Tutorial 1: Classify Iris - Preparing the data

6/13/2018 • 7 minutes to read • Edit Online

Azure Machine Learning service (preview) is an integrated, end-to-end data science and advanced analytics solution for professional data scientists to prepare data, develop experiments, and deploy models at cloud scale.

This tutorial is **part one of a three-part series**. In this tutorial, you walk through the basics of Azure Machine Learning services (preview) and learn how to:

- Create a project in Azure Machine Learning Workbench
- Create a data preparation package
- Generate Python/PySpark code to invoke a data preparation package

This tutorial uses the timeless Iris flower data set.

Prerequisites

If you don't have an Azure subscription, create a free account before you begin.

To complete this tutorial, you must have:

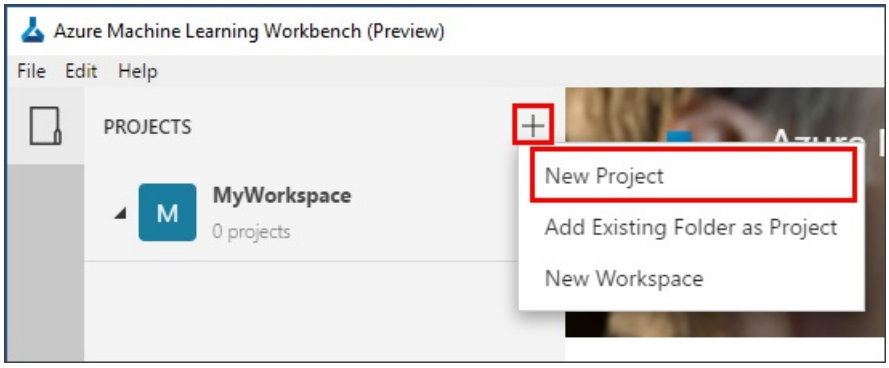
- An Azure Machine Learning Experimentation account
- Azure Machine Learning Workbench installed

If you don't have these prerequisites already, follow the steps in the Quickstart: Install and start article to set up your accounts and install the Azure Machine Learning Workbench application.

Create a new project in Workbench

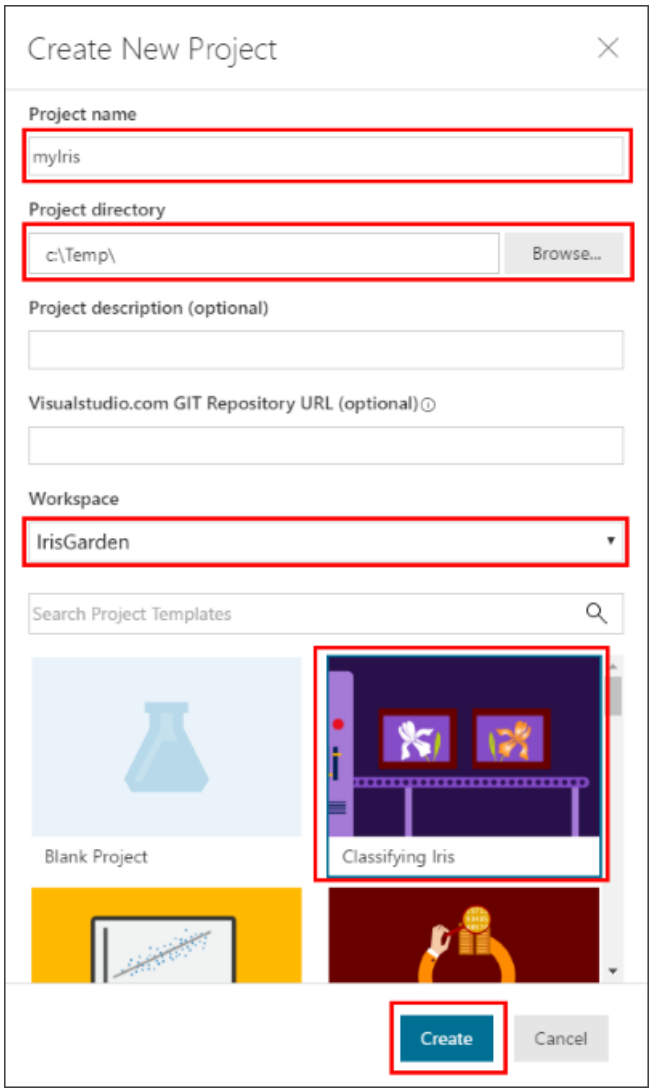
If you followed the steps in the Quickstart: Install and start article you should already have this project and can skip to the next section.

1. Open the Azure Machine Learning Workbench app, and log in if needed.
   - On Windows, launch it using the **Machine Learning Workbench** desktop shortcut.
   - On macOS, select **Azure ML Workbench** in Launchpad.
2. Select the plus sign (+) in the **PROJECTS** pane and choose **New Project**.

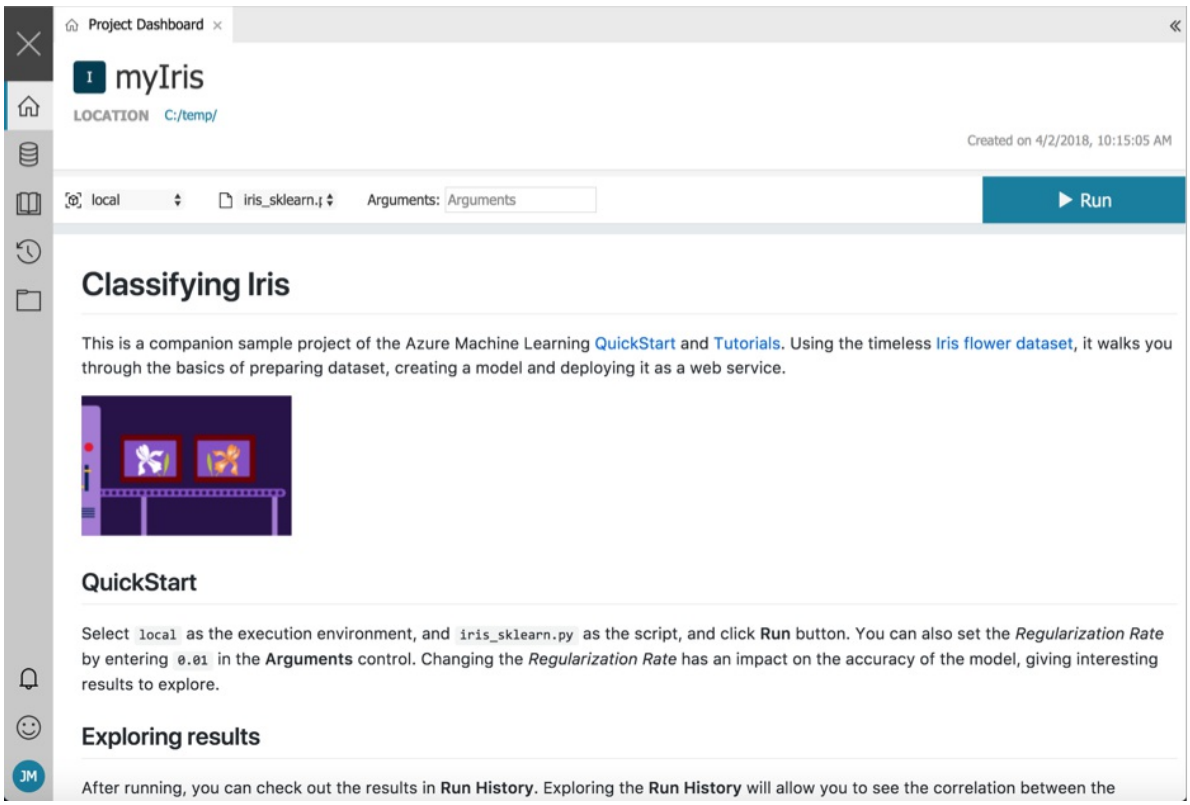


3. Fill out of the form fields and select the **Create** button to create a new project in the Workbench.

FIELD	SUGGESTED VALUE FOR TUTORIAL	DESCRIPTION
Project name	myIris	Enter a unique name that identifies your account. You can use your own name, or a departmental or project name that best identifies the experiment. The name should be 2 to 32 characters. It should include only alphanumeric characters and the dash (-) character.
Project directory	c:\Temp\	Specify the directory in which the project is created.
Project description	*leave blank*	Optional field useful for describing the projects.
Visualstudio.com GIT Repository URL	*leave blank*	Optional field. You can associate a project with a Git repository on Visual Studio Team Services for source control and collaboration. Learn how to set that up.
Selected workspace	IrisGarden (if it exists)	Choose a workspace that you have created for your Experimentation account in the Azure portal. If you followed the Quickstart, you should have a workspace by the name IrisGarden. If not, select the one you created when you created your Experimentation account or any other you want to use.
Project template	Classifying Iris	Templates contain scripts and data you can use to explore the product. This template contains the scripts and data you need for this quickstart and other tutorials in this documentation site.



A new project is created and the project dashboard opens with that project. At this point, you can explore the project home page, data sources, notebooks, and source code files.



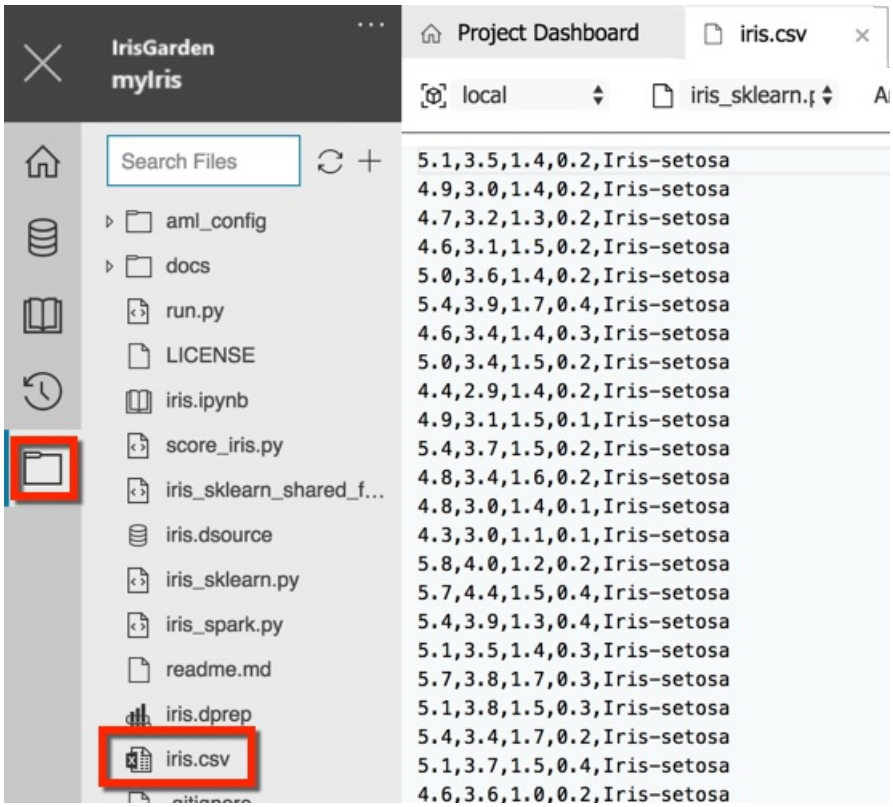
Create a data preparation package

Next, you can explore and start preparing the data in Azure Machine Learning Workbench. Each transformation you perform in Workbench is stored in a JSON format in a local data preparation package (*.dprep file). This data preparation package is the primary container for your data preparation work in Workbench.

This data preparation package can be handed off later to a runtime, such as local-C#/CoreCLR, Scala/Spark, or Scala/HDI.

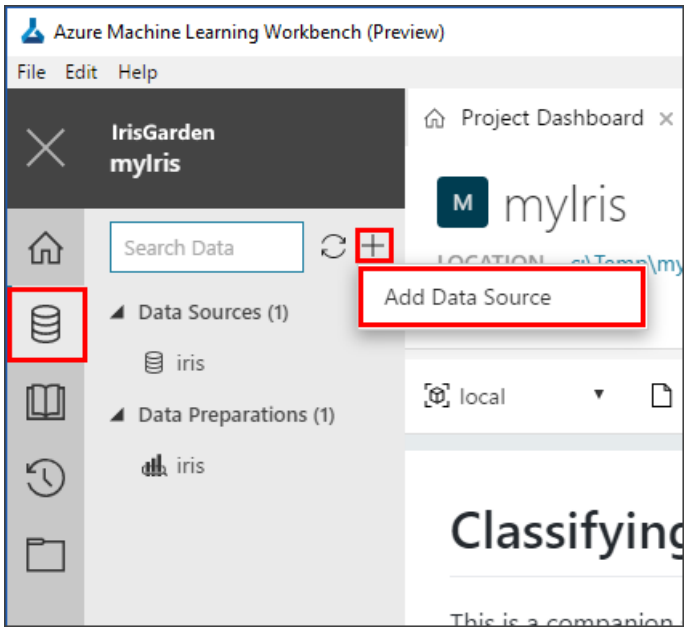
1. Select the folder icon to open the Files view, then select **iris.csv** to open that file.

   This file contains a table with 5 columns and 50 rows. Four columns are numerical feature columns. The fifth column is a string target column. None of the columns have header names.

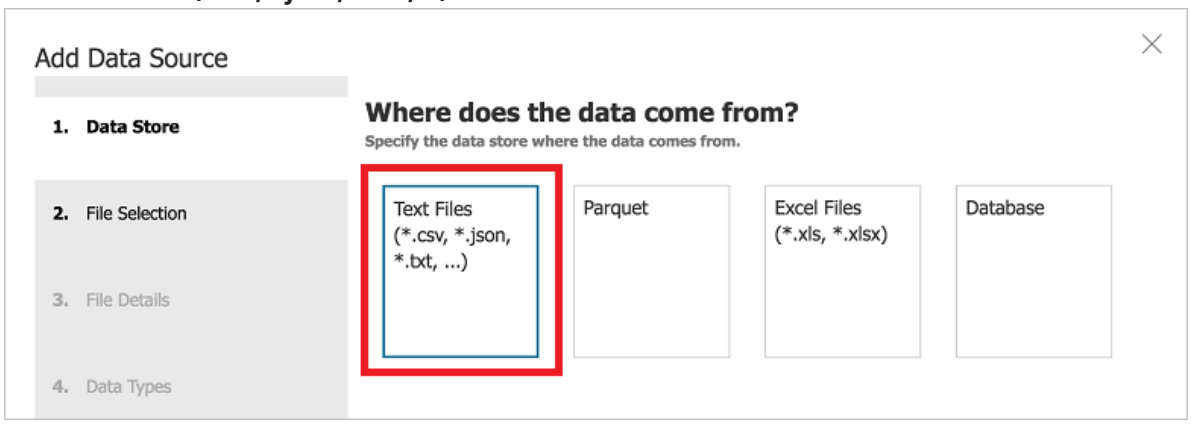


> **NOTE**
>
> Do not include data files in your project folder, particularly when the file size is large. Because the **iris.csv** data file is tiny, it was included in this template for demonstration purposes. For more information, see How to read and write large data files.

2. In the **Data view**, select the plus sign (+) to add a new data source. The **Add Data Source** page opens.

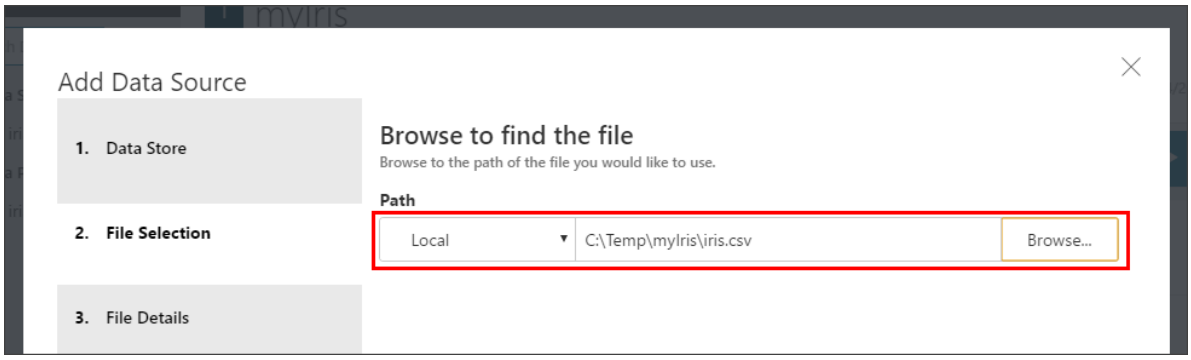


3. Select **Text Files(*.csv, *.json, *.txt., ...)** and click **Next**.



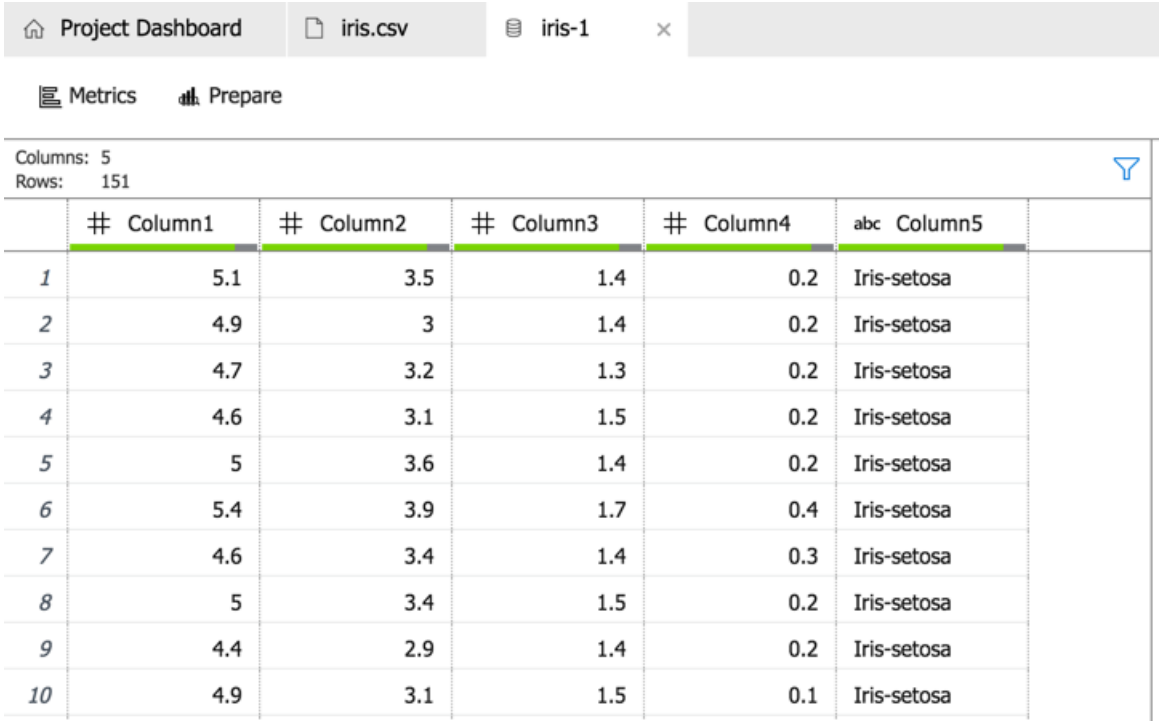
4. Browse to the file **iris.csv**, and click **Finish**. This will use default values for parameters such as the separator and data types.

> **IMPORTANT**
>
> Make sure you select the **iris.csv** file from within the current project directory for this exercise. Otherwise, later steps might fail.



5. A new file named **iris-1.dsource** is created. The file is named uniquely with "-1" because the sample project already comes with an unnumbered **iris.dsource** file.

   The file opens, and the data is shown. A series of column headers, from **Column1** to **Column5**, is automatically added to this data set. Scroll to the bottom and notice that the last row of the data set is empty. The row is empty because there is an extra line break in the CSV file.

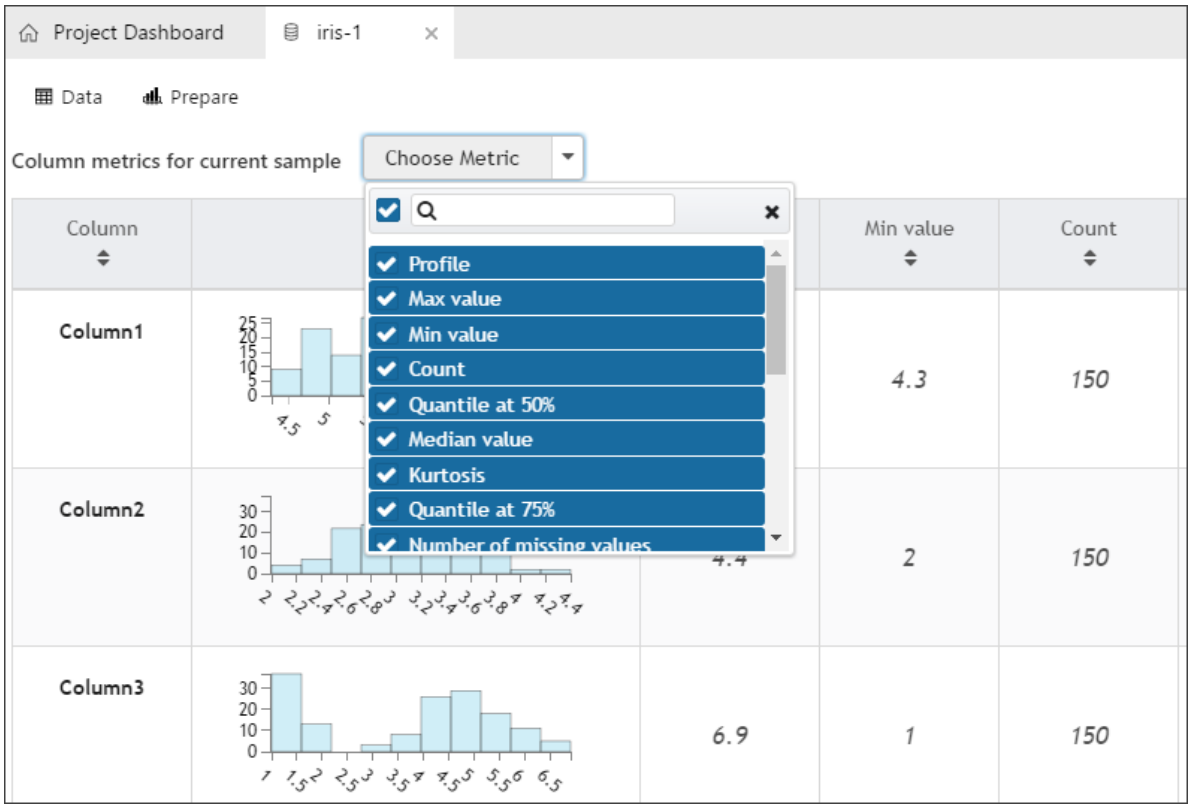


	Column1	Column2	Column3	Column4	Column5
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	5	3.6	1.4	0.2	Iris-setosa
6	5.4	3.9	1.7	0.4	Iris-setosa
7	4.6	3.4	1.4	0.3	Iris-setosa
8	5	3.4	1.5	0.2	Iris-setosa
9	4.4	2.9	1.4	0.2	Iris-setosa
10	4.9	3.1	1.5	0.1	Iris-setosa

6. Select the **Metrics** button. Histograms are generated and displayed.

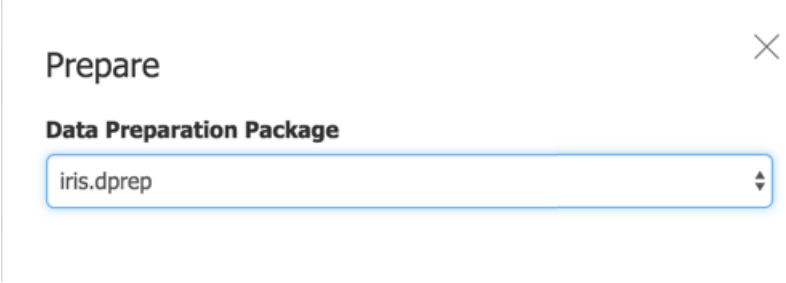
   You can switch back to the data view by selecting the **Data** button.

7. Observe the histograms. A complete set of statistics has been calculated for each column.

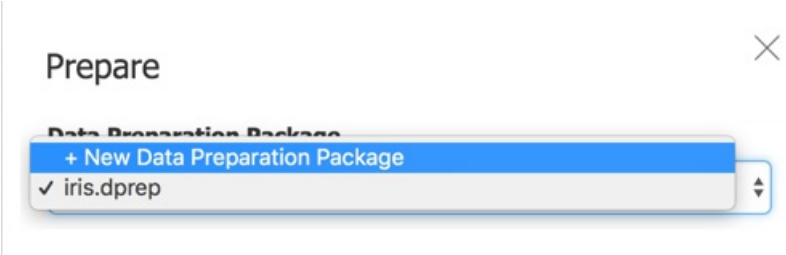


8. Begin creating a data preparation package by selecting the **Prepare** button. The **Prepare** dialog box opens.

   The sample project contains a **iris.dprep** data preparation file that is selected by default.

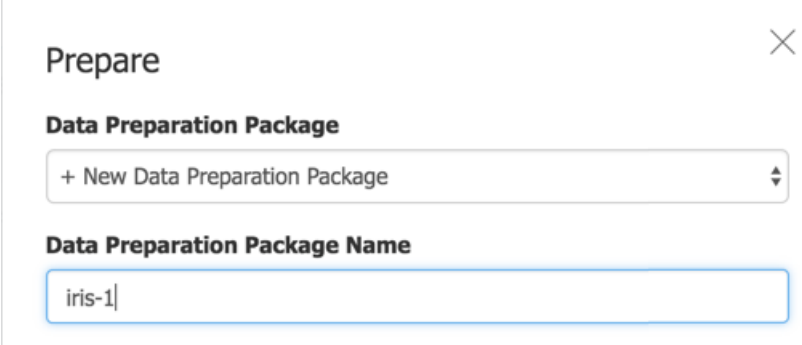


9. Create a new data preparation package by selecting **+ New Data Preparation Package** from the drop-down menu.



10. Enter a new value for the package name (use **iris-1**) and then select **OK**.

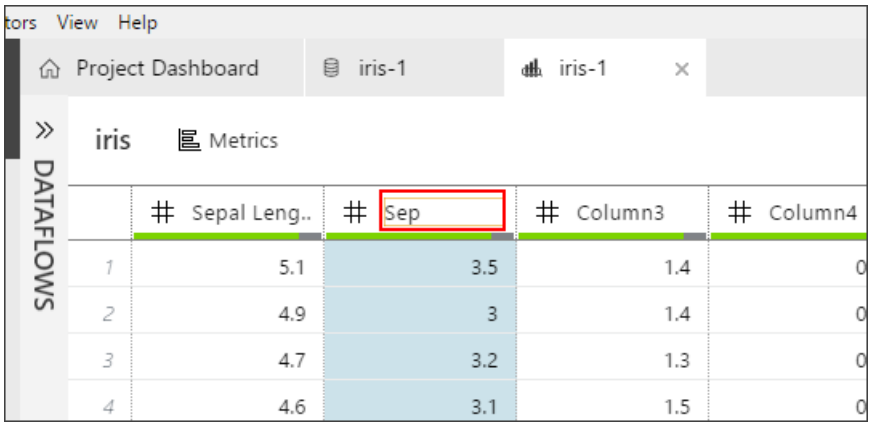
    A new data preparation package named **iris-1.dprep** is created and opened in the data preparation editor.



Now, let's do some basic data preparation.

11. Select each column header to make the header text editable. Then, rename each column as follows:

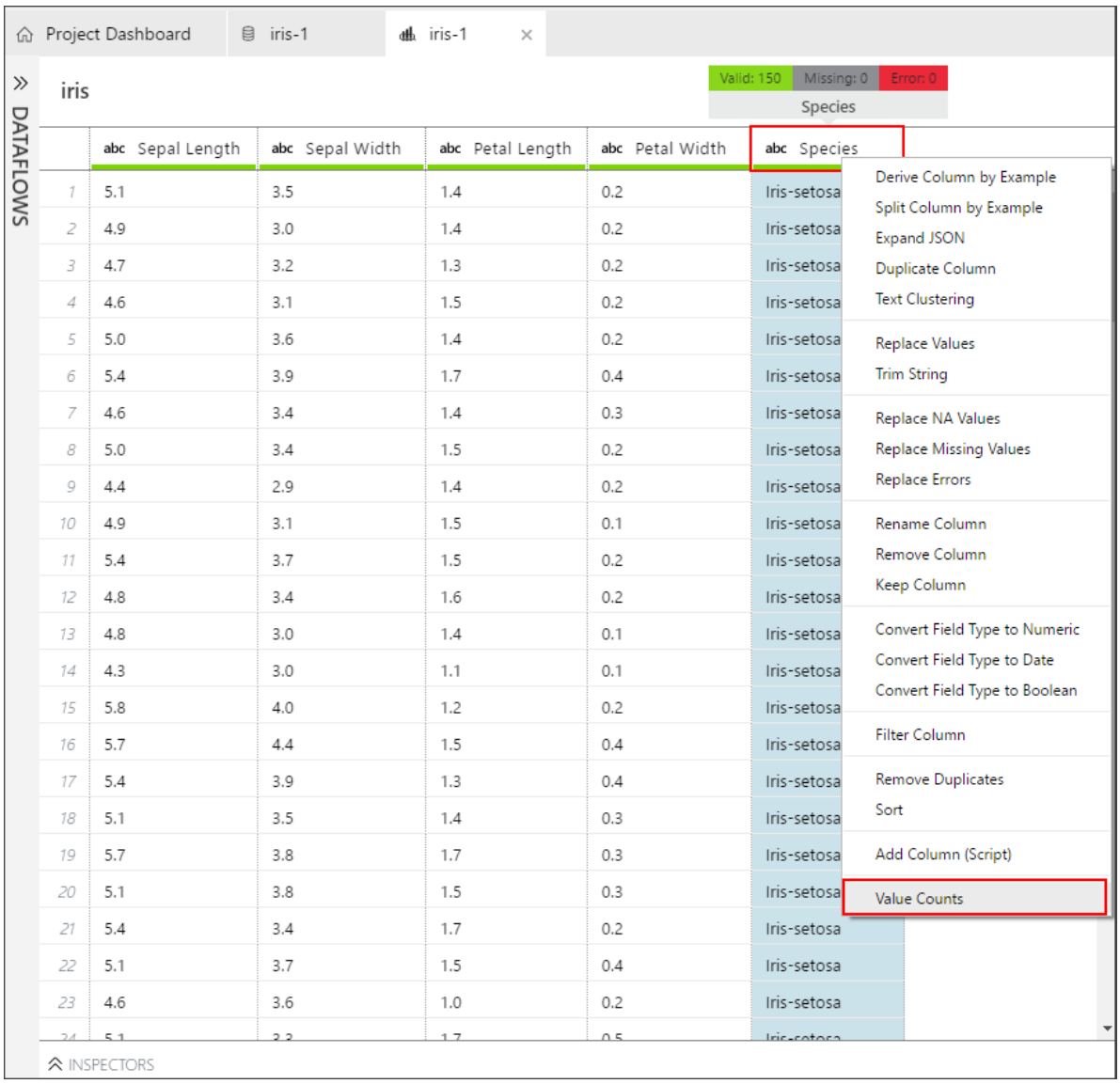
    In order, enter **Sepal Length**, **Sepal Width**, **Petal Length**, **Petal Width**, and **Species** for the five columns respectively.



12. Count distinct values:

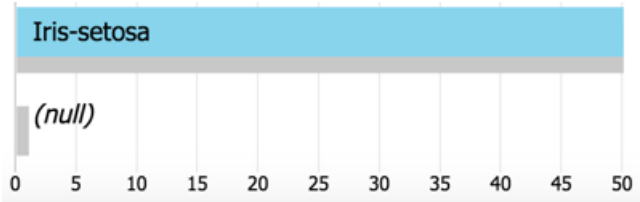
    a. Select the **Species** column
    b. Right-click to select it.
    c. Select **Value Counts** from the drop-down menu.

The **Inspectors** pane opens below the data. A histogram with four bars appears. The target column has four distinct values: **Iris-virginica**, **Iris-versicolor**, **Iris-setosa**, and a **(null)** value.

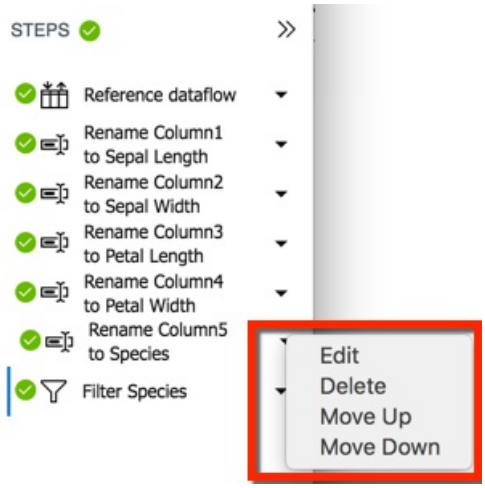


13. To filter out the null values, select the "(null)" bar and then select the minus sign (**-**).

    Then, the (null) row turns gray to indicate that it was filtered out.



14. Take notice of the individual data preparation steps that are detailed in the **STEPS** pane. As you renamed the columns and filtered the null value rows, each action was recorded as a data preparation step. You can edit individual steps to adjust their settings, reorder the steps, and remove steps.



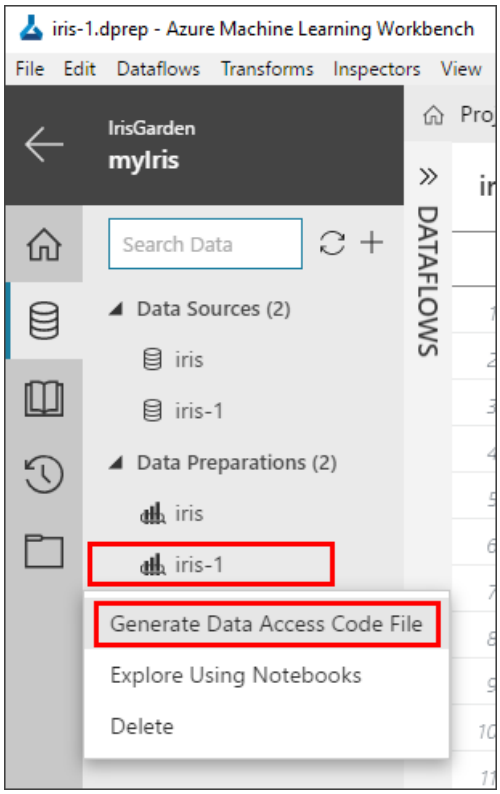
15. Close the data preparation editor. Select the **x** icon on the **iris-1** tab with the graph icon to close the tab. Your work is automatically saved into the **iris-1.dprep** file shown under the **Data Preparations** heading.

Generate Python/PySpark code to invoke a data preparation package

The output of a data preparation package can be explored directly in Python or in a Jupyter Notebook. The packages can be executed across multiple runtimes including local Python, Spark (including in Docker), and HDInsight.

1. Find the **iris-1.dprep** file under the Data Preparations tab.

2. Right-click the **iris-1.dprep** file, and select **Generate Data Access Code File** from the context menu.



A new file named **iris-1.py** opens with the following lines of code to invoke the logic you created as a data preparation package:

```
# Use the Azure Machine Learning data preparation package
from azureml.dataprep import package

# Use the Azure Machine Learning data collector to log various metrics
from azureml.logging import get_azureml_logger
logger = get_azureml_logger()

# This call will load the referenced package and return a DataFrame.
# If run in a PySpark environment, this call returns a
# Spark DataFrame. If not, it will return a Pandas DataFrame.
df = package.run('iris-1.dprep', dataflow_idx=0)

# Remove this line and add code that uses the DataFrame
df.head(10)
```

Depending on the context in which this code is run, `df` represents a different kind of DataFrame:

- When executing on a Python runtime, a pandas DataFrame is used.
- When executing in a Spark context, a Spark DataFrame is used.

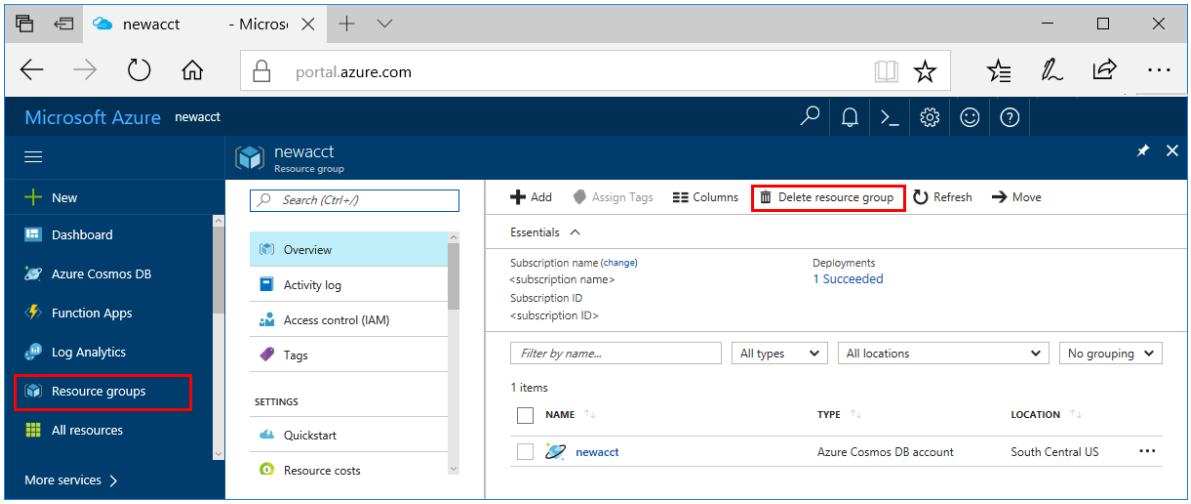
To learn more about how to prepare data in Azure Machine Learning Workbench, see the Get started with data preparation guide.

Clean up resources

If you're not going to continue to use this app, delete all resources created by this quickstart with the following steps so you don't incur any charges:

> **NOTE**
>
> These resources are useful when following the other Azure Machine Learning services tutorials now.

1. In the Azure portal, select **Resource groups** on the far left.



2. From the list of resource groups, select the resource group you created, and then click **Delete resource group**.

3. Type the name of the resource group to delete, and then click **Delete**.

Next steps

In this tutorial, you used Azure Machine Learning Workbench to:

- Create a new project
- Create a data preparation package
- Generate Python/PySpark code to invoke a data preparation package

You are ready to move on to the next part in the tutorial series, where you learn how to build an Azure Machine Learning model:

Tutorial 2 - Build models

Tutorial 2: Classify Iris - Build a model

6/13/2018 • 17 minutes to read • Edit Online

Azure Machine Learning services (preview) are an integrated, data science and advanced analytics solution for professional data scientists to prepare data, develop experiments, and deploy models at cloud scale.

This tutorial is **part two of a three-part series**. In this part of the tutorial, you use Azure Machine Learning services to:

- Open scripts and review code
- Execute scripts in a local environment
- Review run histories
- Execute scripts in a local Azure CLI window
- Execute scripts in a local Docker environment
- Execute scripts in a remote Docker environment
- Execute scripts in a cloud Azure HDInsight environment

This tutorial uses the timeless Iris flower data set.

Prerequisites

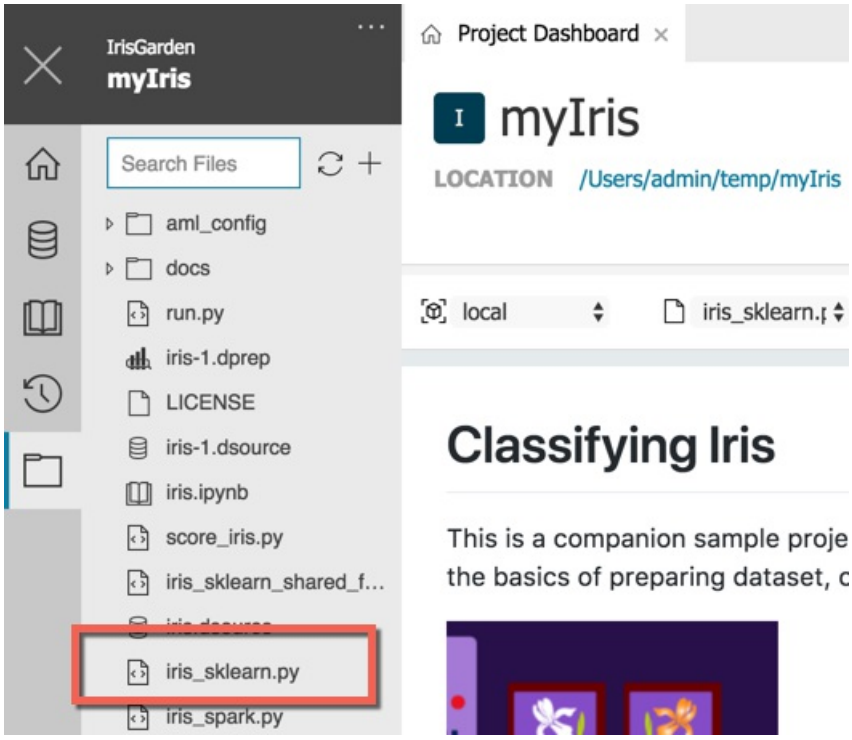
To complete this tutorial, you need:

- An Azure subscription. If you don't have an Azure subscription, create a free account before you begin.
- An experimentation account and Azure Machine Learning Workbench installed as described in this quickstart
- The project and prepared Iris data from Tutorial part 1
- A Docker engine installed and running locally. Docker's Community Edition is sufficient. Learn how to install Docker here: https://docs.docker.com/engine/installation/.

Review iris_sklearn.py and the configuration files

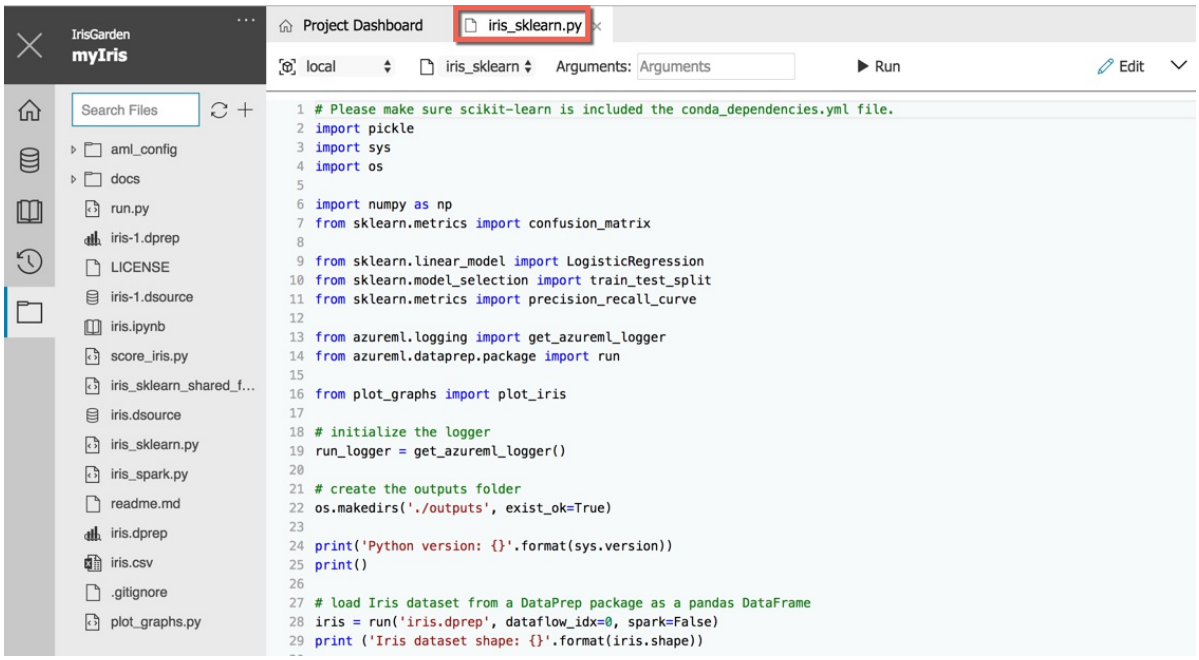
1. Launch the Azure Machine Learning Workbench application.
2. Open the **myIris** project you created in Part 1 of the tutorial series.
3. In the open project, select the **Files** button (the folder icon) on the far-left pane to open the file list in your project folder.

4. Select the **iris_sklearn.py** Python script file.



The code opens in a new text editor tab inside the Workbench. This is the script you use throughout this part of the tutorial.

> **NOTE**
>
> The code you see might not be exactly the same as the preceding code, because this sample project is updated frequently.



5. Inspect the Python script code to become familiar with the coding style.

   The script **iris_sklearn.py** performs the following tasks:

   - Loads the default data preparation package called **iris.dprep** to create a pandas DataFrame.
   - Adds random features to make the problem more difficult to solve. Randomness is necessary because Iris is a small data set that is easily classified with nearly 100% accuracy.
   - Uses the scikit-learn machine learning library to build a logistic regression model. This library comes with Azure Machine Learning Workbench by default.
   - Serializes the model using the pickle library into a file in the `outputs` folder.
   - Loads the serialized model, and then deserializes it back into memory.
   - Uses the deserialized model to make a prediction on a new record.
   - Plots two graphs, a confusion matrix and a multi-class receiver operating characteristic (ROC) curve, using the matplotlib library, and then saves them in the `outputs` folder. You can install this library in your environment if it isn't there already.
   - Plots the regulatization rate and model accuracy in the run history automatically. The `run_logger` object is used throughout to record the regularization rate and the model accuracy into the logs.

Run iris_sklearn.py in your local environment

1. Start the Azure Machine Learning command-line interface (CLI):

   a. Launch the Azure Machine Learning Workbench.

   b. From the Workbench menu, select **File** > **Open Command Prompt**.

   The Azure Machine Learning command-line interface (CLI) window starts in the project folder `C:\Temp\myIris\>` on Windows. This project is the same as the one you created in Part 1 of the tutorial.

> **IMPORTANT**
>
> You must use this CLI window to accomplish the next steps.

2. In the CLI window, install the Python plotting library, **matplotlib**, if you do not already have the library.

   The **iris_sklearn.py** script has dependencies on two Python packages: **scikit-learn** and **matplotlib**. The **scikit-learn** package is installed by Azure Machine Learning Workbench for your convenience. But, you need to install **matplotlib** if you don't have it installed yet.

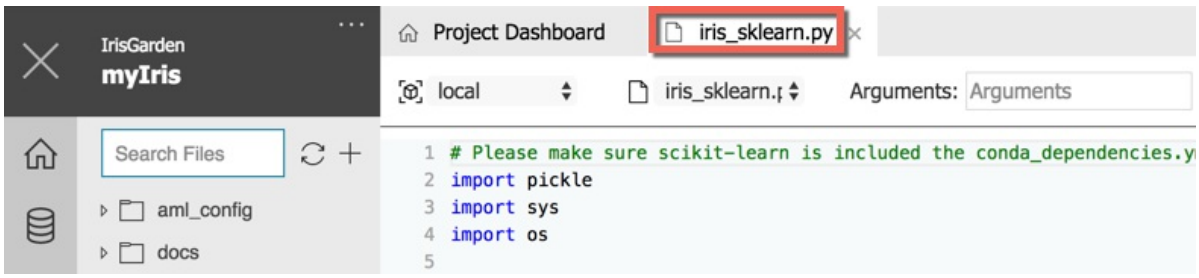
   If you move on without installing **matplotlib**, the code in this tutorial can still run successfully. However, the code will not be able to produce the confusion matrix output and the multi-class ROC curve plots shown in the history visualizations.

   ```
   pip install matplotlib
   ```

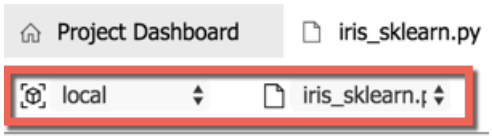
   This install takes about a minute.

3. Return to the Workbench application.

4. Find the tab called **iris_sklearn.py**.



5. In the toolbar of that tab, select **local** as the execution environment, and `iris_sklearn.py` as the script to run. These may already be selected.



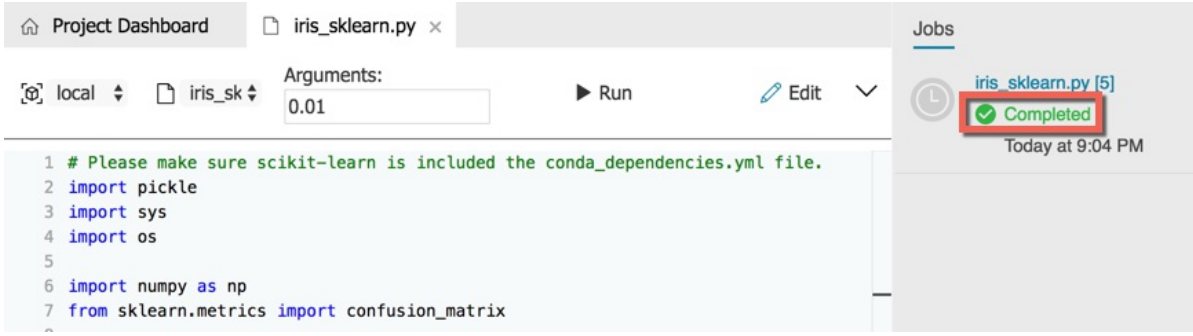
6. Move to the right side of the toolbar and enter `0.01` in the **Arguments** field.

   This value corresponds to the regularization rate of the logistic regression model.

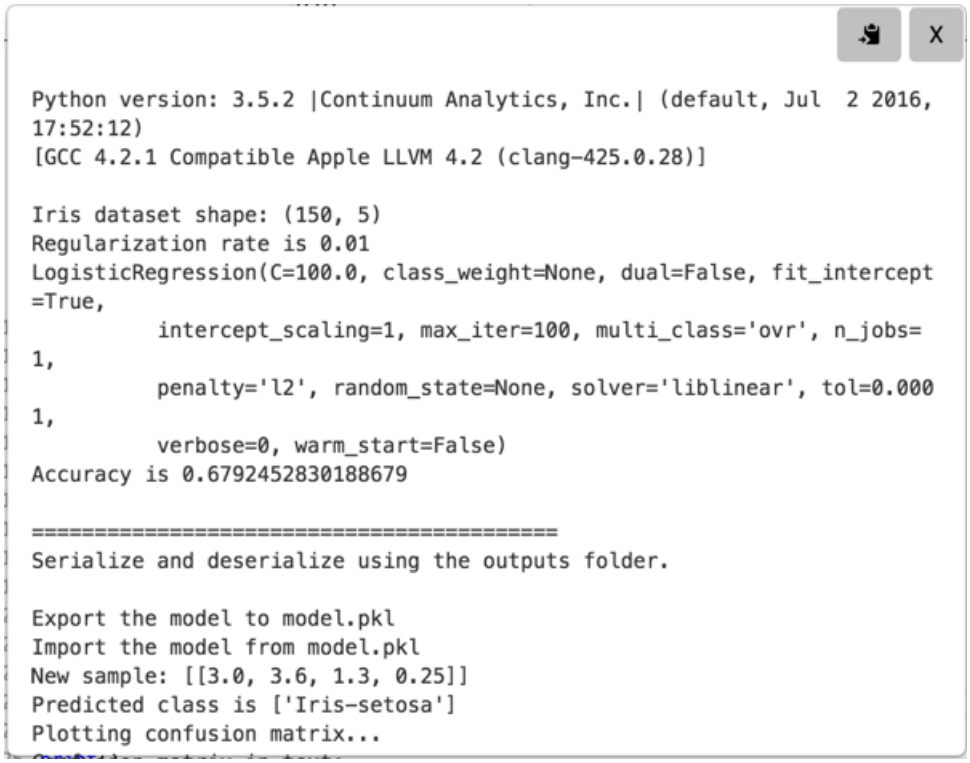
7. Select the **Run** button. A job is immediately scheduled. The job is listed in the **Jobs** pane on the right side of the workbench window.

   After a few moments, the status of the job transitions from **Submitting**, to **Running**, and finally to **Completed**.

8. Select **Completed** in the job status text in the **Jobs** pane.



A pop-up window opens and displays the standard output (stdout) text for the run. To close the stdout text, select the **Close** (**x**) button on the upper right of the pop-up window.



9. In the same job status in the **Jobs** pane, select the blue text **iris_sklearn.py [n]** (*n* is the run number) just above the **Completed** status and the start time. The **Run Properties** window opens and shows the following information for that particular run:

   - **Run Properties** information
   - **Outputs**
   - **Metrics**
   - **Visualizations**, if any
   - **Logs**

   When the run is finished, the pop-up window shows the following results:

> **NOTE**
>
> Because the tutorial introduced some randomization into the training set earlier, your exact results might vary from the results shown here.

```
Python version: 3.5.2 |Continuum Analytics, Inc.| (default, Jul  5 2016, 11:41:13) [MSC v.1900 64 bit
(AMD64)]

Iris dataset shape: (150, 5)
Regularization rate is 0.01
LogisticRegression(C=100.0, class_weight=None, dual=False, fit_intercept=True,
       intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
       penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
       verbose=0, warm_start=False)
Accuracy is 0.6792452830188679

========================================
Serialize and deserialize using the outputs folder.

Export the model to model.pkl
Import the model from model.pkl
New sample: [[3.0, 3.6, 1.3, 0.25]]
Predicted class is ['Iris-setosa']
Plotting confusion matrix...
Confusion matrix in text:
[[50  0  0]
 [ 1 37 12]
 [ 0  4 46]]
Confusion matrix plotted.
Plotting ROC curve....
ROC curve plotted.
Confusion matrix and ROC curve plotted. See them in Run History details pane.
```

10. Close the **Run Properties** tab, and then return to the **iris_sklearn.py** tab.

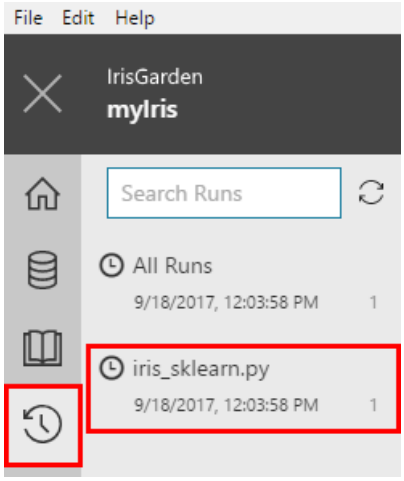
11. Repeat for additional runs.

    Enter a series of values in the **Arguments** field ranging from `0.001` to `10`. Select **Run** to execute the code a few more times. The argument value you change each time is fed to the logistic regression model in the code, resulting in different findings each time.

Review the run history in detail

In Azure Machine Learning Workbench, every script execution is captured as a run history record. If you open the **Runs** view, you can view the run history of a particular script.

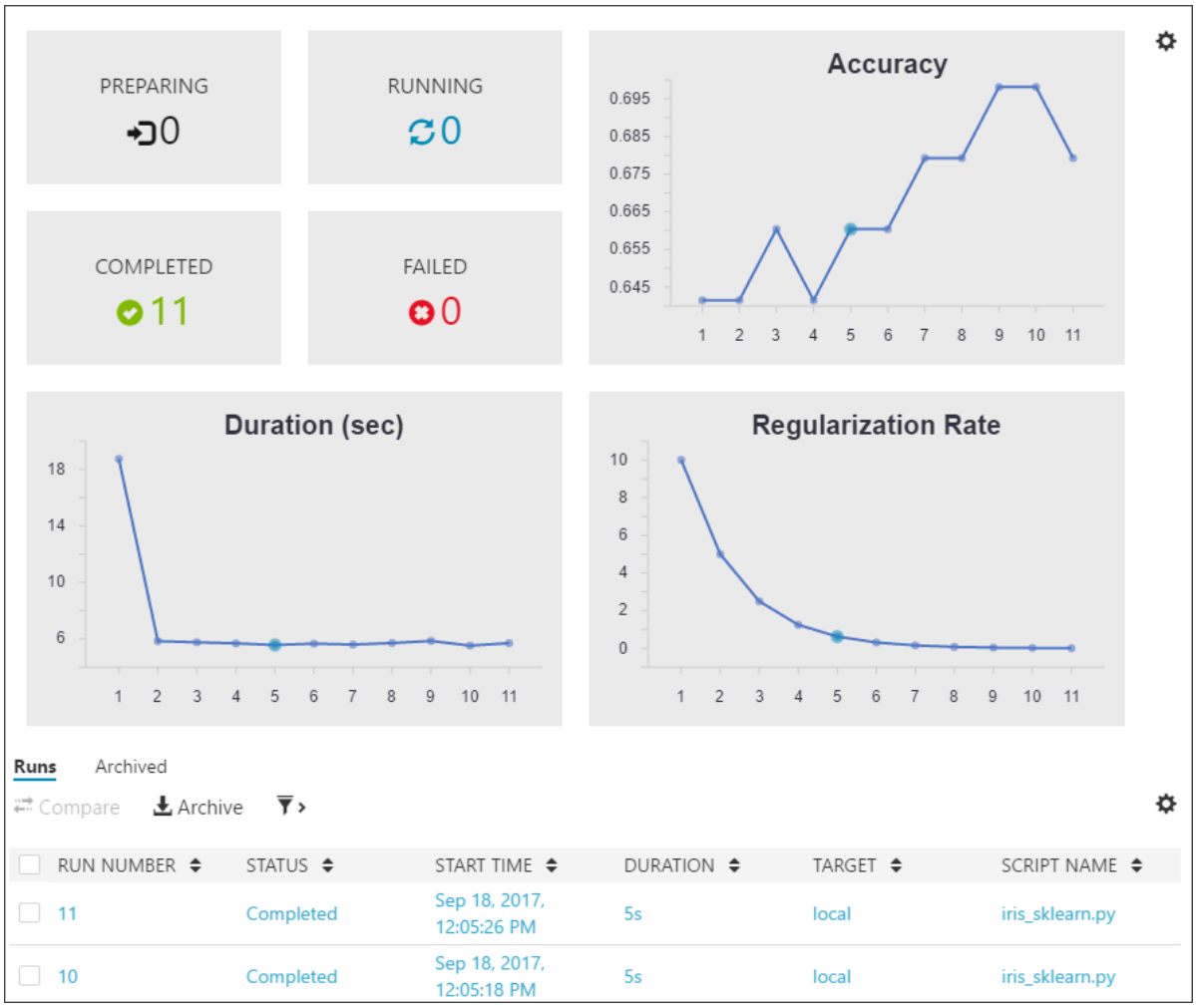
1. To open the list of **Runs**, select the **Runs** button (clock icon) on the left toolbar. Then select **iris_sklearn.py** to show the **Run Dashboard** of `iris_sklearn.py`.



2. The **Run Dashboard** tab opens.

   Review the statistics captured across the multiple runs. Graphs render in the top of the tab. Each run has a

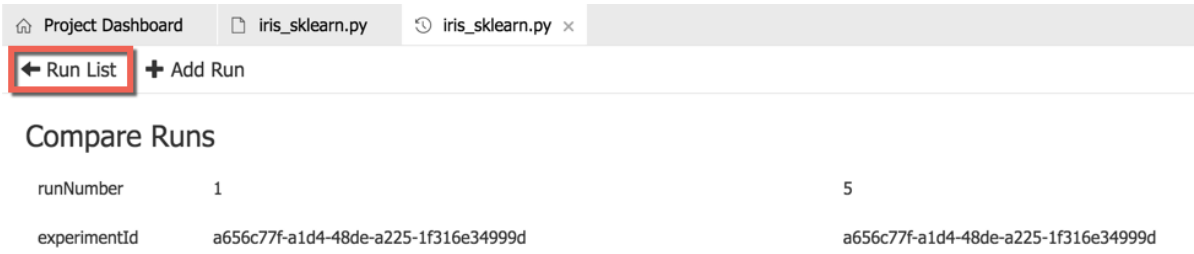
consecutive number, and the run details are listed in the table at the bottom of the screen.



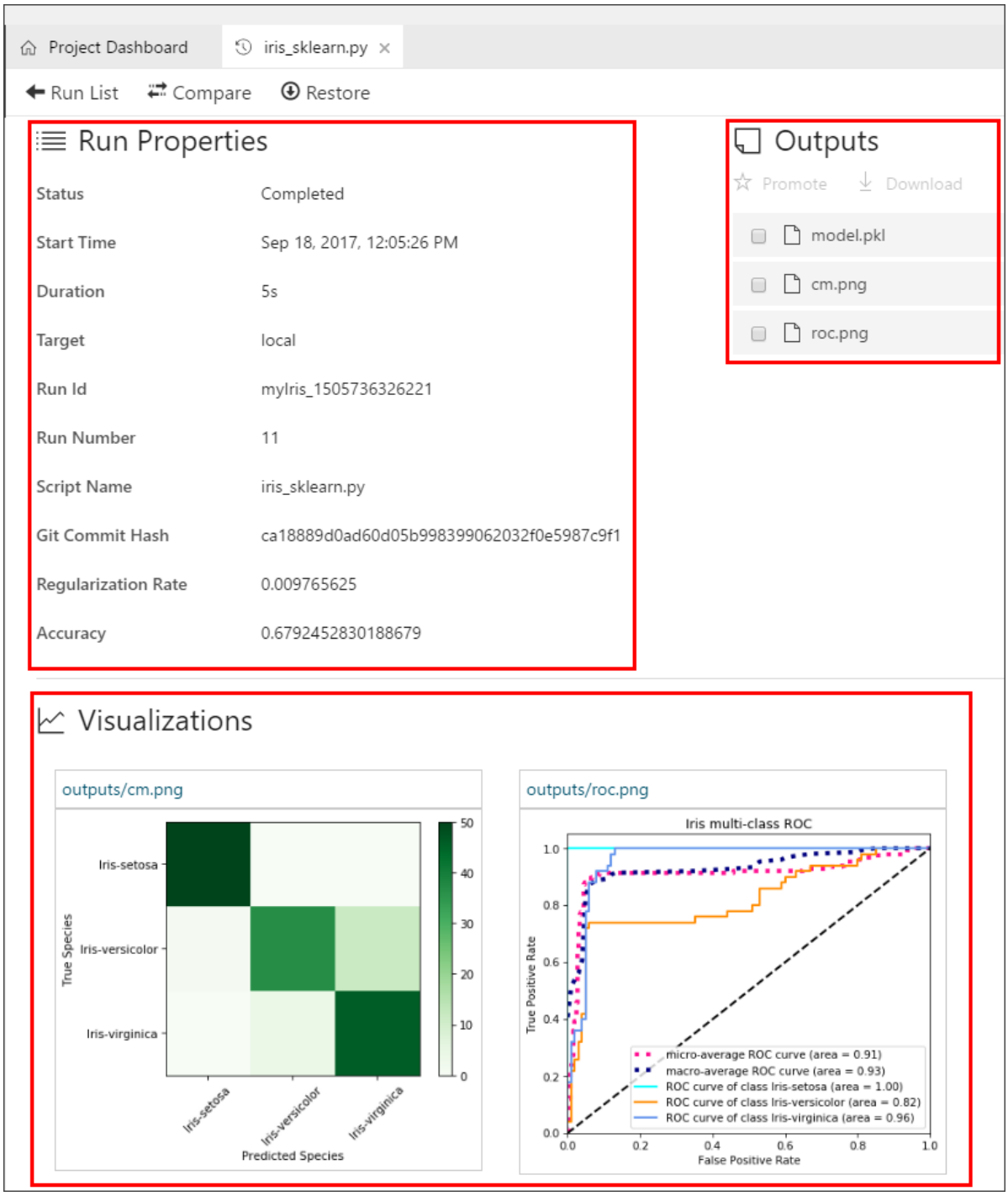
3. Filter the table, and then select any of the graphs to view the status, duration, accuracy, and regularization rate of each run.

4. Select the checkboxes next to two or more runs in the **Runs** table. Select the **Compare** button to open a detailed comparison pane. Review the side-by-side comparison.

5. To return to the **Run Dashboard**, select the **Run List** back button on the upper left of the **Comparison** pane.



6. Select an individual run to see the run detail view. Notice that the statistics for the selected run are listed in the **Run Properties** section. The files written into the output folder are listed in the **Outputs** section, and you can download the files from there.



The two plots, the confusion matrix and the multi-class ROC curve, are rendered in the **Visualizations** section. All the log files can also be found in the **Logs** section.

Run scripts in local Docker environments

Optionally, you can experiment with running scripts against a local Docker container. You can configure additional execution environments, such as Docker, and run your script in those environments.

> **NOTE**
>
> To experiment with dispatching scripts to run in a Docker container in a remote Azure VM or an Azure HDInsight Spark cluster, you can follow the instructions to create an Ubuntu-based Azure Data Science Virtual Machine or HDInsight cluster.

1. If you have not yet done so, install and start Docker locally on your Windows or MacOS machine. For more information, see the Docker installation instructions at https://docs.docker.com/install/. Community edition is sufficient.

2. On the left pane, select the **Folder** icon to open the **Files** list for your project. Expand the `aml_config` folder.

3. There are several environments that are preconfigured: **docker-python**, **docker-spark**, and **local**.

   Each environment has two files, such as `docker.compute` (for both **docker-python** and **docker-spark**) and `docker-python.runconfig`. Open each file to see that certain options are configurable in the text editor.

   To clean up, select **Close** (**x**) on the tabs for any open text editors.

4. Run the **iris_sklearn.py** script by using the **docker-python** environment:

   - On the left toolbar, select the **Clock** icon to open the **Runs** pane. Select **All Runs**.
   - On the top of the **All Runs** tab, select **docker-python** as the targeted environment instead of the default **local**.
   - Next, move to the right side and select **iris_sklearn.py** as the script to run.
   - Leave the **Arguments** field blank because the script specifies a default value.
   - Select the **Run** button.

5. Observe that a new job starts. It appears in the **Jobs** pane on the right side of the workbench window.

   When you run against Docker for the first time, the job takes a few extra minutes to finish.

   Behind the scenes, Azure Machine Learning Workbench builds a new Docker file. The new file references the base Docker image specified in the `docker.compute` file and the dependency Python packages specified in the `conda_dependencies.yml` file.

   The Docker engine performs the following tasks:

   - Downloads the base image from Azure.
   - Installs the Python packages specified in the `conda_dependencies.yml` file.
   - Starts a Docker container.
   - Copies or references, depending on the run configuration, the local copy of the project folder.
   - Executes the `iris_sklearn.py` script.

   In the end, you should see the exact same results as you do when you target **local**.

6. Now, let's try Spark. The Docker base image contains a preinstalled and configured Spark instance that you can use to execute a PySpark script. Using this base image is an easy way to develop and test your Spark program, without having to spend time installing and configuring Spark yourself.

   Open the `iris_spark.py` file. This script loads the `iris.csv` data file, and uses the logistic regression algorithm from the Spark Machine Learning library to classify the Iris data set. Now change the run environment to **docker-spark** and the script to **iris_spark.py**, and then run it again. This process takes a little longer because a Spark session has to be created and started inside the Docker container. You can also see the stdout is different than the stdout of `iris_spark.py`.

7. Start a few more runs and play with different arguments.

8. Open the `iris_spark.py` file to see the logistic regression model built using the Spark Machine Learning library.

9. Interact with the **Jobs** pane, run a history list view, and run a details view of your runs across different execution environments.

Run scripts in the CLI window

1. Start the Azure Machine Learning command-line interface (CLI):

   a. Launch the Azure Machine Learning Workbench.

   b. From the Workbench menu, select **File** > **Open Command Prompt**.

   The CLI prompt starts in the project folder `C:\Temp\myIris\>` on Windows. This is the project you created in Part 1 of the tutorial.

   > **IMPORTANT**
   > You must use this CLI window to accomplish the next steps.

2. In the CLI window, log in to Azure. Learn more about az login.

   You may already be logged in. In that case, you can skip this step.

   a. At the command prompt, enter:

   ```
   az login
   ```

   This command returns a code to use in your browser at https://aka.ms/devicelogin.

   b. Go to https://aka.ms/devicelogin in your browser.

   c. When prompted, enter the code, which you received in the CLI, into your browser.

   The Workbench app and CLI use independent credential caches when authenticating against Azure resources. After you log in, you won't need to authenticated again until the cached token expires.

3. If your organization has multiple Azure subscriptions (enterprise environment), you must set the subscription to be used. Find your subscription, set it using the subscription ID, and then test it.

   a. List every Azure subscription to which you have access using this command:

   ```
   az account list -o table
   ```

   The **az account list** command returns the list of subscriptions available to your login. If there is more than one, identify the subscription ID value for the subscription you want to use.

   b. Set the Azure subscription you want to use as the default account:

   ```
   az account set -s <your-subscription-id>
   ```

   where <your-subscription-id> is ID value for the subscription you want to use. Do not include the brackets.

   c. Confirm the new subscription setting by requesting the details for the current subscription.

   ```
   az account show
   ```

4. In the CLI window, install the Python plotting library, **matplotlib**, if you do not already have the library.

   ```
   pip install matplotlib
   ```

5. In the CLI window, submit the **iris_sklearn.py** script as an experiment.

   The execution of iris_sklearn.py is run against the local compute context.

   - On Windows:

     ```
     az ml experiment submit -c local .\iris_sklearn.py
     ```

   - On MacOS:

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

   Your output should be similar to:

   ```
   RunId: myIris_1521077190506

   Executing user inputs .....
   ==========================

   Python version: 3.5.2 |Continuum Analytics, Inc.| (default, Jul  2 2016, 17:52:12)
   [GCC 4.2.1 Compatible Apple LLVM 4.2 (clang-425.0.28)]

   Iris dataset shape: (150, 5)
   Regularization rate is 0.01
   LogisticRegression(C=100.0, class_weight=None, dual=False, fit_intercept=True,
             intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
             penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
             verbose=0, warm_start=False)
   Accuracy is 0.6792452830188679

   ==========================================
   Serialize and deserialize using the outputs folder.

   Export the model to model.pkl
   Import the model from model.pkl
   New sample: [[3.0, 3.6, 1.3, 0.25]]
   Predicted class is ['Iris-setosa']
   Plotting confusion matrix...
   Confusion matrix in text:
   [[50  0  0]
    [ 1 37 12]
    [ 0  4 46]]
   Confusion matrix plotted.
   Plotting ROC curve....
   ROC curve plotted.
   Confusion matrix and ROC curve plotted. See them in Run History details page.

   Execution Details
   =================
   RunId: myIris_1521077190506
   ```

6. Review the output. You have the same output and results that you had when you used the Workbench to run the script.

7. In the CLI window, run the Python script, **iris_sklearn.py**, again using a Docker execution environment (if you have Docker installed on your machine).

   - If your container is on Windows:

EXECUTION ENVIRONMENT	COMMAND ON WINDOWS
Python	`az ml experiment submit -c docker-python .\iris_sklearn.py 0.01`
Spark	`az ml experiment submit -c docker-spark .\iris_spark.py 0.1`

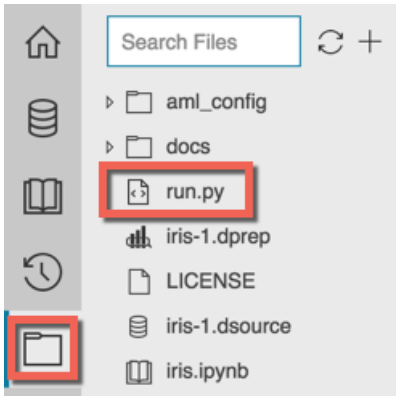
- If your container is on MacOS:

EXECUTION ENVIRONMENT	COMMAND ON WINDOWS
Python	`az ml experiment submit -c docker-python iris_sklearn.py 0.01`
Spark	`az ml experiment submit -c docker-spark iris_spark.py 0.1`

8. Go back to the Workbench, and:

   a. Select the folder icon on the left pane to list the project files.

   b. Open the Python script named **run.py**. This script is useful to loop over various regularization rates.



9. Run the experiment multiple times with those rates.

   This script starts an `iris_sklearn.py` job with a regularization rate of `10.0` (a ridiculously large number). The script then cuts the rate to half in the following run, and so on, until the rate is no smaller than `0.005`.

   The script contains the following code:

```
# run iris_sklearn.py with descending regularization rates
# run this with just "python run.py". It will fail if you run using az ml execute.

import os

# set regularization rate as an argument
reg = 10
while reg > 0.005:
    os.system('az ml experiment submit -c local ./iris_sklearn.py {}'.format(reg))
    # cut regularization rate to half
    reg = reg / 2
```

10. Run the **run.py** script from the command line as follows:

```
python run.py
```

This command submits iris_sklearn.py multiple times with different regularization rates

When `run.py` finishes, you can see graphs of different metrics in your run history list view in the workbench.

Run scripts in a remote Docker container

To execute your script in a Docker container on a remote Linux machine, you need to have SSH access (username and password) to that remote machine. In addition, the machine must have a Docker engine installed and running. The easiest way to obtain such a Linux machine is to create an Ubuntu-based Data Science Virtual Machine (DSVM) on Azure. Learn how to create an Ubuntu DSVM to use in Azure ML Workbench.

> **NOTE**
> The CentOS-based DSVM is *not* supported.

1. After the VM is created, you can attach the VM as an execution environment by generating a pair of `.runconfig` and `.compute` files. Use the following command to generate the files.

   Let's name the new compute target `myvm`.

   ```
   az ml computetarget attach remotedocker --name myvm --address <your-IP> --username <your-username> --
   password <your-password>
   ```

   > **NOTE**
   > The IP address can also be a publicly addressable fully-qualified domain name (FQDN) such as `vm-name.southcentralus.cloudapp.azure.com`. It is a good practice to add an FQDN to your DSVM and use it instead of an IP address. This practice is a good idea because you might turn off the VM at some point to save on cost. Additionally, the next time you start the VM, the IP address might have changed.

   > **NOTE**
   > In addition to username and password authentication, you can specify a private key and the corresponding passphrase (if any) using the `--private-key-file` and (optionally) the `--private-key-passphrase` options.

   Next, prepare the **myvm** compute target by running this command.

   ```
   az ml experiment prepare -c myvm
   ```

   The preceding command constructs the Docker image in the VM to get it ready to run the scripts.

   > **NOTE**
   > You can also change the value of `PrepareEnvironment` in `myvm.runconfig` from the default value `false` to `true`. This change automatically prepares the Docker container as part of the first run.

2. Edit the generated `myvm.runconfig` file under `aml_config` and change the framework from the default value `PySpark` to `Python`:

```
Framework: Python
```

> **NOTE**
> While PySpark should also work, using Python is more efficient if you don't actually need a Spark session to run your Python script.

3. Issue the same command as you did before in the CLI window, using target *myvm* this time to execute iris_sklearn.py in a remote Docker container:

   ```
   az ml experiment submit -c myvm iris_sklearn.py
   ```

   The command executes as if you're in a `docker-python` environment, except that the execution happens on the remote Linux VM. The CLI window displays the same output information.

4. Let's try using Spark in the container. Open File Explorer. Make a copy of the `myvm.runconfig` file and name it `myvm-spark.runconfig`. Edit the new file to change the `Framework` setting from `Python` to `PySpark`:

   ```
   Framework: PySpark
   ```

   Don't make any changes to the `myvm.compute` file. The same Docker image on the same VM gets used for the Spark execution. In the new `myvm-spark.runconfig`, the `Target` field points to the same `myvm.compute` file via its name `myvm`.

5. Type the following command to run the **iris_spark.py** script in the Spark instance running inside the remote Docker container:

   ```
   az ml experiment submit -c myvm-spark .\iris_spark.py
   ```

Run scripts in HDInsight clusters

You can also run this script in an HDInsight Spark cluster. Learn how to create an HDInsight Spark Cluster to use in Azure ML Workbench.

> **NOTE**
> The HDInsight cluster must use Azure Blob as the primary storage. Using Azure Data Lake storage is not supported yet.

1. If you have access to a Spark for Azure HDInsight cluster, generate an HDInsight run configuration command as shown here. Provide the HDInsight cluster name and your HDInsight username and password as the parameters.

   Use the following command to create a compute target that points to a HDInsight cluster:

   ```
   az ml computetarget attach cluster --name myhdi --address <cluster head node FQDN> --username <your-
   username> --password <your-password>
   ```

   To prepare the HDInsight cluster, run this command:

```
az ml experiment prepare -c myhdi
```

The cluster head node FQDN is typically `<your_cluster_name>-ssh.azurehdinsight.net`.

> **NOTE**
> The `username` is the cluster SSH username defined during HDInsight setup. By default, the value is `sshuser`. The value is not `admin`, which is the other user created during setup to enable access to the cluster's admin website.

2. Run the **iris_spark.py** script in the HDInsight cluster with this command:

```
az ml experiment submit -c myhdi .\iris_spark.py
```

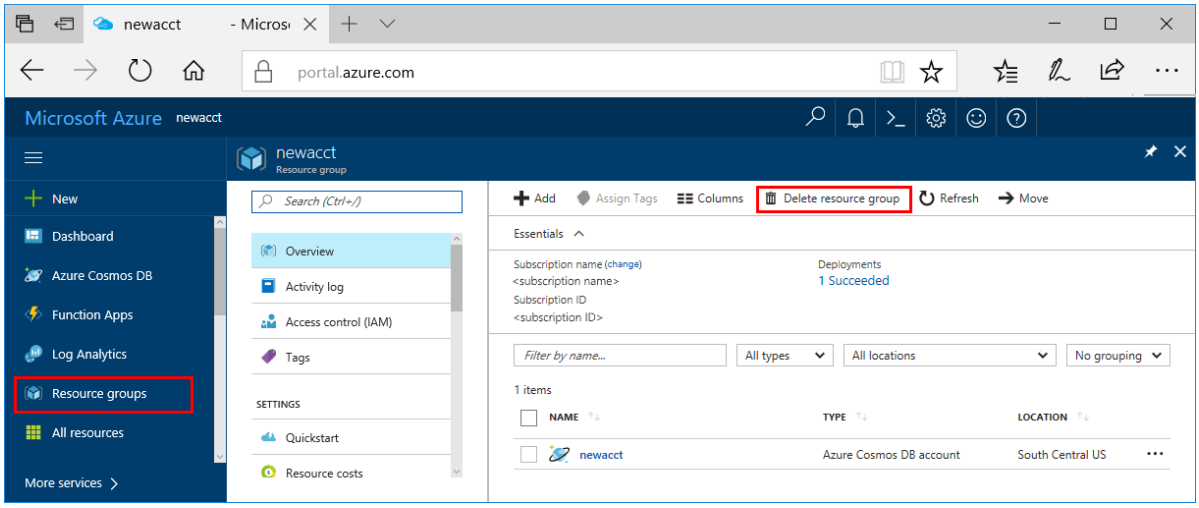
> **NOTE**
> When you execute against a remote HDInsight cluster, you can also view the Yet Another Resource Negotiator (YARN) job execution details at `https://<your_cluster_name>.azurehdinsight.net/yarnui` by using the `admin` user account.

Clean up resources

If you're not going to continue to use this app, delete all resources created by this quickstart with the following steps so you don't incur any charges:

> **NOTE**
> These resources are useful when following the other Azure Machine Learning services tutorials now.

1. In the Azure portal, select **Resource groups** on the far left.



2. From the list of resource groups, select the resource group you created, and then click **Delete resource group**.

3. Type the name of the resource group to delete, and then click **Delete**.

Next steps

In this second part of the three-part tutorial series, you learned how to:

- Open scripts and review the code in Workbench
- Execute scripts in a local environment
- Review the run history
- Execute scripts in a local Docker environment

Now, you can try out the third part of this tutorial series in which you can deploy the logistic regression model you created as a real-time web service.

Tutorial 3 - Deploy models

Tutorial 3: Classify Iris: Deploy a model

6/13/2018 • 13 minutes to read • Edit Online

Azure Machine Learning (preview) is an integrated, end-to-end data science and advanced analytics solution for professional data scientists. Data scientists can use it to prepare data, develop experiments, and deploy models at cloud scale.

This tutorial is **part three of a three-part series**. In this part of the tutorial, you use Machine Learning (preview) to:

- Locate the model file.
- Generate a scoring script and schema file.
- Prepare the environment.
- Create a real-time web service.
- Run the real-time web service.
- Examine the output blob data.

This tutorial uses the timeless Iris flower data set.

Prerequisites

To complete this tutorial, you need:

- An Azure subscription. If you don't have an Azure subscription, create a free account before you begin.
- An experimentation account and Azure Machine Learning Workbench installed as described in this quickstart
- The classification model from Tutorial part 2
- A Docker engine installed and running locally

Download the model pickle file

In the previous part of the tutorial, the **iris_sklearn.py** script was run in the Machine Learning Workbench locally. This action serialized the logistic regression model by using the popular Python object-serialization package pickle.

1. Open the Machine Learning Workbench application. Then open the **myIris** project you created in the previous parts of the tutorial series.

2. After the project is open, select the **Files** button (folder icon) on the left pane to open the file list in your project folder.

3. Select the **iris_sklearn.py** file. The Python code opens in a new text editor tab inside the workbench.

4. Review the **iris_sklearn.py** file to see where the pickle file was generated. Select Ctrl+F to open the **Find** dialog box, and then find the word **pickle** in the Python code.

   This code snippet shows how the pickle output file was generated. The output pickle file is named **model.pkl** on the disk.

   ```
   print("Export the model to model.pkl")
   f = open('./outputs/model.pkl', 'wb')
   pickle.dump(clf1, f)
   f.close()
   ```

5. Locate the model pickle file in the output files of a previous run.

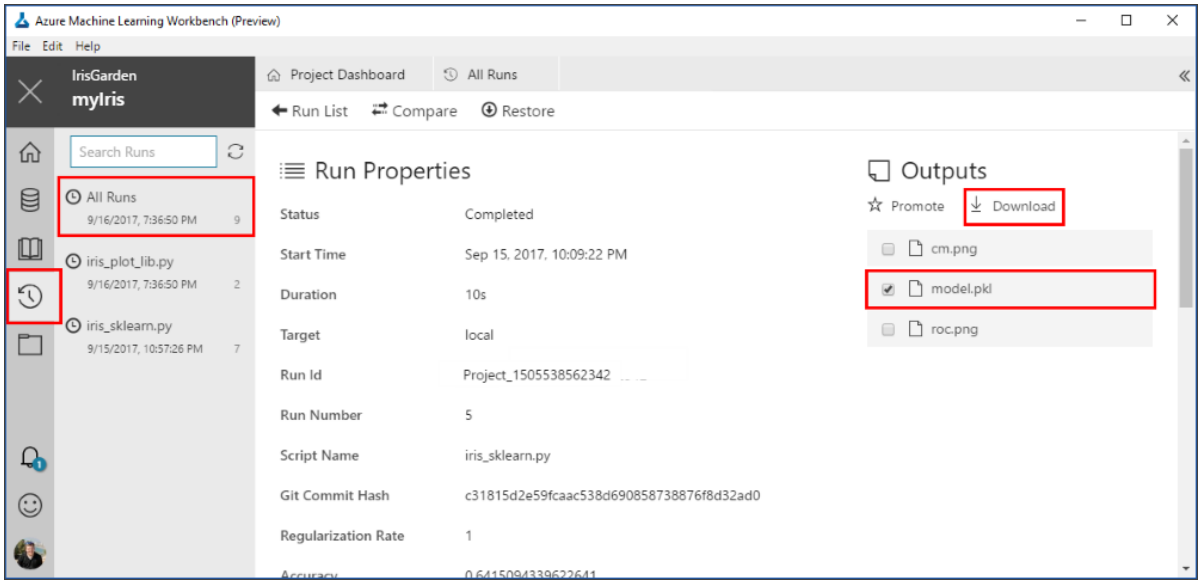
   When you ran the **iris_sklearn.py** script, the model file was written to the **outputs** folder with the name **model.pkl**. This folder lives in the execution environment that you choose to run the script, and not in your local project folder.

   a. To locate the file, select the **Runs** button (clock icon) on the left pane to open the list of **All Runs**.

   b. The **All Runs** tab opens. In the table of runs, select one of the recent runs where the target was **local** and the script name was **iris_sklearn.py**.

   c. The **Run Properties** pane opens. In the upper-right section of the pane, notice the **Outputs** section.

   d. To download the pickle file, select the check box next to the **model.pkl** file, and then select **Download**. Save the file to the root of your project folder. The file is needed in the upcoming steps.



   Read more about the `outputs` folder in How to read and write large data files.

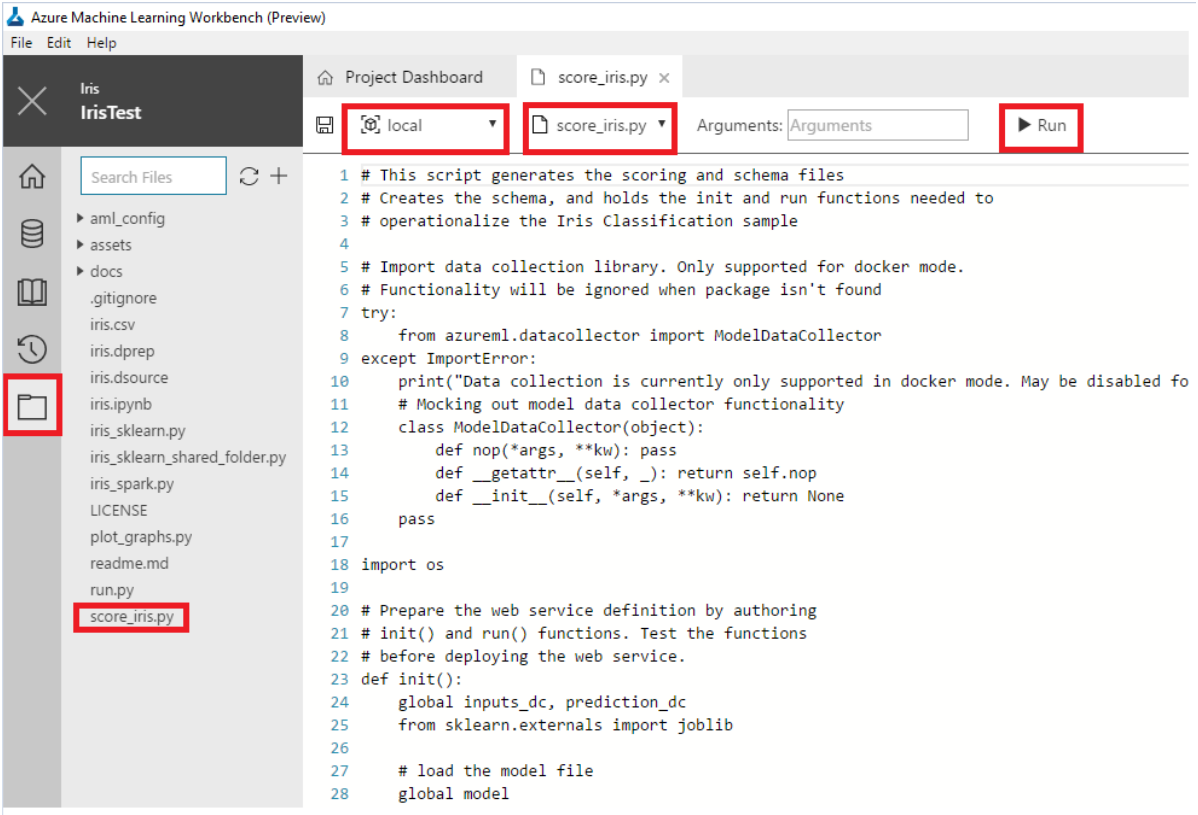
Get the scoring script and schema files

To deploy the web service along with the model file, you also need a scoring script. Optionally, you need a schema for the web service input data. The scoring script loads the **model.pkl** file from the current folder and uses it to produce new predictions.

1. Open the Machine Learning Workbench application. Then open the **myIris** project you created in the previous part of the tutorial series.

2. After the project is open, select the **Files** button (folder icon) on the left pane to open the file list in your project folder.

3. Select the **score_iris.py** file. The Python script opens. This file is used as the scoring file.



4. To get the schema file, run the script. Select the **local** environment and the **score_iris.py** script in the command bar, and then select **Run**.

   This script creates a JSON file in the **Outputs** section, which captures the input data schema required by the model.

5. Note the **Jobs** pane on the right side of the **Project Dashboard** pane. Wait for the latest **score_iris.py** job to display the green **Completed** status. Then select the hyperlink **score_iris.py** for the latest job run to see the run details.

6. On the **Run Properties** pane, in the **Outputs** section, select the newly created **service_schema.json** file. Select the check box next to the file name, and then select **Download**. Save the file into your project root folder.

7. Return to the previous tab where you opened the **score_iris.py** script. By using data collection, you can capture model inputs and predictions from the web service. The following steps are of particular interest for data collection.

8. Review the code at the top of the file, which imports class **ModelDataCollector**, because it contains the model data collection functionality:

```
from azureml.datacollector import ModelDataCollector
```

9. Review the following lines of code in the **init()** function that instantiates **ModelDataCollector**:

```
global inputs_dc, prediction_dc
inputs_dc = ModelDataCollector('model.pkl',identifier="inputs")
prediction_dc = ModelDataCollector('model.pkl', identifier="prediction")`
```

10. Review the following lines of code in the **run(input_df)** function as it collects the input and prediction data:

```
inputs_dc.collect(input_df)
prediction_dc.collect(pred)
```

Now you're ready to prepare your environment to operationalize the model.

Prepare to operationalize locally [For development and testing your service]

Use *local mode* deployment to run in Docker containers on your local computer.

You can use *local mode* for development and testing. The Docker engine must be running locally to complete the following steps to operationalize the model. You can use the `-h` flag at the end of each command to show the corresponding help message.

> **NOTE**
> If you don't have the Docker engine locally, you can still proceed by creating a cluster in Azure for deployment. You can keep the cluster for re-use, or delete it after the tutorial so you don't incur ongoing charges.

> **NOTE**
> Web services deployed locally do not show up in Azure Portal's list of services. They will be running in Docker on the local machine.

1. Open the command-line interface (CLI). In the Machine Learning Workbench application, on the **File** menu, select **Open Command Prompt**.

   The command-line prompt opens in your current project folder location **c:\temp\myIris>**.

2. Make sure the Azure resource provider **Microsoft.ContainerRegistry** is registered in your subscription. You must register this resource provider before you can create an environment in step 3. You can check to see if it's already registered by using the following command:

   ```
   az provider list --query "[].{Provider:namespace, Status:registrationState}" --out table
   ```

   You should see output like this:

   ```
   Provider                                  Status
   --------                                  ------
   Microsoft.Authorization                   Registered
   Microsoft.ContainerRegistry               Registered
   microsoft.insights                        Registered
   Microsoft.MachineLearningExperimentation  Registered
   ...
   ```

   If **Microsoft.ContainerRegistry** is not registered, you can register it by using the following command:

   ```
   az provider register --namespace Microsoft.ContainerRegistry
   ```

   Registration can take a few minutes. You can check on its status by using the previous **az provider list** command or the following command:

```
az provider show -n Microsoft.ContainerRegistry
```

The third line of the output displays **"registrationState": "Registering"**. Wait a few moments and repeat the **show** command until the output displays **"registrationState": "Registered"**.

> **NOTE**
>
> If you are deploying to an ACS cluster, you need register the **Microsoft.ContainerService** resource provider as well using the exact same approach.

3. Create the environment. You must run this step once per environment. For example, run it once for development environment, and once for production. Use *local mode* for this first environment. You can try the `-c` or `--cluster` switch in the following command to set up an environment in *cluster mode* later.

   The following setup command requires you to have Contributor access to the subscription. If you don't have that, you need at least Contributor access to the resource group that you are deploying to. In the latter case, you need to specify the resource group name as part of the setup command by using the `-g` flag.

```
az ml env setup -n <new deployment environment name> --location <e.g. eastus2>
```

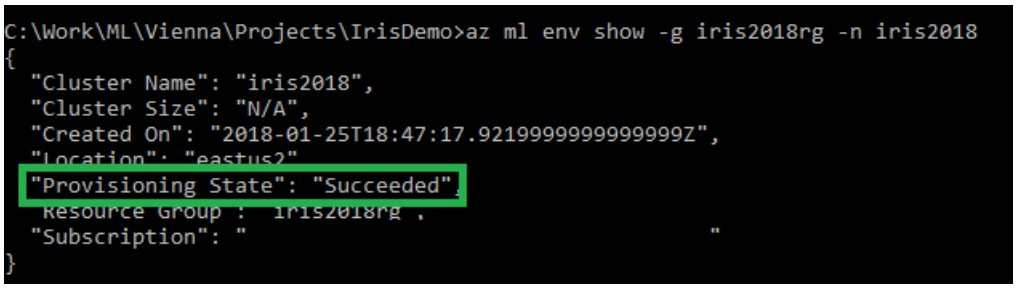
   Follow the on-screen instructions to provision a storage account for storing Docker images, an Azure container registry that lists the Docker images, and an Azure Application Insights account that gathers telemetry. If you use the `-c` switch, the command will additionally create a Container Service cluster.

   The cluster name is a way for you to identify the environment. The location should be the same as the location of the Model Management account you created from the Azure portal.

   To make sure that the environment is set up successfully, use the following command to check the status:

```
az ml env show -n <deployment environment name> -g <existing resource group name>
```

   Make sure that "Provisioning State" has the value "Succeeded", as shown, before you set the environment in step 5:



4. If you didn't create a Model Management account in previous parts of this tutorial, do so now. This is a one-time setup.

```
az ml account modelmanagement create --location <e.g. eastus2> -n <new model management account name> -
g <existing resource group name> --sku-name S1
```

5. Set the Model Management account.

```
az ml account modelmanagement set -n <youracctname> -g <yourresourcegroupname>
```

6. Set the environment.

   After the setup finishes, use the following command to set the environment variables required to operationalize the environment. Use the same environment name that you used previously in step 3. Use the same resource group name that was output in the command window when the setup process finished.

   ```
   az ml env set -n <deployment environment name> -g <existing resource group name>
   ```

7. To verify that you have properly configured your operationalized environment for local web service deployment, enter the following command:

   ```
   az ml env show
   ```

Now you're ready to create the real-time web service.

> **NOTE**
> You can reuse your Model Management account and environment for subsequent web service deployments. You don't need to create them for each web service. An account or an environment can have multiple web services associated with it.

Create a real-time web service in one command

1. To create a real-time web service, use the following command:

   ```
   az ml service create realtime -f score_iris.py --model-file model.pkl -s service_schema.json -n irisapp
   -r python --collect-model-data true -c aml_config\conda_dependencies.yml
   ```

   This command generates a web service ID you can use later.

   The following switches are used with the **az ml service create realtime** command:

   - `-f`: The scoring script file name.
   - `--model-file`: The model file. In this case, it's the pickled model.pkl file.
   - `-s`: The service schema. This was generated in a previous step by running the **score_iris.py** script locally.
   - `-n`: The app name, which must be all lowercase.
   - `-r`: The runtime of the model. In this case, it's a Python model. Valid runtimes are `python` and `spark-py`.
   - `--collect-model-data true`: This switch enables data collection.
   - `-c`: Path to the conda dependencies file where additional packages are specified.

   > **IMPORTANT**
   > The service name, which is also the new Docker image name, must be all lowercase. Otherwise, you get an error.

2. When you run the command, the model and the scoring files are uploaded to the storage account you created as part of the environment setup. The deployment process builds a Docker image with your model, schema, and scoring file in it, and then pushes it to the Azure container registry:

<ACR_name>.azureacr.io/<imagename>:<version>.

The command pulls down the image locally to your computer and then starts a Docker container based on that image. If your environment is configured in cluster mode, the Docker container is deployed into the Azure Cloud Services Kubernetes cluster instead.

As part of the deployment, an HTTP REST endpoint for the web service is created on your local machine. After a few minutes, the command should finish with a success message. Your web service is ready for action!

3. To see the running Docker container, use the **docker ps** command:

```
docker ps
```

[Optional alternative] Create a real-time web service by using separate commands

As an alternative to the **az ml service create realtime** command shown previously, you also can perform the steps separately.

First, register the model. Then generate the manifest, build the Docker image, and create the web service. This step-by-step approach gives you more flexibility at each step. Additionally, you can reuse the entities generated in previous steps and rebuild the entities only when needed.

1. Register the model by providing the pickle file name.

```
az ml model register --model model.pkl --name model.pkl
```

This command generates a model ID.

2. Create a manifest.

To create a manifest, use the following command and provide the model ID output from the previous step:

```
az ml manifest create --manifest-name <new manifest name> -f score_iris.py -r python -i <model ID> -s
service_schema.json -c aml_config\conda_dependencies.yml
```

This command generates a manifest ID.

3. Create a Docker image.

To create a Docker image, use the following command and provide the manifest ID value output from the previous step. You also can optionally include the conda dependencies by using the `-c` switch.

```
az ml image create -n irisimage --manifest-id <manifest ID>
```

This command generates a Docker image ID.

4. Create the service.

To create a service, use the following command and provide the image ID output from the previous step:

```
az ml service create realtime --image-id <image ID> -n irisapp --collect-model-data true
```

This command generates a web service ID.

You are now ready to run the web service.

Run the real-time web service

To test the **irisapp** web service that's running, use a JSON-encoded record containing an array of four random numbers.

1. The web service includes sample data. When running in local mode, you can call the **az ml service usage realtime** command. That call retrieves a sample run command that you can use to test the service. The call also retrieves the scoring URL that you can use to incorporate the service into your own custom app.

   ```
   az ml service usage realtime -i <web service ID>
   ```

2. To test the service, execute the returned service run command:

   ```
   az ml service run realtime -i <web service ID> -d "{\"input_df\": [{\"petal width\": 0.25, \"sepal length\": 3.0, \"sepal width\": 3.6, \"petal length\": 1.3}]}"
   ```

   The output is **"Iris-setosa"**, which is the predicted class. (Your result might be different.)

View the collected data in Azure Blob storage

1. Sign in to the Azure portal.

2. Locate your storage accounts. To do so, select **All Services**.

3. In the search box, enter **Storage accounts**, and then select Enter.

4. From the **Storage accounts** search box, select the **Storage account** resource matching your environment.

   > **TIP**
   > To determine which storage account is in use:
   > 1. Open Machine Learning Workbench.
   > 2. Select the project you're working on.
   > 3. Open a command line prompt from the **File** menu.
   > 4. At the command line prompt, enter `az ml env show -v`, and check the *storage_account* value. This is the name of your storage account.

5. After the **Storage account** pane opens, select **Blobs** from the **Services** section. Locate the container named **modeldata**.

   If you don't see any data, you might need to wait up to 10 minutes after the first web-service request to see the data propagate to the storage account.

   Data flows into blobs with the following container path:

   ```
   /modeldata/<subscription_id>/<resource_group_name>/<model_management_account_name>/<webservice_name>/<model_id>-<model_name>-<model_version>/<identifier>/<year>/<month>/<day>/data.csv
   ```

6. You can consume this data from Azure Blob storage. There are a variety of tools that use both Microsoft software and open-source tools, such as:

- Machine Learning: Open the CSV file by adding the CSV file as a data source.
- Excel: Open the daily CSV files as a spreadsheet.
- Power BI: Create charts with data pulled from the CSV data in blobs.
- Hive: Load the CSV data into a Hive table, and perform SQL queries directly against the blobs.
- Spark: Create a DataFrame with a large portion of CSV data.

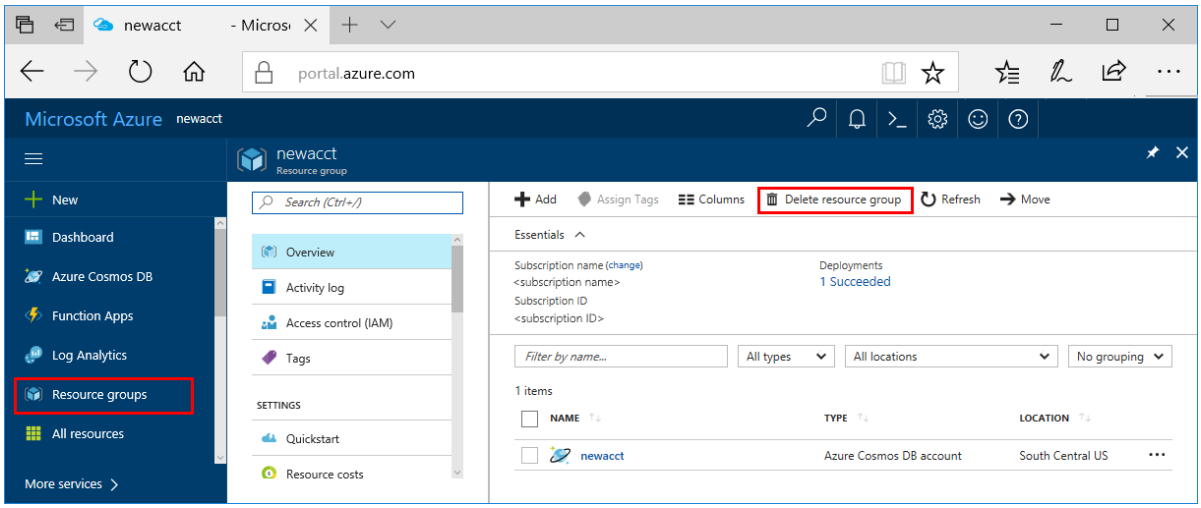
```
var df =
spark.read.format("com.databricks.spark.csv").option("inferSchema","true").option("header","true"
).load("wasb://modeldata@<storageaccount>.blob.core.windows.net/<subscription_id>/<resource_group
_name>/<model_management_account_name>/<webservice_name>/<model_id>-<model_name>-
<model_version>/<identifier>/<year>/<month>/<date>/*")
```

Clean up resources

If you're not going to continue to use this app, delete all resources created by this quickstart with the following steps so you don't incur any charges:

> **NOTE**
>
> These resources are useful when following the other Azure Machine Learning services tutorials now.

1. In the Azure portal, select **Resource groups** on the far left.



2. From the list of resource groups, select the resource group you created, and then click **Delete resource group**.
3. Type the name of the resource group to delete, and then click **Delete**.

Next steps

In this third part of the three-part tutorial series, you have learned how to use Machine Learning to:

- Locate the model file.
- Generate a scoring script and schema file.
- Prepare the environment.
- Create a real-time web service.
- Run the real-time web service.

- Examine the output blob data.

You have successfully run a training script in various compute environments. You have also created, serialized, and operationalized a model through a Docker-based web service.

You are now ready to do advanced data preparation:

Tutorial 4 - Advanced data preparation