Microsoft

Azure Machine Learning Work Bench

Contents

Table of Contents

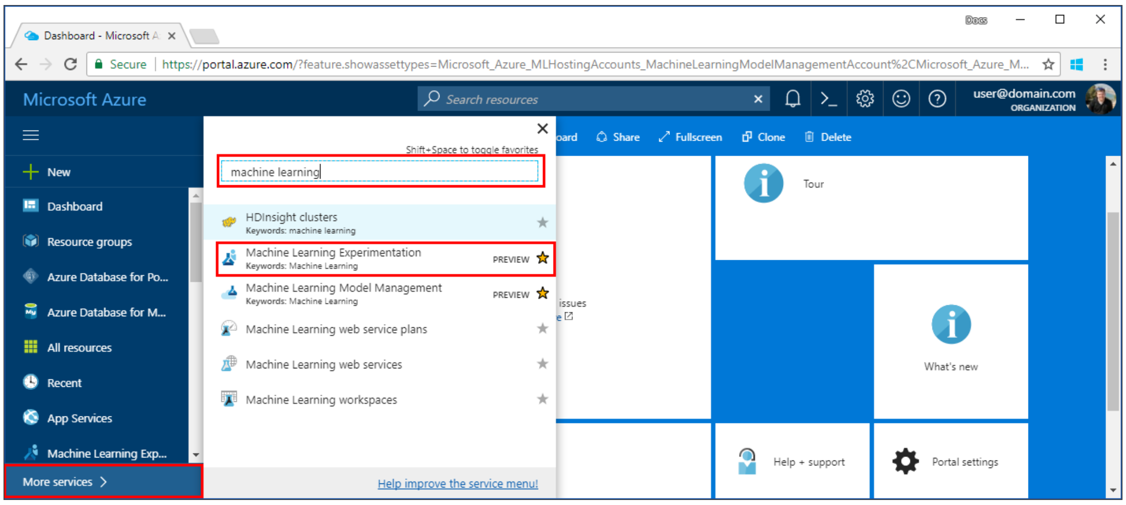
Goals and Requirements

Logistic Regression

Create Azure Machine Learning accounts

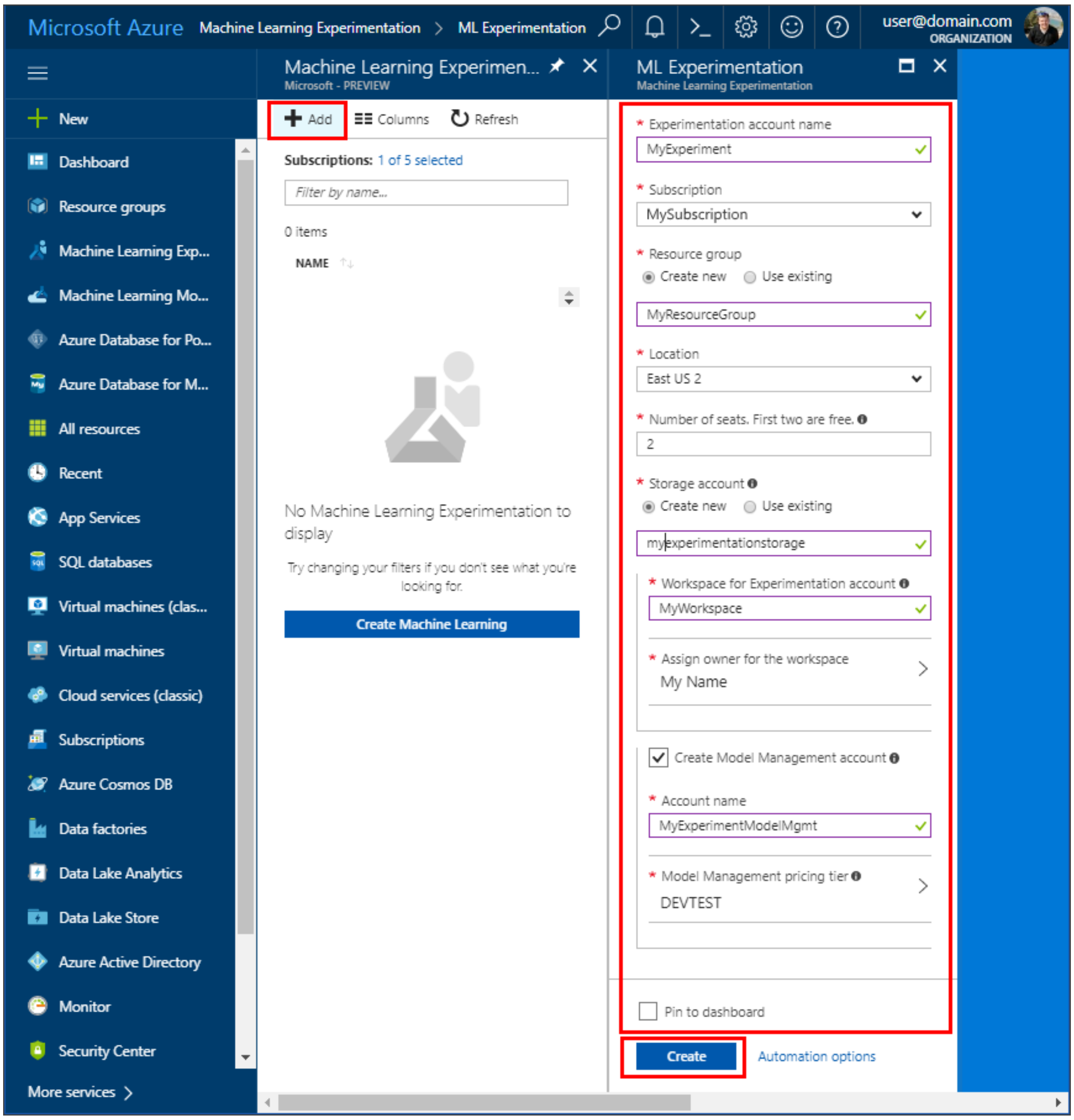
Use the Azure portal to provision Azure Machine Learning accounts:

1. Select the New button (+) in the upper-left corner of the portal.
2. Enter Machine Learning in the search bar. Select the search result named Machine Learning Experimentation (preview). Click the star icon to make this selection a favorite in the Azure portal.



Dashboard - Microsoft A
https://portal.azure.com/?feature.showassettypes=Microsoft_Azure_MLHostingAccounts_MachineLearningModelManagementAccount%2CMicrosoft_Azure_M...
Microsoft Azure
Search resources
user@domain.com
ORGANIZATION
New
Dashboard
Resource groups
Azure Database for Po...
Azure Database for M...
All resources
Recent
App Services
Machine Learning Exp...
More services
Shift+Space to toggle favorites
machine learning
HDInsight clusters
Keywords: machine learning
Machine Learning Experimentation
Keywords: Machine Learning
PREVIEW
Machine Learning Model Management
Keywords: Machine Learning
PREVIEW
Machine Learning web service plans
Machine Learning web services
Machine Learning workspaces
Help improve the service menu!
Share
Fullscreen
Clone
Delete
Tour
What's new
Help + support
Portal settings

3. Select + Add to configure a new Machine Learning Experimentation account. The detailed form opens.



Subscriptions: 1 of 5 selected

0 items

No Machine Learning Experimentation to display

Try changing your filters if you don't see what you're looking for.

* Experimentation account name: MyExperiment

* Subscription: MySubscription

* Resource group: [✓ Create new] Use existing — MyResourceGroup

* Location: East US 2

* Number of seats. First two are free.: 2

* Storage account: [✓ Create new] Use existing — myexperimentationstorage

* Workspace for Experimentation account: MyWorkspace

* Assign owner for the workspace: My Name

☑ Create Model Management account

* Account name: MyExperimentModelMgmt

* Model Management pricing tier: DEVTEST

☐ Pin to dashboard

4. Fill out the Machine Learning Experimentation form with the following information:

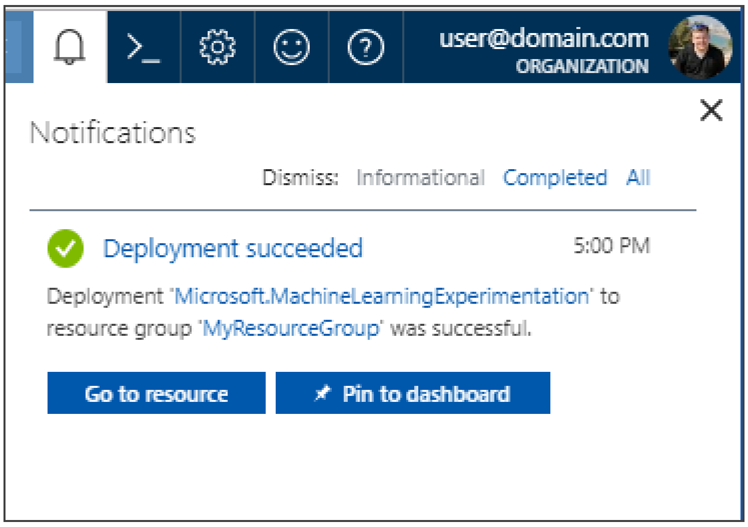
Setting	Suggested value	Description
Experimentation account name	*Unique name*	Choose a unique name that identifies your account. You can use your own name, or a departmental or project name that best identifies the experiment. The name should be 2 to 32 characters. It should include only alphanumeric characters and the dash (-) character.

Setting	Suggested value	Description
Subscription	*Your subscription*	Choose the Azure subscription that you want to use for your experiment. If you have multiple subscriptions, choose the appropriate subscription in which the resource is billed.
Resource group	*Your resource group*	You can make a new resource group name, or you can use an existing one from your subscription.
Location	*The region closest to your users*	Choose the location that's closest to your users and the data resources.
Number of seats	2	Enter the number of seats. This selection affects the pricing. The first two seats are free. Use two seats for the purposes of this Quickstart. You can update the number of seats later as needed in the Azure portal.
Storage account	*Unique name*	Select Create new and provide a name to create an Azure storage account. Or, select Use existing and select your existing storage account from the drop-down list. The storage account is required and is used to hold project artifacts and run history data.
Workspace for Experimentation account	*Unique name*	Provide a name for the new workspace. The name should be 2 to 32 characters. It should include only alphanumeric characters and the dash (-) character.
Assign owner for the workspace	*Your account*	Select your own account as the workspace owner.
Create Model Management account	*check*	As part of the Experimentation account creation experience, you have the option of also creating the Machine Learning Model Management account. This resource is used when you're ready to deploy and manage your models as real-time web services. We recommend creating the Model Management account at the same time as the Experimentation account.
Account name	*Unique name*	Choose a unique name that identifies your Model Management account. You can use your own name, or a departmental or project name that best identifies

Setting	Suggested value	Description
		the experiment. The name should be 2 to 32 characters. It should include only alphanumeric characters and the dash (-) character.
Model Management pricing tier	DEVTEST	Select No pricing tier selected to specify the pricing tier for your new Model Management account. For cost savings, select the DEVTEST pricing tier if it's available on your subscription (limited availability). Otherwise, select the S1 pricing tier for cost savings. Click Select to save the pricing tier selection.
Pin to dashboard	*check*	Select the Pin to dashboard option to allow easy tracking of your Machine Learning Experimentation account on the front dashboard page of the Azure portal.

5. Select Create to begin the creation process.
6. On the Azure portal toolbar, click Notifications (bell icon) to monitor the deployment process.

   The notification shows Deployment in progress. The status changes to Deployment succeeded when it's done. Your Machine Learning Experimentation account page opens upon success.

   Notifications

   Dismiss: Informational Completed All

   Deployment succeeded 5:00 PM

   Deployment 'Microsoft.MachineLearningExperimentation' to resource group 'MyResourceGroup' was successful.

Now, depending on which operating system you use on your local computer, follow one of the next two sections to install Azure Machine Learning Workbench.

Install Azure Machine Learning Workbench on Windows

Install Azure Machine Learning Workbench on your computer running Windows 10, Windows Server 2016, or newer.

1. Download the latest Azure Machine Learning Workbench installerAmlWorkbenchSetup.msi.
2. Double-click the downloaded installer AmlWorkbenchSetup.msi from File Explorer.

   Important

   Download the installer fully on disk, and then run it from there. Do not run it directly from your browser's download widget.

3. Finish the installation by following the on-screen instructions.

   The installer downloads all the necessary dependent components, such as Python, Miniconda, and other related libraries. The installation might take around half an hour to finish all the components.

4. Azure Machine Learning Workbench is now installed in the following directory:

   ```
   C:\Users\<user>\AppData\Local\AmlWorkbench
   ```

Run Azure Machine Learning Workbench to sign in for the first time

1. After the installation process is complete, select the Launch Workbench button on the last screen of the installer. If you have closed the installer, find the shortcut to Machine Learning Workbench on your desktop and Start menu named Azure Machine Learning Workbench to start the app.
2. Sign in to Workbench by using the same account that you used earlier to provision your Azure resources.
3. When the sign-in process has succeeded, Workbench attempts to find the Machine Learning Experimentation accounts that you created earlier. It searches for all Azure subscriptions to which your credential has access. When at least one Experimentation account is found, Workbench opens with that account. It then lists the workspaces and projects found in that account.

   Tip

   If you have access to more than one Experimentation account, you can switch to a different one by selecting the avatar icon in the lower-left corner of the Workbench app.

Create a new project

1. Start the Azure Machine Learning Workbench app and sign in.
2. Select File > New Project (or select the + sign in the PROJECTS pane).
3. Fill in the Project name and Project directory boxes. Project description is optional but helpful. Leave the Visualstudio.com GIT Repository URL box blank for now. Choose a workspace, and select Classifying Iris as the project template.

   Tip

   Optionally, you can fill in the Git repo text box with the URL of a Git repo that is hosted in a Visual Studio Team Services project. This Git repo must already exist, and it must be empty with no master branch. And you must have write access to it. Adding a Git repo now lets you enable roaming and sharing scenarios later. Read more.

4. Select the Create button to create the project. A new project is created and opened for you. At this point, you can explore the project home page, data sources, notebooks, and source code files.

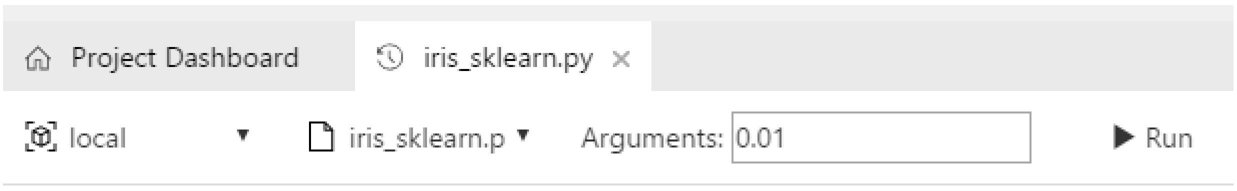
   Tip

   You can also open the project in Visual Studio Code or other editors simply by configuring an integrated development environment (IDE) link, and then opening the project directory in it. Read more.

Run a Python script

Let's run a script on your local computer.

1. Each project opens to its own Project Dashboard page. Select local as the execution target from the command bar near the top of the application, and select iris_sklearn.pyas the script to run. There are other files included in the sample that you can check out later.



Arguments: 0.01

2. In the Arguments text box, enter 0.01. This number is used in the code to set the regularization rate. It's a value that's used to configure how the linear regression model is trained.
3. Select the Run button to begin running iris_sklearn.py on your computer.

This code uses the logistic regression algorithm from the popular Python scikit-learnlibrary to build the model.

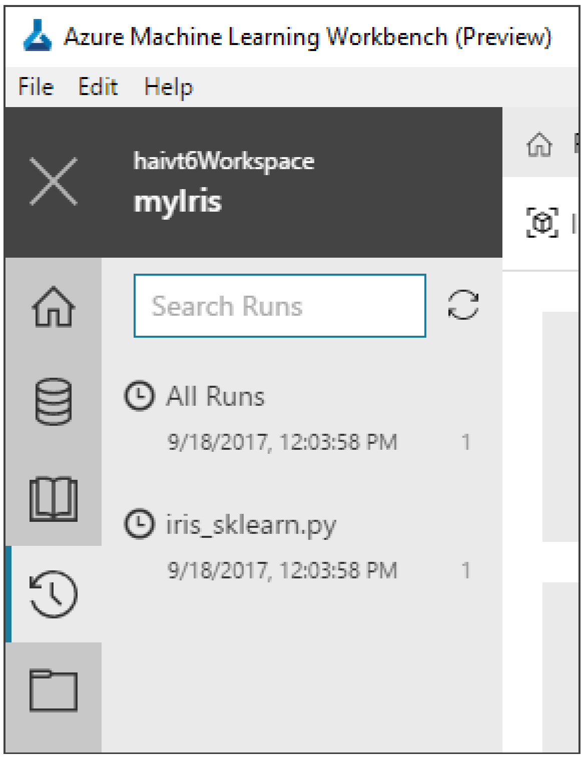
4. The Jobs panel slides out from the right if it is not already visible, and an iris_sklearn job is added in the panel. Its status transitions from Submitting to Running as the job begins to run, and then to Completed in a few seconds.

   Congratulations. You have successfully executed a Python script in Azure Machine Learning Workbench.

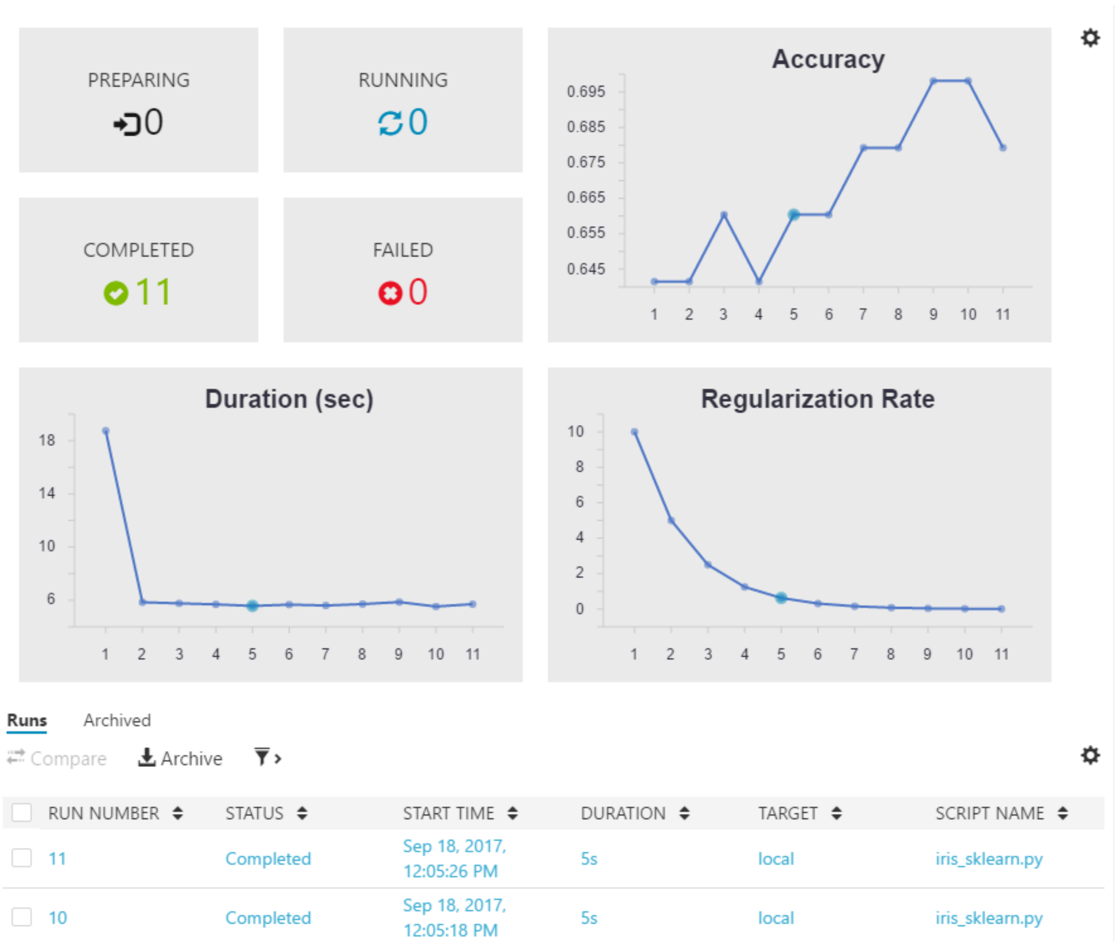
5. Repeat steps 2 to 4 several times. Each time, use different argument values that range from 10 to 0.001.

View run history

1. Go to the Runs view, and select iris_sklearn.py in the run list. The run history dashboard for iris_sklearn.py opens. It shows every run that was executed on iris_sklearn.py.



2. The run history dashboard also displays the top metrics, a set of default graphs, and a list of metrics for each run. You can customize this view by sorting, filtering, and adjusting the configurations. Just select the configuration icon or the filter icon.



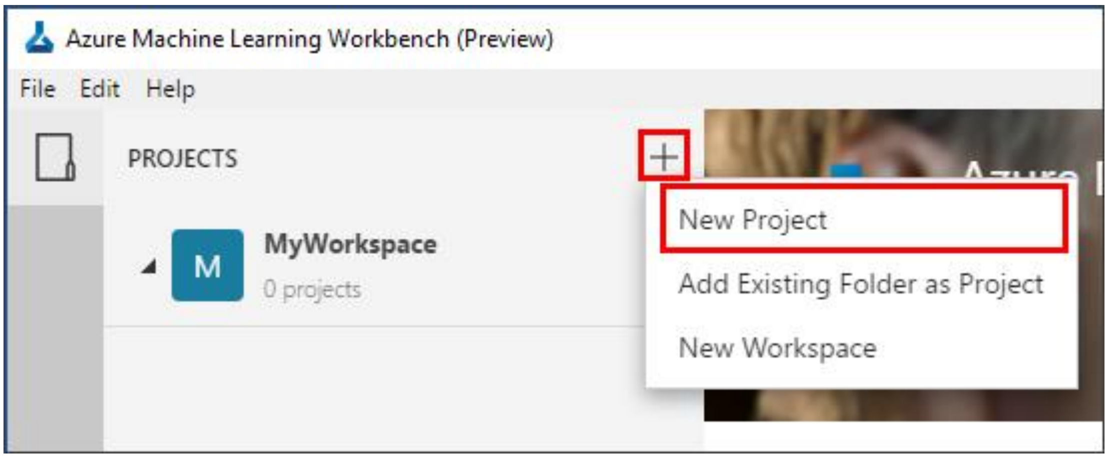
RUN NUMBER	STATUS	START TIME	DURATION	TARGET	SCRIPT NAME
11	Completed	Sep 18, 2017, 12:05:26 PM	5s	local	iris_sklearn.py
10	Completed	Sep 18, 2017, 12:05:18 PM	5s	local	iris_sklearn.py

3. Select a completed run, and you can see a detailed view for that specific execution. Details include additional metrics, the files that it produced, and other potentially useful logs.

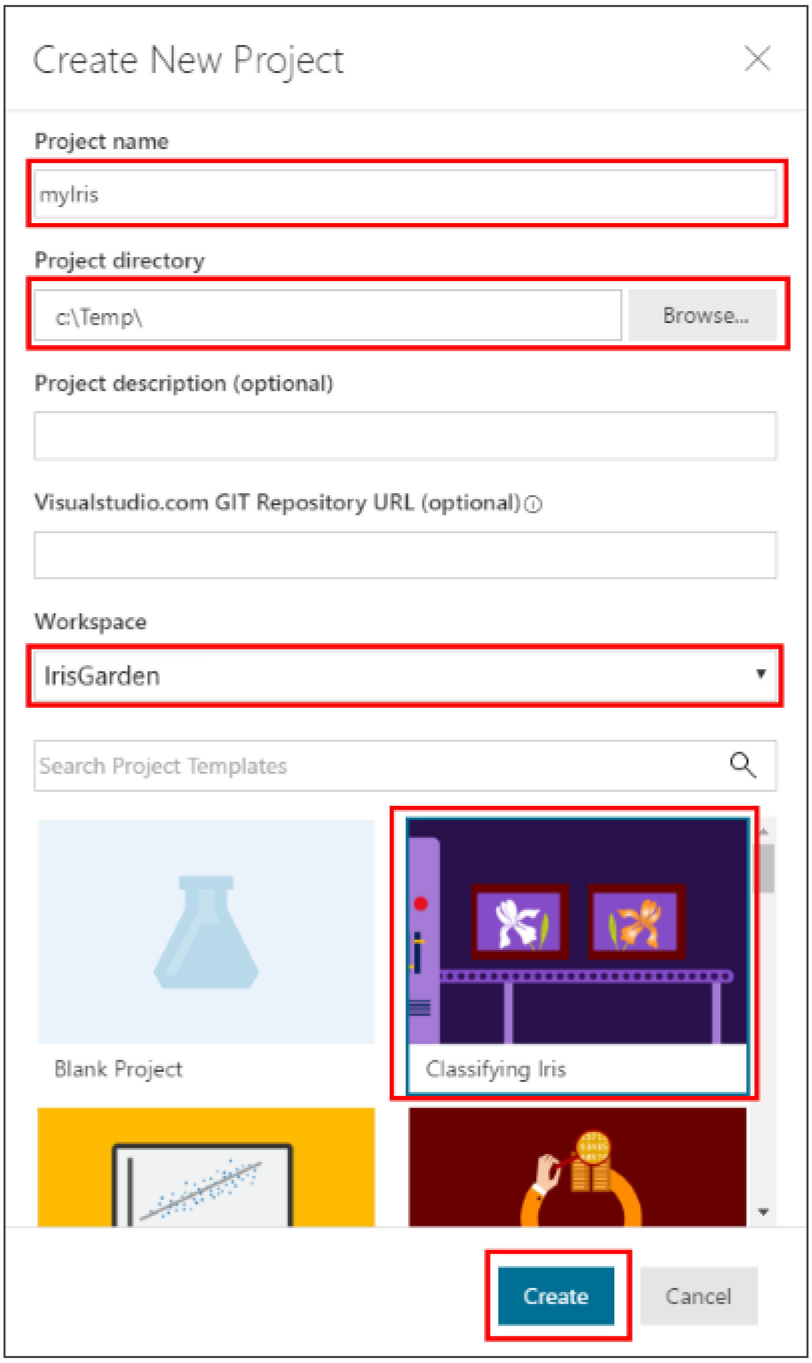
Classify Iris part 1: Prepare the data

Create a new project in Azure Machine Learning Workbench

1. Open the Azure Machine Learning Workbench app, and log in if needed. In the PROJECTS pane, select the plus sign (+) to create a New Project.


Azure Machine Learning Workbench (Preview)
File Edit Help
PROJECTS
MyWorkspace
0 projects
New Project
Add Existing Folder as Project
New Workspace

2. Fill in the Create New Project details:


Create New Project

Project name

myIris

Project directory

c:\Temp\

Project description (optional)

Visualstudio.com GIT Repository URL (optional)

Workspace

IrisGarden

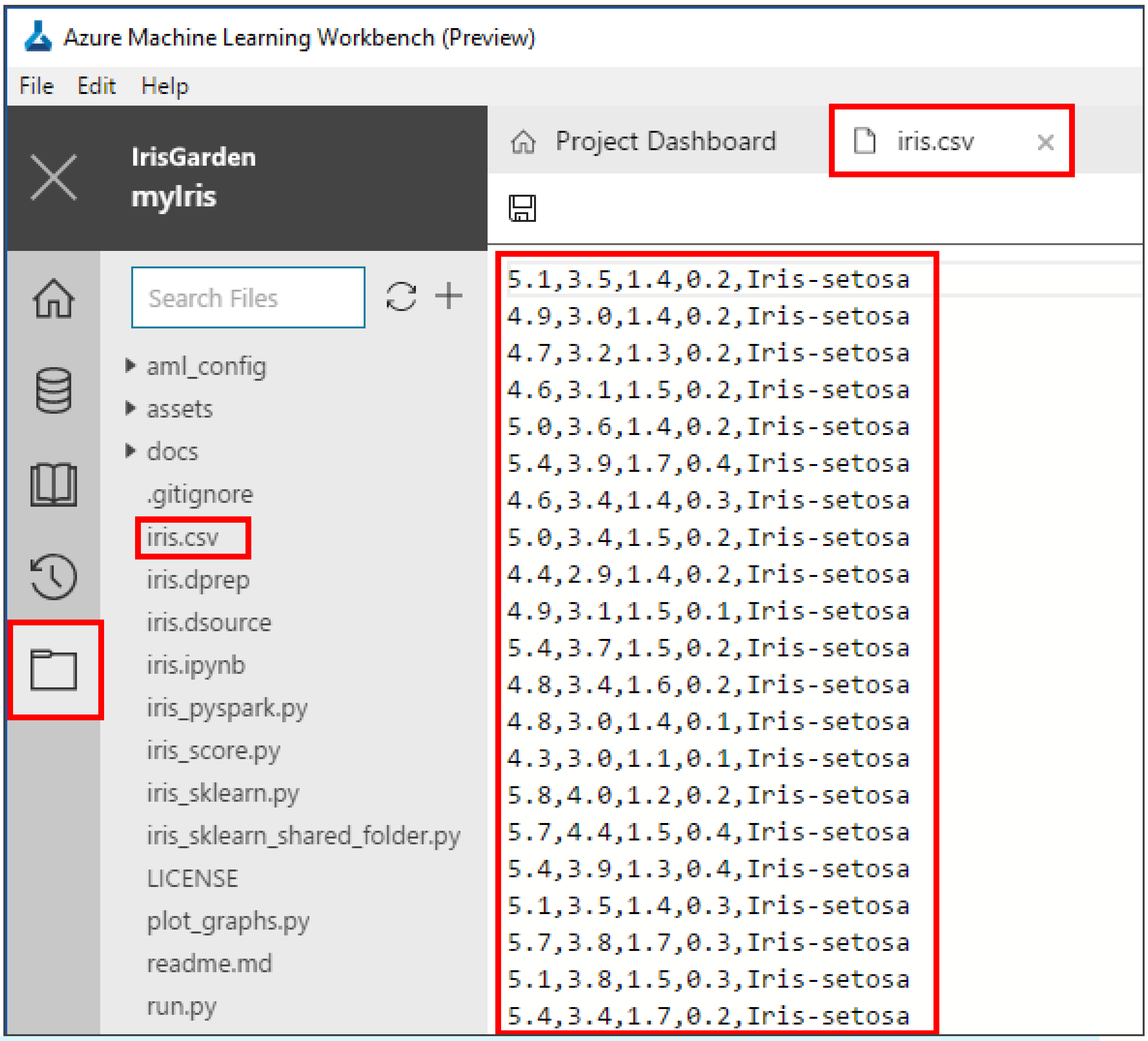
- Fill in the Project name box with a name for the project. For example, use the value myIris.
- Select the Project directory in which the project is created. For example, use the value `C:\Temp\`.
- Enter the Project description, which is optional.
- The Git Repository field is also optional and can be left blank. You can provide an existing empty Git repo (a repo with no master branch) on Visual Studio Team Services. If you use a Git repo that already exists, you can enable the roaming and sharing scenarios later. For more information, see [Use Git repo](#).

- Select a Workspace, for example, this tutorial uses IrisGarden.
- Select the Classifying Iris template from the project template list.

3. Select the Create button. The project is now created and opened for you.

Create a data preparation package

1. Open the iris.csv file from the File View. The file is a table with 5 columns and 150 rows. It has four numerical feature columns and a string target column. It does not have column headers.

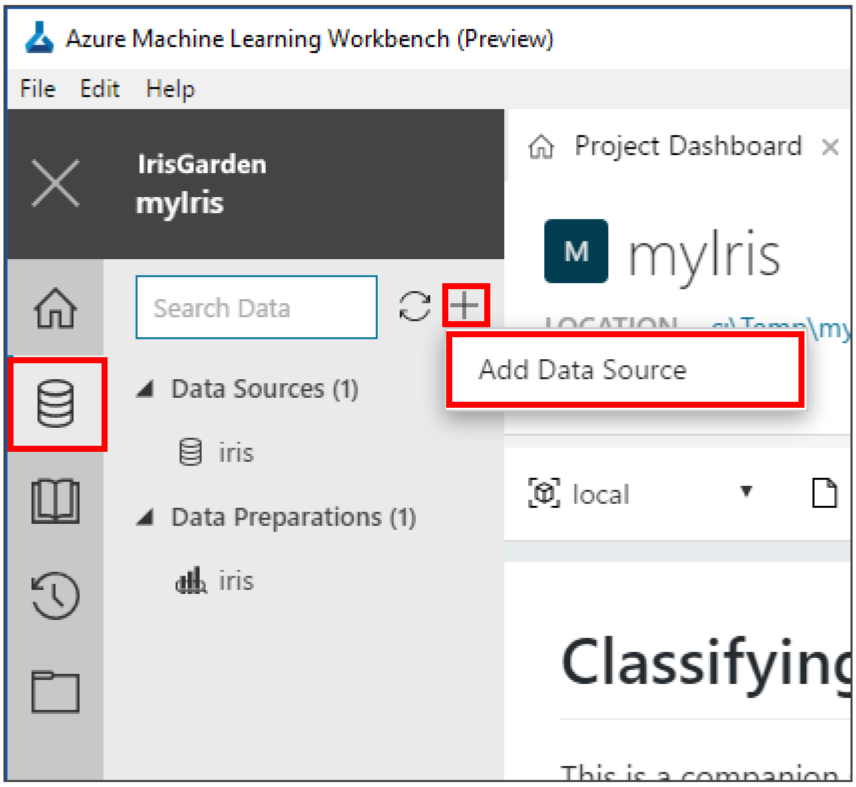


```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
5.4,3.7,1.5,0.2,Iris-setosa
4.8,3.4,1.6,0.2,Iris-setosa
4.8,3.0,1.4,0.1,Iris-setosa
4.3,3.0,1.1,0.1,Iris-setosa
5.8,4.0,1.2,0.2,Iris-setosa
5.7,4.4,1.5,0.4,Iris-setosa
5.4,3.9,1.3,0.4,Iris-setosa
5.1,3.5,1.4,0.3,Iris-setosa
5.7,3.8,1.7,0.3,Iris-setosa
5.1,3.8,1.5,0.3,Iris-setosa
5.4,3.4,1.7,0.2,Iris-setosa
```

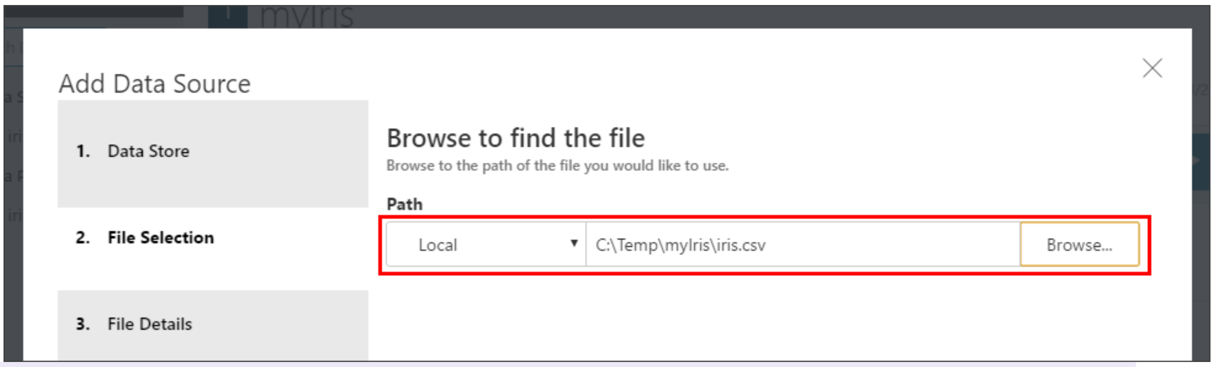
Note

Do not include data files in your project folder, particularly when the file size is large. We include iris.csv in this template for demonstration purposes because it's tiny. 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. Leave the default values, and then select the Next button.


Add Data Source

1. Data Store
2. File Selection
3. File Details

Browse to find the file

Browse to the path of the file you would like to use.

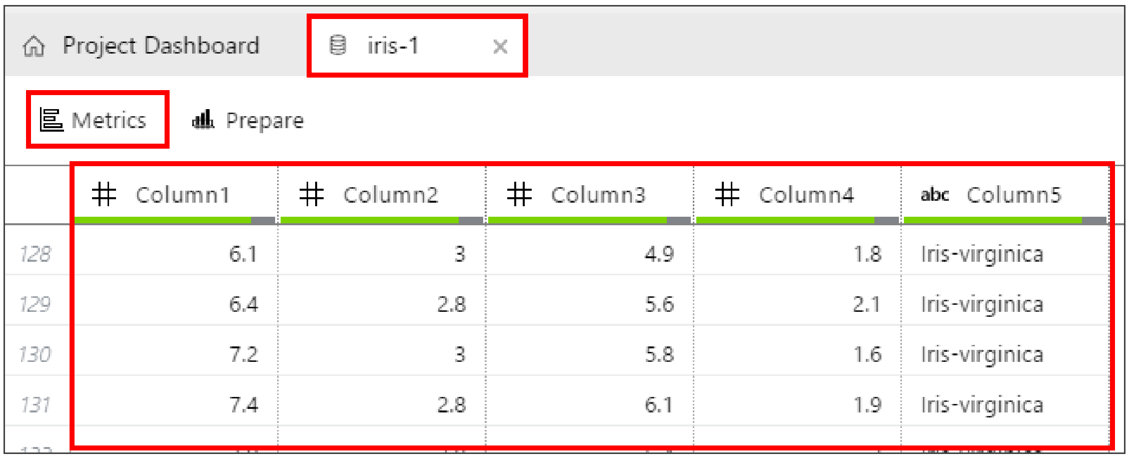
Path: Local | C:\Temp\myIris\iris.csv | Browse...

Important

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

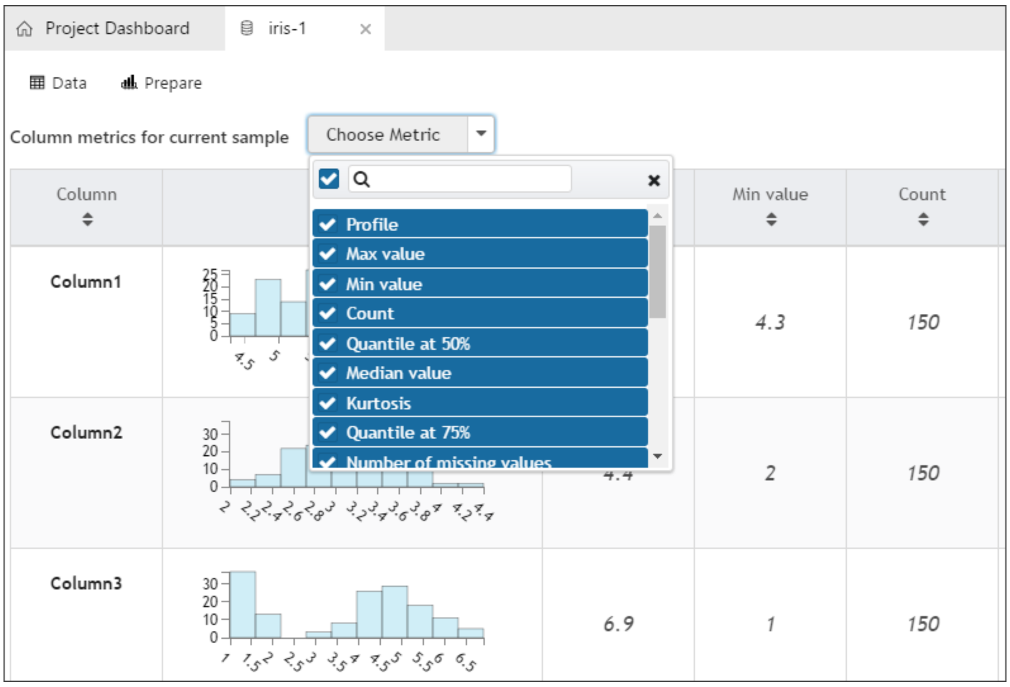
4. After selecting the file, select the Finish button.
5. A new file named iris-1.dsource is created. The file is named uniquely with a dash-1, because the sample project already comes with an unnumbered iris.dsource file.

   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
128	6.1	3	4.9	1.8	Iris-virginica
129	6.4	2.8	5.6	2.1	Iris-virginica
130	7.2	3	5.8	1.6	Iris-virginica
131	7.4	2.8	6.1	1.9	Iris-virginica

6. Select the Metrics button. Observe the histograms. A complete set of statistics has been calculated for each column. You can also select the Data button to see the data again.

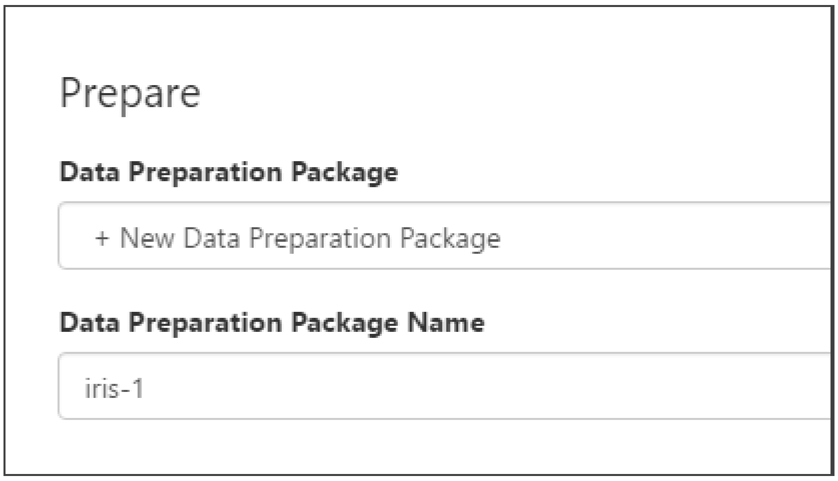


Column			Min value	Count
Column1			4.3	150
Column2		4.4	2	150
Column3		6.9	1	150

7. Select the Prepare button. The Prepare dialog box opens.

   The sample project comes with an iris.dprep file. By default, it asks you to create a new data flow in the iris.dprep data preparation package that already exists.

   Select + New Data Preparation Package from the drop-down menu, enter a new value for the package name, use iris-1, and then select OK.



Prepare

Data Preparation Package

+ New Data Preparation Package

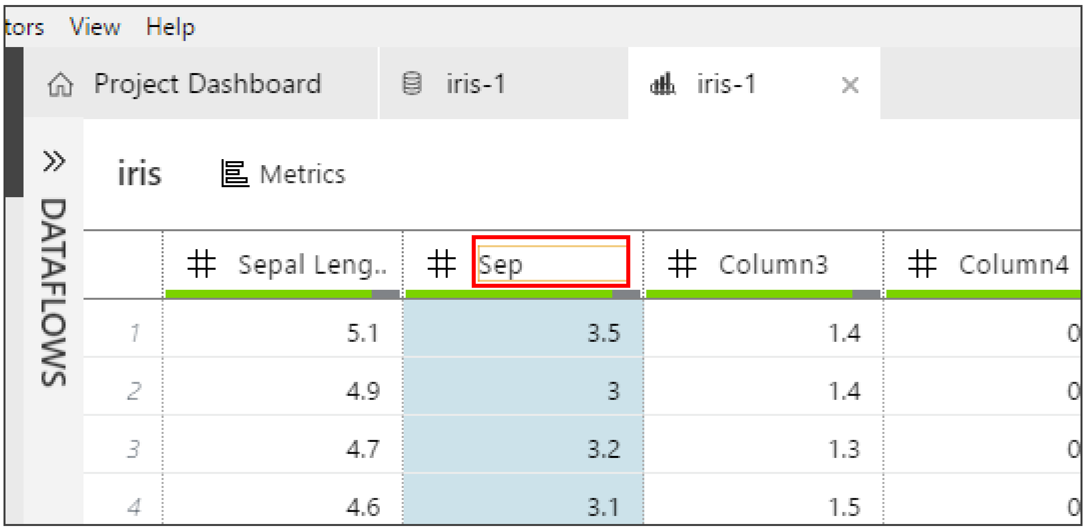
Data Preparation Package Name

iris-1

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

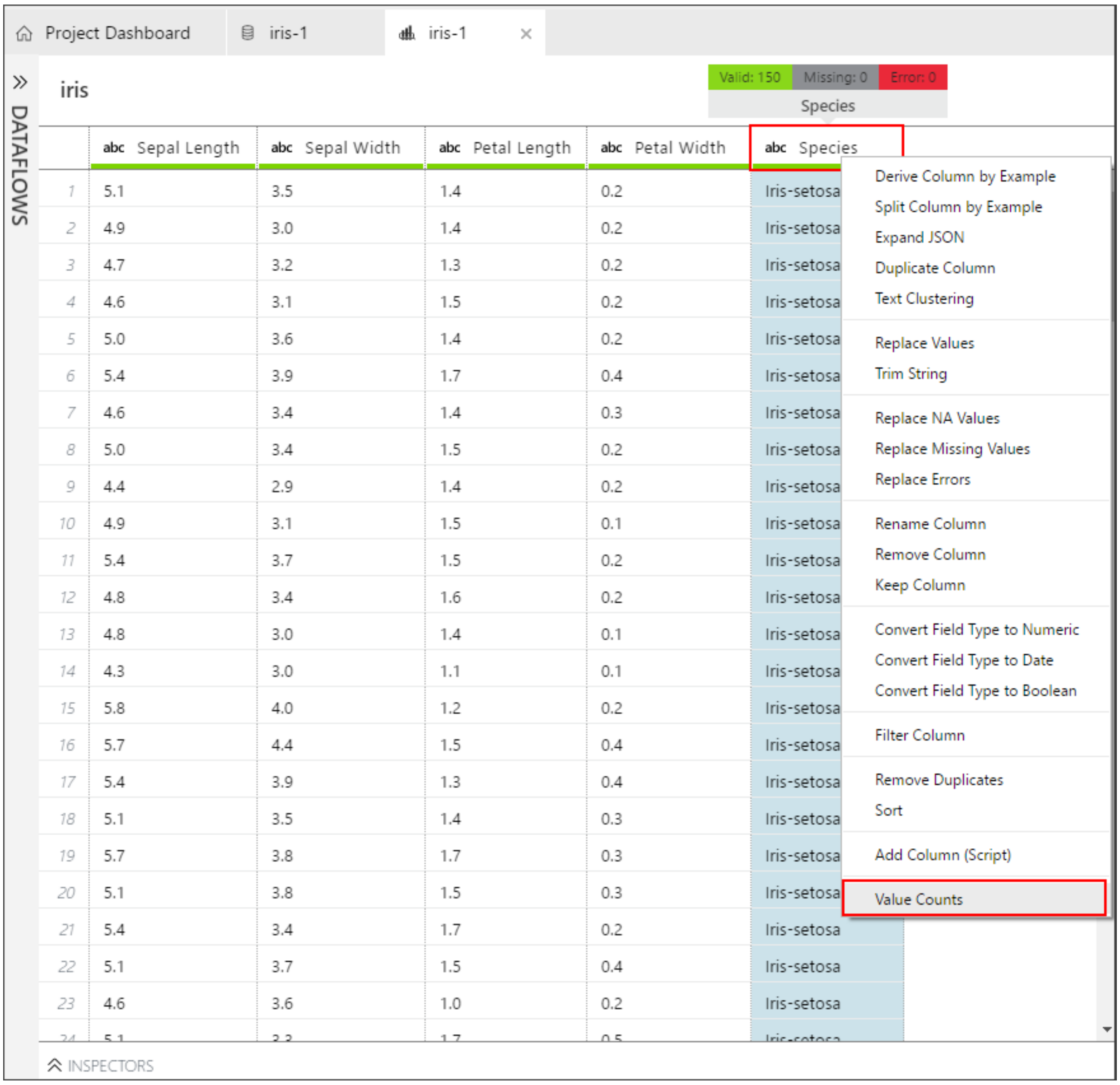
8. Now let's do some basic data preparation. Rename the column names. Select each column header to make the header text editable.

   Enter Sepal Length, Sepal Width, Petal Length, Petal Width, and Species for the five columns respectively.



	Sepal Leng..	Sep	Column3	Column4
1	5.1	3.5	1.4	0
2	4.9	3	1.4	0
3	4.7	3.2	1.3	0
4	4.6	3.1	1.5	0

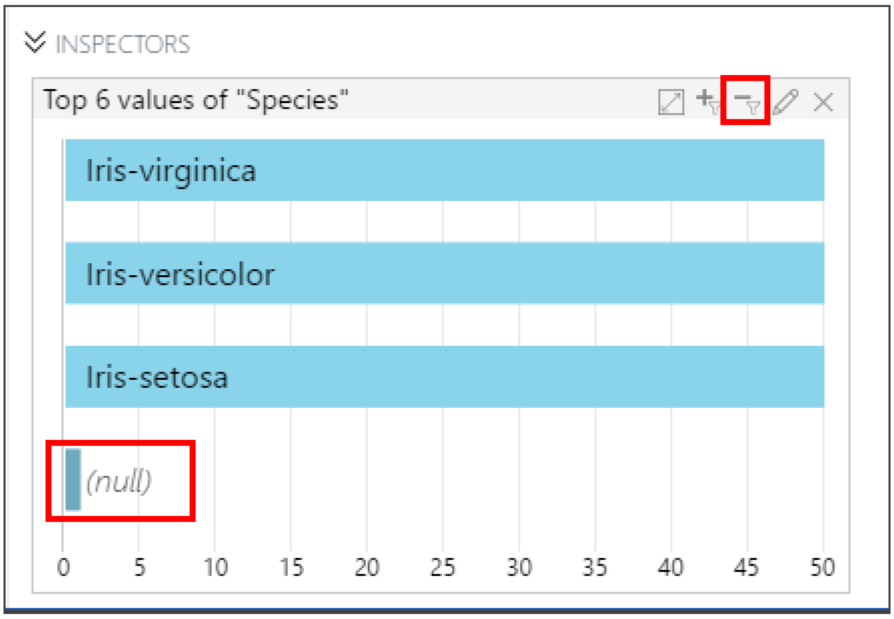
9. To count distinct values, select the Species column, and then right-click to select it. Select Value Counts from the drop-down menu.



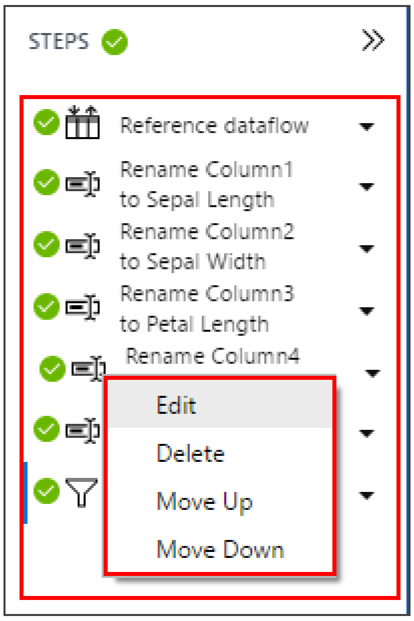
	abc Sepal Length	abc Sepal Width	abc Petal Length	abc Petal Width	abc Species
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3.0	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	5.0	3.6	1.4	0.2	Iris-setosa
6	5.4	3.9	1.7	0.4	Iris-setosa
7	4.6	3.4	1.4	0.3	Iris-setosa
8	5.0	3.4	1.5	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
11	5.4	3.7	1.5	0.2	Iris-setosa
12	4.8	3.4	1.6	0.2	Iris-setosa
13	4.8	3.0	1.4	0.1	Iris-setosa
14	4.3	3.0	1.1	0.1	Iris-setosa
15	5.8	4.0	1.2	0.2	Iris-setosa
16	5.7	4.4	1.5	0.4	Iris-setosa
17	5.4	3.9	1.3	0.4	Iris-setosa
18	5.1	3.5	1.4	0.3	Iris-setosa
19	5.7	3.8	1.7	0.3	Iris-setosa
20	5.1	3.8	1.5	0.3	Iris-setosa
21	5.4	3.4	1.7	0.2	Iris-setosa
22	5.1	3.7	1.5	0.4	Iris-setosa
23	4.6	3.6	1.0	0.2	Iris-setosa

This action opens the Inspectors pane, and displays a histogram with four bars. The target column has three distinct values: Iris_virginica, Iris_versicolor, Iris-setosa, and a (null) value.

10. To filter out nulls, select the bar from the graph that represents the null value. There is one row with a (null) value. To remove this row, select the minus sign (-).



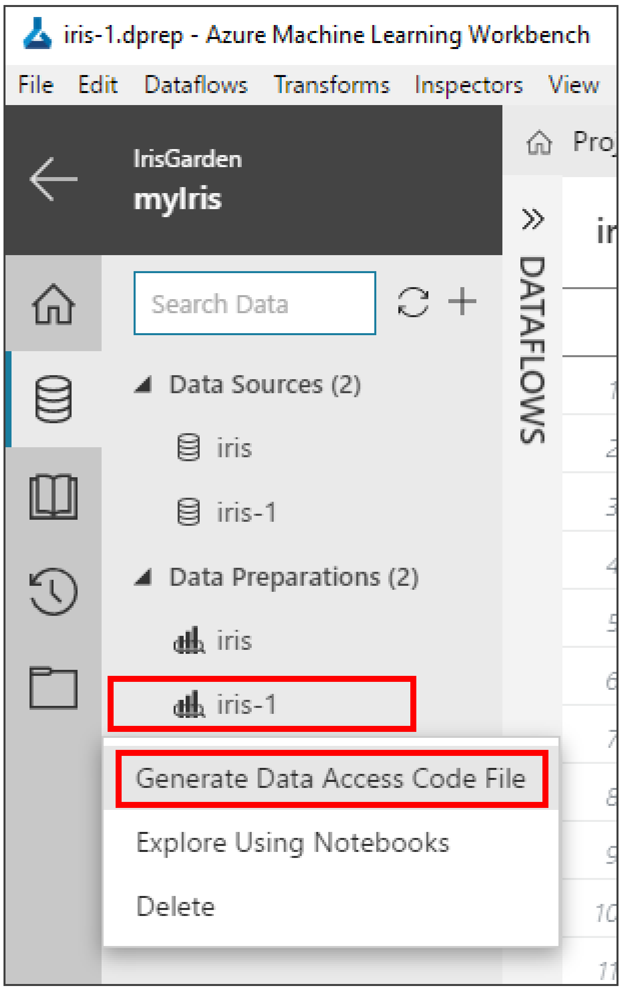
11. Notice the individual steps detailed in the STEPS pane. As you renamed the columns and filtered the null value rows, each action was recorded as a data-preparation step. You can edit individual steps to adjust the settings, reorder the steps, and remove steps.


STEPS
Reference dataflow
Rename Column1 to Sepal Length
Rename Column2 to Sepal Width
Rename Column3 to Petal Length
Rename Column4
Edit
Delete
Move Up
Move Down

12. Close the data preparation editor. Select Close (x) on the iris-1 tab with the graph icon. 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

1. Right-click the iris-1.dprep file to bring up the context menu, and then select Generate Data Access Code File.



2. A new file named iris-1.py opens with the following lines of code:

PythonCopy

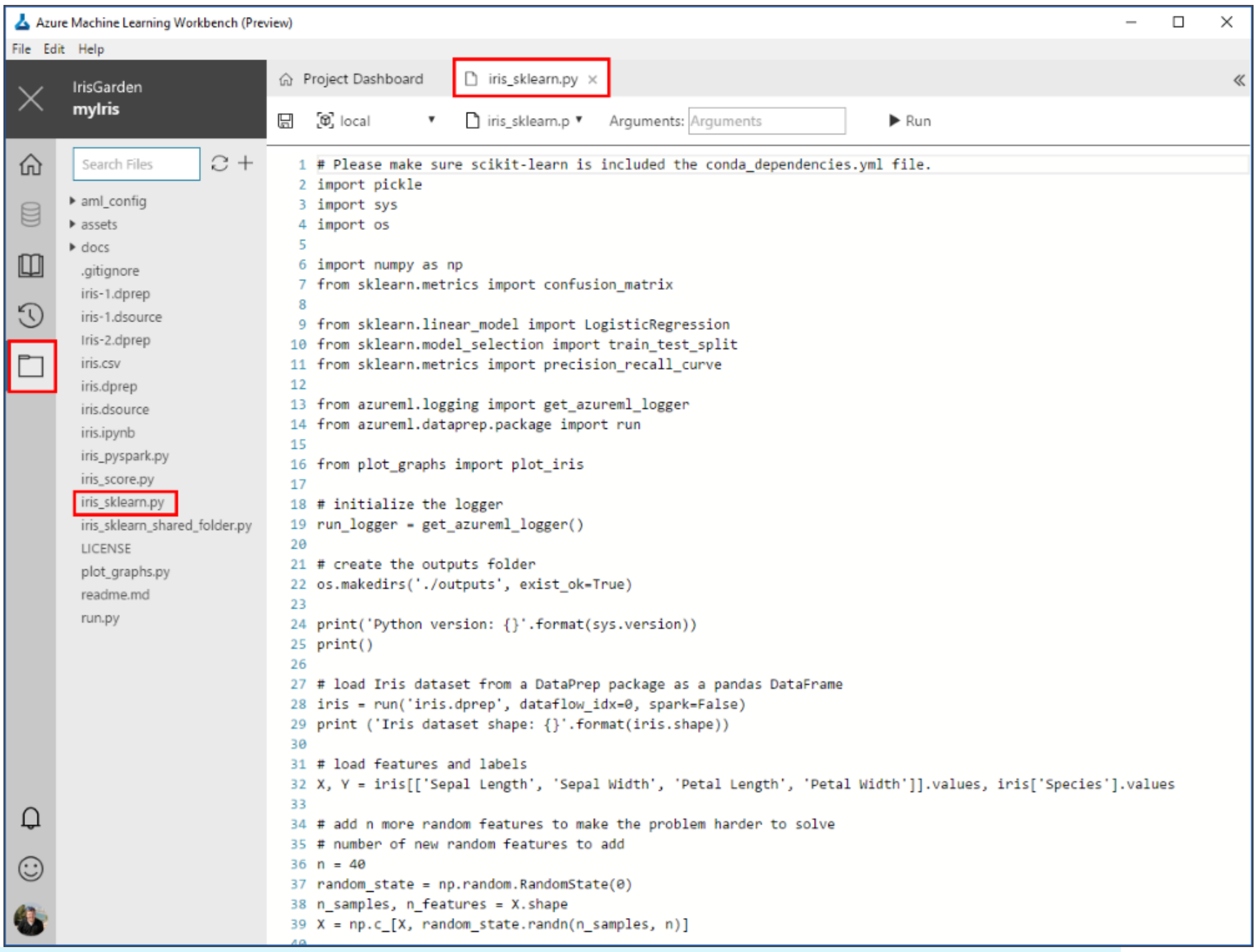
```python
# Use the Azure Machine Learning data preparation package
from azureml.dataprep import package

# 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)
```

This code snippet invokes the logic you created as a data preparation package. Depending on the context in which this code is run, `df` represents the various kinds of dataframes. A pandas DataFrame is used when executed in Python runtime, or a Spark DataFrame is used when executed in a Spark context.

Review iris_sklearn.py and the configuration files

1. Open the Azure Machine Learning Workbench application, and open the myIris project you created in the previous part of the tutorial series.
2. After the project is open, select the Files button (the folder icon) on the far-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.



```python
# Please make sure scikit-learn is included the conda_dependencies.yml file.
import pickle
import sys
import os

import numpy as np
from sklearn.metrics import confusion_matrix

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_recall_curve

from azureml.logging import get_azureml_logger
from azureml.dataprep.package import run

from plot_graphs import plot_iris

# initialize the logger
run_logger = get_azureml_logger()

# create the outputs folder
os.makedirs('./outputs', exist_ok=True)

print('Python version: {}'.format(sys.version))
print()

# load Iris dataset from a DataPrep package as a pandas DataFrame
iris = run('iris.dprep', dataflow_idx=0, spark=False)
print ('Iris dataset shape: {}'.format(iris.shape))

# load features and labels
X, Y = iris[['Sepal Length', 'Sepal Width', 'Petal Length', 'Petal Width']].values, iris['Species'].values

# add n more random features to make the problem harder to solve
# number of new random features to add
n = 40
random_state = np.random.RandomState(0)
n_samples, n_features = X.shape
X = np.c_[X, random_state.randn(n_samples, n)]
```

Note

4. Review the Python script code to become familiar with the coding style. The script performs the following tasks:
   - Loads the data preparation package iris.dprep to create a pandas DataFrame.

     Note

     Use the `iris.dprep` data preparation package that comes with the sample project, which should be the same as the `iris-1.dprep` file you built in part 1 of this tutorial.

   - Adds random features to make the problem more difficult to solve. Randomness is necessary because Iris is a small data set that's easily classified with nearly 100% accuracy.
   - Uses the scikit-learn machine learning library to build a logistic regression model.
   - Serializes the model by inserting the pickle library into a file in the `outputs` folder. The script then loads it and 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, by using the matplotlib library, and then saves them in the `outputs` folder.
   - The `run_logger` object is used throughout to record the regularization rate and to model accuracy into the logs. The logs are automatically plotted in the run history.

Execute iris_sklearn.py script in a local environment

Let's prepare to run the iris_sklearn.py script for the first time. This script requires the scikit-learn and matplotlib packages. The scikit-learn package is already installed by Azure Machine Learning Workbench. You still need to install matplotlib.

1. In Azure Machine Learning Workbench, select the File menu, and then select Open Command Prompt to open the command prompt. The command-line interface window is referred to as the *Azure Machine Learning Workbench CLI window*, or *CLI window* for short.
2. In the CLI window, enter the following command to install the matplotlib Python package. It should finish in less than a minute.

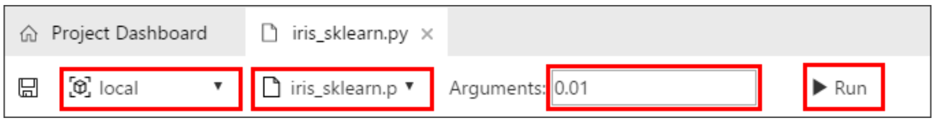
   Azure CLICopy

```
pip install matplotlib
```

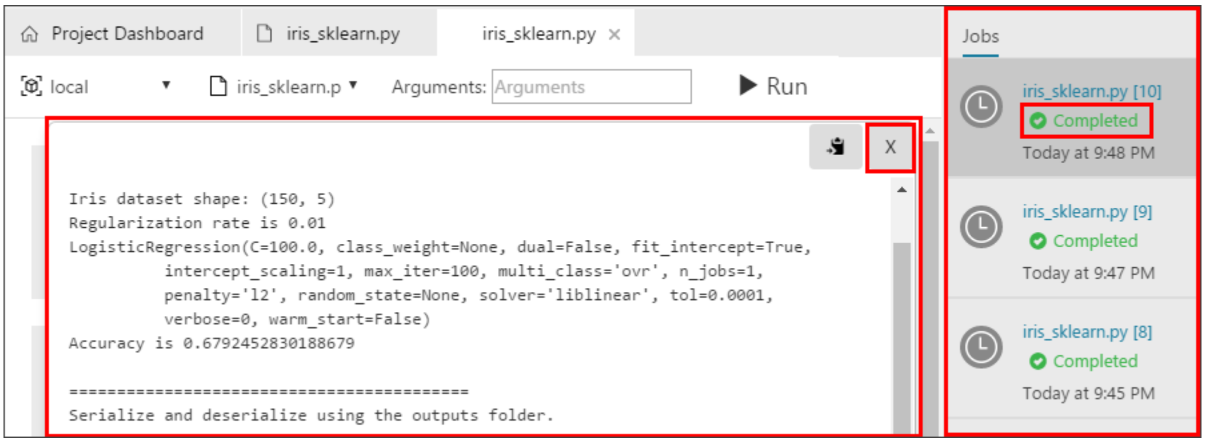
Note

If you skip the previous `pip install` command, the code in `iris_sklearn.py` runs successfully. If you only run `iris_sklearn.py`, the code doesn't produce the confusion matrix output and the multi-class ROC curve plots as shown in the history visualizations.

3. Return to the workbench app window.
4. In the toolbar at the top of the iris_sklearn.py tab, select to open the drop-down menu that is next to the Save icon, and then select Run Configuration. Select local as the execution environment, and then enter `iris_sklearn.py` as the script to run.
5. Next, move to the right side of the toolbar and enter `0.01` in the Arguments field.



6. 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.
7. After a few moments, the status of the job transitions from Submitting, to Running, and then to Completed.



```
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.
```

8. Select Completed in the job status text in the Jobs pane. A pop-up window opens and displays the standard output (stdout) text of the running script. 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 Propertieswindow opens and shows the following information for that particular run:

- Run Properties information
- Outputs files
- Visualizations, if any
- Logs

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

Note

Because we introduced some randomization into the training set earlier, your exact results might vary from the results shown here.

textCopy

```
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.

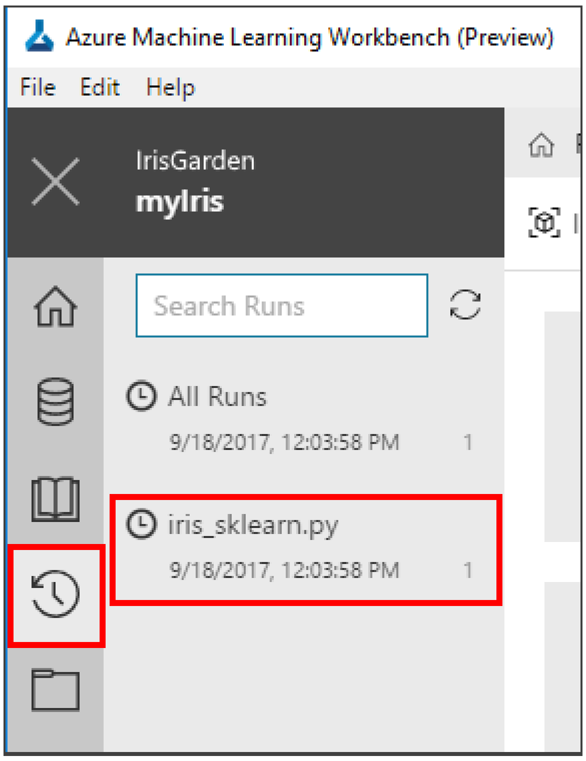
11. Repeat additional runs.

    Enter a series of different numerical 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 algorithm in the code, and that results in different findings each time.

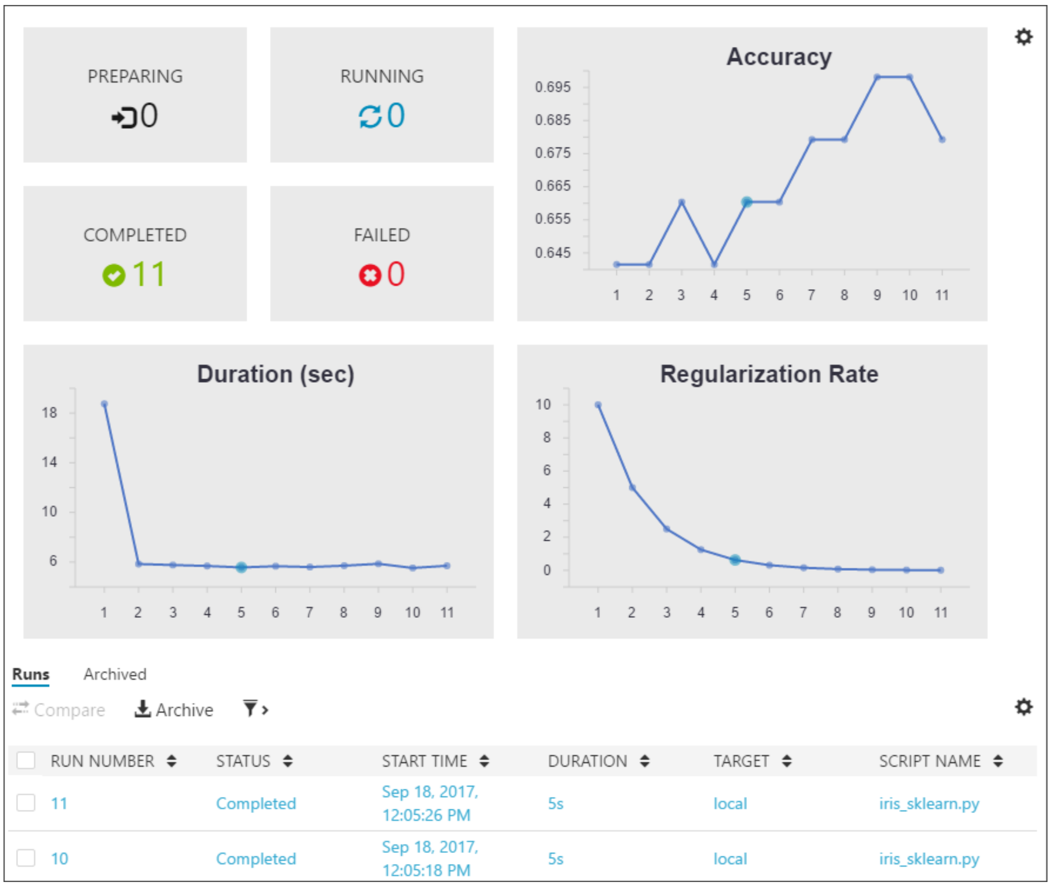
Review the run history in detail

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`.

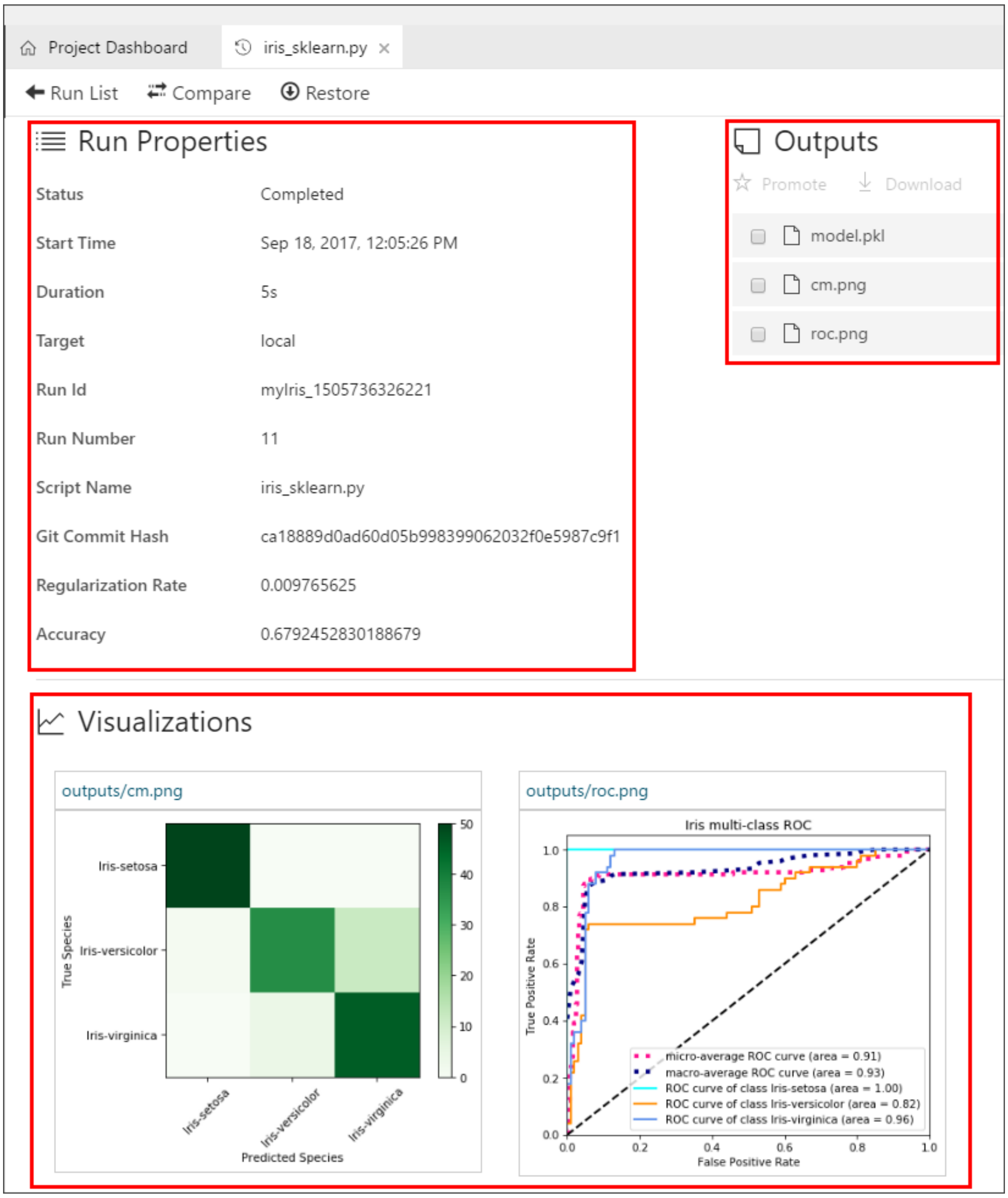


2. The Run Dashboard tab opens. Review the statistics captured across the multiple runs. Graphs render in the top of the tab. Each run has a consecutive number, and the run details are listed in the table at the bottom of the screen.



RUN NUMBER	STATUS	START TIME	DURATION	TARGET	SCRIPT NAME
11	Completed	Sep 18, 2017, 12:05:26 PM	5s	local	iris_sklearn.py
10	Completed	Sep 18, 2017, 12:05:18 PM	5s	local	iris_sklearn.py

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 two or three runs in the Runs table, and select the Compare button to open a detailed comparison pane. Review the side-by-side comparison. Select the Run List back button on the upper left of the Comparison pane to return to the Run Dashboard.
5. 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.



Run Properties

Status	Completed
Start Time	Sep 18, 2017, 12:05:26 PM
Duration	5s
Target	local
Run Id	myIris_1505736326221
Run Number	11
Script Name	iris_sklearn.py
Git Commit Hash	ca18889d0ad60d05b998399062032f0e5987c9f1
Regularization Rate	0.009765625
Accuracy	0.6792452830188679

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.

Execute scripts in the local Docker environment

With Machine Learning, you can easily configure additional execution environments, such as Docker, and run your script in those environments.1

Important

To accomplish this step, you must have a Docker engine locally installed and started. For more information, see the Docker installation instructions.

1. On the left pane, select the Folder icon to open the Files list for your project. Expand the `aml_config` folder.
2. There are several environments that are preconfigured, such as docker-python, docker-spark, and local.

   Each environment has two files, such as `docker.compute` 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.

3. 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.
4. 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, it 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 does 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 result as you do when you target local.

5. Now, let's try Spark. The Docker base image contains a preinstalled and configured Spark instance. Because of this instance, you can execute a PySpark script in it. This is an easy way to develop and test your Spark program, without having to spend the time to install and configure 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`.

6. Do a few more runs and play with different arguments.
7. Open the `iris_spark.py` file to see the logistic regression model built by using the Spark Machine Learning library.
8. Interact with the Jobs pane, run a history list view, and run a details view of your runs across different execution environments.

Execute scripts in the Azure Machine Learning CLI window

1. In Azure Machine Learning Workbench, open the command-line window, select the Filemenu, and then select Open Command Prompt. Your command prompt starts in the project folder with the prompt `C:\Temp\myIris\>`.

   Important

   You must use the command-line window (opened from the workbench) to accomplish the steps that follow.

2. Use the command prompt to log in to Azure.

   The workbench app and CLI use independent credential caches when authenticating against Azure resources. You only need to do this once until the cached token expires. The az account list command returns the list of subscriptions available to your login. If there is more than one, use the ID value from the desired subscription. Set that subscription as the default account to use with the az account set -s command, and then provide the subscription ID value. Then confirm the setting by using the account show command.

Azure CLICopy

```
REM login by using the aka.ms/devicelogin site
az login

REM lists all Azure subscriptions you have access to
az account list -o table

REM sets the current Azure subscription to the one you want to use
az account set -s <subscriptionId>

REM verifies that your current subscription is set correctly
az account show
```

3. After authentication finishes and the current Azure subscription context is set, enter the following commands in the CLI window to install matplotlib, and then submit the Python script as an experiment to run.

Azure CLICopy

```
REM you don't need to run this command if you have installed matplotlib
locally from the previous steps
pip install matplotlib

REM kicks off an execution of the iris_sklearn.py file against the local
compute context
az ml experiment submit -c local .\iris_sklearn.py
```

4. Review the output. You have the same output and results that you had when you used the workbench to run the script.
5. Run the same script again by using the Docker execution environment if you have Docker installed on your machine.

Azure CLICopy

```
REM executes iris_sklearn.py in the local Docker container Python
environment
az ml experiment submit -c docker-python .\iris_sklearn.py 0.01

REM executes iris_spark.py in the local Docker container Spark environment
az ml experiment submit -c docker-spark .\iris_spark.py 0.1
```

6. In the workbench, select the Folder icon on the left pane to list the project files, and open the Python script named run.py.

   This script is useful to loop over various regularization rates. 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`.

   PythonCopy

```python
# run.py
import os

reg = 10
while reg > 0.005:
    os.system('az ml experiment submit -c local ./iris_sklearn.py
{}'.format(reg))
    reg = reg / 2
```

   To open the run.py script from the command line, run the following commands:

   cmdCopy

```
REM submits iris_sklearn.py multiple times with different regularization
rates
python run.py
```

   When `run.py` finishes, you see a graph in your run history list view in the workbench.

Execute in a Docker container on a remote machine

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, that remote 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 if you generate a pair of `.runconfig` and `.compute` files. Use the following command to generate the files. Let's name the new environment `myvm`.

   Azure CLICopy

   ```
   REM creates an myvm compute target
   az ml computetarget attach remotedocker --name myvm --address <IP address> --username <username> --password <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 FQDN to your DSVM and use it here 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.

   Next, run the following command the construct the Docker image in the VM to get it ready to run the scripts:

   Azure CLICopy

   ```
   REM prepares the myvm compute target
   az ml experiment prepare -c myvm
   ```

   Note

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

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

   yamlCopy

   ```
   "Framework": "Python"
   ```

   Note

If you leave the framework setting as PySpark, that should also work. But it's less inefficient 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, except this time target *myvm*:

   Azure CLICopy

```
REM executes 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. You can also do this from the CLI window if you're comfortable with basic file-manipulation commands. 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`:

   yamlCopy

```
"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 it in the Spark instance in the remote Docker container:

   azureliCopy

```
REM executes iris_spark.py in a Spark instance on a remote Docker container
az ml experiment submit -c myvm-spark .\iris_spark.py
```

Download the model pickle file

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

1. Open the Machine Learning Workbench application, and 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 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.

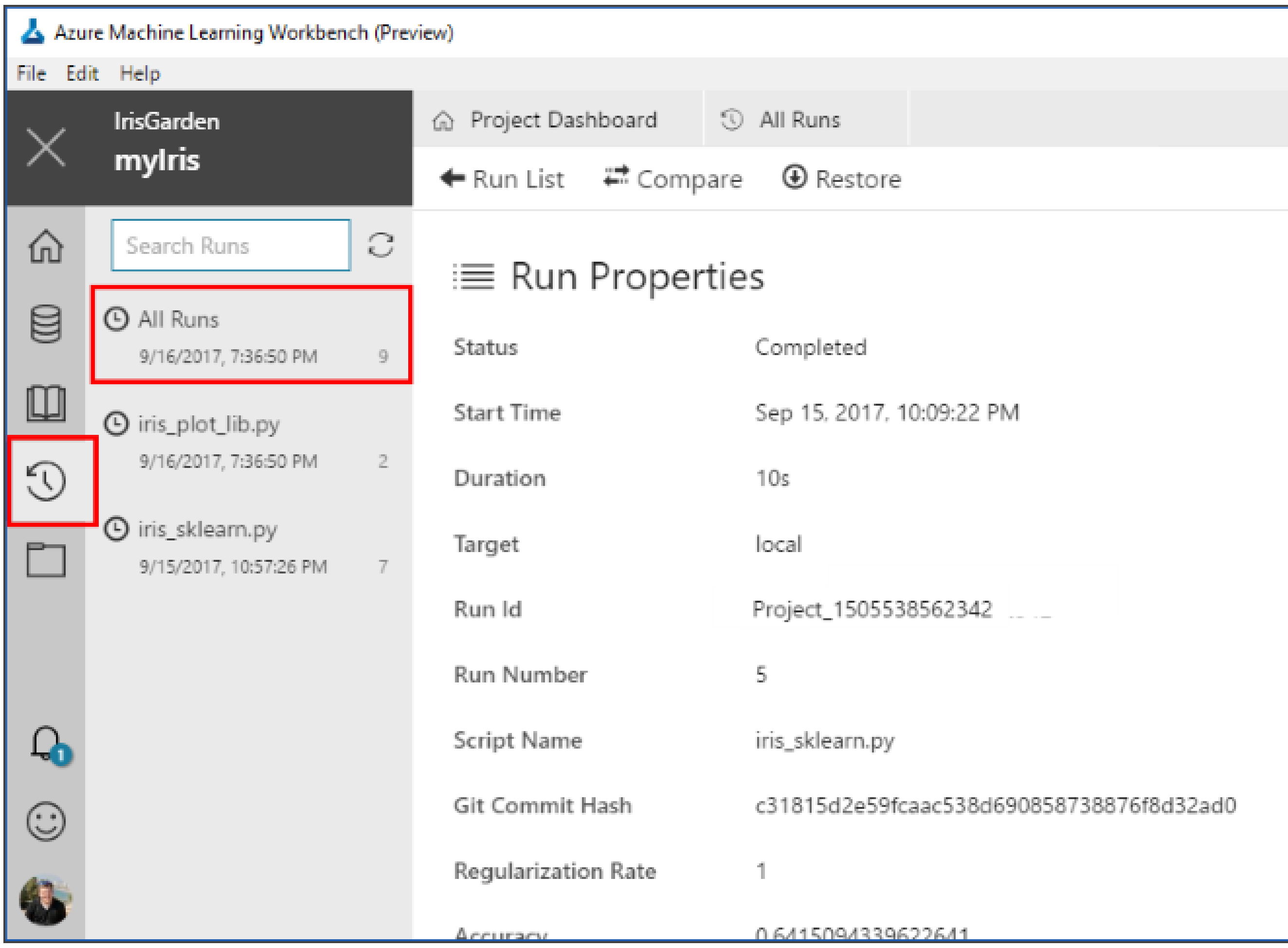
   PythonCopy

```python
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.

   - To locate the file, select the Runs button (clock icon) on the left pane to open the list of All Runs.
   - 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.
   - The Run Properties pane opens. In the upper-right section of the pane, notice the Outputs section.
   - To download the pickle file, select the check box next to the model.pkl file, and then select the Download button. Save it to the root of your project folder. The file is needed in the upcoming steps.



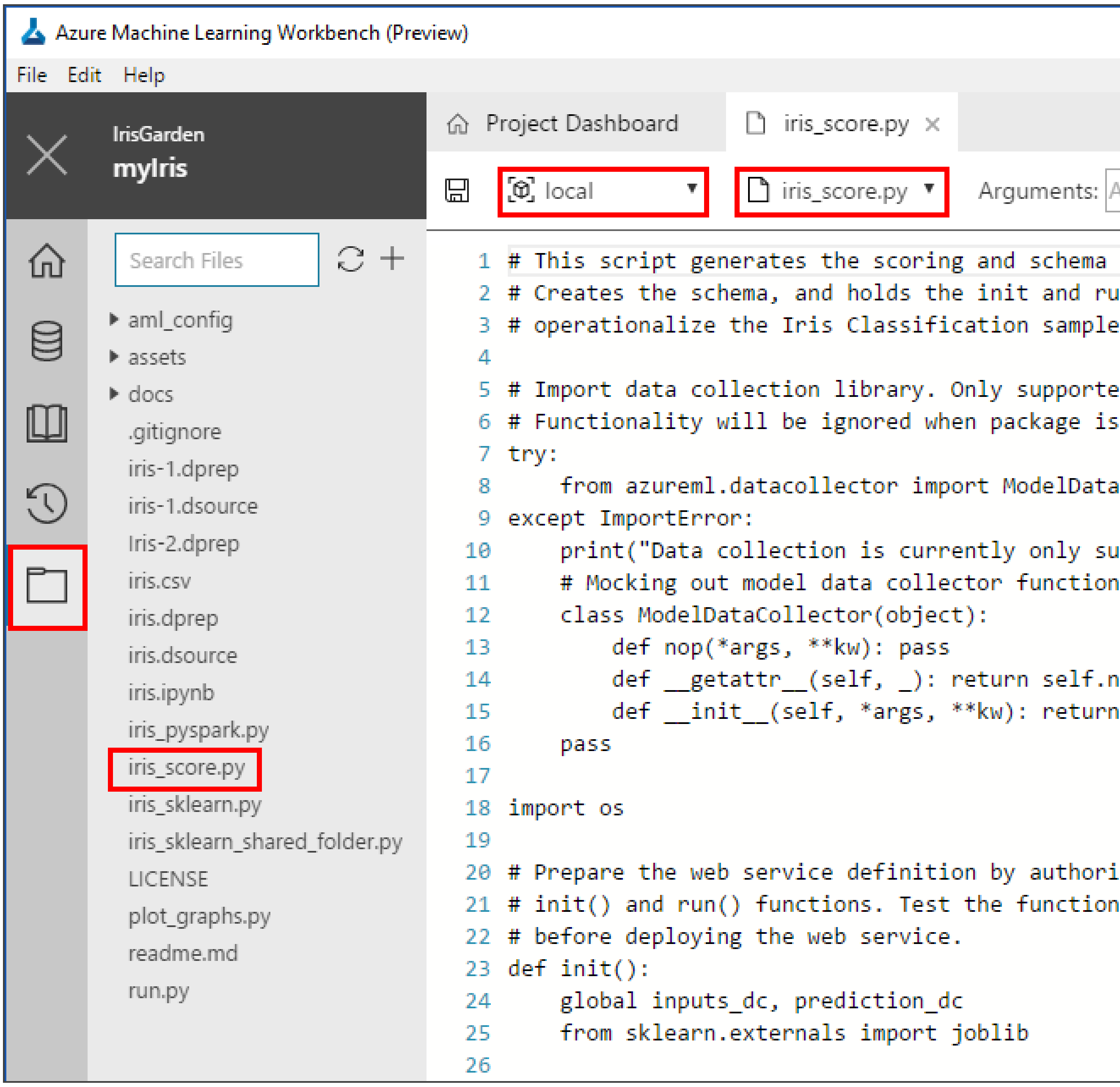
Status	Completed
Start Time	Sep 15, 2017, 10:09:22 PM
Duration	10s
Target	local
Run Id	Project_1505538562342
Run Number	5
Script Name	iris_sklearn.py
Git Commit Hash	c31815d2e59fcaac538d690858738876f8d32ad0
Regularization Rate	1
Accuracy	0.6415094339622641

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

Get the scoring script and schema files

To deploy the web service along with the model file, you also need a scoring script, and optionally, 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 a newly predicted Iris class.

1. Open the Azure Machine Learning Workbench application, and then open the myIrisproject 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.



```
# This script generates the scoring and schema
# Creates the schema, and holds the init and ru
# operationalize the Iris Classification sample

# Import data collection library. Only supporte
# Functionality will be ignored when package is
try:
    from azureml.datacollector import ModelData
except ImportError:
    print("Data collection is currently only su
    # Mocking out model data collector function
    class ModelDataCollector(object):
        def nop(*args, **kw): pass
        def __getattr__(self, _): return self.n
        def __init__(self, *args, **kw): return
    pass

import os

# Prepare the web service definition by authori
# init() and run() functions. Test the function
# before deploying the web service.
def init():
    global inputs_dc, prediction_dc
    from sklearn.externals import joblib

```

4. To get the schema file, run the script. Select the local environment and the score_iris.pyscript in the command bar, and then select the Run button.
5. This script creates a JSON file in the Outputs section, which captures the input data schema required by the model.
6. 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 [1] for the latest job run to see the run details from the score_iris.py run.
7. 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.

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

   PythonCopy

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

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

    PythonCopy

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

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

    PythonCopy

```python
global clf2, inputs_dc, prediction_dc
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

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 run locally to complete the following steps to operationalize the model. You can use the `-h` flag at the end of the commands for command Help.

Note

If you don't have the Docker engine locally, you can still proceed by creating a cluster in Azure for deployment. Just be sure to delete the cluster after the tutorial so you don't incur ongoing charges.

1. Open the command-line interface (CLI). In the Azure 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:

   Copy

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

   You should see output like this:

   Copy

```
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:

   Copy

```
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:

Copy

```
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".

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.

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

   Azure CLICopy

   ```
   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 AppInsight account that gathers telemetry. If you used the `-c` switch, it creates an Azure Container Service cluster too.

   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.

4. Create a Model Management account. (This is a one-time setup.)

   Azure CLICopy

   ```
   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.

   Azure CLICopy

   ```
   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 4. Use the same resource group name that was output in the command window when the setup process finished.

   Azure CLICopy

   ```
   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:

   Azure CLICopy

   ```
   az ml env show
   ```
   1

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:

   Azure CLICopy

```
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
```

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

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

- `-n`: The app name, which must be all lowercase.
- `-f`: The scoring script file name.
- `--model-file`: The model file. In this case, it's the pickled model.pkl file.
- `-r`: The type of model. In this case, it's a Python model.
- `--collect-model-data true`: This enables data collection.

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 file upload 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 and your web service is ready for action.

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

   Azure CLICopy

```
docker ps
```

Create a real-time web service by using separate commands

As an alternate to the az ml service create realtime command shown previously, you can also 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 from the previous step and rebuild the entities only when needed.

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

   Azure CLICopy

   ```
   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:

   Azure CLICopy

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

   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:

   Azure CLICopy

   ```
   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:

   Azure CLICopy

```
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's useful for you to 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:

   Azure CLICopy

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

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

   Azure CLICopy

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

   The output is "2", which is the predicted class. (Your result might be different.)

3. To run the service from outside the CLI, you need to get the keys for authentication:

   Azure CLICopy

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

View the collected data in Azure Blob storage

1. Sign in to the [Azure portal](#).
2. Locate your storage accounts. To do so, select More 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:

   a. Open Azure Machine Learning Workbench.
   b. Select the project you're working on.
   c. Open a command-line prompt from the File menu.
   d. 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 Containers in the list to the left. 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:

   Copy

   ```
   /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:
   - Azure 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.

PythonCopy

```
var df =
spark.read.format("com.databricks.spark.csv").option("inferSchema","tru
e").option("header","true").load("wasb://modeldata@<storageaccount>.blo
b.core.windows.net/<subscription_id>/<resource_group_name>/<model_manag
ement_account_name>/<webservice_name>/<model_id>-<model_name>-
<model_version>/<identifier>/<year>/<month>/<date>/*")
```