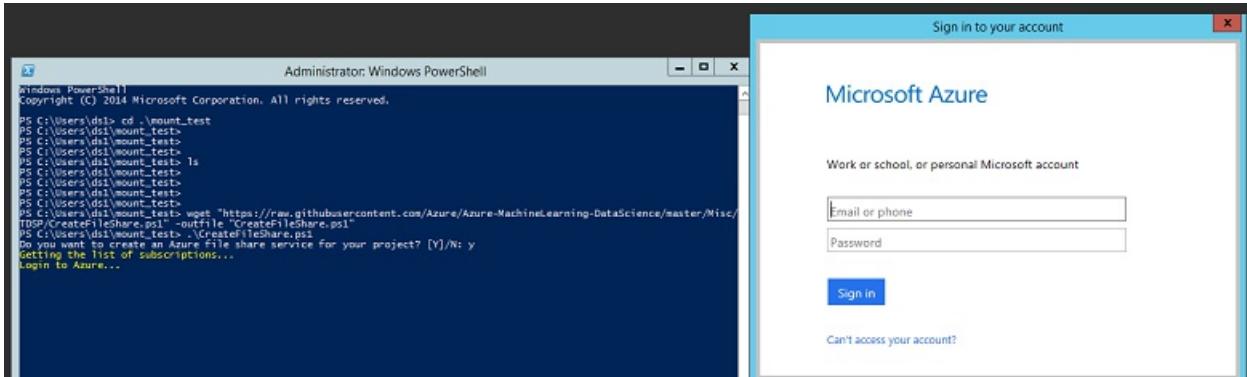
```
 wget "https://raw.githubusercontent.com/Azure/Azure-MachineLearning-DataScience/master/Misc/TDSP/CreateFileShare.ps1" -outfile "CreateFileShare.ps1"
 .\CreateFileShare.ps1
```

```

PS D:\AML_Projects\TDSP-Linux\Tutorial_testing\test_scripts> wget "https://raw.githubusercontent.com/Azure/Azure-MachineLearning-DataScience/master/Misc/TDSP/CreateFileShare.ps1" -outfile "CreateFileShare.ps1"
PS D:\AML_Projects\TDSP-Linux\Tutorial_testing\test_scripts> .\CreateFileShare.ps1
Do you want to create an Azure file share service for your team? [Y]/N: y
Getting the list of subscriptions...
[1]: Microsoft Azure Internal Consumption
[2]: Microsoft Azure Internal - Wei
[3]: Microsoft Azure Internal - Demos
[4]: ACE-Test Subscription
[5]: BDHadoopTeamPMTTestDemo
[6]: ADLTrainingMS
[7]: Office CLE - External
[8]: BigDataDemosExternal
Enter the index of the subscription name where resources will be created(1-8): 1
You selected subscription [1]:Microsoft Azure Internal Consumption. [Y]-yes to continue/N-no to reselect:

```

Log in to your Microsoft Azure account when prompted:



Select the Azure subscription you want to use:

```

[1]: Microsoft Azure Internal Consumption
[2]: Microsoft Azure Internal - Wei
[3]: Microsoft Azure Internal - Demos
[4]: ACE-Test Subscription
[5]: BDHadoopTeamPMTTestDemo
[6]: ADLTrainingMS
[7]: Office CLE - External
[8]: BigDataDemosExternal
Enter the index of the subscription name where resources will be created(1-8): 2

```

Select which storage account to use or create a new one under your selected subscription:

```

Here are the storage account names under your subscription Microsoft Azure Internal - Wei
[1]: weigdsvm2rsc458
[2]: weiglinuxdsvm5664
[3]: weiglinuxresource39870
[4]: weiglinuxtest8
[5]: weigresourcegroup1825
[6]: weigstorage08092016
[7]: weigstoragefordsvm
Do you want to create a new storage account for your Azure file share?[Y]/N: n
Enter the index of the storage account to use(1-7): 6

```

Enter the name of the Azure file storage to create. Only lower case characters, numbers and - are accepted:

```

Enter the name of the file share service to create (lower case characters, numbers, and - are accepted): fs0823
ServiceClient : Microsoft.WindowsAzure.Storage.File.CloudFileClient
Uri          : https://weigstorage08092016.file.core.windows.net/fs0823/data
StorageUri   : Primary = 'https://weigstorage08092016.file.core.windows.net/fs0823/data'; Secondary = ''
Properties    : Microsoft.WindowsAzure.Storage.File.FileDirectoryProperties
Metadata     : {}
Share        : Microsoft.WindowsAzure.Storage.File.CloudFileShare
Parent       : Microsoft.WindowsAzure.Storage.File.CloudFileDirectory
Name         : data

ServiceClient : Microsoft.WindowsAzure.Storage.File.CloudFileClient
Uri          : https://weigstorage08092016.file.core.windows.net/fs0823/data
StorageUri   : Primary = 'https://weigstorage08092016.file.core.windows.net/fs0823/data'; Secondary = ''
Properties    : Microsoft.WindowsAzure.Storage.File.FileDirectoryProperties
Metadata     : {}
Share        : Microsoft.WindowsAzure.Storage.File.CloudFileShare
Parent       : Microsoft.WindowsAzure.Storage.File.CloudFileDirectory
Name         : data

An Azure file share service created. It can be later mounted to the Azure virtual machines created for your team project
$.
Please keep a note for the information of the Azure file share service. It will be needed in the future when mounting it
to Azure virtual machines
Do you want to output the file share information to a file in your current directory?[Y]/N: 

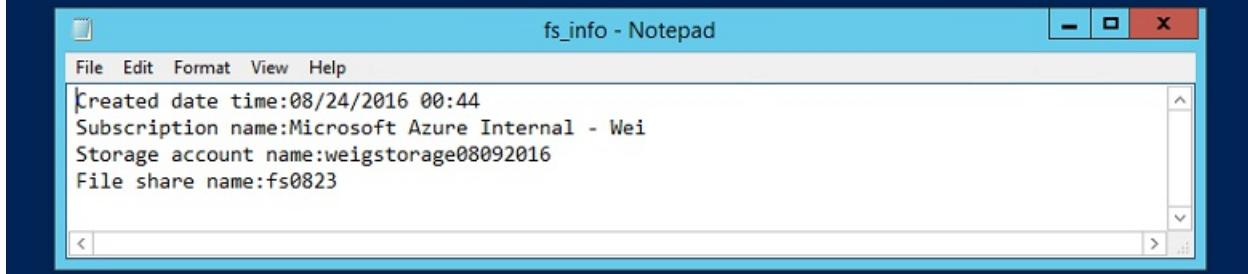
```

To facilitate mounting and sharing this storage after it is created, save the Azure file storage information into a text file and make a note of the path to its location. In particular, you need this file to mount your Azure file storage to your Azure virtual machines in the next section.

It is a good practice to check in this text file into your team ProjectTemplate repository. We recommend to put in the directory **Docs\DataDictionaries**. Therefore, this data asset can be accessed by all projects in your team.

```
Do you want to output the file share information to a file in your current directory? [Y]/N: y
Please provide the file name. This file under the current working directory will be created to store the file share information.: fs_info.txt
File share information output to C:\Users\ds1\mount_test\fs_info.txt. Share it with your team members who want to mount it to their virtual machines.
```

```
PS C:\Users\ds1\mount_test> notepad.exe .\fs_info.txt
PS C:\Users\ds1\mount_test>
```



Create Azure file storage with a Linux script

Run this script from the Linux shell:

```
wget "https://raw.githubusercontent.com/Azure/Azure-MachineLearning-DataScience/master/Misc/TDSP/CreateFileShare.sh"
bash CreateFileShare.sh
```

Log in to your Microsoft Azure account following the instructions on this screen:

```
Do you want to create an Azure file share service for your team? [Y]/N y
info: Executing command config mode
info: New mode is arm
info: config mode command OK
Follow directions on screen to login to your Azure account
info: Executing command login
Authenticating...info: To sign in, use a web browser to open the page https://aka.ms/devicelogin. Enter the code FDUJKQJGQ to authenticate.
/
```

Select the Azure subscription that you want to use:

```
Here are your subscriptions:
1 Microsoft Azure Internal Consumption
2 Microsoft Azure Internal - Wei
3 Microsoft Azure Internal - Demos
4 ACE-Test Subscription
5 BDHadoopTeamPMTestDemo
6 ADLTrainingMS
7 Office CLE - External
8 BigDataDemosExternal

Enter the index of the subscription name where resources will be created (1-8): 2
You selected 2: Microsoft Azure Internal - Wei. [Y]-yes to continue/N-no to reselect: y
info: Executing command account set
info: Setting subscription to "Microsoft Azure Internal - Wei" with id "49bb74df-a9b8-4275-9439-198b33ae0f5f".
```

Select which storage account to use or create a new one under your selected subscription:

```
Here are the storage account names under your subscription Microsoft Azure Internal - Wei:
1 weigdsvm2rsc458
2 weiglinuxdsvm5664
3 weiglinuxresource39870
4 weiglinuxtest8
5 weigresourcegroup1825
6 weigstorage08092016
7 weigstoragefordsvm

Do you want to create a new storage storage account for your Azure file share? [Y]/N: n
Enter the index of the storage account to use (1 - 7 ): 6
You selected storage account weigstorage08092016. [Y]-continue/N-reselect: y
```

Enter the name of the Azure file storage to create, only lower case characters, numbers and - are accepted:

```

Enter the name of the file share service to create (lower case characters, numbers, and - are accepted): linuxfs0823
info: Executing command storage share create
+ Creating storage file share linuxfs0823
+ Getting Storage share information
data:  {
data:    name: 'linuxfs0823',
data:    metadata: {},
data:    etag: '"0x803CBBB44FCB459"',
data:    lastModified: 'Wed, 24 Aug 2016 01:09:16 GMT',
data:    requestId: '22e551c5-001a-012f-52a4-fd5137000000',
data:    quota: '5120',
data:    shareUsage: '0'
data:  }
info: storage share create command OK
An Azure file share service created. It can be later mounted to the Azure virtual machine created for your team projects.
Please keep a note for the information of the Azure file share service. It will be needed in the future when mounting it to Azure virtual machines.
Do you want to output the file share information to a file in your current directory? [Y]/N: ■

```

To facilitate accessing this storage after it is created, save the Azure file storage information into a text file and make a note of the path to its location. In particular, you need this file to mount your Azure file storage to your Azure virtual machines in the next section.

It is a good practice to check in this text file into your team ProjectTemplate repository. We recommend to put in the directory **Docs\DataDictionaries**. Therefore, this data asset can be accessed by all projects in your team.

```

Do you want to output the file share information to a file in your current directory? [Y]/N: y
Please provide the file name. This file under the current working directory will be created to store the file share information: linuxfs0823.txt
File share information output to linuxfs0823.txt. Share it with your team members who want to mount it to their virtual machines.
[ds1@weiqlinuxdsvm5 ~]$ vim linuxfs0823.txt
[ds1@weiqlinuxdsvm5 ~]$
Created date time:Wed Aug 24 01:10:23 UTC 2016
Subscription name:Microsoft Azure Internal - Wei
Storage account name:weigstorage08092016
File share name:linuxfs0823

```

4. Mount Azure file storage (Optional)

After Azure file storage is created successfully, it can be mounted to your local machine using the one of the following PowerShell or Linux scripts.

Mount Azure file storage with PowerShell from Windows

```

wget "https://raw.githubusercontent.com/Azure/Azure-MachineLearning-
DataScience/master/Misc/TDSP/AttachFileShare.ps1" -outfile "AttachFileShare.ps1"
.\AttachFileShare.ps1

```

You are asked to log in first, if you have not logged in.

Click **Enter** or **y** to continue when you are asked if you have an Azure file storage information file, and then input the ****complete path and name*** of the file you create in previous step. The information to mount an Azure file storage is read directly from that file and you are ready to go to the next step.

```

PS C:\Users\ds1\mount_test> wget "https://raw.githubusercontent.com/Azure/Azure-MachineLearning-Datascience/master/Misc/
TDSP/AttachFileShare.ps1" -outfile "AttachFileShare.ps1"
PS C:\Users\ds1\mount_test> .\AttachFileShare.ps1
Do you want to mount an Azure file share service to your Azure virtual machine? [Y]-yes/N-no to quit.: y
Start getting the list of subscriptions under your Azure account...
Here are the subscription names under your Azure account:
[1]: Microsoft Azure Internal Consumption
[2]: Microsoft Azure Internal - Wei
[3]: Microsoft Azure Internal - Demos
[4]: ACE-Test Subscription
[5]: BDHadoopTeamPMTestDemo
[6]: ADLTrainingMS
[7]: Office CLE - External
[8]: BigDataDemosExternal
Do you have a file with the information of the file share you want to mount?[Y]-yes/N-no: y
Please provide the name of the file with the information of the file share you want to mount: fs_info.txt

```

[AZURE.NOTE] If you do not have a file containing the Azure file storage information, the steps to input the information from keyboard are provided at the end of this section.

Then you are asked to enter the name of the drive to be added to your virtual machine. A list of existing drive names is printed on the screen. You should provide a drive name that does not already exist in the list.

```

Environment      : AzureCloud
Account         : weig@microsoft.com
TenantId        : 72f988bf-86f1-41af-91ab-2d7cd011db47
SubscriptionId  : 49bb74df-a9b8-4275-9439-198b33ae0f5f
SubscriptionName : Microsoft Azure Internal - Wei
CurrentStorageAccount : weigstorage08092016

Start getting the list of storage accounts under your subscription Microsoft Azure Internal - Wei
Here are the storage account names under subscription Microsoft Azure Internal - Wei
[1]: weigdsvm2rsc458
[2]: weiglinuxdsvm5664
[3]: weiglinuxresource39870
[4]: weiglinuxtest8
[5]: weigresourcegroup1825
[6]: weigstorage08092016
[7]: weigstoragefordsvm
Existing disk names are:
[1]: A:\ 
[2]: C:\ 
[3]: D:\ 
[4]: E:\ 
Enter the name of the drive to be added to your virtual machine. This name should be different from the disk names your
virtual machine has.: F
File share fs0823 will be mounted to your virtual machine as drive F:
CMDKEY: Credential added successfully.
The command completed successfully.

Do you want to mount other Azure file share services? [Y]-yes/N-no to quit.: n

```

Confirm that a new F drive has been successfully mounted to your machine.

```

PS C:\Users\ds1\mount_test> gdr -PSProvider 'FileSystem'

  Name      Used (GB)    Free (GB) Provider   Root           CurrentLocation
  ----      -----    -----   -----   -----
A          40.04       86.95 FileSystem  A:\ 
C          1.26        283.74 FileSystem  C:\ 
D          5120.00      FileSystem  D:\ 
E          5120.00      FileSystem  E:\ 
F          5120.00      FileSystem  F:\ 


```

How to enter the Azure file storage information manually: If you do not have your Azure file storage information on a text file, you can follow the instructions on the following screen to type in the required subscription, storage account, and Azure file storage information:

```

PS C:\Users\ds1\mount_test> .\AttachFileShare.ps1
Do you want to mount an Azure file share service to your Azure virtual machine? [Y]-yes/N-no to quit.: y
Start getting the list of subscriptions under your Azure account...
Here are the subscription names under your Azure account:
[1]: Microsoft Azure Internal Consumption
[2]: Microsoft Azure Internal - Wei
[3]: Microsoft Azure Internal - Demos
[4]: ACE-Test Subscription
[5]: BDHadoopTeamPMTTestDemo
[6]: ADLTrainingMS
[7]: Office CLE - External
[8]: BigDataDemosExternal
Do you have a file with the information of the file share you want to mount?[Y]-yes/N-no: n

```

Type in your Azure subscription name, select the storage account where the Azure file storage is created, and type in the Azure file storage name:

```

Enter the subscription name where the Azure file share service has been created: Microsoft Azure Internal - Wei

Environment      : AzureCloud
Account         : weig@microsoft.com
TenantId        : 72f988bf-86f1-41af-91ab-2d7cd011db47
SubscriptionId  : 49bb74df-a9b8-4275-9439-198b33ae0f5f
SubscriptionName : Microsoft Azure Internal - Wei
CurrentStorageAccount : weigstorage08092016

Start getting the list of storage accounts under your subscription Microsoft Azure Internal - Wei
Here are the storage account names under subscription Microsoft Azure Internal - Wei
[1]: weigdsvm2rsc458
[2]: weiglinuxdsvm5664
[3]: weiglinuxresource39870
[4]: weiglinuxtest8
[5]: weigresourcegroup1825
[6]: weigstorage08092016
[7]: weigstoragefordsvm
Enter the index of the storage account name where your Azure file share you want to mount is created (1-7): 6
Enter the name of the file share to mount (lower case only): fs0823

```

Mount Azure file storage with a Linux script

```

wget "https://raw.githubusercontent.com/Azure/Azure-MachineLearning-
DataScience/master/Misc/TDSP/AttachFileShare.sh"
bash AttachFileShare.sh

```

```

Do you want to mount an Azure File share service to your Azure Virtual Machine? [Y]-yes/N -no to quit: y
Start getting the list of subscriptions under your Azure account...info: Executing command config mode
info: New mode is arm
info: config mode command OK
Here are the subscriptions :

1 Microsoft Azure Internal Consumption
2 Microsoft Azure Internal - Wei
3 Microsoft Azure Internal - Demos
4 ACE-Test Subscription
5 BDHadoopTeamPMTTestDemo
6 ADLTrainingMS
7 Office CLE - External
8 BigDataDemosExternal

```

You are asked to log in first, if you have not logged in.

Click **Enter** or **y** to continue when you are asked if you have an Azure file storage information file, and then input the ****complete path and name*** of the file you create in previous step. The information to mount an Azure file storage is read directly from that file and you are ready to go to the next step.

```

Do you have a file with the information of the file share you want to mount? Y/N y
Please provide the name of the file with the information of the file share you want to mount: linuxfs0823.txt
info: Executing command account set
info: Setting subscription to "Microsoft Azure Internal - Wei" with id "49bb74df-a9b8-4275-9439-190b33ae0f5f".
info: Changed saved
info: account set command OK
Start getting the list of storage accounts under your subscription Microsoft Azure Internal - Wei Here are the storage account names under subscription Microsoft Azure Internal - Wei
1 weigdsvm2rsc458
2 weiglinuxdsvm564
3 weiglinuxresource39870
4 weiglinuxtest8
5 weigresourcegroup1825
6 weigstorage08092016
7 weigstoragefordsvm

```

Then you are asked to enter the name of the drive to be added to your virtual machine. A list of existing drive names is printed on the screen. You should provide a drive name that does not already exist in the list.

```

Existing disk names are:
Filesystem      Size  Used Avail Use% Mounted on
/dev/sdal       30G   14G   17G  47% /
devtmpfs        3.4G   0     3.4G  0% /dev
tmpfs          3.5G   0     3.5G  0% /dev/shm
tmpfs          3.5G  361M  3.1G  11% /run
tmpfs          3.5G   0     3.5G  0% /sys/fs/cgroup
/dev/sdb1       281G  65M   267G  1% /mnt/resource
tmpfs          697M   0    697M  0% /run/user/1005
Enter the name of the drive to be added to your virtual machine. This name should be different from the disk names your virtual machine has: fs0823
File share linuxfs0823 will be mounted to your virtual machine as drive fs0823 [sudo] password for dsl:
Do you want to mount other Azure File share services? [Y]-yes/N -no to quit: n

```

Confirm that a new F drive has been successfully mounted to your machine.

```
[dsl@weiglinuxdsvm5 ~]$ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sdal       30G   14G   17G  47% /
devtmpfs        3.4G   0     3.4G  0% /dev
tmpfs          3.5G   0     3.5G  0% /dev/shm
tmpfs          3.5G  361M  3.1G  11% /run
tmpfs          3.5G   0     3.5G  0% /sys/fs/cgroup
/dev/sdb1       281G  65M   267G  1% /mnt/resource
tmpfs          697M   0    697M  0% /run/user/1005
//weigstorage08092016,file.core.windows.net/linuxfs0823  5.0T   0   5.0T  0% /fs0823
```

How to enter the Azure file storage information manually: If you do not have your Azure file storage information on a text file, you can follow the instructions on the following screen to type in the required subscription, storage account, and Azure file storage information:

- Input **n**.
- Select the index of the subscription name where the Azure file storage was created in the previous step:

```

Do you want to mount an Azure File share service to your Azure Virtual Machine? [Y]-yes/N -no to quit: y
Start getting the list of subscriptions under your Azure account...info: Executing command config mode
info: New mode is arm
info: config mode command OK
Here are the subscriptions :

1 Microsoft Azure Internal Consumption
2 Microsoft Azure Internal - Wei
3 Microsoft Azure Internal - Demos
4 ACE-Test Subscription
5 BDHadoopTeamPMTTestDemo
6 ADLTrainingMS
7 Office CLE - External
8 BigDataDemosExternal

Do you have a file with the information of the file share you want to mount? Y/N n
Enter the subscription name where the Azure file share service has been created: Microsoft Azure Internal - Wei

```

- Select the storage account under your subscription and type in the Azure file storage name:

```
info: Executing command account set
info: Setting subscription to "Microsoft Azure Internal - Wei" with id "49bb74df-a9b8-4275-9439-198b33ae0f5f".
info: Changes saved
info: account set command OK
Start getting the list of storage accounts under your subscription Microsoft Azure Internal - Wei Here are the storage account names:
Internal - Wei
1 weigdsvm2rsc458
2 weiglinuxdsvm5664
3 weiglinuxresource39870
4 weiglinuxtest8
5 weigresourcegroup1825
6 weigstorage08092016
7 weigstoragefordsvm
Enter the index of the storage account name where your Azure file share you want to mount is created: 6
Enter the name for the file share to mount (lower case only): fs0823
```

- Enter the name of drive to be added to your machine, which should be distinct from any existing ones:

```
Existing disk names are:
Filesystem      Size  Used  Avail Use% Mounted on
/dev/sdal       30G   14G   17G  47% /
/devtmpfs        3.4G    0    3.4G  0% /dev
tmpfs           3.5G    0    3.5G  0% /dev/shm
tmpfs           3.5G  361M  3.1G  1% /run
tmpfs           3.5G    0    3.5G  0% /sys/fs/cgroup
/dev/sdb1       281G  65M  287G  1% /mnt/resource
tmpfs          697M    0   697M  0% /run/user/1005
Enter the name of the drive to be added to your virtual machine. This name should be different from the disk names your virtual machine has: fileshare0823
file share fs0823 will be mounted to your virtual machine as drive fileshare0823 Do you want to mount other Azure File share services? [Y]-yes/N -no to quit: n
```

5. Set up security control policy

From your group VSTS server's homepage, click the **gear icon** next to your user name in the upper right corner, then select the **Security** tab. You can add members to your team here with various permissions.

Display Name	Username Or Scope	Action
Wei Guo	wguo123@outlook.com	Remove

Next steps

Here are links to the more detailed descriptions of the roles and tasks defined by the Team Data Science Process:

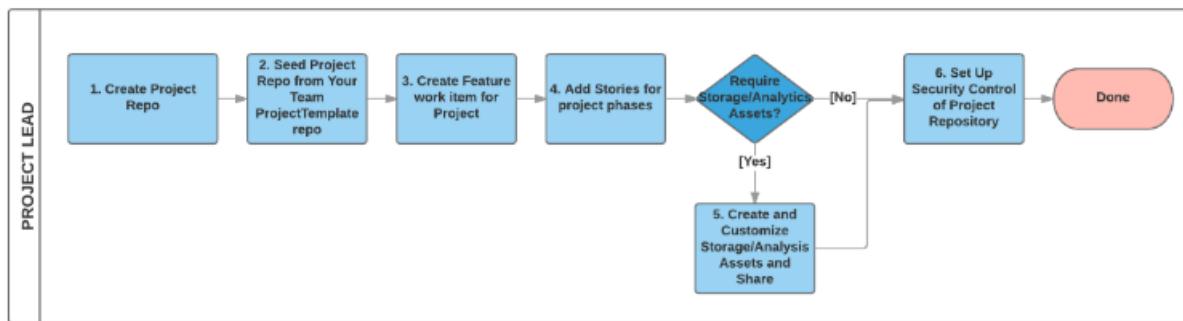
- [Group Manager tasks for a data science team](#)
- [Team Lead tasks for a data science team](#)
- [Project Lead tasks for a data science team](#)
- [Project Individual Contributors for a data science team](#)

Project Lead tasks

11/14/2017 • 7 min to read • [Edit Online](#)

This tutorial outlines the tasks that a project lead is expected to complete for his/her project team. The objective is to establish collaborative team environment that standardizes on the [Team Data Science Process \(TDSP\)](#). The TDSP is a framework developed by Microsoft that provides a structured sequence of activities to execute cloud-based, predictive analytics solutions efficiently. For an outline of the personnel roles and their associated tasks that are handled by a data science team standardizing on this process, see [Team Data Science Process roles and tasks](#).

A **Project Lead** manages the daily activities of individual data scientists on a specific data science project. The workflow for the tasks to be completed by project leads to set up this environment are depicted in the following figure:



This topic currently covers tasks 1,2 and 6 of this workflow for project leads.

[AZURE.NOTE] We outline the steps needed to set up a TDSP team environment for a project using Visual Studio Team Services (VSTS) in the following instructions. We specify how to accomplish these tasks with VSTS because that is how we implement TDSP at Microsoft. If another code-hosting platform is used for your group, the tasks that need to be completed by the team lead generally do not change. But the way to complete these tasks is going to be different.

Repositories and directories

This tutorial uses abbreviated names for repositories and directories. These names make it easier to follow the operations between the repositories and directories. This notation (R for Git repositories and D for local directories on your DSVM) is used in the following sections:

- **R3:** The team **ProjectTemplate** repository on Git your team lead has set up.
- **R5:** The project repository on Git you setup for your project.
- **D3:** The local directory cloned from R3.
- **D5:** The local directory cloned from R5.

0. Prerequisites

The prerequisites are satisfied by completing the tasks assigned to your group manager outlined in [Group Manager tasks for a data science team](#) and to your team lead outlined in [Team lead tasks for a data science team](#).

To summarize here, the following requirements need to meet before you begin the team lead tasks:

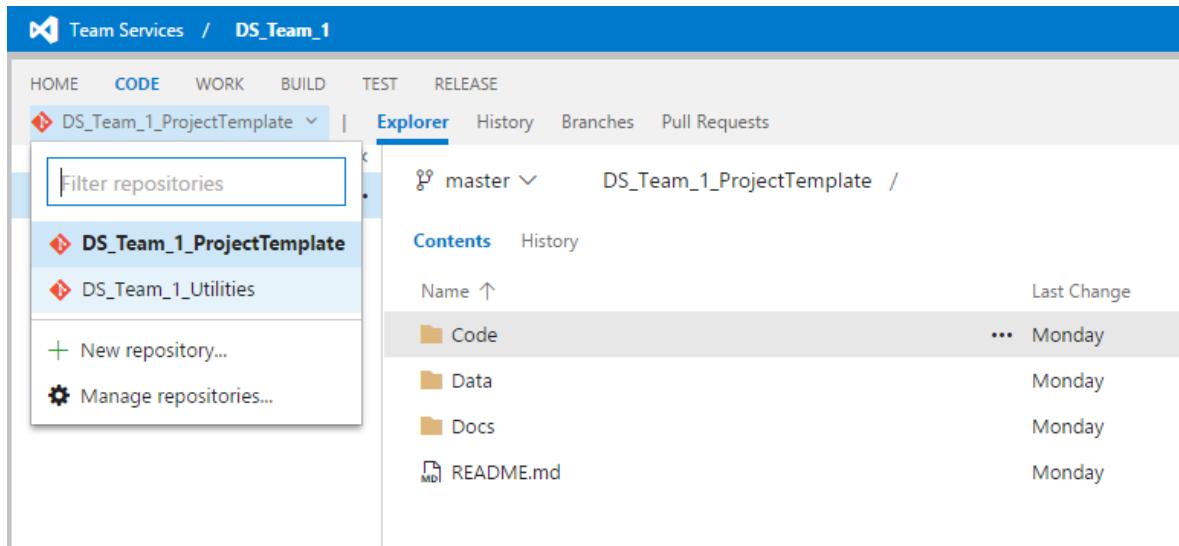
- Your **group VSTS server** (or group account on some other code-hosting platform) has been set up by your

group manager.

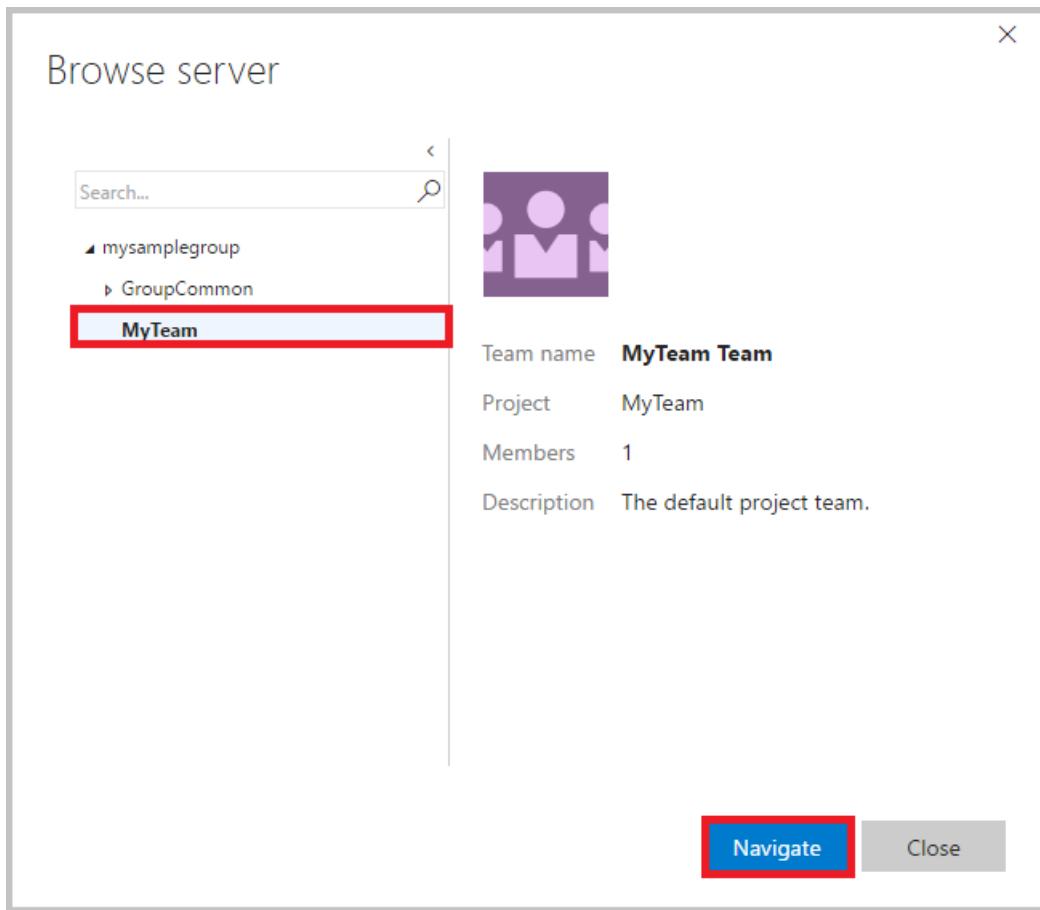
- Your **TeamProjectTemplate repository** (R3) has been set up under your group account by your team lead on the code-hosting platform you plan to use.
- You have been **authorized** by your team lead to create repositories on your group account for your team.
- Git must be installed on your machine. If you are using a Data Science Virtual Machine (DSVM), Git has been pre-installed and you are good to go. Otherwise, see the [Platforms and tools appendix](#).
- If you are using a **Windows DSVM**, you need to have [Git Credential Manager \(GCM\)](#) installed on your machine. In the README.md file, scroll down to the **Download and Install** section and click the *latest installer*. This takes you to the latest installer page. Download the .exe installer from here and run it.
- If you are using **Linux DSVM**, create an SSH public key on your DSVM and add it to your group VSTS server. For more information about SSH, see the **Create SSH public key** section in the [Platforms and tools appendix](#).

1. Create a project repository (R5)

- Log in to your group VSTS server at <https://<VSTS Server Name>.visualstudio.com>.
- Under **Recent projects & teams**, click **Browse**. A window that pops up lists all team projects on the VSTS server.



- Click the team project name in which you are going to create your project repository. In this example, click **MyTeam**.
- Then, click **Navigate** to be directed to the home page of the team project **MyTeam**:



- Click **Collaborate on code** to be directed to the git home page of your team project.

Welcome

Get started using Visual Studio Team Services to make the most of your team dashboard.

Manage Work
Add work to your board

Collaborate on code
Add code to your repository

Continuously integrate
Automate your builds

Visualize progress
Learn how to add charts

Open User Stories

Query returned no results. Create new work items or edit query to see results.

Team Members

It's lonely in here...
Invite a friend

Work

Backlog
Board
Task board
Queries

Sprint Burndown

Set iteration dates to use the sprint burndown widget
Set iteration dates

New Work Item

Enter title
Bug
Create

Open User Stories

0 Work items

Visual Studio

Open in Visual Studio
Requires Visual Studio 2013+

Get Visual Studio
See Visual Studio downloads

- Click the downward arrow at the top left corner, and select **+ New repository**.

MyTeamProjectTemplate

Explorer History Branches Pull Requests

Filter repositories

MyTeamProjectTemplate

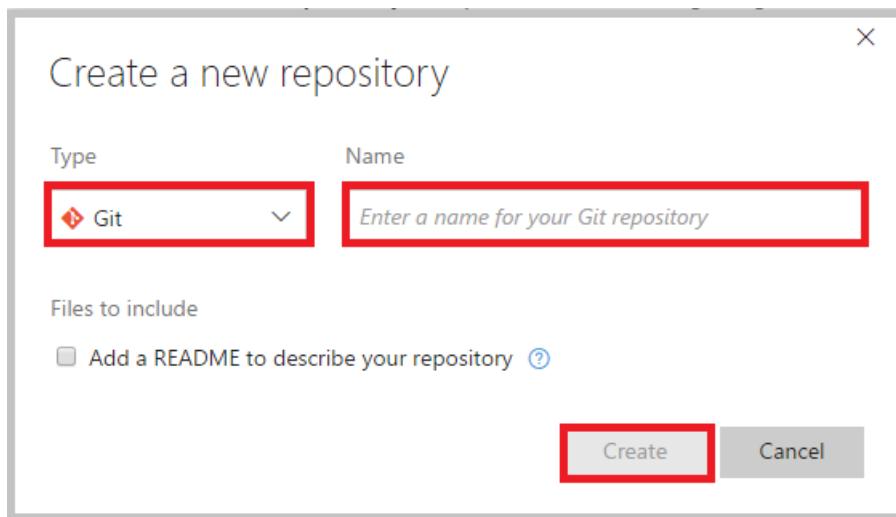
New repository

Manage repositories

Contents History

Name ↑	Last Change	Comments
Code	just now	seeded the MyTeamProjectTemplate from DGADS - Hang Zhang
Data	just now	seeded the MyTeamProjectTemplate from DGADS - Hang Zhang
Docs	just now	seeded the MyTeamProjectTemplate from DGADS - Hang Zhang
README.md	just now	seeded the MyTeamProjectTemplate from DGADS - Hang Zhang

- In the **Create a new repository** window, input a name for your project git repository. Make sure that you select **Git** as the type of the repository. In this example, we use the name **DSPProject1**.



- To create your **DSPProject1** project git repository, click **Create**.

2. Seed the **DSPProject1** project repository

The task here is to seed the **DSPProject1** project repository (R5) from your team project template repository (R3). The seeding procedure uses the directories D3 and D5 on your local DSVM as intermediate staging sites. In summary, the seeding path is: R3 -> D3 -> D5 -> R5.

If you need to customize your **DSPProject1** project repository to meet some specific project needs, you do so in the penultimate step of following procedure. Here is a summary of the steps used to seed the content of the **DSPProject1** project repository. The individual steps correspond to the subsections in the seeding procedure:

- Clone team project template repository into local directory: team R3 - cloned to -> local D3.
- Clone **DSPProject1** repository to a local directory: team R5 - cloned to -> local D5.
- Copy cloned team project template content to local clone of **DSPProject1** repository: D3 - contents copied to -> D5.
- (Optional) Customization local D5.
- Push local **DSPProject1** content to team repositories: D5 - contents add to -> team R5.

Clone your team project template repository (R3) to a directory (D3) on your local machine.

On your local machine, create a directory:

- C:\GitRepos\MyTeamCommon* for Windows
- \$home/GitRepos/MyTeamCommon* for Linux

Change to that directory. Then, run the following command to clone your team project template repository to your local machine.

Windows

```
git clone <the HTTPS URL of the TeamProjectTemplate repository>
```

If you are using VSTS as the code-hosting platform, typically, the *HTTPS URL of your team project template repository* is:

https://<VSTS Server Name>.visualstudio.com/<Your team project name>/_git/<Your team project template repository name>.

In this example, we have:

https://mysamplegroup.visualstudio.com/MyTeam/_git/MyTeamProjectTemplate.

```
PS C:\GitRepo\MyTeamCommon> git clone https://mysamplegroup.visualstudio.com/MyTeam/_git/MyTeamProjectTemplate
Cloning into 'MyTeamProjectTemplate'...
remote:
remote:           vSTS
remote:           vSTSvSTSV
remote:           vSTSvSTSVST
remote: VSTS       vSTSvSTSVSTSV
remote: VSTSvS     vSTSvSTSV STSVS
remote: VSTSvSTSVsTSVSTSVS   TSVST
remote: VS tSVSTSVSTv    STSVS
remote: VS tSVSTSVST    SVSTS
remote: VS tSVSTSVSTSVsts  VSTSv
remote: VSTSvST    SVSTSvSTS VSTSv
remote: VSTSv      STSVSTSVSTS
remote:           VSTSvSTSV
remote:           VSTSvSTS
remote:           VSTS (TM)
remote:
remote: Microsoft (R) Visual Studio (R) Team Services
remote:
Unpacking objects: 100% (32/32), done.
Checking connectivity... done.
PS C:\GitRepo\MyTeamCommon>
```

Linux

```
git clone <the SSH URL of the TeamProjectTemplate repository>
```

```
benchmark@benchmark-vm-l:~/GitRepos/MyTeamCommon$ git clone ssh://mysamplegroup@mysamplegroup.visualstudio.com:22/MyTeam/_git/MyTeamProjectTemplate
Cloning into 'MyTeamProjectTemplate'...
remote:
remote:           vSTS
remote:           vSTSvSTSV
remote:           vSTSvSTSVST
remote: VSTS       vSTSvSTSVSTSV
remote: VSTSvS     vSTSvSTSV STSVS
remote: VSTSvSTSVsTSVSTSVS   TSVST
remote: VS tSVSTSVSTv    STSVS
remote: VS tSVSTSVST    SVSTS
remote: VS tSVSTSVSTSVsts  VSTSv
remote: VSTSvST    SVSTSvSTS VSTSv
remote: VSTSv      STSVSTSVSTS
remote:           VSTSvSTSVS
remote:           VSTSvSTS
remote:           VSTS (TM)
remote:
remote: Microsoft (R) Visual Studio (R) Team Services
remote:
Receiving objects: 100% (32/32), 14.35 KiB | 0 bytes/s, done.
Resolving deltas: 100% (1/1), done.
Checking connectivity... done.
```

If you are using VSTS as the code-hosting platform, typically, the *SSH URL of the team project template repository* is:

ssh://<VSTS Server Name>@<VSTS Server Name>.visualstudio.com:22/<Your Team Project Name>/_git/<Your team project template repository name>.

In this example, we have:

ssh://mysamplegroup@mysamplegroup.visualstudio.com:22/MyTeam/_git/MyTeamProjectTemplate.

Clone DSProject1 repository (R5) to a directory (D5) on your local machine

Change directory to **GitRepos**, and run the following command to clone your project repository to your local machine.

Windows

```
git clone <the HTTPS URL of the Project repository>
```

```
PS C:\GitRepo> git clone https://mysamplegroup.visualstudio.com/MyTeam/_git/DSProject1
Cloning into 'DSProject1'...
warning: You appear to have cloned an empty repository.
Checking connectivity... done.
```

If you are using VSTS as the code-hosting platform, typically, the *HTTPS URL* of the Project repository is ***https://<VSTS Server Name>.visualstudio.com/<Your Team Project Name>/_git/<Your project repository name>***. In this example, we have ***https://mysamplegroup.visualstudio.com/MyTeam/_git/DSProject1***.

Linux

```
git clone <the SSH URL of the Project repository>
```

```
benchmark@benchmark-vm-l:~/GitRepos$ git clone ssh://mysamplegroup@mysamplegroup.visualstudio.com:22/MyTeam/_git/DSProject1
Cloning into 'DSProject1'...
warning: You appear to have cloned an empty repository.
Checking connectivity... done.
```

If you are using VSTS as the code-hosting platform, typically, the *SSH URL of the project repository* is
`_ssh://<VSTS Server Name>@<VSTS Server Name>.visualstudio.com:22/_git/<Your project repository name>.`

In this example, we have

`ssh://mysamplegroup@mysamplegroup.visualstudio.com:22/MyTeam/_git/DSPProject1`

Copy contents of D3 to D5

Now in your local machine, you need to copy the content of $D3$ to $D5$, except the git metadata in .git directory. The following scripts will do the job. Make sure to type in the correct and full paths to the directories. Source folder is the one for your team ($D3$); destination folder is the one for your project ($D5$).

Windows

```
 wget "https://raw.githubusercontent.com/Azure/Azure-MachineLearning-  
DataScience/master/Misc/TDSP/tdsp_local_copy_win.ps1" -outfile "tdsp_local_copy_win.ps1"  
.\\tdsp local copy win.ps1 -role 3
```

```
PS D:\AML_Projects\TDSP-Linux\test0912> .\tdsp_local_copy_win.ps1 3  
Please input the full path to Your TeamProjectTemplate repository (source directory): D:\AML_Projects\TDSP-Linux\test0912\DS_Team_1_ProjectTemplate  
Please input the full path to Your Project repository (destination directory): D:\AML_Projects\TDSP-Linux\test0912\DSProject1  
Start copying files (except files in .git directory) from D:\AML_Projects\TDSP-Linux\test0912\DS_Team_1_ProjectTemplate to D:\AML_Projects\TDSP-Linux\test0912\DSProject1...
```

Now you can see in *DSPProject1* folder, all the files (excluding the .git) are copied from *MyTeamProjectTemplate*.

Linux

```
 wget "https://raw.githubusercontent.com/Azure/Azure-MachineLearning-  
DataScience/master/Misc/TDSP/tdsp_local_copy_linux.sh"  
 bash tdsp local copy linux.sh 3
```

```
[dsl0@linuxdsvm6 test0912]$ bash tdsp_local_copy_linux.sh 3
Please input the full path to Your TeamProjectTemplate repository (source directory): /home/dsl/test0912/DS_Team_1_ProjectTemplate/
Please input the full path to Your Project repository (destination directory): /home/dsl/test0912/DSProject1/
Code/
Code/DataPrep/
Code/DataPrep/dataPrep.py
Code/Model/
Code/Model/model.R
Code/Operationalization/
Code/Operationalization/operationalization.py
Data/
Data/Modeling/
Data/Modeling/modelling.nd
Data/Processed/
Data/Processed/processed.nd
Data/Raw/
Data/Raw/rwData.md
docs/
docs/DataDictionaries/
docs/DataDictionaries/ReadMe.md
docs/DataDictionaries/dict1.css
```

Now you can see in *DSPProject1* folder, all the files (except the metadata in .git) are copied from *MyTeamProjectTemplate*.

```
[ds1@linuxdsvm6 test0912]$ ll ./DSPProject1/
total 4
drwxrwxr-x. 5 ds1 ds1 58 Sep 12 20:51 Code
drwxrwxr-x. 5 ds1 ds1 47 Sep 12 20:51 Data
drwxrwxr-x. 6 ds1 ds1 72 Sep 12 20:51 Docs
-rw-rw-r--. 1 ds1 ds1 80 Sep 12 20:51 README.md
```

Customize D5 if you need to (Optional)

If your project needs some specific directories or documents, other than the ones you get from your team project template (copied to your D5 directory in the previous step), you can customize the content of D5 now.

Add contents of *DSPProject1* in D5 to R5 on your group VSTS server

You now need to push contents in ***DSPProject1*** to **R5** repository in your team project on your group's VSTS server.

- Change to directory **D5**.
- Use the following git commands to add the contents in **D5** to **R5**. The commands are the same for both Windows and Linux systems.

```
git status
git add .
git commit -m "push from win DSVM"
git push
```

- Commit the change and push.

[AZURE.NOTE] If this is the first time you commit to a Git repository, you need to configure global parameters *user.name* and *user.email* before you run the `git commit` command. Run the following two commands:

```
git config --global user.name <your name>
git config --global user.email <your email address>
```

If you are committing to multiple Git repositories, use the same name and email address across all of them. Using the same name and email address proves convenient later on when you build PowerBI dashboards to track your Git activities on multiple repositories.

```
[ds1@linuxdsvm6 ~]$
[ds1@linuxdsvm6 ~]$ git config --global user.name "weig"
[ds1@linuxdsvm6 ~]$ git config --global user.email "weig@microsoft.com"
[ds1@linuxdsvm6 ~]$
```

6. Create and mount Azure file storage as project resources (Optional)

If you want to create Azure file storage to share data, such as the project raw data or the features generated for your project, so that all project members have access to the same datasets from multiple DSVMs, follow the instructions in sections 3 and 4 of [Team Lead tasks for a data science team](#).

Next steps

Here are links to the more detailed descriptions of the roles and tasks defined by the Team Data Science Process:

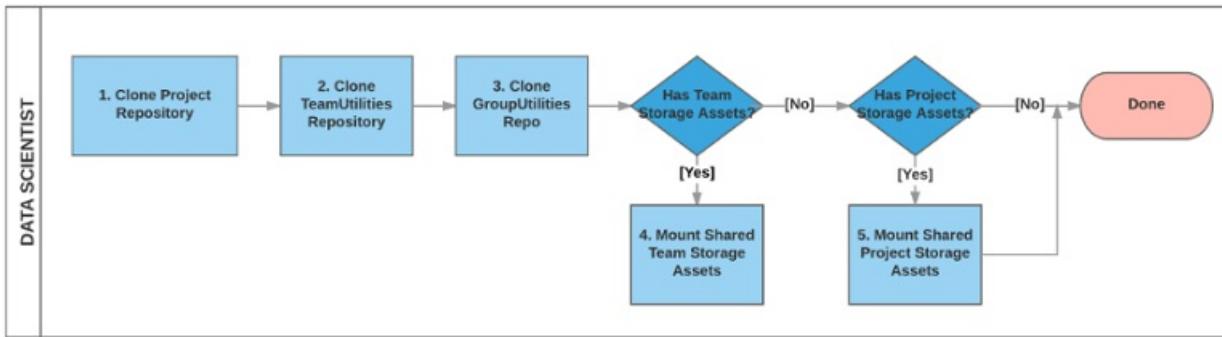
- [Group Manager tasks for a data science team](#)
- [Team Lead tasks for a data science team](#)
- [Project Lead tasks for a data science team](#)
- [Project Individual Contributors for a data science team](#)

Individual Contributor tasks

11/14/2017 • 4 min to read • [Edit Online](#)

This topic outlines the tasks that an individual contributor is expected to complete for their data science team. The objective is to establish collaborative team environment that standardizes on the [Team Data Science Process](#) (TDSP). For an outline of the personnel roles and their associated tasks that are handled by a data science team standardizing on this process, see [Team Data Science Process roles and tasks](#).

The tasks of project individual contributors (data scientists) to set up the TDSP environment for the project are depicted as follows:



- **GroupUtilities** is the repository that your group is maintaining to share useful utilities across the entire group.
- **TeamUtilities** is the repository that your team is maintaining specifically for your team.

For instructions on how to execute a data science project under TDSP, see [Execution of Data Science Projects](#).

[AZURE.NOTE] We outline the steps needed to set up a TDSP team environment using Visual Studio Team Services (VSTS) in the following instructions. We specify how to accomplish these tasks with VSTS because that is how we implement TDSP at Microsoft. If another code-hosting platform is used for your group, the tasks that need to be completed by the team lead generally do not change. But the way to complete these tasks is going to be different.

Repositories and directories

This tutorial uses abbreviated names for repositories and directories. These names make it easier to follow the operations between the repositories and directories. This notation (**R** for Git repositories and **D** for local directories on your DSVM) is used in the following sections:

- **R2**: The GroupUtilities repository on Git that your group manager has set up on your VSTS group server.
- **R4**: The TeamUtilities repository on Git that your team lead has set up.
- **R5**: The Project repository on Git that has been set up by your project lead.
- **D2**: The local directory cloned from R2.
- **D4**: The local directory cloned from R4.
- **D5**: The local directory cloned from R5.

Step-0: Prerequisites

The prerequisites are satisfied by completing the tasks assigned to your group manager outlined in [Group Manager tasks for a data science team](#). To summarize here, the following requirements need to be met before you begin the team lead tasks:

- Your group manager has set up the **GroupUtilities** repository (if any).
- Your team lead has set up the **TeamUtilities** repository (if any).
- Your project lead has set up the project repository.
- You have been added to your project repository by your project lead with the privilege to clone from and push back to the project repository.

The second, **TeamUtilities** repository, prerequisite is optional, depending on whether your team has a team-specific utility repository. If any of other three prerequisites has not been completed, contact your team lead, your project lead, or their delegates to set it up by following the instructions for [Team Lead tasks for a data science team](#) or for [Project Lead tasks for a data science team](#).

- Git must be installed on your machine. If you are using a Data Science Virtual Machine (DSVM), Git has been pre-installed and you are good to go. Otherwise, see the [Platforms and tools appendix](#).
- If you are using a **Windows DSVM**, you need to have [Git Credential Manager \(GCM\)](#) installed on your machine. In the README.md file, scroll down to the **Download and Install** section and click the *latest installer*. This takes you to the latest installer page. Download the .exe installer from here and run it.
- If you are using **Linux DSVM**, create an SSH public key on your DSVM and add it to your group VSTS server. For more information about SSH, see the **Create SSH public key** section in the [Platforms and tools appendix](#).
- If your team and/or project lead has created some Azure file storage that you need to mount to your DSVM, you should get the Azure file storage information from them.

Step 1-3: Clone group, team, and project repositories to local machine

This section provides instructions on completing the first three tasks of project individual contributors:

- Clone the **GroupUtilities** repository R2 to D2
- Clone the **TeamUtilities** repository R4 to D4
- Clone the **Project** repository R5 to D5.

On your local machine, create a directory **C:\GitRepos** (for Windows) or **\$home/GitRepos** (for Linux), and then change to that directory.

Run the one of the following commands (as appropriate for your OS) to clone your **GroupUtilities**, **TeamUtilities**, and **Project** repositories to directories on your local machine:

Windows

```
git clone <the URL of the GroupUtilities repository>
git clone <the URL of the TeamUtilities repository>
git clone <the URL of the Project repository>
```

```

PS D:\AML_Projects\TDSP-Linux\Tutorial_testing\Project_1> git clone https://weig-ds.visualstudio.com/GroupCommon/_git/GroupUtilities
Cloning into 'GroupUtilities'...
remote:
remote:           VSTS
remote:           VSTSvSVTSV
remote:           VSTSvSVSvST
remote: VSTS       VSTSvSVTSvSVS
remote: VSTSvS     VSTSvSVTSV STSVS
remote: VSTSvTSvSvSVSvSVS   TSvST
remote: VS    tSVSVTSvSVS   STSVS
remote: VS    tSVSVTSvST   SVTS
remote: VS    tSVSVTSvSVsts  VSTSv
remote: VSTSvST   SVTSvSVsts VSTSv
remote: VSTSvS     STSVSVTSvSVS
remote:           VSTSvSVSvST
remote:           VSTSvSVTs
remote:           VSTS   (TM)
remote:
remote: Microsoft (R) Visual Studio (R) Team Services
remote:
Receiving objects: 100% (272/272), 38.20 MiB | 1.29 MiB/s, done.
Resolving deltas: 100% (41/41), done.
Checking connectivity... done.
Checking out files: 100% (213/213), done.
PS D:\AML_Projects\TDSP-Linux\Tutorial_testing\Project_1> git clone https://weig-ds.visualstudio.com/DS_Team_1/_git/DS_Team_1_Utils
Cloning into 'DS_Team_1_Utils'...
remote:
remote:           VSTS
remote:           VSTSvSVTSV
remote:           VSTSvSVSvST
remote: VSTS       VSTSvSVTSvSVS
remote: VSTSvS     VSTSvSVTSV STSVS
remote: VSTSvTSvSvSVSvSVS   TSvST
remote: VS    tSVSVTSvSVS   STSVS
remote: VS    tSVSVTSvST   SVTS
remote: VS    tSVSVTSvSVsts  VSTSv
remote: VSTSvST   SVTSvSVsts VSTSv
remote: VSTSvS     STSVSVTSvSVS
remote:           VSTSvSVSvST
remote:           VSTS   (TM)
remote:
remote: Microsoft (R) Visual Studio (R) Team Services
remote:
Receiving objects: 100% (243/243), 38.12 MiB | 1.82 MiB/s, done.
Resolving deltas: 100% (21/21), done.
Checking connectivity... done.
Checking out files: 100% (213/213), done.
PS D:\AML_Projects\TDSP-Linux\Tutorial_testing\Project_1> git clone https://weig-ds.visualstudio.com/DS_Team_1/_git/Project_1
Cloning into 'Project_1'...
remote:
remote:           VSTS
remote:           VSTSvSVTSV
remote:           VSTSvSVSvST
remote: VSTS       VSTSvSVTSvSVS
remote: VSTSvS     VSTSvSVTSV STSVS
remote: VSTSvTSvSvSVSvSVS   TSvST
remote: VS    tSVSVTSvSVS   STSVS
remote: VS    tSVSVTSvST   SVTS
remote: VS    tSVSVTSvSVsts  VSTSv
remote: VSTSvST   SVTSvSVsts VSTSv
remote: VSTSvS     STSVSVTSvSVS
remote:           VSTSvSVSvST
remote:           VSTSvSVTs
remote:           VSTS   (TM)
remote:
remote: Microsoft (R) Visual Studio (R) Team Services
remote:
Unpacking objects: 100% (34/34), done.
Checking connectivity... done.

```

Confirm that you see the three folders under your project directory.

```

PS D:\AML_Projects\TDSP-Linux\Tutorial_testing\Project_1> ls

Directory: D:\AML_Projects\TDSP-Linux\Tutorial_testing\Project_1

Mode                LastWriteTime          Length Name
----              - - - - - - - - - - - - - - - - - - -
d-----        8/11/2016   3:57 PM           0 DS_Team_1_Utils
d-----        8/11/2016   3:55 PM           0 GroupUtilities
d-----        8/11/2016   3:57 PM           0 Project_1

```

Linux

```

git clone <the SSH URL of the GroupUtilities repository>
git clone <the SSH URL of the TeamUtilities repository>
git clone <the SSH URL of the Project repository>

```

```

Cloning into 'GroupUtilities'...
The authenticity of host 'weig-ds.visualstudio.com (23.98.150.230)' can't be established.
RSA key fingerprint is 97:70:33:82:fd:29:3a:73:39:af:6a:07:ad:f8:80:49.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'weig-ds.visualstudio.com,23.98.150.230' (RSA) to the list of known hosts.
remote:
remote:           vSTs
remote:           vSTSVSTSv
remote:           vSTSVSTSvST
remote: VSTS       vSTSVSTSvSTSV
remote: VSTSvS     vSTSVSTSv STSVs
remote: VSTSvSTSvTsTSVSTSvS   TSVST
remote: VS   tSVSTSvSTSv      STSVs
remote: VS   tSVSTSvST       SVSTS
remote: VS   tSVSTSvSTSvSts  VSTSv
remote: VSTSvST    SVSTSvSTSv VSTSv
remote: VSTSv      STSVSTSvSTSv
remote:           VSTSVSTSvST
remote:           VSTS   (TM)
remote:
remote: Microsoft (R) Visual Studio (R) Team Services
remote:
Receiving objects: 100% (272/272), 38.20 MiB | 6.92 MiB/s, done.
Resolving deltas: 100% (41/41), done.
Checking out files: 100% (213/213), done.
[dsl@weiglinuxdsvm5 Project_1]$ git clone ssh://weig-ds@weig-ds.visualstudio.com:22/DS_Team_1/_git/DS_Team_1_Utilities
Cloning into 'DS_Team_1_Utilities'...
remote:
remote:           vSTs
remote:           vSTSVSTSv
remote:           vSTSVSTSvST
remote: VSTS       vSTSVSTSvSTSV
remote: VSTSvS     vSTSVSTSv STSVs
remote: VSTSvSTSvTsTSVSTSvS   TSVST
remote: VS   tSVSTSvSTSv      STSVs
remote: VS   tSVSTSvST       SVSTS
remote: VS   tSVSTSvSTSvSts  VSTSv
remote: VSTSvST    SVSTSvSTSv VSTSv
remote: VSTSv      STSVSTSvSTSv
remote:           VSTSVSTSvST
remote:           VSTSvSTSs
remote:           VSTS   (TM)
remote:
remote: Microsoft (R) Visual Studio (R) Team Services
remote:
Receiving objects: 100% (243/243), 38.12 MiB | 6.24 MiB/s, done.
Resolving deltas: 100% (21/21), done.
Checking out files: 100% (213/213), done.
[dsl@weiglinuxdsvm5 Project_1]$ git clone ssh://weig-ds@weig-ds.visualstudio.com:22/DS_Team_1/_git/Project_1
Cloning into 'Project_1'...
remote:
remote:           vSTs
remote:           vSTSVSTSv
remote:           vSTSVSTSvST
remote: VSTS       vSTSVSTSvSTSV
remote: VSTSvS     vSTSVSTSv STSVs
remote: VSTSvSTSvTsTSVSTSvS   TSVST
remote: VS   tSVSTSvSTSv      STSVs
remote: VS   tSVSTSvST       SVSTS
remote: VS   tSVSTSvSTSvSts  VSTSv
remote: VSTSvST    SVSTSvSTSv VSTSv
remote: VSTSv      STSVSTSvSTSv
remote:           VSTSVSTSvST
remote:           VSTSvSTSs
remote:           VSTS   (TM)
remote:
remote: Microsoft (R) Visual Studio (R) Team Services
remote:
Receiving objects: 100% (34/34), 14.65 KiB | 0 bytes/s, done.
Resolving deltas: 100% (2/2), done.

```

Confirm that you see the three folders under your project directory.

```
[dsl@weiglinuxdsvm5 Project_1]$ ll
total 0
drwxrwxr-x. 7 dsl dsl 80 Aug 11 23:09 DS_Team_1_Utilities
drwxrwxr-x. 7 dsl dsl 80 Aug 11 23:08 GroupUtilities
drwxrwxr-x. 6 dsl dsl 66 Aug 11 23:09 Project_1
```

Step 4-5: Mount Azure file storage to your DSVM (Optional)

To mount Azure file storage to your DSVM, see the instructions in Section 4 of the [Team lead tasks for a data science team](#)

Next steps

Here are links to the more detailed descriptions of the roles and tasks defined by the Team Data Science Process:

- Group Manager tasks for a data science team
- Team Lead tasks for a data science team
- Project Lead tasks for a data science team
- Project Individual Contributors for a data science team

Team Data Science Process project planning

12/7/2017 • 1 min to read • [Edit Online](#)

The Team Data Science Process (TDSP) provides a lifecycle to structure the development of your data science projects. This article provides links to Microsoft Project and Excel templates that help you plan and manage these project stages.

The lifecycle outlines the major stages that projects typically execute, often iteratively:

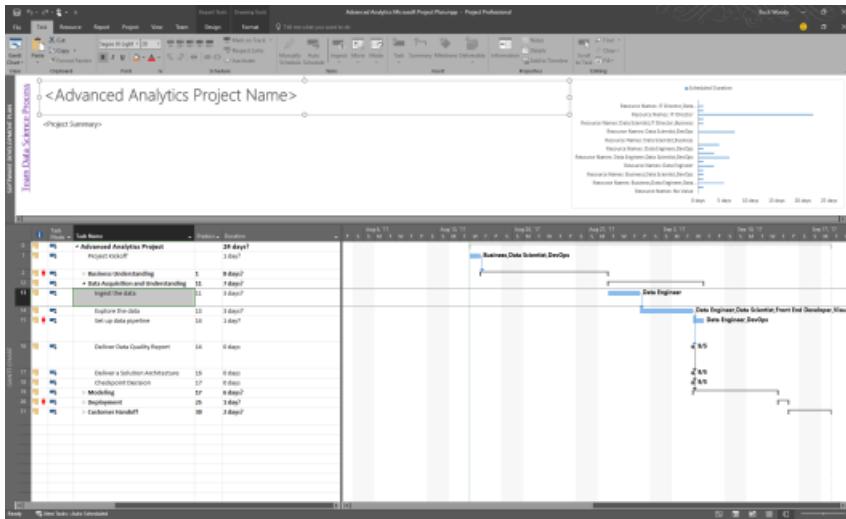
- Business Understanding
- Data Acquisition and Understanding
- Modeling
- Deployment
- Customer Acceptance

For descriptions of each of these stages, see [The Team Data Science Process lifecycle](#).

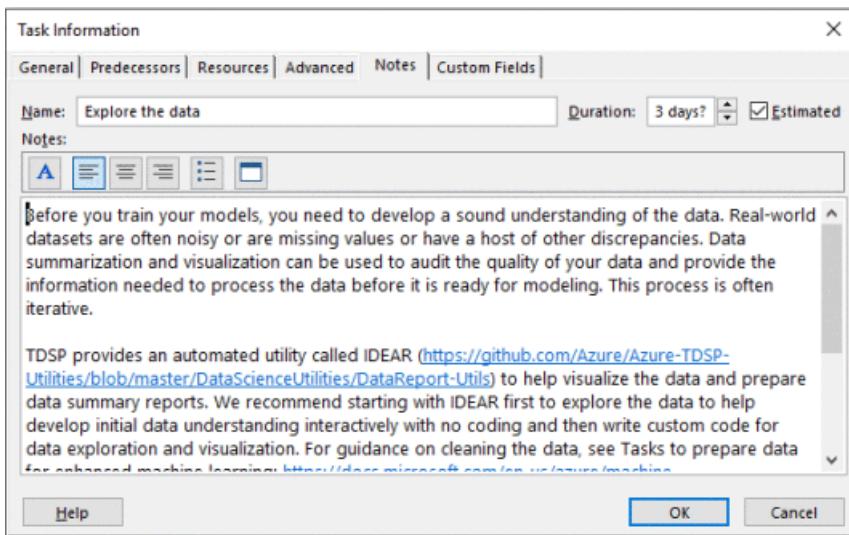
Microsoft Project template

The Microsoft Project template for the Team Data Science Process is available from here: [Microsoft Project template](#)

When you open the plan, click the link to the far left for the TDSP. Change the name and description and then add in any other team resources you need. Estimate the dates required from your experience.



Each task has a note. Open those tasks to see what resources have already been created for you.



Excel template

If don't have access to Microsoft Project, an Excel worksheet with all the same data is also available for download here: [Excel template](#) You can pull it in to whatever tool you prefer to use.

Use these templates at your own risk. The [usual disclaimers](#) apply.

Next steps

[Agile development of data science projects](#) This document describes to execute a data science project in a systematic, version controlled, and collaborative way within a project team by using the Team Data Science Process.

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 [Example walkthroughs](#) topic. They illustrate how to combine cloud, on-premises tools, and services into a workflow or pipeline to create an intelligent application.

Agile development of data science projects

11/29/2017 • 6 min to read • [Edit Online](#)

This document describes how developers can execute a data science project in a systematic, version controlled, and collaborative way within a project team by using the [Team Data Science Process](#) (TDSP). The TDSP is a framework developed by Microsoft that provides a structured sequence of activities to execute cloud-based, predictive analytics solutions efficiently. For an outline of the personnel roles, and their associated tasks that are handled by a data science team standardizing on this process, see [Team Data Science Process roles and tasks](#).

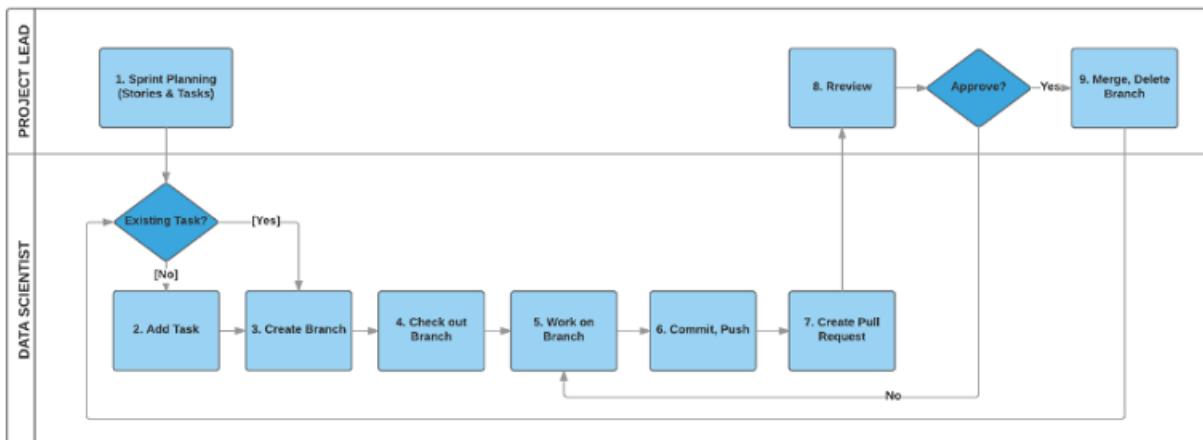
This article includes instructions on how to:

1. do **sprint planning** for work items involved in a project.
If you are unfamiliar with sprint planning, you can find details and general information [here](#).
2. **add work items** to sprints.

NOTE

The steps needed to set up a TDSP team environment using Visual Studio Team Services (VSTS) are outlined in the following set of instructions. They specify how to accomplish these tasks with VSTS because that is how to implement TDSP at Microsoft. If you choose to use VSTS, items (3) and (4) in the previous list are benefits that you get naturally. If another code hosting platform is used for your group, the tasks that need to be completed by the team lead generally do not change. But the way to complete these tasks is going to be different. For example, the item in section six, [Link a work item with a Git branch](#), might not be as easy as it is on VSTS.

The following figure illustrates a typical sprint planning, coding, and source-control workflow involved in implementing a data science project:



1. Terminology

In the TDSP sprint planning framework, there are four frequently used types of **work items**: **Feature**, **User Story**, **Task**, and **Bug**. Each team project maintains a single backlog for all work items. There is no backlog at the Git repository level under a team project. Here are their definitions:

- **Feature**: A feature corresponds to a project engagement. Different engagements with a client are considered different features. Similarly, it is best to consider different phases of a project with a client as different features. If you choose a schema such as **ClientName-EngagementName** to name your features, then you can easily recognize the context of the project/engagement from the names themselves.

- **Story:** Stories are different work items that are needed to complete a feature (project) end-to-end. Examples of stories include:
 - Getting Data
 - Exploring Data
 - Generating Features
 - Building Models
 - Operationalizing Models
 - Retraining Models
- **Task:** Tasks are assignable code or document work items or other activities that need to be done to complete a specific story. For example, tasks in the story *Getting Data* could be:
 - Getting Credentials of SQL Server
 - Uploading Data to SQL Data Warehouse.
- **Bug:** Bugs usually refer to fixes that are needed for an existing code or document that are done when completing a task. If the bug is caused by missing stages or tasks respectively, it can escalate to being a story or a task.

NOTE

Concepts are borrowed of features, stories, tasks, and bugs from software code management (SCM) to be used in data science. They might differ slightly from their conventional SCM definitions.

NOTE

Data scientists may feel more comfortable using an agile template that specifically aligns with the TDSP lifecycle stages. With that in mind, an Agile-derived sprint planning template has been created, where Epics, Stories etc. are replaced by TDSP lifecycle stages or substages. For instructions on how to create an agile template, see [Set up agile data science process in Visual Studio Online](#).

2. Sprint planning

Sprint planning is useful for project prioritization, and resource planning and allocation. Many data scientists are engaged with multiple projects, each of which can take months to complete. Projects often proceed at different paces. On the VSTS server, you can easily create, manage, and track work items in your team project and conduct sprint planning to ensure that your projects are moving forward as expected.

Follow [this link](#) for the step-by-step instructions on sprint planning in VSTS.

3. Add a feature

After your project repository is created under a team project, go to the team **Overview** page and click **Manage work**.

The screenshot shows the Visual Studio Team Services dashboard for a project named 'Client_Project_1'. The 'WORK' tab is selected. The 'Overview' section contains several cards:

- Welcome**: Get started using Visual Studio Team Services to make the most of your team dashboard.
- Manage Work**: Add work to your board (highlighted with a red box).
- Collaborate on code**: Add code to your repository.
- Continuously integrate**: Automate your builds.
- Visualize progress**: Learn how to add charts.

The 'Work assigned to Wei Guo (0)' card shows a large purple '0' icon.

To include a feature in the backlog, click **Backlogs** --> **Features** --> **New**, type in the feature **Title** (usually your project name), and then click **Add**.

The screenshot shows the 'Features' backlog view. The 'Backlogs' tab is selected. A new feature is being created with the following details:

Type	Feature
Title	Client Project 1

The 'Add' button is highlighted with a red box.

Double-click the feature you created. Fill in the descriptions, assign team members for this feature, and set planning parameters for this feature.

You can also link this feature to the project repository. Click **Add link** under the **Development** section. After you have finished editing the feature, click **Save & Close** to exit.

4. Add Story under feature

Under the feature, stories can be added to describe major steps needed to finish the (feature) project. To add a new story, click the + sign to the left of the feature in backlog view.

The screenshot shows the Microsoft Team Services interface for a project named "Client_Project_1". The "WORK" tab is selected. In the "Backlogs" section, a "Features" item is selected. A modal window titled "Features" is open, showing a table with one row. The table has columns for "Order", "Work Item Type", and "Title". The first row has an "Order" of 1, a "Work Item Type" of "Feature", and a "Title" of "Client Project 1". A red box highlights the "+" button in the top-left corner of the table header.

You can edit the details of the story, such as the status, description, comments, planning, and priority in the pop-up window.

The screenshot shows the Microsoft Team Services interface for a project named "Client_Project_1". The "WORK" tab is selected. In the "Backlogs" section, a "Stories" item is selected. A modal window titled "NEW STORY" is open, showing a form for a story titled "Data Ingestion". The form includes fields for "Status" (set to "New"), "Area" (set to "Client_Project_1"), "Reason" (set to "New"), "Iteration" (set to "Client_Project_1"), "Description" (with a rich text editor), "Acceptance Criteria" (with a rich text editor), "Discussion" (with a rich text editor), "Planning" (with fields for "Story Points", "Priority", and "Risk"), and "Development" (with a link to "Add link"). A red box highlights the "Save & Close" button in the top-right corner.

You can link this story to an existing repository by clicking + Add link under Development.

The screenshot shows the Microsoft Team Services interface for a project named "Client_Project_1". The "WORK" tab is selected. In the "Backlogs" section, a "Stories" item is selected. A modal window titled "Add Link to User Story 27: Feature Engineering" is open, showing a "Select the link type and the details" dialog. The "Link type" dropdown is set to "Repository". The "Branch" dropdown is set to "Client_Project_1". The "Branch" dropdown is expanded, showing "master" and "data-ingestion". A red box highlights the "Branch" dropdown. The "Development" section of the main window is also highlighted with a red box, showing a link to "Add link".

5. Add a task to a story

Tasks are specific detailed steps that are needed to complete each story. After all tasks of a story are completed, the story should be completed too.

To add a task to a story, click the + sign next to the story item, select **Task**, and then fill in the detailed information

of this task in the pop-up window.

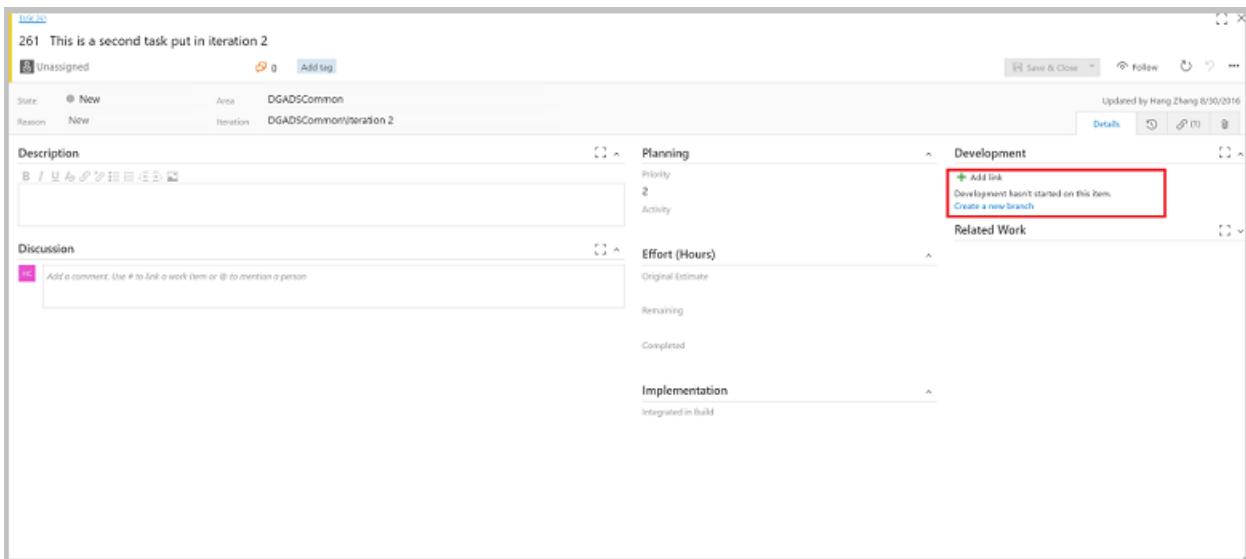
The screenshot shows the Microsoft Team Services interface for a project named "Client_Project_1". The "WORK" tab is selected. On the left, there's a navigation pane with "Backlogs" and "Queries" tabs, and sections for "Features", "Stories", "Current", "Iteration 1", "Future", "Iteration 2", and "Iteration 3". The main area is titled "Features" and shows a "Backlog" view. A modal dialog is open for creating a new item, with "Type" set to "Feature". The "Title" field is empty. Below the title, there are buttons for "Task" and "Bug", with "Task" highlighted and a red box drawn around it. The backlog table has columns for "Order", "Work Item Type", "Title", and "State". It lists one feature item under "Client Project 1" and several user stories under "Data Ingestion".

Order	Work Item Type	Title	State
1	Feature	Client Project 1	New
	User Story	Data Ingestion	New
	User Story	Feature Engineering	New
	User Story	Modeling	New
	User Story	Model deployment	New

After the features, stories, and tasks are created, you can view them in the **Backlog** or **Board** views to track their status.

This screenshot shows the same Microsoft Team Services interface after some items have been added. The "Backlog" view is still active. The backlog table now includes a feature item for "Data Ingestion" with two child tasks: "Get access to project database" and "Data Exploration". The "Data Exploration" task is currently selected, highlighted with a blue background. Other items like "Feature Engineering", "Modeling", and "Model deployment" remain as user stories under the "Data Ingestion" feature.

Order	Work Item Type	Title	State
1	Feature	Client Project 1	New
	User Story	Data Ingestion	New
	Task	Get access to project database	New
	Task	Data Exploration	New
	User Story	Feature Engineering	New
	User Story	Modeling	New
	User Story	Model deployment	New

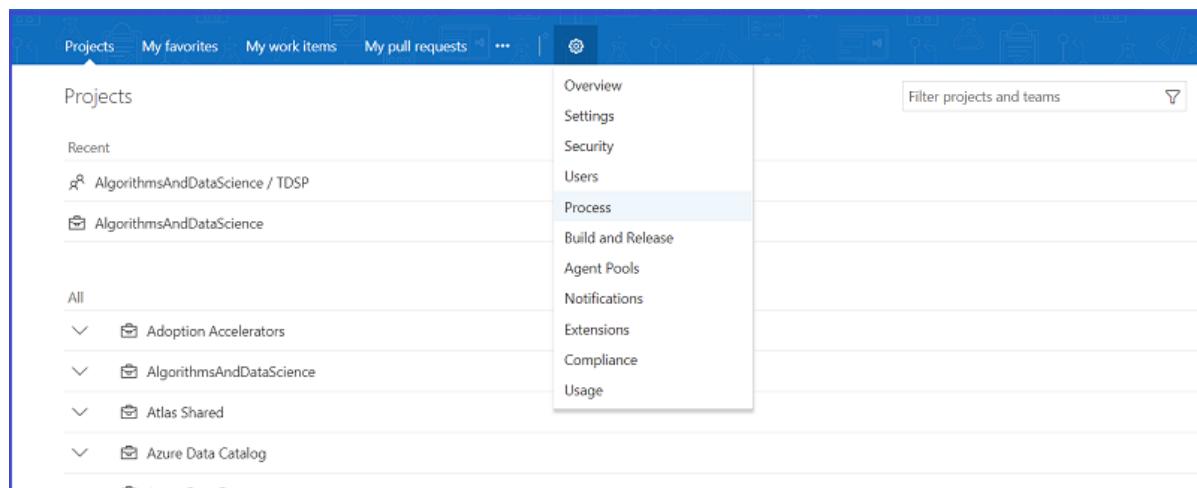


6. Set up an Agile TDSP work template in Visual Studio Online

This article explains how to set up an agile data science process template that uses the TDSP data science lifecycle stages and tracks work items with Visual Studio Online (vso). The steps below walk through an example of setting up the data science-specific agile process template *AgileDataScienceProcess* and show how to create data science work items based on the template.

Agile Data Science Process Template Setup

1. Navigate to server homepage, **Configure** -> **Process**.



2. Navigate to **All processes** -> **Processes**, under **Agile** and click on **Create inherited process**. Then put the process name "AgileDataScienceProcess" and click **Create process**.

Create inherited process from Agile

Create a new inherited process to enable customizations.

Agile [system process]

AgileDataScienceProcess

Description

Learn more

Create process **Cancel**

3. Under the **AgileDataScienceProcess -> Work item types** tab, disable **Epic**, **Feature**, **User Story**, and **Task** work item types by **Configure -> Disable**

Work item types	Backlog levels	Projects
+ New work item type		
Name	Description	
Bug	Describes a divergence betw...	
Epic	... Epics help teams effectively...	
Feature	... will be...	
Issue		
Task	Tracks work that needs to b...	

The screenshot shows the 'Work item types' tab of the AgileDataScienceProcess configuration. It lists several work item types: Bug, Epic, Feature, Issue, and Task. The 'Epic' row is currently selected. A context menu is open over the 'Epic' row, with the 'Disable' option highlighted. Other rows like 'Bug' and 'Task' also have their descriptions partially visible.

4. Navigate to **AgileDataScienceProcess -> Backlog levels** tab. Rename "Epics" to "TDSP Projects" by clicking on the **Configure -> Edit/Rename**. In the same dialog box, click **+New work item type** in "Data

Science Project" and set the value of **Default work item type** to "TDSP Project"

Edit backlog level

The following fields are automatically added to all work item types on the Portfolio backlogs: Stack Rank

Name

 TDSP Projects 

Work item types on this backlog level

 Epic (disabled)

 TDSP Project

 New work item type

Default work item type



[Save](#) [Cancel](#)

5. Similarly, change Backlog name "Features" to "TDSP Stages" and add the following to the **New work item type**:

- Business Understanding
- Data Acquisition
- Modeling
- Deployment

6. Rename "User Story" to "TDSP Substages" with default work item type set to newly created "TDSP Substage" type.

7. Set the "Tasks" to newly created Work item type "TDSP Task"

8. After these steps, the Backlog levels should look like this:

Work item types	Backlog levels	Projects
Portfolio backlog		
Portfolio backlogs provide a way to group related items into a hierarchical structure. You can rename and edit any portfolio backlog level.		
	+ New top level portfolio backlog	
Backlog	Work item types	
TDSP Projects	Epic (disabled)	
	TDSP Project	
TDSP Stages	Business Understanding	
	Data Aquisition	
	Deployment	
	Feature (disabled)	
	Modeling	
Requirement backlog		
The requirement backlog level contains your base level work items. There is only one requirement backlog and it cannot be removed, but can be renamed and edited.		
Backlog	Work item types	
TDSP Substages	TDSP Substage	
	User Story (disabled)	
Iteration backlog		
The iteration backlog contains your task work items. There is only one level of iteration backlog and it cannot be removed. The iteration backlog does not have an associated color.		
Backlog	Work item types	
Tasks	Task (disabled)	
	TDSP Task	

Create Data Science Work Items

After the data science process template is created, you can create and track your data science work items that correspond to the TDSP lifecycle.

- When you create a new team project, select "Agile\AgileDataScienceProcess" as the **Work item process**:



Create new project

Projects contain your source code, work items, automated builds and more.

Project name *

TDSP CustomerX Project



Description

Version control

Git



Work item process

Agile\AgileDataScienceProcess



[Show description](#)

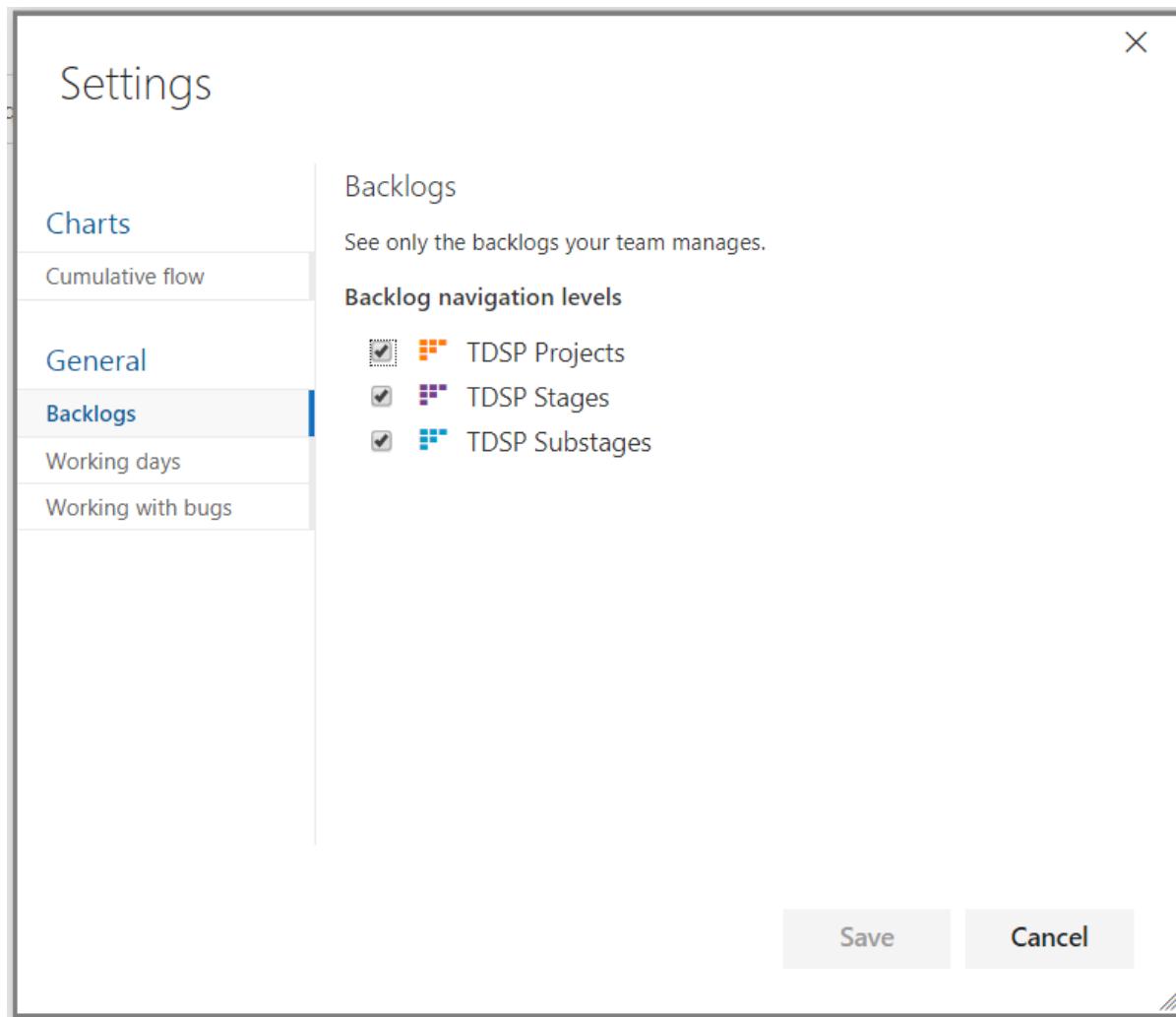
Share with

Everyone in Microsoft

Create

Cancel

2. Navigate to the newly created team project, and click on **Work -> Backlogs**.
3. Make "TDSP Projects" visible by clicking on **Configure team settings** and check "TDSP Projects"; then save.



4. Now you can start creating the data science-specific work items.

The screenshot shows the 'TDSP Projects' backlog board. The left sidebar has 'Backlogs' (selected) and 'Queries'. Under 'Backlogs', it lists 'TDSP Projects' (selected), 'TDSP Stages', and 'TDSP Substages'. It also shows 'Current' and 'Future' sections with 'Iteration 1', 'Iteration 2', and 'Iteration 3'. The main board area has tabs for 'Backlog' and 'Board'. The 'Board' tab is active, showing columns for 'New', '+', 'Create query', 'Column options', and 'Email'. The backlog table has columns for 'Order', 'Work Item Type', and 'Title'.

5. Here is an example of how the data science project work items should appear:

The screenshot shows the Azure DevOps Backlog board for a project named "TDSP Projects". The backlog is organized into iterations: Current, Future, Iteration 1, Iteration 2, and Iteration 3. The tasks are categorized by work item type: TDSP Project, TDSP Substage, TDSP Task, and Data Aquisition. Each task has a title, a detailed description, and a status (New). The tasks are listed in a hierarchical manner, with sub-tasks under their respective parent tasks.

Iteration	Work Item Type	Title	State	Effort
Current	TDSP Project	Fraud Detection CompanyABC	New	
Future	Business Understanding	Define Objectives	New	
Iteration 1	TDSP Substage	Identify business variables	New	
Iteration 1	TDSP Substage	Define success metrics	New	
Iteration 1	TDSP Task	Define accuracy	New	
Iteration 1	Data Aquisition	Ingest data	New	
Iteration 1	Data Aquisition	Explore data	New	
Iteration 1	TDSP Substage	Apply IDEAR to check data quality	New	
Iteration 1	TDSP Task	Find missing data	New	
Iteration 1	TDSP Task	Find numerical data distribution	New	
Iteration 1	Modeling	Feature engineering	New	
Iteration 1	TDSP Substage	Derive features	New	
Iteration 1	TDSP Task	Compute categorical features	New	
Iteration 1	TDSP Task	Compute statistical features	New	
Iteration 1	Modeling	Model creation	New	
Iteration 1	TDSP Substage	Traing model	New	
Iteration 1	TDSP Task	Split data	New	
Iteration 1	TDSP Task	sweep parameter over a model	New	
Iteration 1	TDSP Substage	Evaluate model	New	
Iteration 1	TDSP Task	Select model eval metrics	New	
Iteration 1	TDSP Task	Generate model performance n...	New	
Iteration 1	Deployment	Operationalize model	New	
Iteration 1	TDSP Substage	Deploy model as a web service	New	
Iteration 1	TDSP Task	Set up web service with Azure c...	New	

Next steps

[Collaborative coding with Git](#) describes how to do collaborative code development for data science projects using Git as the shared code development framework and how to link these coding activities to the work planned with the agile process.

Here are additional links to resources on agile processes.

- Agile process <https://www.visualstudio.com/en-us/docs/work/guidance/agile-process>
- Agile process work item types and workflow <https://www.visualstudio.com/en-us/docs/work/guidance/agile-process-workflow>

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 [Example walkthroughs](#) article. They illustrate how to combine cloud, on-premises tools, and services into a workflow or pipeline to create an intelligent application.

Collaborative coding with Git

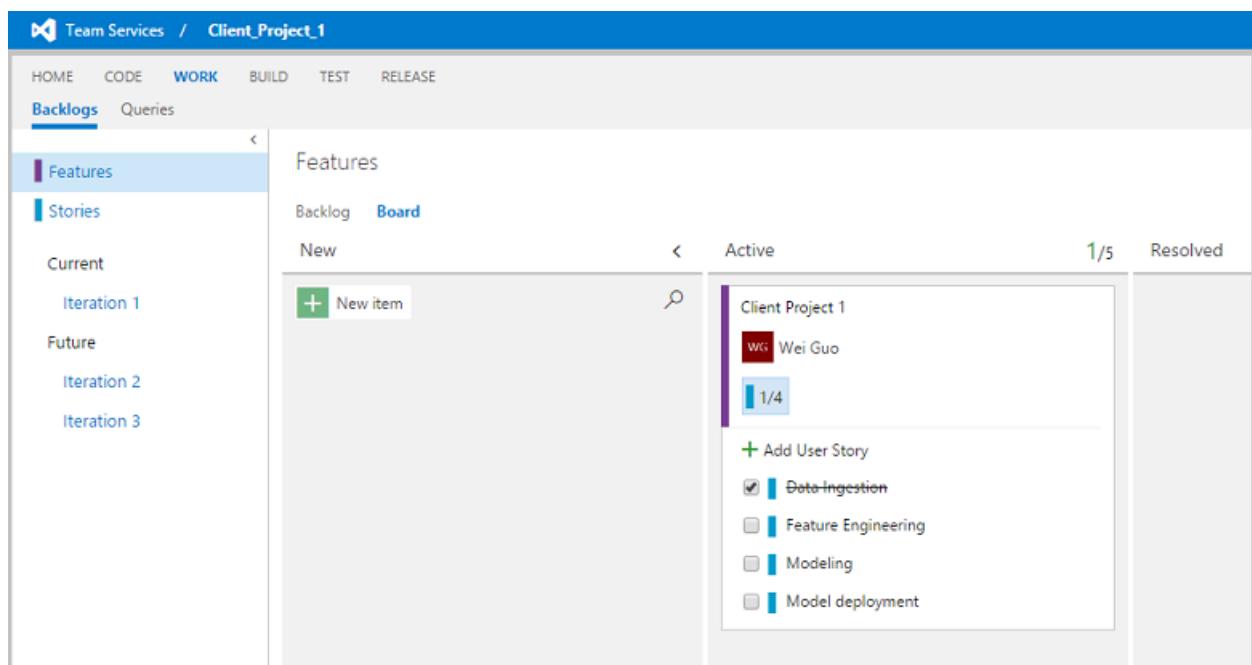
11/29/2017 • 4 min to read • [Edit Online](#)

In this article we describe how to do collaborative code development for data science projects using Git as the shared code development framework. It covers how to link these coding activities to the work planned in [Agile development](#) and how to do code reviews.

1. Link a work item with a Git branch

VSTS provides a convenient way to connect a work item (a story or task) with a Git branch. This enables you to link your story or task directly to the code associated with it.

To connect a work item to a new branch, double-click a work item, and in the pop-up window, click **Create a new branch** under **+ Add link**.



Provide the information for this new branch, such as the branch name, base Git repository, and the branch. The Git repository chosen must be the repository under the same team project that the work item belongs to. The base branch can be the master branch or some other existing branch.

X

Create a branch

Name

Enter your branch name

Based on

Utilities▼master▼

Work items to link

Search work items by ID or title ▼

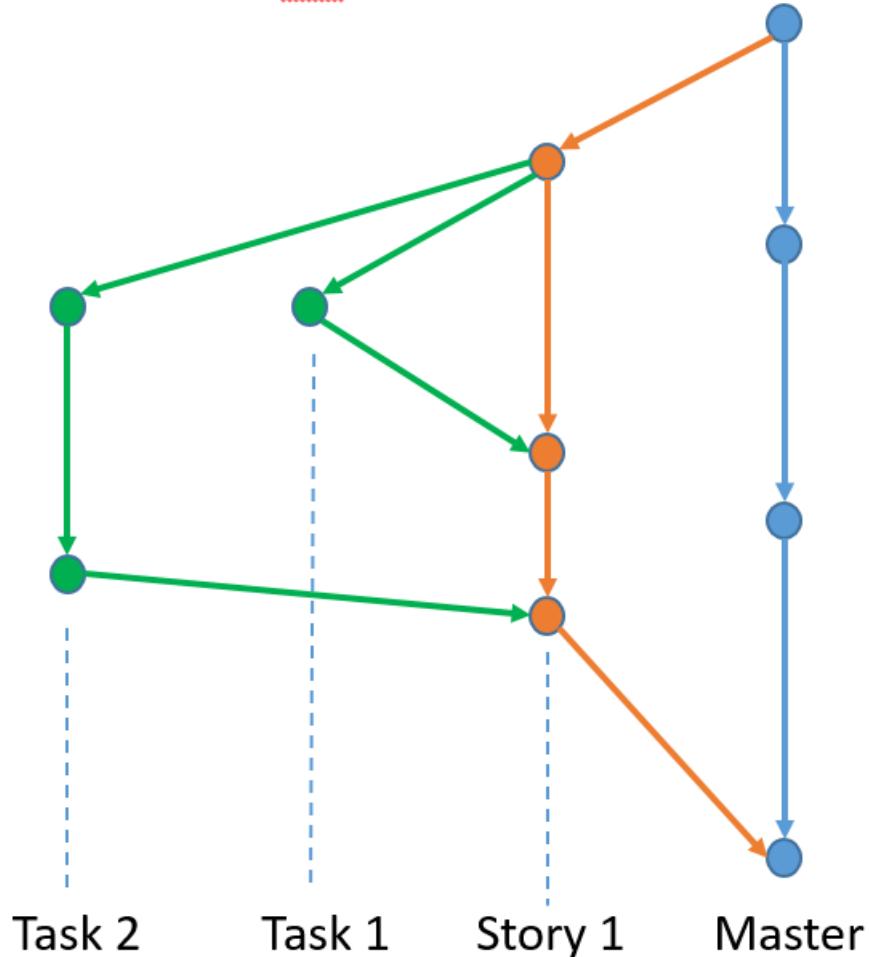
! 261 This is a second task put in iteration 2
Updated 8/30/2016, ● New

Create branchCancel

A good practice is to create a Git branch for each story work item. Then, for each task work item, you create a branch based on the story branch. Organizing the branches in this hierarchical way that corresponds to the story-task relationships is helpful when you have multiple people working on different stories of the same project, or you have multiple people working on different tasks of the same story. Conflicts can be minimized when each team member works on a different branch and when each member works on different codes or other artifacts when sharing a branch.

The following picture depicts the recommended branching strategy for TDSP. You might not need as many branches as are shown here, especially when you only have one or two people working on the same project, or only one person works on all tasks of a story. But separating the development branch from the master branch is always a good practice. This can help prevent the release branch from being interrupted by the development activities. More complete description of Git branch model can be found in [A Successful Git Branching Model](#).

Git Branches



To switch to the branch that you want to work on, run the following command in a shell command (Windows or Linux).

```
git checkout <branch name>
```

Changing the *<branch name>* to **master** switches you back to the **master** branch. After you switch to the working branch, you can start working on that work item, developing the code or documentation artifacts needed to complete the item.

You can also link a work item to an existing branch. In the **Detail** page of a work item, instead of clicking **Create a new branch**, you click **+ Add link**. Then, select the branch you want to link the work item to.



Add Link to Task 261: This is a second task put in it...

Select the link type and the details.

Link type

Branch

Repository

Utilities ▾

Branch

dev ▾

Comment

OK

Cancel

You can also create a new branch in Git Bash commands. If <base branch name> is missing, the <new branch name> is based on *master* branch.

```
git checkout -b <new branch name> <base branch name>
```

2. Work on a branch and commit the changes

Now suppose you make some change to the *data_ingestion* branch for the work item, such as adding an R file on the branch in your local machine. You can commit the R file added to the branch for this work item, provided you are in that branch in your Git shell, using the following Git commands:

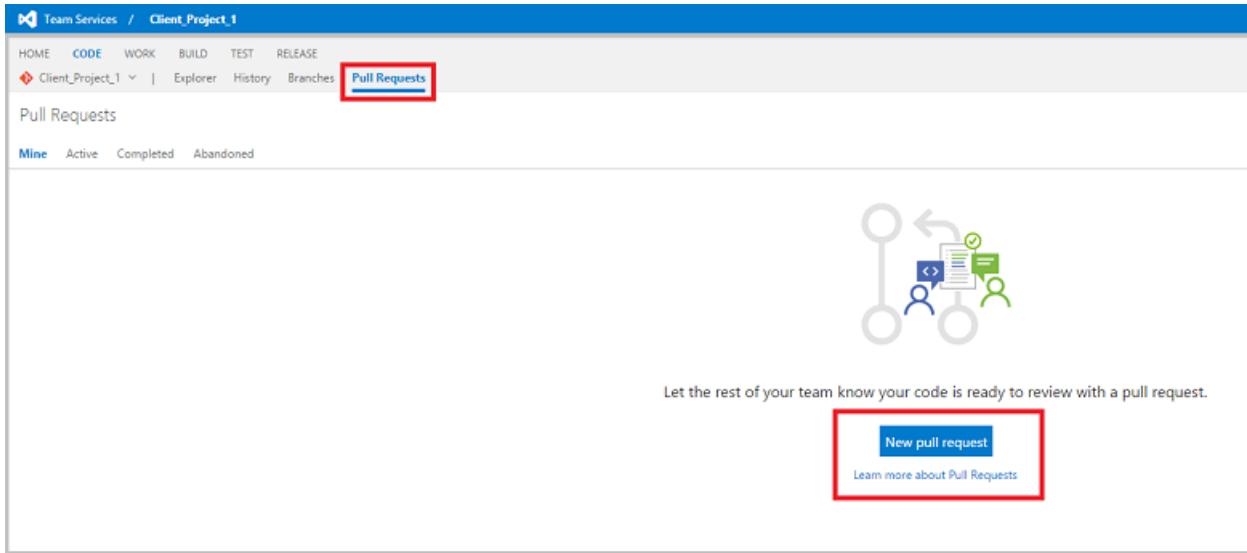
```
git status  
git add .  
git commit -m"added a R scripts"  
git push origin data_ingestion
```

```
PS D:\AML_Projects\TDSP-Linux\sprint_test\Client_Project_1\Code> git status  
On branch data_ingestion  
Untracked files:  
  (use "git add <file>..." to include in what will be committed)  
  
    process_data.R  
  
nothing added to commit but untracked files present (use "git add" to track)  
PS D:\AML_Projects\TDSP-Linux\sprint_test\Client_Project_1\Code> git add .\process_data.R  
PS D:\AML_Projects\TDSP-Linux\sprint_test\Client_Project_1\Code> git commit -m"Wei added a R script"  
[data_ingestion a5a2ff0] Wei added a R script  
 1 file changed, 5 insertions(+)  
  create mode 100644 Code/process data.R  
PS D:\AML_Projects\TDSP-Linux\sprint_test\Client_Project_1\Code> git push origin data_ingestion  
Counting objects: 4, done.  
Delta compression using up to 8 threads.  
Compressing objects: 100% (4/4), done.  
Writing objects: 100% (4/4), 393 bytes | 0 bytes/s, done.  
Total 4 (delta 2), reused 0 (delta 0)  
remote: Analyzing objects... (4/4) (12 ms)  
remote: Storing packfile... done (37 ms)  
remote: Storing index... done (66 ms)  
To https://weig-ds.visualstudio.com/_git/Client_Project_1  
  c3c5ee4..a5a2ff0  data ingestion -> data ingestion
```

3. Create a pull request on VSTS

When you are ready after a few commits and pushes, to merge the current branch into its base branch, you can submit a **pull request** on VSTS server.

Go to the main page of your team project and click **CODE**. Select the branch to be merged and the Git repository name that you want to merge the branch into. Then click **Pull Requests**, click **New pull request** to create a pull request review before the work on the branch is merged to its base branch.



Fill in some description about this pull request, add reviewers, and send it out.

Please review the R script

Description

- add a note
- Wei added a R script

Reviewers

yuso Search users and groups

Work items

Search work items by ID or title

25 Data Ingestion Updated 59 minutes ago, Closed

New pull request fewer options

COMMITS (2)

Client_Project_1

Wednesday, August 31, 2016

- Wei added a R script a5a2ff0f by Wei Guo, 21 minutes ago
- add a note c3c5ee4f by Wei Guo, an hour ago

FILES (2)

Showing: All files

Notes.txt /Notes.txt

```
1 This is a note.
```

process_data.R /Code/process_data.R

```
1 X <- rnorm(1000)
2 y <- 3*x + rnorm(1000)
3 plot(x,y)
4 ln(y~x)
5
6
```

4. Review and merge

When the pull request is created, your reviewers get an email notification to review the pull requests. The reviewers need to check whether the changes are working or not and test the changes with the requester if possible. Based on their assessment, the reviewers can approve or reject the pull request.

Discussion Files (1) Commits (1)

- weig push from local branch

WG Please review my R code,thanks!

Wei Guo - an hour ago

Wei Guo commented on the file `wei_explore_data.R`

```

^ 9
10 #work on branch
8 11 y2 <- 3*x+2+rnorm(1000)
9 12 lm(y2~x)

```

WG Can we compare the model estimates?
Wei Guo - an hour ago - reply

Status: Resolved ▾

Y it works
yuso - 55 minutes ago

yuso closed the pull request with commit `c226dc97`
54 minutes ago

Wei Guo commented on the file `wei_explore_data.R`

```

1 1 x <- rnorm(1000)
2 2 y <- 3*x+rnorm(1000)
3 hist(x)

```

WG we can make more plots to explore
Wei Guo - 39 minutes ago - reply

Status: Active ▾

```

4 4 plot(x,y)
5 5 lm(y~x)
6 6

```

Wei Guo - save - cancel - workitems (#) - people (@)

Team Services / Client_Project_1

HOME CODE WORK BUILD TEST RELEASE

Client_Project_1 | Explorer History Branches Pull Requests

1 active 3 Please review the R script

Wei Guo IP data_ingestion into master

Overview Files Updates Comments

All changes ▾ Group ▾ All ▾ Showing All files

Client_Project_1

- Notes.txt [-]
- Code
- process_data.R [-]
- plot y vs x [last rev]

process_data.R

```

1 x <- rnorm(1000)
2 y <- 3*x + rnorm(1000)
3 hist(x,y)
4 lm(y~x)
5

```

Wei Guo 3 minutes ago plot y vs x

Wei Guo - reply

Approve Approve with suggestions Wait for author Reject Reset feedback

After the review is done, the working branch is merged to its base branch by clicking the **Complete** button. You may choose to delete the working branch after it has merged.

Team Services / Client_Project_1

HOME CODE WORK BUILD TEST RELEASE

Client_Project_1 | Explorer History Branches Pull Requests

1 active 3 Please review the R script

Wei Guo IP data_ingestion into master

Overview Files Updates Comments

All changes ▾ Group ▾ All ▾ Showing All files

Client_Project_1

- Notes.txt [-]
- Code
- process_data.R [-]
- plot y vs x [last rev]

process_data.R

```

1 x <- rnorm(1000)
2 y <- 3*x + rnorm(1000)
3 hist(x,y)
4 lm(y~x)

```

Wei Guo 3 minutes ago plot y vs x

Wei Guo - reply

Approve Approve with suggestions Wait for author Reject Complete

Complete pull request

PR 3: Please review the R script

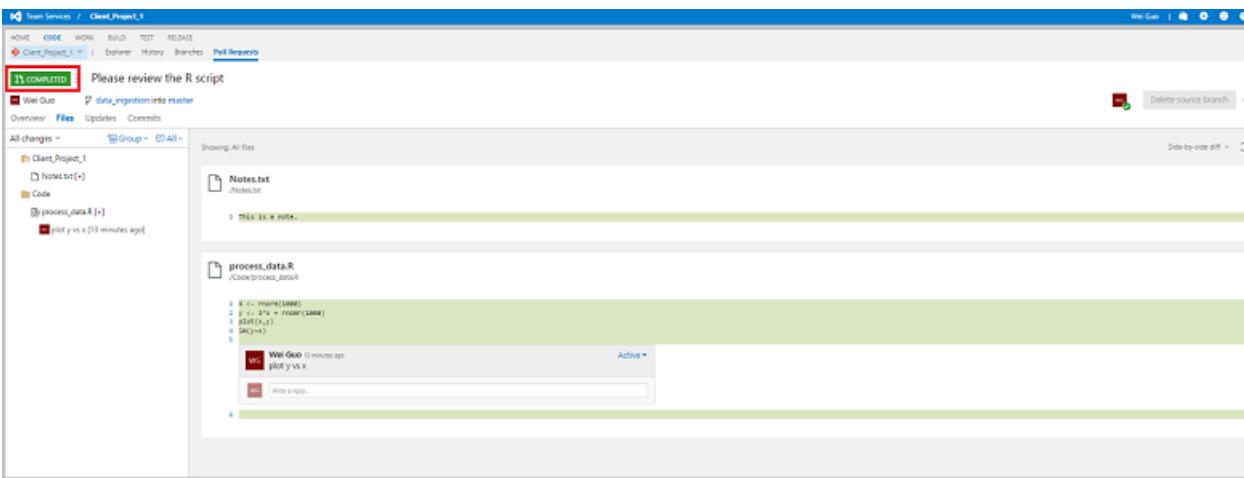
Add a note → We added a R script

Related work items #25

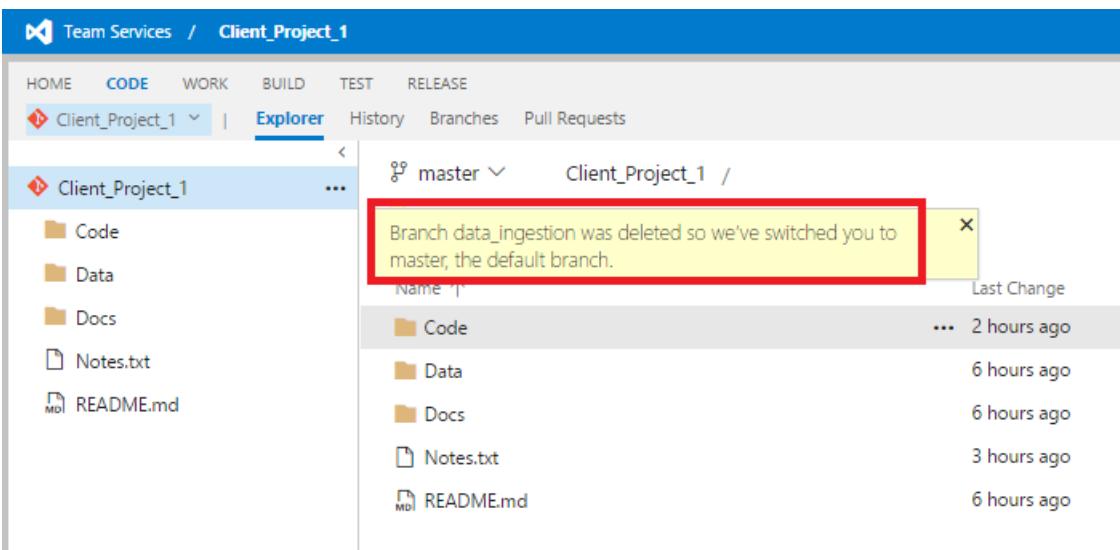
Delete data_ingestion after merging ↗ squash changes when merging

Complete merge Cancel

Confirm on the top left corner that the request is marked as **COMPLETED**.



When you go back to the repository under **CODE**, you are told that you have been switched to the master branch.



You can also use the following Git commands to merge your working branch to its base branch and delete the working branch after merging:

```
git checkout master
git merge data_ingestion
git branch -d data_ingestion
```

```
PS D:\AML_Projects\TDSP-Linux\sprint_test\Client_Project_1\Code> git checkout master
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.
PS D:\AML_Projects\TDSP-Linux\sprint_test\Client_Project_1\Code> git merge data_ingestion
Updating 22a62d5..82533d4
Fast-forward
 Code/process_data.R | 8 ++++++++
 Notes.txt          | 1 +
 2 files changed, 9 insertions(+)
 create mode 100644 Code/process_data.R
 create mode 100644 Notes.txt
PS D:\AML_Projects\TDSP-Linux\sprint_test\Client_Project_1\Code> git branch -d data_ingestion
Deleted branch data_ingestion (was 82533d4).
```

Next steps

[Execute of data science tasks](#) shows how to use utilities to complete several common data science tasks such as interactive data exploration, data analysis, reporting, and model creation.

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 [Example walkthroughs](#) article. They illustrate how to combine

cloud, on-premises tools, and services into a workflow or pipeline to create an intelligent application.

Execute data science tasks: exploration, modeling, and deployment

2/13/2018 • 5 min to read • [Edit Online](#)

Typical data science tasks include data exploration, modeling, and deployment. This article shows how to use the **Interactive Data Exploration, Analysis, and Reporting (IDEAR)** and **Automated Modeling and Reporting (AMAR)** utilities to complete several common data science tasks such as interactive data exploration, data analysis, reporting, and model creation. It also outlines options for deploying a model into a production environment using a variety of toolkits and data platforms, such as the following:

- [Azure Machine Learning](#)
- [SQL-Server with ML services](#)
- [Microsoft Machine Learning Server](#)

1. Exploration

A data scientist can perform exploration and reporting in a variety of ways: by using libraries and packages available for Python (matplotlib for example) or with R (ggplot or lattice for example). Data scientists can customize such code to fit the needs of data exploration for specific scenarios. The needs for dealing with structured data are different than for unstructured data such as text or images.

Products such as Azure Machine Learning Workbench also provide [advanced data preparation](#) for data wrangling and exploration, including feature creation. The user should decide on the tools, libraries, and packages that best suite their needs.

The deliverable at the end of this phase is a data exploration report. The report should provide a fairly comprehensive view of the data to be used for modeling and an assessment of whether the data is suitable to proceed to the modeling step. The Team Data Science Process (TDSP) utilities discussed in the following sections for semi-automated exploration, modeling, and reporting also provide standardized data exploration and modeling reports.

Interactive data exploration, analysis, and reporting using the IDEAR utility

This R markdown-based or Python notebook-based utility provides a flexible and interactive tool to evaluate and explore data sets. Users can quickly generate reports from the data set with minimal coding. Users can click buttons to export the exploration results in the interactive tool to a final report, which can be delivered to clients or used to make decisions on which variables to include in the subsequent modeling step.

At this time, the tool only works on data-frames in memory. A YAML file is needed to specify the parameters of the data-set to be explored. For more information, see [IDEAR in TDSP Data Science Utilities](#).

2. Modeling

There are numerous toolkits and packages for training models in a variety of languages. Data scientists should feel free to use whichever ones they are comfortable with, as long as performance considerations regarding accuracy and latency are satisfied for the relevant business use cases and production scenarios.

The next section shows how to use an R-based TDSP utility for semi-automated modeling. This AMAR utility can be used to generate base line models quickly as well as the parameters that need to be tuned to provide a better performing model. The following model management section shows how to have a system for registering and managing multiple models.

Model training: modeling and reporting using the AMAR utility

The [Automated Modeling and Reporting \(AMAR\) Utility](#) provides a customizable, semi-automated tool to perform model creation with hyper-parameter sweeping and to compare the accuracy of those models.

The model creation utility is an R Markdown file that can be run to produce self-contained HTML output with a table of contents for easy navigation through its different sections. Three algorithms are executed when the Markdown file is run (knit): regularized regression using the `glmnet` package, random forest using the `randomForest` package, and boosting trees using the `xgboost` package). Each of these algorithms produces a trained model. The accuracy of these models is then compared and the relative feature importance plots are reported. Currently, there are two utilities: one is for a binary classification task and one is for a regression task. The primary differences between them is the way control parameters and accuracy metrics are specified for these learning tasks.

A YAML file is used to specify:

- the data input (a SQL source or an R-Data file)
- what portion of the data is used for training and what portion for testing
- which algorithms to run
- the choice of control parameters for model optimization:
 - cross-validation
 - bootstrapping
 - folds of cross-validation
- the hyper-parameter sets for each algorithm.

The number of algorithms, the number of folds for optimization, the hyper-parameters, and the number of hyper-parameter sets to sweep over can also be modified in the Yaml file to run the models quickly. For example, they can be run with a lower number of CV folds, a lower number of parameter sets. If it is warranted, they can also be run more comprehensively with a higher number of CV folds or a larger number of parameter sets.

For more information, see [Automated Modeling and Reporting Utility in TDSP Data Science Utilities](#).

Model management

After multiple models have been built, you usually need to have a system for registering and managing the models. Typically you need a combination of scripts or APIs and a backend database or versioning system. A few options that you can consider for these management tasks are:

1. [Azure Machine Learning - model management service](#)
2. [ModelDB from MIT](#)
3. [SQL-severer as a model management system](#)
4. [Microsoft Machine Learning Server](#)

3. Deployment

Production deployment enables a model to play an active role in a business. Predictions from a deployed model can be used for business decisions.

Production platforms

There are various approaches and platforms to put models into production. Here are a few options:

- [Model deployment in Azure Machine Learning](#)
- [Deployment of a model in SQL-server](#)
- [Microsoft Machine Learning Server](#)

NOTE: Prior to deployment, one has to insure the latency of model scoring is low enough to use in production.

Further examples are available in walkthroughs that demonstrate all the steps in the process for **specific scenarios**. They are listed and linked with thumbnail descriptions in the [Example walkthroughs](#) article. They illustrate how to combine cloud, on-premises tools, and services into a workflow or pipeline to create an intelligent application.

NOTE: For deployment using Azure Machine Learning Studio, see [Deploy an Azure Machine Learning web service](#).

A/B testing

When multiple models are in production, it can be useful to perform [A/B testing](#) to compare performance of the models.

Next steps

[Track progress of data science projects](#) shows how a data scientist can track the progress of a data science project.

Track progress of data science projects

11/29/2017 • 1 min to read • [Edit Online](#)

Data science group managers, team leads, and project leads need to track the progress of their team projects, what work has been done on them and by whom, and remains on the to-do lists.

VSTS dashboards

If you are using Visual Studio Team Services (VSTS), you are able to build dashboards to track the activities and the work items associated with a given Agile project.

For more information on how to create and customize dashboards and widgets on Visual Studio Team Services, see the following sets of instructions:

- [Add and manage dashboards](#)
- [Add widgets to a dashboard](#).

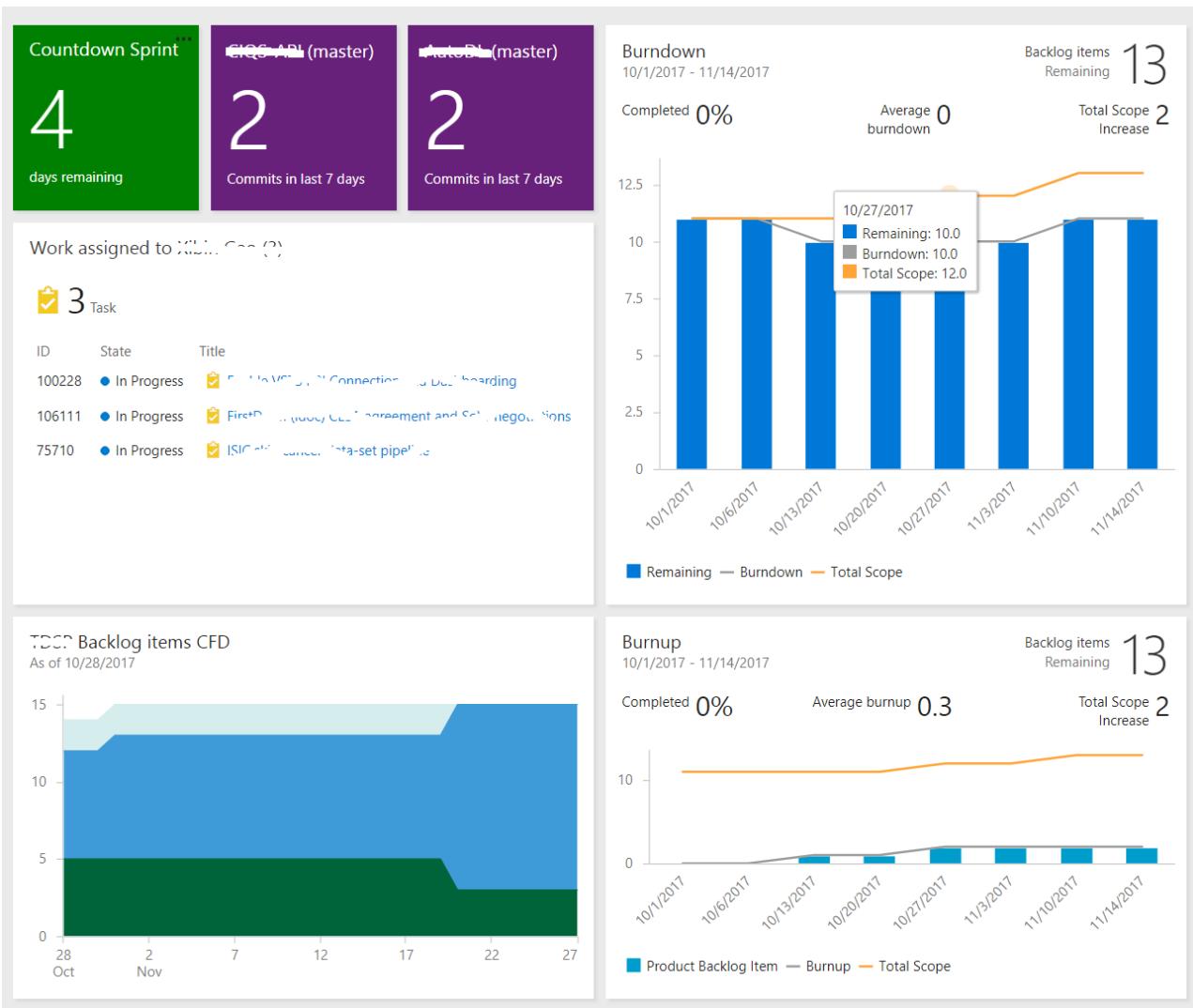
Example dashboard

Here is a simple example dashboard that is built to track the sprint activities of an Agile data science project, as well as the number of commits to associated repositories. The **top left** panel shows:

- the countdown of the current sprint,
- the number of commits for each repository in the last 7 days
- the work item for specific users.

The remaining panels show the cumulative flow diagram (CFD), burndown, and burnup for a project:

- **Bottom left:** CFD the quantity of work in a given state, showing approved in gray, committed in blue, and done in green.
- **Top right:** burndown chart the work left to complete versus the time remaining).
- **Bottom right:** burnup chart the work that has been completed versus the total amount of work.



For a description of how to build these charts, see the quickstarts and tutorials at [Dashboards](#).

Next steps

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 [Example walkthroughs](#) article. They illustrate how to combine cloud, on-premises tools, and services into a workflow or pipeline to create an intelligent application.

Walkthroughs executing the Team Data Science Process

9/25/2017 • 1 min to read • [Edit Online](#)

These **end-to-end walkthroughs** demonstrate the steps in the Team Data Science Process for specific scenarios. They illustrate how to combine cloud, on-premises tools, and services into a workflow or pipeline to create an **intelligent application**. The walkthroughs are grouped by **platform** that they use.

Walkthrough descriptions

Here are brief descriptions of what these walkthrough examples provide on their respective platforms:

- [Azure Machine Learning Workbench](#) These walkthrough tutorials and samples show how to create Team Data Science structured projects with Azure Machine Learning Workbench to do natural language processing and classification.
- [HDInsight Spark walkthroughs using PySpark and Scala](#) These walkthroughs use PySpark and Scala on an Azure Spark cluster to do predictive analytics.
- [HDInsight Hadoop walkthroughs using Hive](#) These walkthroughs use Hive with an HDInsight Hadoop cluster to do predictive analytics.
- [Azure Data Lake walkthroughs using U-SQL](#) These walkthroughs use U-SQL with Azure Data Lake to do predictive analytics.
- [SQL Server](#) These walkthroughs use SQL Server, SQL Server R Services, and SQL Server Python Services to do predictive analytics.
- [SQL Data Warehouse](#) These walkthroughs use SQL Data Warehouse to do predictive analytics.

Next steps

For a discussion of the key components that comprise the Team Data Science Process, see [Team Data Science Process overview](#).

For a discussion of the Team Data Science Process lifecycle that you can use to structure your data science projects, see [Team Data Science Process lifecycle](#). The lifecycle outlines the steps, from start to finish, that projects usually follow when they are executed.

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

Team Data Science structured projects in Azure Machine Learning Workbench

11/2/2017 • 1 min to read • [Edit Online](#)

These tutorials and samples show how to create Team Data Science structured projects with Azure Machine Learning Workbench. For an overview of the Team Data Science Process, see [Data Science Process](#). For an overview of Spark on HDInsight, see [Introduction to Spark on HDInsight](#).

Additional data science walkthroughs that execute the Team Data Science Process are grouped by the **platform** that they use. See [Walkthroughs executing the Team Data Science Process](#) for an itemization of these examples.

Classify UCI incomes in Azure Machine Learning Workbench

For a tutorial showing how to create a TDSP project in Azure Machine Learning Workbench, see [Team Data Science Process Tutorial: Classify UCI incomes in Azure Machine Learning Workbench](#)

Biomedical entity recognition using Natural Language Processing with Deep Learning

For a sample that uses a TDSP-instantiated project in Azure Machine Learning Workbench, see [Biomedical entity recognition using Natural Language Processing with Deep Learning](#)

Next steps

For a discussion of the key components that comprise the Team Data Science Process, see [Team Data Science Process overview](#).

For a discussion of the Team Data Science Process lifecycle that you can use to structure your data science projects, see [Team Data Science Process lifecycle](#). The lifecycle outlines the steps, from start to finish, that projects usually follow when they are executed.

Predict Twitter sentiment with word embeddings by using the Team Data Science Process

2/13/2018 • 6 min to read • [Edit Online](#)

This article shows you how to collaborate effectively by using the *Word2Vec* word embedding algorithm and the *Sentiment-Specific Word Embedding (SSWE)* algorithm to predict Twitter sentiment with [Azure Machine Learning](#). For more information on predicting Twitter sentiment polarity, see the [MachineLearningSamples-TwitterSentimentPrediction](#) repository on GitHub. The key to facilitating effective team collaboration on data-science projects is to standardize the structure and documentation of the projects with an established data-science lifecycle. The [Team Data Science Process \(TDSP\)](#) provides this type of structured [lifecycle](#).

Creating data-science projects with the *TDSP template* provides the standardized framework for Azure Machine Learning projects. Previously, the TDSP team released a [GitHub repository for the TDSP project structure and templates](#). Now Machine Learning projects that are instantiated with [TDSP templates for Azure Machine Learning](#) are enabled. For instructions, see how to use [TDSP structure projects with the TDSP template](#) in Azure Machine Learning.

Twitter sentiment polarity sample

This article uses a sample to show you how to instantiate and execute a Machine Learning project. The sample uses the TDSP structure and templates in Azure Machine Learning Workbench. The complete sample is provided in [this walkthrough](#). The modeling task predicts sentiment polarity (positive or negative) by using the text from tweets. This article outlines the data-modeling tasks that are described in the walkthrough. The walkthrough covers the following tasks:

- Data exploration, training, and deployment of a machine learning model that address the prediction problem that's described in the use case overview. [Twitter sentiment data](#) is used for these tasks.
- Execution of the project by using the TDSP template from Azure Machine Learning for this project. For project execution and reporting, the TDSP lifecycle is used.
- Operationalization of the solution directly from Azure Machine Learning in Azure Container Service.

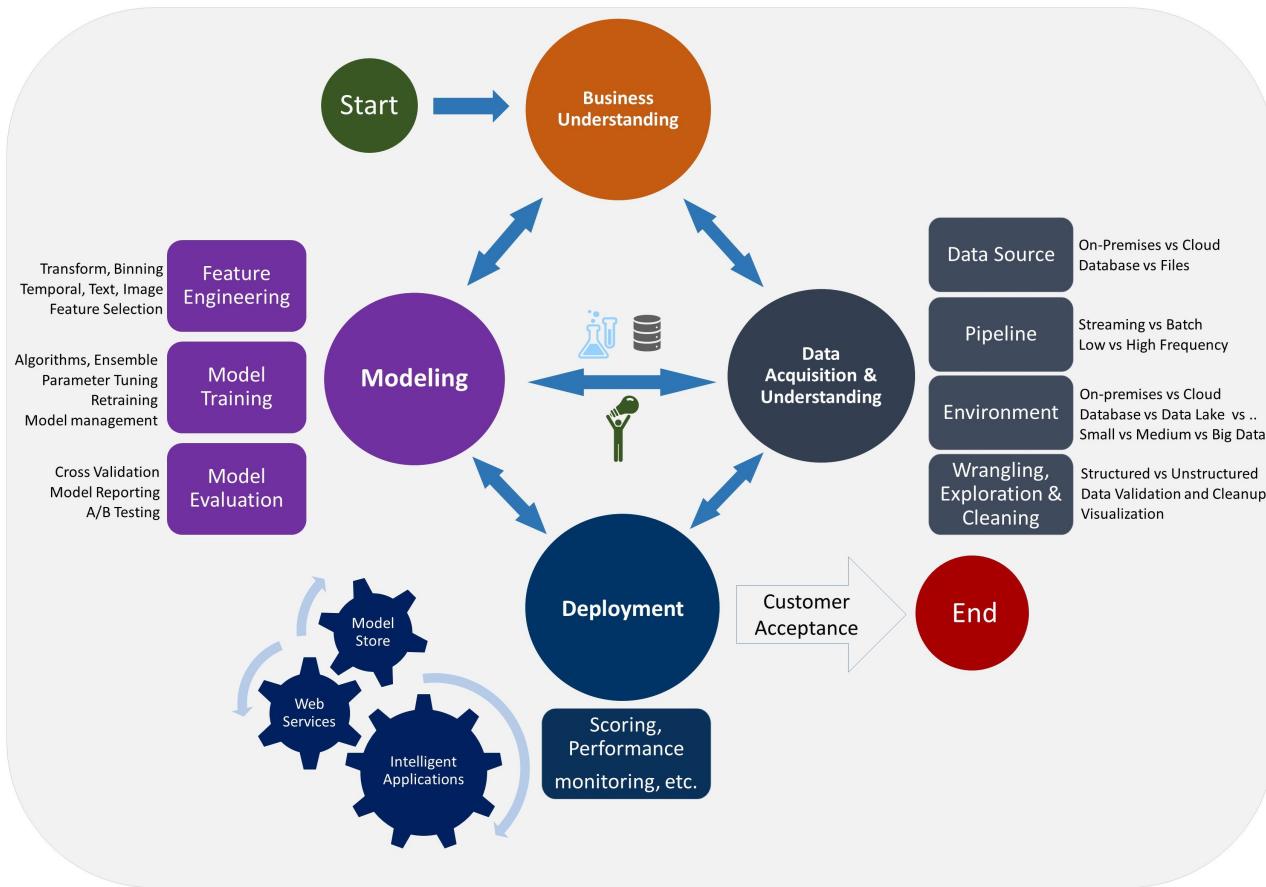
The project highlights the following features of Azure Machine Learning:

- Instantiation and use of the TDSP structure.
- Execution of code in Azure Machine Learning Workbench.
- Easy operationalization in Container Service by using Docker and Kubernetes.

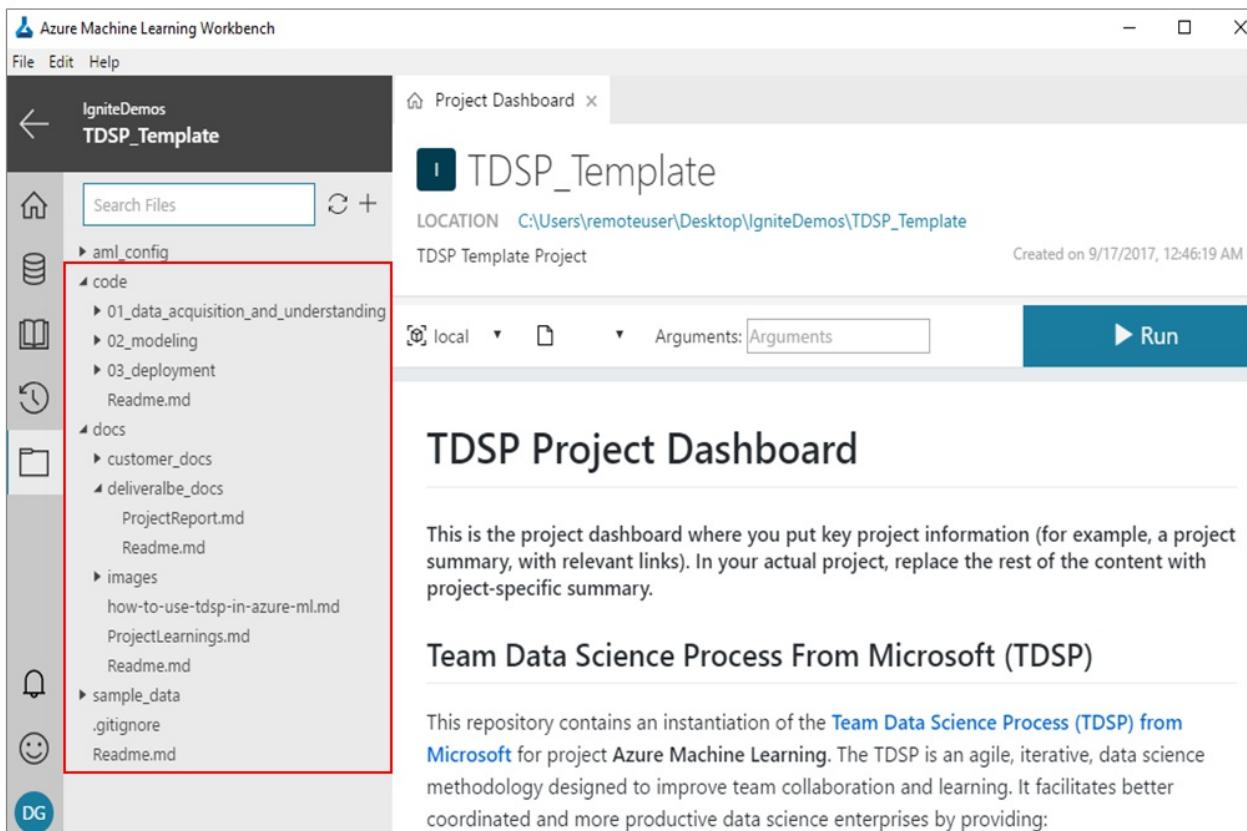
Team Data Science Process

To execute this sample, you use the TDSP project structure and documentation templates in Azure Machine Learning Workbench. The sample implements the [TDSP lifecycle](#), as shown in the following figure:

Data Science Lifecycle



The TDSP project is created in Azure Machine Learning Workbench based on [these instructions](#), as shown in the following figure:



Data acquisition and preparation

The first step in this sample is to download the sentiment140 dataset and divide the data into training and testing datasets. The sentiment140 dataset contains the actual content of the tweet (with emoticons removed). The dataset also contains the polarity of each tweet (negative=0, positive=4) with the neutral tweets removed. After the data is divided, the training data has 1.3 million rows and the testing data has 320,000 rows.

Model development

The next step in the sample is to develop a model for the data. The modeling task is divided into three parts:

- Feature engineering: Generate features for the model by using different word embedding algorithms.
- Model creation: Train different models to predict the sentiment of the input text. Examples of these models include *Logistic Regression* and *Gradient Boosting*.
- Model evaluation: Evaluate the trained models over the testing data.

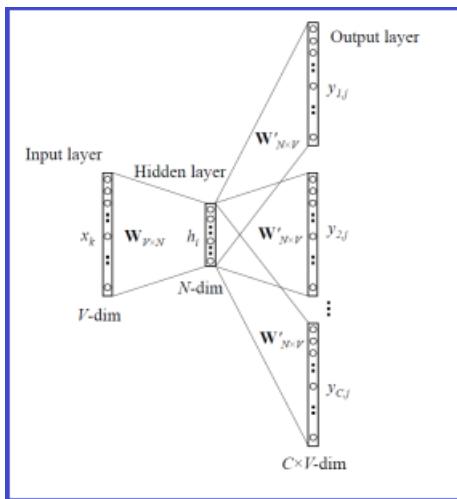
Feature Engineering

The Word2Vec and SSWE algorithms are used to generate word embeddings.

Word2Vec algorithm

The Word2Vec algorithm is used in the Skip-Gram model. This model is explained in the paper by Tomas Mikolov, et al. "[Distributed Representations of Words and Phrases and their Compositionality. Advances in neural information processing systems.](#)" 2013.

The Skip-Gram model is a shallow neural network. The input is the target word that's encoded as a one-hot vector, which is used to predict nearby words. If V is the size of the vocabulary, then the size of the output layer is $C \times V$ where C is the size of the context window. The following figure shows an architecture that's based on the Skip-Gram model:



The detailed mechanics of the Word2Vec algorithm and Skip-Gram model are beyond the scope of this sample. For more information, see the following references:

- [02_A_Word2Vec.py code with referenced TensorFlow examples](#)
- [Vector Representations of Words](#)
- [How exactly does word2vec work?](#)
- [Notes on Noise Contrastive Estimation and Negative Sampling](#)

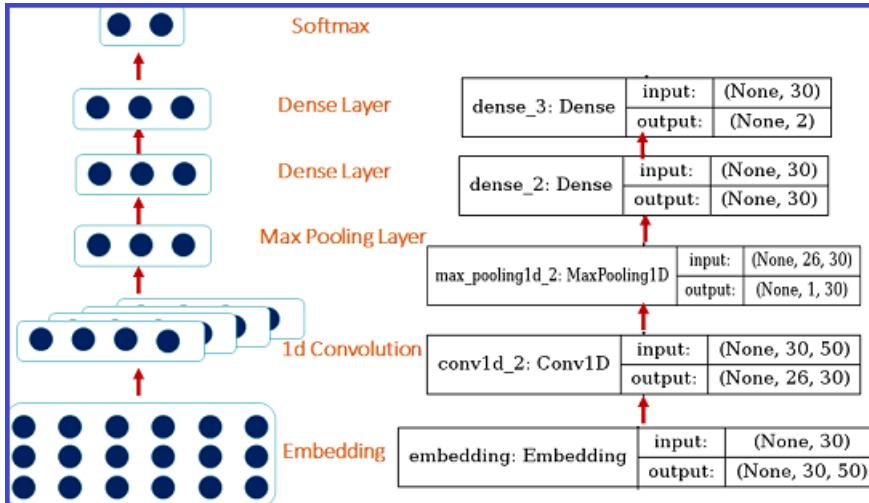
Sentiment-Specific Word Embedding algorithm

The SSWE algorithm tries to overcome a weakness of the Word2Vec algorithm where words with similar contexts and opposite polarity can have similar word vectors. The similarities can cause the Word2Vec algorithm to not perform accurately for tasks like sentiment analysis. The SSWE algorithm tries to handle this weakness by incorporating both the sentence polarity and the word's context into its loss function.

The sample uses a variant of the SSWE algorithm as follows:

- Both the original *ngram* and the corrupted *ngram* are used as inputs.
- A ranking style hinge loss function is used for both the syntactic loss and the semantic loss.
- The ultimate loss function is the weighted combination of both the syntactic loss and the semantic loss.
- For simplicity, only the semantic cross entropy is used as the loss function.

The sample shows that even with the simpler loss function, the performance of the SSWE embedding is better than the Word2Vec embedding. The following figure shows the convolutional model that's used to generate sentiment-specific word embedding:



After the training process is done, two embedding files in the tab-separated values (TSV) format are generated for the modeling stage.

For more information about the SSWE algorithms, see the paper by Duyu Tang, et al. "[Learning Sentiment-Specific Word Embedding for Twitter Sentiment Classification](#)." Association for Computational Linguistics (1). 2014.

Model creation

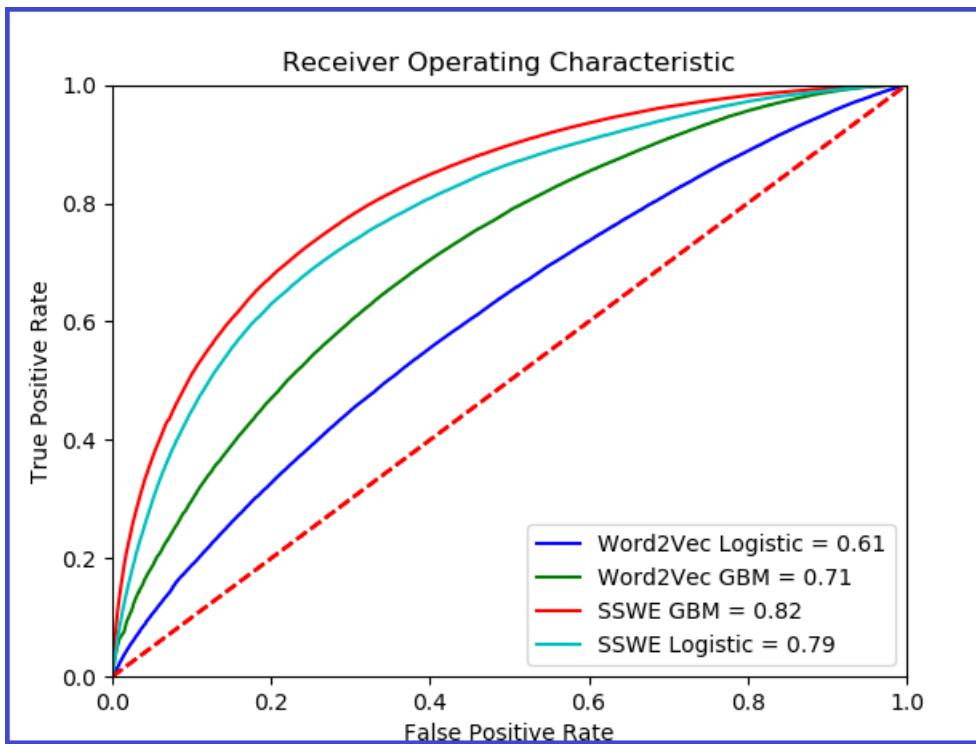
After the word vectors are generated by using the SSWE or Word2Vec algorithm, the classification models are trained to predict the actual sentiment polarity. Two types of features: Word2Vec and SSWE, are applied to two models: the Gradient Boosting model and the Logistic Regression model. As a result, four different models are trained.

Model evaluation

After the models are trained, the models are used to test Twitter text data and evaluate each model's performance. The sample evaluates the following four models:

- Gradient Boosting over SSWE embedding.
- Logistic Regression over SSWE embedding.
- Gradient Boosting over Word2Vec embedding.
- Logistic Regression over Word2Vec embedding.

A comparison of the performance of the four models is shown in the following figure:



The Gradient Boosting model with the SSWE feature gives the best performance when comparing the models by using the area under curve (AUC) metric.

Deployment

The final step is deployment of the trained sentiment prediction model to a web service on a cluster in Azure Container Service. The sample uses the Gradient Boosting model with the SSWE embedding algorithm as the trained model. The operationalization environment provisions Docker and Kubernetes in the cluster to manage the web-service deployment, as shown in the following figure:

For more information on the operationalization process, see [Deploying an Azure Machine Learning model as a web service](#).

Conclusion

In this article, you learned how to train a word-embedding model by using the Word2Vec and Sentiment-Specific Word Embedding algorithms. The extracted embeddings were used as features to train several models to predict sentiment scores for Twitter text data. The SSWE feature used with the Gradient Boosting model gave the best performance. The model was then deployed as a real-time web service in Container Service by using Azure Machine Learning Workbench.

References

- [Team Data Science Process](#)
- [How to use Team Data Science Process \(TDSP\) in Azure Machine Learning](#)
- [TDSP project templates for Azure Machine Learning](#)

- Azure Machine Learning Workbench
- US income data-set from UCI ML repository
- Biomedical entity recognition by using TDSP templates
- Mikolov, Tomas, et al. "Distributed Representations of Words and Phrases and their Compositionality. Advances in neural information processing systems." 2013.
- Tang, Duyu, et al. "Learning Sentiment-Specific Word Embedding for Twitter Sentiment Classification." ACL (1). 2014.

Income classification with Team Data Science Process (TDSP) project

12/18/2017 • 7 min to read • [Edit Online](#)

Introduction

Standardization of the structure and documentation of data science projects, that is anchored to an established [data science lifecycle](#), is key to facilitating effective collaboration in data science teams. Creating Azure Machine Learning projects with the [Team Data Science Process \(TDSP\)](#) template provides a framework for such standardization.

We had previously released a [GitHub repository for the TDSP project structure and templates](#). But it was not possible, until now to instantiate the TDSP structure and templates within a data science tool. We have now enabled creation of Azure Machine Learning projects that are instantiated with [TDSP structure and documentation templates for Azure Machine Learning](#). Instructions on how to use TDSP structure and templates in Azure Machine Learning is provided [here](#). Here we provide an example of how an actual machine learning project can be created using TDSP structure, populated with project-specific code, artifacts and documents, and executed within the Azure Machine Learning.

Link to GitHub repository

We provide summary documentation [here](#) about the sample. More extensive documentation can be found on the GitHub site.

Purpose

The primary purpose of this sample is to show how to instantiate and execute a machine learning project using the [Team Data Science Process \(TDSP\)](#) structure and templates in Azure Machine Learning. For this purpose, we use the well-known [1994 US Census data from the UCI Machine Learning Repository](#). The modeling task is to predict US annual income classes from US Census information (for example, age, race, education level, country of origin, etc.)

Scope

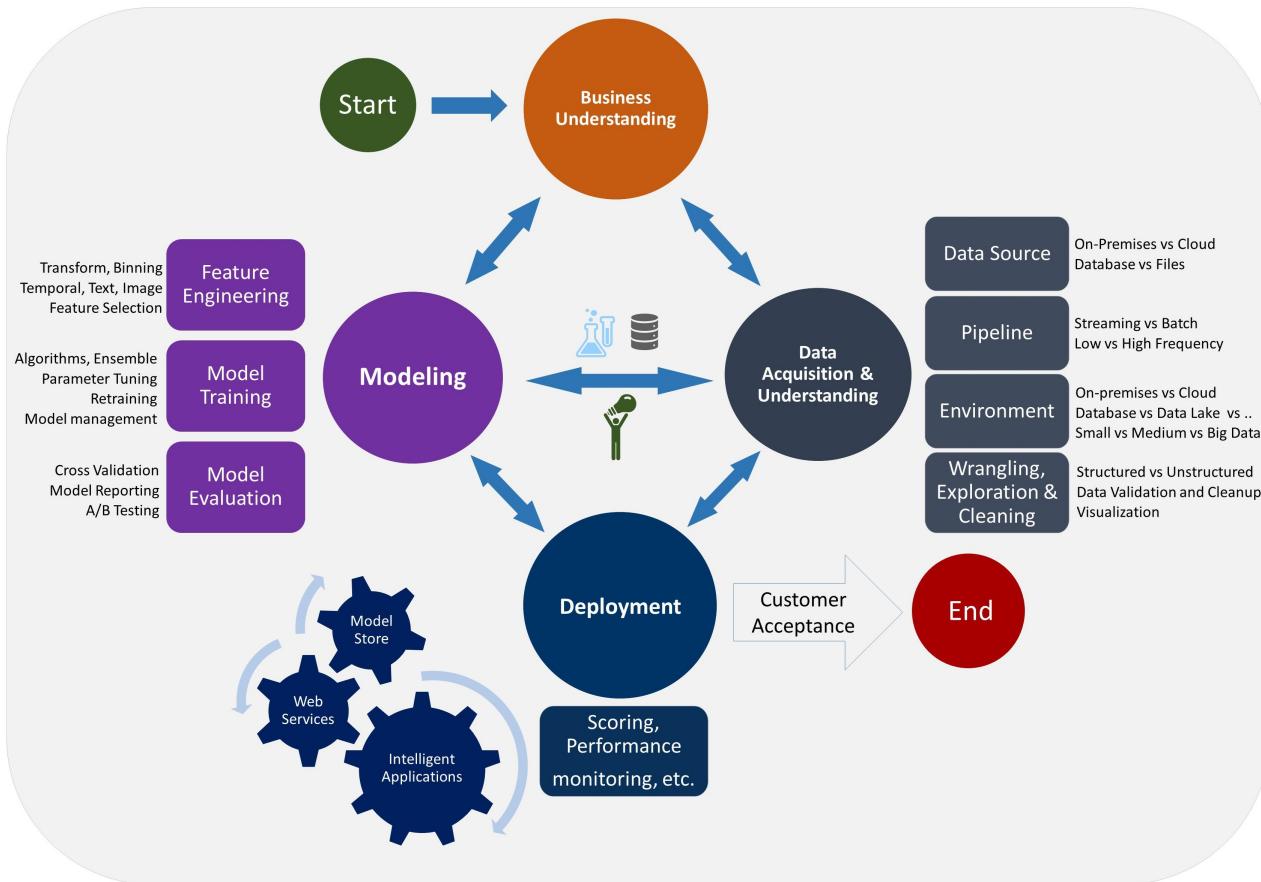
- Data exploration, training, and deployment of a machine learning model which address the prediction problem described in the Use Case Overview.
- Execution of the project in Azure Machine Learning using the Team Data Science Process (TDSP) template from Azure Machine Learning for this project. For project execution and reporting, we're going to use the TDSP lifecycle.
- Operationalization of the solution directly from Azure Machine Learning in Azure Container Services.

The project highlights several features of Azure Machine Learning, such TDSP structure instantiation and use, execution of code in Jupyter notebooks as well as Python files, and easy operationalization in Azure Container Services using Docker and Kubernetes.

Team Data Science Process (TDSP) lifecycle

See [Team Data Science Process \(TDSP\) Lifecycle](#)

Data Science Lifecycle



Prerequisites

Required: subscription, hardware, software

1. An Azure [subscription](#). You can get a [free subscription](#) to execute this sample also.
2. An [Azure Data Science Virtual Machine \(DSVM\) Windows Server 2016](#), (VM Size: [DS3_V2](#), with 4 virtual CPUs and 14-Gb RAM). Although tested on an Azure DSVM, it is likely to work on any Windows 10 machine.
3. Review documentation on Azure Machine Learning and its related services (see below for links).
4. Make sure that you have properly installed Azure Machine Learning by the [quick start installation guide](#).

The dataset for this sample is from the UCI ML Repository [\[link\]](#). It is taken from the 1994 US Census database and contains census and income information for about 50,000 individuals. This is structured dataset having numerical and categorical features, and a categorical target consisting of two income categories ('>50 K' or '<=50 K').

Optional: Version control repository

If you would like to save and version your project and its contents, you need to have a version control repository where this can be done. You can enter the Git repository location while creating the new project using the TDSP template in Azure Machine Learning. See [how to use Git in Azure Machine Learning](#) for further details.

Informational: about Azure Machine Learning

- [FAQ - How to get started](#)
- [Overview](#)
- [Installation](#)
- [Execution](#)
- [Using TDSP](#)
- [Read and write files](#)
- [Using Git with Azure Machine Learning](#)
- [Deploying an ML model as a web service](#)

Create a new workbench project

Create a new project using this example as a template:

1. Open Azure Machine Learning Workbench
2. On the **Projects** page, click the + sign and select **New Project**
3. In the **Create New Project** pane, fill in the information for your new project
4. In the **Search Project Templates** search box, type "Classify US incomes - TDSP project" and select the template
5. Click **Create**

If you provide an empty Git repository location during creating the project (in the appropriate box), then that repository will be populated with the project structure and contents after creation of the project.

Use case overview

The problem is to understand how socio-economic data captured in US Census can help predict annual income of individuals in US. Based on such Census features, the machine learning task is to predict if the income of an individual is above \$50,000 or not (binary classification task).

Data description

For detailed information about the data, see the [description](#) in the UCI repository.

This data was extracted from the Census Bureau database found at: <https://www.census.gov/en.html>.

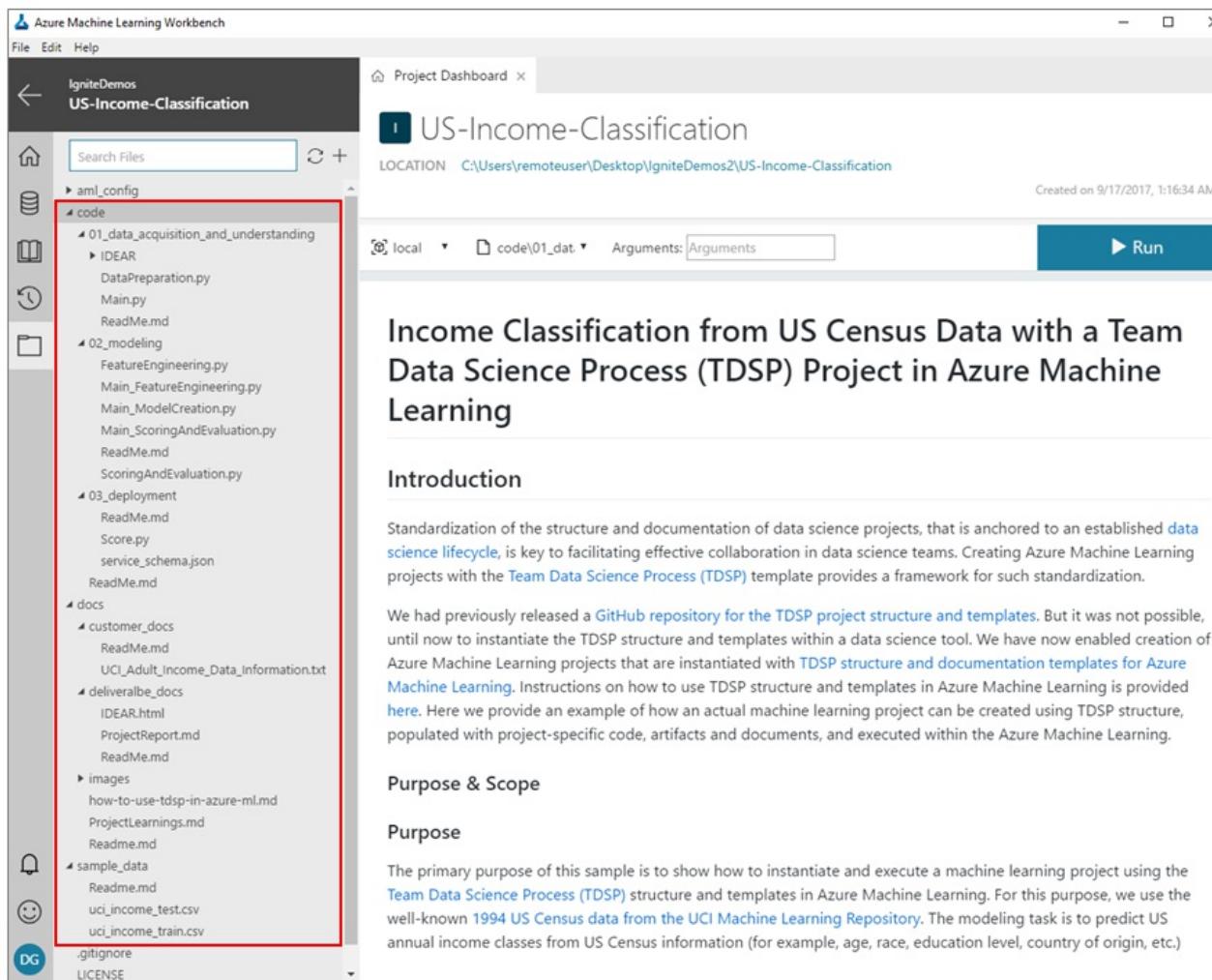
- There are a total of 48,842 instances (prior to any filtering), mix of continuous and discrete (train=32,561, test=16,281)
- Probability for the label '>50 K' : 23.93% / 24.78% (without unknowns)
- Probability for the label '<=50 K': 76.07% / 75.22% (without unknowns)
- **TARGET**: Income class '>50 K', '<=50 K'. These are replaced by 1 and 0 respectively in data preparation phase.
- **FEATURES**: Age, work class, education level, education level, race, sex, hours of work per week, etc.

Project structure, execution, and reporting

Structure

For this project, we use the TDSP folder structure and documentation templates (below), which follows the [TDSP lifecycle](#).

Project is created based on instructions provided [here](#). After it is filled with the project's code and artifacts, the structure looks as follows (see project structure boxed in red in figure below).



Execution

In this example, we execute code in **local compute environment**. Refer to Azure Machine Learning documents for further details on [execution options](#).

Executing a Python script in a local Python runtime is easy:

```
az ml experiment submit -c local my_script.py
```

IPython notebook files can be double-clicked from the project structure on the left of the Azure Machine Learning UI and run in the Jupyter Notebook Server.

The step-by-step data science workflow was as follows:

- **Data acquisition and understanding**

Data was downloaded in .csv form from URLs at UCI ML Repository [\[link\]](#). Features, target, and their transformations are described in detail in the ProjectReport.md file.

Code for data acquisition and understanding is located in: /code/01_data_acquisition_and_understanding.

Data exploration is performed using the Python 3 [IDEAR \(Interactive Data Exploration and Reporting\) utility](#) published as a part of [TDSP suite of data science tools](#). This utility helps to generate standardized data exploration reports for data containing numerical and categorical features and target. Details of how the Python 3 IDEAR utility was used is provided below.

The location of the final data exploration report is [IDEAR.html](#). A view of the IDEAR report is shown below:

Interactive Data Exploration, Analysis, and Reporting

- Author: Team Data Science Process from Microsoft
- Date: 2017/03
- Supported Data Sources: CSV files on the machine where the Jupyter notebook runs or data stored in SQL server
- Output: IDEAR_Report.ipynb

This is the **Interactive Data Exploration, Analysis and Reporting (IDEAR)** in **Python** running on Jupyter Notebook. The data can be stored in CSV file on the machine where the Jupyter notebook runs or from a query running against a SQL server. A yaml file has to be pre-configured before running this tool to provide information about the data.

Step 1: Configure and Set up IDEAR

Before start utilizing the functionalities provided by IDEAR, you need to first [configure and set up](#) the utilities by providing the yaml file and load necessary Python modules and libraries.

Step 2: Start using IDEAR

This tool provides various functionalities to help users explore the data and get insights through interactive visualization and statistical testing.

- [Read and Summarize the data](#)
- [Extract Descriptive Statistics of Data](#)
- [Explore Individual Variables](#)
- [Explore Interactions between Variables](#)
 - [Rank variables](#)
 - [Interaction between two categorical variables](#)
 - [Interaction between two numerical variables](#)
 - [Interaction between numerical and categorical variables](#)
 - [Interaction between two numerical variables and a categorical variable](#)
- [Visualize High Dimensional Data via Projecting to Lower Dimension Principal Component Spaces](#)
- [Generate Data Report](#)

After you are done with exploring the data interactively, you can choose to [show/hide the source code](#) to make your notebook look neater.

Note:

- Change the working directory and yaml file before running IDEAR in Jupyter Notebook.
- Run the cells and click *Export* button to export the code that generates the visualization/analysis result to temporary Jupyter notebooks.
- Run the last cell and click [Generate Final Report](#) to create *IDEAR_Report.ipynb* in the working directory. *If you do not export codes in some sections, you may see some warnings complaining that some temporary Jupyter Notebook files are missing.*
- Upload *IDEAR_Report.ipynb* to Jupyter Notebook server, and run it to generate report.

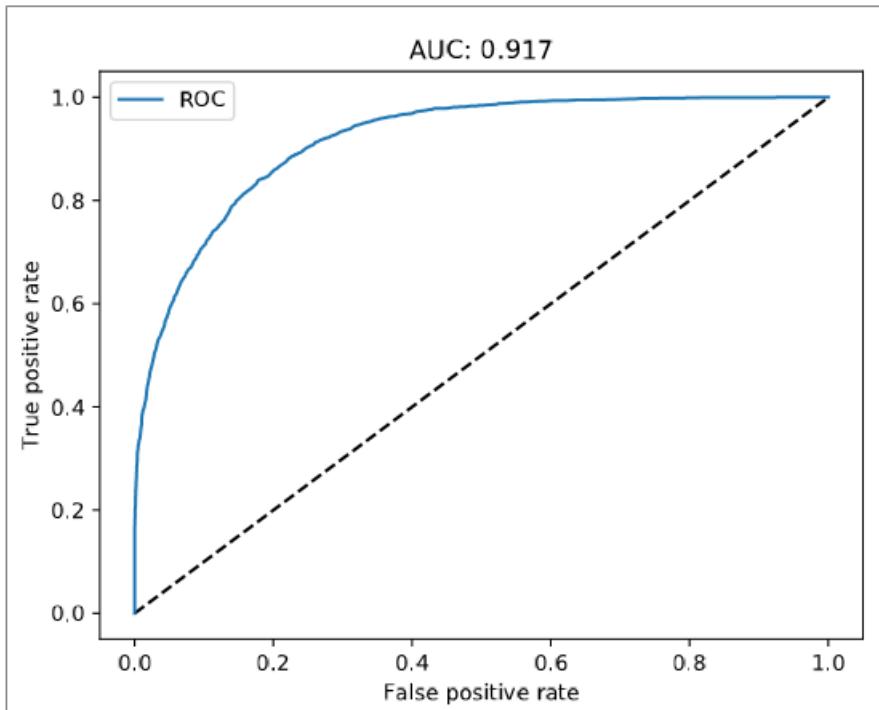
• Modeling

We created two models with 3-fold cross-validation: Elastic Net and Random forest. We used [59-point sampling](#) for random grid search as a strategy for cross-validation and model parameter optimization. Accuracy of the models were measured using AUC (Area under curve) on the test data set.

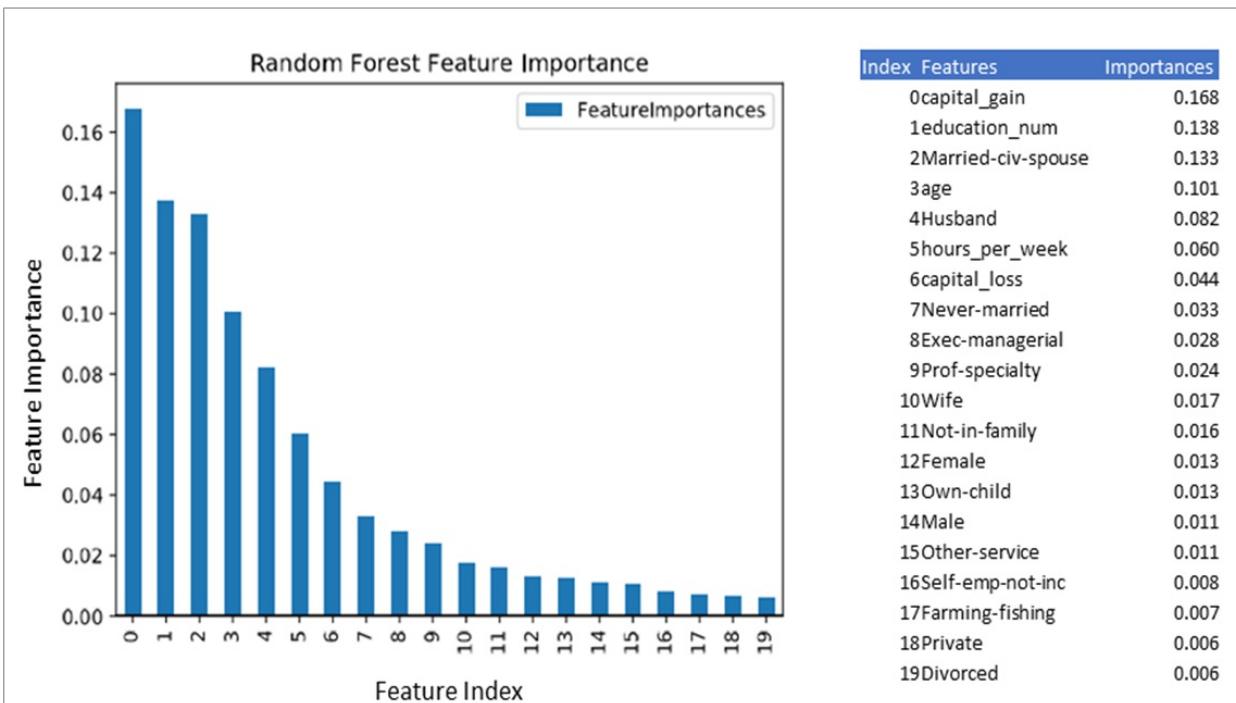
Code for modeling is located in: /code/02_modeling.

AUC of both Elastic Net and Random Forest models were > 0.85. We save both models in pickled.pkl files, and output the ROC plots for both models. AUC of Random Forest model was 0.92 and that of the Elastic Net model was 0.90. In addition, for model interpretation, feature importance for the Random Forest model are output in a .csv file and plotted in a pdf (top 20 predictive features only).

ROC curve of **Random Forest model** on test data is shown below. This was the model that was deployed:



Feature importance (top 20) of Random Forest model is shown below. It shows features capital gain amount, education, marital status, have highest feature importance.



• Deployment

We deployed the Random Forest model as a web-service on a cluster in the [Azure Container Service \(ACS\)](#). The operationalization environment provisions Docker and Kubernetes in the cluster to manage the web-service deployment. You can find further information on the operationalization process [here](#).

Code for deployment is located in: /code/03_deployment.

Final project report

Details about each of the above sections are provided in the compiled final project report [ProjectReport](#). The project report also contains further details about the use case, model performance metrics, deployment, and infrastructure on which the project was developed and deployed.

The project report, together with the contents of the entire project folder, and version control repository may be delivered to the client.

Conclusion

In this sample, we showed how to use TDSP structure and templates in Azure Machine Learning. Through the document and artifact templates you can:

1. Properly define purpose and scope of a project
2. Create a project team with distributed roles and responsibilities
3. Structure and execute projects according to the TDSP lifecycle stages
4. Develop standardized reports using TDSP data science utilities (such as the IDEAR data exploration and visualization report).
5. Prepare a final data science project report that can be delivered to a client

We hope you use this feature of Azure Machine Learning to facilitate with standardization and collaboration within your data science teams.

Next Steps

See references below to get started:

[How to use Team Data Science Process \(TDSP\) in Azure Machine Learning](#)

[Team Data Science Process \(TDSP\)](#)

[TDSP project template for Azure Machine Learning](#)

[US Income data-set from UCI ML repository](#)

Biomedical entity recognition using Team Data Science Process (TDSP) Template

3/12/2018 • 12 min to read • [Edit Online](#)

Entity extraction is a subtask of information extraction (also known as [Named-entity recognition \(NER\)](#), entity chunking, and entity identification). The aim of this real-world scenario is to highlight how to use Azure Machine Learning Workbench to solve a complicated Natural Language Processing (NLP) task such as entity extraction from unstructured text:

1. How to train a neural word embeddings model on a text corpus of about 18 million PubMed abstracts using [Spark Word2Vec implementation](#).
2. How to build a deep Long Short-Term Memory (LSTM) recurrent neural network model for entity extraction on a GPU-enabled Azure Data Science Virtual Machine (GPU DS VM) on Azure.
3. Demonstrate that domain-specific word embeddings model can outperform generic word embeddings models in the entity recognition task.
4. Demonstrate how to train and operationalize deep learning models using Azure Machine Learning Workbench.
5. Demonstrate the following capabilities within Azure Machine Learning Workbench:
 - Instantiation of [Team Data Science Process \(TDSP\) structure and templates](#)
 - Automated management of your project dependencies including the download and the installation
 - Execution of Python scripts on different compute environments
 - Run history tracking for Python scripts
 - Execution of jobs on remote Spark compute contexts using HDInsight Spark 2.1 clusters
 - Execution of jobs in remote GPU VMs on Azure
 - Easy operationalization of deep learning models as web services on Azure Container Services (ACS)

Use case overview

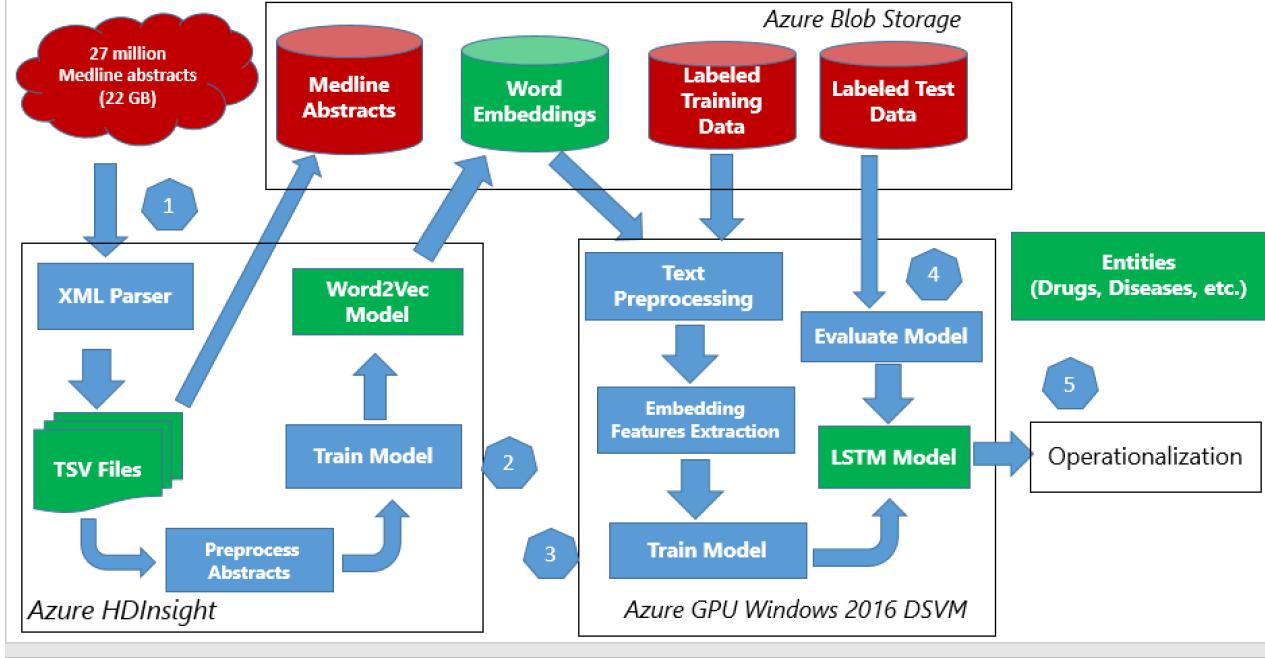
Biomedical named entity recognition is a critical step for complex biomedical NLP tasks such as:

- Extracting the mentions of named entities such diseases, drugs, chemicals, and symptoms from electronic medical or health records.
- Drug discovery
- Understanding the interactions between different entity types such as drug-drug interaction, drug-disease relationship, and gene-protein relationship.

Our use case scenario focuses on how a large amount of unstructured data corpus such as Medline PubMed abstracts can be analyzed to train a word embedding model. Then the output embeddings are considered as automatically generated features to train a neural entity extractor.

Our results show that the biomedical entity extraction model training on the domain-specific word embedding features outperforms the model trained on the generic feature type. The domain-specific model can detect 7012 entities correctly (out of 9475) with F1-score of 0.73 compared to 5274 entities with F1-score of 0.61 for the generic model.

The following figure shows the architecture that was used to process data and train models.



Data description

1. Word2Vec model training data

We first downloaded the raw MEDLINE abstract data from [MEDLINE](#). The data is publically available in the form of XML files on their [FTP server](#). There are 892 XML files available on the server and each of the XML files has the information of 30,000 articles. More details about the data collection step are provided in the Project Structure section. The fields present in each file are

```

abstract
affiliation
authors
country
delete: boolean if False means paper got updated so you might have two XMLs for the same paper.
file_name
issn_linking
journal
keywords
medline_ta: this is abbreviation of the journal name
mesh_terms: list of MeSH terms
nlm_unique_id
other_id: Other IDs
pmc: Pubmed Central ID
pmid: Pubmed ID
pubdate: Publication date
title

```

2. LSTM model training data

The neural entity extraction model has been trained and evaluated on publicly available datasets. To obtain a detailed description about these datasets, you could refer to the following sources:

- [Bio-Entity Recognition Task at BioNLP/NLPBA 2004](#)
- [BioCreative V CDR task corpus](#)
- [Semeval 2013 - Task 9.1 \(Drug Recognition\)](#)

Link to the Azure gallery GitHub repository

Following is the link to the public GitHub repository of the real-world scenario that contains the code and more

detailed description:

<https://github.com/Azure/MachineLearningSamples-BiomedicalEntityExtraction>

Prerequisites

- An Azure [subscription](#)
- Azure Machine Learning Workbench. See [installation guide](#). Currently the Azure Machine Learning Workbench can be installed on the following operating systems only:
 - Windows 10 or Windows Server 2016
 - macOS Sierra (or newer)

Azure services

- [HDInsight Spark cluster](#) version Spark 2.1 on Linux (HDI 3.6) for scale-out computation. To process the full amount of MEDLINE abstracts discussed below, you need the minimum configuration of:
 - Head node: [D13_V2](#) size
 - Worker nodes: At least 4 of [D12_V2](#). In our work, we used 11 worker nodes of D12_V2 size.
- [NC6 Data Science Virtual Machine \(DSVM\)](#) for scale-up computation.

Python packages

All the required dependencies are defined in the aml_config/conda_dependencies.yml file under the scenario project folder. The dependencies defined in this file are automatically provisioned for runs against docker, VM, and HDI cluster targets. For details about the Conda environment file format, refer to [here](#).

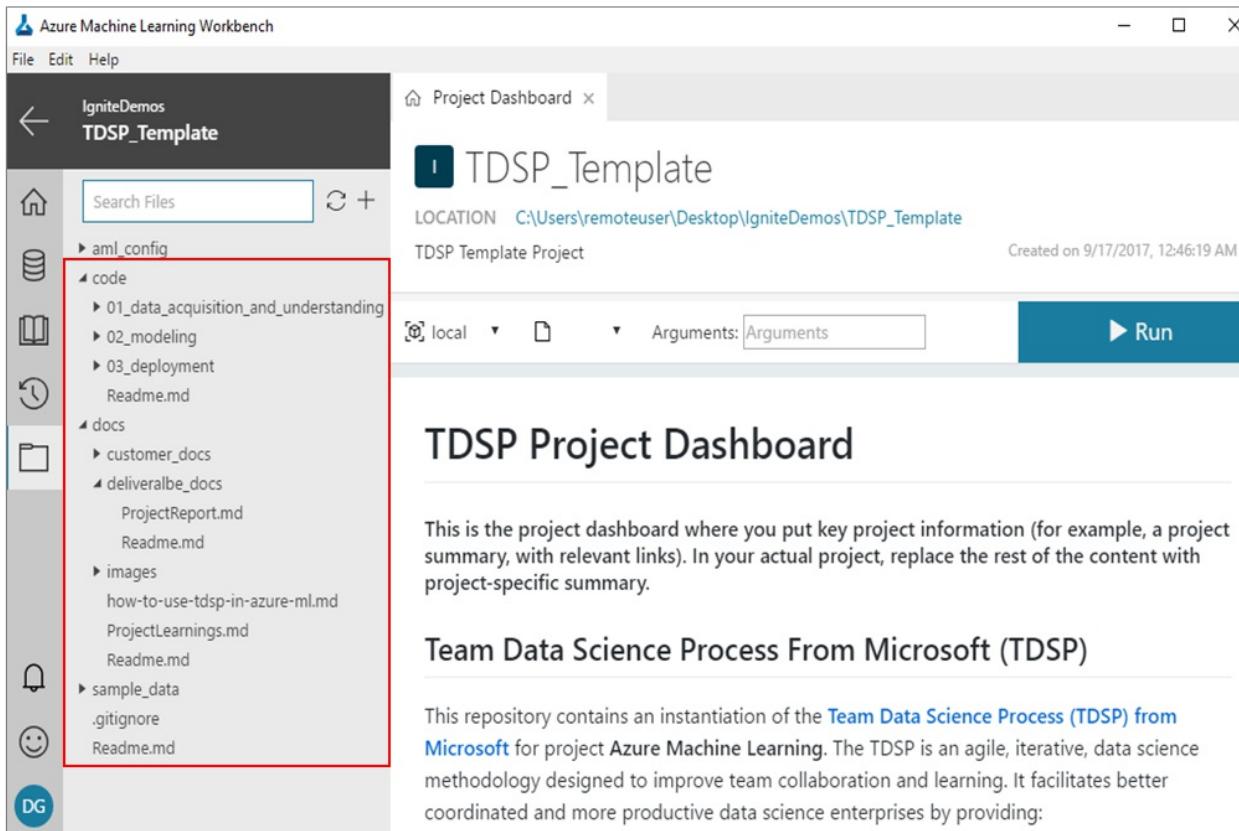
- [TensorFlow](#)
- [CNTK 2.0](#)
- [Keras](#)
- [NLTK](#)
- [Fastparquet](#)

Basic instructions for Azure Machine Learning (AML) workbench

- [Overview](#)
- [Installation](#)
- [Using TDSP](#)
- [How to read and write files](#)
- [How to use Jupyter Notebooks](#)
- [How to use GPU](#)

Scenario structure

For the scenario, we use the TDSP project structure and documentation templates (Figure 1), which follows the [TDSP lifecycle](#). Project is created based on instructions provided [here](#).



The step-by-step data science workflow is as follows:

1. Data acquisition and understanding

See [Data Acquisition and Understanding](#).

The raw MEDLINE corpus has a total of 27 million abstracts where about 10 million articles have an empty abstract field. Azure HDInsight Spark is used to process big data that cannot be loaded into the memory of a single machine as a [Pandas DataFrame](#). First, the data is downloaded into the Spark cluster. Then the following steps are executed on the [Spark DataFrame](#):

- parse the XML files using Medline XML Parser
- preprocess the abstract text including sentence splitting, tokenization, and case normalization.
- exclude articles where abstract field is empty or has short text
- create the word vocabulary from the training abstracts
- train the word embedding neural model. For more information, see [GitHub code link](#) to get started.

After parsing XML files, data has the following format:

abstract	Association of dilution hyponatremia and normal natriuresis produce an inappropriate secretion of antidiuretic hormone. In this pulmonary tuberculosis case we report manifestations along with the other biological criteria of inappropriate secretion of antidiuretic hormone. All disturbances cease when pulmonary tuberculosis is cured. This syndrome may be due either to disturbance in the regulatory apparatus of antidiuretic hormone secretion or to the antidiuretic principle in tubercular lung tissue.
affiliation	
author	P Peltier; R Grossette; G Dabouis; J Coroller
country	FRANCE
delete	FALSE
file_name	medline16n0189.xml.gz
issn_linking	
journal	La semaine des hôpitaux : organe fondé par l'Association d'enseignement médical des hôpitaux de Paris
keywords	
medline_ta	Sem Hop
mesh_terms	Aged; Humans; Hyponatremia; Inappropriate ADH Syndrome; Male; Natriuresis; Tuberculosis, Pulmonary
nlm_unique_id	9410059
other_id	
pmc	
pmid	6244674
pubdate	1980
title	[Dilution hyponatremia and conserved natriuresis in pulmonary tuberculosis (author's transl)].

The neural entity extraction model has been trained and evaluated on publicly available datasets. To obtain a detailed description about these datasets, you could refer to the following sources:

- [Bio-Entity Recognition Task at BioNLP/NLPBA 2004](#)
- [BioCreative V CDR task corpus](#)
- [Semeval 2013 - Task 9.1 \(Drug Recognition\)](#)

2. Modeling

See [Modeling](#).

Modeling is the stage where we show how you can use the data downloaded in the previous section for training your own word embedding model and use it for other downstream tasks. Although we are using the PubMed data, the pipeline to generate the embeddings is generic and can be reused to train word embeddings for any other domain. For embeddings to be an accurate representation of the data, it is essential that the word2vec is trained on a large amount of data. Once we have the word embeddings ready, we can train a deep neural network model that uses the learned embeddings to initialize the Embedding layer. We mark the embedding layer as non-trainable but that is not mandatory. The training of the word embedding model is unsupervised and hence we are able to take advantage of unlabeled texts. However, the training of the entity recognition model is a supervised learning task and its accuracy depends on the amount and the quality of a manually-annotated data.

2.1. Feature generation

See [Feature generation](#).

Word2Vec is the word embedding unsupervised learning algorithm that trains a neural network model from an unlabeled training corpus. It produces a continuous vector for each word in the corpus that represents its semantic information. These models are simple neural networks with a single hidden layer. The word vectors/embeddings are learned by backpropagation and stochastic gradient descent. There are two types of word2vec models, namely, the Skip-Gram and the continuous-bag-of-words (check [here](#) for more details). Since we are using the MLLib's implementation of the word2vec, which supports the Skip-gram model, we briefly describe it here (image taken from [link](#)):

Skip-gram Model

Input: A single word W_i

Output: Words in W_i 's context $\{W_{01}, \dots, W_{0C}\}$ where C is the window size

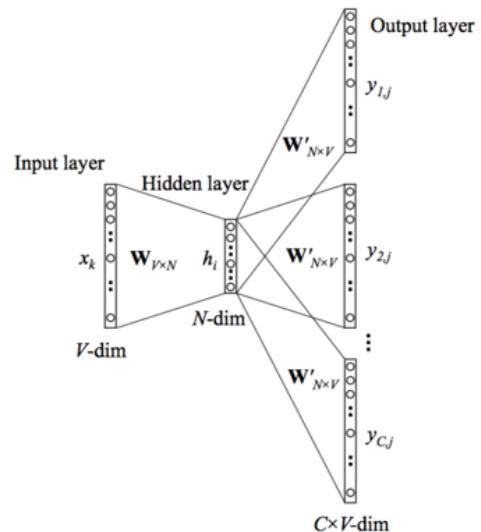
Example: I love deep **learning** for NLP

Input: learning

Output: "love", "deep", "for", "NLP" for window size of 2

$$\text{Objective Function: } p(w_{c,j} = w_{O,c} | w_I) = y_{c,j} = \frac{\exp(u_{c,j})}{\sum_{j'=1}^V \exp(u_{j'})}$$

$$\begin{aligned} \text{Gradient Descent Updates: } w_{ij}^{(new)} &= w_{ij}^{(old)} - \eta \cdot \sum_{c=1}^C (y_{c,j} - t_{c,j}) \cdot h_i \\ w_{ij}^{(new)} &= w_{ij}^{(old)} - \eta \cdot \sum_{j=1}^V \sum_{c=1}^C (y_{c,j} - t_{c,j}) \cdot w_{ij}' \cdot x_j \end{aligned}$$



Optimization using Hierarchical SoftMax and Negative Sampling

The model uses Hierarchical Softmax and Negative sampling to optimize the performance. Hierarchical SoftMax (H-SoftMax) is an approximation inspired by binary trees. H-SoftMax essentially replaces the flat SoftMax layer with a hierarchical layer that has the words as leaves. This allows us to decompose calculating the probability of one word into a sequence of probability calculations, which saves us from having to calculate the expensive normalization over all words. Since a balanced binary tree has a depth of $\log_2(V)$ (V is the Vocabulary), we only need to evaluate at most $\log_2(V)$ nodes to obtain the final probability of a word. The probability of a word w given its context c is then simply the product of the probabilities of taking right and left turns respectively that lead to its leaf node. We can build a Huffman Tree based on the frequency of the words in the dataset to ensure that more frequent words get shorter representations. For more information, see [this link](#). Image taken from [here](#).

Visualization

Once we have the word embeddings, we would like to visualize them and see the relationship between semantically similar words.

```
model.findSynonyms("cancer", 20).select("word").head(20) #Returns types of Cancers, hormones responsible for Cancer etc.
```

```
[Row(word=u'crc'), Row(word=u'colorectal'), Row(word=u'eoc'), Row(word=u'breast'), Row(word=u'hormone-refractory'), Row(word=u'nsclc'), Row(word=u'crpc'), Row(word=u'prostate'), Row(word=u'cancers'), Row(word=u'early-stage'), Row(word=u'carcinoma'), Row(word=u'adenocarcinoma'), Row(word=u'her2-positive'), Row(word=u'head-and-neck'), Row(word=u'hcc'), Row(word=u'metastatic'), Row(word=u'hnscc'), Row(word=u'gbc'), Row(word=u'mcrc'), Row(word=u'tnbc)]
```

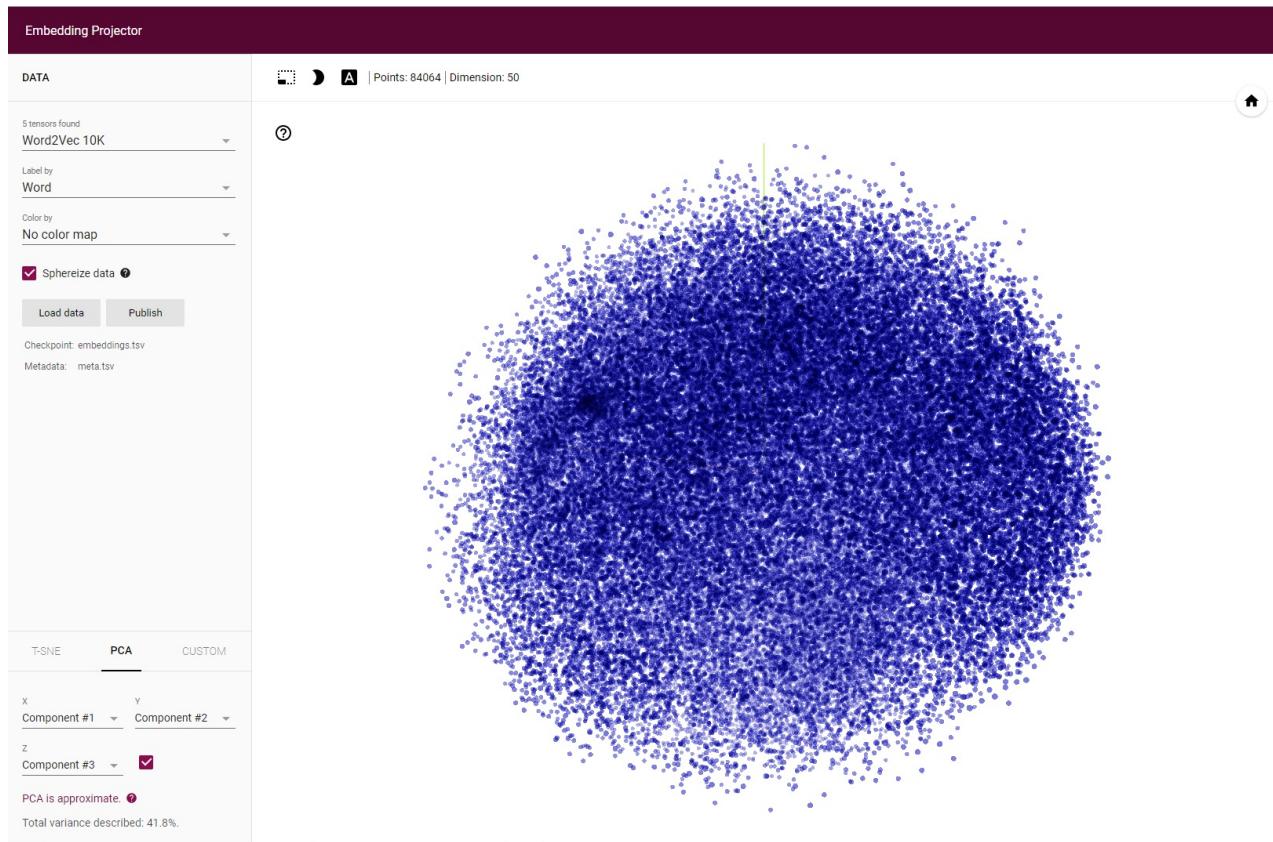
```
model.findSynonyms("brain", 20).select("word").head(20) # Returns Different Parts of the Brain
```

```
[Row(word=u'cerebrum'), Row(word=u'cerebellum'), Row(word=u'forebrain'), Row(word=u'neocortex'), Row(word=u'cerebrocortical'), Row(word=u'hippocampi'), Row(word=u'brains'), Row(word=u'white-matter'), Row(word=u'hippocampus'), Row(word=u'striatum'), Row(word=u'demyelinated'), Row(word=u'diencephalon'), Row(word=u'striatal'), Row(word=u'gerbil'), Row(word=u'infarcted'), Row(word=u'hypoxic-ischemic'), Row(word=u'retina'), Row(word=u'neurohypophysis'), Row(word=u'telencephalon'), Row(word=u'neocortical')]
```

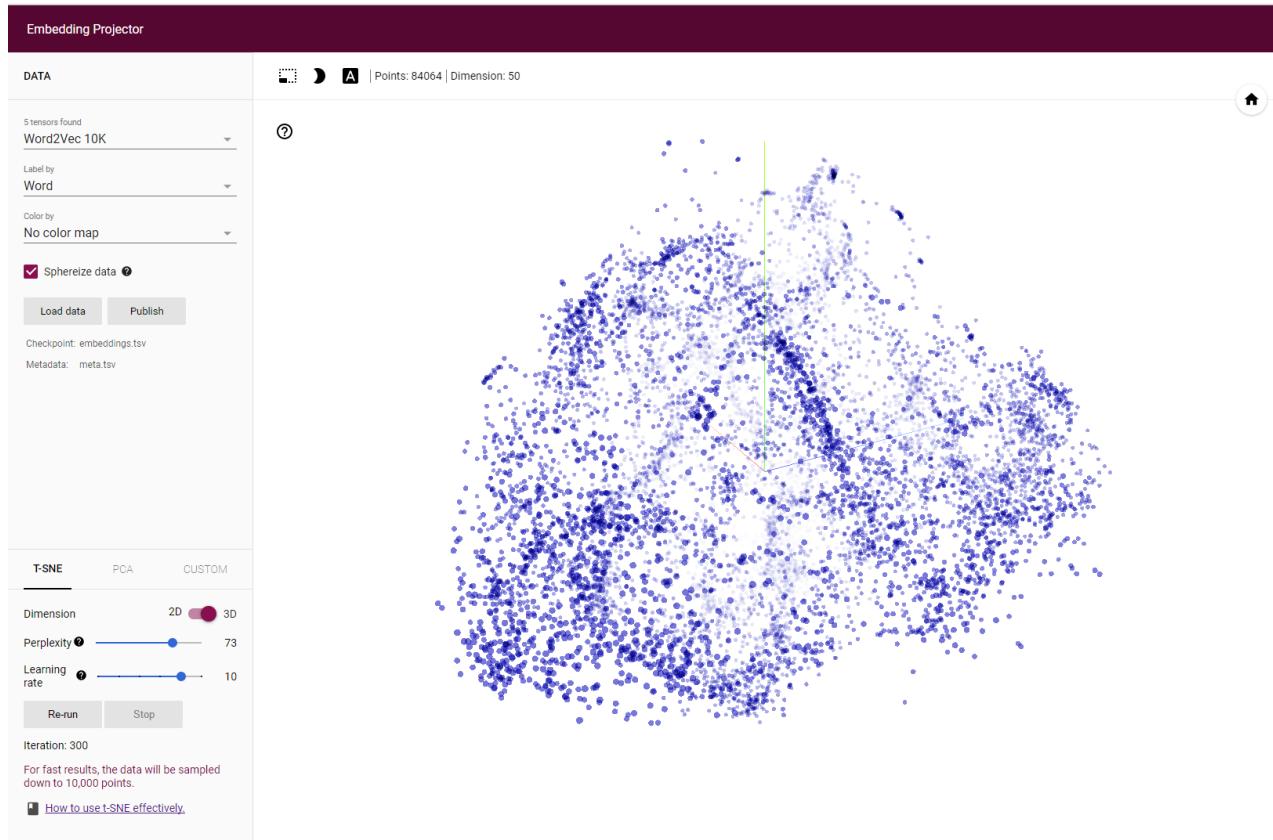
We have shown two different ways of visualizing the embeddings. The first one uses a PCA to project the high dimensional vector to a 2-D vector space. This leads to a significant loss of information and the visualization is not as accurate. The second is to use PCA with **t-SNE**. t-SNE is a nonlinear dimensionality reduction technique that is well-suited for embedding high-dimensional data into a space of two or three dimensions, which can then be visualized in a scatter plot. It models each high-dimensional object by a two- or three-dimensional point in such a way that similar objects are modeled by nearby points and dissimilar objects are modeled by distant points. It works in two parts. First, it creates a probability distribution over the pairs in the higher dimensional space in a way that similar objects have a high probability of being picked and dissimilar points have low probability of getting picked. Second, it defines a similar probability distribution over the points in a low dimensional map and minimizes the KL Divergence between the two distributions with respect to location of points on the map. The location of the points in the low dimension is obtained by minimizing the KL Divergence using Gradient Descent. But t-SNE might not be always reliable. Implementation details can be found [here](#).

As shown in the following figure, the t-SNE visualization provides more separation of word vectors and potential clustering patterns.

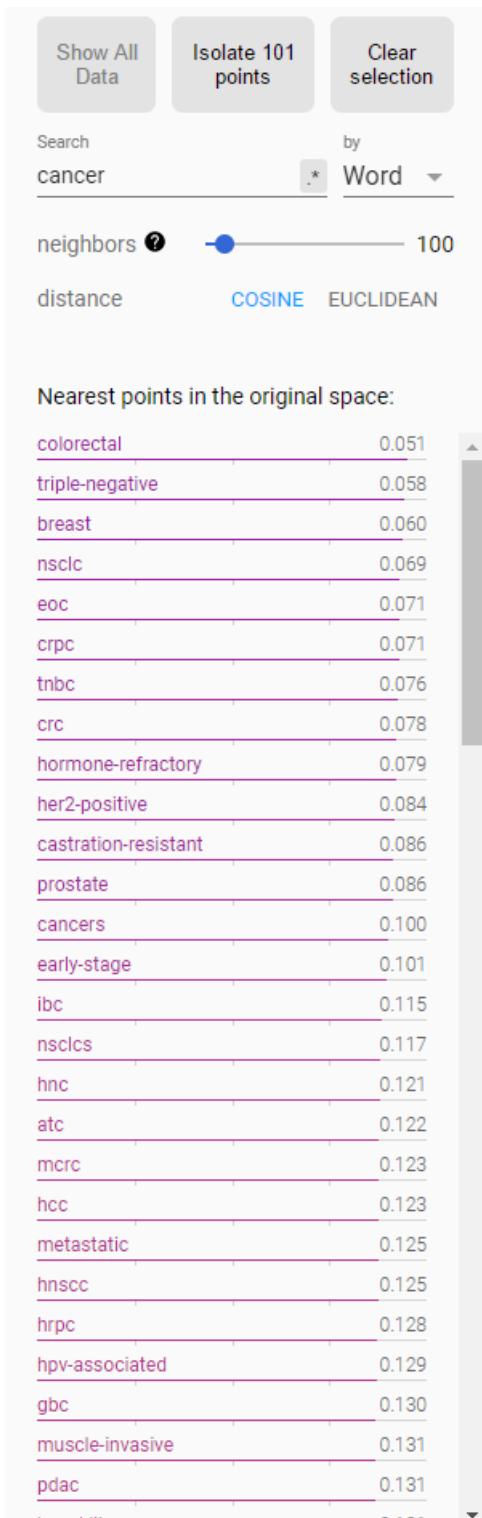
- Visualization with PCA



- Visualization with t-SNE



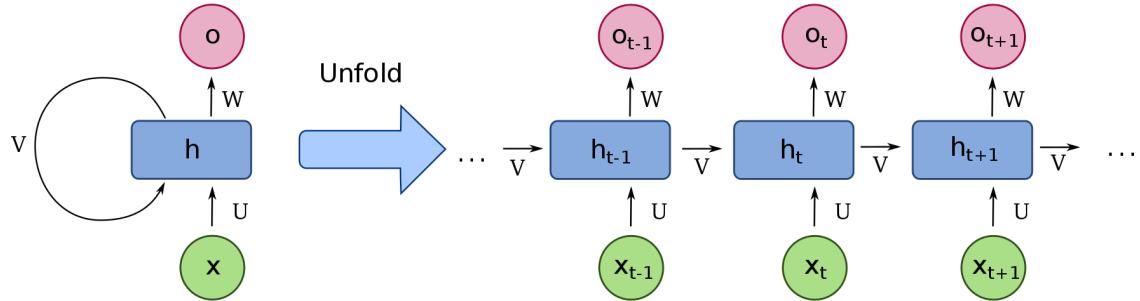
- Points closest to "Cancer" (they are all subtypes of Cancer)



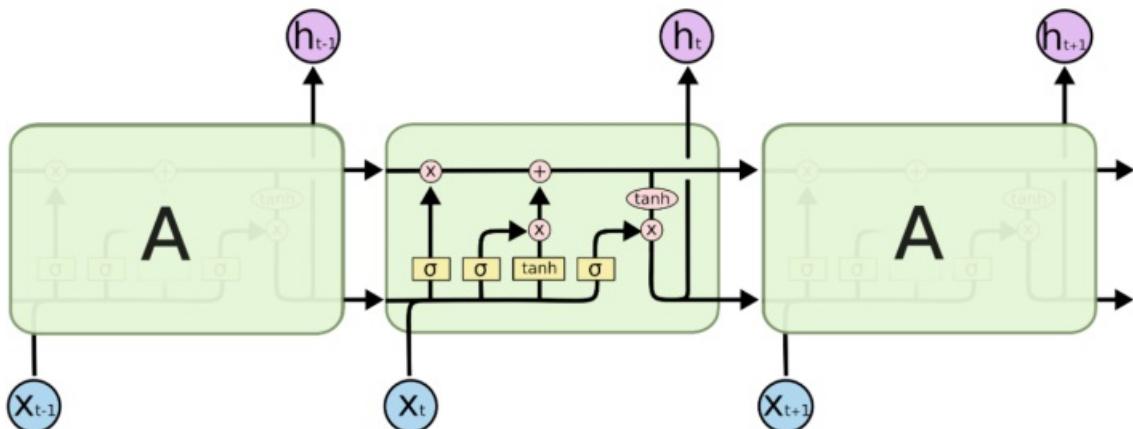
2.2. Train the neural entity extractor

See [Train the neural entity extractor](#).

The feed-forward neural network architecture suffers from a problem that they treat each input and output as independent of the other inputs and outputs. This architecture can't model sequence-to-sequence labeling tasks such as machine translation and entity extraction. Recurrent neural network models overcome this problem as they can pass information computed until now to the next node. This property is called having memory in the network since it is able to use the previously computed information as shown in the following figure:



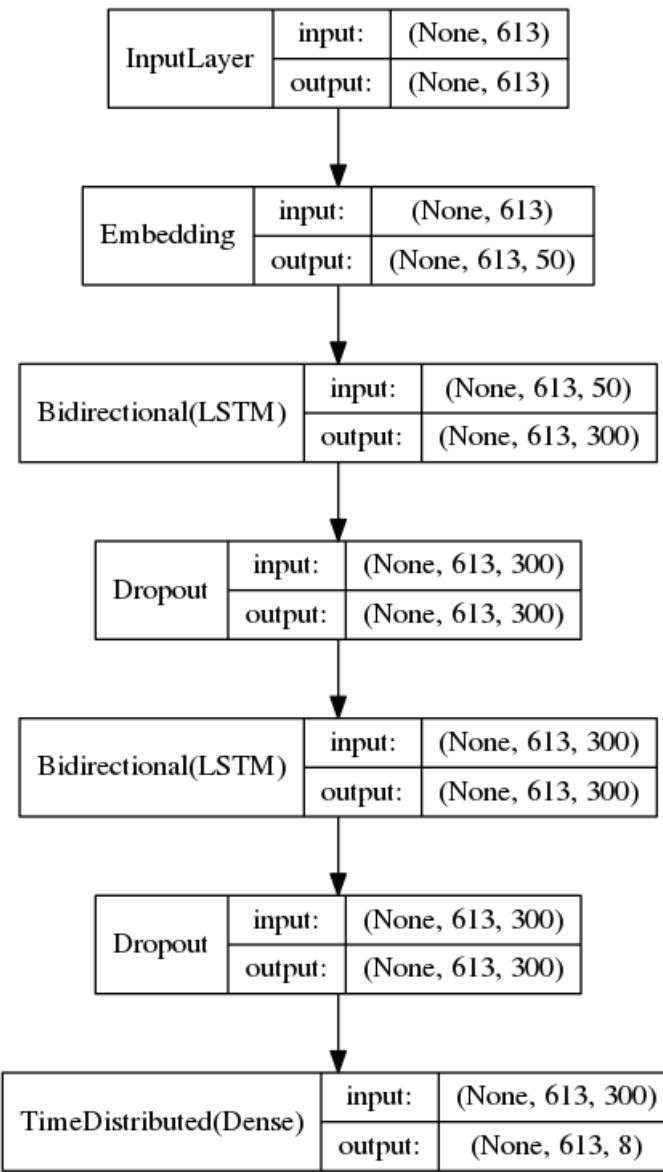
Vanilla RNNs actually suffer from the [Vanishing Gradient Problem](#) due to which they are not able to utilize all the information they have seen before. The problem becomes evident only when a large amount of context is required to make a prediction. But models like LSTM do not suffer from such a problem, in fact they are designed to remember long-term dependencies. Unlike vanilla RNNs that have a single neural network, the LSTMs have the interactions between four neural networks for each cell. For a detailed explanation of how LSTM work, refer to [this post](#).



The repeating module in an LSTM contains four interacting layers.

Let's try to put together our own LSTM-based recurrent neural network and try to extract entity types like drug, disease and symptom mentions from PubMed data. The first step is to obtain a large amount of labeled data and as you would have guessed, that's not easy! Most of the medical data contains lot of sensitive information about the person and hence are not publicly available. We rely on a combination of two different datasets that are publicly available. The first dataset is from Semeval 2013 - Task 9.1 (Drug Recognition) and the other is from BioCreative V CDR task. We are combining and auto labeling these two datasets so that we can detect both drugs and diseases from medical texts and evaluate our word embeddings. For implementation details, refer to [GitHub code link](#).

The model architecture that we have used across all the codes and for comparison is presented below. The parameter that changes for different datasets is the maximum sequence length (613 here).



2.3. Model evaluation

See [Model evaluation](#).

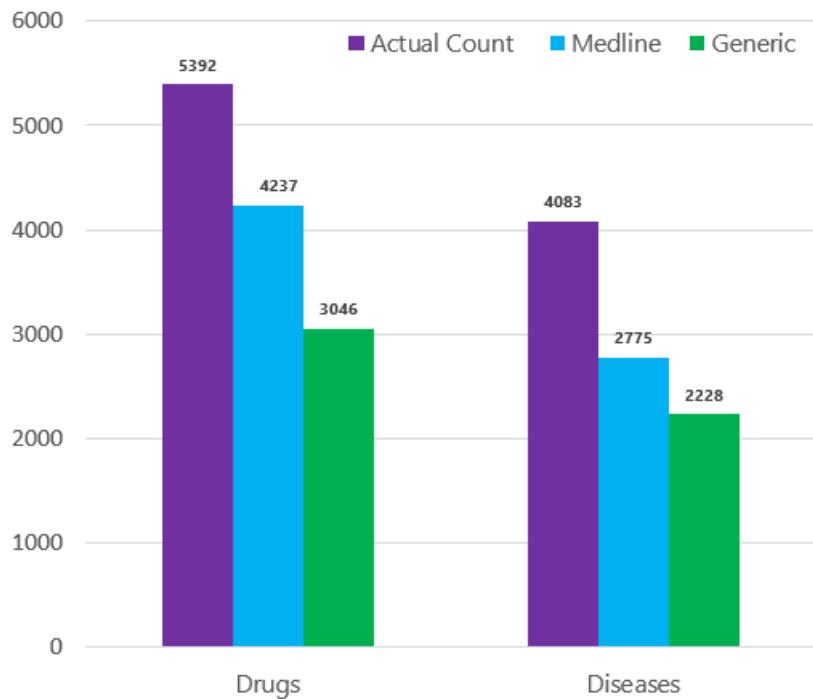
We use the evaluation script from the shared task [Bio-Entity Recognition Task at Bio NLP/NLPBA 2004](#) to evaluate the precision, recall, and F1 score of the model.

In-domain versus generic word embedding models

The following is a comparison between the accuracy of two feature types: (1) word embeddings trained on PubMed abstracts and (2) word embeddings trained on Google News. We clearly see that the in-domain model outperforms the generic model. Hence having a specific word embedding model rather than using a generic one is much more helpful.

- Task #1: Drugs and Diseases Detection

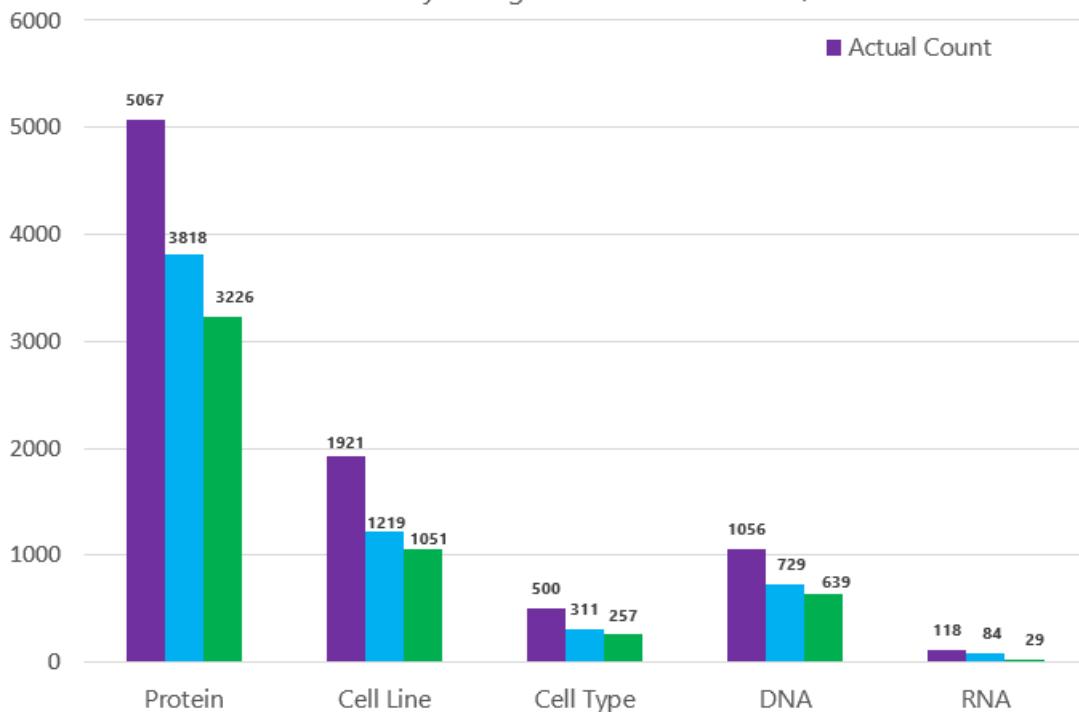
Drug and Disease dataset (auto-labeled)



We perform the evaluation of the word embeddings on other datasets in the similar fashion and see that in-domain model is always better.

- Task #2: Proteins, Cell Line, Cell Type, DNA, and RNA Detection

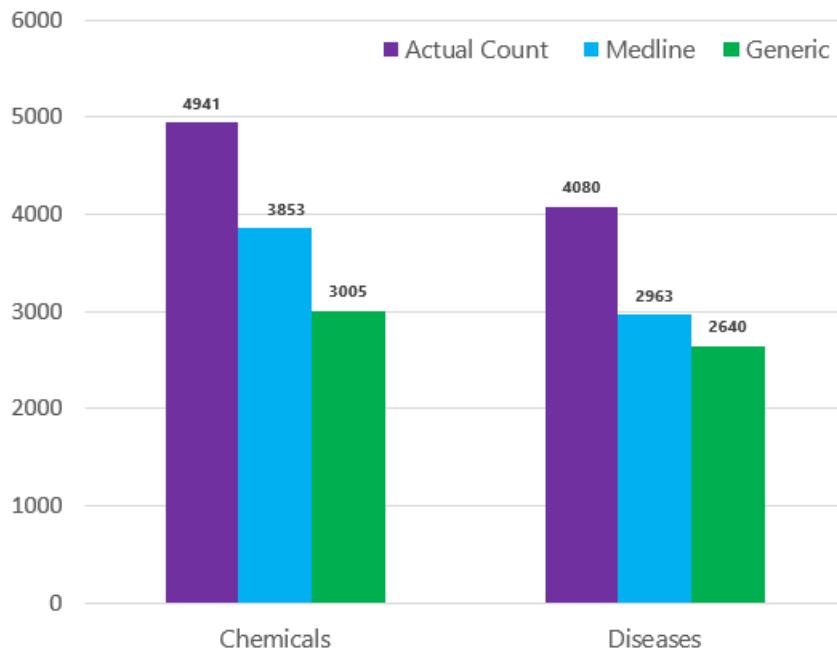
Bio-Entity Recognition Task at Bio NLP/NLPBA 2004



Embedding Dataset	Precision	Recall	F1 Score
MEDLINE	0.65	0.71	0.68
Google News	0.55	0.60	0.57

- Task #3: Chemicals and Diseases Detection

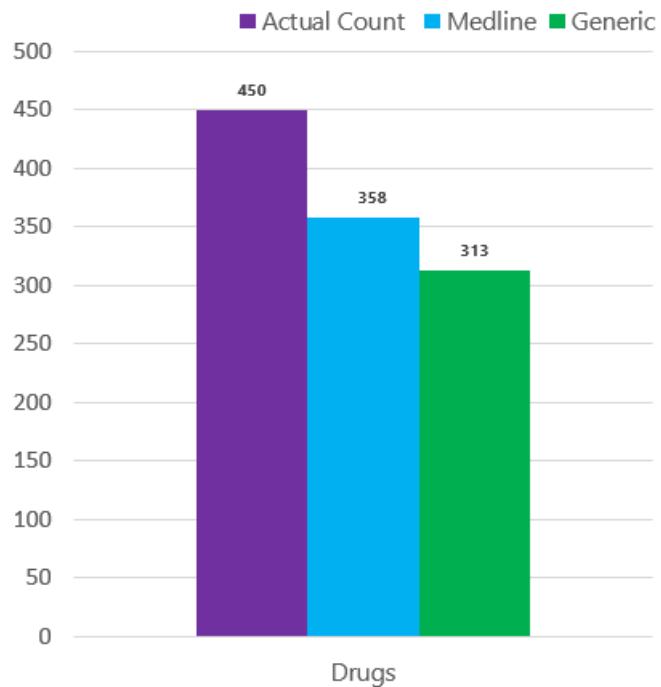
Bio Creative V CDR task corpus



Embedding Dataset	Precision	Recall	F1 Score
MEDLINE	0.77	0.76	0.76
Google News	0.74	0.63	0.67

- Task #4: Drugs Detection

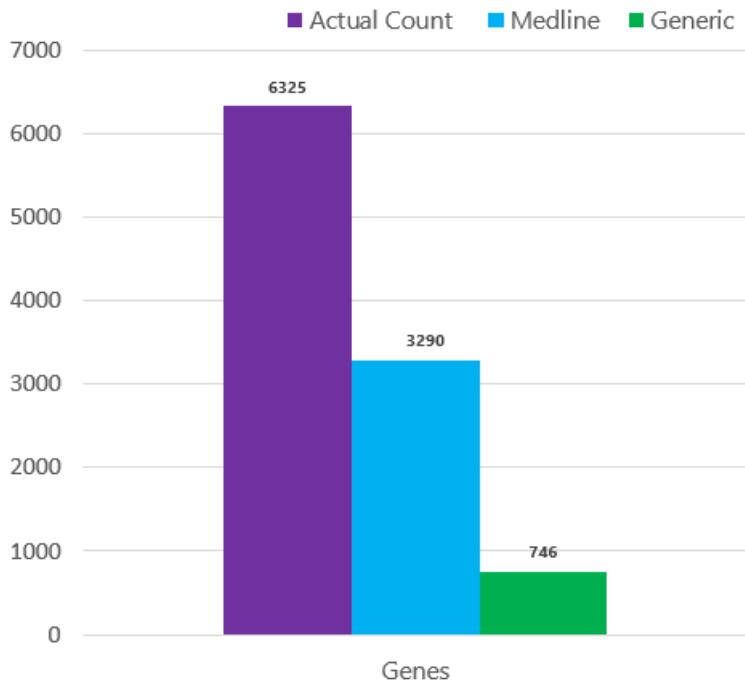
Semeval 2013 (Task 9.1 Drug Recognition)



Embedding Dataset	Precision	Recall	F1 Score
MEDLINE	0.93	0.79	0.85
Google News	0.92	0.69	0.79

- Task #5: Genes Detection

Bio creative 2 (GENE Recognition)

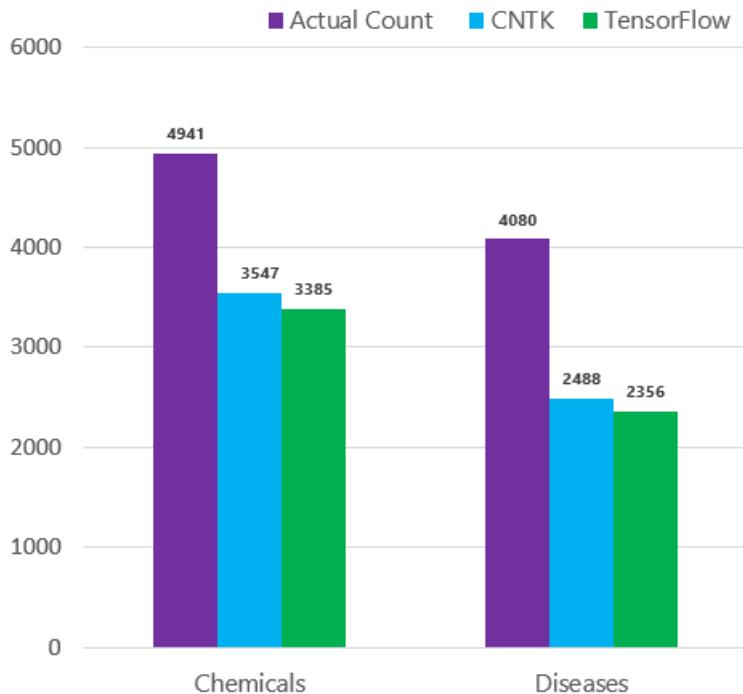


Embedding Dataset	Precision	Recall	F1 Score
MEDLINE	0.57	0.52	0.54
Google News	0.37	0.12	0.18

TensorFlow versus CNTK

All the reported models are trained using Keras with TensorFlow as backend. Keras with CNTK backend does not support "reverse" at the time this work was done. Therefore, for the sake of comparison, we have trained a unidirectional LSTM model with the CNTK backend and compared it to a unidirectional LSTM model with TensorFlow backend. Install CNTK 2.0 for Keras from [here](#).

Bio Creative V CDR task corpus



We concluded that CNTK performs as good as Tensorflow both in terms of the training time taken per epoch (60 secs for CNTK and 75 secs for Tensorflow) and the number of test entities detected. We are using the Unidirectional layers for evaluation.

3. Deployment

See [Deployment](#).

We deployed a web service on a cluster in the [Azure Container Service \(ACS\)](#). The operationalization environment provisions Docker and Kubernetes in the cluster to manage the web-service deployment. You can find further information about the operationalization process [here](#).

Conclusion

We went over the details of how you could train a word embedding model using Word2Vec algorithm on Spark and then use the extracted embeddings as features to train a deep neural network for entity extraction. We have applied the training pipeline on the biomedical domain. However, the pipeline is generic enough to be applied to detect custom entity types of any other domain. You just need enough data and you can easily adapt the workflow presented here for a different domain.

References

- Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. In Proceedings of ICLR.
- Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013b. Distributed representations of words and phrases and their compositionality. In Proceedings of NIPS, pages 3111–3119.
- Billy Chiu, Gamal Crichton, Anna Korhonen, and Sampo Pyysalo. 2016. [How to Train Good Word Embeddings for Biomedical NLP](#), In Proceedings of the fifteenth Workshop on Biomedical Natural Language Processing, pages 166–174.

- Vector Representations of Words
- Recurrent Neural Networks
- Problems encountered with Spark ml Word2Vec
- Spark Word2Vec: lessons learned

HDInsight Spark data science walkthroughs using PySpark and Scala on Azure

11/2/2017 • 2 min to read • [Edit Online](#)

These walkthroughs use PySpark and Scala on an Azure Spark cluster to do predictive analytics. They follow the steps outlined in the Team Data Science Process. For an overview of the Team Data Science Process, see [Data Science Process](#). For an overview of Spark on HDInsight, see [Introduction to Spark on HDInsight](#).

Additional data science walkthroughs that execute the Team Data Science Process are grouped by the **platform** that they use. See [Walkthroughs executing the Team Data Science Process](#) for an itemization of these examples.

Predict taxi tips using PySpark on Azure Spark

The [Use Spark on Azure HDInsight](#) walkthrough uses data from New York taxis to predict whether a tip is paid and the range of amounts expected to be paid. It uses the Team Data Science Process in a scenario using an [Azure HDInsight Spark cluster](#) to store, explore, and feature engineer data from the publicly available NYC taxi trip and fare dataset. This overview topic sets you up with an HDInsight Spark cluster and the Jupyter PySpark notebooks used in the rest of the walkthrough. These notebooks show you 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.

Predict taxi tips using Scala on Azure Spark

The [Use Scala with Spark on Azure](#) walkthrough uses data from New York taxis to predict whether a tip is paid and the range of amounts expected to be paid. It shows how to use Scala for supervised machine learning tasks with the Spark 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.

Next steps

For a discussion of the key components that comprise the Team Data Science Process, see [Team Data Science Process overview](#).

For a discussion of the Team Data Science Process lifecycle that you can use to structure your data science projects, see [Team Data Science Process lifecycle](#). The lifecycle outlines the steps, from start to finish, that projects usually follow when they are executed.

Data exploration and modeling with Spark

2/9/2018 • 31 min to read • [Edit Online](#)

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 a Spark 1.6 (or Spark 2.0) HDInsight 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.

Spark clusters and notebooks

Setup steps and code are provided in this walkthrough for using an HDInsight Spark 1.6. But Jupyter notebooks are provided for both HDInsight Spark 1.6 and Spark 2.0 clusters. A description of the notebooks and links to them are provided in the [Readme.md](#) for the GitHub repository containing them. Moreover, the code here and in the linked 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. For convenience, here are the links to the Jupyter notebooks for Spark 1.6 (to be run in the pySpark kernel of the Jupyter Notebook server) and Spark 2.0 (to be run in the pySpark3 kernel of the Jupyter Notebook server):

Spark 1.6 notebooks

[pySpark-machine-learning-data-science-spark-data-exploration-modeling.ipynb](#): Provides information on how to perform data exploration, modeling, and scoring with several different algorithms.

Spark 2.0 notebooks

The regression and classification tasks that are implemented using a Spark 2.0 cluster are in separate notebooks and the classification notebook uses a different data set:

- [Spark2.0-pySpark3-machine-learning-data-science-spark-advanced-data-exploration-modeling.ipynb](#): This file provides information on how to perform data exploration, modeling, and scoring in Spark 2.0 clusters using the NYC Taxi trip and fare data-set described [here](#). This notebook may be a good starting point for quickly exploring the code we have provided for Spark 2.0. For a more detailed notebook analyzes the NYC Taxi data, see the next notebook in this list. See the notes following this list that compare these notebooks.
- [Spark2.0-pySpark3_NYC_Taxi_Tip_Regression.ipynb](#): This file shows how to perform data wrangling (Spark SQL and dataframe operations), exploration, modeling and scoring using the NYC Taxi trip and fare data-set described [here](#).
- [Spark2.0-pySpark3_Airline_Departure_Delay_Classification.ipynb](#): This file shows how to perform data wrangling (Spark SQL and dataframe operations), exploration, modeling and scoring using the well-known Airline On-time departure dataset from 2011 and 2012. We integrated the airline dataset with the airport weather data (e.g. windspeed, temperature, altitude etc.) prior to modeling, so these weather features can be included in the model.

NOTE

The airline dataset was added to the Spark 2.0 notebooks to better illustrate the use of classification algorithms. See the following links for information about airline on-time departure dataset and weather dataset:

- Airline on-time departure data: <http://www.transtats.bts.gov/ONTIME/>
- Airport weather data: <https://www.ncdc.noaa.gov/>

NOTE

The Spark 2.0 notebooks on the NYC taxi and airline flight delay data-sets can take 10 mins or more to run (depending on the size of your HDI cluster). The first notebook in the above list shows many aspects of the data exploration, visualization and ML model training in a notebook that takes less time to run with down-sampled NYC data set, in which the taxi and fare files have been pre-joined: [Spark2.0-pySpark3-machine-learning-data-science-spark-advanced-data-exploration-modeling.ipynb](#) This notebook takes a much shorter time to finish (2-3 mins) and may be a good starting point for quickly exploring the code we have provided for Spark 2.0.

WARNING

Billing for HDInsight clusters is prorated per minute, whether you are using them or not. Be sure to delete your cluster after you have finished using it. For more information, see [How to delete an HDInsight cluster](#).

NOTE

The descriptions below are related to using Spark 1.6. For Spark 2.0 versions, please use the notebooks described and linked above.

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.Tra
in.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

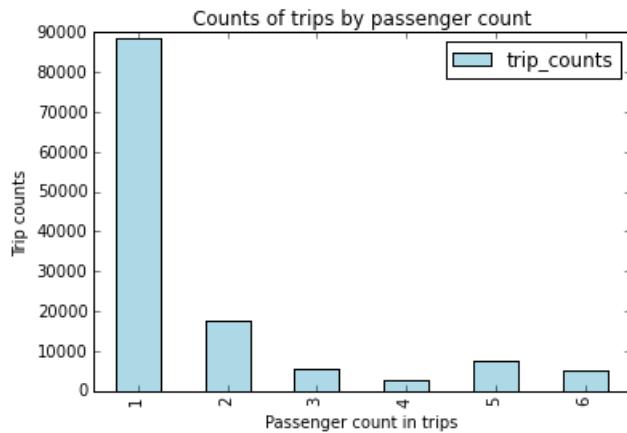
```

# 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_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.

```

#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

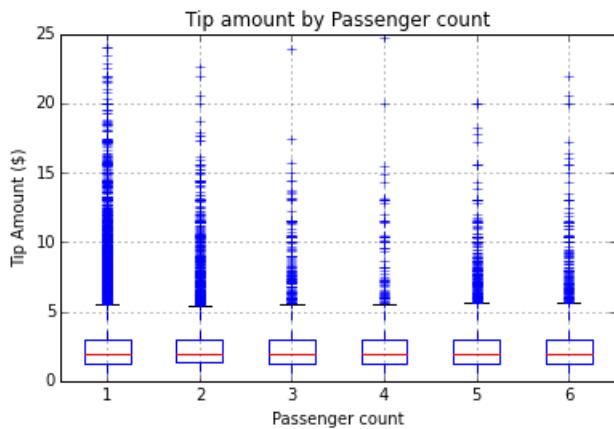
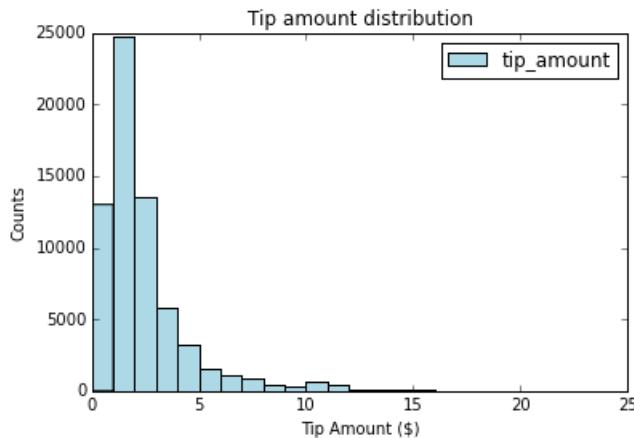
# HISTOGRAM OF TIP AMOUNTS AND PASSENGER COUNT
ax1 = sqlResults[['tip_amount']].plot(kind='hist', bins=25, facecolor='lightblue')
ax1.set_title('Tip amount distribution')
ax1.set_xlabel('Tip Amount ($)')
ax1.set_ylabel('Counts')
plt.suptitle('')
plt.show()

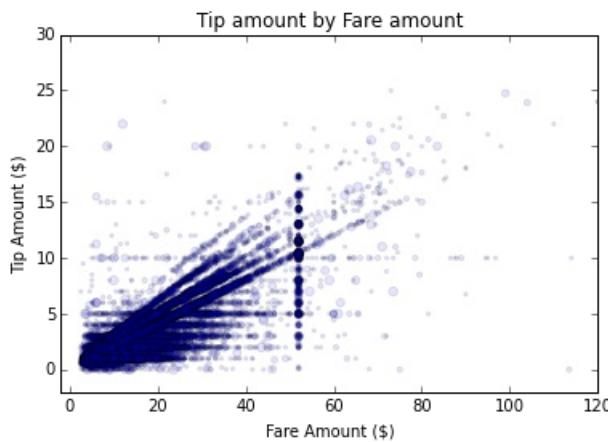
# TIP BY PASSENGER COUNT
ax2 = sqlResults.boxplot(column=['tip_amount'], by=['passenger_count'])
ax2.set_title('Tip amount by Passenger count')
ax2.set_xlabel('Passenger count')
ax2.set_ylabel('Tip Amount ($)')
plt.suptitle('')
plt.show()

# 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(parseRowIndexingBinary)
oneHotTESTbinary = testData.map(parseRowIndexingBinary)

# FOR REGRESSION TRAINING AND TESTING
indexedTRAINreg = trainData.map(parseRowIndexingRegression)
indexedTESTreg = testData.map(parseRowIndexingRegression)
oneHotTRAINreg = trainData.map(parseRowIndexingRegression)
oneHotTESTreg = testData.map(parseRowIndexingRegression)

# 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 to 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.

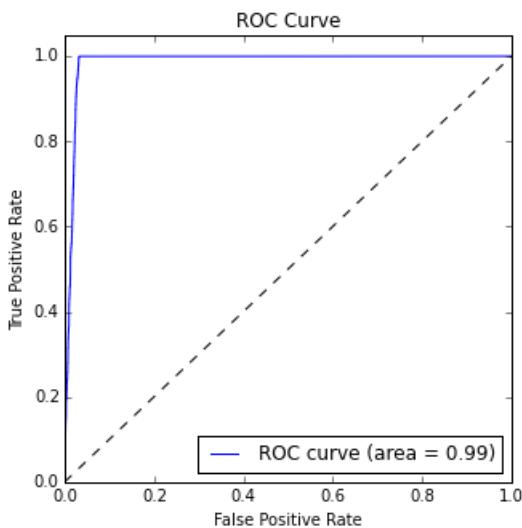
```
# MAKE PREDICTIONS AND PLOT ROC-CURVE

# 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(':', '_');
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 = 0.985297691373

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)

# CONVERT RESULTS TO DF AND REGISTER 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.

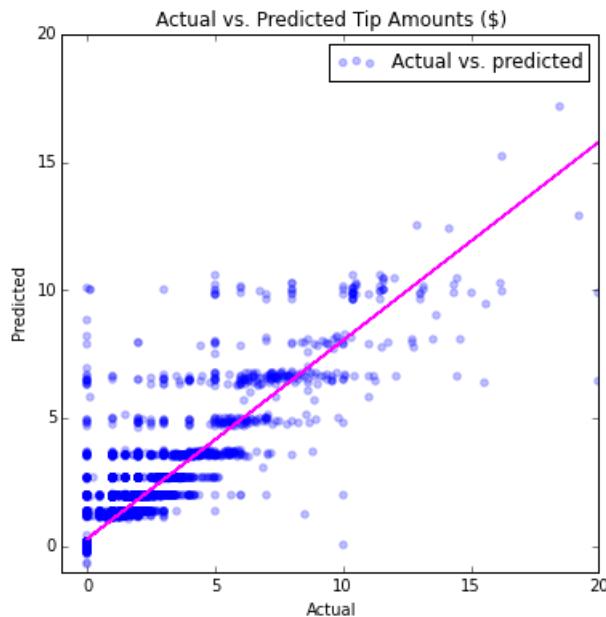
```

# 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

Advanced data exploration and modeling with Spark

11/2/2017 • 38 min to read • [Edit Online](#)

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 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. 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. The ability of the model to predict accurately when tested against the independent dataset in this fold not used to train the model is assessed.

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.

Setup: Spark clusters and notebooks

Setup steps and code are provided in this walkthrough for using an HDInsight Spark 1.6. But Jupyter notebooks are provided for both HDInsight Spark 1.6 and Spark 2.0 clusters. A description of the notebooks and links to them are provided in the [Readme.md](#) for the GitHub repository containing them. Moreover, the code here and in the linked 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. For convenience, here are the links to the Jupyter notebooks for Spark 1.6 and 2.0 to be run in the pyspark kernel of the Jupyter Notebook server:

Spark 1.6 notebooks

[pySpark-machine-learning-data-science-spark-advanced-data-exploration-modeling.ipynb](#): Includes topics in notebook #1, and model development using hyperparameter tuning and cross-validation.

Spark 2.0 notebooks

[Spark2.0-pySpark3-machine-learning-data-science-spark-advanced-data-exploration-modeling.ipynb](#): This file provides information on how to perform data exploration, modeling, and scoring in Spark 2.0 clusters.

WARNING

Billing for HDInsight clusters is prorated per minute, whether you are using them or not. Be sure to delete your cluster after you have finished using it. For more information, 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.Tra
in.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 after the `%%sql -o` magic is used to run a query. The `-o` parameter would persist the output of the SQL query locally. Then the `%%local` magic triggers the next set of code snippets to run locally against the output of the SQL queries that has been 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 -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.

```
# 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

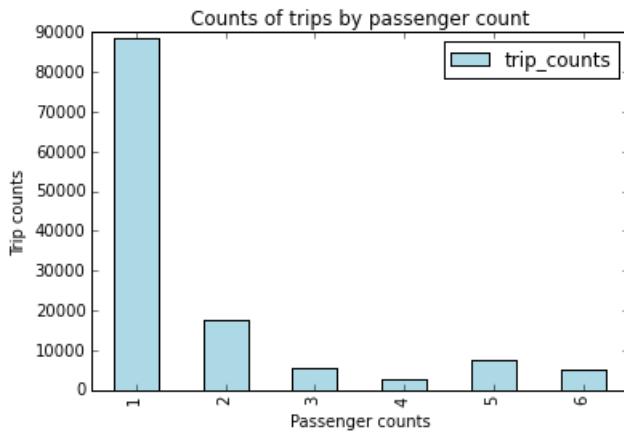
```

# 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 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

```

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

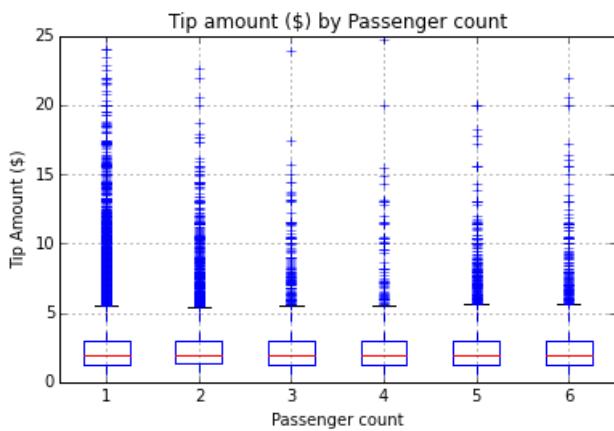
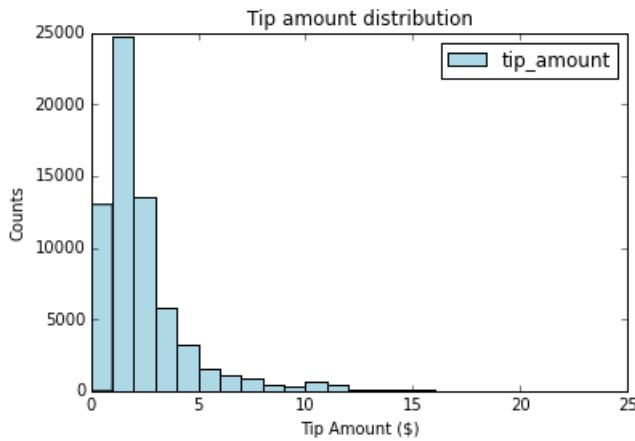
# TIP BY PAYMENT TYPE AND PASSENGER COUNT
ax1 = resultsPDDF[['tip_amount']].plot(kind='hist', bins=25, facecolor='lightblue')
ax1.set_title('Tip amount distribution')
ax1.set_xlabel('Tip Amount ($)')
ax1.set_ylabel('Counts')
plt.suptitle('')
plt.show()

# 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 partitioning hours into traffic time bins
- 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 partitioning traffic times into bins

This code shows how to create a new feature by partitioning traffic times into bins and then how to cache the resulting data frame in memory. Caching leads to improved execution time where Resilient Distributed Datasets (RDDs) and data-frames are used repeatedly. So, we cache RDDs and data-frames at several stages in this 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 that features with categorical input data 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 how to encode it. This prepares it to 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 the data here. Then you know how to use it for your own problem if 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(parseRowIndexingBinary)
oneHotTESTbinary = testData.map(parseRowIndexingBinary)

# FOR REGRESSION TRAINING AND TESTING
indexedTRAINreg = trainData.map(parseRowIndexingRegression)
indexedTESTreg = testData.map(parseRowIndexingRegression)
oneHotTRAINreg = trainData.map(parseRowIndexingRegression)
oneHotTESTreg = testData.map(parseRowIndexingRegression)

# 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). SGD is 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 to 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 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 TRAIING 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), TrafficTimeBinsVec
    (4)
    print("Coefficients: " + str(logitBest.weights))
    print("Intercept: " + str(logitBest.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

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.

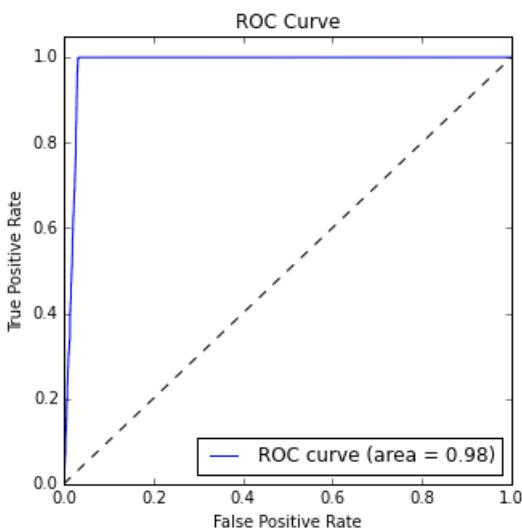
```
# MAKE PREDICTIONS AND PLOT ROC-CURVE

# 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.

```

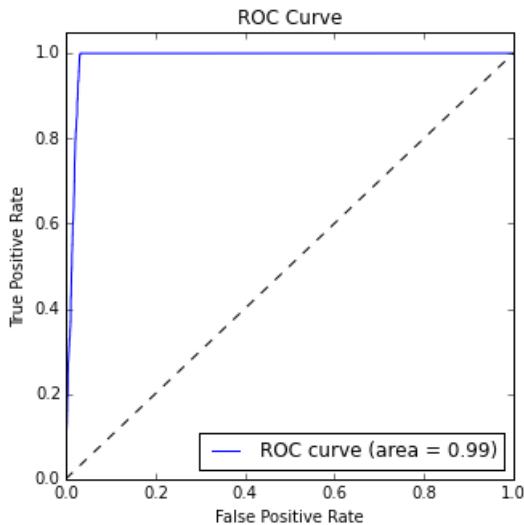
# RUN THE CODE LOCALLY ON THE JUPYTER SERVER AND IMPORT LIBRARIES
%%local
from sklearn.metrics import roc_curve,auc

# 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,
                                         numTrees=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: predict the tip amount 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), 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("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

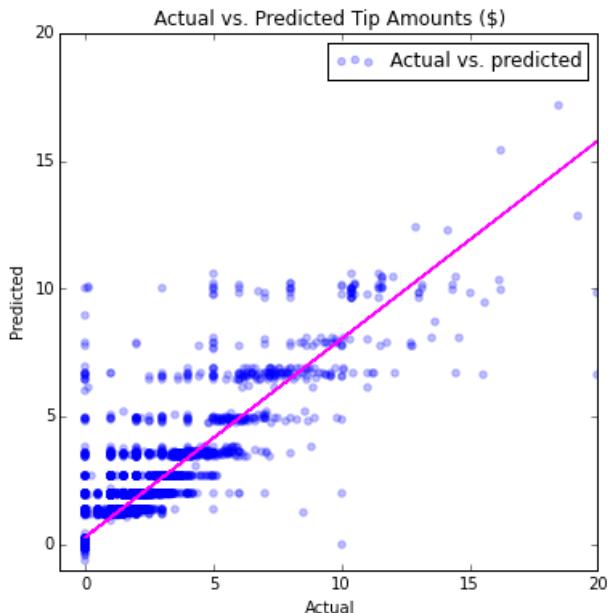
```
# PLOT SCATTER-PLOT BETWEEN ACTUAL AND PREDICTED TIP VALUES

# SELECT RESULTS
%%sql -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
import numpy as np

# PLOT
ax = sqlResults.plot(kind='scatter', figsize = (6,6), x='_1', y='_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](#).

Operationalize Spark-built machine learning models

11/2/2017 • 18 min to read • [Edit Online](#)

This topic shows how to operationalize a saved machine learning model (ML) using Python on HDInsight Spark clusters. It describes how to load machine learning 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.

Spark clusters and Jupyter notebooks

Setup steps and the code to operationalize an ML model are provided in this walkthrough for using an HDInsight Spark 1.6 cluster as well as a Spark 2.0 cluster. The code for these procedures is also provided in Jupyter notebooks.

Notebook for Spark 1.6

The [pySpark-machine-learning-data-science-spark-model-consumption.ipynb](#) Jupyter notebook shows how to operationalize a saved model using Python on HDInsight clusters.

Notebook for Spark 2.0

To modify the Jupyter notebook for Spark 1.6 to use with an HDInsight Spark 2.0 cluster, replace the Python code file with [this file](#). This code shows how to consume the models created in Spark 2.0.

Prerequisites

1. You need an Azure account and a Spark 1.6 (or Spark 2.0) HDInsight 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.
2. You must also create the machine learning models to be scored here by working through the [Data exploration and modeling with Spark](#) topic for the Spark 1.6 cluster or the Spark 2.0 notebooks.
3. The Spark 2.0 notebooks use an additional data set for the classification task, the well-known Airline On-time departure dataset from 2011 and 2012. A description of the notebooks and links to them are provided in the [Readme.md](#) for the GitHub repository containing them. Moreover, the code here and in the linked 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.

WARNING

Billing for HDInsight clusters is prorated per minute, whether you are using them or not. Be sure to delete your cluster after you have finished using it. For more information, 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 `StandardScalar` 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 RERRESSION MODELS
def parseRowOneHotBinary(line):
    features = np.concatenate(([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(([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 LINEARREGRESSIONWITHSGD 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 runs 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 AVAILABLE 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}

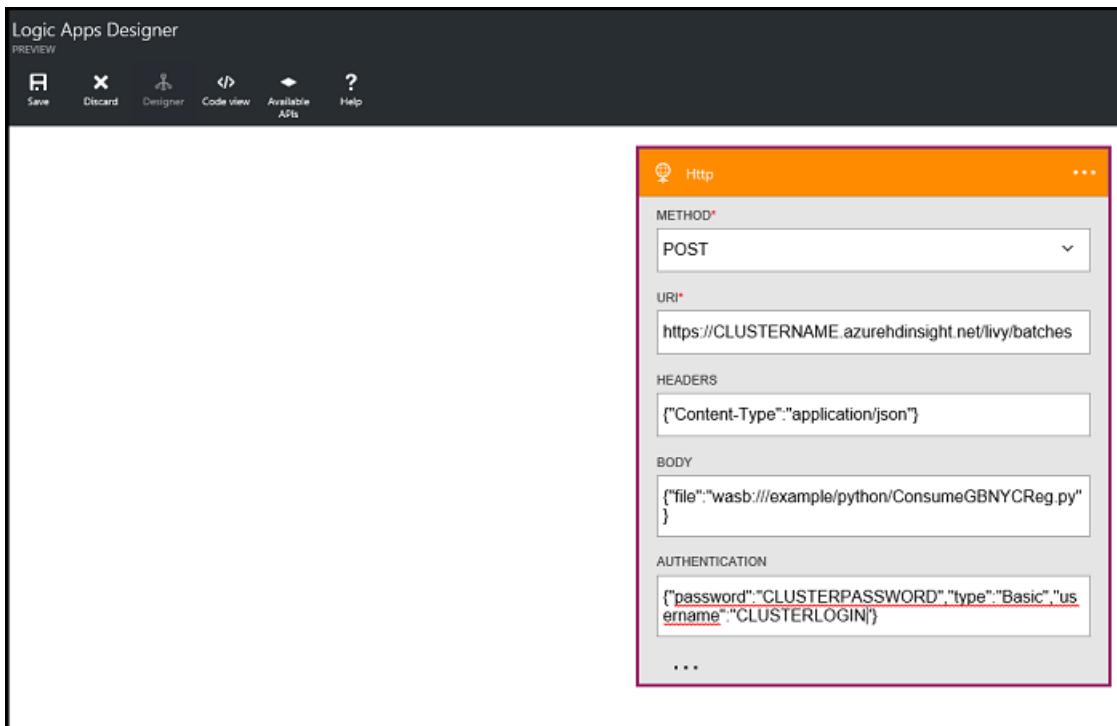
# SPECIFY THE PYTHON SCRIPT TO RUN ON THE SPARK CLUSTER
# IN THE FILE PARAMETER OF THE JSON POST REQUEST BODY
r=conn.request("POST", '/livy/batches', '{"file": "wasb:///example/python/ConsumeGBNYCReg.py"}', headers )
response = conn.getresponse().read()
print(response)
conn.close()

```

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.

HDInsight Hadoop data science walkthroughs using Hive on Azure

11/2/2017 • 1 min to read • [Edit Online](#)

These walkthroughs use Hive with an HDInsight Hadoop cluster to do predictive analytics. They follow the steps outlined in the Team Data Science Process. For an overview of the Team Data Science Process, see [Data Science Process](#). For an introduction to Azure HDInsight, see [Introduction to Azure HDInsight, the Hadoop technology stack, and Hadoop clusters](#).

Additional data science walkthroughs that execute the Team Data Science Process are grouped by the **platform** that they use. See [Walkthroughs executing the Team Data Science Process](#) for an itemization of these examples.

Predict taxi tips using Hive with HDInsight Hadoop

The [Use HDInsight Hadoop clusters](#) walkthrough uses data from New York taxis to predict:

- Whether a tip is paid
- The distribution of tip amounts

The scenario is implemented using Hive with an [Azure HDInsight Hadoop cluster](#). You learn how to store, explore, and feature engineer data from a publicly available NYC taxi trip and fare dataset. You also use Azure Machine Learning to build and deploy the models.

Predict advertisement clicks using Hive with HDInsight Hadoop

The [Use Azure HDInsight Clusters on a 1-TB dataset](#) walkthrough uses a publicly available [Criteo](#) click dataset to predict whether a tip is paid and the range of amounts expected. The scenario is implemented using Hive with an [Azure HDInsight Hadoop cluster](#) to store, explore, feature engineer, and down sample data. It uses Azure Machine Learning to build, train, and score a binary classification model predicting whether a user clicks on an advertisement. The walkthrough concludes showing how to publish one of these models as a Web service.

Next steps

For a discussion of the key components that comprise the Team Data Science Process, see [Team Data Science Process overview](#).

For a discussion of the Team Data Science Process lifecycle that you can use to structure your data science projects, see [Team Data Science Process lifecycle](#). The lifecycle outlines the steps, from start to finish, that projects usually follow when they are executed.

Azure Data Lake data science walkthroughs using U-SQL

9/25/2017 • 1 min to read • [Edit Online](#)

These walkthroughs use U-SQL with Azure Data Lake to do predictive analytics. They follow the steps outlined in the Team Data Science Process. For an overview of the Team Data Science Process, see [Data Science Process](#). For an introduction to Azure Data Lake, see [Overview of Azure Data Lake Store](#).

Additional data science walkthroughs that execute the Team Data Science Process are grouped by the **platform** that they use. See [Walkthroughs executing the Team Data Science Process](#) for an itemization of these examples.

Predict taxi tips using U-SQL with Azure Data Lake

The [Use Azure Data Lake for data science](#) 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.

Next steps

For a discussion of the key components that comprise the Team Data Science Process, see [Team Data Science Process overview](#).

For a discussion of the Team Data Science Process lifecycle that you can use to structure your data science projects, see [Team Data Science Process lifecycle](#). The lifecycle outlines the steps, from start to finish, that projects usually follow when they are executed.

SQL Server data science walkthroughs using R, Python and T-SQL

12/7/2017 • 1 min to read • [Edit Online](#)

These walkthroughs use SQL Server, SQL Server R Services, and SQL Server Python Services to do predictive analytics. R and Python code is deployed in stored procedures. They follow the steps outlined in the Team Data Science Process. For an overview of the Team Data Science Process, see [Data Science Process](#).

Additional data science walkthroughs that execute the Team Data Science Process are grouped by the **platform** that they use. See [Walkthroughs executing the Team Data Science Process](#) for an itemization of these examples.

Predict taxi tips using Python and SQL queries with SQL Server

The [Use SQL Server](#) walkthrough shows how you build and deploy machine learning classification and regression models using SQL Server and a publicly available NYC taxi trip and fare dataset.

Predict taxi tips using Microsoft R with SQL Server

The [Use 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. The walkthrough is designed to introduce R developers to R Services (In-Database).

Predict taxi tips using R from T-SQL or stored procedures with SQL Server

The [Data science walkthrough for R and SQL Server](#) provides SQL programmers with experience building an advanced analytics solution with Transact-SQL using SQL Server R Services to operationalize an R solution.

Predict taxi tips using Python in SQL Server stored procedures

The [Use T-SQL with SQL Server Python Services](#) walkthrough provides SQL programmers with experience building a machine learning solution in SQL Server. It demonstrates how to incorporate Python into an application by adding Python code to stored procedures.

Next steps

For a discussion of the key components that comprise the Team Data Science Process, see [Team Data Science Process overview](#).

For a discussion of the Team Data Science Process lifecycle that you can use to structure your data science projects, see [Team Data Science Process lifecycle](#). The lifecycle outlines the steps, from start to finish, that projects usually follow when they are executed.

SQL Data Warehouse data science walkthroughs using T-SQL and Python on Azure

9/25/2017 • 1 min to read • [Edit Online](#)

These walkthroughs use of SQL Data Warehouse to do predictive analytics. They follow the steps outlined in the Team Data Science Process. For an overview of the Team Data Science Process, see [Data Science Process](#). For an introduction to SQL Data Warehouse, see [What is Azure SQL Data Warehouse?](#)

Additional data science walkthroughs that execute the Team Data Science Process are grouped by the **platform** that they use. See [Walkthroughs executing the Team Data Science Process](#) for an itemization of these examples.

Predict taxi tips using T-SQL and IPython notebooks with SQL Data Warehouse

The [Use 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.

Next steps

For a discussion of the key components that comprise the Team Data Science Process, see [Team Data Science Process overview](#).

For a discussion of the Team Data Science Process lifecycle that you can use to structure your data science projects, see [Team Data Science Process lifecycle](#). The lifecycle outlines the steps, from start to finish, that projects usually follow when they are executed.

Team Data Science Process for data scientists

2/13/2018 • 8 min to read • [Edit Online](#)

This article provides guidance to a set of objectives that are typically used to implement comprehensive data science solutions with Azure technologies. You are guided through:

- understanding an analytics workload
- using the Team Data Science Process
- using Azure Machine Learning
- the foundations of data transfer and storage
- providing data source documentation
- using tools for analytics processing

These training materials are related to the Team Data Science Process (TDSP) and Microsoft and open-source software and toolkits, which are helpful for envisioning, executing and delivering data science solutions.

Lesson Path

You can use the items in the following table to guide your own self-study. Read the *Description* column to follow the path, click on the *Topic* links for study references, and check your skills using the *Knowledge Check* column.

OBJECTIVE	TOPIC	DESCRIPTION	KNOWLEDGE CHECK
Understand the processes for developing analytic projects	An introduction to the Team Data Science Process	We begin by covering an overview of the Team Data Science Process – the TDSP. This process guides you through each step of an analytics project. Read through each of these sections to learn more about the process and how you can implement it.	Review and download the TDSP Project Structure artifacts to your local machine for your project.
	Agile Development	The Team Data Science Process works well with many different programming methodologies. In this Learning Path, we use Agile software development. Read through the "What is Agile Development?" and "Building Agile Culture" articles, which cover the basics of working with Agile. There are also other references at this site where you can learn more.	Explain Continuous Integration and Continuous Delivery to a colleague.

OBJECTIVE	TOPIC	DESCRIPTION	KNOWLEDGE CHECK
	DevOps for Data Science	Developer Operations (DevOps) involves people, processes, and platforms you can use to work through a project and integrate your solution into an organization's standard IT. This integration is essential for adoption, safety, and security. In this online course, you learn about DevOps practices as well as understand some of the toolchain options you have.	Prepare a 30-minute presentation to a technical audience on how DevOps is essential for analytics projects.
Understand the Technologies for Data Storage and Processing	Microsoft Business Analytics and AI	We focus on a few technologies in this Learning Path that you can use to create an analytics solution, but Microsoft has many more. To understand the options you have, it's important to review the platforms and features available in Microsoft Azure, the Azure Stack, and on-premises options. Review this resource to learn the various tools you have available to answer analytics question.	Download and review the presentation materials from this workshop.
Setup and Configure your training, development, and production environments	Microsoft Azure	Now let's create an account in Microsoft Azure for training and learn how to create development and test environments. These free training resources get you started. Complete the "Beginner" and "Intermediate" paths.	If you do not have an Azure Account, create one. Log in to the Microsoft Azure portal and create one Resource Group for training.
	The Microsoft Azure Command-Line Interface (CLI)	There are multiple ways of working with Microsoft Azure – from graphical tools like VSCode and Visual Studio, to Web interfaces such as the Azure portal, and from the command line, such as Azure PowerShell commands and functions. In this article, we cover the Command-Line Interface (CLI), which you can use locally on your workstation, in Windows and other Operating Systems, as well as in the Azure portal.	Set your default subscription with the Azure CLI.

OBJECTIVE	TOPIC	DESCRIPTION	KNOWLEDGE CHECK
	Microsoft Azure Storage	You need a place to store your data. In this article, you learn about Microsoft Azure's storage options, how to create a storage account, and how to copy or move data to the cloud. Read through this introduction to learn more.	Create a Storage Account in your training Resource Group , create a container for a Blob object, and upload and download data.
	Microsoft Azure Active Directory	Microsoft Azure Active Directory (AAD) forms the basis of securing your application. In this article, you learn more about accounts, rights, and permissions. Active Directory and security are complex topics, so just read through this resource to understand the fundamentals.	Add one user to Azure Active Directory . NOTE: You may not have permissions for this action if you are not the administrator for the subscription. If that's the case, simply review this tutorial to learn more .
	The Microsoft Azure Data Science Virtual Machine	You can install the tools for working with Data Science locally on multiple operating systems. But the Microsoft Azure Data Science Virtual Machine (DSVM) contains all of the tools you need and plenty of project samples to work with. In this article, you learn more about the DVSM and how to work through its examples. This resource explains the Data Science Virtual Machine, how you can create one, and a few options for developing code with it. It also contains all the software you need to complete this learning path – so make sure you complete the Knowledge Path for this topic.	Create a Data Science Virtual Machine and work through at least one lab.
Install and Understand the tools and technologies for working with Data Science solutions			

OBJECTIVE	TOPIC	DESCRIPTION	KNOWLEDGE CHECK
Working with git	To follow our DevOps process with the TDSP, we need to have a version-control system. Microsoft Azure Machine Learning Services uses git, a popular open-source distributed repository system. In this article, you learn more about how to install, configure, and work with git and a central repository – github.	Clone this github project for your learning path project structure.	
	VSCode	VSCode is a cross-platform Integrated Development Environment (IDE) that you can use with multiple languages and Azure tools. You can use this single environment to create your entire solution. Watch these introductory videos to get started.	Install VSCode, and work through the VS Code features in the Interactive Editor Playground.
	Programming with Python	In this solution we use Python, one of the most popular languages in Data Science. This article covers the basics of writing analytic code with Python, and resources to learn more. Work through sections 1-9 of this reference, then check your knowledge.	Add one entity to an Azure Table using Python.
	Working with Notebooks	Notebooks are a way of introducing text and code in the same document. Azure Machine Learning Services work with Notebooks, so it is beneficial to understand how to use them. Read through this tutorial and give it a try in the Knowledge Check section.	Open this page, and click on the "Welcome to Python.ipynb" link. Work through the examples on that page.
	Machine Learning		

OBJECTIVE	TOPIC	DESCRIPTION	KNOWLEDGE CHECK
Creating advanced Analytic solutions involves working with data, using Machine Learning, which also forms the basis of working with Artificial Intelligence and Deep Learning. This course teaches you more about Machine Learning. For a comprehensive course on Data Science, check out this certification.	Locate a resource on Machine Learning Algorithms. (Hint: Search on "azure machine learning algorithm cheat sheet")		
	scikit-learn	The scikit-learn set of tools allows you to perform data science tasks in Python. We use this framework in our solution. This article covers the basics and explains where you can learn more.	Using the Iris dataset, persist an SVM model using Pickle.
	Working with Docker	Docker is a distributed platform used to build, ship, and run applications, and is used frequently in Azure Machine Learning services. This article covers the basics of this technology and explains where you can go to learn more.	Open Visual Studio Code, and install the Docker Extension . Create a simple Node Docker container.
	HDInsight	HDInsight is the Hadoop open-source infrastructure, available as a service in Microsoft Azure. Your Machine Learning algorithms may involve large sets of data, and HDInsight has the ability to store, transfer and process data at large scale. This article covers working with HDInsight.	Create a small HDInsight cluster . Use HiveQL statements to project columns onto an /example/data/sample.log file . Alternatively, you can complete this knowledge check on your local system .

OBJECTIVE	TOPIC	DESCRIPTION	KNOWLEDGE CHECK
Create a Data Processing Flow from Business Requirements	Determining the Question, following the TDSP	With the development environment installed and configured, and the understanding of the technologies and processes in place, it's time to put everything together using the TDSP to perform an analysis. We need to start by defining the question, selecting the data sources, and the rest of the steps in the Team Data Science Process. Keep in mind the DevOps process as we work through this process. In this article, you learn how to take the requirements from your organization and create a data flow map through your application to define your solution using the Team Data Science Process.	
Locate a resource on " The 5 data science questions " and describe one question your organization might have in these areas. Which algorithms should you focus on for that question?			
Use Azure Machine Learning Services to create a predictive solution	Azure Machine Learning Services	Microsoft Azure Machine Learning includes working with data sources, using AI for data wrangling and feature engineering, creating experiments, and tracking model runs. All of this works in a single environment and most functions can run locally or in Azure. You can use the Cognitive Network Toolkit (CNTK), TensorFlow, and other frameworks to create your experiments. In this article, we focus on a complete example of this process, using everything you've learned so far.	Complete this tutorial on Azure Machine Learning Services and the TDSP.

OBJECTIVE	TOPIC	DESCRIPTION	KNOWLEDGE CHECK
Use Power BI to visualize results	Power BI	Power BI is Microsoft's data visualization tool. It is available on multiple platforms from Web to mobile devices and desktop computers. In this article you learn how to work with the output of the solution you've created by accessing the results from Azure storage and creating visualizations using Power BI.	Complete this tutorial on Power BI . Then connect Power BI to the Blob CSV created in an experiment run.
Monitor your Solution	Application Insights	There are multiple tools you can use to monitor your end solution. Azure Application Insights makes it easy to integrate built-in monitoring into your solution.	Set up Application Insights to monitor an Application .
	Azure Log Analytics	Another method to monitor your application is to integrate it into your DevOps process. The Azure Log Analytics system provides a rich set of features to help you watch your analytic solutions after you deploy them.	Complete this tutorial on using Azure Log Analytics .
Complete this Learning Path	Additional Projects to try	Congratulations! You've completed this learning path. There is a lot more to learn. A more advanced example is building a customer churn model in Azure Machine Learning Services. Try it out here .	

Next steps

[Team Data Science Process for Developer Operations](#) This article explores the Developer Operations (DevOps) functions that are specific to an Advanced Analytics and Cognitive Services solution implementation.

Team Data Science Process for Developer Operations

11/29/2017 • 10 min to read • [Edit Online](#)

This article explores the Developer Operations (DevOps) functions that are specific to an Advanced Analytics and Cognitive Services solution implementation. These training materials are related to the Team Data Science Process (TDSP) and Microsoft and open-source software and toolkits, which are helpful for envisioning, executing and delivering data science solutions. It references topics that cover the DevOps Toolchain that is specific to Data Science and AI projects and solutions.

Lesson Path

The following table provides guidance at specified levels to help complete the DevOps objectives that are needed to implement data science solutions with Azure technologies.

OBJECTIVE	TOPIC	RESOURCE	TECHNOLOGIES	LEVEL	PREREQUISITES
Understand Advanced Analytics	The Team Data Science Process Lifecycle	This technical walkthrough describes the Team Data Science Process	Data Science	Intermediate	General technology background, familiarity with data solutions, Familiarity with IT projects and solution implementation
Understand the Microsoft Azure Platform for Advanced Analytics	Information Management				
This reference gives and overview of Azure Data Factory to build pipelines to collect and orchestrate data from the services you use for analysis	Microsoft Azure Data Factory	Experienced	General technology background, familiarity with data solutions, Familiarity with IT projects and solution implementation		

OBJECTIVE	TOPIC	RESOURCE	TECHNOLOGIES	LEVEL	PREREQUISITES
This reference covers an overview of the Azure Data Catalog which you can use to document and manage metadata on your data sources	Microsoft Azure Data Catalog	Intermediate	General technology background, familiarity with data solutions, familiarity with Relational Database Management Systems (RDBMS) and NoSQL data sources		
This reference covers an overview of the Azure Event Hubs system and how you and use it to ingest data into your solution	Azure Event Hubs	Intermediate	General technology background, familiarity with data solutions, familiarity with Relational Database Management Systems (RDBMS) and NoSQL data sources, familiarity with the Internet of Things (IoT) terminology and use		
	Big Data Stores				
This reference covers an overview of using the Azure SQL Data Warehouse to store and process large amounts of data	Azure SQL Data Warehouse	Experienced	General technology background, familiarity with data solutions, familiarity with Relational Database Management Systems (RDBMS) and NoSQL data sources, familiarity with HDFS terminology and use		

OBJECTIVE	TOPIC	RESOURCE	TECHNOLOGIES	LEVEL	PREREQUISITES
		This reference covers an overview of using Azure Data Lake to capture data of any size, type, and ingestion speed in one single place for operational and exploratory analytics	Azure Data Lake Store	Intermediate	General technology background, familiarity with data solutions, familiarity with NoSQL data sources, familiarity with HDFS
	Machine Learning and Analytics	This reference covers an introduction to Machine Learning, predictive analytics, and Artificial Intelligence systems	Azure Machine Learning	Intermediate	General technology background, familiarity with data solutions, familiarity with Data Science terms, familiarity with Machine Learning and Artificial Intelligence terms
		This article provides an introduction to Azure HDInsight, a cloud distribution of the Hadoop technology stack. It also covers what a Hadoop cluster is and when you would use it	Azure HDInsight	Intermediate	General technology background, familiarity with data solutions, familiarity with NoSQL data sources
		This reference covers an overview of the Azure Data Lake Analytics job service	Azure Data Lake Analytics	Intermediate	General technology background, familiarity with data solutions, familiarity with NoSQL data sources
		This overview covers using Azure Stream Analytics as a fully-managed event-processing engine to perform real-time analytic computations on streaming data	Azure Stream Analytics	Intermediate	General technology background, familiarity with data solutions, familiarity with structured and unstructured data concepts

OBJECTIVE	TOPIC	RESOURCE	TECHNOLOGIES	LEVEL	PREREQUISITES
	Intelligence	This reference covers an overview of the available Cognitive Services (such as vision, text, and search) and how to get started using them	Cognitive Services	Experienced	General technology background, familiarity with data solutions, software development
		This reference covers and introduction to the Microsoft Bot Framework and how to get started using it	Bot Framework	Experienced	General technology background, familiarity with data solutions
	Visualization	This self-paced, online course covers the Power BI system, and how to create and publish reports	Microsoft Power BI	Beginner	General technology background, familiarity with data solutions
	Solutions	This resource page covers multiple applications you can review, test and implement to see a complete solution from start to finish	Microsoft Azure, Azure Machine Learning, Cognitive Services, Microsoft R, Azure Search, Python, Azure Data Factory, Power BI, Azure Document DB, Application Insights, Azure SQL DB, Azure SQL Data Warehouse, Microsoft SQL Server, Azure Data Lake, Cognitive Services, Bot Framework, Azure Batch,	Intermediate	General technology background, familiarity with data solutions

OBJECTIVE	TOPIC	RESOURCE	TECHNOLOGIES	LEVEL	PREREQUISITES
Understand and Implement DevOps Processes	DevOps Fundamentals	This video series explains the covers the fundamentals of DevOps and helps you understand how they map to DevOps practices, and how they can be implemented by a variety of products and tools	DevOps, Microsoft Azure Platform, Visual Studio Team Services	Experienced	Used an SDLC, familiarity with Agile and other Development Frameworks, IT Operations Familiarity
Use the DevOps Toolchain for Data Science	Configure	This reference covers the basics of choosing the proper visualization in Visio to communicate your project design	Visio	Intermediate	General technology background, familiarity with data solutions
		This reference describes the Azure Resource Manager, terms, and serves as the primary root source for samples, getting started, and other references	Azure Resource Manager, Azure PowerShell, Azure CLI	Intermediate	General technology background, familiarity with data solutions
		This reference explains the Azure Data Science Virtual Machines for Linux and Windows	Data Science Virtual Machine	Experienced	Familiarity with Data Science Workloads, Linux
		This walkthrough explains configuring Azure cloud service roles with Visual Studio - pay close attention to the connection strings specifically for storage accounts	Visual Studio	Intermediate	Software Development

OBJECTIVE	TOPIC	RESOURCE	TECHNOLOGIES	LEVEL	PREREQUISITES
		This series teaches you how to use Microsoft Project to schedule time, resources and goals for an Advanced Analytics project	Microsoft Project	Intermediate	Understand Project Management Fundamentals
		This Microsoft Project template provides a time, resources and goals tracking for an Advanced Analytics project	Microsoft Project	Intermediate	Understand Project Management Fundamentals
		This tutorial helps you get started with Azure Data Catalog, a fully managed cloud service that serves as a system of registration and system of discovery for enterprise data assets	Azure Data Catalog	Beginner	Familiarity with Data Sources and Structures
		This Microsoft Virtual Academy course explains how to set up Dev/Test with Visual Studio Online and Microsoft Azure	Visual Studio Online	Experienced	Software Development, familiarity with Dev/Test environments
		This Management Pack download for Microsoft System Center contains a Guidelines Document to assist in working with Azure assets	System Center	Intermediate	Experience with System Center for IT Management

OBJECTIVE	TOPIC	RESOURCE	TECHNOLOGIES	LEVEL	PREREQUISITES
		This document is intended for developer and operations teams to understand the benefits of PowerShell Desired State Configuration	PowerShell DSC	Intermediate	Experience with PowerShell coding, enterprise architectures, scripting
	Code	This download also contains documentation on using Visual Studio Online Code for creating Data Science and AI applications	Visual Studio Online	Intermediate	Software Development
		This getting started site teaches you about DevOps and Visual Studio	Visual Studio	Beginner	Software Development
		You can write code directly from the Azure Portal using the App Service Editor. Learn more at this resource about Continuous Integration with this tool	Azure Portal	Highly Experienced	Data Science background - but read this anyway
		This resource explains how to code and create Predictive Analytics experiments using the web-based Azure ML Studio tool	Azure ML Studio	Experienced	Software Development
		This reference contains a list and a study link to all of the development tools on the Data Science Virtual Machine in Azure	Data Science Virtual Machine	Experienced	Software Development, Data Science

OBJECTIVE	TOPIC	RESOURCE	TECHNOLOGIES	LEVEL	PREREQUISITES
		Read and understand each of the references in this Azure Security Trust Center for Security, Privacy, and Compliance - VERY important	Azure Security	Intermediate	System Architecture Experience, Security Development experience
	Build	This course teaches you about enabling DevOps Practices with Visual Studio Online Build	Visual Studio Online	Experienced	Software Development, Familiarity with an SDLC
		This reference explains compiling and building using Visual Studio	Visual Studio	Intermediate	Software Development, Familiarity with an SDLC
		This reference explains how to orchestrate processes such as software builds with Runbooks	System Center	Experienced	Experience with System Center Orchestrator
	Test	Use this reference to understand how to use Visual Studio Online for Test Case Management	Visual Studio Online	Experienced	Software Development, Familiarity with an SDLC
		Use this previous reference for Runbooks to automate tests using System Center	System Center	Experienced	Experience with System Center Orchestrator
		As part of not only testing but development, you should build in Security. The Microsoft SDL Threat Modeling Tool can help in all phases. Learn more and download it here	Threat Monitoring Tool	Experienced	Familiarity with security concepts, software development

OBJECTIVE	TOPIC	RESOURCE	TECHNOLOGIES	LEVEL	PREREQUISITES
		This article explains how to use the Microsoft Attack Surface Analyzer to test your Advanced Analytics solution	Attack Surface Analyzer	Experienced	Familiarity with security concepts, software development
	Package	This reference explains the concepts of working with Packages in TFS and VSO	Visual Studio Online	Experienced	Software development, familiarity with an SDLC
		Use this previous reference for Runbooks to automate packaging using System Center	System Center	Experienced	Experience with System Center Orchestrator
		This reference explains how to create a data pipeline for your solution, which you can save as a JSON template as a "package"	Azure Data Factory	Intermediate	General computing background, data project experience
		This topic describes the structure of an Azure Resource Manager template	Azure Resource Manager	Intermediate	Familiarity with the Microsoft Azure Platform
		DSC is a management platform in PowerShell that enables you to manage your IT and development infrastructure with configuration as code, saved as a package. This reference is an overview for that topic	PowerShell Desired State Configuration	Intermediate	PowerShell coding, familiarity with enterprise architectures, scripting

OBJECTIVE	TOPIC	RESOURCE	TECHNOLOGIES	LEVEL	PREREQUISITES
	Release	This head-reference article contains concepts for build, test, and release for CI/CD environments	Visual Studio Online	Experienced	Software development, familiarity with CI/CD environments, familiarity with an SDLC
		Use this previous reference for Runbooks to automate release management using System Center	System Center	Experienced	Experience with System Center Orchestrator
		This article helps you determine the best option to deploy the files for your web app, mobile app backend, or API app to Azure App Service, and then guides you to appropriate resources with instructions specific to your preferred option	Microsoft Azure Deployment	Intermediate	Software development, experience with the Microsoft Azure platform
	Monitor	This reference explains Application Insights and how you can add it to your Advanced Analytics Solutions	Application Insights	Intermediate	Software Development, familiarity with the Microsoft Azure platform
		This topic explains basic concepts about Operations Manager for the administrator who manages the Operations Manager infrastructure and the operator who monitors and supports the Advanced Analytics Solution	System Center	Experienced	Familiarity with enterprise monitoring, System Center Operations Manager

OBJECTIVE	TOPIC	RESOURCE	TECHNOLOGIES	LEVEL	PREREQUISITES
		This blog entry explains how to use the Azure Data Factory to monitor and manage the Advanced Analytics pipeline	Azure Data Factory	Intermediate	Familiarity with Azure Data Factory
		This video shows how to monitor a log with Azure Log Analytics	Azure Logs, PowerShell	Experienced	Familiarity with the Azure Platform
Understand how to use Open Source DevOps Tools with DevOps on Azure	Open Source DevOps Tools and Azure	This reference page contains two videos and a whitepaper on using Chef with Azure deployments	Chef	Experienced	Familiarity with the Azure Platform, Familiarity with DevOps
		This site has a toolchain selection path	DevOps, Microsoft Azure Platform, Visual Studio Team Services, Open Source Software	Experienced	Used an SDLC, familiarity with Agile and other Development Frameworks, IT Operations Familiarity
		This tutorial explains how to automate the build and test phase of application development, using a continuous integration and deployment CI/CD pipeline	Jenkins	Experienced	Familiarity with the Azure Platform, Familiarity with DevOps, Familiarity with Jenkins
		This contains an overview of working with Docker and Azure as well as additional references for implementation for Data Science applications	Docker	Intermediate	Familiarity with the Azure Platform, Familiarity with Server Operating Systems

OBJECTIVE	TOPIC	RESOURCE	TECHNOLOGIES	LEVEL	PREREQUISITES
		This installation and explanation explains how to use Visual Studio Code with Azure assets	VSCODE	Intermediate	Software Development, familiarity with the Microsoft Azure Platform
		This blog entry explains how to use R Studio with Microsoft R	R Studio	Intermediate	R Language experience
		This blog entry shows how to use continuous integration with Azure and GitHub	git, github	Intermediate	Software Development

Next steps

[Team Data Science Process for data scientists](#) This article provides guidance to a set of objectives that are typically used to implement comprehensive data science solutions with Azure technologies.

Structure projects with the Team Data Science Process template

12/18/2017 • 5 min to read • [Edit Online](#)

This document provides instructions on how to create data science projects in Azure Machine Learning with Team Data Science Process (TDSP) templates. These templates help to structure projects for collaboration and reproducibility.

What is the Team Data Science Process?

The TDSP is an agile, iterative, data science process for executing and delivering advanced analytics solutions. It's designed to improve the collaboration and efficiency of data science teams in enterprise organizations. It supports these objectives with four key components:

- A standard [data science lifecycle](#) definition.
- A standardized project structure, [project documentation, and reporting templates](#).
- An infrastructure and resources for project execution, such as, respectively, a compute and storage infrastructure and code repositories.
- [Tools and utilities](#) for data science project tasks, such as:
 - Collaborative version control
 - Code review
 - Data exploration and modeling
 - Work planning

For a more complete discussion of the TDSP, see the [Team Data Science Process overview](#).

Why should you use the TDSP structure and templates?

Standardization of the structure, lifecycle, and documentation of data science projects is key to facilitating effective collaboration on data science teams. Create machine learning projects with the TDSP template to provide such a framework for coordinated teamwork.

We previously released a [GitHub repository for the TDSP project structure and templates](#) to help achieve these objectives. But it was not possible, until now, to instantiate the TDSP structure and templates within a data science tool. It's now possible to create a machine learning project that instantiates the TDSP structure and documentation templates.

Things to note before creating a new project

Review the following items *before* you create a new project:

- Review the TDSP Machine Learning [template](#).
- The contents (other than what is already present in the "docs" folder) are required to be less than 25 MB in size. See the note that follows this list.
- The sample_data folder is only for small data files (less than 5 MB) with which you can test your code or start early development.
- Storing files, such as Word and PowerPoint files, can increase the size of the "docs" folder substantially. We advise that you to find a collaborative Wiki, [SharePoint](#), or other collaborative resource to store such files.
- To learn how to handle large files and outputs in Machine Learning, read [Persisting changes and dealing with](#)

large files.

NOTE

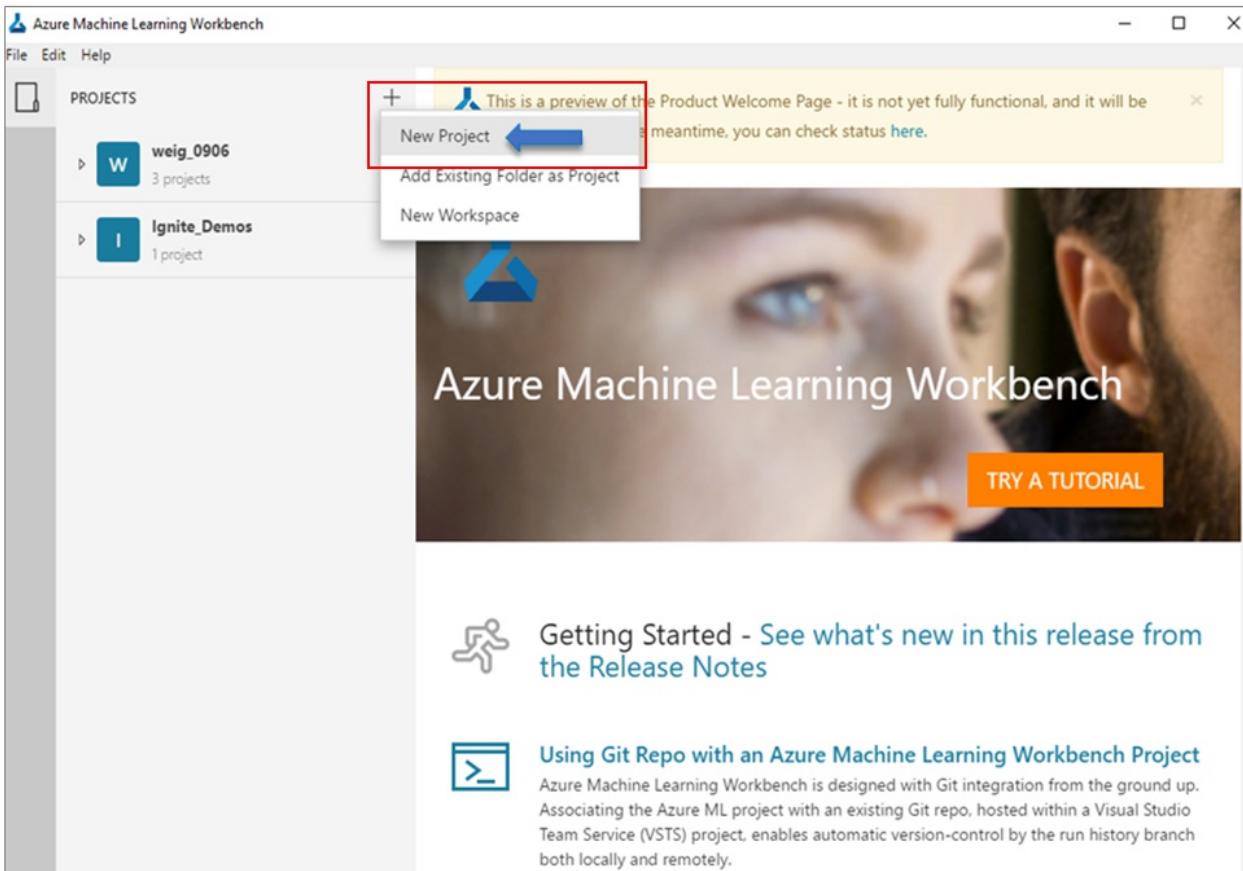
All documentation-related content (text, markdowns, images, and other document files) that is *not* used during project execution, other than the readme.md file, must reside in the folder named "docs" (all lowercase). The "docs" folder is a special folder ignored by Machine Learning execution so that the contents in this folder don't get copied to compute targets unnecessarily. Objects in this folder also don't count toward the 25 MB cap for the project size. The "docs" folder, for example, is the place to store large image files needed in your documentation. These files are still tracked by Git through the run history.

Instantiate the TDSP structure and templates from the Machine Learning template gallery

To create a new project with the TDSP structure and documentation templates, complete the following procedures.

Create a new project

To create a new project, open Azure Machine Learning. Under **Projects** on the top-left pane, select the plus sign (+), and then select **New Project**.



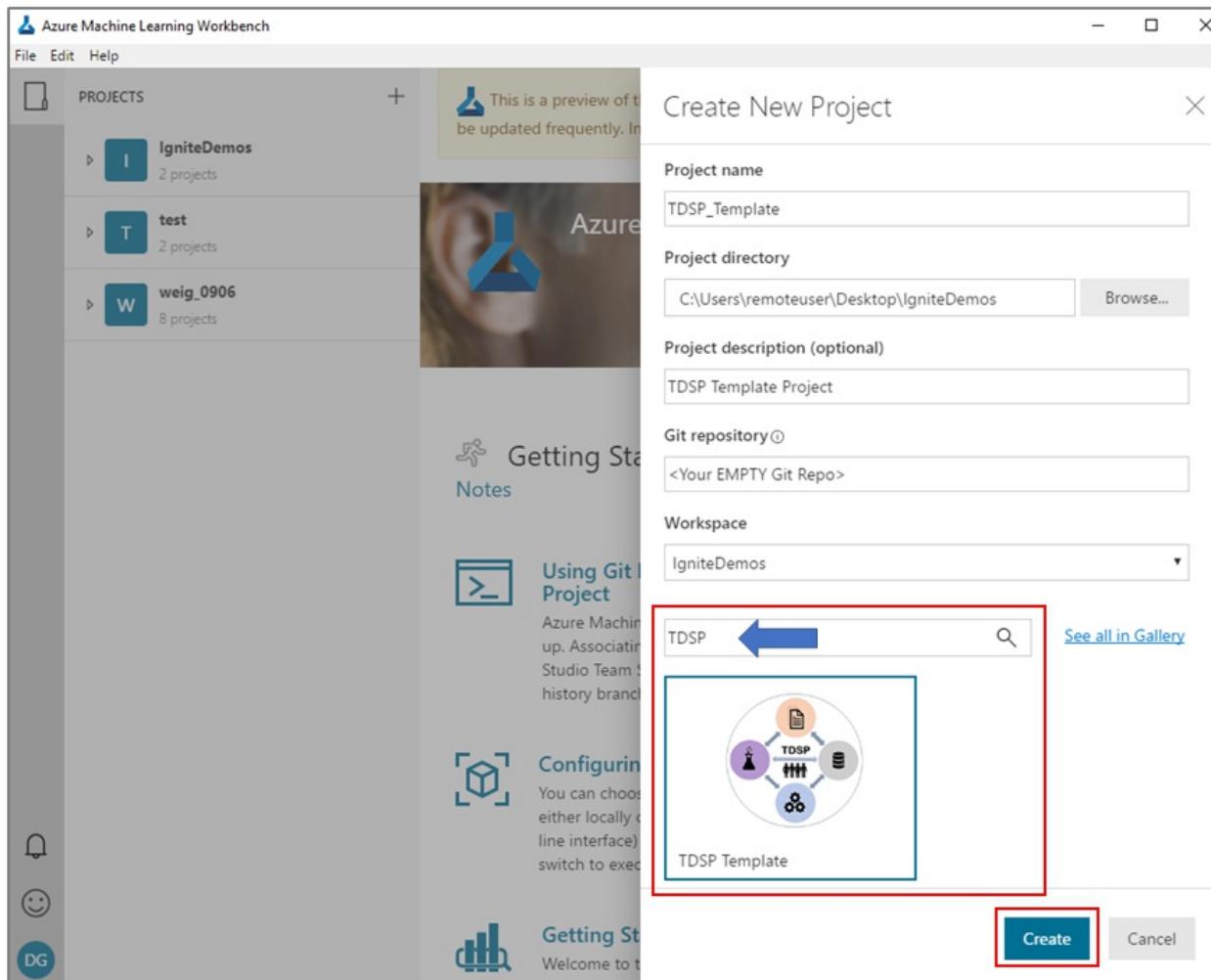
Create a new TDSP-structured project

1. Specify the parameters and information in the relevant box or list:

- Project name
- Project directory
- Project description
- An empty Git repository path
- Workspace name

2. Then in the **Search** box, enter **TDSP**.

3. When the **Structure a project with TDSP** option appears, select that template.
4. Select the **Create** button to create your new project with a TDSP structure. If you provide an empty Git repository when you create the project (in the appropriate text box), then that repository will populate with the project structure and contents after creation of the project.



Examine the TDSP project structure

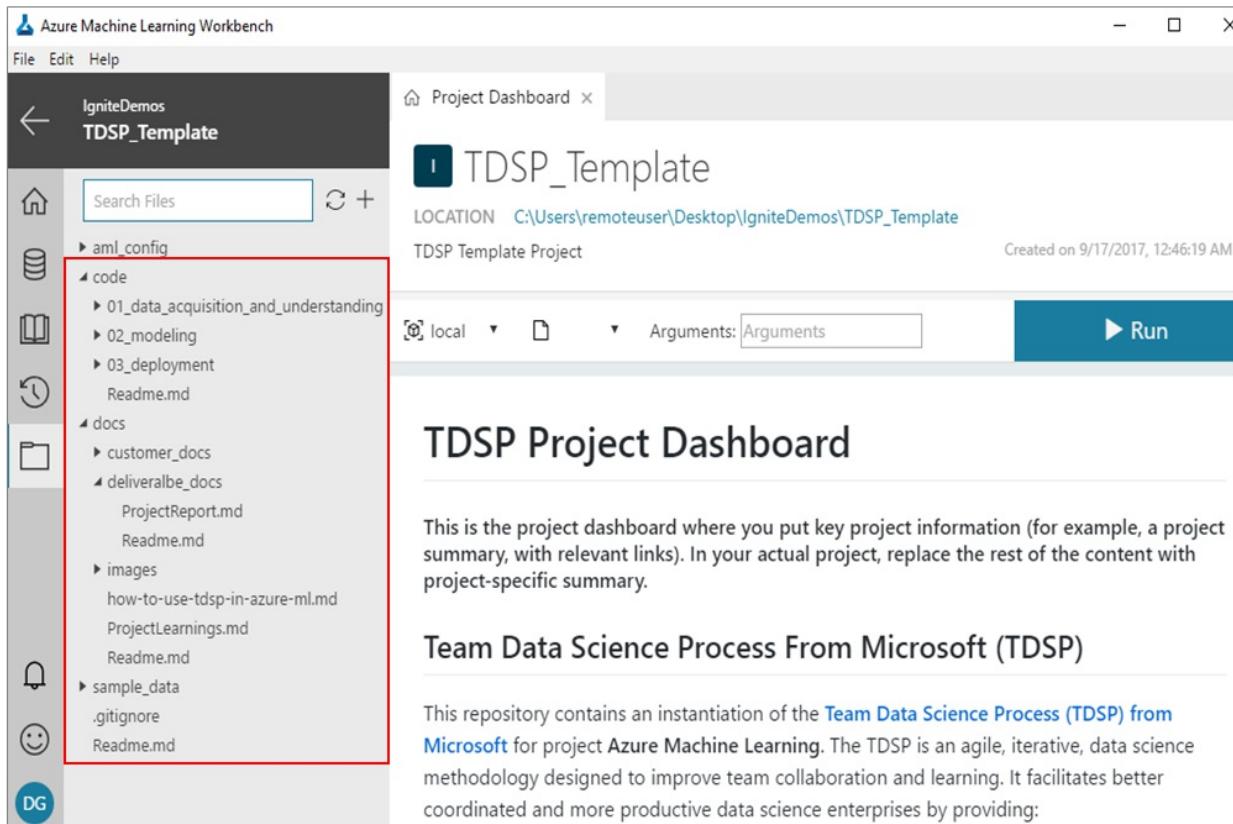
After your new project has been created, you can examine its structure (see the left panel in the following figure). It contains all the aspects of standardized documentation for business understanding. These items include the stages of the TDSP lifecycle, data location, definitions, and the architecture of this documentation template.

The structure shown is derived from the TDSP structure that is published in [TDSP project structure, documents, and artifact templates](#), with some modifications. For example, several of the document templates are merged into one markdown, namely, [ProjectReport](#).

Project folder structure

The TDSP project template contains the following top-level folders:

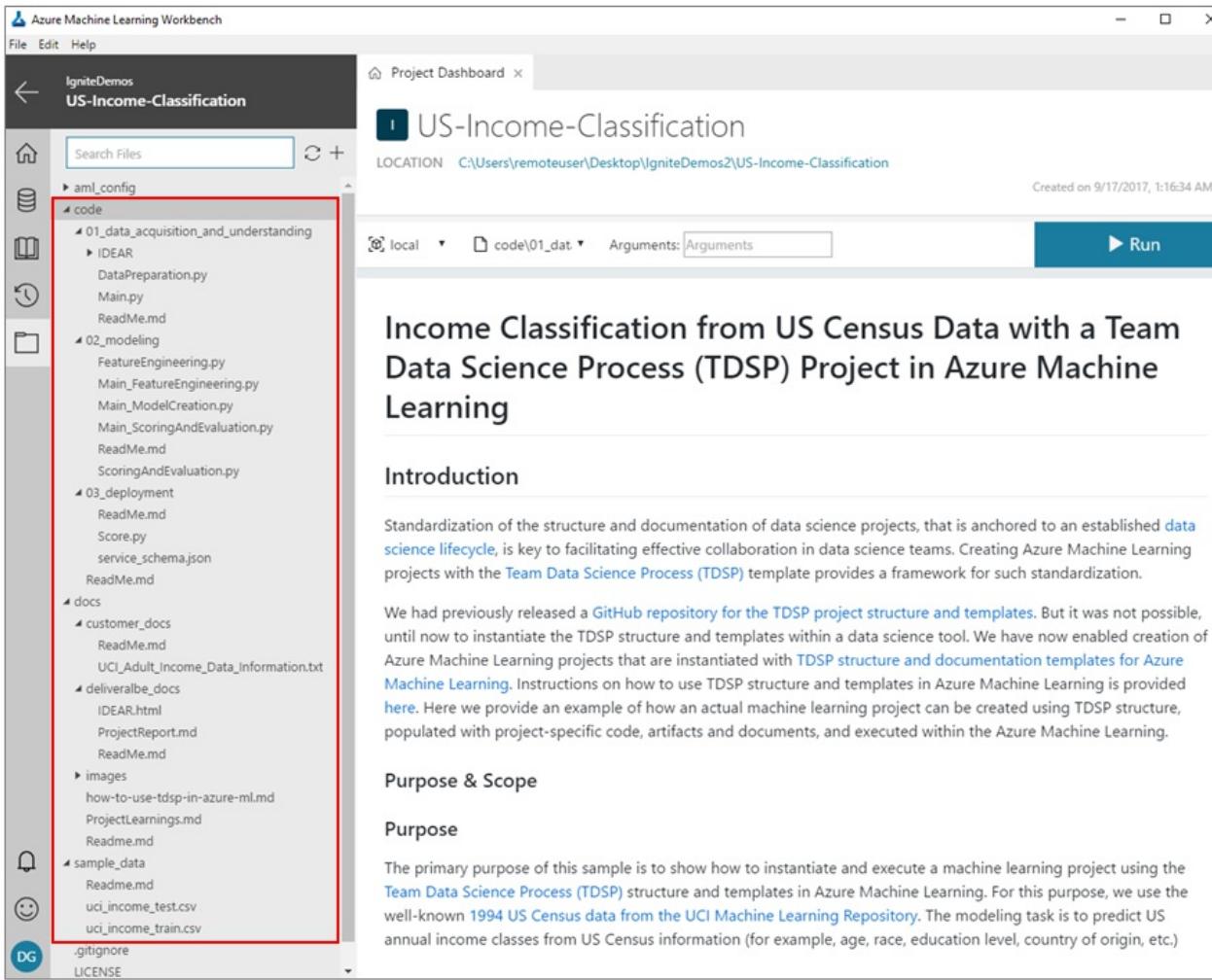
- **code**: Contains code.
- **docs**: Contains necessary documentation about the project (for example, markdown files and related media).
- **sample_data**: Contains **SAMPLE (small)** data that you can use for early development or testing. Typically, these sets are not larger than several (5) MB. This folder is not for full or large data sets.



Use the TDSP structure and templates

You need to add project-specific information to the structure and templates. You're expected to populate these with the code and the information necessary to execute and deliver your project. The [ProjectReport](#) file is a template that you need to modify with information relevant to your project. It comes with a set of questions that help you fill out the information for each of the four stages of the [TDSP lifecycle](#).

An example of what a project structure looks like during execution or after completion is shown in the left panel in the following figure. This project is from the [Team Data Science Process sample project: Classify incomes from US census data in Azure Machine Learning](#) sample project.



Document your project

Refer to the [TDSP documentation templates](#) for information on how to document your project. In the current Machine Learning TDSP documentation template, we recommend that you include all the information in the [ProjectReport](#) file. This template should be filled out with information that is specific to your project.

We also provide a [ProjectLearnings](#) template. You can use this template to include any information that is not included in the primary project document, but that is still useful to document.

Example project report

You can get an [example project report](#). This project report for the [US income classification sample project](#) shows how to instantiate and use the TDSP template for a data science project.

Next steps

To facilitate your understanding on how to use the TDSP structure and templates in Machine Learning projects, we provide several completed project examples in the documentation for Machine Learning:

- For a sample that shows how to create a TDSP project in Machine Learning, see [Team Data Science Process sample project: Classify incomes from US Census data in Azure Machine Learning](#).
- For a sample that uses deep learning in natural language processing (NLP) in a TDSP-instantiated project in Machine Learning, see [Bio-medical entity recognition using natural language processing with deep learning](#).

Set up data science environments for use in the Team Data Science Process

11/30/2017 • 1 min to read • [Edit Online](#)

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](#).

About Azure storage accounts

12/18/2017 • 10 min to read • [Edit Online](#)

TIP

The content in this article applies to the original Azure Table storage. However, there is now a premium offering for table storage, the Azure Cosmos DB Table API that offers throughput-optimized tables, global distribution, and automatic secondary indexes. To learn more and try out the premium experience, please check out [Azure Cosmos DB Table API](#).

Overview

An Azure storage account provides a unique namespace to store and access your Azure Storage data objects. All objects in a storage account are billed together as a group. By default, the data in your account is available only to you, the account owner.

There are two types of storage accounts:

General-purpose Storage Accounts

A general-purpose storage account gives you access to Azure Storage services such as Tables, Queues, Files, Blobs and Azure virtual machine disks under a single account. This type of storage account has two performance tiers:

- A standard storage performance tier which allows you to store Tables, Queues, Files, Blobs and Azure virtual machine disks.
- A premium storage performance tier which currently only supports Azure virtual machine disks. See [Premium Storage: High-Performance Storage for Azure Virtual Machine Workloads](#) for an in-depth overview of Premium storage.

Blob Storage Accounts

A Blob storage account is a specialized storage account for storing your unstructured data as blobs (objects) in Azure Storage. Blob storage accounts are similar to your existing general-purpose storage accounts and share all the great durability, availability, scalability, and performance features that you use today including 100% API consistency for block blobs and append blobs. For applications requiring only block or append blob storage, we recommend using Blob storage accounts.

NOTE

Blob storage accounts support only block and append blobs, and not page blobs.

Blob storage accounts expose the **Access Tier** attribute which can be specified during account creation and modified later as needed. There are two types of access tiers that can be specified based on your data access pattern:

- A **Hot** access tier which indicates that the objects in the storage account will be more frequently accessed. This allows you to store data at a lower access cost.
- A **Cool** access tier which indicates that the objects in the storage account will be less frequently accessed. This allows you to store data at a lower data storage cost.

If there is a change in the usage pattern of your data, you can also switch between these access tiers at any time. Changing the access tier may result in additional charges. Please see [Pricing and billing for Blob storage accounts](#) for more details.

For more details on Blob storage accounts, see [Azure Blob Storage: Cool and Hot tiers](#).

Before you can create a storage account, you must have an Azure subscription, which is a plan that gives you access to a variety of Azure services. You can get started with Azure with a [free account](#). Once you decide to purchase a subscription plan, you can choose from a variety of [purchase options](#). If you're an [MSDN subscriber](#), you get free monthly credits that you can use with Azure services, including Azure Storage. See [Azure Storage Pricing](#) for information on volume pricing.

To learn how to create a storage account, see [Create a storage account](#) for more details. You can create up to 200 uniquely named storage accounts with a single subscription. See [Azure Storage Scalability and Performance Targets](#) for details about storage account limits.

Storage account billing

You are billed for Azure Storage usage based on your storage account. Storage costs are based on the following factors: region/location, account type, storage capacity, replication scheme, storage transactions, and data egress.

- Region refers to the geographical region in which your account is based.
- Account type refers to whether you are using a general-purpose storage account or a Blob storage account. With a Blob storage account, the access tier also determines the billing model for the account.
- Storage capacity refers to how much of your storage account allotment you are using to store data.
- Replication determines how many copies of your data are maintained at one time, and in what locations.
- Transactions refer to all read and write operations to Azure Storage.
- Data egress refers to data transferred out of an Azure region. When the data in your storage account is accessed by an application that is not running in the same region, you are charged for data egress. (For Azure services, you can take steps to group your data and services in the same data centers to reduce or eliminate data egress charges.)

The [Azure Storage Pricing](#) page provides detailed pricing information based on account type, storage capacity, replication, and transactions. The [Data Transfers Pricing Details](#) provides detailed pricing information for data egress. You can use the [Azure Storage Pricing Calculator](#) to help estimate your costs.

NOTE

When you create an Azure virtual machine, a storage account is created for you automatically in the deployment location if you do not already have a storage account in that location. So it's not necessary to follow the steps below to create a storage account for your virtual machine disks. The storage account name will be based on the virtual machine name. See the [Azure Virtual Machines documentation](#) for more details.

Storage account endpoints

Every object that you store in Azure Storage has a unique URL address. The storage account name forms the subdomain of that address. The combination of subdomain and domain name, which is specific to each service, forms an *endpoint* for your storage account.

For example, if your storage account is named *mystorageaccount*, then the default endpoints for your storage account are:

- Blob service: <http://mystorageaccount.blob.core.windows.net>
- Table service: <http://mystorageaccount.table.core.windows.net>
- Queue service: <http://mystorageaccount.queue.core.windows.net>
- File service: <http://mystorageaccount.file.core.windows.net>

NOTE

A Blob storage account only exposes the Blob service endpoint.

The URL for accessing an object in a storage account is built by appending the object's location in the storage account to the endpoint. For example, a blob address might have this format:

`http://mystorageaccount.blob.core.windows.net/mycontainer/myblob`.

You can also configure a custom domain name to use with your storage account. For more information, see [Configure a custom domain Name for your Blob Storage Endpoint](#). You can also configure it with PowerShell. For more information, see the [Set-AzureRmStorageAccount](#) cmdlet.

Create a storage account

1. Sign in to the [Azure portal](#).
2. In the Azure portal, expand the menu on the left side to open the menu of services, and choose **More Services**. Then, scroll down to **Storage**, and choose **Storage accounts**. On the **Storage Accounts** window that appears, choose **Add**.
3. Enter a name for your storage account. See [Storage account endpoints](#) for details about how the storage account name will be used to address your objects in Azure Storage.

NOTE

Storage account names must be between 3 and 24 characters in length and may contain numbers and lowercase letters only.

Your storage account name must be unique within Azure. The Azure portal will indicate if the storage account name you select is already in use.

4. Specify the deployment model to be used: **Resource Manager** or **Classic**. **Resource Manager** is the recommended deployment model. For more information, see [Understanding Resource Manager deployment and classic deployment](#).

NOTE

Blob storage accounts can only be created using the Resource Manager deployment model.

5. Select the type of storage account: **General purpose** or **Blob storage**. **General purpose** is the default.

If **General purpose** was selected, then specify the performance tier: **Standard** or **Premium**. The default is **Standard**. For more details on standard and premium storage accounts, see [Introduction to Microsoft Azure Storage](#) and [Premium Storage: High-Performance Storage for Azure Virtual Machine Workloads](#).

If **Blob Storage** was selected, then specify the access tier: **Hot** or **Cool**. The default is **Hot**. See [Azure Blob Storage: Cool and Hot tiers](#) for more details.

6. Select the replication option for the storage account: **LRS**, **GRS**, **RA-GRS**, or **ZRS**. The default is **RA-GRS**. For more details on Azure Storage replication options, see [Azure Storage replication](#).
7. Select the subscription in which you want to create the new storage account.
8. Specify a new resource group or select an existing resource group. For more information on resource groups, see [Azure Resource Manager overview](#).
9. Select the geographic location for your storage account. See [Azure Regions](#) for more information about what services are available in which region.

10. Click **Create** to create the storage account.

Manage your storage account

Change your account configuration

After you create your storage account, you can modify its configuration, such as changing the replication option used for the account or changing the access tier for a Blob storage account. In the [Azure portal](#), navigate to your storage account, find and click **Configuration** under **SETTINGS** to view and/or change the account configuration.

NOTE

Depending on the performance tier you chose when creating the storage account, some replication options may not be available.

Changing the replication option will change your pricing. For more details, see [Azure Storage Pricing](#) page.

For Blob storage accounts, changing the access tier may incur charges for the change in addition to changing your pricing. Please see the [Blob storage accounts - Pricing and Billing](#) for more details.

Manage your storage access keys

When you create a storage account, Azure generates two 512-bit storage access keys, which are used for authentication when the storage account is accessed. By providing two storage access keys, Azure enables you to regenerate the keys with no interruption to your storage service or access to that service.

NOTE

We recommend that you avoid sharing your storage access keys with anyone else. To permit access to storage resources without giving out your access keys, you can use a *shared access signature*. A shared access signature provides access to a resource in your account for an interval that you define and with the permissions that you specify. See [Using Shared Access Signatures \(SAS\)](#) for more information.

View and copy storage access keys

In the [Azure portal](#), navigate to your storage account, click **All settings** and then click **Access keys** to view, copy, and regenerate your account access keys. The **Access Keys** blade also includes pre-configured connection strings using your primary and secondary keys that you can copy to use in your applications.

Regenerate storage access keys

We recommend that you change the access keys to your storage account periodically to help keep your storage connections secure. Two access keys are assigned so that you can maintain connections to the storage account by using one access key while you regenerate the other access key.

WARNING

Regenerating your access keys can affect services in Azure as well as your own applications that are dependent on the storage account. All clients that use the access key to access the storage account must be updated to use the new key.

Media services - If you have media services that are dependent on your storage account, you must re-sync the access keys with your media service after you regenerate the keys.

Applications - If you have web applications or cloud services that use the storage account, you will lose the connections if you regenerate keys, unless you roll your keys.

Storage Explorers - If you are using any [storage explorer applications](#), you will probably need to update the storage key used by those applications.

Here is the process for rotating your storage access keys:

1. Update the connection strings in your application code to reference the secondary access key of the storage account.
2. Regenerate the primary access key for your storage account. On the **Access Keys** blade, click **Regenerate Key1**, and then click **Yes** to confirm that you want to generate a new key.
3. Update the connection strings in your code to reference the new primary access key.
4. Regenerate the secondary access key in the same manner.

Delete a storage account

To remove a storage account that you are no longer using, navigate to the storage account in the [Azure portal](#), and click **Delete**. Deleting a storage account deletes the entire account, including all data in the account.

WARNING

It's not possible to restore a deleted storage account or retrieve any of the content that it contained before deletion. Be sure to back up anything you want to save before you delete the account. This also holds true for any resources in the account—once you delete a blob, table, queue, or file, it is permanently deleted.

If you try to delete a storage account associated with an Azure virtual machine, you may get an error about the storage account still being in use. For help troubleshooting this error, please see [Troubleshoot errors when you delete storage accounts](#).

Next steps

- [Microsoft Azure Storage Explorer](#) is a free, standalone app from Microsoft that enables you to work visually with Azure Storage data on Windows, macOS, and Linux.
- [Azure Blob Storage: Cool and Hot tiers](#)
- [Azure Storage replication](#)
- [Configure Azure Storage Connection Strings](#)
- [Transfer data with the AzCopy Command-Line Utility](#)
- Visit the [Azure Storage Team Blog](#).

Platforms and tools for data science team projects

11/2/2017 • 9 min to read • [Edit Online](#)

Microsoft provides a full spectrum of data and analytics services and resources for both cloud or on-premise platforms. They can be deployed to make the execution of your data science projects efficient and scalable. Guidance for teams implementing data science projects in a trackable, version controlled, and collaborative way is provided by the [Team Data Science Process](#) (TDSP). For an outline of the personnel roles, and their associated tasks that are handled by a data science team standardizing on this process, see [Team Data Science Process roles and tasks](#).

The data and analytics services available to data science teams using the TDSP include:

- Data Science Virtual Machines (both Windows and Linux CentOS)
- HDInsight Spark Clusters
- SQL Data Warehouse
- Azure Data Lake
- HDInsight Hive Clusters
- Azure File Storage
- SQL Server 2016 R Services

In this document, we briefly describe the resources and provide links to the tutorials and walkthroughs the TDSP teams have published. They can help you learn how to use them step by step and start using them to build your intelligent applications. More information on these resources is available on their product pages.

Data Science Virtual Machine (DSVM)

The data science virtual machine offered on both Windows and Linux by Microsoft, contains popular tools for data science modeling and development activities. It includes tools such as:

- Microsoft R Server Developer Edition
- Anaconda Python distribution
- Jupyter notebooks for Python and R
- Visual Studio Community Edition with Python and R Tools on Windows / Eclipse on Linux
- Power BI desktop for Windows
- SQL Server 2016 Developer Edition on Windows / Postgres on Linux

It also includes **ML and AI tools** like CNTK (an Open Source Deep Learning toolkit from Microsoft), xgboost, mxnet and Vowpal Wabbit.

Currently DSVM is available in **Windows** and **Linux CentOS** operating systems. Choose the size of your DSVM (number of CPU cores and the amount of memory) based on the needs of the data science projects that you are planning to execute on it.

For more information on Windows edition of DSVM, see [Microsoft Data Science Virtual Machine](#) on the Azure marketplace. For the Linux edition of the DSVM, see [Linux Data Science Virtual Machine](#).

To learn how to execute some of the common data science tasks on the DSVM efficiently, see [Ten things you can do on the Data science Virtual Machine](#)

Azure HDInsight Spark clusters

Apache Spark is an open-source parallel processing framework that supports in-memory processing to boost the performance of big-data analytic applications. The Spark processing engine is built for speed, ease of use, and sophisticated analytics. Spark's in-memory computation capabilities make it a good choice for iterative algorithms in machine learning and for graph computations. Spark is also compatible with Azure Blob storage (WASB), so your existing data stored in Azure can easily be processed using Spark.

When you create a Spark cluster in HDInsight, you create Azure compute resources with Spark installed and configured. It takes about 10 minutes to create a Spark cluster in HDInsight. Store the data to be processed in Azure Blob storage. For information on using Azure Blob Storage with a cluster, see [Use HDFS-compatible Azure Blob storage with Hadoop in HDInsight](#).

TDSP team from Microsoft has published two end-to-end walkthroughs on how to use Azure HDInsight Spark Clusters to build data science solutions, one using Python and the other Scala. For more information on Azure HDInsight **Spark Clusters**, see [Overview: Apache Spark on HDInsight Linux](#). To learn how to build a data science solution using **Python** on an Azure HDInsight Spark Cluster, see [Overview of Data Science using Spark on Azure HDInsight](#). To learn how to build a data science solution using **Scala** on an Azure HDInsight Spark Cluster, see [Data Science using Scala and Spark on Azure](#).

Azure SQL Data Warehouse

Azure SQL Data Warehouse allows you to scale compute resources easily and in seconds, without over-provisioning or over-paying. It also offers the unique option to pause the use of compute resources, giving you the freedom to better manage your cloud costs. The ability to deploy scalable compute resources makes it possible to bring all your data into Azure SQL Data Warehouse. Storage costs are minimal and you can run compute only on the parts of datasets that you want to analyze.

For more information on Azure SQL Data Warehouse, see the [SQL Data Warehouse](#) website. To learn how to build end-to-end advanced analytics solutions with SQL Data Warehouse, see [The Team Data Science Process in action: using SQL Data Warehouse](#).

Azure Data Lake

Azure data lake is as an enterprise-wide repository of every type of data collected in a single location, prior to any formal requirements or schema being imposed. This flexibility allows every type of data to be kept in a data lake, regardless of its size or structure or how fast it is ingested. Organizations can then use Hadoop or advanced analytics to find patterns in these data lakes. Data lakes can also serve as a repository for lower-cost data preparation before curating the data and moving it into a data warehouse.

For more information on Azure Data Lake, see [Introducing Azure Data Lake](#). To learn how to build a scalable end-to-end data science solution with Azure Data Lake, see [Scalable Data Science in Azure Data Lake: An end-to-end Walkthrough](#)

Azure HDInsight Hive (Hadoop) clusters

Apache Hive is a data warehouse system for Hadoop, which enables data summarization, querying, and the analysis of data using HiveQL, a query language similar to SQL. Hive can be used to interactively explore your data or to create reusable batch processing jobs.

Hive allows you to project structure on largely unstructured data. After you define the structure, you can use Hive to query that data in a Hadoop cluster without having to use, or even know, Java or MapReduce. HiveQL (the Hive query language) allows you to write queries with statements that are similar to T-SQL.

For data scientists, Hive can run Python User-Defined Functions (UDFs) in Hive queries to process records. This ability extends the capability of Hive queries in data analysis considerably. Specifically, it allows data scientists to conduct scalable feature engineering in languages they are mostly familiar with: the SQL-like HiveQL and Python.

For more information on Azure HDInsight Hive Clusters, see [Use Hive and HiveQL with Hadoop in HDInsight](#). To learn how to build a scalable end-to-end data science solution with Azure HDInsight Hive Clusters, see [The Team Data Science Process in action: using HDInsight Hadoop clusters](#).

Azure File Storage

Azure File Storage is a service that offers file shares in the cloud using the standard Server Message Block (SMB) Protocol. Both SMB 2.1 and SMB 3.0 are supported. With Azure File storage, you can migrate legacy applications that rely on file shares to Azure quickly and without costly rewrites. Applications running in Azure virtual machines or cloud services or from on-premises clients can mount a file share in the cloud, just as a desktop application mounts a typical SMB share. Any number of application components can then mount and access the File storage share simultaneously.

Especially useful for data science projects is the ability to create an Azure file store as the place to share project data with your project team members. Each of them then has access to the same copy of the data in the Azure file storage. They can also use this file storage to share feature sets generated during the execution of the project. If the project is a client engagement, your clients can create an Azure file storage under their own Azure subscription to share the project data and features with you. In this way, the client has full control of the project data assets. For more information on Azure File Storage, see [Get started with Azure File storage on Windows](#) and [How to use Azure File Storage with Linux](#).

SQL Server 2016 R Services

R Services (In-database) provide a platform for developing and deploying intelligent applications that can uncover new insights. You can use the rich and powerful R language, including the many packages provided by the R community, to create models and generate predictions from your SQL Server data. Because R Services (In-database) integrate the R language with SQL Server, analytics are kept close to the data, which eliminates the costs and security risks associated with moving data.

R Services (In-database) support the open source R language with a comprehensive set of SQL Server tools and technologies. They offer superior performance, security, reliability, and manageability. You can deploy R solutions using convenient and familiar tools. Your production applications can call the R runtime and retrieve predictions and visuals using Transact-SQL. You also use the ScaleR libraries to improve the scale and performance of your R solutions. For more information, see [SQL Server R Services](#)

The TDSP team from Microsoft has published two end-to-end walkthroughs that show how to build data science solutions in SQL Server 2016 R Services: one for R programmers and one for SQL developers. For **R Programmers**, see [Data Science End-to-End Walkthrough](#). For **SQL Developers**, see [In-Database Advanced Analytics for SQL Developers \(Tutorial\)](#).

Appendix: Tools to set up data science projects

Install Git Credential Manager on Windows

If you are following the TDSP on **Windows**, you need to install the **Git Credential Manager (GCM)** to communicate with the Git repositories. To install GCM, you first need to install **Chocolatey**. To install Chocolatey and the GCM, run the following commands in Windows PowerShell as an **Administrator**:

```
iwr https://chocolatey.org/install.ps1 -UseBasicParsing | iex  
choco install git-credential-manager-for-windows -y
```

Install Git on Linux (CentOS) machines

Run the following bash command to install Git on Linux (CentOS) machines:

```
sudo yum install git
```

Generate public SSH key on Linux (CentOS) machines

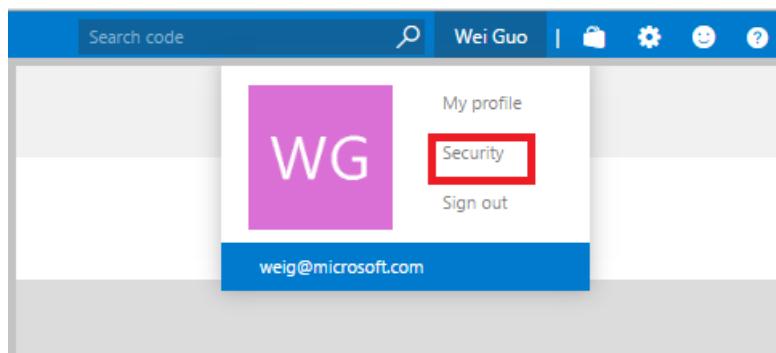
If you are using Linux (CentOS) machines to run the git commands, you need to add the public SSH key of your machine to your VSTS server, so that this machine is recognized by the VSTS server. First, you need to generate a public SSH key and add the key to SSH public keys in your VSTS security setting page.

- To generate the SSH key, run the following two commands:

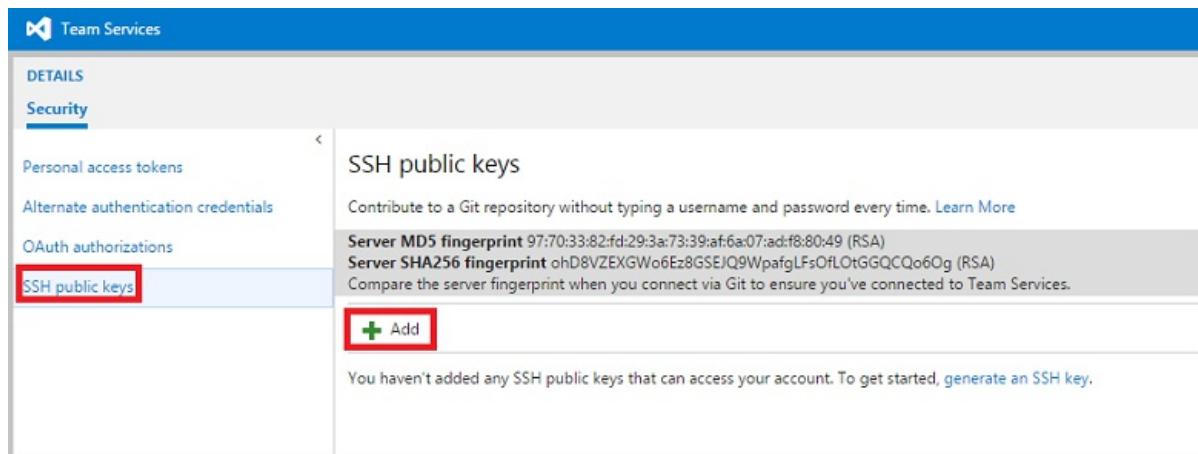
```
ssh-keygen  
cat .ssh/id_rsa.pub
```

```
[ds1@weiglinuxdsvm3 ~]$ ssh-keygen  
Generating public/private rsa key pair.  
Enter file in which to save the key (/home/ds1/.ssh/id_rsa):  
Created directory '/home/ds1/.ssh'.  
Enter passphrase (empty for no passphrase):  
Enter same passphrase again:  
Your identification has been saved in /home/ds1/.ssh/id_rsa.  
Your public key has been saved in /home/ds1/.ssh/id_rsa.pub.  
The key's randomart image is:  
+--[ RSA 2048]----+  
|          . . . . |  
|       . . . . . |  
|      . . . . . . |  
|     . . . . . . . |  
|    . . . . . . . . |  
|   . . . . . . . . . |  
|  . . . . . . . . . . |  
| . . . . . . . . . . . |  
+-----+  
[ds1@weiglinuxdsvm3 ~]$ ls cat .ssh/id_rsa.pub  
ssh-rsa AAAAB3NzC1yc2EAAAQABAAQG5b1xMv5n1wtAhqogn6fGcbqpuPUVkjAf13555Lg2JfpEMU3duhe0x2RBLsFcgralVdzPdtsHNC4/0Y4Jhxz1dKEW2pSLubGcaE2xR1bM0f2Z3N10cf+zb2qAHobt01m9pSPR3euaPrR2t  
[ds1@weiglinuxdsvm3 ~]$ Show ds1@weiglinuxdsvm3  
[ds1@weiglinuxdsvm3 ~]$
```

- Copy the entire ssh key including `ssh-rsa`.
- Log in to your VSTS server.
- Click <Your Name> at the top right corner of the page and click **security**.



- Click **SSH public keys**, and click **+Add**.



- Paste the ssh key just copied into the text box and save.

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 [Example walkthroughs](#) topic. They illustrate how to combine cloud, on-premises tools, and services into a workflow or pipeline to create an intelligent application.

For examples executing steps in the Team Data Science Process that use Azure Machine Learning Studio, see the [With Azure ML](#) learning path.

Introduction to Spark on HDInsight

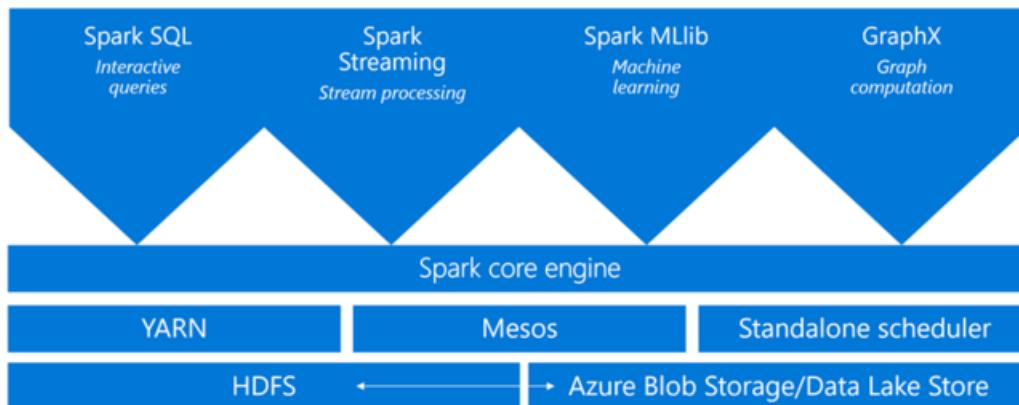
2/15/2018 • 7 min to read • [Edit Online](#)

This article provides you with an introduction to Spark on HDInsight. [Apache Spark](#) is an open-source parallel processing framework that supports in-memory processing to boost the performance of big-data analytic applications. Spark cluster on HDInsight is compatible with Azure Storage (WASB) as well as Azure Data Lake Store. Hence, your existing data stored in Azure can easily be processed via a Spark cluster.

NOTE

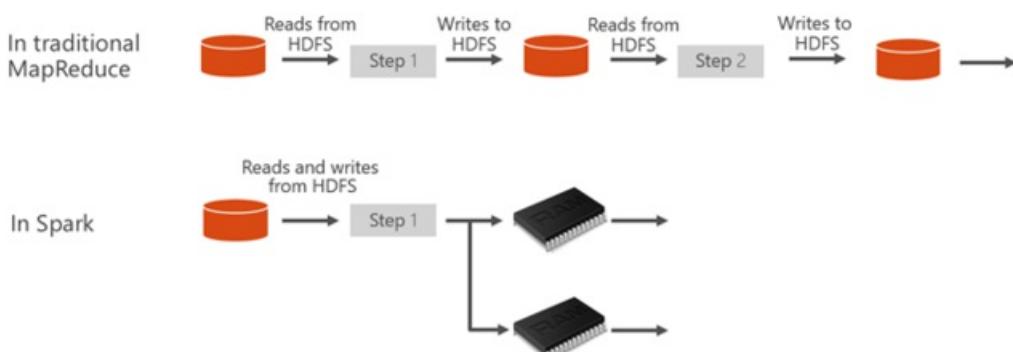
[Learn more about upcoming enhancements and capabilities.](#)

When you create a Spark cluster on HDInsight, you create Azure compute resources with Spark installed and configured. It only takes about 10 minutes to create a Spark cluster in HDInsight. The data to be processed is stored in Azure Storage or Azure Data Lake Store. See [Use Azure Storage with HDInsight](#).



Spark vs. traditional MapReduce

What makes Spark fast? How is the architecture of Apache Spark different than traditional MapReduce, allowing it to offer better performance for data sharing?



Spark provides primitives for in-memory cluster computing. A Spark job can load and cache data into memory and query it repeatedly, much more quickly than disk-based systems. Spark also integrates into the Scala programming language to let you manipulate distributed data sets like local collections. There's no need to structure everything as map and reduce operations.

In Spark, data sharing between operations is faster since data is in-memory. In contrast, Hadoop shares data through HDFS, which takes longer to process.

What is Apache Spark on Azure HDInsight?

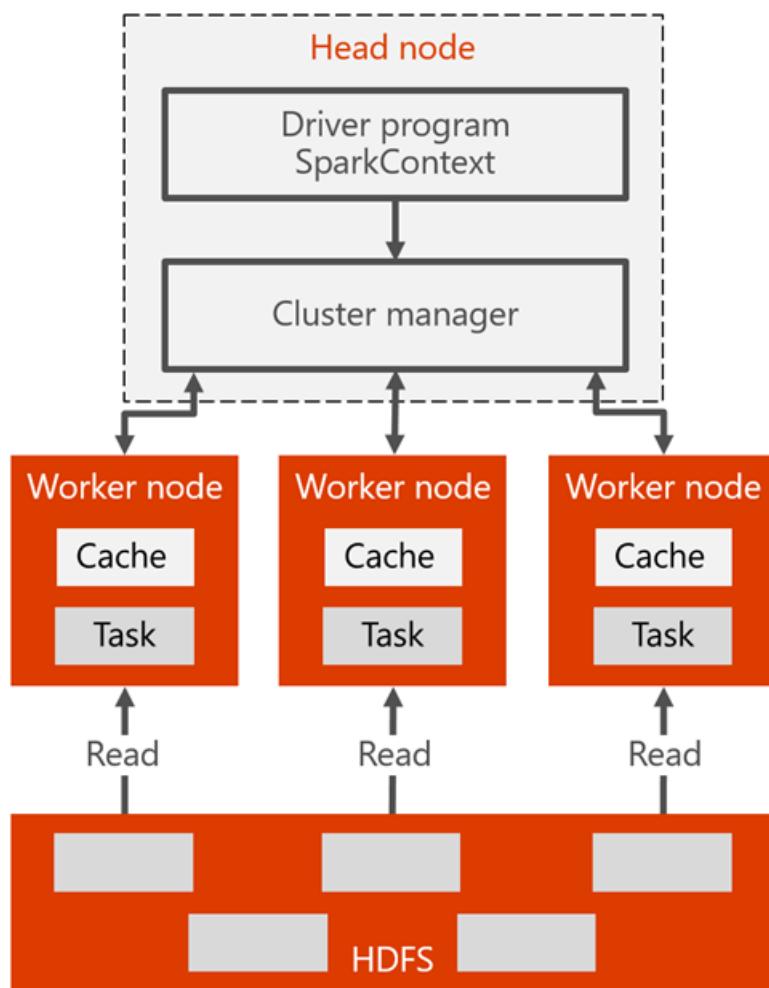
Spark clusters on HDInsight offer a fully managed Spark service. Benefits of creating a Spark cluster on HDInsight are listed here.

FEATURE	DESCRIPTION
Ease of creating Spark clusters	You can create a new Spark cluster on HDInsight in minutes using the Azure portal, Azure PowerShell, or the HDInsight .NET SDK. See Get started with Spark cluster in HDInsight
Ease of use	Spark cluster in HDInsight include Jupyter and Zeppelin notebooks. You can use these notebooks for interactive data processing and visualization.
REST APIs	Spark clusters in HDInsight include Livy , a REST API-based Spark job server to remotely submit and monitor jobs.
Support for Azure Data Lake Store	Spark cluster on HDInsight can be configured to use Azure Data Lake Store as an additional storage, as well as primary storage (only with HDInsight 3.5 clusters). For more information on Data Lake Store, see Overview of Azure Data Lake Store .
Integration with Azure services	Spark cluster on HDInsight comes with a connector to Azure Event Hubs. Customers can build streaming applications using the Event Hubs, in addition to Kafka , which is already available as part of Spark.
Support for R Server	You can set up a R Server on HDInsight Spark cluster to run distributed R computations with the speeds promised with a Spark cluster. For more information, see Get started using R Server on HDInsight .
Integration with third-party IDEs	HDInsight provides plugins for IDEs like IntelliJ IDEA and Eclipse that you can use to create and submit applications to an HDInsight Spark cluster. For more information, see Use Azure Toolkit for IntelliJ IDEA and Use Azure Toolkit for Eclipse .
Concurrent Queries	Spark clusters in HDInsight support concurrent queries. This enables multiple queries from one user or multiple queries from various users and applications to share the same cluster resources.
Caching on SSDs	You can choose to cache data either in memory or in SSDs attached to the cluster nodes. Caching in memory provides the best query performance but could be expensive; caching in SSDs provides a great option for improving query performance without the need to create a cluster of a size that is required to fit the entire dataset in memory.
Integration with BI Tools	Spark clusters on HDInsight provide connectors for BI tools such as Power BI and Tableau for data analytics.
Pre-loaded Anaconda libraries	Spark clusters on HDInsight come with Anaconda libraries pre-installed. Anaconda provides close to 200 libraries for machine learning, data analysis, visualization, etc.

FEATURE	DESCRIPTION
Scalability	Although you can specify the number of nodes in your cluster during creation, you may want to grow or shrink the cluster to match workload. All HDInsight clusters allow you to change the number of nodes in the cluster. Also, Spark clusters can be dropped with no loss of data since all the data is stored in Azure Storage or Data Lake Store.
24/7 Support	Spark clusters on HDInsight come with enterprise-level 24/7 support and an SLA of 99.9% up-time.

Spark cluster architecture

Here is the Spark cluster architecture and how it works:



The head node has the Spark master that manages the number of applications, the apps are mapped to the Spark driver. Every app is managed by Spark master in various ways. Spark can be deployed on top of Mesos, YARN, or the Spark cluster manager, which allocates worker node resources to an application. In HDInsight, Spark runs using the YARN cluster manager. The resources in the cluster are managed by Spark master in HDInsight. That means the Spark master has knowledge of which resources, like memory, are occupied or available on the worker node.

The driver runs the user's main function and executes the various parallel operations on the worker nodes. Then, the driver collects the results of the operations. The worker nodes read and write data from and to the Hadoop distributed file system (HDFS). The worker nodes also cache transformed data in-memory as Resilient Distributed Datasets (RDDs).

Once the app is created in the Spark master, the resources are allocated to the apps by Spark master, creating an execution called the Spark driver. The Spark driver also creates the SparkContext and also starts creating the RDDs. The metadata of the RDDs are stored on the Spark driver.

The Spark driver connects to the Spark master and is responsible for converting an application to a directed graph (DAG) of individual tasks that get executed within an executor process on the worker nodes. Each application gets its own executor processes, which stay up for the duration of the whole application and run tasks in multiple threads.

What are the use cases for Spark on HDInsight?

Spark clusters in HDInsight enable the following key scenarios:

Interactive data analysis and BI

[Look at a tutorial](#)

Apache Spark in HDInsight stores data in Azure Storage or Azure Data Lake Store. Business experts and key decision makers can analyze and build reports over that data and use Microsoft Power BI to build interactive reports from the analyzed data. Analysts can start from unstructured/semi structured data in cluster storage, define a schema for the data using notebooks, and then build data models using Microsoft Power BI. Spark clusters in HDInsight also support a number of third-party BI tools such as Tableau making it an ideal platform for data analysts, business experts, and key decision makers.

Spark Machine Learning

[Look at a tutorial: Predict building temperatures using HVAC data](#)

[Look at a tutorial: Predict food inspection results](#)

Apache Spark comes with [MLlib](#), a machine learning library built on top of Spark that you can use from a Spark cluster in HDInsight. Spark cluster on HDInsight also includes Anaconda, a Python distribution with a variety of packages for machine learning. Couple this with a built-in support for Jupyter and Zeppelin notebooks, and you have a top-of-the-line environment for creating machine learning applications.

Spark streaming and real-time data analysis

Spark clusters in HDInsight offer a rich support for building real-time analytics solutions. While Spark already has connectors to ingest data from many sources like Kafka, Flume, Twitter, ZeroMQ, or TCP sockets, Spark in HDInsight adds first-class support for ingesting data from Azure Event Hubs. Event Hubs is the most widely used queuing service on Azure. Having an out-of-the-box support for Event Hubs makes Spark clusters in HDInsight an ideal platform for building real time analytics pipeline.

What components are included as part of a Spark cluster?

Spark clusters in HDInsight include the following components that are available on the clusters by default.

- [Spark Core](#). Includes Spark Core, Spark SQL, Spark streaming APIs, GraphX, and MLlib.
- [Anaconda](#)
- [Livy](#)
- [Jupyter notebook](#)
- [Zeppelin notebook](#)

Spark clusters on HDInsight also provide an [ODBC driver](#) for connectivity to Spark clusters in HDInsight from BI tools such as Microsoft Power BI and Tableau.

Where do I start?

Start with creating a Spark cluster on HDInsight. See [QuickStart: create a Spark cluster on HDInsight Linux and run interactive query using Jupyter](#).

Next Steps

Scenarios

- [Spark with BI: Perform interactive data analysis using Spark in HDInsight with BI tools](#)
- [Spark with Machine Learning: Use Spark in HDInsight for analyzing building temperature using HVAC data](#)
- [Spark with Machine Learning: Use Spark in HDInsight to predict food inspection results](#)
- [Website log analysis using Spark in HDInsight](#)

Create and run applications

- [Create a standalone application using Scala](#)
- [Run jobs remotely on a Spark cluster using Livy](#)

Tools and extensions

- [Use HDInsight Tools Plugin for IntelliJ IDEA to create and submit Spark Scala applications](#)
- [Use HDInsight Tools Plugin for IntelliJ IDEA to debug Spark applications remotely](#)
- [Use Zeppelin notebooks with a Spark cluster on HDInsight](#)
- [Kernels available for Jupyter notebook in Spark cluster for HDInsight](#)
- [Use external packages with Jupyter notebooks](#)
- [Install Jupyter on your computer and connect to an HDInsight Spark cluster](#)

Manage resources

- [Manage resources for the Apache Spark cluster in Azure HDInsight](#)
- [Track and debug jobs running on an Apache Spark cluster in HDInsight](#)
- [Known issues of Apache Spark in Azure HDInsight.](#)

Create an Apache Spark cluster in Azure HDInsight

3/1/2018 • 4 min to read • [Edit Online](#)

Learn how to create Apache Spark cluster on Azure HDInsight, and how to run Spark SQL queries against Hive tables. For information on Spark on HDInsight, see [Overview: Apache Spark on Azure HDInsight](#).

Prerequisites

- **An Azure subscription.** Before you begin this tutorial, you must have an Azure subscription. See [Create your free Azure account](#).

Create HDInsight Spark cluster

Create an HDInsight Spark cluster using an [Azure Resource Manager template](#). The template can be found in [github](#). For other cluster creation methods, see [Create HDInsight clusters](#).

1. Click the following image to open the template in the Azure portal.

[Deploy to Azure >](#)

2. Enter the following values:

The screenshot shows the configuration page for an HDInsight template named "101-hdinsight-spark-linux". The page is divided into sections: **TEMPLATE**, **BASICS**, **SETTINGS**, and **TERMS AND CONDITIONS**. In the **BASICS** section, fields include "Subscription" (dropdown), "Resource group" (radio buttons: "Create new" selected, "Use existing" unselected), "myresgroup" (selected resource group), "Location" (dropdown: "East US 2"). In the **SETTINGS** section, fields include "Cluster Name" (mysparkcluster), "Cluster Login User Name" (admin), "Cluster Login Password" (redacted), "Ssh User Name" (sshuser), and "Ssh Password" (redacted). The **TERMS AND CONDITIONS** section contains links to "Template information", "Azure Marketplace Terms", and "Azure Marketplace". It also includes a terms and conditions agreement box with the text: "By clicking "Purchase," I (a) agree to the applicable legal terms associated with the offering; (b) authorize Microsoft to charge or bill my current payment method for the fees associated the offering(s), including applicable taxes, with the same billing frequency as my Azure subscription, until I discontinue use of the offering(s); and (c) agree that, if the deployment involves 3rd party offerings, Microsoft may share my contact information and other details of such deployment with the publisher of that offering." Below this is a checkbox for "I agree to the terms and conditions stated above" which is checked. At the bottom are "Pin to dashboard" and a large blue "Purchase" button.

- **Subscription:** Select your Azure subscription used for creating this cluster.
- **Resource group:** Create a resource group or select an existing one. Resource group is used to manage Azure resources for your projects.
- **Location:** Select a location for the resource group. The template uses this location for creating the cluster as well as for the default cluster storage.
- **ClusterName:** Enter a name for the HDInsight cluster that you want to create.
- **Cluster login name and password:** The default login name is admin. Choose a password for the cluster login.
- **SSH user name and password.** Choose a password for the SSH user.

3. Select **I agree to the terms and conditions stated above**, select **Pin to dashboard**, and then click **Purchase**. You can see a new tile titled **Deploying Template deployment**. It takes about 20 minutes to create the cluster.

If you run into an issue with creating HDInsight clusters, it could be that you do not have the right permissions to do so. For more information, see [Access control requirements](#).

NOTE

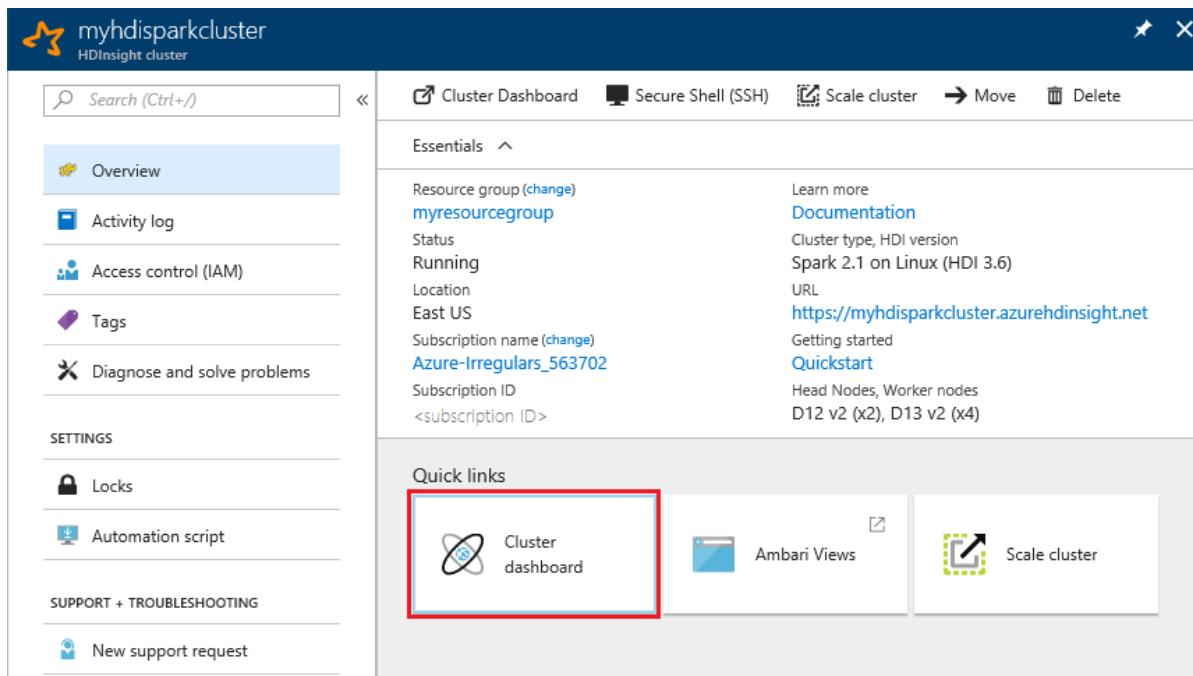
This article creates a Spark cluster that uses [Azure Storage Blobs as the cluster storage](#). You can also create a Spark cluster that uses [Azure Data Lake Store](#) as the default storage. For instructions, see [Create an HDInsight cluster with Data Lake Store](#).

Create a Jupyter notebook

[Jupyter Notebook](#) is an interactive notebook environment that supports various programming languages that allow you to interact with your data, combine code with markdown text and perform simple visualizations. Spark on HDInsight also includes [Zeppelin Notebook](#). Jupyter Notebook is used in this tutorial.

To create a Jupyter notebook

1. Open the [Azure portal](#).
2. Open the Spark cluster you created. For the instructions, see [List and show clusters](#).
3. From the portal, click **Cluster dashboards**, and then click **Jupyter Notebook**. If prompted, enter the admin credentials for the cluster.

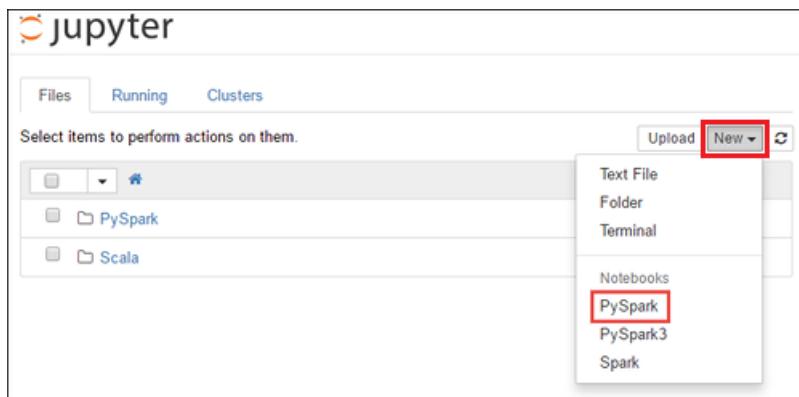


NOTE

You may also access the Jupyter Notebook for your cluster by opening the following URL in your browser. Replace **CLUSTERNAME** with the name of your cluster:

```
https://CLUSTERNAME.azurehdinsight.net/jupyter
```

4. Click **New**, and then click **PySpark** to create a notebook. Jupyter notebooks on HDInsight clusters support three kernels - **PySpark**, **PySpark3**, and **Spark**. The **PySpark** kernel is used in this tutorial. For more information about the kernels, and the benefits of using **PySpark**, see [Use Jupyter notebook kernels with Apache Spark clusters in HDInsight](#).



A new notebook is created and opened with the name Untitled(Untitled.ipynb).

5. Click the notebook name at the top, and enter a friendly name if you want.



Run Spark SQL statements on a Hive table

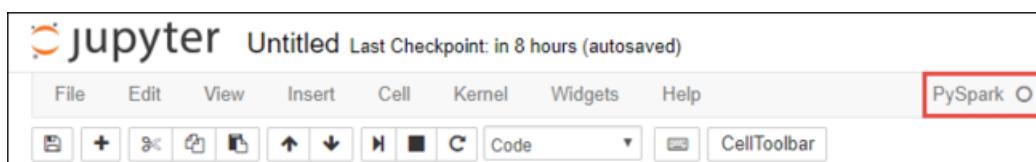
SQL (Structured Query Language) is the most common and widely used language for querying and defining data. Spark SQL functions as an extension to Apache Spark for processing structured data, using the familiar SQL syntax.

Spark SQL supports both SQL and HiveQL as query languages. Its capabilities include binding in Python, Scala, and Java. With it, you can query data stored in many locations, such as external databases, structured data files (example: JSON), and Hive tables.

For an example of reading data from a csv file instead of a Hive table, see [Run interactive queries on Spark clusters in HDInsight](#).

To run Spark SQL

1. When you start the notebook for the first time, the kernel performs some tasks in the background. Wait for the kernel to be ready. The kernel is ready when you see a hollow circle next to the kernel name in the notebook. Solid circle denotes that the kernel is busy.



2. When the kernel is ready, paste the following code in an empty cell, and then press **SHIFT + ENTER** to run the code. The command lists the Hive tables on the cluster:

```
%%sql  
SHOW TABLES
```

When you use a Jupyter Notebook with your HDInsight Spark cluster, you get a preset `sqlContext` that you can use to run Hive queries using Spark SQL. `%%sql` tells Jupyter Notebook to use the preset `sqlContext` to run the Hive query. The query retrieves the top 10 rows from a Hive table (`hivesampletable`) that comes with all HDInsight clusters by default. It takes about 30 seconds to get the results. The output looks like:

```
In [1]: %%sql
SHOW TABLES
Starting Spark application
```

ID	YARN Application ID	Kind	State	Spark UI	Driver log	Current session?
0	application_1516304789330_0004	pyspark	idle	Link	Link	✓

SparkSession available as 'spark'.

```
Out[1]:   database  tableName  isTemporary
          0         default  hivesamplatable  False
```

For more information on the `%%sql` magic and the preset contexts, see [Jupyter kernels available for an HDInsight cluster](#).

Every time you run a query in Jupyter, your web browser window title shows a (**Busy**) status along with the notebook title. You also see a solid circle next to the **PySpark** text in the top-right corner.

- Run another query to see the data in `hivesamplatable`.

```
%%sql
SELECT * FROM hivesamplatable LIMIT 10
```

The screen shall refresh to show the query output.

```
In [1]: %%sql
SELECT * FROM hivesamplatable LIMIT 10
```

Type: [Table](#) [Pie](#) [Scatter](#) [Line](#) [Area](#) [Bar](#)

clientid	querytime	market	deviceplatform	devicemake	devicemodel	state
8	2017-07-18 18:54:20	en-US	Android	Samsung	SCH-i500	California
23	2017-07-18 19:19:44	en-US	Android	HTC	Incredible	Pennsylvania
23	2017-07-18 19:19:46	en-US	Android	HTC	Incredible	Pennsylvania
23	2017-07-18 19:19:47	en-US	Android	HTC	Incredible	Pennsylvania
28	2017-07-18 01:37:50	en-US	Android	Motorola	Droid X	Colorado

- From the **File** menu on the notebook, click **Close and Halt**. Shutting down the notebook releases the cluster resources.
- If you plan to complete the next steps at a later time, make sure you delete the HDInsight cluster you created in this article.

WARNING

Billing for HDInsight clusters is prorated per minute, whether you are using them or not. Be sure to delete your cluster after you have finished using it. For more information, see [How to delete an HDInsight cluster](#).

Next steps

In this article, you learned how to create an HDInsight Spark cluster and run a basic Spark SQL query. Advance to

the next article to learn how to use an HDInsight Spark cluster to run interactive queries on sample data.

[Run interactive queries on an HDInsight Spark cluster](#)

Kernels for Jupyter notebook on Spark clusters in Azure HDInsight

2/23/2018 • 8 min to read • [Edit Online](#)

HDInsight Spark clusters provide kernels that you can use with the Jupyter notebook on Spark for testing your applications. A kernel is a program that runs and interprets your code. The three kernels are:

- **PySpark** - for applications written in Python2
- **PySpark3** - for applications written in Python3
- **Spark** - for applications written in Scala

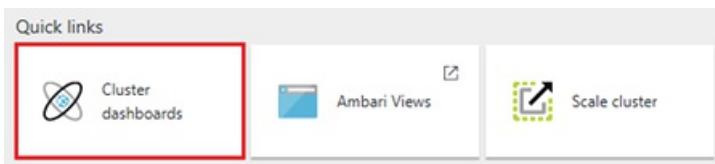
In this article, you learn how to use these kernels and the benefits of using them.

Prerequisites

- An Apache Spark cluster in HDInsight. For instructions, see [Create Apache Spark clusters in Azure HDInsight](#).

Create a Jupyter notebook on Spark HDInsight

1. From the [Azure portal](#), open your cluster. See [List and show clusters](#) for the instructions. The cluster is opened in a new portal blade.
2. From the **Quick links** section, click **Cluster dashboards** to open the **Cluster dashboards** blade. If you don't see **Quick Links**, click **Overview** from the left menu on the blade.



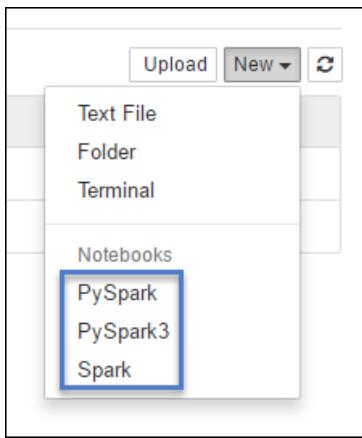
3. Click **Jupyter Notebook**. If prompted, enter the admin credentials for the cluster.

NOTE

You may also reach the Jupyter notebook on Spark cluster by opening the following URL in your browser. Replace **CLUSTERNAME** with the name of your cluster:

```
https://CLUSTERNAME.azurehdinsight.net/jupyter
```

4. Click **New**, and then click either **Pyspark**, **PySpark3**, or **Spark** to create a notebook. Use the Spark kernel for Scala applications, PySpark kernel for Python2 applications, and PySpark3 kernel for Python3 applications.



5. A notebook opens with the kernel you selected.

Benefits of using the kernels

Here are a few benefits of using the new kernels with Jupyter notebook on Spark HDInsight clusters.

- **Preset contexts.** With **PySpark**, **PySpark3**, or the **Spark** kernels, you do not need to set the Spark or Hive contexts explicitly before you start working with your applications. These are available by default. These contexts are:
 - **sc** - for Spark context
 - **sqlContext** - for Hive context

So, you don't have to run statements like the following to set the contexts:

```
sc = SparkContext('yarn-client')
sqlContext = HiveContext(sc)
```

Instead, you can directly use the preset contexts in your application.

- **Cell magics.** The PySpark kernel provides some predefined “magics”, which are special commands that you can call with `%%` (for example, `%%MAGIC`). The magic command must be the first word in a code cell and allow for multiple lines of content. The magic word should be the first word in the cell. Adding anything before the magic, even comments, causes an error. For more information on magics, see [here](#).

The following table lists the different magics available through the kernels.

MAGIC	EXAMPLE	DESCRIPTION
help	<code>%%help</code>	Generates a table of all the available magics with example and description
info	<code>%%info</code>	Outputs session information for the current Livy endpoint

MAGIC	EXAMPLE	DESCRIPTION
configure	<pre>%%configure -f {"executorMemory": "1000M", "executorCores": 4}</pre>	Configures the parameters for creating a session. The force flag (-f) is mandatory if a session has already been created, which ensures that the session is dropped and recreated. Look at Livy's POST /sessions Request Body for a list of valid parameters. Parameters must be passed in as a JSON string and must be on the next line after the magic, as shown in the example column.
sql	<pre>%%sql -o <variable name> SHOW TABLES</pre>	Executes a Hive query against the sqlContext. If the <code>-o</code> parameter is passed, the result of the query is persisted in the %%local Python context as a Pandas dataframe.
local	<pre>%%local a=1</pre>	All the code in subsequent lines is executed locally. Code must be valid Python2 code even irrespective of the kernel you are using. So, even if you selected PySpark3 or Spark kernels while creating the notebook, if you use the <code>%%local</code> magic in a cell, that cell must only have valid Python2 code..
logs	<pre>%%logs</pre>	Outputs the logs for the current Livy session.
delete	<pre>%%delete -f -s <session number></pre>	Deletes a specific session of the current Livy endpoint. You cannot delete the session that is initiated for the kernel itself.
cleanup	<pre>%%cleanup -f</pre>	Deletes all the sessions for the current Livy endpoint, including this notebook's session. The force flag -f is mandatory.

NOTE

In addition to the magics added by the PySpark kernel, you can also use the [built-in IPython magics](#), including `%%sh`. You can use the `%%sh` magic to run scripts and block of code on the cluster headnode.

- **Auto visualization.** The **Pyspark** kernel automatically visualizes the output of Hive and SQL queries. You can choose between several different types of visualizations including Table, Pie, Line, Area, Bar.

Parameters supported with the %%sql magic

The `%%sql` magic supports different parameters that you can use to control the kind of output that you receive when you run queries. The following table lists the output.

PARAMETER	EXAMPLE	DESCRIPTION
-o	-o <VARIABLE NAME>	Use this parameter to persist the result of the query, in the %%local Python context, as a Pandas dataframe. The name of the dataframe variable is the variable name you specify.
-q	-q	Use this to turn off visualizations for the cell. If you don't want to auto-visualize the content of a cell and just want to capture it as a dataframe, then use -q -o <VARIABLE>. If you want to turn off visualizations without capturing the results (for example, for running a SQL query, like a CREATE TABLE statement), use -q without specifying a -o argument.
-m	-m <METHOD>	Where METHOD is either take or sample (default is take). If the method is take , the kernel picks elements from the top of the result data set specified by MAXROWS (described later in this table). If the method is sample , the kernel randomly samples elements of the data set according to -r parameter, described next in this table.
-r	-r <FRACTION>	Here FRACTION is a floating-point number between 0.0 and 1.0. If the sample method for the SQL query is sample , then the kernel randomly samples the specified fraction of the elements of the result set for you. For example, if you run a SQL query with the arguments -m sample -r 0.01, then 1% of the result rows are randomly sampled.
-n	-n <MAXROWS>	MAXROWS is an integer value. The kernel limits the number of output rows to MAXROWS . If MAXROWS is a negative number such as -1, then the number of rows in the result set is not limited.

Example:

```
%%sql -q -m sample -r 0.1 -n 500 -o query2
SELECT * FROM hivesampletable
```

The statement above does the following:

- Selects all records from **hivesampletable**.
- Because we use -q, it turns off auto-visualization.
- Because we use -m sample -r 0.1 -n 500 it randomly samples 10% of the rows in the hivesampletable and limits the size of the result set to 500 rows.

- Finally, because we used `-o query2` it also saves the output into a dataframe called **query2**.

Considerations while using the new kernels

Whichever kernel you use, leaving the notebooks running consumes the cluster resources. With these kernels, because the contexts are preset, simply exiting the notebooks does not kill the context and hence the cluster resources continue to be in use. A good practice is to use the **Close and Halt** option from the notebook's **File** menu when you are finished using the notebook, which kills the context and then exits the notebook.

Show me some examples

When you open a Jupyter notebook, you see two folders available at the root level.

- The **PySpark** folder has sample notebooks that use the new **Python** kernel.
- The **Scala** folder has sample notebooks that use the new **Spark** kernel.

You can open the **00 - [READ ME FIRST] Spark Magic Kernel Features** notebook from the **PySpark** or **Spark** folder to learn about the different magics available. You can also use the other sample notebooks available under the two folders to learn how to achieve different scenarios using Jupyter notebooks with HDInsight Spark clusters.

Where are the notebooks stored?

If your cluster uses Azure Storage as the default storage account, Jupyter notebooks are saved to storage account under the **/HdiNotebooks** folder. Notebooks, text files, and folders that you create from within Jupyter are accessible from the storage account. For example, if you use Jupyter to create a folder **myfolder** and a notebook **myfolder/mynotebook.ipynb**, you can access that notebook at `/HdiNotebooks/myfolder/mynotebook.ipynb` within the storage account. The reverse is also true, that is, if you upload a notebook directly to your storage account at `/HdiNotebooks/mynotebook1.ipynb`, the notebook is visible from Jupyter as well. Notebooks remain in the storage account even after the cluster is deleted.

NOTE

HDInsight clusters with Azure Data Lake Store as the default storage do not store notebooks in associated storage.

The way notebooks are saved to the storage account is compatible with HDFS. So, if you SSH into the cluster you can use file management commands as shown in the following snippet:

```
hdfs dfs -ls /HdiNotebooks          # List everything at the root directory – everything
in this directory is visible to Jupyter from the home page
hdfs dfs -copyToLocal /HdiNotebooks    # Download the contents of the HdiNotebooks folder
hdfs dfs -copyFromLocal example.ipynb /HdiNotebooks  # Upload a notebook example.ipynb to the root folder so
it's visible from Jupyter
```

Irrespective of whether the cluster uses Azure Storage or Azure Data Lake Store as the default storage account, the notebooks are also saved on the cluster headnode at `/var/lib/jupyter`.

Supported browser

Jupyter notebooks on Spark HDInsight clusters are supported only on Google Chrome.

Feedback

The new kernels are in evolving stage and will mature over time. This could also mean that APIs could change as these kernels mature. We would appreciate any feedback that you have while using these new kernels. This is

useful in shaping the final release of these kernels. You can leave your comments/feedback under the **Comments** section at the bottom of this article.

See also

- [Overview: Apache Spark on Azure HDInsight](#)

Scenarios

- [Spark with BI: Perform interactive data analysis using Spark in HDInsight with BI tools](#)
- [Spark with Machine Learning: Use Spark in HDInsight for analyzing building temperature using HVAC data](#)
- [Spark with Machine Learning: Use Spark in HDInsight to predict food inspection results](#)
- [Website log analysis using Spark in HDInsight](#)

Create and run applications

- [Create a standalone application using Scala](#)
- [Run jobs remotely on a Spark cluster using Livy](#)

Tools and extensions

- [Use HDInsight Tools Plugin for IntelliJ IDEA to create and submit Spark Scala applications](#)
- [Use HDInsight Tools Plugin for IntelliJ IDEA to debug Spark applications remotely](#)
- [Use Zeppelin notebooks with a Spark cluster on HDInsight](#)
- [Use external packages with Jupyter notebooks](#)
- [Install Jupyter on your computer and connect to an HDInsight Spark cluster](#)

Manage resources

- [Manage resources for the Apache Spark cluster in Azure HDInsight](#)
- [Track and debug jobs running on an Apache Spark cluster in HDInsight](#)

Introduction to R Server and open-source R capabilities on HDInsight

1/9/2018 • 8 min to read • [Edit Online](#)

Microsoft R Server is available as a deployment option when you create HDInsight clusters in Azure. This new capability provides data scientists, statisticians, and R programmers with on-demand access to scalable, distributed methods of analytics on HDInsight.

NOTE

[Learn more about upcoming enhancements and capabilities.](#)

Clusters can be sized appropriately to the projects and tasks at hand and then torn down when they're no longer needed. Since they're part of Azure HDInsight, these clusters come with enterprise-level 24/7 support, an SLA of 99.9% up-time, and the ability to integrate with other components in the Azure ecosystem.

R Server on HDInsight provides the latest capabilities for R-based analytics on datasets of virtually any size, loaded to either Azure Blob or Data Lake storage. Since R Server is built on open source R, the R-based applications you build can leverage any of the 8000+ open source R packages. The routines in ScaleR, Microsoft's big data analytics package included with R Server, are also available.

The edge node of a cluster provides a convenient place to connect to the cluster and to run your R scripts. With an edge node, you have the option of running the parallelized distributed functions of ScaleR across the cores of the edge node server. You can also run them across the nodes of the cluster by using ScaleR's Hadoop Map Reduce or Spark compute contexts.

The models or predictions that result from analyses can be downloaded for use on-premises. They can also be operationalized elsewhere in Azure, in particular through [Azure Machine Learning Studio web service](#).

Get started with R on HDInsight

To include R Server in an HDInsight cluster, you must select the R Server cluster type when creating an HDInsight cluster using the Azure portal. The R Server cluster type includes R Server on the data nodes of the cluster and on an edge node, which serves as a landing zone for R Server-based analytics. See [Getting Started with R Server on HDInsight](#) for a walkthrough on how to create the cluster.

Learn about data storage options

Default storage for the HDFS file system of HDInsight clusters can be associated with either an Azure Storage account or an Azure Data Lake store. This association ensures that whatever data is uploaded to the cluster storage during analysis is made persistent. There are various tools for handling the data transfer to the storage option that you select, including the portal-based upload facility of the storage account and the [AzCopy](#) utility.

You have the option of adding access to additional Blob and Data lake stores during the cluster provisioning process regardless of the primary storage option in use. See [Getting started with R Server on HDInsight](#) for information on adding access to additional accounts. See the supplementary [Azure Storage options for R Server on HDInsight](#) article to learn more about using multiple storage accounts.

You can also use [Azure Files](#) as a storage option for use on the edge node. Azure Files enables you to mount a file share that was created in Azure Storage to the Linux file system. For more information about these data storage

options for R Server on HDInsight cluster, see [Azure Storage options for R Server on HDInsight clusters](#).

Access R Server on the cluster

You can connect to R Server on the edge node using a browser. It is installed by default during cluster creation. For more information, see [Get started with R Server on HDInsight](#).

You can also connect to the R Server from the commandline by using SSH/PuTTY to access the R console.

Develop and run R scripts

The R scripts you create and run can use any of the 8000+ open source R packages in addition to the parallelized and distributed routines available in the ScaleR library. In general, a script that is run with R Server on the edge node runs within the R interpreter on that node. The exceptions are those steps that need to call a ScaleR function with a compute context that is set to Hadoop Map Reduce (RxHadoopMR) or Spark (RxSpark). In this case, the function runs in a distributed fashion across those data (task) nodes of the cluster that are associated with the data referenced. For more information about the different compute context options, see [Compute context options for R Server on HDInsight](#).

Operationalize a model

When your data modeling is complete, you can operationalize the model to make predictions for new data either from Azure and on-premises. This process is known as scoring. Scoring can be done in HDInsight, Azure Machine Learning, or on-premises.

Score in HDInsight

To score in HDInsight, write an R function that calls your model to make predictions for a new data file that you've loaded to your storage account. Then save the predictions back to the storage account. You can run the routine on-demand on the edge node of your cluster or by using a scheduled job.

Score in Azure Machine Learning (AML)

To score using an AML web service, use the open source Azure Machine Learning R package known as [AzureML](#) to publish your model as an Azure web service. For convenience, this package is pre-installed on the edge node. Next, use the facilities in Machine Learning to create a user interface for the web service, and then call the web service as needed for scoring.

If you choose this option, you need to convert any ScaleR model objects to equivalent open-source model objects for use with the web service. Use ScaleR coercion functions, such as `as.randomForest()` for ensemble-based models, for this conversion.

Score on-premises

To score on-premises after creating your model, you can serialize the model in R, download it, de-serialize it, and then use it for scoring new data. You can score new data by using the approach described earlier in [Scoring in HDInsight](#) or by using [DeployR](#).

Maintain the cluster

Install and maintain R packages

Most of the R packages that you use are required on the edge node since most steps of your R scripts run there. To install additional R packages on the edge node, you can use the usual `install.packages()` method in R.

If you are just using routines from the ScaleR library across the cluster, you do not usually need to install additional R packages on the data nodes. However, you might need additional packages to support the use of **rxExec** or **RxDatapStep** execution on the data nodes.

In these cases, the additional packages can be installed with a script action after you create the cluster. For more information, see [Creating an HDInsight cluster with R Server](#).

Change Hadoop Map Reduce memory settings

A cluster can be modified to change the amount of memory that's available to R Server when it's running a Map Reduce job. To modify a cluster, use the Apache Ambari UI that's available through the Azure portal blade for your cluster. For instructions about how to access the Ambari UI for your cluster, see [Manage HDInsight clusters using the Ambari Web UI](#).

It's also possible to change the amount of memory that's available to R Server by using Hadoop switches in the call to **RxHadoopMR** as follows:

```
hadoopSwitches = "-libjars /etc/hadoop/conf -Dmapred.job.map.memory.mb=6656"
```

Scale your cluster

An existing cluster can be scaled up or down through the portal. By scaling, you can gain the additional capacity that you might need for larger processing tasks, or you can scale back a cluster when it is idle. For instructions about how to scale a cluster, see [Manage HDInsight clusters](#).

Maintain the system

Maintenance to apply OS patches and other updates is performed on the underlying Linux VMs in an HDInsight cluster during off-hours. Typically, maintenance is done at 3:30 AM (based on the local time for the VM) every Monday and Thursday. Updates are performed in such a way that they don't impact more than a quarter of the cluster at a time.

Since the head nodes are redundant and not all data nodes are impacted, any jobs that are running during this time might slow down. They should still run to completion, however. Any custom software or local data that you have is preserved across these maintenance events unless a catastrophic failure occurs that requires a cluster rebuild.

Learn about IDE options for R Server on an HDInsight cluster

The Linux edge node of an HDInsight cluster is the landing zone for R-based analysis. Recent versions of HDInsight provide a default installation of RStudio Server on the edge node as a browser-based IDE. Use of RStudio Server as an IDE for the development and execution of R scripts can be considerably more productive than just using the R console.

Another full IDE option is to install a desktop IDE and use it to access the cluster through use of a remote Map Reduce or Spark compute context. Options include Microsoft's [R Tools for Visual Studio](#) (RTVS), RStudio, and Walware's Eclipse-based [StatET](#).

Lastly, you can access the R Server console on the edge node by typing **R** at the Linux command prompt after connecting via SSH or PuTTY. When using the console interface, it is convenient to run a text editor for R script development in another window, and cut and paste sections of your script into the R console as needed.

Learn about pricing

The fees that are associated with an HDInsight cluster with R Server are structured similarly to the fees for the standard HDInsight clusters. They are based on the sizing of the underlying VMs across the name, data, and edge nodes, with the addition of a core-hour uplift. For more information about HDInsight pricing see [HDInsight pricing](#).

Next steps

To learn more about how to use R Server with HDInsight clusters, see the following topics:

- [Getting started with R Server on HDInsight](#)
- [Compute context options for R Server on HDInsight](#)
- [Azure Storage options for R Server on HDInsight](#)

Get started with R Server on HDInsight

2/23/2018 • 25 min to read • [Edit Online](#)

Azure HDInsight includes an R Server option to be integrated into your HDInsight cluster. This option allows R scripts to use Spark and MapReduce to run distributed computations. In this article, you learn how to create an R Server on HDInsight cluster. You then learn how to run an R script that demonstrates using Spark for distributed R computations.

Prerequisites

- **An Azure subscription:** Before you begin this tutorial, you must have an Azure subscription. For more information, see [Get Microsoft Azure free trial](#).
- **A Secure Shell (SSH) client:** An SSH client is used to remotely connect to the HDInsight cluster and run commands directly on the cluster. For more information, see [Use SSH with HDInsight](#).
- **SSH keys (optional):** You can secure the SSH account that's used to connect to the cluster by using either a password or a public key. Using a password is easier, and it enables you to get started without having to create a public/private key pair. However, using a key is more secure.

NOTE

The steps in this article assume that you are using a password.

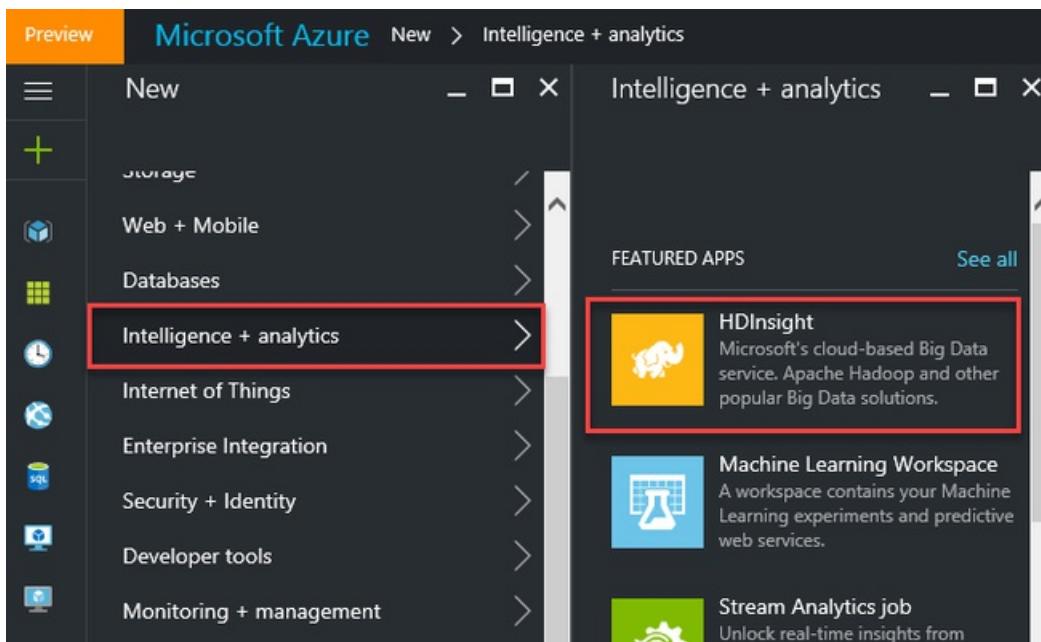
Automate cluster creation

You can automate the creation of HDInsight R Server instances by using Azure Resource Manager templates, the SDK, and PowerShell.

- To create an R Server instance by using an Azure Resource Manager template, see [Deploy an R server HDInsight cluster](#).
- To create an R Server instance by using the .NET SDK, see [Create Linux-based clusters in HDInsight using the .NET SDK](#).
- To deploy R Server by using PowerShell, see [Create Linux-based clusters in HDInsight using Azure PowerShell](#).

Create a cluster by using the Azure portal

1. Sign in to the [Azure portal](#).
2. Select **New > Intelligence + analytics > HDInsight**.



3. In the **Quick create** experience, enter a name for the cluster in the **Cluster name** box. If you have multiple Azure subscriptions, use the **Subscription** box to select the one that you want to use.

HDInsight by Microsoft

Quick create Custom (size, settings, apps)

Basics

* Cluster name
Enter new cluster name .azurehdinsight.net

* Subscription

Storage Set storage settings

Summary Confirm configurations

This cluster may take up to 20 minutes to create.

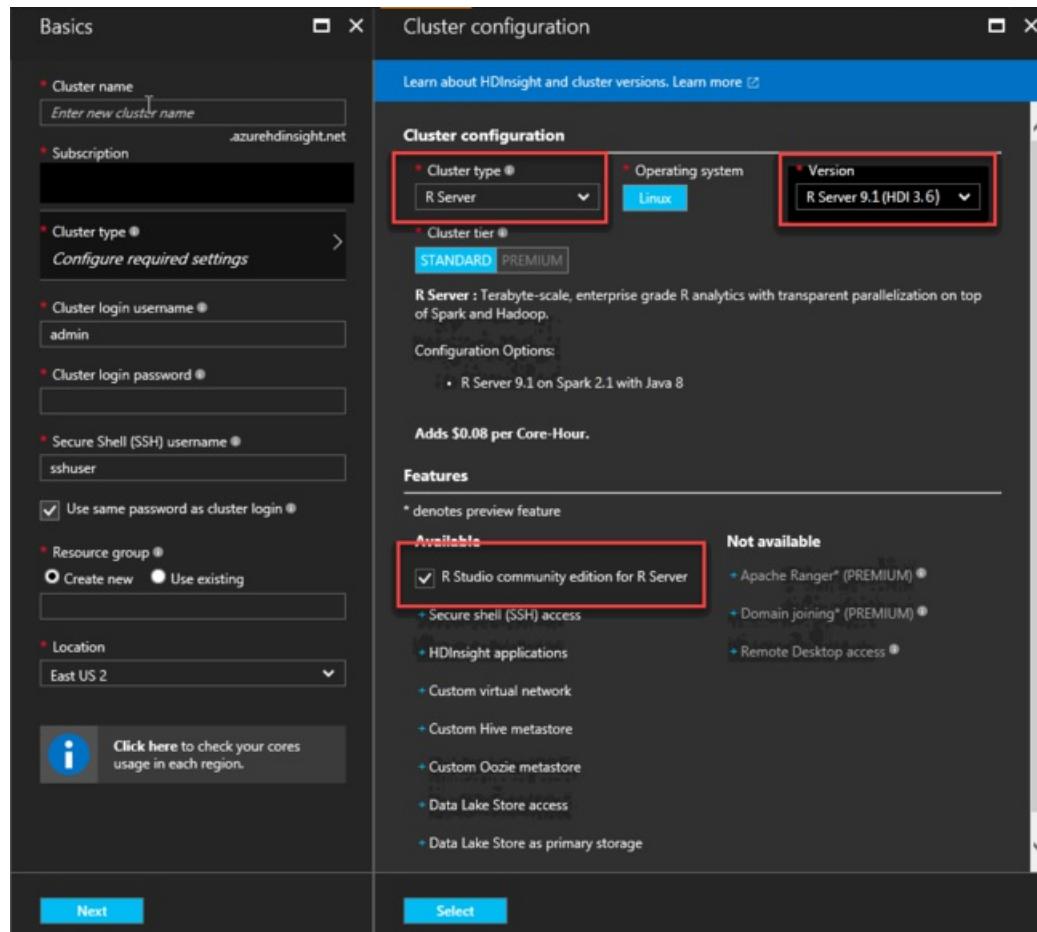
Click here to check your cores usage in each region.

4. Select **Cluster type** to open the **Cluster configuration** pane. In the **Cluster configuration** pane, select the following options:

- **Cluster type:** Select **R Server**.
- **Version:** Select the version of R Server to install on the cluster. The version currently available is **R Server**

9.1 (HDI 3.6). Release notes for the available versions of R Server are available on docs.microsoft.com.

- **R Studio community edition for R Server:** This browser-based IDE is installed by default on the edge node. If you prefer to not have it installed, clear the check box. If you choose to have it installed, the URL for accessing the RStudio Server login will be in a portal application pane for your cluster after it's created.
- Leave the other options at the default values, and use the **Select** button to save the cluster type.



5. In the **Basics** pane, in the **Cluster login username** and **Cluster login password** boxes, enter a username and a password (respectively) for the cluster.
6. In the **Secure Shell (SSH) username** box, specify the SSH username. SSH is used to remotely connect to the cluster by using an SSH client. You can specify the SSH user in this box or after the cluster is created (on the **Configuration** tab for the cluster).

NOTE

R Server is configured to expect an SSH username of "remoteuser." If you use a different username, you must perform an additional step after the cluster is created.

7. Leave the **Use same password as cluster login** check box selected to use **PASSWORD** as the authentication type, unless you prefer to use a public key. You need a public/private key pair to access R Server on the cluster via a remote client, such as R Tools for Visual Studio, RStudio, or another desktop IDE. If you install the RStudio Server community edition, you need to choose an SSH password.

To create and use a public/private key pair, clear **Use same password as cluster login**. Then select **PUBLIC KEY** and proceed as follows. These instructions assume that you have Cygwin with ssh-keygen or an equivalent installed.

- a. Generate a public/private key pair from the command prompt on your laptop:

```
ssh-keygen -t rsa -b 2048
```

- b. Follow the prompt to name a key file and then enter a passphrase for added security. Your screen should look something like the following image:

```
jeffs@furiosa ~
$ ssh-keygen -t rsa -b 2048
Generating public/private rsa key pair.
Enter file in which to save the key (/home/jeffs/.ssh/id_rsa): furiosa
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in furiosa.
Your public key has been saved in furiosa.pub.
The key fingerprint is:
SHA256:ggjBzrchiLmQsX6Q3J0+68whYB+kgN2PRjDtf4T+kJM jeffs@furiosa
The key's randomart image is:
+---[RSA 2048]---+
| o.
| .. +.
| o .oo .
| o oo.+.
| +=+=.=o++S
| *@oo.oE..
| O+.=.. =
| o.o.*o. .
| ....=o
+---[SHA256]---+
jeffs@furiosa ~
$ |
```

This command creates a private key file and a public key file under the name .pub. In this example, the files are furiosa and furiosa.pub:

```
jeffs@furiosa ~
$ dir
furiosa  furiosa.pub
jeffs@furiosa ~
$ |
```

- c. Specify the public key file (*.pub) when assigning HDI cluster credentials. Confirm your resource group and region, and select **Next**.

* Cluster login username ⓘ
admin

* Cluster login password ⓘ

* Secure Shell (SSH) username ⓘ
sshuser

Use same password as cluster login ⓘ

SSH authentication type
PASSWORD PUBLIC KEY

* SSH public key

```
ssh-rsa
AAAAB3NzaC1yc2EAAAQABAA...
QD1ET5or73bwMmprdRyICpaBZx...
ZEez72GAgd0sGevuM/bChiitSLnc...
dY0GuhgOReg6NOgZ/f2XmwtMn70M...
g5fej8W6k4kJZSys+I2NxrB750Uo...
uEcuVjvdX9zpdFeU6ZIAIEA1/+6E...
BeaSr/7
```

furiosa.pub

! RStudio requires a SSH user with a password. If you choose SSH user with public key you will be required to add password later for the SSH user in order to login to RStudio.

d. Change permissions on the private key file on your laptop:

```
chmod 600 <private-key-filename>
```

e. Use the private key file with SSH for remote login:

```
ssh -i <private-key-filename> remoteuser@<hostname public ip>
```

Or, use the private key file as part the definition of your Hadoop Spark compute context for R Server on the client. For more information, see [Create a Compute Context for Spark](#).

8. The quick create transitions you to the **Storage** pane. There, you select the storage account settings to be used for the primary location of the HDFS file system that the cluster uses. Select a new or existing Azure storage account, or select an existing Azure Data Lake Store account.

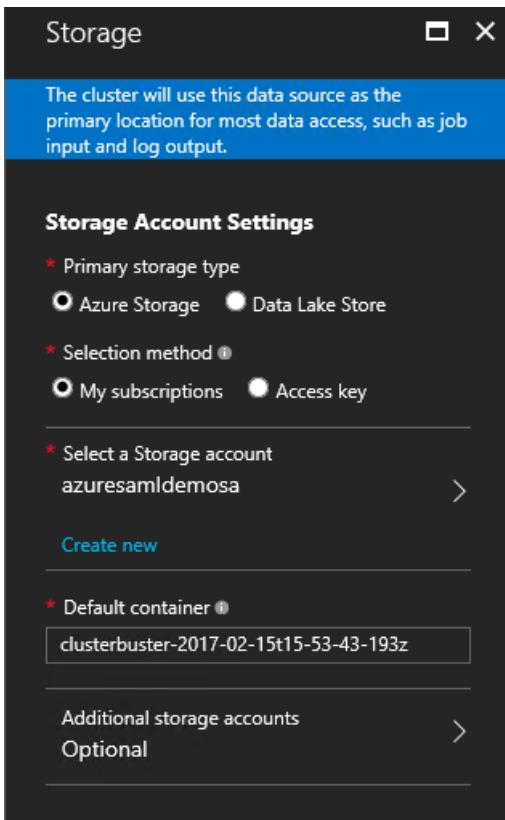
- If you select an Azure storage account, you can choose an existing account by selecting **Select a storage account** and then selecting the relevant account. Create a new account by using the **Create new** link in the **Select a storage account** section.

NOTE

If you select **New**, you must enter a name for the new storage account. A green check appears if the name is accepted.

Default container defaults to the name of the cluster. Leave this default as the value.

If you selected a new storage account, use **Location** in the prompt to select a region for it.



IMPORTANT

Selecting the location for the default data source also sets the location of the HDInsight cluster. The cluster and default data source must be in the same region.

- If you want to use an existing Data Lake Store account, select the account to use. Then add the cluster ADD identity to your cluster to allow access to the store. For more information on this process, see [Create HDInsight clusters with Data Lake Store by using the Azure portal](#).

Use the **Select** button to save the data source configuration.

9. The **Summary** pane then appears so you can validate all your settings. Here you can change your cluster size to modify the number of servers in your cluster. You can also specify any script actions that you want to run. Unless you know that you need a larger cluster, leave the number of worker nodes at the default of **4**. The pane also shows the estimated cost of the cluster.

Cluster summary

Click '**Create**' below to deploy your cluster. Once created you will be billed until your cluster is deleted.

Basics (Edit)

Cluster name	clusterbuster
Subscription	[REDACTED]
Cluster type	R Server 9.0 on Linux (HDI 3.5)
Cluster login username	admin
SSH username	sshuser
Resource group	hdinsight5M3F5ZXYDDJO2HOUJUZXMP3AFHXMDHL2C36B...
Location	East US 2

Storage (Edit)

Azure Storage account	azuresamldemosa
Additional Storage accounts	---
Metastores	---

Applications (optional) (Edit)

Applications (optional)	---
--------------------------------	-----

Cluster size (Edit)

Nodes	Head (2 x D12 v2), Worker (4 x D4 v2), RServer (1 x Standard...)
--------------	--

Advanced settings (Edit)

Script actions	---
Virtual network	---

11.58 USD
USD/HOUR (ESTIMATED)

7 NODES (2 HEAD + 4 WORKER + 1 RSERVER) 48 CORES
R SERVER 9.0 ON LINUX (HDI 3.5) AZURESAMLDEMOSA (EAST US 2)

Create [Download template and parameters](#)

NOTE

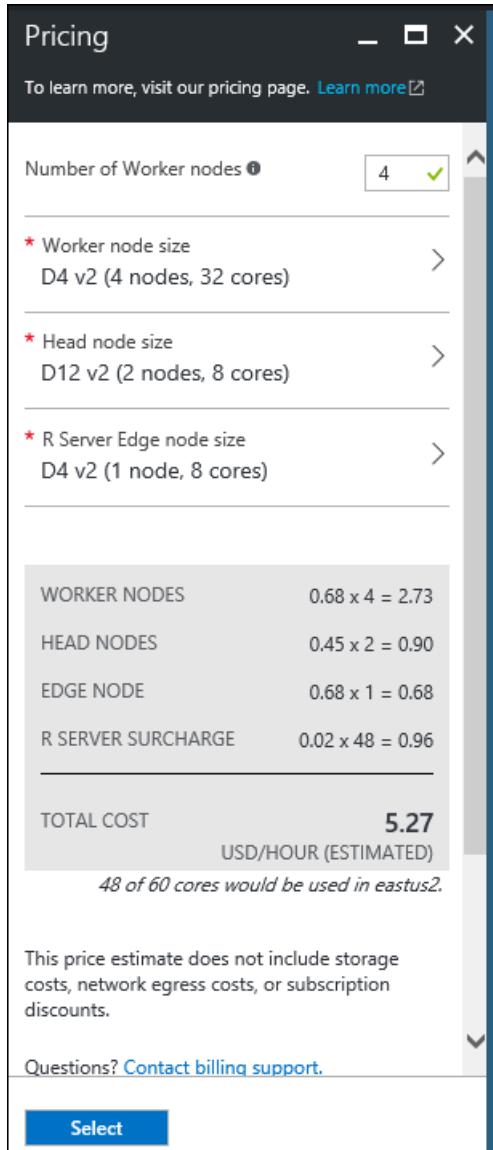
If needed, you can resize your cluster later through the portal (**Cluster** > **Settings** > **Scale cluster**) to increase or decrease the number of worker nodes. This resizing can be useful for idling down the cluster when it's not in use, or for adding capacity to meet the needs of larger tasks.

Factors to keep in mind when sizing your cluster, the data nodes, and the edge node include:

- The performance of distributed R Server analyses on Spark is proportional to the number of worker nodes when the data is large.
- The performance of R Server analyses is linear in the size of data being analyzed. For example:
 - For small to modest data, performance is best when the data is analyzed in a local compute context on the edge node. For more information on the scenarios under which the local and Spark compute contexts work best, see [Compute context options for R Server on HDInsight](#).

- If you log in to the edge node and run your R script, all but the ScaleR rx-functions are executed *locally* on the edge node. So the memory and number of cores on the edge node should be sized accordingly. The same applies if you use R Server on HDI as a remote compute context from your laptop.

Use the **Select** button to save the node pricing configuration.



There is also a link for **Download template and parameters**. Select this link to display scripts that can be used to automate the creation of a cluster with the selected configuration. These scripts are also available from the Azure portal entry for your cluster after it's created.

NOTE

It takes some time for the cluster to be created, usually around 20 minutes. To check on the creation process, use the tile on the Startboard or the **Notifications** entry on the left of the page.

Connect to RStudio Server

If you chose to include RStudio Server community edition in your installation, you can access the RStudio login via two methods:

- Go to the following URL (where *CLUSTERNAMESPACE* is the name of the cluster that you created):

<https://CLUSTERNAMESPACE.azurehdinsight.net/rstudio/>

- Open the entry for your cluster in the Azure portal, select the **R Server dashboards** quick link, and then select **R Studio dashboard**:

Resource group: hdsigp
Cluster type, HDI version: R Server on Linux (HDI 3.5.1000.0)
Status: Running
Location: East US 2

Usage

TYPE	NODE SIZE	CORES	NODES
Head	D12 v2	8	2
Worker	D4 v2	32	4
Zookeeper	A2	6	3
Edge nodes	D4 v2	8	1

Cores in East US 2 for subscription

THIS CLUSTER: 48
OTHER CLUSTERS: 0
TOTAL: 60
AVAILABLE: 12

Cluster dashboard

HDInsight cluster dashboard

Jupyter Notebook

Zeppelin is only available on Spark 2.4.2 and cluster k version 3.5

R Studio server

In the first login prompt, provide your cluster login credentials. In the second login prompt for R Studio user credentials, provide your SSH login credentials.

Spark History Server

Yarn

IMPORTANT

Regardless of the method used, the first time you log in, you need to authenticate twice. At the first authentication, provide the cluster admin user ID and password. At the second prompt, provide the SSH user ID and password. Subsequent logins require only the SSH user ID and password.

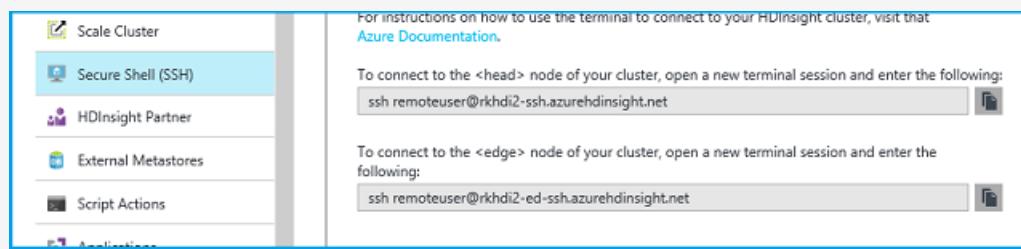
Connect to the R Server edge node

Connect to the R Server edge node of the HDInsight cluster by using SSH via this command:

```
ssh USERNAME@CLUSTERNAME-ed-ssh.azurehdinsight.net
```

NOTE

You can find the `USERNAME@CLUSTERNAME-ed-ssh.azurehdinsight.net` address in the Azure portal. Select your cluster, and then select **All Settings > Apps > RServer**. This displays the SSH endpoint information for the edge node.



If you used a password to help secure your SSH user account, you are prompted to enter it. If you used a public key, you might have to use the `-i` parameter to specify the matching private key. For example:

```
ssh -i ~/.ssh/id_rsa USERNAME@CLUSTERNAME-ed-ssh.azurehdinsight.net
```

For more information, see [Connect to HDInsight \(Hadoop\) using SSH](#).

After you're connected, you arrive at a prompt that's similar to the following:

```
sername@ed00-myrsr:~$
```

Enable multiple concurrent users

You can enable multiple concurrent users by adding more users for the edge node on which the RStudio community version runs.

When you create an HDInsight cluster, you must provide two users: an HTTP user and an SSH user.

Basics

* Cluster name
 .azurehdinsight.net

* Subscription

* Cluster type [i](#) >
Configure required settings

* Cluster login username [i](#)

* Cluster login password [i](#)
 ✓

Secure Shell (SSH) username [i](#)

Use same password as cluster login [i](#)

* Resource group [i](#)
 Create new Use existing

* Location

 Click here to view cores usage.

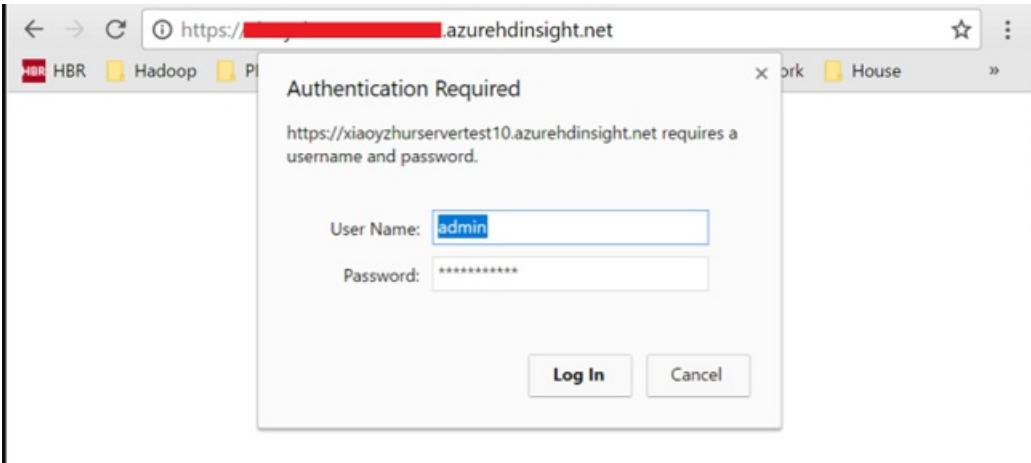
Next

- **Cluster login username:** An HTTP user for authentication through the HDInsight gateway that's used to protect the HDInsight clusters that you created. This HTTP user is used to access the Ambari UI, the YARN UI, and other UI components.
- **Secure Shell (SSH) username:** An SSH user to access the cluster through Secure Shell. This user is a user in the

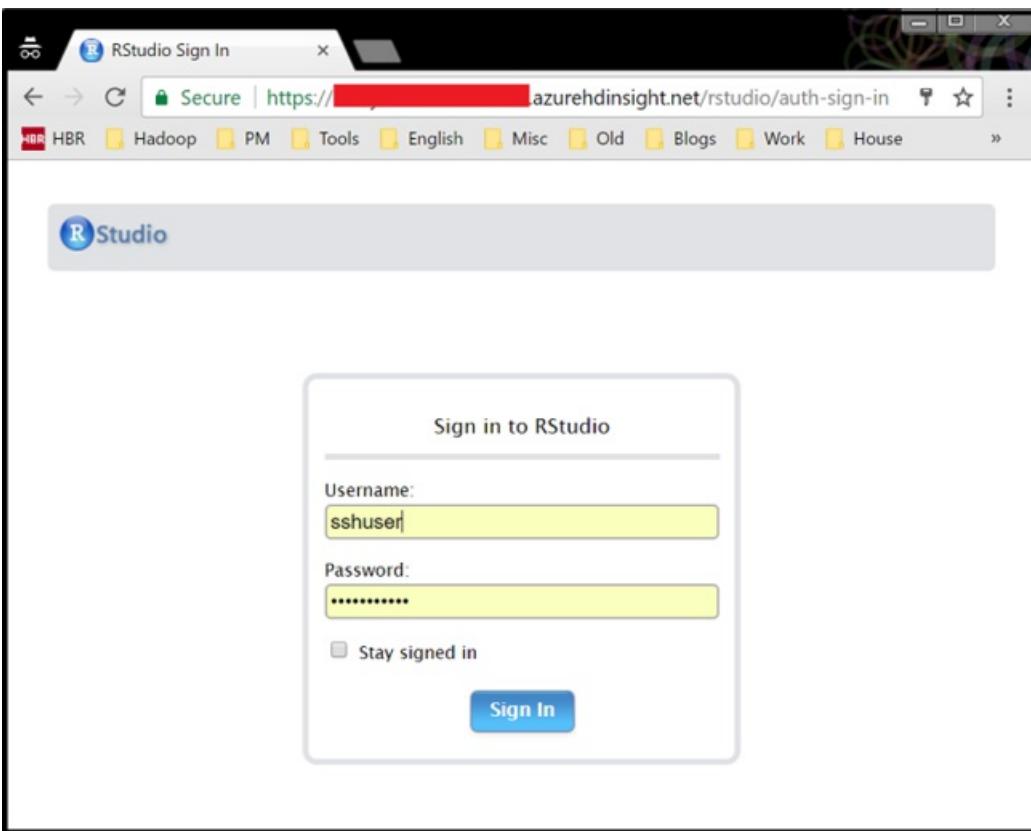
Linux system for all the head nodes, worker nodes, and edge nodes. So you can use SSH to access any of the nodes in a remote cluster.

The RStudio Server community version used in the Microsoft R Server on HDInsight type cluster accepts only the Linux username and password as a login mechanism. It does not support passing tokens. So if you have created a new cluster and want to use RStudio to access it, you need to log in twice.

1. Log in by using the HTTP user credentials through the HDInsight gateway:



2. Use the SSH user credentials to sign in to RStudio:



Currently, you can create only one SSH user account when provisioning an HDInsight cluster. To enable multiple users to access Microsoft R Server on HDInsight clusters, you need to create additional users in the Linux system.

Because RStudio Server community version is running on the cluster's edge node, there are three steps here:

1. Use the created SSH user to log in to the edge node.
2. Add more Linux users to the edge node.
3. Use the RStudio community version with the created user.

Use the created SSH user to log in to the edge node

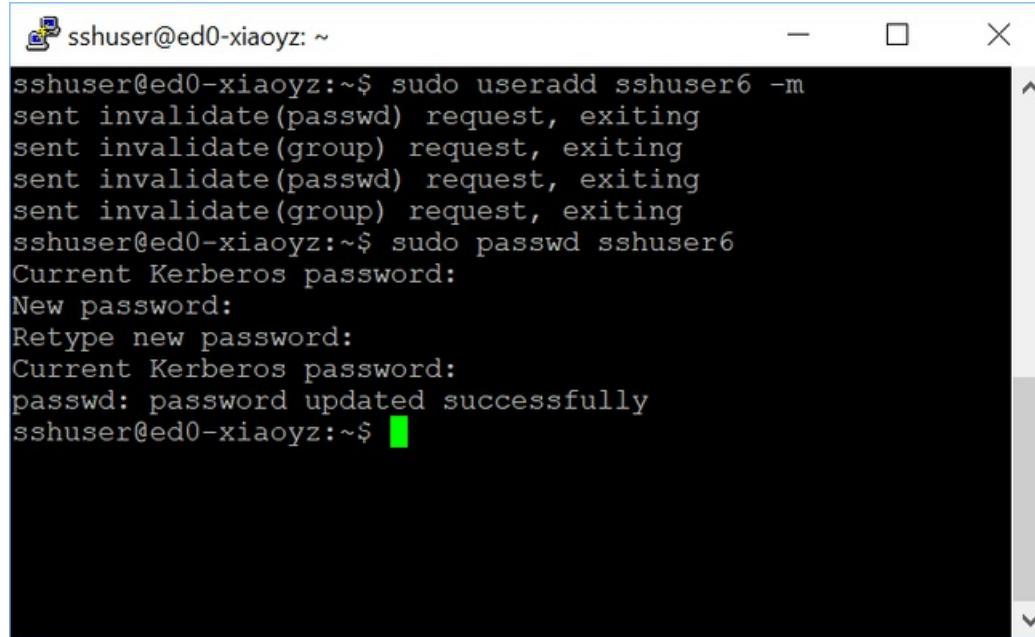
Download any SSH tool (such as PuTTY) and use the existing SSH user to log in. Then follow the instructions provided in [Connect to HDInsight \(Hadoop\) using SSH](#) to access the edge node. The edge node address for the R Server on HDInsight cluster is: **clusternode-ed-ssh.azurehdinsight.net**

Add more Linux users to the edge node

To add a user to the edge node, run these commands:

```
sudo useradd yournewusername -m  
sudo passwd yourusername
```

You should see the following items returned:

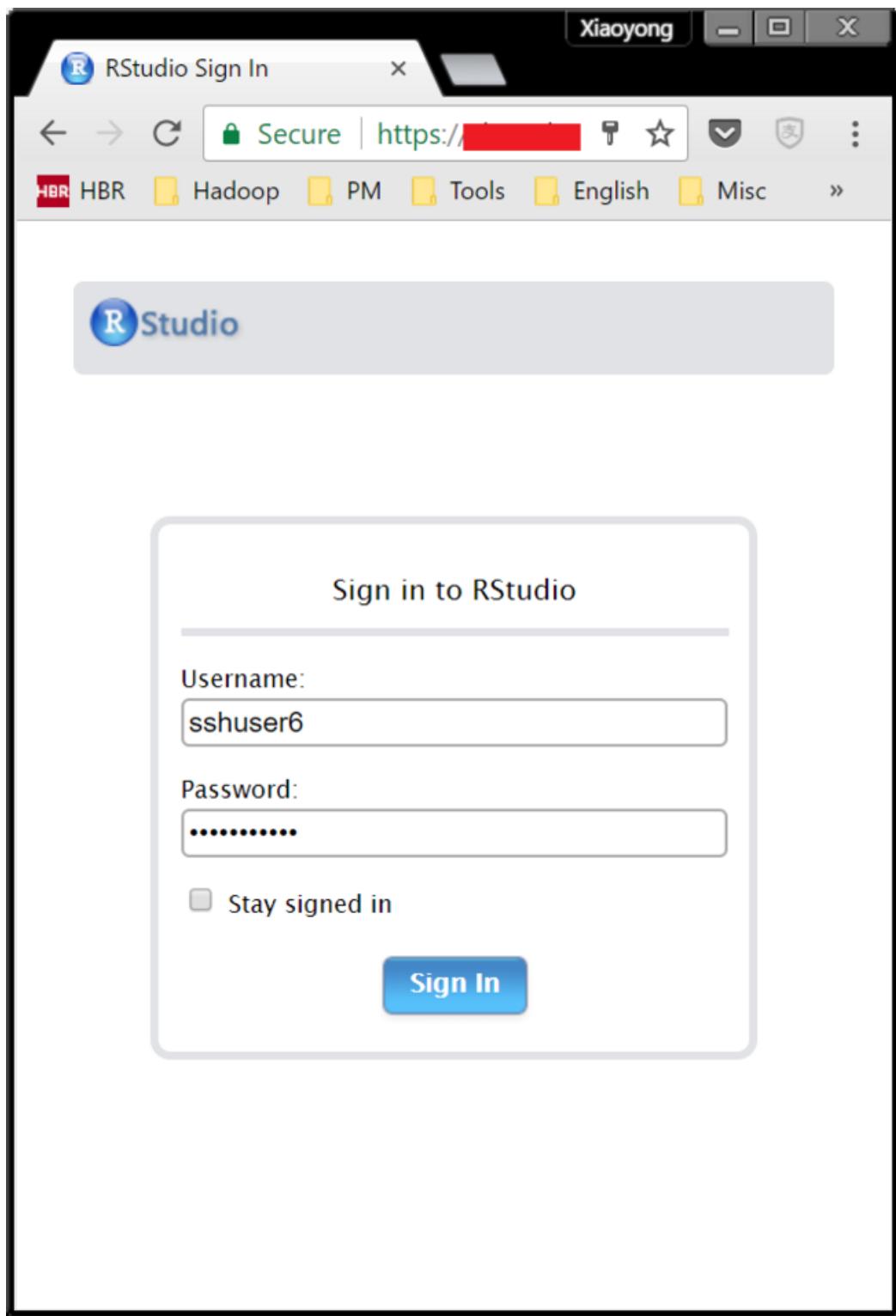


The screenshot shows a terminal window titled "sshuser@ed0-xiaoyz: ~". It displays the command "sudo useradd sshuser6 -m" followed by several "sent invalidate(passwd) request, exiting" messages. Then it shows "sudo passwd sshuser6", "Current Kerberos password:", "New password:", "Retype new password:", "Current Kerberos password:", and finally "passwd: password updated successfully". The terminal has a standard window title bar with minimize, maximize, and close buttons.

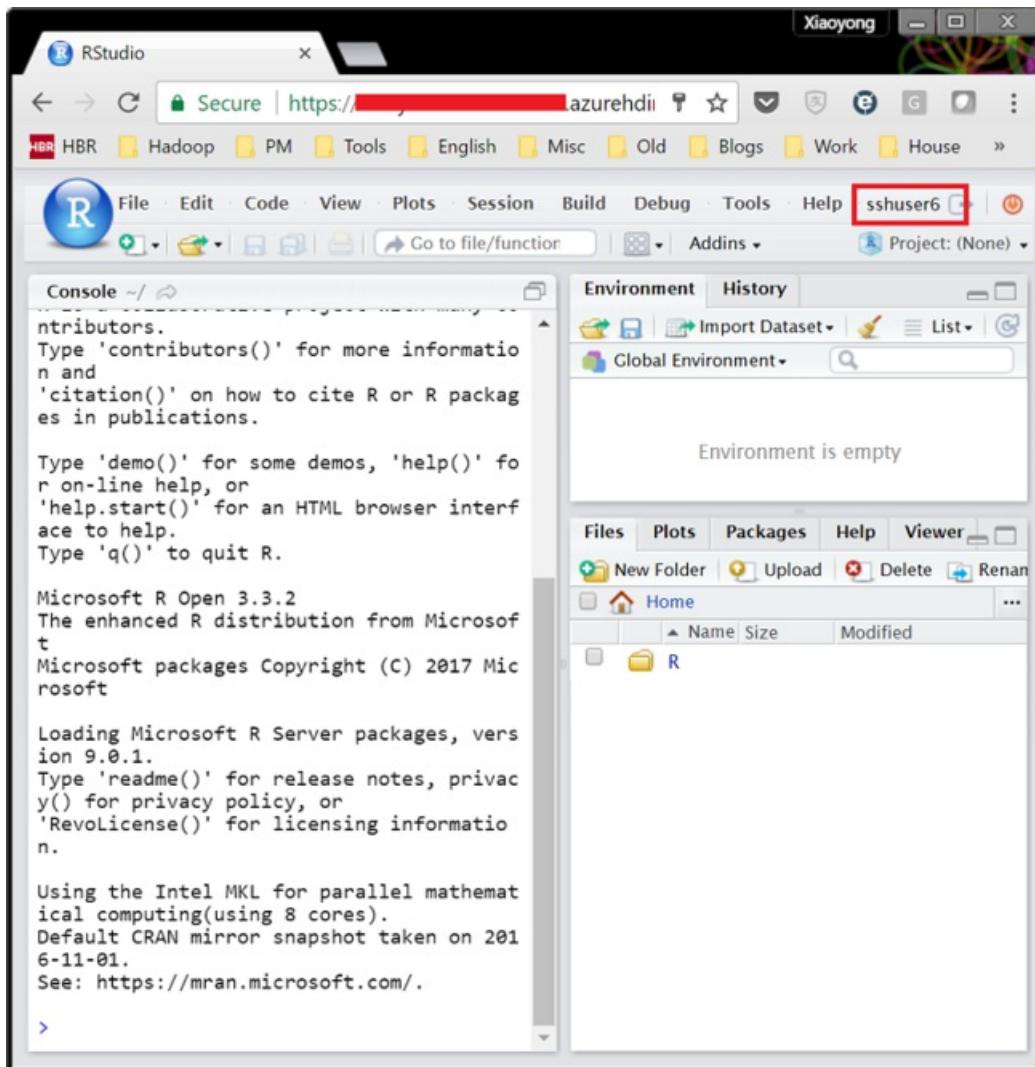
When you're prompted for the current Kerberos password, just select the Enter key to ignore it. The `-m` option in the `useradd` command indicates that the system will create a home folder for the user. This folder is required for the RStudio community version.

Use the RStudio community version with the created user

Use the created user to sign in to RStudio:



Notice that RStudio indicates that you are using the new user (here, for example, **sshuser6**) to log in to the cluster:



You can also log in by using the original credentials (by default, it's **sshuser**) concurrently from another browser window.

You can submit a job by using ScaleR functions. Here is an example of the commands for running a job:

```
# Set the HDFS (Azure Blob storage) location of example data
bigDataDirRoot <- "/example/data"

# Create a local folder for storing data temporarily
source <- "/tmp/AirOnTimeCSV2012"
dir.create(source)

# Download data to the tmp folder
remoteDir <- "https://packages.revolutionanalytics.com/datasets/AirOnTimeCSV2012"
download.file(file.path(remoteDir, "airOT201201.csv"), file.path(source, "airOT201201.csv"))
download.file(file.path(remoteDir, "airOT201202.csv"), file.path(source, "airOT201202.csv"))
download.file(file.path(remoteDir, "airOT201203.csv"), file.path(source, "airOT201203.csv"))
download.file(file.path(remoteDir, "airOT201204.csv"), file.path(source, "airOT201204.csv"))
download.file(file.path(remoteDir, "airOT201205.csv"), file.path(source, "airOT201205.csv"))
download.file(file.path(remoteDir, "airOT201206.csv"), file.path(source, "airOT201206.csv"))
download.file(file.path(remoteDir, "airOT201207.csv"), file.path(source, "airOT201207.csv"))
download.file(file.path(remoteDir, "airOT201208.csv"), file.path(source, "airOT201208.csv"))
download.file(file.path(remoteDir, "airOT201209.csv"), file.path(source, "airOT201209.csv"))
download.file(file.path(remoteDir, "airOT201210.csv"), file.path(source, "airOT201210.csv"))
download.file(file.path(remoteDir, "airOT201211.csv"), file.path(source, "airOT201211.csv"))
download.file(file.path(remoteDir, "airOT201212.csv"), file.path(source, "airOT201212.csv"))

# Set the directory in bigDataDirRoot to load the data
inputDir <- file.path(bigDataDirRoot, "AirOnTimeCSV2012")

# Create the directory
```

```

rxHadoopMakeDir(inputDir)

# Copy the data from source to input
rxHadoopCopyFromLocal(source, bigDataDirRoot)

# Define the HDFS (Blob storage) file system
hdfsFS <- RxHdfsFileSystem()

# Create an info list for the airline data
airlineColInfo <- list(
  DAY_OF_WEEK = list(type = "factor"),
  ORIGIN = list(type = "factor"),
  DEST = list(type = "factor"),
  DEP_TIME = list(type = "integer"),
  ARR_DEL15 = list(type = "logical"))

# Get all the column names
varNames <- names(airlineColInfo)

# Define the text data source in HDFS
airOnTimeData <- RxTextData(inputDir, colInfo = airlineColInfo, varsToKeep = varNames, fileSystem = hdfsFS)

# Define the text data source in the local system
airOnTimeDataLocal <- RxTextData(source, colInfo = airlineColInfo, varsToKeep = varNames)

# Specify the formula to use
formula = "ARR_DEL15 ~ ORIGIN + DAY_OF_WEEK + DEP_TIME + DEST"

# Define the Spark compute context
mySparkCluster <- RxSpark()

# Set the compute context
rxSetComputeContext(mySparkCluster)

# Run a logistic regression
system.time(
  modelSpark <- rxLogit(formula, data = airOnTimeData)
)

# Display a summary
summary(modelSpark)

```

Notice that the submitted jobs are under different user names in the YARN UI:

ID	User	Name	Application Type	Queue	Application Priority	StartTime	FinishTime	State
application_149464829433_0004	sshuser6	scaleR-spark-won2nQFHQA-sshuser6-48254-89568F09223C480D8E910EA8DAE13874	SPARK	default	0	Sat May 13	N/A	RUNN
application_149464829433_0003	zxzytest12	scaleR-spark-won2nQFHQA-zxzytest2-29775-AB88989529344D9B9E0752D98BAE7199	SPARK	default	0	Sat May 13	Sat May 13	FINISH
application_149464829433_0002	sshuser	scaleR-spark-won2nQFHQA-sshuser-58112-93FAFA1E4A994A00928663A6F934884A	SPARK	default	0	Sat May 13	Sat May 13	FINISH

Note also that the newly added users do not have root privileges in Linux system. But they do have the same access to all the files in the remote HDFS file system and Blob storage.

Use the R console

- From the SSH session, use the following command to start the R console:

```
R
```

- You should see output similar to the following:

```
R version 3.2.2 (2015-08-14) -- "Fire Safety"
Copyright (C) 2015 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

Microsoft R Server version 8.0: an enhanced distribution of R
Microsoft packages Copyright (C) 2016 Microsoft Corporation

Type 'readme()' for release notes.
>
```

- From the `>` prompt, you can enter R code. R Server includes packages that you can use to easily interact with Hadoop and run distributed computations. For example, use the following command to view the root of the default file system for the HDInsight cluster:

```
rxHadoopListFiles("/")
```

- You can also use the Blob storage style of addressing:

```
rxHadoopListFiles("wasb:///")
```

Use R Server on HDI from a remote instance of Microsoft R Server or Microsoft R Client

It's possible to set up access to the HDI Hadoop Spark compute context from a remote instance of Microsoft R Server or Microsoft R Client running on a desktop or laptop. For more information, see the "Using Microsoft R Server as a Hadoop Client" section in [Creating a Compute Context for Spark](#). To do so, specify the following options when defining the RxSpark compute context on your laptop: hdfsShareDir, shareDir, sshUsername, sshHostname, sshSwitches, and sshProfileScript. Here's an example:

```

myNameNode <- "default"
myPort <- 0

mySshHostname <- 'rkrrehdi1-ed-ssh.azurehdinsight.net' # HDI secure shell hostname
mySshUsername <- 'remoteuser'# HDI SSH username
mySshSwitches <- '-i /cygdrive/c/Data/R/davec' # HDI SSH private key

myHdfsShareDir <- paste("/user/RevoShare", mySshUsername, sep="/")
myShareDir <- paste("/var/RevoShare" , mySshUsername, sep="/")

mySparkCluster <- RxSpark(
  hdfsShareDir = myHdfsShareDir,
  shareDir      = myShareDir,
  sshUsername   = mySshUsername,
  sshHostname   = mySshHostname,
  sshSwitches   = mySshSwitches,
  sshProfileScript = '/etc/profile',
  nameNode      = myNameNode,
  port          = myPort,
  consoleOutput= TRUE
)

```

Use a compute context

You can use a compute context to control whether computation is performed locally on the edge node or is distributed across the nodes in the HDInsight cluster.

1. From RStudio Server or the R console (in an SSH session), use the following code to load example data into the default storage for HDInsight:

```

# Set the HDFS (Blob storage) location of example data
bigDataDirRoot <- "/example/data"

# Create a local folder for storing data temporarily
source <- "/tmp/AirOnTimeCSV2012"
dir.create(source)

# Download data to the tmp folder
remoteDir <- "https://packages.revolutionanalytics.com/datasets/AirOnTimeCSV2012"
download.file(file.path(remoteDir, "airOT201201.csv"), file.path(source, "airOT201201.csv"))
download.file(file.path(remoteDir, "airOT201202.csv"), file.path(source, "airOT201202.csv"))
download.file(file.path(remoteDir, "airOT201203.csv"), file.path(source, "airOT201203.csv"))
download.file(file.path(remoteDir, "airOT201204.csv"), file.path(source, "airOT201204.csv"))
download.file(file.path(remoteDir, "airOT201205.csv"), file.path(source, "airOT201205.csv"))
download.file(file.path(remoteDir, "airOT201206.csv"), file.path(source, "airOT201206.csv"))
download.file(file.path(remoteDir, "airOT201207.csv"), file.path(source, "airOT201207.csv"))
download.file(file.path(remoteDir, "airOT201208.csv"), file.path(source, "airOT201208.csv"))
download.file(file.path(remoteDir, "airOT201209.csv"), file.path(source, "airOT201209.csv"))
download.file(file.path(remoteDir, "airOT201210.csv"), file.path(source, "airOT201210.csv"))
download.file(file.path(remoteDir, "airOT201211.csv"), file.path(source, "airOT201211.csv"))
download.file(file.path(remoteDir, "airOT201212.csv"), file.path(source, "airOT201212.csv"))

# Set the directory in bigDataDirRoot to load the data into
inputDir <- file.path(bigDataDirRoot,"AirOnTimeCSV2012")

# Make the directory
rxHadoopMakeDir(inputDir)

# Copy the data from source to input
rxHadoopCopyFromLocal(source, bigDataDirRoot)

```

2. Create some data info and define two data sources so that you can work with the data:

```

# Define the HDFS (Blob storage) file system
hdfsFS <- RxHdfsFileSystem()

# Create an info list for the airline data
airlineColInfo <- list(
  DAY_OF_WEEK = list(type = "factor"),
  ORIGIN = list(type = "factor"),
  DEST = list(type = "factor"),
  DEP_TIME = list(type = "integer"),
  ARR_DEL15 = list(type = "logical"))

# Get all the column names
varNames <- names(airlineColInfo)

# Define the text data source in HDFS
airOnTimeData <- RxTextData(inputDir, colInfo = airlineColInfo, varsToKeep = varNames, fileSystem =
hdfsFS)

# Define the text data source in the local system
airOnTimeDataLocal <- RxTextData(source, colInfo = airlineColInfo, varsToKeep = varNames)

# Formula to use
formula = "ARR_DEL15 ~ ORIGIN + DAY_OF_WEEK + DEP_TIME + DEST"

```

3. Run a logistic regression over the data by using the local compute context:

```

# Set a local compute context
rxSetComputeContext("local")

# Run a logistic regression
system.time(
  modelLocal <- rxLogit(formula, data = airOnTimeDataLocal)
)

# Display a summary
summary(modelLocal)

```

You should see output that ends with lines similar to the following:

```

Data: airOnTimeDataLocal (RxTextData Data Source)
File name: /tmp/AirOnTimeCSV2012
Dependent variable(s): ARR_DEL15
Total independent variables: 634 (Including number dropped: 3)
Number of valid observations: 6005381
Number of missing observations: 91381
-2*LogLikelihood: 5143814.1504 (Residual deviance on 6004750 degrees of freedom)

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -3.370e+00 1.051e+00 -3.208 0.00134 **
ORIGIN=JFK    4.549e-01 7.915e-01  0.575 0.56548
ORIGIN=LAX    5.265e-01 7.915e-01  0.665 0.50590
.....
DEST=SHD      5.975e-01 9.371e-01  0.638 0.52377
DEST=TTN      4.563e-01 9.520e-01  0.479 0.63172
DEST=LAR     -1.270e+00 7.575e-01 -1.676 0.09364 .
DEST=BPT      Dropped   Dropped Dropped Dropped

---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Condition number of final variance-covariance matrix: 11904202
Number of iterations: 7

```

- Run the same logistic regression by using the Spark context. The Spark context distributes the processing over all the worker nodes in the HDInsight cluster.

```

# Define the Spark compute context
mySparkCluster <- RxSpark()

# Set the compute context
rxSetComputeContext(mySparkCluster)

# Run a logistic regression
system.time(
  modelSpark <- rxLogit(formula, data = airOnTimeData)
)

# Display a summary
summary(modelSpark)

```

NOTE

You can also use MapReduce to distribute computation across cluster nodes. For more information on compute context, see [Compute context options for R Server on HDInsight](#).

Distribute R code to multiple nodes

With R Server, you can easily take existing R code and run it across multiple nodes in the cluster by using `rxExec`. This function is useful when you're doing a parameter sweep or simulations. The following code is an example of how to use `rxExec`:

```
rxExec( function() {Sys.info()["nodename"]}, timesToRun = 4 )
```

If you are still using the Spark or MapReduce context, this command returns the `nodename` value for the worker nodes that the code `(Sys.info()["nodename"])` is run on. For example, on a four-node cluster, you expect to receive

output similar to the following:

```
$rxElem1
  nodename
"wn3-myrsr"

$rxElem2
  nodename
"wn0-myrsr"

$rxElem3
  nodename
"wn3-myrsr"

$rxElem4
  nodename
"wn3-myrsr"
```

Access data in Hive and Parquet

A feature available in R Server 9.1 allows direct access to data in Hive and Parquet for use by ScaleR functions in the Spark compute context. These capabilities are available through new ScaleR data source functions called RxHiveData and RxParquetData. These functions work through use of Spark SQL to load data directly into a Spark DataFrame for analysis by ScaleR.

The following code provides some examples of how to use the new functions:

```
#Create a Spark compute context
myHadoopCluster <- rxSparkConnect(reset = TRUE)

#Retrieve some sample data from Hive and run a model
hiveData <- RxHiveData("select * from hivesampletable",
                       colInfo = list(devicemake = list(type = "factor")))
rxGetInfo(hiveData, getVarInfo = TRUE)

rxLinMod(querydwelltime ~ devicemake, data=hiveData)

#Retrieve some sample data from Parquet and run a model
rxHadoopMakeDir('/share')
rxHadoopCopyFromLocal(file.path(rxGetOption('sampleDataDir'), 'claimsParquet/'), '/share/')
pqData <- RxParquetData('/share/claimsParquet',
                        colInfo = list(
                          age      = list(type = "factor"),
                          car.age = list(type = "factor"),
                          type    = list(type = "factor")
                        ))
rxGetInfo(pqData, getVarInfo = TRUE)

rxNaiveBayes(type ~ age + cost, data = pqData)

#Check on Spark data objects, clean up, and close the Spark session
lsObj <- rxSparkListData() #Two data objects are cached
lsObj
rxSparkRemoveData(lsObj)
rxSparkListData() #It should show an empty list
rxSparkDisconnect(myHadoopCluster)
```

For more information about these new functions, see the online help in R Server by using the `?RxHivedata` and `?RxParquetData` commands.

Install additional R packages on the edge node

If you want to install additional R packages on the edge node, you can use `install.packages()` directly from within the R console when connected to the edge node through SSH. However, if you need to install R packages on the worker nodes of the cluster, you must use a script action.

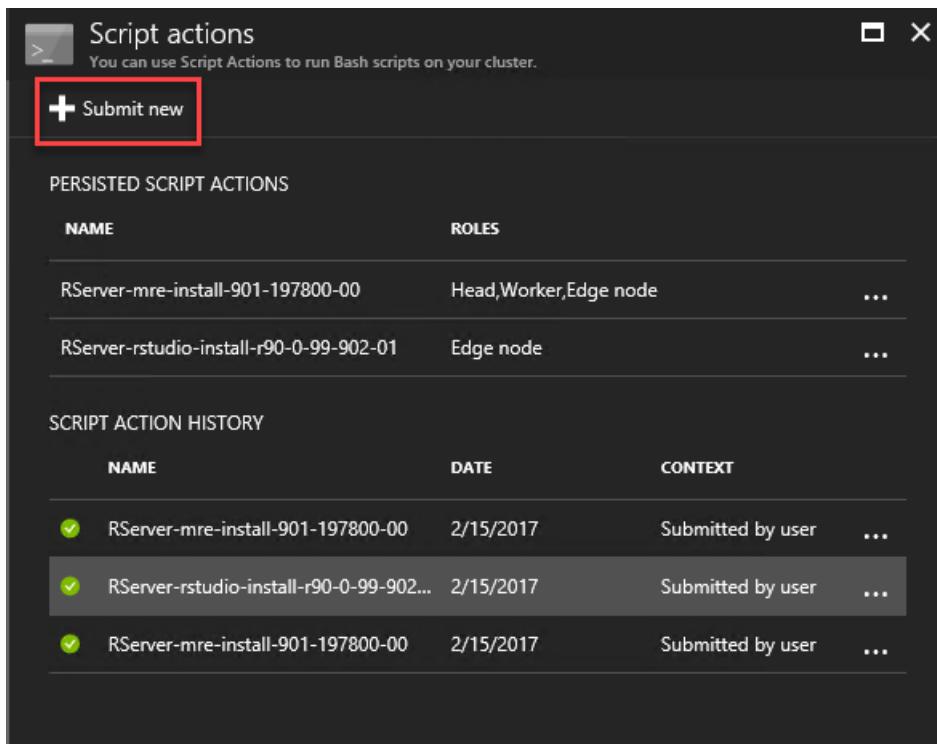
Script actions are Bash scripts that are used to make configuration changes to the HDInsight cluster or to install additional software, such as additional R packages. To install additional packages by using a script action, use the following steps.

IMPORTANT

You can use script actions to install additional R packages only after the cluster is created. Do not use this procedure during cluster creation, because the script relies on R Server being completely installed and configured.

1. From the [Azure portal](#), select your R Server on HDInsight cluster.

2. From the **Settings** pane, select **Script Actions** > **Submit New**.



3. From the **Submit script action** pane, provide the following information:

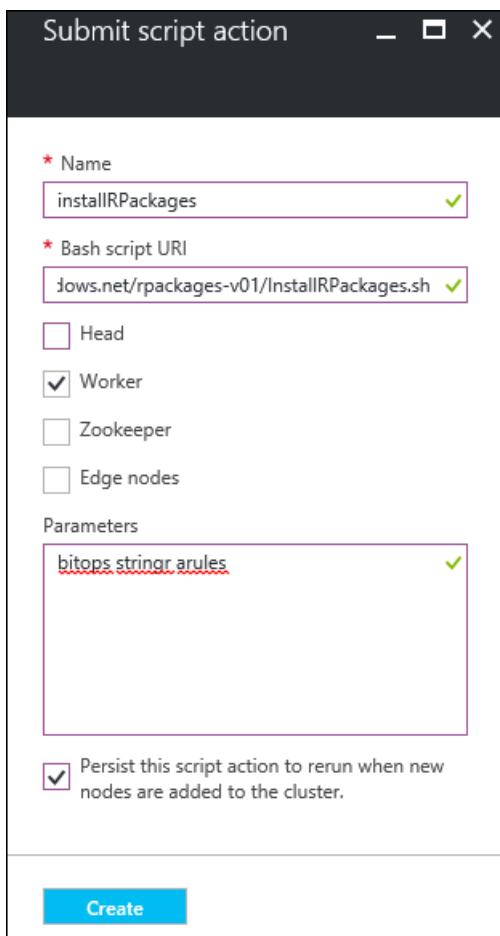
- **Name:** A friendly name to identify this script.
- **Bash script URI:** `http://mrsactionscripts.blob.core.windows.net/rpackages-v01/InstallRPackages.sh`
- **Head:** This item should be cleared.
- **Worker:** This item should be cleared.
- **Zookeeper:** This item should be cleared.
- **Edge nodes:** This item should be selected.
- **Parameters:** The R packages to be installed--for example, `bitops stringr arules`.
- **Persist this script:** This item should be selected.

NOTE

By default, all R packages are installed from a snapshot of the Microsoft R Application Network repository that's consistent with the installed version of R Server. If you want to install newer versions of packages, there is some risk of incompatibility. However, this kind of installation is possible if you specify `useCRAN` as the first element of the package list--for example, `useCRAN bitops, stringr, arules`.

Some R packages require additional Linux system libraries. For convenience, we have pre-installed the dependencies that the 100 most popular R packages need. If the R packages that you install require libraries beyond these, you must download the base script used here and add steps to install the system libraries. You must then upload the modified script to a public blob container in Azure Storage and use the modified script to install the packages.

For more information on developing script actions, see [Script action development](#).



4. Select **Create** to run the script. After the script finishes, the R packages are available on all worker nodes.

Configure Microsoft R Server operationalization

When your data modeling is complete, you can operationalize the model to make predictions. To configure Microsoft R Server operationalization, perform the following steps:

1. Use the `ssh` command for the edge node--for example:

```
ssh -L USERNAME@CLUSTERNAME-ed-ssh.azurehdinsight.net
```

2. Change directory for the relevant version, and use the `sudo dotnet` command for the .dll file.

For Microsoft R Server 9.1:

```
cd /usr/lib64/microsoft-r/rserver/o16n/9.1.0
sudo dotnet Microsoft.RServer.Utils.AdminUtil/Microsoft.RServer.Utils.AdminUtil.dll
```

For Microsoft R Server 9.0:

```
cd /usr/lib64/microsoft-deployr/9.0.1
sudo dotnet Microsoft.DeployR.Utils.AdminUtil/Microsoft.DeployR.Utils.AdminUtil.dll
```

3. To configure Microsoft R Server operationalization with a one-box configuration, do the following:

- a. Select **Configure R Server for Operationalization**.
- b. Select **A. One-box (web + compute nodes)**.
- c. Enter a password for the **admin** user.

```
root@ed10-jasozh:/usr/lib64/microsoft-deployr/9.0.1/Microsoft.DeployR.Utils.AdminUtil# dotnet Microsoft.DeployR.Utils.AdminUtil.dll

*****
 Administration Utility (v9.0.1)
 *****

1. Configure R Server for Operationalization
2. Set a local admin password
3. Stop and start services
4. Change service ports
5. Encrypt credentials
6. Run diagnostic tests
7. Evaluate capacity
8. Exit

Please enter an option:
1

Configuration for Operationalization:

A. One-box (web + compute nodes)
B. Web node
C. Compute node
D. Reset machine to default install state
E. Return to main menu

Please enter an option:
1
```

4. As an optional step, you can perform a diagnostic test as follows:

- a. Select **6. Run diagnostic tests**.
- b. Select **A. Test configuration**.
- c. Enter **admin** for the username, and enter the password from the previous configuration step.
- d. Confirm **Overall Health = pass**.
- e. Exit the admin utility.
- f. Exit SSH.

```

root@ed10-jasozh:/usr/lib64/microsoft-deployr/9.0.1/Microsoft.DeployR.Utils.AdminUtil# dotnet Microsoft.DeployR.Utils.AdminUtil.dll
*****
 Administration Utility (v9.0.1)
*****


1. Configure R Server for Operationalization
2. Set a local admin password
3. Stop and start services
4. Change service ports
5. Encrypt credentials
6. Run diagnostic tests
7. Evaluate capacity
8. Exit

Web node endpoint: http://localhost:12800/
Please enter an option:
6

Diagnostic Tests:
A. Test configuration
B. Get raw server status
C. Trace code execution
D. Trace service execution
E. Return to main menu
Please enter an option:
a

Preparing to run diagnostics...
Please authenticate...

Username:
admin
Password:
*****



*****DIAGNOSTIC RESULTS:*****
*****Overall Health: pass

Web Node Details:
Logs: /usr/lib64/microsoft-deployr/9.0.1/Microsoft.DeployR.Server.WebAPI/logs
Available compute nodes: 1

Compute Node Details:
Health of 'http://localhost:12805/': pass
Logs: /usr/lib64/microsoft-deployr/9.0.1/Microsoft.DeployR.Server.BackEnd/logs

Authentication Details:
A local admin account was found. No other form of authentication is configured.

Database Details:
Health: pass
Type: sqlite

Code Execution Test: PASS
Code: 'y <- cumprod(c(1500, 1+rnorm(n=25,mean=.05, sd = 1.4)/100))'

```

NOTE

If you encounter long delays when trying to consume a web service created with mrsdeploy functions in a Spark compute context, you might need to add some missing folders. The Spark application belongs to a user called *rserve2* whenever it's invoked from a web service through mrsdeploy functions. To work around this issue:

```

#Create these required folders for user rserve2 in local and HDFS
hadoop fs -mkdir /user/RevoShare/rserve2
hadoop fs -chmod 777 /user/RevoShare/rserve2

mkdir /var/RevoShare/rserve2
chmod 777 /var/RevoShare/rserve2

#Create a new Spark compute context
rxSparkConnect(reset = TRUE)

```

At this stage, the configuration for operationalization is complete. Now you can use the mrsdeploy package on R Client to connect to the operationalization on the edge node. You can then start using its features, like [remote execution](#) and [web services](#). Depending on whether your cluster is set up on a virtual network or not, you might need to set up port forward tunneling through an SSH login.

R Server cluster on a virtual network

Make sure that you allow traffic through port 12800 to the edge node. That way, you can use the edge node to connect to the operationalization feature.

```

library(mrsdeploy)

remoteLogin(
  deployr_endpoint = "http://[your-cluster-name]-ed-ssh.azurehdinsight.net:12800",
  username = "admin",
  password = "xxxxxx"
)

```

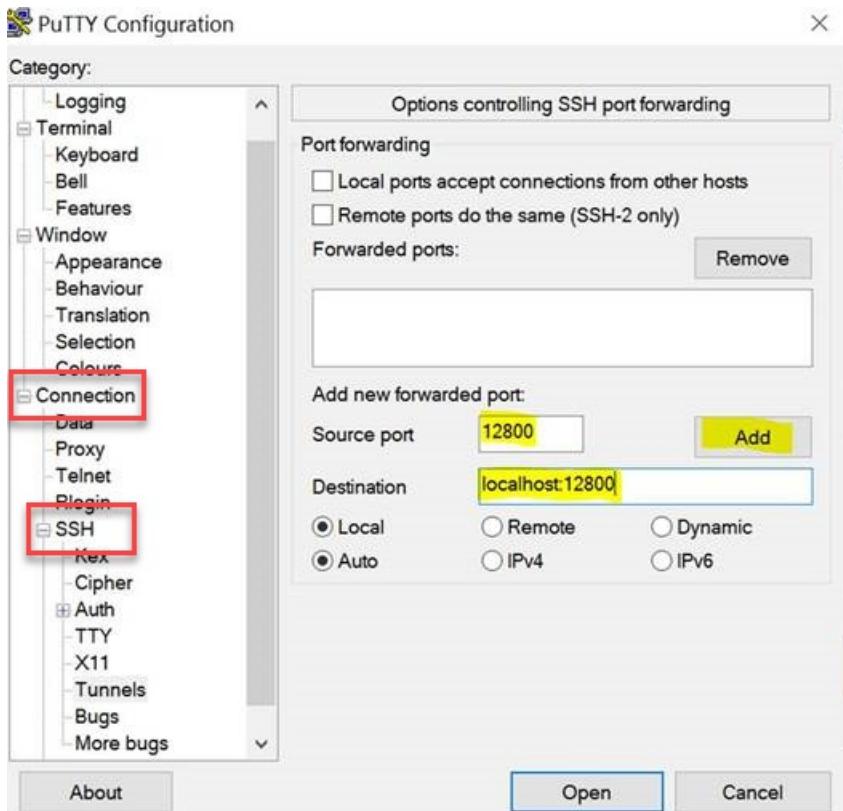
If `remoteLogin()` cannot connect to the edge node, but you can use SSH to connect to the edge node, you need to check whether the rule to allow traffic on port 12800 is set properly. If you continue to face the issue, you can work around it by setting up port forward tunneling through SSH. For instructions, see the following section.

R Server cluster not set up on a virtual network

If your cluster is not set up on a virtual network, or if you're having trouble with connectivity through a virtual network, you can use SSH port forward tunneling:

```
ssh -L localhost:12800:localhost:12800 USERNAME@CLUSTERNAME-ed-ssh.azurehdinsight.net
```

You can also set it up on PuTTY:



After your SSH session is active, the traffic from your machine's port 12800 is forwarded to the edge node's port 12800 through the SSH session. Make sure that you use `127.0.0.1:12800` in your `remoteLogin()` method. This method logs in to the edge node's operationalization through port forwarding.

```

library(mrsdeploy)

remoteLogin(
  deployr_endpoint = "http://127.0.0.1:12800",
  username = "admin",
  password = "xxxxxx"
)

```

Scale Microsoft R Server operationalization compute nodes on HDInsight worker nodes

Decommission the worker nodes

Microsoft R Server is currently not managed through Yarn. If the worker nodes are not decommissioned, Yarn ResourceManager will not work as expected because it will not be aware of the resources that the server is using. To avoid this situation, we recommend decommissioning the worker nodes before you scale out the compute nodes.

To decommission worker nodes:

1. Log in to the HDI cluster's Ambari console and select the **Hosts** tab.
2. Select worker nodes to be decommissioned, and then select **Actions > Selected Hosts > Hosts > Turn On Maintenance Mode**. For example, in the following image, we have selected wn3 and wn4 to decommission.

The screenshot shows the Ambari Hosts tab with the following details:

- Actions dropdown: Actions ▾
- Filter: All (10) ▾
- Table headers: IP Address, Rack
- Table rows:
 - Selected Hosts (2) ▾
 - Filtered Hosts (10) ▾
 - All Hosts (10) ▾
 - hn0-deploy.sdaosvcem5... (10.0.0.16)
 - hn1-deploy.sdaosvcem5... (10.0.0.17)
 - wn0-deploy.sdaosvcem5... (10.0.0.15) /default-rack
 - wn1-deploy.sdaosvcem5... (10.0.0.14) /default-rack
 - wn3-deploy.sdaosvcem5... (10.0.0.13) /default-rack** (highlighted with a yellow box)
 - wn4-deploy.sdaosvcem5... (10.0.0.11) /default-rack** (highlighted with a yellow box)
- Context menu (open over wn3 and wn4):
 - Start All Components
 - Stop All Components
 - Restart All Components
 - Turn On Maintenance Mode** (highlighted with a yellow box)
 - Turn Off Maintenance Mode
 - Set Rack

3. Select **Actions > Selected Hosts > DataNodes > Decommission**.
4. Select **Actions > Selected Hosts > NodeManagers > Decommission**.
5. Select **Actions > Selected Hosts > DataNodes > Stop**.
6. Select **Actions > Selected Hosts > NodeManagers > Stop**.
7. Select **Actions > Selected Hosts > Hosts > Stop All Components**.
8. Unselect the worker nodes and select the head nodes.
9. Select **Actions > Selected Hosts > Hosts > Restart All Components**.

Configure compute nodes on each decommissioned worker node

1. Use SSH to connect to each decommissioned worker node.
2. Run an admin utility by using

```
dotnet /usr/lib64/microsoft-
deployr/9.0.1/Microsoft.DeployR.Utils.AdminUtil/Microsoft.DeployR.Utils.AdminUtil.dll
```
3. Enter **1** to select the option **Configure R Server for Operationalization**.
4. Enter **c** to select the option **C. Compute node**. This step configures the compute node on the worker node.
5. Exit the admin utility.

Add compute nodes' details on the web node

After all decommissioned worker nodes are configured to run on the compute node, go back to the edge node and add decommissioned worker nodes' IP addresses in the Microsoft R Server web node's configuration:

1. Use SSH to connect to the edge node.

2. Run `vi /usr/lib64/microsoft-deployr/9.0.1/Microsoft.DeployR.Server.WebAPI/appsettings.json`.

3. Look for the `URIs` section, and add the worker nodes' IP and port details.

```
"UrIs": {
  "Description": "Update 'Values' section to point to your backend machines. Using HTTPS is highly recommended",
  "Values": [
    "http://localhost:12805", "https://[worker-node1-ip]:12805", "https://[worker-node2-ip]:12805"
  ]
}
```

Troubleshoot

If you have problems with creating HDInsight clusters, see [Access control requirements](#).

Next steps

Now you should understand how to create an HDInsight cluster that includes R Server. You should also understand the basics of using the R console from an SSH session. The following topics explain other ways of managing and working with R Server on HDInsight:

- [Compute context options for R Server on HDInsight](#)
- [Azure Storage options for R Server on HDInsight](#)

Create and share an Azure Machine Learning workspace

12/7/2017 • 2 min to read • [Edit Online](#)

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

You can try Azure Machine Learning for free. No credit card or Azure subscription is required. [Get started now.](#)

To create a workspace

1. Sign in to the [Azure portal](#)

NOTE

To sign in and create a workspace, you need to be an Azure subscription administrator.

2. Click **+New**
3. In the search box, type **Machine Learning Studio Workspace** and select the matching item. Then, select click **Create** at the bottom of the page.
4. Enter your workspace information:
 - The *workspace name* may be up to 260 characters, not ending in a space. The name can't include these characters: < > * % & : \ ? + /
 - The *web service plan* you choose (or create), along with the associated *pricing tier* you select, is used if you deploy web services from this workspace.

Machine Learning Workspace □ X

Machine Learning Workspace

* Workspace name
My-workspace ✓

* Subscription
My-subscription

* Resource group ⓘ
 Create new Use existing
My-resource-group ✓

* Location
South Central US

* Storage account ⓘ
 Create new Use existing
storageformyworkspace ✓

Workspace pricing tier ⓘ
Standard

* Web service plan ⓘ
 Create new Use existing
My-web-service-plan ✓

* Web service plan pricing tier ⓘ >
S1 Standard

5. Click **Create**.

Once the workspace is deployed, you can open it in Machine Learning Studio.

1. Browse to Machine Learning Studio at <https://studio.azureml.net/>.
2. Select your workspace in the upper-right-hand corner.



3. Click **my experiments**.

Welcome back luisa!

MY RECENT WORKSPACES:

My-workspace

MY RECENT EXPERIMENTS:

my experiments →

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 a Machine Learning workspace](#).

Sharing an Azure Machine Learning workspace

Once a Machine Learning workspace is created, you can invite users to your workspace to share access to your workspace and all its experiments, datasets, notebooks, etc. You can add users in one of two roles:

- **User** - A workspace user can create, open, modify, and delete experiments, datasets, etc. in the workspace.
- **Owner** - An owner can invite and remove users in the workspace, in addition to what a user can do.

NOTE

The administrator account that creates the workspace is automatically added to the workspace as workspace Owner. However, other administrators or users in that subscription are not automatically granted access to the workspace - you need to invite them explicitly.

To share a workspace

1. Sign in to Machine Learning Studio at <https://studio.azureml.net/Home>
2. In the left panel, click **SETTINGS**
3. Click the **USERS** tab
4. Click **INVITE MORE USERS** at the bottom of the page

The screenshot shows the Microsoft Azure Machine Learning Studio interface. On the left, there's a sidebar with icons for Projects, Experiments, Web Services, Notebooks, Datasets, Trained Models, and Settings. The 'SETTINGS' icon is highlighted with a red box. The main area has a title 'settings'. At the top, there are tabs: NAME, AUTHORIZATION TOKENS, USERS (which is highlighted with a red box), and DATA GATEWAYS. Below the tabs is a table with columns: NAME, EMAIL, ROLE, and STATUS. One row is visible: Luisa, luisa@contoso.com, Owner, Active. At the bottom of the page, there are three buttons: '+ NEW', 'INVITE MORE USERS' (which is highlighted with a red box), and 'REMOVE'.

5. Enter one or more email addresses. The users need a valid Microsoft account or an organizational account (from Azure Active Directory).
6. Select whether you want to add the users as Owner or User.
7. Click the **OK** checkmark button.

Each user you add will receive an email with instructions on how to sign in to the shared workspace.

NOTE

For users to be able to deploy or manage web services in this workspace, they must be a contributor or administrator in the Azure subscription.

Provision the Windows Data Science Virtual Machine on Azure

3/6/2018 • 12 min to read • [Edit Online](#)

The Microsoft Data Science Virtual Machine is a Windows 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:

- [Azure Machine Learning Workbench](#)
- [Microsoft Machine Learning Server](#) Developer Edition
- Anaconda Python distribution
- Jupyter notebook (with R, Python, PySpark kernels)
- Visual Studio Community Edition
- Power BI desktop
- SQL Server 2017 Developer Edition
- Standalone Spark instance for local development and testing
- [JuliaPro](#)
- Machine learning and Data Analytics tools
 - Deep Learning Frameworks: A rich set of AI frameworks including [Microsoft Cognitive Toolkit](#), [TensorFlow](#), [Chainer](#), mxNet, Keras are included on the VM.
 - [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. It includes GUI-based data exploration and modeling with automatic R code generation.
 - [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 and provides several popular Linux command-line utilities (including awk, sed, perl, grep, find, wget, curl, etc.) accessible both on git-bash and command prompt.

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 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.

Other Versions of the Data Science Virtual Machine

An [Ubuntu](#) image is available as well, with many similar tools plus a few additional deep learning frameworks. A [CentOS](#) image is also available. We also offer a [Windows Server 2012 edition](#) of the data science virtual machine though a few tools are only available on the Windows Server 2016 edition. Otherwise, this article also applies to the Windows Server 2012 edition.

Prerequisites

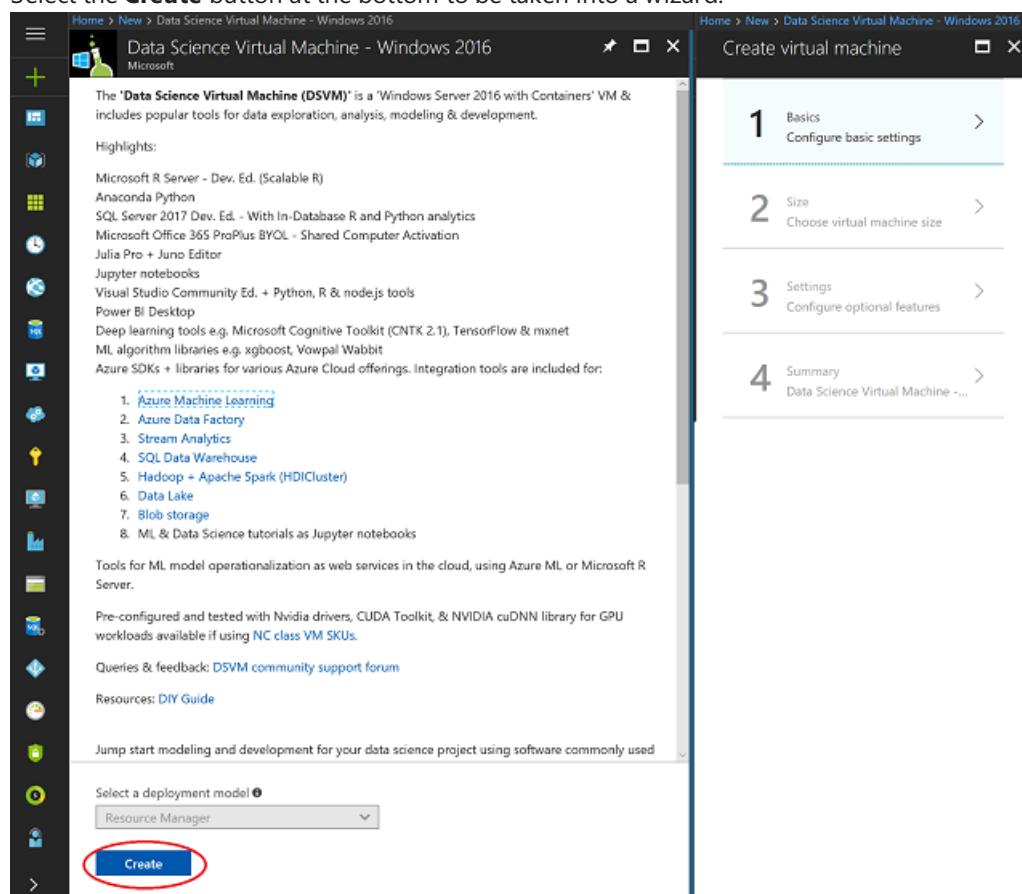
Before you can create a Microsoft Data Science Virtual Machine, you must have the following:

- **An Azure subscription:** To obtain one, see [Get Azure free trial](#).

Create your Microsoft Data Science Virtual Machine

To create an instance of the Microsoft Data Science Virtual Machine, follow these steps:

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 **four steps** enumerated on the right of this figure. Here are the inputs needed to configure each of these steps:

- a. **Basics**

- a. **Name:** Name of your data science server you are creating.
 - b. **VM Disk Type:** Choose between SSD or HDD. For NC_v1 GPU instance (NVidia Tesla K80 based), choose **HDD** as the disk type.
 - c. **User Name:** Admin account login id.
 - d. **Password:** Admin account password.
 - e. **Subscription:** If you have more than one subscription, select the one on which the machine is to be created and billed.
 - f. **Resource Group:** You can create a new one or use an existing group.
 - g. **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:**
- a. **Use Managed Disks:** Choose Managed if you want Azure to manage the disks for the VM. Otherwise you need to specify a new or existing storage account.
 - b. **Other parameters:** Usually you just use the default values. If you want to consider using non-default values, hover over the informational link for help on the specific fields. a. **Summary:** Verify that all information you entered is correct and click **Create**. **NOTE:** 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.

Tools installed on the Microsoft Data Science Virtual Machine

Microsoft ML Server Developer Edition

If you wish to use Microsoft enterprise libraries for scalable R or Python for your analytics, the VM has Microsoft ML Server Developer edition (Previously known as Microsoft R Server) installed. Microsoft ML Server is a broadly deployable enterprise-class analytics platform available for both R and Python and is scalable, commercially supported and secure. Supporting a variety of big data statistics, predictive modeling and machine learning capabilities, ML Server supports the full range of analytics – exploration, analysis, visualization, and modeling. By using and extending open source R and Python, Microsoft ML Server is fully compatible with R / Python scripts, functions and CRAN / pip / Conda 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 and Python tools for Visual Studio extension that provides a full IDE for working with R or Python. We also provide other IDEs as well such as [RStudio](#) and [PyCharm Community edition](#) on the VM.

Python

For development using Python, Anaconda Python distribution 2.7 and 3.6 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 2017 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, you need to create custom environments for each version. To set these environment paths in the Visual Studio 2017 Community Edition, navigate to **Tools -> Python Tools -> Python Environments** and then click **+ Custom**.

Anaconda Python 3.6 is installed under C:\Anaconda and Anaconda Python 2.7 is installed under c:\Anaconda\envs\python2. 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.x, PySpark, Julia and R kernels. There is a desktop icon named "Jupyter Notebook" to start the Jupyter server and launch the browser to access the Notebook server.

We have packaged several sample notebooks in Python and in R. The Jupyter notebooks show how to work with Microsoft ML Server, SQL Server ML Services (In-database analytics), Python, Microsoft Cognitive Toolkit, Tensorflow and other Azure technologies once you access 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 2017 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. Now there is also a plugin called [Visual Studio Tools for AI](#) that seamlessly integrates to Azure Machine Learning and helps you rapidly build AI applications.

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 2017 Developer edition

A developer version of SQL Server 2017 with ML Services to run in-database analytics is provided on the VM in either R or Python. ML Services provide a platform for developing and deploying intelligent applications. You can use the rich and powerful these languages 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 ML Services (In-database) integrates both the R and Python language within the SQL Server. This eliminates the costs and security risks associated with data movement.

NOTE

The SQL Server 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 SQL ML Services, run the following command as a one time action in SQL Server management studio after logging in as the server administrator.

```
CREATE LOGIN [%COMPUTERNAME%\SQLUserGroup] 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 Cosmos DB, a NoSQL database on the cloud. Type **dtui** on command prompt.
- **Azure Data Factory Integration Runtime**: 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.

Azure Machine Learning Workbench

Azure Machine Learning Workbench is a desktop application and command-line interface. The Workbench has built-in data preparation that learns your data preparation steps as you perform them. It also provides project management, run history, and notebook integration to bolster your productivity. You can take advantage of the best open-source frameworks, including TensorFlow, Cognitive Toolkit, Spark ML, and scikit-learn to develop your models. On the DSVM, we provide a desktop icon to install the Azure Machine Learning workbench into the individual user's %LOCALAPPDATA% directory. Each user that needs to use the Workbench needs to do a one time action of double-clicking the **AzureML Workbench Setup** desktop icon to install their instance of the Workbench.

Azure Machine Learning also creates and uses a per-user Python environment that is extracted in the %LOCALAPPDATA%\amlworkbench\python.

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 and c\users<username>\notebooks
Other samples	c\dsvm\samples
Anaconda (default: Python 3.6)	c\Anaconda
Anaconda Python 2.7 environment	c\Anaconda\envs\python2
Microsoft ML Server Standalone Python	C:\Program Files\Microsoft\ML Server\PYTHON_SERVER
Default R instance (ML Server Standalone)	C:\Program Files\Microsoft\ML Server\R_SERVER
SQL ML Services In-database instance directory	C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER
Azure Machine Learning Workbench (per user)	%localappdata%\amlworkbench
Miscellaneous tools	c\dsvm\tools

NOTE

On Windows Server 2012 edition of the DSVM and Windows Server 2016 edition before March 2018, the default Anaconda environment is Python 2.7. The secondary environment is Python 3.5 located at c\Anaconda\envs\py35.

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.
- Learn about Azure Machine Learning Services and Workbench by visiting the product [quickstart and tutorials page](#).
- Navigate to **C:\Program Files\Microsoft\ML Server\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 [Azure AI Gallery](#) for machine learning and data analytics samples that use Azure Machine learning and related data services on Azure. We have also provided an icon on the **Start** menu and on the desktop of the virtual machine to this gallery.

Provision the Data Science Virtual Machine for Linux (Ubuntu)

9/25/2017 • 23 min to read • [Edit Online](#)

The Data Science Virtual Machine for Linux is an Ubuntu-based virtual machine image that makes it easy to get started with deep learning on Azure. Deep learning tools include:

- [Caffe](#): A deep learning framework built for speed, expressivity, and modularity
- [Caffe2](#): A cross-platform version of Caffe
- [Microsoft Cognitive Toolkit](#): A deep learning software toolkit from Microsoft Research
- [H2O](#): An open-source big data platform and graphical user interface
- [Keras](#): A high-level neural network API in Python for Theano and TensorFlow
- [MXNet](#): A flexible, efficient deep learning library with many language bindings
- [NVIDIA DIGITS](#): A graphical system that simplifies common deep learning tasks
- [TensorFlow](#): An open-source library for machine intelligence from Google
- [Theano](#): A Python library for defining, optimizing, and efficiently evaluating mathematical expressions involving multi-dimensional arrays
- [Torch](#): A scientific computing framework with wide support for machine learning algorithms
- CUDA, cuDNN, and the NVIDIA driver
- Many sample Jupyter notebooks

All libraries are the GPU versions, though they also run on the CPU.

The Data Science Virtual Machine for Linux also contains popular tools for data science and development activities, including:

- Microsoft R Server Developer Edition with Microsoft R Open
- Anaconda Python distribution (versions 2.7 and 3.5), including popular data analysis libraries
- JuliaPro - a curated distribution of Julia language with popular scientific and data analytics libraries
- Standalone Spark instance and single node Hadoop (HDFS, Yarn)
- JupyterHub - a multiuser Jupyter notebook server supporting R, Python, PySpark, Julia kernels
- Azure Storage Explorer
- Azure command-line interface (CLI) for managing Azure resources
- Machine learning tools
 - [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](#): A graphical tool that makes getting started with data analytics and machine learning in R easy
 - [LightGBM](#): A fast, distributed, high-performance gradient boosting framework
- 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 (RStudio, PyCharm, IntelliJ, Emacs, vim)

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 Data Science Virtual Machine for Linux 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++. 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. More details on the compute fees can be found on the [VM listing page on the Azure Marketplace](#).

Other Versions of the Data Science Virtual Machine

A [CentOS](#) image is also available, with many of the same tools as the Ubuntu image. A [Windows](#) image is available as well.

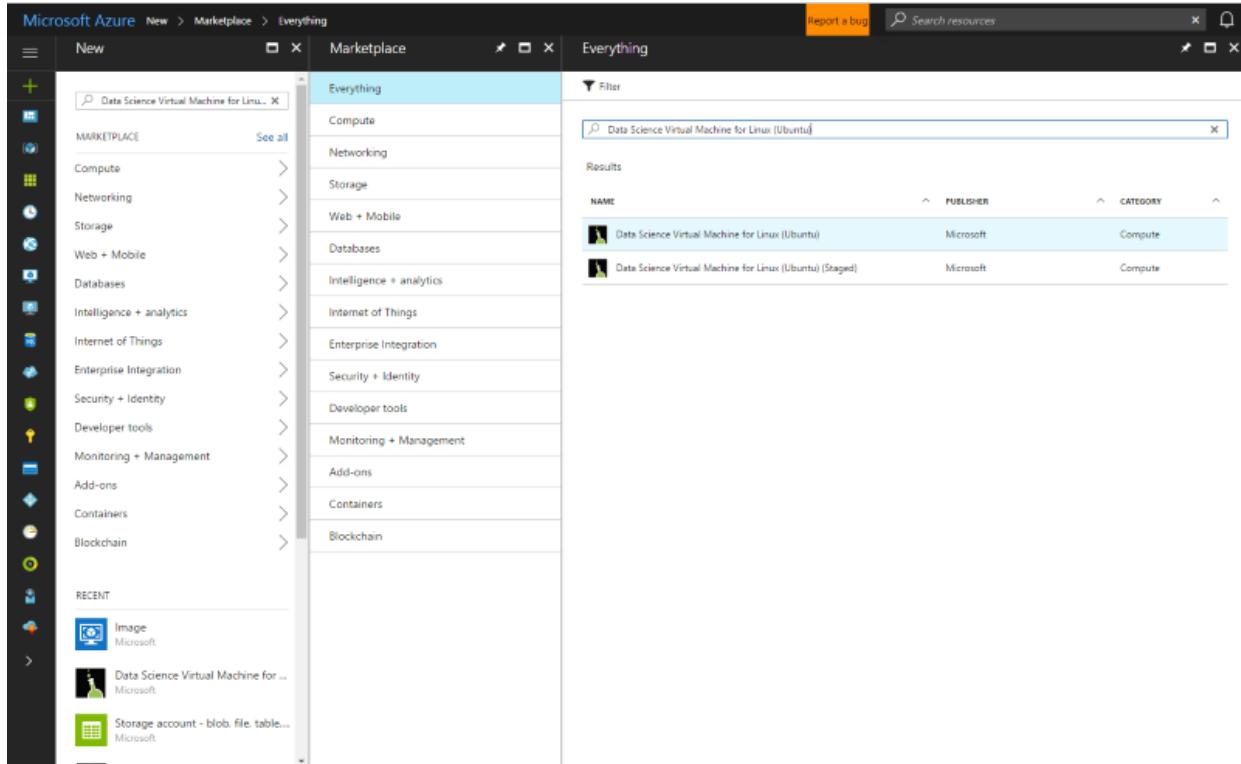
Prerequisites

Before you can create a Data Science Virtual Machine for Linux, you must have an Azure subscription. To obtain one, see [Get Azure free trial](#).

Create your Data Science Virtual Machine for Linux

Here are the steps to create an instance of the Data Science Virtual Machine for Linux:

1. Navigate to the virtual machine listing on the [Azure portal](#).
2. Click **Create** (at the bottom) to bring up the wizard.



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. Select an NC-class VM for GPU training.

c. **Settings:**

- **Disk Type:** Choose **Premium** if you prefer a solid-state drive (SSD). Otherwise, choose **Standard**. GPU VMs require a Standard disk.
- **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 5-10 minutes. The status of the provisioning is displayed on the Azure portal.

How to access the Data Science Virtual Machine for Linux

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 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, complete the following procedure 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 Data Science Virtual Machine for Linux

Deep Learning Libraries

CNTK

The Microsoft Cognitive Toolkit is an open source, deep learning toolkit. Python bindings are available in the root and py35 Conda environments. It also has a command-line tool (cntk) that is already in the PATH.

Sample Python notebooks are available in JupyterHub. To run a basic sample at the command-line, execute the following commands in the shell:

```
cd /home/[USERNAME]/notebooks/CNTK/HelloWorld-LogisticRegression
cntk configFile=lr_bs.cntk makeMode=false command=Train
```

For more information, see the CNTK section of [GitHub](#), and the [CNTK wiki](#).

Caffe

Caffe is a deep learning framework from the Berkeley Vision and Learning Center. It is available in /opt/caffe. Examples can be found in /opt/caffe/examples.

Caffe2

Caffe2 is a deep learning framework from Facebook that is built on Caffe. It is available in Python 2.7 in the Conda root environment. To activate it, run the following from the shell:

```
source /anaconda/bin/activate root
```

Some example notebooks are available in JupyterHub.

H2O

H2O is a fast, in-memory, distributed machine learning and predictive analytics platform. A Python package is installed in both the root and py35 Anaconda environments. An R package is also installed. To start H2O from the command-line, run `java -jar /dsvm/tools/h2o/current/h2o.jar`; there are various [command line options](#) that you may like to configure. The Flow Web UI can be accessed by browsing to <http://localhost:54321> to get started. Sample notebooks are also available in JupyterHub.

Keras

Keras is a high-level neural network API in Python that is capable of running on top of either TensorFlow or Theano. It is available in the root and py35 Python environments.

MXNet

MXNet is a deep learning framework designed for both efficiency and flexibility. It has R and Python bindings included on the DSVM. Sample notebooks are included in JupyterHub, and sample code is available in /dsvm/samples/mxnet.

NVIDIA DIGITS

The NVIDIA Deep Learning GPU Training System, known as DIGITS, is a system to simplify common deep learning tasks like managing data, designing and training neural networks on GPU systems, and monitoring performance in real time with advanced visualization.

DIGITS is available as a service, called digits. Start the service and browse to <http://localhost:5000> to get started.

DIGITS is also installed as a Python module in the Conda root environment.

TensorFlow

TensorFlow is Google's deep learning library. It is an open source software library for numerical computation using data flow graphs. TensorFlow is available in the py35 Python environment, and some sample notebooks are included in JupyterHub.

Theano

Theano is a Python library for efficient numerical computation. It is available in the root and py35 Python environments.

Torch

Torch is a scientific computing framework with wide support for machine learning algorithms. It is available in /dsvm/tools/torch, and the th interactive session and luarocks package manager are available at the command line. Examples are available in /dsvm/samples/torch.

PyTorch is also available in the root Anaconda environment. Examples are in /dsvm/samples/pytorch.

Microsoft R Server

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 Server (MRS) with the Microsoft R Open (MRO) and 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. MRS gives you scaling and operationalization of R models into web services. You can edit your R programs in one of the default editors, like RStudio, vi, or Emacs. If you prefer using the Emacs editor, it has been pre-installed. The Emacs package ESS (Emacs Speaks Statistics) simplifies working with R files within the Emacs editor.

To launch R console, 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, and then run the scripts within R. With 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 (conda environment) you want to work on in the current session. The activation process sets the PATH variable to the desired version of Python.

To activate the Python 2.7 conda environment, run the following command from the shell:

```
source /anaconda/bin/activate root
```

Python 2.7 is installed at */anaconda/bin*.

To activate the Python 3.5 conda environment , 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 **pycharm** to launch the PyCharm Python IDE.

To install additional Python libraries, you need to run `conda` or `pip` command under sudo and provide full path of the Python package manager (conda or pip) to install to the correct Python environment. For example:

```
sudo /anaconda/bin/pip install <package> #for Python 2.7 environment  
sudo /anaconda/envs/py35/bin/pip install <package> # for Python 3.5 environment
```

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.

Apache Spark Standalone

A standalone instance of Apache Spark is preinstalled on the Linux DSVM to help you develop Spark applications locally first before testing and deploying on large clusters. You can run PySpark programs through the Jupyter kernel. When you open Jupyter, click the **New** button and you should see a list of available kernels. The "Spark - Python" is the PySpark kernel that lets you build Spark applications using Python language. You can also use a Python IDE like PyCharm or Spyder to build your Spark program. Since, this is a standalone instance, the Spark stack runs within the calling client program. This makes it faster and easier to troubleshoot issues compared to developing on a Spark cluster.

A sample PySpark notebook is provided on Jupyter that you can find in the "SparkML" directory under the home directory of Jupyter (`$HOME/notebooks/SparkML/pySpark`).

If you are programming in R for Spark, you can use Microsoft R Server, SparkR or sparklyr.

Before running in Spark context in Microsoft R Server, you need to do a one time setup step to enable a local single node Hadoop HDFS and Yarn instance. By default, Hadoop services are installed but disabled on the DSVM. In order

to enable it, you need to run the following commands as root the first time:

```
echo -e 'y\n' | ssh-keygen -t rsa -P '' -f ~hadoop/.ssh/id_rsa
cat ~hadoop/.ssh/id_rsa.pub >> ~hadoop/.ssh/authorized_keys
chmod 0600 ~hadoop/.ssh/authorized_keys
chown hadoop:hadoop ~hadoop/.ssh/id_rsa
chown hadoop:hadoop ~hadoop/.ssh/id_rsa.pub
chown hadoop:hadoop ~hadoop/.ssh/authorized_keys
systemctl start hadoop-namenode hadoop-datanode hadoop-yarn
```

You can stop the Hadoop related services when you dont need them by running

```
systemctl stop hadoop-namenode hadoop-datanode hadoop-yarn
```

A sample demonstrating how to develop and test MRS in remote Spark context (which is the standalone Spark instance on the DSVM) is provided and available in the `/dsvm/samples/MRS` directory.

IDEs and editors

You have a choice of several code editors. This includes vi/VIM, Emacs, PyCharm, RStudio, and IntelliJ. IntelliJ, RStudio and PyCharm 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](#).

LaTex is installed through the texlive package along with an Emacs add-on [auctex](#) package, which simplifies authoring your LaTex documents within Emacs.

Databases

Graphical SQL client

SQuirrel SQL, a graphical SQL client, has been provided to connect to different databases (such as Microsoft SQL Server, 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.

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 Cosmos DB (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, Azure Cosmos DB, 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:

- **Vowpal Wabbit**: A fast online learning algorithm.
- **xgboost**: A tool that provides optimized, boosted tree algorithms.
- **Rattle**: An R-based graphical tool for easy data exploration and modeling.
- **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.
- **LightGBM**: A fast, distributed, high-performance gradient boosting framework based on decision tree algorithms.
- **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.

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 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** **A**nalytical **T**ool **T**o **L**earn **E**asily) 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 Data Science Virtual Machine for Linux](#) 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.

Provision a Linux CentOS Data Science Virtual Machine on Azure

9/25/2017 • 21 min to read • [Edit Online](#)

The Linux Data Science Virtual Machine is a CentOS-based 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:

- Operating System: Linux CentOS distribution.
- Microsoft R Server Developer Edition
- Anaconda Python distribution (versions 2.7 and 3.5), including popular data analysis libraries
- JuliaPro - a curated distribution of Julia language with popular scientific and data analytics libraries
- Standalone Spark instance and single node Hadoop (HDFS, Yarn)
- JupyterHub - a multiuser Jupyter notebook server supporting R, Python, PySpark, Julia kernels
- Azure Storage Explorer
- Azure command-line interface (CLI) for managing Azure resources
- PostgreSQL Database
- Machine learning tools
 - [Cognitive Toolkit](#): 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 (RStudio, PyCharm, IntelliJ, 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](#).

Other Versions of the Data Science Virtual Machine

An [Ubuntu](#) image is also available, with many of the same tools as the CentOS image plus deep learning frameworks. A [Windows](#) image is available as well.

Prerequisites

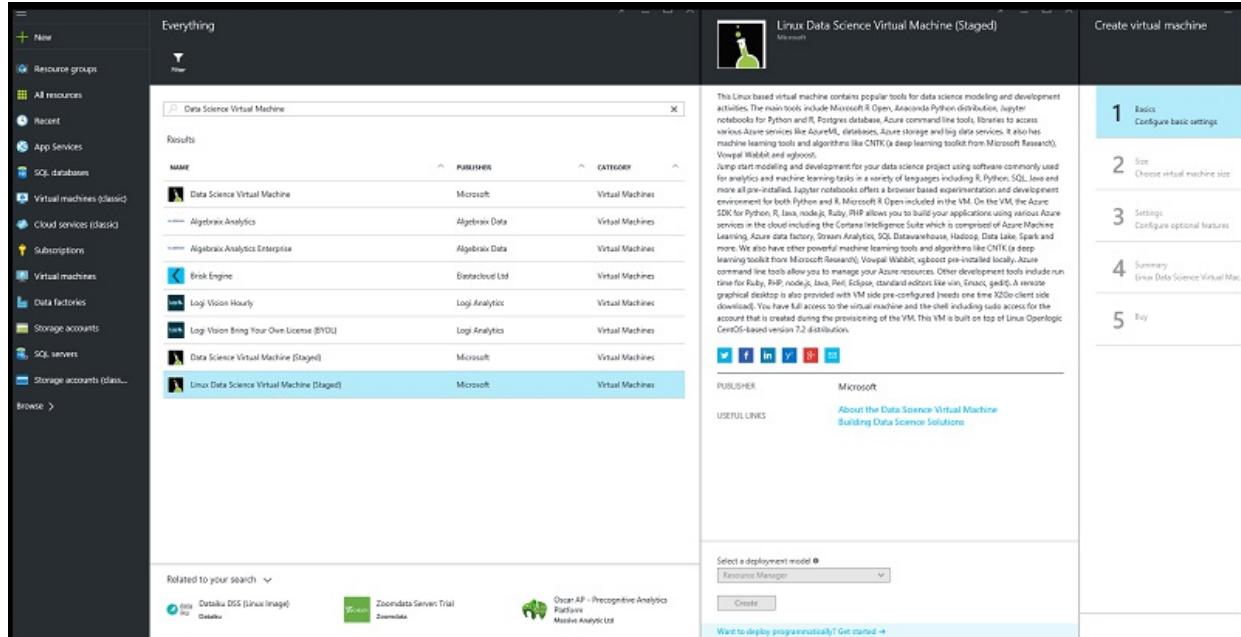
Before you can create a Linux Data Science Virtual Machine, you must have the following:

- **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, if you do not want to use an existing account, the storage account can be created as part of the process of creating the VM.

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.



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:** If you don't need to use sound support and client printing, you can turn them off.
 - **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 Server

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 Server (MRS) with the Microsoft R Open (MRO) and 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. MRS gives you scaling and operationalization of R models into web services. You can edit your R programs in one of the default editors, like RStudio, vi, Emacs, or gedit. 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 console, 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. With 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 (conda environment) you want to work on in the current session. The activation process sets the PATH variable to the desired version of Python.

To activate the Python 2.7 conda environment, run the following command from the shell:

```
source /anaconda/bin/activate root
```

Python 2.7 is installed at */anaconda/bin*.

To activate the Python 3.5 conda environment, 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 **pycharm** to launch the PyCharm Python IDE.

To install additional Python libraries, you need to run **conda** or **pip** command under sudo and provide full path of the Python package manager (conda or pip) to install to the correct Python environment. For example:

```
sudo /anaconda/bin/pip install <package> #for Python 2.7 environment  
sudo /anaconda/envs/py35/bin/pip install <package> # for Python 3.5 environment
```

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.

Apache Spark Standalone

A standalone instance of Apache Spark is preinstalled on the Linux DSVM to help you develop Spark applications locally first before testing and deploying on large clusters. You can run PySpark programs through the Jupyter kernel. When you open Jupyter and click the **New** button, you should see a list of available kernels. The "Spark - Python" is the PySpark kernel that lets you build Spark applications using Python language. You can also use a Python IDE like PyCharm or Spyder to build your Spark program. Since, this is a standalone instance, the Spark stack runs within the calling client program. This makes it faster and easier to troubleshoot issues compared to developing on a Spark cluster.

A sample PySpark notebook is provided on Jupyter that you can find in the "SparkML" directory under the home directory of Jupyter (`$HOME/notebooks/SparkML/pySpark`).

If you are programming in R for Spark, you can use Microsoft R Server, SparkR or sparklyr.

Before running in Spark context in Microsoft R Server, you need to do a one time setup step to enable a local single node Hadoop HDFS and Yarn instance. By default, Hadoop services are installed but disabled on the DSVM. In order to enable it, you need to run the following commands as root the first time:

```
echo -e 'y\n' | ssh-keygen -t rsa -P '' -f ~hadoop/.ssh/id_rsa
cat ~hadoop/.ssh/id_rsa.pub >> ~hadoop/.ssh/authorized_keys
chmod 0600 ~hadoop/.ssh/authorized_keys
chown hadoop:hadoop ~hadoop/.ssh/id_rsa
chown hadoop:hadoop ~hadoop/.ssh/id_rsa.pub
chown hadoop:hadoop ~hadoop/.ssh/authorized_keys
systemctl start hadoop-namenode hadoop-datanode hadoop-yarn
```

You can stop the Hadoop related services when you don't need them by running

```
systemctl stop hadoop-namenode hadoop-datanode hadoop-yarn
```

A sample demonstrating how to develop and test MRS in remote Spark context (which is the standalone Spark instance on the DSVM) is provided and available in the `/dsvm/samples/MRS` directory.

IDEs and editors

You have a choice of several code editors. This includes vi/VIM, Emacs, gEdit, PyCharm, RStudio, Eclipse, and IntelliJ. gEdit, Eclipse, IntelliJ, RStudio and PyCharm 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 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 Cosmos DB (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, Azure Cosmos DB, 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:

- **Microsoft Cognitive Toolkit**: 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.

Microsoft Cognitive Toolkit

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:

```
cd /home/[USERNAME]/notebooks/CNTK/HelloWorld-LogisticRegression  
cntk configFile=lr_bs.cntk makeMode=false command=Train
```

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 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** **A**nalytical **T**ool **T**o **L**earn **E**asily) 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.

Provision a Deep Learning Virtual Machine on Azure

9/25/2017 • 4 min to read • [Edit Online](#)

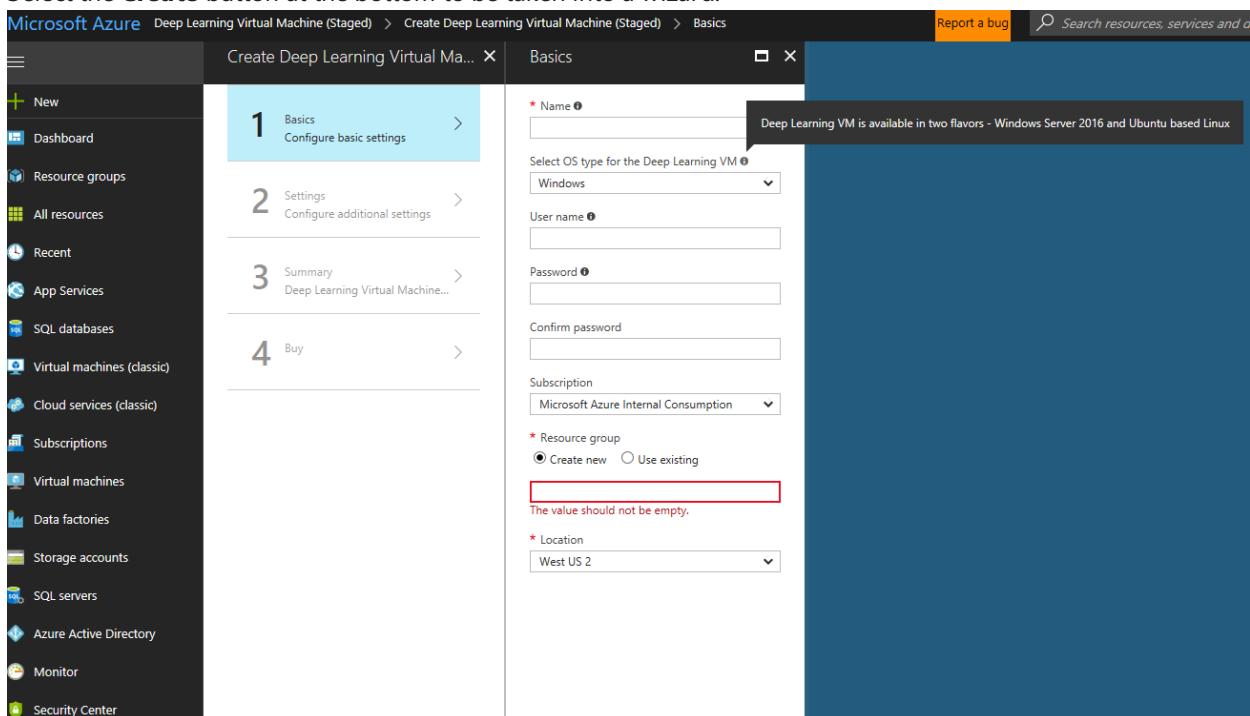
The Deep Learning Virtual Machine (DLVM) is a specially configured variant of the popular [Data Science Virtual Machine](#) (DSVM) to make it easier to use GPU-based VM instances for rapidly training deep learning models. It is supported with either Windows 2016, or the Ubuntu DSVM as the base. The DLVM shares the same core VM images and hence all the rich toolset that is available on DSVM.

The DLVM contains several tools for AI including GPU editions of popular deep learning frameworks like Microsoft Cognitive Toolkit, TensorFlow, Keras, Caffe2, Chainer; tools to acquire and pre-process image, textual data, tools for data science modeling and development activities such as Microsoft R Server Developer Edition, Anaconda Python, Jupyter notebooks for Python and R, IDEs for Python and R, SQL databases and many other data science and ML tools.

Create your Deep Learning Virtual Machine

Here are the steps to create an instance of the Deep Learning 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 DLVM requires **inputs** for each of the **four steps** enumerated on the right of this figure. Here are the inputs needed to configure each of these steps:

a. Basics

- a. **Name:** Name of your data science server you are creating.
- b. **Select OS type for the Deep Learning VM:** Choose Windows or Linux (for Windows 2016 and Ubuntu Linux base DSVM)
- c. **User Name:** Admin account login id.
- d. **Password:** Admin account password.
- e. **Subscription:** If you have more than one subscription, select the one on which the machine is to be created and billed.

- f. **Resource Group:** You can create a new one or use an **empty** existing Azure resource group in your subscription.
- g. **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.

NOTE

Since DLVM is provisioned on Azure NC-Series GPU VM instances, you must choose one of the locations in Azure that has GPUs. Currently the locations that have GPU VMs are: **East US, North Central US, South Central US, West US 2, North Europe, West Europe**. For the latest list, check the [Azure Products by Region Page](#) and look for **NC-Series** under **Compute**.

1. **Settings:** Select one of the NC-Series GPU virtual machine size that meets your functional requirement and cost constraints. Create a storage account for your VM.

Size	vCPUs	Memory (GB)	Data Disks	Graphics
NC6 Standard	6	56	8 Data disks 8x500 GB Local SSD	1x K80 Graphics
NC12 Standard	12	112	16 Data disks 16x500 GB Local SSD	2x K80 Graphics
NC24 Standard	24	224	32 Data disks 32x500 GB Local SSD	4x K80 Graphics

2. **Summary:** Verify that all information you entered is correct.
3. **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 Deep Learning Virtual Machine

Windows Edition

Once the VM is created, you can remote desktop into it using the Admin account credentials that you configured in the preceding **Basics** section.

Linux Edition

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 a Windows client, 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 better than X11 forwarding in our testing. We recommend using the X2Go client for a graphical desktop interface.

Installing and configuring X2Go client

The Linux DLVM is already provisioned with X2Go server and ready to accept client connections. To connect to the Linux VM graphical desktop, complete the following procedure 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 DSVM 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.

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 identify scenarios and plan for advanced analytics data processing

11/13/2017 • 4 min to read • [Edit Online](#)

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 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 requires 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

You can try Azure Machine Learning for free. No credit card or Azure subscription is required. [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 articles:

- [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-premises 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-premises SQL server to SQL Azure with Azure Data Factory](#)

5. **How much of the data is to be moved to Azure?** Datasets that are extremely 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 2 GB
- Medium: Greater than 2 GB and less than 10 GB
- Large: Greater than 10 GB

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 to gain an understand of 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](#).

Load data into storage environments for analytics

11/10/2017 • 1 min to read • [Edit Online](#)

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

11/9/2017 • 1 min to read • [Edit Online](#)

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

11/9/2017 • 2 min to read • [Edit Online](#)

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.

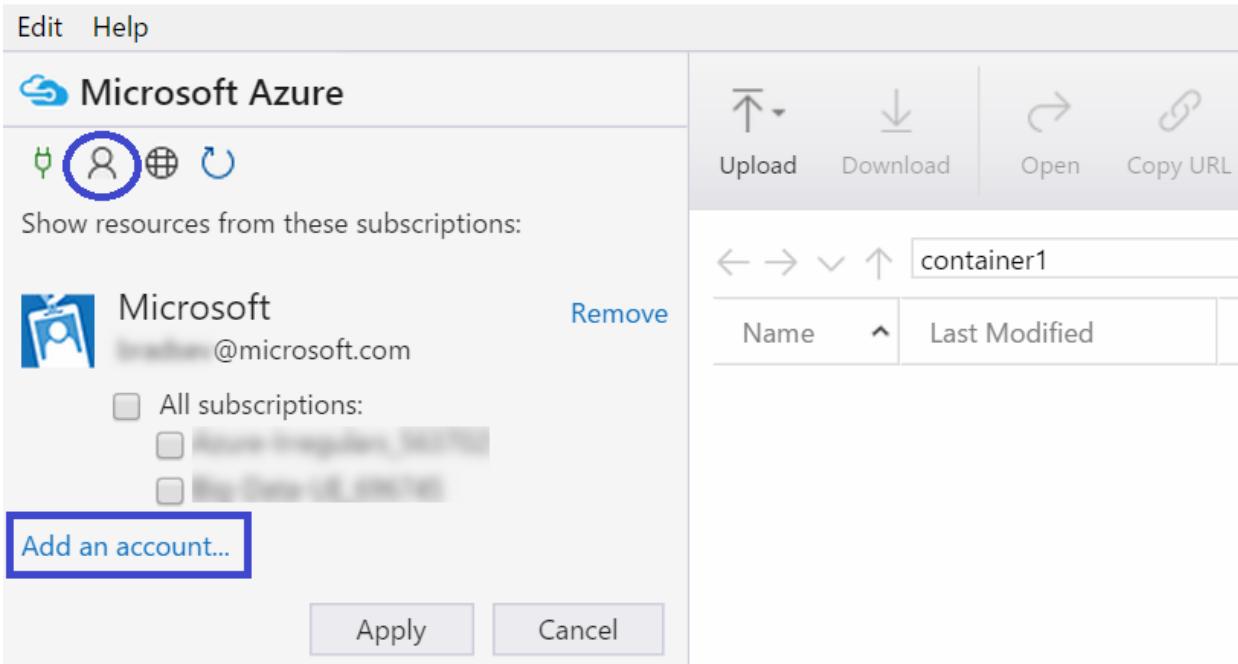
- To set up an Azure subscription, see [Free one-month trial](#).
- For instructions on creating a storage account and for getting account and key information, see [About Azure storage accounts](#). Make a note the access key for your storage account as you need this key to connect to the account with 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

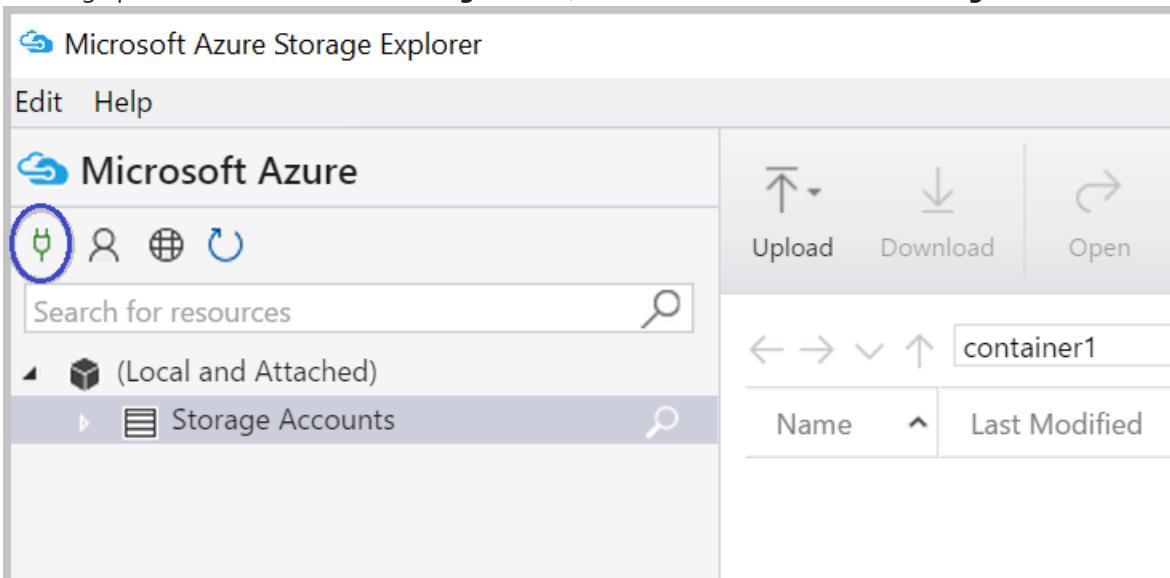
The following steps document how to upload/download data using Azure Storage Explorer.

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



3. To bring up the **Connect to Azure Storage** wizard, select the **Connect to Azure storage** icon.



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

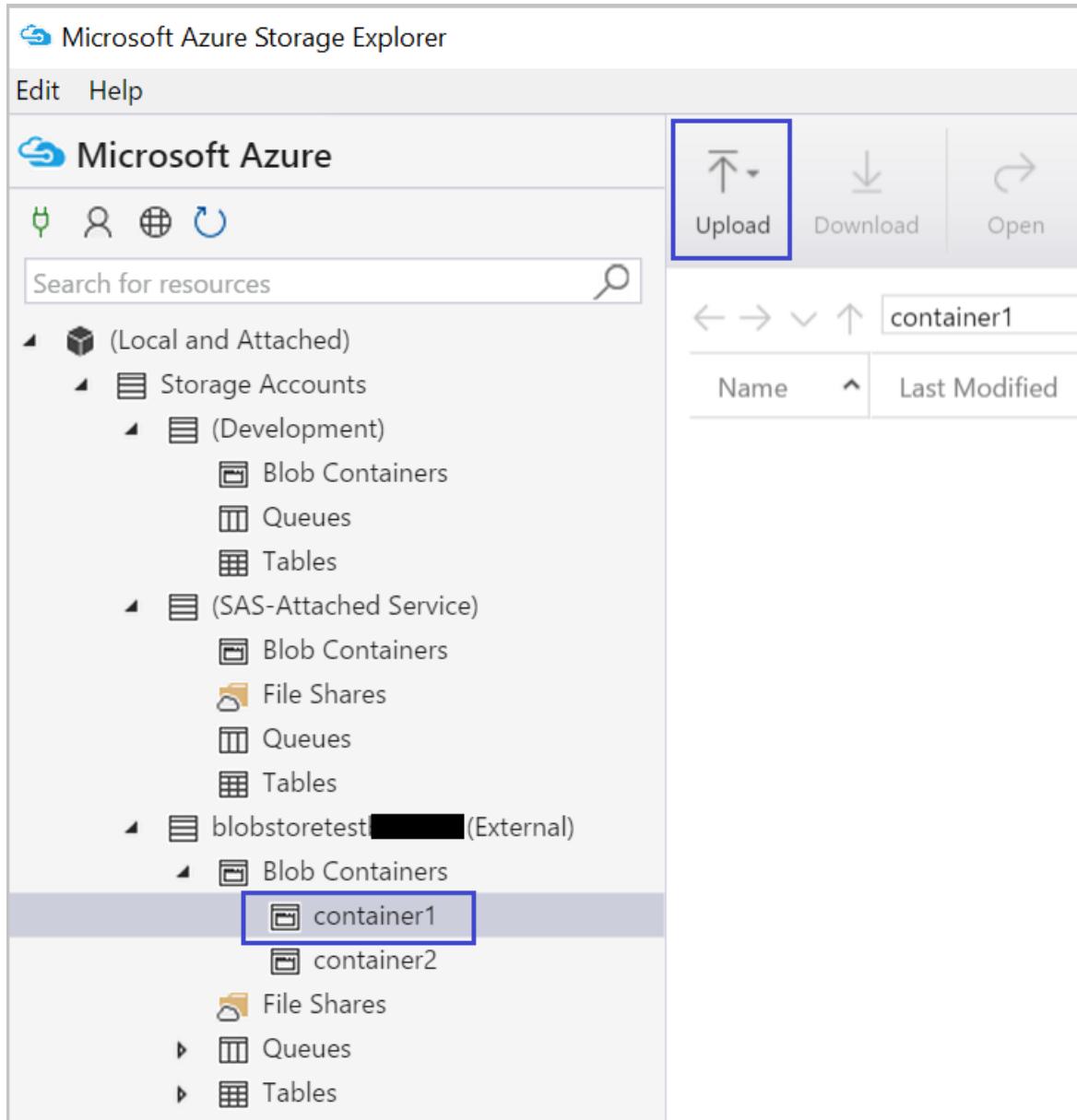
Next

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's 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, there's a toolbar with buttons for Upload, Download (which is highlighted with a blue box), Open, Copy URL, Select all, and Copy. Below the toolbar, a list of files in 'container1' is shown, with 'trip_data_1.csv.zip' selected and highlighted with a blue box. The file details show it was last modified on Wednesday, 31 Aug 2016 at 14:31:13 GMT.

Name	Last Modified
trip_data_1.csv.zip	Wed, 31 Aug 2016 14:31:13 GMT

Move data to and from Azure Blob Storage using AzCopy

11/9/2017 • 2 min to read • [Edit Online](#)

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.

4 min to read •

Move data to or from Azure Blob Storage using SSIS connectors

11/9/2017 • 3 min to read • [Edit Online](#)

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](#).

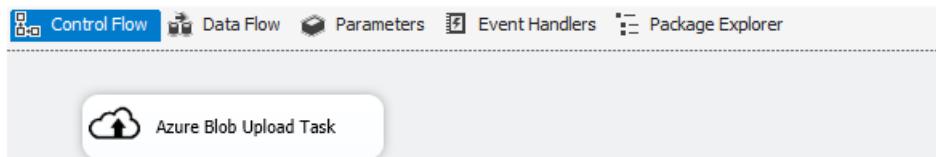
Download NYC Taxi dataset

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*.xlsx* includes files such as MySheet001.xlsx 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-premises 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-premises 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

11/9/2017 • 8 min to read • [Edit Online](#)

This topic outlines the options for moving data either from flat files (CSV or TSV formats) or from an on-premises 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 -x -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_attemp -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
)

```

Built-in Utilities in SQL Server

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
```

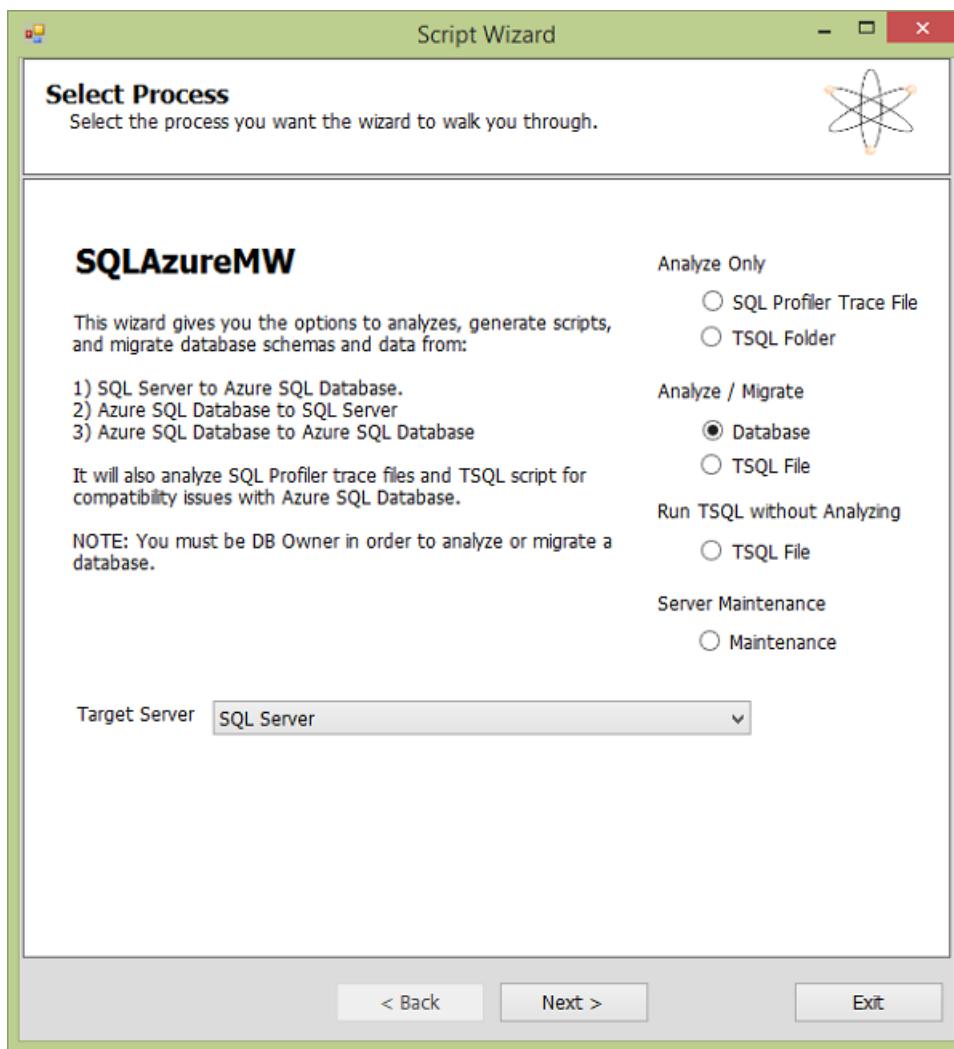
Format file generation when running BCP remotely against a SQL Server

```
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.

SQlL Server Management Studio window showing a database named 'disprocess' selected in Object Explorer. The Tasks menu is open, showing various options like Detach, Take Offline, Bring Online, Shrink, Back Up, Restore, Mirror, etc. A large table is displayed in the main pane, showing data from a temporary table 'tmp.asp'. The table has columns: skid_id, skid_lanularity, skid_latitude, skid_longitude, skid_location, skid_tmid, and skid_VDago. The data includes rows for locations like San Francisco, CA; California - LA; Mexico; Memphis, TN; New Jersey; Charleston, SC; San Francisco, CA; London; and a row with NULL values.

skid_id	skid_lanularity	skid_latitude	skid_longitude	skid_location	skid_tmid	skid_VDago
0.629902515495	0.461316337541	APRNUJY112598800C91	0	NULL	6aa6a01f-91d7-46cc-ba74-5e6e5d20c54	NULL
0.67170530534	0.386606364066	APR73M011870940578	37.77916	San Francisco, CA	-122.42005	911066462638-4ab9-9d34069644b453
0.581793765845	0.401995453884	APD77NE1187B939FB1	0	California - LA	0	e77e51a4-4781-45d3-9847-2051811e366
0.476943791297	0.307060168813	APL75QJ1187F935EFE	23.62574	Mexico	-101.95625	0163c840-4b3d-93d3-9d62-9956a9d04344
0.63063000759	0.417495644971	APRNUAH1187FB46P3	35.16988	Memphis, TN	-90.04892	1c7ba6b6-d334-4333-8d0b-7e8d118492
0.0131579730569	0.379262747332	APR909C1187F956098	0	New Jersey	0	c0129404-7814-4418-9d57-c3e85f707081
0.531334211069	0.3382691594	APG909811187B990479	64.56663	Charleston, SC	12.66938	c795a5a8-7327-4375-b389-88614116027
0.627827825331	0.387516855902	APC7361187B99428D	37.77916	San Francisco, CA	-122.42005	2e753be2-25d4-4409-99e4-6999e0592e1
0.42179593845	0.336655666667	APRQZWT1187B9900F	51.50632	London	-0.12714	c1632a11-2c7e-44d4-a4f6-b05ed680055
0.487056793939	0.343426978297	APRUR1F1187B998404	0	NULL	0	7e273984-e593-4451-9c4d-3983805ebcd

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

11/9/2017 • 4 min to read • [Edit Online](#)

This topic outlines the options for moving data either from flat files (CSV or TSV formats) or from data stored in an on-premises 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-premises 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-premises 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-premises 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-premises 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-premises SQL Server to an Azure SQL database

If the source data is stored in an on-premises 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-premises SQL server to SQL Azure with Azure Data Factory](#). This topic shows how to move data from an on-premises 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-premises 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 Hive tables and load data from Azure Blob Storage

2/28/2018 • 10 min to read • [Edit Online](#)

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.

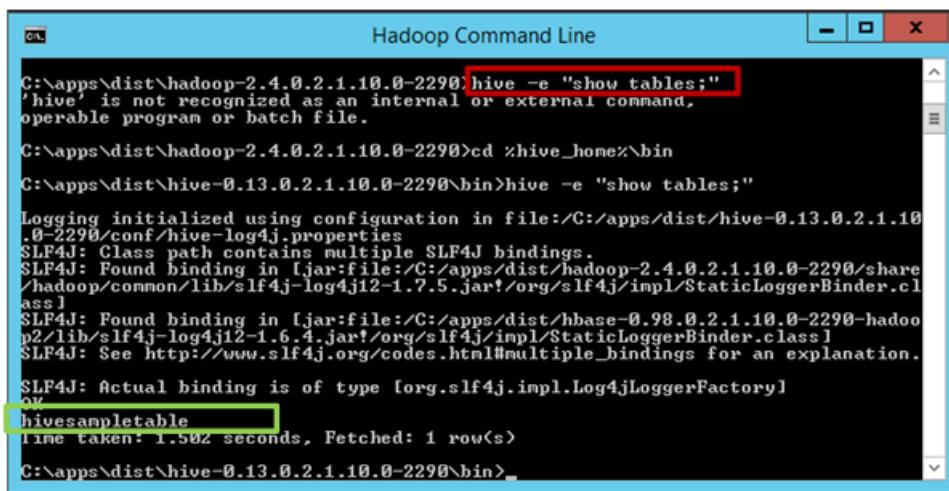
Log in to the head node of the Hadoop cluster, open the Hadoop Command Line on the desktop of the head node, and enter command `cd %hive_home%\bin`.

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]

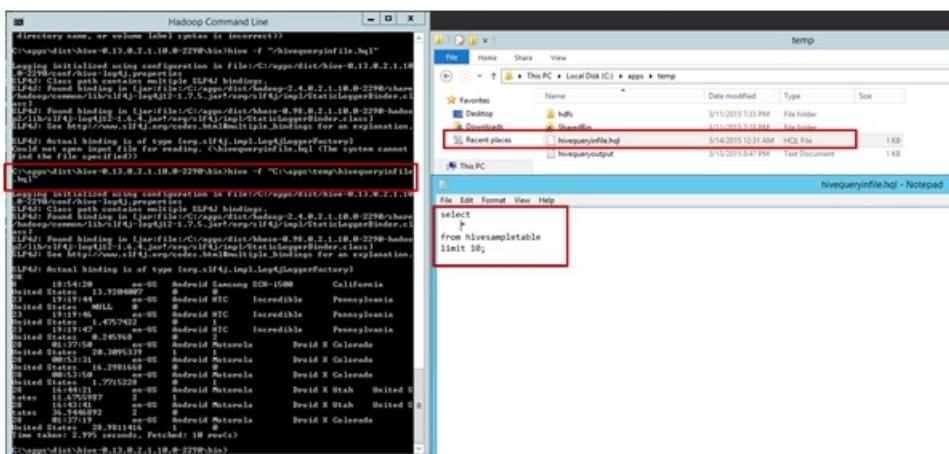
```
hivequeryoutput
Time taken: 1.502 seconds, Fetched: 1 row(s)
```

```
C:\>
```

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>"
```



```
cd %hive_home%\bin
hive -f "C:\app\dist\hivequeryfile.hql"
```

Logging initialized using configuration in file:/C:/App/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:/app/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:/app/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]

```
hivequeryoutput
Time taken: 1.502 seconds, Fetched: 1 row(s)
```

```
C:\>
```

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
hivesampletable
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 hivesampletable limit 3;
+-----+-----+-----+-----+-----+
| id | ts |          |          |          |
|-----+-----+-----+-----+-----+
| 8  | 18:54:20 | en-US | Android | Samsung SGH-i500 | California
| United States | 13.9204007 | 0 | 0 |
| 23 | 19:19:44 | en-US | Android | HTC      | Incredibile
| United States | NULL    | 0 | 0 |
| 23 | 19:19:46 | en-US | Android | HTC      | Incredibile
| United States | 1.4757422 | 0 | 1 |
+-----+-----+-----+-----+-----+
Time taken: 2.897 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
C:\apps\dist\hive-0.13.0.2.1.10.0-2290\bin>hive -e "select * from hivesampletable limit 10" > C:\apps\temp\hivequeryoutput.txt
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 hivesampletable limit 10;
+-----+-----+-----+-----+-----+
| id | ts |          |          |          |
|-----+-----+-----+-----+-----+
| 8  | 18:54:20 | en-US | Android | Samsung SGH-i500 | California
| United States | 13.9204007 | 0 | 0 |
| 23 | 19:19:44 | en-US | Android | HTC      | Incredibile
| United States | NULL    | 0 | 0 |
| 23 | 19:19:46 | en-US | Android | HTC      | Incredibile
| United States | 1.4757422 | 0 | 1 |
+-----+-----+-----+-----+-----+
Time taken: 2.897 seconds, Fetched: 3 rows(s)
C:\apps\dist\hive-0.13.0.2.1.10.0-2290\bin>hive -e "select * from hivesampletable limit 10" > C:\apps\temp\hivequeryoutput.txt
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 hivesampletable limit 10;
+-----+-----+-----+-----+-----+
| id | ts |          |          |          |
|-----+-----+-----+-----+-----+
| 8  | 18:54:20 | en-US | Android | Samsung SGH-i500 | California
| United States | 13.9204007 | 0 | 0 |
| 23 | 19:19:44 | en-US | Android | HTC      | Incredibile
| United States | NULL    | 0 | 0 |
| 23 | 19:19:46 | en-US | Android | HTC      | Incredibile
| United States | 1.4757422 | 0 | 1 |
+-----+-----+-----+-----+-----+
Time taken: 25.041 seconds, Fetched: 10 rows(s)
C:\apps\dist\hive-0.13.0.2.1.10.0-2290\bin>
```

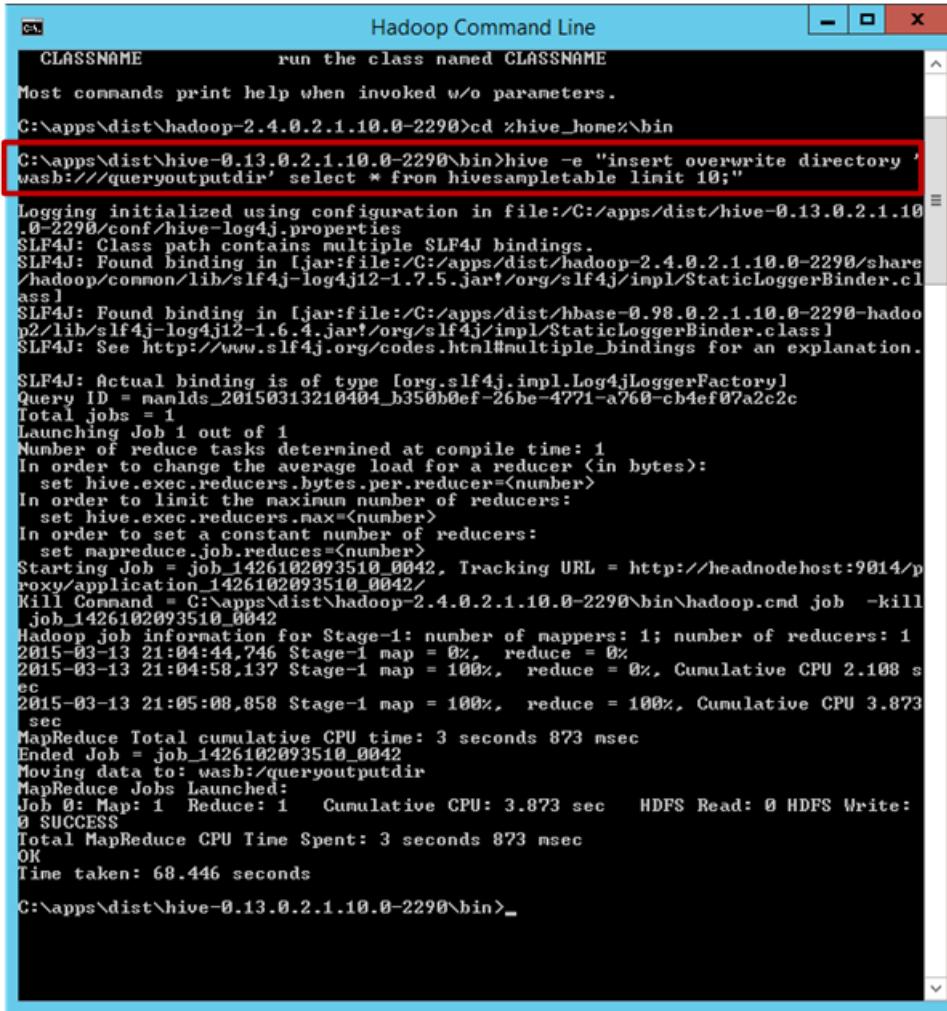
Output Hive query results to an Azure blob

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`.

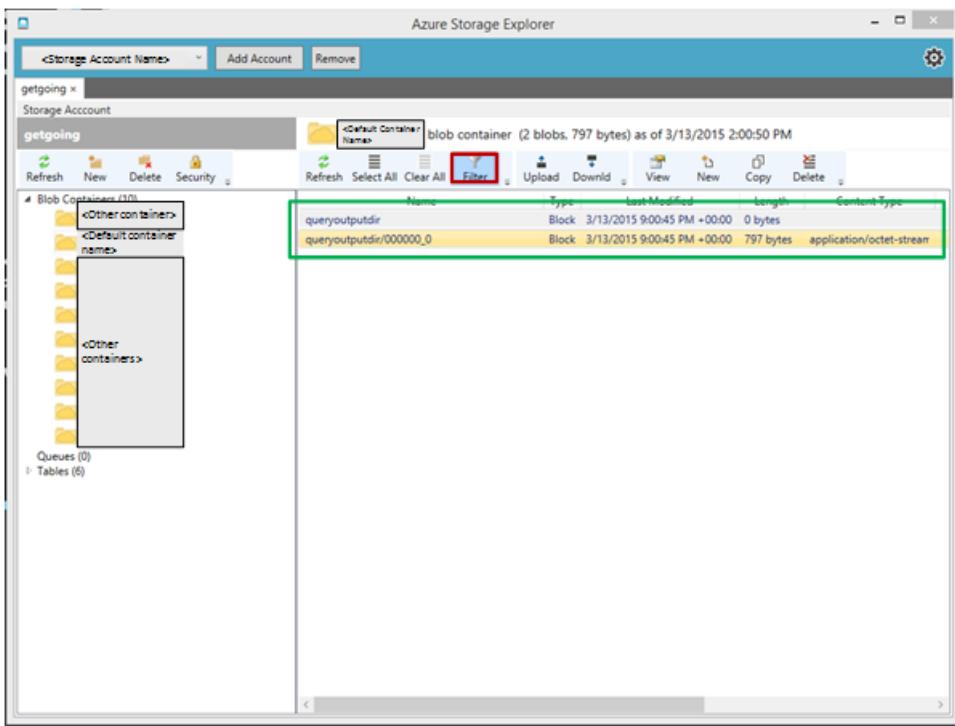


The screenshot shows a terminal window titled "Hadoop Command Line". The command entered is:

```
C:\apps\dist\hive-0.13.0.2.1.10.0-2290\bin>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 the configuration of the job, the execution of the MapReduce job, and the final success message. A red box highlights the command line where the directory path is specified.

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://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:

- : 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.
- : 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 without .

- `:` the separator that delimits fields in the data file to be uploaded to the Hive table.
- `:` the separator that delimits lines in the data file.
- `:` the Azure storage location to save the data of Hive tables. If you do not specify `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>;
```

- `:` If the blob file to be uploaded to the Hive table is in the default container of the HDInsight Hadoop cluster, the should be in the format '`wasb:///`'. The blob file can also be in an additional container of the HDInsight Hadoop cluster. In this case, should be in the format '`wasb://blob.core.windows.net/`'.

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 . 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 . fails since . 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 . 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 when using the following query after all data has been inserted into .:

```
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-premises SQL server to SQL Azure with Azure Data Factory

2/21/2018 • 8 min to read • [Edit Online](#)

This topic shows how to move data from an on-premises SQL Server Database to a SQL Azure Database via Azure Blob Storage using the Azure Data Factory (ADF).

For a table that summarizes various options for moving data to an Azure SQL Database, see [Move data to an Azure SQL Database for Azure Machine Learning](#).

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-premises 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-premises SQL database and an Azure SQL Database in the cloud. The two activities are:

- copy data from an on-premises 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-premises 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-premises 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-premises 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*.

Install and configure up the Data Management Gateway

To enable your pipelines in an Azure data factory to work with an on-premises SQL Server, you need to add it as a Linked Service to the data factory. To create a Linked Service for an on-premises SQL Server, you must:

- download and install Microsoft Data Management Gateway onto the on-premises 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. We have three resources in this scenario for which linked services are needed:

1. On-premises SQL Server
2. Azure Blob Storage
3. Azure SQL database

The step-by-step procedure for creating linked services is provided in [Create linked services](#).

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-premises SQL server is *nyctaxi_data*
- the **container name** in the Azure Blob Storage account is *containernamespace*

Three table definitions are needed for this ADF pipeline:

1. [SQL on-premises 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-premises Table

The table definition for the on-premises 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 AFDsp -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-premises 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 adfdsprg -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": "AMLDSPipeline",
  "properties":
  {
    "description" : "This pipeline has one Copy activity that copies data from an on-premises SQL to  
Azure blob",
    "activities":
    [
      {
        "name": "CopyFromSQLtoBlob",
        "description": "Copy data from on-premises SQL server to blob",
        "type": "CopyActivity",
        "inputs": [ {"name": "OnPremSQLTable"} ],
        "outputs": [ {"name": "OutputBlobTable"} ],
        "script": "copyFromSQLToBlob"
      }
    ]
  }
}
```

```

    "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

```

Start the Pipeline

The pipeline can now be run using the following command:

```
Set-AzureDataFactoryPipelineActivePeriod -ResourceGroupName ADFdsprg -DataFactoryName ADFdsp -StartTime startdateZ -EndTime enddateZ -Name AMLDSPProcessPipeline
```

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

11/10/2017 • 5 min to read • [Edit Online](#)

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 already.
- Add database filegroups to the database, which holds 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 is 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

To create partitioned table(s) according to the data schema, mapped to the database filegroups created in the previous step, you must first create a partition function and scheme. When data is bulk imported to the partitioned table(s), records are distributed among the filegroups according to a partition scheme, as described below.

1. Create a partition function

[Create a partition function](#) This function defines the range of values/boundaries to be included in each individual partition table, for example, 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' )
```

2. Create a partition scheme

[Create a partition scheme](#). This scheme maps each partition range in the partition function to a physical filegroup, for example:

```
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,
       prng.value AS PartitionValue,
       prng.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 prng ON prng.function_id=pfun.function_id
 WHERE pfun.name = <DatetimeFieldPFN>
```

3. Create a partition table

[Create partitioned table](#)(s) according to your data schema, and specify the partition scheme and constraint field used to partition the table, for example:

```
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, for example:

```
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
$tbname = "<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, $tbname, $basename, $fmtfile, $indir, $logdir, $num)
    bcp ($dbname + ".." + $tbname) in ($indir + "\" + $basename + "_" + $num + ".csv") -o ($logdir + "\" +
    $tbname + "_" + $num + ".txt") -h "TABLOCK" -F 2 -C "RAW" -f ($fmtfile) -T -b 2500 -t "," -r \n
}

# BCP example using SQL authentication
$ScriptBlock2 = {
    param($dbname, $tbname, $basename, $fmtfile, $indir, $logdir, $num, $sqlusr, $server, $pass)
    bcp ($dbname + ".." + $tbname) in ($indir + "\" + $basename + "_" + $num + ".csv") -o ($logdir + "\" +
    $tbname + "_" + $num + ".txt") -h "TABLOCK" -F 2 -C "RAW" -f ($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, $tbname, $basename, $fmtfile, $indir, $logdir, $i)
    }
    else {
        # Use SQL authentication
        Start-Job -ScriptBlock $ScriptBlock2 -Arg ($dbname, $tbname, $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 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 example:

```
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 slows down the data loading.

Advanced Analytics Process and Technology in Action Example

For an end-to-end walkthrough example using the Team Data Science Process with a public dataset, see [Team Data Science Process in Action: using SQL Server](#).

Tasks to prepare data for enhanced machine learning

11/10/2017 • 6 min to read • [Edit Online](#)

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

11/10/2017 • 1 min to read • [Edit Online](#)

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

11/10/2017 • 2 min to read • [Edit Online](#)

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 .

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, ':', 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

11/10/2017 • 2 min to read • [Edit Online](#)

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 <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>
```

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 (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 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](#).

The Team Data Science 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

11/10/2017 • 1 min to read • [Edit Online](#)

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

2/2/2018 • 1 min to read • [Edit Online](#)

This document links to articles that cover 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

2/2/2018 • 1 min to read • [Edit Online](#)

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 `random.choice` 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

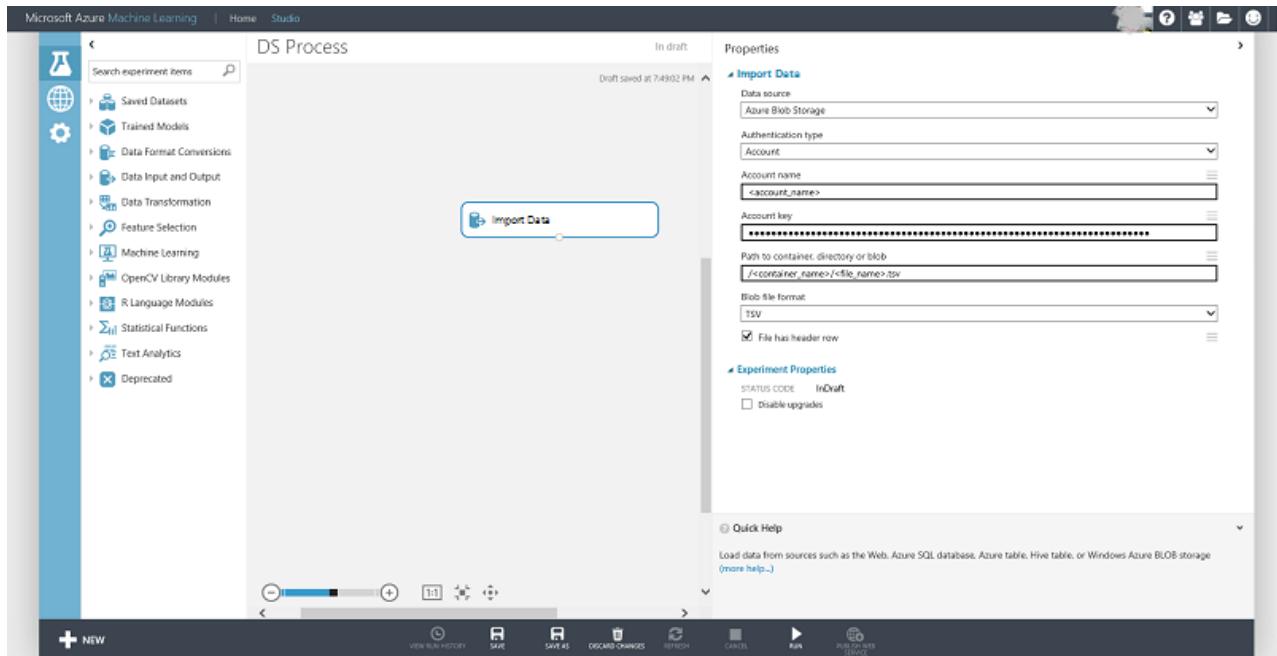
STORAGEACCOUNTNAME= <storage_account_name>
LOCALFILENAME= <local_file_name>
STORAGEACCOUNTKEY= <storage_account_key>
CONTAINERNAME= <container_name>
BLOBNAME= <blob_name>

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

2/2/2018 • 3 min to read • [Edit Online](#)

This article 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, refer to [Move data to SQL Server on Azure](#) article for instructions on how to move your data to SQL Server on Azure.

The following **menu** links to articles 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. Choose a method based on your data size and its distribution.

The following two items 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 following sample code 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 the data as well. 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.

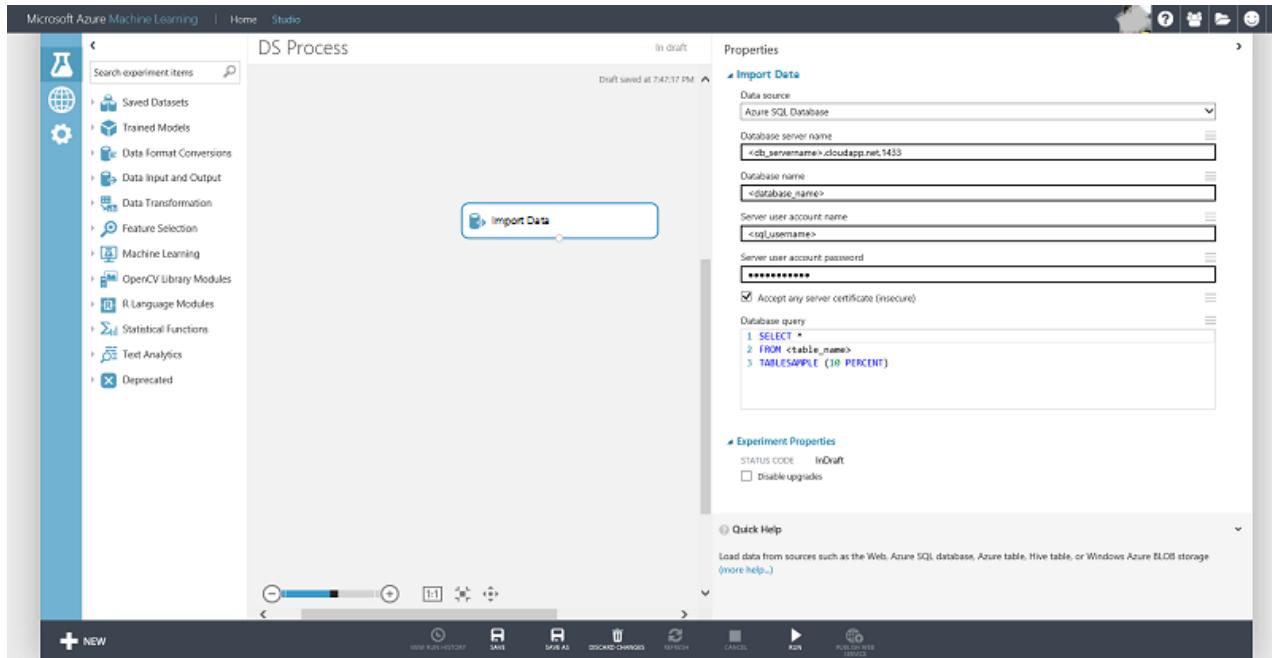
```
SELECT *  
FROM <table_name>  
TABLESAMPLE (10 PERCENT)
```

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 here:



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 following code reads a 0.1% sample of the data from a table in Azure SQL database into a Pandas data frame:

```
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

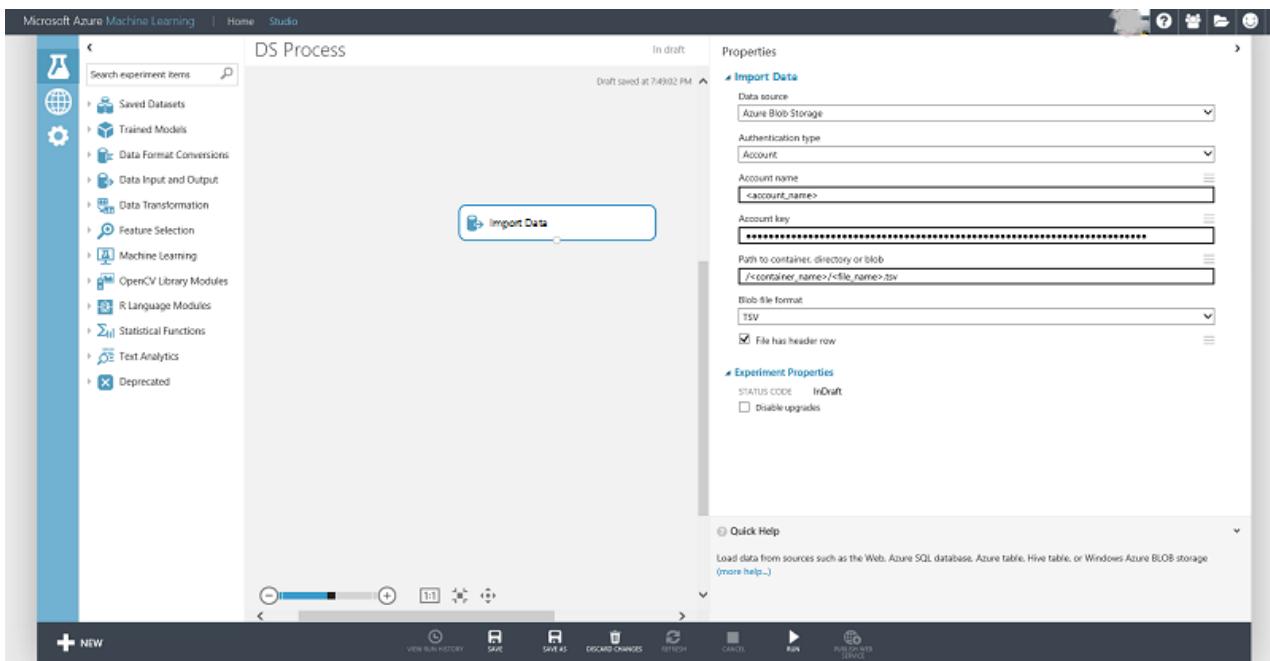
STORAGEACCOUNTNAME= <storage_account_name>
LOCALFILENAME= <local_file_name>
STORAGEACCOUNTKEY= <storage_account_key>
CONTAINERNAME= <container_name>
BLOBNAME= <blob_name>

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 following screen grab:



The Team Data Science Process in Action example

To walkthrough an example of the Team Data Science Process a using a public dataset, see [Team Data Science Process in Action: using SQL Server](#).

Sample data in Azure HDInsight Hive tables

11/13/2017 • 2 min to read • [Edit Online](#)

This article describes how to down-sample data stored in Azure HDInsight Hive tables using Hive queries to reduce it to a size more manageable for analysis. It covers 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. Down-sampling 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. It 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 for some value of the categorical variable. This sort of sampling is called "sampling by group". For example, if you have a categorical variable "State", which has values such as NY, MA, CA, NJ, and PA, you want records from each state to be 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 categorical values that are present in the same ratio as they were in the parent population. Using the same example as above, suppose your data has the following observations by states: 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

11/14/2017 • 8 min to read • [Edit Online](#)

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

You can try Azure Machine Learning for free. No credit card or Azure subscription is required. [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 red circle. The main area has a title 'settings' and tabs for NAME, AUTHORIZATION TOKENS, and USERS. The USERS tab is selected. It shows a table with columns: NAME, EMAIL, ROLE, and STATUS. One row is visible, showing a blacked-out name, a blacked-out email, 'Owner' in the ROLE column, and 'Active' in the STATUS column. There's also a magnifying glass icon for filtering.

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 authorization 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 'AUTHORIZATION TOKENS' tab selected. The left sidebar shows the SETTINGS icon circled in red. The main area has a title 'settings' and tabs for NAME, AUTHORIZATION TOKENS, and USERS. The AUTHORIZATION TOKENS tab is selected. It shows two sections: 'PRIMARY AUTHORIZATION TOKEN' and 'SECONDARY AUTHORIZATION TOKEN'. Each section has a blacked-out token value and a 'Regenerate' button. The entire row for the secondary token is also 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 sidebar with icons for EXPERIMENTS, WEB SERVICES, MODULES, DATASETS (which is selected), TRAINED MODELS, and SETTINGS. The main area is titled "datasets" and shows a table of datasets. The table has columns: NAME, SUBMITTED BY, DESCRIPTION, DATA TYPE, and CREATION DATE. One row is visible: "My Data.tsv", "ptvsazure", "My Test Data", "GenericTSV", and "1/20/2015 12:3...". At the bottom of the page is a toolbar with buttons for NEW, DOWNLOAD, DELETE, and GENERATE DATA ACCESS CODE. The "GENERATE DATA ACCESS CODE" button is highlighted with a red box.

4. Select the code snippet from the window that appears and copy it to your clipboard.

The screenshot shows the "GENERATE DATA ACCESS CODE" dialog box. It contains instructions to "Use this code to access your data" and notes that the code includes a workspace access token. A "CODE SNIPPET" section shows Python code for connecting to a workspace and retrieving a dataset. The code is as follows:

```
from azureml import Workspace
ws = Workspace(
    workspace_id='[REDACTED]',
    authorization_token='[REDACTED]')
ds = ws.datasets['My Data.tsv']
frame = ds.to_dataframe()
```

Below the code, there is a checkbox for "USE SECONDARY TOKEN" and a checkmark icon.

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 from the "My Data.tsv" dataset. The columns are labeled fLength, fWidth, fSize, fConc, fConcl, fAsym, fM3Long, fM3Trans, fAlpha, fDist, and Class. The data includes numerical values and categorical labels "g" and "h".

	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):

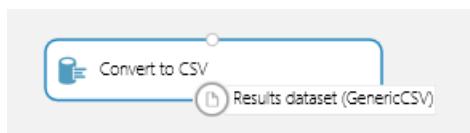
- PlainText
- GenericCSV
- GenericTSV
- GenericCSVNoHeader
- GenericTSVNoHeader

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.

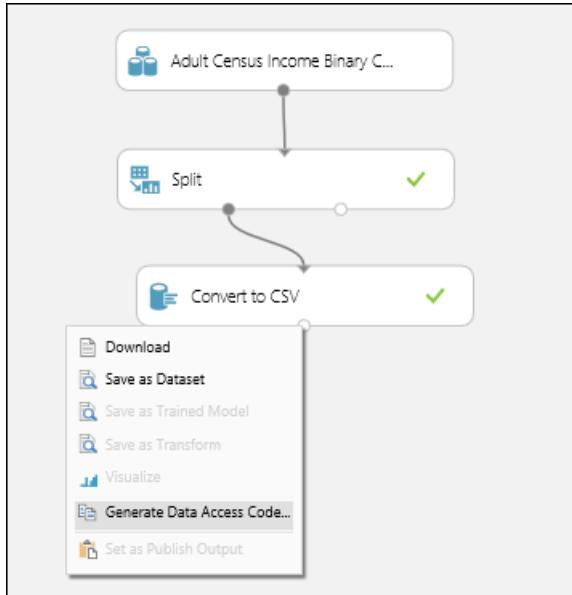


You need to use a conversion module, such as [Convert to CSV](#), to get an output into a supported format.

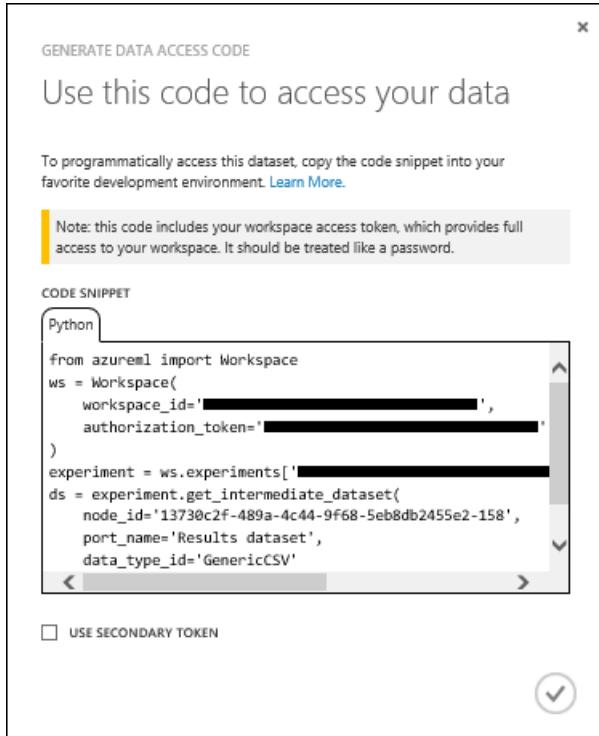


The following steps show an example that creates an experiment, runs it and accesses the intermediate dataset.

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

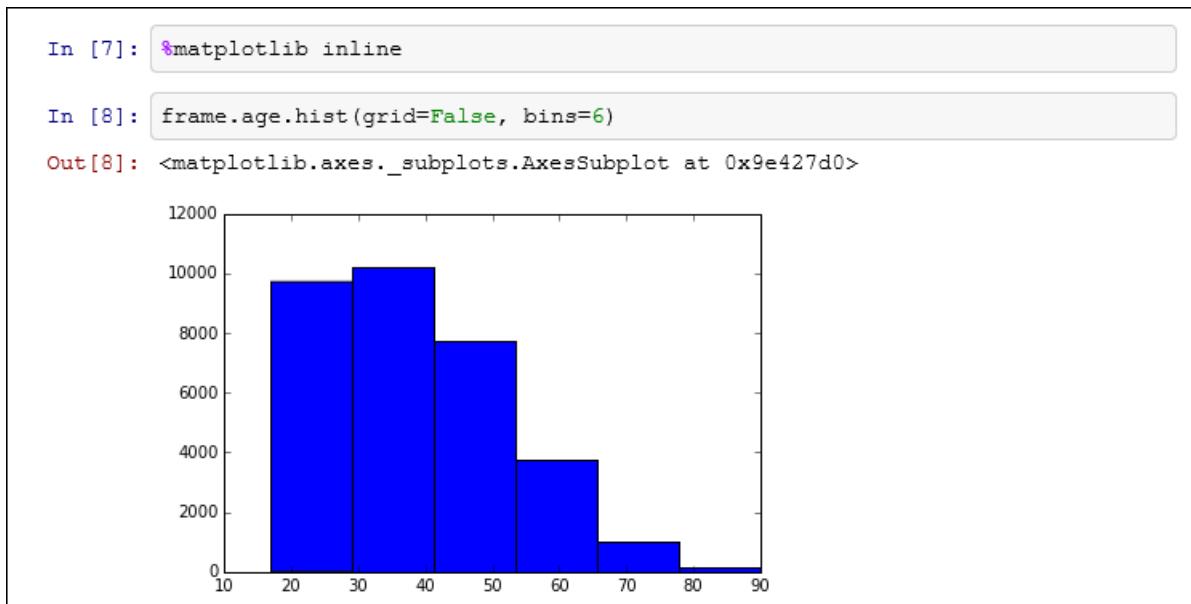
```
In [6]: from azureml import Workspace
ws = Workspace(
    workspace_id='[REDACTED]',
    authorization_token='[REDACTED]'
)
experiment = ws.experiments['[REDACTED]']
ds = experiment.get_intermediate_dataset(
    node_id='13730c2f-489a-4c44-9f68-5eb8db2455e2-158',
    port_name='Results dataset',
    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(
    datafram=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(
    datafram=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

2/2/2018 • 3 min to read • [Edit Online](#)

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

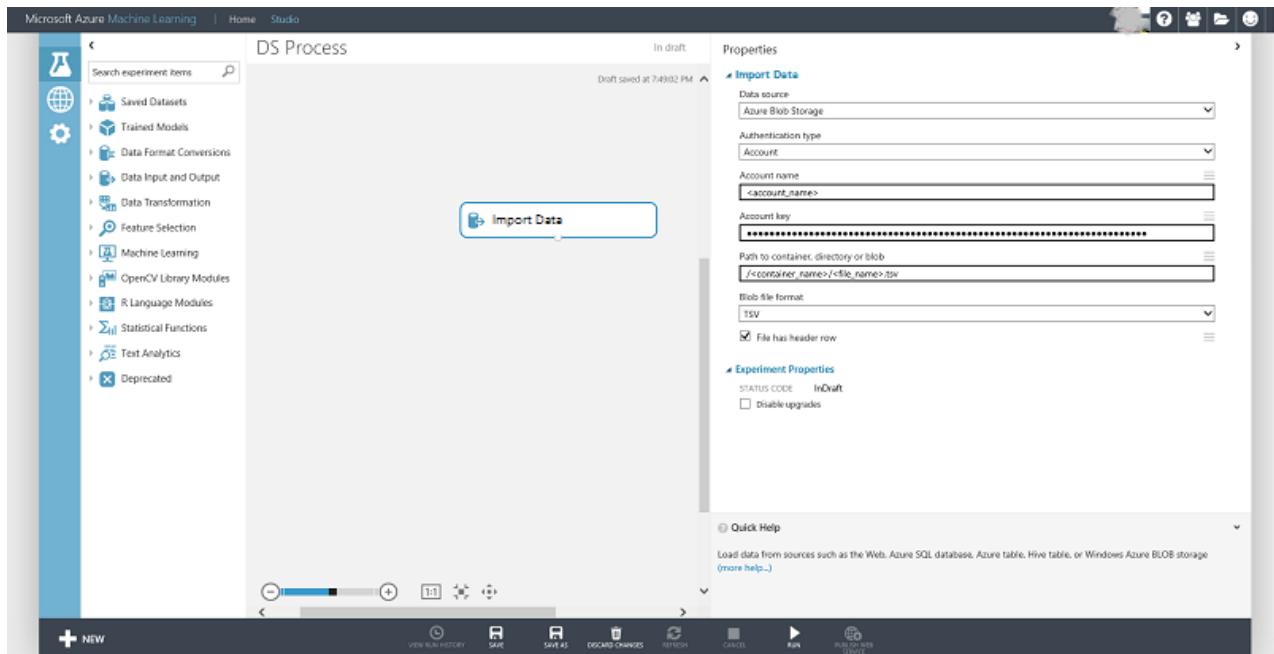
try:

#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 with Azure Data Lake: An end-to-end Walkthrough

11/14/2017 • 19 min to read • [Edit Online](#)

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 is 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 how to install the prerequisites and resources that are needed to complete data science process tasks. 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. It provides a Jupyter notebook with the Python scripts for the 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 later. If you do not already have one of these versions installed, you can download a free Community version from [Visual Studio Community](#).

NOTE

Instead of Visual Studio, you can also use the Azure portal to submit Azure Data Lake queries. Instructions are provided on how to do so both with Visual Studio and on the portal in the section titled **Process data with U-SQL**.

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)

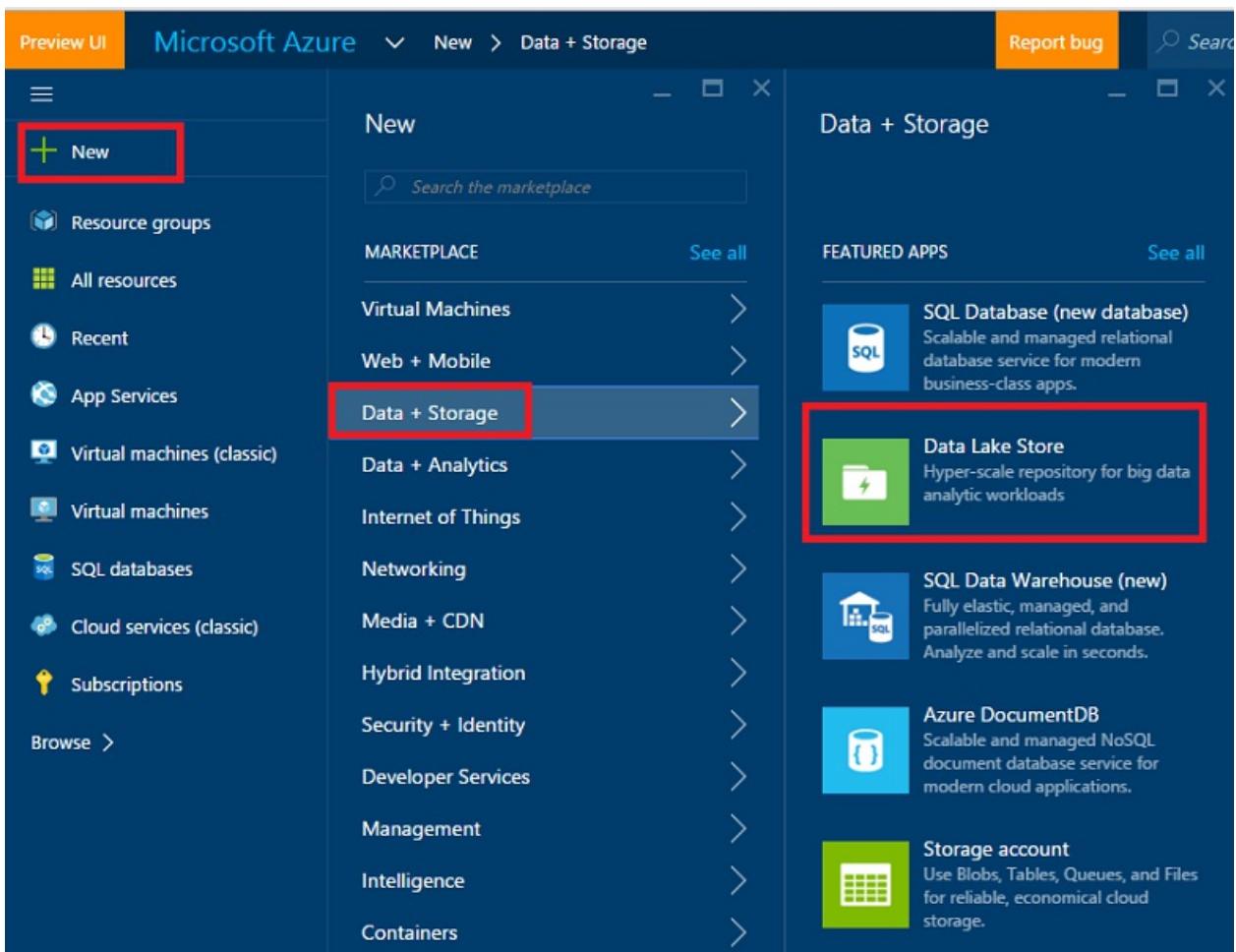
This section provides instructions on how to create each of these resources. If you choose to use Hive tables with Azure Machine Learning, instead of Python, to build a model, you also need to provision an HDInsight (Hadoop) cluster. This alternative procedure is described in the Option 2 section.

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, but the Data Lake storage account need not be created separately.

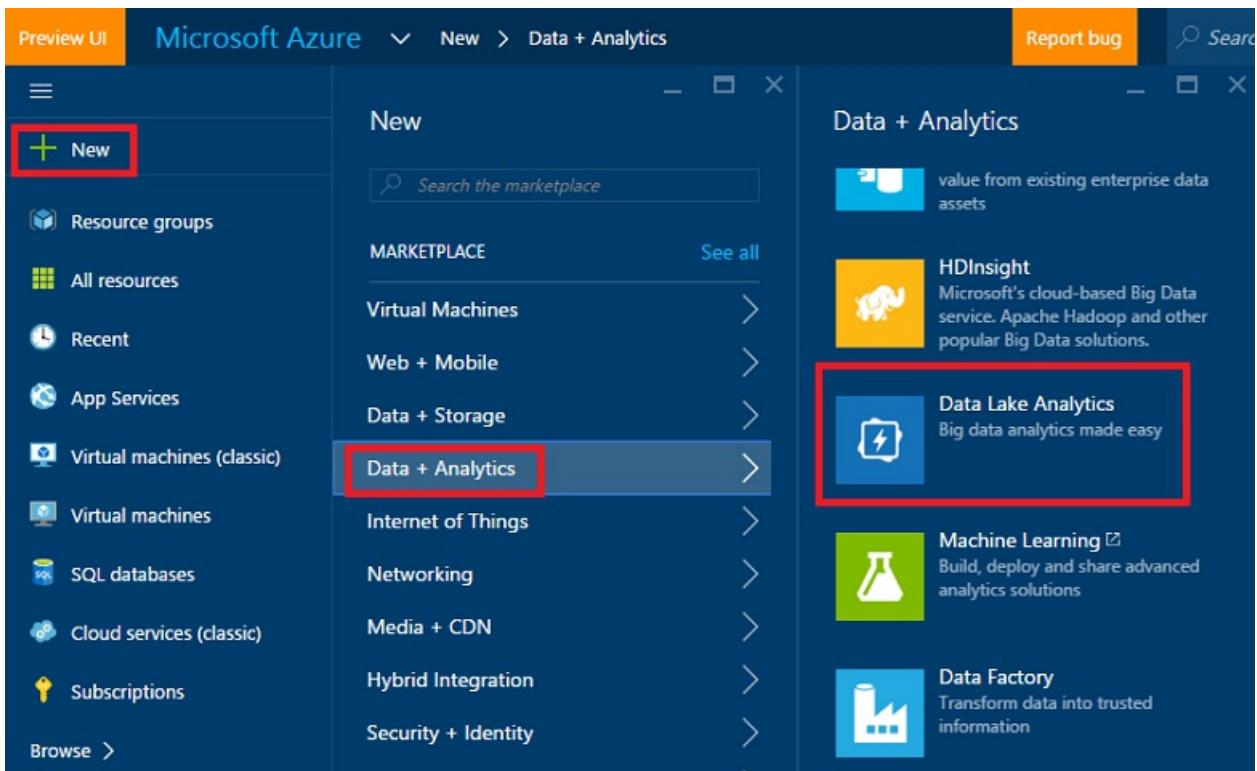
Create an Azure Data Lake Store

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.



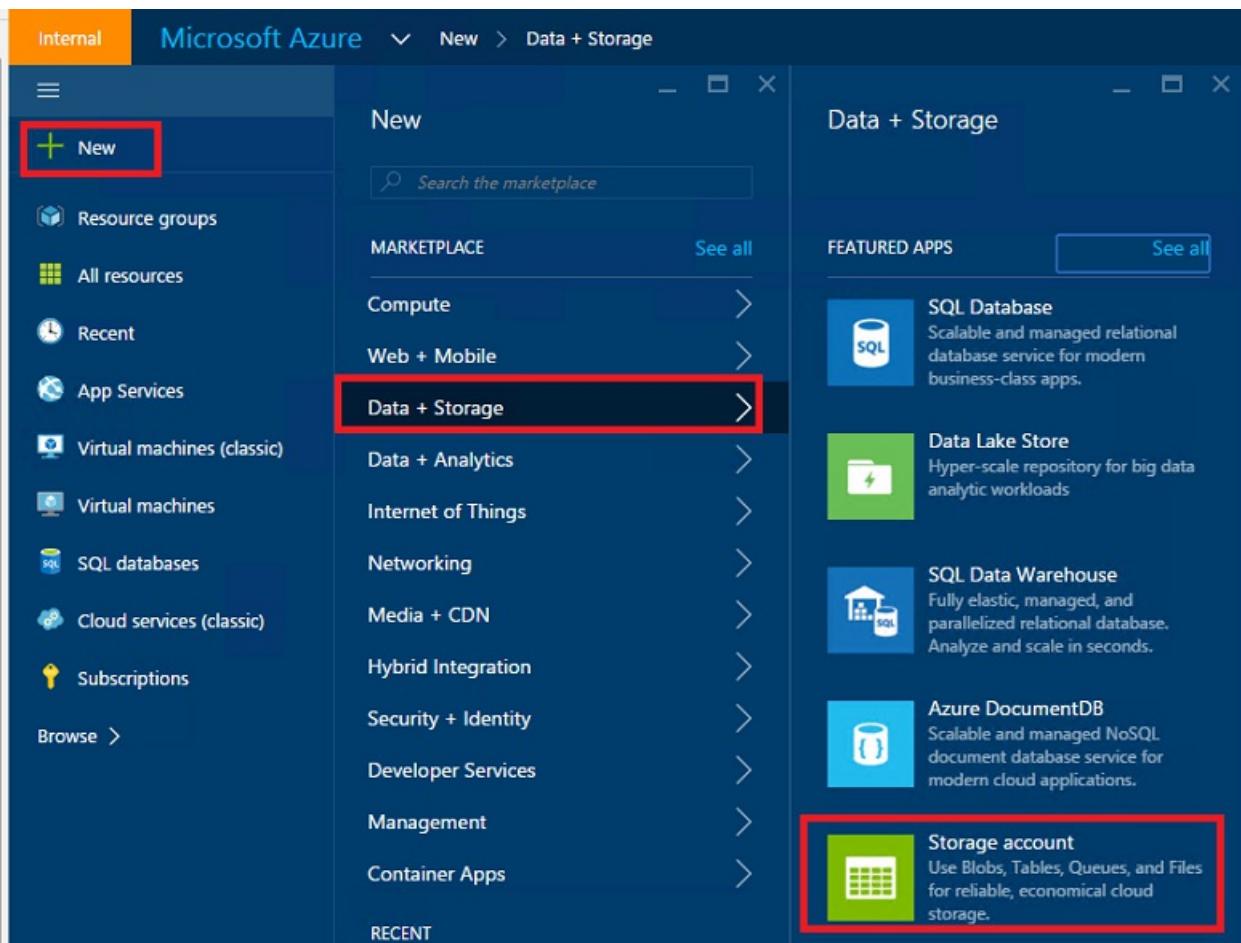
Create an Azure Data Lake Analytics account

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

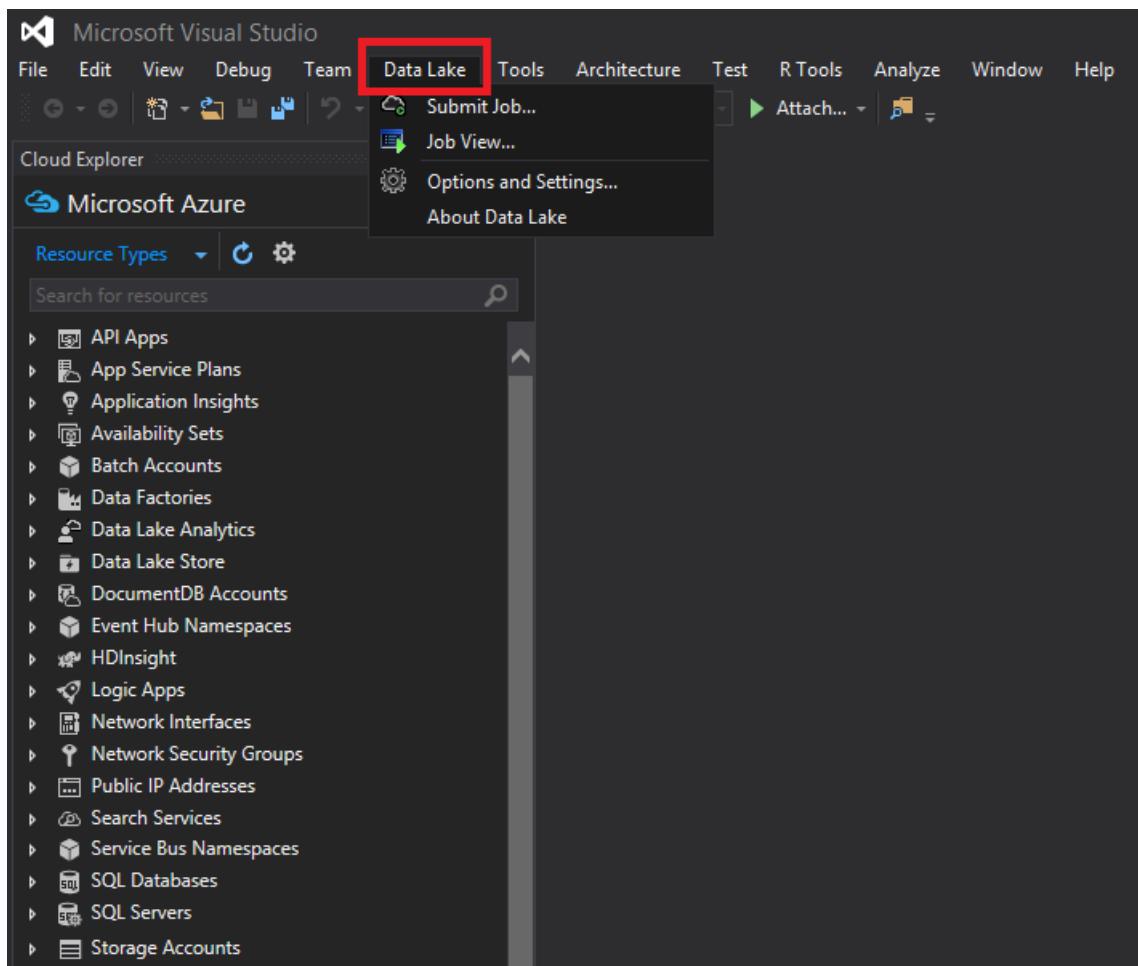
Sign up/into Azure Machine Learning Studio from the [Azure Machine Learning](#) page. Click on the **Get started now** button and then choose a "Free Workspace" or "Standard Workspace". Now you are ready to create experiments in Azure ML Studio.

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](#).

The screenshot shows the 'Azure Data Lake Tools for Visual Studio' page. At the top, the title 'Azure Data Lake Tools for Visual Studio' is displayed. Below it, there's a language selection 'Language: English' and a large orange 'Download' button, which is circled in blue. The text 'Plug-in for Azure Data Lake development using Visual Studio' is present. At the bottom, there are three expandable sections: '+ Details', '+ System Requirements', and '+ 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 used here is a publicly available dataset -- the [NYC Taxi Trips dataset](#). The NYC Taxi Trip data consists of about 20 GB of compressed CSV files (~48 GB 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,9.5,0.5,0.5,0,0,10.5

```

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.

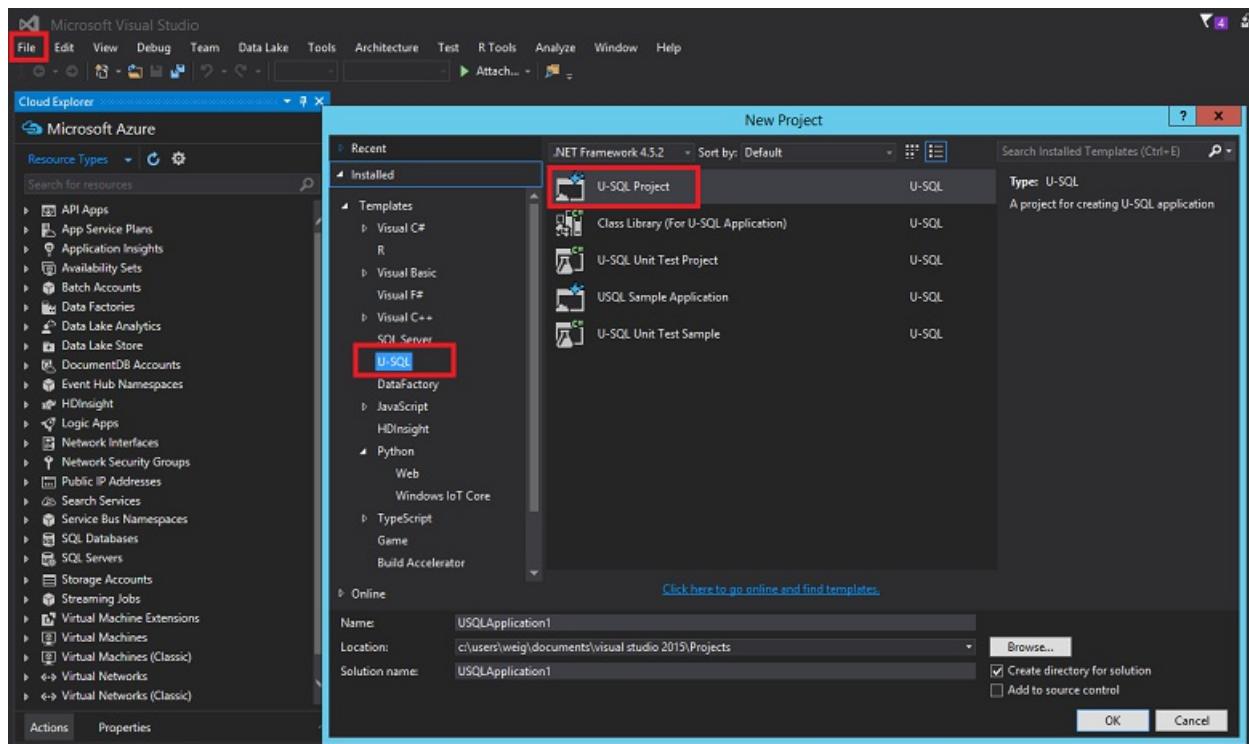
Process data with U-SQL

The data processing tasks illustrated in this section include ingesting, checking quality, exploring, and sampling the data. How to join trip and fare tables is also shown. 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](#)

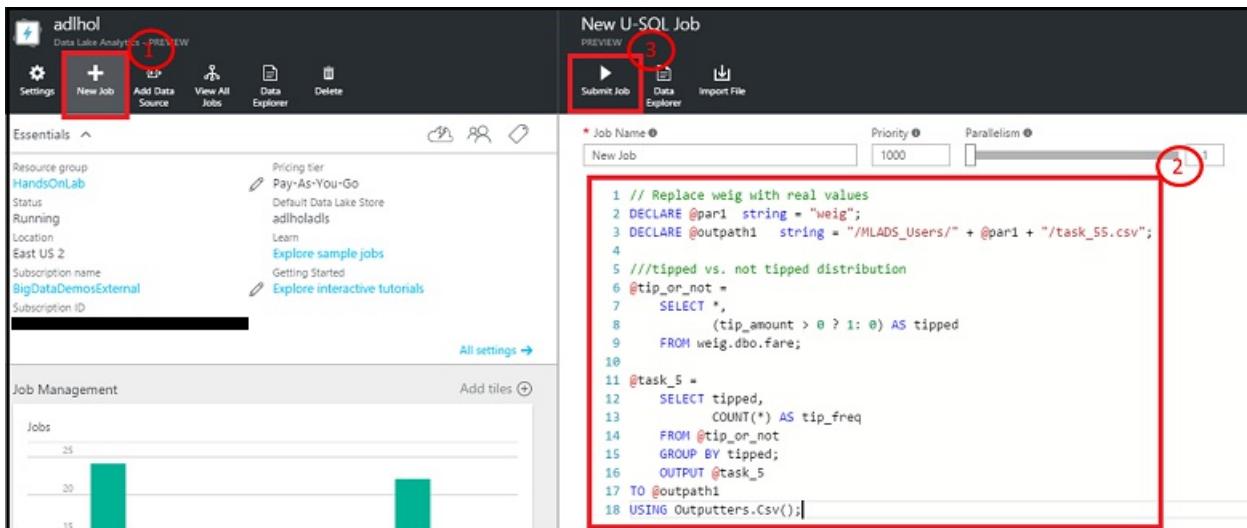
The U-SQL scripts are described here and provided in a separate file. You can download the full **U-SQL scripts** from [GitHub](#).

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:



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, it is possible to 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, you need to remove the headers and change column types into appropriate ones. You 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.

```

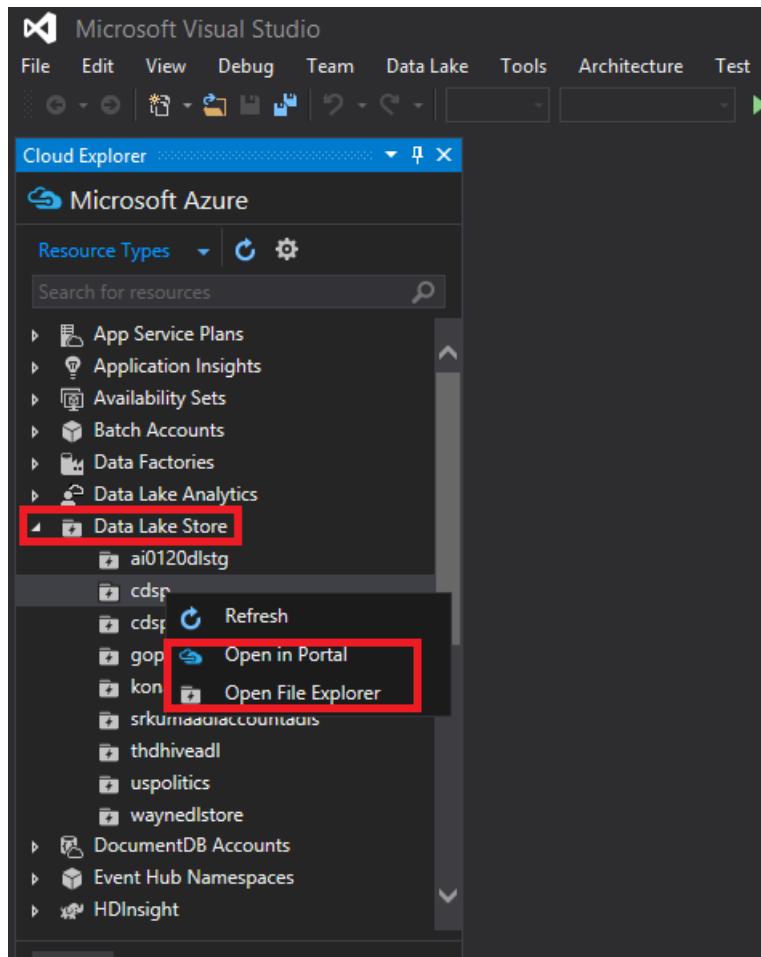
// change data types
@trip =
    SELECT
        medallion,
        hack_license,
        vendor_id,
        rate_code,
        store_and_fwd_flag,
        DateTime.Parse(pickup_datetime) AS pickup_datetime,
        DateTime.Parse(dropoff_datetime) AS dropoff_datetime,
        Int32.Parse(passenger_count) 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";

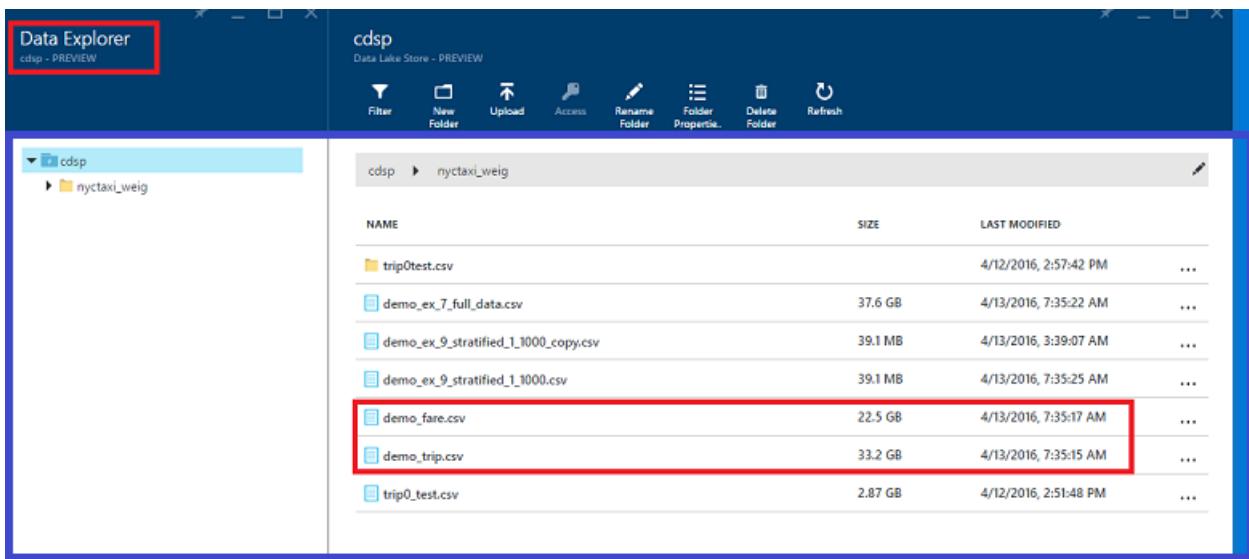
////output data to ADL
OUTPUT @trip
TO "swebhdfs://data_lake_storage_name.azuredatalakestore.net/nyctaxi_folder/demo_trip.csv"
USING Outputters.Csv();

////Output data to blob
OUTPUT @trip
TO "wasbs://container_name@blob_storage_account_name.blob.core.windows.net/demo_trip.csv"
USING Outputters.Csv();

```

Similarly you 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

Do some data exploration with the following scripts 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/nytaxi_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, randomly select 0.1% of the data from the joined table:

```

//random select 1/1000 data for modeling purpose
@addrownumberres_randomsample =
SELECT *,
ROW_NUMBER() OVER() AS rounum
FROM @model_data_full;

@model_data_random_sample_1_1000 =
SELECT *
FROM @addrownumberres_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 do stratified sampling by binary variable tip_class:

```

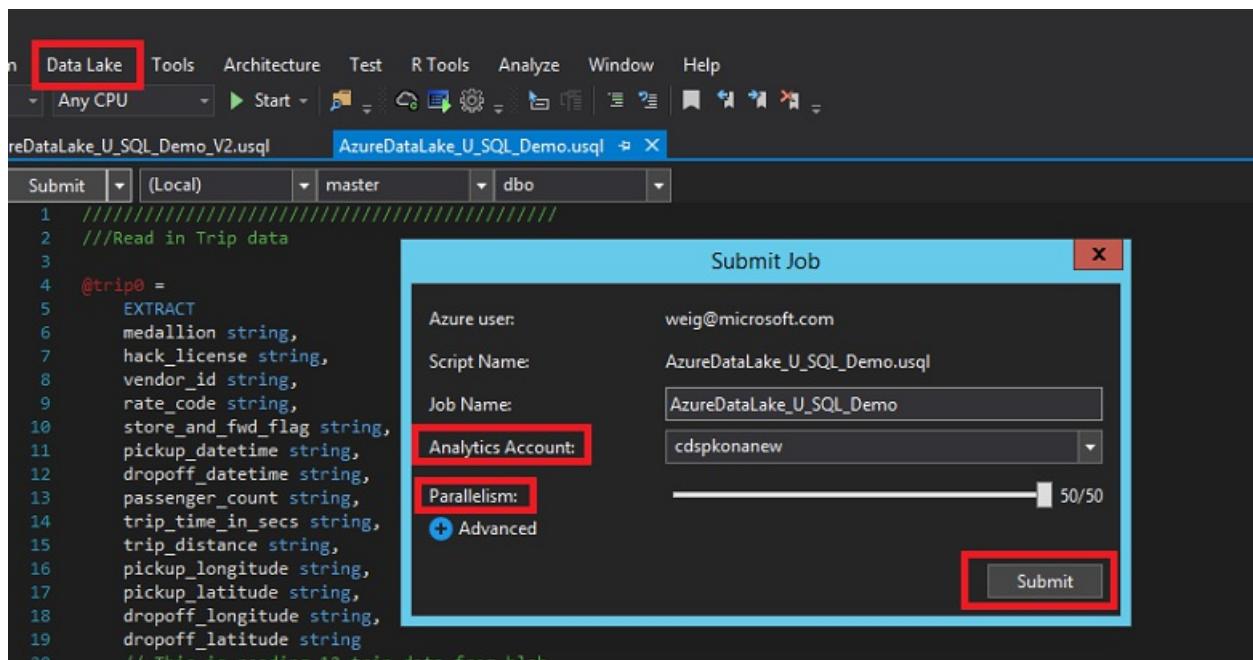
//stratified random select 1/1000 data for modeling purpose
@addrownumberres_stratifiedsample =
SELECT *,
       ROW_NUMBER() OVER(PARTITION BY tip_class) AS rounum
FROM @model_data_full;

@model_data_stratified_sample_1_1000 =
SELECT *
FROM @addrownumberres_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/nyctaxi_folder/demo_ex_9_stratified_1_1000.csv"
USING Outputters.Csv();

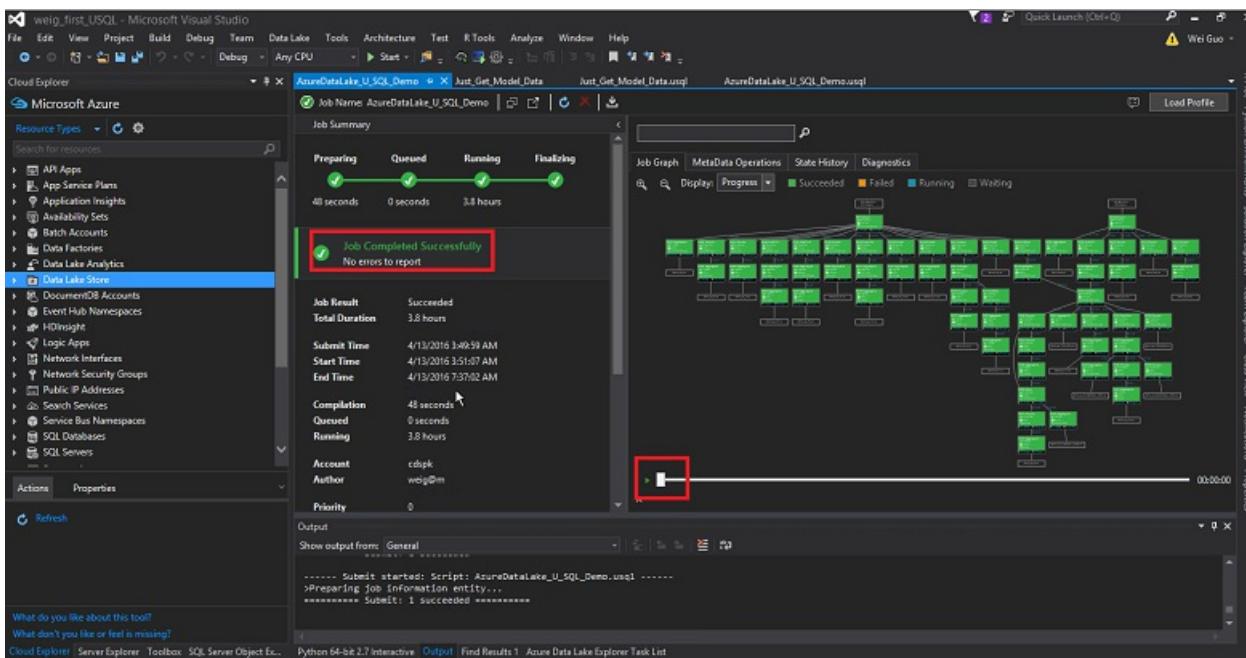
```

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.



When the job is complied successfully, the status of your job is 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.



The screenshot shows the Azure Data Lake Analytics portal. On the left, the 'cdspk' resource group details are shown, including a 'View All Jobs' button highlighted with a red box. On the right, the 'All Jobs' list shows a total of 238 jobs. One job, 'AzureDataLake_U_SQL_Demo', is listed with a green checkmark indicating it succeeded. The 'Status' column shows 'Succeeded' for this job, while others show 'U-SQL'.

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. Use the stratified sample data for our modeling in the next step.

The screenshot shows the Azure Storage Explorer interface. The left sidebar lists 'Storage Account' (weigstorageforsv) and 'Blob Containers (7)' containing 'Slogs' and 'test1'. The 'test1' container is selected, showing a list of 98 blobs. A red box highlights the 'test1' folder in the left navigation pane. The table below lists the blob names, types, last modified times, lengths, and content types.

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 2:35:26 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_fare.csv	Block	4/13/2016 10:39:08 AM +0:00	37.32M	text/plain; charset=utf-8
demo_trip.csv	Block	4/13/2016 2:35:18 PM +0:00	20.97G	text/plain; charset=utf-8

The screenshot shows the Azure Data Lake Store - PREVIEW interface. At the top, there's a toolbar with icons for Filter, New Folder, Upload, Access, Rename Folder, Folder Properties, Delete Folder, and Refresh. Below the toolbar, the path 'cdsp > nyctaxi_weig' is displayed. The main area is a table listing files in the 'nyctaxi_weig' folder:

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

Two options are 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. Both these options are discussed in the following sections.

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, just the modeling and deployment are covered.

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 = 'YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY'
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', 'równum']
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 you 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 you need to create dummy variables that can be used in scikit-learn models

```

df1_payment_type_dummmy = pd.get_dummies(df1['payment_type'], prefix='payment_type_dummmy')
df1_vendor_id_dummmy = pd.get_dummies(df1['vendor_id'], prefix='vendor_id_dummmy')

```

- Create data frame for the modeling

```

cols_to_keep = ['tipped', 'trip_distance', 'passenger_count']
data = df1[cols_to_keep].join([df1_payment_type_dummmy,df1_vendor_id_dummmy])

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/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/data-lake-walkthrough/c2-py-logit-evaluation.PNG)

```

Build Web Service API and consume it in Python

You want to operationalize the machine learning model after it has been built. The binary logistic model is used here 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.**

NAME	AUTHORIZATION TOKENS	USERS
WORKSPACE NAME	Wei-WorkSpace-Azure-ML	
WORKSPACE DESCRIPTION		
WORKSPACE TYPE	Standard	Learn More
WORKSPACE ID		
CREATION TIME	4/3/2015, 1:33:43 PM	
OWNER'S EMAIL	outlook.com	
SUBSCRIPTION ID	e8:	
STORAGE ACCOUNT	acc1	

```

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_fit.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/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](#).

The screenshot shows the Microsoft Azure Preview UI. In the top left, there's a 'New' button with a red box around it. The main area is titled 'New' and has a search bar 'Search the marketplace'. Below it is a list of categories: MARKETPLACE, Virtual Machines, Web + Mobile, Data + Storage, Data + Analytics (which is highlighted with a red box), Internet of Things, Networking, Media + CDN, Hybrid Integration, Security + Identity, Developer Services, and Management.

On the right side, under 'Data + Analytics', there are five items:

- Power BI Embedded**: Embed fully interactive, stunning data visualizations in your applications.
- Cognitive Services APIs**: Microsoft Cognitive Services lets you build apps with powerful algorithms using just a few lines of code.
- Data Catalog**: Data source discovery to get more value from existing enterprise data assets.
- HDInsight**: Microsoft's cloud-based Big Data service. Apache Hadoop and other popular Big Data solutions. (This item is also highlighted with a red box).
- Data Lake Analytics**: Big data analytics made easy.

Create Hive table in HDInsight

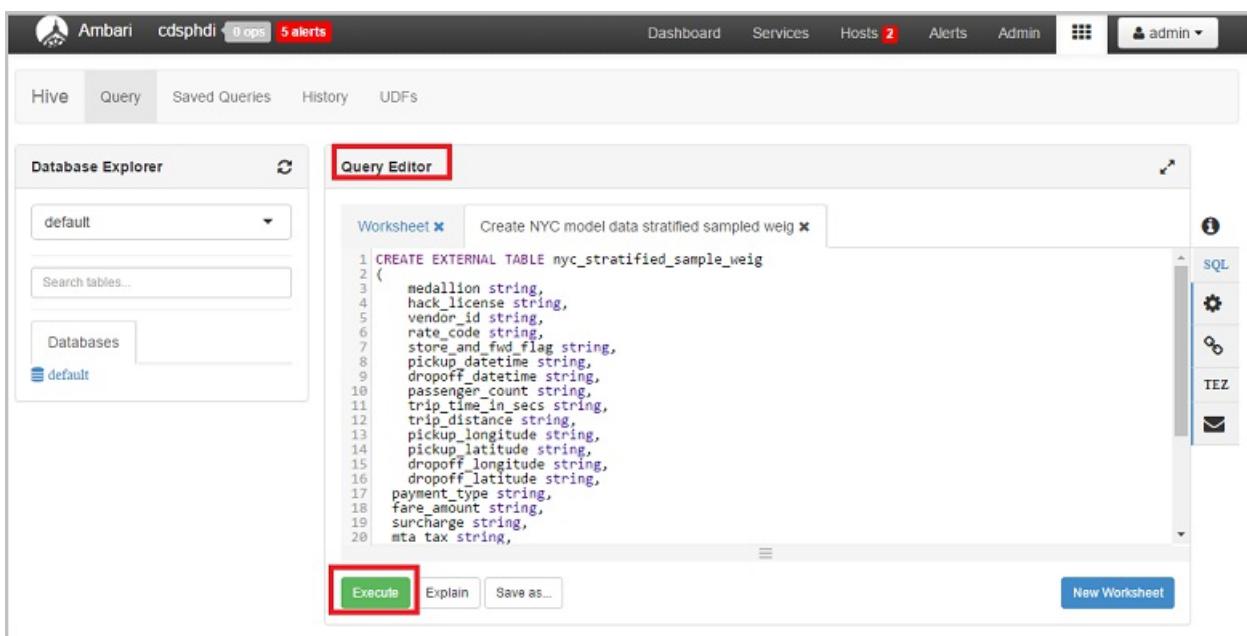
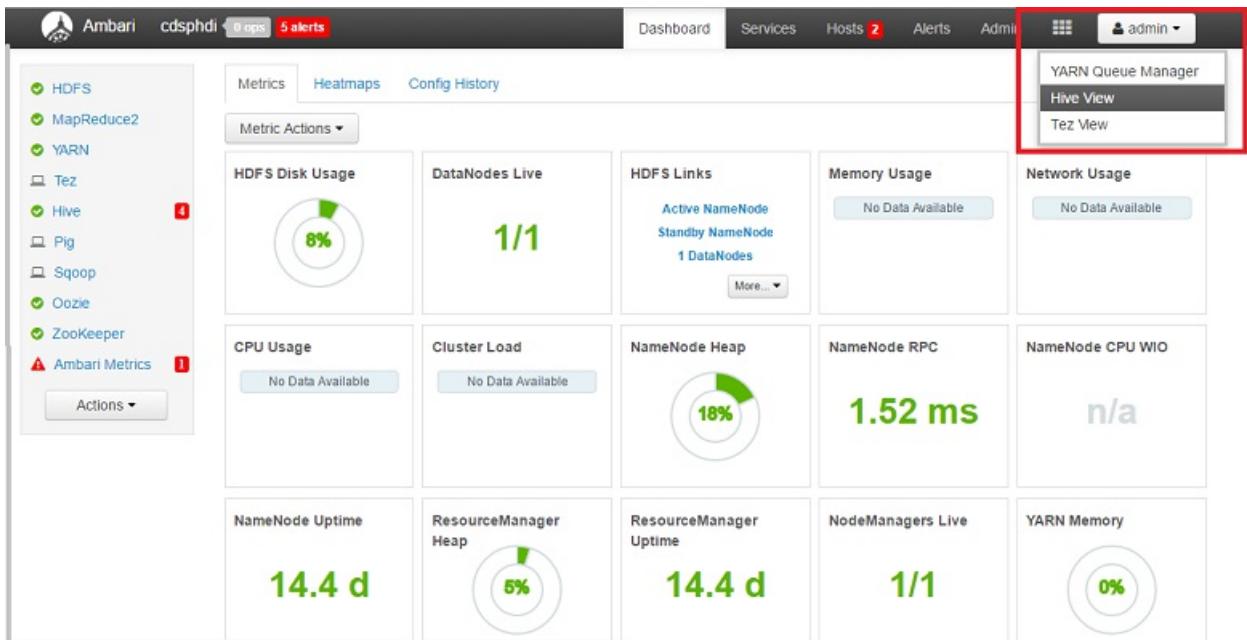
Now you create Hive tables to be used in Azure Machine Learning Studio in the HDInsight cluster using the data stored in Azure Data Lake Store in the previous step. Go to the HDInsight cluster created. Click **Settings** --> **Properties** --> **Cluster AAD Identity** --> **ADLS Access**, make sure your Azure Data Lake Store account is added in the list with read, write and execute rights.

The screenshot shows the Azure HDInsight Cluster Settings page. On the left, there's a summary card with cores information: 2,100 cores, 12 clusters, 0 other clusters, 2088 available, and 2100 total. Below it are 'Quick Links' for Cluster Dashboard, Ambari View, and Scale Cluster.

The main area is titled 'Settings' and shows the 'Cluster AAD Identity' configuration. The 'ADLS Access' tab is selected. It displays the 'SERVICE PRINCIPAL' settings and a table for 'Other ADLS accounts'. The table has columns for 'DATA LAKE STORE', 'READ', 'WRITE', and 'EXECUTE'. One row for 'adlscat' has all three checkboxes checked and is highlighted with a red box.

DATA LAKE STORE	READ	WRITE	EXECUTE
adlscat	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
adlscat	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
adlscat	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
adlscat	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
adlscat	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
adlscat	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
adlscat	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
adlscat	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
adlscat	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
adlscat	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
adlscat	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Then click **Dashboard** next to the **Settings** button and a window pops up. Click **Hive View** in the upper right corner of the page and you should see the **Query Editor**.



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/nytaxi_folder/demo_ex_9_stratified_1_1000_copy.csv';

```

When the query finishes running, you should 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	nyc_stratified_sample_weig.rate_code
"0053334C798EC6C8E637657962030F99"	"9EF690115D60940E7A8039A67542642E"	"VTS"	"1"
"00B99071EE4DC8266384113B91E6AC13"	"081B28EDA7F73E5E2C97573F0DBAC25D"	"CMT"	"0"
"011E4CBA1987A8553F8EA5681FFBF7F1"	"F53B26859F6C46C24E39EEAEE8096268"	"CMT"	"1"
"1109955CCAABC BCE1A22BCED5F1DBFF5"	"EAA69F43239E5C6609CD8FF4D6725836"	"CMT"	"0"
"0A415B814A6479EE706972D2FD6DA08A"	"12A32F655B8C9CE3AC7872477A641974"	"VTS"	"1"
"02B29197FB7470B583ED12167D19E998"	"C1EEBC8298A619F86637D7E97F6BDD5C"	"CMT"	"0"
"03055B956C21B1F915DCDB118AA79F21"	"E0C7A67293DE535DB04F8AFD8BF28F73"	"VTS"	"1"
"037673EEAE0DCB912D06BED04E89D89D"	"3B247BD1230E95A0D9FED3E47FECB8D9"	"CMT"	"0"
"03BF54085C92C385889B957D804780D1"	"776D5633A2041CEBDE03092E401D61DB"	"CMT"	"1"
"0437660BC3704C2F185301D539434A64"	"AE9AB8C79A2A0EB6DDA76B7024FF6029"	"CMT"	"0"

Build and deploy models in Azure Machine Learning Studio

You 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 it is only shown 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

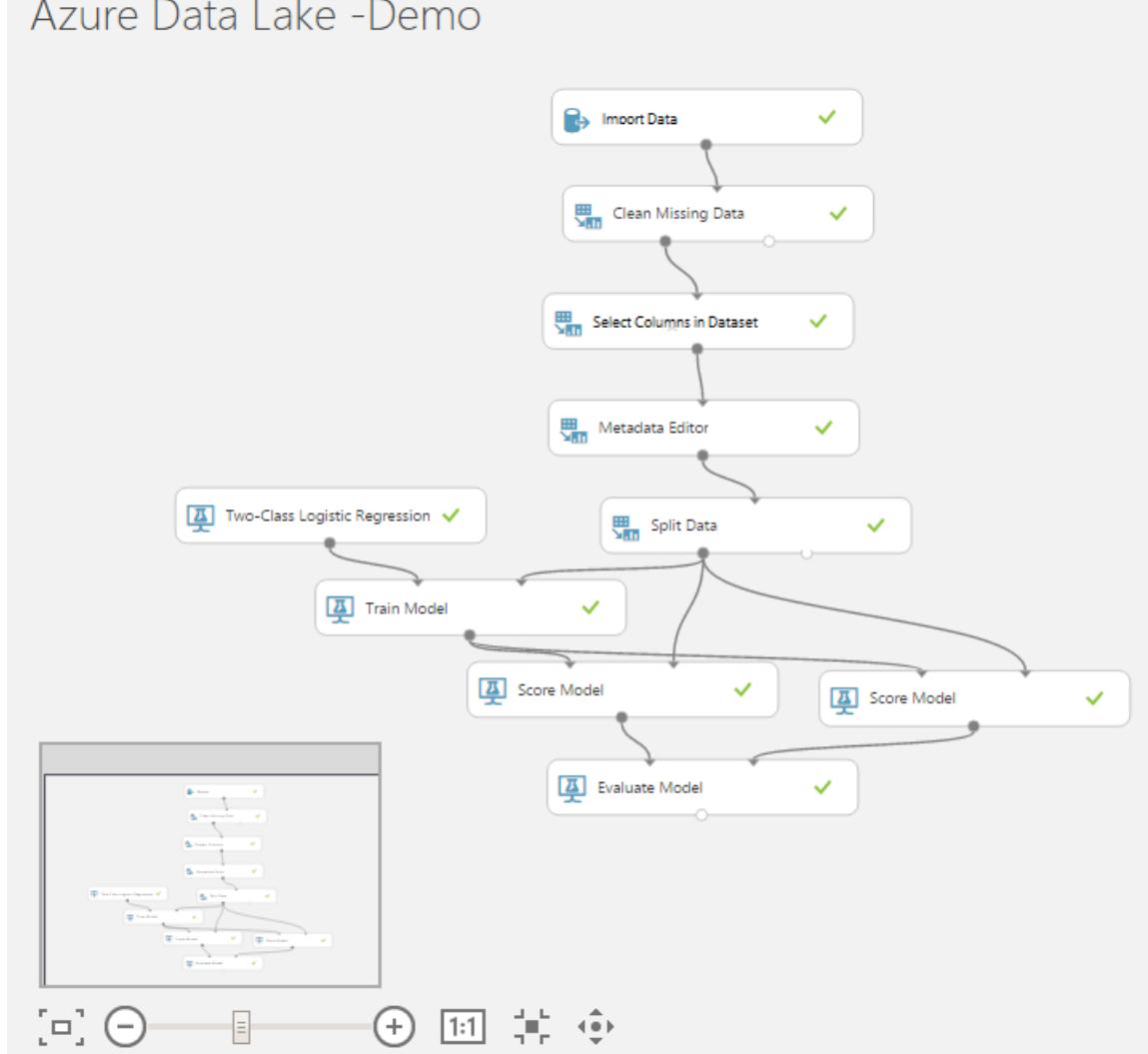
```
select * from nyc_stratified_sample;
```

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 following figure:

Azure Data Lake -Demo

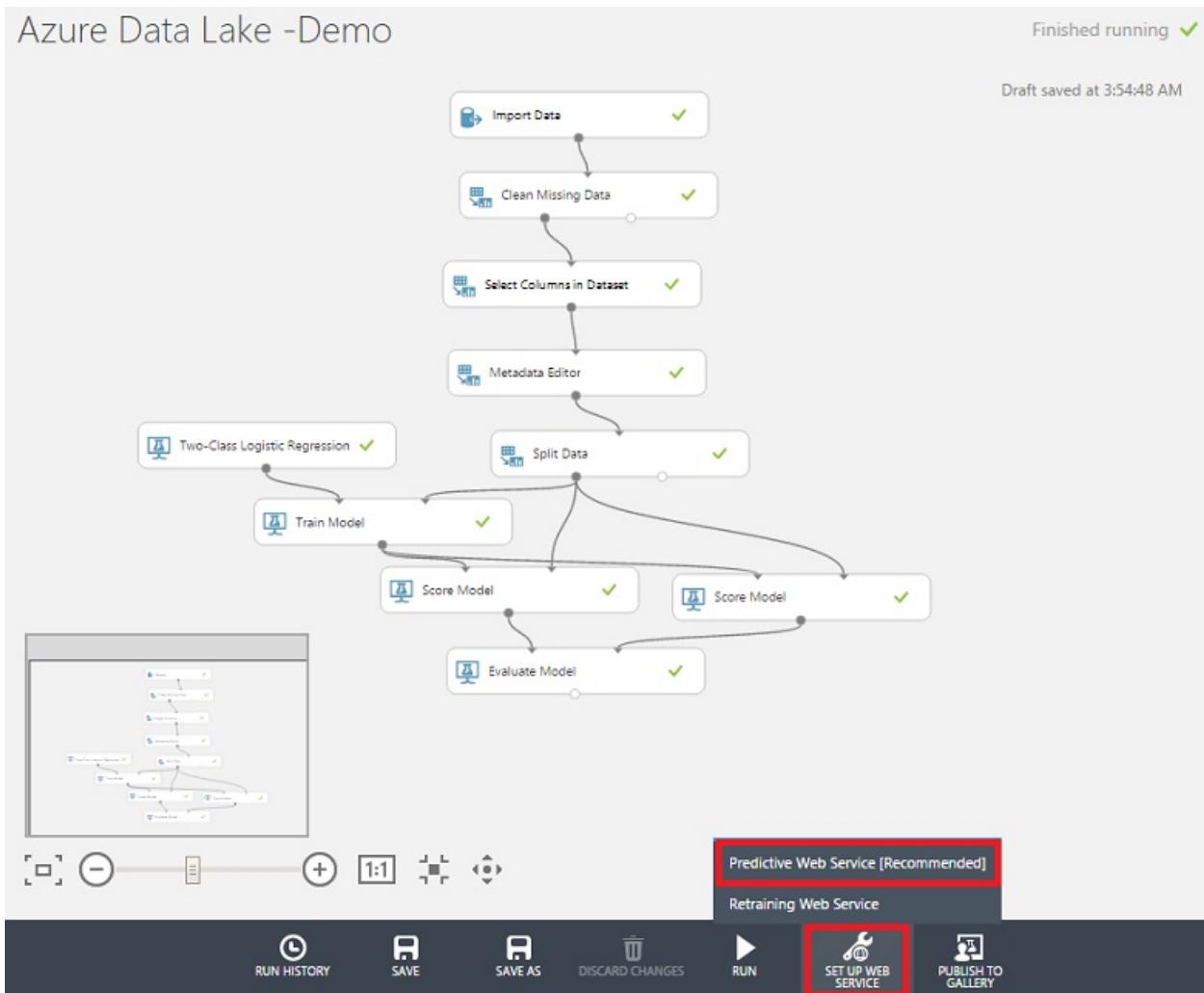


After the experiment is created, click **Set Up Web Service --> Predictive Web Service**

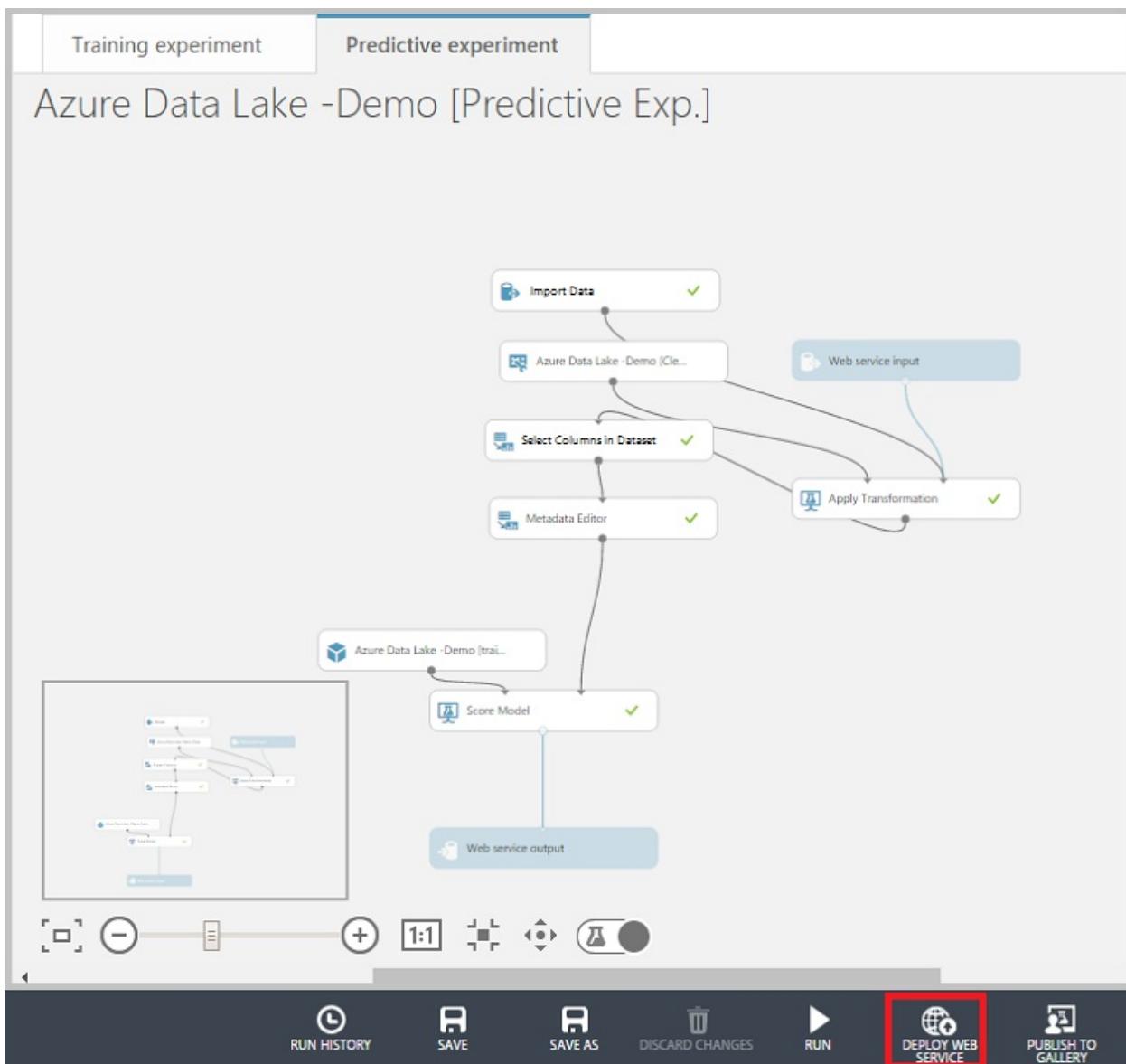
Azure Data Lake -Demo

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 displays 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
 h86t ... 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

2/2/2018 • 6 min to read • [Edit Online](#)

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
,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) 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>

```

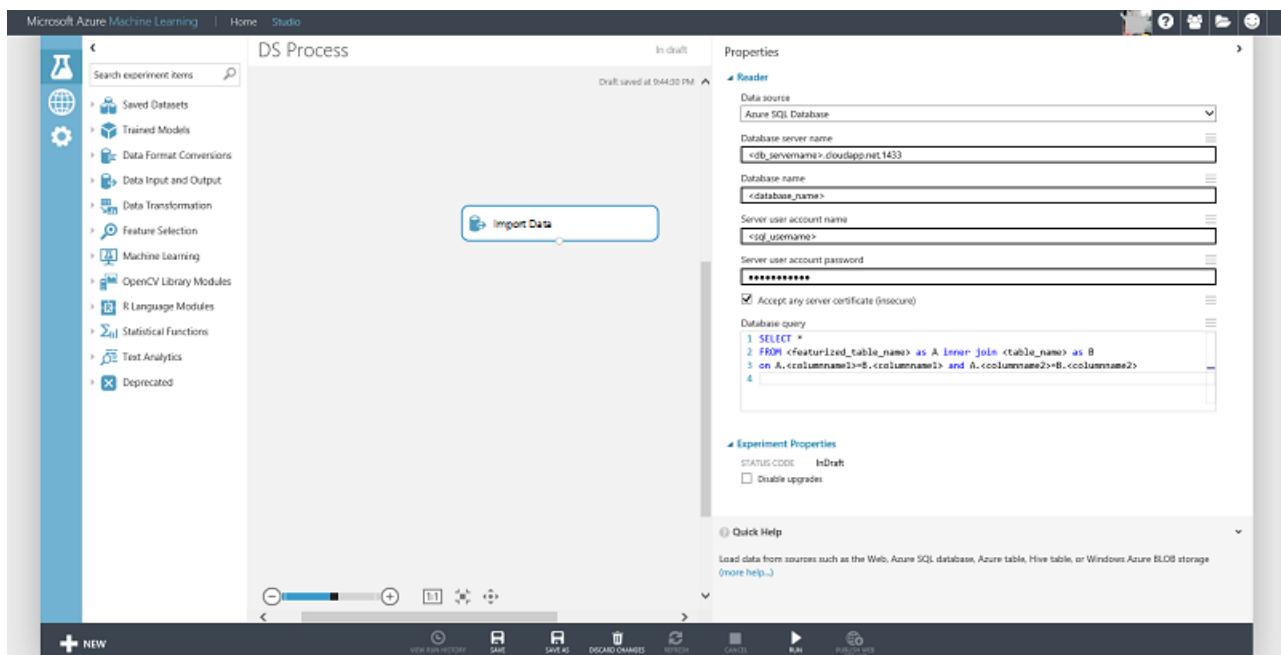
These location-based features can be further used to generate additional count features as described earlier.

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

9/25/2017 • 1 min to read • [Edit Online](#)

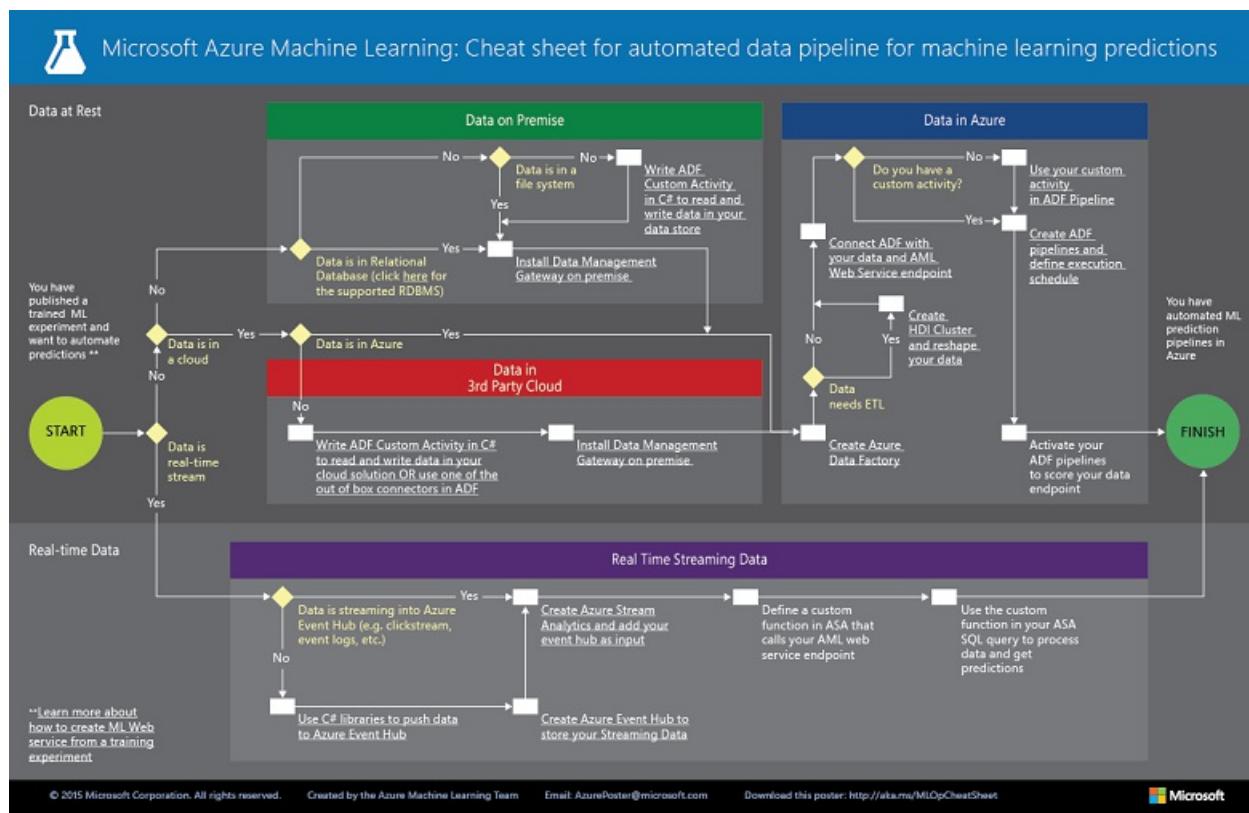
The **Microsoft Azure Machine Learning automated data pipeline cheat sheet** helps you navigate through the technology you can use to get your data to your Machine Learning web service where it can be scored by your predictive analytics model.

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 need to make, and it offers links to articles that can 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](#).

NOTE

You can try Azure Machine Learning for free. No credit card or Azure subscription is required. [Get started now.](#)

Overview of data science using Spark on Azure HDInsight

12/7/2017 • 9 min to read • [Edit Online](#)

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 references the topics that describe how to set up the Spark cluster that you need to complete the steps in the walkthroughs provided.

Spark and MLlib

Spark is an open-source parallel processing framework that supports in-memory processing to boost the performance of big-data analytic 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 the iterative algorithms used in machine learning and graph computations. **MLlib** is Spark's scalable machine learning library that brings the algorithmic modeling capabilities to this distributed environment.

HDInsight Spark

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.

Setup: Spark clusters and Jupyter notebooks

Setup steps and code are provided in this walkthrough for using an HDInsight Spark 1.6. But Jupyter notebooks are provided for both HDInsight Spark 1.6 and Spark 2.0 clusters. A description of the notebooks and links to them are provided in the [Readme.md](#) for the GitHub repository containing them. Moreover, the code here and in the linked 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. For convenience, here are the links to the Jupyter notebooks for Spark 1.6 (to be run in the pySpark kernel of the Jupyter Notebook server) and Spark 2.0 (to be run in the pySpark3 kernel of the Jupyter Notebook server):

Spark 1.6 notebooks

These notebooks are to be run in the pySpark kernel of Jupyter notebook server.

- [pySpark-machine-learning-data-science-spark-data-exploration-modeling.ipynb](#): Provides information on how to perform data exploration, modeling, and scoring with several different algorithms.
- [pySpark-machine-learning-data-science-spark-advanced-data-exploration-modeling.ipynb](#): Includes topics in notebook #1, and model development using hyperparameter tuning and cross-validation.
- [pySpark-machine-learning-data-science-spark-model-consumption.ipynb](#): Shows how to operationalize a saved model using Python on HDInsight clusters.

Spark 2.0 notebooks

These notebooks are to be run in the pySpark3 kernel of Jupyter notebook server.

- [Spark2.0-pySpark3-machine-learning-data-science-spark-advanced-data-exploration-modeling.ipynb](#): This file provides information on how to perform data exploration, modeling, and scoring in Spark 2.0 clusters using the NYC Taxi trip and fare data-set described [here](#). This notebook may be a good starting point for quickly exploring the code we have provided for Spark 2.0. For a more detailed notebook analyzes the NYC Taxi data, see the next notebook in this list. See the notes following this list that compare these notebooks.
- [Spark2.0-pySpark3_NYC_Taxi_Tip_Regression.ipynb](#): This file shows how to perform data wrangling (Spark SQL and dataframe operations), exploration, modeling and scoring using the NYC Taxi trip and fare data-set described [here](#).
- [Spark2.0-pySpark3_Airline_Departure_Delay_Classification.ipynb](#): This file shows how to perform data wrangling (Spark SQL and dataframe operations), exploration, modeling and scoring using the well-known Airline On-time departure dataset from 2011 and 2012. We integrated the airline dataset with the airport weather data (e.g. windspeed, temperature, altitude etc.) prior to modeling, so these weather features can be included in the model.

NOTE

The airline dataset was added to the Spark 2.0 notebooks to better illustrate the use of classification algorithms. See the following links for information about airline on-time departure dataset and weather dataset:

- Airline on-time departure data: <http://www.transtats.bts.gov/ONTIME/>
- Airport weather data: <https://www.ncdc.noaa.gov/>

NOTE

The Spark 2.0 notebooks on the NYC taxi and airline flight delay data-sets can take 10 mins or more to run (depending on the size of your HDI cluster). The first notebook in the above list shows many aspects of the data exploration, visualization and ML model training in a notebook that takes less time to run with down-sampled NYC data set, in which the taxi and fare files have been pre-joined: [Spark2.0-pySpark3-machine-learning-data-science-spark-advanced-data-exploration-modeling.ipynb](#) This notebook takes a much shorter time to finish (2-3 mins) and may be a good starting point for quickly exploring the code we have provided for Spark 2.0.

For guidance on the operationalization of a Spark 2.0 model and model consumption for scoring, see the [Spark 1.6 document on consumption](#) for an example outlining the steps required. To use this on Spark 2.0, replace the Python code file with [this file](#).

Prerequisites

The following procedures are related to Spark 1.6. For the Spark 2.0 version, use the notebooks described and linked to previously.

1. You must have an Azure subscription. If you do not already have one, see [Get Azure free trial](#).
2. You need a 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 price of '+ 0.00 USD/CORE/HOUR'. The PREMIUM (PREVIEW) tier includes 'Administration', 'Scalability', '99.9% Uptime SLA', 'Automatic patching', and 'Microsoft R Server for HDInsight', with a price 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

Billing for HDInsight clusters is prorated per minute, whether you are using them or not. Be sure to delete your cluster after you have finished using it. For more information, 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
89D227B655E5C82AECE13C3F540D4CF4,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
89D227B655E5C82AEFC13C3F540D4CF4,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

FIELD	BRIEF DESCRIPTION
dropoff_latitude	Dropoff latitude
direct_distance	Direct distance between pick up and dropoff locations
payment_type	Payment type (cas, credit-card etc.)
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 the Azure portal interface for managing an HDInsight cluster. On the left, the 'HDInsight Cluster' blade is open, displaying essential information like the resource group, status (Running), location (South Central US), and subscription details. A red circle highlights the 'Cluster Dashboards' button in the 'Quick Links' section. On the right, the 'Cluster Dashboards' blade is open, showing a dashboard with four tiles: 'HDInsight Cluster Dashboard' (with a yellow elephant icon), 'Jupyter Notebook' (with a circular orange icon), 'Spark History Server' (with a blue square icon), and 'Yarn' (with a blue square icon). The 'Jupyter Notebook' tile is also 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 notebook interface. At the top, there are tabs for 'Files', 'Running', and 'Clusters'. Below the tabs, there's a message: 'Select items to perform actions on them.' A file list shows three entries: 'PySpark' (selected and highlighted with a red circle), 'Scala', and a folder icon. On the right side of the interface, there are buttons for 'Upload', 'New', and a refresh icon. The 'Upload' button is circled in red.

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**.

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

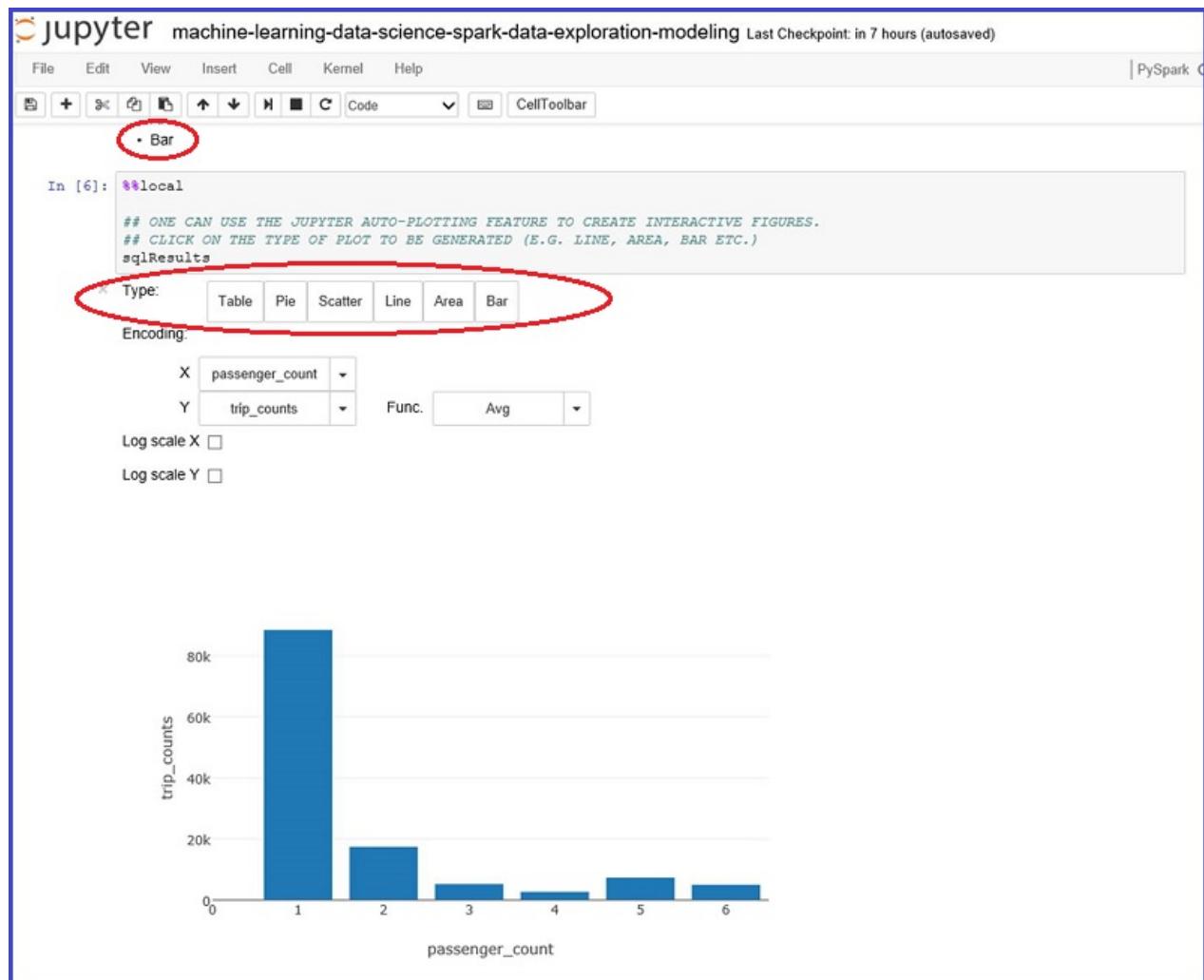
You can right-click the links on your browser and select **Copy Link** to get the github raw content URL. You can paste this URL into the Jupyter Upload file explorer dialog box.

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 Science using Scala and Spark on Azure

11/14/2017 • 34 min to read • [Edit Online](#)

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
Configure required settings 🔒

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

Billing for HDInsight clusters is prorated per minute, whether you are using them or not. Be sure to delete your cluster after you have finished using it. For more information, 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. On the left, there's a summary card with resource group, status, location, and subscription information. Below it is a donut chart showing cluster usage. On the right, there's a 'Cluster Dashboards' section with a list of tiles. The 'Cluster Dashboards' link in the 'Quick Links' area is circled in red. The 'Jupyter Notebook' tile is also 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 that is a file explorer with a list of items. In the top right, there are buttons for 'Upload', 'New', and a refresh icon. A red circle highlights the 'Upload' button. In the file tree, there are two entries: 'PySpark' and 'Scala'. The 'Scala' entry is 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, GBTRRegressor, GBTRRegressionModel}
import org.apache.spark.ml.classification.{LogisticRegression, LogisticRegressionModel,
RandomForestClassifier, RandomForestClassificationModel, GBTCClassifier, GBTCClassificationModel}
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@cdsparksamples.blob.core.windows.net/Data/NYCTaxi/JoinedTaxiTripFare.P
oint1Pct.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

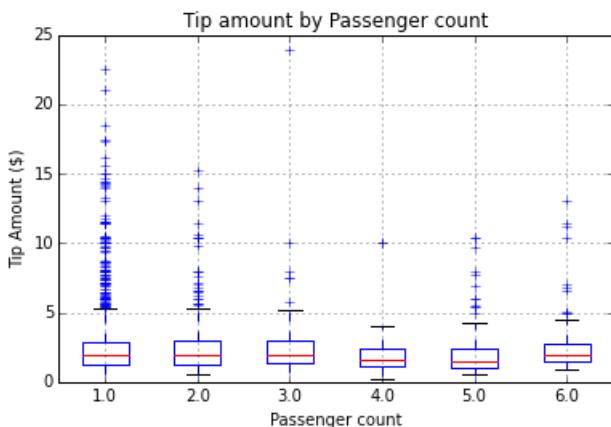
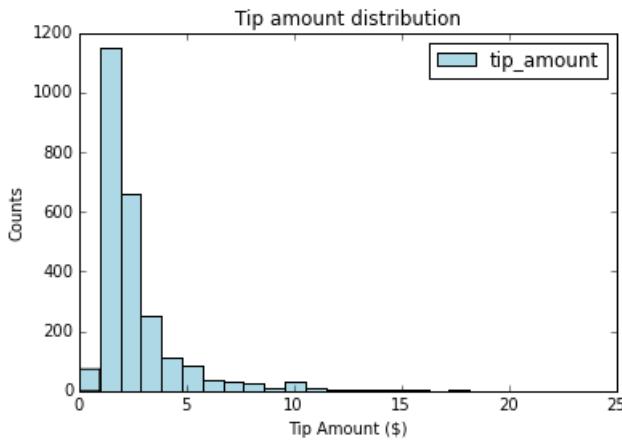
# PLOT TIP BY PAYMENT TYPE AND PASSENGER COUNT
ax1 = sqlResults[['tip_amount']].plot(kind='hist', bins=25, facecolor='lightblue')
ax1.set_title('Tip amount distribution')
ax1.set_xlabel('Tip Amount ($)')
ax1.set_ylabel('Counts')
plt.suptitle('')
plt.show()

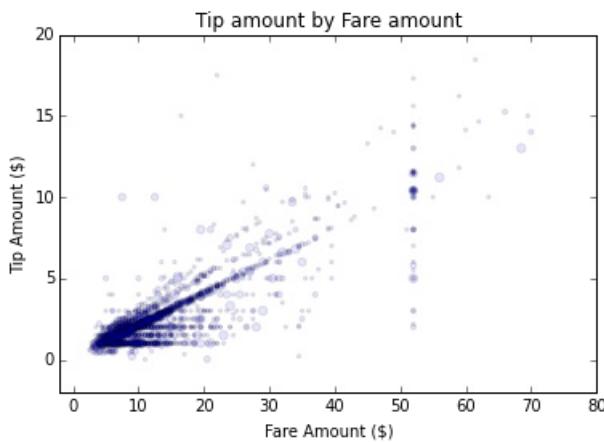
# PLOT TIP BY PASSENGER COUNT
ax2 = sqlResults.boxplot(column=['tip_amount'], by=['passenger_count'])
ax2.set_title('Tip amount by Passenger count')
ax2.set_xlabel('Passenger count')
ax2.set_ylabel('Tip Amount ($)')
plt.suptitle('')
plt.show()

# 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).setElast
icNetParam(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("areaUn
derROC")
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("areaUn
derROC")
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

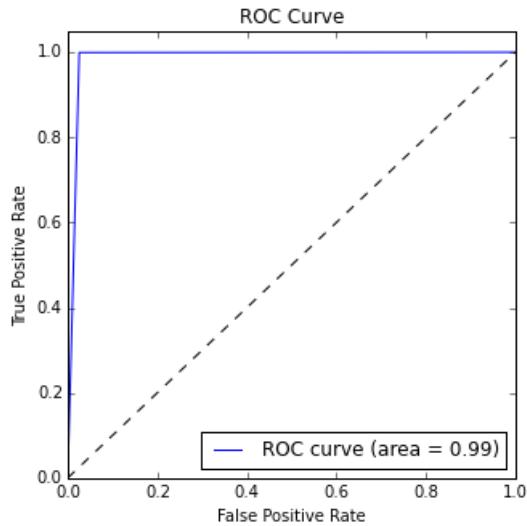
sqlResults['probFloat'] = sqlResults.apply(lambda row: row['probability'].values()[0][1], axis=1)
predictions_pddf = sqlResults[["tipped","probFloat"]]

# 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 datestamp = Calendar.getInstance().getTime().toString.replaceAll(" ", ".").replaceAll(":", "_");
val modelName = "GBT_Classification_"
val filename = modelDir.concat(modelName).concat(datestamp)
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 `GBTRegressor()` 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 timestamp = Calendar.getInstance().getTime().toString.replaceAll(" ", ".").replaceAll(":", "_");
val modelName = "LinearRegression_"
val filename = modelDir.concat(modelName).concat(timestamp)
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.

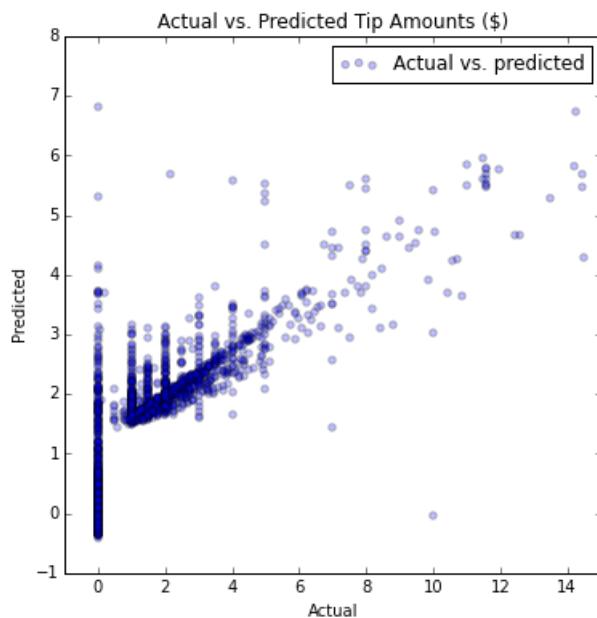
```

# 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 GBTRRegressor().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).set
FeatureSubsetStrategy("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),(2,6),(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(datestamp)
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

11/22/2017 • 7 min to read • [Edit Online](#)

This article 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 articles 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

You can try Azure Machine Learning for free. No credit card or Azure subscription is required. [Get started now.](#)

Create 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. The new features are expected 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: Add temporal features for a 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 the "cnt" 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 exists 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 leftmost branch, each of these branches contains an [Execute R Script](#) module, in which the 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.



A comparison of the performance results of the four models is summarized in the following table:

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

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 the 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.

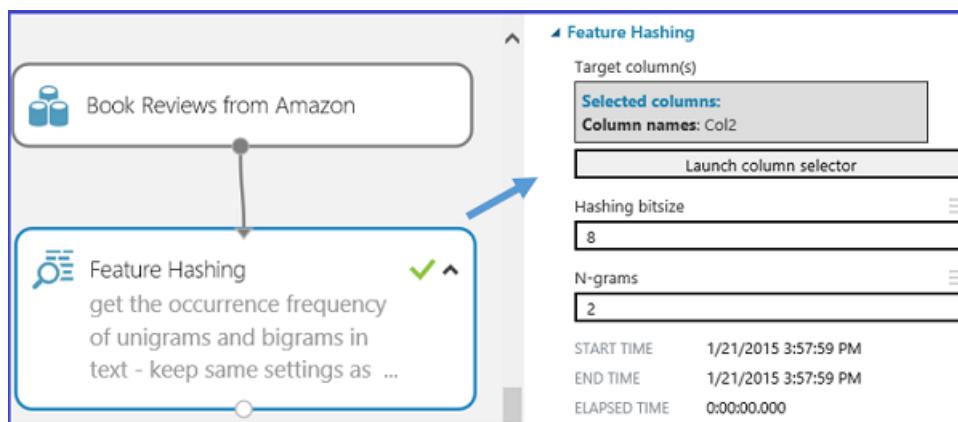
Example 2: Creating features in text mining

Feature engineering is widely applied in tasks related to text mining, such as document classification and sentiment analysis. For example, when you 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.

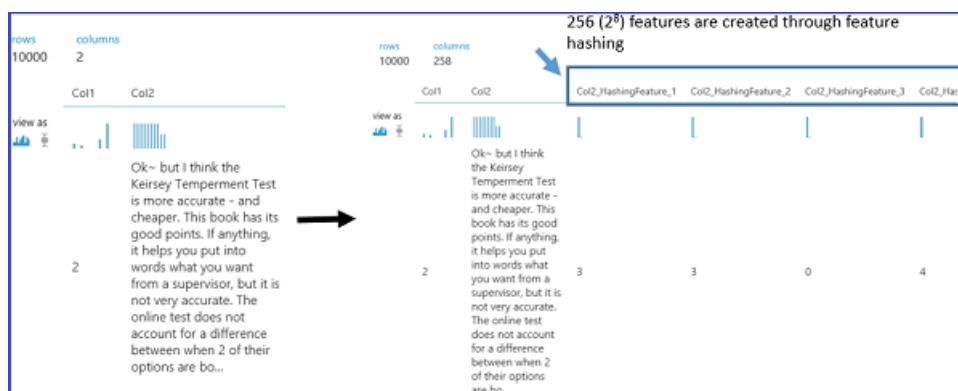
To achieve this task, a technique called **feature hashing** is applied to efficiently turn arbitrary text features into indices. Instead of associating each text feature (words/phrases) to a particular index, this method functions by applying a hash function to the features and using their hash values 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, 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 value 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 feature 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.

It is not always necessarily to perform feature engineering or feature selection. Whether it is needed or not depends on the data to hand or collected, the algorithm selected, and the objective of the experiment.

Create features for Azure blob storage data using Panda

11/22/2017 • 2 min to read • [Edit Online](#)

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 explore and manipulate a dataset, downloaded it from the blob source to a local file. Then load it into 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 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.

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, 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 to consume it in Azure Machine Learning

To consume the data in Azure Machine Learning that you have explored, sampled or featurized, upload the data to an Azure blob. Additional features can be created in the Azure Machine Learning Studio as well. The following steps show how to upload the data:

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>

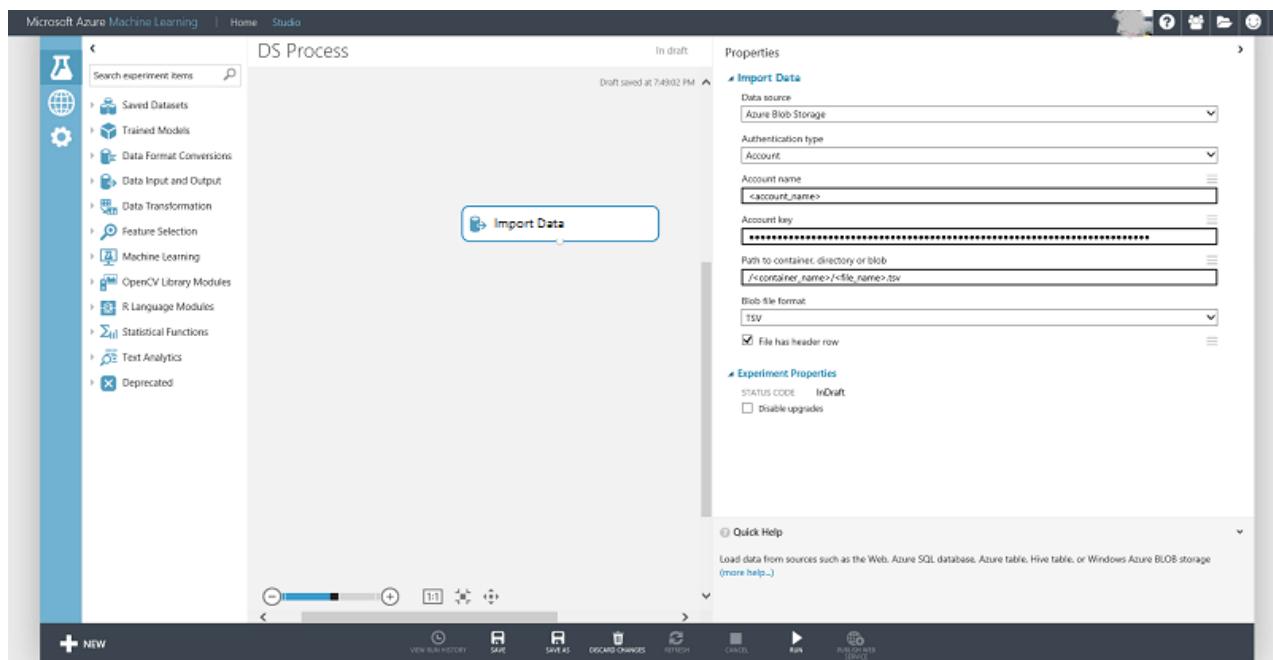
output_blob_service=BlobService(account_name=STORAGEACCOUNTNAME,account_key=STORAGEACCOUNTKEY)
localfileprocessed = os.path.join(os.getcwd(),LOCALFILENAME) #assuming file is in current working directory

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 following screen shot:



Create features for data in SQL Server using SQL and Python

2/2/2018 • 5 min to read • [Edit Online](#)

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. You can use SQL or a programming language like Python to accomplish this task. Both approaches 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>). Here are some useful things to understand about location data before creating features from the field:

- The sign indicates whether we are north or south, east or west on the globe.
- A nonzero hundreds digit indicates longitude, not latitude is being used.
- The tens digit gives a position to about 1,000 kilometers. It gives 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 indicates, 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 by 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/ff701710.aspx> 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)
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>

```

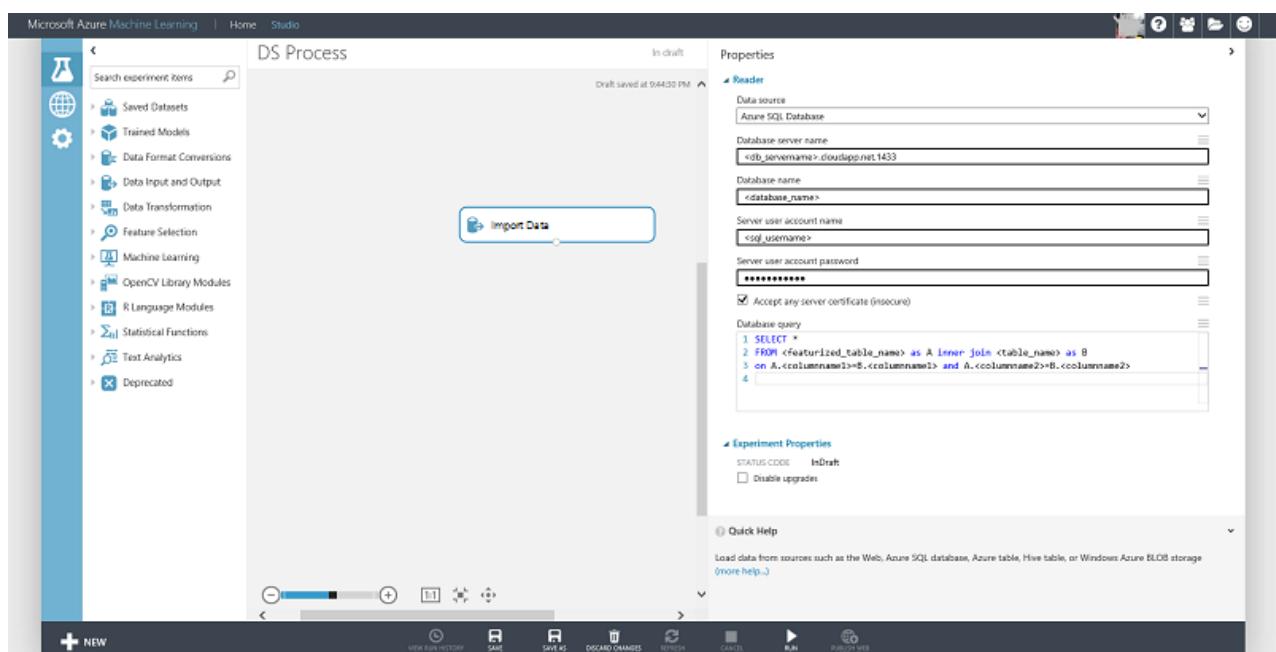
These location based features can be further used to generate additional count features as described earlier.

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. [Here is an example of how to do this using pyodbc](#). 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. For comparison, see [Process Azure Blob data in your data science environment](#). Load the data from the database into a pandas data frame to process it further. The process of connecting to the database and loading the data into the data frame is documented 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 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 data frame as covered in topics [Create features for Azure blob storage data using Panda](#).

Create features for data in a Hadoop cluster using Hive queries

2/28/2018 • 7 min to read • [Edit Online](#)

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, 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, non-numeric categorical variables must be converted into numeric features when the models being used only take numeric features. This conversion is done by replacing each non-numeric level with a numeric risk. This section shows 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 risk > 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). This section shows 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 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 has the pattern like *03/26/2015 12:04:39*, the ' 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 function 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,
    ${hiveconf:R}*2*2*atan((1-sqrt(1-pow(sin((dropoff_latitude-pickup_latitude)
        *${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 this 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. This section discusses some parameters that users can tune to 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 errors. This error can be avoided 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 128 MB, then any data of size less than and up to 128 MB is stored in a single block. Data that is larger than 128 MB is allotted extra blocks.
3. Choosing a 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;
```

4. **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, set:

```
set hive.auto.convert.join=true;
```

5. **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 64 MB.

As we can see, given the data size, tuning these parameters by "setting" them allows us to tune the number of mappers used.

6. Here are a few other more **advanced options** for optimizing Hive performance. These allow you to set the memory allocated to map and reduce tasks, and can be useful in tweaking performance. 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)

11/22/2017 • 4 min to read • [Edit Online](#)

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

You can try Azure Machine Learning for free. No credit card or Azure subscription is required. [Get started now.](#)

The engineering and selection of features is one part of the Team Data Science Process (TDSP) outlined in the article [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.

Filter features from your data - feature selection

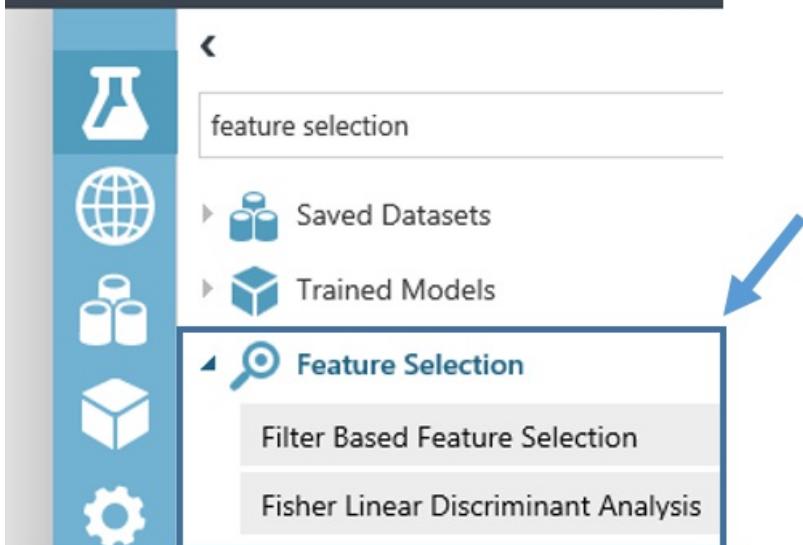
Feature selection is a process that is commonly applied for the construction of training datasets for predictive modeling tasks such as classification or regression tasks. The goal is to select a subset of the features from the original dataset that reduce its dimensions by using a minimal set of features to represent the maximum amount of variance in the data. This subset of features is used 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 the model training process more efficient. Efficiency is particularly important for learners that are expensive to train such as support vector machines.

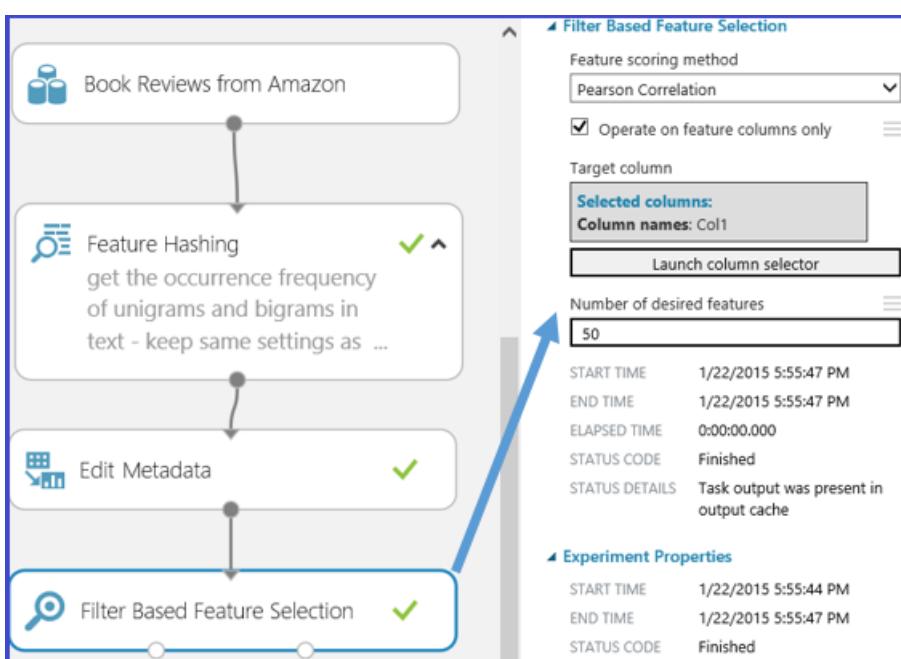
Although feature selection does seek to reduce the number of features in the dataset used to train the model, it is not referred to by the term "dimensionality reduction". Feature selection methods extract a subset of original features in the data without changing them. Dimensionality reduction methods 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 convenience, continue using the text mining example. Assume that you 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" that contains 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](#) produces a dataset containing 50 features together with the target attribute "Col1". The following figure shows the flow of this experiment and the input parameters:



The following figure shows the resulting datasets:

view as	Col1	Col2_Hash	Col2_HashFeature_203	Col2_HashFeature_146	Col2_HashFeature_122	Col2_HashFeature_109
...	2	6	1	2	6	5
	2	6	4	7	5	3
	1	9	2	1	3	0
	2	5	2	2	3	0
	2	3	1	1	0	0
	2	11	6	5	5	5
	1	2	2	3	1	1
	2	4	3	2	1	1
	2	2	2	6	3	3
	2	2	0	1	0	0
	1	11	4	3	5	5
	2	1	0	5	0	0
	2	2	1	2	2	2
	1	1	3	3	3	3

Each feature is scored based on the Pearson Correlation between itself and the target attribute "Col1". The features with top scores are kept.

The corresponding scores of the selected features are shown in the following figure:

view as	Col1	Col2_Hash	Col2_HashFeature_203	Col2_HashFeature_146	Col2_HashFeature_122	Col2_HashFeature_109
...	1	0.083607	0.060681	0.05716	0.056381	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.

It is not always necessarily to perform feature engineering or feature selection. Whether it is needed or not depends on the data collected, the algorithm selected, and the objective of the experiment.

How to choose algorithms for Microsoft Azure Machine Learning

1/8/2018 • 15 min to read • [Edit Online](#)

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 points 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 many 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 applies. Sometimes none of them are 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 [Azure AI 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.

Supervised learning is a popular and useful type of machine learning. With one exception, all 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, it's 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](#).

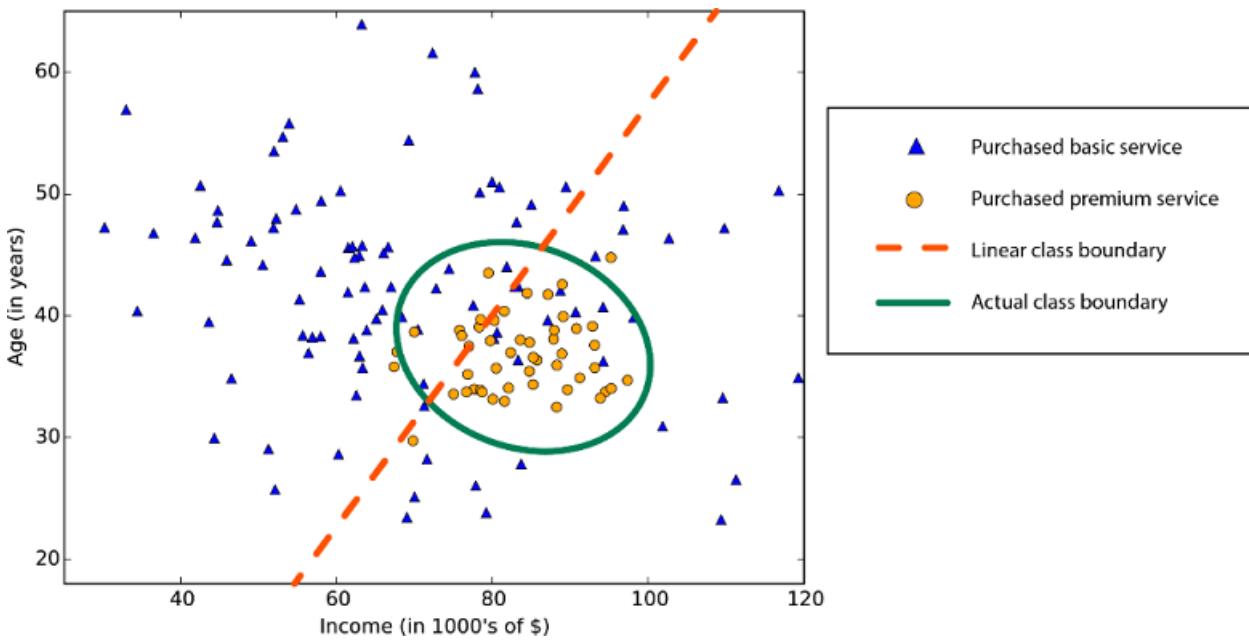
Training time

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.

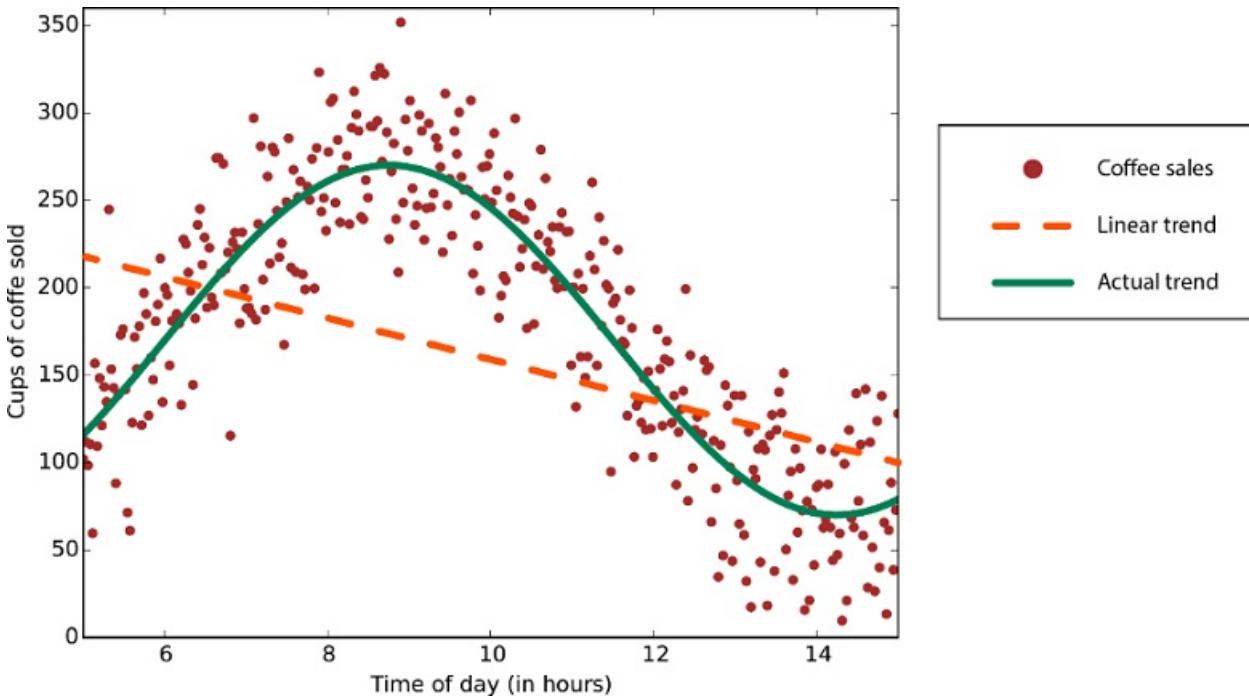
Linearity

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					
logistic regression		●	●	5	

ALGORITHM	ACCURACY	TRAINING TIME	LINEARITY	PARAMETERS	NOTES
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:

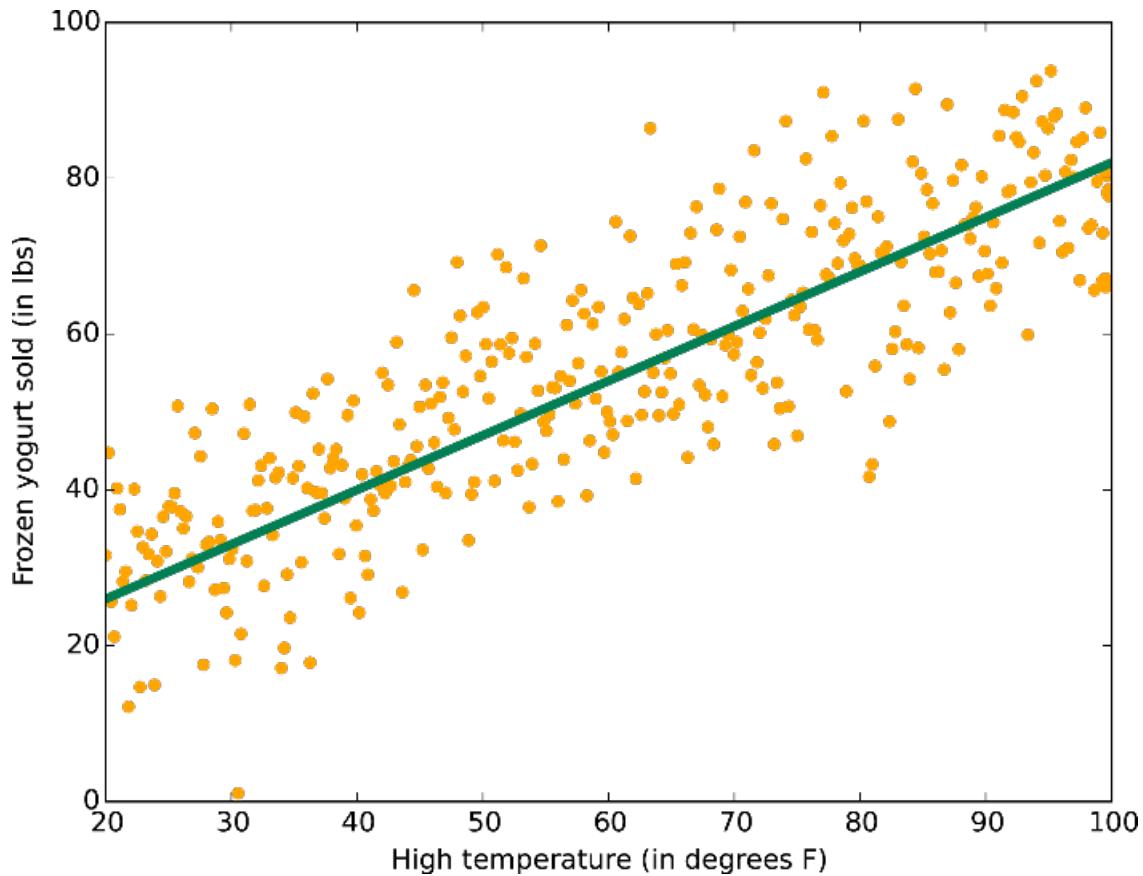
● - shows excellent accuracy, fast training times, and the use of linearity

○ - shows good accuracy and moderate training times

Algorithm notes

Linear regression

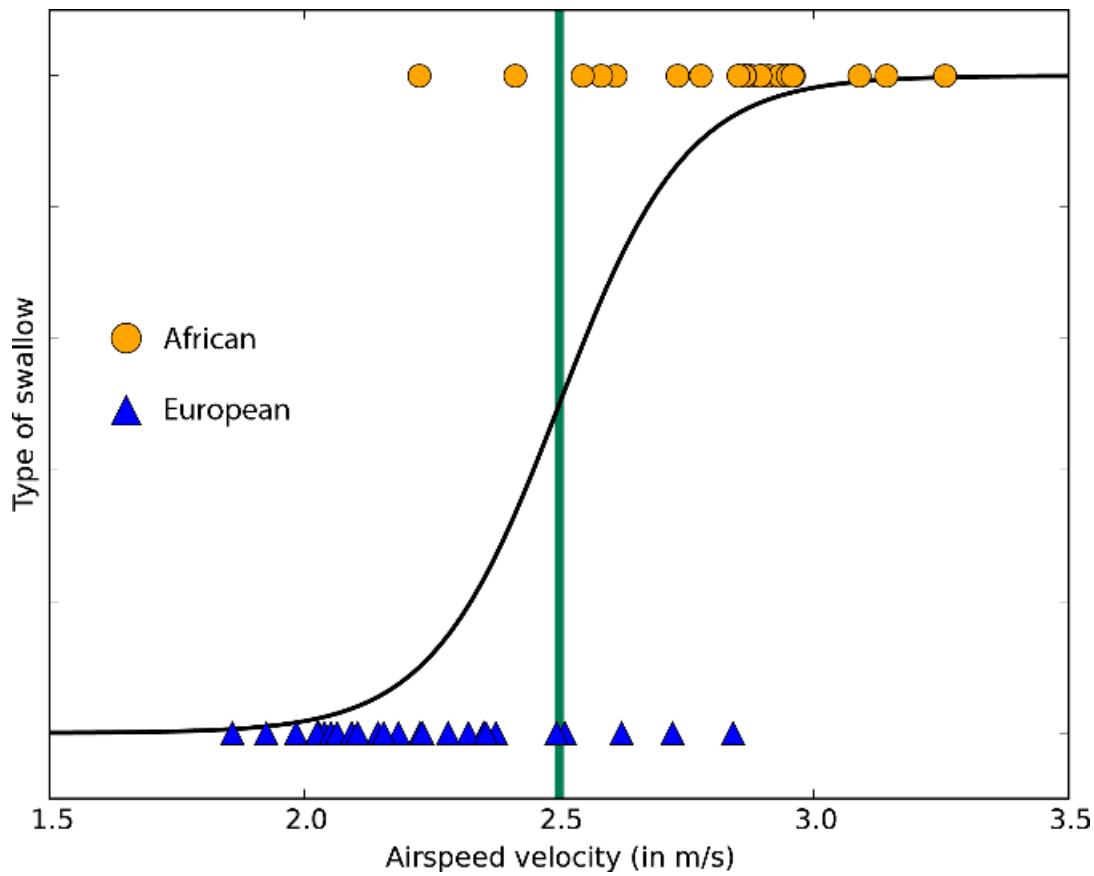
As mentioned previously, [linear regression](#) fits a line (or plane, or hyperplane) to the data set. It's a workhorse, simple and fast, but it may be overly simplistic for some problems. Check here for a [linear regression tutorial](#).



Data with a linear trend

Logistic regression

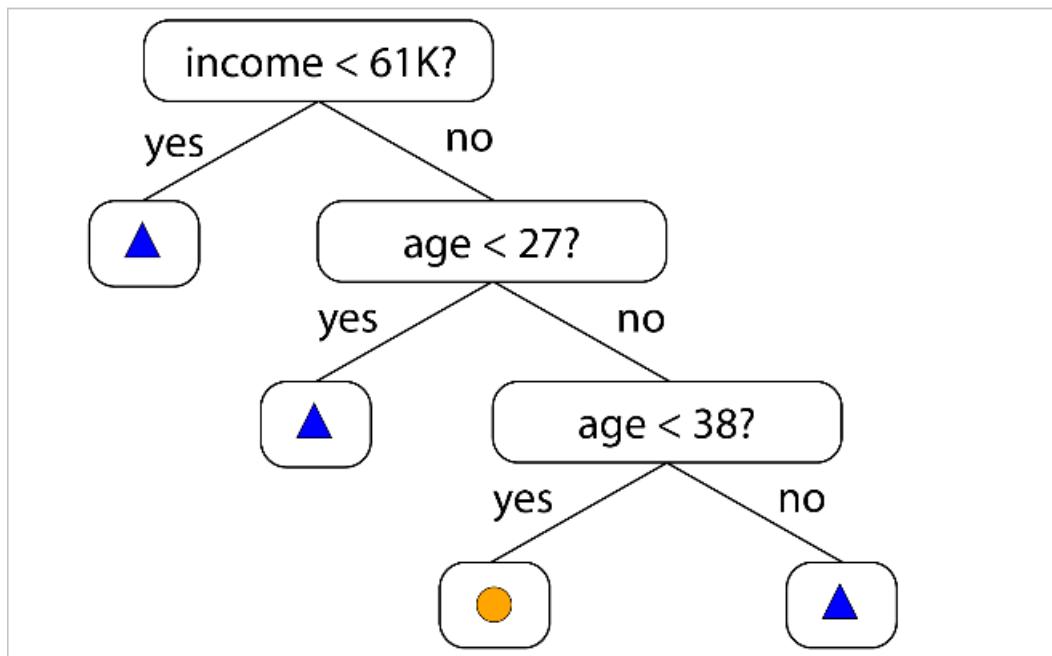
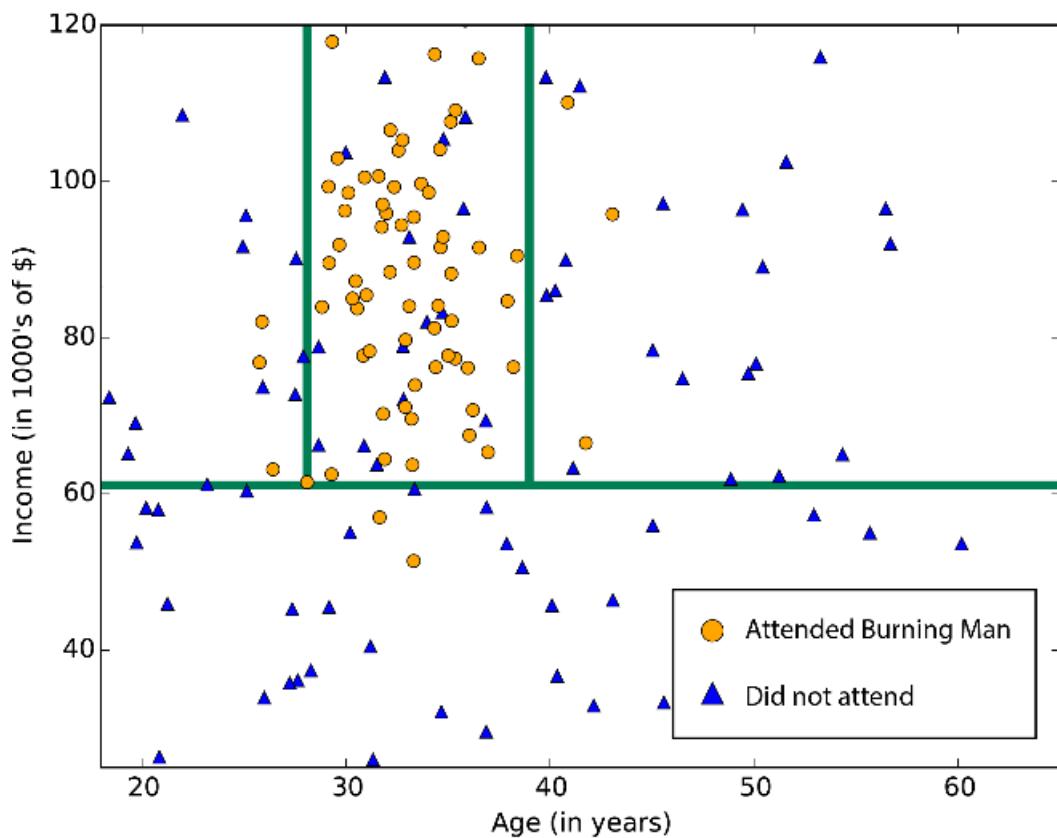
Although it confusingly includes 'regression' in the name, logistic regression is actually a powerful tool for [two-class](#) and [multiclass](#) classification. It's fast and simple. The fact that it uses an 'S'-shaped curve instead of a straight line makes it a natural fit for dividing data into groups. 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. This is 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.

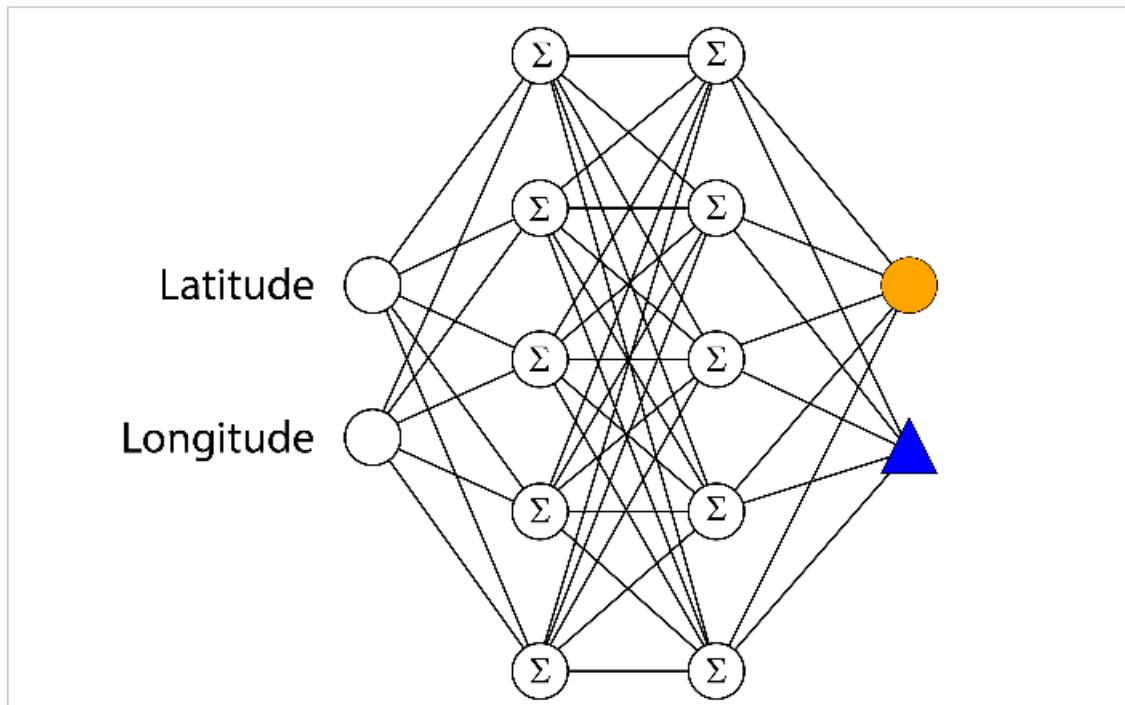
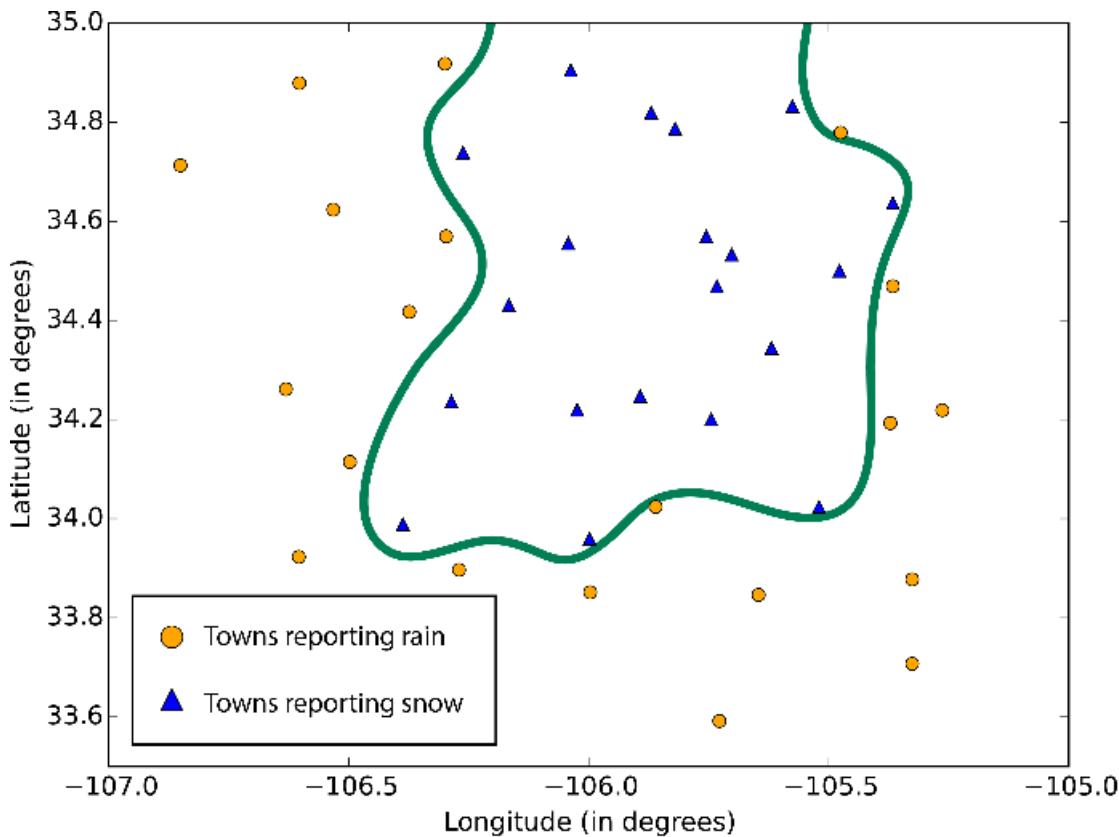
Boosted decision trees avoid overfitting by limiting how many times they can subdivide and how few data points are allowed in each region. The algorithm constructs a sequence of trees, each of which learns to compensate for the error left by the tree before. The result is a very accurate learner that 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.

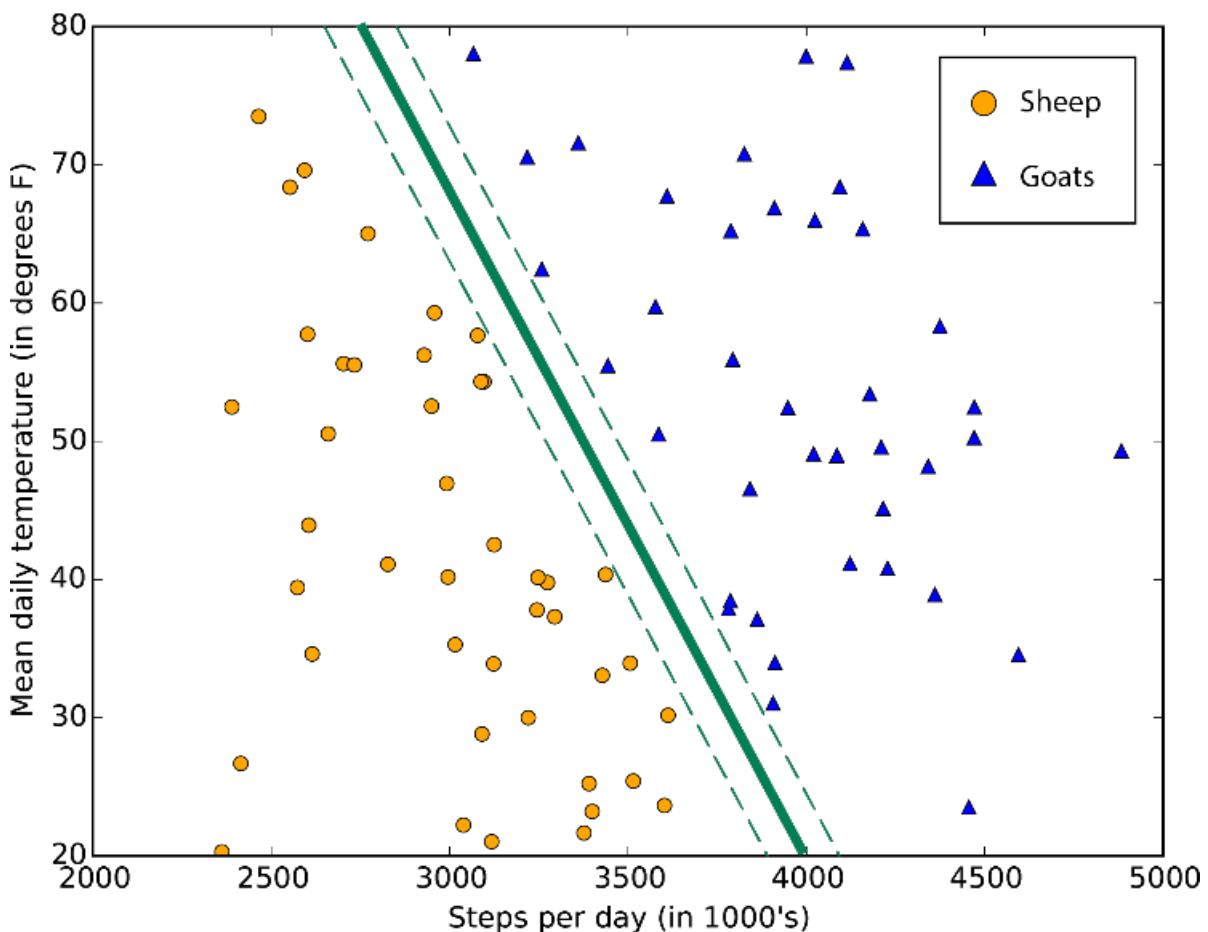


boundaries learned by neural networks can be complex and irregular

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 down the problem 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

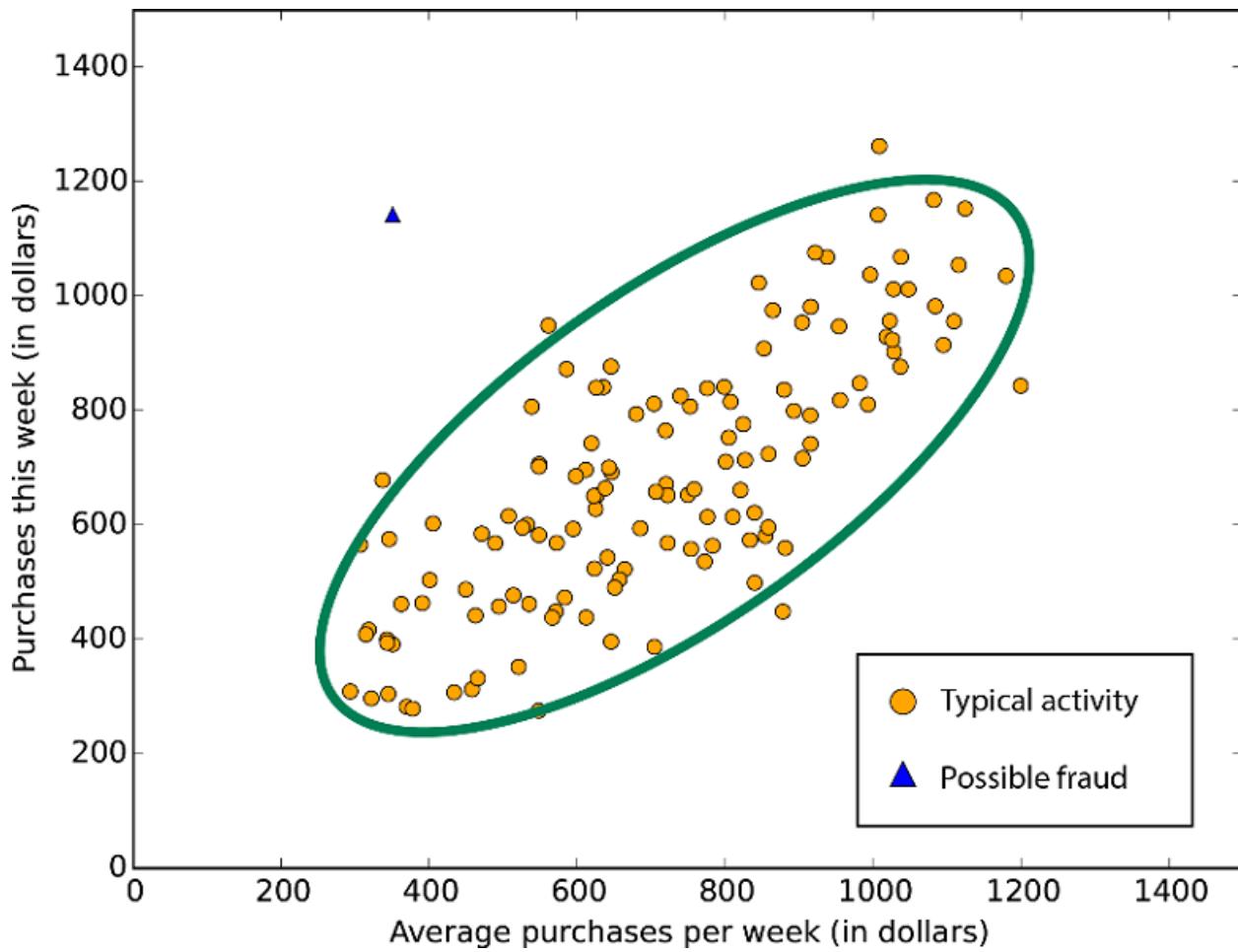
Bayesian methods have a highly desirable quality: they avoid overfitting. They do this by making some assumptions beforehand about the likely distribution of the answer. Another byproduct of this approach is that they have very few parameters. Azure Machine Learning has both Bayesian algorithms for both 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 a 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](#).

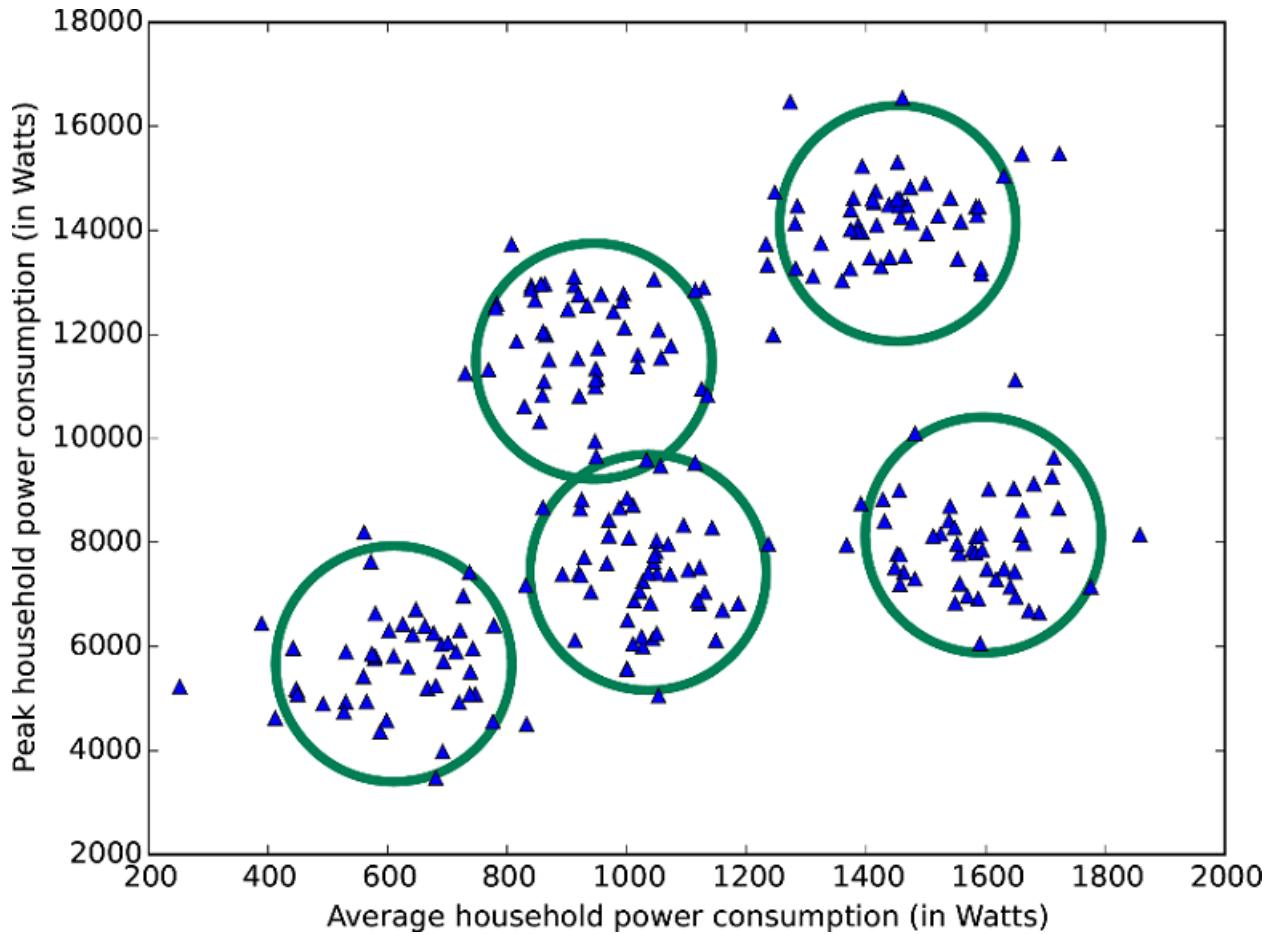
Specialized algorithms

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](#)),
- anomaly detection (one based on [principal components analysis](#) and one based on [support vector machines](#))
- clustering ([K-means](#))

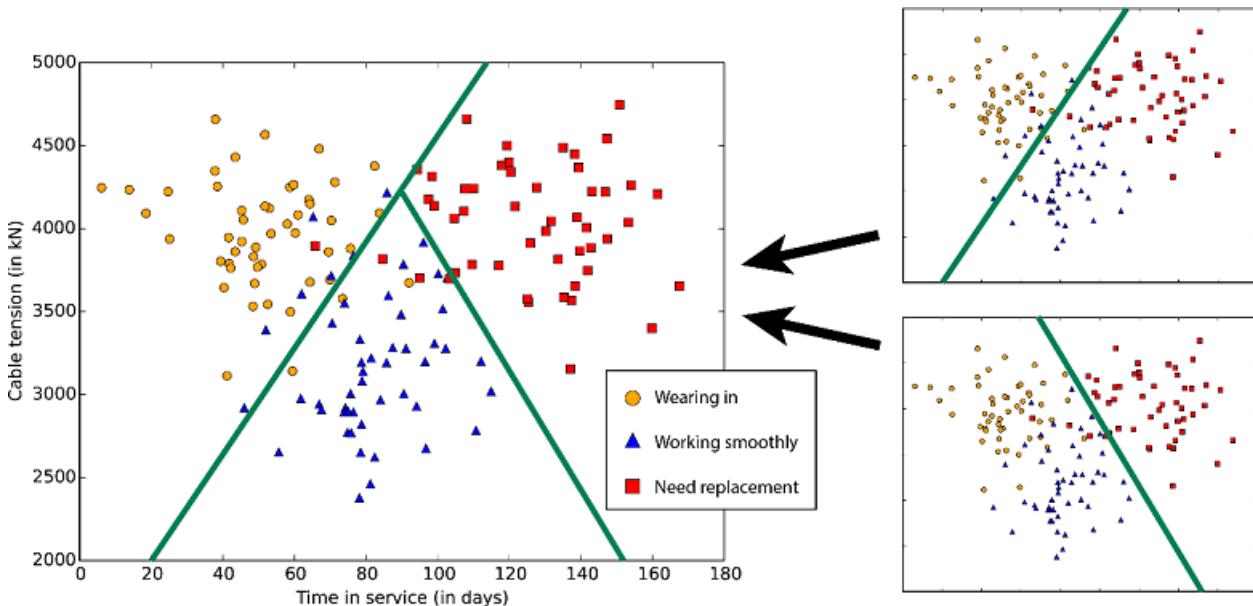


PCA-based anomaly detection - the vast majority of the data falls into a stereotypical distribution; points deviating dramatically from that distribution are suspect



A data set is grouped into five 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](#) 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](#) 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

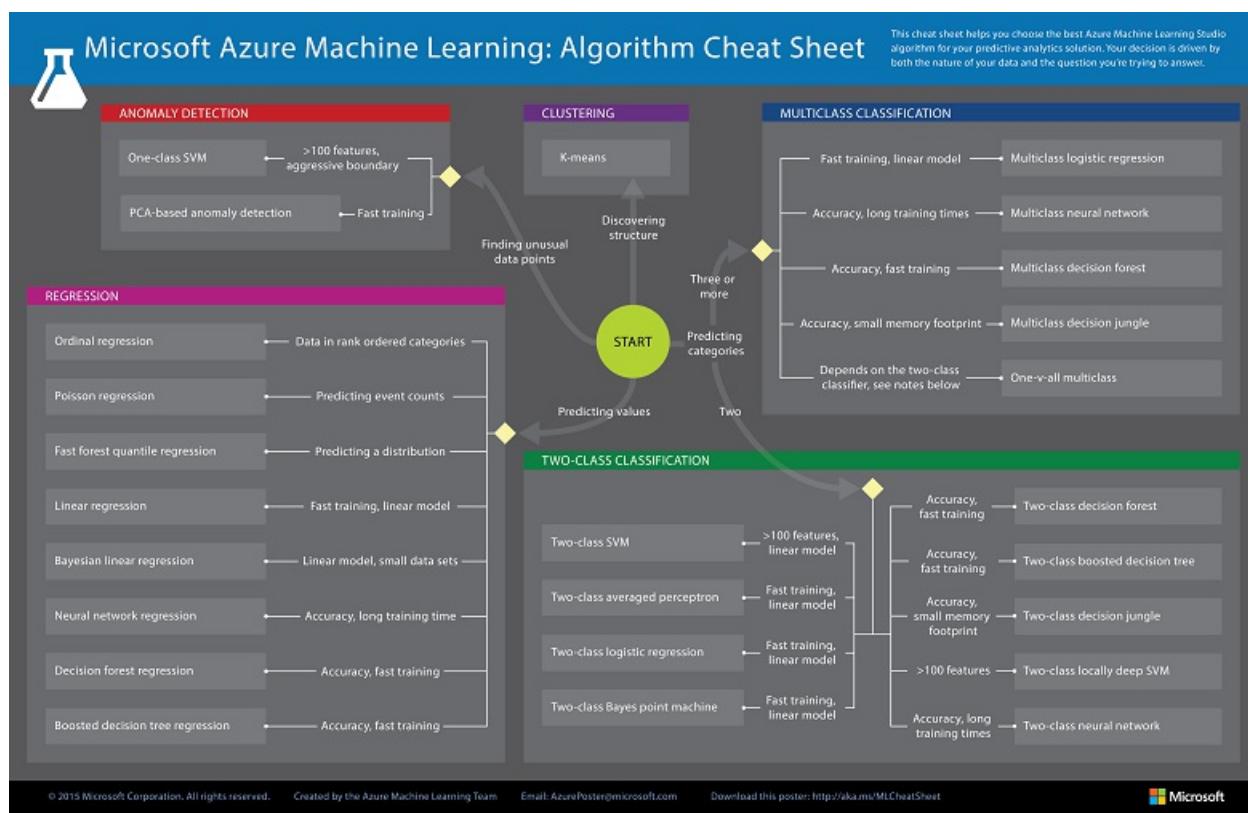
1/8/2018 • 5 min to read • [Edit Online](#)

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

You can try Azure Machine Learning for free. No credit card or Azure subscription is required. [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 it's not always possible to know beforehand which is the best fit. 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 [Azure AI 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, identifying new photos with the correct animal or assigning 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, they assume 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, setting it to 1 in cases where the category applies, 0 where it doesn't.

Deploy models in production

2/13/2018 • 1 min to read • [Edit Online](#)

Production deployment enables a model to play an active role in a business. Predictions from a deployed model can be used for business decisions.

Production platforms

There are various approaches and platforms to put models into production. Here are a few options:

- [Model deployment in Azure Machine Learning](#)
- [Deployment of a model in SQL-server](#)
- [Microsoft Machine Learning Server](#)

NOTE

Prior to deployment, one has to insure the latency of model scoring is low enough to use in production.

NOTE

For deployment using Azure Machine Learning Studio, see [Deploy an Azure Machine Learning web service](#).

A/B testing

When multiple models are in production, it can be useful to perform [A/B testing](#) to compare performance of the models.

Next steps

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 [Example walkthroughs](#) article. They illustrate how to combine cloud, on-premises tools, and services into a workflow or pipeline to create an intelligent application.

Cortana Intelligence AppSource publishing guide

9/25/2017 • 6 min to read • [Edit Online](#)

Overview

AppSource is the single destination for Business Decision Makers (BDMs) to discover and seamlessly try business solutions/apps built by partners and evaluated by Microsoft. Watch [this video](#) to learn how AppSource works.

As a Microsoft Partner, you can really benefit from publishing on AppSource if you have:

- Built an intelligent solution/app using [Cortana Intelligence Suite](#).
- Your solution or app addresses a specific business problem.
- You built modules or intellectual property that your customers can reuse relatively quickly in a predictable manner.

Take a look at [Cortana Intelligence solutions](#) that are already published on AppSource.

This article will walk through the steps to get a partner's Cortana Intelligence solution published to AppSource

Getting started

1. In the [Partner Community Benefits Guide](#) (PDF), see page 9 to get listed as an Advanced Analytics partner.
2. Complete the [Submit your app](#) form.

For the question "*Choose the products that your app is built for*", check the **Other** checkbox and list "Cortana Intelligence" in the edit control.

3. Before a Cortana Intelligence app can get onboarded to AppSource, it must get certified by Cortana Intelligence's Partner Solution Technology team. To get that process started, kindly share details about your app by filling in survey form at [Cortana Intelligence solution evaluation for AppSource publishing](#). This site is password protected and the site has instructions on how to get the password.

Solution evaluation criteria

Here is the list of criteria the app needs to meet

1. App needs to address specific business use case problem within a given functional area in a repeatable manner with minimal configurations for predefined value propositions implementable within a short period.
2. Solution should use at least one of the following components:
 - HDInsight
 - Machine Learning
 - Data Lake Analytics
 - Stream Analytics
 - Cognitive Services
 - Bot Framework
 - Analysis Services
 - Microsoft R Server stand alone
 - R services on SQL 2016 or HDInsight Premium
3. Solution should be generating at least \$1000 a month per customer using DPOR/CSP.
4. Solution should use the Azure Active Directory federated single signon (AAD federated SSO) with consent

enabled for user authentication and resource access controls. You will need to show to the evaluation team that your solution is AAD federated SSO enabled before your app can be onboarded to AppSource.

To see what it means to be AAD federated SSO enabled, seek to position 02:35 in [AppSource trial experience walk through](#) video. If your app is not enabled with AAD federated SSO yet, here is some documentation about it

- a. [One-click sign in](#).
 - b. [Integrating applications with Azure Active Directory](#).
5. Use Power BI in your solution: Optional but highly recommended as it is proven to generate higher number of leads.

DevCenter account setup

This is the process of registering your company to become a publisher with Microsoft. If you are already a registered publisher with an existing DevCenter account, share the email ID associated with your DevCenter account. Once your application is approved for publishing, you will get access to full documentation about [Cloud Portal Manage publisher profile](#)

If you don't have one, below are the key steps to setup a DevCenter account.

1. Create a Microsoft account [here](#).
2. Go to the Microsoft DevCenter [registration page](#) and click "sign up".
3. When prompted for payment, use the code that we provided to you. If you don't have a code, contact appsourcecissupport@microsoft.com.
4. Select the [country/region](#) in which you live, or where your business is located. **You won't be able to change this later.**
5. Select your [developer account type](#) For AppSource, select **Company**. For a company account, be sure to review these [guidelines](#).
6. Enter the contact info you want to use for your developer account. For detailed step by step instruction on how to setup DevCenter account, see the instructions [here](#).

Provide info for Microsoft sellers

One of the key value propositions of AppSource for partners is to be able to collaborate with Microsoft Sellers in positioning partner apps in front of potential customers.

Fill up [Partner Solution Info for Microsoft Sellers](#) and send it to appsourcecissupport@microsoft.com. This is required step for a Cortana Intelligence app to get approved for publishing.

Build a compelling customer walkthrough on AppSource

First, take a look at [Neal Analytics Inventory Optimization](#) on AppSource. Every app entry in AppSource has title, summary (100 chars max), description (1300 chars max), images, videos (optional), pdf documents apart from entry point for a trial experience. Partners should leverage all these to build a compelling customer experience.

Partners should think of the content they put on AppSource as an end to end sales orchestration. Customers read the title and description to understand the value proposition and then go through images & videos to understand what the solution is about. Case studies help potential customers see how existing customers are getting value.

All this should make the customer feel interested and wanting to know more. Think of these as pitch decks based presentation a good technical sales person would walk the new customers through. The suggested format of description is to break up the text into sub-sections based on value propositions, each with highlighted with a sub-heading.

Visitors usually glance over the "offer summary" field and sub-headings to get gist of what the app addresses and why should they consider the app in just a quick glance. So, it is important to get the user's attention give them a reason to read on to get the specifics.

See what these partners have done.

- [Neal Analytics Inventory Optimization](#)
- [Plexure Retail Personalization](#)
- [AvePoint Citizen Services](#)

The final act of sales is to show a demo of the app/solution showing how the value proposition is delivered. That is exactly what a customer trial experience on AppSource is meant for. A good demo will do the following:

- Re-summarize the value proposition of the app in short and list out the personas within the customer firm who would interact with the solution
- Tell a story and sets the context about a sample customer dealing with specific issues
- Explain how the situation can generally devolve and impact the customer (before) VS what would happen if the solution is in use. This can be shown using PowerBI dashboards etc.
- Summarize how the solution makes it happen by using any specific Machine Learning algorithms etc.

The content being added in AppSource should be of high quality and stitched up enough to enable the following:

- A partner's technical sales folks should be using it for their sales orchestration. Your sales teams using it is a good sign that you can expect MS sales folks also being able to do the same. This will enable customer point of contact to be able to consistently relay the same story to their team mates and higher ups to get budget and approvals before a purchase deal can be done.
- A customer visiting the site organically can go through the content all by themselves and feel excited to respond back to partner communication to move ahead with next steps.

That's why partners should think of the content they put on AppSource as an end to end sales orchestration. Based on the story line and features to be shown in trial experience, type of offer can be decided.

Publish your app on the publishing portal

Once we have evaluated the above steps for your application, you will get access to the publishing portal and can see [How to publish a Cortana Intelligence offer via Cloud Partner Portal](#) for detailed instructions on next steps.

If you need information about any of the fields, email appsourcecissupport@microsoft.com.

My app is published on AppSource - now what?

Firstly, Congratulations on getting your app published. The level of returns you get from publishing your app on AppSource heavily depends on how you influence the target audience. See [Growth-Hacking your Cortana Intelligence app on AppSource](#) for more details of how you can maximize the returns.

If you have any questions or suggestions, kindly reach out to us at appsourcecissupport@microsoft.com.

Cortana Intelligence solution evaluation tool

12/7/2017 • 8 min to read • [Edit Online](#)

Overview

You can use the Cortana Intelligence solution evaluation tool to assess your advanced analytics solutions for compliance with Microsoft-recommended best practices. Microsoft is excited to work with our partners (ISVs / SIs) to provide high-quality solutions for customers, resellers and implementation. This guide will walk through the process of using the Solution evaluation tool with your solution and describe the specific best practices in checks for.

Getting started

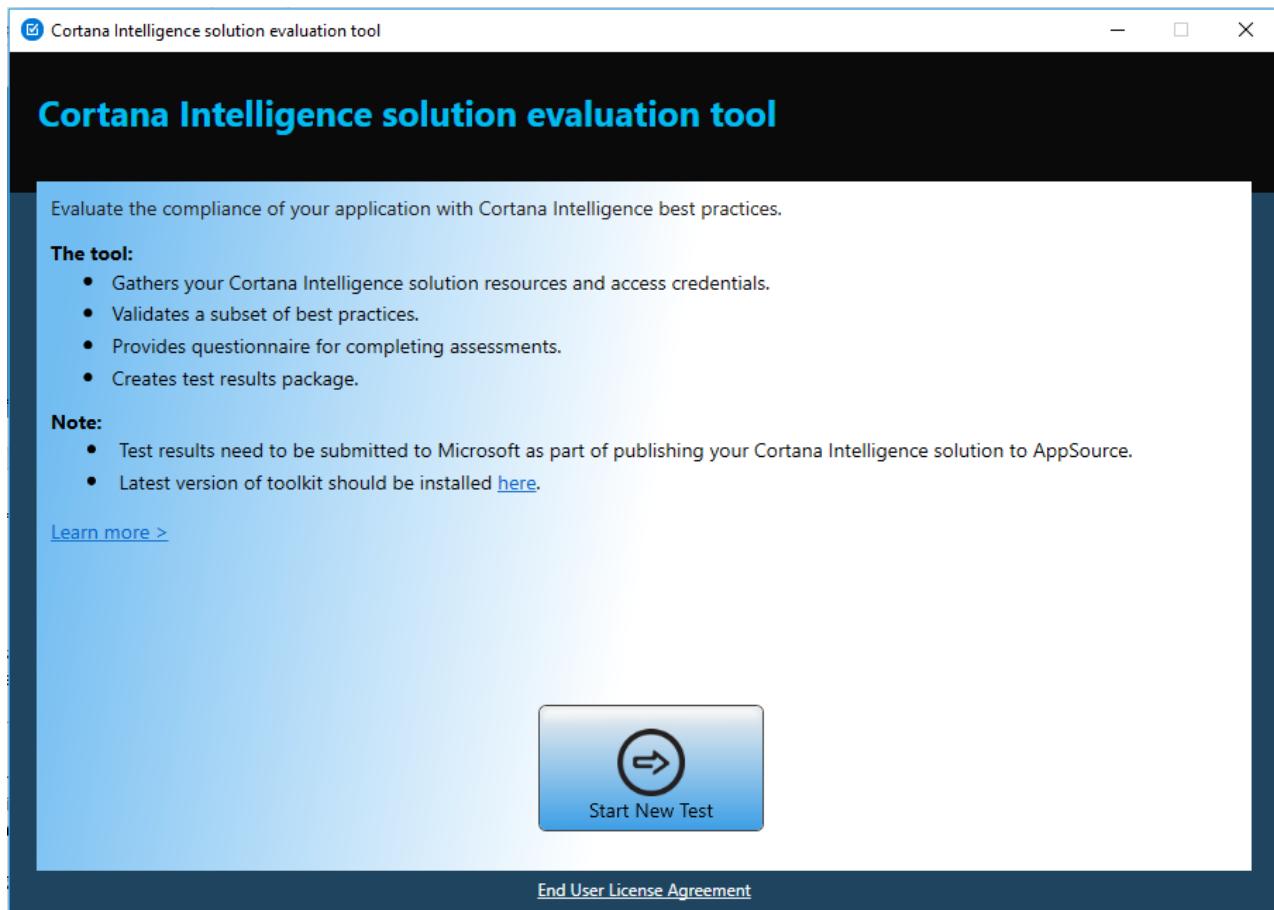
Please [download](#) and install the Cortana Intelligence solution evaluation tool.

Prerequisites:

- Windows 10: [Official site for Windows 10](#)
- Azure Powershell: [Install and configure Azure PowerShell](#).

Identifying your app

After installation completes, open the tool and begin your first evaluation.



Provide identifying information about your solution.

Cortana Intelligence solution evaluation tool

Information Test Prerequisites Execute Test Assessment View Report

Test Information

*Indicates required field.

Test Name*	: <input type="text"/>	Organization Name*	: <input type="text"/>
Application Name*	: <input type="text"/>	Application Version	: <input type="text"/>

Verify your application's readiness

This tool will:

- Login to your Azure account to identify your app's resources.
- Evaluate the configuration and architecture of your app.
- Ask a series of questions regarding your app.
- Request explanation/justification for any non-compliance.

Additional Information:

- See the [Cortana Intelligence app criteria](#)



[!\[\]\(ce64933ae03a59683c2699ec6bd194fd_img.jpg\) Back](#) [!\[\]\(85aabff03ca89ec313ee5827e7ad733a_img.jpg\) Next](#) [!\[\]\(83776bdc4659be40795d98ed6095e657_img.jpg\) Cancel](#)

Connect to your Azure subscription and provide the Resource Group containing your app.

Cortana Intelligence solution evaluation tool

Information Test Prerequisites Execute Test Assessment View Report

Cortana Intelligence - Azure Connectivity

*Indicates required field.

Subscription*	: <input type="text"/>	Disconnect
Resources Group Name*	: <input type="text"/>	Load Resources

Specify the data role for each data storage resource listed below:

Name	Resource Type	Data Role
[REDACTED]	Microsoft.Web/sites	
[REDACTED]	Microsoft.AnalysisServices/servers	Consumption
[REDACTED]	Microsoft.DataLakeStore/accounts	Internal
[REDACTED]	Microsoft.Storage/storageAccounts	Ingestion
[REDACTED]	Microsoft.Compute/virtualMachines	
[REDACTED]	Microsoft.Compute/disks	
[REDACTED]	Microsoft.Network/networkInterfaces	
[REDACTED]	Microsoft.Network/publicIPAddresses	

[!\[\]\(07007bc2b2f3124408434726cfb72749_img.jpg\) Back](#) [!\[\]\(2c55bbb3f6826b07192e76435706aec6_img.jpg\) Next](#) [!\[\]\(146568efd38bfcc4e927d427dd166761_img.jpg\) Cancel](#)

Once the resource group has been loaded, please select the resources that are included in your solution and identify the accessibility of any data resources as either:

- Ingestion
- Consumption
- Internal

We use this information to better understand how your solution is utilizing various components and to ensure user-facing components are consistent with best practices.

Ingestion

Ingestion in this case means any data sources that are used to pull in data from outside the solution or that any services outside the solution use to push data into it.

Consumption

Consumption in this case means any datasets that are used to push data to end users, either directly or indirectly. For example:

- Datasets used in direct query from PowerBI.
- Datasets queried in a WebApp.

NOTE

If a specific resource is used for both ingestion and consumption, please choose **Consumption**.

Internal

Use internal for any data resources that are used only in internal application processing.

Next, you will be prompted to provide valid credentials for any databases specified in the prior step:

Cortana Intelligence solution evaluation tool

Information Test Prerequisites Execute Test Assessment View Report

Cortana Intelligence - Test Prerequisites Documentation | Feedback | Support

*Indicates required field.
Verify all the selected databases connection
Note: Please ensure the client IP address added as a firewall rule at all your database servers you are selecting.

SQL Databases and SQL Data Warehouses

ciscert2xy3bldasjvivusrv/certdbdw_temp

Connection String*: Server = tcp:xxxxxxxx.database.windows.net, 1433; Initial Catalog = xxxxxxxx; Persist Security Info = False; User ID = xxxx
User ID*: Password*:

ciscert2xy3bldasjvivusrv/ciscert2db

Connection String*: Server = tcp:xxxxxxxx.database.windows.net, 1433; Initial Catalog = xxxxxxxx; Persist Security Info = False; User ID = xxxx
User ID*: Password*:

ciscert2xy3bldasjvivusrvsecondary/ciscert2db

Connection String*: Server = tcp:xxxxxxxx.database.windows.net, 1433; Initial Catalog = xxxxxxxx; Persist Security Info = False; User ID = xxxx
User ID*: Password*:

Test Connection

Back Next Cancel

Solution test cases

The solution tool will perform a collection of automated tests on your solution.

Cortana Intelligence solution evaluation tool

The interface shows a navigation bar with five steps: Information, Test Prerequisites, Execute Test, Assessment, and View Report. The 'Execute Test' step is highlighted in blue. Below the navigation bar, the title 'Cortana Intelligence - Test Execution' is displayed, along with links to Documentation, Feedback, and Support. The main content area is titled 'Test Cases' and contains several sections: 'Security', 'Availability', 'Scalability', and 'Other'. Each section lists specific requirements with their status (e.g., Log, Pass, Fail) and execution progress (e.g., Executing...). A progress bar at the bottom indicates 'Please wait...'.

Test Cases	Logs	Status
Security		
Databases should use Azure Active Directory authentication.	Log	Pass
Azure WebApps should use Azure Active Directory authentication.	Log	Pass
Datasets accessible to end-users should support role-based access control.	Log	Fail
Azure Data Lake Store should use at-rest encryption.	Log	Pass
Azure SQL and Azure SQL DW should use encryption.	Log	Executing...
Azure blob storage should use encryption.	Log	Executing...
Any Virtual Machines must be deployed from the Azure Marketplace.	Log	Executing...
Availability		
Datasets accessed by end users should support many concurrent users.	Log	Executing...
Azure SQL resources should have a read-only replica for failover.	Log	Executing...
Azure SQL DW should have geo-redundant backups enabled.	Log	Executing...
Virtual machines should be configured with availability sets.	Log	Executing...
Scalability		
Cortana Intelligence apps should include a scalable big data platform.	Log	Executing...
Cortana Intelligence apps should include dedicated "ingestion" data environments.	Log	Executing...
Azure SQL DW should use Polybase for data ingestion.	Log	Executing...
Other		
Should be using Data Factory.	Log	Executing...
<small>Machine Learning models should be retuned using Azure Data Factory.</small>	Log	Executing...

Please wait...

Back Next Cancel

After the tests complete, you will be asked to provide an explanation or justification for why your solution does not comply with the requirement.

Cortana Intelligence solution evaluation tool

The interface shows a navigation bar with five steps: Information, Test Prerequisites, Execute Test, Assessment, and View Report. The 'Execute Test' step is highlighted in blue. Below the navigation bar, the title 'Cortana Intelligence - Test Execution - Business Justification' is displayed, along with links to Documentation, Feedback, and Support. The main content area is titled 'Test Cases' and lists failed test cases with their details and status. At the bottom, there are 'Back', 'Next', and 'Cancel' buttons.

Test Cases	Logs	Status
Datasets accessible to end-users should support role-based access control.		
Access the the data warehouse instance is strictly limited to power uses	Log	Fail
Cortana Intelligence apps should include dedicated "ingestion" data environments.	Log	Fail

Back Next Cancel

For example, if your solution publishes to Azure SQL DW, the evaluation tests require you to also publish to Azure Analysis Services.

Your solution may use IaaS virtual machines running Sql Server Analysis Services instead of Azure Analysis Services. This would be an acceptable reason for failure of the test.

Packaging your evaluation results

After completing the test cases, your evaluation package will be exported to a zip file and you will be asked to provide feedback on the evaluation tool.

You need to share this test results zip file with Microsoft for your solution to be evaluated before getting approval to be added to AppSource

The screenshot shows the Cortana Intelligence solution evaluation tool. The top navigation bar includes tabs for 'Information', 'Test Prerequisites', 'Execute Test', 'Assessment', and 'View Report'. The 'View Report' tab is currently selected. Below the tabs, there's a 'Report' section with fields for 'User Name*' and 'Email Address*'. A question asks 'How likely are you to recommend Cortana Intelligence solution evaluation tool to a friend or colleague?' with a scale from 1 (Not at all likely) to 10 (Extremely likely). A text area for 'What is the primary reason for the score you chose?' is present. At the bottom, there's a note about generating a test case report and a 'Generate Report' button. A yellow warning box contains a privacy statement: 'Please note that the test results package may contain personally identifiable information which may be present in the information we collect from the operating system. This includes information such as directory structures and file names; application name, description and version; system information; event log (error data); record of crash/dump (not the actual dump file); user account.' It also links to the 'Microsoft Privacy Statement'. Navigation buttons 'Back' and 'Close' are at the bottom right.

Above section of this article covers various features of the tool, now let us review types of best practices that this tool evaluates.

Security evaluation considerations

Databases should use Azure Active Directory authentication

Any Azure SQL or Azure SQL DW resources in the sloution should be enabled with Azure Active Directory (AAD) authentication. AAD provides a single place to manage all of your identities and roles.

FOR MORE INFORMATION ABOUT	SEE THIS ARTICLE
AAD with SQL Database and SQL Data Warehouse	Use Azure Active Directory Authentication for authentication with SQL Database or SQL Data Warehouse
Configure and manage AAD	Configure and manage Azure Active Directory authentication with SQL Database or SQL Data Warehouse

FOR MORE INFORMATION ABOUT	SEE THIS ARTICLE
Azure WebApps authentication	Authentication and authorization in Azure App Service
Configure WebApps with AAD	How to configure your App Service application to use Azure Active Directory login

Datasets accessible to end-users should support role-based access control

While executing the evaluation tool, you will be asked to specify any reporting or publishing resources. It is assumed that these resources are intended for access by end-users, not developers. These resources should be provide role-based access control (RBAC) in order to ensure that end users are only able to access authorized data.

Specifically, any of the following Azure resources can be configured with RBAC and are considered acceptable:

- Secure HDInsight, see [An introduction to Hadoop security with domain-joined HDInsight clusters](#)
- Azure SQL, see [AAD authentication with Azure SQL](#)
- Azure Analysis Services, see [Manage database roles and users for Azure Analysis Services](#)
- Azure SQL Data Warehouse (Be aware that because SQL DW does support RBAC, it is not recommended for direct end-user access.)

If you are using a different resource type that supports RBAC, please specify that in the test case justification.

Azure Data Lake Store should use at-rest encryption

Azure Data Lake Store (ADLS) supports at-rest encryption by default using ADLS-managed encryption keys. You may also configure encryption using Azure Key Vault.

For information about specifying ADLS encryption settings, [Create an Azure Data Lake Store account](#).

Azure SQL and Azure SQL Data Warehouse should use encryption

Azure SQL and Azure SQL DW both support Transparent Data Encryption (TDE), which provides real-time encryption and decryption of both data and log files.

FOR MORE INFORMATION ABOUT	SEE THIS ARTICLE
Transparent Data Encryption (TDE)	Transparent Data Encryption
Azure SQL Data Warehouse with TDE	SQL Data Warehouse Encryption TDE TSQL
Configure Azure SQL with TDE	Transparent Data Encryption with Azure SQL Database
Configure Azure SQL with Always Encrypted	SQL Database Always Encrypted Azure Key Vault

In addition to TDE, Azure SQL also supports Always Encrypted, a new data encryption technology that ensures data is encrypted not only at-rest and during movement between client and server, but also while data is in use while executing commands on the server.

Any Virtual Machines must be deployed from the Azure Marketplace

In order to provide a consistent level of security across AppSource, we require that any virtual machines deployed as part of a Cortana Intelligence solution be certified and published on the Azure Marketplace.

To search the current list of Azure Marketplace images, see [Microsoft Azure Marketplace](#).

For information on how to publish a virtual machine image for Azure Marketplace, see [Guide to create a virtual machine image for the Azure Marketplace](#).

Scalability evaluation considerations

Cortana Intelligence solutions should include a scalable big data platform

Cortana Intelligence solutions should scale to very large data sizes. In Azure, this means they should include one of the two Petabyte-scale data platforms:

- Azure Data Lake Store
- Azure SQL Data Warehouse

If your solution does not require support for these data sizes or if you are using an alternative data platform, please explain this in the test case justification.

Cortana Intelligence solutions should include dedicated ingestion data environments

Cortana Intelligence solutions should generally avoid directly inserting data into relational data sources. Instead, raw data should be stored in an unstructured environment, with idempotent inserts/updates into any relational stores using Azure Data Factory.

For more information on copying data with Azure Data Factory, [Tutorial: Create a pipeline with Copy Activity using Visual Studio](#).

Azure SQL Data Warehouse should use PolyBase for data ingestion

Azure SQL DW supports PolyBase for highly scalable, parallel data ingestion. PolyBase allows you to use Azure SQL DW to issue queries against external datasets stored in either Azure Blob Storage or Azure Data Lake Store. This provides superior performance to alternative methods of bulk updates.

For instructions on getting started with PolyBase and Azure SQL DW, see [Load data with PolyBase in SQL Data Warehouse](#).

For more information about best practices with PolyBase and Azure SQL DW, see [Guide for using PolyBase in SQL Data Warehouse](#).

Availability evaluation considerations

Datasets accessible to end users should support a large volume of concurrent users

While executing the evaluation tool, you will be asked to specify any reporting or publishing resources. It is assumed that these resources are intended for access by end-users, not developers. These resources should support medium-large numbers of concurrent users.

Specifically, Azure SQL Data Warehouse should NOT be the sole data source available to end users. If Azure SQL DW is provided as a resource for Power Users, Azure Analysis Services should be made available to typical users.

For more information about Azure SQL DW concurrency limits, see [Concurrency and workload management in SQL Data Warehouse](#).

For more information about Azure Analysis Services, see [Analysis Services Overview](#).

Azure SQL resources should have a read-only replica for failover

Azure SQL databases support geo-replication to a secondary instance. This instance can then be used as a failover instance to provide high availability applications.

For more information about geo-replication for Azure SQL databases, see [SQL Database GEO Replication overview](#).

For instructions on how to configure geo-replication for Azure SQL, see [Configure active geo-replication for Azure SQL Database with Transact-SQL](#).

Azure SQL Data Warehouse should have geo-redundant backups enabled

Azure SQL DW supports daily backups to geo-redundant storage. This geo-replication ensures you can restore the

data warehouse even in situations where you cannot access snapshots stored in your primary region. This feature is on by default and should not be disabled for Cortana Intelligence solutions.

For more information about Azure SQL DW backups and restoration, see here [SQL Data Warehouse Backups](#).

Virtual machines should be configured with availability sets

Azure virtual machines should be configured in availability sets in order to minimize the impact of planned and unplanned maintenance events.

For more information about Azure virtual machine availability, see [Manage the availability of Windows virtual machines in Azure](#).

Other evaluation considerations

Cortana Intelligence apps should use a centralized tool for data orchestration

Using a single tool for managing and scheduling data movement and transformation ensures consistency around mission-critical data. It provides a clear logic around retry logic, dependency management, alert/logging, etc. We recommend the use of [Azure Data Factory](#) for data orchestration in Azure.

If you are using a tool other than Azure Data Factory for data orchestration, please describe which tool or tools you are using.

Azure Machine Learning models should be retrained using Azure Data Factory

Azure Machine Learning (AzureML) provides easy to use tools for the creation and deployment of predictive modeling and machine learning pipelines. However, it is important that production deployments of these AzureML models is not based on a single fixed dataset, but instead adapts to the shifting dynamics of real-world phenomena.

For more information on creating retraining web services in AzureML, see [Retrain Machine Learning models programmatically](#).

For more information about automating the model training process using Azure Data Factory, see [Updating Azure Machine Learning models using Update Resource Activity](#).

Existing documentation

[Microsoft Azure Certified to grow your cloud business](#)

[Microsoft Azure Certified for Cortana Intelligence](#)

Machine Learning Anomaly Detection API

1/6/2018 • 10 min to read • [Edit Online](#)

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 [Azure AI 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: `Inputs` 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.

```
{  
    "Inputs": {  
        "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"  
    }  
}
```

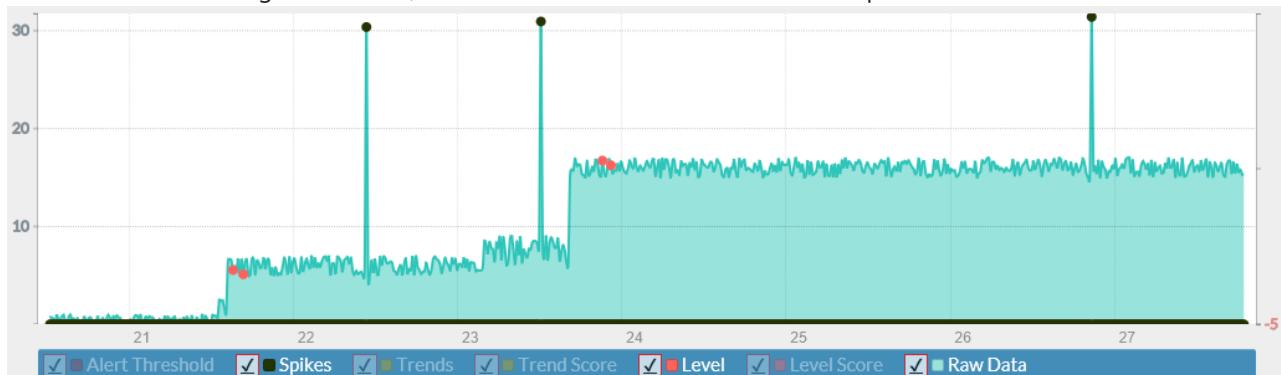
Sample Response

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", "-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", "-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", "PosTrendAlert", "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

Detector Category	Detector	Description	Input Parameters	Outputs
Spike Detectors	ZSpike Detector	Detect spikes and dips based on how far the datapoints are from their mean	<code>zspikedetector.sensitivity</code> : 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
Slow Trend Detector	Slow Trend Detector	Detect slow positive trend as per the set sensitivity	<code>trenddetector.sensitivity</code> : 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	<code>bileveldetector.sensitivity</code> : 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
<code>detectors.historyWindow</code>	History (in # of data points) used for anomaly score computation	500	integer	10-2000	Time-series dependent
<code>detectors.spikesdips</code>	Whether to detect only spikes, only dips, or both	Both	enumerated	Both, Spikes, Dips	Both
<code>bileveldetector.sensitivity</code>	Sensitivity for bidirectional level change detector.	3.25	double	None	3.25-5 (Lesser values mean more sensitive)
<code>trenddetector.sensitivity</code>	Sensitivity for positive trend detector.	3.25	double	None	3.25-5 (Lesser values mean more sensitive)
<code>tspikedetector.sensitivity</code>	Sensitivity for TSpike Detector	3	integer	1-10	3-5 (Lesser values mean more sensitive)

INPUT PARAMETERS	DESCRIPTION	DEFAULT SETTING	TYPE	VALID RANGE	SUGGESTED RANGE
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.

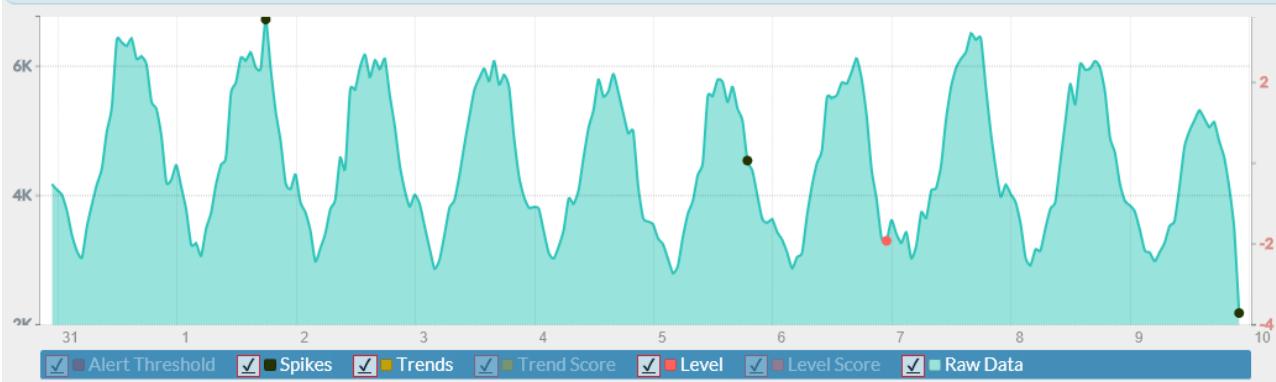
OUTPUTS	DESCRIPTION
Time	Timestamps from raw data, or aggregated (and/or) imputed data if aggregation (and/or) missing data imputation is applied
Data	Values from raw data, or aggregated (and/or) imputed data if aggregation (and/or) missing data imputation is applied
TSpike	Binary indicator to indicate 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)

INPUT PARAMETERS	DESCRIPTION	DEFAULT SETTING	TYPE	VALID RANGE	SUGGESTED RANGE
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.enabled	Whether seasonality analysis is to be performed	true	boolean	true, false	Time-series dependent
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

OUTPUTS	DESCRIPTION
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
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

Cortana Intelligence Solution Template Playbook for predictive maintenance in aerospace and other businesses

12/7/2017 • 48 min to read • [Edit Online](#)

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.

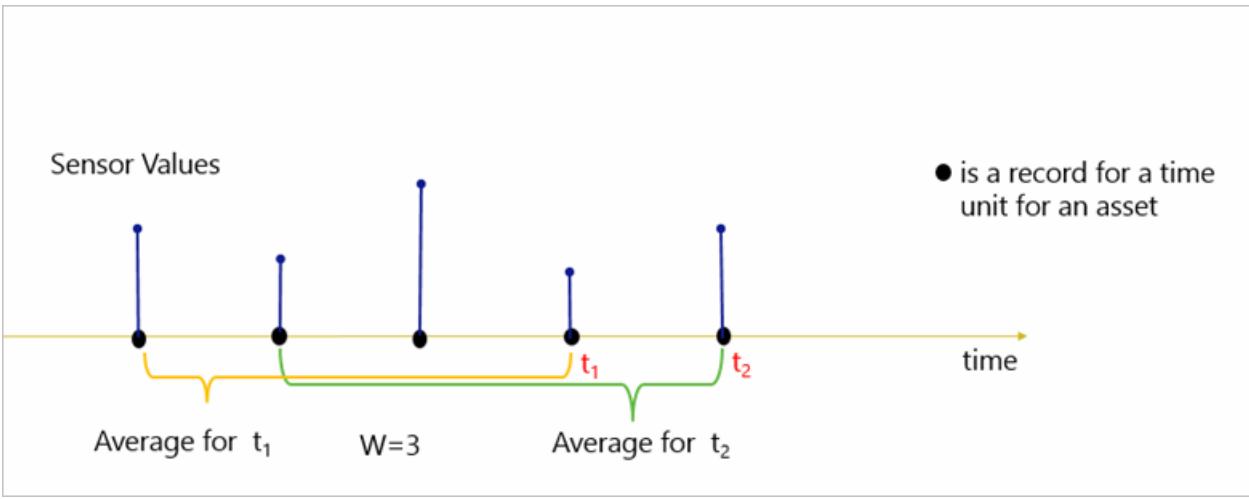


Figure 1. Rolling aggregate features

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.

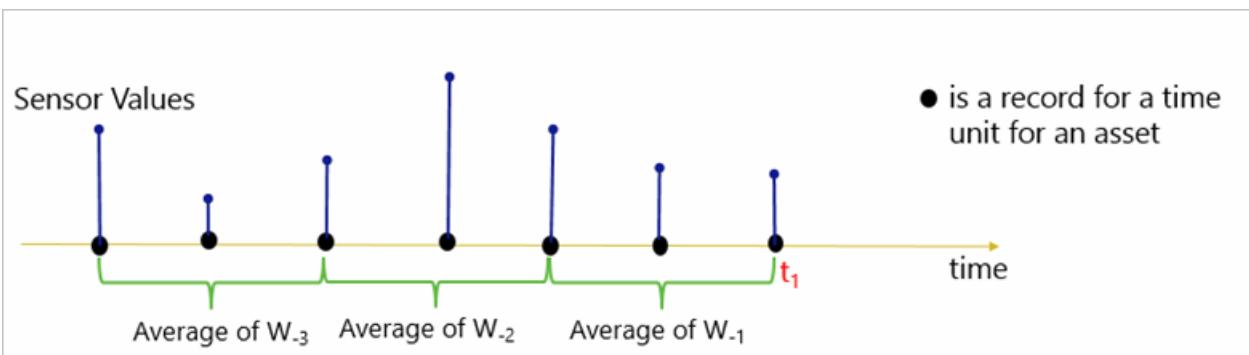


Figure 2. Tumbling aggregate features

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).

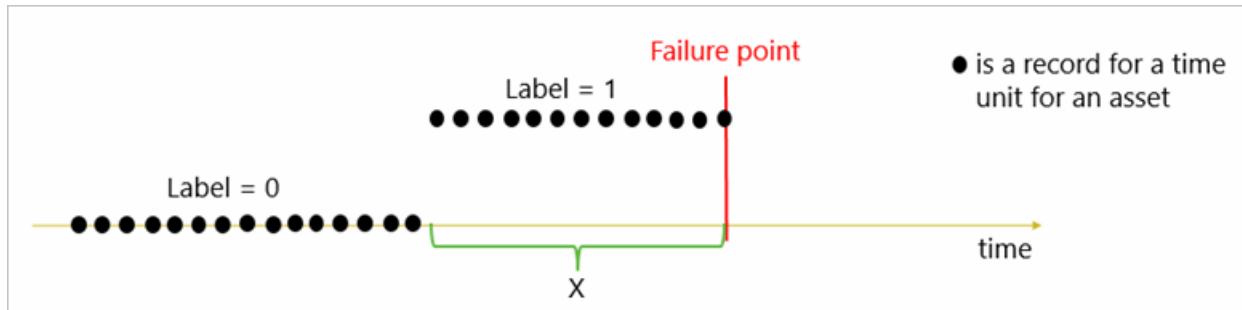


Figure 3. Labeling for binary classification

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).

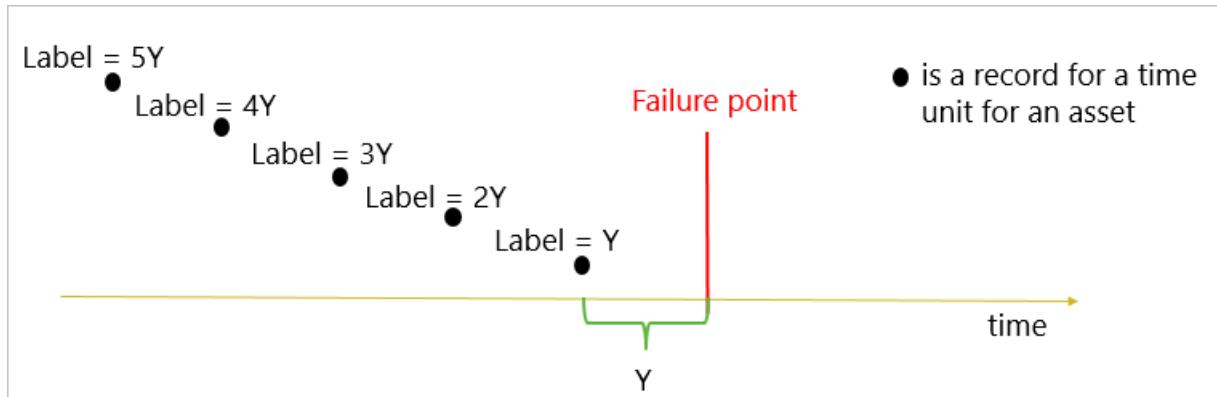


Figure 4. Labeling for regression

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).

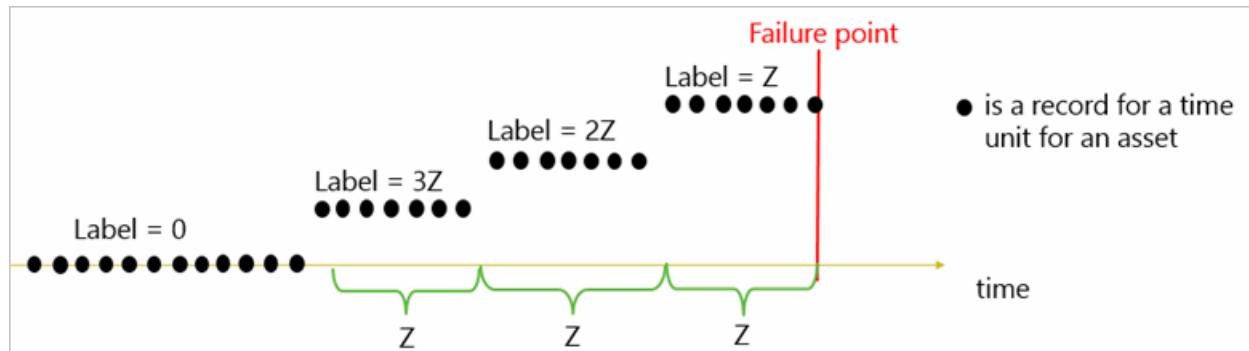


Figure 5. Labeling for multiclass classification for failure time prediction

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).

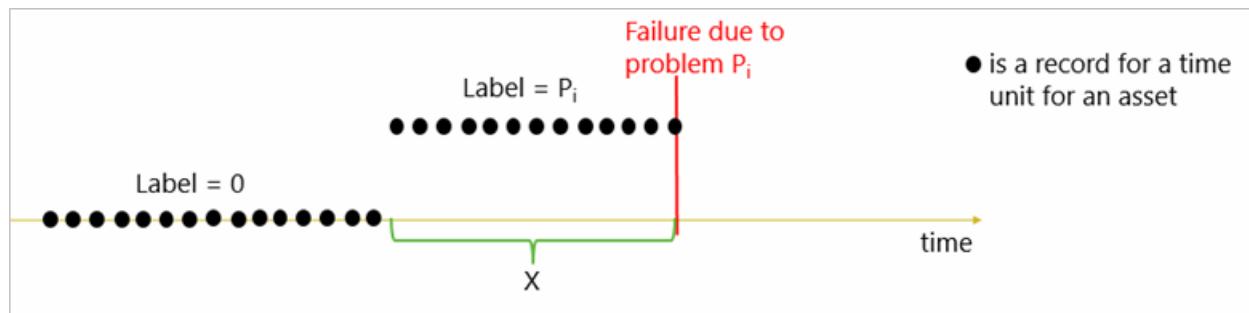


Figure 6. Labeling for multiclass classification for root cause prediction

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-premises 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.

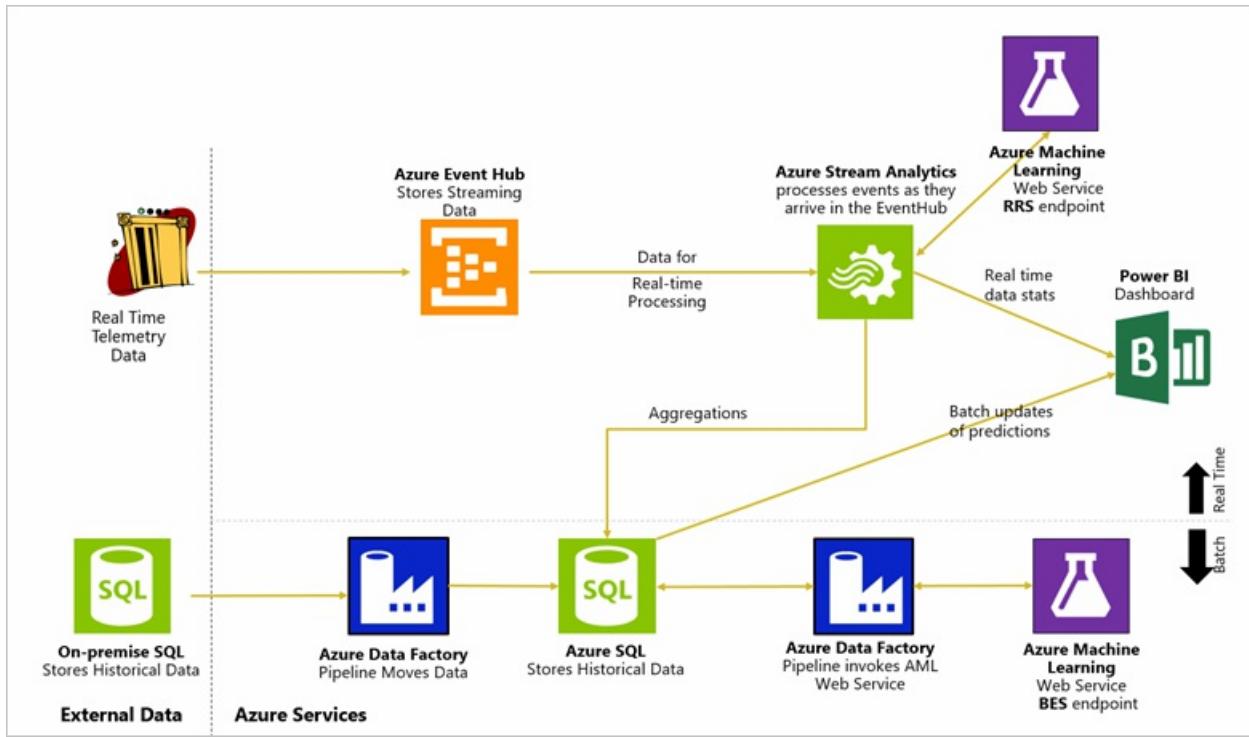


Figure 8. Example solution architecture for predictive maintenance

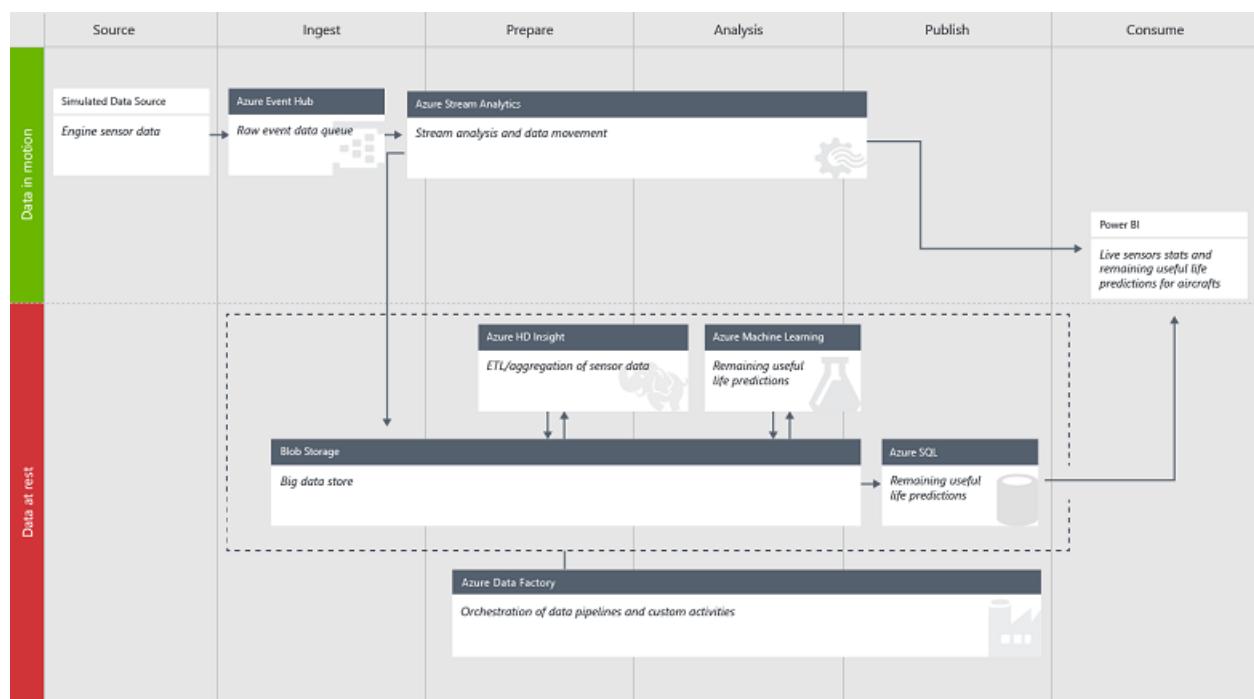
For more information about each of the components of the architecture please refer to [Azure documentation](#).

Architecture of the Cortana Intelligence Solution Template for predictive maintenance in aerospace and other businesses

9/25/2017 • 1 min to read • [Edit Online](#)

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

11/27/2017 • 16 min to read • [Edit Online](#)

IMPORTANT

This article has been deprecated. The discussion about Predictive Maintenance in Aerospace is still relevant, but for current information, refer to [Solution Overview for Business Audiences](#).

Solution templates are designed to accelerate the process of building an E2E demo on top of Cortana Intelligence Suite. A deployed template provisions your subscription with necessary Cortana Intelligence components and then builds the relationships between them. It also seeds the data pipeline with sample data from a data generator application, which you download and install on your local machine after you deploy the solution template. The data from the generator hydrates the data pipeline and start generating machine learning predictions, which can then be visualized on the Power BI dashboard.

The deployment process guides you through several steps to set up your solution credentials. Make sure you record the credentials such as solution name, username, and password that you provide during the deployment.

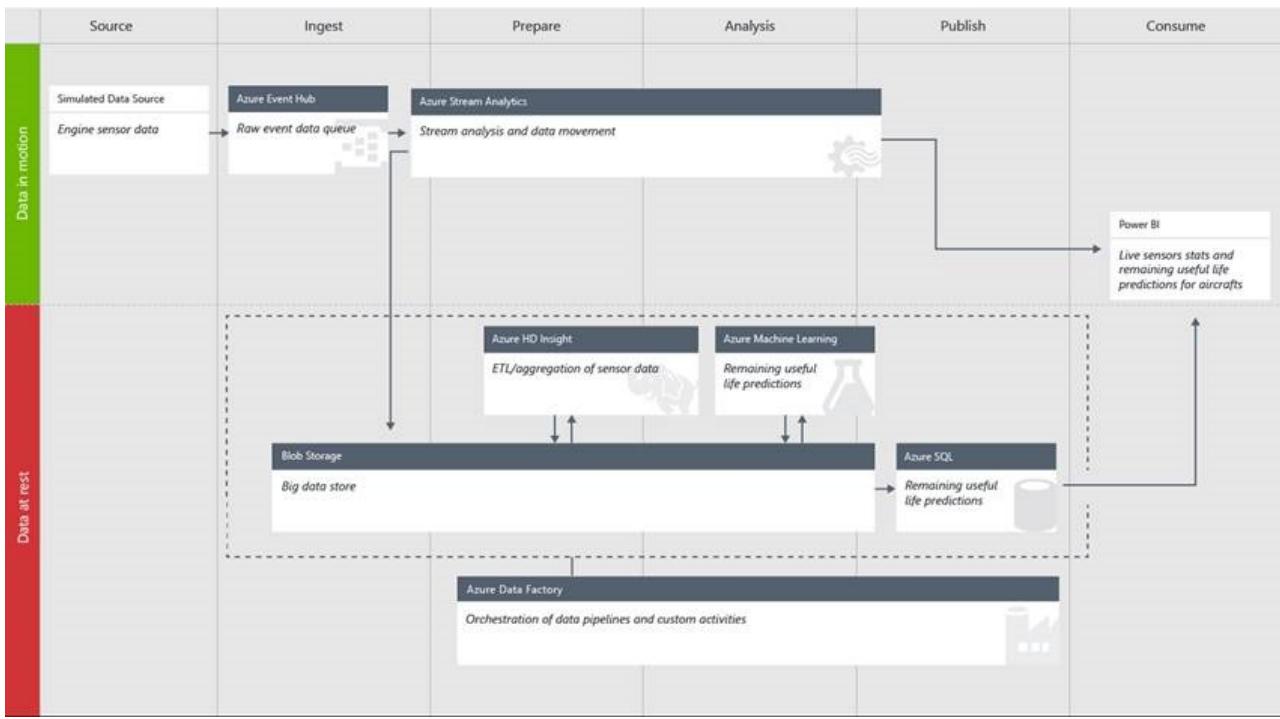
The goals of this article are to:

- Describe the reference architecture and components provisioned in your subscription.
- Demonstrate how to replace the sample data with your own data.
- Show how to modify the solution template.

TIP

You can download and print a [PDF version of this article](#).

Overview



When you deploy the solution, it activates Azure services within the Cortana Analytics Suite (including Event Hub, Stream Analytics, HDInsight, Data Factory, and Machine Learning). The architecture diagram shows how the Predictive Maintenance for Aerospace Solution Template is constructed. You can investigate these services in the Azure portal by clicking them in the solution template diagram created with the solution deployment (except for HDInsight, which is provisioned on demand when the related pipeline activities are required to run and are deleted afterwards). Download a [full-size version of the diagram](#).

The following sections describe the solution parts.

Data source and ingestion

Synthetic data source

For this template, the data source used is generated from a desktop application that you download and run locally after successful deployment.

To find the instructions to download and install this application, select the first node, Predictive Maintenance Data Generator, on the solution template diagram. The instructions are found in the Properties bar. This application feeds the [Azure Event Hub](#) service with data points, or events, used in the rest of the solution flow. This data source is derived from publicly available data from the [NASA data repository](#) using the [Turbofan Engine Degradation Simulation Data Set](#).

The event generation application populates 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.

Data preparation and analysis

Azure Stream Analytics

Use [Azure Stream Analytics](#) to provide near real-time analytics on the input stream from the [Azure Event Hub](#) service. You then publish results onto a [Power BI](#) dashboard as well as archive all raw incoming events to the [Azure Storage](#) service for later processing by the [Azure Data Factory](#) service.

HDInsight custom aggregation

Run [Hive](#) scripts (orchestrated by Azure Data Factory) using HDInsight to provide aggregations on the raw events

archived using the Azure Stream Analytics service.

Azure Machine Learning

Make predictions on the remaining useful life (RUL) of a particular aircraft engine using the inputs received with [Azure Machine Learning Service](#) (orchestrated by Azure Data Factory).

Data publishing

Azure SQL Database

Use [Azure SQL Database](#) to store the predictions received by the Azure Machine Learning service, which are then consumed in the [Power BI](#) dashboard.

Data consumption

Power BI

Use [Power BI](#) to show a dashboard that contains aggregations and alerts provided by [Azure Stream Analytics](#), as well as RUL predictions stored in [Azure SQL Database](#) that were produced using [Azure Machine Learning](#).

How to bring in your own data

This section describes how to bring your own data to Azure, and what areas require changes for the data you bring into this architecture.

It's unlikely that your dataset matches the dataset used by the [Turbofan Engine Degradation Simulation Data Set](#) used for this solution template. Understanding your data and the requirements are crucial in how you modify this template to work with your own data.

The following sections discuss the parts of the template that require modifications when a new dataset is introduced.

Azure Event Hub

Azure Event Hub is generic; 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 APIs.

Azure Stream Analytics

Use the Azure Stream Analytics service 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 query consuming events from the Azure Event Hub service, with 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:

- Connect to the Azure portal
- Locating the Stream Analytics jobs  that were generated when the solution was deployed (*for example, **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 provided as part of this solution template. Because there's implicit knowledge about the incoming data format, these queries must be altered based on your data format.

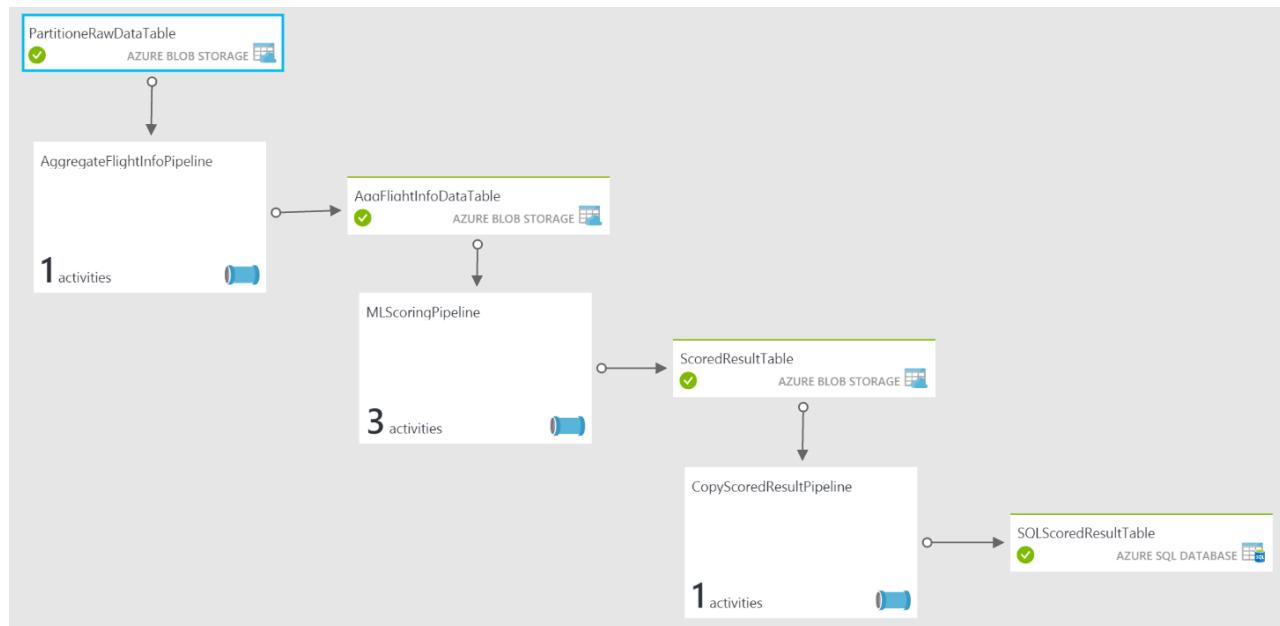
The query in the second Stream Analytics job **maintenancesa02asblob** simply outputs all [Event Hub](#) events to [Azure Storage](#) and hence requires no alteration regardless of your data format as the full event information is streamed to storage.

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. Access your data factory by opening the Data Factory node at the bottom of the solution template diagram created with the deployment of the solution. Errors under your datasets are due to data factory being deployed before the data generator was started. Those errors can be ignored and do not prevent your data factory from functioning

The screenshot shows the Azure Data Factory Essentials interface. The left sidebar has sections for 'Author and display', 'Diagram', and 'Sample pipelines'. The main area has tabs for 'Datasets', 'Pipelines', and 'Linked services'. Under 'Datasets', there are 6 items, with 1 having errors. Under 'Pipelines', there are 3 items. Under 'Linked services', there are 5 items. A right-hand panel titled 'FAILED DATASETS GENERATED WITH EXEC' shows 'PartitioneRawDataTable'.

This section discusses the necessary [pipelines](#) and [activities](#) contained in the [Azure Data Factory](#). Here is a diagram view of the solution.



Two of the pipelines of this factory contain [Hive](#) scripts used to partition and aggregate the data. When noted, the scripts are located in the [Azure Storage](#) account created during setup. Their location is:
maintenancesascript\\script\\hive\\ (or [https://\[Your solution name\].blob.core.windows.net/maintenancesascript](https://[Your solution name].blob.core.windows.net/maintenancesascript)).

Similar to [Azure Stream Analytics](#) queries, the [Hive](#) scripts have implicit knowledge about the incoming data format and must be altered based on your data format.

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 whose end result is the scored predictions from the [Azure Machine Learning](#) experiment associated with this solution template.

Activities included 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 accessed by the [AzureMLBatchScoring](#) activity.
- [AzureMLBatchScoring](#) activity calls the [Azure Machine Learning](#) experiment, with results 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](#) 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 requires modification or replacement specific to the data 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 dehydrate, and the different components of your solution start kicking into action following the commands issued by the data factory. There are two ways to monitor the pipeline.

1. One of the Stream Analytics jobs 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 takes you to the [Azure portal](#). Once there, click on Blobs. In the next panel, you see a list of Containers. Click on **maintenancesadata**. In the next panel is the **rawdata** folder. Inside the rawdata folder are folders with names such as hour=17, and hour=18. The presence of these folders indicates raw data is being generated on your computer and stored in blob storage. You should see csv files with finite sizes in MB in those folders.
2. The last step of the pipeline is to write data (for example 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 the [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 (for example select count(*) from

PMResult). As your database grows, the number of rows in the table should increase.

Power BI Dashboard

Set up a Power BI dashboard to visualize your Azure Stream Analytics data (hot path) and batch prediction results from Azure machine learning (cold path).

Set up the cold path dashboard

In the cold path data pipeline, the 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) On deploying your solution, a prediction will appear 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 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 (for example, prediction results) for visualization.

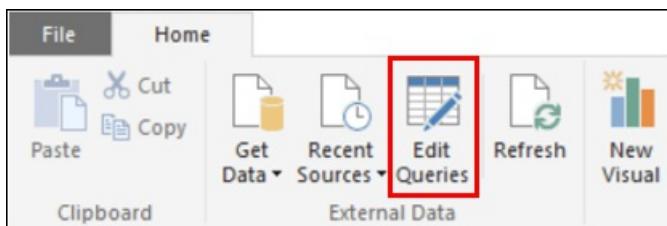
1. Get the database credentials.

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 this icon (for example, '**pmaintainededb**'), 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 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**'.



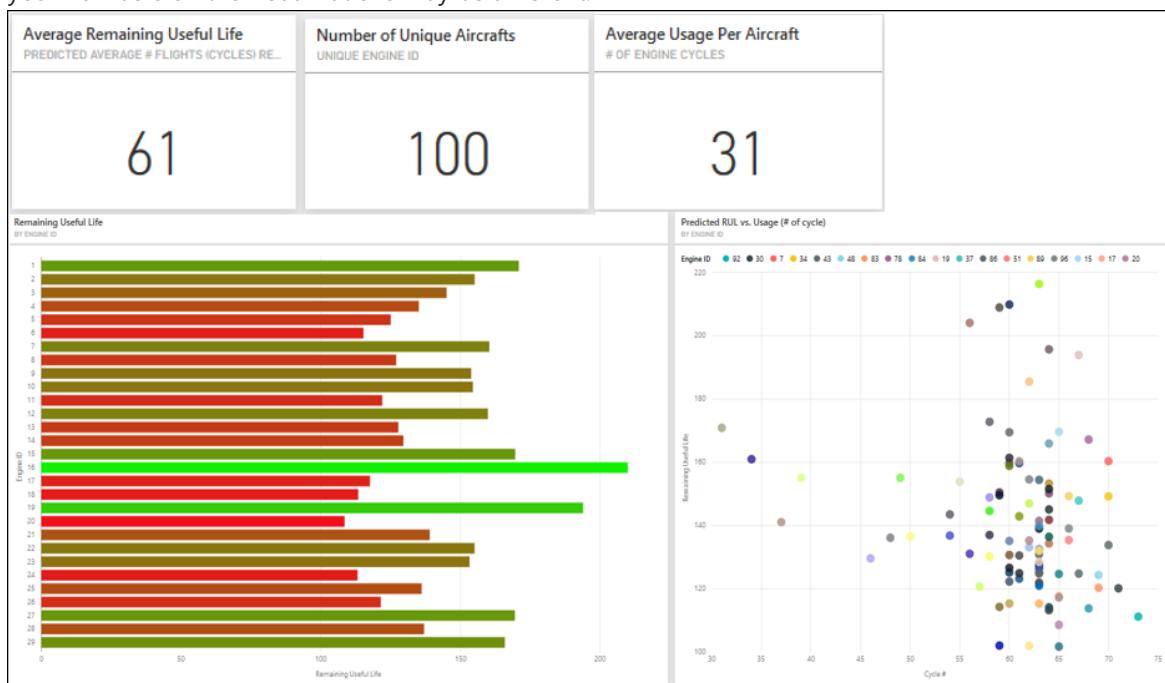
- You'll see two tables, **RemainingUsefulLife** and **PMResult**. Select the first table and click  next to '**Source**' under '**APPLIED STEPS**' on the right '**Query Settings**' panel. Ignore any warning messages that appear.
- In the pop out window, replace '**Server**' and '**Database**' with your 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.

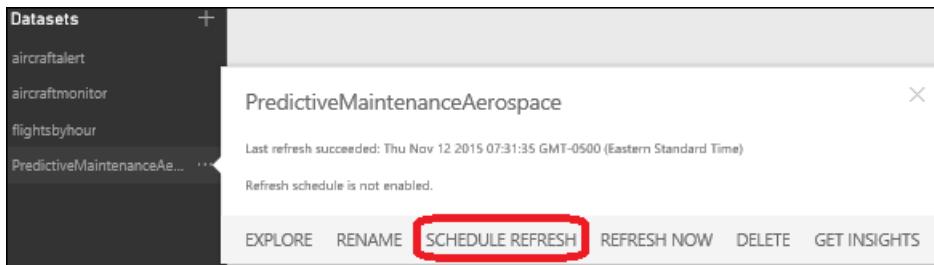
- In the next pop out window, you'll see two options on the left pane (**Windows** and **Database**). Click '**Database**', fill in your '**Username**' and '**Password**' (this is the username and password you entered when you first deployed the solution and created an Azure SQL database). In **Select 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 displays - 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 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.

3. (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

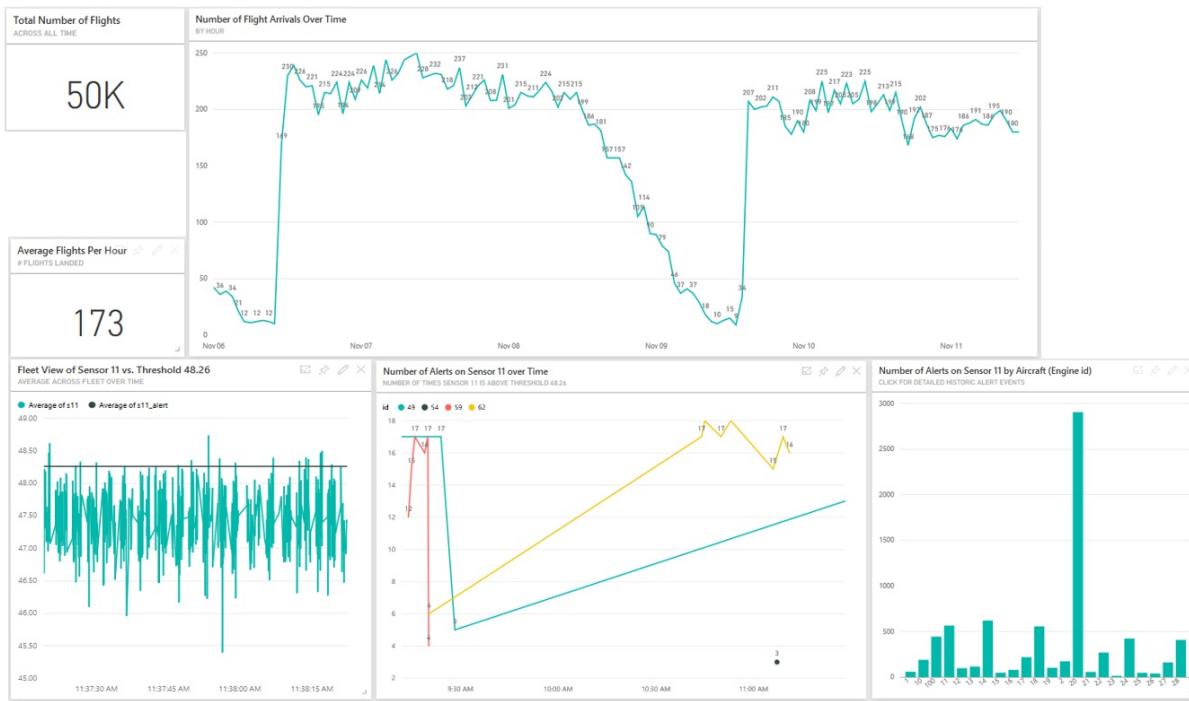
The following steps guide you how to visualize data output from Stream Analytics jobs that were generated at the time of solution deployment. A [Power BI online](#) account is required to perform the following steps. If you don't have an account, you can [create one](#).

1. Add Power BI output in Azure Stream Analytics (ASA).

- You must follow the instructions in [Azure Stream Analytics & Power BI: An 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 need to add an output to ASA. When you add the first output (**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 ("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" is 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."

Delete your solution

Ensure that you stop the data generator when not actively using the solution as running the data generator will incur higher costs. Delete the solution if you are not using it. Deleting your solution deletes all the components provisioned in your subscription when you deployed the solution. To delete the solution, click your solution name in the left panel of the solution template, and then click **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

3/15/2018 • 1 min to read • [Edit Online](#)

This menu links to the chapters in this playbook:

Overview

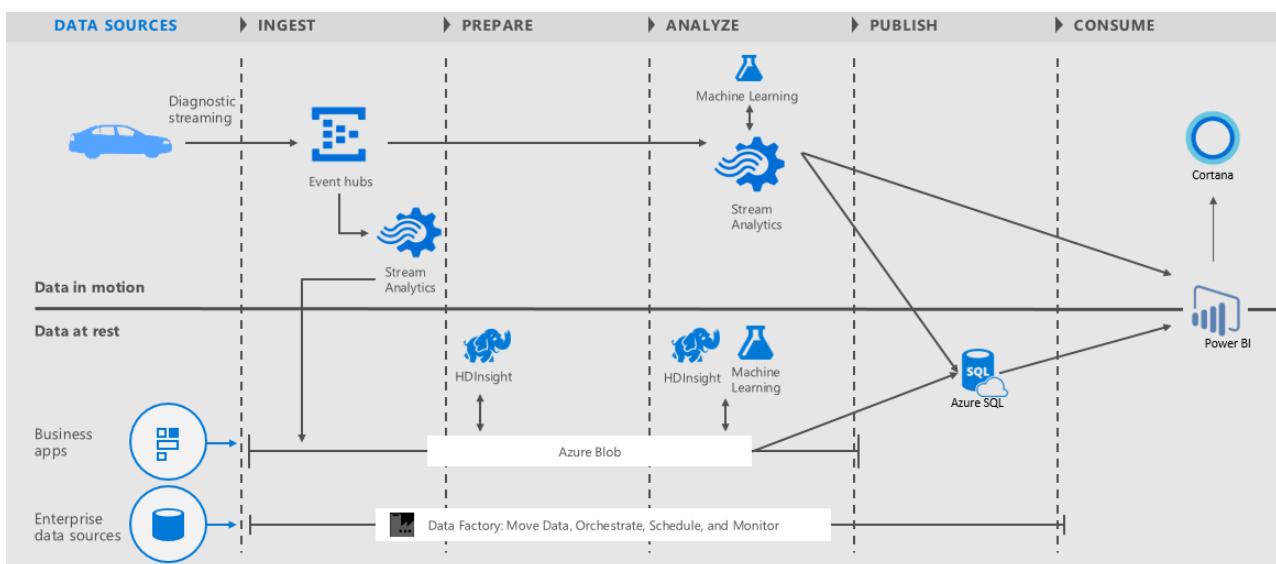
Super computers have moved out of the lab and are now parked in garages. These are now being placed in cutting-edge automobiles that contain myriad sensors. These sensors give them the ability to track and monitor millions of events every second. By 2020, most of these vehicles will be connected to the Internet. Tapping into this wealth of data provides greater safety, reliability, and so a better driving experience. Microsoft makes this dream a reality with Cortana Intelligence.

Cortana Intelligence is a fully managed big data and advanced analytics suite that you can use to transform your data into intelligent action. The Cortana Intelligence Vehicle Telemetry Analytics Solution Template demonstrates how car dealerships, automobile manufacturers, and insurance companies are able to obtain real-time and predictive insights on vehicle health and driving habits.

The solution is implemented as a [lambda architecture pattern](#), which shows the full potential of the Cortana Intelligence platform for real-time and batch processing.

Architecture

The Vehicle Telemetry Analytics Solution architecture is illustrated in this diagram:



This solution includes the following Cortana Intelligence components and showcases their integration:

- **Azure Event Hubs** ingests millions of vehicle telemetry events into Azure.
- **Azure Stream Analytics** provides real-time insights on vehicle health and persists that data into long-term storage for richer batch analytics.
- **Azure Machine Learning** detects anomalies in real time and uses batch processing to provide predictive insights.
- **Azure HDInsight** transforms data at scale.
- **Azure 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:** This reference data set maps VIN numbers to models.

Vehicle Telemetry Analytics Solution playbook: Deep dive into the solution

3/15/2018 • 17 min to read • [Edit Online](#)

This menu links to the sections of this playbook:

This document drills down into each of the stages depicted in the solution architecture. Instructions and pointers for customization are included.

Data sources

The solution uses two different data sources:

- Simulated vehicle signals and diagnostic data set
- Vehicle catalog

A vehicle telematics simulator is included as part of this solution, as shown in the following screenshot. It emits diagnostic information and signals that correspond to the state of the vehicle and to the driving pattern at a given point in time. To download the Vehicle Telematics Simulator Visual Studio Solution for customizations based on your requirements, go to the [Vehicle telematics simulator](#) webpage. The vehicle catalog contains a reference data set that maps vehicle identification numbers (VINs) to models.

```
*****
Virtual Car Telematics Application
*****
Press Ctrl-C to stop the sender process
11/28/2015 11:57:05 PM > Sending message: {"vin":"UUMDK8HD35HJ6IQ4","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":"75I14TF2XUU5QB8Q","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
```

This JSON-formatted data set contains the following schema.

COLUMN	DESCRIPTION	VALUES
VIN	Randomly generated VIN	Obtained from a master list of 10,000 randomly generated VINs

COLUMN	DESCRIPTION	VALUES
Outside temperature	The outside temperature where the vehicle is driving	Randomly generated number from 0 to 100
Engine temperature	The engine temperature of the vehicle	Randomly generated number from 0 to 500
Speed	The engine speed at which the vehicle is driving	Randomly generated number from 0 to 100
Fuel	The fuel level of the vehicle	Randomly generated number from 0 to 100 (indicates fuel level percentage)
EngineOil	The engine oil level of the vehicle	Randomly generated number from 0 to 100 (indicates engine oil level percentage)
Tire pressure	The tire pressure of the vehicle	Randomly generated number from 0 to 50 (indicates tire pressure level percentage)
Odometer	The odometer reading of the vehicle	Randomly generated number from 0 to 200,000
Accelerator_pedal_position	The accelerator pedal position of the vehicle	Randomly generated number from 0 to 100 (indicates accelerator level percentage)
Parking_brake_status	Indicates whether the vehicle is parked or not	True or False
Headlamp_status	Indicates whether 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 on or not	True or False
ABS	Indicates whether ABS is engaged or not	True or False
Timestamp	The time stamp when the data point is created	Date
City	The location of the vehicle	Four cities in this solution: Bellevue, Redmond, Sammamish, Seattle

The vehicle model reference data set maps VINs to models.

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
VCYWW0WUZNBTM594J	Sports car
HNVCE6YFZSA5M82NY	Medium SUV
4R30FOR7NUOBL05GJ	Station wagon
WYNIIY42VKV6OQS1J	Large SUV
8Y5QKG27QET1RBK7I	Large SUV
DF6OX2WSRA6511BVG	Coupe
Z2EOZWZBXAEW3E60T	Sedan
M4TV6IEALD5QDS3IR	Hybrid
VHRA1Y2TGTAA84F00H	Family saloon
ROJAUHT1L1R3BIKIO	Sedan
9230C202Z60XX84AU	Hybrid
T8DNDN5UDCWL7M72H	Family saloon

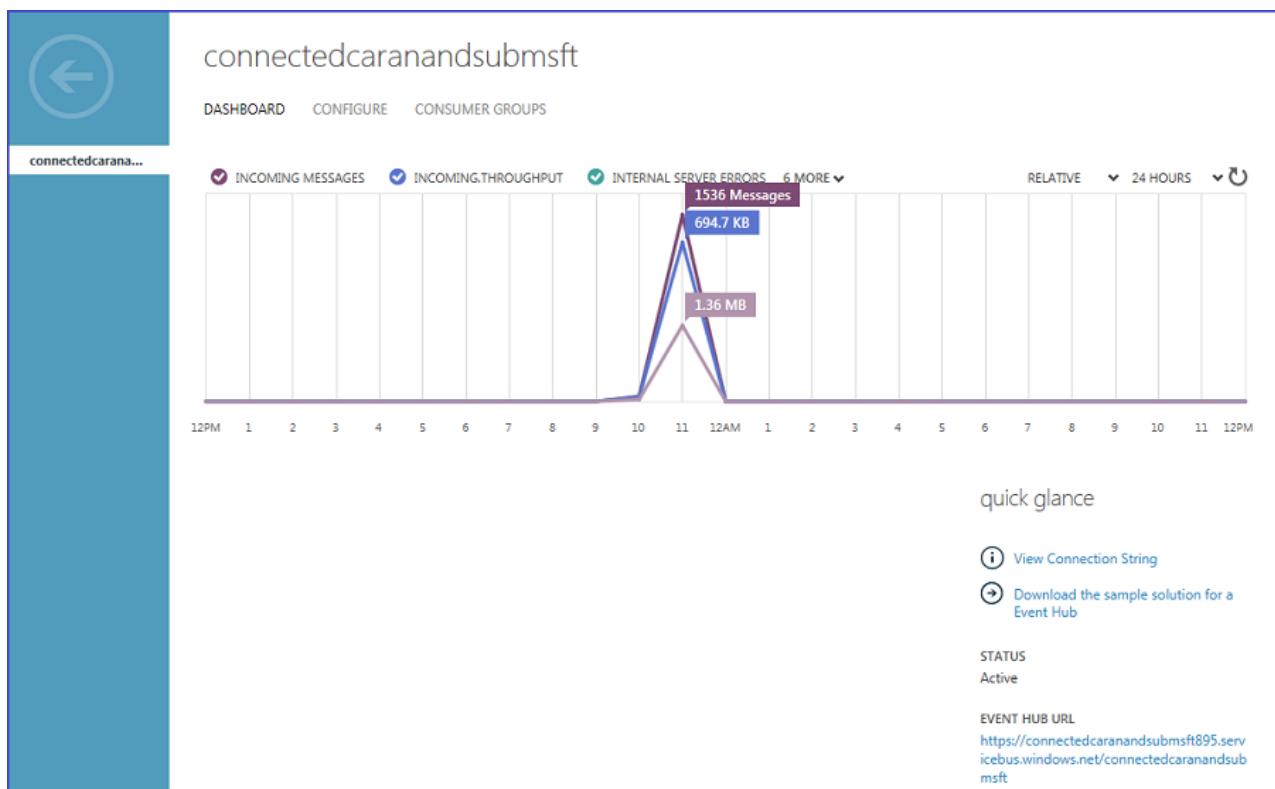
VIN	MODEL
4WPYRUZII5YV7YA42	Sedan
D1ZVY26UV2BFGHZNO	Hybrid
XUF99EW9OIQOMV7Q7	Family saloon
80MCL3LGI7XNCC21U	Convertible
.....	

Ingestion

Combinations of Azure Event Hubs, Azure Stream Analytics, and Azure Data Factory are used 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 by 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.

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.

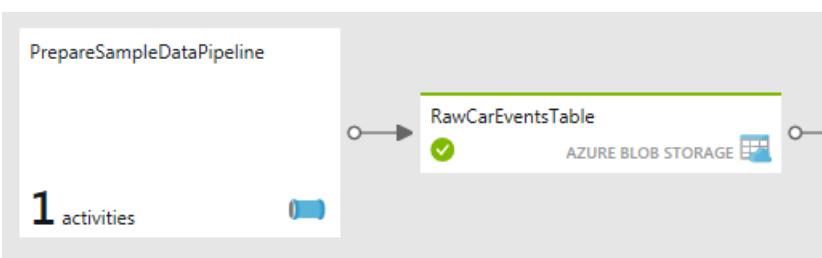
The following Stream Analytics query is used to persist the data into Blob storage:

```
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
```

Batch analysis

An additional volume of simulated vehicle signals and diagnostic data set is also generated for richer batch analytics. This additional volume is required to ensure a good representative data volume for batch processing. For this purpose, PrepareSampleDataPipeline is used in the Data Factory workflow to generate one year's worth of simulated vehicle signals and diagnostic data set. To download the Data Factory custom .NET activity Visual Studio solution for customizations based on your requirements, go to the [Data Factory custom activity](#) webpage.

This workflow shows sample data prepared for batch processing.



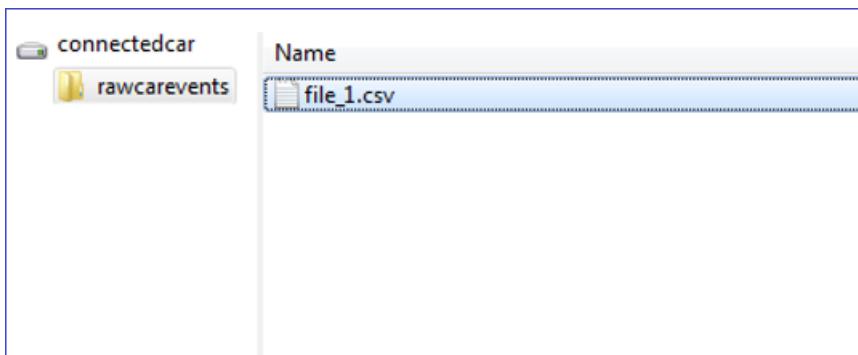
The pipeline consists of a custom Data Factory .NET activity.

```

{
  "name": "PrepareSampleDataPipeline",
  "properties": {
    "description": "Prepare Sample Data for Connected Cars Use Case",
    "activities": [
      {
        "type": "DotNetActivity",
        "transformation": {
          "assemblyName": "ConnectedCarsDataGenerator.dll",
          "entryPoint": "ConnectedCarsDataGenerator.DataGenerator",
          "packageLinkedService": "StorageLinkedService",
          "packageFile": "connectedcar/scripts/connectedcar.zip",
          "extendedProperties": {
            "sliceStart": "$$Text.Format('{0:yyyyMMddHHmm}', Time.AddMinutes(SliceStart, 0))"
          }
        },
        "outputs": [
          {
            "name": "RawCarEventsTable"
          }
        ],
        "policy": {
          "timeout": "02:00:00",
          "concurrency": 1,
          "executionPriorityOrder": "NewestFirst",
          "retry": 1
        },
        "name": "PrepareSampleDataActivity",
        "description": "Prepare Sample Data for ConnectedCars Use Case",
        "linkedServiceName": "HDInsightLinkedService"
      }
    ],
    "start": "2014-04-01T00:00:00Z",
    "end": "2015-04-01T00:00:00Z",
    "isPaused": false,
    "hubName": "connectedcarandsubmsft_hub"
  }
}

```

After the pipeline executes successfully and the RawCarEventsTable data set is marked "Ready," one year's 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:



Partition the data set

In the data preparation step, the raw semi-structured vehicle signals and diagnostic data set are partitioned into a YEAR/MONTH format. This partitioning promotes more efficient querying and scalable long-term storage by enabling fault-over. For example, as the first blob account fills up, it faults over to the next account.

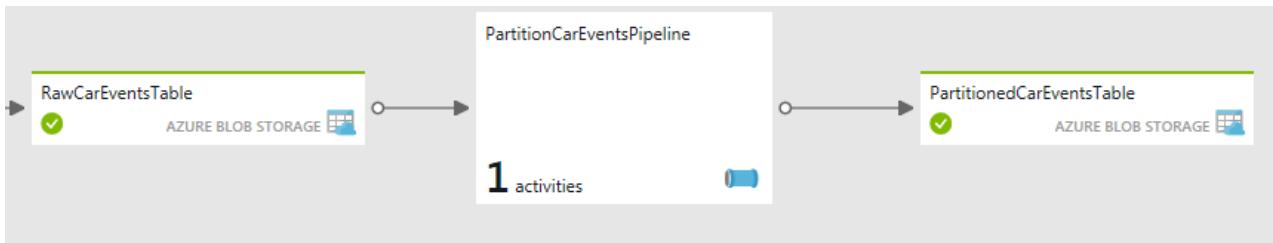
NOTE

This step in the solution applies only to batch processing.

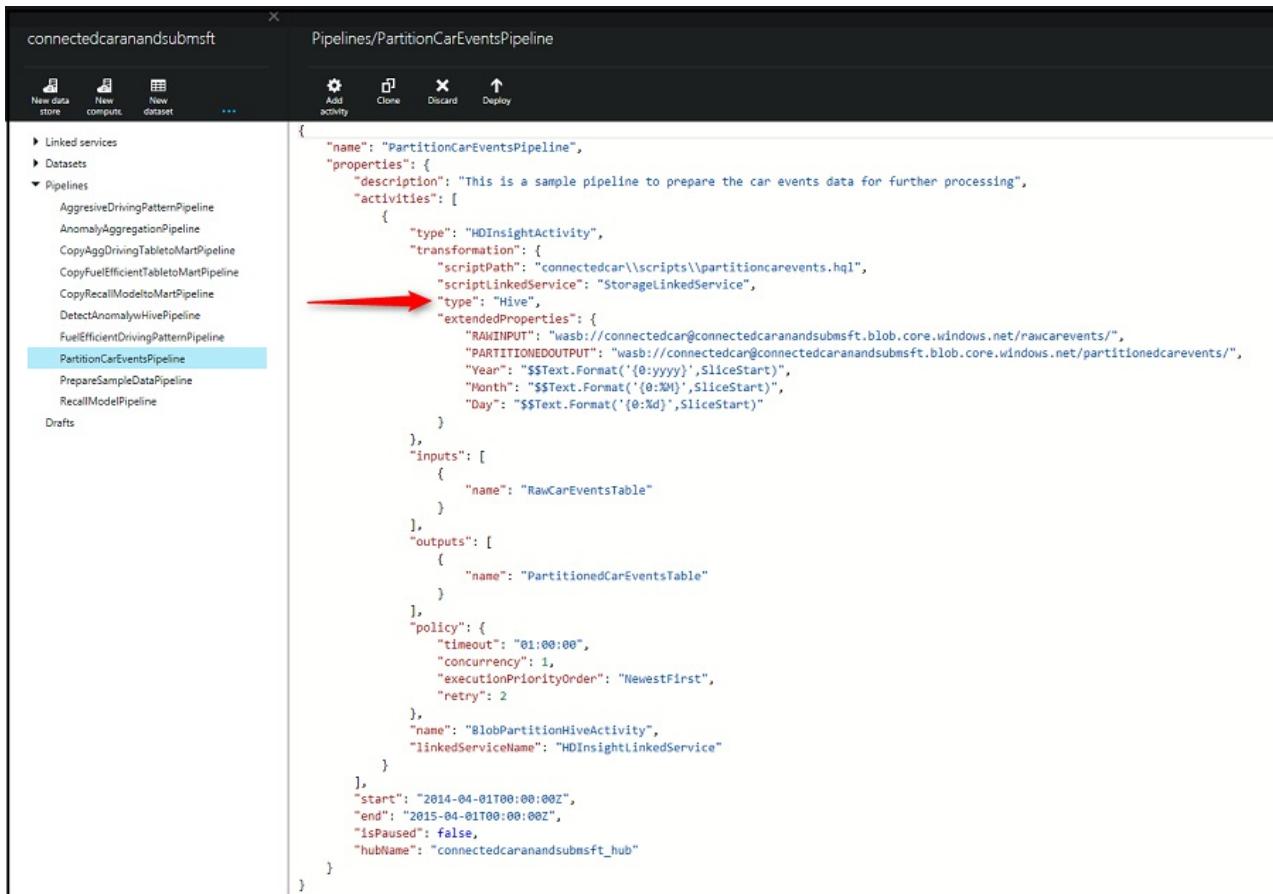
Input and output data management:

- **Output data** (labeled PartitionedCarEventsTable) is kept for a long period of time as the foundational/"rawest" form of data in the customer's data lake.
- **Input data** to this pipeline is typically discarded because the output data has full fidelity to the input. It's stored (partitioned) better for subsequent use.

The partition car events workflow.



The raw data is partitioned by using a Hive Azure HDInsight activity in PartitionCarEventsPipeline, as shown in the following screenshot. The sample data generated for a year in the data preparation step is partitioned by YEAR/MONTH. The partitions are used to generate vehicle signals and diagnostic data for each month (total of 12 partitions) of a year.



PartitionConnectedCarEvents Hive script

The Hive script partitioncarevents.hql is used for partitioning. It's located in the \demo\src\connectedcar\scripts folder of the downloaded zip file.

```
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 '10' 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
'10' 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 '10';

```

```

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)

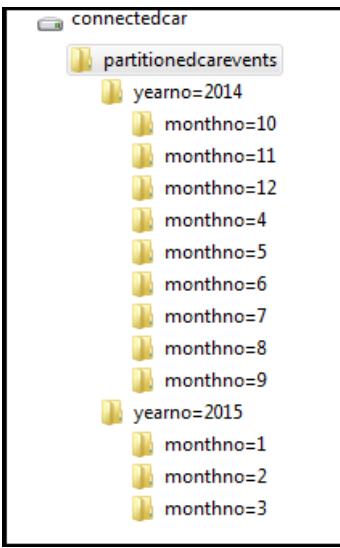
FROM RawCarEvents WHERE Year(gendate) = ${hiveconf:Year} AND Month(gendate) = ${hiveconf:Month};

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};

```

After the pipeline executes successfully, you see the following partitions generated in your storage account under the connectedcar container:



The data is now optimized, more manageable, and ready for further processing to gain rich batch insights.

Data analysis

In this section, you see how to combine Stream Analytics, Azure Machine Learning, Data Factory, and HDInsight for rich advanced analytics on vehicle health and driving habits.

Machine learning

The goal here is to predict the vehicles that require maintenance or recall based on certain heath statistics, based on the following assumptions:

- If one of the following three conditions is true, the vehicles require servicing maintenance:
 - The tire pressure is low.
 - The engine oil level is low.
 - The engine temperature is high.
- If one of the following conditions is true, the vehicles might have a safety issue and require recall:
 - The engine temperature is high, but the outside temperature is low.
 - The engine temperature is low, but the outside temperature is high.

Based on the previous requirements, two separate models detect anomalies. One model is for vehicle maintenance detection, and one model is for vehicle recall detection. In both 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, only these three variables need to be consider in building the model. In the experiment in machine learning, the **Select Columns in Dataset** module is used to extract these three variables. Next, the PCA-based anomaly detection module is used to build the anomaly detection model.

PCA is an established technique in machine learning that can be applied to feature selection, classification, and anomaly detection. PCA converts a set of cases that contain 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 to more easily identify features and anomalies.

For each new input to the detection model, the anomaly detector first computes its projection on the eigenvectors. It then computes the normalized reconstruction error. This normalized error is the anomaly score: the higher the error, the more anomalous the instance.

In the maintenance detection problem, each record is considered as a point in a three-dimensional space defined by tire pressure, engine oil, and engine temperature coordinates. To capture these anomalies, PCA is used to project the original data in the three-dimensional space onto a two-dimensional space. Thus, the parameter number of components to use in PCA is set to two. This parameter plays an important role in applying PCA-based anomaly detection. After using PCA to project data, these anomalies are identified more easily.

Recall anomaly detection model

In the recall anomaly detection model, the **Select Columns in Dataset** and PCA-based anomaly detection modules are used in a similar way. Specifically, three variables--engine temperature, outside temperature, and speed--are extracted first by using the **Select Columns in Dataset** module. The speed variable also is included, because the engine temperature typically correlates to the speed. Next, the PCA-based anomaly detection module is used to project the data from the three-dimensional space onto a two-dimensional space. The recall criteria are satisfied. The vehicle requires recall when engine temperature and outside temperature are highly negatively correlated. After PCA is performed, the PCA-based anomaly detection algorithm is used to capture the anomalies.

When training either model, normal data is used as the input data to train the PCA-based anomaly detection model. (Normal data doesn't require maintenance or recall.) In the scoring experiment, the trained anomaly detection model is used to detect whether 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. These parameters include 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 a specific region. The averages are then correlated 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)
```

All the averages are calculated over a three-second tumbling window. A tumbling window is used because non-overlapping and contiguous time intervals are required.

To learn more about the windowing capabilities in Stream Analytics, see [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. The application RealTimeDashboardApp is created and configured as part of the solution deployment. The application:

- Listens to an event hub instance where Stream Analytics publishes 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
```

- Receives events. For every event that this application receives:
 - The data is processed by using a 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 data set by using the push APIs.

This pattern is also applicable to scenarios in which you want to integrate a line-of-business application with the real-time analytics flow. These scenarios include alerts, notifications, and messaging.

To download the RealtimeDashboardApp Visual Studio solution for customizations, see the

RealtimeDashboardApp download webpage.

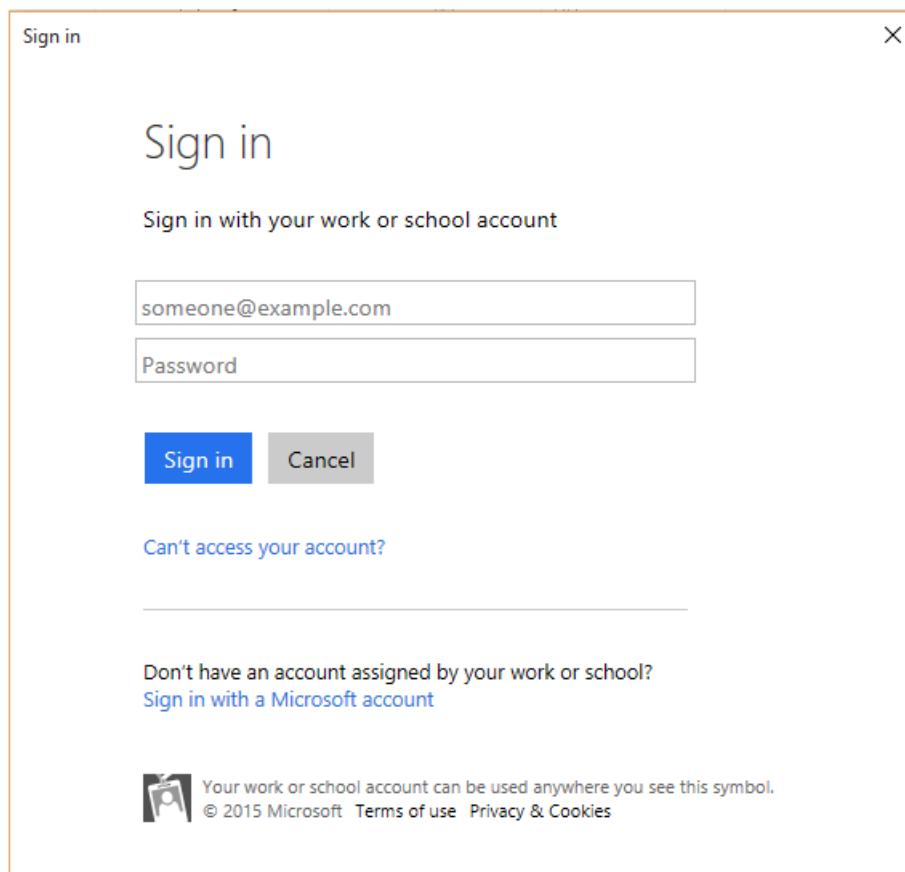
Execute the real-time dashboard application

1. Extract the RealtimeDashboardApp, and save it locally.

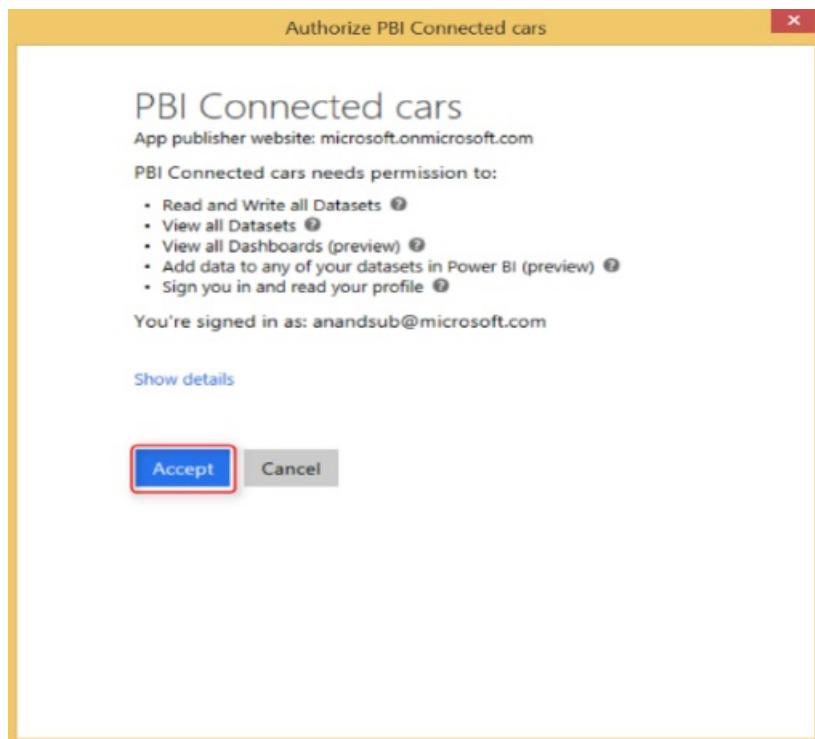
Name	Date modified	Type	Size
ADF	7/21/2015 6:33 PM	File folder	
ASA	7/21/2015 6:33 PM	File folder	
bin	7/21/2015 6:33 PM	File folder	
CarEventGenerator	7/27/2015 11:26 PM	File folder	
RealTimeDashboardApp	7/21/2015 6:34 PM	File folder	
referencedata	7/21/2015 6:34 PM	File folder	
scripts	7/21/2015 6:34 PM	File folder	

2. Execute the application RealtimeDashboardApp.exe.

3. Enter your valid Power BI credentials, and select **Sign in**.



4. Select **Accept**.



NOTE

If you want to flush the Power BI data set, 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. This data reveals rich insights on driving patterns, usage behavior, and vehicle health. This information 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 patterns to govern business decisions and provide best-in-class products and services.

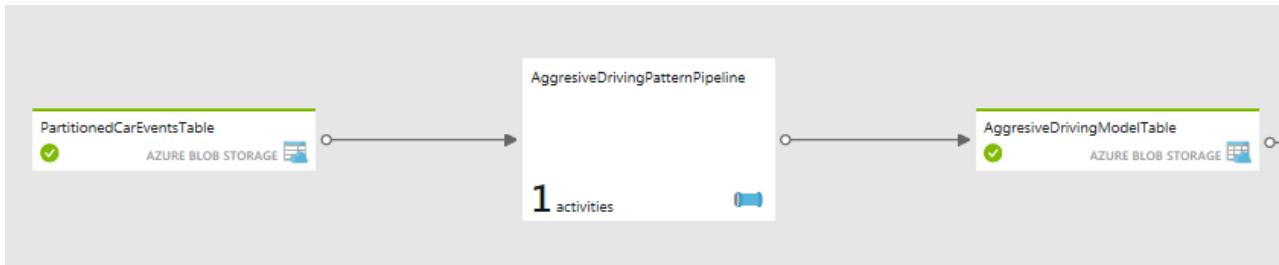
In this solution, the following metrics are targeted:

- **Aggressive driving behavior:** Identifies the trend of the models, locations, driving conditions, and time of year to gain insights on aggressive driving patterns. Contoso Motors can use these insights for marketing campaigns to introduce new personalized features and usage-based insurance.
- **Fuel-efficient driving behavior:** Identifies the trend of the models, locations, driving conditions, and time of year to gain insights on fuel-efficient driving patterns. Contoso Motors can use these insights for marketing campaigns to introduce new features and proactive reporting to drivers for cost-effective and environment-friendly driving habits.
- **Recall models:** Identifies models that require recalls by operationalizing the anomaly detection machine learning experiment.

Let's look into the details of each of these metrics.

Aggressive driving behavior patterns

The partitioned vehicle signals and diagnostic data are processed in `AggresiveDrivingPatternPipeline`, as shown in the following workflow. Hive is used to determine the models, location, vehicle, driving conditions, and other parameters that exhibit aggressive driving patterns.



Aggressive driving pattern Hive query

The Hive script `agresivedriving.hql` is used to analyze aggressive driving condition patterns. It's located in the `\demo\src\connectedcar\scripts` folder of the downloaded zip file.

```

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
)

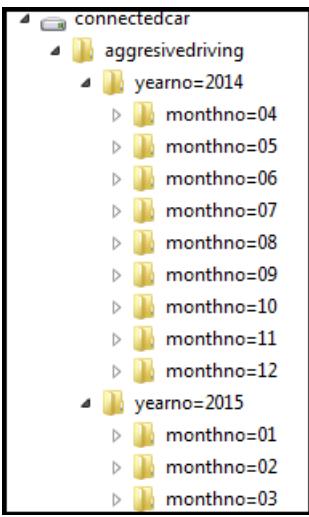
) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '10' STORED AS TEXTFILE LOCATION
`${hiveconf:AGGRESIVEOUTPUT}`;

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', 'eight') AND brake_pedal_status =
'1' AND speed >= '50'

```

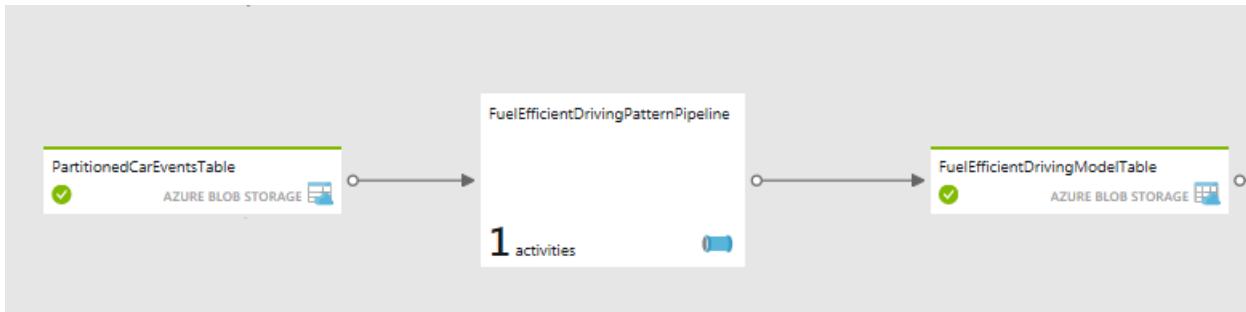
The script uses the combination of a vehicle's transmission gear position, brake pedal status, and speed to detect reckless/aggressive driving behavior based on braking patterns at high speed.

After the pipeline executes successfully, you see the following partitions generated in your storage account under the connectedcar container:



Fuel-efficient driving behavior patterns

The partitioned vehicle signals and diagnostic data are processed in FuelEfficientDrivingPatternPipeline, as shown in the following workflow. Hive is used to determine the models, location, vehicle, driving conditions, and other properties that exhibit fuel-efficient driving patterns.



Fuel-efficient driving pattern Hive query

The Hive script `fuelefficientdriving.hql` is used to analyze fuel-efficient driving condition patterns. It's located in the `\demo\src\connectedcar\scripts` folder of the downloaded zip file.

```

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}`;

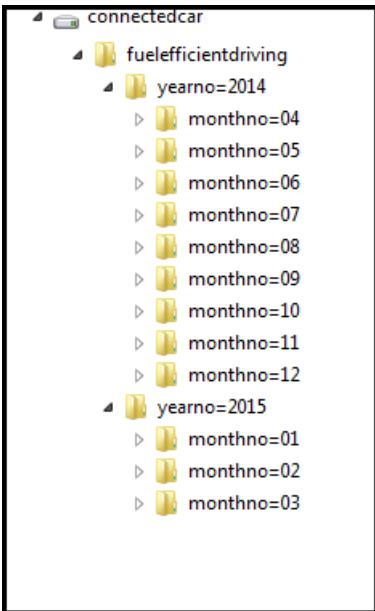
DROP TABLE IF EXISTS FuelEfficientDriving;
CREATE EXTERNAL TABLE FuelEfficientDriving
(
    vin                      string,
    model                     string,
    city                      string,
    speed                     string,
    transmission_gear_position string,
    brake_pedal_status         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', 'eight') AND parking_brake_status
= '0' AND brake_pedal_status = '0' AND speed <= '60' AND accelerator_pedal_position >= '50'

```

The script uses the combination of a 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.

After the pipeline executes successfully, you see the following partitions generated in your storage account under the connectedcar container:



Recall model predictions

The machine learning experiment is provisioned and published as a web service as part of the solution deployment. The batch scoring endpoint is used in this workflow. It's registered as a data factory linked service and operationalized by using the data factory batch scoring activity.

```
{
  "name": "AzureMLAnomalydetectionendpoint",
  "properties": {
    "hubName": "connectedcaranandsubmsf_hub",
    "type": "AzureML",
    "typeProperties": {
      "mlEndpoint": "https://ussouthcentral.services.azureml.net/workspaces/e501a87b96754eae8c2512b6d591097a/services/99d9a2e7c480467283ea95d1e1075274/jobs",
      "apiKey": "*****"
    }
  }
}
```

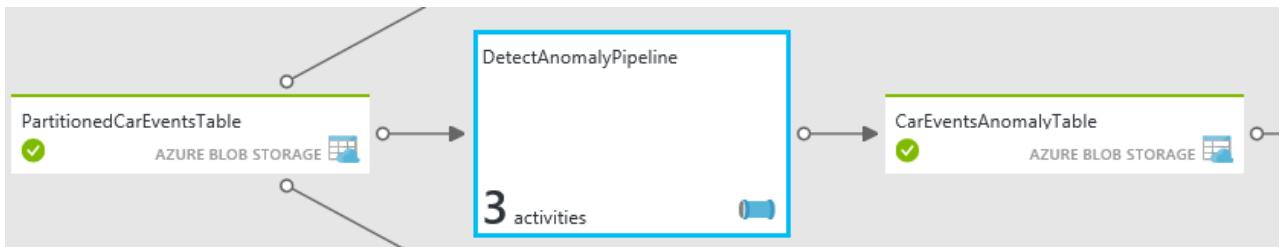
The registered linked service is used in DetectAnomalyPipeline to score the data by using the anomaly detection model.

```

{
    "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"
}
}

```

A few steps are performed in this pipeline for data preparation so that it can be operationalized with the batch scoring web service.



Anomaly detection Hive query

After the scoring is finished, an HDInsight activity processes and aggregates the data that the model categorized as anomalies. The model uses 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}';

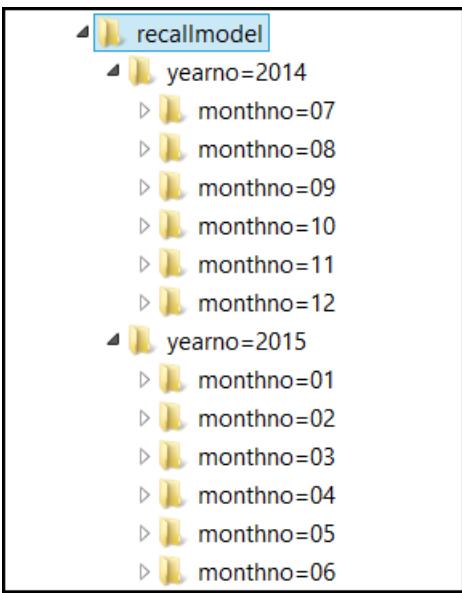
DROP TABLE IF EXISTS RecallModel;
CREATE EXTERNAL TABLE RecallModel
(
    vin                      string,
    model                     string,
    city                      string,
    outsidetemperature        string,
    enginetemperature         string,
    speed                     string,
    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'

```

After the pipeline executes successfully, you see the following partitions generated in your storage account under the connectedcar container:



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
SQLSink	SQL Database	✓ OK

The following Stream Analytics query is used to publish to the 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

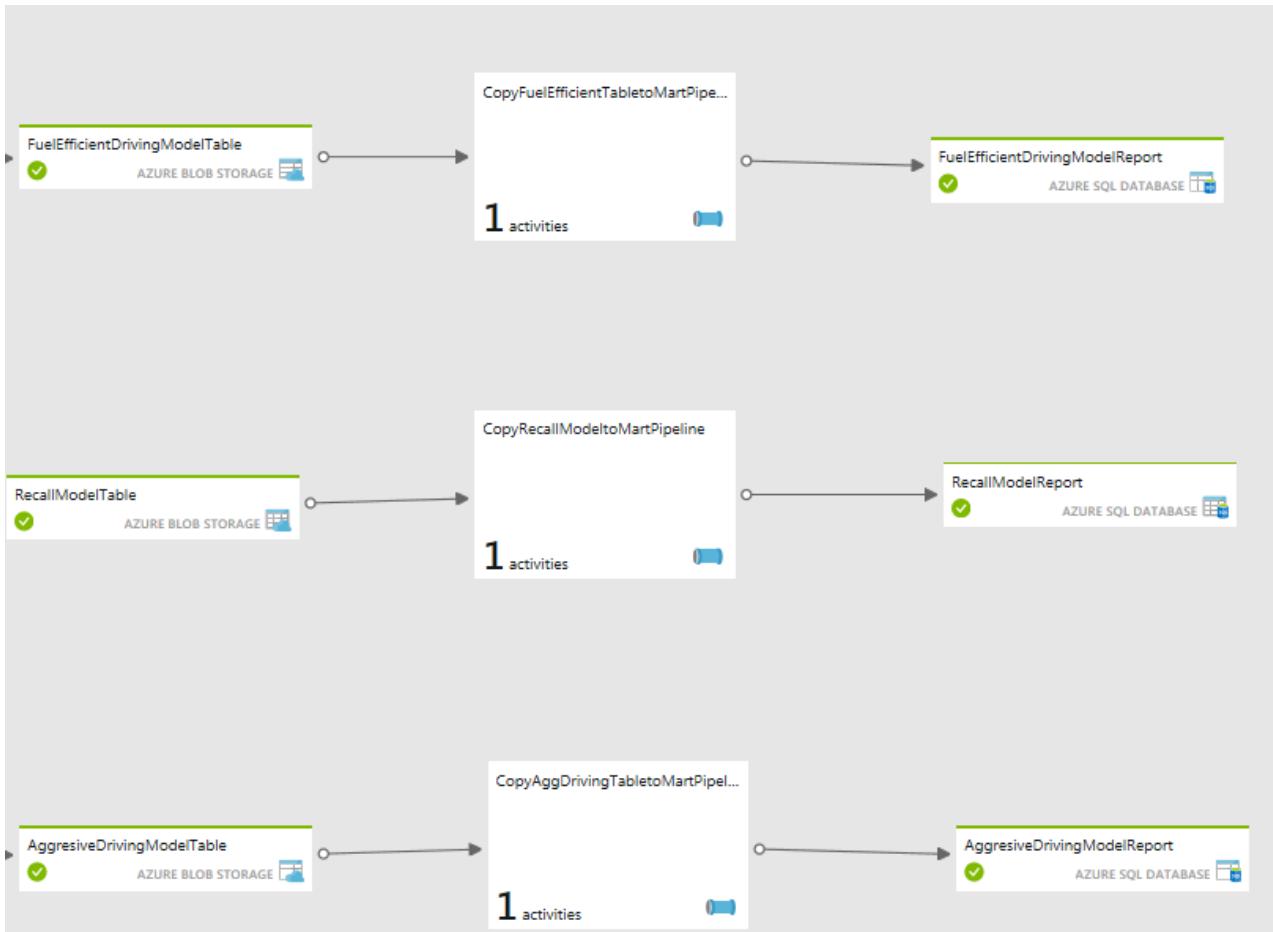
```

This stream of events is consumed by the RealTimeDashboardApp that's included in the solution. This application uses the machine learning request-response web service for real-time scoring. It publishes the resultant data to a Power BI data set for consumption.

Batch analysis

The results of the batch and real-time processing are published to Azure SQL Database tables for consumption. The SQL server, the database, and the tables are created automatically as part of the setup script.

The batch processing results are copied to the data mart workflow.



The Stream Analytics job is published to the data mart.

The screenshot shows the Stream Analytics job configuration page for the job "connectedcaranandsubmsft".

- Outputs:** A message states: "Output can't be edited while a job is running. You can stop the job to edit the output."
- Diagnosis:** Two sinks are listed:

NAME	TYPE	DIAGNOSIS
BlobSink	Blob storage	✓ OK
SQLSink	SQL Database	✓ OK

The data mart setting is in the Stream Analytics job.

BlobSink
SQISink

sqlsink

Output can't be edited while a job is running. You can stop the job to edit the output.

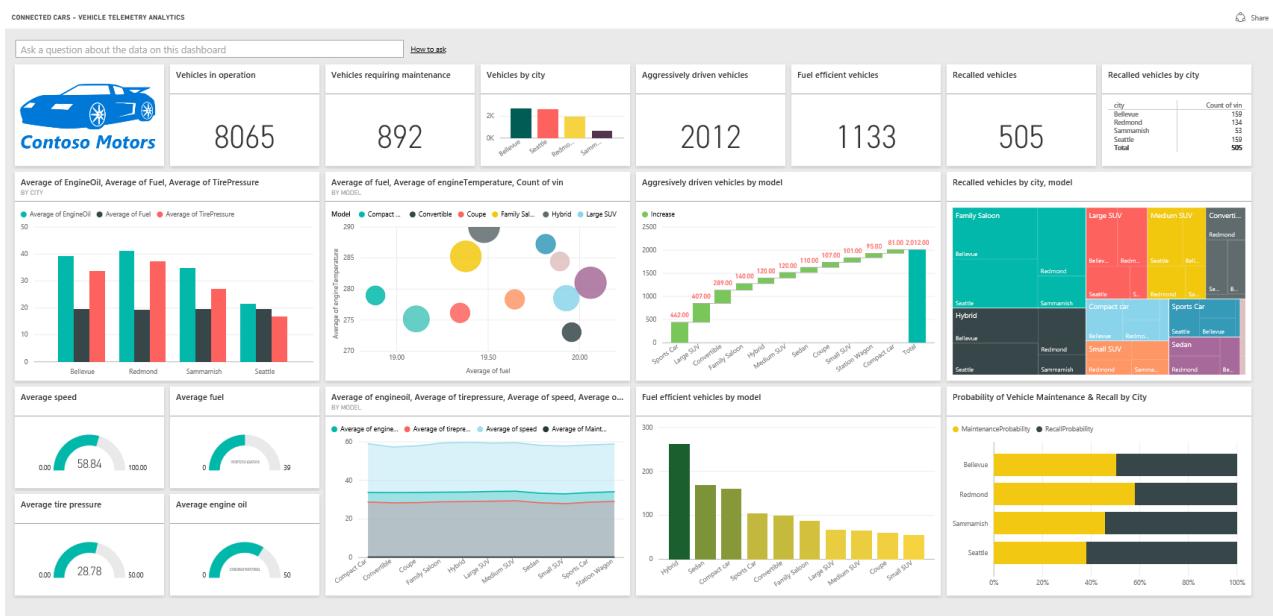
general

SUBSCRIPTION	<input type="checkbox"/> Use SQL Database from Another Subscription <input checked="" type="checkbox"/>
SERVER NAME	zrzel4v9yp
DATABASE	connectedcar
USERNAME	mylogin
PASSWORD	*****
TABLE	NearRealTimeAveragesReport

Consume

Power BI gives this solution a rich dashboard for real-time data and predictive analytics visualizations.

The final dashboard looks like this example:



Summary

This document contains a detailed drill-down of the Vehicle Telemetry Analytics Solution. The lambda architecture pattern is used 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.

References

- [Vehicle Telematics Simulator Visual Studio Solution](#)
- [Azure Event Hubs](#)
- [Azure Data Factory](#)
- [Azure Event Hubs SDK for stream ingestion](#)
- [Azure Data Factory data movement capabilities](#)
- [Azure Data Factory .NET activity](#)
- [Azure Data Factory .NET activity Visual Studio solution used to prepare sample data](#)

Vehicle Telemetry Analytics Solution Template Power BI dashboard setup instructions

3/15/2018 • 11 min to read • [Edit Online](#)

This menu links to the chapters in this playbook:

The Vehicle Telemetry Analytics Solution showcases how car dealerships, automobile manufacturers, and insurance companies are able to use the capabilities of Cortana Intelligence. They can obtain real-time and predictive insights on vehicle health and driving habits to improve customer experience, research and development, and marketing campaigns. These step-by-step instructions show how to configure the Power BI reports and dashboard after you deploy the solution in your subscription.

Prerequisites

- Deploy the [Vehicle Telemetry Analytics](#) Solution.
- [Install Power BI Desktop](#).
- Obtain an [Azure subscription](#). If you don't have an Azure subscription, get started with the Azure free subscription.
- Open a 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:

- **Azure Event Hubs** ingests millions of vehicle telemetry events into Azure.
- **Azure Stream Analytics** provides real-time insights on vehicle health and persists that data into long-term storage for richer batch analytics.
- **Azure Machine Learning** detects anomalies in real time, and uses batch processing to provide predictive insights.
- **Azure HDInsight** transforms data at scale.
- **Azure Data Factory** handles orchestration, scheduling, resource management, and monitoring of the batch processing pipeline.

Power BI gives this solution a rich dashboard for data and predictive analytics visualizations.

The solution uses two different data sources:

- Simulated vehicle signals and diagnostic data sets
- Vehicle catalog

A vehicle telematics simulator is included as part of this solution. It emits diagnostic information and signals that correspond to the state of the vehicle and driving patterns at a given point in time.

The vehicle catalog is a reference data set that maps VINs to models.

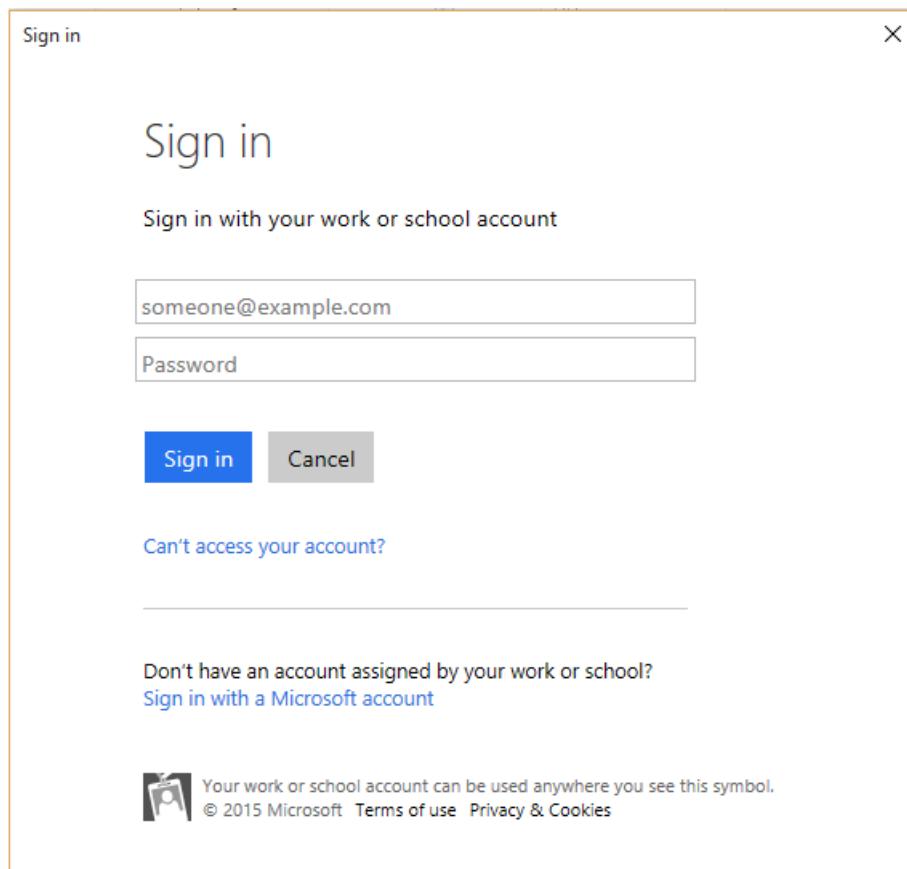
Power BI dashboard preparation

[Set up the Power BI real-time dashboard](#)

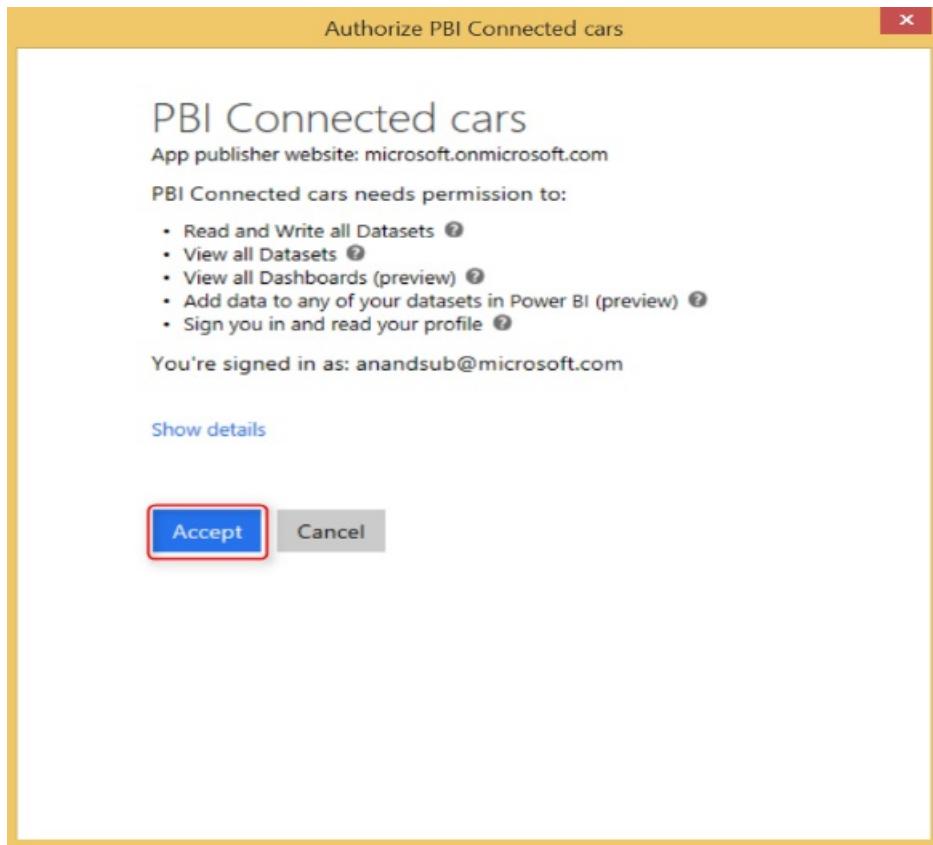
[Start the real-time dashboard application](#)

After the deployment is finished, follow the manual operation instructions.

1. Download the real-time dashboard application RealtimeDashboardApp.zip, and unzip it.
2. In the unzipped folder, open the app config file RealtimeDashboardApp.exe.config. Replace appSettings for Event Hubs, Azure Blob storage, and Azure Machine Learning service connections with the values in the manual operation instructions. Save your changes.
3. Run the application RealtimeDashboardApp.exe. On the sign-in window, enter your valid Power BI credentials.



4. Select **Accept**. The app starts to run.



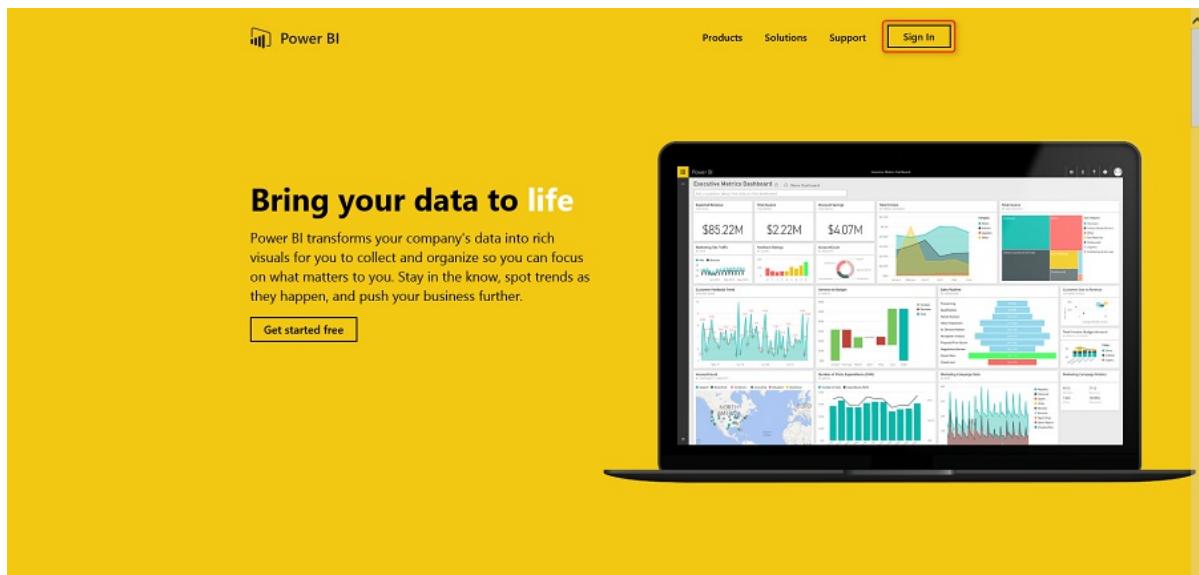
5. Sign in to the Power BI website, and create a real-time dashboard.

Now you're ready to configure the Power BI dashboard.

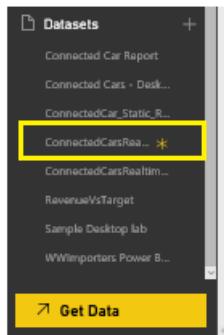
Configure Power BI reports

The real-time reports and the dashboard take about 30 to 45 minutes to finish.

1. Browse to the [Power BI](#) webpage, and sign in.



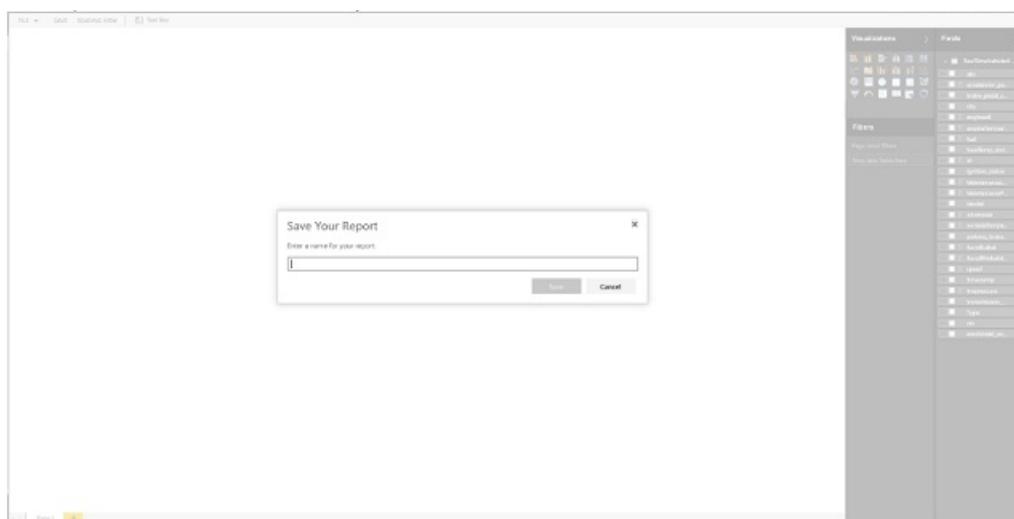
2. A new data set is generated in Power BI. Select the **ConnectedCarsRealtime** data set.



3. To save the blank report, press Ctrl+S.



4. Enter the report name **Vehicle Telemetry Analytics Real-time - Reports**.



Real-time reports

Three real-time reports are in this solution:

- Vehicles in Operation
- Vehicles Requiring Maintenance
- Vehicle Health Statistics

You can configure all three of the reports, or you can stop after any stage. You then can proceed to the next section on how to configure batch reports. We recommend that you create all three reports to visualize the full insights of the real-time path of the solution.

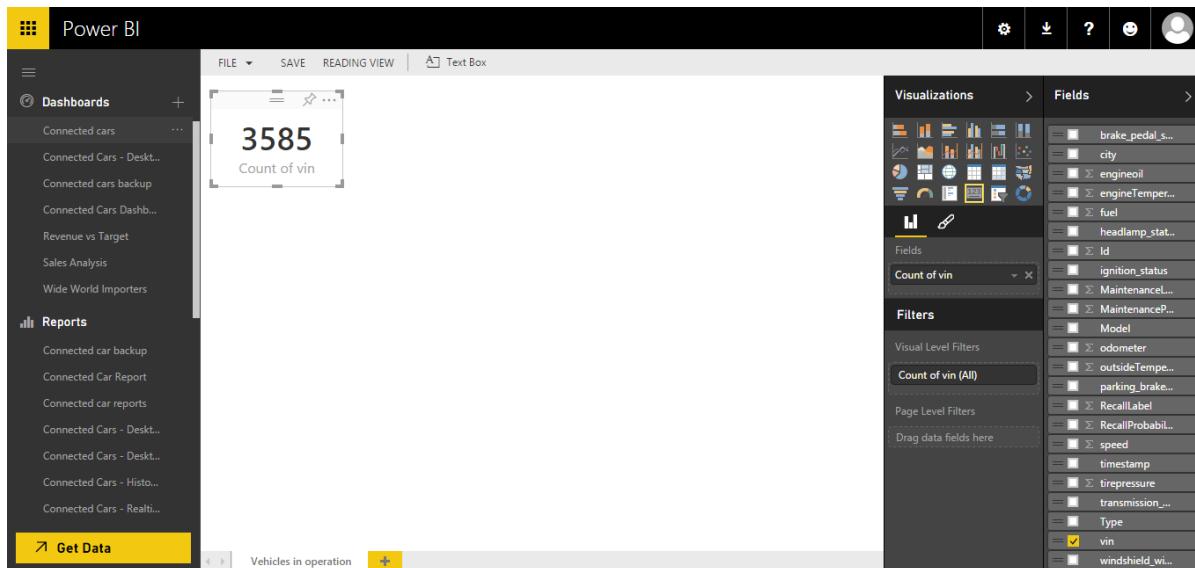
Vehicles in Operation report

- Double-click **Page 1**, and rename it **Vehicles in Operation**.



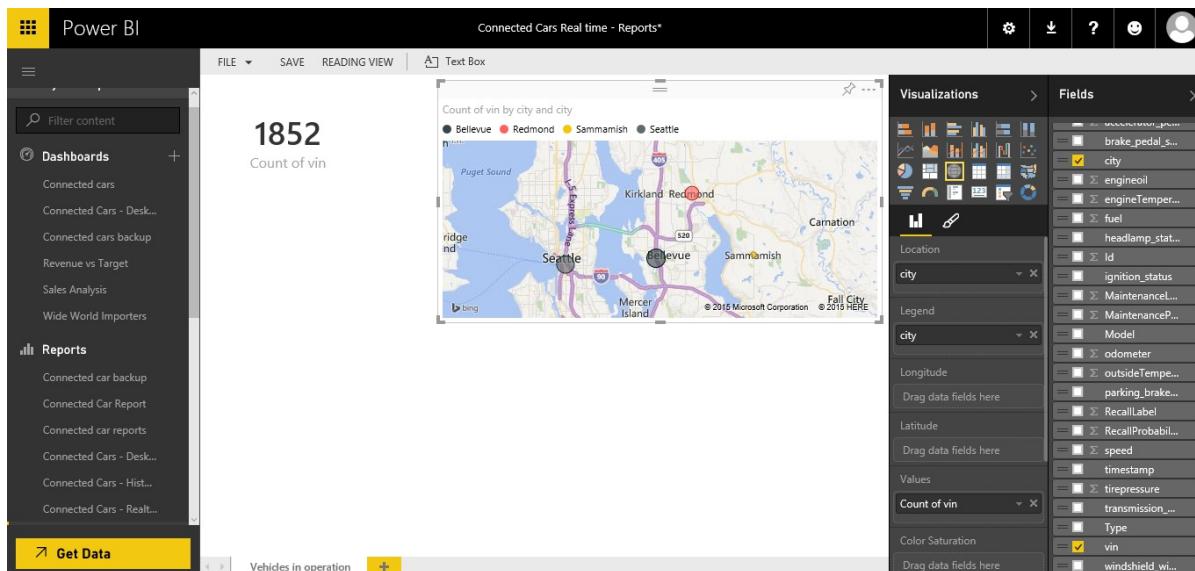
- On the **Fields** tab, select **vin**. On the **Visualizations** tab, select the **Card** visualization.

The **Card** visualization is created as shown in the following figure:

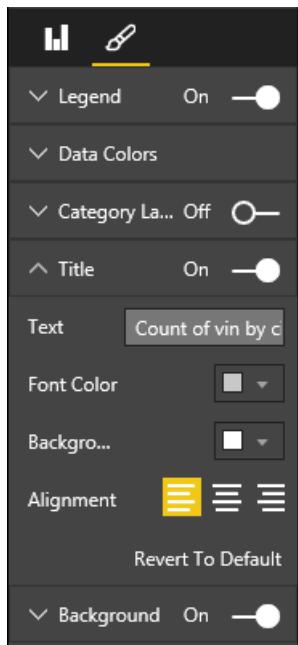


- Select the blank area to add a new visualization.

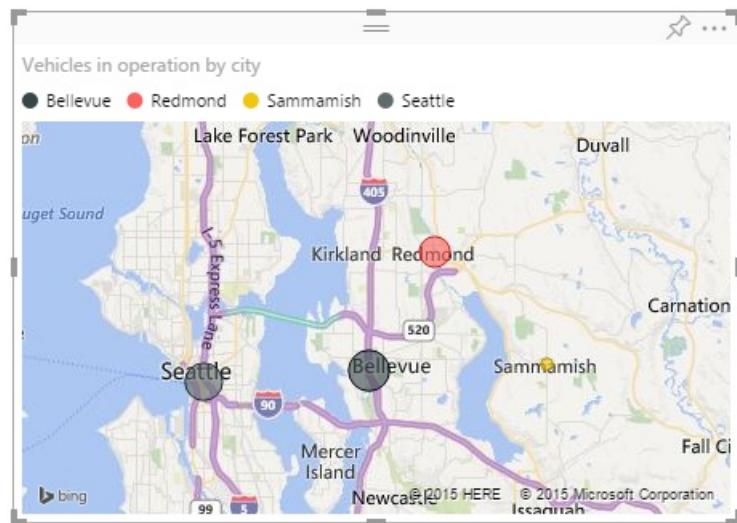
- On the **Fields** tab, select **city** and **vin**. On the **Visualizations** tab, select the **Map** visualization. Drag **vin** to the **Values** area. Drag **city** to the **Legend** area.



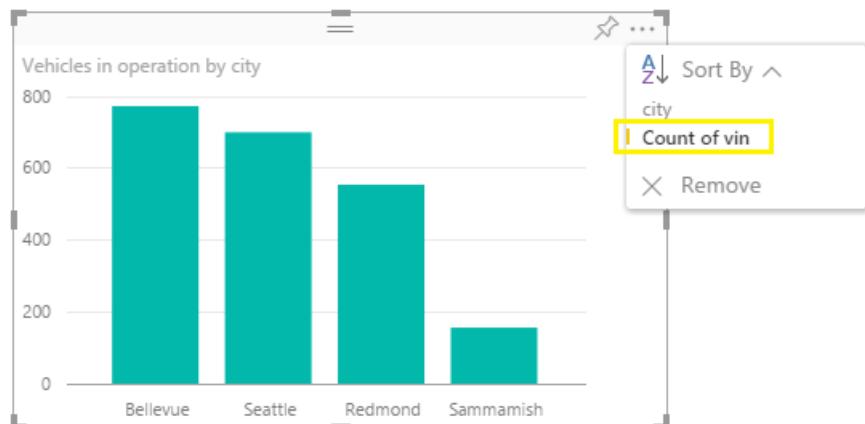
- On the **Visualizations** tab, select the **Format** section. Select **Title**, and change **Text** to **Vehicles in operation by city**.



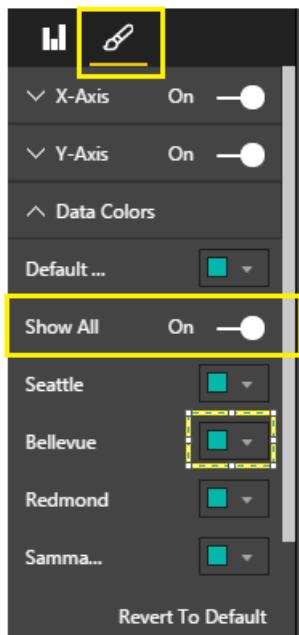
The final visualization looks like the following example:



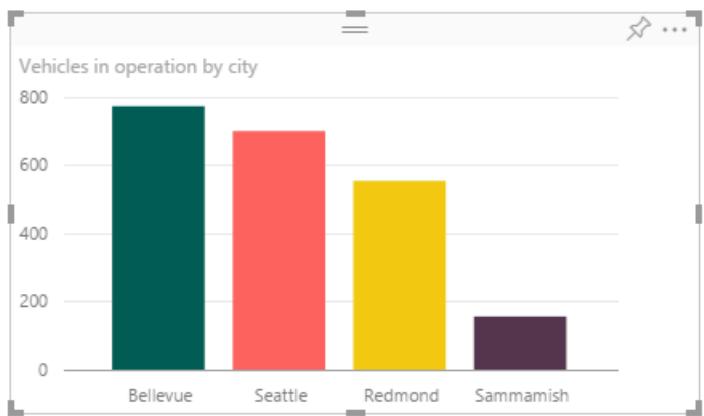
6. Select the blank area to add a new visualization.
7. On the **Fields** tab, select **city** and **vin**. On the **Visualizations** tab, select the **Clustered Column Chart** visualization. Drag **city** to the **Axis** area. Drag **vin** to the **Value** area.
8. Sort the chart by **Count of vin**.



9. Change the chart **Title** to **Vehicles in operation by city**.
10. Select the **Format** section, and then select **Data Colors**. Change **Show All** to **On**.



11. Change the color of an individual city by selecting the color symbol.

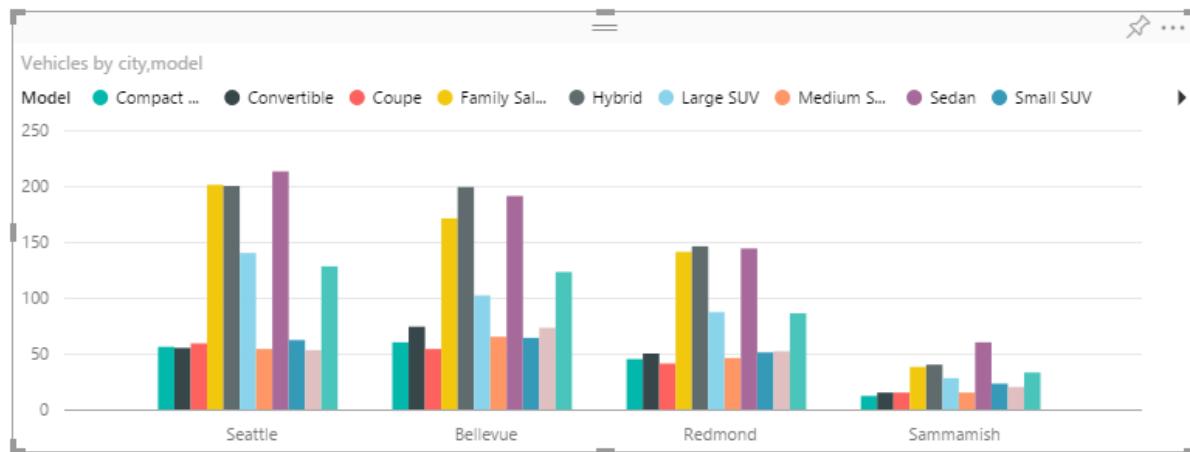


12. Select the blank area to add a new visualization.

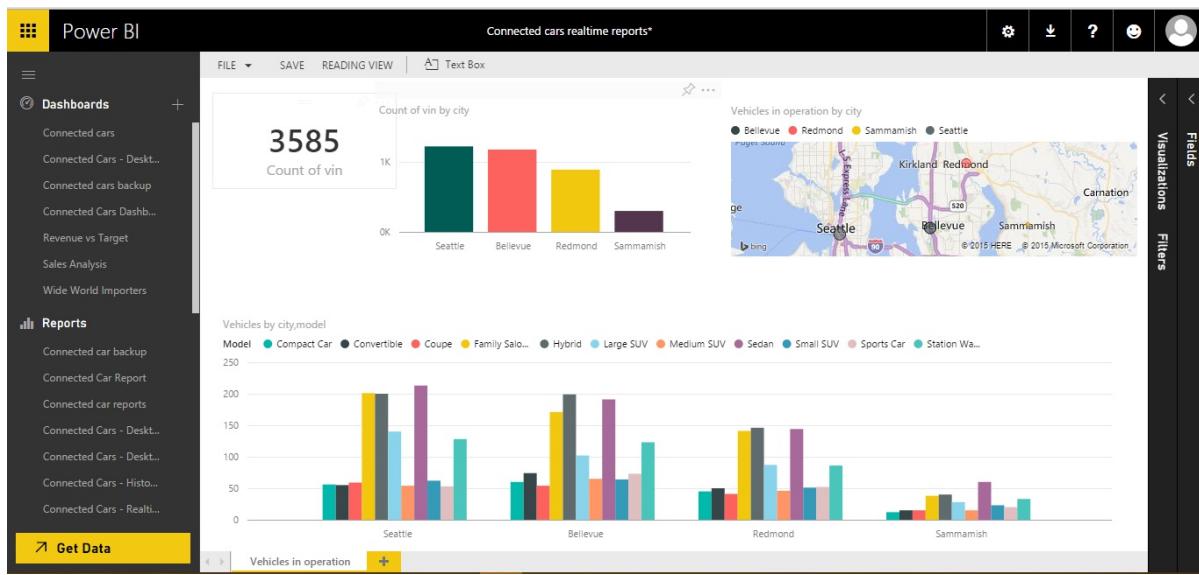
13. On the **Visualizations** tab, select the **Clustered Column Chart** visualization. On the **Fields** tab, drag **city** to the **Axis** area. Drag **Model** to the **Legend** area. Drag **vin** to the **Value** area.

The screenshot shows the Power BI Fields pane. On the left, there's a navigation bar with icons for different types of visualizations. Below it, there are sections for Axis, Legend, Model, Value, Color Saturation, and Filters. Under the Filters section, there's a 'Visual Level Filters' dropdown containing 'city (All)'. On the right, a list of fields is shown, many of which have checkboxes next to them. The checked fields are: city, Model, and vin.

The chart looks like the following image:



14. Rearrange all the visualizations so that the page looks like the following example:



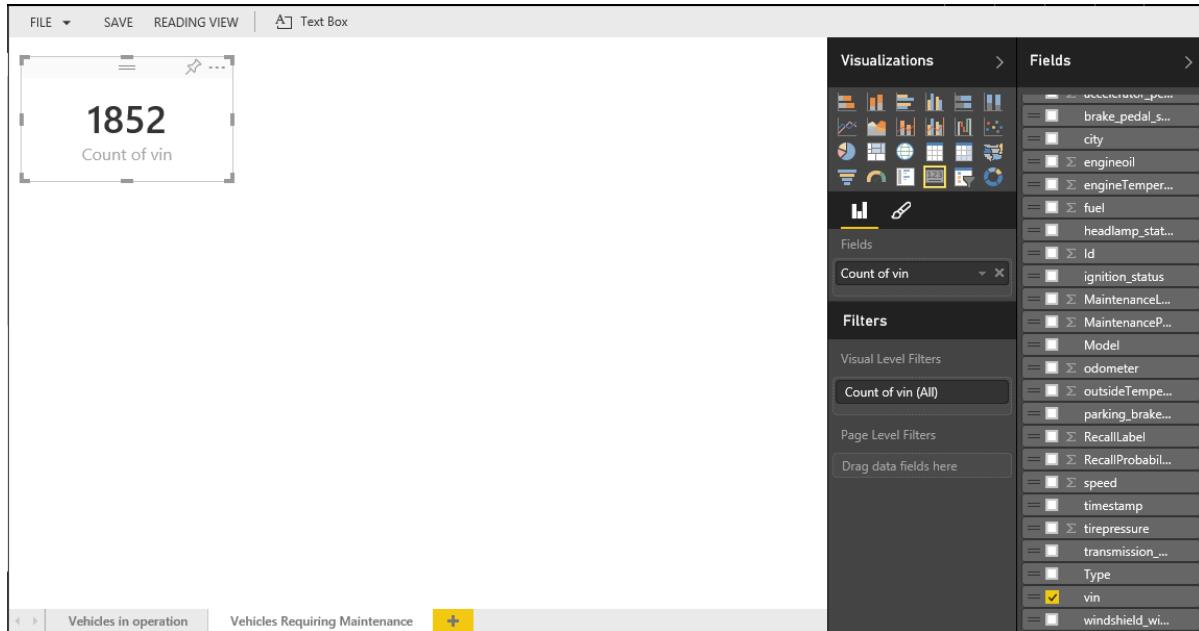
You successfully configured the "Vehicles in Operation" report. You can create the next real-time report, or you can stop here and configure the dashboard.

Vehicles Requiring Maintenance report

1. Select to add a new report. Rename it **Vehicles Requiring Maintenance**.



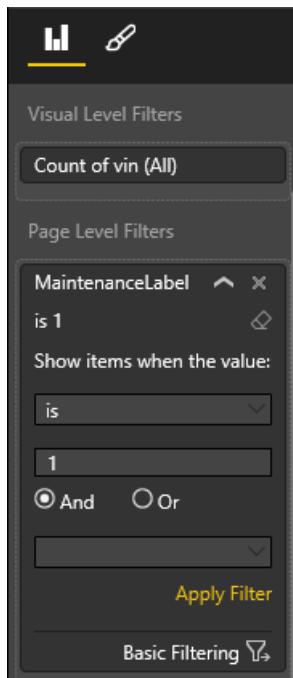
2. On the **Fields** tab, select **vin**. On the **Visualizations** tab, select the **Card** visualization.



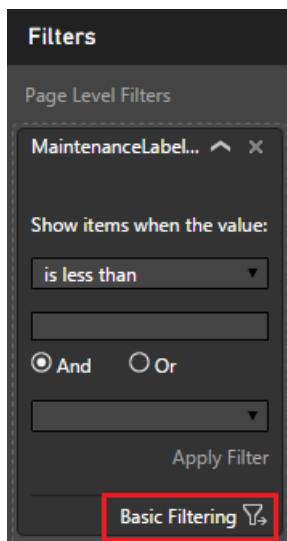
The data set contains a field named **MaintenanceLabel**. This field can have a value of "0" or "1." The value is set by the machine learning model that's provisioned as part of the solution. It's integrated with the real-time path. The value "1" indicates that a vehicle requires maintenance.

3. To add a **Page Level Filter** to show data for the vehicles that require maintenance:

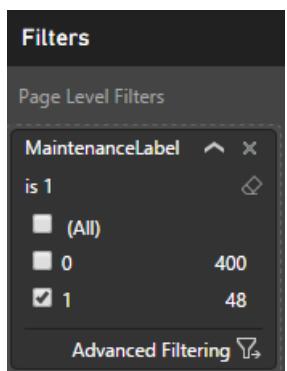
- a. Drag the **MaintenanceLabel** field to **Page Level Filters**.



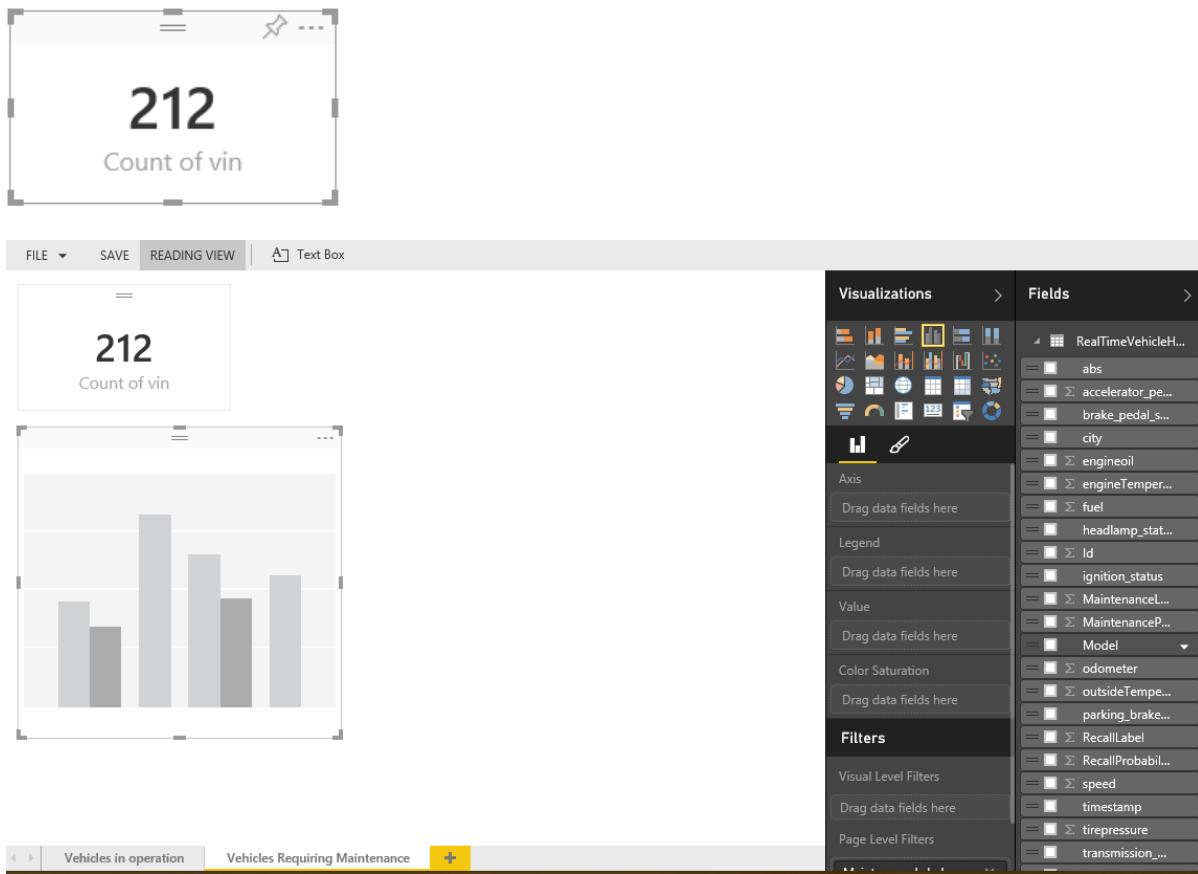
b. At the bottom of **Page Level Filters MaintenanceLabel**, select **Basic Filtering**.



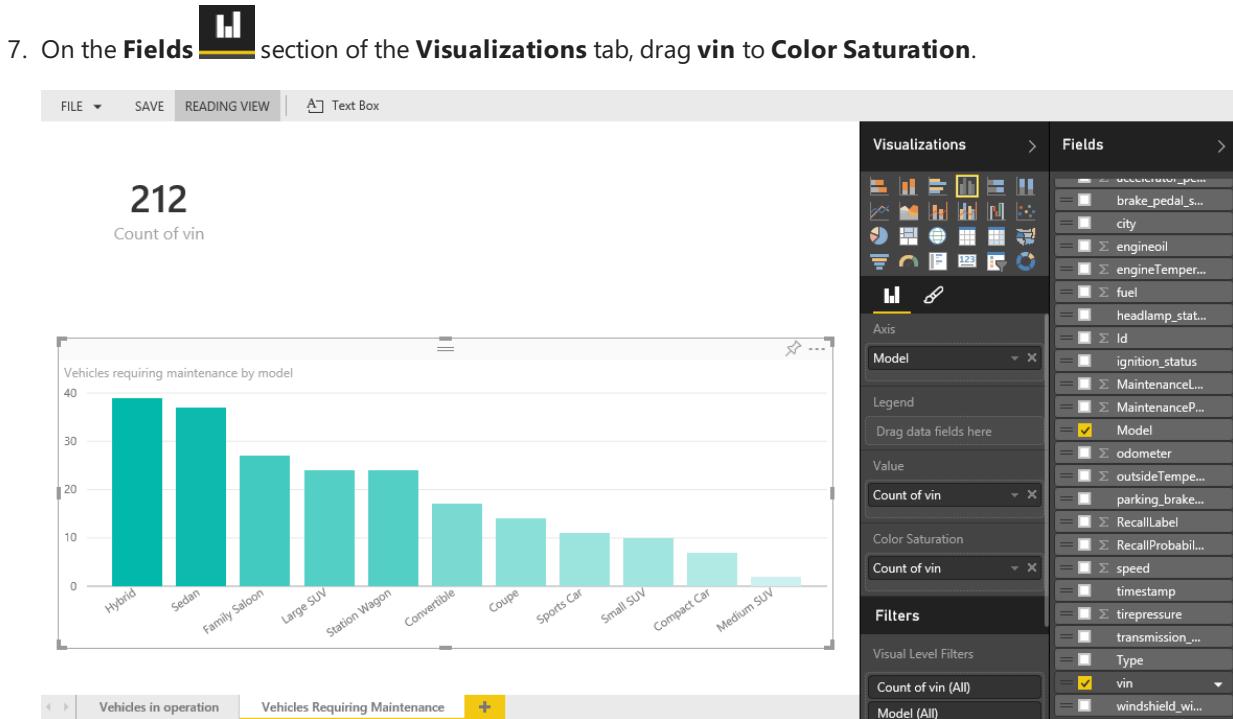
c. Set the filter value to **1**.



4. Select the blank area to add a new visualization.
5. On the **Visualizations** tab, select the **Clustered Column Chart** visualization.

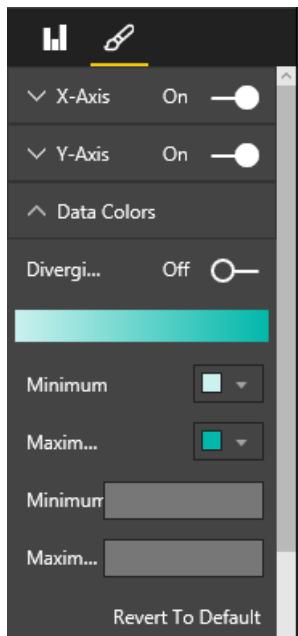


- On the **Fields** tab, drag **Model** to the **Axis** area. Drag **vin** to the **Value** area. Then sort the visualization by **Count of vin**. Change the chart **Title** to **Vehicles requiring maintenance by model**.

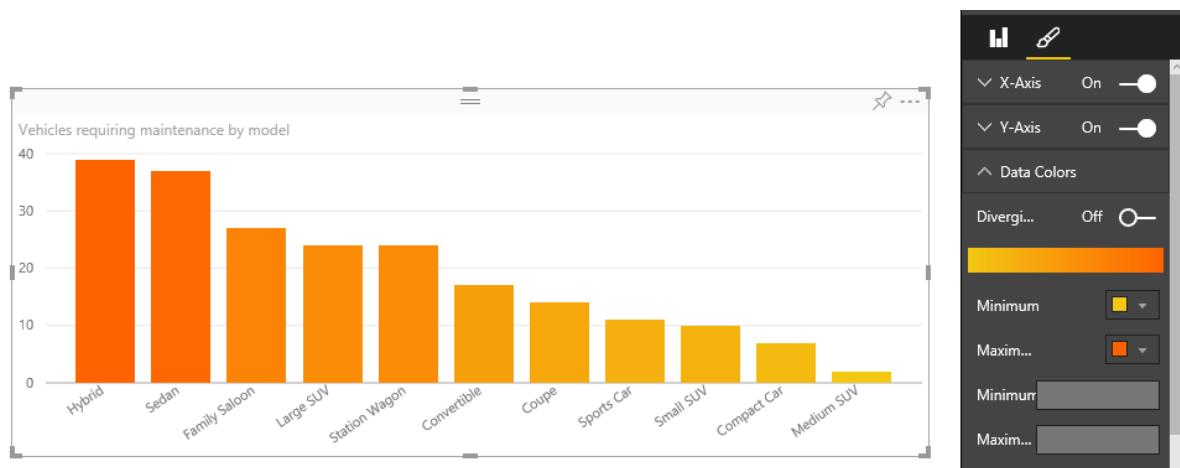


- On the **Format** section, change **Data Colors** in the visualization:

- Change the **Minimum** color to **F2C812**.
- Change the **Maximum** color to **FF6300**.



The new visualization colors look like the following example:



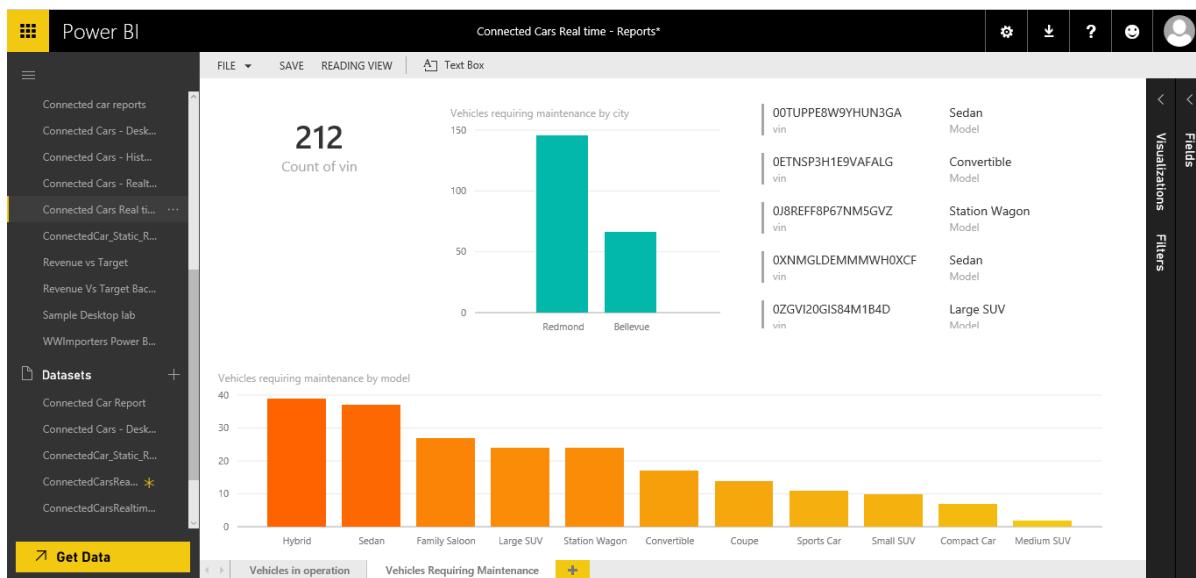
9. Select the blank area to add a new visualization.
10. On the **Visualizations** tab, select **Clustered Column Chart**. Drag **vin** to the **Value** area. Drag **city** to the **Axis** area. Sort the chart by **Count of vin**. Change the chart **Title** to **Vehicles requiring maintenance by city**.



11. Select the blank area to add a new visualization.
12. On the **Visualizations** tab, select the **Multi-Row Card** visualization. Drag **Model** and **vin** to the **Fields** area.

00TUPPE8W9YH...	Sedan
vin	Model
0ETNSP3H1E9VA...	Convertible
vin	Model
0J8REFF8P67NM5...	Station Wagon
vin	Model
0XNMGLDEMMM...	Sedan
vin	Model
0ZGVI20GIS84M1...	Large SUV
vin	Model

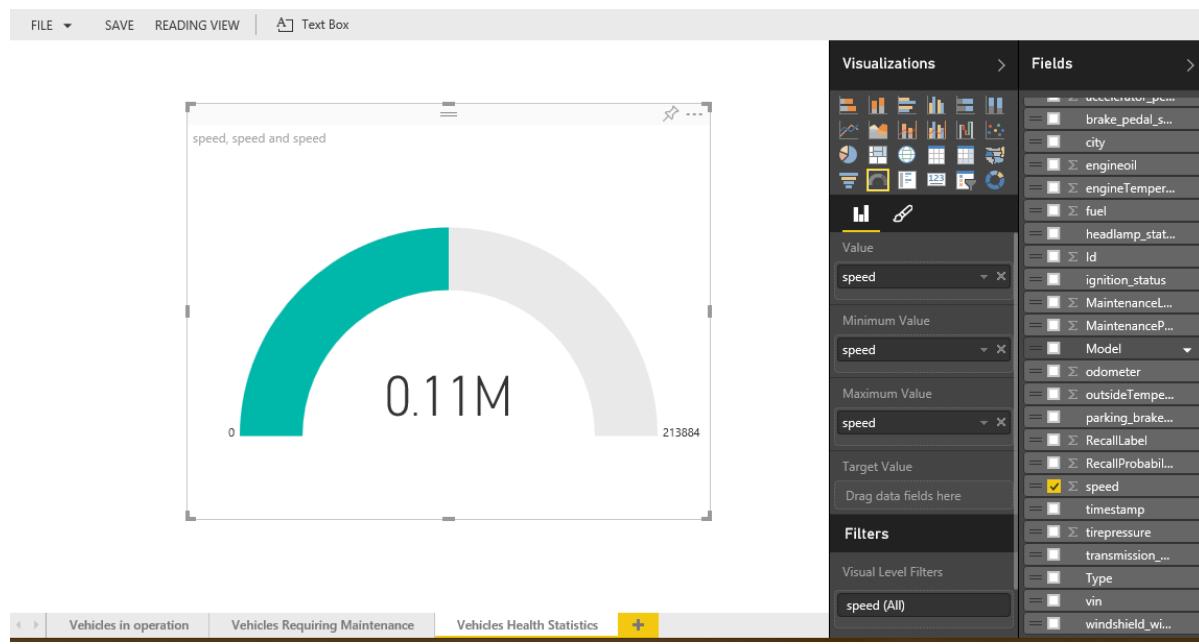
13. Rearrange all the visualizations so that the final report looks like the following example:



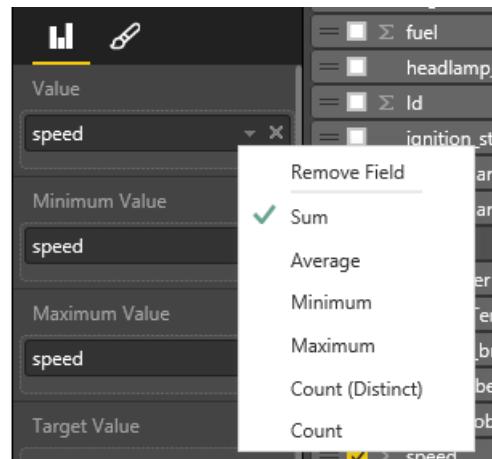
You successfully configured the "Vehicles Requiring Maintenance" real-time report. You can create the next real-time report, or you can stop here and configure the dashboard.

Vehicle Health Statistics report

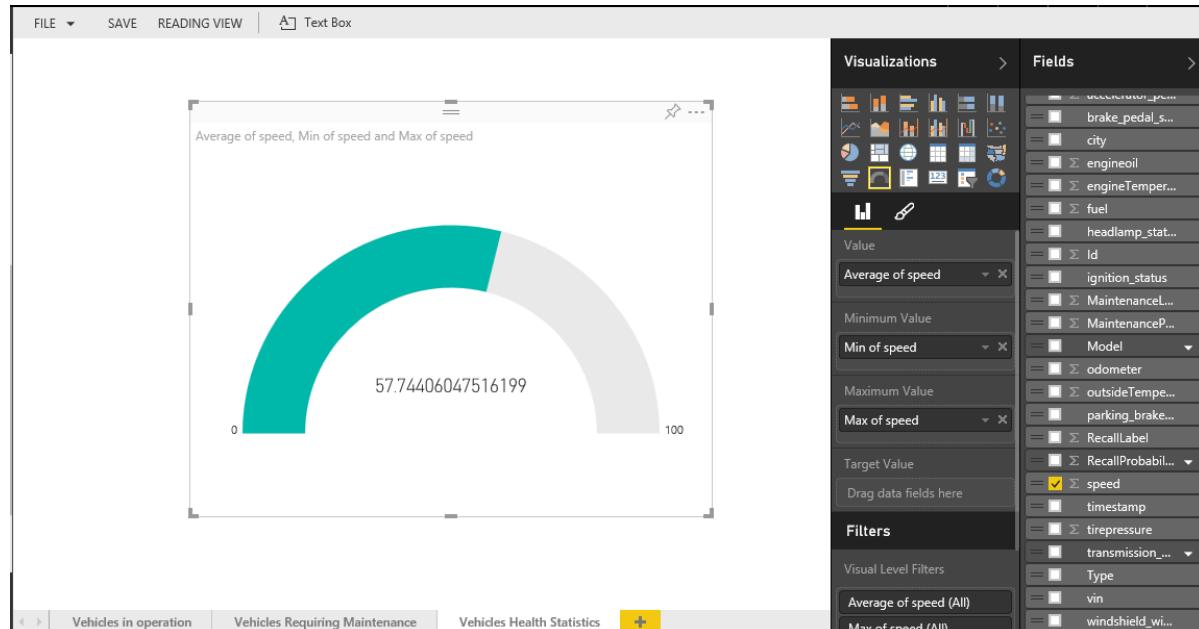
- Select to add a new report. Rename it **Vehicles Health Statistics**.
- On the **Visualizations** tab, select the **Gauge** visualization. Drag **speed** to the **Value**, **Minimum Value**, and **Maximum Value** areas.



3. In the **Value** area, change the default aggregation of **speed** to **Average**.
4. In the **Minimum Value** area, change the default aggregation of **speed** to **Minimum**.
5. In the **Maximum Value** area, change the default aggregation of **speed** to **Maximum**.



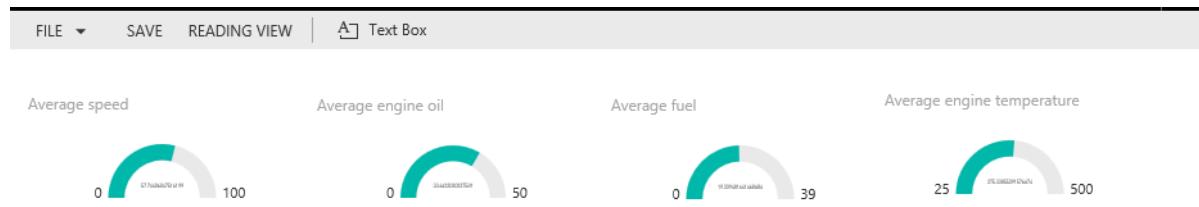
6. Rename the **Gauge Title** to **Average speed**.



7. Select the blank area to add a new visualization.

Similarly, add a **Gauge** for **Average engine oil**, **Average fuel**, and **Average engine temperature**.

8. Change the default aggregation of fields in each gauge like you did in the previous steps in the **Average speed** gauge.



9. Select the blank area to add a new visualization.

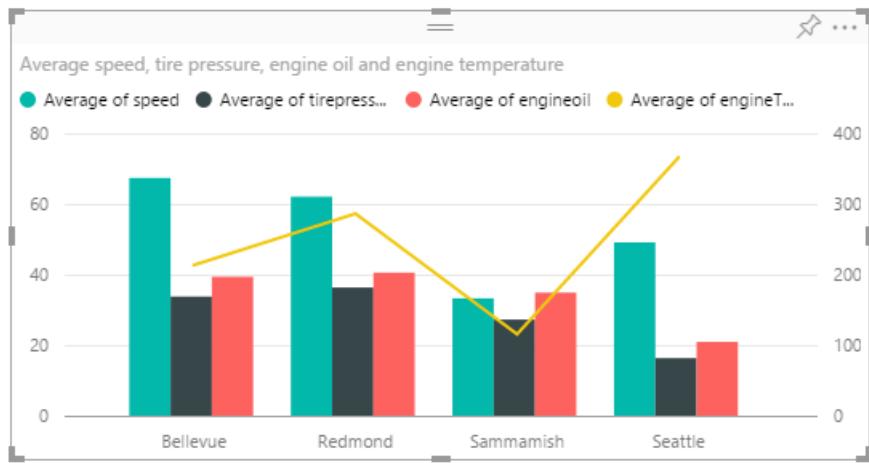
10. On the **Visualizations** tab, select the **Line and Clustered Column Chart** visualization. Drag **city** to **Shared Axis**. Drag **tirepressure**, **engineoil**, and **speed** to the **Column Values** area. Change their aggregation type to **Average**.

11. Drag **engineTemperature** to the **Line Values** area. Change the aggregation type to **Average**.

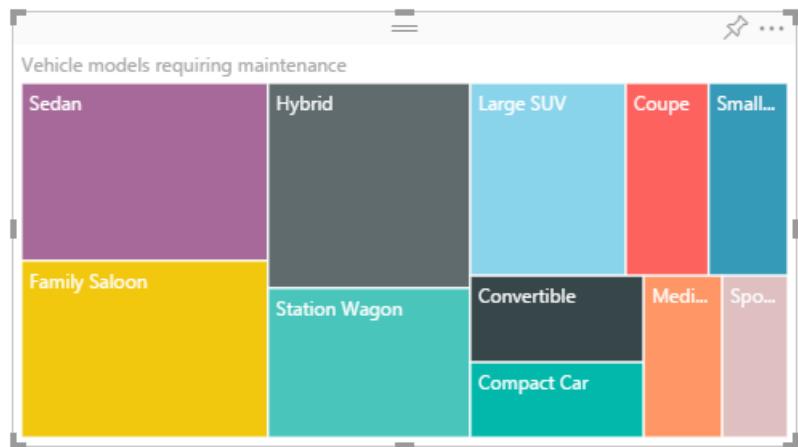
The screenshot shows the Power BI Visualizations pane with the following configuration:

- Visualizations**: Shows various chart icons, with the "Line and Clustered Column Chart" icon selected.
- Fields**:
 - Shared Axis**: Set to **city**.
 - Column Series**: "Drag data fields here".
 - Column Values**: Contains three items with aggregation type set to **Average**:
 - Average of tirepressure
 - Average of engineoil
 - Average of speed
 - Line Values**: Contains one item with aggregation type set to **Average**:
 - Average of engineTe...
 - Filters**: No filters applied.
 - Visual Level Filters**: No filters applied.

12. Change the chart **Title** to **Average speed, tire pressure, engine oil, and engine temperature**.



13. Select the blank area to add a new visualization.
14. On the **Visualizations** tab, select the **Treemap** visualization. Drag **Model** to the **Group** area. Drag **MaintenanceProbability** to the **Values** area.
15. Change the chart **Title** to **Vehicle models requiring maintenance**.

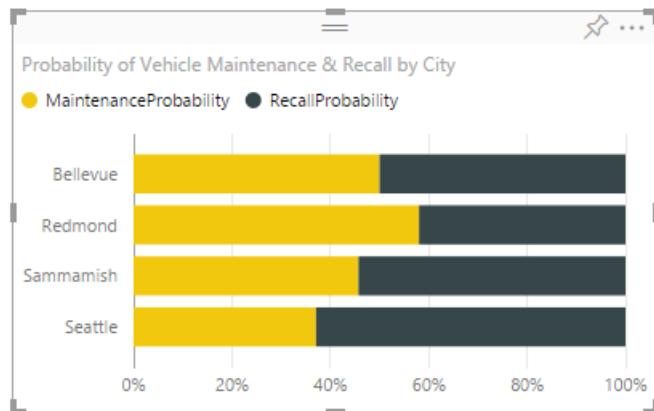


16. Select the blank area to add a new visualization.
17. On the **Visualizations** tab, select the **100% Stacked Bar Chart** visualization. Drag **city** to the **Axis** area. Drag **MaintenanceProbability** and **RecallProbability** to the **Value** area.

The screenshot shows the Power BI Fields pane. On the left, there's a grid of visualization icons. Below it, the 'Axis' section has 'city' selected. In the 'Value' section, 'MaintenanceProbability' and 'RecallProbability' are listed under dropdown menus. The 'Filters' section contains a single filter: 'city (All)'. The main pane lists numerous fields, many with aggregation operators (Σ) and checkmarks indicating they are used in the current view. Some visible fields include 'brake_pedal_s...', 'city', 'engineoil', 'engineTemper...', 'fuel', 'headlamp_stat...', 'Id', 'ignition_status', 'MaintenanceL...', 'MaintenanceP...', 'Model', 'odometer', 'outsideTempe...', 'parking_brake...', 'RecallLabel', 'RecallProbabil...', 'speed', 'timestamp', 'tirepressure', 'transmission...', 'Type', 'vin', and 'windshield_wi...'. Most fields have a small yellow square icon next to them.

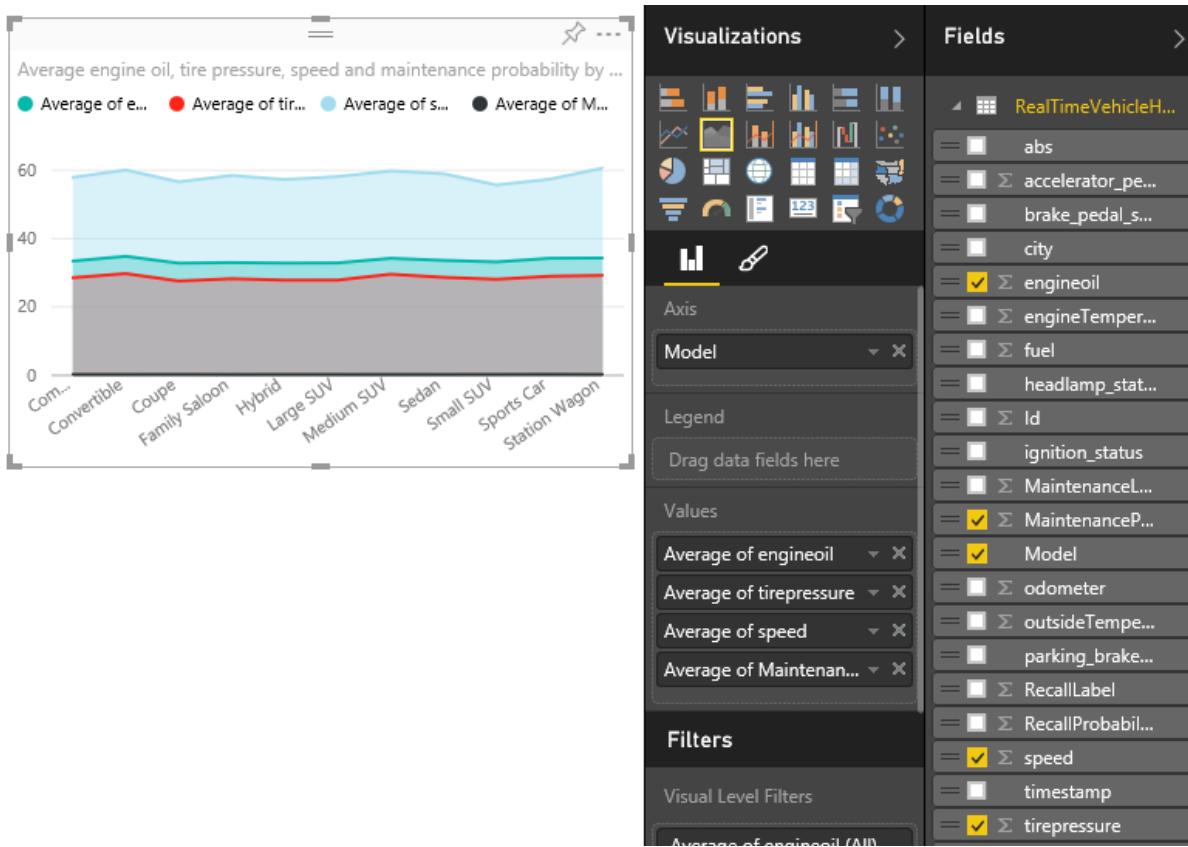
18. On the **Format** section, select **Data Colors**. Set the **MaintenanceProbability** color to the value **F2C80F**.

19. Change the chart **Title** to **Probability of Vehicle Maintenance & Recall by City**.

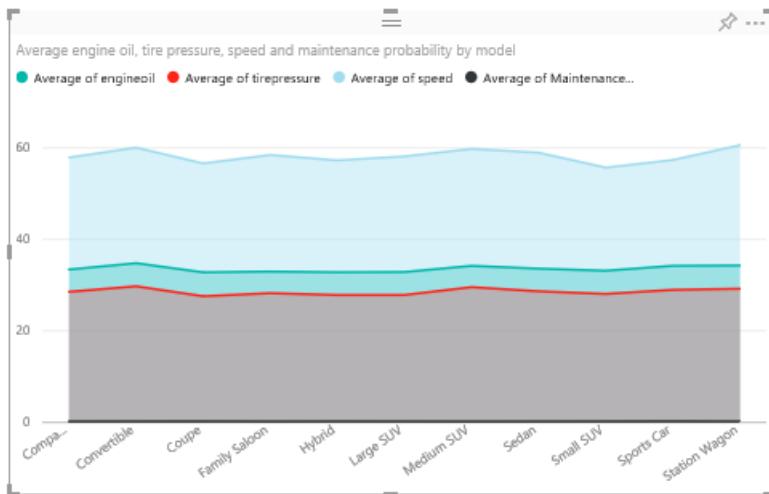


20. Select the blank area to add a new visualization.

21. On the **Visualizations** tab, select the **Area Chart** visualization. Drag **Model** to the **Axis** area. Drag **engineOil**, **tirepressure**, **speed**, and **MaintenanceProbability** to the **Values** area. Change their aggregation type to **Average**.



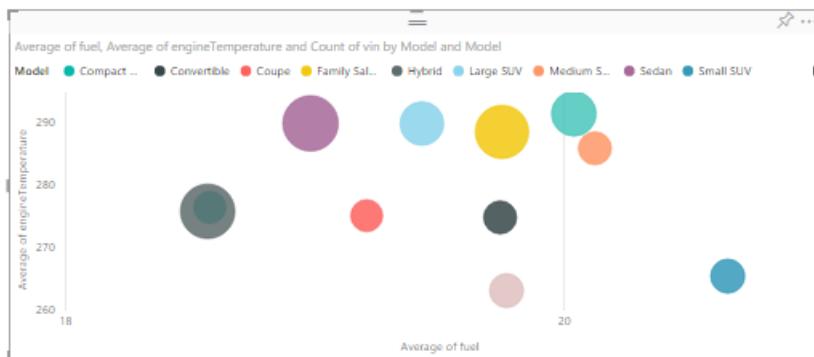
22. Change the chart **Title** to **Average engine oil, tire pressure, speed, and maintenance probability by model**.



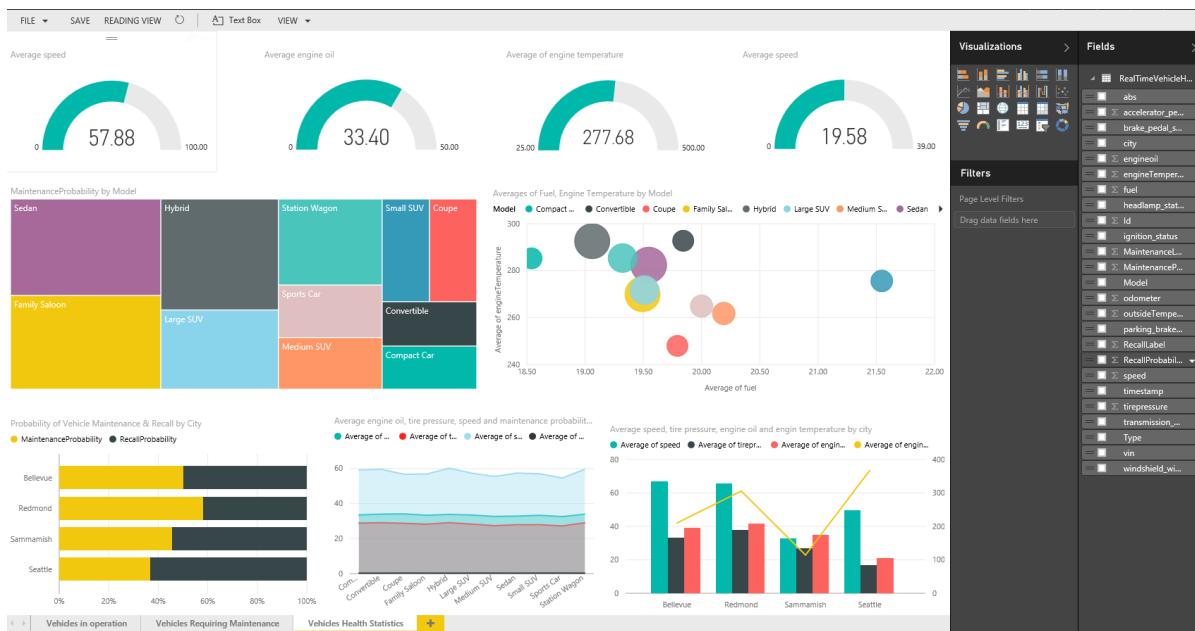
23. Select the blank area to add a new visualization.

24. On the **Visualizations** tab, select the **Scatter Chart** visualization. Drag **Model** to the **Details** and **Legend** areas. Drag **fuel** to the **X-Axis** area. Change the aggregation to **Average**. Drag **engineTemperature** to the **Y-Axis** area. Change the aggregation to **Average**. Drag **vin** to the **Size** area.

25. Change the chart **Title** to **Average of fuel, Average of engineTemperature, and Count of vin by Model and Model.**

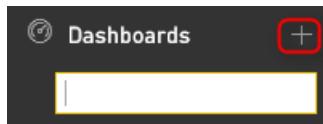


The final report looks like the following example:

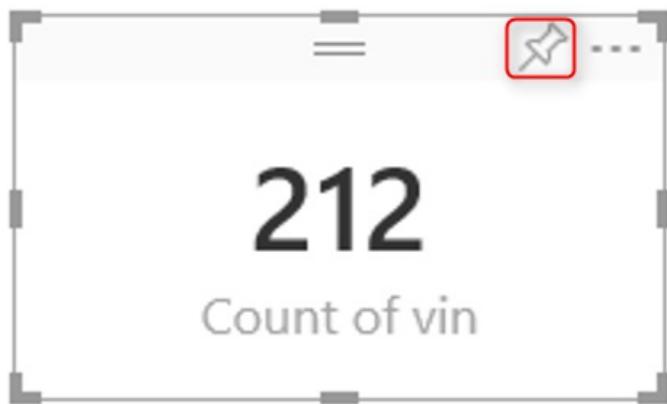


Pin visualizations from the reports to the real-time dashboard

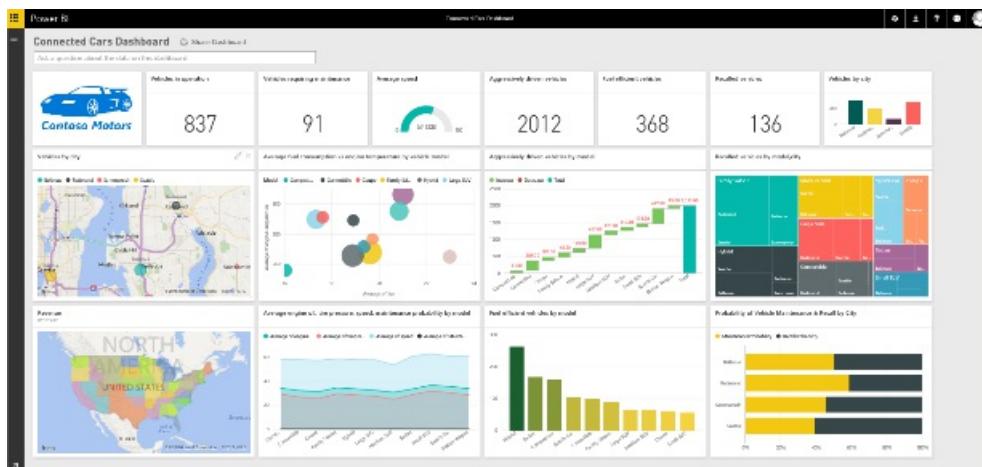
1. Create a blank dashboard by selecting the plus symbol next to **Dashboards**. Enter the name **Vehicle Telemetry Analytics Dashboard**.



2. Pin the visualizations from the previous reports to the dashboard.



When all three reports are pinned to the dashboard, it should look like the following example. If you didn't create all the reports, your dashboard might look different.



You successfully created the real-time dashboard. As you continue to execute CarEventGenerator.exe and RealtimeDashboardApp.exe, you see live updates on the dashboard. The following steps take about 10 to 15 minutes to complete.

Set up the Power BI batch processing dashboard

NOTE

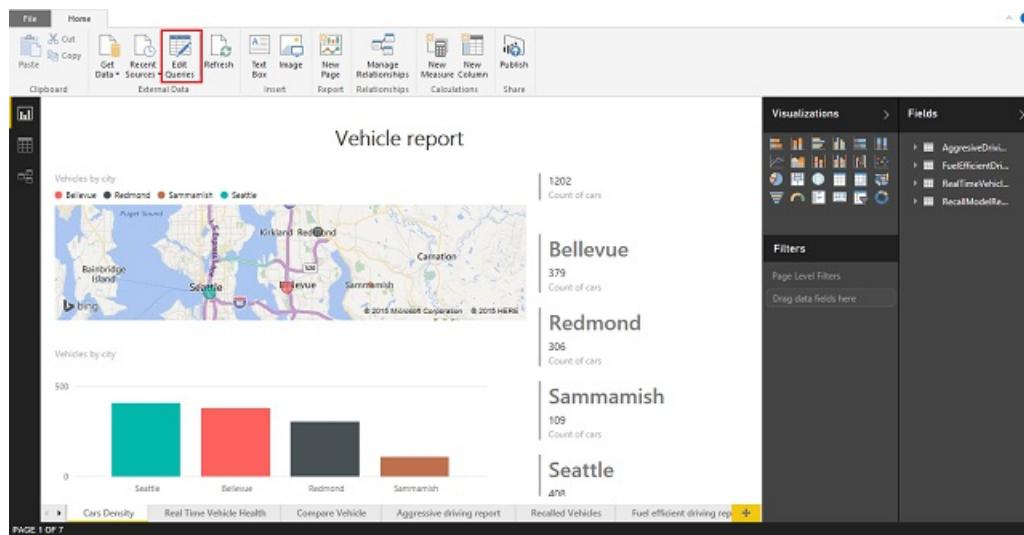
It takes about two hours (from the successful completion of the deployment) for the end-to-end batch processing pipeline to finish execution and process a year's worth of generated data. Wait for the processing to finish before you proceed with the following steps:

Download the Power BI designer file

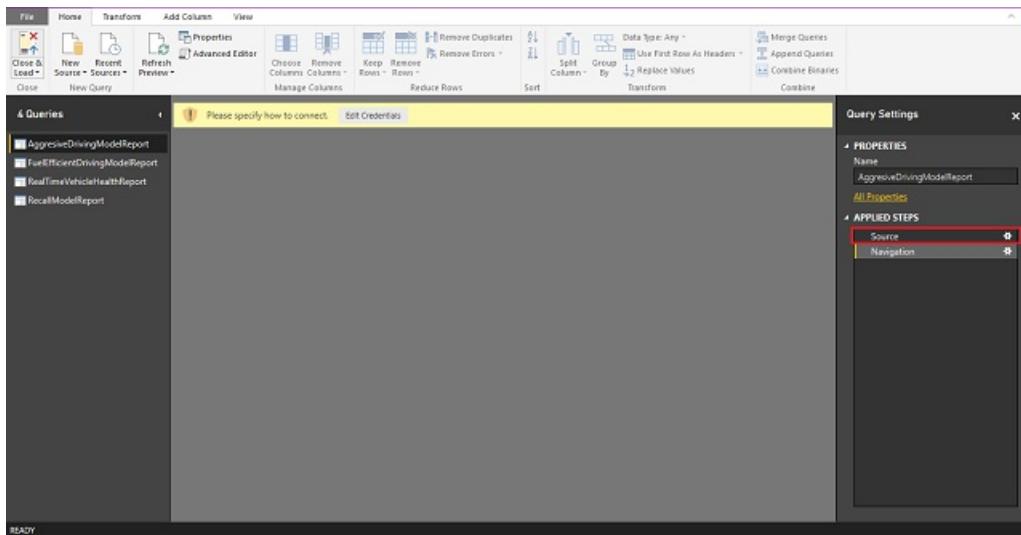
1. A preconfigured Power BI designer file is included as part of the deployment manual operation instructions. Look for "2. Set up the PowerBI batch processing dashboard."
2. Download the Power BI template for batch processing dashboard here called **ConnectedCarsPbiReport.pbix**.
3. Save it locally.

Configure Power BI reports

1. Open the designer file **ConnectedCarsPbiReport.pbix** by using the Power BI Desktop. If you don't already have it, install the Power BI Desktop from the [Power BI Desktop installation website](#).
2. Select **Edit Queries**.



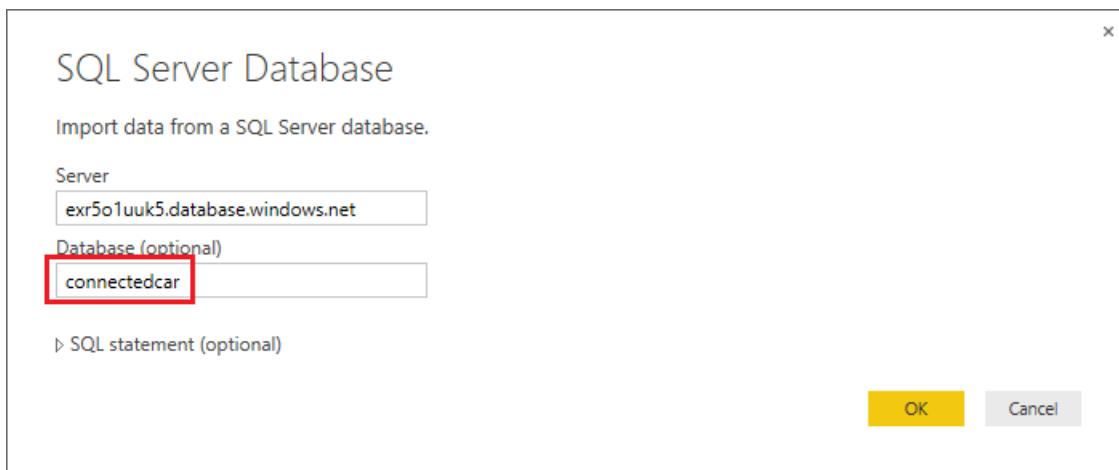
3. Double-click **Source**.



4. Update the server connection string with the Azure SQL server that got provisioned as part of the deployment. Look in the manual operation instructions under Azure SQL database:

- Server: somethingsrv.database.windows.net
- Database: connectedcar
- Username: username
- Password: You can manage your SQL Server password from the Azure portal.

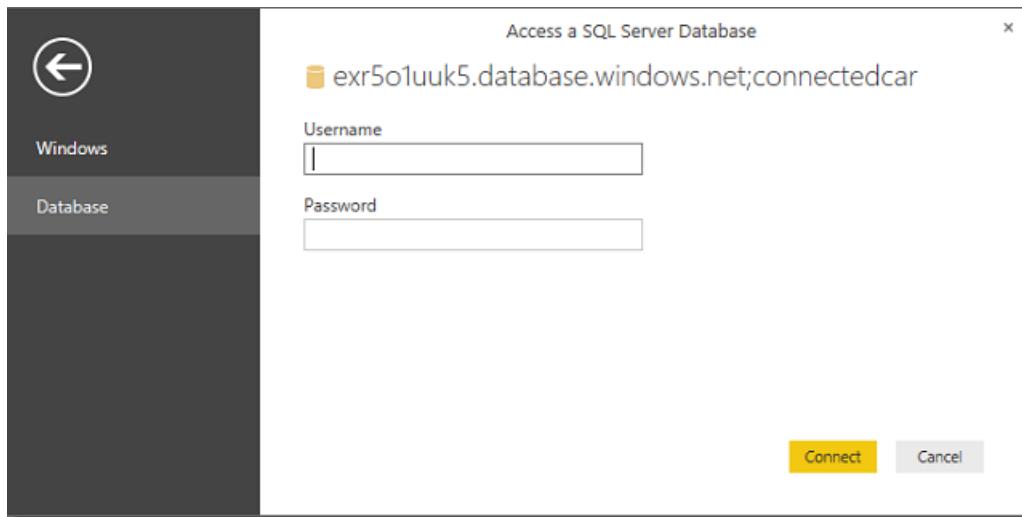
5. Leave **Database** as **connectedcar**.



6. Select **OK**.

7. The **Windows credential** tab is selected by default. Change it to **Database credentials** by selecting the **Database** tab at the right.

8. Enter the **Username** and **Password** of your Azure SQL database that was specified during its deployment setup.



9. Select **Connect**.

10. Repeat the previous steps for each of the three remaining queries present in the right pane. Then update the data source connection details.
11. Select **Close and Load**. Power BI Desktop file data sets are connected to SQL database tables.
12. Select **Close** to close the Power BI Desktop file.

13. Select **Save** to save the changes.

You have now configured all the reports that correspond to the batch processing path in the solution.

Upload to powerbi.com

1. Go to the [Power BI web portal](#), and sign in.
2. Select **Get Data**.
3. Upload the Power BI Desktop file. Select **Get Data > Files Get > Local file**.
4. Go to **ConnectedCarsPbiReport.pbix**.
5. After the file is uploaded, go back to your Power BI work space. A dataset, a report, and a blank dashboard are created for you.
6. Pin charts to a new dashboard called **Vehicle Telemetry Analytics Dashboard** in Power BI. Select the blank dashboard that was previously created, and then go to the **Reports** section. Select the newly uploaded report.

The screenshot shows the Power BI interface with the 'Connected Cars Dashboard' selected. On the left, there's a navigation pane with various reports and datasets listed under categories like 'Reports' and 'Datasets'. A yellow box highlights a report card icon in the center of the dashboard area.

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 Motors logo

This screenshot shows the 'Connected Cars - Desktop Report' page. It features a bar chart titled 'Vehicles in operation' comparing car counts across four cities: Bellevue, Redmond, Sammamish, and Seattle. To the right is a map of the Seattle metropolitan area with specific locations marked. Below the chart is a waterfall chart showing vehicle models by city and model type.

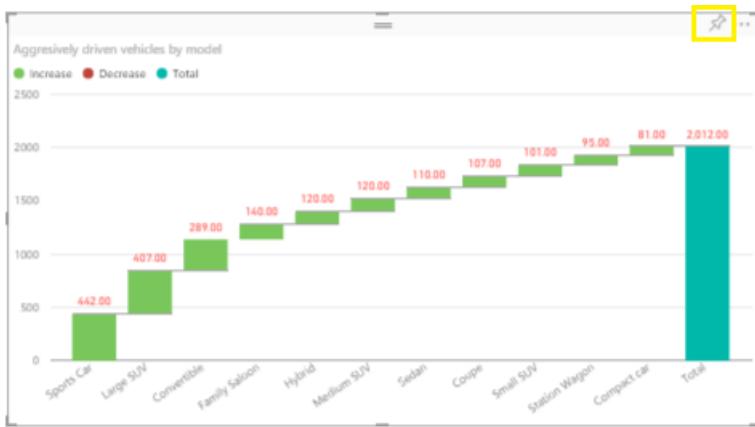
City	Count of cars
Bellevue	379
Redmond	306
Sammamish	109
Seattle	408

7. From **Page 3**, pin the following content:

a. Count of vin

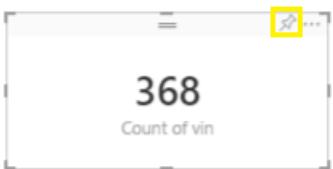


b. Aggressively driven vehicles by model – Waterfall chart

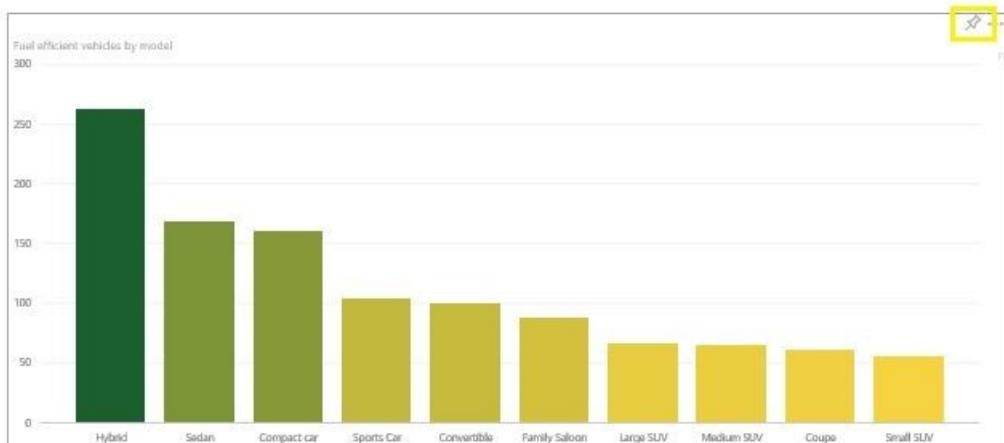


8. From **Page 5**, pin the following content:

a. **Count of vin**

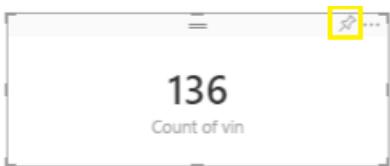


b. **Fuel-efficient vehicles by model: Clustered column chart**



9. From **Page 4**, pin the following content:

a. **Count of vin**



b. **Recalled vehicles by city, model: Treemap**



10. From **Page 6**, pin the following content:

- **Contoso Motors logo**



Organize the dashboard

1. Go to the dashboard.
2. Hover over each chart. Rename each chart based on the naming provided in the following finished dashboard example:

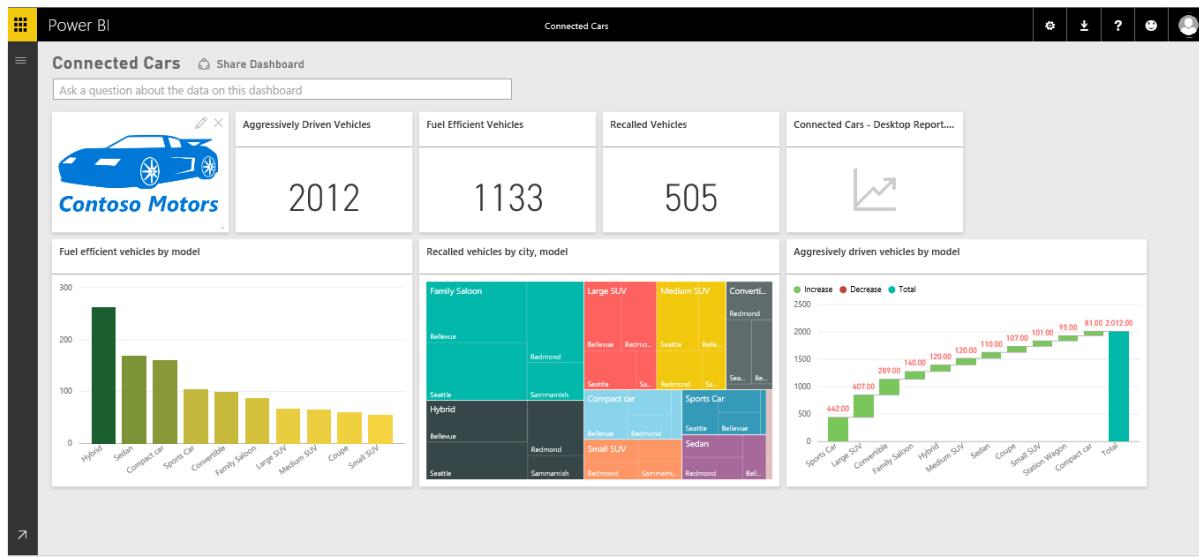
The dashboard contains the following tiles:

- Aggressively Driven Vehicles**: A bar chart showing fuel efficiency by vehicle model for the year 2012. The models include Hybrid, Sedan, Compact car, Sports Car, Convertible, Family Saloon, Large SUV, Medium SUV, Coupe, and Small SUV. The chart shows a significant difference between the top (Hybrid) and bottom (Small SUV) models.
- Fuel Efficient Vehicles**: A summary tile showing the count of 1133 fuel-efficient vehicles.
- Recalled Vehicles**: A treemap chart showing recalled vehicles categorized by city (Bellevue, Redmond, Seattle, Sammamish) and vehicle type (Family Saloon, Large SUV, Medium SUV, Compact car, Sports Car, Sedan, Coupe, Small SUV).
- Connected Cars - Desktop Report...**: A line chart showing the number of aggressively driven vehicles over time, with data points for Sports Car, Large SUV, Convertible, Family Saloon, Hybrid, Medium SUV, and Sedan.

Tile Details pane (right side):

- Title Title**: Aggressively Driven Vehicles (highlighted with a yellow box)
- Set custom link: []
- Apply** | **Discard**

3. Move the charts around to look like the following dashboard example:



4. After you create all the reports mentioned in this document, the final finished dashboard looks like the following example:



You have successfully created the reports and the dashboard to gain real-time, predictive, and batch insights on vehicle health and driving habits.