

Contents

Team Data Science Process Documentation

Overview

Lifecycle

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

Roles and tasks

Group manager

Team lead

Project lead

Individual contributor

Project planning

Development

Agile development

Collaborative coding with Git

Execute data science tasks

Code testing

Track progress

Operationalization

DevOps - CI/CD

Worked-out examples

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 Azure Synapse Analytics](#)

Training

[For data scientists](#)

[For DevOps](#)

How To

[Set up data science environments](#)

[Azure storage accounts](#)

[Platforms and tools](#)

[R and Python on HDInsight clusters](#)

[Introduction to Spark on HDInsight](#)

[Create an Apache Spark cluster in Azure HDInsight](#)

[PySpark kernels for Jupyter Notebook](#)

[R Server on HDInsight](#)

[Get started using R Server on HDInsight](#)

[Machine Learning Studio \(classic\) workspace](#)

[Analyze business needs](#)

[Identify your scenario](#)

[Acquire and understand data](#)

[Ingest data](#)

[Overview](#)

[Move to/from Blob storage](#)

[Overview](#)

[Use Storage Explorer](#)

[Use AzCopy](#)

[Use Python](#)

[Use SSIS](#)

[Move to SQL on a VM](#)

[Move to Azure SQL Database](#)

[Move to Hive tables](#)

[Move to SQL partitioned tables](#)

[Move from on-prem SQL](#)

[Explore and visualize data](#)

[Prepare data](#)

[Explore data](#)

[Overview](#)

[Explore Azure Blob Storage](#)

[Explore SQL on a VM](#)

[Explore Hive tables](#)

[Sample data](#)

[Overview](#)

[Use Blob Storage](#)

[Use SQL Server](#)

[Use Hive tables](#)

[Process data](#)

[Access with Python](#)

[Process blob data](#)

[Use Azure Data Lake](#)

[Use SQL VM](#)

[Use data pipeline](#)

[Use Spark](#)

[Use Scala and Spark](#)

[Develop models](#)

[Engineer features](#)

[Overview](#)

[Use SQL+Python](#)

[Use Hive queries](#)

[Select features](#)

[Create and train models](#)

[Choose algorithms](#)

[Algorithm cheat sheet](#)

[Deploy models in production](#)

[Related](#)

[Cognitive Services](#)

[Anomaly detection](#)

[Predictive maintenance](#)

[Overview](#)

[Architecture](#)

[Technical guide](#)

[Resources](#)

[Organizations using TDSP](#)

[New Signature](#)

[Blue Granite](#)

[Related Microsoft resources](#)

[Microsoft Q&A question page](#)

[Stack Overflow](#)

[Videos](#)

What is the Team Data Science Process?

3/5/2021 • 4 minutes 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 by suggesting how team roles work best together. TDSP includes best practices and structures from Microsoft and other industry leaders to help toward 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 different kinds 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 has the following key components:

- **A data science lifecycle** definition
- **A standardized project structure**
- **Infrastructure and resources** recommended for data science projects
- **Tools and utilities** recommended 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 full steps that successful projects follow.

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.

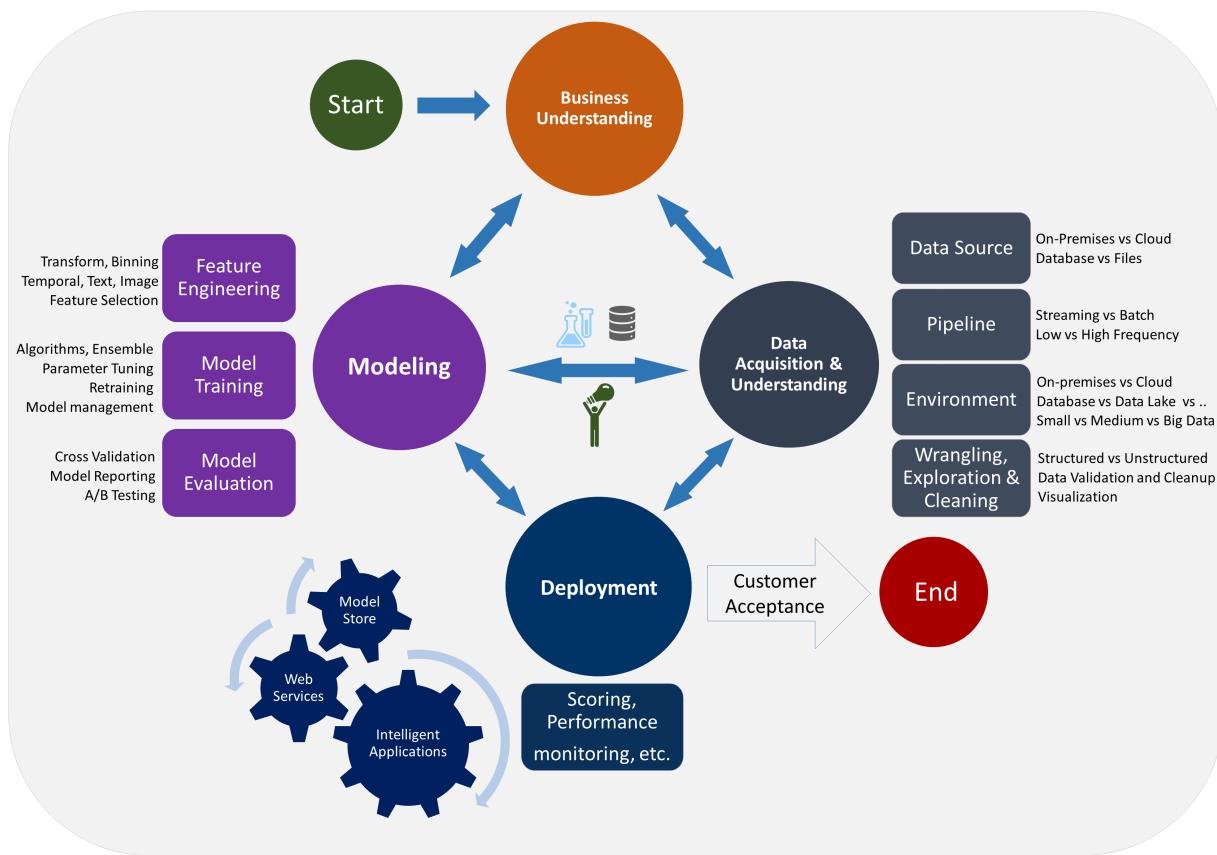
This lifecycle has been designed for data science projects that ship as part of intelligent applications. These applications deploy machine learning or artificial intelligence models for predictive analytics. Exploratory data science projects or improvised analytics projects can also benefit from using 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**

Here is a visual representation of the [Team Data Science Process lifecycle](#).

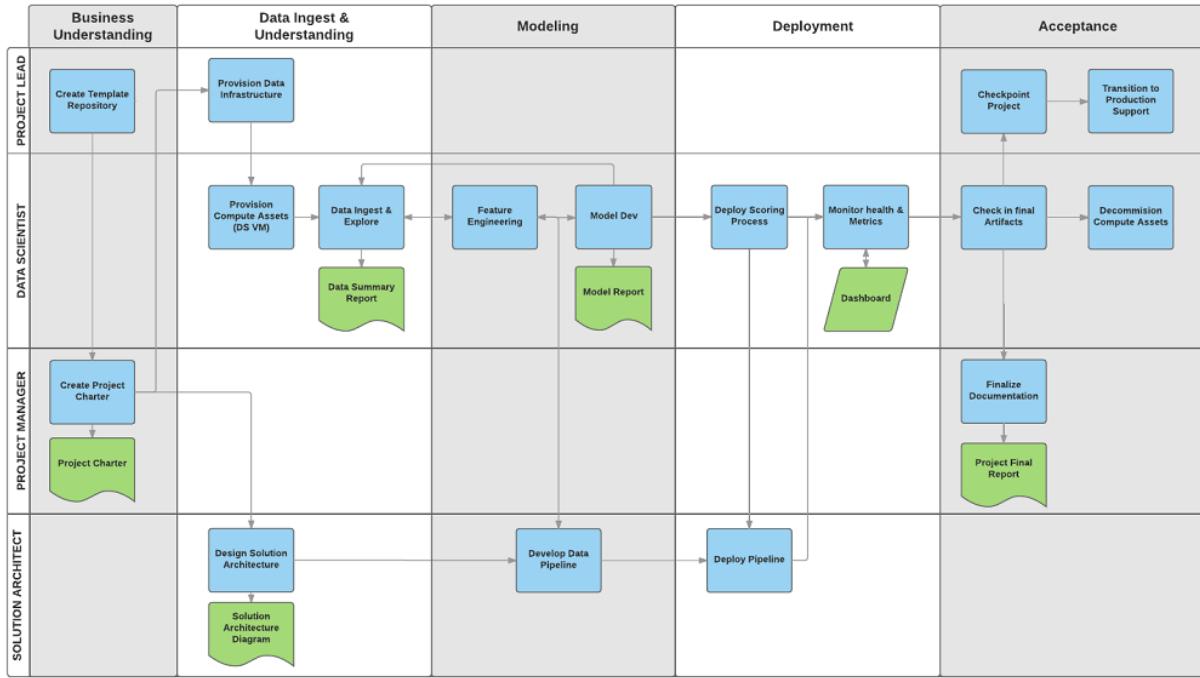
Data Science Lifecycle



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 engineer
- Data scientist
- Application developer
- 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).

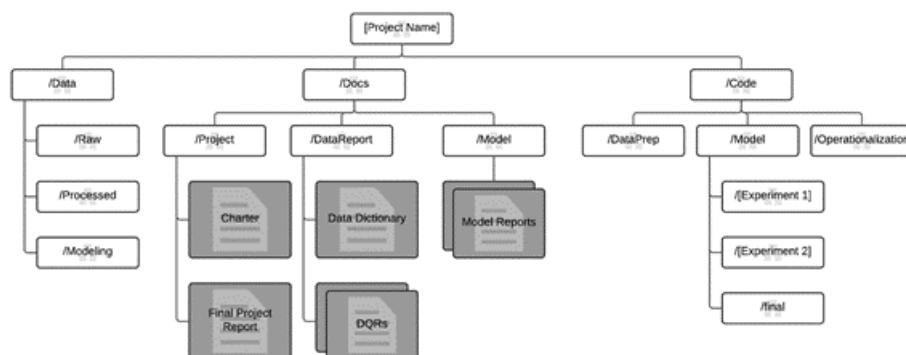


Standardized project structure

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, and Azure DevOps 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](#).

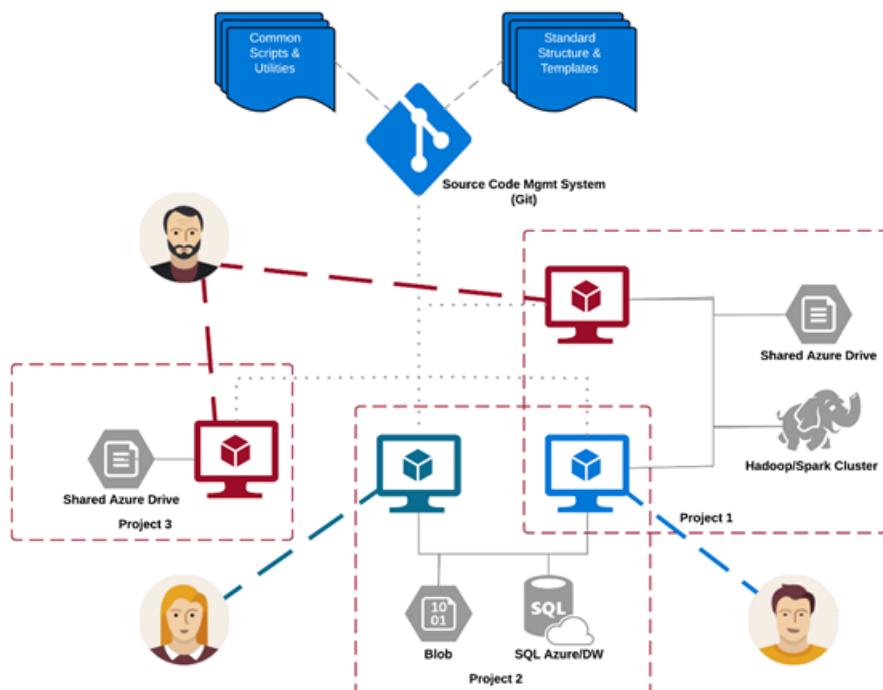
Infrastructure and resources for data science projects

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

- cloud file systems for storing datasets
- databases
- big data (SQL or Spark) clusters
- machine learning service

The analytics and storage infrastructure, where raw and processed datasets are stored, may be in the cloud or on-premises. This infrastructure enables reproducible analysis. It also avoids duplication, which may 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 to 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.

Microsoft provides extensive tooling inside [Azure Machine Learning](#) supporting both open-source (Python, R, ONNX, and common deep-learning frameworks) and also Microsoft's own tooling (AutoML).

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

3/5/2021 • 2 minutes 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 complete steps that successful projects follow. 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.

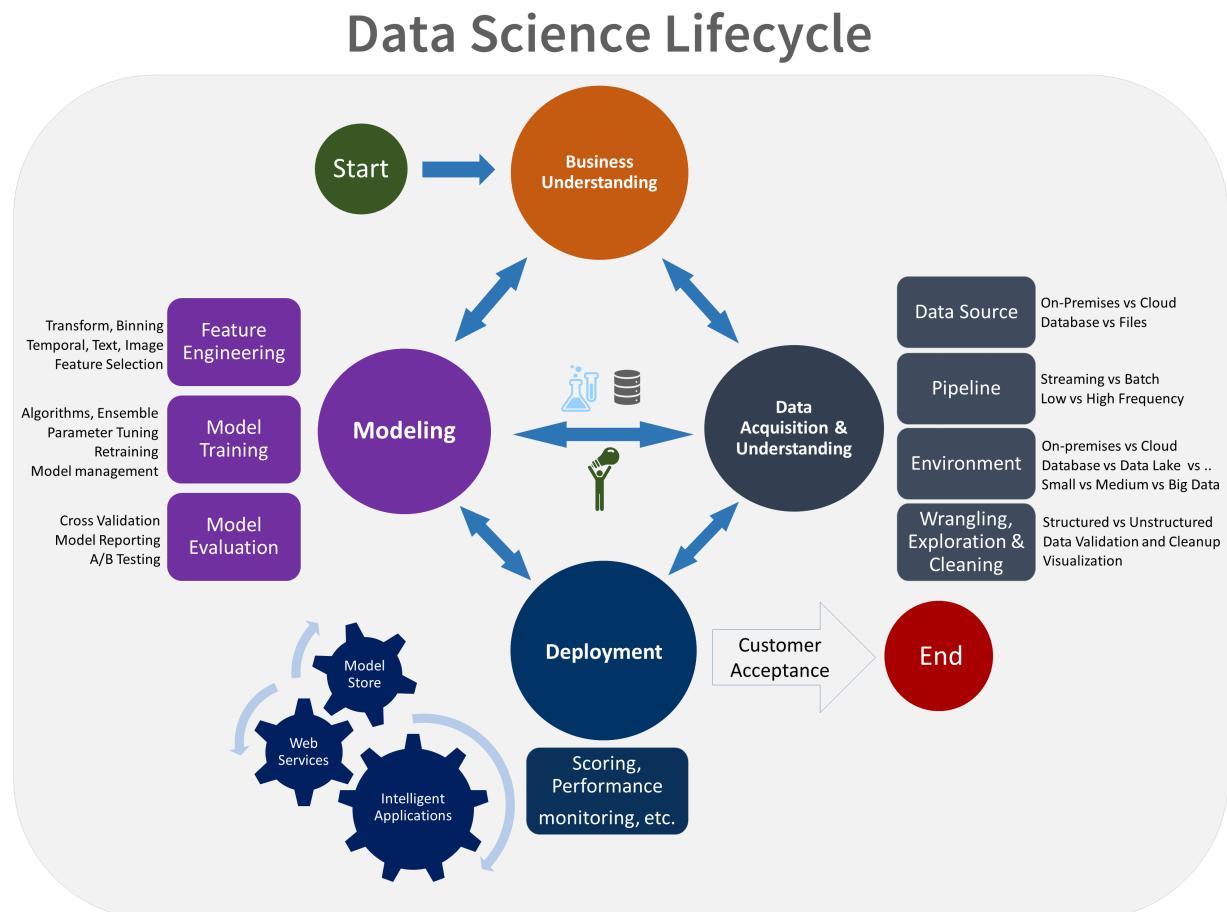
This lifecycle is designed for data-science projects that are intended to ship as part of intelligent applications. These applications deploy machine learning or artificial intelligence models for predictive analytics. Exploratory data-science projects and improvised analytics projects can also benefit from the use of this process. But for those projects, some of the steps described here might not be needed.

Five lifecycle stages

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](#).

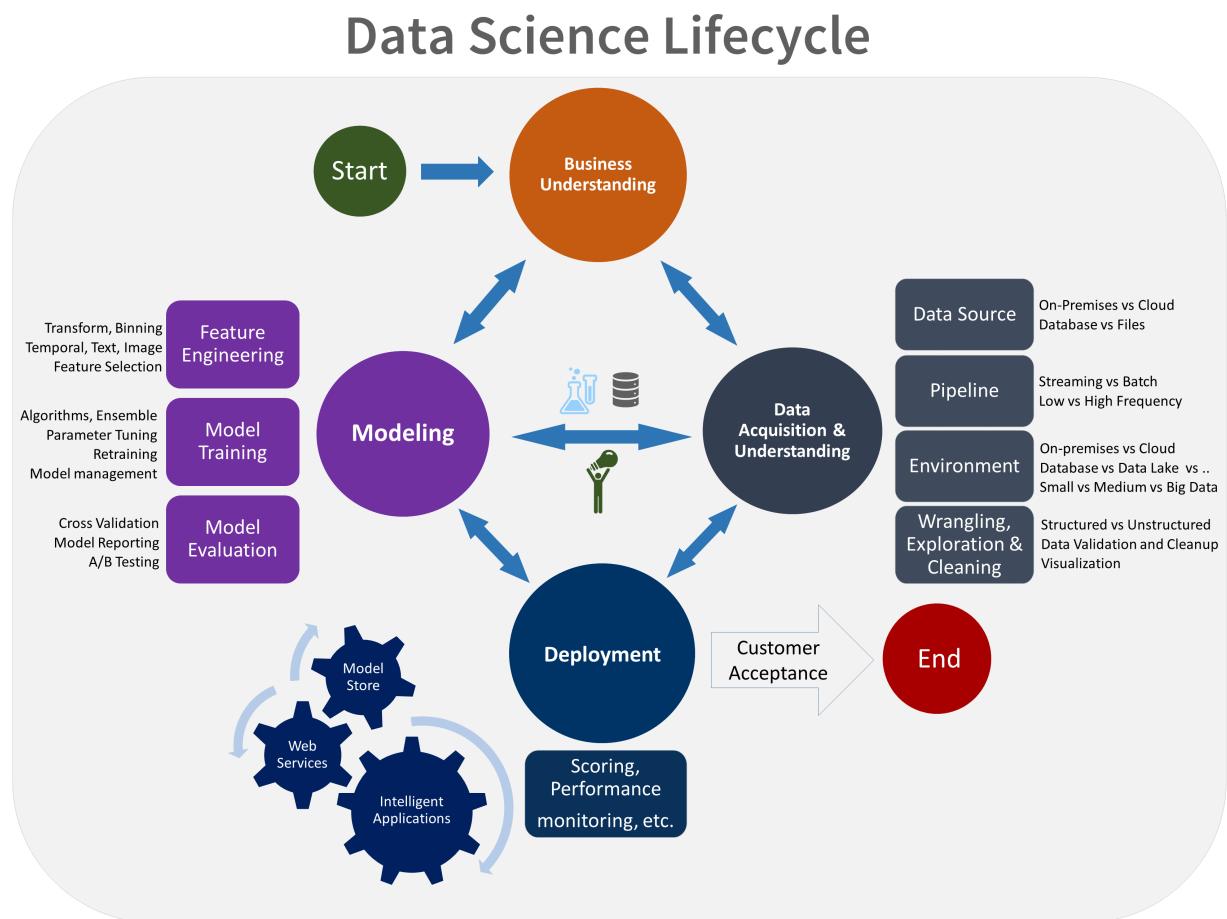
The business understanding stage of the Team Data Science Process lifecycle

1/23/2020 • 3 minutes 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. 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 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.

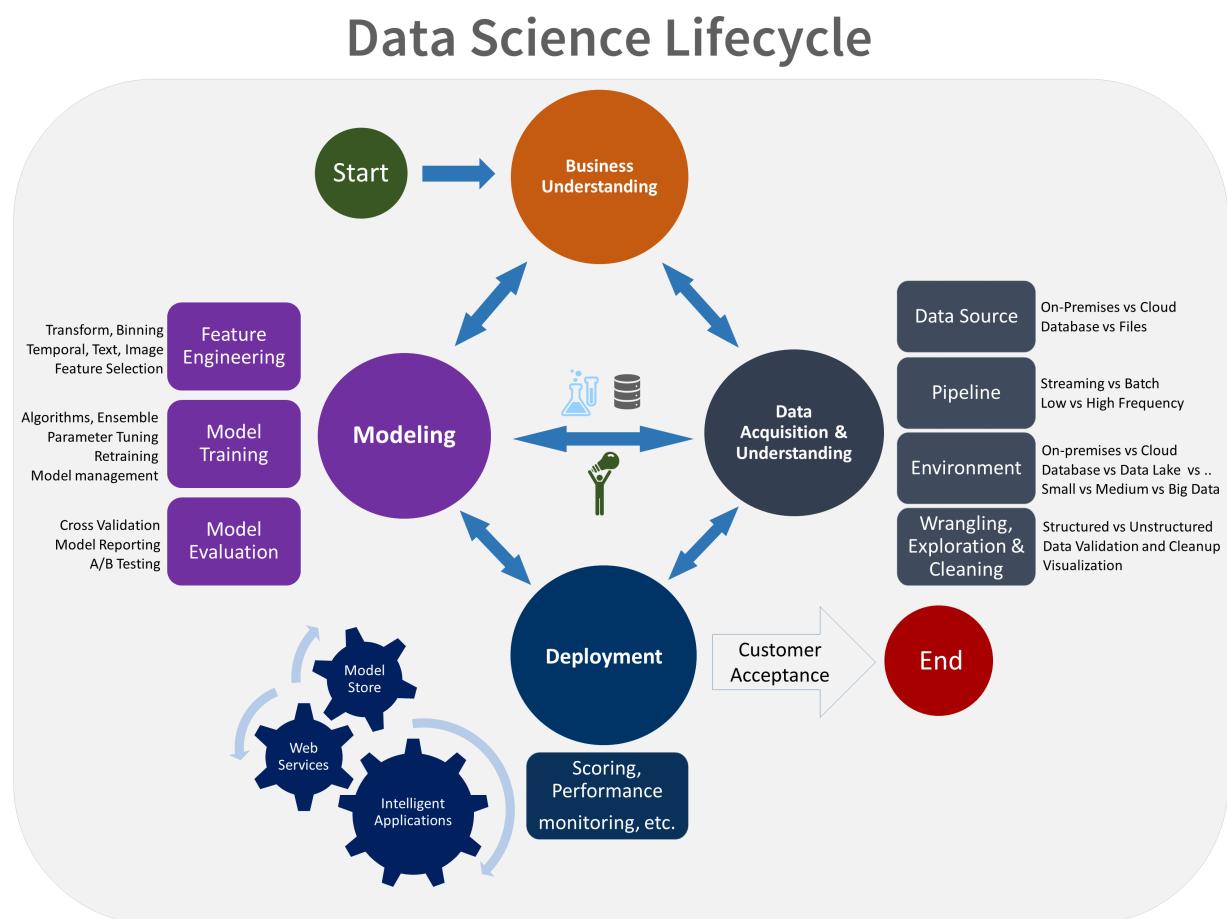
Data acquisition and understanding stage of the Team Data Science Process

3/5/2021 • 3 minutes 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. 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 data analysis 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. Scoring may be completed with a data pipeline or workflow. The [Move data from a 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 options:

- 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.
- **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 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 .

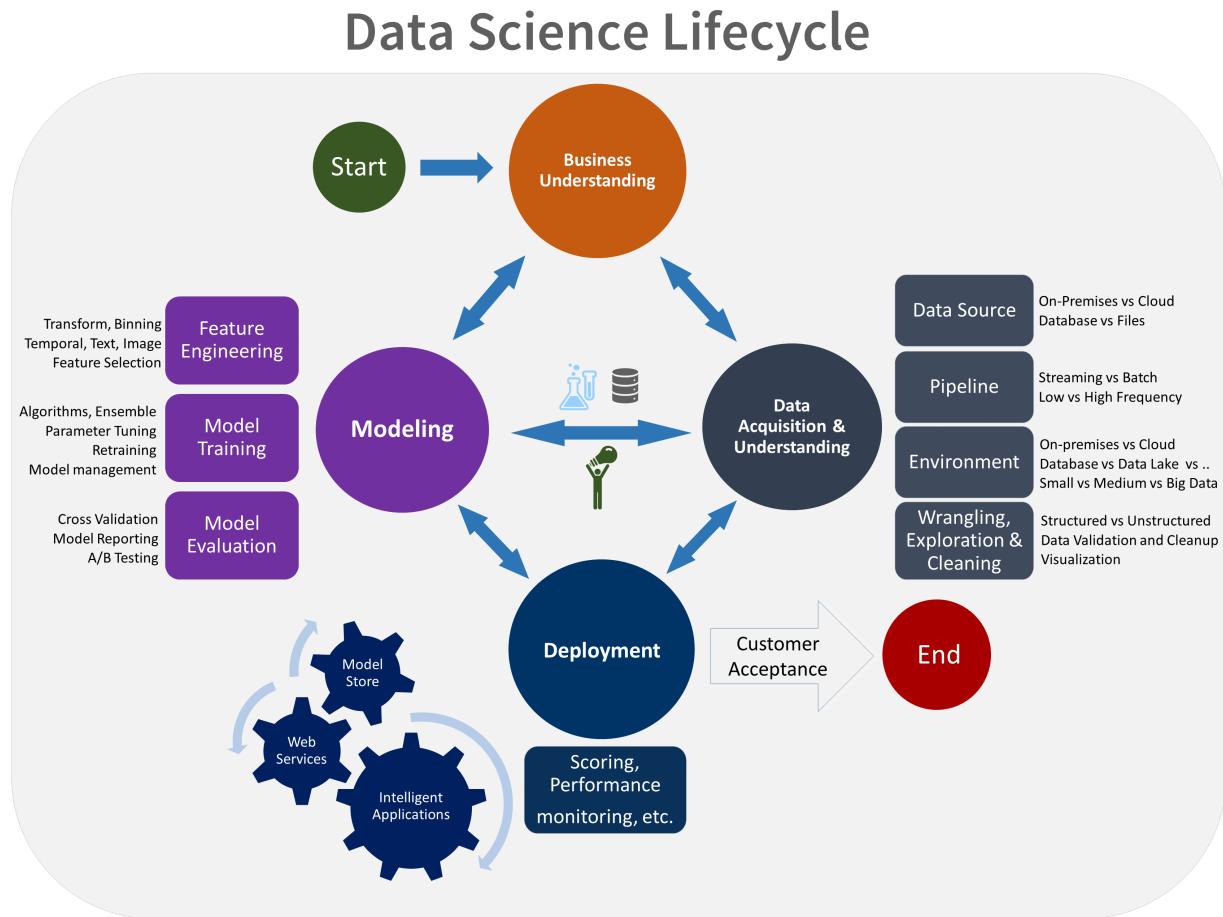
Modeling stage of the Team Data Science Process lifecycle

3/5/2021 • 3 minutes 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.

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 sufficiently for production. 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](#).

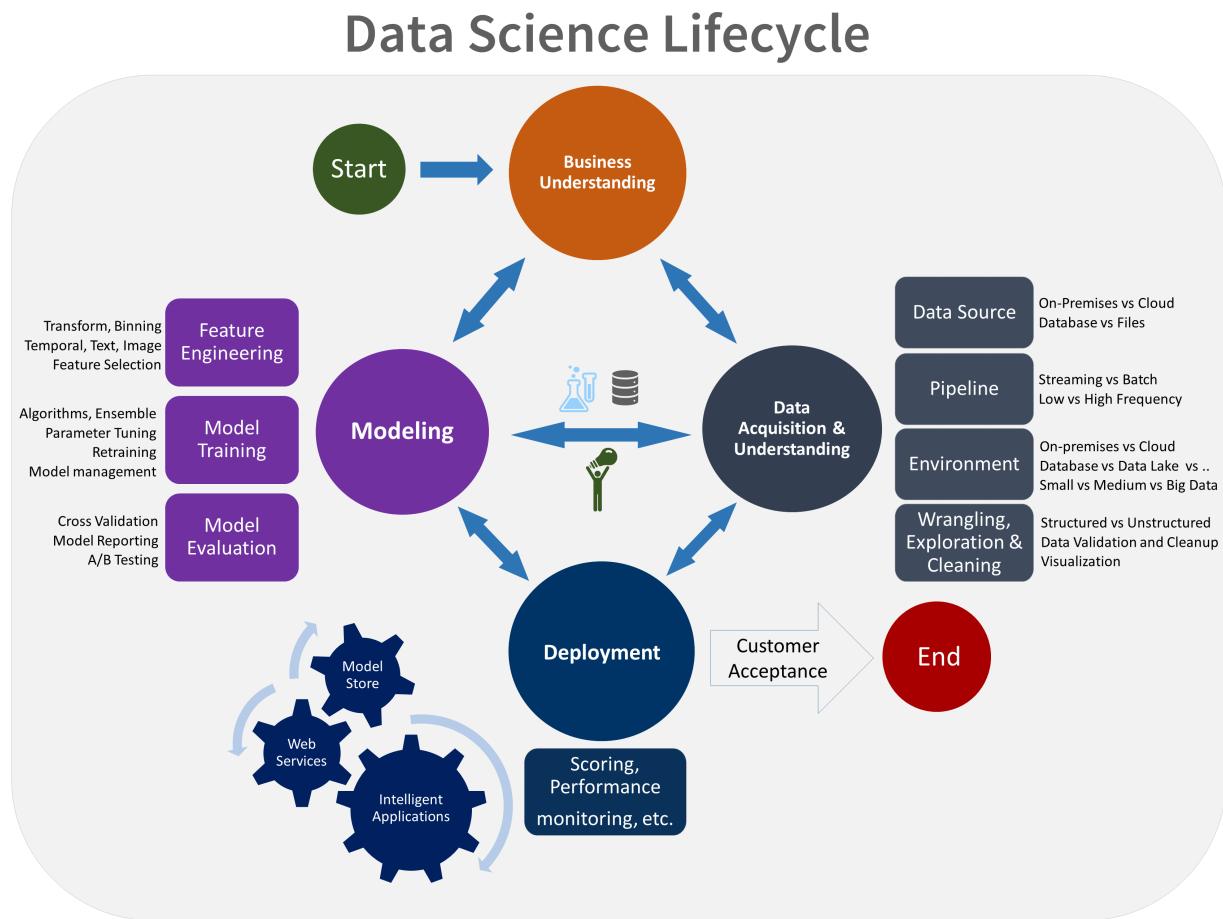
Deployment stage of the Team Data Science Process lifecycle

3/5/2021 • 2 minutes 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 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](#).

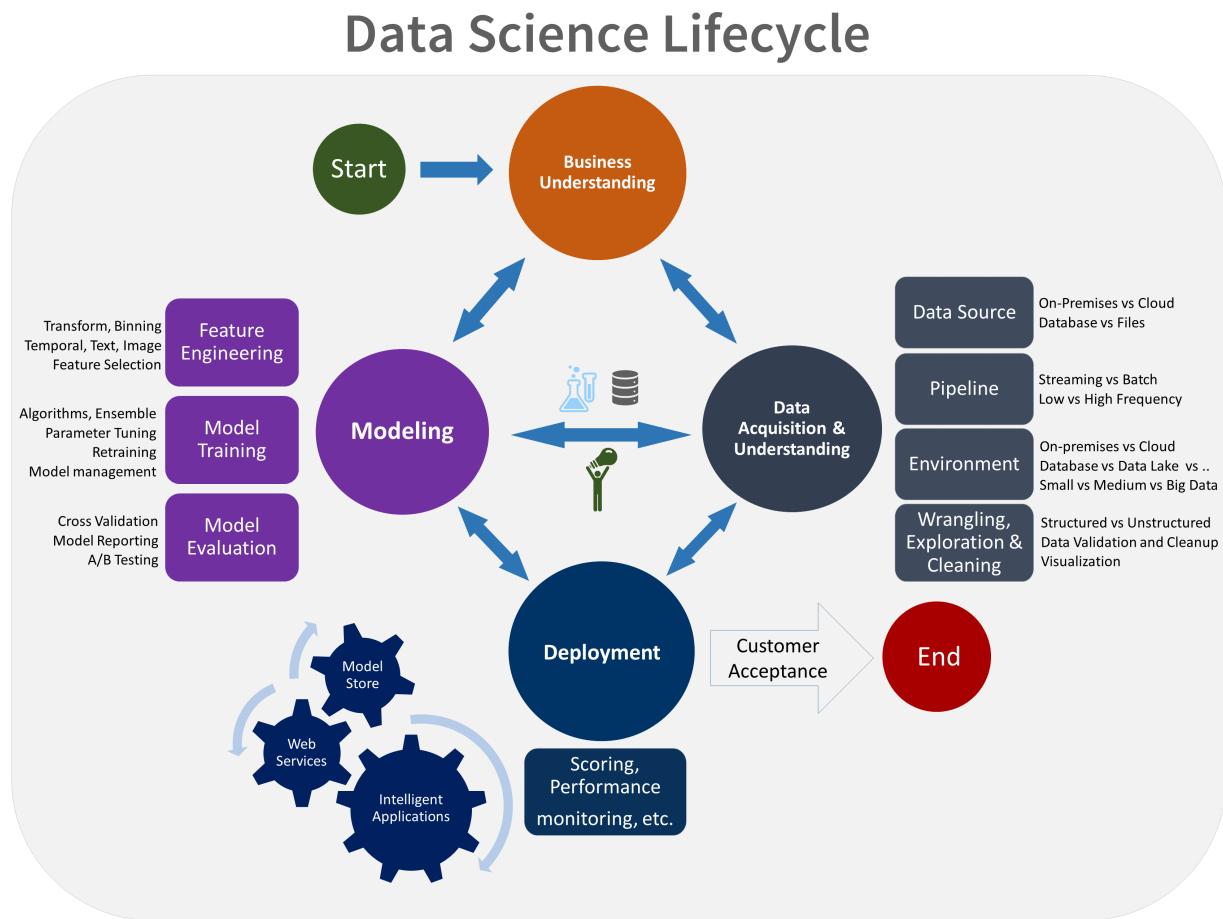
Customer acceptance stage of the Team Data Science Process lifecycle

3/5/2021 • 2 minutes 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:



Goal

Finalize 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 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

11/2/2020 • 6 minutes to read • [Edit Online](#)

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

This introductory article links to tutorials on how to set up the TDSP environment. The tutorials provide detailed guidance for using Azure DevOps Projects, Azure Repos repositories, and Azure Boards. The motivating goal is moving from concept through modeling and into deployment.

The tutorials use Azure DevOps because that is how to implement TDSP at Microsoft. Azure DevOps facilitates collaboration by integrating role-based security, work item management and tracking, and code hosting, sharing, and source control. The tutorials also use an Azure [Data Science Virtual Machine](#) (DSVM) as the analytics desktop, which has several popular data science tools pre-configured and integrated with Microsoft software and Azure services.

You can use the tutorials to implement TDSP using other code-hosting, agile planning, and development tools and environments, but some features may not be available.

Structure of data science groups and teams

Data science functions in enterprises are often organized in the following hierarchy:

- Data science group
 - Data science team/s within the group

In such a structure, there are group leads and team leads. Typically, a data science project is done by a data science team. Data science teams have project leads for project management and governance tasks, and individual data scientists and engineers to perform the data science and data engineering parts of the project. The initial project setup and governance is done by the group, team, or project leads.

Definition and tasks for the four TDSP roles

With the assumption that the data science unit consists of teams within a group, there are four distinct roles for TDSP personnel:

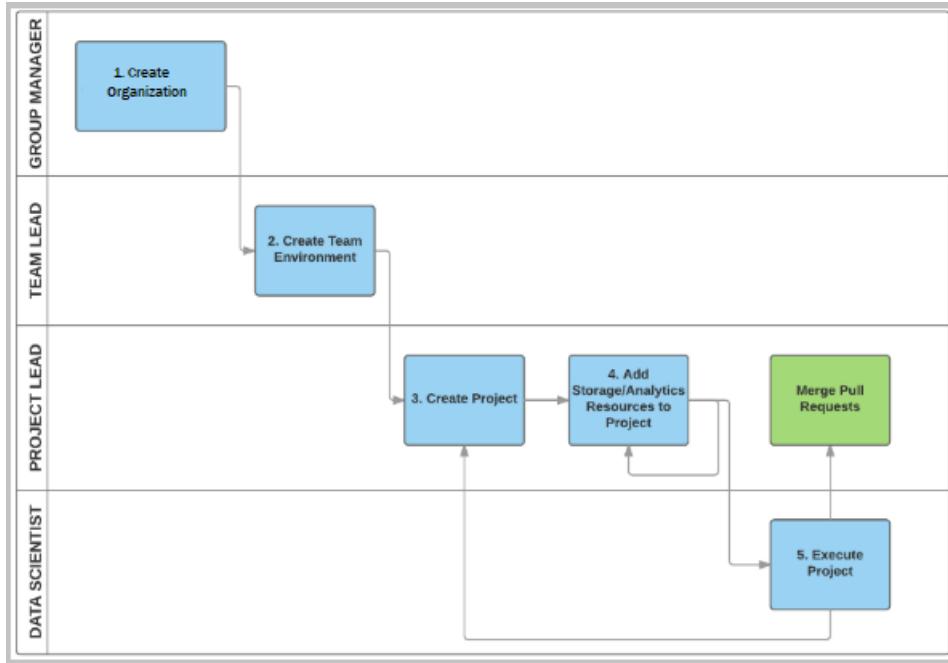
1. **Group Manager:** Manages the entire data science unit in an enterprise. A data science unit might have multiple teams, each of which is working on multiple data science projects in distinct business verticals. A Group Manager might delegate their tasks to a surrogate, but the tasks associated with the role do not change.
2. **Team Lead:** Manages a team in the data science unit of an enterprise. A team consists of multiple data scientists. For a small data science unit, the Group Manager and the Team Lead might be the same person.
3. **Project Lead:** Manages the daily activities of individual data scientists on a specific data science project.
4. **Project Individual Contributors:** Data Scientists, Business Analysts, Data Engineers, Architects, and others who execute a data science project.

NOTE

Depending on the structure and size of an enterprise, a single person may play more than one role, or more than one person may fill a role.

Tasks to be completed by the four roles

The following diagram shows the top-level tasks for each Team Data Science Process role. This schema and the following, more detailed outline of tasks for each TDSP role can help you choose the tutorial you need based on your responsibilities.



Group Manager tasks

The Group Manager or a designated TDSP system administrator completes the following tasks to adopt the TDSP:

- Creates an Azure DevOps **organization** and a group project within the organization.
- Creates a **project template repository** in the Azure DevOps group project, and seeds it from the project template repository developed by the Microsoft TDSP team. The Microsoft TDSP project template repository provides:
 - A **standardized directory structure**, including directories for data, code, and documents.
 - A set of **standardized document templates** to guide an efficient data science process.
- Creates a **utility repository**, and seeds it from the utility repository developed by the 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. The Microsoft utility repository includes utilities for interactive data exploration, analysis, reporting, and baseline modeling and reporting.
- Sets up the **security control policy** for the organization account.

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

Team Lead tasks

The Team Lead or a designated project administrator completes the following tasks to adopt the TDSP:

- Creates a team **project** in the group's Azure DevOps organization.
- Creates the **project template repository** in the project, and seeds it from the group project template

repository set up by the Group Manager or delegate.

- Creates the **team utility repository**, seeds it from the group utility repository, and adds team-specific utilities to the repository.
- Optionally creates **Azure file storage** to store useful data assets for the team. Other team members can mount this shared cloud file store on their analytics desktops.
- Optionally mounts the Azure file storage on the team's **DSVM** and adds team data assets to it.
- Sets up **security control** by adding team members and configuring their permissions.

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

Project Lead tasks

The Project Lead completes the following tasks to adopt the TDSP:

- Creates a **project repository** in the team project, and seeds it from the project template repository.
- Optionally creates **Azure file storage** to store the project's data assets.
- Optionally mounts the Azure file storage to the **DSVM** and adds project data assets to it.
- Sets up **security control** by adding project members and configuring their permissions.

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

Project Individual Contributor tasks

The Project Individual Contributor, usually a Data Scientist, conducts the following tasks using the TDSP:

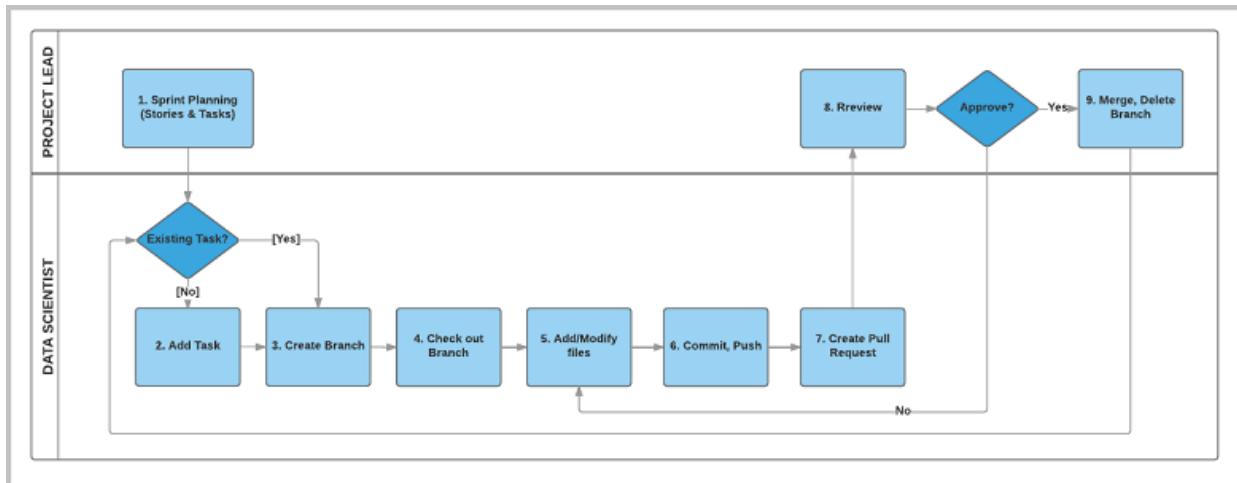
- Clones the **project repository** set up by the project lead.
- Optionally mounts the shared team and project **Azure file storage** on their **Data Science Virtual Machine (DSVM)**.
- Executes the project.

For detailed instructions for onboarding onto a project, see [Project Individual Contributor tasks for a data science team](#).

Data science project execution workflow

By following the relevant tutorials, data scientists, project leads, and team leads can create work items to track all tasks and stages for project from beginning to end. Using Azure Repos promotes collaboration among data scientists and ensures that the artifacts generated during project execution are version controlled and shared by all project members. Azure DevOps lets you link your Azure Boards work items with your Azure Repos repository branches and easily track what has been done for a work item.

The following figure outlines the TDSP workflow for project execution:



The workflow steps can be grouped into three activities:

- Project Leads conduct sprint planning
- Data Scientists develop artifacts on `git` branches to address work items
- Project Leads or other team members do code reviews and merge working branches to the primary branch

For detailed instructions on project execution workflow, see [Agile development of data science projects](#).

TDSP project template repository

Use the Microsoft TDSP team's [project template repository](#) to support efficient project execution and collaboration. The repository gives you a standardized directory structure and document templates you can use for your own TDSP projects.

Next steps

Explore more detailed descriptions of the roles and tasks defined by the Team Data Science Process:

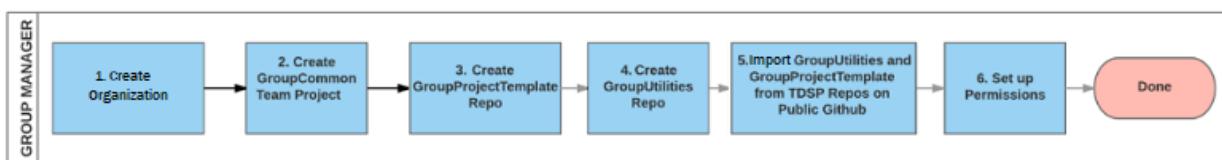
- [Group Manager tasks for a data science team](#)
- [Team Lead tasks for a data science team](#)
- [Project Lead tasks for a data science team](#)
- [Project Individual Contributor tasks for a data science team](#)

Team Data Science Process group manager tasks

3/5/2021 • 7 minutes to read • [Edit Online](#)

This article describes the tasks that a *group manager* completes for a data science organization. The group manager manages the entire data science unit in an enterprise. A data science unit may have several teams, each of which is working on many data science projects in distinct business verticals. The group manager's objective is to establish a collaborative group environment that standardizes on the [Team Data Science Process](#) (TDSP). For an outline of all the personnel roles and associated tasks handled by a data science team standardizing on the TDSP, see [Team Data Science Process roles and tasks](#).

The following diagram shows the six main group manager setup tasks. Group managers may delegate their tasks to surrogates, but the tasks associated with the role don't change.



1. Set up an [Azure DevOps organization](#) for the group.
2. Create the default **GroupCommon** project in the Azure DevOps organization.
3. Create the **GroupProjectTemplate** repository in Azure Repos.
4. Create the **GroupUtilities** repository in Azure Repos.
5. Import the contents of the Microsoft TDSP team's **ProjectTemplate** and **Utilities** repositories into the group common repositories.
6. Set up **membership** and **permissions** for team members to access the group.

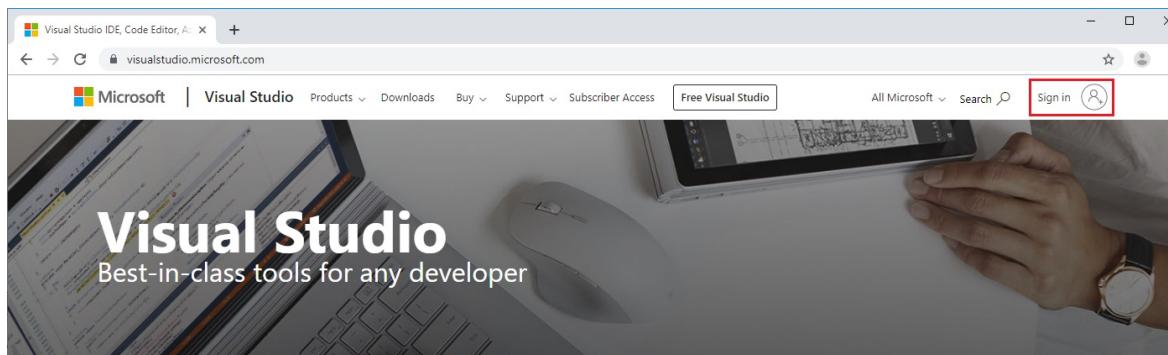
The following tutorial walks through the steps in detail.

NOTE

This article uses Azure DevOps to set up a TDSP group environment, because that is how to implement TDSP at Microsoft. If your group uses other code hosting or development platforms, the Group Manager's tasks are the same, but the way to complete them may be different.

Create an organization and project in Azure DevOps

1. Go to visualstudio.microsoft.com, select **Sign in** at upper right, and sign into your Microsoft account.



If you don't have a Microsoft account, select **Sign up now**, create a Microsoft account, and sign in using this account. If your organization has a Visual Studio subscription, sign in with the credentials for that

subscription.

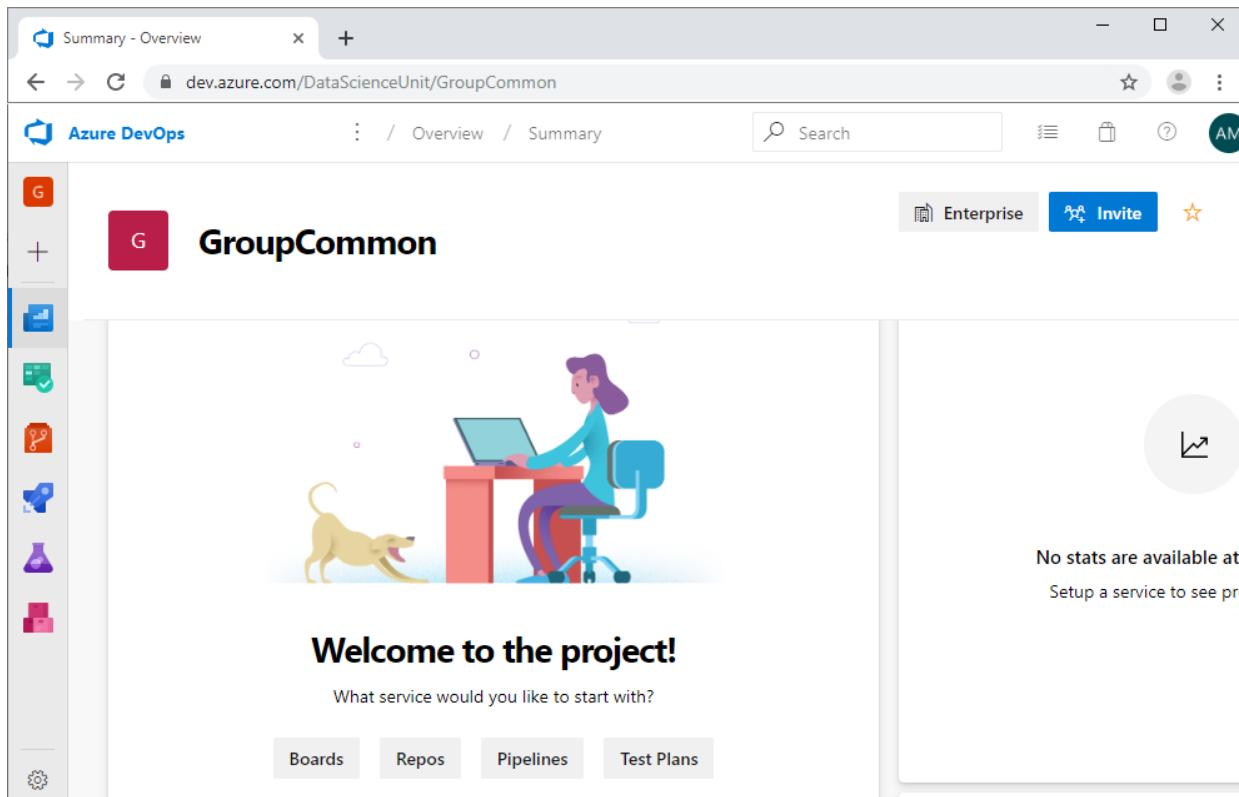
2. After you sign in, at upper right on the Azure DevOps page, select **Create new organization**.

The screenshot shows the Azure DevOps Organizations page. On the left, there's a profile card for 'A Group Manager' with a large circular icon containing 'AM'. Below the icon are fields for email (gm@fabrikam.com) and location (United States). On the right, under 'Azure DevOps Organizations', there's a list of existing organizations: 'dev.azure.com/fabrikam-org' (Owner), which contains projects like 'TDSP Customer Project', 'FabrikamFiber', and 'microprofile1'. A red box highlights the 'Create new organization' button at the top right of the main content area.

3. If you're prompted to agree to the Terms of Service, Privacy Statement, and Code of Conduct, select **Continue**.
4. In the signup dialog, name your Azure DevOps organization and accept the host region assignment, or drop down and select a different region. Then select **Continue**.
5. Under **Create a project to get started**, enter *GroupCommon*, and then select **Create project**.

The screenshot shows the 'Create a project to get started' dialog. On the left, there's a sidebar with organization names: 'DataScienceUnit' (selected), 'mseng', 'ceapex', and 'fabrikam-org'. Below the sidebar are links for '8 more organizations' and 'New organization'. The main area has a title 'Create a project to get started'. It includes a 'Project name *' field with 'GroupCommon' entered, a 'Visibility' section with three options: 'Public', 'Enterprise' (selected, highlighted with a red box), and 'Private', and a large blue 'Create project' button at the bottom.

The **GroupCommon** project **Summary** page opens. The page URL is <https://<servername>/<organization-name>/GroupCommon>.



Set up the group common repositories

Azure Repos hosts the following types of repositories for your group:

- **Group common repositories:** General-purpose repositories that multiple teams within a data science unit can adopt for many data science projects.
- **Team repositories:** Repositories for specific teams within a data science unit. These repositories are specific for a team's needs, and may be used for multiple projects within that team, but are not general enough to be used across multiple teams within a data science unit.
- **Project repositories:** Repositories for specific projects. Such repositories may not be general enough for multiple projects within a team, or for other teams in a data science unit.

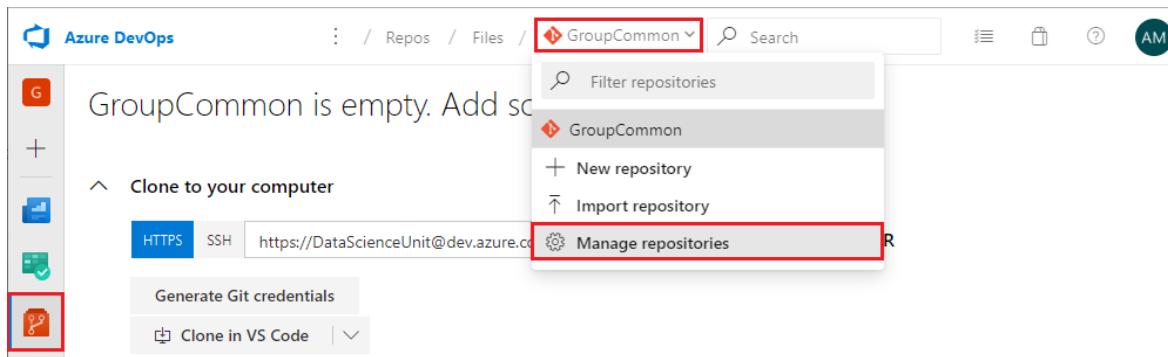
To set up the group common repositories in your project, you:

- Rename the default **GroupCommon** repository to **GroupProjectTemplate**
- Create a new **GroupUtilities** repository

Rename the default project repository to **GroupProjectTemplate**

To rename the default **GroupCommon** project repository to **GroupProjectTemplate**:

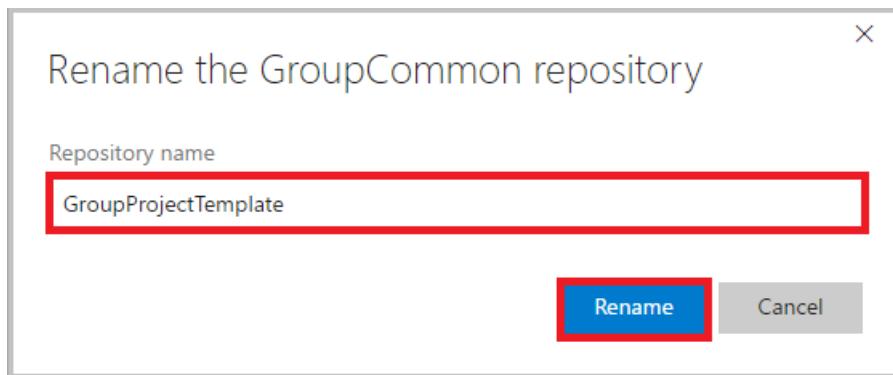
1. On the **GroupCommon** project **Summary** page, select **Repos**. This action takes you to the default **GroupCommon** repository of the **GroupCommon** project, which is currently empty.
2. At the top of the page, drop down the arrow next to **GroupCommon** and select **Manage repositories**.



3. On the **Project Settings** page, select the ... next to **GroupCommon**, and then select **Rename repository**.

The screenshot shows the 'Repositories' section of the 'GroupCommon' settings. The 'GroupCommon' repository is listed under 'Git repositories'. A context menu is open over 'GroupCommon', with the 'Rename repository...' option highlighted.

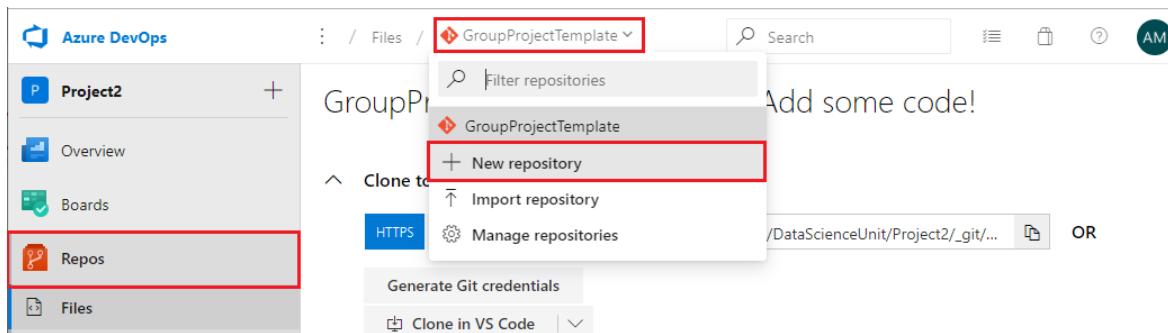
4. In the **Rename the GroupCommon repository** popup, enter *GroupProjectTemplate*, and then select **Rename**.



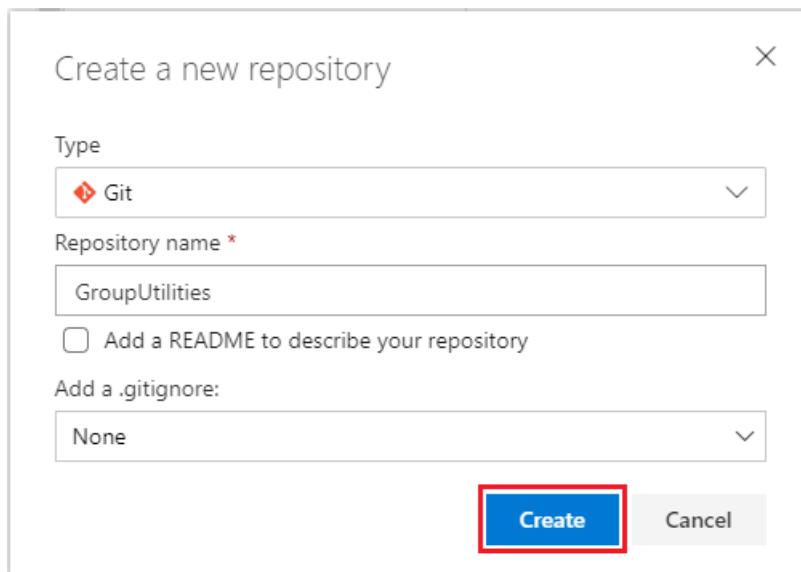
Create the GroupUtilities repository

To create the **GroupUtilities** repository:

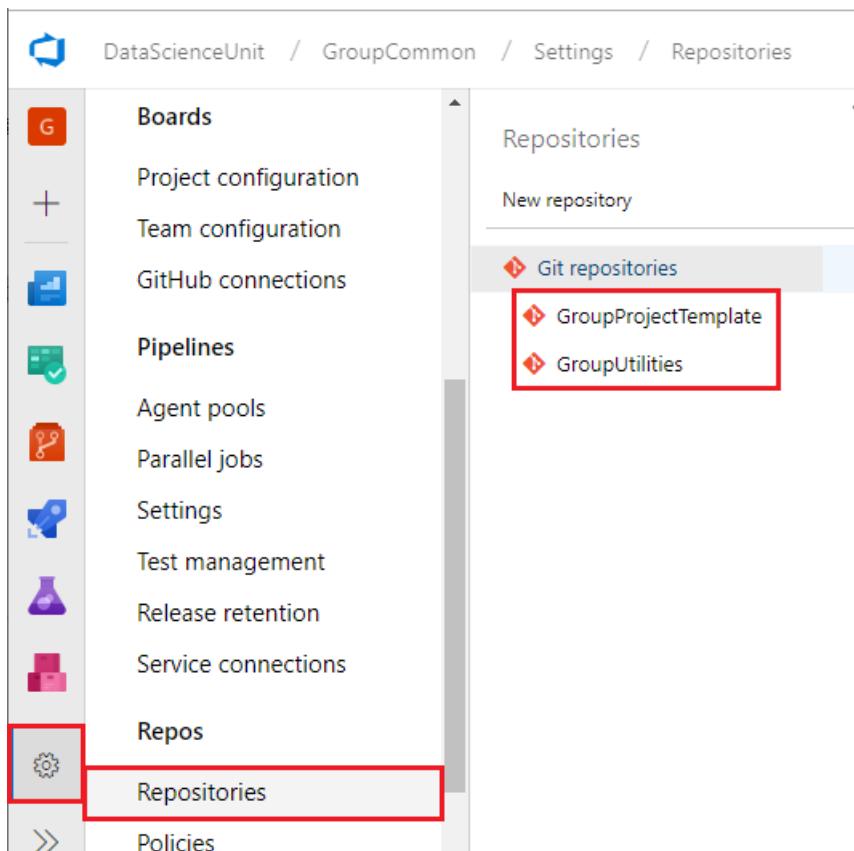
1. On the **GroupCommon** project **Summary** page, select **Repos**.
2. At the top of the page, drop down the arrow next to **GroupProjectTemplate** and select **New repository**.



3. In the **Create a new repository** dialog, select **Git** as the **Type**, enter *GroupUtilities* as the **Repository name**, and then select **Create**.



4. On the **Project Settings** page, select **Repositories** under **Repos** in the left navigation to see the two group repositories: **GroupProjectTemplate** and **GroupUtilities**.



Import the Microsoft TDSP team repositories

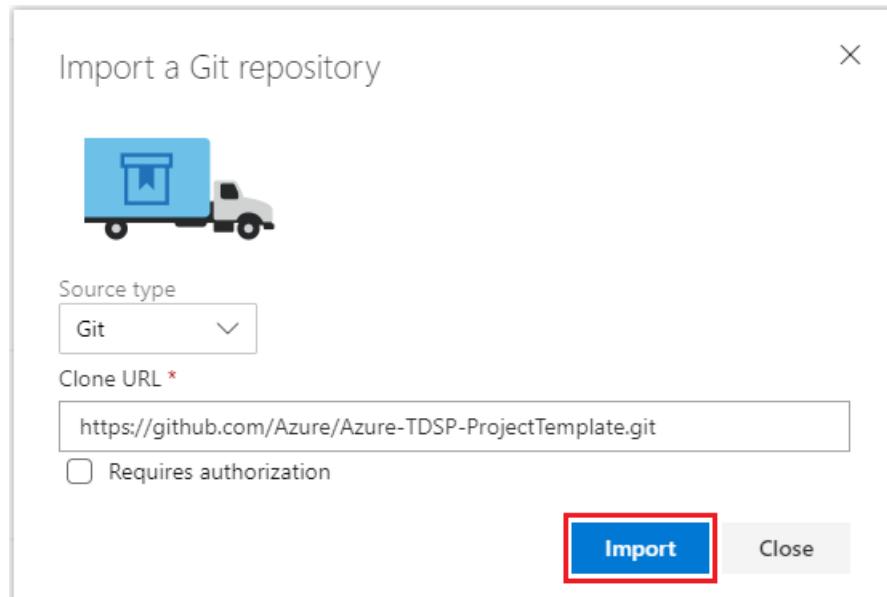
In this part of the tutorial, you import the contents of the **ProjectTemplate** and **Utilities** repositories managed by the Microsoft TDSP team into your **GroupProjectTemplate** and **GroupUtilities** repositories.

To import the TDSP team repositories:

1. From the **GroupCommon** project home page, select **Repos** in the left navigation. The default **GroupProjectTemplate** repo opens.
2. On the **GroupProjectTemplate** is empty page, select **Import**.

The screenshot shows the Azure DevOps interface for the 'GroupCommon' project. The left sidebar has 'Repos' selected, highlighted with a red box. The main content area displays the message 'GroupProjectTemplate is empty. Add some code!'. Below this, there are sections for cloning the repository via HTTPS or SSH, generating Git credentials, and cloning it into VS Code. Another section shows commands for pushing an existing repository from the command line. At the bottom, there's a link to import a repository, which is also highlighted with a red box.

3. In the **Import a Git repository** dialog, select **Git** as the **Source type**, and enter <https://github.com/Azure/Azure-TDSP-ProjectTemplate.git> for the **Clone URL**. Then select **Import**. The contents of the Microsoft TDSP team ProjectTemplate repository are imported into your GroupProjectTemplate repository.



4. At the top of the **Repos** page, drop down and select the **GroupUtilities** repository.

Each of your two group repositories now contains all the files, except those in the `.git` directory, from the Microsoft TDSP team's corresponding repository.

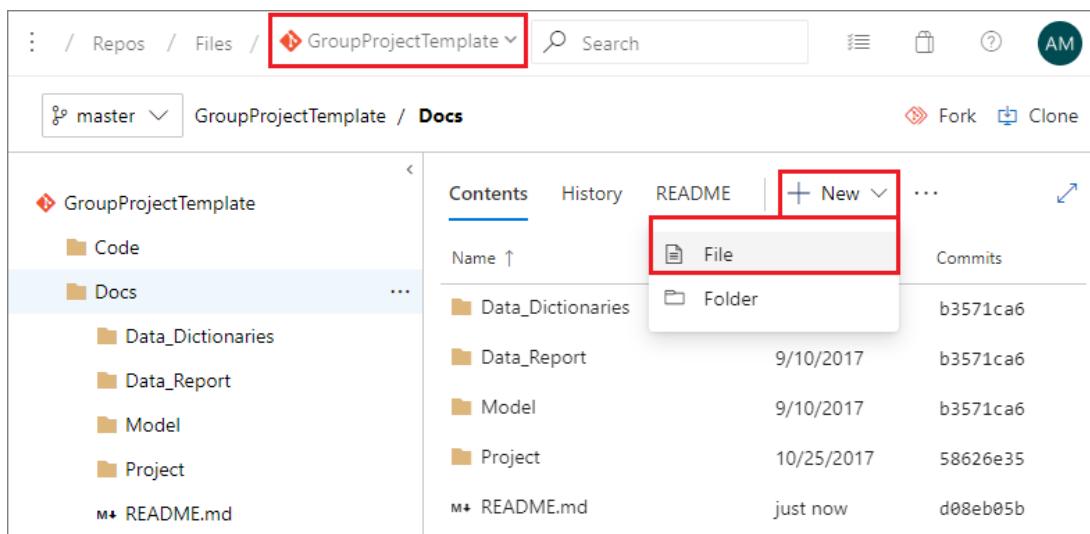
Customize the contents of the group repositories

If you want to customize the contents of your group repositories to meet the specific needs of your group, you can do that now. You can modify the files, change the directory structure, or add files that your group has developed or that are helpful for your group.

Make changes in Azure Repos

To customize repository contents:

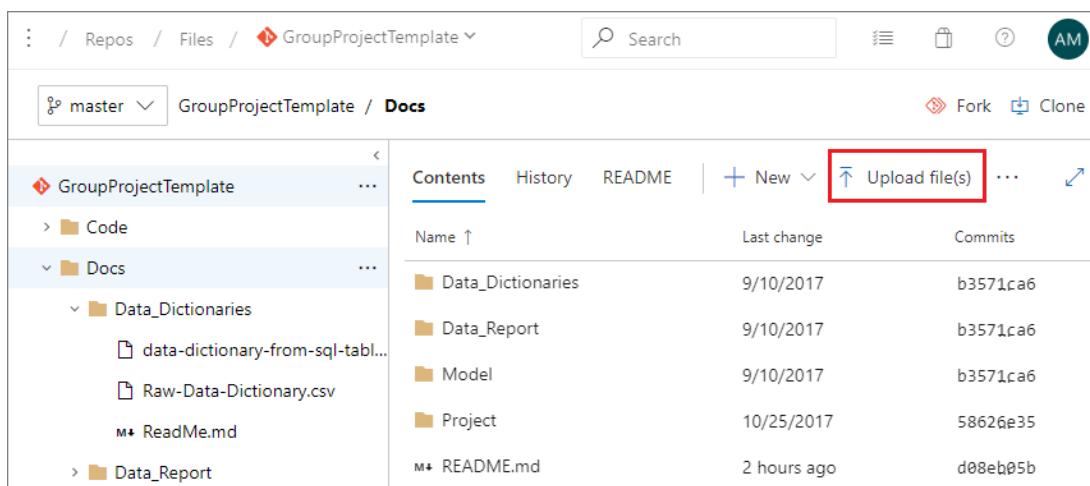
1. On the **GroupCommon** project **Summary** page, select **Repos**.
2. At the top of the page, select the repository you want to customize.
3. In the repo directory structure, navigate to the folder or file you want to change.
 - To create new folders or files, select the arrow next to **New**.



The screenshot shows the Azure Repos interface for the 'GroupProjectTemplate' repository. The left sidebar shows the directory structure: 'Code', 'Docs' (which is expanded to show 'Data_Dictionaries', 'Data_Report', 'Model', 'Project', and 'README.md'), and 'README'. The main area displays a table of files and folders under the 'Docs' folder. A red box highlights the '+ New' dropdown menu, which is open to show options for 'File' and 'Folder'. The table includes columns for Name, Last change, and Commits.

Name	Last change	Commits
Data_Dictionaries	9/10/2017	b3571ca6
Data_Report	9/10/2017	b3571ca6
Model	9/10/2017	b3571ca6
Project	10/25/2017	58626e35
README.md	just now	d08eb05b

- To upload files, select **Upload file(s)**.



The screenshot shows the Azure Repos interface for the 'GroupProjectTemplate' repository. The left sidebar shows the directory structure: 'Code', 'Docs' (which is expanded to show 'Data_Dictionaries' containing 'data-dictionary-from-sql-table...', 'Raw-Data-Dictionary.csv', and 'ReadMe.md', and 'Data_Report'). The main area displays a table of files and folders under the 'Docs' folder. A red box highlights the '**Upload file(s)**' button. The table includes columns for Name, Last change, and Commits.

Name	Last change	Commits
Data_Dictionaries	9/10/2017	b3571ca6
Data_Report	9/10/2017	b3571ca6
Model	9/10/2017	b3571ca6
Project	10/25/2017	58626e35
README.md	2 hours ago	d08eb05b

- To edit existing files, navigate to the file and then select **Edit**.

The screenshot shows a GitHub repository named 'GroupProjectTemplate'. The 'master' branch is selected. In the 'Docs' folder, the 'README.md' file is open. The 'Edit' button at the top right of the preview area is highlighted with a red box. The content of the README.md file describes a folder for hosting documents related to Data Science Unit, listing items 1 through 6.

```
Folder for hosting all documents for the Data Science Unit
Documents will contain information about the following
1. System architecture
2. Data dictionaries
3. Reports related to data understanding, modeling
4. Project management and planning docs
5. Information obtained from a business owner or client about the project
6. Docs and presentations prepared to share information about the project
```

4. After adding or editing files, select **Commit**.

The screenshot shows the 'Commit' dialog box. It includes fields for 'Comment' (containing 'Updated README.md'), 'Branch name' (set to 'master'), and 'Work items to link' (a search bar). At the bottom, there are 'Commit' and 'Cancel' buttons, with 'Commit' highlighted with a red box.

Make changes using your local machine or DSVM

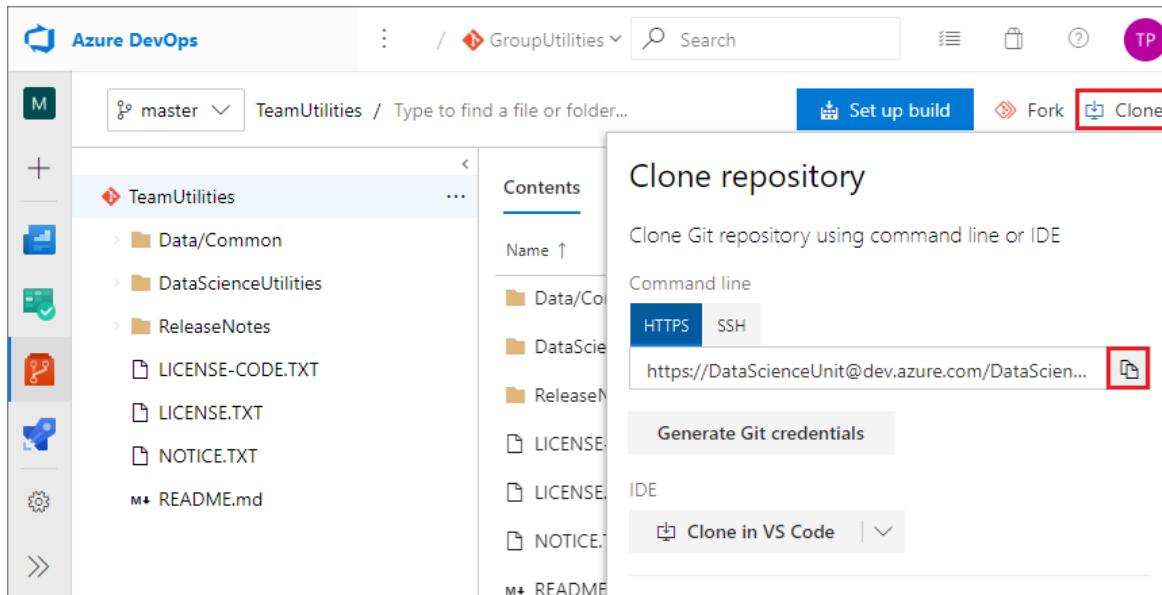
If you want to make changes using your local machine or DSVM and push the changes up to the group repositories, make sure you have the prerequisites for working with Git and DSVMs:

- An Azure subscription, if you want to create a DSVM.
- Git installed on your machine. If you're using a DSVM, Git is pre-installed. Otherwise, see the [Platforms and tools appendix](#).
- If you want to use a DSVM, the Windows or Linux DSVM created and configured in Azure. For more information and instructions, see the [Data Science Virtual Machine Documentation](#).
- For a Windows DSVM, [Git Credential Manager \(GCM\)](#) installed on your machine. In the *README.md* file, scroll down to the **Download and Install** section and select the **latest installer**. Download the .exe installer from the installer page and run it.

- For a Linux DSVM, an SSH public key set up on your DSVM and added in Azure DevOps. For more information and instructions, see the [Create SSH public key](#) section in the [Platforms and tools appendix](#).

First, copy or *clone* the repository to your local machine.

- On the **GroupCommon** project **Summary** page, select **Repos**, and at the top of the page, select the repository you want to clone.
- On the repo page, select **Clone** at upper right.
- In the **Clone repository** dialog, select **HTTPS** for an HTTP connection, or **SSH** for an SSH connection, and copy the clone URL under **Command line** to your clipboard.



- On your local machine, create the following directories:

- For Windows: `C:\GitRepos\GroupCommon`
- For Linux, `~/GitRepos/GroupCommon` on your home directory

- Change to the directory you created.

- In Git Bash, run the command `git clone <clone URL>`.

For example, either of the following commands clones the **GroupUtilities** repository to the **GroupCommon** directory on your local machine.

HTTPS connection:

```
git clone https://DataScienceUnit@dev.azure.com/DataScienceUnit/_git/GroupUtilities
```

SSH connection:

```
git clone git@ssh.dev.azure.com:v3/DataScienceUnit/_git/GroupUtilities
```

After making whatever changes you want in the local clone of your repository, you can push the changes to the shared group common repositories.

Run the following Git Bash commands from your local **GroupProjectTemplate** or **GroupUtilities** directory.

```
git add .
git commit -m "push from local"
git push
```

NOTE

If this is the first time you commit to a Git repository, you may 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're committing to several Git repositories, use the same name and email address for all of them. Using the same name and email address is convenient when building Power BI dashboards to track your Git activities in multiple repositories.

Add group members and configure permissions

To add members to the group:

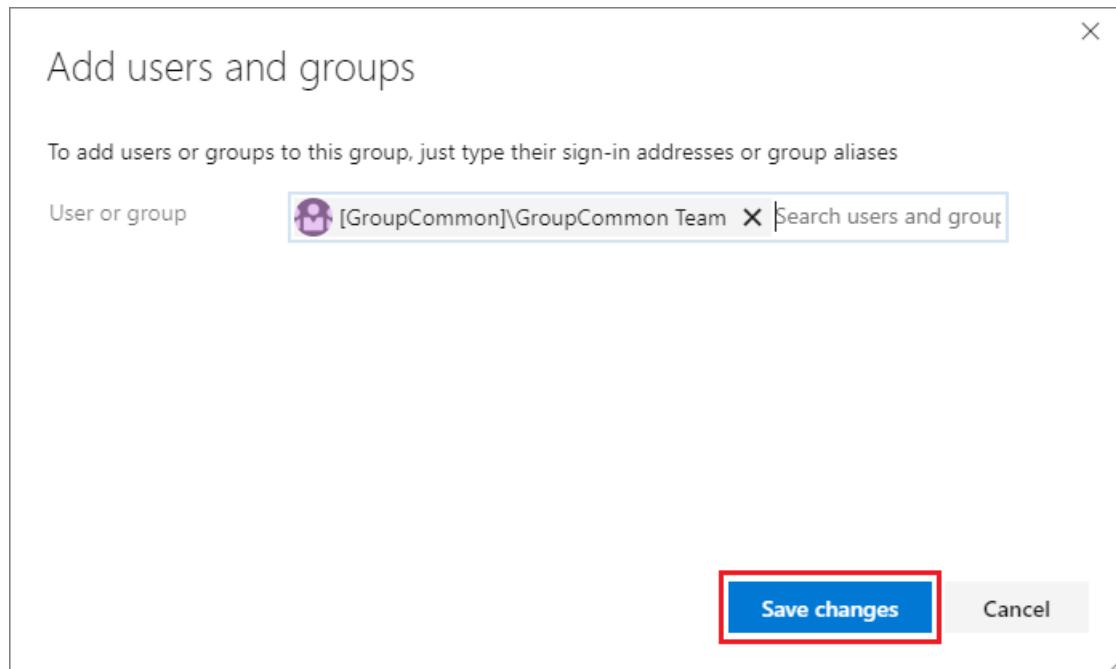
1. In Azure DevOps, from the **GroupCommon** project home page, select **Project settings** from the left navigation.
2. From the **Project Settings** left navigation, select **Teams**, then on the **Teams** page, select the **GroupCommon Team**.

The screenshot shows the Azure DevOps Project Settings - Teams page. The left sidebar has a 'Groups' icon highlighted with a red box. Under 'Project Settings', 'Teams' is selected, also highlighted with a red box. The main pane displays the 'Teams' section with a table showing one team: 'GroupCommon Team'. The 'GroupCommon Team' row is highlighted with a red box. The table columns are 'Team Name', 'Members', and 'Description'. The description for the team is 'The default project team.'

3. On the **Team Profile** page, select **Add**.

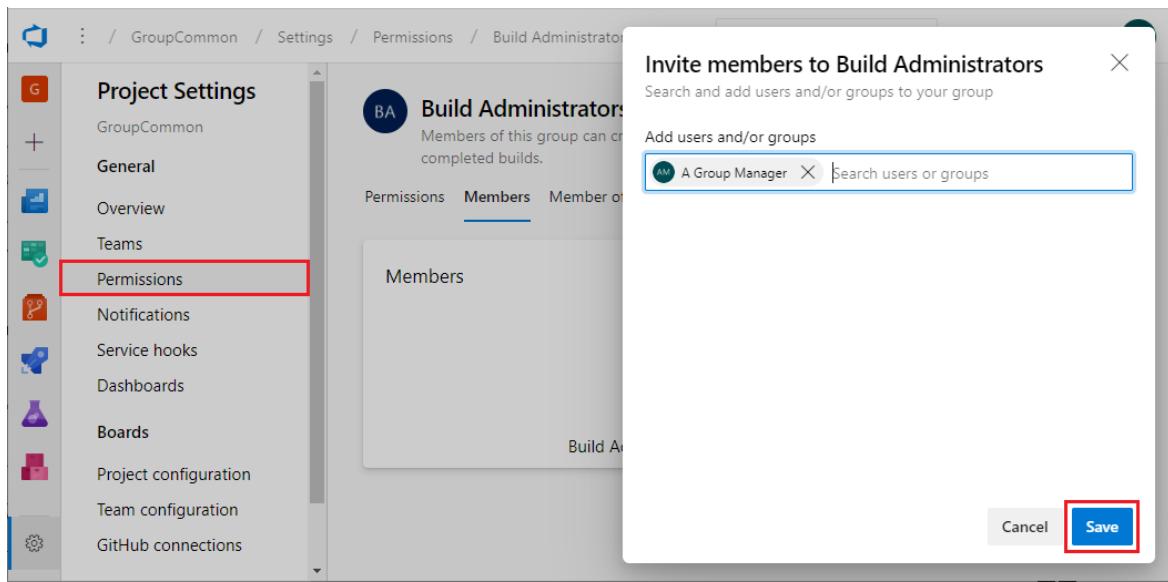
<p>Team Profile</p>  <p>Name GroupCommon Team</p> <p>Description The default project team.</p>	<p>GroupCommon Team</p> <p>Members</p> <p>+ Add...</p> <p>Display Name Username Or Scope AM A Group Manager gm@fabrikam.com</p> <p>membership direct</p>
--	---

4. In the **Add users and groups** dialog, search for and select members to add to the group, and then select **Save changes**.



To configure permissions for members:

1. From the **Project Settings** left navigation, select **Permissions**.
2. On the **Permissions** page, select the group you want to add members to.
3. On the page for that group, select **Members**, and then select **Add**.
4. In the **Invite members** popup, search for and select members to add to the group, and then select **Save**.



Next steps

Here are links to detailed descriptions of the other roles and tasks in the Team Data Science Process:

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

Tasks for the team lead on a Team Data Science Process team

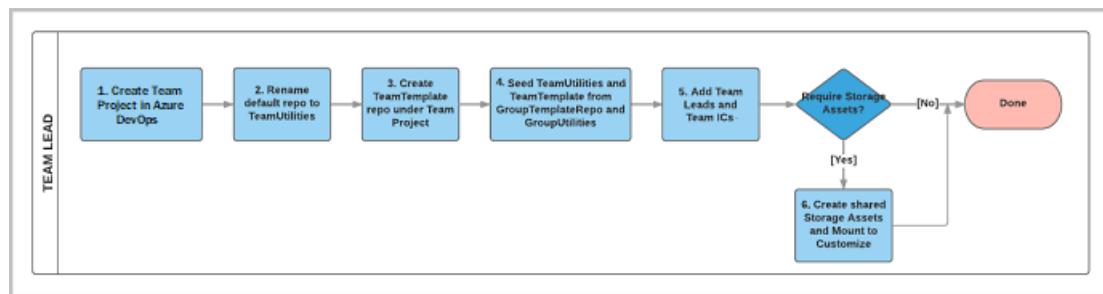
3/5/2021 • 11 minutes to read • [Edit Online](#)

This article describes the tasks that a *team lead* completes for their data science team. The team lead's objective is to establish a collaborative team environment that standardizes on the [Team Data Science Process](#) (TDSP). The TDSP is designed to help improve collaboration and team learning.

The TDSP is an agile, iterative data science methodology to efficiently deliver predictive analytics solutions and intelligent applications. The process distills the best practices and structures from Microsoft and the industry. The goal is successful implementation of data science initiatives and fully realizing the benefits of their analytics programs. For an outline of the personnel roles and associated tasks for a data science team standardizing on the TDSP, see [Team Data Science Process roles and tasks](#).

A team lead manages a team consisting of several data scientists in the data science unit of an enterprise. Depending on the data science unit's size and structure, the [group manager](#) and the team lead might be the same person, or they could delegate their tasks to surrogates. But the tasks themselves do not change.

The following diagram shows the workflow for the tasks the team lead completes to set up a team environment:



1. Create a **team project** in the group's organization in Azure DevOps.
2. Rename the default team repository to **TeamUtilities**.
3. Create a new **TeamTemplate** repository in the team project.
4. Import the contents of the group's **GroupUtilities** and **GroupProjectTemplate** repositories into the **TeamUtilities** and **TeamTemplate** repositories.
5. Set up **security control** by adding team members and configuring their permissions.
6. If required, create team data and analytics resources:
 - Add team-specific utilities to the **TeamUtilities** repository.
 - Create **Azure file storage** to store data assets that can be useful for the entire team.
 - Mount the Azure file storage to the team lead's **Data Science Virtual Machine** (DSVM) and add data assets to it.

The following tutorial walks through the steps in detail.

NOTE

This article uses Azure DevOps and a DSVM to set up a TDSP team environment, because that is how to implement TDSP at Microsoft. If your team uses other code hosting or development platforms, the team lead tasks are the same, but the way to complete them may be different.

Prerequisites

This tutorial assumes that the following resources and permissions have been set up by your [group manager](#):

- The Azure DevOps **organization** for your data unit
- **GroupProjectTemplate** and **GroupUtilities** repositories, populated with the contents of the Microsoft TDSP team's **ProjectTemplate** and **Utilities** repositories
- Permissions on your organization account for you to create projects and repositories for your team

To be able to clone repositories and modify their content on your local machine or DSVM, or set up Azure file storage and mount it to your DSVM, you need the following:

- An Azure subscription.
- Git installed on your machine. If you're using a DSVM, Git is pre-installed. Otherwise, see the [Platforms and tools appendix](#).
- If you want to use a DSVM, the Windows or Linux DSVM created and configured in Azure. For more information and instructions, see the [Data Science Virtual Machine Documentation](#).
- For a Windows DSVM, [Git Credential Manager \(GCM\)](#) installed on your machine. In the *README.md* file, scroll down to the **Download and Install** section and select the **latest installer**. Download the *.exe* installer from the installer page and run it.
- For a Linux DSVM, an SSH public key set up on your DSVM and added in Azure DevOps. For more information and instructions, see the **Create SSH public key** section in the [Platforms and tools appendix](#).

Create a team project and repositories

In this section, you create the following resources in your group's Azure DevOps organization:

- The **MyTeam** project in Azure DevOps
- The **TeamTemplate** repository
- The **TeamUtilities** repository

The names specified for the repositories and directories in this tutorial assume that you want to establish a separate project for your own team within your larger data science organization. However, the entire group can choose to work under a single project created by the group manager or organization administrator. Then, all the data science teams create repositories under this single project. This scenario might be valid for:

- A small data science group that doesn't 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.

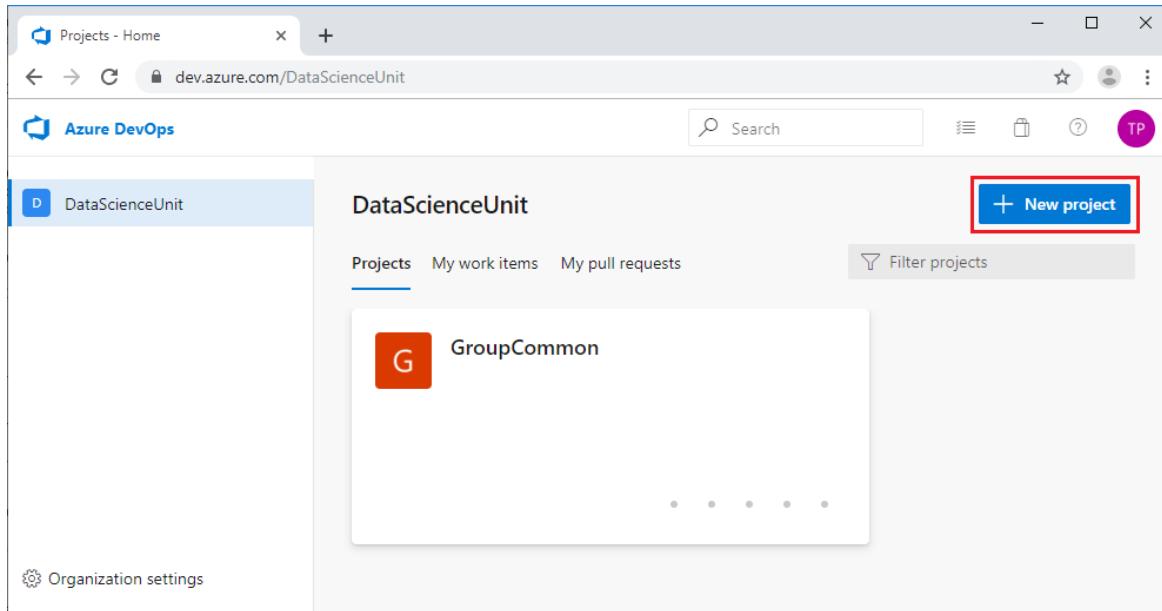
If teams choose to have their team-specific repositories under a single group project, the team leads should create the repositories with names like *<TeamName>Template* and *<TeamName>Utilities*. For instance: *TeamATemplate* and *TeamAUtilities*.

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

Create the MyTeam project

To create a separate project for your team:

1. In your web browser, go to your group's Azure DevOps organization home page at URL <https://<server name>/<organization name>>, and select **New project**.



2. In the **Create project** dialog, enter your team name, such as *MyTeam*, under **Project name**, and then select **Advanced**.
3. Under **Version control**, select **Git**, and under **Work item process**, select **Agile**. Then select **Create**.

The screenshot shows the 'Create new project' dialog. It includes fields for 'Project name' (containing 'MyTeam'), 'Description' (empty), and 'Visibility' (with 'Enterprise' selected). The 'Enterprise' visibility option is highlighted with a blue box. Below these are sections for 'Version control' (set to 'Git') and 'Work item process' (set to 'Agile'). At the bottom right, there are 'Cancel' and 'Create' buttons, with the 'Create' button being highlighted by a red box.

The team project **Summary** page opens, with page URL <https://<server name>/<organization name>/<team name>>.

Rename the MyTeam default repository to TeamUtilities

1. On the **MyTeam** project **Summary** page, under **What service would you like to start with?**, select **Repos**.

The screenshot shows the Azure DevOps interface for the 'MyTeam' project. The top navigation bar includes 'Enterprise' and 'Invite' buttons. The main area features a cartoon illustration of a person working at a desk with a dog. Below the illustration, the text 'Welcome to the project!' is displayed. A callout asks 'What service would you like to start with?' with options for 'Boards', 'Repos' (which is highlighted with a red box), 'Pipelines', and 'Test Plans'. To the right, a 'Project stats' section shows 'No stats are available.' and a 'Setup a service to see progress' button.

2. On the **MyTeam** repo page, select the **MyTeam** repository at the top of the page, and then select **Manage repositories** from the dropdown.

The screenshot shows the Azure DevOps interface for the 'MyTeam' repository. The top navigation bar includes 'Enterprise' and 'Invite' buttons. The main area displays the message 'MyTeam is empty. Add some code...'. A dropdown menu is open next to the repository name 'MyTeam', listing options: 'Filter repositories', 'MyTeam' (selected and highlighted with a red box), '+ New repository', and 'Import repository'. At the bottom of the page, there are links for 'Generate Git credentials' and 'Clone in VS Code'.

3. On the **Project Settings** page, select the ... next to the **MyTeam** repository, and then select **Rename repository**.

The screenshot shows the 'Project Settings' page for a project named 'MyTeam'. On the left sidebar, the 'Pipelines' icon is highlighted with a red box. In the main content area, under the 'Repositories' section, there is a list of repositories. The 'MyTeam' repository is selected, and a red box highlights the '... More options' button next to it. A tooltip 'Rename repository...' is shown below the button. To the right, a panel titled 'Security for MyTeam repository' lists various security roles and groups.

4. In the Rename the MyTeam repository popup, enter *TeamUtilities*, and then select Rename.

Create the TeamTemplate repository

1. On the Project Settings page, select New repository.

The screenshot shows the 'Project Settings' page for a project named 'MyTeam'. On the left sidebar, the 'Pipelines' icon is highlighted with a red box. In the main content area, under the 'Repositories' section, there is a list of repositories. The 'New repository' button is highlighted with a red box. To the right, a panel titled 'Security for MyTeam repository' lists various security roles and groups.

Or, select **Repos** from the left navigation of the **MyTeam** project **Summary** page, select a repository at the top of the page, and then select **New repository** from the dropdown.

2. In the **Create a new repository** dialog, make sure **Git** is selected under **Type**. Enter *TeamTemplate* under **Repository name**, and then select **Create**.

Create a new repository

Type
Git

Repository name *
TeamTemplate

Add a README to describe your repository

Add a .gitignore:
None

Create **Cancel**

3. Confirm that you can see the two repositories **TeamUtilities** and **TeamTemplate** on your project settings page.

Project Settings

- General
- Overview
- Teams
- Permissions
- Notifications
- Service hooks
- Dashboards
- Boards
- Project configuration
- Team configuration
- GitHub connections
- Pipelines

Repositories

New repository

Git repositories

- TeamTemplate
- TeamUtilities

Security for MyTeam repository

Security Options Policies

+ Add... Inheritance ▾

Search

Azure DevOps Groups

- Project Collection Administrators
- Project Collection Build Service Accounts
- Project Collection Service Accounts
- Build Administrators
- Contributors
- Project Administrators
- Readers

Import the contents of the group common repositories

To populate your team repositories with the contents of the group common repositories set up by your group manager:

1. From your **MyTeam** project home page, select **Repos** in the left navigation. If you get a message that the **MyTeam** template is not found, select the link in **Otherwise, navigate to your default TeamTemplate repository.**

The default **TeamTemplate** repository opens.

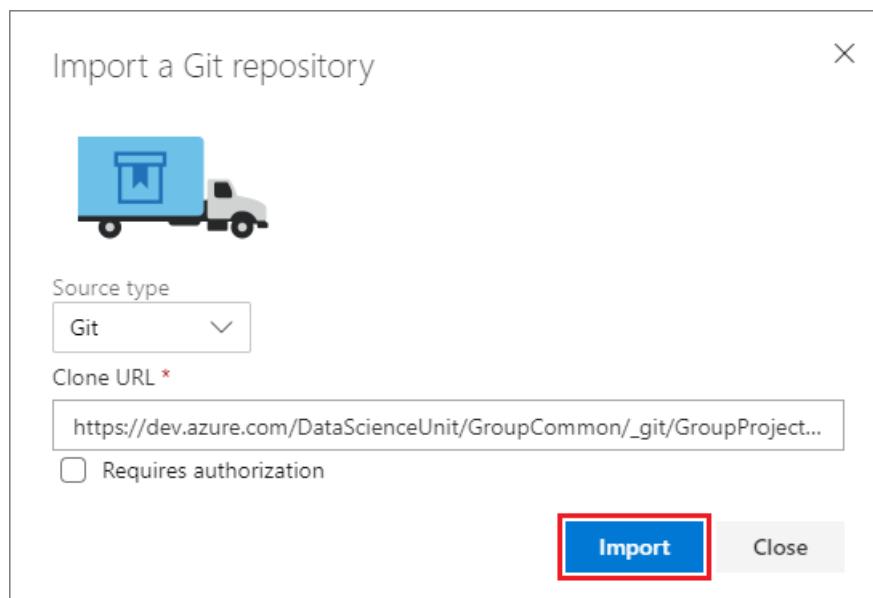
2. On the **TeamTemplate is empty** page, select **Import**.

The screenshot shows the Azure DevOps interface for a 'TeamTemplate' repository. On the left, there's a sidebar with various icons. One icon, which looks like a red square with a white letter 'P', is highlighted with a red box. The main content area displays the message 'TeamTemplate is empty. Add some code!'. Below this, there are sections for cloning the repository:

- Clone to your computer**: Shows 'HTTPS' selected, with the URL https://DataScienceUnit@dev.azure.com/DataScienceUnit/_git/MyTeam. There's also an 'SSH' option and a 'Generate Git credentials' button.
- or push an existing repository from command line**: Shows a command-line interface with the following text:

```
git remote add origin https://DataScienceUnit@dev.azure.com/DataScienceUnit/_git/TeamTemplate
git push -u origin --all
```
- or import a repository**: Contains an 'Import' button.

3. In the **Import a Git repository** dialog, select **Git** as the **Source type**, and enter the URL for your group common template repository under **Clone URL**. The URL is *https://<server name>/<organization name>/_git/<repository name>*. For example: https://dev.azure.com/DataScienceUnit/GroupCommon/_git/GroupProjectTemplate.
4. Select **Import**. The contents of your group template repository are imported into your team template repository.



5. At the top of your project's **Repos** page, drop down and select the **TeamUtilities** repository.
6. Repeat the import process to import the contents of your group common utilities repository, for example *GroupUtilities*, into your **TeamUtilities** repository.

Each of your two team repositories now contains the files from the corresponding group common repository.

Customize the contents of the team repositories

If you want to customize the contents of your team repositories to meet your team's specific needs, you can do that now. You can modify files, change the directory structure, or add files and folders.

To modify, upload, or create files or folders directly in Azure DevOps:

1. On the **MyTeam** project **Summary** page, select **Repos**.
2. At the top of the page, select the repository you want to customize.
3. In the repo directory structure, navigate to the folder or file you want to change.
 - To create new folders or files, select the arrow next to **New**.

The screenshot shows a list of files and folders in a repository. At the top right, there is a 'New' button with a plus sign and a dropdown arrow. A red box highlights this button. A dropdown menu is open, showing two options: 'File' and 'Folder'. The main list below includes:

Name	Last change	Commits
Data_Dictionaries	9/10/2017	b3571ca6
Data_Report	9/10/2017	b3571ca6
Model	9/10/2017	b3571ca6
Project	10/25/2017	58626e35
README.md	just now	d08eb05b

- To upload files, select **Upload file(s)**.

The screenshot shows the same repository interface as above, but with a different view. At the top right, there is a 'New' button with a plus sign and a dropdown arrow. A red box highlights the 'Upload file(s)' button, which has a cloud icon and a downward arrow. The main list below includes:

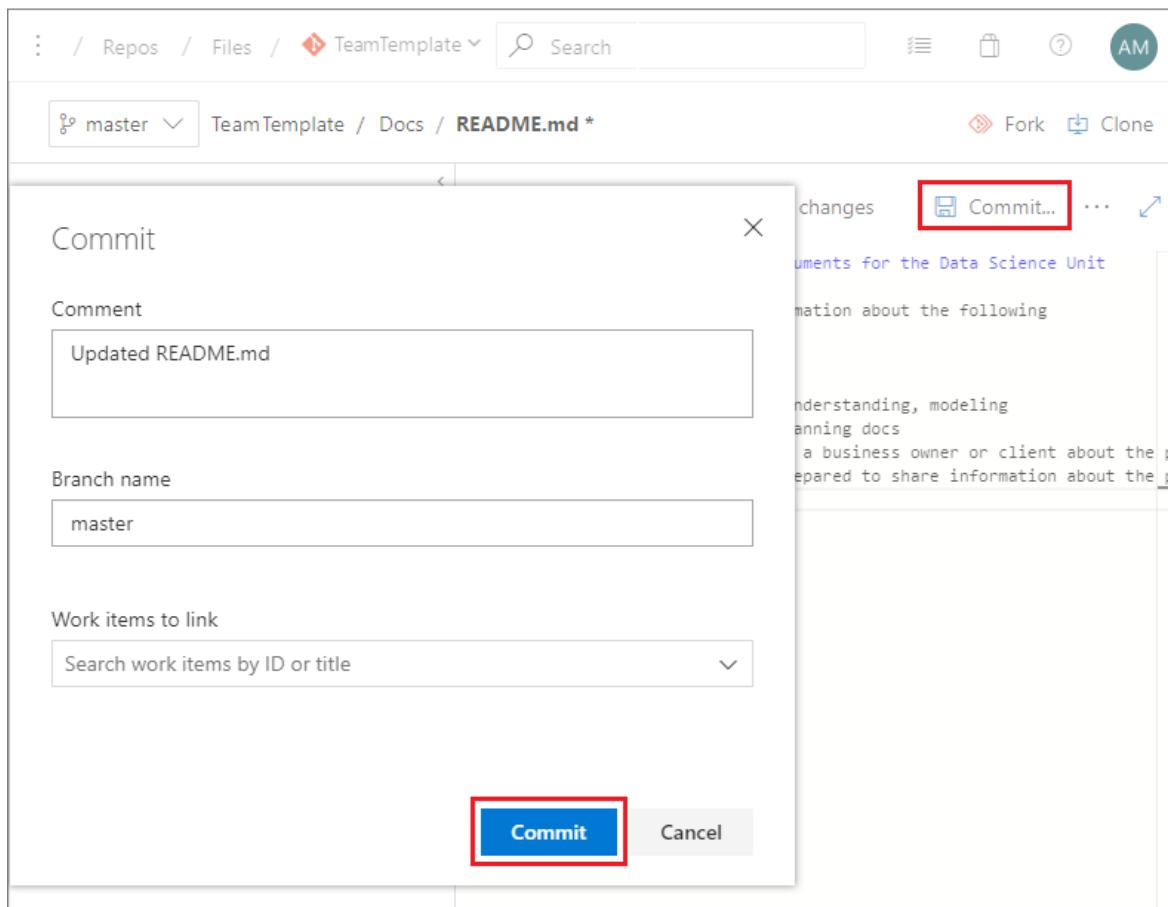
Name	Last change	Commits
Data_Dictionaries	9/10/2017	b3571ca6
Data_Report	9/10/2017	b3571ca6
Model	9/10/2017	b3571ca6
Project	10/25/2017	58626e35
README.md	2 hours ago	d08eb05b

- To edit existing files, navigate to the file and then select **Edit**.

The screenshot shows a detailed view of a file named 'Data_Dictionaries'. At the top right, there is a 'New' button with a plus sign and a dropdown arrow. A red box highlights the 'Edit' button, which has a pencil icon. Below the file name, there is a section titled 'Folder for hosting all documents for the Data Science Unit'. Underneath, it says 'Documents will contain information about the following' and lists six items:

1. System architecture
2. Data dictionaries
3. Reports related to data understanding, modeling
4. Project management and planning docs
5. Information obtained from a business owner or client about the project
6. Docs and presentations prepared to share information about the project

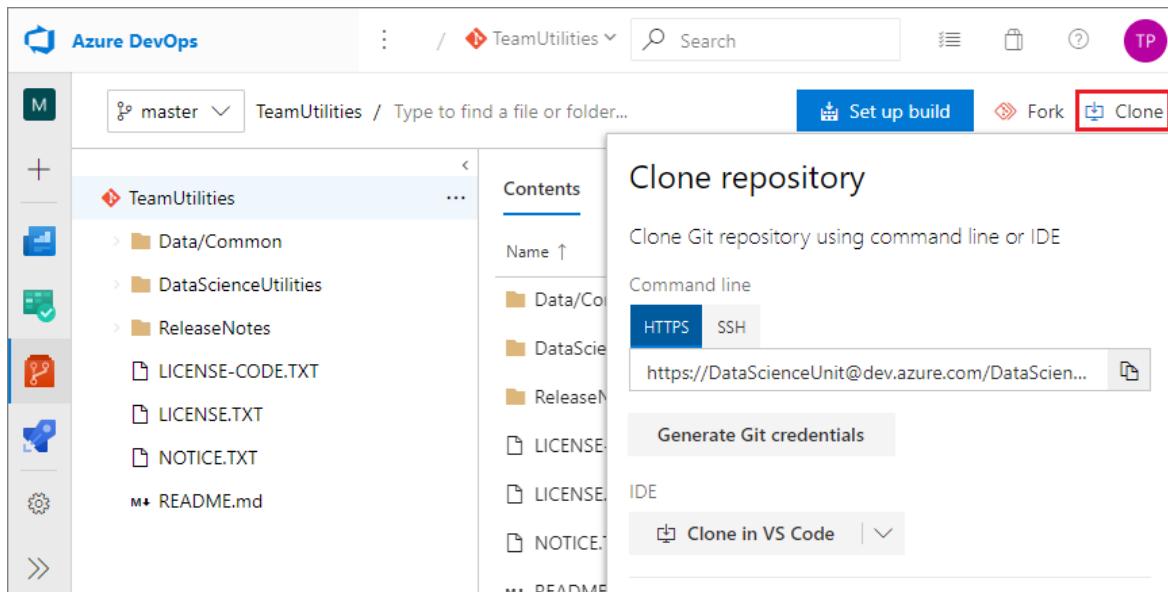
- After adding or editing files, select **Commit**.



To work with repositories on your local machine or DSVM, you first copy or *clone* the repositories to your local machine, and then commit and push your changes up to the shared team repositories,

To clone repositories:

- On the **MyTeam** project **Summary** page, select **Repos**, and at the top of the page, select the repository you want to clone.
- On the repo page, select **Clone** at upper right.
- In the **Clone repository** dialog, under **Command line**, select **HTTPS** for an HTTP connection or **SSH** for an SSH connection, and copy the clone URL to your clipboard.



4. On your local machine, create the following directories:

- For Windows: C:\GitRepos\MyTeam
- For Linux, \$home/GitRepos/MyTeam

5. Change to the directory you created.

6. In Git Bash, run the command `git clone <clone URL>`, where <clone URL> is the URL you copied from the **Clone** dialog.

For example, use one of the following commands to clone the **TeamUtilities** repository to the *MyTeam* directory on your local machine.

HTTPS connection:

```
git clone https://DataScienceUnit@dev.azure.com/DataScienceUnit/MyTeam/_git/TeamUtilities
```

SSH connection:

```
git clone git@ssh.dev.azure.com:v3/DataScienceUnit/MyTeam/TeamUtilities
```

After making whatever changes you want in the local clone of your repository, commit and push the changes to the shared team repositories.

Run the following Git Bash commands from your local **GitRepos\MyTeam\TeamTemplate** or **GitRepos\MyTeam\TeamUtilities** directory.

```
git add .
git commit -m "push from local"
git push
```

NOTE

If this is the first time you commit to a Git repository, you may need to configure global parameters `username` 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're committing to several Git repositories, use the same name and email address for all of them. Using the same name and email address is convenient when building Power BI dashboards to track your Git activities in multiple repositories.

Add team members and configure permissions

To add members to the team:

1. In Azure DevOps, from the **MyTeam** project home page, select **Project settings** from the left navigation.
2. From the **Project Settings** left navigation, select **Teams**, then on the **Teams** page, select the **MyTeam Team**.

The screenshot shows the 'Project Settings' page for 'MyTeam'. The left sidebar has a red box around the 'Teams' icon. The main area shows a table with one team entry:

Team Name	Members	Description
MyTeam Team	1	The default project team.

3. On the **Team Profile** page, select **Add**.

The screenshot shows the 'Team Profile' page for 'MyTeam Team'. The 'Members' section has a red box around the '+ Add...' button. The table shows one member:

Display Name	Username Or Scope
AM A Group Manager	gm@fabrikam.com

4. In the **Add users and groups** dialog, search for and select members to add to the group, and then select **Save changes**.

The screenshot shows the 'Add users and groups' dialog. The search bar contains '[MyTeam]\MyTeam Team'. The 'Save changes' button is highlighted with a red box.

To configure permissions for team members:

1. From the **Project Settings** left navigation, select **Permissions**.
2. On the **Permissions** page, select the group you want to add members to.
3. On the page for that group, select **Members**, and then select **Add**.
4. In the **Invite members** popup, search for and select members to add to the group, and then select **Save**.

The screenshot shows the 'Project Administrators' page in the Azure DevOps interface. On the left, there's a sidebar with various project settings like General, Overview, Teams, Permissions (which is highlighted with a red box), Notifications, Service hooks, and Dashboards. The main area is titled 'Project Administrators' and describes the group as having full operations in the team project. It has tabs for Permissions, Members (which is selected and highlighted with a blue box), Member of, and Settings. A search bar at the top right says 'Search users and groups'. Below the tabs is a table titled 'Members' with columns for Name, Type, and Username or scope. One row is shown: 'Team Lead' (fabrikamfiber4@outlook.com) is a user type. At the bottom right of the table is a blue 'Add' button with a red box around it.

Create team data and analytics resources

This step is optional, but 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. You can create Azure file storage and mount it on your DSVM to share with team members.

For information about sharing other resources with your team, such as Azure HDInsight Spark clusters, see [Platforms and tools](#). That topic provides 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.

NOTE

To avoid transmitting data across data centers, which might be slow and costly, make sure that your Azure resource group, storage account, and DSVM are all hosted in the same Azure region.

Create Azure file storage

1. Run the following script to create Azure file storage for data assets that are useful for your entire team.

The script prompts you for your Azure subscription information, so have that ready to enter.

- On a Windows machine, run the script from the PowerShell command prompt:

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

- On a Linux machine, run the script from the Linux shell:

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

2. Log in to your Microsoft Azure account when prompted, and select the subscription you want to use.

3. Select the storage account to use, or create a new one under your selected subscription. You can use lowercase characters, numbers, and hyphens for the Azure file storage name.

4. To facilitate mounting and sharing the storage, press Enter or enter *Y* to save the Azure file storage information into a text file in your current directory. You can check in this text file to your **TeamTemplate** repository, ideally under **Docs\Dictionary**, so all projects in your team can access it. You also need the file information to mount your Azure file storage to your Azure DSVM in the next section.

Mount Azure file storage on your local machine or DSVM

1. To mount your Azure file storage to your local machine or DSVM, use the following script.

- On a Windows machine, run the script from the PowerShell command prompt:

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

- On a Linux machine, run the script from the Linux shell:

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

2. Press Enter or enter *Y* to continue, if you saved an Azure file storage information file in the previous step.

Enter the complete path and name of the file you created.

If you don't have an Azure file storage information file, enter *n*, and follow the instructions to enter your subscription, Azure storage account, and Azure file storage information.

3. Enter the name of a local or TDSP drive to mount the file share on. The screen displays a list of existing drive names. Provide a drive name that doesn't already exist.
4. Confirm that the new drive and storage is successfully mounted on your machine.

Next steps

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

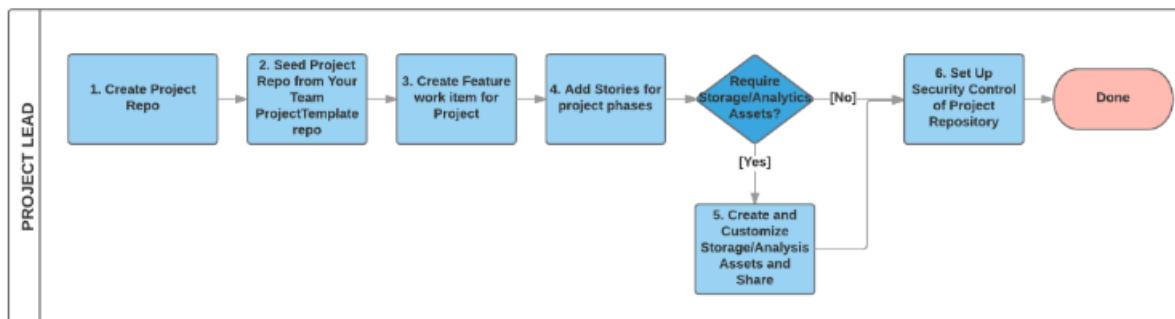
- [Group Manager tasks for a data science team](#)
- [Project Lead tasks for a data science team](#)
- [Project Individual Contributor tasks for a data science team](#)

Project lead tasks in the Team Data Science Process

3/5/2021 • 3 minutes to read • [Edit Online](#)

This article describes tasks that a *project lead* completes to set up a repository for their project team in the [Team Data Science Process](#) (TDSP). The TDSP is a framework developed by Microsoft that provides a structured sequence of activities to efficiently execute cloud-based, predictive analytics solutions. The TDSP is designed to help improve collaboration and team learning. For an outline of the personnel roles and associated tasks for a data science team standardizing on the TDSP, 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 in the TDSP. The following diagram shows the workflow for project lead tasks:



This tutorial covers Step 1: Create project repository, and Step 2: Seed project repository from your team ProjectTemplate repository.

For Step 3: Create Feature work item for project, and Step 4: Add Stories for project phases, see [Agile development of data science projects](#).

For Step 5: Create and customize storage/analysis assets and share, if necessary, see [Create team data and analytics resources](#).

For Step 6: Set up security control of project repository, see [Add team members and configure permissions](#).

NOTE

This article uses Azure Repos to set up a TDSP project, because that is how to implement TDSP at Microsoft. If your team uses another code hosting platform, the project lead tasks are the same, but the way to complete them may be different.

Prerequisites

This tutorial assumes that your [group manager](#) and [team lead](#) have set up the following resources and permissions:

- The Azure DevOps **organization** for your data unit
- A team **project** for your data science team
- Team template and utilities **repositories**
- **Permissions** on your organization account for you to create and edit repositories for your project

To clone repositories and modify content on your local machine or Data Science Virtual Machine (DSVM), or set up Azure file storage and mount it to your DSVM, you also need to consider this checklist:

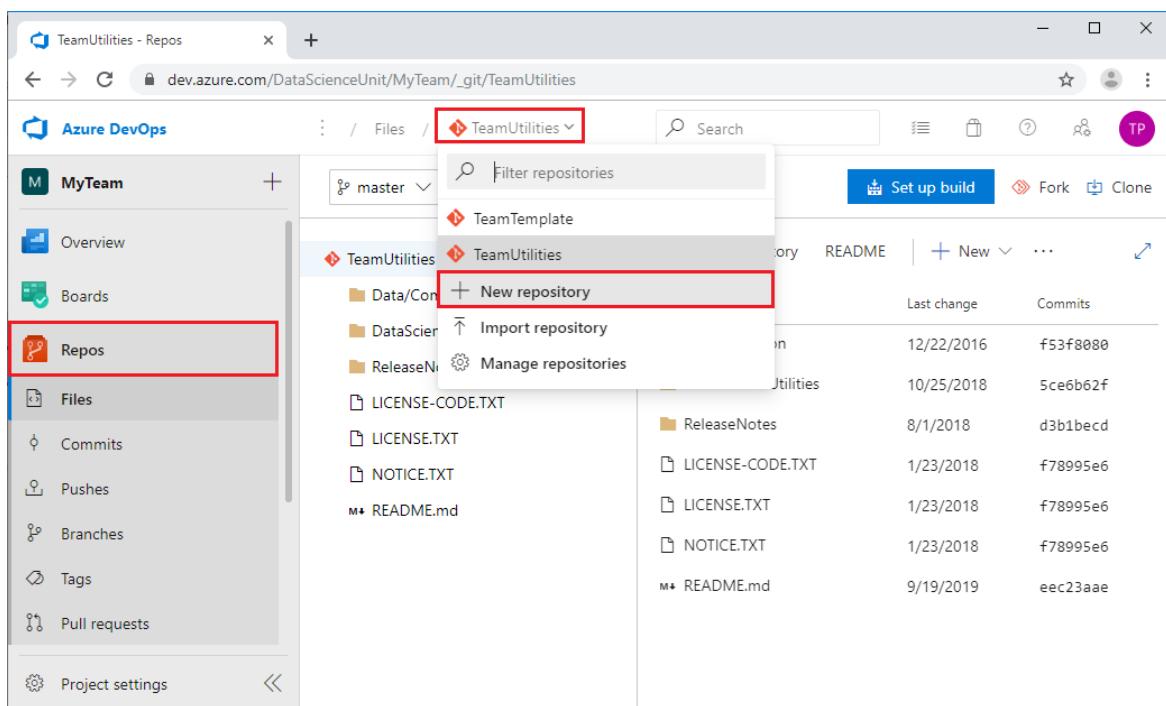
- An Azure subscription.

- Git installed on your machine. If you're using a DSVM, Git is pre-installed. Otherwise, see the [Platforms and tools appendix](#).
- If you want to use a DSVM, the Windows or Linux DSVM created and configured in Azure. For more information and instructions, see the [Data Science Virtual Machine Documentation](#).
- For a Windows DSVM, [Git Credential Manager \(GCM\)](#) installed on your machine. In the *README.md* file, scroll down to the **Download and Install** section and select the **latest installer**. Download the .exe installer from the installer page and run it.
- For a Linux DSVM, an SSH public key set up on your DSVM and added in Azure DevOps. For more information and instructions, see the [Create SSH public key](#) section in the [Platforms and tools appendix](#).

Create a project repository in your team project

To create a project repository in your team's MyTeam project:

1. Go to your team's project **Summary** page at <https://<server name>/<organization name>/<team name>>, for example, <https://dev.azure.com/DataScienceUnit/MyTeam>, and select **Repos** from the left navigation.
2. Select the repository name at the top of the page, and then select **New repository** from the dropdown.



3. In the **Create a new repository** dialog, make sure **Git** is selected under **Type**. Enter *DSProject1* under **Repository name**, and then select **Create**.

Create a new repository

Type
Git

Repository name *

DSProject1

Add a README to describe your repository

Add a .gitignore:
None

Create **Cancel**

4. Confirm that you can see the new **DSProject1** repository on your project settings page.

DataScienceUnit / MyTeam / Settings / Repositories

M Pipelines

Agent pools

Parallel jobs

Settings

Test management

Release retention

Service connections

Repos

Repositories

Policies

Git repositories

DSProject1

TeamTemplate

TeamUtilities

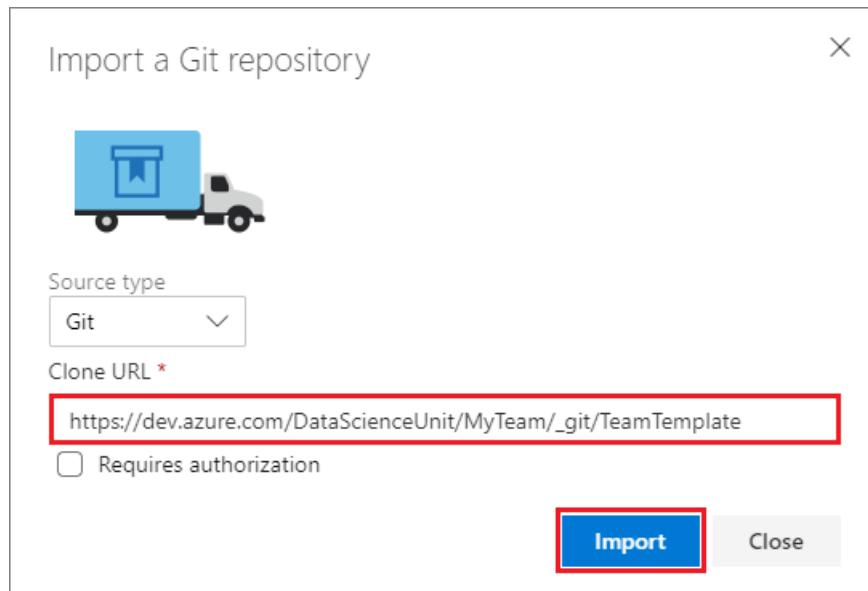
Import the team template into your project repository

To populate your project repository with the contents of your team template repository:

1. From your team's project **Summary** page, select **Repos** in the left navigation.
2. Select the repository name at the top of the page, and select **DSProject1** from the dropdown.
3. On the **DSProject1 is empty** page, select **Import**.

The screenshot shows the Azure DevOps interface for a project named 'DSProject1'. On the left, there's a sidebar with various icons. One icon, which looks like a folder with a question mark, is highlighted with a red box. The main content area displays instructions for cloning the repository. It shows two cloning methods: 'Clone to your computer' (using HTTPS or SSH) and 'or push an existing repository from command line'. Below these, there's a section for 'or import a repository' with a red box around the 'Import' button.

4. In the **Import** a Git repository dialog, select **Git** as the **Source type**, and enter the URL for your **TeamTemplate** repository under **Clone URL**. The URL is `https://<server name>/<organization name>/<team name>/_git/<team template repository name>`. For example: `https://dev.azure.com/DataScienceUnit/_git/TeamTemplate`.
5. Select **Import**. The contents of your team template repository are imported into your project repository.



If you need to customize the contents of your project repository to meet your project's specific needs, you can add, delete, or modify repository files and folders. You can work directly in Azure Repos, or clone the repository to your local machine or DSVM, make changes, and commit and push your updates to the shared project repository. Follow the instructions at [Customize the contents of the team repositories](#).

Next steps

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

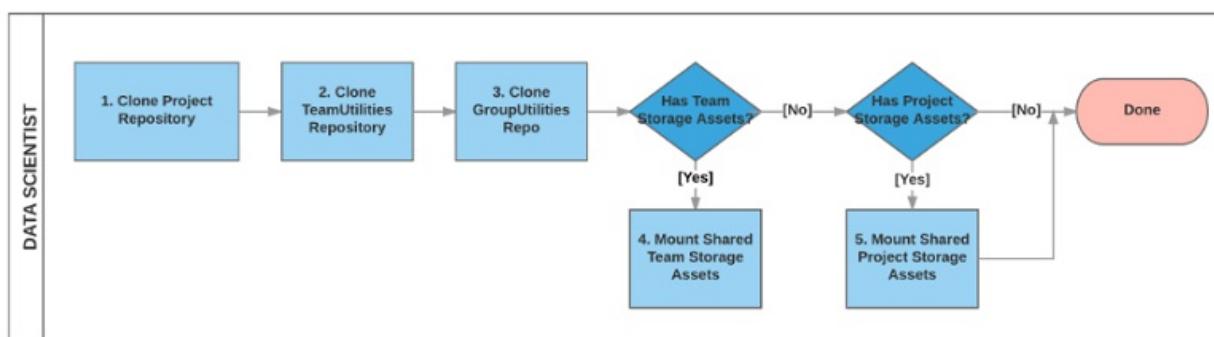
- [Group Manager tasks for a data science team](#)
- [Team Lead tasks for a data science team](#)
- [Individual Contributor tasks for a data science team](#)

Tasks for an individual contributor in the Team Data Science Process

3/5/2021 • 3 minutes to read • [Edit Online](#)

This topic outlines the tasks that an *individual contributor* completes to set up a project in the [Team Data Science Process](#) (TDSP). The objective is to work in a collaborative team environment that standardizes on the TDSP. The TDSP is designed to help improve collaboration and team learning. For an outline of the personnel roles and their associated tasks that are handled by a data science team standardizing on the TDSP, see [Team Data Science Process roles and tasks](#).

The following diagram shows the tasks that project individual contributors (data scientists) complete to set up their team environment. For instructions on how to execute a data science project under the TDSP, see [Execution of data science projects](#).



- **ProjectRepository** is the repository your project team maintains to share project templates and assets.
- **TeamUtilities** is the utilities repository your team maintains specifically for your team.
- **GroupUtilities** is the repository your group maintains to share useful utilities across the entire group.

NOTE

This article uses Azure Repos and a Data Science Virtual Machine (DSVM) to set up a TDSP environment, because that is how to implement TDSP at Microsoft. If your team uses other code hosting or development platforms, the individual contributor tasks are the same, but the way to complete them may be different.

Prerequisites

This tutorial assumes that the following resources and permissions have been set up by your [group manager](#), [team lead](#), and [project lead](#):

- The Azure DevOps **organization** for your data science unit
- A **project repository** set up by your project lead to share project templates and assets
- **GroupUtilities** and **TeamUtilities** repositories set up by the group manager and team lead, if applicable
- Azure **file storage** set up for shared assets for your team or project, if applicable
- **Permissions** for you to clone from and push back to your project repository

To clone repositories and modify content on your local machine or DSVM, or mount Azure file storage to your DSVM, you need to consider this checklist:

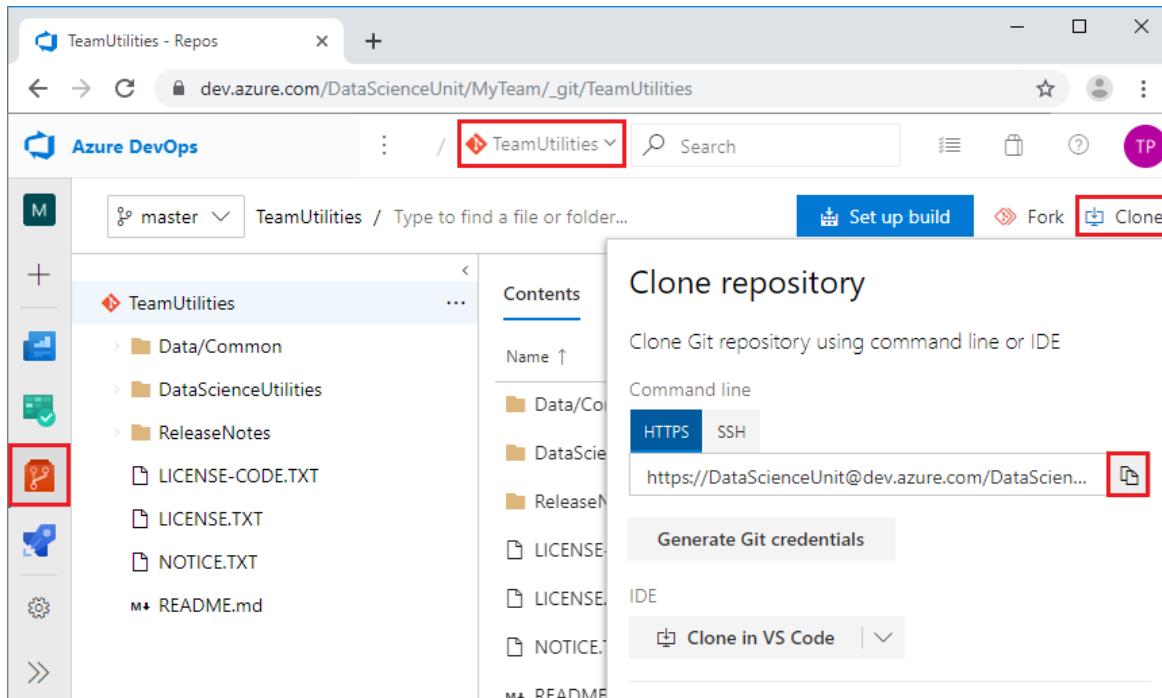
- An Azure subscription.

- Git installed on your machine. If you're using a DSVM, Git is pre-installed. Otherwise, see the [Platforms and tools appendix](#).
- If you want to use a DSVM, the Windows or Linux DSVM created and configured in Azure. For more information and instructions, see the [Data Science Virtual Machine Documentation](#).
- For a Windows DSVM, [Git Credential Manager \(GCM\)](#) installed on your machine. In the *README.md* file, scroll down to the **Download and Install** section and select the **latest installer**. Download the *.exe* installer from the installer page and run it.
- For a Linux DSVM, an SSH public key set up on your DSVM and added in Azure DevOps. For more information and instructions, see the **Create SSH public key** section in the [Platforms and tools appendix](#).
- The Azure file storage information for any Azure file storage you need to mount to your DSVM.

Clone repositories

To work with repositories locally and push your changes up to the shared team and project repositories, you first copy or *clone* the repositories to your local machine.

1. In Azure DevOps, go to your team's project Summary page at <https://<server name>/<organization name>/<team name>>, for example, <https://dev.azure.com/DataScienceUnit/MyTeam>.
2. Select **Repos** in the left navigation, and at the top of the page, select the repository you want to clone.
3. On the repo page, select **Clone** at upper right.
4. In the **Clone repository** dialog, select **HTTPS** for an HTTP connection, or **SSH** for an SSH connection, and copy the clone URL under **Command line** to your clipboard.



5. On your local machine or DSVM, create the following directories:
 - For Windows: `C:\GitRepos`
 - For Linux: `$home/GitRepos`
6. Change to the directory you created.
7. In Git Bash, run the command `git clone <clone URL>` for each repository you want to clone.

For example, the following command clones the **TeamUtilities** repository to the *MyTeam* directory on your local machine.

HTTPS connection:

```
git clone https://DataScienceUnit@dev.azure.com/DataScienceUnit/MyTeam/_git/TeamUtilities
```

SSH connection:

```
git clone git@ssh.dev.azure.com:v3/DataScienceUnit/MyTeam/TeamUtilities
```

8. Confirm that you can see the folders for the cloned repositories in your local project directory.

```
azureuser@LinuxDSVM1:~/Project_1$  
azureuser@LinuxDSVM1:~/Project_1$ dir  
GroupUtilities ProjectRepository TeamUtilities
```

Mount Azure file storage to your DSVM

If your team or project has shared assets in Azure file storage, mount the file storage to your local machine or DSVM. Follow the instructions at [Mount Azure file storage on your local machine or DSVM](#).

Next steps

Here are links to detailed descriptions of the other 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](#)

Team Data Science Process project planning

3/5/2021 • 2 minutes 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:

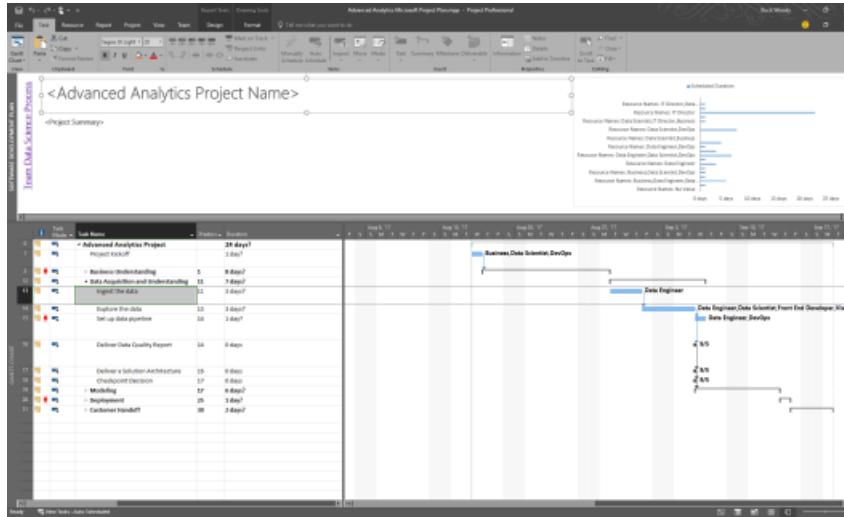
- Business Understanding
- Data Acquisition and Understanding
- Modeling
- Deployment
- Customer Acceptance

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

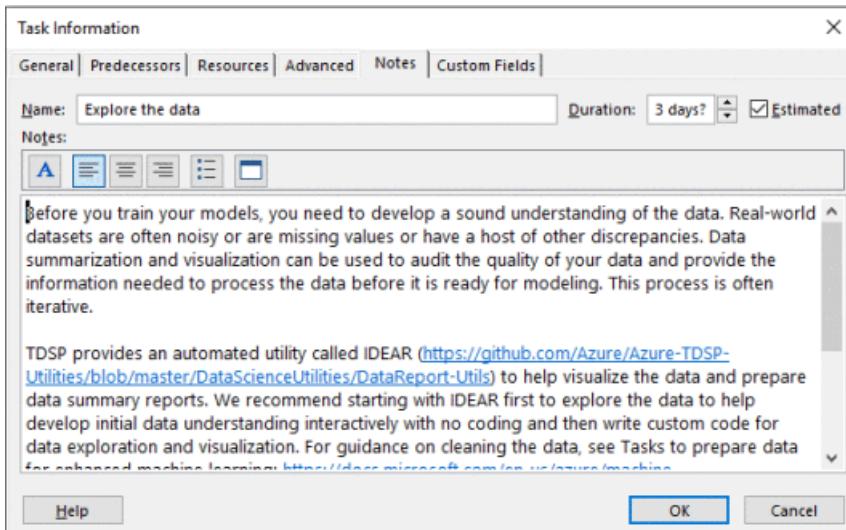
Microsoft Project template

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.

Repository template

Use this [project template repository](#) to support efficient project execution and collaboration. This repository gives you a standardized directory structure and document templates you can use for your own TDSP project.

Next steps

[Agile development of data science projects](#) This document describes a data science project in a systematic, version controlled, and collaborative way 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](#) article. 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

11/2/2020 • 7 minutes to read • [Edit Online](#)

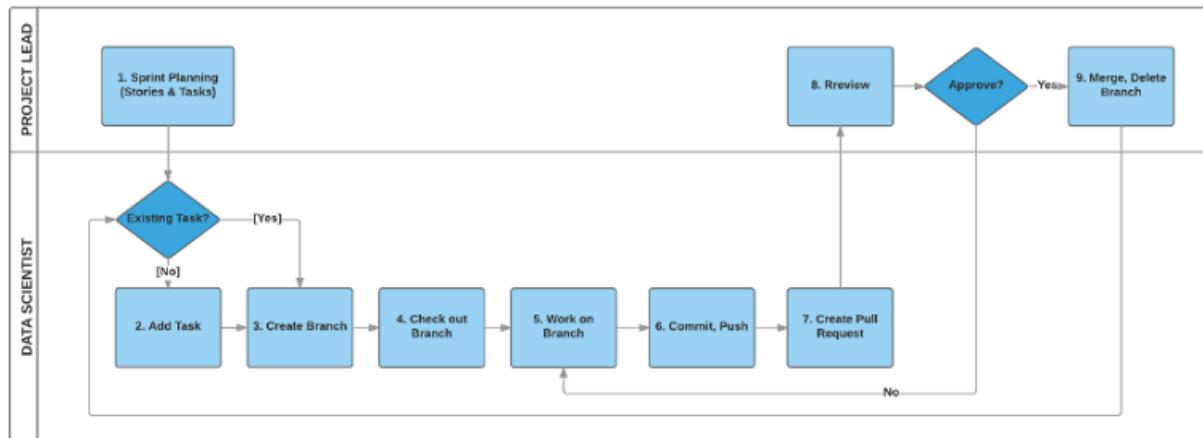
This document describes how developers can execute a data science project in a systematic, version controlled, and collaborative way within a project team by using the [Team Data Science Process](#) (TDSP). The TDSP is a framework developed by Microsoft that provides a structured sequence of activities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

This article includes instructions on how to:

- Do *sprint planning* for work items involved in a project.
- Add *work items* to sprints.
- Create and use an *agile-derived work item template* that specifically aligns with TDSP lifecycle stages.

The following instructions outline the steps needed to set up a TDSP team environment using Azure Boards and Azure Repos in Azure DevOps. The instructions use Azure DevOps because that is how to implement TDSP at Microsoft. If your group uses a different code hosting platform, the team lead tasks generally don't change, but the way to complete the tasks is different. For example, linking a work item with a Git branch might not be the same with GitHub as it is with Azure Repos.

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



Work item types

In the TDSP sprint planning framework, there are four frequently used *work item* types: *Features*, *User Stories*, *Tasks*, and *Bugs*. The backlog for all work items is at the project level, not the Git repository level.

Here are the definitions for the work item types:

- **Feature:** A Feature corresponds to a project engagement. Different engagements with a client are different Features, and it's best to consider different phases of a project as different Features. If you choose a schema such as `<ClientName>-<EngagementName>` to name your Features, you can easily recognize the context of the project and engagement from the names themselves.
- **User Story:** User Stories are work items needed to complete a Feature end-to-end. Examples of User Stories include:

- Get data
 - Explore data
 - Generate features
 - Build models
 - Operationalize models
 - Retrain models
- **Task:** Tasks are assignable work items that need to be done to complete a specific User Story. For example, Tasks in the User Story *Get data* could be:
 - Get SQL Server credentials
 - Upload data to Azure Synapse Analytics
 - **Bug:** Bugs are issues in existing code or documents that must be fixed to complete a Task. If Bugs are caused by missing work items, they can escalate to be User Stories or Tasks.

Data scientists may feel more comfortable using an agile template that replaces Features, User Stories, and Tasks with TDSP lifecycle stages and substages. To create an agile-derived template that specifically aligns with the TDSP lifecycle stages, see [Use an agile TDSP work template](#).

NOTE

TDSP borrows the concepts of Features, User Stories, Tasks, and Bugs from software code management (SCM). The TDSP concepts might differ slightly from their conventional SCM definitions.

Plan sprints

Many data scientists are engaged with multiple projects, which can take months to complete and proceed at different paces. Sprint planning is useful for project prioritization, and resource planning and allocation. In Azure Boards, you can easily create, manage, and track work items for your projects, and conduct sprint planning to ensure projects are moving forward as expected.

For more information about sprint planning, see [Scrum sprints](#).

For more information about sprint planning in Azure Boards, see [Assign backlog items to a sprint](#).

Add a Feature to the backlog

After your project and project code repository are created, you can add a Feature to the backlog to represent the work for your project.

1. From your project page, select **Boards** > **Backlogs** in the left navigation.
2. On the **Backlog** tab, if the work item type in the top bar is **Stories**, drop down and select **Features**. Then select **New Work Item**.

The screenshot shows the Azure DevOps interface for the 'FabrikamFiber Team Stories Backlog'. The left sidebar has 'Backlogs' selected. The main area features a 'Get started with your product backlog' message and a 'New Work Item' button. The top navigation bar includes a 'Features' dropdown, which is also highlighted with a red box.

3. Enter a title for the Feature, usually your project name, and then select **Add to top**.

The screenshot shows the backlog page with a new feature named 'FabrikamFiber'. The 'Add to top' button is highlighted with a red box. The backlog navigation bar also has a 'Features' dropdown highlighted with a red box.

4. From the **Backlog** list, select and open the new Feature. Fill in the description, assign a team member, and set planning parameters.

You can also link the Feature to the project's Azure Repos code repository by selecting **Add link** under the **Development** section.

After you edit the Feature, select **Save & Close**.

The screenshot shows the details page for 'FEATURE 19'. The 'Development' section is highlighted with a red box, containing a '+ Add link' button. Other sections like 'Description', 'Planning', and 'Related Work' are also visible.

Add a User Story to the Feature

Under the Feature, you can add User Stories to describe major steps needed to complete the project.

To add a new User Story to a Feature:

1. On the **Backlog** tab, select the + to the left of the Feature.

The screenshot shows the Azure Boards interface. At the top, it displays the organization path: fabrikam-org / FabrikamFiber / Boards / Backlogs. Below this, the team name 'FabrikamFiber Team' is shown with a dropdown arrow. The main navigation bar includes 'Backlog' (selected), 'Analytics', '+ New Work Item', and 'View as Board'. Under the backlog, there are filters for 'Order', 'Work Item Type', and 'Title'. A specific work item is selected, highlighted with a red box around its icon. The work item details are: ID 1, Type Feature, Title 'FabrikamFiber'. In the bottom-left corner of the backlog area, there is a button labeled 'Add: User Story'.

2. Give the User Story a title, and edit details such as assignment, status, description, comments, planning, and priority.

You can also link the User Story to a branch of the project's Azure Repos code repository by selecting **Add link** under the **Development** section. Select the repository and branch you want to link the work item to, and then select **OK**.

The image contains two side-by-side screenshots. The left screenshot is a modal dialog titled 'Add link'. It contains the following fields:

- 'You are adding a link from:' dropdown set to 'Get data'.
- 'Link type' dropdown set to 'Branch'.
- 'Repository' dropdown set to 'FabrikamFiber'.
- 'Branch' dropdown set to 'master'.
- 'Comment' text area (empty).

A red box highlights the 'OK' button at the bottom of the dialog. The right screenshot shows a work item details page. The 'Development' tab is selected, showing the following:

- 'Business' section.
- 'Development' section, which has a red box around the '+ Add link' button.
- 'Related Work' section.

3. When you're finished editing the User Story, select **Save & Close**.

Add a Task to a User Story

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

To add a Task to a User Story, select the + next to the User Story item, and select **Task**. Fill in the title and other information in the Task.

The screenshot shows the 'Backlog' view for the 'FabrikamFiber Team'. At the top, there's a navigation bar with 'fabrikam-org / FabrikamFiber / Boards / Backlogs'. Below it is a header with 'FabrikamFiber Team' and icons for 'Backlog', 'Analytics', 'New Work Item', and 'View as Board'. The main area shows a table with one row: '1 Feature' under 'FabrikamFiber'. Below this is a dropdown menu with three options: 'Task' (highlighted with a red box), 'Get data', and 'Bug'.

After you create Features, User Stories, and Tasks, you can view them in the **Backlogs** or **Boards** views to track their status.

This screenshot shows the 'Backlogs' view for the 'FabrikamFiber Team'. The left sidebar has 'Backlogs' selected with a red box. The main area displays a table with three rows: '1 Feature' (State: New, Effort: 1), 'User Story' (State: New), and 'Task' (State: New). There are also buttons for 'New Work Item' and 'Features'.

This screenshot shows the 'Boards' view for the 'FabrikamFiber Team'. The left sidebar has 'Boards' selected with a red box. The main area shows a board with columns: 'New' (0/5), 'Active' (0/5), 'Resolved' (0/5), and 'Closed' (0/5). A card for '19 FabrikamFiber' (Team Lead) is visible, along with buttons for 'New item', 'Add User Story', and 'Get data'.

Use an agile TDSP work template

Data scientists may feel more comfortable using an agile template that replaces Features, User Stories, and Tasks with TDSP lifecycle stages and substages. In Azure Boards, you can create an agile-derived template that uses TDSP lifecycle stages to create and track work items. The following steps walk through setting up a data science-specific agile process template and creating data science work items based on the template.

Set up an Agile Data Science Process template

1. From your Azure DevOps organization main page, select **Organization settings** from the left navigation.
2. In the **Organization Settings** left navigation, under **Boards**, select **Process**.

3. In the All processes pane, select the ... next to Agile, and then select Create inherited process.

The screenshot shows the 'Organization Settings' section under 'fabrikam-org'. The 'Process' tab is selected. A list of processes is shown, with 'Agile (default)' being the first item. A context menu is open over 'Agile (default)', with the 'Create inherited process' option highlighted and enclosed in a red box. Other options in the menu include '...', '+ New team project', and 'Edit'.

4. In the Create inherited process from Agile dialog, enter the name *AgileDataScienceProcess*, and select Create process.

The dialog title is 'Create inherited process from Agile'. It displays a tree structure where 'Agile [system process]' has a child node 'AgileDataScienceProcess' (which is highlighted with a red box). Below the tree is a 'Description' text area and a 'Learn more' link. At the bottom are 'Create process' and 'Cancel' buttons, with 'Create process' also highlighted with a red box.

5. In All processes, select the new AgileDataScienceProcess.

6. On the Work item types tab, disable Epic, Feature, User Story, and Task by selecting the ... next to each item and then selecting Disable.

All processes > AgileDataScienceProcess		Help	Filter by work item type name
Work item types	Backlog levels	Projects	
+ New work item type			
Name	Description		
Bug	Describes a divergence between required and actual behavior, and trac...		
Epic	Epics help teams effectively manage and groom their product backlog	...	
Feature	I will be released with the product	Edit	
Issue	Tracks an obstacle or problem.	Disable	
Task	Tracks work that needs to be done.		
Test Case	Server-side data for a set of steps to be tested.		
Test Plan	Tracks test activities for a specific milestone or release.		
Test Suite	Tracks test activities for a specific feature, requirement, or user story.		
User Story	Tracks an activity the user will be able to perform with the product		

7. In All processes, select the Backlog levels tab. Under Portfolios backlogs, select the ... next to Epic (disabled), and then select Edit/Rename.
8. In the Edit backlog level dialog box:
 - a. Under Name, replace Epic with TDSP Projects.
 - b. Under Work item types on this backlog level, select New work item type, enter TDSP Project, and select Add.
 - c. Under Default work item type, drop down and select TDSP Project.
 - d. Select Save.

Edit backlog level

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

Name

Work item types on this backlog level

Epic (disabled)
 TDSP Project
[+ New work item type](#)

Default work item type

Save **Cancel**

9. Follow the same steps to rename Features to TDSP Stages, and add the following new work item types:

- *Business Understanding*
- *Data Acquisition*
- *Modeling*
- *Deployment*

10. Under **Requirement backlog**, rename **Stories** to *TDSP Substages*, add the new work item type *TDSP Substage*, and set the default work item type to **TDSP Substage**.

11. Under **Iteration backlog**, add a new work item type *TDSP Task*, and set it to be the default work item type.

After you complete the steps, the backlog levels should look like this:

All processes > AgileDataScienceProcess		
Work item types	Backlog levels	Projects
Portfolio backlogs		
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 Stages	Business Understanding	
	Data Acquisition	
	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 (default)	
	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 (default)	

Create Agile Data Science Process work items

You can use the data science process template to create TDSP projects and track work items that correspond to TDSP lifecycle stages.

1. From your Azure DevOps organization main page, select **New project**.
2. In the **Create new project** dialog, give your project a name, and then select **Advanced**.
3. Under **Work item process**, drop down and select **AgileDataScienceProcess**, and then select **Create**.

Create new project

Project name *

TDSP Customer Project



Description

Visibility



Public

Anyone on the internet can view the project. Certain features like TFVC are not supported.



Enterprise

[Members of your enterprise](#) can view the project.



Private

Only people you give access to will be able to view this project.

Advanced

Version control

Git

Work item process

Agile

Cancel

Create

Agile

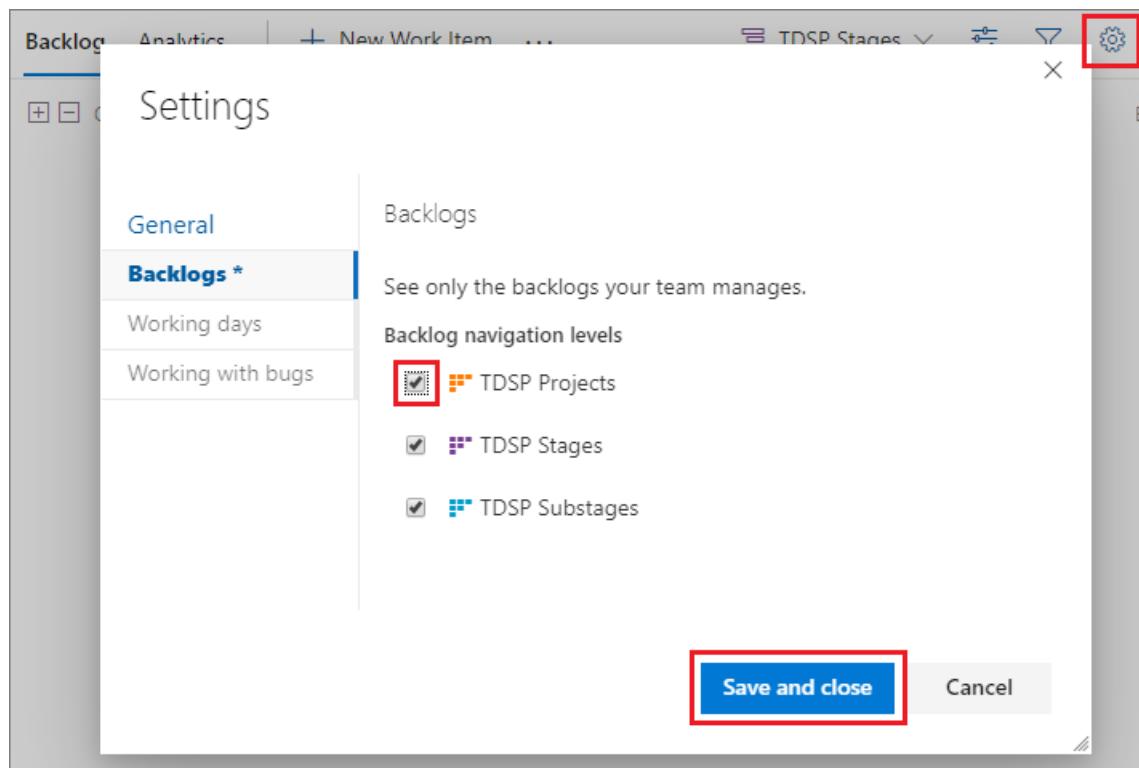
AgileDataScienceProcess

Basic

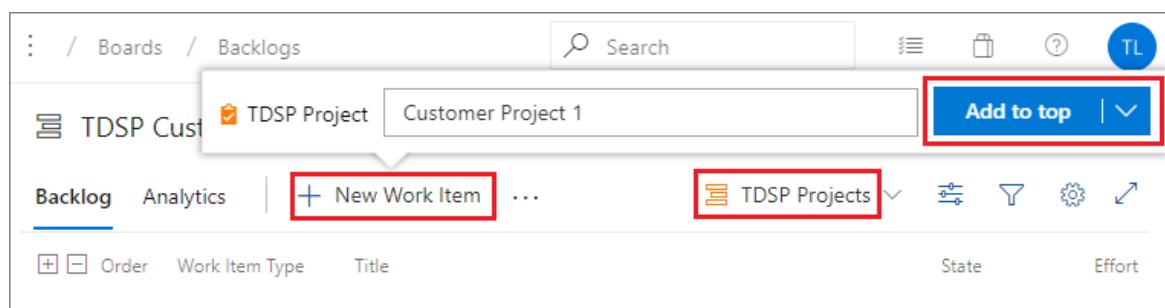
CMMI

Scrum

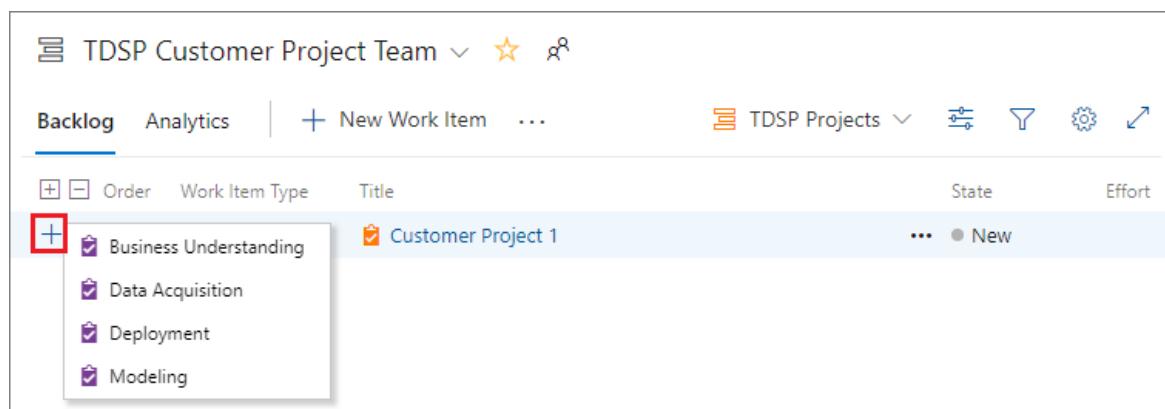
4. In the newly created project, select **Boards > Backlogs** in the left navigation.
5. To make TDSP Projects visible, select the **Configure team settings** icon. In the **Settings** screen, select the **TDSP Projects** check box, and then select **Save and close**.



6. To create a data science-specific TDSP Project, select TDSP Projects in the top bar, and then select **New work item**.
7. In the popup, give the TDSP Project work item a name, and select **Add to top**.



8. To add a work item under the TDSP Project, select the + next to the project, and then select the type of work item to create.



9. Fill in the details in the new work item, and select **Save & Close**.
10. Continue to select the + symbols next to work items to add new TDSP Stages, Substages, and Tasks.

Here is an example of how the data science project work items should appear in **Backlogs** view:

The screenshot shows the backlog for the 'TDSP Customer Project Team'. The backlog is organized into columns: Order, Work Item Type, Title, and State. The backlog items are:

Order	Work Item Type	Title	State
1	TDSP Project	Customer Project 1	New
	Business Understanding	Define objectives	New
	TDSP Substage	Interview customers	New
	TDSP Task	Interview CEO	New
	TDSP Task	Interview CFO	New
	TDSP Substage	Prioritize tasks	New
	Data Acquisition	Get data	New
	TDSP Substage	Establish data connection	New
	TDSP Substage	Set up data store	New
	TDSP Task	Get datastore credentials	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.

[Example walkthroughs](#) lists walkthroughs of specific scenarios, with links and thumbnail descriptions. The linked scenarios illustrate how to combine cloud and on-premises tools and services into workflows or pipelines to create intelligent applications.

Additional resources on agile processes:

- [Agile process](#)
- [Agile process work item types and workflow](#)

Collaborative coding with Git

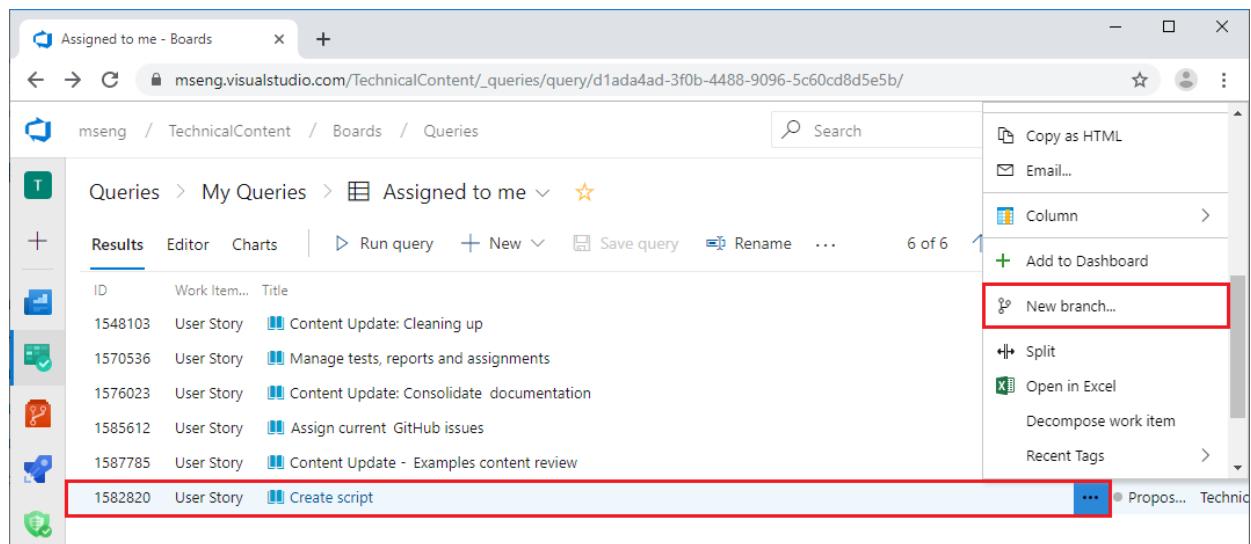
3/5/2021 • 4 minutes to read • [Edit Online](#)

This article describes how to use Git as the collaborative code development framework for data science projects. The article covers how to link code in Azure Repos to [agile development](#) work items in Azure Boards, how to do code reviews, and how to create and merge pull requests for changes.

Link a work item to an Azure Repos branch

Azure DevOps provides a convenient way to connect an Azure Boards User Story or Task work item with an Azure Repos Git repository branch. You can link your User Story or Task directly to the code associated with it.

To connect a work item to a new branch, select the **Actions** ellipsis (...) next to the work item, and on the context menu, scroll to and select **New branch...**.



In the **Create a branch** dialog, provide the new branch name and the base Azure Repos Git repository and branch. The base repository must be in the same Azure DevOps project as the work item. The base branch can be any existing branch. Select **Create branch**.

Create a branch

Name
script

Based on
TechnicalContent
master

Work items to link
Search work items by ID or title
1582820 Create script
Updated 10 minutes ago, • Proposed

Create branch Cancel

You can also create a new branch using the following Git bash command in Windows or Linux:

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

If you don't specify a <base branch name>, the new branch is based on `main`.

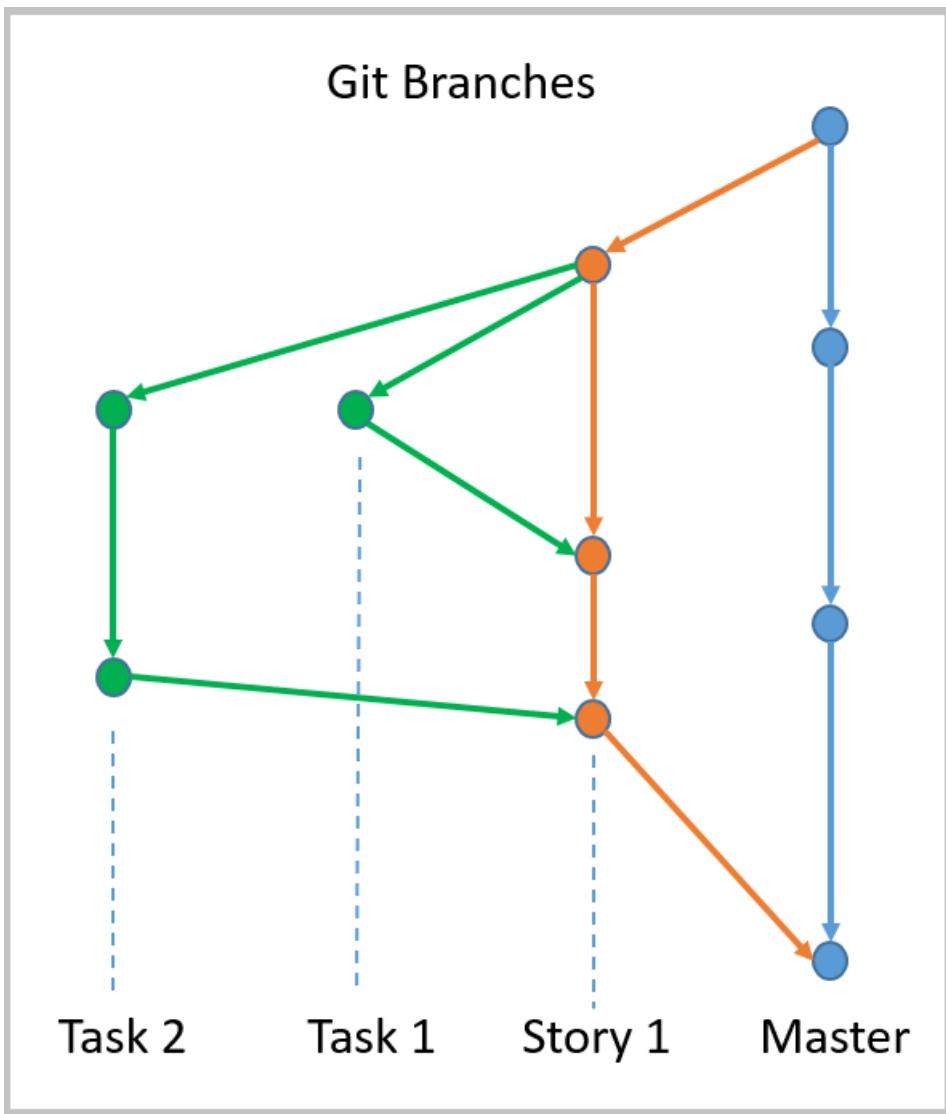
To switch to your working branch, run the following command:

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

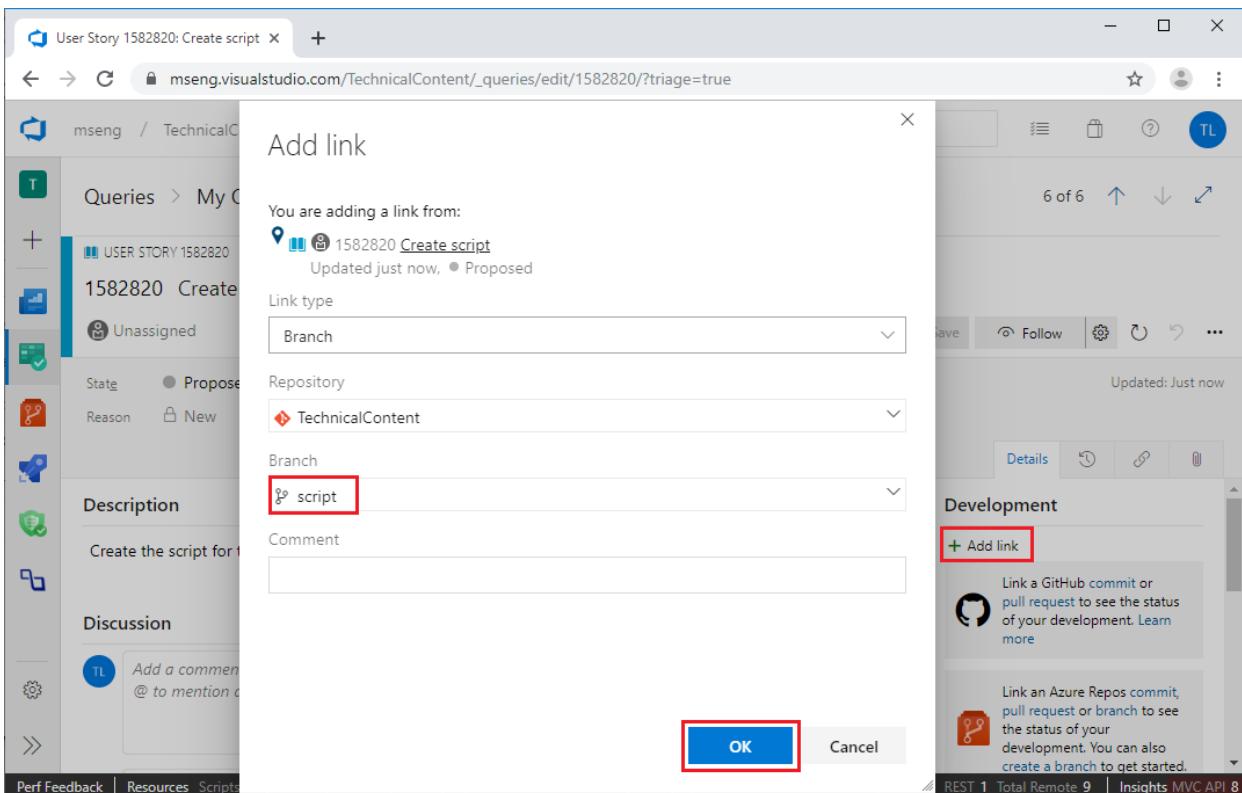
After you switch to the working branch, you can start developing code or documentation artifacts to complete the work item. Running `git checkout main` switches you back to the `main` branch.

It's a good practice to create a Git branch for each User Story work item. Then, for each Task work item, you can create a branch based on the User Story branch. Organize the branches in a hierarchy that corresponds to the User Story-Task relationship when you have multiple people working on different User Stories for the same project, or on different Tasks for the same User Story. You can minimize conflicts by having each team member work on a different branch, or on different code or other artifacts when sharing a branch.

The following diagram shows the recommended branching strategy for TDSP. You might not need as many branches as shown here, especially when only one or two people work on a project, or only one person works on all Tasks of a User Story. But separating the development branch from the primary branch is always a good practice, and can help prevent the release branch from being interrupted by development activities. For a complete description of the Git branch model, see [A Successful Git Branching Model](#).



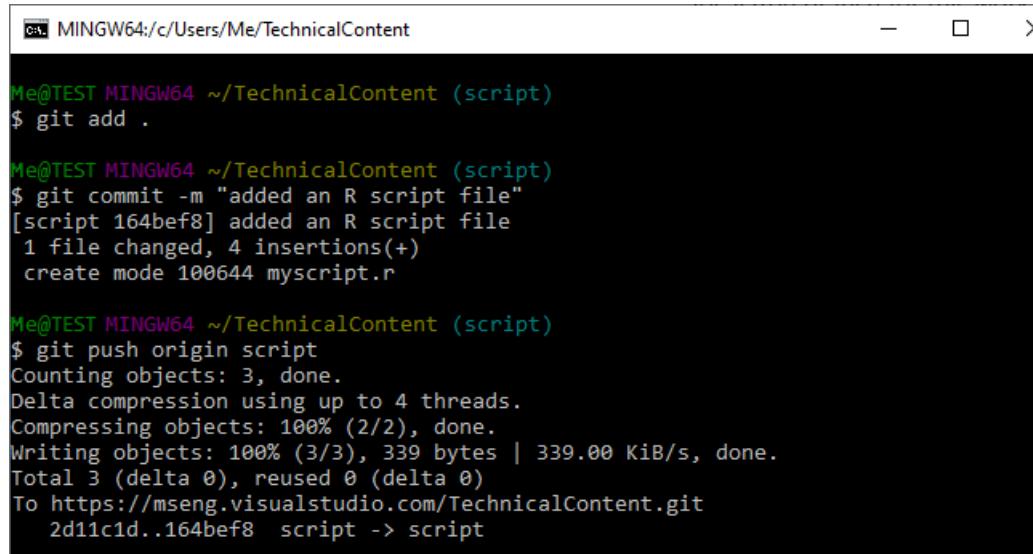
You can also link a work item to an existing branch. On the Detail page of a work item, select **Add link**. Then select an existing branch to link the work item to, and select **OK**.



Work on the branch and commit changes

After you make a change for your work item, such as adding an R script file to your local machine's `script` branch, you can commit the change from your local branch to the upstream working branch by using the following Git bash commands:

```
git status  
git add .  
git commit -m "added an R script file"  
git push origin script
```



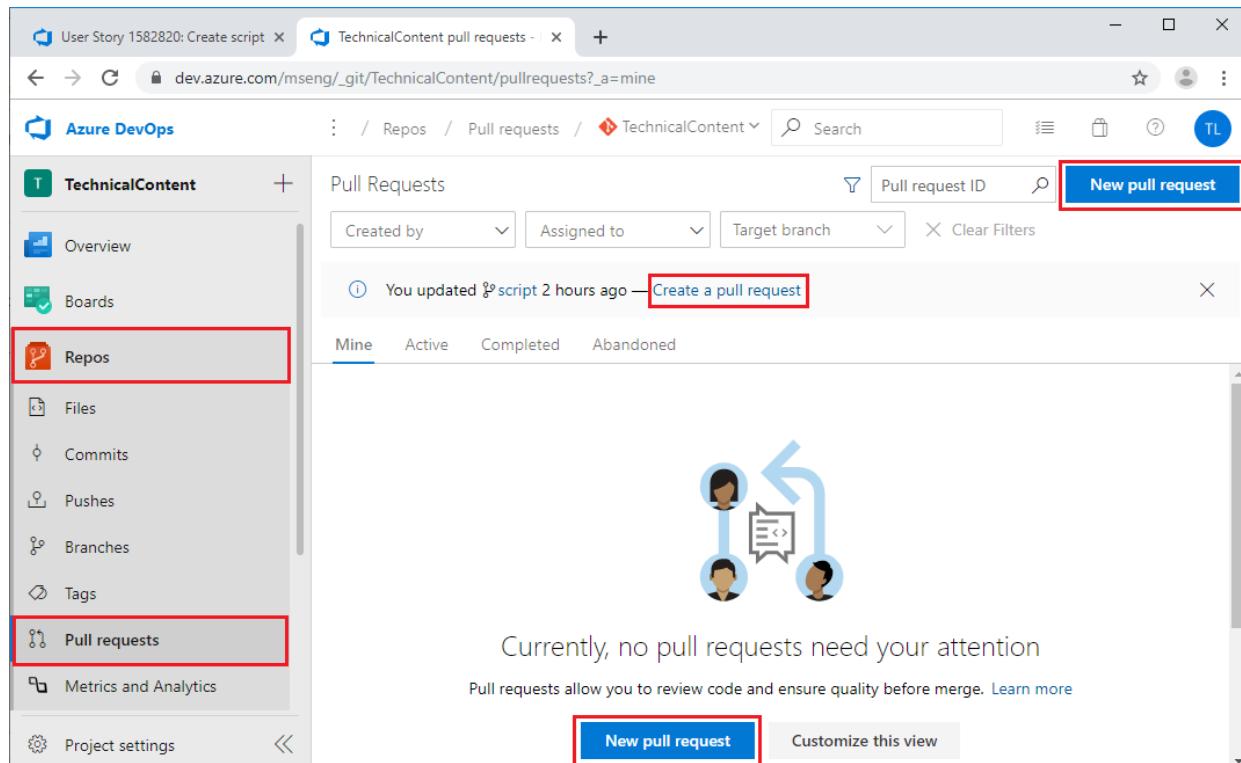
The screenshot shows a terminal window titled "MINGW64:c/Users/Me/TechnicalContent". The command history is as follows:

```
Me@TEST MINGW64 ~/TechnicalContent (script)  
$ git add .  
  
Me@TEST MINGW64 ~/TechnicalContent (script)  
$ git commit -m "added an R script file"  
[script 164bef8] added an R script file  
1 file changed, 4 insertions(+)  
create mode 100644 myscript.r  
  
Me@TEST MINGW64 ~/TechnicalContent (script)  
$ git push origin script  
Counting objects: 3, done.  
Delta compression using up to 4 threads.  
Compressing objects: 100% (2/2), done.  
Writing objects: 100% (3/3), 339 bytes | 339.00 KiB/s, done.  
Total 3 (delta 0), reused 0 (delta 0)  
To https://mseng.visualstudio.com/TechnicalContent.git  
 2d11c1d..164bef8  script -> script
```

Create a pull request

After one or more commits and pushes, when you're ready to merge your current working branch into its base branch, you can create and submit a *pull request* in Azure Repos.

From the main page of your Azure DevOps project, point to **Repos > Pull requests** in the left navigation. Then select either of the **New pull request** buttons, or the **Create a pull request** link.



The screenshot shows the Azure DevOps interface for the "TechnicalContent" repository. The left sidebar is highlighted with red boxes around the "Repos" and "Pull requests" items. The main area shows the "Pull Requests" list with a "New pull request" button highlighted with a red box. A tooltip above the button says "You updated script 2 hours ago — Create a pull request". Below the list, there is a message: "Currently, no pull requests need your attention". At the bottom, there is another "New pull request" button and a "Customize this view" link.

On the **New Pull Request** screen, if necessary, navigate to the Git repository and branch you want to merge your changes into. Add or change any other information you want. Under **Reviewers**, add the names of the reviewers, and then select **Create**.

The screenshot shows the 'Create Pull Request - Repos' page in Azure DevOps. The left sidebar is titled 'TechnicalContent' and includes links for Overview, Boards, Repos, Files, Commits, Pushes, Branches, Tags, Pull requests, Metrics and Analytics, Push Notifications, Pipelines, Compliance, and Project settings. The 'Pull requests' link is currently selected. The main area is titled 'New Pull Request' and shows the following fields:

- From dropdown: 'script' (selected), To dropdown: 'master'.
- Title: 'Added myscript.r'.
- Description: 'Added myscript.r'. A note below says 'Markdown supported.'
- Reviewers: A search bar containing 'Admin' and '[Documentation]\Test'. A red box highlights the 'Reviewers' label.
- Work Items: A search bar containing '1582820 Create script'. A red box highlights the work item entry.
- Bottom right: A blue 'Create' button with a downward arrow, also highlighted with a red box.

Review and merge

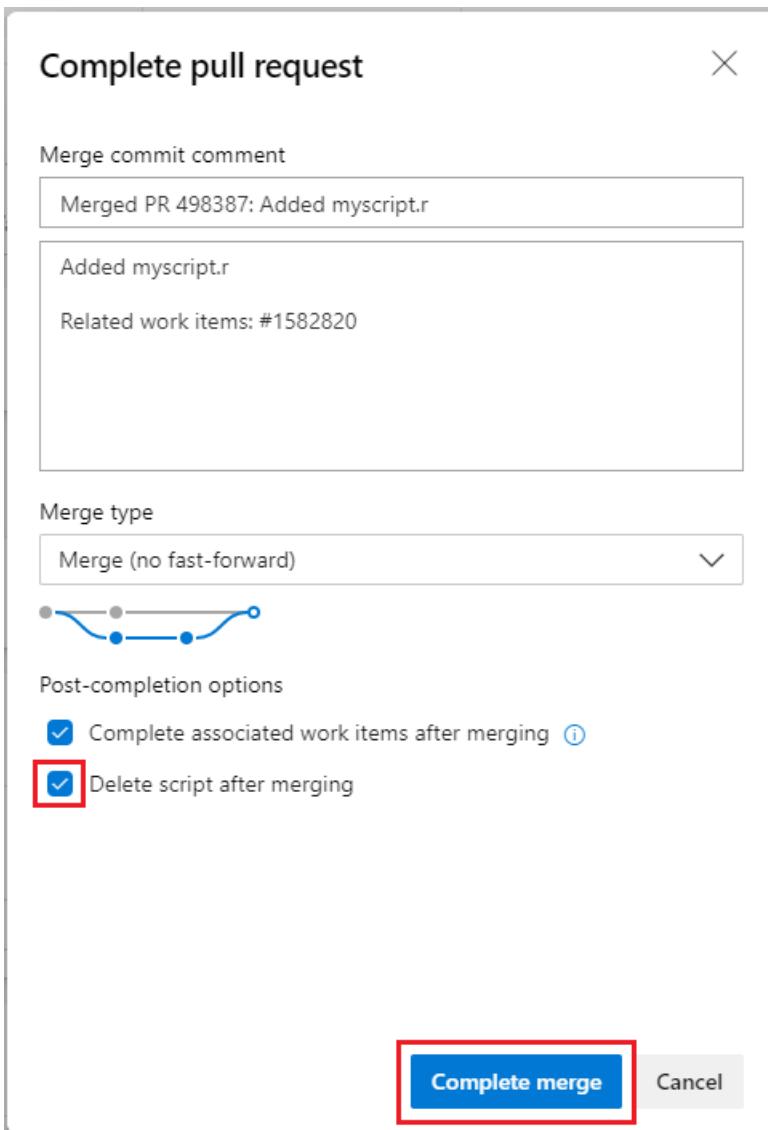
Once you create the pull request, your reviewers get an email notification to review the pull request. The reviewers test whether the changes work, and check the changes with the requester if possible. The reviewers can make comments, request changes, and approve or reject the pull request based on their assessment.

The screenshot shows the Azure DevOps interface for a pull request titled "Added myscript.r". The pull request is active and has been tested and approved by "script" into the "master" branch. The "Description" field contains the text "Added myscript.r". There are two comments listed:

- A comment from "Tester" 2 minutes ago: "I need to review this". The status is "Resolved".
- A comment from "Test" just now: "OK.". A reply box is open for "Tester".

On the right side, there is a feedback menu with options: Approve, Approve with suggestions, Wait for author, Reject, and Reset feedback. The "Approve" option is selected. A "Labels" section is also visible.

After the reviewers approve the changes, you or someone else with merge permissions can merge the working branch to its base branch. Select **Complete**, and then select **Complete merge** in the **Complete pull request** dialog. You can choose to delete the working branch after it has merged.



Confirm that the request is marked as COMPLETED.

498387 **COMPLETED** Added myscript.r

Test script into master

Overview Files Updates Commits Additional Validations Conflicts

Delete source branch

When you go back to **Repos** in the left navigation, you can see that you've been switched to the main branch since the **script** branch was deleted.

User Story 1582820: Create script X TechnicalContent - Repos +

dev.azure.com/mseng/_git/TechnicalContent

Azure DevOps / Repos / Files / TechnicalContent Type to find a file or folder... Search Fork Clone

TechnicalContent +

Overview Boards Repos Files Commits Pushes

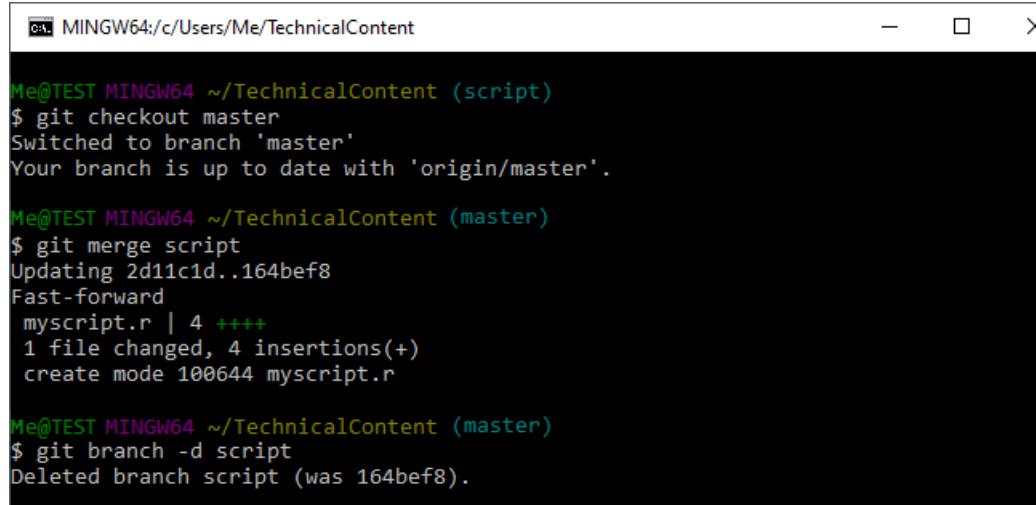
master TechnicalContent / Type to find a file or folder...

Branch script was deleted so we've switched you to master, the default branch.

Name ↑	Last change	Commits
myscript.r	3 hours ago	67a2baac
README.md	8/12/2019	f954d0c1
TeamList.md	8/16/2019	882421ed

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

```
git checkout main
git merge script
git branch -d script
```



```
Me@TEST MINGW64 ~/TechnicalContent (script)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

Me@TEST MINGW64 ~/TechnicalContent (master)
$ git merge script
Updating 2d11c1d..164bef8
Fast-forward
 myscript.r | 4 +++
 1 file changed, 4 insertions(+)
 create mode 100644 myscript.r

Me@TEST MINGW64 ~/TechnicalContent (master)
$ git branch -d script
Deleted branch script (was 164bef8).
```

Next steps

[Execute 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.

[Example walkthroughs](#) lists walkthroughs of specific scenarios, with links and thumbnail descriptions. The linked scenarios illustrate how to combine cloud and on-premises tools and services into workflows or pipelines to create intelligent applications.

Execute data science tasks: exploration, modeling, and deployment

3/5/2021 • 2 minutes to read • [Edit Online](#)

Typical data science tasks include data exploration, modeling, and deployment. This article outlines the tasks to complete several common data science tasks such as interactive data exploration, data analysis, reporting, and model creation. Options for deploying a model into a production environment may include:

- [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 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.

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.

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-server 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.

[Model operation and CI/CD](#) shows how CI/CD can be performed with developed models.

Data science code testing on Azure with the Team Data Science Process and Azure DevOps Services

3/10/2021 • 4 minutes to read • [Edit Online](#)

This article gives preliminary guidelines for testing code in a data science workflow. Such testing gives data scientists a systematic and efficient way to check the quality and expected outcome of their code. We use a Team Data Science Process (TDSP) [project that uses the UCI Adult Income dataset](#) that we published earlier to show how code testing can be done.

Introduction on code testing

"Unit testing" is a longstanding practice for software development. But for data science, it's often not clear what "unit testing" means and how you should test code for different stages of a data science lifecycle, such as:

- Data preparation
- Data quality examination
- Modeling
- Model deployment

This article replaces the term "unit testing" with "code testing." It refers to testing as the functions that help to assess if code for a certain step of a data science lifecycle is producing results "as expected." The person who's writing the test defines what's "as expected," depending on the outcome of the function--for example, data quality check or modeling.

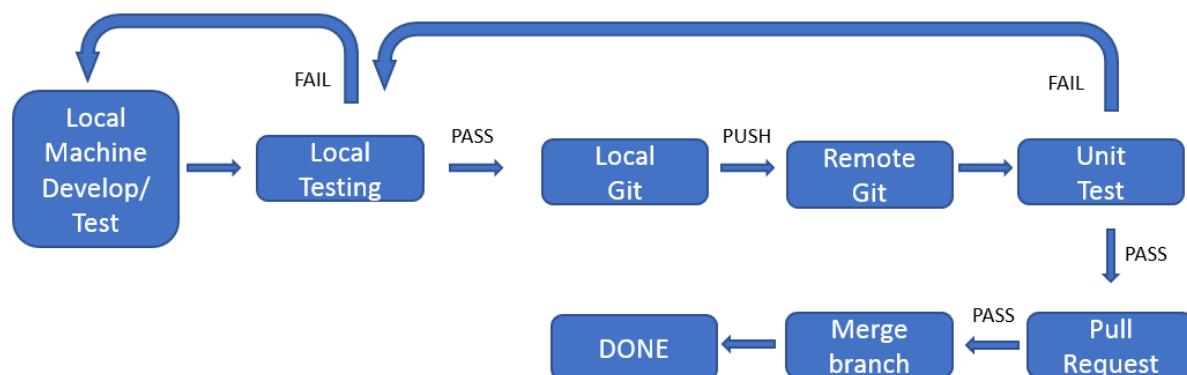
This article provides references as useful resources.

Azure DevOps for the testing framework

This article describes how to perform and automate testing by using Azure DevOps. You might decide to use alternative tools. We also show how to set up an automatic build by using Azure DevOps and build agents. For build agents, we use Azure Data Science Virtual Machines (DSVMs).

Flow of code testing

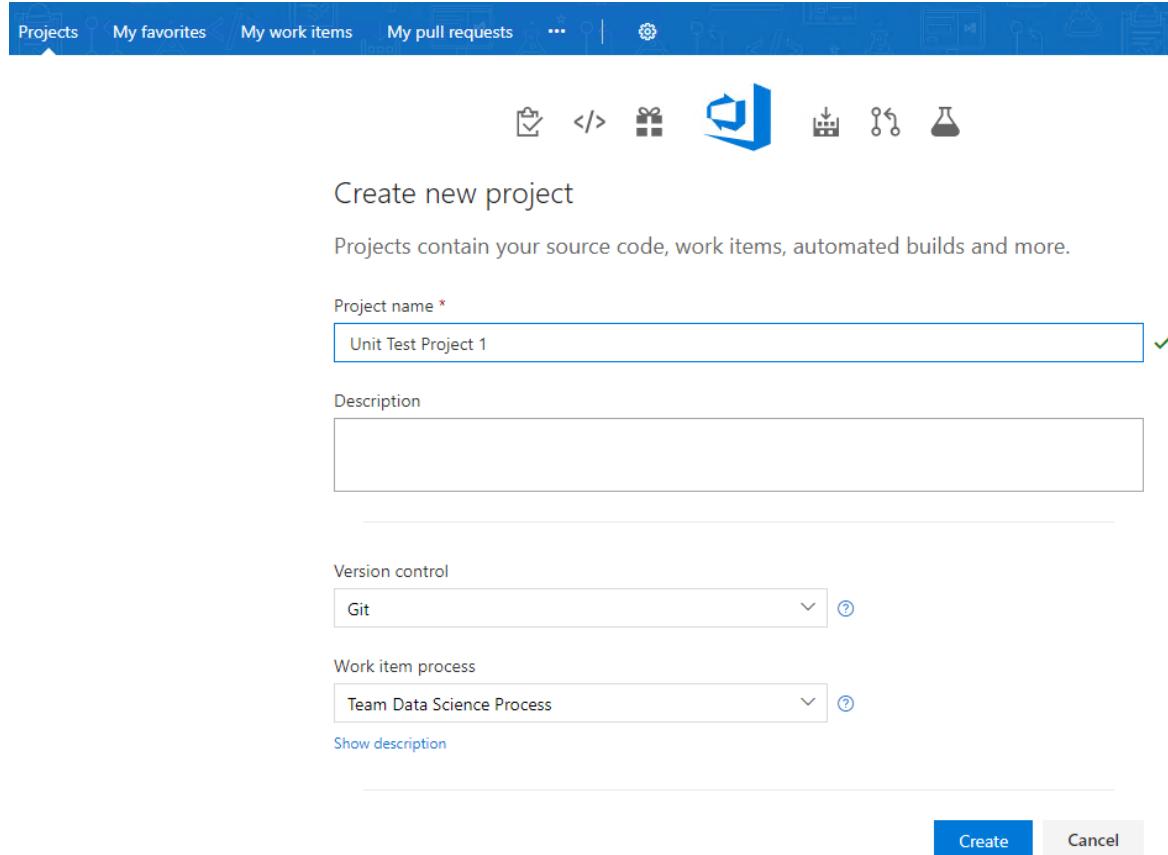
The overall workflow of testing code in a data science project looks like this:



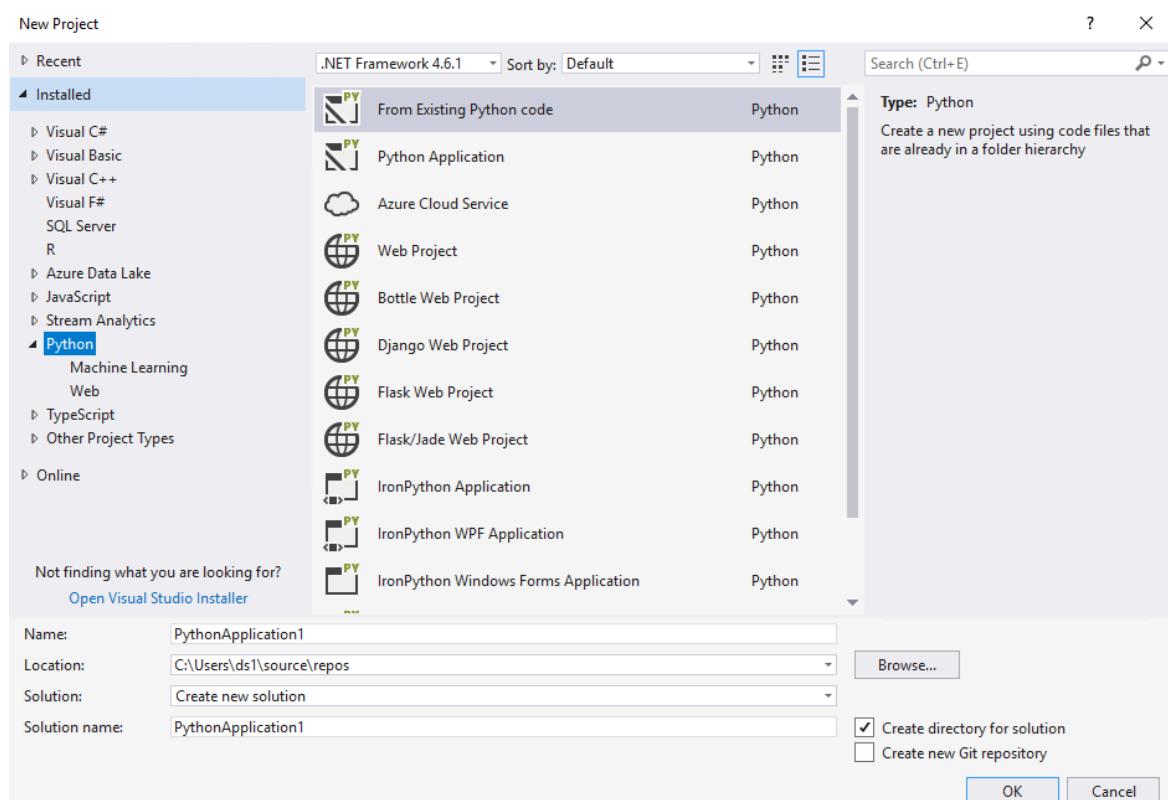
Detailed steps

Use the following steps to set up and run code testing and an automated build by using a build agent and Azure DevOps:

1. Create a project in the Visual Studio desktop application:



After you create your project, you'll find it in Solution Explorer in the right pane:



The screenshot shows the Microsoft Visual Studio Code interface. On the left, the code editor displays `test_funcs.py` with several functions for data validation. On the right, the Solution Explorer shows a project named `unit_test_1` containing files like `data_ingestion.py`, `test_funcs.py`, and `test1.py`.

```

1 #class check_data(object):
2 #     """description of class"""
3 import pandas as pd
4 import pickle
5 import numpy as np
6
7
8 def checkColumnNames(csv_file, required_columns):
9     df = pd.read_csv(csv_file)
10    if set(df.columns) == set(required_columns):
11        print("Columns match!")
12    else:
13        print("Columns do not matched!")
14    return set(df.columns) == set(required_columns)
15
16 def checkResponseLevels(csv_file, response_column):
17     df = pd.read_csv(csv_file)
18     levels=list(df[response_column].unique())
19     if levels == [0,1]:
20         print("Levels match!")
21     else:
22         print("Levels do not match!")
23     return levels == [0,1]
24
25 def checkResponsePercent(csv_file, response_column):
26     df = pd.read_csv(csv_file)

```

2. Feed your project code into the Azure DevOps project code repository:

The screenshot shows the Azure DevOps Repository interface. It displays the `Unit_Test_UCI_Income` repository with the `master` branch selected. The history shows several commits, including one from `wei` adding testing for model prediction.

Name	Last change	Commits
<code>__pycache__</code>	3/14/2018	7785cf02 modified for py35 wei
<code>.vs</code>	3/14/2018	7785cf02 modified for py35 wei
<code>adult_income_model.pkl</code>	3/14/2018	ecf9822b add testing for model prediction wei
<code>adult_income_to_score.csv</code>	3/14/2018	0fcfa1fc7 add testing for model prediction wei
<code>data_ingestion.py</code>	1/11/2018	254c6c6f put some code wei
<code>test1.py</code>		
<code>test1.nvc</code>		

3. Suppose you've done some data preparation work, such as data ingestion, feature engineering, and creating label columns. You want to make sure your code is generating the results that you expect. Here's some code that you can use to test whether the data-processing code is working properly:

- Check that column names are right:

```

def checkColumnNames(csv_file, required_columns):
    df = pd.read_csv(csv_file)
    if set(df.columns) == set(required_columns):
        print("Columns match!")
    else:
        print("Columns do not matched!")
    return set(df.columns) == set(required_columns)

```

- Check that response levels are right:

```

def checkResponseLevels(csv_file, response_column):
    df = pd.read_csv(csv_file)
    levels=list(df[response_column].unique())
    if levels == [0,1]:
        print("Levels match!")
    else:
        print("Levels do not match!")
    return levels == [0,1]

```

- Check that response percentage is reasonable:

```

def checkResponsePercent(csv_file,response_column):
    df = pd.read_csv(csv_file)
    ratio1 = df.loc[df[response_column] == 1].shape[0]/df.shape[0]
    if ratio1 > 0.5:
        print("Response levels(0/1) are messed up!")
    else:
        print("Response 0/1 are OK, and OK, Happy Friday!")
    return ratio1 < 0.5

```

- Check the missing rate of each column in the data:

```

def checkMissingRate2(csv_file, threshold):
    df = pd.read_csv(csv_file)
    for x in df.columns:
        miss_rate = df[x].isnull().sum()/df.shape[0]
        if miss_rate > threshold:
            print("{} has more than {} missing values!".format(x,threshold))
            return False
    print("No column has missing values more than {}!".format(threshold))
    return True

```

4. After you've done the data processing and feature engineering work, and you've trained a good model, make sure that the model you trained can score new datasets correctly. You can use the following two tests to check the prediction levels and distribution of label values:

- Check prediction levels:

```

def checkPredictionLevels(csv_file, model_file):
    df = pd.read_csv(csv_file)
    X_to_score = df[['education_num','age','hours_per_week']].values
    loaded_model = pickle.load(open(model_file, 'rb'))
    y_hat = loaded_model.predict(X_to_score)
    if list(set(y_hat)) == [0,1]:
        print("Prediction Levels match!")
    else:
        print("Prediction Levels do not match!")
    return list(set(y_hat)) == [0,1]

```

- Check the distribution of prediction values:

```

def checkPredictionPercent(csv_file,model_file):
    df = pd.read_csv(csv_file)
    X_to_score = df[['education_num','age','hours_per_week']].values
    loaded_model = pickle.load(open(model_file, 'rb'))
    y_hat = loaded_model.predict(X_to_score)
    ratio1 = sum(y_hat == 1)/len(y_hat)
    if ratio1 > 0.5:
        print("Response levels(0/1) are messed up!")
    else:
        print("Response 0/1 percent is OK, Happy prediction!")
    return ratio1 < 0.5

```

5. Put all test functions together into a Python script called `test_funcs.py`:

```

test_funcs.py ✘ X

1  #class check_data(object):
2  #    """description of class"""
3  import pandas as pd
4  import pickle
5  import numpy as np
6
7
8  def checkColumnNames(csv_file, required_columns):
9      df = pd.read_csv(csv_file)
10     if set(df.columns) == set(required_columns):
11         print("Columns match!")
12     else:
13         print("Columns do not matched!")
14     return set(df.columns) == set(required_columns)
15
16  def checkResponseLevels(csv_file, response_column):
17      df = pd.read_csv(csv_file)
18      levels=list(df[response_column].unique())
19      if levels == [0,1]:
20          print("Levels match!")
21      else:
22          print("Levels do not match!")
23      return levels == [0,1]
24
25  def checkResponsePercent(csv_file, response_column):
26      df = pd.read_csv(csv_file)
27      ratio1 = df.loc[df[response_column] == 1].shape[0]/df.shape[0]
28      if ratio1 > 0.5:
29          print("Response levels(0/1) are messed up!")
30      else:
31          print("Response 0/1 are OK, and OK, Happy Friday!")
32      return ratio1 < 0.5
33

```

6. After the test codes are prepared, you can set up the testing environment in Visual Studio.

Create a Python file called **test1.py**. In this file, create a class that includes all the tests you want to do.

The following example shows six tests prepared:

```

6  class Test_A(unittest.TestCase):
7
8      def checkColumnNames(self):
9          assert checkColumnNames(file_name, required_columns) == True
10
11     def checkResponseLevels(self):
12         assert checkResponseLevels(file_name, response_column) == True
13
14     def checkResponsePercent(self):
15         assert checkResponsePercent(file_name, response_column) == True
16
17     def checkMissingRate(self):
18         assert checkMissingRate2(file_name, threshold) == True
19
20     def checkPredictionLevels(self):
21         assert checkPredictionLevels(score_file_name, model_file_name) == True
22
23     def checkPredictionPercent(self):
24         assert checkPredictionPercent(score_file_name, model_file_name) == True
~~

```

7. Those tests can be automatically discovered if you put **codetest.testCase** after your class name. Open Test Explorer in the right pane, and select **Run All**. All the tests will run sequentially and will tell you if the test is successful or not.

The screenshot shows the Visual Studio Code interface with the following details:

- Test Explorer:** On the left, it lists test cases categorized by execution time:
 - Fast < 100 ms (1):** test_checkPredictionLevels < 1 ms
 - Not Run (5):** test_checkColumnNames, test_checkMissingRate, test_checkPredictionPercent, test_checkResponseLevels, test_checkResponsePercent
- Code Editor:** The main window displays the Python file `test1.py` with the following content:

```
import unittest
import pandas as pd
from test_funcs import *

class UCI_unit_test(unittest.TestCase):
    def test_checkColumnNames(self):
        assert checkColumnNames()

    def test_checkResponseLevel(self):
        assert checkResponseLevel()

    def test_checkResponsePerce(self):
        assert checkResponsePerce()

    def test_checkMissingRate(self):
        assert checkMissingRate()

    def test_checkPredictionLev(self):
        assert checkPredictionLev()

    def test_checkPredictionPer(self):
        assert checkPredictionPer()
```

8. Check in your code to the project repository by using Git commands. Your most recent work will be reflected shortly in Azure DevOps.

```
PS C:\Unit_Test_UCI_Income> git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   .vs/unit_test_1/v15.suo
        modified:   __pycache__/_test_funcs.cpython-35.pyc
        modified:   test1.py
        modified:   test1.pyc
        modified:   test_funcs.py
        modified:   unit_test.pyproj

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

    __pycache__/_test1.cpython-35.pyc
    __pycache__/_test2.cpython-35.pyc
    __pycache__/_test_model.cpython-35.pyc

no changes added to commit (use "git add" and/or "git commit -a")
PS C:\Unit_Test_UCI_Income> git add .
PS C:\Unit_Test_UCI_Income> git commit -m"revised code wei"
[master 60841e1] revised code wei
 9 files changed, 7 insertions(+), 19 deletions(-)
 rewrite .vs/unit_test_1/v15.suo (63%)
 create mode 100644 __pycache__/_test1.cpython-35.pyc
 create mode 100644 __pycache__/_test2.cpython-35.pyc
 create mode 100644 __pycache__/_test_model.cpython-35.pyc
 rewrite test1.pyc (64%)
PS C:\Unit_Test_UCI_Income> git push
Counting objects: 15, done.
Delta compression using up to 24 threads.
Compressing objects: 100% (13/13), done.
Writing objects: 100% (15/15) 6.16 KiB | 2.05 MiB/s, done.
Total 15 (delta 6), reused 0 (delta 0)
remote: Analyzing objects... (15/15) (26 ms)
remote: Storing packfile... done (213 ms)
remote: Storing index... done (32 ms)
To https://dg-ads.visualstudio.com/_git/Unit_Test_UCI_Income
 7785cf0..60841e1 master -> master
PS C:\Unit_Test_UCI_Income> _
```

Unit_Test_UCI_Income / Type to find a file or folder...				
		Contents	History	
		Name	Last change ↓	Commits
e_		.vs	2 minutes ago	60841e16 revised code wei wei
ome_model.pkl		__pycache__	2 minutes ago	60841e16 revised code wei wei
income_to_score.csv		unit_test.pyproj	2 minutes ago	60841e16 revised code wei wei
estion.py		test1.py	2 minutes ago	60841e16 revised code wei wei
		test1.pyc	2 minutes ago	60841e16 revised code wei wei
		test_funcs.py	2 minutes ago	60841e16 revised code wei wei
			...	

9. Set up automatic build and test in Azure DevOps:

- a. In the project repository, select **Build and Release**, and then select **+ New** to create a new build process.

The screenshot shows the Azure DevOps interface with the 'Build and Release' tab selected. Below it, the 'Definitions' tab is active. At the top right, there are buttons for '+ New', '+ Import', and 'Security'. Below the tabs, there are two buttons: 'Definitions' and 'Queued'.

- b. Follow the prompts to select your source code location, project name, repository, and branch information.

The screenshot shows the 'Select a source' step in the build setup wizard. It includes a grid of source control icons: VSTS Git (selected), TFVC, GitHub, GitHub Enterprise, Subversion, Bitbucket Cloud, and External Git. Below the grid, there are dropdown menus for 'Team project' (set to 'Unit_Test_UCI_Income'), 'Repository' (set to 'Unit_Test_UCI_Income'), and 'Default branch for manual and scheduled builds' (set to 'master'). At the bottom is a 'Continue' button.

- c. Select a template. Because there's no Python project template, start by selecting **Empty process**.

Select a template

Or start with an  [Empty process](#)



Azure Service Fabric Application with Docker Support

Build and package an Azure Service Fabric application that contains Docker images to be pushed to a Docker registry.



Azure Web App for Java

Build your Java WAR file and deploy it directly to an Azure Web App.



Container

Build an image and push to Docker or Azure Container registry.

d. Name the build and select the agent. You can choose the default here if you want to use a DSVM to complete the build process. For more information about setting agents, see [Build and release agents](#).

Agent phase 

X Remove

Display name  *

Phase 1

Agent selection 

Agent queue  | [Manage](#) 

<inherit from definition>  

Demands 

Name	Condition	Value
DotNetFramework	exists	

+ Add

Execution plan 

Parallelism 

None Multi-configuration Multi-agent

Timeout  *

0

Additional options 

Allow scripts to access OAuth token 

Run this phase 

Only when all previous phases have succeeded 

e. Select + in the left pane, to add a task for this build phase. Because we're going to run the Python script **test1.py** to complete all the checks, this task is using a PowerShell command to run Python code.

The screenshot shows the Azure DevOps pipeline editor. At the top, there are tabs for Tasks, Variables, Triggers, Options, Retention, and History. The Tasks tab is selected. In the main area, there's a sidebar with sections for Process (Build process), Get sources (Unit_Test_UCI_Income, master), and Phase 1 (Run on agent). Below this, a search bar with the placeholder 'Add tasks' and a refresh button is followed by a search input field containing 'powershell'. A search result for 'PowerShell' is displayed, showing its icon (a blue PowerShell logo), name, description ('Run a PowerShell script'), provider ('by Microsoft Corporation'), and an 'Add' button. Another result for 'Service Fabric PowerShell' is partially visible below it.

f. In the PowerShell details, fill in the required information, such as the name and version of PowerShell. Choose **Inline Script** as the type.

In the box under **Inline Script**, you can type `python test1.py`. Make sure the environment variable is set up correctly for Python. If you need a different version or kernel of Python, you can explicitly specify the path as shown in the figure:

The screenshot shows the Azure DevOps pipeline editor with the Tasks tab selected. The pipeline structure is identical to the previous screenshot. The 'PowerShell Script' task in 'Phase 1' is selected, revealing its configuration details. The 'PowerShell' task has a 'Version' dropdown set to '1.*'. The 'Display name' is 'PowerShell Script'. The 'Type' is set to 'Inline Script'. The 'Arguments' field is empty. The 'Inline Script' field contains the following code:

```
# You can write your powershell scripts inline here.  
# You can also pass predefined and custom variables to this scripts using arguments  
  
Write-Host "Hello World"  
#python test1.py  
C:\Anaconda\envs\py35\python.exe ./test1.py
```

g. Select **Save & queue** to complete the build pipeline process.

ment Groups*



Save & queue ...

History

PowerShell ⓘ

Link settings

Remove

Version 1.*

Display name *

PowerShell Script

Type * ⓘ

Inline Script

Arguments ⓘ

Inline Script * ⓘ

Now every time a new commit is pushed to the code repository, the build process will start automatically. You can define any branch. The process runs the `test1.py` file in the agent machine to make sure that everything defined in the code runs correctly.

If alerts are set up correctly, you'll be notified in email when the build is finished. You can also check the build status in Azure DevOps. If it fails, you can check the details of the build and find out which piece is broken.

Reply Reply All Forward IM

Wed 3/21/2018 10:51 AM



Visual Studio Team Services

Unit_Test_UCI_Income Build 47 succeeded

To •

47 - Succeeded

[Open Build Report in Web Access](#)

Continuous Integration Build of Unit_Test_UCI_Income-CI (2) (Unit_Test_UCI_Income)
Ran for 0.3 minutes (Default), completed at Wed 03/21/2018 05:50 PM

Request Summary

Request 47

Completed

Summary

| Finalize build

0 error(s), 0 warning(s)

| Phase 1

0 error(s), 0 warning(s)

Notes:

- All dates and times are shown in UTC

We sent you this notification due to a default subscription | [Unsubscribe](#) | [View](#)

Provided by [Microsoft Visual Studio® Team Foundation Server](#)

Builds Releases Library Task Groups Deployment Groups*

✓ Build 47

✓ Phase 1

- ✓ Job
 - ✓ Initialize Job
 - ✓ Get Sources
 - ✓ PowerShell Script
 - ✓ Post Job Cleanup
- ✓ Finalize build
- ✓ Report build status

Unit_Test_UCI_Income-CI (2) / Build 47

Edit build definition Queue new build... ▾

Build succeeded

Build 47 ↗
Ran for 16 seconds (Default), com...

Summary Timeline Code coverage* Tests

Build details

Definition	Unit_Test_UCI_Income-CI (2) (edit)
Source	master
Source version	Commit 60841e16
Requested by	Microsoft.VisualStudio.Services.TFS on
Queue name	Default
Queued	Wednesday, March 21, 2018 5:50 PM
Started	Wednesday, March 21, 2018 5:50 PM
Finished	Wednesday, March 21, 2018 5:50 PM
Retained state	Build not retained

Next steps

- See the [UCI income prediction repository](#) for concrete examples of unit tests for data science scenarios.
- Follow the preceding outline and examples from the UCI income prediction scenario in your own data science projects.

References

- [Team Data Science Process](#)
- [Visual Studio Testing Tools](#)
- [Azure DevOps Testing Resources](#)
- [Data Science Virtual Machines](#)

Track the progress of data science projects

11/2/2020 • 2 minutes to read • [Edit Online](#)

Data science group managers, team leads, and project leads can track the progress of their projects. Managers want to know what work has been done, who did the work, and what work remains. Managing expectations is an important element of success.

Azure DevOps dashboards

If you're using Azure DevOps, you can build dashboards to track the activities and work items associated with a given Agile project. For more information about dashboards, see [Dashboards, reports, and widgets](#).

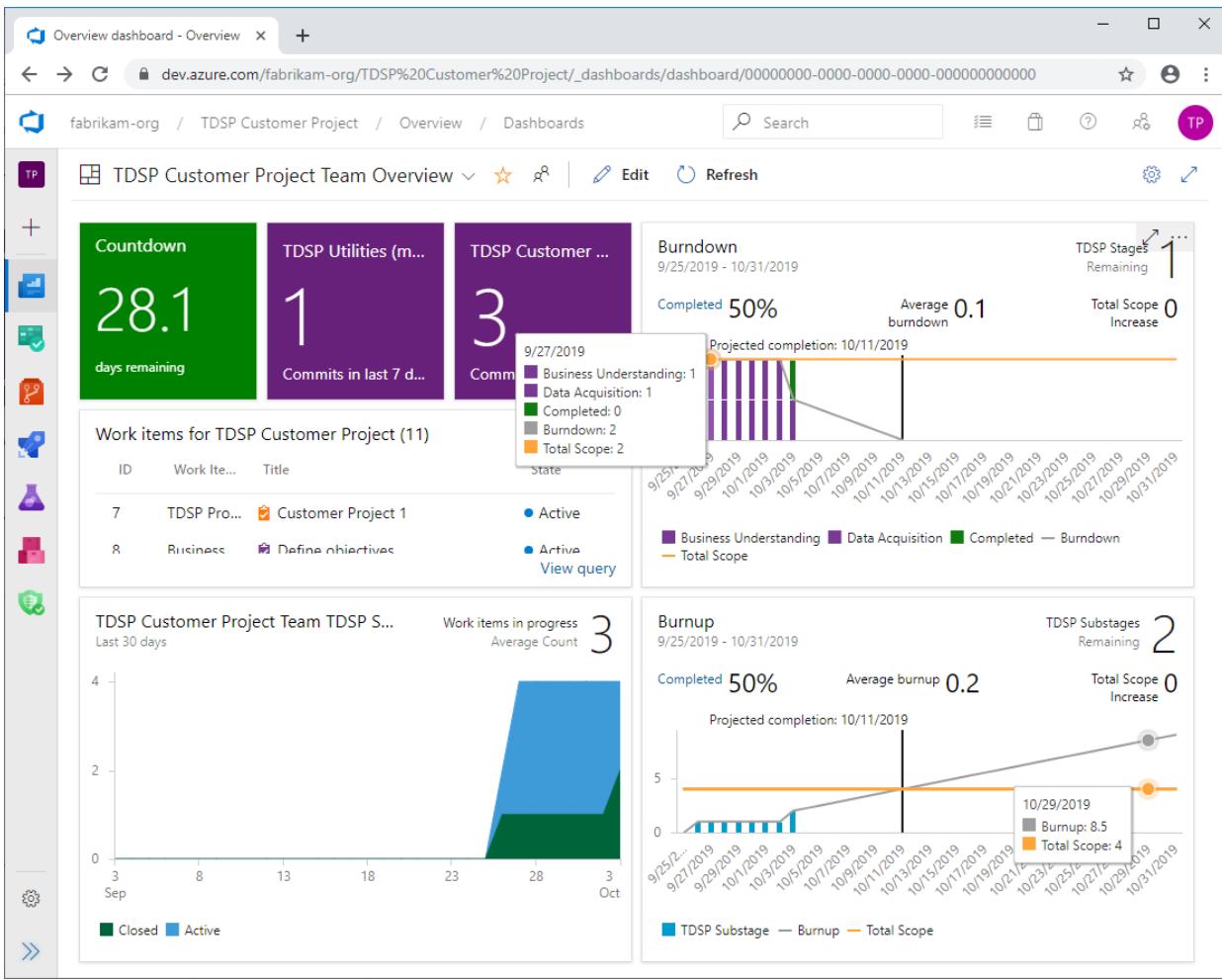
For instructions on how to create and customize dashboards and widgets in Azure DevOps, see the following quickstarts:

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

Example dashboard

Here is a simple example dashboard that tracks the sprint activities of an Agile data science project, including the number of commits to associated repositories.

- The **countdown** tile shows the number of days that remain in the current sprint.
- The two **code tiles** show the number of commits in the two project repositories for the past seven days.
- **Work items for TDSP Customer Project** shows the results of a query for all work items and their status.
- A **cumulative flow diagram** (CFD) shows the number of Closed and Active work items.
- The **burndown chart** shows work still to complete against remaining time in the sprint.
- The **burnup chart** shows completed work compared to total amount of work in the sprint.



Next steps

[Walkthroughs executing the Team Data Science Process](#) lists walkthroughs that demonstrate all the process steps. The linked scenarios illustrate how to manage the cloud and on-premise resources into intelligent applications.

Create CI/CD pipelines for AI apps using Azure Pipelines, Docker, and Kubernetes

3/5/2021 • 2 minutes to read • [Edit Online](#)

An Artificial Intelligence (AI) application is application code embedded with a pretrained machine learning (ML) model. There are always two streams of work for an AI application: Data scientists build the ML model, and app developers build the app and expose it to end users to consume. This article describes how to implement a continuous integration and continuous delivery (CI/CD) pipeline for an AI application that embeds the ML model into the app source code. The sample code and tutorial use a Python Flask web application, and fetch a pretrained model from a private Azure blob storage account. You could also use an AWS S3 storage account.

NOTE

The following process is one of several ways to do CI/CD. There are alternatives to this tooling and the prerequisites.

Source code, tutorial, and prerequisites

You can download [source code](#) and a [detailed tutorial](#) from GitHub. Follow the tutorial steps to implement a CI/CD pipeline for your own application.

To use the downloaded source code and tutorial, you need the following prerequisites:

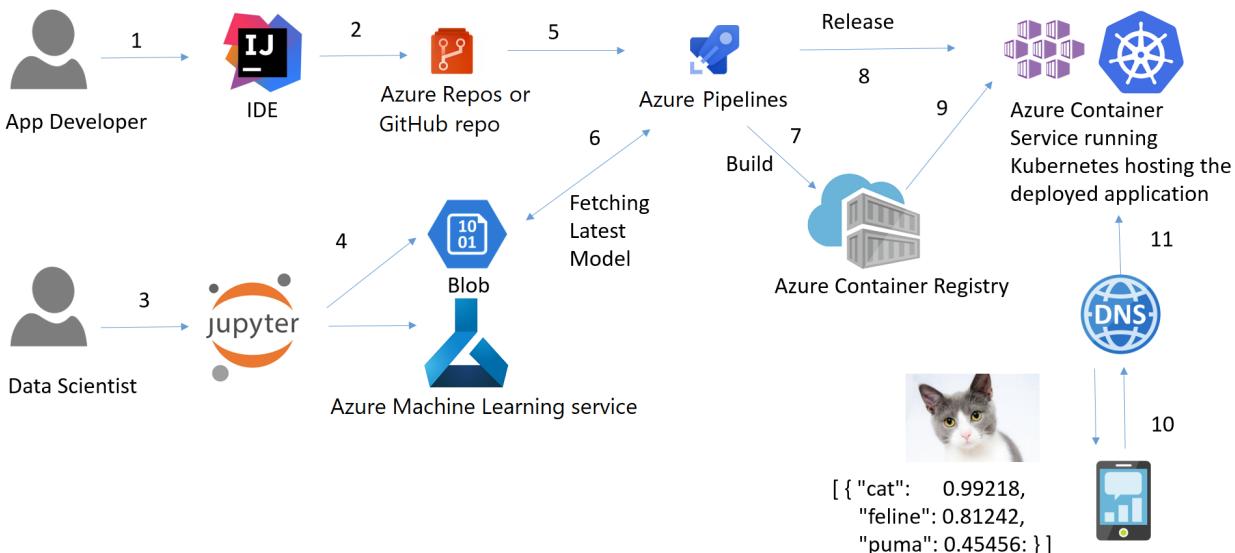
- The [source code repository](#) forked to your GitHub account
- An [Azure DevOps Organization](#)
- [Azure CLI](#)
- An [Azure Container Service for Kubernetes \(AKS\) cluster](#)
- [Kubectl](#) to run commands and fetch configuration from the AKS cluster
- An [Azure Container Registry \(ACR\) account](#)

CI/CD pipeline summary

Each new Git commit kicks off the Build pipeline. The build securely pulls the latest ML model from a blob storage account, and packages it with the app code in a single container. This decoupling of the application development and data science workstreams ensures that the production app is always running the latest code with the latest ML model. If the app passes testing, the pipeline securely stores the build image in a Docker container in ACR. The release pipeline then deploys the container using AKS.

CI/CD pipeline steps

The following diagram and steps describe the CI/CD pipeline architecture:



1. Developers work on the application code in the IDE of their choice.
2. The developers commit the code to Azure Repos, GitHub, or other Git source control provider.
3. Separately, data scientists work on developing their ML model.
4. The data scientists publish the finished model to a model repository, in this case a blob storage account.
5. Azure Pipelines kicks off a build based on the Git commit.
6. The Build pipeline pulls the latest ML model from blob storage and creates a container.
7. The pipeline pushes the build image to the private image repository in ACR.
8. The Release pipeline kicks off based on the successful build.
9. The pipeline pulls the latest image from ACR and deploys it across the Kubernetes cluster on AKS.
10. User requests for the app go through the DNS server.
11. The DNS server passes the requests to a load balancer, and sends responses back to the users.

See also

- [Team Data Science Process \(TDSP\)](#)
- [Azure Machine Learning \(AML\)](#)
- [Azure DevOps](#)
- [Azure Kubernetes Services \(AKS\)](#)

Walkthroughs executing the Team Data Science Process

3/5/2021 • 2 minutes to read • [Edit Online](#)

These **comprehensive 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 for 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:

- [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.
- [Azure Synapse Analytics](#) These walkthroughs use Azure Synapse Analytics 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, see [Team Data Science Process lifecycle](#). This lifecycle outlines the steps, from start to finish, that projects usually follow when they are executed.

For an overview, see [Data Science Process](#).

HDInsight Spark data science walkthroughs using PySpark and Scala on Azure

3/5/2021 • 2 minutes 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

Using New York taxi data, the [Use Spark on Azure HDInsight](#) walkthrough predicts whether a tip is paid and the range of expected amounts. This example 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 uses an HDInsight Spark cluster and Jupyter PySpark notebooks. 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 predicts 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 an overview of the Team Data Science Process, see [Team Data Science Process overview](#).

For a discussion of the Team Data Science Process lifecycle, see [Team Data Science Process lifecycle](#). This lifecycle outlines the steps, from start to finish, that projects usually follow when they are executed.

Data exploration and modeling with Spark

3/21/2021 • 27 minutes to read • [Edit Online](#)

Learn how to use HDInsight Spark to train machine learning models for taxi fare prediction using Spark MLlib.

This sample showcases the various steps in the [Team Data Science Process](#). A subset of the NYC taxi trip and fare 2013 dataset is used to load, explore and prepare data. Then, using Spark MLlib, binary classification and regression models are trained to predict whether a tip will be paid for the trip and estimate the tip amount.

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](#): Provide information on how to perform data exploration, modeling, and scoring with several different algorithms.
- [Spark 2.0 notebooks](#): Provide information on how to perform regression and classification tasks. Datasets may vary, but the steps and concepts are applicable to various datasets.

WARNING

Billing for HDInsight clusters is prorated per minute, whether you use them or not. Be sure to delete your cluster after you finish using it. 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

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.T
rain.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 <variable name>** 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 Jupyter notebook kernels and the predefined "magics", see [Kernels available for Jupyter notebooks with HDInsight Spark Linux clusters on HDInsight](#).

Load the data

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],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

Explore the data

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 setting makes the output 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

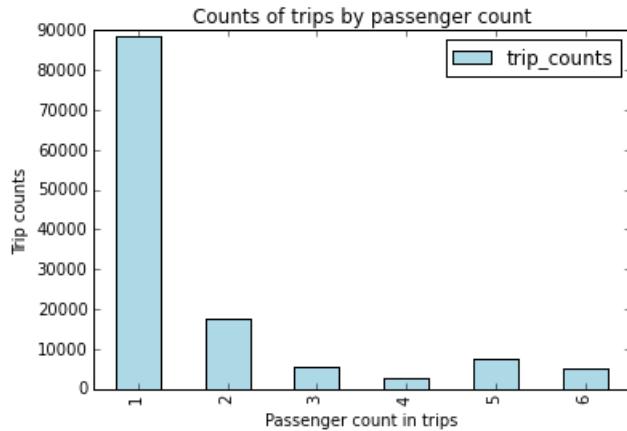
```

# 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

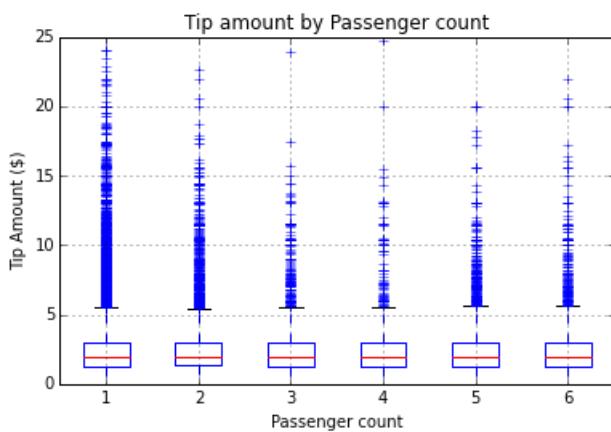
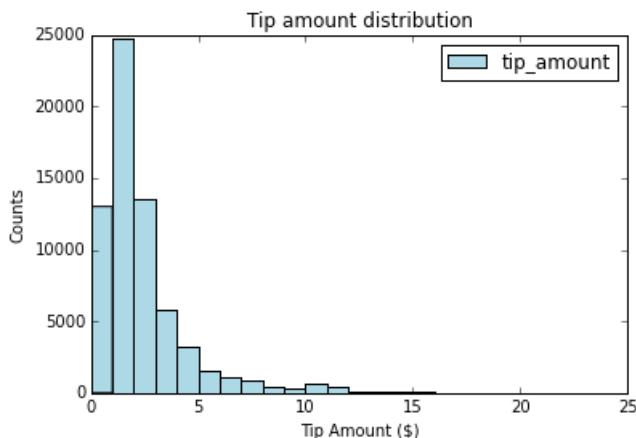
# HISTOGRAM OF TIP AMOUNTS AND PASSENGER COUNT
ax1 = sqlResults[['tip_amount']].plot(kind='hist', bins=25, facecolor='lightblue')
ax1.set_title('Tip amount distribution')
ax1.set_xlabel('Tip Amount ($)')
ax1.set_ylabel('Counts')
plt.suptitle('')
plt.show()

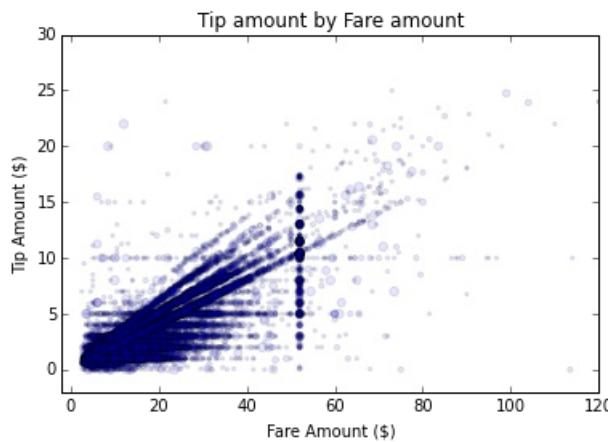
# TIP BY PASSENGER COUNT
ax2 = sqlResults.boxplot(column=['tip_amount'], by=['passenger_count'])
ax2.set_title('Tip amount by Passenger count')
ax2.set_xlabel('Passenger count')
ax2.set_ylabel('Tip Amount ($)')
plt.suptitle('')
plt.show()

# 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:





Prepare the data

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 subsampling 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 algorithm 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 subsampling 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, sampling can save significant time while training models. Next we split the sample into a training part (75% here) and a testing part (25% here) to use in classification and regression modeling.

```
# RECORD START TIME
timestart = datetime.datetime.now()

# LOAD PYSPARK LIBRARIES
from pyspark.sql.functions import rand

# SPECIFY SAMPLING AND SPLITTING FRACTIONS
samplingFraction = 0.25;
trainingFraction = 0.75; testingFraction = (1-trainingFraction);
seed = 1234;
encodedFinalSampled = encodedFinal.sample(False, samplingFraction, seed=seed)

# SPLIT SAMPLED DATA-FRAME INTO TRAIN/TEST
# INCLUDE RAND COLUMN FOR CREATING CROSS-VALIDATION FOLDS (FOR USE LATER IN AN ADVANCED TOPIC)
dfTmpRand = encodedFinalSampled.select("*", rand(0).alias("rand"));
trainData, testData = dfTmpRand.randomSplit([trainingFraction, testingFraction], seed=seed);

# FOR BINARY CLASSIFICATION TRAINING AND TESTING
indexedTRAINbinary = trainData.map(parseRowIndexingBinary)
indexedTESTbinary = testData.map(parseRowIndexingBinary)
oneHotTRAINbinary = trainData.map(parseRowOneHotBinary)
oneHotTESTbinary = testData.map(parseRowOneHotBinary)

# FOR REGRESSION TRAINING AND TESTING
indexedTRAINreg = trainData.map(parseRowIndexingRegression)
indexedTESTreg = testData.map(parseRowIndexingRegression)
oneHotTRAINreg = trainData.map(parseRowOneHotRegression)
oneHotTESTreg = testData.map(parseRowOneHotRegression)

# PRINT ELAPSED TIME
timeend = datetime.datetime.now()
timedelta = round((timeend-timestart).total_seconds(), 2)
print "Time taken to execute above cell: " + str(timedelta) + " seconds";
```

OUTPUT:

Time taken to execute above cell: 0.24 second

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 second

Train a binary classification model

This section shows how use three models for the binary classification task of predicting whether or not a tip is paid for a taxi trip. The models presented are:

- Regularized logistic regression
- Random forest model
- Gradient Boosting Trees

Each model building code section is split into steps:

1. **Model training** data with one parameter set
2. **Model evaluation** on a test data set with metrics
3. **Saving model** in blob for future consumption

Classification using logistic regression

The code in this section shows how to train, evaluate, and save a logistic regression model with [LBFGS](#) that predicts whether or not a tip is paid for a trip in the NYC taxi trip and fare dataset.

Train the logistic regression model using CV and hyperparameter sweeping

```
# LOGISTIC REGRESSION CLASSIFICATION WITH CV AND HYPERPARAMETER SWEEPING

# GET ACCURACY FOR HYPERPARAMETERS BASED ON CROSS-VALIDATION IN TRAINING DATA-SET

# RECORD START TIME
timestart = datetime.datetime.now()

# LOAD LIBRARIES
from pyspark.mllib.classification import LogisticRegressionWithLBFGS
from sklearn.metrics import roc_curve, auc
from pyspark.mllib.evaluation import BinaryClassificationMetrics
from pyspark.mllib.evaluation import MulticlassMetrics

# CREATE MODEL WITH ONE SET OF PARAMETERS
logitModel = LogisticRegressionWithLBFGS.train(oneHotTRAINbinary, iterations=20, initialWeights=None,
                                                regParam=0.01, regType='l2', intercept=True, corrections=10,
                                                tolerance=0.0001, validateData=True, numClasses=2)

# PRINT COEFFICIENTS AND INTERCEPT OF THE MODEL
# NOTE: There are 20 coefficient terms for the 10 features,
#       and the different categories for features: vendorVec (2), rateVec, paymentVec (6),
TrafficTimeBinsVec (4)
print("Coefficients: " + str(logitModel.weights))
print("Intercept: " + str(logitModel.intercept))

# PRINT ELAPSED TIME
timeend = datetime.datetime.now()
timedelta = round((timeend-timestart).total_seconds(), 2)
print "Time taken to execute above cell: " + str(timedelta) + " seconds";
```

OUTPUT:

Coefficients: [0.0082065285375, -0.0223675576104, -0.0183812028036, -3.48124578069e-05, -0.00247646947233, -0.00165897881503, 0.0675394837328, -0.111823113101, -0.324609912762, -0.204549780032, -1.36499216354, 0.591088507921, -0.664263411392, -1.00439726852, 3.46567827545, -3.51025855172, -0.0471341112232, -0.043521833294, 0.000243375810385, 0.054518719222]

Intercept: -0.0111216486893

Time taken to execute above cell: 14.43 seconds

Evaluate the binary classification model with standard metrics

```
#EVALUATE LOGISTIC REGRESSION MODEL WITH LBFGS

# RECORD START TIME
timestart = datetime.datetime.now()

# PREDICT ON TEST DATA WITH MODEL
predictionAndLabels = oneHotTESTbinary.map(lambda lp: (float(logitModel.predict(lp.features)), lp.label))

# INSTANTIATE METRICS OBJECT
metrics = BinaryClassificationMetrics(predictionAndLabels)

# AREA UNDER PRECISION-RECALL CURVE
print("Area under PR = %s" % metrics.areaUnderPR)

# AREA UNDER ROC CURVE
print("Area under ROC = %s" % metrics.areaUnderROC)
metrics = MulticlassMetrics(predictionAndLabels)

# OVERALL STATISTICS
precision = metrics.precision()
recall = metrics.recall()
f1Score = metrics.fMeasure()
print("Summary Stats")
print("Precision = %s" % precision)
print("Recall = %s" % recall)
print("F1 Score = %s" % f1Score)

## SAVE MODEL WITH DATE-STAMP
datestamp = unicode(datetime.datetime.now()).replace(' ', '_').replace(':', '_');
logisticregressionfilename = "LogisticRegressionWithLBFGS_" + datestamp;
dirfilename = modelDir + logisticregressionfilename;
logitModel.save(sc, dirfilename);

# OUTPUT PROBABILITIES AND REGISTER TEMP TABLE
logitModel.clearThreshold(); # This clears threshold for classification (0.5) and outputs probabilities
predictionAndLabelsDF = predictionAndLabels.toDF()
predictionAndLabelsDF.registerTempTable("tmp_results");

# PRINT ELAPSED TIME
timeend = datetime.datetime.now()
timedelta = round((timeend-timestart).total_seconds(), 2)
print "Time taken to execute above cell: " + str(timedelta) + " seconds";
```

OUTPUT:

Area under PR = 0.985297691373

Area under ROC = 0.983714670256

Summary Stats

Precision = 0.984304060189

Recall = 0.984304060189

F1 Score = 0.984304060189

Time taken to execute above cell: 57.61 seconds

Plot the ROC curve.

The `predictionAndLabelsDF` is registered as a table, `tmp_results`, in the previous cell. `tmp_results` can be used to do queries and output results into the `sqlResults` data-frame for plotting. Here is the code.

```
# QUERY RESULTS
%%sql -q -o sqlResults
SELECT * from tmp_results
```

Here is the code to make predictions and plot the ROC-curve.

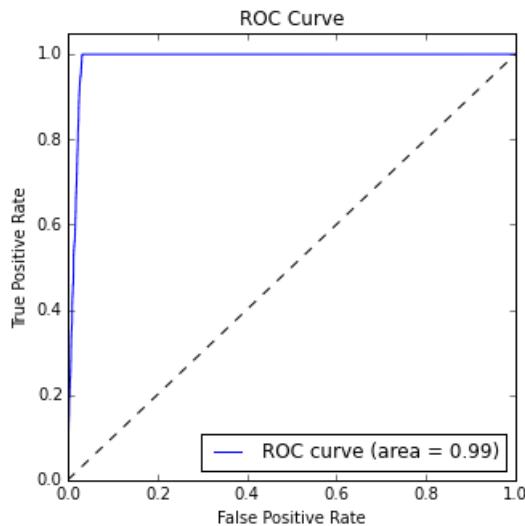
```
# MAKE PREDICTIONS AND PLOT ROC-CURVE

# RUN THE CODE LOCALLY ON THE JUPYTER SERVER AND IMPORT LIBRARIES
%%local
%matplotlib inline
from sklearn.metrics import roc_curve,auc

# MAKE PREDICTIONS
predictions_pddf = test_predictions.rename(columns={'_1': 'probability', '_2': 'label'})
prob = predictions_pddf["probability"]
fpr, tpr, thresholds = roc_curve(predictions_pddf['label'], prob, pos_label=1);
roc_auc = auc(fpr, tpr)

# PLOT ROC CURVE
plt.figure(figsize=(5,5))
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()
```

OUTPUT:



Random forest classification

The code in this section shows how to train, evaluate, and save a random forest model that predicts whether or not a tip is paid for a trip in the NYC taxi trip and fare dataset.

```

#PREDICT WHETHER A TIP IS PAID OR NOT USING RANDOM FOREST

# RECORD START TIME
timestart = datetime.datetime.now()

# LOAD PYSPARK LIBRARIES
from pyspark.mllib.tree import RandomForest, RandomForestModel
from pyspark.mllib.util import MLUtils
from pyspark.mllib.evaluation import BinaryClassificationMetrics
from pyspark.mllib.evaluation import MulticlassMetrics

# SPECIFY NUMBER OF CATEGORIES FOR CATEGORICAL FEATURES. FEATURE #0 HAS 2 CATEGORIES, FEATURE #2 HAS 2 CATEGORIES, AND SO ON
categoricalFeaturesInfo={0:2, 1:2, 2:6, 3:4}

# TRAIN RANDOMFOREST MODEL
rfModel = RandomForest.trainClassifier(indexedTRAINbinary, numClasses=2,
                                         categoricalFeaturesInfo=categoricalFeaturesInfo,
                                         numTrees=25, featureSubsetStrategy="auto",
                                         impurity='gini', maxDepth=5, maxBins=32)

## UN-COMMENT IF YOU WANT TO PRINT TREES
#print('Learned classification forest model:')
#print(rfModel.toDebugString())

# PREDICT ON TEST DATA AND EVALUATE
predictions = rfModel.predict(indexedTESTbinary.map(lambda x: x.features))
predictionAndLabels = indexedTESTbinary.map(lambda lp: lp.label).zip(predictions)

# AREA UNDER ROC CURVE
metrics = BinaryClassificationMetrics(predictionAndLabels)
print("Area under ROC = %s" % metrics.areaUnderROC)

# PERSIST MODEL IN BLOB
datestamp = unicode(datetime.datetime.now()).replace(' ', '').replace(':', '_');
rfclassificationfilename = "RandomForestClassification_" + datestamp;
dirfilename = modelDir + rfclassificationfilename;

rfModel.save(sc, dirfilename);

# PRINT ELAPSED TIME
timeend = datetime.datetime.now()
timedelta = round((timeend-timestart).total_seconds(), 2)
print "Time taken to execute above cell: " + str(timedelta) + " seconds";

```

OUTPUT:

Area under ROC = 0.985297691373

Time taken to execute above cell: 31.09 seconds

Gradient boosting trees classification

The code in this section shows how to train, evaluate, and save a gradient boosting trees model that predicts whether or not a tip is paid for a trip in the NYC taxi trip and fare dataset.

```

#PREDICT WHETHER A TIP IS PAID OR NOT USING GRADIENT BOOSTING TREES

# RECORD START TIME
timestart = datetime.datetime.now()

# LOAD PYSPARK LIBRARIES
from pyspark.mllib.tree import GradientBoostedTrees, GradientBoostedTreesModel

# SPECIFY NUMBER OF CATEGORIES FOR CATEGORICAL FEATURES. FEATURE #0 HAS 2 CATEGORIES, FEATURE #2 HAS 2 CATEGORIES, AND SO ON
categoricalFeaturesInfo={0:2, 1:2, 2:6, 3:4}

gbtModel = GradientBoostedTrees.trainClassifier(indexedTRAINbinary,
categoricalFeaturesInfo=categoricalFeaturesInfo, numIterations=5)
## UNCOMMENT IF YOU WANT TO PRINT TREE DETAILS
#print('Learned classification GBT model:')
#print(bgtModel.toDebugString())

# PREDICT ON TEST DATA AND EVALUATE
predictions = gbtModel.predict(indexedTESTbinary.map(lambda x: x.features))
predictionAndLabels = indexedTESTbinary.map(lambda lp: lp.label).zip(predictions)

# AREA UNDER ROC CURVE
metrics = BinaryClassificationMetrics(predictionAndLabels)
print("Area under ROC = %s" % metrics.areaUnderROC)

# PERSIST MODEL IN A BLOB
datestamp = unicode(datetime.datetime.now()).replace(' ','').replace(':', '_');
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

Train a regression model

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.

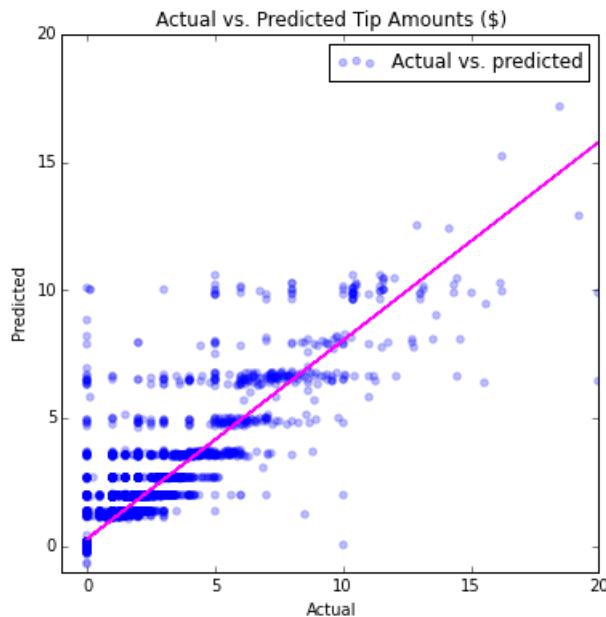
```

# 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()

```

Save the models

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

3/21/2021 • 37 minutes 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 build 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 prediction 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 search goes through a 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 is 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 use them or not. Be sure to delete your cluster after you finish using it. 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.T
rain.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 <variable name>** 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

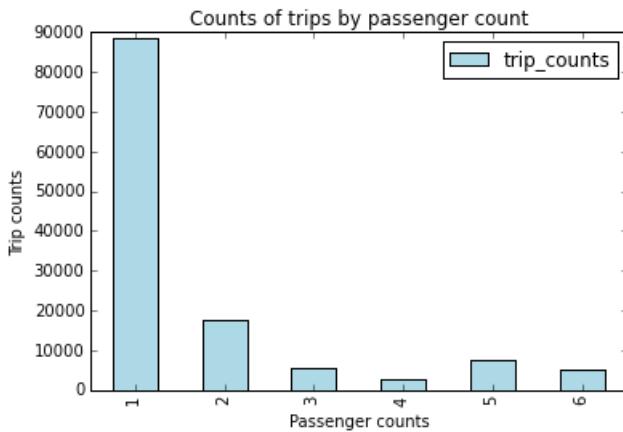
```

# 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 SQUERY
%%sql -q -o sqlResults
SELECT fare_amount, passenger_count, tip_amount, tipped
FROM taxi_train
WHERE passenger_count > 0
AND passenger_count < 7
AND fare_amount > 0
AND fare_amount < 200
AND payment_type in ('CSH', 'CRD')
AND tip_amount > 0
AND tip_amount < 25

```

This code cell uses the SQL query to create three plots the data.

```

# RUN THE CODE LOCALLY ON THE JUPYTER SERVER AND IMPORT LIBRARIES
%%local
%matplotlib inline

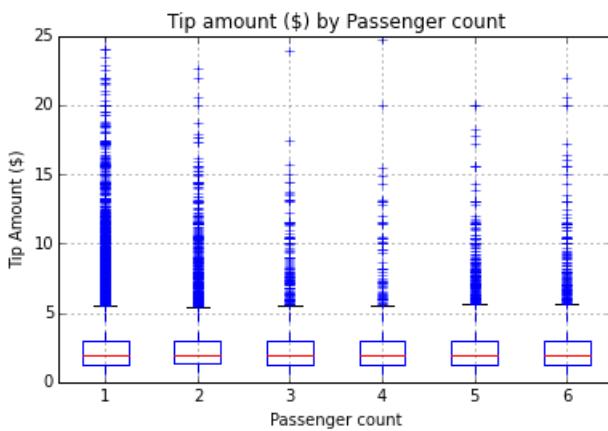
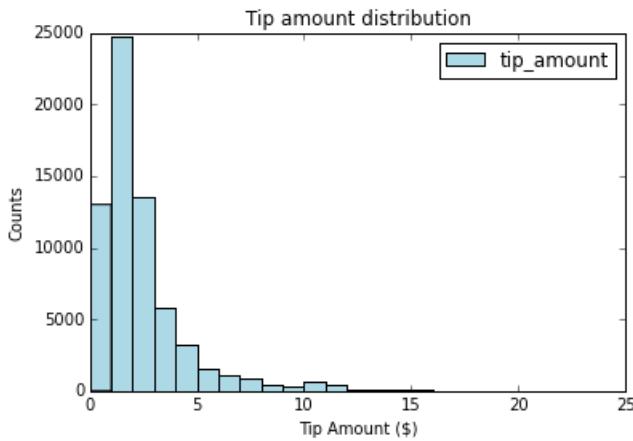
# TIP BY PAYMENT TYPE AND PASSENGER COUNT
ax1 = resultsPDDF[['tip_amount']].plot(kind='hist', bins=25, facecolor='lightblue')
ax1.set_title('Tip amount distribution')
ax1.set_xlabel('Tip Amount ($)')
ax1.set_ylabel('Counts')
plt.suptitle('')
plt.show()

# TIP BY PASSENGER COUNT
ax2 = resultsPDDF.boxplot(column=['tip_amount'], by=['passenger_count'])
ax2.set_title('Tip amount ($) by Passenger count')
ax2.set_xlabel('Passenger count')
ax2.set_ylabel('Tip Amount ($)')
plt.suptitle('')
plt.show()

# TIP AMOUNT BY FARE AMOUNT, POINTS ARE SCALED BY PASSENGER COUNT
ax = resultsPDDF.plot(kind='scatter', x= 'fare_amount', y = 'tip_amount', c='blue', alpha = 0.10, s=5*(resultsPDDF.passenger_count))
ax.set_title('Tip amount by Fare amount ($)')
ax.set_xlabel('Fare Amount')
ax.set_ylabel('Tip Amount')
plt.axis([-2, 120, -2, 30])
plt.show()

```

OUTPUT:





Feature engineering, transformation, and data preparation for modeling

This section describes and provides the code for procedures used to prepare data for use in ML modeling. It shows how to do the following tasks:

- Create a new feature by 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 subsampling 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 transformation prepares text data 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 subsampling 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, sampling can save significant time while training models. Next we split the sample into a training part (75% here) and a testing part (25% here) to use in classification and regression modeling.

```

# RECORD START TIME
timestart = datetime.datetime.now()

# SPECIFY SAMPLING AND SPLITTING FRACTIONS
from pyspark.sql.functions import rand

samplingFraction = 0.25;
trainingFraction = 0.75; testingFraction = (1-trainingFraction);
seed = 1234;
encodedFinalSampled = encodedFinal.sample(False, samplingFraction, seed=seed)

# SPLIT SAMPLED DATA-FRAME INTO TRAIN/TEST, WITH A RANDOM COLUMN ADDED FOR DOING CV (SHOWN LATER)
# INCLUDE RAND COLUMN FOR CREATING CROSS-VALIDATION FOLDS
dfTmpRand = encodedFinalSampled.select("*", rand(0).alias("rand"));
trainData, testData = dfTmpRand.randomSplit([trainingFraction, testingFraction], seed=seed);

# CACHE TRAIN AND TEST DATA
trainData.cache()
testData.cache()

# FOR BINARY CLASSIFICATION TRAINING AND TESTING
indexedTRAINbinary = trainData.map(parseRowIndexingBinary)
indexedTESTbinary = testData.map(parseRowIndexingBinary)
oneHotTRAINbinary = trainData.map(parseRowOneHotBinary)
oneHotTESTbinary = testData.map(parseRowOneHotBinary)

# FOR REGRESSION TRAINING AND TESTING
indexedTRAINreg = trainData.map(parseRowIndexingRegression)
indexedTESTreg = testData.map(parseRowIndexingRegression)
oneHotTRAINreg = trainData.map(parseRowOneHotRegression)
oneHotTESTreg = testData.map(parseRowOneHotRegression)

# PRINT ELAPSED TIME
timeend = datetime.datetime.now()
timedelta = round((timeend-timestart).total_seconds(), 2)
print "Time taken to execute above cell: " + str(timedelta) + " seconds";

```

OUTPUT

Time taken to execute above cell: 0.31 second

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 second

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 that can be applied to any algorithm in MLlib and to any parameter sets in an algorithm.
2. Using the **pySpark CrossValidator pipeline function**. CrossValidator has a few limitations for Spark 1.5.0:
 - Pipeline models cannot be saved or 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.

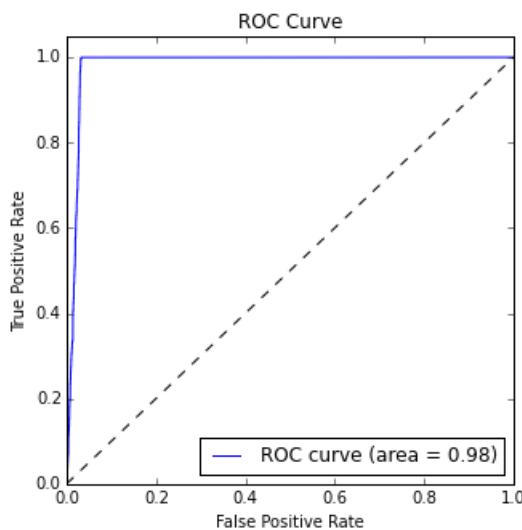
```
# MAKE PREDICTIONS AND PLOT ROC-CURVE

# 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.

```

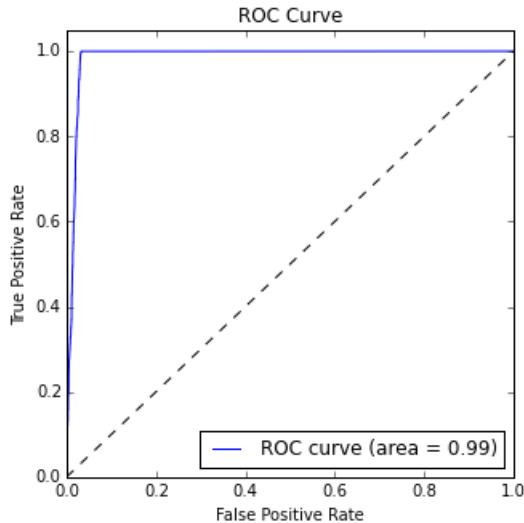
# RUN THE CODE LOCALLY ON THE JUPYTER SERVER AND IMPORT LIBRARIES
%%local
from sklearn.metrics import roc_curve,auc

# 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

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.

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

```

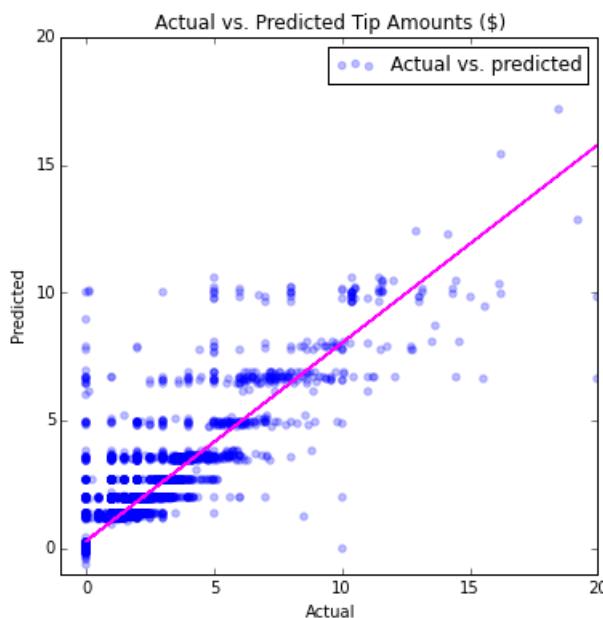
Here is the code to plot the data using the Jupyter server.

```

# 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 REGISTER 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 *sql/Results* 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

**Output 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

3/21/2021 • 17 minutes 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. 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 use them or not. Be sure to delete your cluster after you finish using it. 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://mlibwalkthroughs@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. Therefore, 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 by default:

- 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 <variable name>**
- 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. The RDDs 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 REGRESSION MODELS
def parseRowOneHotBinary(line):
    features = np.concatenate((np.array([line.pickup_hour, line.weekday, line.passenger_count,
                                         line.trip_time_in_secs, line.trip_distance, line.fare_amount]),
                               line.vendorVec.toArray(), line.rateVec.toArray(),
                               line.paymentVec.toArray(), line.TrafficTimeBinsVec.toArray()),
                              axis=0)
    return features

# ONE-HOT ENCODING OF CATEGORICAL TEXT FEATURES FOR INPUT INTO TREE-BASED MODELS
def parseRowIndexingRegression(line):
    features = np.array([line.paymentIndex, line.vendorIndex, line.rateIndex, line.TrafficTimeBinsIndex,
                        line.pickup_hour, line.weekday, line.passenger_count, line.trip_time_in_secs,
                        line.trip_distance, line.fare_amount])
    return features

# INDEXING CATEGORICAL TEXT FEATURES FOR INPUT INTO LINEAR REGRESSION MODELS
def parseRowOneHotRegression(line):
    features = np.concatenate((np.array([line.pickup_hour, line.weekday, line.passenger_count,
                                         line.trip_time_in_secs, line.trip_distance, line.fare_amount]),
                               line.vendorVec.toArray(), line.rateVec.toArray(),
                               line.paymentVec.toArray(), line.TrafficTimeBinsVec.toArray()),
                              axis=0)
    return features

# FOR BINARY CLASSIFICATION TRAINING AND TESTING
indexedTESTbinary = encodedFinal.map(parseRowIndexingBinary)
oneHotTESTbinary = encodedFinal.map(parseRowOneHotBinary)

# FOR REGRESSION CLASSIFICATION TRAINING AND TESTING
indexedTESTreg = encodedFinal.map(parseRowIndexingRegression)
oneHotTESTreg = encodedFinal.map(parseRowOneHotRegression)

# SCALING FEATURES FOR 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. This algorithm 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}

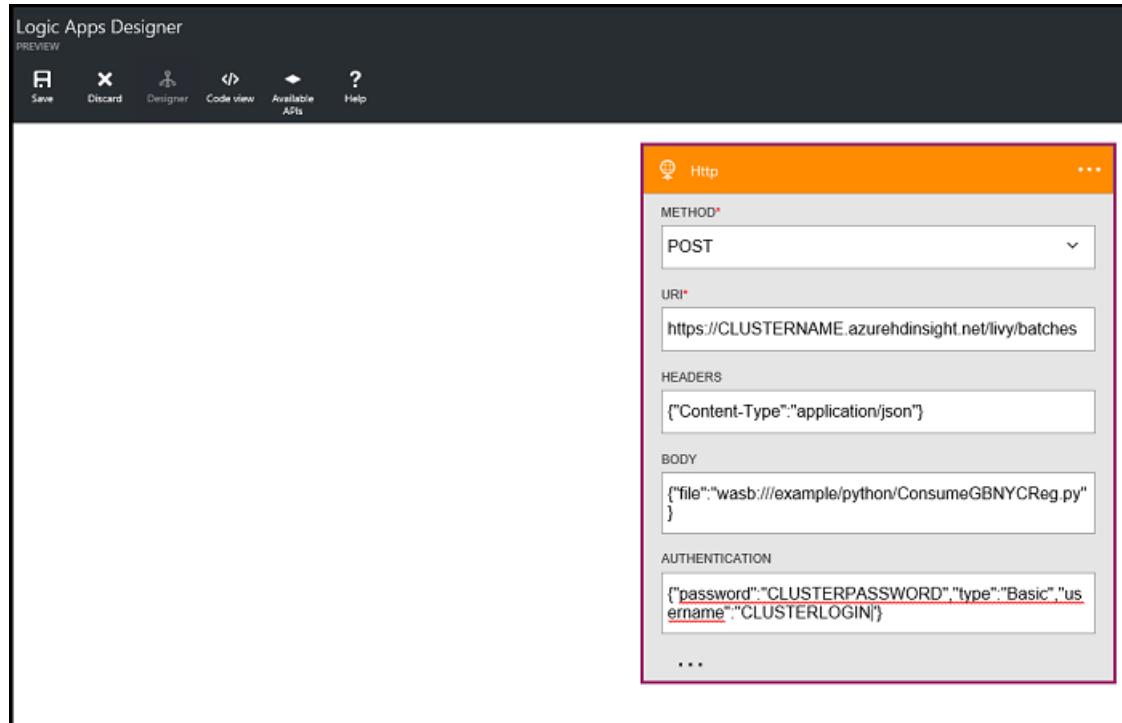
# SPECIFY THE PYTHON SCRIPT TO RUN ON THE SPARK CLUSTER
# IN THE FILE PARAMETER OF THE JSON POST REQUEST BODY
r=conn.request("POST", '/livy/batches', '{"file": "wasb:///example/python/ConsumeGBNYCReg.py"}', headers )
response = conn.getresponse().read()
print(response)
conn.close()

```

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

1/10/2020 • 2 minutes 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 Hadoop Clusters on a 1-TB dataset](#) walkthrough uses a publicly available [Criteo](#) click dataset to predict whether a tip is paid and the expected amounts. 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

1/10/2020 • 2 minutes 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. The data are a sample of the NYC taxi dataset. The task is predicting whether or not a tip is paid by a customer.

Next steps

For an overview of the Team Data Science Process, see [Team Data Science Process overview](#).

For a discussion of the Team Data Science Process lifecycle, see [Team Data Science Process lifecycle](#). This lifecycle outlines the steps that projects usually follow when they are executed.

SQL Server data science walkthroughs using R, Python, and T-SQL

3/5/2021 • 2 minutes 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. The data are 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 shows how 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.

Azure Synapse Analytics data science walkthroughs using T-SQL and Python on Azure

11/2/2020 • 2 minutes to read • [Edit Online](#)

These walkthroughs use of Azure Synapse Analytics 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 Synapse Analytics, see [What is Azure Synapse Analytics?](#)

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 Azure Synapse Analytics

The [Use Azure Synapse Analytics walkthrough](#) shows you how to build and deploy machine learning classification and regression models using Azure Synapse Analytics. The data are 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, see [Team Data Science Process lifecycle](#). This lifecycle outlines the steps, from start to finish, that projects usually follow when they are executed.

Team Data Science Process for data scientists

3/5/2021 • 8 minutes 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	<p>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.</p>	<p>Prepare a 30-minute presentation to a technical audience on how DevOps is essential for analytics projects.</p>
Understand the Technologies for Data Storage and Processing	Microsoft Business Analytics and AI	<p>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.</p>	<p>Download and review the presentation materials from this workshop.</p>
Setup and Configure your training, development, and production environments	Microsoft Azure	<p>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.</p>	<p>If you do not have an Azure Account, create one. Log in to the Microsoft Azure portal and create one Resource Group for training.</p>
	The Microsoft Azure Command-Line Interface (CLI)	<p>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.</p>	<p>Set your default subscription with the Azure CLI.</p>

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 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 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. 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 to create a predictive solution	Azure Machine Learning	Microsoft Azure Machine Learning uses AI for data wrangling and feature engineering, manages experiments, and tracks model runs. All of this works in a single environment and most functions can run locally or in Azure. You can use the PyTorch, 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.	

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 Monitor logs	Another method to monitor your application is to integrate it into your DevOps process. The Azure Monitor logs system provides a rich set of features to help you watch your analytic solutions after you deploy them.	Complete this tutorial on using Azure Monitor logs .
Complete this Learning Path		Congratulations! You've completed this learning path.	

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

3/10/2021 • 9 minutes 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 implement the Team Data Science Process (TDSP) and Microsoft and open-source software and toolkits, 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 level-based guidance to help complete the DevOps objectives for implementing data science solutions on Azure.

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 for analytics data solutions	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 can 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 Synapse Analytics to store and process large amounts of data	Azure Synapse Analytics	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

OBJECTIVE	TOPIC	RESOURCE	TECHNOLOGIES	LEVEL	PREREQUISITES
		This overview covers using Azure Stream Analytics as a fully-managed event-processing engine to support 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
	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 an 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

OBJECTIVE	TOPIC	RESOURCE	TECHNOLOGIES	LEVEL	PREREQUISITES
	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 Cognitive Search, Python, Azure Data Factory, Power BI, Azure Document DB, Application Insights, Azure SQL DB, Azure Synapse Analytics, Microsoft SQL Server, Azure Data Lake, Cognitive Services, Bot Framework, Azure Batch,	Intermediate	General technology background, familiarity with data solutions
Understand and Implement DevOps Processes	DevOps Fundamentals	This video series explains the covers the fundamentals of DevOps and helps explain how they map to DevOps practices	DevOps, Microsoft Azure Platform, Azure DevOps	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

OBJECTIVE	TOPIC	RESOURCE	TECHNOLOGIES	LEVEL	PREREQUISITES
		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
		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 Azure Data Catalog tutorial describes a system of registration and 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 Codespace and Microsoft Azure	Visual Studio Codespace	Experienced	Software Development, familiarity with Dev/Test environments

OBJECTIVE	TOPIC	RESOURCE	TECHNOLOGIES	LEVEL	PREREQUISITES
		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
		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 Codespace Code for creating Data Science and AI applications	Visual Studio Codespace	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

OBJECTIVE	TOPIC	RESOURCE	TECHNOLOGIES	LEVEL	PREREQUISITES
		This resource explains how to code and create Predictive Analytics experiments using the web-based Azure Machine Learning Studio (classic) tool	Azure Machine Learning Studio (classic)	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
		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 Codespace Build	Visual Studio Codespace	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

OBJECTIVE	TOPIC	RESOURCE	TECHNOLOGIES	LEVEL	PREREQUISITES
	Test	Use this reference to understand how to use Visual Studio Codespace for Test Case Management	Visual Studio Codespace	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
		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 Visual Studio Codespace	Visual Studio Codespace	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

OBJECTIVE	TOPIC	RESOURCE	TECHNOLOGIES	LEVEL	PREREQUISITES
		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
	Release	This head-reference article contains concepts for build, test, and release for CI/CD environments	Visual Studio Codespace	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

OBJECTIVE	TOPIC	RESOURCE	TECHNOLOGIES	LEVEL	PREREQUISITES
		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
		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 Monitor logs	Azure Logs, PowerShell	Experienced	Familiarity with the Azure Platform

OBJECTIVE	TOPIC	RESOURCE	TECHNOLOGIES	LEVEL	PREREQUISITES
Understand how to use Open Source 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, Azure DevOps, Open Source Software	Experienced	Used an SDLC, familiarity with Agile and other Development Frameworks, IT Operations Familiarity
		This tutorial automates 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
		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 for implementing data science solutions with Azure.

Set up data science environments for use in the Team Data Science Process

3/5/2021 • 2 minutes 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](#)

The following articles describe how to set up the various data science environments used by the Team Data Science Process.

- [Azure storage-account](#)
- [HDInsight \(Hadoop\) cluster](#)
- [Azure Machine Learning Studio \(classic\) workspace](#)

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 more information, see [Introduction to the cloud-based Data Science Virtual Machine for Linux and Windows](#).

Learn how to create:

- [Windows DSVM](#)
- [Ubuntu DSVM](#)
- [CentOS DSVM](#)

Create a storage account

3/24/2021 • 8 minutes to read • [Edit Online](#)

An Azure storage account contains all of your Azure Storage data objects: blobs, files, queues, tables, and disks. The storage account provides a unique namespace for your Azure Storage data that is accessible from anywhere in the world over HTTP or HTTPS. Data in your Azure storage account is durable and highly available, secure, and massively scalable.

In this how-to article, you learn to create a storage account using the [Azure portal](#), [Azure PowerShell](#), [Azure CLI](#), or an [Azure Resource Manager template](#).

NOTE

This article has been updated to use the Azure Az PowerShell module. The Az PowerShell module is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see [Install Azure PowerShell](#). To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

Prerequisites

If you don't have an Azure subscription, create a [free account](#) before you begin.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [Template](#)

None.

Sign in to Azure

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [Template](#)

Sign in to the [Azure portal](#).

Create a storage account

Every storage account must belong to an Azure resource group. A resource group is a logical container for grouping your Azure services. When you create a storage account, you have the option to either create a new resource group, or use an existing resource group. This article shows how to create a new resource group.

A **general-purpose v2** storage account provides access to all of the Azure Storage services: blobs, files, queues, tables, and disks. The steps outlined here create a general-purpose v2 storage account, but the steps to create any type of storage account are similar. For more information about types of storage accounts and other storage account settings, see [Azure storage account overview](#).

- [Portal](#)
- [PowerShell](#)

- [Azure CLI](#)
- [Template](#)

To create a general-purpose v2 storage account in the Azure portal, follow these steps:

1. On the Azure portal menu, select **All services**. In the list of resources, type **Storage Accounts**. As you begin typing, the list filters based on your input. Select **Storage Accounts**.
2. On the **Storage Accounts** window that appears, choose **Add**.
3. On the **Basics** tab, select the subscription in which to create the storage account.
4. Under the **Resource group** field, select your desired resource group, or create a new resource group. For more information on Azure resource groups, see [Azure Resource Manager overview](#).
5. Next, enter a name for your storage account. The name you choose must be unique across Azure. The name also must be between 3 and 24 characters in length, and may include only numbers and lowercase letters.
6. Select a location for your storage account, or use the default location.
7. Select a performance tier. The default tier is *Standard*.
8. Set the **Account kind** field to *Storage V2 (general-purpose v2)*.
9. Specify how the storage account will be replicated. The default replication option is *Read-access geo-redundant storage (RA-GRS)*. For more information about available replication options, see [Azure Storage redundancy](#).
10. Additional options are available on the **Networking**, **Data protection**, **Advanced**, and **Tags** tabs. To use Azure Data Lake Storage, choose the **Advanced** tab, and then set **Hierarchical namespace** to **Enabled**. For more information, see [Azure Data Lake Storage Gen2 Introduction](#)
11. Select **Review + Create** to review your storage account settings and create the account.
12. Select **Create**.

The following image shows the settings on the **Basics** tab for a new storage account:

Create storage account

X

[Basics](#) [Networking](#) [Data protection](#) [Advanced](#) [Tags](#) [Review + create](#)

Azure Storage is a Microsoft-managed service providing cloud storage that is highly available, secure, durable, scalable, and redundant. Azure Storage includes Azure Blobs (objects), Azure Data Lake Storage Gen2, Azure Files, Azure Queues, and Azure Tables. The cost of your storage account depends on the usage and the options you choose below.

[Learn more about Azure storage accounts](#)

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription *	Azure Storage content development and testing
Resource group *	storagesamples-rg
	Create new

Instance details

The default deployment model is Resource Manager, which supports the latest Azure features. You may choose to deploy using the classic deployment model instead. [Choose classic deployment model](#)

Storage account name * ⓘ	createacctstorage
Location *	(US) West US
Performance ⓘ	<input checked="" type="radio"/> Standard <input type="radio"/> Premium
Account kind ⓘ	StorageV2 (general purpose v2)
Replication ⓘ	Read-access geo-redundant storage (RA-GRS)

[Review + create](#)[< Previous](#)[Next : Networking >](#)

Delete a storage account

Deleting a storage account deletes the entire account, including all data in the account, and cannot be undone.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [Template](#)

1. Navigate to the storage account in the [Azure portal](#).
2. Click **Delete**.

Alternately, you can delete the resource group, which deletes the storage account and any other resources in that resource group. For more information about deleting a resource group, see [Delete resource group and resources](#).

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, see [Troubleshoot errors when you delete storage accounts](#).

Next steps

- [Storage account overview](#)
- [Upgrade to a general-purpose v2 storage account](#)
- [Move an Azure Storage account to another region](#)
- [Recover a deleted storage account](#)

Platforms and tools for data science projects

3/5/2021 • 9 minutes to read • [Edit Online](#)

Microsoft provides a full spectrum of analytics resources for both cloud or on-premises 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 analytics resources available to data science teams using the TDSP include:

- Data Science Virtual Machines (both Windows and Linux CentOS)
- HDInsight Spark Clusters
- Azure Synapse Analytics
- Azure Data Lake
- HDInsight Hive Clusters
- Azure File Storage
- SQL Server 2019 R and Python Services
- Azure Databricks

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 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 [10 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 Synapse Analytics

Azure Synapse Analytics 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 Synapse Analytics. 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 Synapse Analytics, see the [Azure Synapse Analytics](#) website. To learn how to build end-to-end advanced analytics solutions with Azure Synapse Analytics, see [The Team Data Science Process in action: using Azure Synapse Analytics](#).

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 2019 R and Python Services

R Services (In-database) provides 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) integrates 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) supports 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

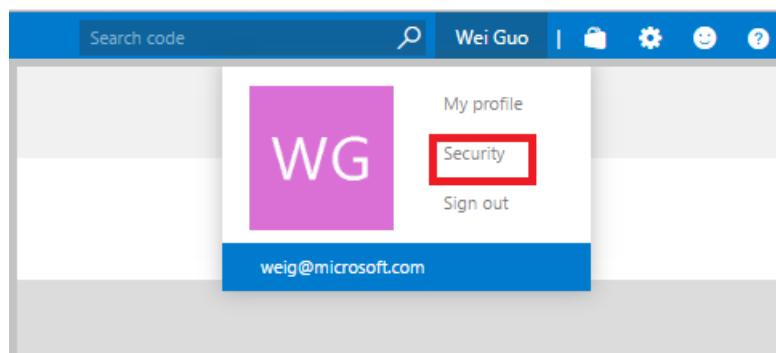
If you are using Linux (CentOS) machines to run the git commands, you need to add the public SSH key of your machine to your Azure DevOps Services, so that this machine is recognized by the Azure DevOps Services. First, you need to generate a public SSH key and add the key to SSH public keys in your Azure DevOps Services security setting page.

1. 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 fingerprint is:  
64:0f:bf:29:78:b0:e7:1b:f3:59:c9:ff:0f:92:bb:75 ds1@weiglinuxdsvm3  
The key's randomart image is:  
+--[ RSA 2048]----  
  
+ * .  
. 5 * +.  
o B o .  
. o + o E  
. . + + .  
+ + + + o +  
[ds1@weiglinuxdsvm3 ~]$ cat .ssh/id_rsa.pub  
-----  
MIIBIjANBgkqhkiG9w0BAQEFAAQIDQgAAMADQdAlMMAdAQc5B1AxW5n1WtATQqJn6fCCbgpP0VXgMAf13555Lg2Jfp#0Jduh0xZRM0LsfCgr#QVbzPSdtNC4/0Y4)lxz1d1KEW7p$1b0gAc2JR10M9f2ZIN10cf1b2gdanibc01ngpSPR3eua/PfR21  
L2UPLMC7cN10DkK6Bw/9Ng5r2454TNU+sdB925XhyuVhc140hR60jd129kOz03seRu09Ygj55beiu19cuFr0IE1+1ld8fdx4LockayFzeENMwciDhOrre03JXqya0siBS#404og17eb15P10Fc5m9PTQ7lIEipzbcaG  
H0X ds1@weiglinuxdsvm3  
[ds1@weiglinuxdsvm3 ~]$
```

2. Copy the entire ssh key including *ssh-rsa*.
3. Log in to your Azure DevOps Services.
4. Click <Your Name> at the top-right corner of the page and click **security**.



5. Click **SSH public keys**, and click **+Add**.



6. Paste the ssh key copied into the text box and save.

Next steps

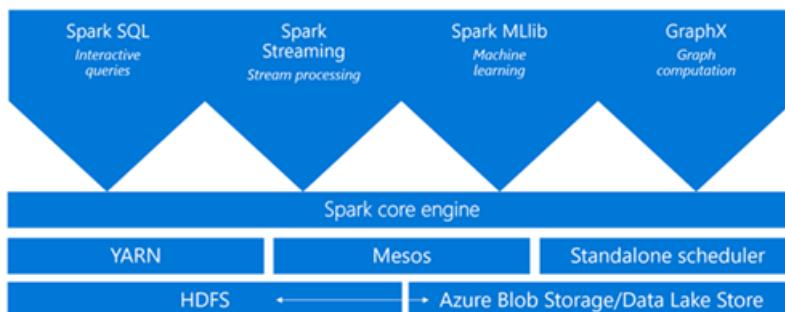
Full end-to-end walkthroughs that demonstrate all the steps in the process for **specific scenarios** are also provided. They are listed and linked with thumbnail descriptions in the [Example walkthroughs](#) topic. They illustrate how to combine cloud, on-premises tools, and services into a workflow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

For examples that show how to execute steps in the Team Data Science Process by using Azure Machine Learning Studio (classic), see the [With Azure ML](#) learning path.

What is Apache Spark in Azure HDInsight

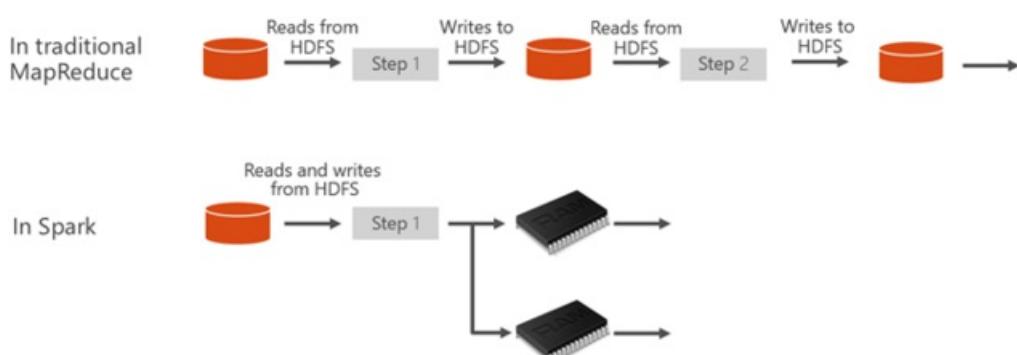
3/23/2021 • 6 minutes to read • [Edit Online](#)

Apache Spark is a parallel processing framework that supports in-memory processing to boost the performance of big-data analytic applications. Apache Spark in Azure HDInsight is the Microsoft implementation of Apache Spark in the cloud. HDInsight makes it easier to create and configure a Spark cluster in Azure. Spark clusters in HDInsight are compatible with [Azure Blob storage](#), [Azure Data Lake Storage Gen1](#), or [Azure Data Lake Storage Gen2](#). So you can use HDInsight Spark clusters to process your data stored in Azure. For the components and the versioning information, see [Apache Hadoop components and versions in Azure HDInsight](#).



What is Apache Spark?

Spark provides primitives for in-memory cluster computing. A Spark job can load and cache data into memory and query it repeatedly. In-memory computing is much faster than disk-based applications, such as Hadoop, which shares data through Hadoop distributed file system (HDFS). 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.



Spark clusters in HDInsight offer a fully managed Spark service. Benefits of creating a Spark cluster in HDInsight are listed here.

FEATURE	DESCRIPTION
Ease creation	You can create a new Spark cluster in HDInsight in minutes using the Azure portal, Azure PowerShell, or the HDInsight .NET SDK. See Get started with Apache Spark cluster in HDInsight .

FEATURE	DESCRIPTION
Ease of use	Spark cluster in HDInsight include Jupyter Notebooks and Apache Zeppelin Notebooks. You can use these notebooks for interactive data processing and visualization. See Use Apache Zeppelin notebooks with Apache Spark and Load data and run queries on an Apache Spark cluster .
REST APIs	Spark clusters in HDInsight include Apache Livy , a REST API-based Spark job server to remotely submit and monitor jobs. See Use Apache Spark REST API to submit remote jobs to an HDInsight Spark cluster .
Support for Azure Storage	Spark clusters in HDInsight can use Azure Data Lake Storage Gen1/Gen2 as both the primary storage or additional storage. For more information on Data Lake Storage Gen1, see Azure Data Lake Storage Gen1 . For more information on Data Lake Storage Gen2, see Azure Data Lake Storage Gen2 .
Integration with Azure services	Spark cluster in HDInsight comes with a connector to Azure Event Hubs. You can build streaming applications using the Event Hubs. Including Apache Kafka, which is already available as part of Spark.
Support for ML Server	Support for ML Server in HDInsight is provided as the ML Services cluster type. You can set up an ML Services cluster to run distributed R computations with the speeds promised with a Spark cluster. For more information, see What is ML Services in Azure HDInsight .
Integration with third-party IDEs	HDInsight provides several IDE plugins that are useful to create and submit applications to an HDInsight Spark cluster. For more information, see Use Azure Toolkit for IntelliJ IDEA , Use Spark & Hive Tools for VSCode , and Use Azure Toolkit for Eclipse .
Concurrent Queries	Spark clusters in HDInsight support concurrent queries. This capability 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. See Improve performance of Apache Spark workloads using Azure HDInsight IO Cache .
Integration with BI Tools	Spark clusters in HDInsight provide connectors for BI tools such as Power BI for data analytics.
Pre-loaded Anaconda libraries	Spark clusters in HDInsight come with Anaconda libraries pre-installed. Anaconda provides close to 200 libraries for machine learning, data analysis, visualization, and so on.

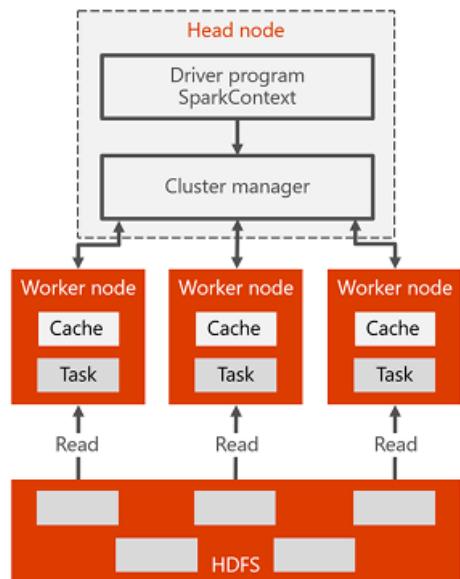
FEATURE	DESCRIPTION
Adaptability	HDInsight allows you to change the number of cluster nodes dynamically with the Autoscale feature. See Automatically scale Azure HDInsight clusters . Also, Spark clusters can be dropped with no loss of data since all the data is stored in Azure Blob storage, Azure Data Lake Storage Gen1 or Azure Data Lake Storage Gen2 .
SLA	Spark clusters in HDInsight come with 24/7 support and an SLA of 99.9% up-time.

Apache 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](#)
- [Apache Livy](#)
- [Jupyter Notebook](#)
- [Apache Zeppelin notebook](#)

HDInsight Spark clusters an [ODBC driver](#) for connectivity from BI tools such as Microsoft Power BI.

Spark cluster architecture



It's easy to understand the components of Spark by understanding how Spark runs on HDInsight clusters.

Spark applications run as independent sets of processes on a cluster. Coordinated by the `SparkContext` object in your main program (called the driver program).

The `SparkContext` can connect to several types of cluster managers, which give resources across applications. These cluster managers include Apache Mesos, Apache Hadoop YARN, or the Spark cluster manager. In HDInsight, Spark runs using the YARN cluster manager. Once connected, Spark acquires executors on workers nodes in the cluster, which are processes that run computations and store data for your application. Next, it sends your application code (defined by JAR or Python files passed to `SparkContext`) to the executors. Finally, `SparkContext` sends tasks to the executors to run.

The `SparkContext` runs the user's main function and executes the various parallel operations on the worker nodes. Then, the `SparkContext` collects the results of the operations. The worker nodes read and write data from

and to the Hadoop distributed file system. The worker nodes also cache transformed data in-memory as Resilient Distributed Datasets (RDDs).

The SparkContext connects to the Spark master and is responsible for converting an application to a directed graph (DAG) of individual tasks. 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.

Spark in HDInsight use cases

Spark clusters in HDInsight enable the following key scenarios:

Interactive data analysis and BI

Apache Spark in HDInsight stores data in Azure Blob Storage, Azure Data Lake Gen1, or Azure Data Lake Storage Gen2. 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 easier for data analysts, business experts, and key decision makers.

- [Tutorial: Visualize Spark data using Power BI](#)

Spark Machine Learning

Apache Spark comes with [MLlib](#). MLlib is a machine learning library built on top of Spark that you can use from a Spark cluster in HDInsight. Spark cluster in HDInsight also includes Anaconda, a Python distribution with different kinds of packages for machine learning. And with built-in support for Jupyter and Zeppelin notebooks, you have an environment for creating machine learning applications.

- [Tutorial: Predict building temperatures using HVAC data](#)
- [Tutorial: Predict food inspection results](#)

Spark streaming and real-time data analysis

Spark clusters in HDInsight offer a rich support for building real-time analytics solutions. 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 complete support for Event Hubs makes Spark clusters in HDInsight an ideal platform for building real-time analytics pipeline.

- [Overview of Apache Spark Streaming](#)
- [Overview of Apache Spark Structured Streaming](#)

Next Steps

In this overview, you've got a basic understanding of Apache Spark in Azure HDInsight. You can use the following articles to learn more about Apache Spark in HDInsight, and you can create an HDInsight Spark cluster and further run some sample Spark queries:

- [Quickstart: Create an Apache Spark cluster in HDInsight and run interactive query using Jupyter](#)
- [Tutorial: Load data and run queries on an Apache Spark job using Jupyter](#)
- [Tutorial: Visualize Spark data using Power BI](#)
- [Tutorial: Predict building temperatures using HVAC data](#)
- [Optimize Spark jobs for performance](#)

Quickstart: Create Apache Spark cluster in Azure HDInsight using ARM template

3/23/2021 • 8 minutes to read • [Edit Online](#)

In this quickstart, you use an Azure Resource Manager template (ARM template) to create an [Apache Spark](#) cluster in Azure HDInsight. You then create a Jupyter Notebook file, and use it to run Spark SQL queries against Apache Hive tables. Azure HDInsight is a managed, full-spectrum, open-source analytics service for enterprises. The Apache Spark framework for HDInsight enables fast data analytics and cluster computing using in-memory processing. Jupyter Notebook lets you interact with your data, combine code with markdown text, and do simple visualizations.

If you're using multiple clusters together, you'll want to create a virtual network, and if you're using a Spark cluster you'll also want to use the Hive Warehouse Connector. For more information, see [Plan a virtual network for Azure HDInsight](#) and [Integrate Apache Spark and Apache Hive with the Hive Warehouse Connector](#).

An [ARM template](#) is a JavaScript Object Notation (JSON) file that defines the infrastructure and configuration for your project. The template uses declarative syntax. In declarative syntax, you describe your intended deployment without writing the sequence of programming commands to create the deployment.

If your environment meets the prerequisites and you're familiar with using ARM templates, select the **Deploy to Azure** button. The template will open in the Azure portal.



Prerequisites

If you don't have an Azure subscription, create a [free account](#) before you begin.

Review the template

The template used in this quickstart is from [Azure Quickstart Templates](#).

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "clusterName": {
      "type": "string",
      "metadata": {
        "description": "The name of the HDInsight cluster to create."
      }
    },
    "clusterLoginUserName": {
      "type": "string",
      "metadata": {
        "description": "These credentials can be used to submit jobs to the cluster and to log into cluster dashboards."
      }
    },
    "clusterLoginPassword": {
      "type": "securestring",
      "minLength": 10,
      "metadata": {
        "description": "The password must be at least 10 characters in length and must contain at least one digit, one upper case letter, one lower case letter, and one non-alphanumeric character except (/single-)
```

"`variables`, one upper case letter, one lower case letter, and one non-alphanumeric character except (single quote, double-quote, backslash, right-bracket, full-stop). Also, the password must not contain 3 consecutive characters from the cluster username or SSH username."

```
        },
        },
        "sshUserName": {
            "type": "string",
            "metadata": {
                "description": "These credentials can be used to remotely access the cluster."
            }
        },
        "sshPassword": {
            "type": "securestring",
            "minLength": 6,
            "maxLength": 72,
            "metadata": {
                "description": "SSH password must be 6-72 characters long and must contain at least one digit, one upper case letter, and one lower case letter. It must not contain any 3 consecutive characters from the cluster login name"
            }
        },
        "location": {
            "type": "string",
            "defaultValue": "[resourceGroup().location]",
            "metadata": {
                "description": "Location for all resources."
            }
        },
        "HeadNodeVirtualMachineSize": {
            "type": "string",
            "defaultValue": "Standard_E4_v3",
            "allowedValues": [
                "Standard_A4_v2",
                "Standard_A8_v2",
                "Standard_E2_v3",
                "Standard_E4_v3",
                "Standard_E8_v3",
                "Standard_E16_v3",
                "Standard_E20_v3",
                "Standard_E32_v3",
                "Standard_E48_v3"
            ],
            "metadata": {
                "description": "This is the headnode Azure Virtual Machine size, and will affect the cost. If you don't know, just leave the default value."
            }
        },
        "WorkerNodeVirtualMachineSize": {
            "type": "string",
            "defaultValue": "Standard_E4_v3",
            "allowedValues": [
                "Standard_A4_v2",
                "Standard_A8_v2",
                "Standard_E2_v3",
                "Standard_E4_v3",
                "Standard_E8_v3",
                "Standard_E16_v3",
                "Standard_E20_v3",
                "Standard_E32_v3",
                "Standard_E48_v3"
            ],
            "metadata": {
                "description": "This is the workernode Azure Virtual Machine size, and will affect the cost. If you don't know, just leave the default value."
            }
        }
    },
    "variables": {
        "defaultStorageAccount": {
            "name": "ResourceGroupStorage" + uniqueString(resourceGroup().id)
        }
    }
}
```

```

        "name": "[concat('storage', uniquestring(resourcegroup().id))]",
        "type": "Standard_LRS"
    },
},
"resources": [
{
    "type": "Microsoft.Storage/storageAccounts",
    "apiVersion": "2019-06-01",
    "name": "[variables('defaultStorageAccount').name]",
    "location": "[parameters('location')]",
    "sku": {
        "name": "[variables('defaultStorageAccount').type]"
    },
    "kind": "Storage",
    "properties": {}
},
{
    "type": "Microsoft.HDInsight/clusters",
    "apiVersion": "2018-06-01-preview",
    "name": "[parameters('clusterName')]",
    "location": "[parameters('location')]",
    "dependsOn": [
        "[resourceId('Microsoft.Storage/storageAccounts', variables('defaultStorageAccount').name)]"
    ],
    "properties": {
        "clusterVersion": "4.0",
        "osType": "Linux",
        "tier": "Standard",
        "clusterDefinition": {
            "kind": "spark",
            "configurations": {
                "gateway": {
                    "restAuthCredential.isEnabled": true,
                    "restAuthCredential.username": "[parameters('clusterLoginUserName')]",
                    "restAuthCredential.password": "[parameters('clusterLoginPassword')]"
                }
            }
        },
        "storageProfile": {
            "storageaccounts": [
                {
                    "name": "[replace(replace(reference(resourceId('Microsoft.Storage/storageAccounts',
variables('defaultStorageAccount').name)).primaryEndpoints.blob, 'https://', ''), '/', '')]]",
                    "isDefault": true,
                    "container": "[parameters('clusterName')]",
                    "key": "[listKeys(resourceId('Microsoft.Storage/storageAccounts',
variables('defaultStorageAccount').name), '2019-06-01').keys[0].value]"
                }
            ]
        },
        "computeProfile": {
            "roles": [
                {
                    "name": "headnode",
                    "targetInstanceCount": 2,
                    "hardwareProfile": {
                        "vmSize": "[parameters('HeadNodeVirtualMachineSize')]"
                    },
                    "osProfile": {
                        "linuxOperatingSystemProfile": {
                            "username": "[parameters('sshUserName')]",
                            "password": "[parameters('sshPassword')]"
                        }
                    }
                },
                {
                    "name": "workernode",
                    "targetInstanceCount": 2,
                    "hardwareProfile": {
                        "vmSize": "[parameters('WorkerVirtualMachineSize')]"
                    },
                    "osProfile": {
                        "linuxOperatingSystemProfile": {
                            "username": "[parameters('sshUserName')]",
                            "password": "[parameters('sshPassword')]"
                        }
                    }
                }
            ]
        }
    }
}
]

```

```

    "hardwareProfile": {
      "vmSize": "[parameters('WorkerNodeVirtualMachineSize')]"
    },
    "osProfile": {
      "linuxOperatingSystemProfile": {
        "username": "[parameters('sshUserName')]",
        "password": "[parameters('sshPassword')]"
      }
    }
  }
],
"outputs": {
  "storage": {
    "type": "object",
    "value": "[reference(resourceId('Microsoft.Storage/storageAccounts',
variables('defaultStorageAccount').name))]"
  },
  "cluster": {
    "type": "object",
    "value": "[reference(resourceId('Microsoft.HDInsight/clusters', parameters('clusterName')))]"
  }
}
}

```

Two Azure resources are defined in the template:

- [Microsoft.Storage/storageAccounts](#): create an Azure Storage Account.
- [Microsoft.HDInsight/cluster](#): create an HDInsight cluster.

Deploy the template

1. Select the **Deploy to Azure** button below to sign in to Azure and open the ARM template.



2. Enter or select the following values:

PROPERTY	DESCRIPTION
Subscription	From the drop-down list, select the Azure subscription that's used for the cluster.
Resource group	From the drop-down list, select your existing resource group, or select Create new .
Location	The value will autopopulate with the location used for the resource group.
Cluster Name	Enter a globally unique name. For this template, use only lowercase letters, and numbers.
Cluster Login User Name	Provide the username, default is admin .

PROPERTY	DESCRIPTION
Cluster Login Password	Provide a password. The password must be at least 10 characters in length and must contain at least one digit, one uppercase, and one lower case letter, one non-alphanumeric character (except characters ' " `).
Ssh User Name	Provide the username, default is sshuser
Ssh Password	Provide the password.

← → C portal.azure.com/?feature...

☰ Microsoft Azure ?

Home > Deploy a Spark cluster in Azure HDInsight

Deploy a Spark cluster in Azure HDInsight

Azure quickstart template

TEMPLATE

101-hdinsight-spark-linux
2 resources Edit template Edit paramet... Learn more

BASICS

Subscription *

Resource group *
[Create new](#)

Location

SETTINGS

Cluster Name *

Cluster Login User Name

Cluster Login Password *

Ssh User Name

Ssh Password *

Location

TERMS AND CONDITIONS

[Template information](#) | [Azure Marketplace Terms](#) | [Azure Marketplace](#)

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.

I agree to the terms and conditions stated above

Purchase

3. Review the **TERMS AND CONDITIONS**. Then select **I agree to the terms and conditions stated above**, then **Purchase**. You'll receive a notification that your deployment is in progress. It takes about 20 minutes to create a cluster.

If you run into an issue with creating HDInsight clusters, it could be that you don't have the right permissions to do so. For more information, see [Access control requirements](#).

Review deployed resources

Once the cluster is created, you'll receive a **Deployment succeeded** notification with a [Go to resource](#) link. Your Resource group page will list your new HDInsight cluster and the default storage associated with the cluster. Each cluster has an [Azure Storage](#), an [Azure Data Lake Storage Gen1](#), or an

[Azure Data Lake Storage Gen2](#) dependency. It's referred as the default storage account. HDInsight cluster and its default storage account must be colocated in the same Azure region. Deleting clusters doesn't delete the storage account dependency. It's referred as the default storage account. The HDInsight cluster and its default storage account must be colocated in the same Azure region. Deleting clusters doesn't delete the storage account.

Create a Jupyter Notebook file

[Jupyter Notebook](#) is an interactive notebook environment that supports various programming languages. You can use a Jupyter Notebook file to interact with your data, combine code with markdown text, and perform simple visualizations.

1. Open the [Azure portal](#).
2. Select **HDInsight clusters**, and then select the cluster you created.

The screenshot shows the Azure portal interface for managing HDInsight clusters. The top navigation bar includes 'Home > HDInsight clusters'. Below this, the title 'HDInsight clusters' is displayed with a Microsoft logo. There are buttons for 'Add', 'Edit columns', 'Refresh', and 'Assign tags'. A message indicates 'Subscriptions: 1 of 18 selected – Don't see a subscription? Open Directory + Subscription settings'. Below this, there are filter options: 'Filter by name...', '<Subscription...>', 'myspark201...', 'All locations', 'All tags', and 'No grouping'. A table lists one item: 'myspark20180403' (with a blue icon), 'myspark201804...', 'East US 2', and '<Subscription name> ***'. The 'myspark20180403' entry is highlighted with a red box.

3. From the portal, in Cluster dashboards section, select **Jupyter Notebook**. If prompted, enter the cluster login credentials for the cluster.

Dashboard > myspark20180403rg > myspark20180403

myspark20180403

HDInsight cluster

Search (Ctrl+ /)

Move Delete Refresh

Overview

- Activity log
- Access control (IAM)
- Tags
- Diagnose and solve problems
- Quick start
- Tools

Settings

- Cluster size
- Quota limits
- SSH + Cluster login
- Data Lake Storage Gen1
- Storage accounts
- Applications
- Script actions

Resource group ([change](#))
myspark20180403rg

Status
Running

Location
East US 2

Subscription ([change](#))
<Azure Subscription name>

Subscription ID
<Azure Subscription ID>

Tags ([change](#))
[Click here to add tags](#)

Learn more
[Documentation](#)

Cluster type, HDI version
Spark 2.3 (HDI 3.6)

URL
<https://myspark20180403.azurehdinsight.net>

Getting started
[Quickstart](#)

Cluster dashboards Cluster management interfaces

- Ambari home
- Ambari views
- Zeppelin notebook
- Jupyter notebook**
- Spark history server
- Yarn

- Select New > PySpark to create a notebook.

jupyter

Files Running Clusters

Select items to perform actions on them.

Upload New

Text File
Folder
Terminal

Notebooks
PySpark

PySpark3
Spark

A new notebook is created and opened with the name Untitled(Untitled.py).py

Run Apache Spark SQL statements

SQL (Structured Query Language) is the most common and widely used language for querying and transforming data. Spark SQL functions as an extension to Apache Spark for processing structured data, using the familiar SQL syntax.

- Verify the kernel is 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.

jupyter Untitled Last Checkpoint: in 8 hours (autosaved)

File Edit View Insert Cell Kernel Widgets Help

PySpark

alt-text="Kernel status"

border="true":

When you start the notebook for the first time, the kernel performs some tasks in the background. Wait for the kernel to be 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 file with your HDInsight cluster, you get a preset `spark` session that you can use to run Hive queries using Spark SQL. `%%sql` tells Jupyter Notebook to use the preset `spark` session 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. The first time you submit the query, Jupyter will create a Spark application for the notebook. It takes about 30 seconds to complete. Once the Spark application is ready, the query is executed in about a second and produces 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_1522771942160_0004	pyspark	idle	Link	Link	✓

SparkSession available as 'spark'.

Out[1]:

	database	tableName	isTemporary
0	default	hivesampletable	False

HDInsight" border="true":

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.

3. Run another query to see the data in `hivesampletable`.

```
%%sql  
SELECT * FROM hivesampletable LIMIT 10
```

The screen shall refresh to show the query output.

The screenshot shows a Jupyter notebook interface. In the top navigation bar, the title is "jupyter My first Jupyter notebook Last Checkpoint: Last Tuesday at 9:11 PM (unsaved changes)". Below the title is a toolbar with various icons for file operations. The main area has two code cells. The first cell, labeled "In [2]", contains the command "%sql SELECT * FROM hivesampletable LIMIT 10". The second cell, labeled "Out[2]", displays a table with 10 rows of data. The columns are: clientid, querytime, market, deviceplatform, devicemake, devicemodel, state, country, querydweltime, sessionid, and sessionpagevieworder. The data shows various mobile devices (Android, Samsung, SCH-I500) from the United States with session IDs ranging from 36 to 48.

border="true":| Insight"

border="true":::

- From the **File** menu on the notebook, select **Close and Halt**. Shutting down the notebook releases the cluster resources, including Spark application.

Clean up resources

After you complete the quickstart, you may want to delete the cluster. With HDInsight, your data is stored in Azure Storage, so you can safely delete a cluster when it isn't in use. You're also charged for an HDInsight cluster, even when it isn't in use. Since the charges for the cluster are many times more than the charges for storage, it makes economic sense to delete clusters when they aren't in use.

From the Azure portal, navigate to your cluster, and select **Delete**.

The screenshot shows the Azure portal's "Dashboard > myspark20180403rg > myspark20180403" page. The main area displays the "myspark20180403" cluster details. On the right, there is a "Delete" button highlighted with a red box. Below the cluster name, the resource group is listed as "Resource group (change) myspark20180403rg". To the right of the cluster details, there are links for "Learn more Documentation", "Cluster type, HDI version Spark 2.3 (HDI 3.6)", "URL https://myspark20180403.azurehdinsight.net", and "Getting started Quickstart".

sight cluster"

border="true":::

You can also select the resource group name to open the resource group page, and then select **Delete resource group**. By deleting the resource group, you delete both the HDInsight cluster, and the default storage account.

Next steps

In this quickstart, you learned how to create an Apache Spark cluster in HDInsight and run a basic Spark SQL query. Advance to the next tutorial to learn how to use an HDInsight cluster to run interactive queries on sample data.

Run interactive queries on Apache Spark

Kernels for Jupyter Notebook on Apache Spark clusters in Azure HDInsight

3/23/2021 • 7 minutes to read • [Edit Online](#)

HDInsight Spark clusters provide kernels that you can use with the Jupyter Notebook on [Apache 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](#), select your Spark cluster. See [List and show clusters](#) for the instructions. The **Overview** view opens.
2. From the **Overview** view, in the **Cluster dashboards** box, select **Jupyter notebook**. If prompted, enter the admin credentials for the cluster.

The screenshot shows the Microsoft Azure portal interface. At the top, there's a navigation bar with back, forward, refresh, and search icons, followed by the URL <https://portal.azure.com>. Below the URL is a search bar with placeholder text "Search resources, services, and docs (G+/-)". On the left, there's a sidebar with various icons for different services like Storage, Functions, and Logic Apps. The main content area shows the "mySpark" HDInsight cluster details. The "Overview" tab is selected. In the "Cluster dashboards" section, there's a list of management interfaces: Ambari home, Ambari views, Zeppelin notebook, **Jupyter notebook** (which is highlighted with a red box), Spark history server, and Yarn. Below this, there's a "Cluster size" section showing "3 nodes" with a table:

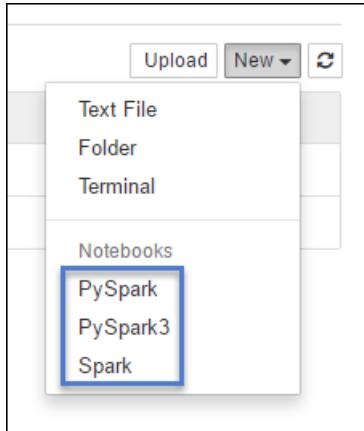
Type	↑↓	Size	↑↓	Cores	↑↓	Nodes	↑↓
Head		D12 v2		8		2	
Worker		D13 v2		8		1	

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
```

3. Select **New**, and then select 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.



4. 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 don't need to set the Spark or Hive contexts explicitly before you start working with your applications. These contexts 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 <args>**). 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
-------	---------	-------------

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
configure	<code>%%configure -f</code> <code>{"executorMemory": "1000M",</code> <code>"executorCores": 4 }</code>	Configures the parameters for creating a session. The force flag (<code>-f</code>) 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	<code>%%sql -o <variable name></code> <code>SHOW TABLES</code>	Executes a Hive query against the <code>sqlContext</code> . If the <code>-o</code> parameter is passed, the result of the query is persisted in the <code>%local</code> Python context as a Pandas dataframe.
local	<code>%%local</code> <code>a=1</code>	All the code in later lines is executed locally. Code must be valid Python2 code no matter which kernel you're 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	<code>%%logs</code>	Outputs the logs for the current Livy session.
delete	<code>%%delete -f -s <session number></code>	Deletes a specific session of the current Livy endpoint. You can't delete the session that is started for the kernel itself.
cleanup	<code>%%cleanup -f</code>	Deletes all the sessions for the current Livy endpoint, including this notebook's session. The force flag <code>-f</code> 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
<code>-o</code>	<code>-o <VARIABLE NAME></code>	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.
<code>-q</code>	<code>-q</code>	Use this parameter to turn off visualizations for the cell. If you don't want to autovisualize the content of a cell and just want to capture it as a dataframe, then use <code>-q -o <VARIABLE></code> . If you want to turn off visualizations without capturing the results (for example, for running a SQL query, like a <code>CREATE TABLE</code> statement), use <code>-q</code> without specifying a <code>-o</code> argument.
<code>-m</code>	<code>-m <METHOD></code>	Where METHOD is either take or sample (default is take). If the method is <code>take</code> , the kernel picks elements from the top of the result data set specified by MAXROWS (described later in this table). If the method is <code>sample</code> , the kernel randomly samples elements of the data set according to <code>-r</code> parameter, described next in this table.
<code>-r</code>	<code>-r <FRACTION></code>	Here FRACTION is a floating-point number between 0.0 and 1.0. If the sample method for the SQL query is <code>sample</code> , 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 <code>-m sample -r 0.01</code> , then 1% of the result rows are randomly sampled.
<code>-n</code>	<code>-n <MAXROWS></code>	MAXROWS is an integer value. The kernel limits the number of output rows to MAXROWS . If MAXROWS is a negative number such as <code>-1</code> , then the number of rows in the result set isn't limited.

Example:

```
%%sql -q -m sample -r 0.1 -n 500 -o query2
SELECT * FROM hivesamplable
```

The statement above does the following actions:

- Selects all records from **hivesamptable**.
- Because we use -q, it turns off autovisualization.
- Because we use `-m sample -r 0.1 -n 500`, it randomly samples 10% of the rows in the hivesamptable 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 doesn't kill the context. And so 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're finished using the notebook. The closure kills the context and then exits the notebook.

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 Storage as the default storage do not store notebooks in associated storage.

The way notebooks are saved to the storage account is compatible with [Apache Hadoop HDFS](#). If you SSH into the cluster you can use the file management commands:

COMMAND	DESCRIPTION
<code>hdfs dfs -ls /HdiNotebooks</code>	# List everything at the root directory – everything in this directory is visible to Jupyter from the home page
<code>hdfs dfs -copyToLocal /HdiNotebooks</code>	# Download the contents of the HdiNotebooks folder
<code>hdfs dfs -copyFromLocal example.ipynb /HdiNotebooks</code>	# Upload a notebook example.ipynb to the root folder so it's visible from Jupyter

Whether the cluster uses Azure Storage or Azure Data Lake Storage 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.

Suggestions

The new kernels are in evolving stage and will mature over time. So the APIs could change as these kernels mature. We would appreciate any feedback that you have while using these new kernels. The feedback is useful

in shaping the final release of these kernels. You can leave your comments/feedback under the **Feedback** section at the bottom of this article.

Next steps

- [Overview: Apache Spark on Azure HDInsight](#)
- [Use Apache Zeppelin notebooks with an Apache Spark cluster on HDInsight](#)
- [Use external packages with Jupyter Notebooks](#)
- [Install Jupyter on your computer and connect to an HDInsight Spark cluster](#)

What is ML Services in Azure HDInsight

3/5/2021 • 8 minutes to read • [Edit Online](#)

Microsoft Machine Learning Server is available as a deployment option when you create HDInsight clusters in Azure. The cluster type that provides this option is called **ML Services**. This capability provides on-demand access to adaptable, distributed methods of analytics on HDInsight.

ML Services on HDInsight provides the latest capabilities for R-based analytics on datasets of virtually any size. The datasets can be loaded to either Azure Blob or Data Lake storage. Your R-based applications can use the 8000+ open-source R packages. The routines in ScaleR, Microsoft's big data analytics package are also available.

The edge node provides a convenient place to connect to the cluster and run your R scripts. The edge node allows running the ScaleR parallelized distributed functions across the cores of the server. You can also run them across the nodes of the cluster by using ScaleR's Hadoop Map Reduce. You can also use Apache Spark compute contexts.

The models or predictions that result from analysis can be downloaded for on-premises use. They can also be [operationalized](#) elsewhere in Azure. In particular, through [Azure Machine Learning Studio \(classic\)](#), and [web service](#).

Get started with ML Services on HDInsight

To create an ML Services cluster in HDInsight, select the **ML Services** cluster type. The ML Services cluster type includes ML Server on the data nodes, and edge node. The edge node serves as a landing zone for ML Services-based analytics. See [Create Apache Hadoop clusters using the Azure portal](#) for a walkthrough on how to create the cluster.

Why choose ML Services in HDInsight?

ML Services in HDInsight provides the following benefits:

AI innovation from Microsoft and open-source

ML Services includes highly adaptable, distributed set of algorithms such as [RevoscaleR](#), [revoscalepy](#), and [microsoftML](#). These algorithms can work on data sizes larger than the size of physical memory. They also run on a wide variety of platforms in a distributed manner. Learn more about the collection of Microsoft's custom [R packages](#) and [Python packages](#) included with the product.

ML Services bridges these Microsoft innovations and contributions coming from the open-source community (R, Python, and AI toolkits). All on top of a single enterprise-grade platform. Any R or Python open-source machine learning package can work side by side with any proprietary innovation from Microsoft.

Simple, secure, and high-scale operationalization and administration

Enterprises relying on traditional paradigms and environments invest much time and effort towards operationalization. This action results in inflated costs and delays including the translation time for: models, iterations to keep them valid and current, regulatory approval, and managing permissions.

ML Services offers enterprise grade [operationalization](#). After a machine learning model completes, it takes just a few clicks to generate web services APIs. These [web services](#) are hosted on a server grid in the cloud and can be integrated with line-of-business applications. The ability to deploy to an elastic grid lets you scale seamlessly with the needs of your business, both for batch and real-time scoring. For instructions, see [Operationalize ML Services on HDInsight](#).

NOTE

The ML Services cluster type on HDInsight is supported only on HDInsight 3.6. HDInsight 3.6 is scheduled to retire on December 31, 2020.

Key features of ML Services on HDInsight

The following features are included in ML Services on HDInsight.

FEATURE CATEGORY	DESCRIPTION
R-enabled	R packages for solutions written in R, with an open-source distribution of R, and run-time infrastructure for script execution.
Python-enabled	Python modules for solutions written in Python, with an open-source distribution of Python, and run-time infrastructure for script execution.
Pre-trained models	For visual analysis and text sentiment analysis, ready to score data you provide.
Deploy and consume	Operationalize your server and deploy solutions as a web service.
Remote execution	Start remote sessions on ML Services cluster on your network from your client workstation.

Data storage options for ML Services on HDInsight

Default storage for the HDFS file system can be an Azure Storage account or Azure Data Lake Storage. Uploaded data to cluster storage during analysis is made persistent. The data is available even after the cluster is deleted. Various tools can handle the data transfer to storage. The tools include the portal-based upload facility of the storage account and the AzCopy utility.

You can enable access to additional Blob and Data lake stores during cluster creation. You aren't limited by the primary storage option in use. See [Azure Storage options for ML Services 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 file shares created in Azure Storage to the Linux file system. For more information, see [Azure Storage options for ML Services on HDInsight](#).

Access ML Services edge node

You can connect to Microsoft ML Server on the edge node using a browser, or SSH/PuTTY. The R console is installed by default during cluster creation.

Develop and run R scripts

Your R scripts can use any of the 8000+ open-source R packages. You can also use the parallelized and distributed routines from the ScaleR library. Scripts run on the edge node run within the R interpreter on that node. Except for steps that call ScaleR functions with a Map Reduce (RxHadoopMR) or Spark (RxSpark) compute context. The functions run in a distributed fashion across the data nodes that are associated with the data. For

more information about context options, see [Compute context options for ML Services on HDInsight](#).

Operationalize a model

When your data modeling is complete, `operationalize` the model to make predictions for new data either from Azure or 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. The function 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 this 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 Azure Machine Learning, 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 Azure Machine Learning to create a user interface for the web service, and then call the web service as needed for scoring. Then convert 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: serialize the model in R, download it, de-serialize it, then use it for scoring new data. You can score new data by using the approach described earlier in Score in HDInsight or by using [web services](#).

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 `install.packages()` method in R.

If you're just using ScaleR library routines, you don't usually need additional R packages. You might need additional packages for `rxExec` or `RxDataStep` execution on the data nodes.

The additional packages can be installed with a script action after you create the cluster. For more information, see [Manage ML Services in HDInsight cluster](#).

Change Apache Hadoop MapReduce memory settings

Available memory to ML Services can be modified when it's running a MapReduce job. To modify a cluster, use the Apache Ambari UI for your cluster. For Ambari UI instructions, see [Manage HDInsight clusters using the Ambari Web UI](#).

Available memory to ML Services can be changed by using Hadoop switches in the call to `RxHadoopMR`:

```
hadoopSwitches = "-libjars /etc/hadoop/conf -Dmapred.job.map.memory.mb=6656"
```

Scale your cluster

An existing ML Services cluster on HDInsight can be scaled up or down through the portal. By scaling up, you gain additional capacity for larger processing tasks. You can scale back a cluster when it's idle. For instructions about how to scale a cluster, see [Manage HDInsight clusters](#).

Maintain the system

OS Maintenance is done on the underlying Linux VMs in an HDInsight cluster during off-hours. Typically,

maintenance is done at 3:30 AM (VM's local time) every Monday and Thursday. Updates don't impact more than a quarter of the cluster at a time.

Running jobs might slow down during maintenance. However, they should still run to completion. Any custom software or local data that you've is preserved across these maintenance events unless a catastrophic failure occurs that requires a cluster rebuild.

IDE options for ML Services on HDInsight

The Linux edge node of an HDInsight cluster is the landing zone for R-based analysis. Recent versions of HDInsight provide a browser-based IDE of RStudio Server on the edge node. RStudio Server is more productive than the R console for development and execution.

A desktop IDE can access the cluster through a remote MapReduce or Spark compute context. Options include: Microsoft's [R Tools for Visual Studio](#) (RTVS), RStudio, and Walware's Eclipse-based StatET.

Access the R console on the edge node by typing `R` at the command prompt. When using the console interface, it's convenient to develop R script in a text editor. Then cut and paste sections of your script into the R console as needed.

Pricing

The prices associated with an ML Services HDInsight cluster are structured similarly to other HDInsight cluster types. They're based on the sizing of the underlying VMs across the name, data, and edge nodes. Core-hour uplifts as well. For more information, see [HDInsight pricing](#).

Next steps

To learn more about how to use ML Services on HDInsight clusters, see the following articles:

- [Execute an R script on an ML Services cluster in Azure HDInsight using RStudio Server](#)
- [Compute context options for ML Services cluster on HDInsight](#)
- [Storage options for ML Services cluster on HDInsight](#)

What is ML Services in Azure HDInsight

3/5/2021 • 8 minutes to read • [Edit Online](#)

Microsoft Machine Learning Server is available as a deployment option when you create HDInsight clusters in Azure. The cluster type that provides this option is called **ML Services**. This capability provides on-demand access to adaptable, distributed methods of analytics on HDInsight.

ML Services on HDInsight provides the latest capabilities for R-based analytics on datasets of virtually any size. The datasets can be loaded to either Azure Blob or Data Lake storage. Your R-based applications can use the 8000+ open-source R packages. The routines in ScaleR, Microsoft's big data analytics package are also available.

The edge node provides a convenient place to connect to the cluster and run your R scripts. The edge node allows running the ScaleR parallelized distributed functions across the cores of the server. You can also run them across the nodes of the cluster by using ScaleR's Hadoop Map Reduce. You can also use Apache Spark compute contexts.

The models or predictions that result from analysis can be downloaded for on-premises use. They can also be [operationalized](#) elsewhere in Azure. In particular, through [Azure Machine Learning Studio \(classic\)](#), and [web service](#).

Get started with ML Services on HDInsight

To create an ML Services cluster in HDInsight, select the **ML Services** cluster type. The ML Services cluster type includes ML Server on the data nodes, and edge node. The edge node serves as a landing zone for ML Services-based analytics. See [Create Apache Hadoop clusters using the Azure portal](#) for a walkthrough on how to create the cluster.

Why choose ML Services in HDInsight?

ML Services in HDInsight provides the following benefits:

AI innovation from Microsoft and open-source

ML Services includes highly adaptable, distributed set of algorithms such as [RevscaleR](#), [revoscalepy](#), and [microsoftML](#). These algorithms can work on data sizes larger than the size of physical memory. They also run on a wide variety of platforms in a distributed manner. Learn more about the collection of Microsoft's custom [R packages](#) and [Python packages](#) included with the product.

ML Services bridges these Microsoft innovations and contributions coming from the open-source community (R, Python, and AI toolkits). All on top of a single enterprise-grade platform. Any R or Python open-source machine learning package can work side by side with any proprietary innovation from Microsoft.

Simple, secure, and high-scale operationalization and administration

Enterprises relying on traditional paradigms and environments invest much time and effort towards operationalization. This action results in inflated costs and delays including the translation time for: models, iterations to keep them valid and current, regulatory approval, and managing permissions.

ML Services offers enterprise grade [operationalization](#). After a machine learning model completes, it takes just a few clicks to generate web services APIs. These [web services](#) are hosted on a server grid in the cloud and can be integrated with line-of-business applications. The ability to deploy to an elastic grid lets you scale seamlessly with the needs of your business, both for batch and real-time scoring. For instructions, see [Operationalize ML Services on HDInsight](#).

NOTE

The ML Services cluster type on HDInsight is supported only on HDInsight 3.6. HDInsight 3.6 is scheduled to retire on December 31, 2020.

Key features of ML Services on HDInsight

The following features are included in ML Services on HDInsight.

FEATURE CATEGORY	DESCRIPTION
R-enabled	R packages for solutions written in R, with an open-source distribution of R, and run-time infrastructure for script execution.
Python-enabled	Python modules for solutions written in Python, with an open-source distribution of Python, and run-time infrastructure for script execution.
Pre-trained models	For visual analysis and text sentiment analysis, ready to score data you provide.
Deploy and consume	Operationalize your server and deploy solutions as a web service.
Remote execution	Start remote sessions on ML Services cluster on your network from your client workstation.

Data storage options for ML Services on HDInsight

Default storage for the HDFS file system can be an Azure Storage account or Azure Data Lake Storage. Uploaded data to cluster storage during analysis is made persistent. The data is available even after the cluster is deleted. Various tools can handle the data transfer to storage. The tools include the portal-based upload facility of the storage account and the AzCopy utility.

You can enable access to additional Blob and Data lake stores during cluster creation. You aren't limited by the primary storage option in use. See [Azure Storage options for ML Services 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 file shares created in Azure Storage to the Linux file system. For more information, see [Azure Storage options for ML Services on HDInsight](#).

Access ML Services edge node

You can connect to Microsoft ML Server on the edge node using a browser, or SSH/PuTTY. The R console is installed by default during cluster creation.

Develop and run R scripts

Your R scripts can use any of the 8000+ open-source R packages. You can also use the parallelized and distributed routines from the ScaleR library. Scripts run on the edge node run within the R interpreter on that node. Except for steps that call ScaleR functions with a Map Reduce (RxHadoopMR) or Spark (RxSpark) compute context. The functions run in a distributed fashion across the data nodes that are associated with the data. For

more information about context options, see [Compute context options for ML Services on HDInsight](#).

Operationalize a model

When your data modeling is complete, `operationalize` the model to make predictions for new data either from Azure or 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. The function 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 this 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 Azure Machine Learning, 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 Azure Machine Learning to create a user interface for the web service, and then call the web service as needed for scoring. Then convert 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: serialize the model in R, download it, de-serialize it, then use it for scoring new data. You can score new data by using the approach described earlier in Score in HDInsight or by using [web services](#).

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 `install.packages()` method in R.

If you're just using ScaleR library routines, you don't usually need additional R packages. You might need additional packages for `rxExec` or `RxDataStep` execution on the data nodes.

The additional packages can be installed with a script action after you create the cluster. For more information, see [Manage ML Services in HDInsight cluster](#).

Change Apache Hadoop MapReduce memory settings

Available memory to ML Services can be modified when it's running a MapReduce job. To modify a cluster, use the Apache Ambari UI for your cluster. For Ambari UI instructions, see [Manage HDInsight clusters using the Ambari Web UI](#).

Available memory to ML Services can be changed by using Hadoop switches in the call to `RxHadoopMR`:

```
hadoopSwitches = "-libjars /etc/hadoop/conf -Dmapred.job.map.memory.mb=6656"
```

Scale your cluster

An existing ML Services cluster on HDInsight can be scaled up or down through the portal. By scaling up, you gain additional capacity for larger processing tasks. You can scale back a cluster when it's idle. For instructions about how to scale a cluster, see [Manage HDInsight clusters](#).

Maintain the system

OS Maintenance is done on the underlying Linux VMs in an HDInsight cluster during off-hours. Typically,

maintenance is done at 3:30 AM (VM's local time) every Monday and Thursday. Updates don't impact more than a quarter of the cluster at a time.

Running jobs might slow down during maintenance. However, they should still run to completion. Any custom software or local data that you've is preserved across these maintenance events unless a catastrophic failure occurs that requires a cluster rebuild.

IDE options for ML Services on HDInsight

The Linux edge node of an HDInsight cluster is the landing zone for R-based analysis. Recent versions of HDInsight provide a browser-based IDE of RStudio Server on the edge node. RStudio Server is more productive than the R console for development and execution.

A desktop IDE can access the cluster through a remote MapReduce or Spark compute context. Options include: Microsoft's [R Tools for Visual Studio](#) (RTVS), RStudio, and Walware's Eclipse-based StatET.

Access the R console on the edge node by typing R at the command prompt. When using the console interface, it's convenient to develop R script in a text editor. Then cut and paste sections of your script into the R console as needed.

Pricing

The prices associated with an ML Services HDInsight cluster are structured similarly to other HDInsight cluster types. They're based on the sizing of the underlying VMs across the name, data, and edge nodes. Core-hour uplifts as well. For more information, see [HDInsight pricing](#).

Next steps

To learn more about how to use ML Services on HDInsight clusters, see the following articles:

- [Execute an R script on an ML Services cluster in Azure HDInsight using RStudio Server](#)
- [Compute context options for ML Services cluster on HDInsight](#)
- [Storage options for ML Services cluster on HDInsight](#)

Create and share an Machine Learning Studio (classic) workspace

3/5/2021 • 4 minutes to read • [Edit Online](#)

APPLIES TO: Machine Learning Studio (classic) Azure Machine Learning

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

Create a Studio (classic) workspace

To open a workspace in Machine Learning Studio (classic), you must be signed in to the Microsoft Account you used to create the workspace, or you need to receive an invitation from the owner to join the workspace. From the Azure portal you can manage the workspace, which includes the ability to configure access.

1. Sign in to the [Azure portal](#)

NOTE

To sign in and create a Studio (classic) workspace, you need to be an Azure subscription administrator.

2. Click **+New**
3. In the search box, type **Machine Learning Studio (classic) 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**.

Machine Learning is currently available in a limited number of regions. If your subscription does not include one of these regions, you may see the error message, "You have no subscriptions in the allowed regions." To request that a region be added to your subscription, create a new Microsoft support request from the Azure portal, choose **Billing** as the problem type, and follow the prompts to submit your request.

NOTE

Machine Learning Studio (classic) relies on an Azure storage account that you provide to save intermediary data when it executes the workflow. After the workspace is created, if the storage account is deleted, or if the access keys are changed, the workspace will stop functioning and all experiments in that workspace will fail. If you accidentally delete the storage account, recreate the storage account with the same name in the same region as the deleted storage account and resync the access key. If you changed storage account access keys, resync the access keys in the workspace by using the Azure portal.

Once the workspace is deployed, you can open it in Machine Learning Studio (classic).

1. Browse to Machine Learning Studio (classic) 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 Studio (classic) workspace, see [Manage an Azure Machine Learning Studio \(classic\) workspace](#). If you encounter a problem creating your workspace, see [Troubleshooting guide: Create and connect to a Machine Learning Studio \(classic\) workspace](#).

Share an Azure Machine Learning Studio (classic) workspace

Once a Machine Learning Studio (classic) workspace is created, you can invite users to your workspace to share access to your workspace and all its experiments, datasets, 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 Studio (classic) workspace

1. Sign in to Machine Learning Studio (classic) 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. At the bottom of the sidebar is a 'NEW' button. The main area is titled 'settings'. It has tabs for NAME, AUTHORIZATION TOKENS, USERS (which is highlighted with a red box), and DATA GATEWAYS. Below these tabs is a table with columns for NAME, EMAIL, ROLE, and STATUS. One row is shown: luisa, luisa@contoso.com, Owner, Active. At the bottom right of the main area is a button labeled 'INVITE MORE USERS' with a red box around it. There's also a 'REMOVE' button.

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.

Troubleshoot storage accounts

The Machine Learning service needs a storage account to store data. You can use an existing storage account, or you can create a new storage account when you create the new Machine Learning Studio (classic) workspace (if you have quota to create a new storage account).

After the new Machine Learning Studio (classic) workspace is created, you can sign in to Machine Learning Studio (classic) by using the Microsoft account you used to create the workspace. If you encounter the error message, "Workspace Not Found" (similar to the following screenshot), please use the following steps to delete your browser cookies.

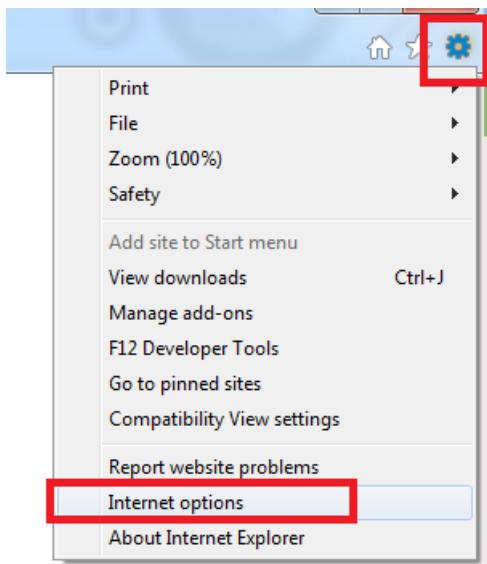
Workspace Not Found

SIGN OUT ⓘ
ABOUT MICROSOFT AZURE ML ⓘ
MACHINE LEARNING CENTER ⓘ
CONTACT MICROSOFT SUPPORT ⓘ

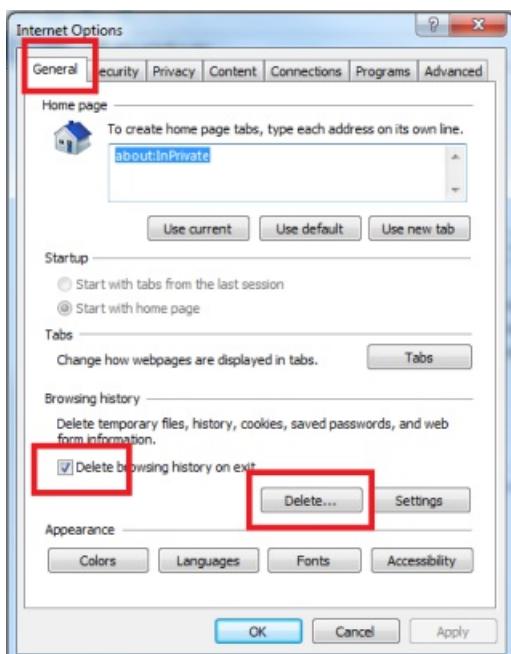
The user is not authorized for the workspace

To delete browser cookies

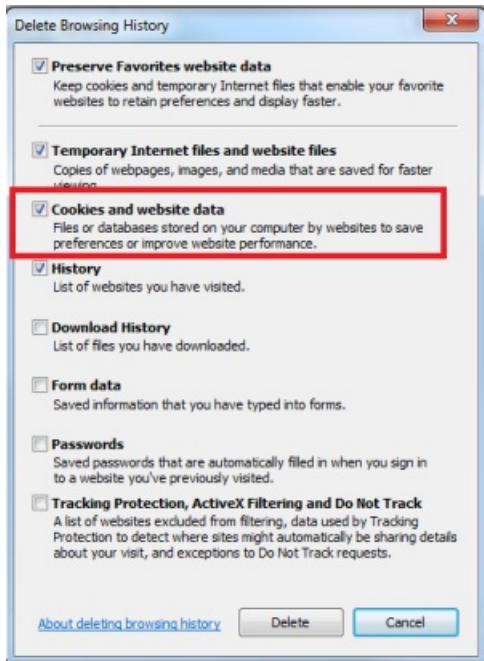
1. If you use Internet Explorer, click the **Tools** button in the upper-right corner and select **Internet options**.



2. Under the **General** tab, click **Delete...**



3. In the **Delete Browsing History** dialog box, make sure **Cookies and website data** is selected, and click **Delete**.



After the cookies are deleted, restart the browser and then go to the [Microsoft Azure Machine Learning Studio \(classic\)](#) page. When you are prompted for a user name and password, enter the same Microsoft account you used to create the workspace.

Next steps

For more information on managing a workspace, see [Manage an Azure Machine Learning Studio \(classic\) workspace](#).

How to identify scenarios and plan for advanced analytics data processing

3/5/2021 • 4 minutes to read • [Edit Online](#)

What resources are required for you to create an environment that can perform advanced analytics processing on a dataset? This article suggests a series of questions to ask that can help identify tasks and resources relevant to your scenario.

To learn about the order of high-level steps for predictive analytics, see [What is the Team Data Science Process \(TDSP\)](#). Each step requires specific resources for the tasks relevant to your particular scenario.

Answer key questions in the following areas to identify your scenario:

- data logistics
- data characteristics
- dataset quality
- preferred tools and languages

Logistic questions: data locations and movement

The logistic questions cover the following items:

- data source location
- target destination in Azure
- requirements for moving the data, including the schedule, amount, and resources involved

You may need to move the data 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.

What is your data source?

Is your data local or in the cloud? Possible locations include:

- a publicly available HTTP address
- a local or network file location
- a SQL Server database
- an Azure Storage container

What is the Azure destination?

Where does your data need to be for processing or modeling?

- 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

How are you going to move the data?

For procedures and resources to ingest or load data into a variety of different storage and processing environments, see:

- Load data into storage environments for analytics
- Import your training data into Azure Machine Learning Studio (classic) from various data sources

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. ADF can be helpful for:

- a hybrid scenario that involves both on-premises and cloud resources
- a scenario where the data is transacted, modified, or changed by business logic in the course of being migrated

For more information, see [Move data from a SQL Server database to SQL Azure with Azure Data Factory](#).

How much of the data is to be moved to Azure?

Large datasets may exceed the storage capacity of certain environments. For an example, see the discussion of size limits for Machine Learning Studio (classic) in the next section. In such cases, you might use a sample of the data 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. They will help you choose the appropriate scenario for your data type and understand any restrictions.

What are the data types?

- Numerical
- Categorical
- Strings
- Binary

How is your data formatted?

- Comma-separated (CSV) or tab-separated (TSV) flat files
- Compressed or uncompressed
- Azure blobs
- Hadoop Hive tables
- SQL Server tables

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 (classic) 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 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

What do you know about your data?

Understand the basic characteristics about your data:

- What patterns or trends it exhibits

- What outliers it has
- How many values are missing

This step is important to help you:

- Determine how much pre-processing is needed
- Formulate hypotheses that suggest the most appropriate features or type of analysis
- Formulate plans for additional data collection

Useful techniques for data inspection include descriptive statistics calculation and visualization plots. For details of how to explore a dataset in various Azure environments, see [Explore data in the Team Data Science Process](#).

Does the data require preprocessing or cleaning?

You might need to preprocess and clean your data before you can use the dataset effectively for machine learning. Raw data is often noisy and unreliable. It might 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 many options for languages, development environments, and tools. Be aware of your needs and preferences.

What languages do you prefer to use for analysis?

- R
- Python
- SQL

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

After 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](#).

Next steps

[What is the Team Data Science Process \(TDSP\)?](#)

Load data into storage environments for analytics

3/5/2021 • 2 minutes to read • [Edit Online](#)

The Team Data Science Process requires that data be ingested or loaded into the most appropriate way in each stage. Data destinations can include Azure Blob Storage, SQL Azure databases, SQL Server on Azure VM, HDInsight (Hadoop), Azure Synapse Analytics, and Azure Machine Learning.

The following articles describe how to ingest data into various target environments where the data is stored and processed.

- To/From [Azure Blob Storage](#)
- To [SQL Server on Azure VM](#)
- To [Azure SQL Database](#)
- To [Hive tables](#)
- To [SQL partitioned tables](#)
- From [On-premises SQL Server](#)

Technical and business needs, as well as the initial location, format, and size of your data will determine the best data ingestion plan. It is not uncommon for a best plan to have several steps. This sequence of tasks can include, for example, data exploration, pre-processing, cleaning, down-sampling, and model training. Azure Data Factory is a recommended Azure resource to orchestrate data movement and transformation.

Move data to and from Azure Blob storage

3/5/2021 • 2 minutes 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.

Different technologies for moving data

The following articles describe how to move data to and from Azure Blob storage using different technologies.

- [Azure Storage-Explorer](#)
- [AzCopy](#)
- [Python](#)
- [SSIS](#)

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](#).

Using Azure Data Factory

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 article 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

3/5/2021 • 2 minutes 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 your credentials.

Microsoft Azure Storage Explorer

Edit Help

Microsoft Azure



Show resources from these subscriptions:



Microsoft

@microsoft.com

Remove

All subscriptions:



[REDACTED]

[REDACTED]

Add an account...

Apply

Cancel



Upload Download

Open

Copy URL

← → ↻ ↺ container1

Name

Last Modified

3. To bring up the **Connect to Azure Storage** wizard, select the **Connect to Azure Storage** icon.

Microsoft Azure Storage Explorer

Edit Help

Microsoft Azure



Search for resources



◀ (Local and Attached)

▶ Storage Accounts



Upload

Download

Open

← → ↻ ↺ container1

Name

Last Modified

4. Enter the access key from your Azure Storage account on the **Connect to Azure Storage** wizard and then **Next**.

Connect to Azure Storage

Enter a connection string, Shared Access Signature (SAS) URI, or an account key.

Back

Next

Connect

Cancel

5. Enter storage account name in the **Account name** box and then select **Next**.

Attach External Storage

Enter information to connect to the Microsoft Azure storage account

Account name:

Account key:

Storage endpoints domain:

- Microsoft Azure Default
- Microsoft Azure China
- Other (specify below)

Use HTTP (Not recommended)

[Online privacy statement](#)

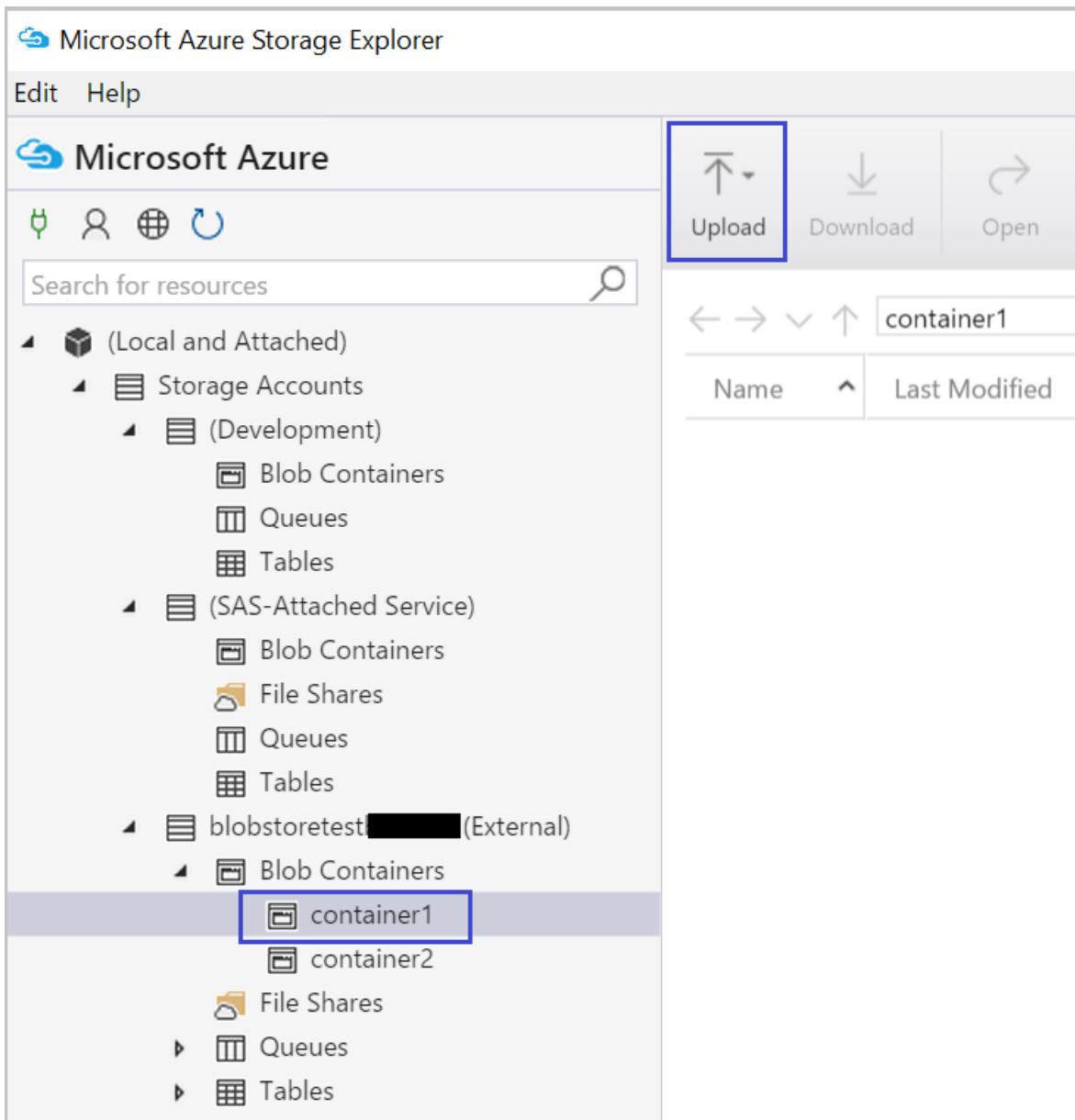
Back

Next

Connect

Cancel

6. The storage account added should now be displayed. 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, the navigation pane lists storage accounts and containers. A specific container named 'container1' is selected. On the right, the contents of 'container1' are displayed in a table format. A file named 'trip_data_1.csv.zip' is selected, and its details are shown in the preview pane at the bottom. The toolbar at the top features several buttons: Upload, Download (which is highlighted with a blue box), Open, Copy URL, Select all, and Copy.

Name	Last Modified
trip_data_1.csv.zip	Wed, 31 Aug 2016 14:31:13 GMT

Get started with AzCopy

3/17/2021 • 5 minutes to read • [Edit Online](#)

AzCopy is a command-line utility that you can use to copy blobs or files to or from a storage account. This article helps you download AzCopy, connect to your storage account, and then transfer files.

NOTE

AzCopy V10 is the currently supported version of AzCopy.

If you need to use a previous version of AzCopy, see the [Use the previous version of AzCopy](#) section of this article.

Download AzCopy

First, download the AzCopy V10 executable file to any directory on your computer. AzCopy V10 is just an executable file, so there's nothing to install.

- [Windows 64-bit \(zip\)](#)
- [Windows 32-bit \(zip\)](#)
- [Linux x86-64 \(tar\)](#)
- [macOS \(zip\)](#)

These files are compressed as a zip file (Windows and Mac) or a tar file (Linux). To download and decompress the tar file on Linux, see the documentation for your Linux distribution.

NOTE

If you want to copy data to and from your [Azure Table storage](#) service, then install [AzCopy version 7.3](#).

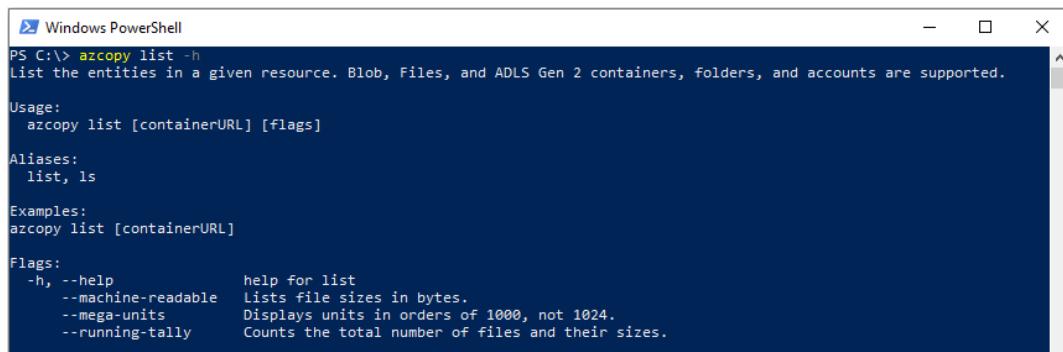
Run AzCopy

For convenience, consider adding the directory location of the AzCopy executable to your system path for ease of use. That way you can type `azcopy` from any directory on your system.

If you choose not to add the AzCopy directory to your path, you'll have to change directories to the location of your AzCopy executable and type `azcopy` or `.\azcopy` in Windows PowerShell command prompts.

To see a list of commands, type `azcopy -h` and then press the ENTER key.

To learn about a specific command, just include the name of the command (For example: `azcopy list -h`).



```
PS C:\> azcopy list -h
List the entities in a given resource. Blob, Files, and ADLS Gen 2 containers, folders, and accounts are supported.

Usage:
  azcopy list [containerURL] [flags]

Aliases:
  list, ls

Examples:
  azcopy list [containerURL]

Flags:
  -h, --help           Help for list
  --machine-readable   Lists file sizes in bytes.
  --mega-units         Displays units in orders of 1000, not 1024.
  --running-tally      Counts the total number of files and their sizes.
```

To find detailed reference documentation for each command and command parameter, see [azcopy](#)

NOTE

As an owner of your Azure Storage account, you aren't automatically assigned permissions to access data. Before you can do anything meaningful with AzCopy, you need to decide how you'll provide authorization credentials to the storage service.

Authorize AzCopy

You can provide authorization credentials by using Azure Active Directory (AD), or by using a Shared Access Signature (SAS) token.

Use this table as a guide:

STORAGE TYPE	CURRENTLY SUPPORTED METHOD OF AUTHORIZATION
Blob storage	Azure AD & SAS
Blob storage (hierarchical namespace)	Azure AD & SAS
File storage	SAS only

Option 1: Use Azure Active Directory

This option is available for blob Storage only. By using Azure Active Directory, you can provide credentials once instead of having to append a SAS token to each command.

NOTE

In the current release, if you plan to copy blobs between storage accounts, you'll have to append a SAS token to each source URL. You can omit the SAS token only from the destination URL. For examples, see [Copy blobs between storage accounts](#).

To authorize access by using Azure AD, see [Authorize access to blobs with AzCopy and Azure Active Directory \(Azure AD\)](#).

Option 2: Use a SAS token

You can append a SAS token to each source or destination URL that use in your AzCopy commands.

This example command recursively copies data from a local directory to a blob container. A fictitious SAS token is appended to the end of the container URL.

```
azcopy copy "C:\local\path" "https://account.blob.core.windows.net/mycontainer1/?sv=2018-03-28&ss=bjqt&srt=sco&sp=rwddgcup&se=2019-05-01T05:01:17Z&st=2019-04-30T21:01:17Z&spr=https&sig=MGCXiyEzbttkr3ewJlh2AR8KrghSy1DGM9ovN734bQF4%3D" --recursive=true
```

To learn more about SAS tokens and how to obtain one, see [Using shared access signatures \(SAS\)](#).

Transfer data

After you've authorized your identity or obtained a SAS token, you can begin transferring data.

NOTE

The [Secure transfer required](#) setting of a storage account determines whether the connection to a storage account is secured with Transport Layer Security (TLS). This setting is enabled by default.

To find example commands, see any of these articles.

SERVICE	ARTICLE
Azure Blob Storage	Upload files to Azure Blob Storage Download blobs from Azure Blob Storage Copy blobs between Azure storage accounts Synchronize with Azure Blob Storage
Azure Files	Transfer data with AzCopy and file storage
Amazon S3	Copy data from Amazon S3 to Azure Storage

Service	Article
Google Cloud Storage	Copy data from Google Cloud Storage to Azure Storage (preview)
Azure Stack storage	Transfer data with AzCopy and Azure Stack storage

Use in a script

Obtain a static download link

Over time, the AzCopy [download link](#) will point to new versions of AzCopy. If your script downloads AzCopy, the script might stop working if a newer version of AzCopy modifies features that your script depends upon.

To avoid these issues, obtain a static (unchanging) link to the current version of AzCopy. That way, your script downloads the same exact version of AzCopy each time that it runs.

To obtain the link, run this command:

Operating System	Command
Linux	<code>curl -s -D- https://aka.ms/downloadazcopy-v10-linux grep ^Location</code>
Windows	<code>(curl https://aka.ms/downloadazcopy-v10-windows -MaximumRedirection 0 -ErrorAction silentlyContinue).headers.location</code>

Note

For Linux, `--strip-components=1` on the `tar` command removes the top-level folder that contains the version name, and instead extracts the binary directly into the current folder. This allows the script to be updated with a new version of `azcopy` by only updating the `wget` URL.

The URL appears in the output of this command. Your script can then download AzCopy by using that URL.

Operating System	Command
Linux	<code>wget -O azcopy_v10.tar.gz https://aka.ms/downloadazcopy-v10-linux && tar -xf azcopy_v10.tar.gz --strip-components=1</code>
Windows	<code>Invoke-WebRequest https://azcopyvnext.azureedge.net/release20190517/azcopy_windows_amd64_ -OutFile azcopyv10.zip <<Unzip here>></code>

Escape special characters in SAS tokens

In batch files that have the `.cmd` extension, you'll have to escape the `%` characters that appear in SAS tokens. You can do that by adding an additional `%` character next to existing `%` characters in the SAS token string.

Run scripts by using Jenkins

If you plan to use [Jenkins](#) to run scripts, make sure to place the following command at the beginning of the script.

```
/usr/bin/keyctl new_session
```

Use in Azure Storage Explorer

[Storage Explorer](#) uses AzCopy to perform all of its data transfer operations. You can use [Storage Explorer](#) if you want to leverage the performance advantages of AzCopy, but you prefer to use a graphical user interface rather than the command line to interact with your files.

Storage Explorer uses your account key to perform operations, so after you sign into Storage Explorer, you won't need to provide additional authorization credentials.

Configure, optimize, and fix

See [Configure, optimize, and troubleshoot AzCopy](#)

Use a previous version

If you need to use the previous version of AzCopy, see either of the following links:

- [AzCopy on Windows \(v8\)](#)
- [AzCopy on Linux \(v7\)](#)

Next steps

If you have questions, issues, or general feedback, submit them [on GitHub](#) page.

Quickstart: Manage blobs with Python v12 SDK

3/23/2021 • 7 minutes to read • [Edit Online](#)

In this quickstart, you learn to manage blobs by using Python. Blobs are objects that can hold large amounts of text or binary data, including images, documents, streaming media, and archive data. You'll upload, download, and list blobs, and you'll create and delete containers.

More resources:

- [API reference documentation](#)
- [Library source code](#)
- [Package \(Python Package Index\)](#)
- [Samples](#)

Prerequisites

- An Azure account with an active subscription. [Create an account for free](#).
- An Azure Storage account. [Create a storage account](#).
- [Python](#) 2.7 or 3.6+.

NOTE

The features described in this article are now available to accounts that have a hierarchical namespace. To review limitations, see the [Blob storage features available in Azure Data Lake Storage Gen2](#) article.

Setting up

This section walks you through preparing a project to work with the Azure Blob Storage client library v12 for Python.

Create the project

Create a Python application named *blob-quickstart-v12*.

1. In a console window (such as cmd, PowerShell, or Bash), create a new directory for the project.

```
mkdir blob-quickstart-v12
```

2. Switch to the newly created *blob-quickstart-v12* directory.

```
cd blob-quickstart-v12
```

3. Inside the *blob-quickstart-v12* directory, create another directory called *data*. This directory is where the blob data files will be created and stored.

```
mkdir data
```

Install the package

While still in the application directory, install the Azure Blob Storage client library for Python package by using

the `pip install` command.

```
pip install azure-storage-blob
```

This command installs the Azure Blob Storage client library for Python package and all the libraries on which it depends. In this case, that is just the Azure core library for Python.

Set up the app framework

From the project directory:

1. Open a new text file in your code editor
2. Add `import` statements
3. Create the structure for the program, including basic exception handling

Here's the code:

```
import os, uuid
from azure.storage.blob import BlobServiceClient, BlobClient, ContainerClient, __version__
try:
    print("Azure Blob Storage v" + __version__ + " - Python quickstart sample")
    # Quick start code goes here
except Exception as ex:
    print('Exception:')
    print(ex)
```

4. Save the new file as `blob-quickstart-v12.py` in the `blob-quickstart-v12` directory.

Copy your credentials from the Azure portal

When the sample application makes a request to Azure Storage, it must be authorized. To authorize a request, add your storage account credentials to the application as a connection string. View your storage account credentials by following these steps:

1. Sign in to the [Azure portal](#).
2. Locate your storage account.
3. In the **Settings** section of the storage account overview, select **Access keys**. Here, you can view your account access keys and the complete connection string for each key.
4. Find the **Connection string** value under **key1**, and select the **Copy** button to copy the connection string. You will add the connection string value to an environment variable in the next step.



Configure your storage connection string

After you have copied your connection string, write it to a new environment variable on the local machine running the application. To set the environment variable, open a console window, and follow the instructions for your operating system. Replace `<yourconnectionstring>` with your actual connection string.

Windows

```
setx AZURE_STORAGE_CONNECTION_STRING "<yourconnectionstring>"
```

After you add the environment variable in Windows, you must start a new instance of the command window.

Linux

```
export AZURE_STORAGE_CONNECTION_STRING=<yourconnectionstring>
```

macOS

```
export AZURE_STORAGE_CONNECTION_STRING=<yourconnectionstring>
```

Restart programs

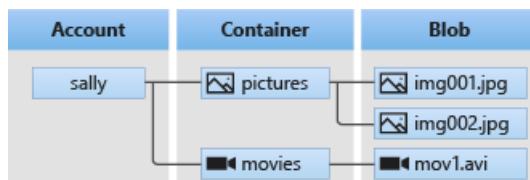
After you add the environment variable, restart any running programs that will need to read the environment variable. For example, restart your development environment or editor before continuing.

Object model

Azure Blob Storage is optimized for storing massive amounts of unstructured data. Unstructured data is data that doesn't adhere to a particular data model or definition, such as text or binary data. Blob storage offers three types of resources:

- The storage account
- A container in the storage account
- A blob in the container

The following diagram shows the relationship between these resources.



Use the following Python classes to interact with these resources:

- [BlobServiceClient](#): The `BlobServiceClient` class allows you to manipulate Azure Storage resources and blob containers.
- [ContainerClient](#): The `ContainerClient` class allows you to manipulate Azure Storage containers and their blobs.
- [BlobClient](#): The `BlobClient` class allows you to manipulate Azure Storage blobs.

Code examples

These example code snippets show you how to do the following tasks with the Azure Blob Storage client library for Python:

- [Get the connection string](#)
- [Create a container](#)
- [Upload blobs to a container](#)
- [List the blobs in a container](#)
- [Download blobs](#)
- [Delete a container](#)

Get the connection string

The code below retrieves the storage account connection string from the environment variable created in the [Configure your storage connection string](#) section.

Add this code inside the `try` block:

```
# Retrieve the connection string for use with the application. The storage
# connection string is stored in an environment variable on the machine
# running the application called AZURE_STORAGE_CONNECTION_STRING. If the environment variable is
# created after the application is launched in a console or with Visual Studio,
# the shell or application needs to be closed and reloaded to take the
# environment variable into account.
connect_str = os.getenv('AZURE_STORAGE_CONNECTION_STRING')
```

Create a container

Decide on a name for the new container. The code below appends a UUID value to the container name to ensure that it's unique.

IMPORTANT

Container names must be lowercase. For more information about naming containers and blobs, see [Naming and Referencing Containers, Blobs, and Metadata](#).

Create an instance of the `BlobServiceClient` class by calling the `from_connection_string` method. Then, call the `create_container` method to actually create the container in your storage account.

Add this code to the end of the `try` block:

```
# Create the BlobServiceClient object which will be used to create a container client
blob_service_client = BlobServiceClient.from_connection_string(connect_str)

# Create a unique name for the container
container_name = str(uuid.uuid4())

# Create the container
container_client = blob_service_client.create_container(container_name)
```

Upload blobs to a container

The following code snippet:

1. Creates a local directory to hold data files.
2. Creates a text file in the local directory.
3. Gets a reference to a `BlobClient` object by calling the `get_blob_client` method on the `BlobServiceClient` from the [Create a container](#) section.
4. Uploads the local text file to the blob by calling the `upload_blob` method.

Add this code to the end of the `try` block:

```

# Create a local directory to hold blob data
local_path = "./data"
os.mkdir(local_path)

# Create a file in the local data directory to upload and download
local_file_name = str(uuid.uuid4()) + ".txt"
upload_file_path = os.path.join(local_path, local_file_name)

# Write text to the file
file = open(upload_file_path, 'w')
file.write("Hello, World!")
file.close()

# Create a blob client using the local file name as the name for the blob
blob_client = blob_service_client.get_blob_client(container=container_name, blob=local_file_name)

print("\nUploading to Azure Storage as blob:\n\t" + local_file_name)

# Upload the created file
with open(upload_file_path, "rb") as data:
    blob_client.upload_blob(data)

```

List the blobs in a container

List the blobs in the container by calling the [list_blobs](#) method. In this case, only one blob has been added to the container, so the listing operation returns just that one blob.

Add this code to the end of the `try` block:

```

print("\nListing blobs...")

# List the blobs in the container
blob_list = container_client.list_blobs()
for blob in blob_list:
    print("\t" + blob.name)

```

Download blobs

Download the previously created blob by calling the [download_blob](#) method. The example code adds a suffix of "DOWNLOAD" to the file name so that you can see both files in local file system.

Add this code to the end of the `try` block:

```

# Download the blob to a local file
# Add 'DOWNLOAD' before the .txt extension so you can see both files in the data directory
download_file_path = os.path.join(local_path, str.replace(local_file_name ,'.txt', 'DOWNLOAD.txt'))
print("\nDownloading blob to \n\t" + download_file_path)

with open(download_file_path, "wb") as download_file:
    download_file.write(blob_client.download_blob().readall())

```

Delete a container

The following code cleans up the resources the app created by removing the entire container using the [delete_container](#) method. You can also delete the local files, if you like.

The app pauses for user input by calling `input()` before it deletes the blob, container, and local files. Verify that the resources were created correctly, before they're deleted.

Add this code to the end of the `try` block:

```
# Clean up
print("\nPress the Enter key to begin clean up")
input()

print("Deleting blob container...")
container_client.delete_container()

print("Deleting the local source and downloaded files...")
os.remove(upload_file_path)
os.remove(download_file_path)
os.rmdir(local_path)

print("Done")
```

Run the code

This app creates a test file in your local folder and uploads it to Azure Blob Storage. The example then lists the blobs in the container, and downloads the file with a new name. You can compare the old and new files.

Navigate to the directory containing the *blob-quickstart-v12.py* file, then execute the following `python` command to run the app.

```
python blob-quickstart-v12.py
```

The output of the app is similar to the following example:

```
Azure Blob Storage v12 - Python quickstart sample

Uploading to Azure Storage as blob:
    quickstartcf275796-2188-4057-b6fb-038352e35038.txt

Listing blobs...
    quickstartcf275796-2188-4057-b6fb-038352e35038.txt

Downloading blob to
    ./data/quickstartcf275796-2188-4057-b6fb-038352e35038DOWNLOAD.txt

Press the Enter key to begin clean up

Deleting blob container...
Deleting the local source and downloaded files...
Done
```

Before you begin the cleanup process, check your *data* folder for the two files. You can open them and observe that they're identical.

After you've verified the files, press the **Enter** key to delete the test files and finish the demo.

Next steps

In this quickstart, you learned how to upload, download, and list blobs using Python.

To see Blob storage sample apps, continue to:

[Azure Blob Storage SDK v12 Python samples](#)

- To learn more, see the [Azure Storage client libraries for Python](#).
- For tutorials, samples, quickstarts, and other documentation, visit [Azure for Python Developers](#).

Move data to or from Azure Blob Storage using SSIS connectors

3/5/2021 • 3 minutes 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 their data from any Azure service to leverage the full power of the suite of Azure technologies. The data may be subsequently used, for example, in Azure Machine Learning or on an HDInsight cluster.

Examples for using these Azure resources are in 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 need the 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 connectors 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](#).

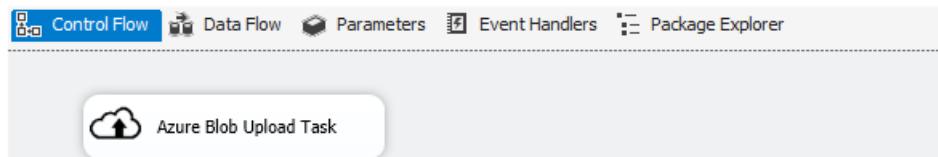
Download NYC Taxi dataset

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, each of which is about 2 GB 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 holds 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 is replaced.
LocalDirectory	Specifies the local directory that contains the files to be uploaded.
FileName	Specifies a name filter to select files with the specified name pattern. For example, MySheet*.xls* includes files such as MySheet001.xls and MySheetABC.xlsx.
TimeRangeFrom/TimeRangeTo	Specifies a time range filter. Files modified after <i>TimeRangeFrom</i> and before <i>TimeRangeTo</i> are included.

NOTE

The **AzureStorageConnection** credentials need to be correct and the **BlobContainer** must exist before the transfer is attempted.

Download data from Azure blob storage

To download data from Azure blob storage to on-premises storage with SSIS, use an instance of the [Azure Blob Download 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

3/5/2021 • 7 minutes to read • [Edit Online](#)

This article 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 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

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 storage account 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, which should work from the command line when SQL Server is installed on same machine:

```
bcp dbname..tablename in datafilename.tsv -f exportformatfilename.xml -S servername\sqlinstancename -U username -P password -b block_size_to_move_in_single_attempt -t \t -r \n
```

Optimizing BCP Inserts Please refer the following article '[Guidelines for Optimizing Bulk Import](#)' to optimize such inserts.

Parallelizing Inserts for Faster Data Movement

If the data you are moving is large, you can speed up things 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 file groups and partition tables. For more information about creating and loading data to partition tables, see [Parallel Load SQL Partition Tables](#).

The following sample PowerShell script demonstrates 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 -
    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 Integration 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 options 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 computer

```
bcp dbname..tablename format nul -c -x -f exportformatfilename.xml -S servername\sqlinstance -T -t \t -r \n
```

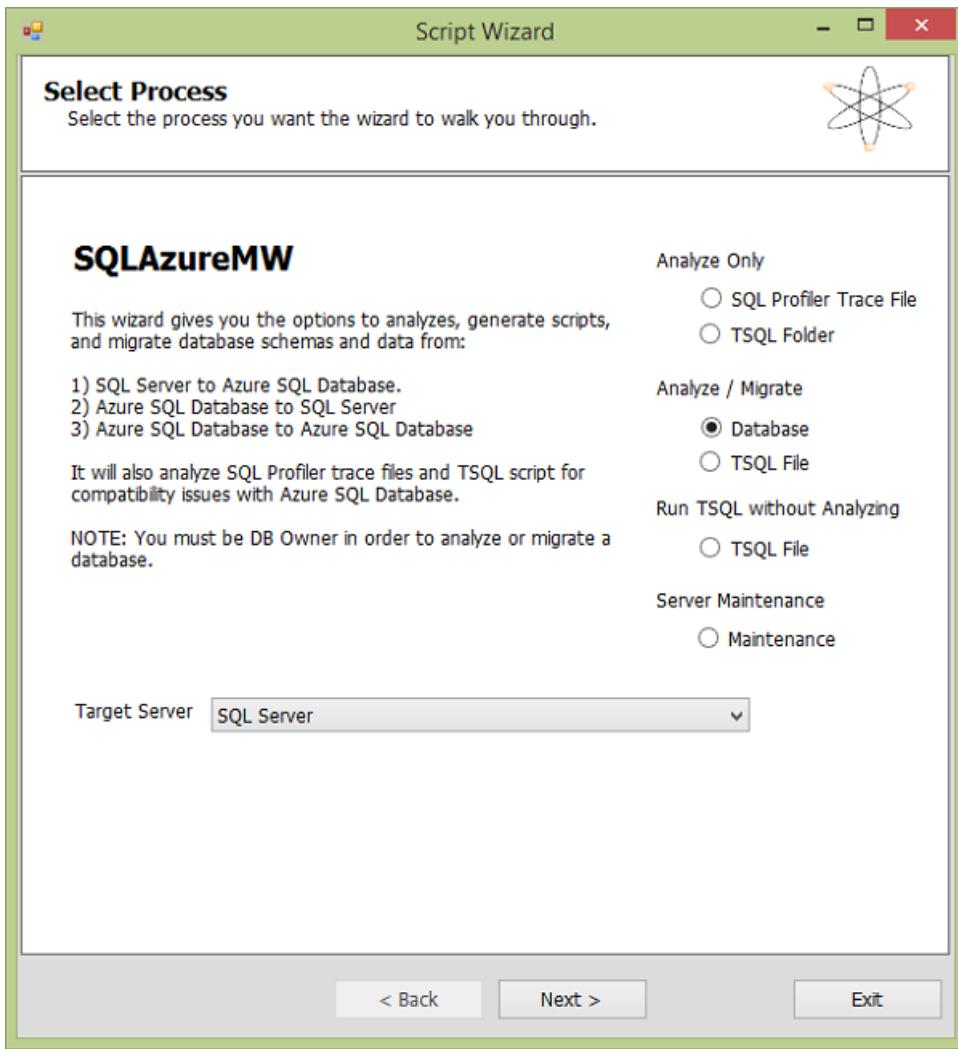
Format file generation when running BCP remotely against a SQL Server

```
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.

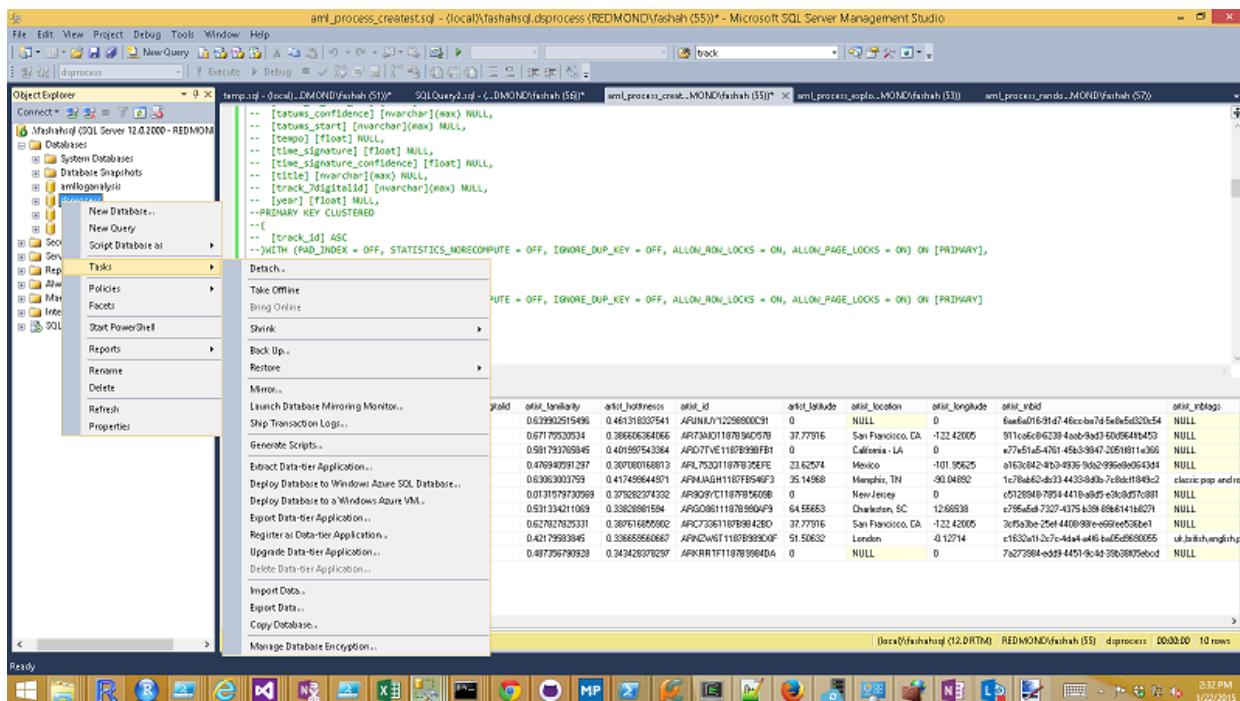


Database back up and restore

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 database in SQL Database. For more information, see [Use the Copy Database Wizard](#).

A screenshot of the Database back up/restore options from SQL Server Management Studio is shown below.



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

3/5/2021 • 3 minutes to read • [Edit Online](#)

This article outlines the options for moving data either from flat files (CSV or TSV formats) or from data stored in 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 SQL Server for Machine Learning, see [Move data to SQL Server on an Azure virtual machine](#).

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	1. Export to Flat File 2. SQL Database Migration Wizard 3. Database back up and restore 4. Azure 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 storage account 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 SQL Server database, follow the procedure outlined in [Bulk Import Data into SQL Server Database](#).

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 the directions 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 SQL Server to an Azure SQL Database

If the source data is stored in 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 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 directions covered in [Export to Flat File](#).

SQL Database Migration Wizard

The steps for using the SQL Database Migration Wizard are similar to those directions covered in [SQL Database Migration Wizard](#).

Database back up and restore

The steps for using database backup and restore are similar to those directions listed in [Database backup and restore](#).

Azure Data Factory

Learn how to move data to an Azure SQL Database with Azure Data Factory (ADF) in this topic, [Move data from a SQL Server to SQL Azure with Azure Data Factory](#). This topic shows how to use ADF to move data from a SQL Server database to an Azure SQL Database via Azure Blob Storage.

Consider using ADF when data needs to be continually migrated with hybrid on-premises and cloud sources. ADF also helps when the data needs transformations, or needs new business logic during migration. 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

11/2/2020 • 10 minutes to read • [Edit Online](#)

This article 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.

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 [Setup Clusters in HDInsight](#).
- Enabled remote access to the cluster, logged in, and opened the Hadoop Command-Line console. If you need instructions, see [Manage Apache Hadoop clusters](#).

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 provide a 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; options for such an account appear at [Use Azure Storage with Azure HDInsight clusters](#) 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:

- [Submit Hive queries through Hadoop Command Line in headnode of Hadoop cluster](#)
- [Submit Hive queries with the Hive Editor](#)
- [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.

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.

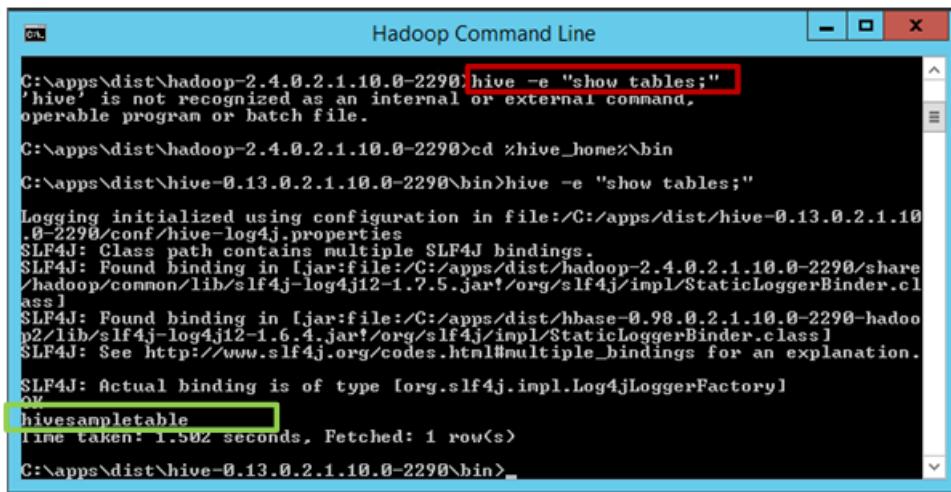
Log in to the head node of the Hadoop cluster, open the Hadoop Command Line on the desktop of the head node, and enter command `cd %hive_home%\bin`.

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.

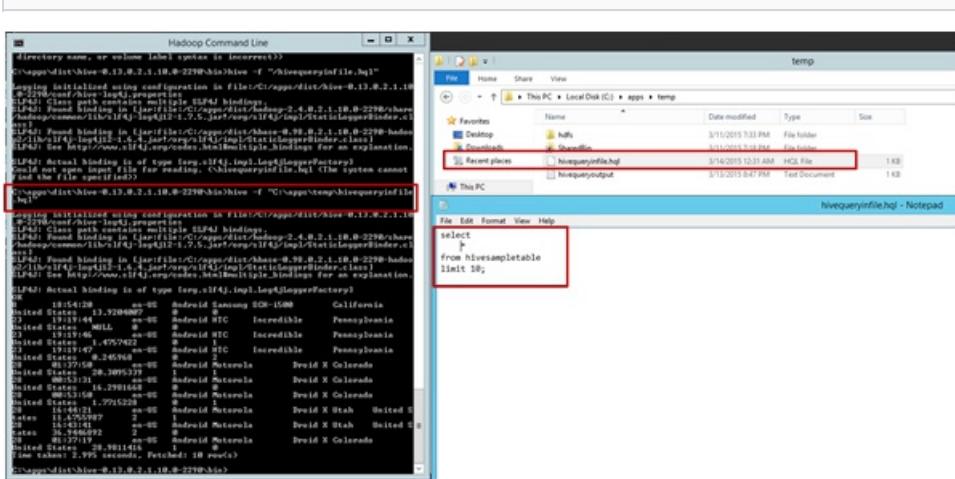


The screenshot shows a Windows Command Prompt window titled "Hadoop Command Line". The command entered is `hive -e "show tables;"`. The output shows the following error message: "hive' is not recognized as an internal or external command, operable program or batch file." This is because the Hive command needs to be run from the `%hive_home%\bin` directory. The next command, `cd %hive_home%\bin`, changes the directory. Finally, the command `hive -e "show tables;"` is run again successfully, displaying the results: "OK" and "hivesamptable". The entire command line is highlighted with a red box, and the output is highlighted with a green box.

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>"
```



The screenshot shows a Windows Command Prompt window titled "Hadoop Command Line". The command entered is `hive -f "C:\apps\temp\hivequeryinfile.hql"`. The output shows the same error message as the previous screenshot: "hive' is not recognized as an internal or external command, operable program or batch file." This is because the Hive command needs to be run from the `%hive_home%\bin` directory. The next command, `cd %hive_home%\bin`, changes the directory. Finally, the command `hive -f "C:\apps\temp\hivequeryinfile.hql"` is run successfully, displaying the results of the Hive query. The entire command line is highlighted with a red box, and the output is highlighted with a green box.

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 "<Hive queries>"
```

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;  
8 18:54:20 en-US Android Samsung SCH-i500 California  
United States 13.9204007 0 0  
23 19:19:44 en-US Android HTC Incredible Pennsylvania  
United States NULL 0 0  
23 19:19:46 en-US Android HTC Incredible Pennsylvania  
United States 1.4757422 0 1  
Time taken: 2.892 seconds. Fetched: 3 row(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`.

```
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 -e "select * from hivesampletable limit 100"  
Logging initialized using configuration in file:/C:/apps/dist/hive-0.13.0.2.1.10.0-2290/conf/hive-log4j.properties  
SLF4J: Class path contains multiple SLF4J bindings.  
SLF4J: Found binding in [jar:file:/C:/apps/dist/hadoop-2.4.0.2.1.10.0-2290/share/hadoop/common/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]  
SLF4J: Found binding in [jar:file:/C:/apps/dist/hbase-0.98.0.2.1.10.0-2290-hadoop2/lib/slf4j-log4j12-1.6.4.jar!/org/slf4j/impl/StaticLoggerBinder.class]  
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.  
SLF4J: Actual binding is of type org.slf4j.impl.Log4jLoggerFactory  
OK  
Time taken: 3.178 seconds. Fetched: 10 rows(s)  
C:\apps\dist\hive-0.13.0.2.1.10.0-2290\bin>hive -e "select * from hivesampletable limit 100" > 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  
OK  
Time taken: 25.081 seconds. Fetched: 10 rows(s)  
C:\apps\dist\hive-0.13.0.2.1.10.0-2290\bin>
```

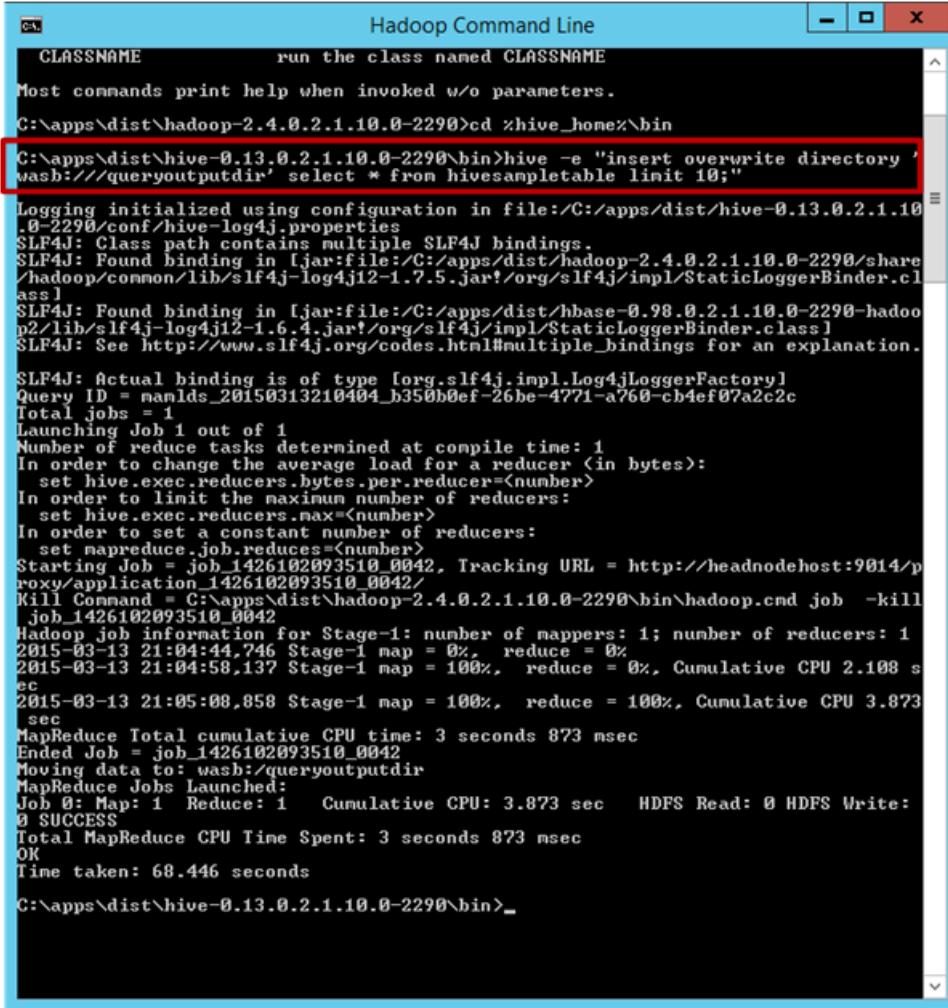
Output Hive query results to an Azure blob

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:

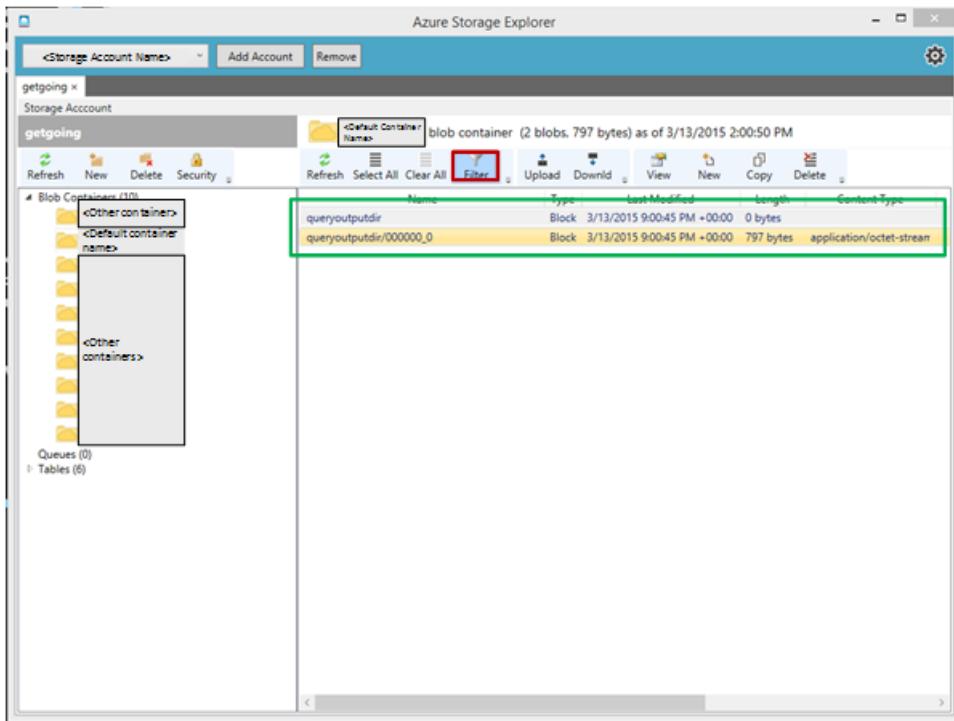
```
C:\apps\dist\hive-0.13.0.2.1.10.0-2290\bin>hive -e "insert overwrite directory 'wasb:///queryoutputdir' select * from hivesampletable limit 10;"
```

The output of the command is displayed below the command line. A red box highlights the error message:

```
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-hadoop-2/lib/slf4j-log4j12-1.6.4.jar!org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.

SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Query ID = namld5_20150313210404_b350b0ef-26be-4771-a760-cb4ef07a2c2c
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer <in bytes>:
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1426102093510_0042, Tracking URL = http://headnodehost:9014/proxy/application_1426102093510_0042/
Kill Command = C:\apps\dist\hadoop-2.4.0.2.1.10.0-2290\bin\hadoop.cmd job -kill job_1426102093510_0042
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2015-03-13 21:04:44,746 Stage-1 map = 0%, reduce = 0%
2015-03-13 21:04:58,137 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 2.108 s
ec
2015-03-13 21:05:08,858 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 3.873 sec
MapReduce Total cumulative CPU time: 3 seconds 873 msec
Ended Job = job_1426102093510_0042
Moving data to: wasb:/queryoutputdir
MapReduce Jobs Launched:
Job 0: Map: 1 Reduce: 1 Cumulative CPU: 3.873 sec    HDFS Read: 0 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 3 seconds 873 msec
OK
Time taken: 68.446 seconds
C:\apps\dist\hive-0.13.0.2.1.10.0-2290\bin>_
```

If you open the default container of the Hadoop cluster using Azure Storage Explorer, you can see the output of the Hive query as shown in the following figure. You can apply the filter (highlighted by red box) to only retrieve the blob with specified letters in names.



Submit Hive queries with the Hive Editor

You can also use the Query Console (Hive Editor) by entering a URL of the form <https://<Hadoop cluster name>.azurehdinsight.net/Home/HiveEditor> into a web browser. You must be logged in to see this console and so you need your Hadoop cluster credentials here.

Submit Hive queries with Azure PowerShell Commands

You can also use PowerShell to submit Hive queries. For instructions, see [Submit Hive jobs using PowerShell](#).

Create Hive database and tables

The Hive queries are shared in the [GitHub repository](#) and can be downloaded from there.

Here is the Hive query that creates a Hive table.

```
create database if not exists <database name>;
CREATE EXTERNAL TABLE if not exists <database name>.<table name>
(
    field1 string,
    field2 int,
    field3 float,
    field4 double,
    ...,
    fieldN string
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '<field separator>' lines terminated by '<line separator>'
STORED AS TEXTFILE LOCATION '<storage location>' TBLPROPERTIES("skip.header.line.count"="1");
```

Here are the descriptions of the fields that you need to plug in and other configurations:

- **<database name>**: the name of the database that you want to create. If you just want to use the default database, the query "create database..." can be omitted.
- **<table name>**: the name of the table that you want to create within the specified database. If you want to use the default database, the table can be directly referred by <table name> without <database name>.
- **<field separator>**: the separator that delimits fields in the data file to be uploaded to the Hive table.
- **<line separator>**: the separator that delimits lines in the data file.
- **<storage location>**: the Azure Storage location to save the data of Hive tables. If you do not specify

LOCATION <storage location>, the database and the tables are stored in *hive/warehouse*/directory in the default container of the Hive cluster by default. If you want to specify the storage location, the storage location has to be within the default container for the database and tables. This location has to be referred as location relative to the default container of the cluster in the format of '*wasb:///<directory 1>/*' or '*wasb:///<directory 1>/<directory 2>/*', etc. After the query is executed, the relative directories are created within the default container.

- **TBLPROPERTIES("skip.header.line.count" = "1")**: If the data file has a header line, you have to add this property at the end of the *create table* query. Otherwise, the header line is loaded as a record to the table. If the data file does not have a header line, this configuration can be omitted in the query.

Load data to Hive tables

Here is the Hive query that loads data into a Hive table.

```
LOAD DATA INPATH '<path to blob data>' INTO TABLE <database name>.<table name>;
```

- **<path to blob data>**: If the blob file to be uploaded to the Hive table is in the default container of the HDInsight Hadoop cluster, the *<path to blob data>* should be in the format '*wasb://<directory in this container>/<blob file name>*'. The blob file can also be in an additional container of the HDInsight Hadoop cluster. In this case, *<path to blob data>* should be in the format '*wasb://<container name>@<storage account name>.blob.core.windows.net/<blob file name>*'.

NOTE

The blob data to be uploaded to Hive table has to be in the default or additional container of the storage account for the Hadoop cluster. Otherwise, the *LOAD DATA* query fails complaining that it cannot access the data.

Advanced topics: partitioned table and store Hive data in ORC format

If the data is large, partitioning the table is beneficial for queries that only need to scan a few partitions of the table. For instance, it is reasonable to partition the log data of a web site by dates.

In addition to partitioning Hive tables, it is also beneficial to store the Hive data in the Optimized Row Columnar (ORC) format. For more information on ORC formatting, see [Using ORC files improves performance when Hive is reading, writing, and processing data](#).

Partitioned table

Here is the Hive query that creates a partitioned table and loads data into it.

```
CREATE EXTERNAL TABLE IF NOT EXISTS <database name>.<table name>
  (field1 string,
  ...
  fieldN string
)
PARTITIONED BY (<partitionfieldname> vartype) ROW FORMAT DELIMITED FIELDS TERMINATED BY '<field separator>'
  lines terminated by '<line separator>' TBLPROPERTIES("skip.header.line.count"="1");
LOAD DATA INPATH '<path to the source file>' INTO TABLE <database name>.<partitioned table name>
  PARTITION (<partitionfieldname>=<partitionfieldvalue>);
```

When querying partitioned tables, it is recommended to add the partition condition in the **beginning** of the `where` clause, which improves the search efficiency.

```

select
    field1, field2, ..., fieldN
from <database name>.<partitioned table name>
where <partitionfieldname>=<partitionfieldvalue> and ...;
```

Store Hive data in ORC format

You cannot directly load data from blob storage into Hive tables that is stored in the ORC format. Here are the steps that you need to take to load data from Azure blobs to Hive tables stored in ORC format.

Create an external table STORED AS TEXTFILE and load data from blob storage to the table.

```

CREATE EXTERNAL TABLE IF NOT EXISTS <database name>.<external textfile table name>
(
    field1 string,
    field2 int,
    ...
    fieldN date
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '<field separator>'
lines terminated by '<line separator>' STORED AS TEXTFILE
LOCATION 'wasb:///<directory in Azure blob>' TBLPROPERTIES("skip.header.line.count"="1");

LOAD DATA INPATH '<path to the source file>' INTO TABLE <database name>.<table name>;
```

Create an internal table with the same schema as the external table in step 1, with the same field delimiter, and store the Hive data in the ORC format.

```

CREATE TABLE IF NOT EXISTS <database name>.<ORC table name>
(
    field1 string,
    field2 int,
    ...
    fieldN date
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '<field separator>' STORED AS ORC;
```

Select data from the external table in step 1 and insert into the ORC table

```

INSERT OVERWRITE TABLE <database name>.<ORC table name>
SELECT * FROM <database name>.<external textfile table name>;
```

NOTE

If the TEXTFILE table <database name>.<external textfile table name> has partitions, in STEP 3, the `SELECT * FROM <database name>.<external textfile table name>` command selects the partition variable as a field in the returned data set. Inserting it into the <database name>.<ORC table name> fails since <database name>.<ORC table name> does not have the partition variable as a field in the table schema. In this case, you need to specifically select the fields to be inserted to <database name>.<ORC table name> as follows:

```

INSERT OVERWRITE TABLE <database name>.<ORC table name> PARTITION (<partition variable>=<partition value>)
SELECT field1, field2, ..., fieldN
FROM <database name>.<external textfile table name>
WHERE <partition variable>=<partition value>;
```

It is safe to drop the <external text file table name> when using the following query after all data has been

inserted into <database name>.<ORC table name>:

```
DROP TABLE IF EXISTS <database name>.<external textfile table name>;
```

After following this procedure, you should have a table with data in the ORC format ready to use.

Build and optimize tables for fast parallel import of data into a SQL Server on an Azure VM

3/5/2021 • 5 minutes to read • [Edit Online](#)

This article 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 database 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 = ''FileGroup2'', FILENAME = ''' + @data_path + '<file_name_2>.ndf' , SIZE = 4096KB ,
FILEGROWTH = 1024KB ),
        FILEGROUP [filegroup_3]
            ( NAME = ''FileGroup3'', FILENAME = ''' + @data_path + '<file_name_3>.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 1 TB 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](#).

Move data from a SQL Server database to SQL Database with Azure Data Factory

3/10/2021 • 8 minutes to read • [Edit Online](#)

This article shows how to move data from a SQL Server database to Azure SQL Database via Azure Blob Storage using the Azure Data Factory (ADF): this method is a supported legacy approach that has the advantages of a replicated staging copy, though [we suggest to look at our data migration page for the latest options](#).

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 needs transformations 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 a SQL Server database and Azure SQL Database. The two activities are:

- Copy data from a SQL Server database to an Azure Blob Storage account
- Copy data from the Azure Blob Storage account to Azure SQL Database.

NOTE

The steps shown here have been adapted from the more detailed tutorial provided by the ADF team: [Copy data from a SQL Server database to Azure Blob storage](#). 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 storage account 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 SQL Server instance

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_far.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 SQL Server database, follow the procedure outlined in [Bulk Import Data into SQL Server database](#).

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 Azure Data Factory Integration Runtime

The Integration Runtime is a customer-managed data integration infrastructure used by Azure Data Factory to provide data integration capabilities across different network environments. This runtime was formerly called "Data Management Gateway".

To set up, [follow the instructions for creating a pipeline](#)

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 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 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 subselect on the column names by including them here (for details check the [ADF documentation](#) topic).

Copy the JSON definition of the table into a file called *onpremtabledef.json* file and save it to a known location (here assumed to be *C:\temp\onpremtabledef.json*). Create the table in ADF with the following Azure PowerShell cmdlet:

```
New-AzureDataFactoryTable -ResourceGroupName ADFdsprg -DataFactoryName ADFdsp -File C:\temp\onpremtabledef.json
```

Blob Table

Definition for the table for the output blob location is in the following (this maps the ingested data from on-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 adfdsprg -DataFactoryName adfdsp -File C:\temp\bloboutputtabledef.json
```

SQL Azure Table

Definition for the table for the SQL Azure output is in the following (this schema maps the data coming from the blob):

```
{
  "name": "OutputSQLAzureTable",
  "properties":
  {
    "structure":
    [
      { "name": "column1", "type": "String"},  

      { "name": "column2", "type": "String"}
    ],
    "location":
    {
      "type": "AzureSqlTableLocation",
      "tableName": "your_db_name",
      "linkedServiceName": "adfdssqlazure_linked_servicename"
    },
    "availability":
    {
      "frequency": "Day",
      "interval": 1
    }
  }
}
```

Copy the JSON definition of the table into a file called *AzureSqlTable.json* file and save it to a known location (here assumed to be *C:\temp\AzureSqlTable.json*). Create the table in ADF with the following Azure PowerShell cmdlet:

```
New-AzureDataFactoryTable -ResourceGroupName adfdsp -DataFactoryName adfdsp -File  
C:\temp\AzureSqlTable.json
```

Define and create the pipeline

Specify the activities that belong to the pipeline and create the pipeline with the following script-based procedures. A JSON file is used to define the pipeline properties.

- The script assumes that the **pipeline name** is *AMLDSPipeline*.
- Also note that we set the periodicity of the pipeline to be executed on daily basis and use the default execution time for the job (12 am UTC).

NOTE

The following procedures use Azure PowerShell to define and create the ADF pipeline. But this task can also be accomplished using the Azure portal. For details, see [Create pipeline](#).

Using the table definitions provided previously, the pipeline definition for the ADF is specified as follows:

```
{
  "name": "AMLDSPProcessPipeline",
  "properties":
  {
    "description" : "This pipeline has two activities: the first one copies data from SQL Server to Azure Blob, and the second one copies from Azure Blob to Azure Database Table",
    "activities":
    [
      {
        "name": "CopyFromSQLtoBlob",
        "description": "Copy data from SQL Server to blob",
        "type": "CopyActivity",
        "inputs": [ {"name": "OnPremSQLTable"} ],
        "outputs": [ {"name": "OutputBlobTable"} ],
        "transformation":
        {
          "source":
          {
            "type": "SqlSource",
            "sqlReaderQuery": "select * from nyctaxi_data"
          },
          "sink":
          {
            "type": "BlobSink"
          }
        },
        "Policy":
        {
          "concurrency": 3,
          "executionPriorityOrder": "NewestFirst",
          "style": "StartOfInterval",
          "retry": 0,
          "timeout": "01:00:00"
        }
      },
      {
        "name": "CopyFromBlobtoSQLAzure",
        "description": "Push data to Sql Azure",
        "type": "CopyActivity",
        "inputs": [ {"name": "OutputBlobTable"} ],
        "outputs": [ {"name": "OutputSQLAzureTable"} ],
        "transformation":
        {
          "source":
          {
            "type": "BlobSource"
          },
          "sink":
          {
            "type": "SqlSink",
            "WriteBatchTimeout": "00:5:00",
          }
        },
        "Policy":
        {
          "concurrency": 3,
          "executionPriorityOrder": "NewestFirst",
          "style": "StartOfInterval",
          "retry": 2,
          "timeout": "02:00:00"
        }
      }
    ]
  }
}
```

Copy this JSON definition of the pipeline into a file called *pipelinedef.json* file and save it to a known location (here assumed to be *C:\temp\pipelinedef.json*). Create the pipeline in ADF with the following Azure PowerShell cmdlet:

```
New-AzureDataFactoryPipeline -ResourceGroupName adfdsp -DataFactoryName adfdsp -File C:\temp\pipelinedef.json
```

Start the Pipeline

The pipeline can now be run using the following command:

```
Set-AzureDataFactoryPipelineActivePeriod -ResourceGroupName AFDsp -DataFactoryName AFDsp -StartTime startdateZ -EndTime enddateZ -Name AMLDSProcessPipeline
```

The *startdate* and *enddate* parameter values need to be replaced with the actual dates between which you want the pipeline to run.

Once the pipeline executes, you should be able to see the data show up in the container selected for the blob, one file per day.

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](#).

Tasks to prepare data for enhanced machine learning

1/23/2020 • 6 minutes to read • [Edit Online](#)

Pre-processing and cleaning data are important tasks that must be conducted before a dataset can be used for model training. 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 (classic), 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 Studio (classic).

For an example of data exploration and pre-processing done inside Azure Machine Learning Studio (classic), see the [Pre-processing data](#) 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-formed** 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. For example, if the data contains student GPA (grade point average), 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 that 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 Studio (classic) in the Machine Learning. 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 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 that may cause data misalignment, for example, embedded tabs in a tab-separated data file, embedded new lines that may break records, for example.

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 rescales 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 that affect columns alignment and/or record boundaries. For example, embedded tabs in a tab-separated file cause column misalignment, and embedded new line characters break record lines. Improper text encoding handling while writing or reading text leads to information loss, inadvertent introduction of unreadable characters (like 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 process also helps you decide on the data processing steps that need to be taken to resolve them. Identifying the final use cases and personas can also be used to prioritize the data processing effort.

References

Data Mining: Concepts and Techniques, Third Edition, Morgan Kaufmann, 2011, Jiawei Han, Micheline Kamber, and Jian Pei

Explore data in the Team Data Science Process

1/23/2020 • 2 minutes to read • [Edit Online](#)

Exploring data is a step in the [Team Data Science Process](#).

The following articles describe how to explore data in three different storage environments that are typically used in the Data Science Process:

- Explore [Azure blob container](#) data using the [Pandas](#) Python package.
- Explore [SQL Server](#) data by using SQL and by using a programming language like Python.
- Explore [Hive table](#) data using Hive queries.

In addition, the video, [Preprocessing Data in Azure Machine Learning Studio](#), describes the commonly used modules for cleaning and transforming data.

Explore data in Azure Blob Storage with pandas

3/5/2021 • 2 minutes to read • [Edit Online](#)

This article covers how to explore data that is stored in Azure blob container using [pandas](#) Python package.

This task is a step in the [Team Data Science Process](#).

Prerequisites

This article assumes that you have:

- Created an Azure storage account. If you need instructions, see [Create an Azure Storage account](#)
- Stored your data in an Azure Blob Storage account. If you need instructions, see [Moving data to and from Azure Storage](#)

Load the data into a pandas DataFrame

To explore and manipulate a dataset, it must first be downloaded from the blob source to a local file, which can then be loaded in a pandas DataFrame. Here are the steps to follow for this procedure:

1. Download the data from Azure blob with the following Python code sample using Blob service. Replace the variable in the following code with your specific values:

```
from azure.storage.blob import BlockBlobService
import pandas as pd
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=BlockBlobService(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 DataFrame from the downloaded file.

```
# LOCALFILE is the file path
dataframe_blobdata = pd.read_csv(LOCALFILENAME)
```

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

11/2/2020 • 2 minutes to read • [Edit Online](#)

This article covers how to explore data that is stored in a SQL Server VM on Azure. Use SQL or Python to examine the data.

This task is a step in the [Team Data Science Process](#).

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](#). Load the data 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

3/5/2021 • 2 minutes to read • [Edit Online](#)

This article provides sample Hive scripts that are used to explore data in Hive tables in an HDInsight Hadoop cluster.

This task is a step in the [Team Data Science Process](#).

Prerequisites

This article assumes that you have:

- Created an Azure storage account. If you need instructions, see [Create an Azure Storage account](#)
- 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 <dbname>.<tblname> group by <partitionfieldname>;
```

2. Get the count of observations per day

```
SELECT to_date(<date_columnname>), count(*) from <dbname>.<tblname> group by  
to_date(<date_columnname>);
```

3. Get the levels in a categorical column

```
SELECT distinct <column_name> from <dbname>.<tblname>
```

4. Get the number of levels in combination of two categorical columns

```
SELECT <column_a>, <column_b>, count(*) from <dbname>.<tblname> group by <column_a>,  
<column_b>
```

5. Get the distribution for numerical columns

```
SELECT <column_name>, count(*) from <dbname>.<tblname> 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

3/5/2021 • 2 minutes to read • [Edit Online](#)

The following articles describe 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.

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. Downsizing may facilitate data understanding, exploration, and feature engineering. This sampling 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

3/5/2021 • 2 minutes to read • [Edit Online](#)

This article covers sampling data stored in Azure Blob Storage by downloading it programmatically and then sampling it using procedures written in Python.

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. Sampling 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 one 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>

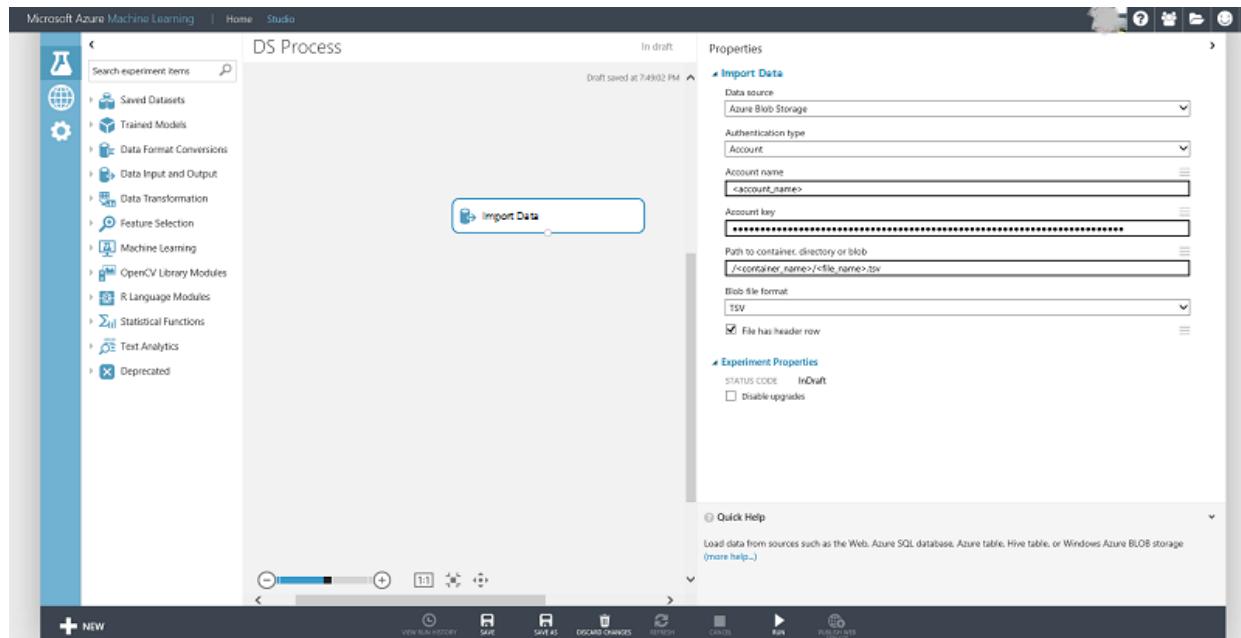
output_blob_service=BlobService(account_name=STORAGEACCOUNTNAME,account_key=STORAGEACCOUNTKEY)
localfileprocessed = os.path.join(os.getcwd(),LOCALFILENAME) #assuming file is in current working directory

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

3/5/2021 • 3 minutes 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.

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. Sampling 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 autogenerated 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 option 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.

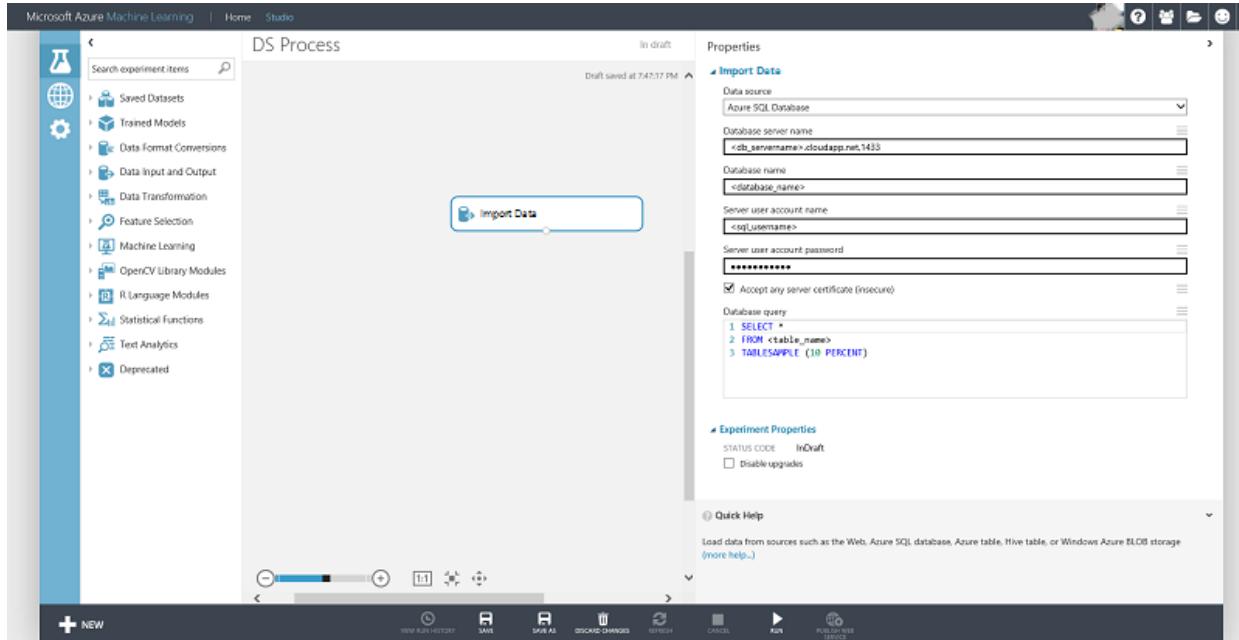
```
SELECT *  
FROM <table_name>  
TABLESAMPLE (10 PERCENT)
```

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 screenshot 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>

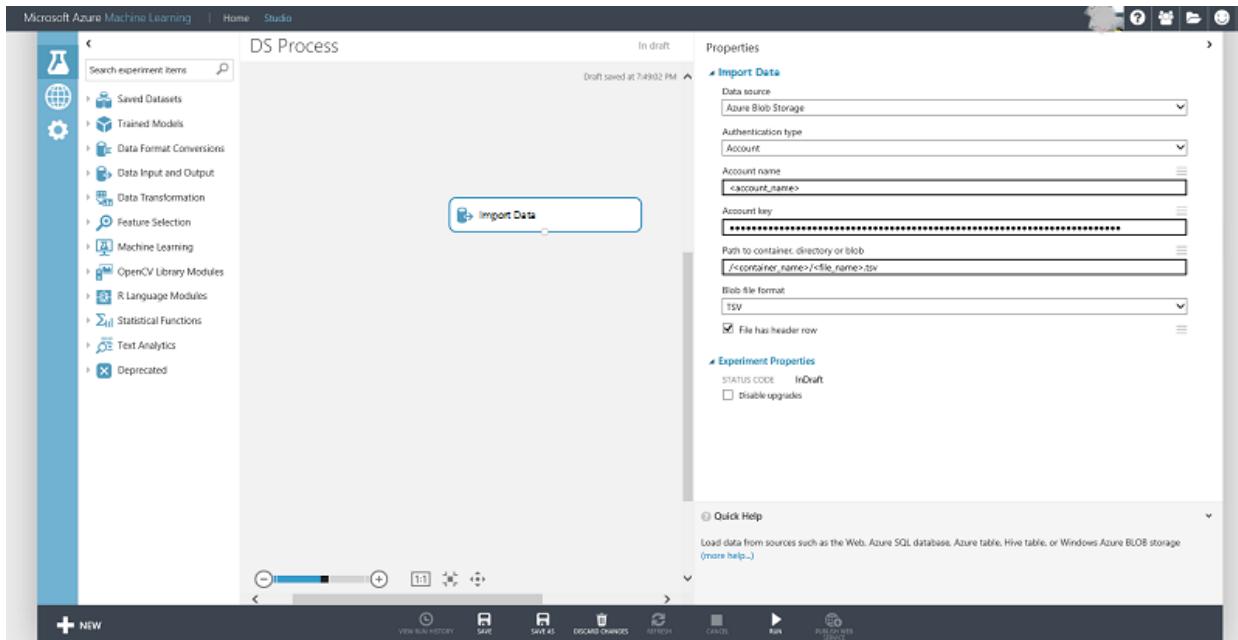
output_blob_service=BlobService(account_name=STORAGEACCOUNTNAME,account_key=STORAGEACCOUNTKEY)
localfileprocessed = os.path.join(os.getcwd(),LOCALFILENAME) #assuming file is in current working directory

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 walk through 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

3/5/2021 • 2 minutes 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

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. 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

3/23/2021 • 8 minutes 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

Prerequisites

The Python client library has been tested under the following environments:

- Windows, Mac, and Linux
- Python 2.7 and 3.6+

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

Install the Azure Machine Learning Python client library to complete the tasks outlined in this topic. This library 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 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 Azure Machine Learning Studio (classic) web interface, you can generate code snippets that include all the necessary information to download and deserialize datasets as pandas DataFrame objects on your local machine.

Security for data access

The code snippets provided by Azure Machine Learning Studio (classic) 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 (classic) on the **USERS** page under **Settings**.

The screenshot shows the 'settings' page in the Azure Machine Learning Studio (classic). On the left, there is a sidebar with icons for EXPERIMENTS, WEB SERVICES, MODULES, DATASETS, TRAINED MODELS, and SETTINGS. The SETTINGS icon is highlighted. The main area is titled 'settings' and contains a table for 'USERS'. The table has columns for NAME, AUTHORIZATION TOKENS, and USERS. The 'NAME' column lists a user with a blacked-out name. The 'ROLE' column shows 'Owner'. The 'STATUS' column shows 'Active'. There is a small 'p' icon in the top right corner of the table.

If your role is not set as **Owner**, you can either request to be reinvited as an owner, or ask the owner of the workspace to provide you with the code snippet.

To obtain the authorization token, you may choose one of these options:

- Ask for a token from an owner. Owners can access their authorization tokens from the Settings page of their workspace in Azure Machine Learning Studio (classic). Select **Settings** from the left pane and click **AUTHORIZATION TOKENS** to see the primary and secondary tokens. Although either the primary or the secondary authorization tokens can be used in the code snippet, it is recommended that owners only share the secondary authorization tokens.

The screenshot shows the 'settings' page in the Azure Machine Learning Studio (classic). On the left, there is a sidebar with icons for EXPERIMENTS, WEB SERVICES, and SETTINGS. The SETTINGS icon is highlighted. The main area is titled 'settings' and contains a table for 'AUTHORIZATION TOKENS'. The table has columns for NAME, PRIMARY AUTHORIZATION TOKEN, and SECONDARY AUTHORIZATION TOKEN. The 'NAME' column is circled in red. The 'PRIMARY AUTHORIZATION TOKEN' and 'SECONDARY AUTHORIZATION TOKEN' columns are also circled in red. Each token field has a 'Regenerate' button to its right.

- Ask to be promoted to role of owner: a current owner of the workspace needs to first remove you from the workspace then reinvite 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 (classic), 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 Machine Learning Studio (classic) interface. On the left, there's a sidebar with icons for EXPERIMENTS, WEB SERVICES, MODULES, DATASETS (which is selected and highlighted in blue), TRAINED MODELS, and SETTINGS. The main area is titled "datasets" and shows a table with one row:

NAME	SUBMITTED BY	DESCRIPTION	DATA TYPE	CREA... ↴	🔍
My Data.tsv	ptvsazure	My Test Data	GenericTSV	1/20/2015 12:3...	

At the bottom, there's a dark 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 has a title bar "GENERATE DATA ACCESS CODE" and a close button "x". The main content area says "Use this code to access your data". Below that, a note says "To programmatically access this dataset, copy the code snippet into your favorite development environment. [Learn More](#)". A callout box contains the note: "Note: this code includes your workspace access token, which provides full access to your workspace. It should be treated like a password." The "CODE SNIPPET" section is titled "Python" and contains the following code:

```
from azureml import Workspace
ws = Workspace(
    workspace_id='[REDACTED]',
    authorization_token='[REDACTED]')
ds = ws.datasets['My Data.tsv']
frame = ds.to_dataframe()
```

At the bottom left is a checkbox labeled "USE SECONDARY TOKEN" with an unchecked state. At the bottom right is a circular checkmark icon.

5. Paste the code into the notebook of your local Python application.

The screenshot shows an IPython Notebook interface. In cell [3], Python code is run to import the Azure ML workspace module and load a dataset named 'My Data.tsv' into a DataFrame. In cell [4], the resulting DataFrame is displayed as a table with 10 rows and 13 columns, labeled 'frame'. The columns are: fLength, fWidth, fSize, fConc, fConcl, fAsym, fM3Long, fM3Trans, fAlpha, fDist, and Class.

	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 Machine Learning Studio (classic), 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 formats are in the `azureml.DataTypeIds` class):

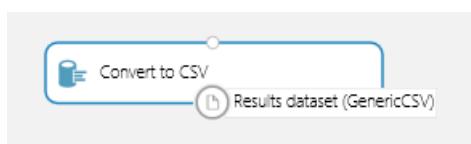
- PlainText
- GenericCSV
- GenericTSV
- GenericCSVNoHeader
- GenericTSVNoHeader

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.



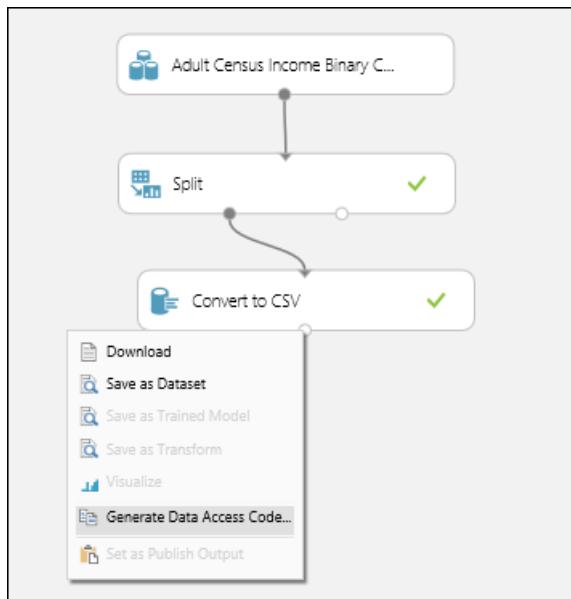
You need to use a conversion module, such as [Convert to CSV](#), to get an output into a supported format.



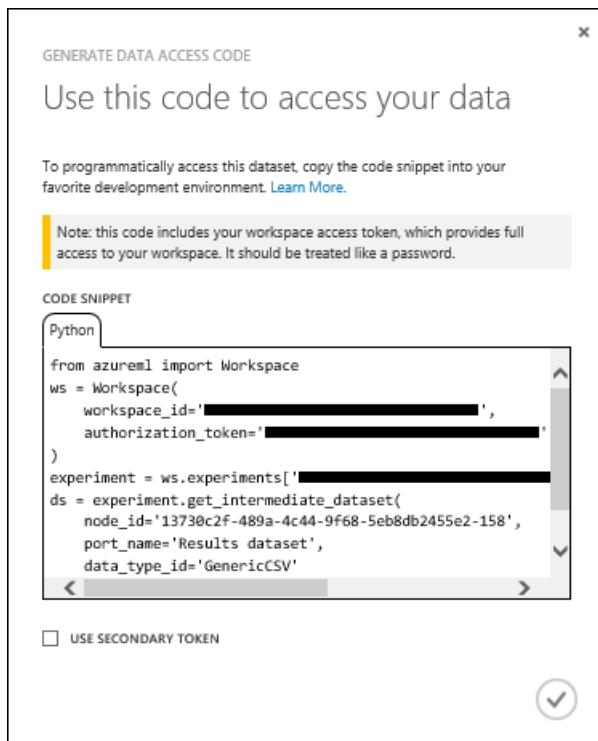
The following steps show an example that creates an experiment, runs it and accesses the intermediate dataset.

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 the job to finish.
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

In [6]:

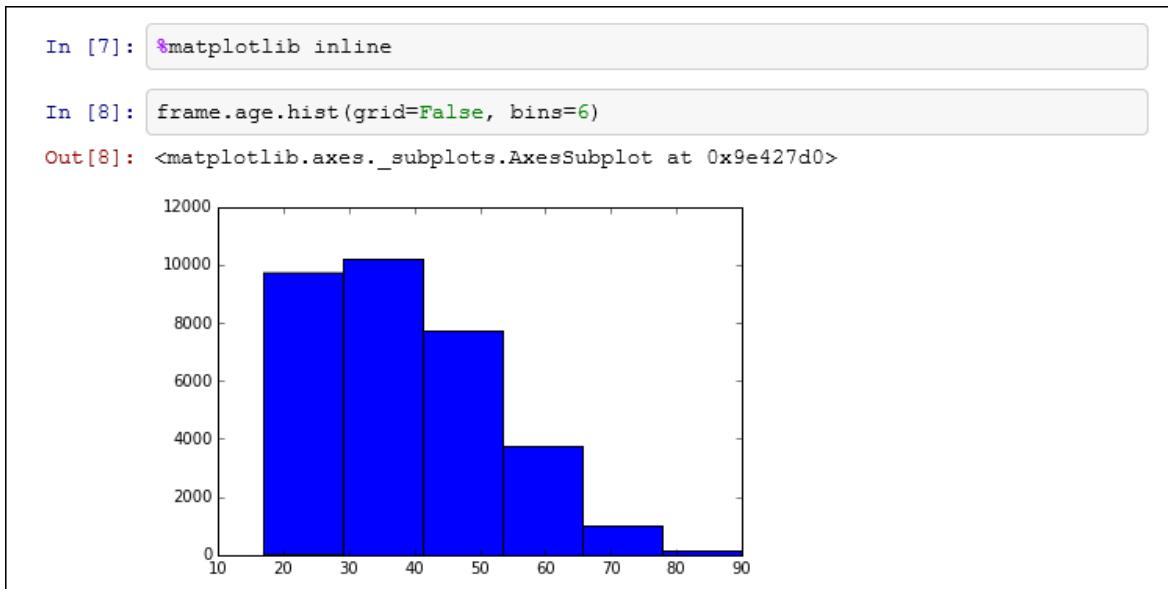
```
from azureml import Workspace
ws = Workspace(
    workspace_id='[REDACTED]',
    authorization_token='[REDACTED]'
)
experiment = ws.experiments['[REDACTED]']
ds = experiment.get_intermediate_dataset(
    node_id='13730c2f-489a-4c44-9f68-5eb8db2455e2-158',
    port_name='Results dataset',
    data_type_id='GenericCSV'
)
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 (classic) 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

3/5/2021 • 3 minutes 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 that 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: Additional features can be created in the Azure Machine Learning Studio (classic) 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

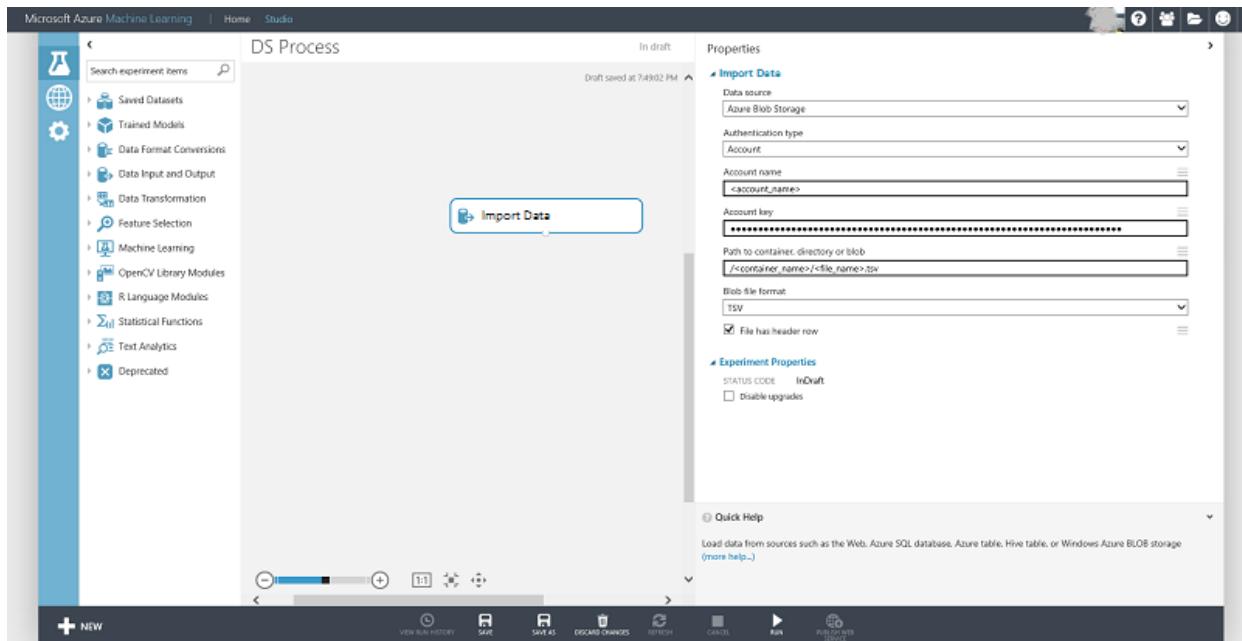
try:

#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

3/5/2021 • 19 minutes 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.

Technologies

These technologies are used in this walkthrough.

- Azure Data Lake Analytics
- U-SQL and Visual Studio
- Python
- Azure Machine Learning
- Scripts

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 Synapse Analytics, Power BI, and Data Factory. This combination 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 (classic) 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 (classic) is used to build and deploy the predictive models 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 Storage (ADLS)
- Azure Data Lake Analytics (ADLA)
- Azure Blob storage account
- Azure Machine Learning Studio (classic) 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 Storage

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.

The screenshot shows the Microsoft Azure portal interface. On the left, the navigation pane has a 'New' button highlighted with a red box. Under the 'Data + Storage' category in the 'Marketplace' list, the 'Data Lake Store' service is highlighted with a red box. The right panel displays the 'Data + Storage' blade with the 'Data Lake Store' service listed under 'FEATURED APPS'.

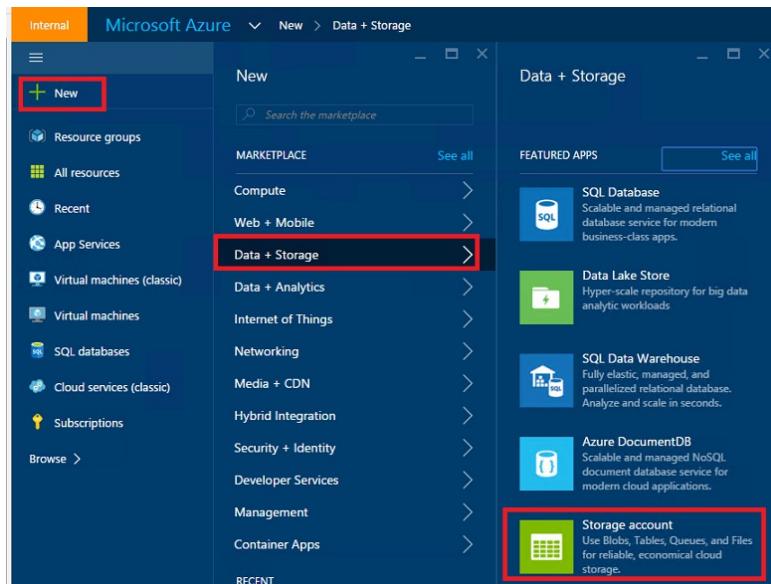
Create an Azure Data Lake Analytics account

Create an ADLA account from the [Azure portal](#). For details, see [Tutorial: get started with Azure Data Lake Analytics using Azure portal](#).

The screenshot shows the Microsoft Azure portal interface. On the left, the navigation pane has a 'New' button highlighted with a red box. Under the 'Data + Analytics' category in the 'Marketplace' list, the 'Data Lake Analytics' service is highlighted with a red box. The right panel displays the 'Data + Analytics' blade with the 'Data Lake Analytics' service listed under 'FEATURED APPS'.

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 (classic) account

Sign up/into Azure Machine Learning Studio (classic) from the [Azure Machine Learning studio](#) 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 Machine Learning studio.

Install Azure Data Lake Tools [Recommended]

Install Azure Data Lake Tools for your version of Visual Studio from [Azure Data Lake Tools for Visual Studio](#).

Azure Data Lake Tools for Visual Studio

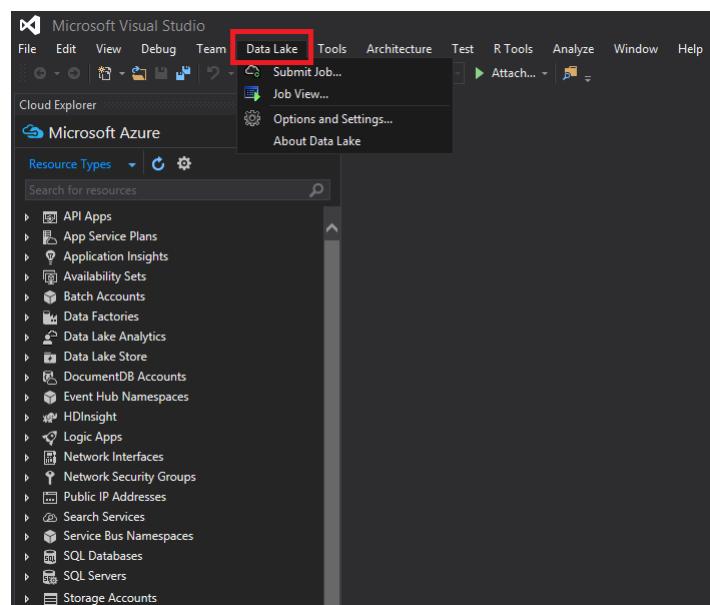
Language: English

Download

Plug-in for Azure Data Lake development using Visual Studio

- [+ Details](#)
- [+ System Requirements](#)
- [+ Install Instructions](#)

After the installation finishes, 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 dropoff 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
89D227B655E5C82AECF13C3F540D4CF4,BA96DE419E711691B9445D6A307C170,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
0BD7C8F5BA12B88E0B67BE2D8BEA73D8,9FD8F69F0804BD85549F40E9DA1BE472,CMT,1,N,2013-01-06 00:18:35,2013-01-06 00:22:54,1,259,1,50,-74.006683,40.731781,-73.994499,40.750666
0BD7C8F5BA12B88E0B67BE2D8BEA73D8,9FD8F69F0804BD85549F40E9DA1BE472,CMT,1,N,2013-01-05 18:49:41,2013-01-05 18:54:23,1,282,1,10,-74.004707,40.73777,-74.009834,40.726082
DFD2202E08F7A8DC9A57B02ACB81FE2,51EE87E3205C985EF8431D850C786310,CMT,1,N,2013-01-07 23:54:15,2013-01-07 23:58:20,2,244,,70,-73.974602,40.759945,-73.984734,40.75938
DFD2202E08F7A8DC9A57B02ACB81FE2,51EE87E3205C985EF8431D850C786310,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,BA96DE419E711691B9445D6A307C170,CMT,2013-01-01 15:11:48,CSH,6,5,0,0,5,0,0,7
0BD7C8F5BA12B88E0B67BE2D8BEA73D8,9FD8F69F0804BD85549F40E9DA1BE472,CMT,2013-01-06 00:18:35,CSH,6,0,5,0,5,0,0,7
0BD7C8F5BA12B88E0B67BE2D8BEA73D8,9FD8F69F0804BD85549F40E9DA1BE472,CMT,2013-01-05 18:49:41,CSH,5,5,1,0,5,0,0,7
DFD2202E08F7A8DC9A57B02ACB81FE2,51EE87E3205C985EF8431D850C786310,CMT,2013-01-07 23:54:15,CSH,5,0,5,0,5,0,0,6
DFD2202E08F7A8DC9A57B02ACB81FE2,51EE87E3205C985EF8431D850C786310,CMT,2013-01-07 23:25:03,CSH,9,5,0,5,0,0,0,10,5

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 an Azure Storage blob. The U-SQL script for this join is in the [Join trip and fare tables](#) section.

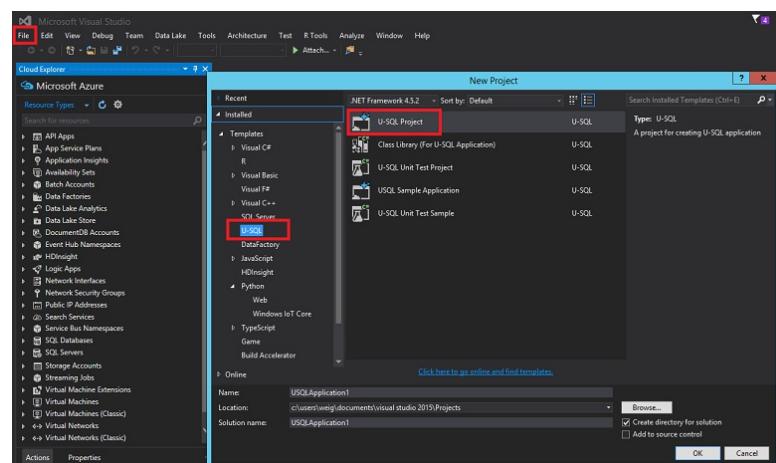
Process data with U-SQL

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 `container_name@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
@tripB =
    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/nytaxitrip/trip_data_{*}.csv"
    USING Extractors.Csv();
```

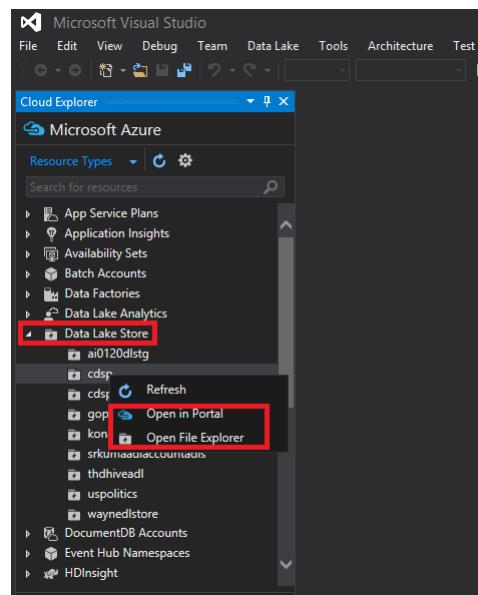
Since there are headers in the first row, you need to remove the headers and change column types into appropriate ones. You can either save the processed data to Azure Data Lake Storage using `swebhdfs://data_lake_storage_name.azuredatalakestorage.net/folder_name/file_name_` or to Azure Blob storage 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/nytaxi_folder/demo_trip.csv"
USING Outputters.Csv();

////Output data to blob
OUTPUT @trip
TO "wasb://container_name@blob_storage_account_name.blob.core.windows.net/demo_trip.csv"
USING Outputters.Csv();
```

Similarly you can read in the fare data sets. Right-click Azure Data Lake Storage, 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 Storage.

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 @trip
    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 @trip
    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,
        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 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 rownum
FROM @model_data_full;

@model_data_random_sample_1_1000 =
SELECT *
FROM @addrownumberres_randomsample
WHERE rownum % 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:

```

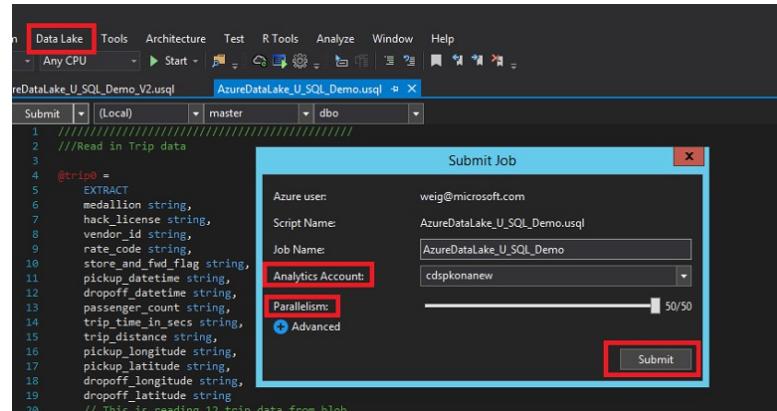
//stratified random select 1/1000 data for modeling purpose
@addrownumberres_stratifiedsample =
SELECT *,
       ROW_NUMBER() OVER(PARTITION BY tip_class) AS rownum
FROM @model_data_full;

@model_data_stratified_sample_1_1000 =
SELECT *
FROM @addrownumberres_stratifiedsample
WHERE rownum % 1000 == 0;
//// output to blob
OUTPUT @model_data_stratified_sample_1_1000
TO "wasb://container_name@blob_storage_account_name.blob.core.windows.net/demo_ex_9_stratified_1_1000.csv"
USING Outputters.Csv();
////output data to ADL
OUTPUT @model_data_stratified_sample_1_1000
TO "swebhdfs://data_lake_storage_name.azuredatalakestore.net/nytaxi_folder/demo_ex_9_stratified_1_1000.csv"
USING Outputters.Csv();

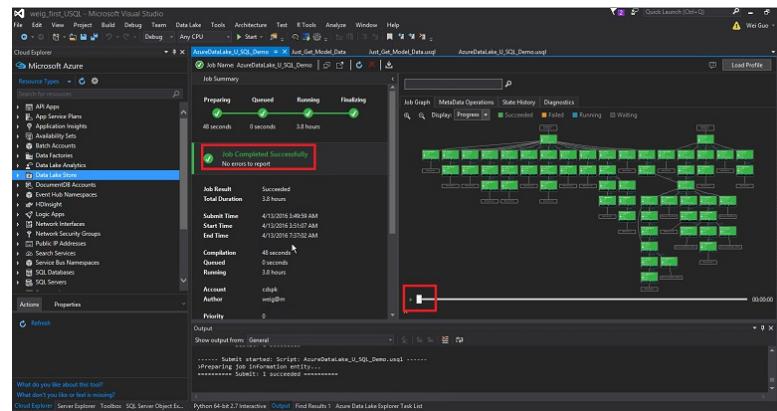
```

Run U-SQL jobs

After 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 compiled successfully, the status of your job is displayed in Visual Studio for monitoring. After the job completes, 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.



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.

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 Azure Machine Learning 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 = 'XXXXXXXXX'
STORAGEACCOUNTKEY = 'YYYYYYYYYYYYYYYYYYYYYYYYYYYY'
BLOBNAME = 'demo_ex_9_stratified_1_1000_copy.csv'
blob_service = BlobService(account_name=STORAGEACCOUNTNAME,account_key=STORAGEACCOUNTKEY)
```

- Read in as text

```
t1 = time.time()
data = blob_service.get_blob_to_text(CONTAINERNAME,BLOBNAME).split("\n")
t2 = time.time()
print(("It takes %s seconds to read in "+BLOBNAME) % (t2 - t1))
```

It takes 1.61118912697 seconds to read in demo_ex_9_stratified_1_1000_copy.csv

- Add column names and separate columns

```
colnames =
['medallion','hack_license','vendor_id','rate_code','store_and_fwd_flag','pickup_datetime','dropoff_datetime',
'passenger_count','trip_time_in_secs','trip_distance','pickup_longitude','pickup_latitude','dropoff_longitude',
'dropoff_latitude','payment_type','fare_amount','surcharge','mta_tax','tolls_amount','total_amount','tip_amount',
'tipped','tip_class','rownum']
df1 = pd.DataFrame([sub.split(",") for sub in data], columns = colnames)
```

- Change some columns to numeric

```
cols_2_float =
['trip_time_in_secs','pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude',
'fare_amount','surcharge','mta_tax','tolls_amount','total_amount','tip_amount',
'passenger_count','trip_distance',
'tipped','tip_class','rownum']
for col in cols_2_float:
    df1[col] = df1[col].astype(float)
```

Build machine learning models

Here 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_dummy = pd.get_dummies(df1['payment_type'], prefix='payment_type_dummy')
df1_vendor_id_dummy = pd.get_dummies(df1['vendor_id'], prefix='vendor_id_dummy')
```

- Create data frame for the modeling

```
cols_to_keep = ['tipped', 'trip_distance', 'passenger_count']
data = df1[cols_to_keep].join([df1_payment_type_dummy,df1_vendor_id_dummy])

X = data.iloc[:,1:]
Y = data.tipped
```

- Training and testing 60-40 split

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.4, random_state=0)
```

- Logistic Regression in training set

```
model = LogisticRegression()
logit_fit = model.fit(X_train, Y_train)
print ('Coefficients: \n', logit_fit.coef_)
Y_train_pred = logit_fit.predict(X_train)
```

```
('Coefficients: \n', array([[-0.05133172, -0.008217 , 5.45355932, -6.63170116, -1.79058312,
 -2.79956737, 4.60387707, -0.33341385, -0.83100141]]))
```

- 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)
```

```
[ 0.          0.03372081  1.          ] [ 0.          0.99994397  1.          ] [ 2.          1.          0.]
[ 0.          0.03342893  1.          ] [ 0.          0.99983179  1.          ] [ 2.          1.          0.]
0.983111577209
0.983201427013
[[46966 1639]
 [ 3 53535]]
[[31343 1084]
 [ 6 35663]]
```

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 (Azure Machine Learning Studio is already at least at this version).

- Find your workspace credentials from Azure Machine Learning Studio (classic) settings. In Azure Machine Learning Studio, click Settings --> Name --> Authorization Tokens.

The screenshot shows the 'settings' page in the Microsoft Azure Machine Learning Studio (classic). On the left sidebar, there are icons for PROJECTS, EXPERIMENTS, WEB SERVICES, NOTEBOOKS, DATASETS, TRAINED MODELS, and SETTINGS. The SETTINGS icon is highlighted with a red box. The main area displays workspace details:

- NAME: Wei-WorkSpace-Azure-ML (highlighted with a red box)
- WORKSPACE DESCRIPTION: (empty text area)
- WORKSPACE TYPE: Standard (Learn More)
- WORKSPACE ID: (highlighted with a red box)
- CREATION TIME: 4/3/2015, 1:33:43 PM
- OWNER'S EMAIL: @outlook.com
- SUBSCRIPTION ID: e8: (with a dropdown arrow)
- STORAGE ACCOUNT: acc: (with a dropdown arrow)

```
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

```

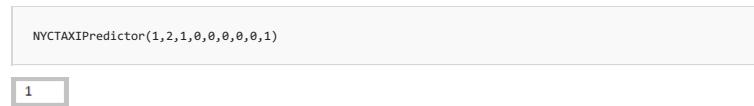
url = predictNYCTAXI.service.url
api_key = predictNYCTAXI.service.api_key

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. Typically, wait 5-10 seconds after the previous step.



Option 2: Create and deploy models directly in Azure Machine Learning

Azure Machine Learning Studio (classic) can read data directly from Azure Data Lake Storage and then be used to create and deploy models. This approach uses a Hive table that points at the Azure Data Lake Storage. A separate Azure HDInsight cluster needs to be provisioned for the Hive table.

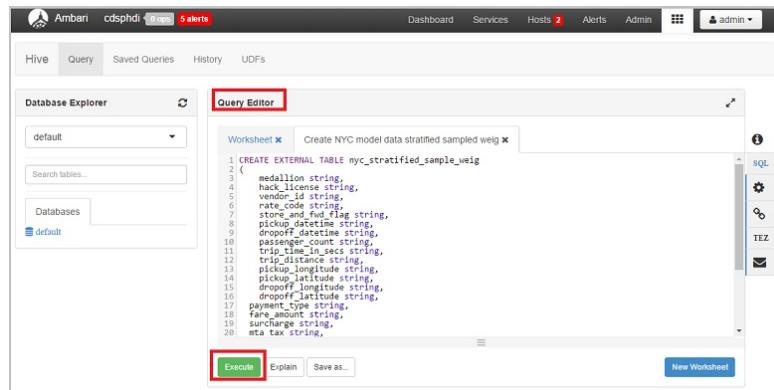
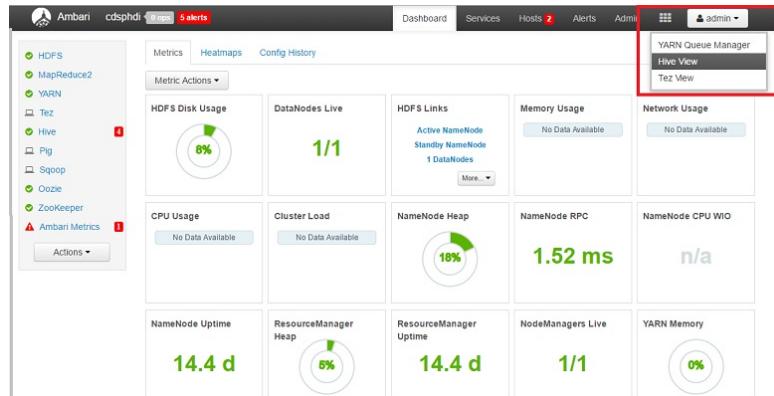
Create an HDInsight Linux Cluster

Create an HDInsight Cluster (Linux) from the [Azure portal](#). For details, see the [Create an HDInsight cluster with access to Azure Data Lake Storage](#) section in [Create an HDInsight cluster with Data Lake Store using Azure portal](#).

Create Hive table in HDInsight

Now you create Hive tables to be used in Azure Machine Learning Studio (classic) in the HDInsight cluster using the data stored in Azure Data Lake Storage in the previous step. Go to the HDInsight cluster created. Click **Settings** --> **Properties** --> **Cluster AAD Identity** --> **ADLS Access**, make sure your Azure Data Lake Storage account is added in the list with read, write, and execute rights.

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 Storage 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 completes, you should see the results like this:

Query Process Results (Status: Succeeded)			Save results...
Logs	Results		
Filter columns...		previous	next
<code>nyc_stratified_sample_weig.medallion</code>	<code>nyc_stratified_sample_weig.hack_license</code>	<code>nyc_stratified_sample_we</code>	
"0053334C798EC6C8E637657962030F99"	"9EF690115D60940E7A8039A67542642E"	"VTS"	
"00B99071EE4DC8266384113B91E6AC13"	"081B28EDAT73E5E2C97573F0DBAC25D"	"CMT"	
"011E4CBA1987A8553F8EA5681FFBF7F1"	"F53B26859F6C46C24E39EEAE8096268"	"CMT"	
"1109955CCAABC BCE1A22BCED5F1DBFF5"	"EAA69F43239E5C6609CD8FF4D6725836"	"CMT"	
"0A415B814A6479EE706972D2FD6DA08A"	"12A32F655B8C9CE3AC7872477A641974"	"VTS"	
"02B29197FB7470B583ED12167D19E998"	"C1EEBC8298A619F86637D7E97F6BDD5C"	"CMT"	
"03055B956C21B1F915DCDB118AA79F21"	"E0C7A67293DE535DB04F8AFD8BF28F73"	"VTS"	
"037673EEAE0DCB912D06BED04E89D89D"	"3B247BD1230E95A0D9FED3E47FECB8D9"	"CMT"	
"03BF54085C92C385889B957D804780D1"	"776D5633A2041CEBDE03092E401D61DB"	"CMT"	
"0437660BC3704C2F185301D539434A64"	"AE9AB8C79A2A0EB6DDA76B7024FF6029"	"CMT"	

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 Machine Learning Studio (classic) 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 URI can be found in Azure portal), Hadoop credentials, location of output data, and Azure Storage account name/key/container name.

Data source

Hive database query

```
1 select * from nyc_stratified_sample;
```

HCatalog server URI

Hadoop user account name

Hadoop user account password

Location of output data

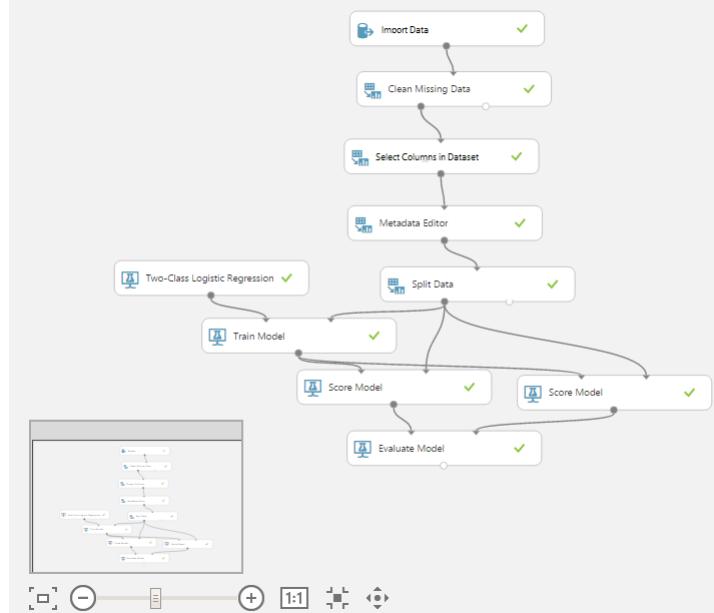
Azure storage account name

Azure storage key

Azure container name

An example of a binary classification experiment reading data from Hive table is shown in the following figure:

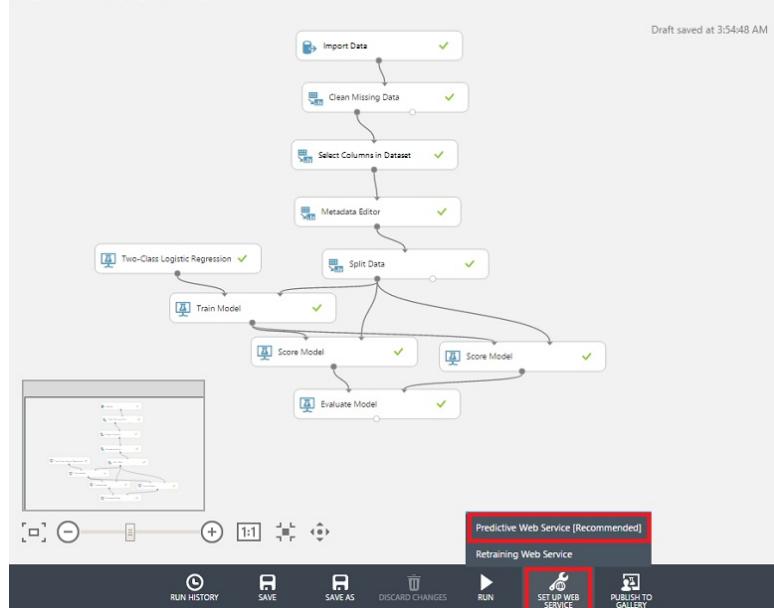
Azure Data Lake -Demo



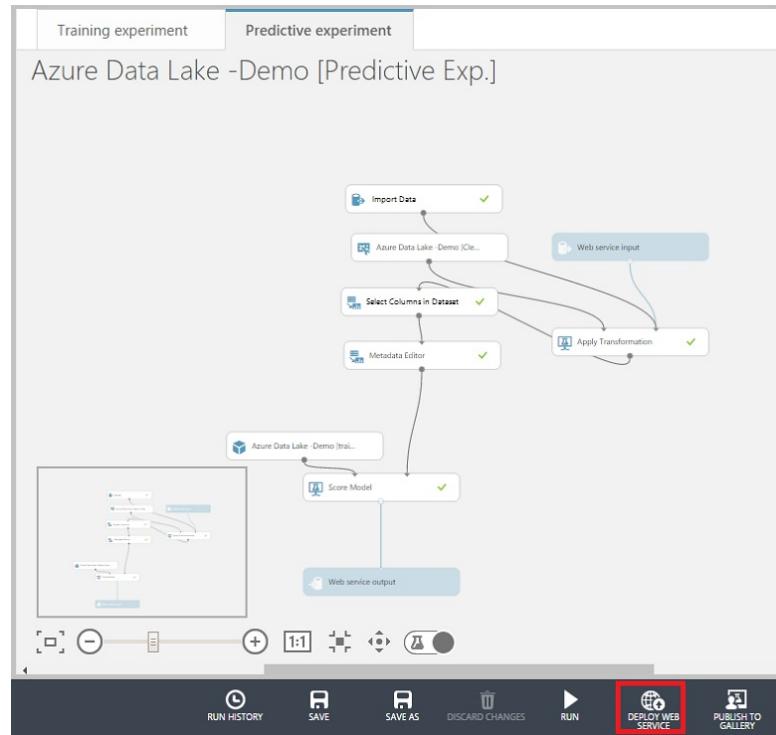
After the experiment is created, click Set Up Web Service --> Predictive Web Service

Azure Data Lake -Demo

Finished running ✓



Run the automatically created scoring experiment, when it finishes, click Deploy Web Service



The web service dashboard displays shortly:

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 (classic) 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 Azure Synapse Analytics](#)
- [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

3/5/2021 • 6 minutes 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 goal may be completed 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 results may 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 guidance is useful to understand before including location as one or more features:

- 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 you roughly what state, country, or region you're 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. You

can also call a REST end point such as Bing Maps API available at [Find a Location by Point](#) to get the region/district information.

```
select
    <location_columnname>
    ,round(<location_columnname>,0) as l1
    ,l2=case when LEN (PARSENAME(round(ABS(<location_columnname>)) - FLOOR(ABS(<location_columnname>)),6),1))
    >= 1 then substring(PARSENAME(round(ABS(<location_columnname>)) -
    FLOOR(ABS(<location_columnname>)),6),1,1) 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) 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) 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) 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) 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) else '0' end
from <tablename>
```

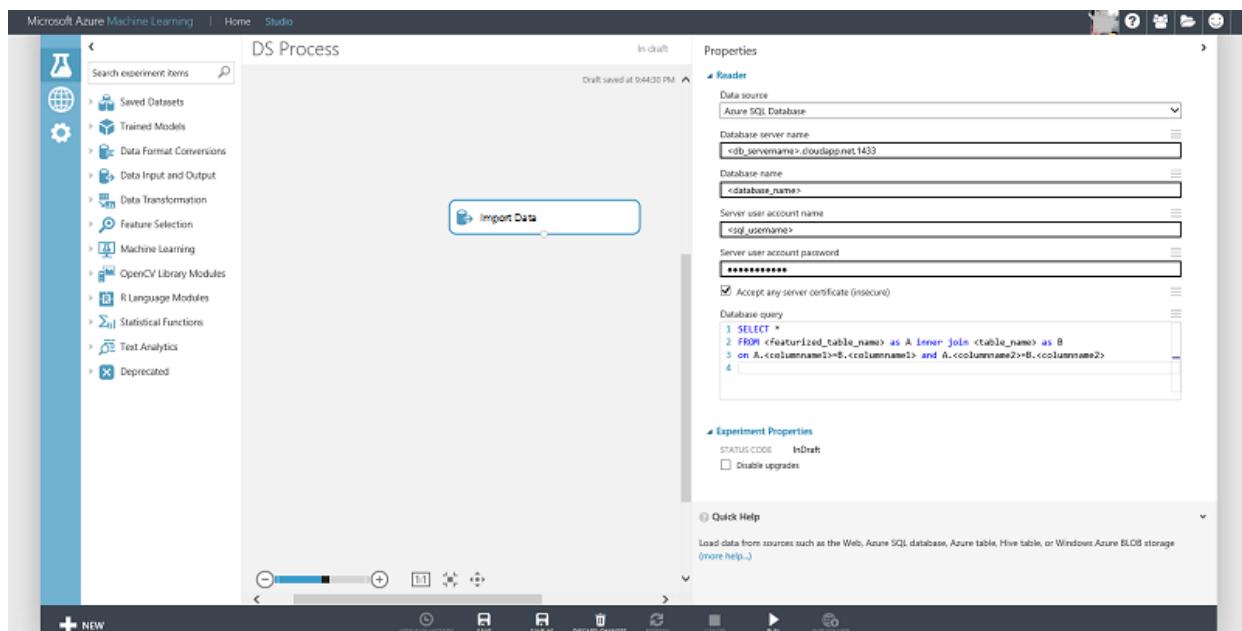
These location-based features can be further used to generate additional count features as described earlier.

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](#). Load the data from the database into a pandas data frame for more processing. 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 (replaceservername, dbname, username, and password with your specific values):

```
#Set up the SQL Azure connection
import pyodbc
conn = pyodbc.connect('DRIVER={SQL Server};SERVER=<servername>;DATABASE=<dbname>;UID=<username>;PWD=<password>')
```

The [Pandas library](#) in Python provides a rich set of data structures and data analysis tools for data manipulation for Python programming. The 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

11/2/2020 • 2 minutes to read • [Edit Online](#)

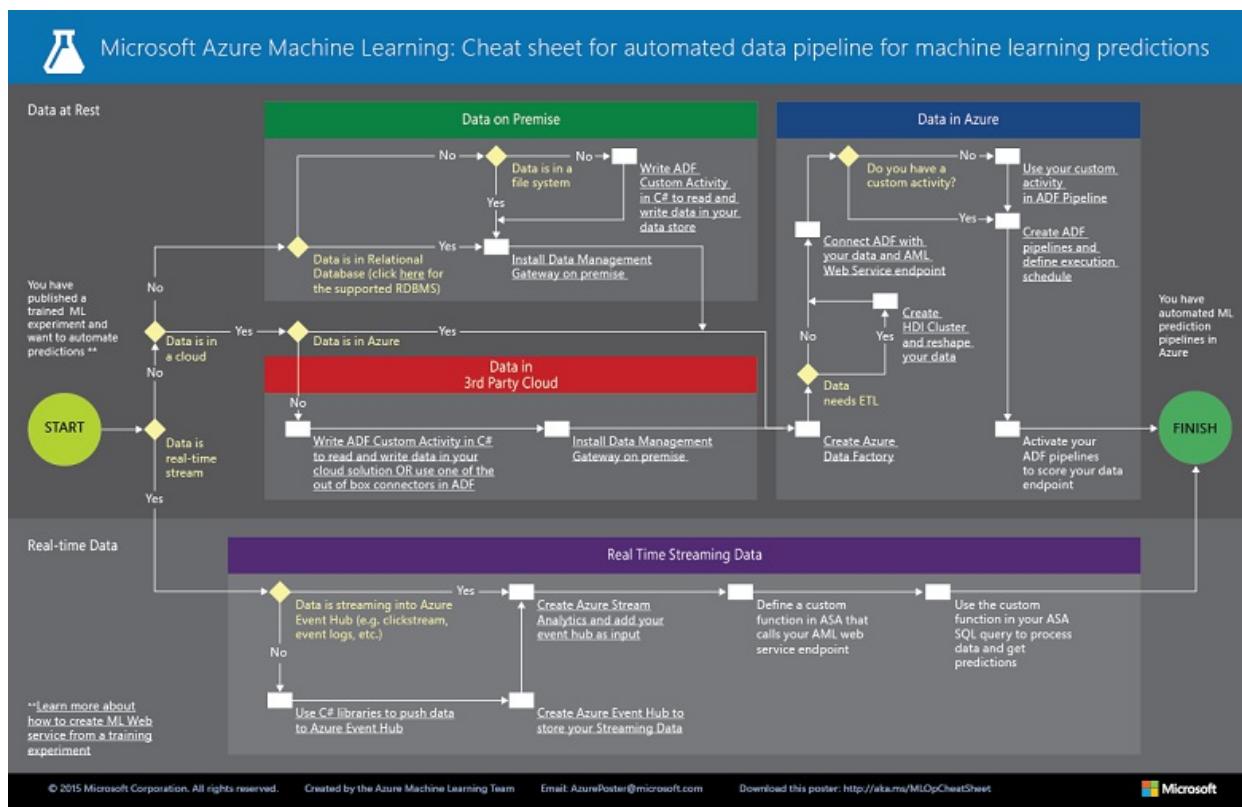
The Microsoft Azure Machine Learning automated data pipeline cheat sheet helps you navigate through the technology you can use to get your data to your Machine Learning web service where it can be scored by your predictive analytics model.

Depending on whether your data is on-premises, in the cloud, or real-time streaming, 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](#).

Overview of data science using Spark on Azure HDInsight

3/5/2021 • 9 minutes 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 compares 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 (for example, 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: <https://www.transtats.bts.gov/ONTIME/>
- Airport weather data: <https://www.ncdc.noaa.gov/>

NOTE

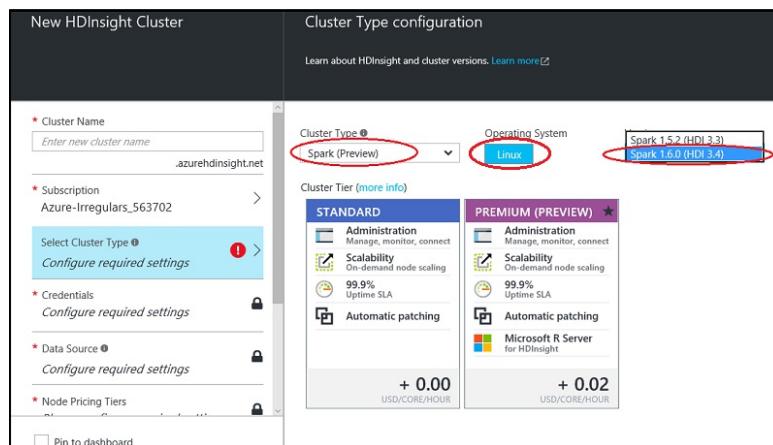
The Spark 2.0 notebooks on the NYC taxi and airline flight delay data-sets can take 10 mins or more to run (depending on the size of your HDI cluster). The first notebook in the above list shows many aspects of the data exploration, visualization and ML model training in a notebook that takes less time 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 example 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.

**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 use them or not. Be sure to delete your cluster after you finish using it. 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 pickup and dropoff 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_longi  
89D227B655E5C82AECF13C3F5404CF4,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  
0BD7C8F5BA12888E0B67BED28BEA73D8,9FD8F69F0804BD85549F40E9DA1BE472,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  
0BD7C8F5BA12888E0B67BED28BEA73D8,9FD8F69F0804BD85549F40E9DA1BE472,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  
DFD2202EE0F7A8DC9A57B02ACB81FE2,51EE87E3205C985EF8431D850C786310,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  
DFD2202EE0F7A8DC9A57B02ACB81FE2,51EE87E3205C985EF8431D850C786310,CMT,1,N,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  
89D227B655E5C82AECF13C3F5404CF4,BA96DE419E711691B9445D6A6307C170,CMT,2013-01-01 15:11:48,CSH,6.5,0,0.5,0,0,7  
0BD7C8F5BA12888E0B67BED28BEA73D8,9FD8F69F0804BD85549F40E9DA1BE472,CMT,2013-01-06 00:18:35,CSH,6,0.5,0,0.5,0,0,7
```

0BD7C8F5BA12B88E0B67BED28BEA73D8,9FD8F69F0804BDB5549F40E9DA1BE472,CMT,2013-01-05 18:49:41,CSH,5.5,1,0.5,0,0,7
DFD2202EE08F7A8DC9A57B02ACB81FE2,51EE87E3205C985EF8431D850C786310,CMT,2013-01-07 23:54:15,CSH,5,0.5,0.5,0,0,6
DFD2202EE08F7A8DC9A57B02ACB81FE2,51EE87E3205C985EF8431D850C786310,CMT,2013-01-07 23:25:03,CSH,9.5,0.5,0.5,0,0,10.5

We have taken a 0.1% sample of these files and joined the trip_data and trip_fare CVS files into a single dataset to use as the input dataset for this walkthrough. The unique key to join trip_data and trip_fare is composed of the fields: medallion, hack_licence and pickup_datetime. Each record of the dataset contains the following attributes representing a NYC Taxi trip:

FIELD	BRIEF DESCRIPTION
medallion	Anonymized taxi medallion (unique taxi id)
hack_license	Anonymized Hackney Carriage License number
vendor_id	Taxi vendor id
rate_code	NYC taxi rate of fare
store_and_fwd_flag	Store and forward flag
pickup_datetime	Pick up date & time
dropoff_datetime	Dropoff date & time
pickup_hour	Pick up hour
pickup_week	Pick up week of the year
weekday	Weekday (range 1-7)
passenger_count	Number of passengers in a taxi trip
trip_time_in_secs	Trip time in seconds
trip_distance	Trip distance traveled in miles
pickup_longitude	Pick up longitude
pickup_latitude	Pick up latitude
dropoff_longitude	Dropoff longitude
dropoff_latitude	Dropoff latitude
direct_distance	Direct distance between pickup and dropoff locations
payment_type	Payment type (cash, credit-card etc.)
fare_amount	Fare amount in
surcharge	Surcharge
mta_tax	MTA Metro Transportation 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**.

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.

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

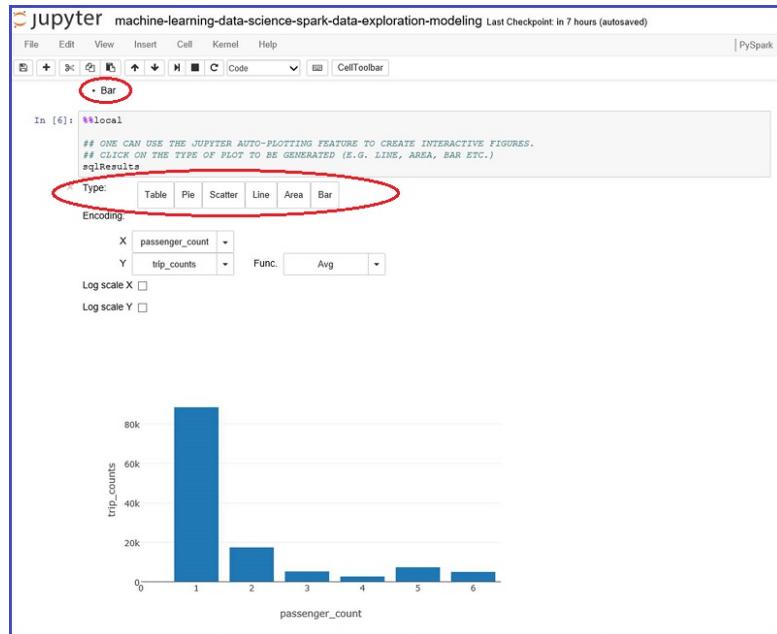
You can right-click the links on your browser and select **Copy Link** to get the GitHub raw content URL. You can paste this URL into the Jupyter Upload file explorer dialog box.

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

3/5/2021 • 34 minutes 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.

The screenshot shows the 'New HDInsight Cluster' configuration page. In the 'Cluster Type configuration' section, the 'Cluster Type' dropdown is set to 'Spark (Preview)', which is highlighted with a red oval. The 'Operating System' dropdown is set to 'Linux', also highlighted with a red oval. The 'Data Source' dropdown is set to 'Spark 1.6.0 (HDI 3.4)', also highlighted with a red oval. Below these, the 'Cluster Tier' section compares two options: 'STANDARD' and 'PREMIUM (PREVIEW)'. Both tiers include 'Administration', 'Scalability', '99.9% Uptime SLA', and 'Automatic patching'. The 'PREMIUM (PREVIEW)' tier adds 'Microsoft R Server for HDInsight'. Both tiers cost '+ 0.02 USD/CORE/HOUR'.

WARNING

Billing for HDInsight clusters is prorated per minute, whether you use them or not. Be sure to delete your cluster after you finish using it. 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 management interface. On the left, the 'Cluster Dashboards' section is highlighted with a red circle. It displays cluster details like 'Cluster Type: Standard Spark on Linux', 'URL: https://<clustername>.azurehdinsight.net', and usage statistics: 'Cores in South Central US for subscription' (2,100 cores total, 12 in this cluster, 756 in other clusters, 1332 available, 2100 total). On the right, the 'Jupyter Notebook' section is also highlighted with a red circle. It lists other cluster components: 'HDInsight Cluster Dashboard', 'Spark History Server', and 'Yarn'. A 'Add tiles +' button is visible at the top right of the dashboard area.

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. The file explorer sidebar on the left shows two directories: 'PySpark' and 'Scala'. The 'Scala' directory is circled with a red marker. At the top right, there is an 'Upload' button, which is also circled with a red marker. The main workspace is currently empty.

Select **Scala** to see a directory that has a few examples of prepackaged notebooks that use the PySpark API. The Exploration Modeling and Scoring using Scala.ipynb notebook that contains the code samples for this suite of Spark topics is available on [GitHub](#).

You can upload the notebook directly from GitHub to the Jupyter Notebook server on your Spark cluster. On your Jupyter home page, click the **Upload** button. In the file explorer, paste the GitHub (raw content) URL of the Scala notebook, and then click **Open**. The Scala notebook is available at the following URL:

[Exploration-Modeling-and-Scoring-using-Scala.ipynb](#)

Setup: Preset Spark and Hive contexts, Spark magics, and Spark

libraries

Preset Spark and Hive contexts

```
# SET THE START TIME
import java.util.Calendar
val beginningTime = Calendar.getInstance().getTime()
```

The Spark kernels that are provided with Jupyter notebooks have preset contexts. You don't need to explicitly set the Spark or Hive contexts before you start working with the application you are developing. The preset contexts are:

- `sc` for `SparkContext`
- `sqlContext` for `HiveContext`

Spark magics

The Spark kernel provides some predefined "magics," which are special commands that you can call with `%%`. Two of these commands are used in the following code samples.

- `%%local` specifies that the code in subsequent lines will be executed locally. The code must be valid Scala code.
- `%%sql -o <variable name>` executes a Hive query against `sqlContext`. If the `-o` parameter is passed, the result of the query is persisted in the `%%local` Scala context as a Spark data frame.

For more information about the kernels for Jupyter notebooks and their predefined "magics" that you call with `%%` (for example, `%%local`), see [Kernels available for Jupyter notebooks with HDInsight Spark Linux clusters on HDInsight](#).

Import libraries

Import the Spark, MLlib, and other libraries you'll need by using the following code.

```

# IMPORT SPARK AND JAVA LIBRARIES
import org.apache.spark.sql.SQLContext
import org.apache.spark.sql.functions._
import java.text.SimpleDateFormat
import java.util.Calendar
import sqlContext.implicits._
import org.apache.spark.sql.Row

# IMPORT SPARK SQL FUNCTIONS
import org.apache.spark.sql.types.{StructType, StructField, StringType, IntegerType, FloatType, DoubleType}
import org.apache.spark.sql.functions.rand

# IMPORT SPARK ML FUNCTIONS
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.feature.{StringIndexer, VectorAssembler, OneHotEncoder, VectorIndexer, Binarizer}
import org.apache.spark.ml.tuning.{ParamGridBuilder, TrainValidationSplit, CrossValidator}
import org.apache.spark.ml.regression.{LinearRegression, LinearRegressionModel, RandomForestRegressor,
RandomForestRegressionModel, GBTRegressor, GBTRegressionModel}
import org.apache.spark.ml.classification.{LogisticRegression, LogisticRegressionModel,
RandomForestClassifier, RandomForestClassificationModel, GBTClassifier, GBTClassificationModel}
import org.apache.spark.ml.evaluation.{BinaryClassificationEvaluator, RegressionEvaluator,
MulticlassClassificationEvaluator}

# IMPORT SPARK MLLIB FUNCTIONS
import org.apache.spark.mllib.linalg.{Vector, Vectors}
import org.apache.spark.mllib.util.MLUtils
import org.apache.spark.mllib.classification.{LogisticRegressionWithLBFGS, LogisticRegressionModel}
import org.apache.spark.mllib.regression.{LabeledPoint, LinearRegressionWithSGD, LinearRegressionModel}
import org.apache.spark.mllib.tree.{GradientBoostedTrees, RandomForest}
import org.apache.spark.mllib.tree.configuration.BoostingStrategy
import org.apache.spark.mllib.tree.model.{GradientBoostedTreesModel, RandomForestModel, Predict}
import org.apache.spark.mllib.evaluation.{BinaryClassificationMetrics, MulticlassMetrics, RegressionMetrics}

# SPECIFY SQLCONTEXT
val sqlContext = new SQLContext(sc)

```

Data ingestion

The first step in the Data Science process is to ingest the data that you want to analyze. You bring the data from external sources or systems where it resides into your data exploration and modeling environment. In this article, the data you ingest is a joined 0.1% sample of the taxi trip and fare file (stored as a .tsv file). The data exploration and modeling environment is Spark. This section contains the code to complete the following series of tasks:

1. Set directory paths for data and model storage.
2. Read in the input data set (stored as a .tsv file).
3. Define a schema for the data and clean the data.
4. Create a cleaned data frame and cache it in memory.
5. Register the data as a temporary table in SQLContext.
6. Query the table and import the results into a data frame.

Set directory paths for storage locations in Azure Blob storage

Spark can read and write to Azure Blob storage. You can use Spark to process any of your existing data, and then store the results again in Blob storage.

To save models or files in Blob storage, you need to properly specify the path. Reference the default container attached to the Spark cluster by using a path that begins with `wasb:///`. Reference other locations by using `wasb://`.

The following code sample specifies the location of the input data to be read and the path to Blob storage that is

attached to the Spark cluster where the model will be saved.

```
# SET PATHS TO DATA AND MODEL FILE LOCATIONS
# INGEST DATA AND SPECIFY HEADERS FOR COLUMNS
val taxi_train_file =
sc.textFile("wasb://mllibwalkthroughs@cdsparksamples.blob.core.windows.net/Data/NYCTaxi/JoinedTaxiTripFare
.Point1Pct.Train.tsv")
val header = taxi_train_file.first;

# SET THE MODEL STORAGE DIRECTORY PATH
# NOTE THAT THE FINAL BACKSLASH IN THE PATH IS REQUIRED.
val modelDir = "wasb:///user/remoteuser/NYCTaxi/Models/";
```

Import data, create an RDD, and define a data frame according to the schema

```
# RECORD THE START TIME
val starttime = Calendar.getInstance().getTime()

# DEFINE THE SCHEMA BASED ON THE HEADER OF THE FILE
val sqlContext = new SQLContext(sc)
val taxi_schema = StructType(
  Array(
    StructField("medallion", StringType, true),
    StructField("hack_license", StringType, true),
    StructField("vendor_id", StringType, true),
    StructField("rate_code", DoubleType, true),
    StructField("store_and_fwd_flag", StringType, true),
    StructField("pickup_datetimestamp", StringType, true),
    StructField("dropoff_datetimestamp", 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 automatic visualization Jupyter feature.

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 setting means the output will 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:

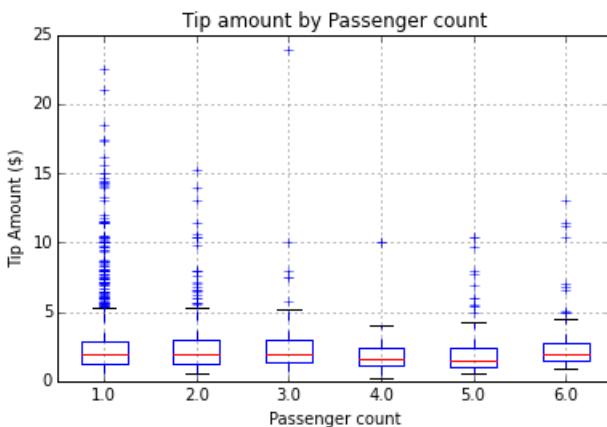
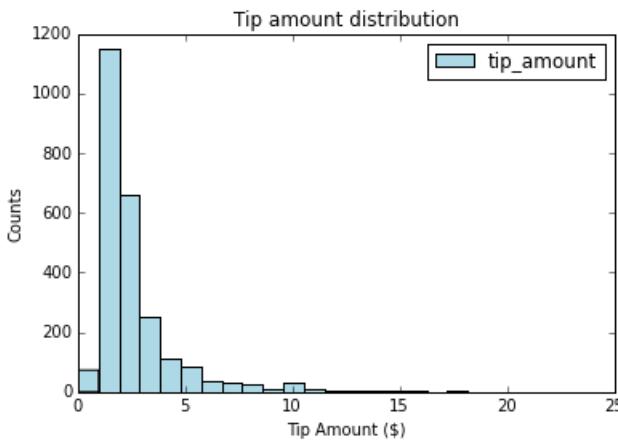
```
# RUN THE CODE LOCALLY ON THE JUPYTER SERVER AND IMPORT LIBRARIES
%%local
import matplotlib.pyplot as plt
%matplotlib inline

# 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).setEla
sticNetParam(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("area
UnderROC")
val ROC = evaluator.evaluate(predictions)
println("ROC on test data = " + ROC)

# SAVE THE MODEL
val datestamp = Calendar.getInstance().getTime().toString.replaceAll(" ", ".").replaceAll(":", "_");
val modelName = "LogisticRegression_"
val filename = modelDir.concat(modelName).concat(datestamp)
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("area
UnderROC")
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

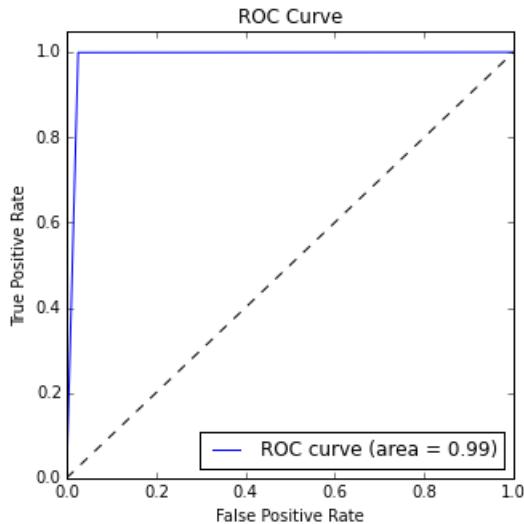
sqlResults['probFloat'] = sqlResults.apply(lambda row: row['probability'].values()[0][1], axis=1)
predictions_pddf = sqlResults[["tipped","probFloat"]]

# 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).setE
lasticNetParam(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.

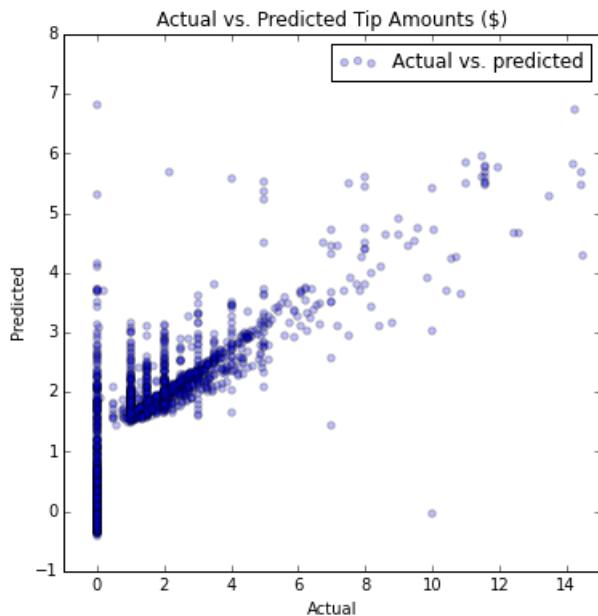
```

# 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 trains decision trees iteratively to minimize a loss function. You can use GBTS for regression and classification. They can handle categorical features, do not require feature scaling, and can capture nonlinearities and feature interactions. You also can use them in a multiclass-classification setting.

```

# RECORD THE START TIME
val starttime = Calendar.getInstance().getTime()

# TRAIN A GBT REGRESSION MODEL
val gbt = new GBTRegressor().setLabelCol("label").setFeaturesCol("featuresCat").setMaxIter(10)
val gbtModel = gbt.fit(indexedTRAINwithCatFeat)

# MAKE PREDICTIONS
val predictions = gbtModel.transform(indexedTESTwithCatFeat)

# COMPUTE TEST SET R2
val evaluator = new
RegressionEvaluator().setLabelCol("label").setPredictionCol("prediction").setMetricName("r2")
val Test_R2 = evaluator.evaluate(predictions)

# GET THE TIME TO RUN THE CELL
val endtime = Calendar.getInstance().getTime()
val elapsedtime = ((endtime.getTime() - starttime.getTime()) / 1000).toString;
println("Time taken to run the above cell: " + elapsedtime + " seconds.")

# PRINT THE RESULTS
println("Test R-sqr is: " + Test_R2);

```

Output:

Test R-sqr is: 0.7655383534596654

Advanced modeling utilities for optimization

In this section, you use machine learning utilities that developers frequently use for model optimization. Specifically, you can optimize machine learning models three different ways by using parameter sweeping and cross-validation:

- Split the data into train and validation sets, optimize the model by using hyper-parameter sweeping on a training set, and evaluate on a validation set (linear regression)
- Optimize the model by using cross-validation and hyper-parameter sweeping by using Spark ML's CrossValidator function (binary classification)
- Optimize the model by using custom cross-validation and parameter-sweeping code to use any machine learning function and parameter set (linear regression)

Cross-validation is a technique that assesses how well a model trained on a known set of data will generalize to predict the features of data sets on which it has not been trained. The general idea behind this technique is that a model is trained on a data set of known data, and then the accuracy of its predictions is tested against an independent data set. A common implementation is to divide a data set into k -folds, and then train the model in a round-robin fashion on all but one of the folds.

Hyper-parameter optimization is the problem of choosing a set of hyper-parameters for a learning algorithm, usually with the goal of optimizing a measure of the algorithm's performance on an independent data set. A hyper-parameter is a value that you must specify outside the model training procedure. Assumptions about hyper-parameter values can affect the flexibility and accuracy of the model. Decision trees have hyper-parameters, for example, such as the desired depth and number of leaves in the tree. You must set a misclassification penalty term for a support vector machine (SVM).

A common way to perform hyper-parameter optimization is to use a grid search, also called a **parameter sweep**. In a grid search, an exhaustive search is performed through the values of a specified subset of the hyper-parameter space for a learning algorithm. Cross-validation can supply a performance metric to sort out the optimal results produced by the grid search algorithm. If you use cross-validation hyper-parameter

sweeping, you can help limit problems like overfitting a model to training data. This way, the model retains the capacity to apply to the general set of data from which the training data was extracted.

Optimize a linear regression model with hyper-parameter sweeping

Next, split data into train and validation sets, use hyper-parameter sweeping on a training set to optimize the model, and evaluate on a validation set (linear regression).

```
# RECORD THE START TIME
val starttime = Calendar.getInstance().getTime()

# RENAME `tip_amount` AS A LABEL
val OneHotTRAINLabeled =
  OneHotTRAIN.select("tip_amount", "features").withColumnRenamed(existingName="tip_amount",newName="label")
val OneHotTESTLabeled =
  OneHotTEST.select("tip_amount", "features").withColumnRenamed(existingName="tip_amount",newName="label")
OneHotTRAINLabeled.cache()
OneHotTESTLabeled.cache()

# DEFINE THE ESTIMATOR FUNCTION: `THE LinearRegression()` FUNCTION
val lr = new LinearRegression().setLabelCol("label").setFeaturesCol("features").setMaxIter(10)

# DEFINE THE PARAMETER GRID
val paramGrid = new ParamGridBuilder().addGrid(lr.regParam, Array(0.1, 0.01,
0.001)).addGrid(lr.fitIntercept).addGrid(lr.elasticNetParam, Array(0.1, 0.5, 0.9)).build()

# DEFINE THE PIPELINE WITH A TRAIN/TEST VALIDATION SPLIT (75% IN THE TRAINING SET), AND THEN THE SPECIFY
ESTIMATOR, EVALUATOR, AND PARAMETER GRID
val trainPct = 0.75
val trainValidationSplit = new TrainValidationSplit().setEstimator(lr).setEvaluator(new
RegressionEvaluator).setEstimatorParamMaps(paramGrid).setTrainRatio(trainPct)

# RUN THE TRAIN VALIDATION SPLIT AND CHOOSE THE BEST SET OF PARAMETERS
val model = trainValidationSplit.fit(OneHotTRAINLabeled)

# MAKE PREDICTIONS ON THE TEST DATA BY USING THE MODEL WITH THE COMBINATION OF PARAMETERS THAT PERFORMS THE
BEST
val testResults = model.transform(OneHotTESTLabeled).select("label", "prediction")

# COMPUTE TEST SET R2
val evaluator = new
RegressionEvaluator().setLabelCol("label").setPredictionCol("prediction").setMetricName("r2")
val Test_R2 = evaluator.evaluate(testResults)

# GET THE TIME TO RUN THE CELL
val endtime = Calendar.getInstance().getTime()
val elapsedtime = ((endtime.getTime() - starttime.getTime())/1000).toString;
println("Time taken to run the above cell: " + elapsedtime + " seconds.");

println("Test R-sqr is: " + Test_R2);
```

Output:

Test R-sqr is: 0.6226484708501209

Optimize the binary classification model by using cross-validation and hyper-parameter sweeping

This section shows you how to optimize a binary classification model by using cross-validation and hyper-parameter sweeping. This uses the Spark ML `CrossValidator` function.

```

# RECORD THE START TIME
val starttime = Calendar.getInstance().getTime()

# CREATE DATA FRAMES WITH PROPERLY LABELED COLUMNS TO USE WITH THE TRAIN AND TEST SPLIT
val indexedTRAINwithCatFeatBinTargetRF =
indexedTRAINwithCatFeatBinTarget.select("labelBin","featuresCat").withColumnRenamed(existingName="labelBin",
newName="label").withColumnRenamed(existingName="featuresCat",newName="features")
val indexedTESTwithCatFeatBinTargetRF =
indexedTESTwithCatFeatBinTarget.select("labelBin","featuresCat").withColumnRenamed(existingName="labelBin",
newName="label").withColumnRenamed(existingName="featuresCat",newName="features")
indexedTRAINwithCatFeatBinTargetRF.cache()
indexedTESTwithCatFeatBinTargetRF.cache()

# DEFINE THE ESTIMATOR FUNCTION
val rf = new
RandomForestClassifier().setLabelCol("label").setFeaturesCol("features").setImpurity("gini").setSeed(1234).s
etFeatureSubsetStrategy("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.toString.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.toString.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

```

```

category_labelEstimatorCategory_labelEstimator,
          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 machine learning

3/5/2021 • 6 minutes to read • [Edit Online](#)

In this article, you learn about feature engineering and its role in enhancing data in machine learning. Learn from illustrative examples drawn from [Azure Machine Learning Studio \(classic\)](#) experiments.

- **Feature engineering:** The process of creating new features from raw data to increase the predictive power of the learning algorithm. Engineered features should capture additional information that is not easily apparent in the original feature set.
- **Feature selection:** The process of selecting the key subset of features to reduce the dimensionality of the training problem.

Normally **feature engineering** is applied first to generate additional features, and then **feature selection** is done to eliminate irrelevant, redundant, or highly correlated features.

Feature engineering and selection are part of the [modeling stage](#) of the Team Data Science Process (TDSP). To learn more about the TDSP and the data science lifecycle, see [What is the TDSP?](#)

What is feature engineering?

Training data consists of a matrix composed of rows and columns. Each row in the matrix is an observation or record. The columns of each row are the features that describe each record. The features specified in the experimental design should characterize the patterns in the data.

Although many of the raw data fields can be used directly to train a model, it's often necessary to create additional (engineered) features for an enhanced training dataset.

Engineered features that enhance training provide information that better differentiates the patterns in the data. But this process is something of an art. Sound and productive decisions often require domain expertise.

Example 1: Add temporal features for a regression model

Let's use the experiment [Demand forecasting of bikes rentals](#) in Azure Machine Learning Studio (classic) to demonstrate how to engineer features for a regression task. The objective of this experiment is to predict the demand for bike rentals within a specific month/day/hour.

Bike rental dataset

The [Bike Rental UCI dataset](#) is based on real data from a bike share company based in the United States. It represents the number of bike rentals within a specific hour of a day for the years 2011 and 2012. It contains 17,379 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 count, which represents the bike rentals within a specific hour. Count ranges from 1 to 977.

Create a feature engineering experiment

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 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.

Feature engineering using Studio (classic)

In the Studio (classic) experiment, these four training datasets are formed via four branches from the pre-processed input dataset. Except for 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 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.



Results

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. 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: Create features for text mining

Feature engineering is widely applied in tasks related to text mining such as document classification and sentiment analysis. Since individual pieces of raw text usually serve as the input data, the feature engineering process is needed to create the features involving word/phrase frequencies.

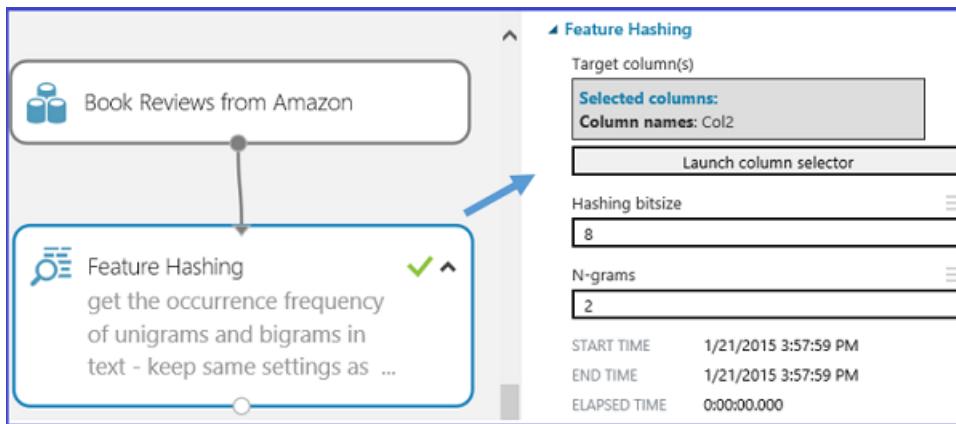
Feature hashing

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 applies a hash function to the features and using their hash values as indices directly.

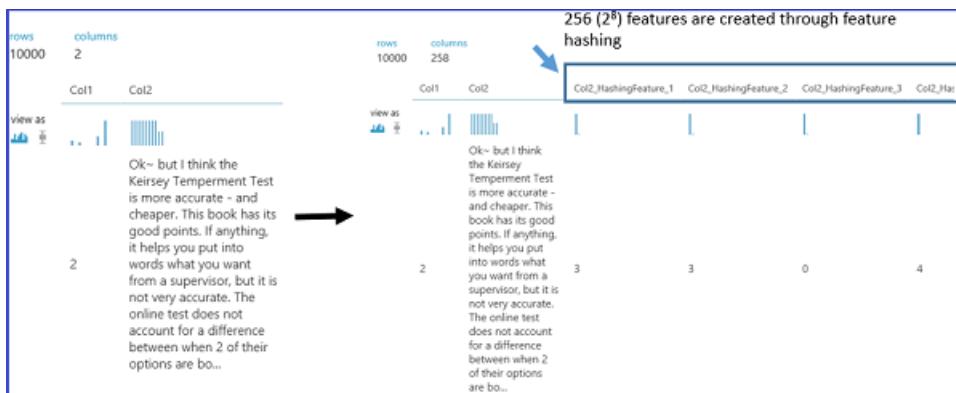
In Studio (classic), there is a [Feature Hashing](#) module that creates word/phrase features conveniently. Following figure shows an example of using this module. The input dataset contains two columns: the book rating ranging from 1 to 5, and the actual review content. The goal of this module is to 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/phrase 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, the selected features are a minimal set of independent variables that explain the patterns in the data and predict outcomes successfully.

It's not always necessarily to perform feature engineering or feature selection. It depends on the data, the algorithm selected, and the objective of the experiment.

Next steps

To create features for data in specific environments, see the following articles:

- Create features for data in SQL Server
- Create features for data in a Hadoop cluster using Hive queries

Create features for data in SQL Server using SQL and Python

3/5/2021 • 5 minutes 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 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:

- [Count based Feature Generation](#)
- [Binning Feature Generation](#)
- [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 new features 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

<https://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/region 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 level 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 goal 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. Once can also call a REST endpoint, such as Bing Maps API (see <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>

```

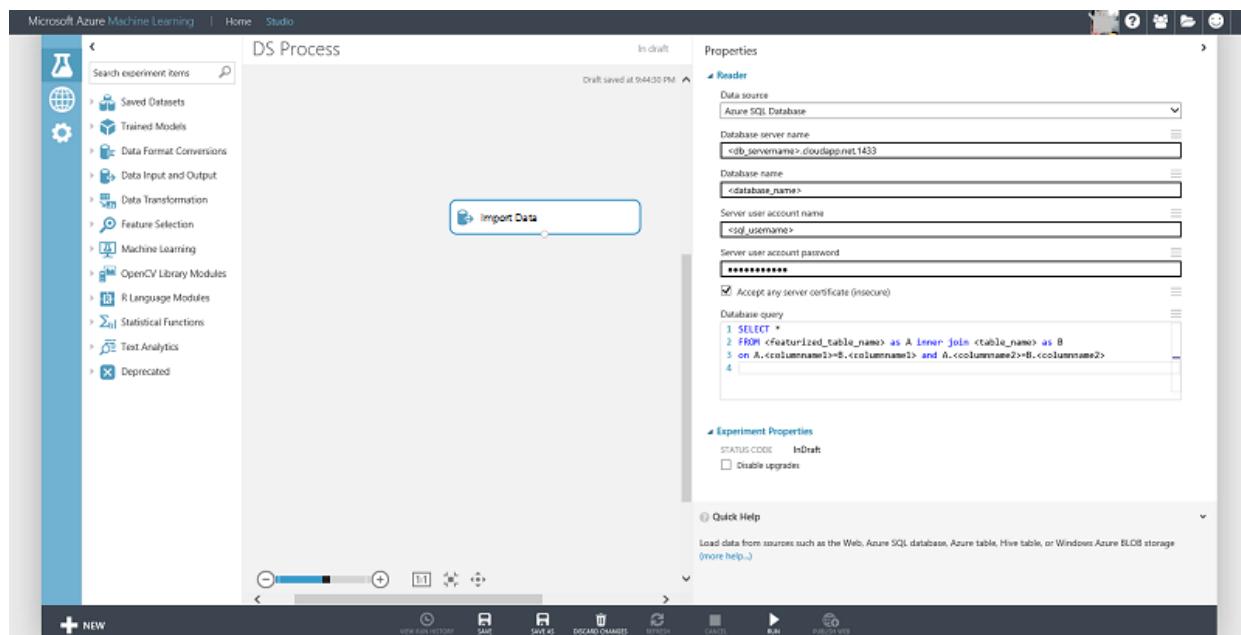
These location-based features can be further used to generate additional count features as described earlier.

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

3/5/2021 • 7 minutes 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 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 `<datetime field>` is in the default datetime format.

If a datetime field is not in the default format, you need to convert the datetime field into Unix time stamp first, and then convert the Unix time stamp to a datetime string that is in the default format. When the datetime is in default format, users can apply the embedded datetime UDFs to extract features.

```
select from_unixtime(unix_timestamp(<datetime field>,'<pattern of the datetime field>'))
from <databasename>.<tablename>;
```

In this query, if the *<datetime field>* has the pattern like *03/26/2015 12:04:39*, the *<pattern of the datetime field>* 'should be `'MM/dd/yyyy HH:mm:ss'`. To test it, users can run

```
select from_unixtime(unix_timestamp('05/15/2015 09:32:10','MM/dd/yyyy HH:mm:ss'))
from hivesampletable limit 1;
```

The *hivesampletable* 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 $\text{lat_or_lon} \pi / 180$, 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 4-GB 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"). Set this option:

```
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 options 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)

3/5/2021 • 3 minutes 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.

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 may be used for 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.

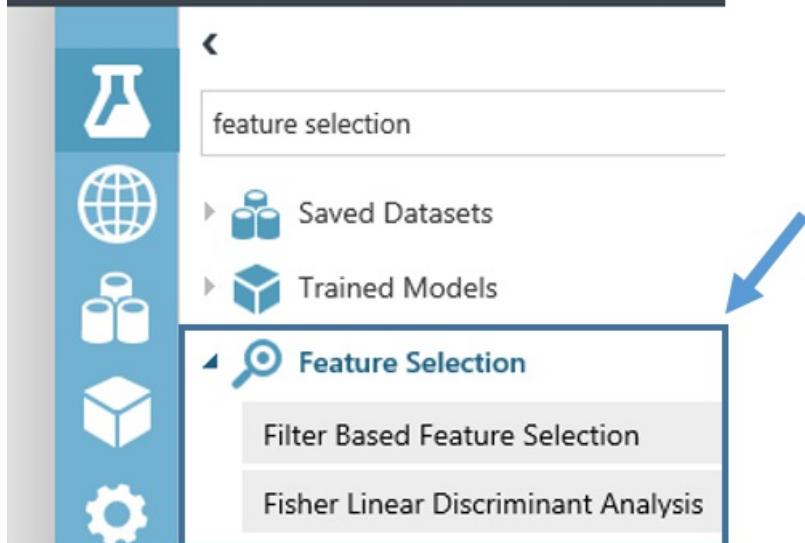
- First, feature selection often increases classification accuracy by eliminating irrelevant, redundant, or highly correlated features.
- Second, it decreases the number of features, which makes the model training process more efficient.

Efficiency is important for learners that are expensive to train such as support vector machines.

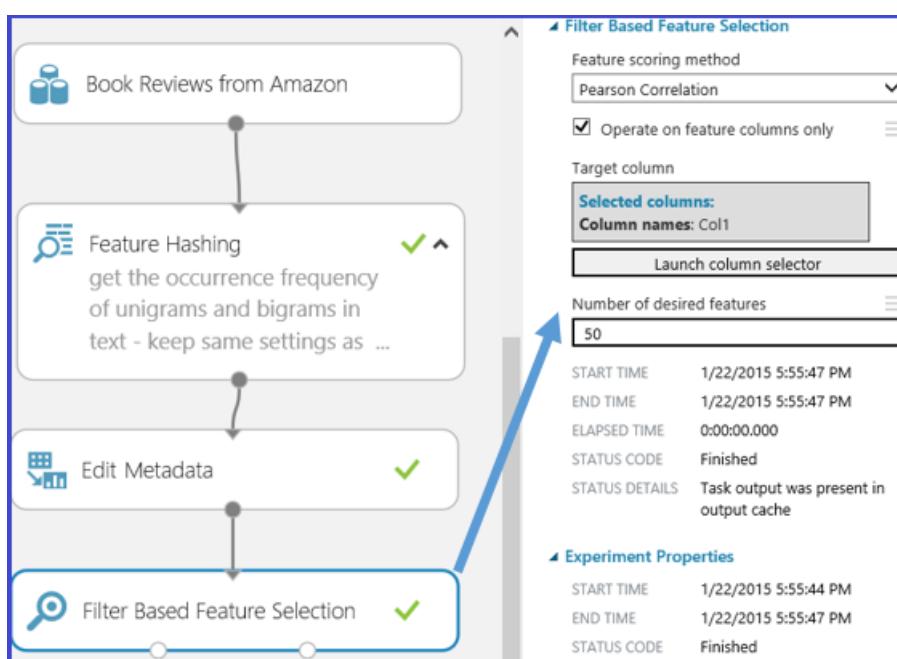
Although feature selection does seek to reduce the number of features in the dataset used to train the model, it is not referred to by the term "dimensionality reduction". Feature selection methods extract a subset of original features in the data without changing them. Dimensionality reduction methods 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:

rows 10000 columns 51

Col1 Col2_HashFeature_203 Col2_HashFeature_146 Col2_HashFeature_122 Col2

view as

	Col1	Col2_HashFeature_203	Col2_HashFeature_146	Col2_HashFeature_122	Col2
2	6	1	2	6	
2	6	4	7	5	
1	9	2	1	3	
2	5	2	2	3	
2	3	1	1	0	
2	11	6	5	5	
1	2	2	3	1	
2	4	3	2	1	
2	2	2	6	3	
2	2	0	1	0	
1	11	4	3	5	
2	1	0	5	0	
2	2	1	2	2	
1	1	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:

rows 1 columns 51

Col1 Col2_HashFeature_203 Col2_HashFeature_146 Col2_HashFeature_122 Col2_Hash

view as

	Col1	Col2_HashFeature_203	Col2_HashFeature_146	Col2_HashFeature_122	Col2_Hash
1		0.083607	0.060681	0.05716	0.056381

By applying this **Filter-Based Feature Selection** module, 50 out of 256 features are selected because they have the most correlated features with the target variable "Col1", based on the scoring method "Pearson Correlation".

Conclusion

Feature engineering and feature selection are two commonly Engineered and selected features increase the efficiency of the training process that 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.

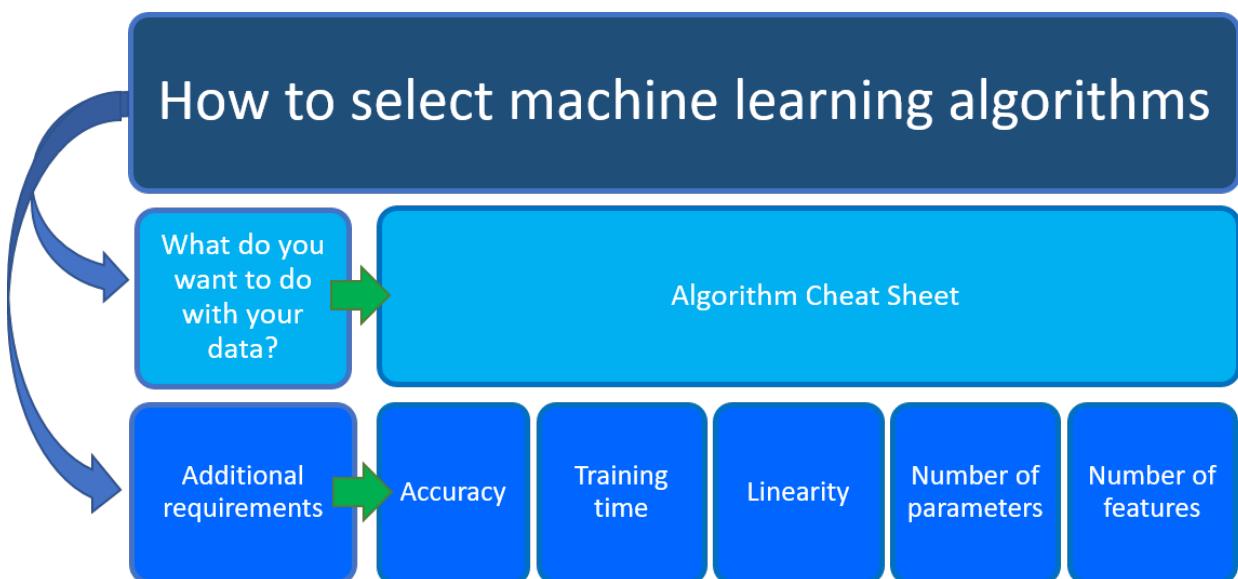
It is not always necessarily to perform feature engineering or feature selection. Whether it is needed or not depends on the data collected, the algorithm selected, and the objective of the experiment.

How to select algorithms for Azure Machine Learning

3/5/2021 • 7 minutes to read • [Edit Online](#)

A common question is "Which machine learning algorithm should I use?" The algorithm you select depends primarily on two different aspects of your data science scenario:

- **What you want to do with your data?** Specifically, what is the business question you want to answer by learning from your past data?
- **What are the requirements of your data science scenario?** Specifically, what is the accuracy, training time, linearity, number of parameters, and number of features your solution supports?



Business scenarios and the Machine Learning Algorithm Cheat Sheet

The [Azure Machine Learning Algorithm Cheat Sheet](#) helps you with the first consideration: **What you want to do with your data?** On the Machine Learning Algorithm Cheat Sheet, look for task you want to do, and then find a [Azure Machine Learning designer](#) algorithm for the predictive analytics solution.

Machine Learning designer provides a comprehensive portfolio of algorithms, such as [Multiclass Decision Forest](#), [Recommendation systems](#), [Neural Network Regression](#), [Multiclass Neural Network](#), and [K-Means Clustering](#). Each algorithm is designed to address a different type of machine learning problem. See the [Machine Learning designer algorithm and module reference](#) for a complete list along with documentation about how each algorithm works and how to tune parameters to optimize the algorithm.

NOTE

To download the machine learning algorithm cheat sheet, go to [Azure Machine learning algorithm cheat sheet](#).

Along with guidance in the Azure Machine Learning Algorithm Cheat Sheet, keep in mind other requirements when choosing a machine learning algorithm for your solution. Following are additional factors to consider, such as the accuracy, training time, linearity, number of parameters and number of features.

Comparison of machine learning algorithms

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.

The following table summarizes some of the most important characteristics of algorithms from the classification, regression, and clustering families:

ALGORITHM	ACCURACY	TRAINING TIME	LINEARITY	PARAMETERS	NOTES
Classification family					
Two-Class logistic regression	Good	Fast	Yes	4	
Two-class decision forest	Excellent	Moderate	No	5	Shows slower scoring times. Suggest not working with One-vs-All Multiclass, because of slower scoring times caused by tread locking in accumulating tree predictions
Two-class boosted decision tree	Excellent	Moderate	No	6	Large memory footprint
Two-class neural network	Good	Moderate	No	8	
Two-class averaged perceptron	Good	Moderate	Yes	4	
Two-class support vector machine	Good	Fast	Yes	5	Good for large feature sets
Multiclass logistic regression	Good	Fast	Yes	4	
Multiclass decision forest	Excellent	Moderate	No	5	Shows slower scoring times
Multiclass boosted decision tree	Excellent	Moderate	No	6	Tends to improve accuracy with some small risk of less coverage

ALGORITHM	ACCURACY	TRAINING TIME	LINEARITY	PARAMETERS	NOTES
Multiclass neural network	Good	Moderate	No	8	
One-vs-all multiclass	-	-	-	-	See properties of the two-class method selected
Regression family					
Linear regression	Good	Fast	Yes	4	
Decision forest regression	Excellent	Moderate	No	5	
Boosted decision tree regression	Excellent	Moderate	No	6	Large memory footprint
Neural network regression	Good	Moderate	No	8	
Clustering family					
K-means clustering	Excellent	Moderate	Yes	8	A clustering algorithm

Requirements for a data science scenario

Once you know what you want to do with your data, you need to determine additional requirements for your solution.

Make choices and possibly trade-offs for the following requirements:

- Accuracy
- Training time
- Linearity
- Number of parameters
- Number of features

Accuracy

Accuracy in machine learning measures the effectiveness of a model as the proportion of true results to total cases. In Machine Learning designer, the [Evaluate Model module](#) computes a set of industry-standard evaluation metrics. You can use this module to measure the accuracy of a trained model.

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 is the case, you may be able to cut your processing time dramatically by sticking with more approximate methods. Approximate methods also naturally tend to avoid overfitting.

There are three ways to use the Evaluate Model module:

- Generate scores over your training data in order to evaluate the model
- Generate scores on the model, but compare those scores to scores on a reserved testing set
- Compare scores for two different but related models, using the same set of data

For a complete list of metrics and approaches you can use to evaluate the accuracy of machine learning models, see [Evaluate Model module](#).

Training time

In supervised learning, training means using historical data to build a machine learning model that minimizes errors. 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. You might choose a specific algorithm because you have a time limitation, especially when the data set is large.

In Machine Learning designer, creating and using a machine learning model is typically a three-step process:

1. Configure a model, by choosing a particular type of algorithm, and then defining its parameters or hyperparameters.
2. Provide a dataset that is labeled and has data compatible with the algorithm. Connect both the data and the model to [Train Model module](#).
3. After training is completed, use the trained model with one of the [scoring modules](#) to make predictions on new data.

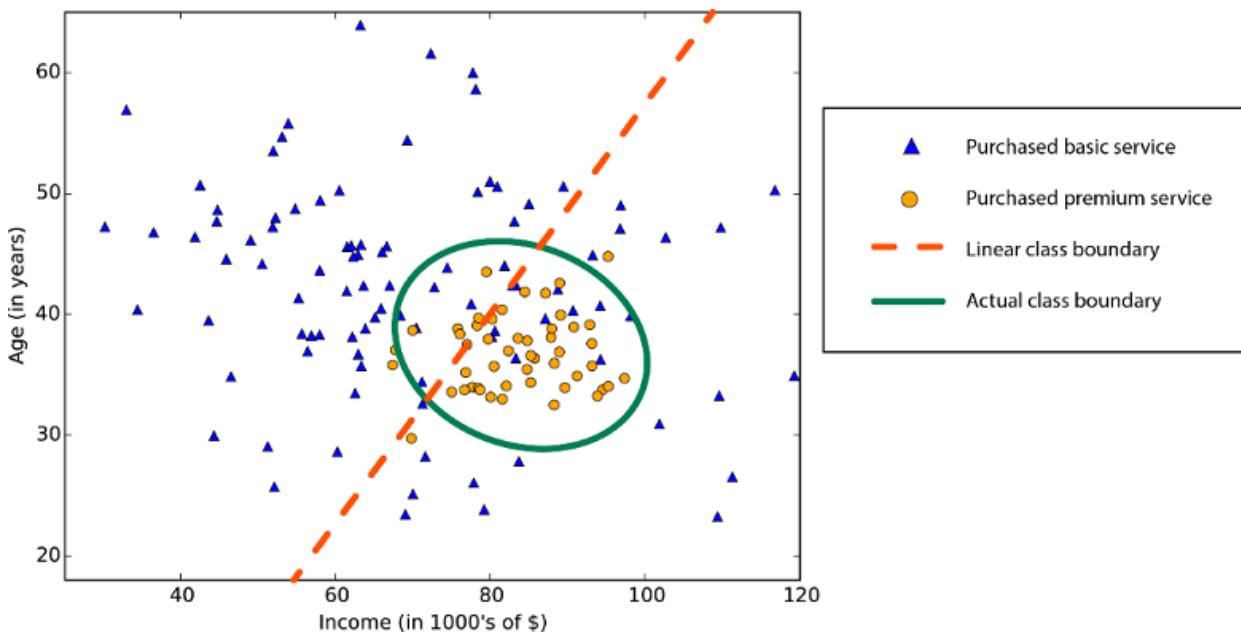
Linearity

Linearity in statistics and machine learning means that there is a linear relationship between a variable and a constant in your dataset. For example, linear classification algorithms assume that classes can be separated by a straight line (or its higher-dimensional analog).

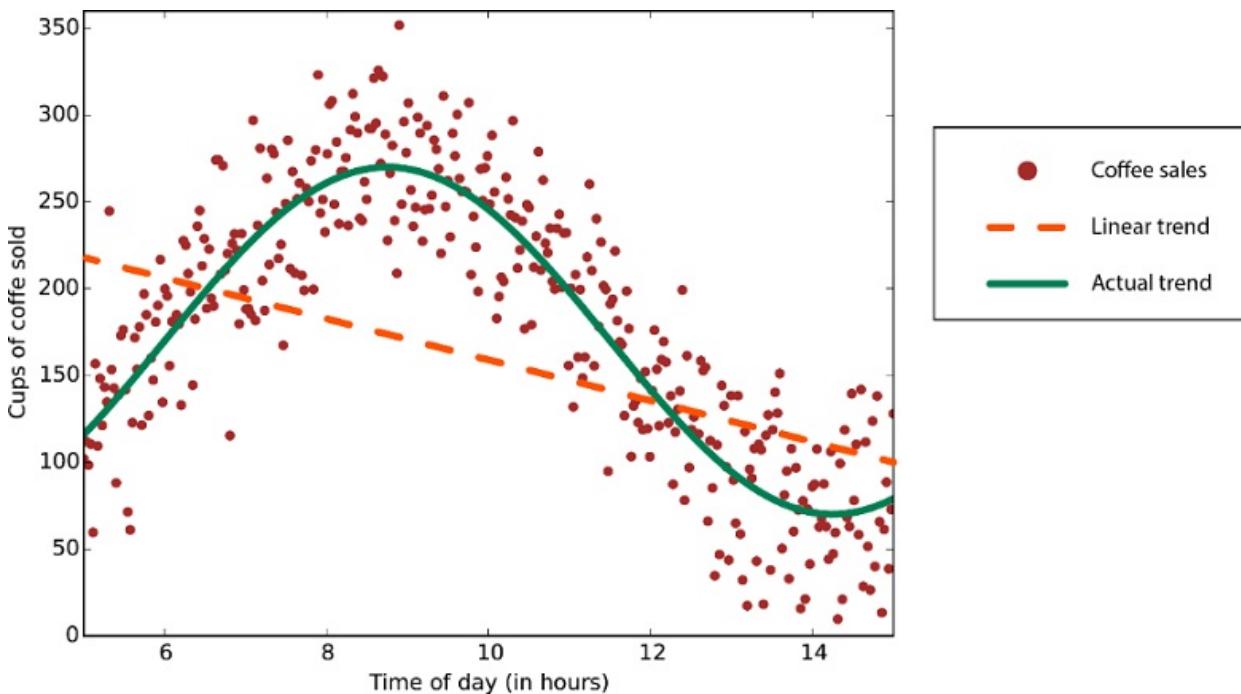
Lots of machine learning algorithms make use of linearity. In Azure Machine Learning designer, they include:

- [Multiclass logistic regression](#)
- [Two-class logistic regression](#)
- [Support vector machines](#)

Linear regression algorithms assume that data trends follow a straight line. This assumption isn't bad for some problems, but for others it reduces accuracy. Despite their drawbacks, linear algorithms are popular as a first strategy. They tend to be algorithmically simple and fast to train.



Nonlinear 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.

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 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 the [Tune Model Hyperparameters module](#) in Machine Learning designer: The goal of this module is to determine the optimum hyperparameters for a machine learning model. The module builds and tests multiple models by using different combinations of settings. It compares metrics over all models to get the combinations of settings.

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

In machine learning, a feature is a quantifiable variable of the phenomenon you are trying to analyze. 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.

A large number of features can bog down some learning algorithms, making training time unfeasibly long.

[Support vector machines](#) are particularly well suited to scenarios with a high number of features. For this reason, they have been used in many applications from information retrieval to text and image classification. Support vector machines can be used for both classification and regression tasks.

Feature selection refers to the process of applying statistical tests to inputs, given a specified output. The goal is to determine which columns are more predictive of the output. The [Filter Based Feature Selection module](#) in Machine Learning designer provides multiple feature selection algorithms to choose from. The module includes correlation methods such as Pearson correlation and chi-squared values.

You can also use the [Permutation Feature Importance module](#) to compute a set of feature importance scores for your dataset. You can then leverage these scores to help you determine the best features to use in a model.

Next steps

- [Learn more about Azure Machine Learning designer](#)
- For descriptions of all the machine learning algorithms available in Azure Machine Learning designer, see [Machine Learning designer algorithm and module reference](#)
- To explore the relationship between deep learning, machine learning, and AI, see [Deep Learning vs. Machine Learning](#)

Machine Learning Algorithm Cheat Sheet for Azure Machine Learning designer

3/5/2021 • 2 minutes to read • [Edit Online](#)

The **Azure Machine Learning Algorithm Cheat Sheet** helps you choose the right algorithm from the designer for a predictive analytics model.

Azure Machine Learning has a large library of algorithms from the *classification, recommender systems, clustering, anomaly detection, regression*, and *text analytics* families. Each is designed to address a different type of machine learning problem.

For additional guidance, see [How to select algorithms](#)

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.

How to use 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 cheat sheet 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 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.

To learn more about the algorithms in Azure Machine Learning designer, go to the [Algorithm and module reference](#).

Kinds of machine learning

There are three main categories of machine learning: *supervised learning*, *unsupervised learning*, and *reinforcement learning*.

Supervised 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.

Unsupervised learning

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. Unsupervised learning groups data into clusters, as K-means does, or finds different ways of looking at complex data so that it appears simpler.

Reinforcement learning

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 signal, the algorithm modifies its strategy in order to achieve the highest reward.

Next steps

- See additional guidance on [How to select algorithms](#)
- [Learn about studio in Azure Machine Learning and the Azure portal.](#)
- [Tutorial: Build a prediction model in Azure Machine Learning designer.](#)
- [Learn about deep learning vs. machine learning.](#)

Deploy models to production to play an active role in making business decisions

3/5/2021 • 2 minutes 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:

- [Where to deploy models with 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, [A/B testing](#) may be used to compare model performance.

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.

Machine Learning Anomaly Detection API

3/5/2021 • 10 minutes to read • [Edit Online](#)

NOTE

This item is under maintenance. We encourage you to use the [Anomaly Detector API service](#) powered by a gallery of Machine Learning algorithms under Azure Cognitive Services to detect anomalies from business, operational, and IoT metrics.

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](#)

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 Azure Machine Learning Studio (classic) 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 [Azure Machine Learning Studio \(classic\) 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 that 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. These two requirements, along with sample code for calling the API, are available from the [Azure Machine Learning Studio \(classic\) web services](#) page. Navigate to the desired API, and then click the "Consume" tab to find them. You can call the API as a Swagger API (that is, with the URL parameter `format=swagger`) or as a non-Swagger API (that is, 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"
    }
}
```

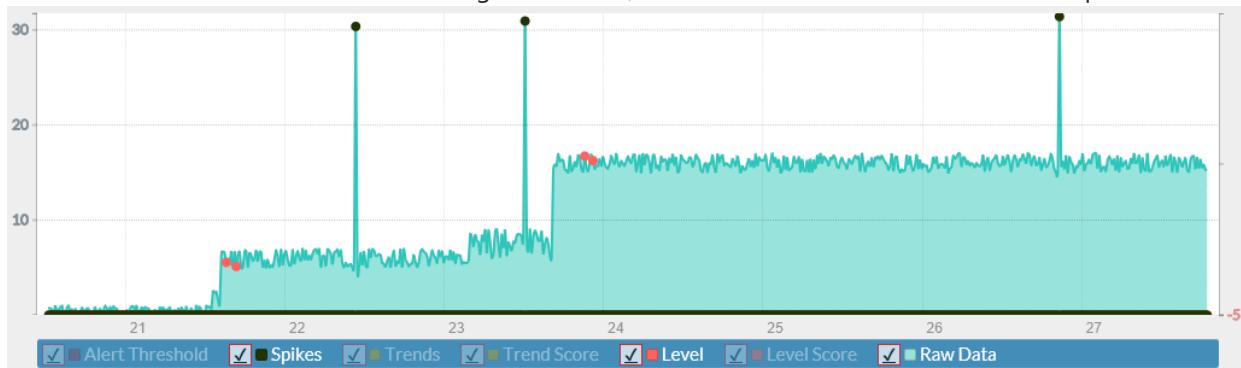
Sample Response

In order to see the `ColumnNames` field, you must include `details=true` as a URL parameter in your request. See the tables below for the meaning behind each of these fields.

```
{
  "Results": {
    "output1": {
      "type": "table",
      "value": {
        "Values": [
          ["5/30/2010 6:07:00 PM", "1", "1", "0", "0", "-0.687952590518378", "0", "-0.687952590518378", "0", "-0.687952590518378", "0"],
          ["5/30/2010 6:08:00 PM", "1.4", "1.4", "0", "0", "-1.07030497733224", "0", "-0.884548154298423", "0", "-1.07030497733224", "0"],
          ["5/30/2010 6:09:00 PM", "1.1", "1.1", "0", "0", "-1.30229513613974", "0", "-1.173800281031", "0", "-1.30229513613974", "0"]
        ],
        "ColumnNames": ["Time", "OriginalData", "ProcessedData", "TSpike", "ZSpike", "BiLevelChangeScore", "BiLevelChangeAlert", "PosTrendScore", "PosTrendAlert", "NegTrendScore", "NegTrendAlert"],
        "ColumnTypes": ["DateTime", "Double", "Double", "Double", "Double", "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 two distinct level changes, and three 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 three 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	<i>zspikedetector:sensitivity</i> : take integer value in the range 1-10, default: 3; Higher values will catch more extreme values making it less sensitive	ZSpike: binary values – '1' if a spike/dip is detected, '0' otherwise
Slow Trend Detector	Slow Trend Detector	Detect slow positive trend as per the set sensitivity	<i>trenddetector:sensitivity</i> : threshold on detector score (default: 3.25, 3.25 – 5 is a reasonable range to select from; The higher the less sensitive)	tscore: floating number representing anomaly score on trend
Level Change Detectors	Bidirectional Level Change Detector	Detect both upward and downward level change as per the set sensitivity	<i>bileveldetector:sensitivity</i> : threshold on detector score (default: 3.25, 3.25 – 5 is a reasonable range to select from; The higher the less sensitive)	rpscore: floating number representing anomaly score on upward and downward level change

Parameters

More detailed information on these input parameters is listed in the table below:

INPUT PARAMETERS	DESCRIPTION	DEFAULT SETTING	TYPE	VALID RANGE	SUGGESTED RANGE
detectors.history.window	History (in # of data points) used for anomaly score computation	500	integer	10-2000	Time-series dependent
detectors.spikesdips	Whether to detect only spikes, only dips, or both	Both	enumerated	Both, Spikes, Dips	Both
bileveldetector.sensitivity	Sensitivity for bidirectional level change detector.	3.25	double	None	3.25-5 (Lesser values mean more sensitive)
trenddetector.sensitivity	Sensitivity for positive trend detector.	3.25	double	None	3.25-5 (Lesser values mean more sensitive)
tspikedetector.sensitivity	Sensitivity for TSpike Detector	3	integer	1-10	3-5 (Lesser values mean more sensitive)

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

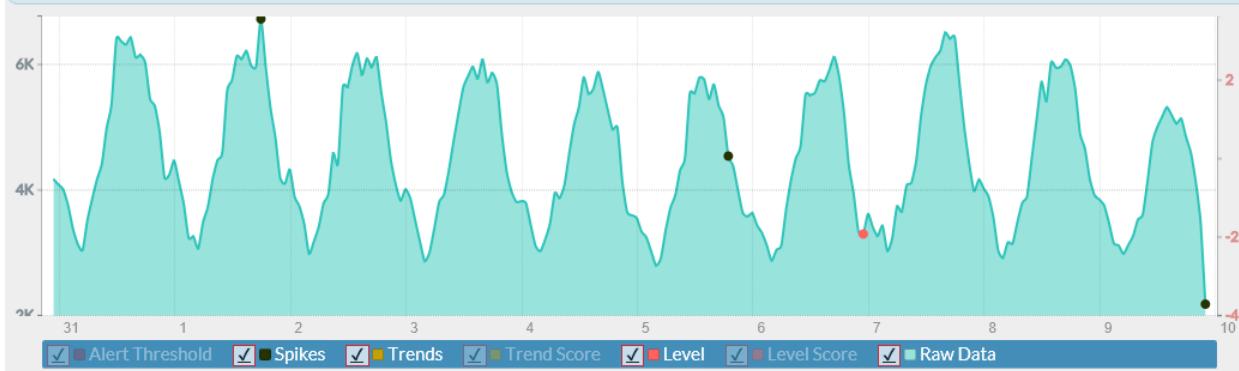
The API runs all detectors on your time series data and returns anomaly scores and binary spike indicators for each point in time. The table below lists outputs from the API.

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 first black dot), two dips (the second black dot and one at the end), and one level change (red dot). 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.spikesAndDips	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.enable	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

OUTPUTS	DESCRIPTION
ProcessedData	<p>Either of the following options:</p> <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 option is the same as OriginalData
TSpike	Binary indicator to indicate whether a spike is detected by TSpike Detector
ZSpike	Binary indicator to indicate whether a spike is detected by ZSpike Detector
BiLevelChangeScore	A floating number representing anomaly score on level change
BiLevelChangeAlert	1/0 value indicating there is a level change anomaly based on the input sensitivity
PosTrendScore	A floating number representing anomaly score on positive trend
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

Azure AI guide for predictive maintenance solutions

3/5/2021 • 42 minutes to read • [Edit Online](#)

Summary

Predictive maintenance (**PdM**) is a popular application of predictive analytics that can help businesses in several industries achieve high asset utilization and savings in operational costs. This guide brings together the business and analytical guidelines and best practices to successfully develop and deploy PdM solutions using the [Microsoft Azure AI platform](#) technology.

For starters, this guide introduces industry-specific business scenarios and the process of qualifying these scenarios for PdM. The data requirements and modeling techniques to build PdM solutions are also provided. The main content of the guide is on the data science process - including the steps of data preparation, feature engineering, model creation, and model operationalization. To complement these key concepts, this guide lists a set of solution templates to help accelerate PdM application development. The guide also points to useful training resources for the practitioner to learn more about the AI behind the data science.

Data Science guide overview and target audience

The first half of this guide describes typical business problems, the benefits of implementing PdM to address these problems, and lists some common use cases. Business decision makers (BDMs) will benefit from this content. The second half explains the data science behind PdM, and provides a list of PdM solutions built using the principles outlined in this guide. It also provides learning paths and pointers to training material. Technical decision makers (TDMs) will find this content useful.

START WITH ...	IF YOU ARE ...
Business case for predictive maintenance	a business decision maker (BDM) looking to reduce downtime and operational costs, and improve utilization of equipment
Data Science for predictive maintenance	a technical decision maker (TDM) evaluating PdM technologies to understand the unique data processing and AI requirements for predictive maintenance
Solution templates for predictive maintenance	a software architect or AI Developer looking to quickly stand up a demo or a proof-of-concept
Training resources for predictive maintenance	any or all of the above, and want to learn the foundational concepts behind the data science, tools, and techniques.

Prerequisite knowledge

The BDM content does not expect the reader to have any prior data science knowledge. For the TDM content, basic knowledge of statistics and data science is helpful. Knowledge of Azure Data and AI services, Python, R, XML, and JSON is recommended. AI techniques are implemented in Python and R packages. Solution templates are implemented using Azure services, development tools, and SDKs.

Business case for predictive maintenance

Businesses require critical equipment to be running at peak efficiency and utilization to realize their return on capital investments. These assets could range from aircraft engines, turbines, elevators, or industrial chillers - that cost millions - down to everyday appliances like photocopiers, coffee machines, or water coolers.

- By default, most businesses rely on *corrective maintenance*, where parts are replaced as and when they fail. Corrective maintenance ensures parts are used completely (therefore not wasting component life), but costs the business in downtime, labor, and unscheduled maintenance requirements (off hours, or inconvenient locations).
- At the next level, businesses practice *preventive maintenance*, where they determine the useful lifespan for a part, and maintain or replace it before a failure. Preventive maintenance avoids unscheduled and catastrophic failures. But the high costs of scheduled downtime, under-utilization of the component during its useful lifetime, and labor still remain.
- The goal of *predictive maintenance* is to optimize the balance between corrective and preventative maintenance, by enabling *just in time* replacement of components. This approach only replaces those components when they are close to a failure. By extending component lifespans (compared to preventive maintenance) and reducing unscheduled maintenance and labor costs (over corrective maintenance), businesses can gain cost savings and competitive advantages.

Business problems in PdM

Businesses face high operational risk due to unexpected failures and have limited insight into the root cause of problems in complex systems. Some of the key business questions are:

- Detect anomalies in equipment or system performance or functionality.
- Predict whether an asset may fail in the near future.
- Estimate the remaining useful life of an asset.
- Identify the main causes of failure of an asset.
- Identify what maintenance actions need to be done, by when, on an asset.

Typical goal statements from PdM are:

- Reduce operational risk of mission critical equipment.
- Increase rate of return on assets by predicting failures before they occur.
- Control cost of maintenance by enabling just-in-time maintenance operations.
- Lower customer attrition, improve brand image, and lost sales.
- Lower inventory costs by reducing inventory levels by predicting the reorder point.
- Discover patterns connected to various maintenance problems.
- Provide KPIs (key performance indicators) such as health scores for asset conditions.
- Estimate remaining lifespan of assets.
- Recommend timely maintenance activities.
- Enable just in time inventory by estimating order dates for replacement of parts.

These goal statements are the starting points for:

- *data scientists* to analyze and solve specific predictive problems.
- *cloud architects and developers* to put together an end to end solution.

Qualifying problems for predictive maintenance

It is important to emphasize that not all use cases or business problems can be effectively solved by PdM. There are three important qualifying criteria that need to be considered during problem selection:

- The problem has to be predictive in nature; that is, there should be a target or an outcome to predict. The problem should also have a clear path of action to prevent failures when they are detected.
- The problem should have a record of the operational history of the equipment that contains *both good and bad outcomes*. The set of actions taken to mitigate bad outcomes should also be available as part of these records. Error reports, maintenance logs of performance degradation, repair, and replace logs are also

important. In addition, repairs undertaken to improve them, and replacement records are also useful.

- The recorded history should be reflected in *relevant* data that is of *sufficient* enough quality to support the use case. For more information about data relevance and sufficiency, see [Data requirements for predictive maintenance](#).
- Finally, the business should have domain experts who have a clear understanding of the problem. They should be aware of the internal processes and practices to be able to help the analyst understand and interpret the data. They should also be able to make the necessary changes to existing business processes to help collect the right data for the problems, if needed.

Sample PdM use cases

This section focuses on a collection of PdM use cases from several industries such as Aerospace, Utilities, and Transportation. Each section starts with a business problem, and discusses the benefits of PdM, the relevant data surrounding the business problem, and finally the benefits of a PdM solution.

BUSINESS PROBLEM	BENEFITS FROM PDM
Aviation	
<i>Flight delay and cancellations</i> due to mechanical problems. Failures that cannot be repaired in time may cause flights to be canceled, and disrupt scheduling and operations.	PdM solutions can predict the probability of an aircraft being delayed or canceled due to mechanical failures.
<i>Aircraft engine parts failure.</i> Aircraft engine part replacements are among the most common maintenance tasks within the airline industry. Maintenance solutions 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.
Finance	
<i>ATM failure</i> is a common problem within the banking industry. The problem here is to report the probability that an ATM cash withdrawal transaction gets interrupted due to a paper jam or part failure in the cash dispenser. Based on predictions of transaction failures, ATMs can be serviced proactively to prevent failures from occurring.	Rather than allow the machine to fail midway through a transaction, the desirable alternative is to program the machine to deny service based on the prediction.
Energy	
<i>Wind turbine failures.</i> Wind turbines are the main energy source in environmentally responsible countries/regions, and involve high capital costs. A key component in wind turbines is the generator motor, whose failure renders the turbine ineffective. It is also highly expensive to fix.	Predicting KPIs such as MTTF (mean time to failure) can help the energy companies prevent turbine failures, and ensure minimal downtime. Failure probabilities will inform technicians to monitor turbines that are likely to fail soon, and schedule time-based maintenance regimes. Predictive models provide insights into different factors that contribute to the failure, which helps technicians better understand the root causes of problems.
<i>Circuit breaker failures.</i> Distribution of electricity to homes and businesses requires power lines to be operational at all times to guarantee energy delivery. Circuit breakers help limit or avoid damage to power lines during overloading or adverse weather conditions. The business problem here is to predict circuit breaker failures.	PdM solutions help reduce repair costs and increase the lifespan of equipment such as circuit breakers. They help improve the quality of the power network by reducing unexpected failures and service interruptions.
Transportation and logistics	

<p><i>Elevator door failures:</i> Large elevator companies provide a full stack service for millions of functional elevators around the world. Elevator safety, reliability, and uptime are the main concerns for their customers. These companies track these and various other attributes via sensors, to help them with corrective and preventive maintenance. In an elevator, the most prominent customer problem is malfunctioning elevator doors. The business problem in this case is to provide a knowledge base predictive application that predicts the potential causes of door failures.</p>	<p>Elevators are capital investments for potentially a 20-30 year lifespan. So each potential sale can be highly competitive; hence expectations for service and support are high. Predictive maintenance can provide these companies with an advantage over their competitors in their product and service offerings.</p>
<p><i>Wheel failures:</i> Wheel failures account for half of all train derailments and cost billions to the global rail industry. Wheel failures also cause rails to deteriorate, sometimes causing the rail to break prematurely. Rail breaks lead to catastrophic events such as derailments. To avoid such instances, railways monitor the performance of wheels and replace them in a preventive manner. The business problem here is the prediction of wheel failures.</p>	<p>Predictive maintenance of wheels will help with just-in-time replacement of wheels</p>
<p><i>Subway train door failures:</i> A major reason for delays in subway operations is door failures of train cars. The business problem here is to predict train door failures.</p>	<p>Early awareness of a door failure, or the number of days until a door failure, will help the business optimize train door servicing schedules.</p>

The next section gets into the details of how to realize the PdM benefits discussed above.

Data Science for predictive maintenance

This section provides general guidelines of data science principles and practice for PdM. It is intended to help a TDM, solution architect, or a developer understand the prerequisites and process for building end-to-end AI applications for PdM. You can read this section along with a review of the demos and proof-of-concept templates listed in [Solution Templates for predictive maintenance](#). You can then use these principles and best practices to implement your PdM solution in Azure.

NOTE

This guide is NOT intended to teach the reader Data Science. Several helpful sources are provided for further reading in the section for [training resources for predictive maintenance](#). The [solution templates](#) listed in the guide demonstrate some of these AI techniques for specific PdM problems.

Data requirements for predictive maintenance

The success of any learning depends on (a) the quality of what is being taught, and (b) the ability of the learner. Predictive models learn patterns from historical data, and predict future outcomes with certain probability based on these observed patterns. A model's predictive accuracy depends on the relevancy, sufficiency, and quality of the training and test data. The new data that is 'scored' using this model should have the same features and schema as the training/test data. The feature characteristics (type, density, distribution, and so on) of new data should match that of the training and test data sets. The focus of this section is on such data requirements.

Relevant data

First, the data has to be *relevant to the problem*. Consider the *wheel failure* use case discussed above - the training data should contain features related to the wheel operations. If the problem was to predict the failure of

the *traction system*, the training data has to encompass all the different components for the traction system. The first case targets a specific component whereas the second case targets the failure of a larger subsystem. The general recommendation is to design prediction systems about specific components rather than larger subsystems, since the latter will have more dispersed data. The domain expert (see [Qualifying problems for predictive maintenance](#)) should help in selecting the most relevant subsets of data for the analysis. The relevant data sources are discussed in greater detail in [Data preparation for predictive maintenance](#).

Sufficient data

Two questions are commonly asked with regard to failure history data: (1) "How many failure events are required to train a model?" (2) "How many records is considered as "enough"?" There are no definitive answers, but only rules of thumb. For (1), more the number of failure events, better the model. For (2), and the exact number of failure events depends on the data and the context of the problem being solved. But on the flip side, if a machine fails too often then the business will replace it, which will reduce failure instances. Here again, the guidance from the domain expert is important. However, there are methods to cope with the issue of *rare events*. They are discussed in the section [Handling imbalanced data](#).

Quality data

The quality of the data is critical - each predictor attribute value must be *accurate* in conjunction with the value of the target variable. Data quality is a well-studied area in statistics and data management, and hence out of scope for this guide.

NOTE

There are several resources and enterprise products to deliver quality data. A sample of references is provided below:

- Dasu, T, Johnson, T, Exploratory Data Mining and Data Cleaning, Wiley, 2003.
- [Exploratory Data Analysis, Wikipedia](#)
- [Hellerstein, J, Quantitative Data Cleaning for Large Databases](#)
- [de Jonge, E, van der loo, M, Introduction to Data Cleaning with R](#)

Data preparation for predictive maintenance

Data sources

The relevant data sources for predictive maintenance include, but are not limited to:

- Failure history
- Maintenance/repair history
- Machine operating conditions
- Equipment metadata

Failure history

Failure events are rare in PdM applications. However, when building prediction models, the algorithm needs to learn about a component's normal operational pattern, as well as its failure patterns. So the training data should contain sufficient number of examples from both categories. Maintenance records and parts replacement history are good sources to find failure events. With the help of some domain knowledge, anomalies in the training data can also be defined as failures.

Maintenance/repair history

Maintenance history of an asset contains details about components replaced, repair activities performed etc. These events record degradation patterns. Absence of this crucial information in the training data can lead to misleading model results. Failure history can also be found within maintenance history as special error codes, or order dates for parts. Additional data sources that influence failure patterns should be investigated and provided by domain experts.

Machine operating conditions

Sensor based (or other) streaming data of the equipment in operation is an important data source. A key assumption in PdM is that a machine's health status degrades over time during its routine operation. The data is expected to contain time-varying features that capture this aging pattern, and any anomalies that lead to degradation. The temporal aspect of the data is required for the algorithm to learn the failure and non-failure patterns over time. Based on these data points, the algorithm learns to predict how many more units of time a machine can continue to work before it fails.

Static feature data

Static features are metadata about the equipment. Examples are the equipment make, model, manufactured date, start date of service, location of the system, and other technical specifications.

Examples of relevant data for the [sample PdM use cases](#) are tabulated below:

USE CASE	EXAMPLES OF RELEVANT DATA
<i>Flight delay and cancellations</i>	Flight route information in the form of flight legs and page logs. Flight leg data includes routing details such as departure/arrival date, time, airport, layovers etc. Page log includes a series of error and maintenance codes recorded by the ground maintenance personnel.
<i>Aircraft engine parts failure</i>	Data collected from sensors in the aircraft that provide information on the condition of the various parts. Maintenance records help identify when component failures occurred and when they were replaced.
<i>ATM Failure</i>	Sensor readings for each transaction (depositing cash/check) and dispensing of cash. Information on gap measurement between notes, note thickness, note arrival distance, check attributes etc. Maintenance records that provide error codes, repair information, last time the cash dispenser was refilled.
<i>Wind turbine failure</i>	Sensors monitor turbine conditions such as temperature, wind direction, power generated, generator speed etc. Data is gathered from multiple wind turbines from wind farms located in various regions. Typically, each turbine will have multiple sensor readings relaying measurements at a fixed time interval.
<i>Circuit breaker failures</i>	Maintenance logs that include corrective, preventive, and systematic actions. Operational data that includes automatic and manual commands sent to circuit breakers such as for open and close actions. Device metadata such as date of manufacture, location, model, etc. Circuit breaker specifications such as voltage levels, geolocation, ambient conditions.
<i>Elevator door failures</i>	Elevator metadata such as type of elevator, manufactured date, maintenance frequency, building type, and so on. Operational information such as number of door cycles, average door close time. Failure history with causes.
<i>Wheel failures</i>	Sensor data that measures wheel acceleration, braking instances, driving distance, velocity etc. Static information on wheels like manufacturer, manufactured date. Failure data inferred from part order database that track order dates and quantities.

USE CASE	EXAMPLES OF RELEVANT DATA
<i>Subway train door failures</i>	Door opening and closing times, other operational data such as current condition of train doors. Static data would include asset identifier, time, and condition value columns.

Data types

Given the above data sources, the two main data types observed in PdM domain are:

- *Temporal data*: Operational telemetry, machine conditions, work order types, priority codes that will have timestamps at the time of recording. Failure, maintenance/repair, and usage history will also have timestamps associated with each event.
- *Static data*: Machine features and operator features in general are static since they describe the technical specifications of machines or operator attributes. If these features could change over time, they should also have timestamps associated with them.

Predictor and target variables should be preprocessed/transformed into [numerical, categorical, and other data types](#) depending on the algorithm being used.

Data preprocessing

As a prerequisite to *feature engineering*, prepare the data from various streams to compose a schema from which it is easy to build features. Visualize the data first as a table of records. Each row in the table represents a training instance, and the columns represent *predictor* features (also called independent attributes or variables). Organize the data such that the last column(s) is the *target* (dependent variable). For each training instance, assign a *label* as the value of this column.

For temporal data, divide the duration of sensor data into time units. Each record should belong to a time unit for an asset, *and should offer distinct information*. Time units are defined based on business needs in multiples of seconds, minutes, hours, days, months, and so on. The time unit *does not have to be the same as the frequency of data collection*. If the frequency is high, the data may not show any significant difference from one unit to the other. For example, assume that ambient temperature was collected every 10 seconds. Using that same interval for training data only inflates the number of examples without providing any additional information. For this case, a better strategy would be to use average the data over 10 minutes, or an hour based on the business justification.

For static data,

- *Maintenance records*: Raw maintenance data has an asset identifier and timestamp with information on maintenance activities that have been performed at a given point in time. Transform maintenance activities into *categorical* columns, where each category descriptor uniquely maps to a specific maintenance action. The schema for maintenance records would include asset identifier, time, and maintenance action.
- *Failure records*: Failures or failure reasons can be recorded as specific error codes or failure events defined by specific business conditions. In cases where the equipment has multiple error codes, the domain expert should help identify the ones that are pertinent to the target variable. Use the remaining error codes or conditions to construct *predictor* features that correlate with these failures. The schema for failure records would include asset identifier, time, failure, or failure reason - if available.
- *Machine and operator metadata*: Merge the machine and operator data into one schema to associate an asset with its operator, along with their respective attributes. The schema for machine conditions would include asset identifier, asset features, operator identifier, and operator features.

Other data preprocessing steps include *handling missing values* and *normalization* of attribute values. A detailed discussion is beyond the scope of this guide - see the next section for some useful references.

With the above preprocessed data sources in place, the final transformation before feature engineering is to join the above tables based on the asset identifier and timestamp. The resulting table would have null values for the failure column when machine is in normal operation. These null values can be imputed by an indicator for normal operation. Use this failure column to create *labels for the predictive model*. For more information, see the section on [modeling techniques for predictive maintenance](#).

Feature engineering

Feature engineering is the first step prior to modeling the data. Its role in the data science process [is described here](#). A *feature* is a predictive attribute for the model - such as temperature, pressure, vibration, and so on. For PdM, feature engineering involves abstracting a machine's health over historical data collected over a sizable duration. In that sense, it is different from its peers such as remote monitoring, anomaly detection, and failure detection.

Time windows

Remote monitoring entails reporting the events that happen as of *points in time*. Anomaly detection models evaluate (score) incoming streams of data to flag anomalies as of points in time. Failure detection classifies failures to be of specific types as they occur points in time. In contrast, PdM involves predicting failures over a *future time period*, based on features that represent machine behavior over *historical time period*. For PdM, feature data from individual points of time are too noisy to be predictive. So the data for each feature needs to be *smoothed* by aggregating data points over time windows.

Lag features

The business requirements define how far the model has to predict into the future. In turn, this duration helps define 'how far back the model has to look' to make these predictions. This 'looking back' period is called the *lag*, and features engineered over this lag period are called *lag features*. This section discusses lag features that can be constructed from data sources with timestamps, and feature creation from static data sources. Lag features are typically *numerical* in nature.

IMPORTANT

The window size is determined via experimentation, and should be finalized with the help of a domain expert. The same caveat holds for the selection and definition of lag features, their aggregations, and the type of windows.

Rolling aggregates

For each record of an asset, a rolling window of size "W" is chosen as the number of units of time to compute the aggregates. Lag features are then computed using the W periods *before the date* of that record. In Figure 1, the blue lines show sensor values recorded for an asset for each unit of time. They denote a rolling average of feature values over a window of size W=3. The rolling average is computed over all records with timestamps in the range t₁ (in orange) to t₂ (in green). The value for W is typically in minutes or hours depending on the nature of the data. But for certain problems, picking a large W (say 12 months) can provide the whole history of an asset until the time of the record.

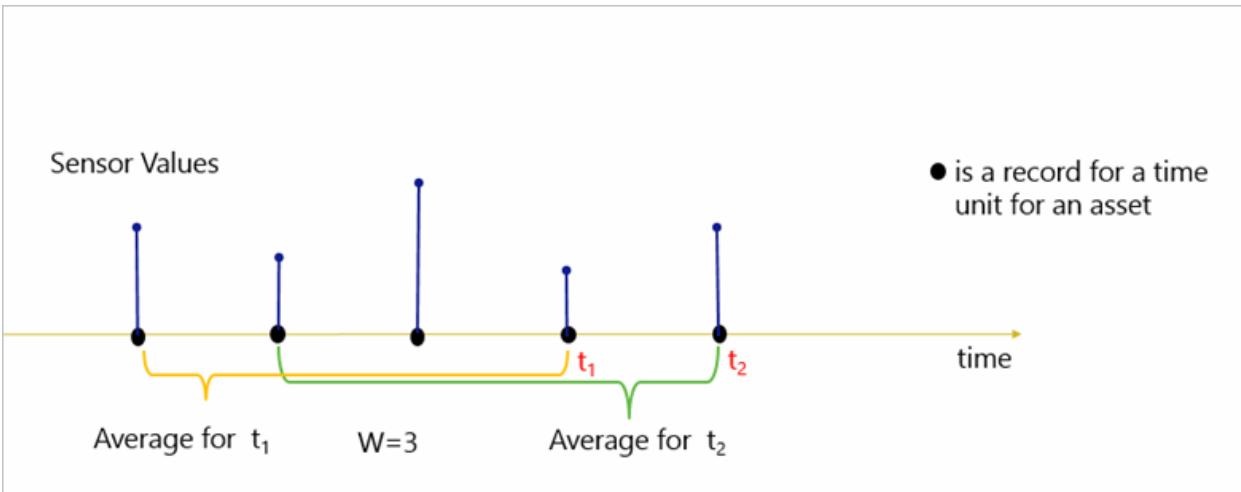


Figure 1. Rolling aggregate features

Examples of rolling aggregates over a time window are count, average, CUMESUM (cumulative sum) measures, min/max values. In addition, variance, standard deviation, and count of outliers beyond N standard deviations are often used. Examples of aggregates that may be applied for the [use cases](#) in this guide are listed below.

- *Flight delay*: count of error codes over the last day/week.
- *Aircraft engine part failure*: rolling means, standard deviation, and sum over the past day, week etc. This metric should be determined along with the business domain expert.
- *ATM failures*: rolling means, median, range, standard deviations, count of outliers beyond three standard deviations, upper and lower CUMESUM.
- *Subway train door failures*: Count of events over past day, week, two weeks etc.
- *Circuit breaker failures*: Failure counts over past week, year, three years etc.

Another useful technique in PdM is to capture trend changes, spikes, and level changes using algorithms that detect anomalies in data.

Tumbling aggregates

For each labeled record of an asset, a window of size $W-k$ is defined, where k is the number of windows of size W . Aggregates are then created over k *tumbling windows* $W-k, W-(k-1), \dots, W_2, W_1$ for the periods before a record's timestamp. k can be a small number to capture short-term effects, or a large number to capture long-term degradation patterns. (see Figure 2).

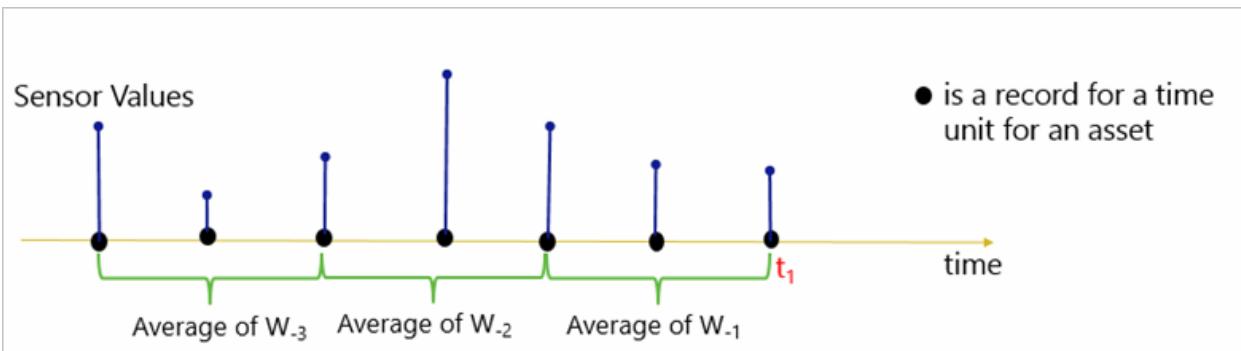


Figure 2. Tumbling aggregate features

For example, lag features for the wind turbines use case may be created with $W=1$ and $k=3$. They imply the lag for each of the past three months using top and bottom outliers.

Static features

Technical specifications of the equipment such as date of manufacture, model number, location, are some examples of static features. They are treated as *categorical* variables for modeling. Some examples for the circuit breaker use case are voltage, current, power capacity, transformer type, and power source. For wheel failures,

the type of tire wheels (alloy vs steel) is an example.

The data preparation efforts discussed so far should lead to the data being organized as shown below. Training, test, and validation data should have this logical schema (this example shows time in units of days).

ASSET ID	TIME	<FEATURE COLUMNS>	LABEL
A123	Day 1
A123	Day 2
...
B234	Day 1
B234	Day 2
...

The last step in feature engineering is the **labeling** of the target variable. This process is dependent on the modeling technique. In turn, the modeling technique depends on the business problem and nature of the available data. Labeling is discussed in the next section.

IMPORTANT

Data preparation and feature engineering are as important as modeling techniques to arrive at successful PdM solutions. The domain expert and the practitioner should invest significant time in arriving at the right features and data for the model. A small sample from many books on feature engineering are listed below:

- Pyle, D. Data Preparation for Data Mining (The Morgan Kaufmann Series in Data Management Systems), 1999
- Zheng, A., Casari, A. Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists, O'Reilly, 2018.
- Dong, G. Liu, H. (Editors), Feature Engineering for Machine Learning and Data Analytics (Chapman & Hall/CRC Data Mining and Knowledge Discovery Series), CRC Press, 2018.

Modeling techniques for predictive maintenance

This section discusses the main modeling techniques for PdM problems, along with their specific label construction methods. Notice that a single modeling technique can be used across different industries. The modeling technique is paired to the data science problem, rather than the context of the data at hand.

IMPORTANT

The choice of labels for the failure cases and the labeling strategy should be determined in consultation with the domain expert.

Binary classification

Binary classification is used to *predict the probability that a piece of equipment fails within a future time period* - called the *future horizon period X*. X is determined by the business problem and the data at hand, in consultation with the domain expert. Examples are:

- *minimum lead time* required to replace components, deploy maintenance resources, perform maintenance to avoid a problem that is likely to occur in that period.
- *minimum count of events* that can happen before a problem occurs.

In this technique, two types of training examples are identified. A positive example, *which indicates a failure*, with label = 1. A negative example, which indicates normal operations, with label = 0. The target variable, and hence the label values, are *categorical*. The model should identify each new example as likely to fail or work normally over the next X time units.

Label construction for binary classification

The question here is: "What is the probability that the asset will fail in the next X units of time?" To answer this question, label X records prior to the failure of an asset as "about to fail" (label = 1), and label all other records as being "normal" (label = 0). (see Figure 3).

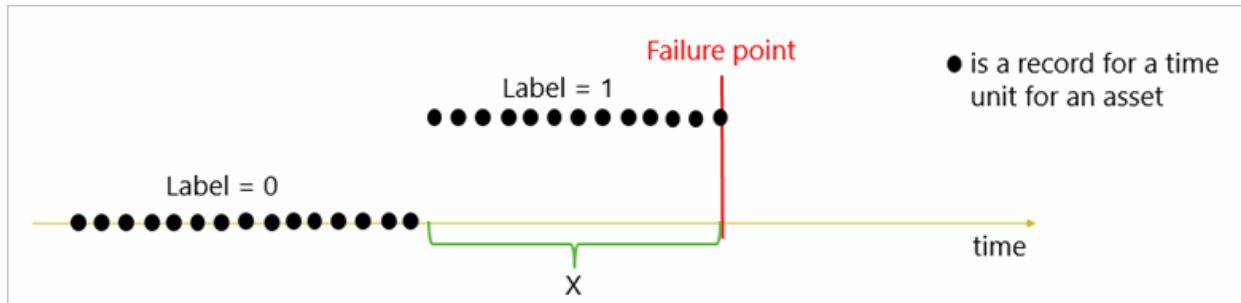


Figure 3. Labeling for binary classification

Examples of labeling strategy for some of the use cases are listed below.

- *Flight delays*: X may be chosen as one day, to predict delays in the next 24 hours. Then all flights that are within 24 hours before failures are labeled as 1.
- *ATM cash dispense failures*: A goal may be to determine failure probability of a transaction in the next one hour. In that case, all transactions that happened within the past hour of the failure are labeled as 1. To predict failure probability over the next N currency notes dispensed, all notes dispensed within the last N notes of a failure are labeled as 1.
- *Circuit breaker failures*: The goal may be to predict the next circuit breaker command failure. In that case, X is chosen to be one future command.
- *Train door failures*: X may be chosen as two days.
- *Wind turbine failures*: X may be chosen as two months.

Regression for predictive maintenance

Regression models are used to *compute the remaining useful life (RUL) of an asset*. RUL is defined as the amount of time that an asset is operational before the next failure occurs. Each training example is a record that belongs to a time unit nY for an asset, where n is the multiple. The model should calculate the RUL of each new example as a *continuous number*. This number denotes the period of time remaining before the failure.

Label construction for regression

The question here is: "What is the remaining useful life (RUL) of the equipment?" For each record prior to the failure, calculate the label to be the number of units of time remaining before the next failure. In this method, labels are continuous variables. (See Figure 4)

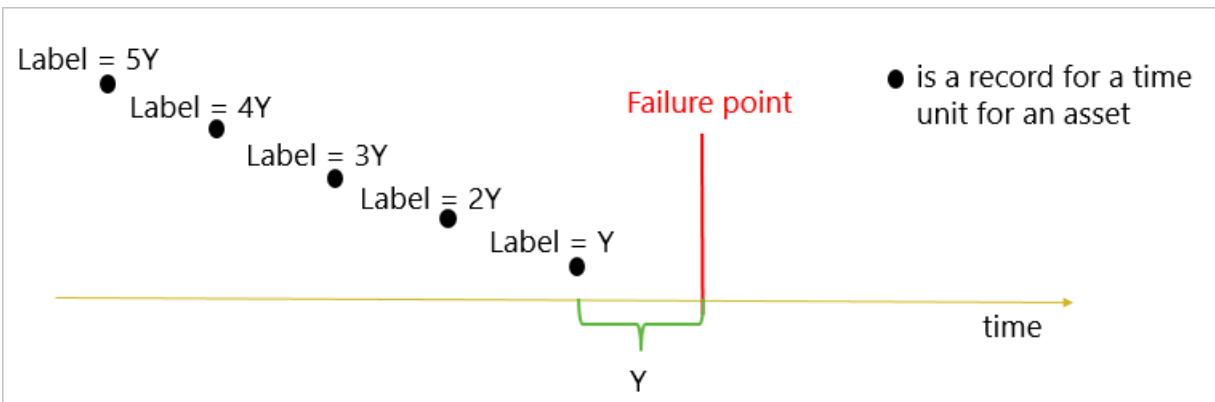


Figure 4. Labeling for regression

For regression, labeling is done with reference to a failure point. Its calculation is not possible without knowing how long the asset has survived before a failure. So in contrast to binary classification, assets without any failures in the data cannot be used for modeling. This issue is best addressed by another statistical technique called [Survival Analysis](#). But potential complications may arise when applying this technique to PdM use cases that involve time-varying data with frequent intervals. For more information on Survival Analysis, see [this one-pager](#).

Multi-class classification for predictive maintenance

Multi-class classification techniques can be used in PdM solutions for two scenarios:

- Predict *two future outcomes*: The first outcome is *a range of time to failure* for an asset. The asset is assigned to one of multiple possible periods of time. The second outcome is the likelihood of failure in a future period due to *one of the multiple root causes*. This prediction enables the maintenance crew to watch for symptoms and plan maintenance schedules.
- Predict *the most likely root cause* of a given failure. This outcome recommends the right set of maintenance actions to fix a failure. A ranked list of root causes and recommended repairs can help technicians prioritize their repair actions after a failure.

Label construction for multi-class classification

The question here is: "What is the probability that an asset will fail in the next nZ units of time where n is the number of periods?" To answer this question, label nZ records prior to the failure of an asset using buckets of time ($3Z$, $2Z$, Z). Label all other records as "normal" (label = 0). In this method, the target variable holds *categorical* values. (See Figure 5).

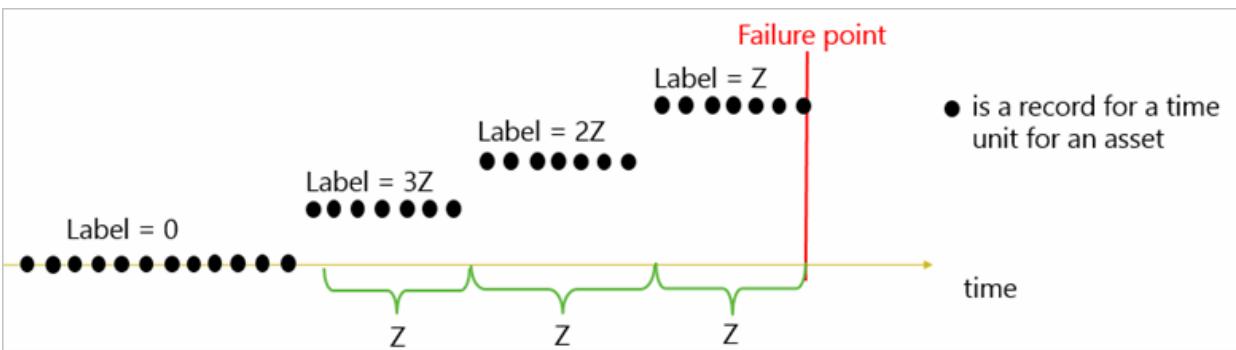


Figure 5. Labeling for multi-class classification for failure time prediction

The question here is: "What is the probability that the asset will fail in the next X units of time due to root cause/problem P_i ?" where i is the number of possible root causes. To answer this question, label X records prior to the failure of an asset as "about to fail due to root cause P_i " (label = P_i). Label all other records as being "normal" (label = 0). In this method also, labels are categorical (See Figure 6).

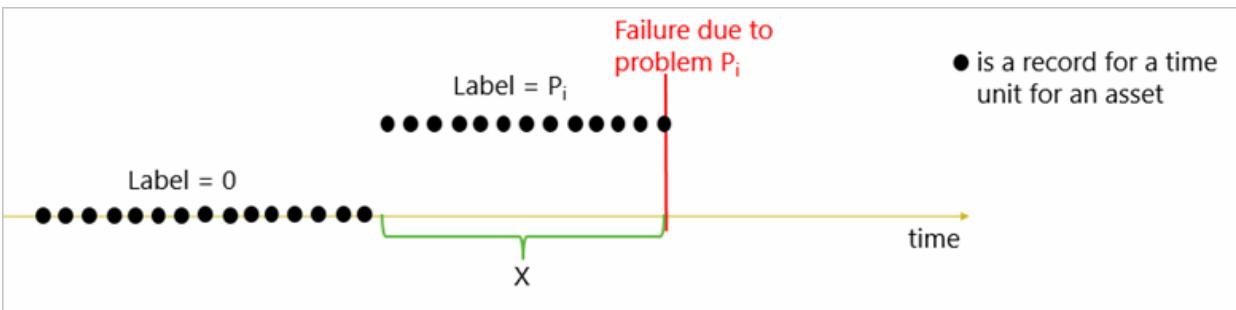


Figure 6. Labeling for multi-class classification for root cause prediction

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.

The question here is: "What maintenance actions do you recommend after a failure?" To answer this question, labeling *does not require a future horizon to be picked*, because the model is not predicting failure in the future. It is just predicting the most likely root cause *once the failure has already happened*.

Training, validation, and testing methods for predictive maintenance

The [Team Data Science Process](#) provides a full coverage of the model train-test-validate cycle. This section discusses aspects unique to PdM.

Cross validation

The goal of [cross validation](#) is to define a data set to "test" the model in the training phase. This data set is called the *validation set*. This technique helps limit problems like *overfitting* and gives an insight on how the model will generalize to an independent data set. That is, an unknown data set, which could be from a real problem. The training and testing routine for PdM needs to take into account the time varying aspects to better generalize on unseen future data.

Many machine learning algorithms depend on a number of hyperparameters that can change the model performance significantly. The optimal values of these hyperparameters are not computed automatically when training the model. They should be specified by the data scientist. There are several ways of finding good values of hyperparameters.

The most common one is *k-fold cross-validation* that splits the examples randomly into k folds. For each set of hyperparameters values, run the learning algorithm k times. At each iteration, use the examples in the current fold as a validation set, and the rest of the examples as a training set. Train the algorithm over training examples and compute the performance metrics over validation examples. At the end of this loop, compute the average of k performance metrics. For each set of hyperparameter values, choose the ones that have the best average performance. The task of choosing hyperparameters is often experimental in nature.

In PdM problems, data is recorded as a time series of events that come from several data sources. These records may be ordered according to the time of labeling. Hence, if the dataset is split *randomly* into training and validation set, *some of the training examples may be later in time than some of validation examples*. Future performance of hyperparameter values will be estimated based on some data that arrived *before* model was trained. These estimations might be overly optimistic, especially if the time-series is not stationary and evolves over time. As a result, the chosen hyperparameter values might be suboptimal.

The recommended way is to split the examples into training and validation set in a *time-dependent* manner, where all validation examples are later in time than all training examples. For each set of hyperparameter values, train the algorithm over the training data set. Measure the model's performance over the same validation set. Choose hyperparameter values that show the best performance. Hyperparameter values chosen by train/validation split result in better future model performance than with the values chosen randomly by cross-validation.

The final model can be generated by training a learning algorithm over entire training data using the best hyperparameter values.

Testing for model performance

Once a model is built, an estimate of its future performance on new data is required. A good estimate is the performance metric of hyperparameter values computed over the validation set, or an average performance metric computed from cross-validation. These estimations are often overly optimistic. The business might often have some additional guidelines on how they would like to test the model.

The recommended way for PdM is to split the examples into training, validation, and test data sets in a *time-dependent* manner. All test examples should be later in time than all the training and validation examples. After the split, generate the model and measure its performance as described earlier.

When time-series are stationary and easy to predict, both random and time-dependent approaches generate similar estimations of future performance. But when time-series are non-stationary, and/or hard to predict, the time-dependent approach will generate more realistic estimates of future performance.

Time-dependent split

This section describes best practices to implement time-dependent split. A time-dependent two-way split between training and test sets is described below.

Assume a stream of timestamped events such as measurements from various sensors. Define features and labels of training and test examples over time frames that contain multiple events. For example, for binary classification, create features based on past events, and create labels based on future events within "X" units of time in the future (see the sections on [feature engineering](#) and modeling techniques). Thus, the labeling time frame of an example comes later than the time frame of its features.

For time-dependent split, pick a *training cutoff time* T_c at which to train a model, with hyperparameters tuned using historical data up to T_c . To prevent leakage of future labels that are beyond T_c into the training data, choose the latest time to label training examples to be X units before T_c . In the example shown in Figure 7, each square represents a record in the data set where features and labels are computed as described above. The figure shows the records that should go into training and testing sets for $X=2$ and $W=3$:



Figure 7. Time-dependent split for binary classification

The green squares represent records belonging to the time units that can be used for training. Each training example is generated by considering the past three periods for feature generation, and two future periods for labeling before T_c . When any part of the two future periods is beyond T_c , exclude that example from the training data set because no visibility is assumed beyond T_c .

The black squares represent the records of the final labeled data set that should not be used in the training data set, given the above constraint. These records will also not be used in testing data, since they are before T_c . In addition, their labeling time frames partially depend on the training time frame, which is not ideal. Training and test data should have separate labeling time frames to prevent label information leakage.

The technique discussed so far allows for overlap between training and testing examples that have timestamps

near T_c . A solution to achieve greater separation is to exclude examples that are within W time units of T_c from the test set. But such an aggressive split depends on ample data availability.

Regression models used for predicting RUL are more severely affected by the leakage problem. Using the random split method leads to extreme over-fitting. For regression problems, the split should be such that the records belonging to assets with failures before T_c go into the training set. Records of assets that have failures after the cutoff go into the test set.

Another best practice for splitting data for training and testing is to use a split by asset ID. The split should be such that none of the assets used in the training set are used in testing the model performance. Using this approach, a model has a better chance of providing more realistic results with new assets.

Handling imbalanced data

In classification problems, if there are more examples of one class than of the others, the data set is said to be *imbalanced*. Ideally, enough representatives of each class in the training data are preferred to enable differentiation between different classes. If one class is less than 10% of the data, the data is deemed to be imbalanced. The underrepresented class is called a *minority class*.

Many PdM problems face such imbalanced datasets, where one class is severely underrepresented compared to the other class, or classes. In some situations, the minority class may constitute only 0.001% of the total data points. Class imbalance is not unique to PdM. Other domains where failures and anomalies are rare occurrences face a similar problem, for example, fraud detection and network intrusion. These failures make up the minority class examples.

With class imbalance in data, performance of most standard learning algorithms is compromised, since they aim to minimize the overall error rate. For a data set with 99% negative and 1% positive examples, a model can be shown to have 99% accuracy by labeling all instances as negative. But the model will mis-classify all positive examples; so even if its accuracy is high, the algorithm is not a useful one. Consequently, conventional evaluation metrics such as *overall accuracy on error rate* are insufficient for imbalanced learning. When faced with imbalanced datasets, other metrics are used for model evaluation:

- Precision
- Recall
- F1 scores
- Cost adjusted ROC (receiver operating characteristics)

For more information about these metrics, see [model evaluation](#).

However, there are some methods that help remedy class imbalance problem. The two major ones are *sampling techniques* and *cost sensitive learning*.

Sampling methods

Imbalanced learning involves the use of sampling methods to modify the training data set to a balanced data set. Sampling methods are not to be applied to the test set. Although there are several sampling techniques, most straight forward ones are *random oversampling* and *under sampling*.

Random oversampling involves selecting a random sample from minority class, replicating these examples, and adding them to training data set. Consequently, the number of examples in minority class is increased, and eventually balance the number of examples of different classes. A drawback of oversampling is that multiple instances of certain examples can cause the classifier to become too specific, leading to over-fitting. The model may show high training accuracy, but its performance on unseen test data may be suboptimal.

Conversely, *random under sampling* is selecting a random sample from a 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 over-sampled and majority class is under-sampled at the same time is another viable approach.

There are many sophisticated sampling techniques. The technique chosen depends on the data properties and results of iterative experiments by the data scientist.

Cost sensitive learning

In PdM, failures that constitute the minority class are of more interest than normal examples. So the focus is mainly on the algorithm's performance on failures. Incorrectly predicting a positive class as a negative class can cost more than vice-versa. This situation is commonly referred as unequal loss or asymmetric cost of mis-classifying elements to different classes. The ideal classifier should deliver high prediction accuracy over the minority class, without compromising on the accuracy for the majority class.

There are multiple ways to achieve this balance. To mitigate the problem of unequal loss, assign a high cost to mis-classification of the minority class, and try to minimize the overall cost. Algorithms like *SVMs (Support Vector Machines)* adopt this method inherently, by allowing cost of positive and negative examples to be specified during training. Similarly, boosting methods such as *boosted decision trees* usually show good performance with imbalanced data.

Model evaluation

Mis-classification is a significant problem for PdM scenarios where the cost of false alarms to the business is high. For instance, a decision to ground an aircraft based on an incorrect prediction of engine failure can disrupt schedules and travel plans. Taking a machine offline from an assembly line can lead to loss of revenue. So model evaluation with the right performance metrics against new test data is critical.

Typical performance metrics used to evaluate PdM models are discussed below:

- **Accuracy** is the most popular metric used for describing a classifier's performance. But accuracy is sensitive to data distributions, and is an ineffective measure for scenarios with imbalanced data sets. Other metrics are used instead. Tools like **confusion matrix** are used to compute and reason about accuracy of the model.
- **Precision** of PdM models relate to the rate of false alarms. Lower precision of the model generally corresponds to a higher rate of false alarms.
- **Recall** rate denotes how many of the failures in the test set were correctly identified by the model. Higher recall rates mean the model is successful in identifying the true failures.
- **F1 score** is the harmonic average of precision and recall, with its value ranging between 0 (worst) to 1 (best).

For binary classification,

- **Receiver operating curves (ROC)** is also a popular metric. In ROC curves, model performance is interpreted based on one fixed operating point on the ROC.
- But for PdM problems, *decile tables* and *lift charts* are more informative. They focus only on the positive class (failures), and provide a more complex picture of the algorithm performance than ROC curves.
 - *Decile tables* are created using test examples in a descending order of failure probabilities. The ordered samples are then grouped into deciles (10% of the samples with highest probability, then 20%, 30%, and so on). The ratio (true positive rate)/(random baseline) for each decile helps estimate the algorithm performance at each decile. The random baseline takes on values 0.1, 0.2, and so on.
 - *Lift charts* plot the decile true positive rate versus random true positive rate for all deciles. The first deciles are usually the focus of results, since they show the largest gains. First deciles can also be seen as representative for "at risk", when used for PdM.

Model operationalization for predictive maintenance

The benefit the data science exercise is realized only when the trained model is made operational. That is, the model must be deployed into the business systems to make predictions based on new, previously unseen, data. The new data must exactly conform to the *model signature* of the trained model in two ways:

- all the features must be present in every logical instance (say a row in a table) of the new data.

- the new data must be pre-processed, and each of the features engineered, in exactly the same way as the training data.

The above process is stated in many ways in academic and industry literature. But all the following statements mean the same thing:

- Score new data* using the model
- Apply the model* to new data
- Operationalize* the model
- Deploy* the model
- Run the model* against new data

As stated earlier, model operationalization for PdM is different from its peers. Scenarios involving anomaly detection and failure detection typically implement *online scoring* (also called *real time scoring*). Here, the model *scores* each incoming record, and returns a prediction. For anomaly detection, the prediction is an indication that an anomaly occurred (Example: One-class SVM). For failure detection, it would be the type or class of failure.

In contrast, PdM involves *batch scoring*. To conform to the model signature, the features in the new data must be engineered in the same manner as the training data. For the large datasets that is typical for new data, features are aggregated over time windows and scored in batch. Batch scoring is typically done in distributed systems like [Spark](#) or [Azure Batch](#). There are a couple of alternatives - both suboptimal:

- Streaming data engines support aggregation over windows in memory. So it could be argued that they support online scoring. But these systems are suitable for dense data in narrow windows of time, or sparse elements over wider windows. They may not scale well for the dense data over wider time windows, as seen in PdM scenarios.
- If batch scoring is not available, the solution is to adapt online scoring to handle new data in small batches at a time.

Solution templates for predictive maintenance

The final section of this guide provides a list of PdM solution templates, tutorials, and experiments implemented in Azure. These PdM applications can be deployed into an Azure subscription within minutes in some cases. They can be used as proof-of-concept demos, sandboxes to experiment with alternatives, or accelerators for actual production implementations. These templates are located in the [Azure AI Gallery](#) or [Azure GitHub](#). These different samples will be rolled into this solution template over time.

#	TITLE	DESCRIPTION
2	Azure Predictive Maintenance Solution Template	An open-source solution template that demonstrates Azure ML modeling and a complete Azure infrastructure capable of supporting Predictive Maintenance scenarios in the context of IoT remote monitoring.
3	Deep Learning for Predictive Maintenance	Azure Notebook with a demo solution of using LSTM (Long Short-Term Memory) networks (a class of Recurrent Neural Networks) for Predictive Maintenance, with a blog post on this sample .

#	TITLE	DESCRIPTION
4	Azure Predictive Maintenance for Aerospace	One of the first PdM solution templates based on Azure ML v1.0 for aircraft maintenance. This guide originated from this project.
5	Azure AI Toolkit for IoT Edge	AI in the IoT Edge using TensorFlow; toolkit packages deep learning models in Azure IoT Edge-compatible Docker containers and expose those models as REST APIs.
6	Azure IoT Predictive Maintenance	Azure IoT Suite PCS - Preconfigured Solution. Aircraft maintenance PdM template with IoT Suite. Another document and walkthrough related to the same project.
7	Predictive Maintenance template using SQL Server R Services	Demo of remaining useful life scenario based on R services.
8	Predictive Maintenance Modeling Guide	Aircraft maintenance dataset feature engineered using R with experiments and datasets and Azure notebook and experiments in AzureML v1.0

Training resources for predictive maintenance

Microsoft Azure offers learning paths for the foundational concepts behind PdM techniques, besides content and training on general AI concepts and practice.

TRAINING RESOURCE	AVAILABILITY
Learning Path for PdM using Trees and Random Forest	Public
Learning Path for PdM using Deep Learning	Public
AI Developer on Azure	Public
Microsoft AI School	Public
Azure AI Learning from GitHub	Public
LinkedIn Learning	Public
Microsoft AI YouTube Webinars	Public
Microsoft AI Show	Public
LearnAI@MS	Partners
Microsoft Partner Network	Partners

In addition, free MOOCs (massive open online courses) on AI are offered online by academic institutions like

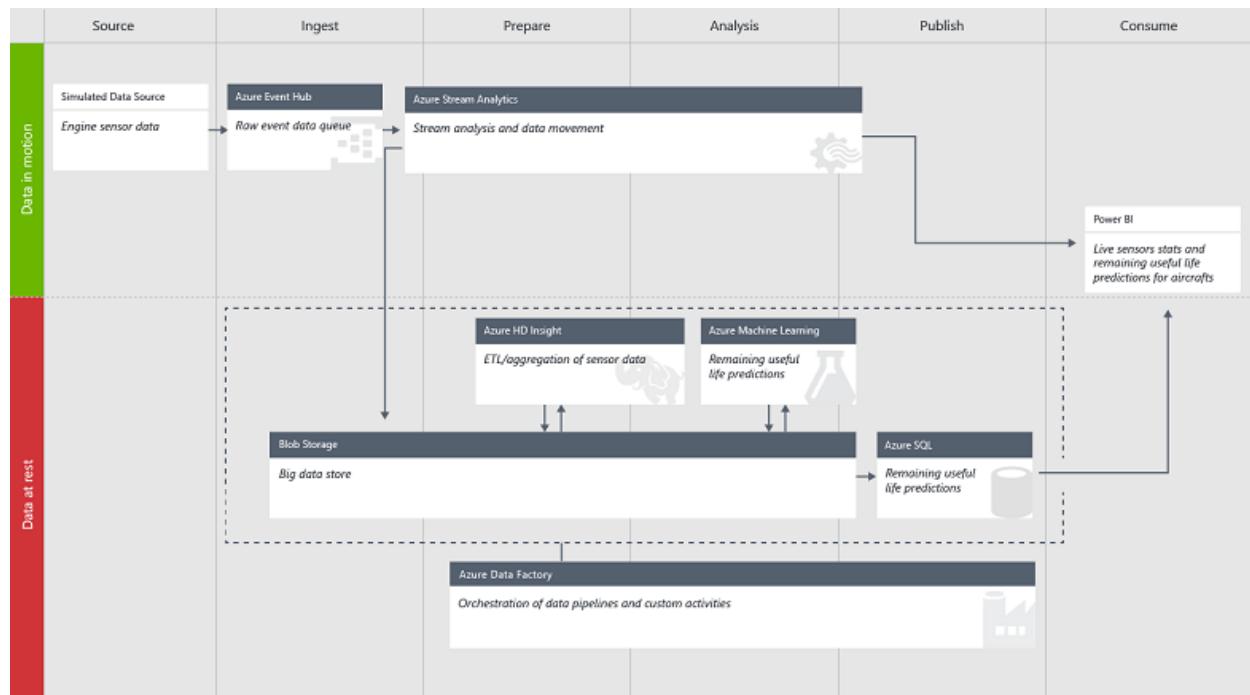
Stanford and MIT, and other educational companies.

Architecture of the Solution Template for predictive maintenance in aerospace

3/11/2020 • 2 minutes to read • [Edit Online](#)

The diagram below provides an architectural overview of the [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 Solution Template for predictive maintenance in aerospace

3/5/2021 • 16 minutes 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](#).

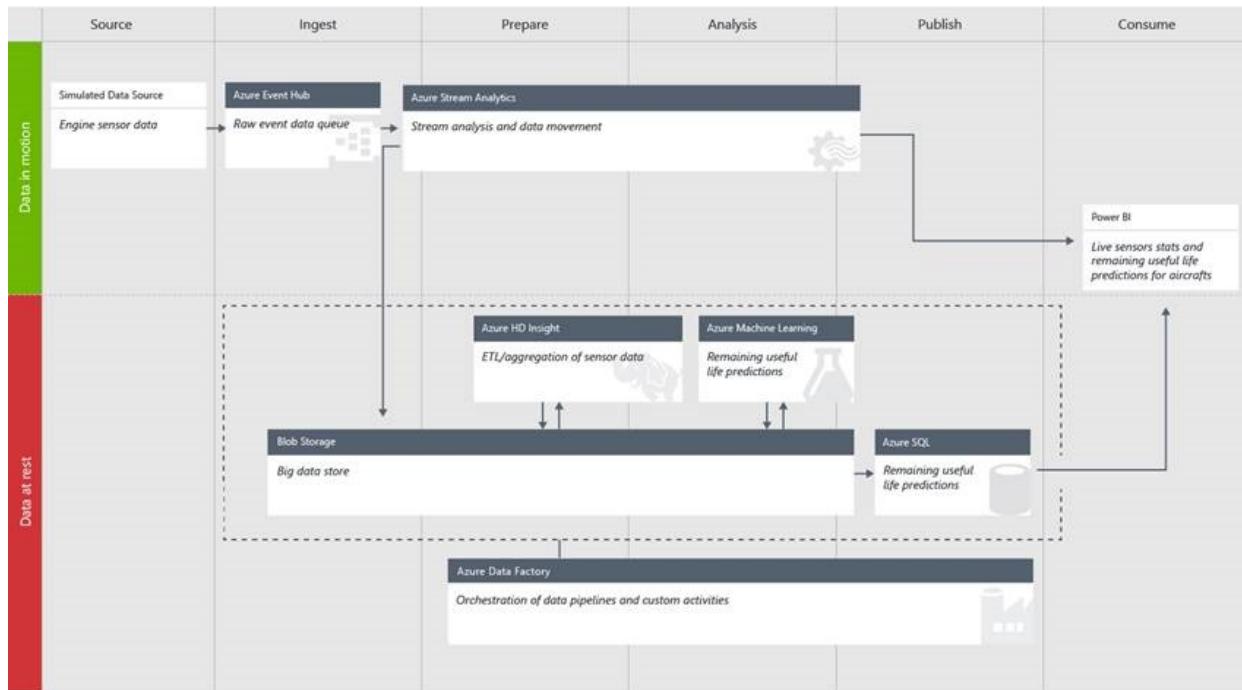
Solution templates are designed to accelerate the process of building an E2E demo. A deployed template provisions your subscription with necessary components and then builds the relationships between them. It also seeds the data pipeline with sample data from a data generator application, which you download 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.

Overview



When you deploy the solution, it activates Azure services 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 downloaded desktop application that you 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 resource.

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, 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 resource 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

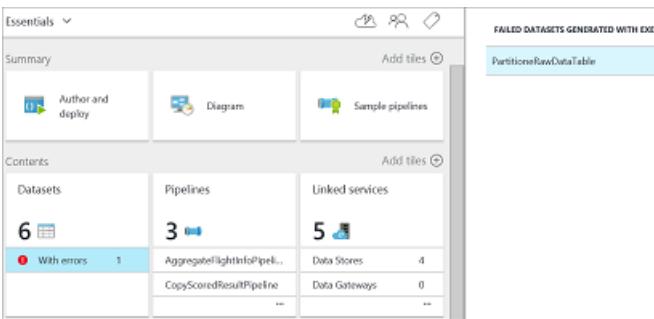
Information about Azure Stream Analytics query construction can be found in the [Stream Analytics Query Reference](#) on MSDN.

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.

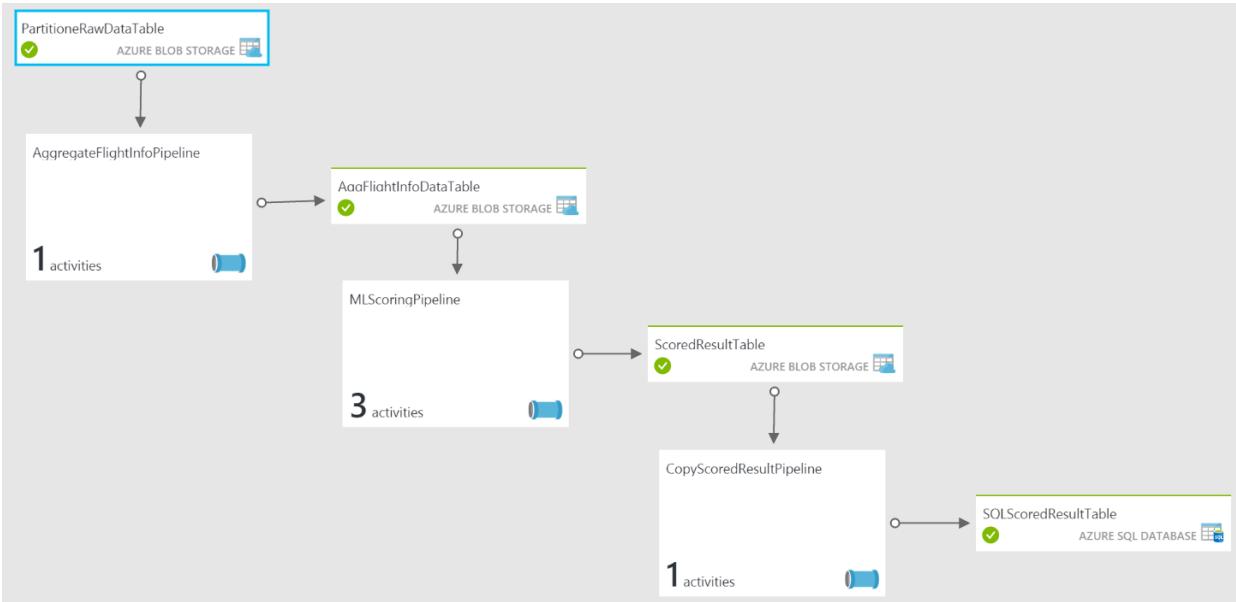
The query in the second Stream Analytics job **maintenancesa02asablob** simply outputs all [Event Hub](#) events to [Azure Storage](#) and hence requires no alteration regardless of your data format as the full event information is streamed to storage.

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.



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.

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.

- 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.
- 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 **pmaintenancedb** 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.

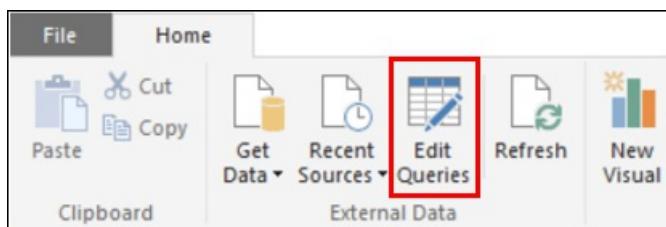
1. Get the database credentials.

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 that 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, 'pmaintenancedb'), and the **database server name** is listed under the Server name property and should look similar to **YourSolutionName.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'.

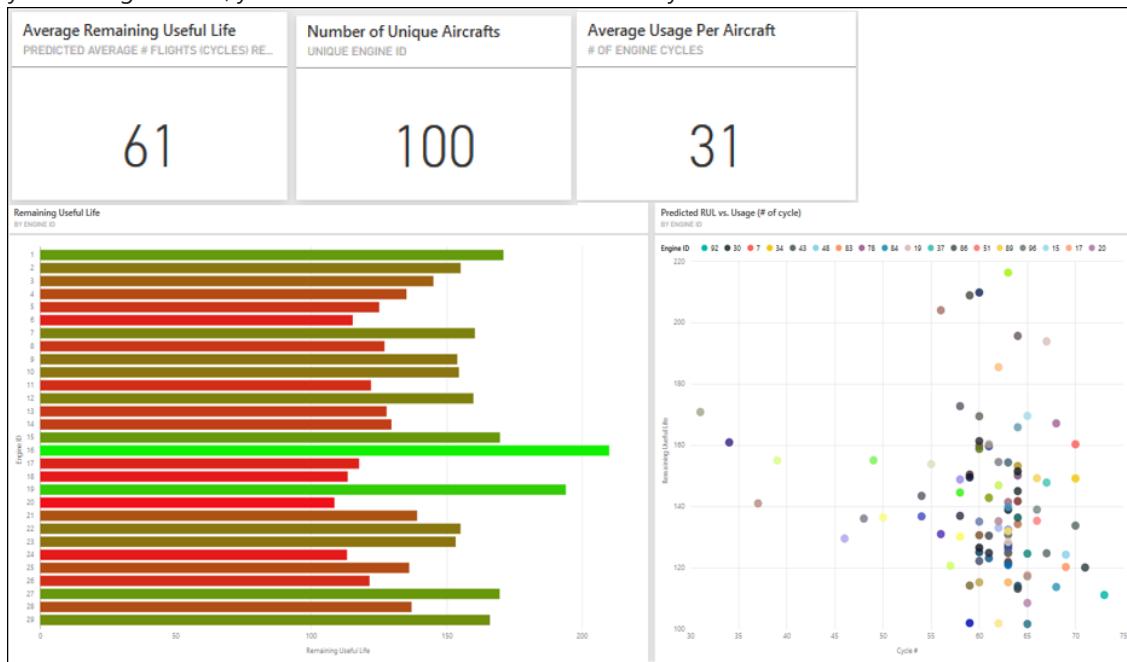


- You'll see two tables, **RemainingUsefullLife** and **PMResult**. Select the first table and click  next to 'Source' under 'APPLIED STEPS' on the right 'Query Settings' panel. Ignore any warning messages that appear.
- In the pop out window, replace 'Server' and 'Database' with your own server and database names, and then click 'OK'. For server name, make sure you specify the port 1433 (**YourSolutionName.database.windows.net, 1433**). Leave the Database field as **pmaintenancedb**. 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' (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](#). This step needs a Power BI account (or a work or school account).

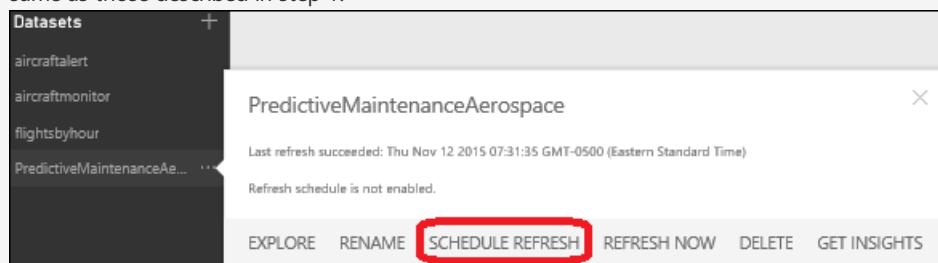
- Click 'Publish' and few seconds later a window appears displaying "Publishing to Power BI Success!" with a green check mark. Click the link below "Open PredictiveMaintenanceAerospace.pbix in Power BI". To find detailed instructions, see [Publish from Power BI Desktop](#).
- To create a new dashboard: 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](#).

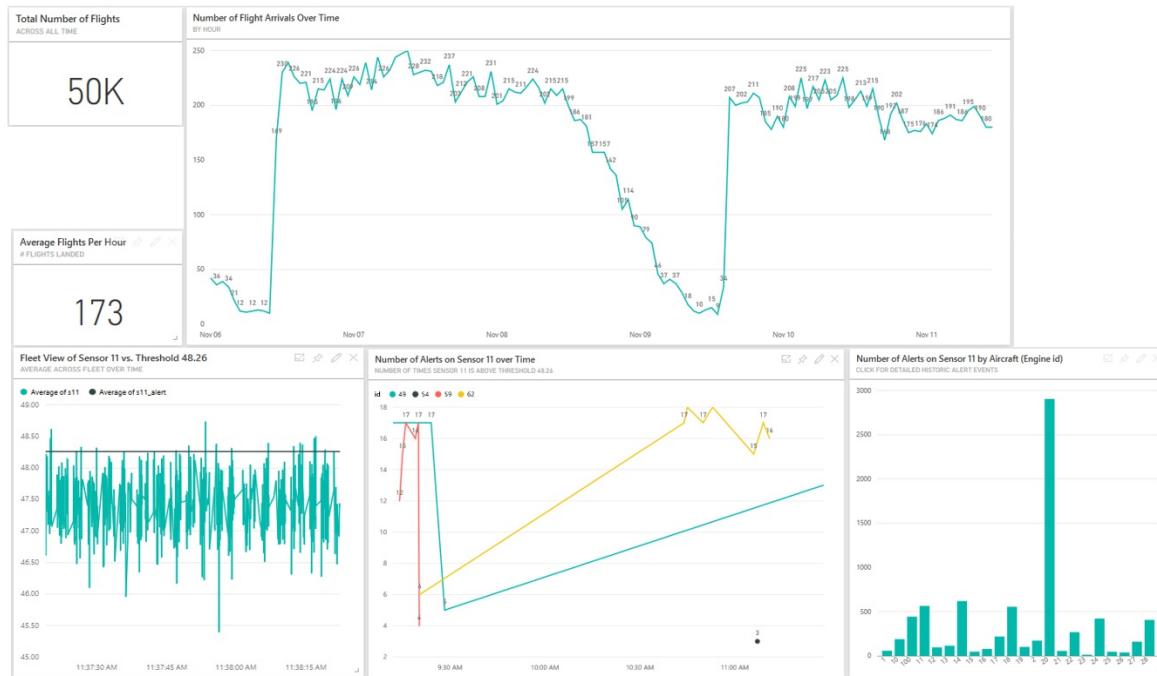
1. Add Power BI output in Azure Stream Analytics (ASA).

- 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 that 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 \(desktop\)](#)