Artificial Intelligence Orientation

Lab 1 – Getting Started with Azure Machine Learning

Overview

In this lab, you will learn how to train and evaluate machine learning models using Azure Machine Learning.

What You'll Need

To complete this lab, you will need the following:

- A Microsoft account (for example, an *outlook.com*, *live.com*, or *hotmail.com* address)
- A Microsoft Azure subscription
- A Windows, Linux, or Mac OS X computer
- The lab files for this course

Note: To set up the required environment for the lab, follow the instructions in the setup guide for this course.

Exercise 1: Provisioning an Azure Machine Learning Workspace

In this exercise, you will create an Azure Machine Learning workspace so that you can experiment with data.

Note: The Microsoft Azure portal is continually improved in response to customer feedback. The steps in this exercise reflect the user interface of the Microsoft Azure portal at the time of writing but may not match the latest design of the portal exactly.

Create an Azure Machine Learning Workspace and Plan

Before you can use Azure Machine Learning, you must provision a workspace. In this case, you will provision the workspace in your Azure subscription (note that you can also provision a free Azure machine learning workspace that does not require an Azure subscription – free workspaces are subject to some restrictions).

1. In a web browser, navigate to http://portal.azure.com, and if prompted, sign in using the Microsoft account that is associated with your Azure subscription.
2. In the Microsoft Azure portal, click **New**. Then search for *Machine Learning Workspace* and create a new Machine learning Workspace with the following settings:
    - **Workspace name**: *Enter a unique name.*
    - **Subscription**: *Select your Azure subscription*
    - **Resource Group**: *Create a new resource group with a unique name.*

- **Location:** *Select any available region.*
- **Storage account:** *Create a new storage account with a unique name.*
- **Workspace pricing tier:** Standard
- **Web service plan:** *Create a new web service plan with a unique name.*
- **Web service plan pricing tier:** DEVTEST Standard
- **Pin to dashboard**: *Not selected*

3. In the Azure portal, view **Notifications** to verify that deployment has started. Then wait for the workspace to be deployed (this can take a few minutes.)
4. Click **All resources** and verify that your subscription now includes the following new resources:
   - A Machine Learning Workspace.
   - A Machine Learning Plan.
   - A Storage Account.

Exercise 2: Exploring Data with Azure Machine Learning

In this exercise, you will create an Azure Machine Learning experiment to explore a dataset that consist of clinical observations for patients in a study into indicators that can be used for early detection of diabetes.

Note: The data used in the exercise was generated by a simulation based on the data in the *Pima Indians Diabetes* dataset published by the University of California, School of Information and Computer Science at https://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes.

Open Azure Machine Learning Studio

Now that you have a workspace, you can use Azure Machine Learning Studio to work with data.

1. In the Azure portal, browse to the workspace you created in the previous procedure.
2. In the blade for your workspace, click **Launch Machine Learning Studio**. This opens a new browser page.
3. In the new browser page, sign into Azure Machine Learning Studio using the Microsoft account associated with your Azure subscription.
4. In Azure Machine Learning Studio, at the top right, ensure that the name of the workspace you created in the previous procedure is displayed.

   Note: If a different workspace name is displayed, you may already have some workspaces associated with your account – in which case select your new workspace in the drop-down list.

View Datasets

The exercise data is provided as comma-separated values (CSV) file. In addition to the samples provided, you can upload your own dataset files, and you can ingest data from multiple sources such as Azure Storage, Azure SQL database, and others.

1. In Azure Machine Learning Studio, click **DATASETS**. You should have no datasets of your own (clicking **Samples** will display some built-in sample datasets).
2. At the bottom left, click **+ NEW**, and ensure that the **DATASET** tab is selected.
3. Click **FROM LOCAL FILE**. Then in the **Upload a new dataset** dialog box, browse to select the **diabetes.csv** file in the **Lab01** folder where you extracted the lab files on your local computer and enter the following details as shown in the image below, and then click the **(✓)** icon.
   - **This is a new version of an existing dataset**: Unselected

- **Enter a name for the new dataset**: diabetes.csv
- **Select a type for the new dataset**: Generic CSV file with a header (.csv)
- **Provide an optional description**: Patient data.

4. Wait for the upload of the dataset to be completed, then verify that it is listed.

Create an Experiment

Now that you have uploaded your data, you can create an experiment to explore it.

1. In Azure Machine Learning Studio, click **EXPERIMENTS**. You should have no experiments in your workspace yet.
2. At the bottom left, click **+ NEW**, and ensure that the **EXPERIMENT** tab is selected. Then click the **Blank Experiment** tile to create a new blank experiment.
3. At the top of the experiment canvas, change the experiment name to **Exercise Analysis**.
4. In the experiment items pane, expand **Saved Datasets** and **My Datasets**, and then drag the **diabetes.csv** dataset onto the experiment canvas, as shown here:
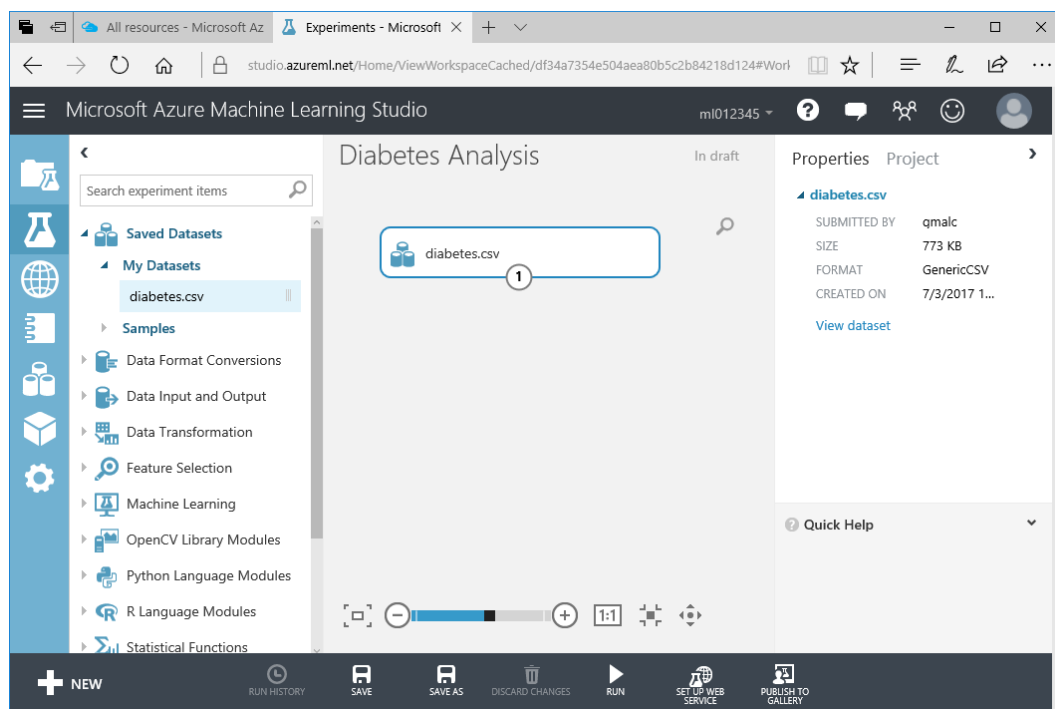

5. Right-click the dataset output of the **diabetes.csv** dataset and click **Visualize** as shown here:
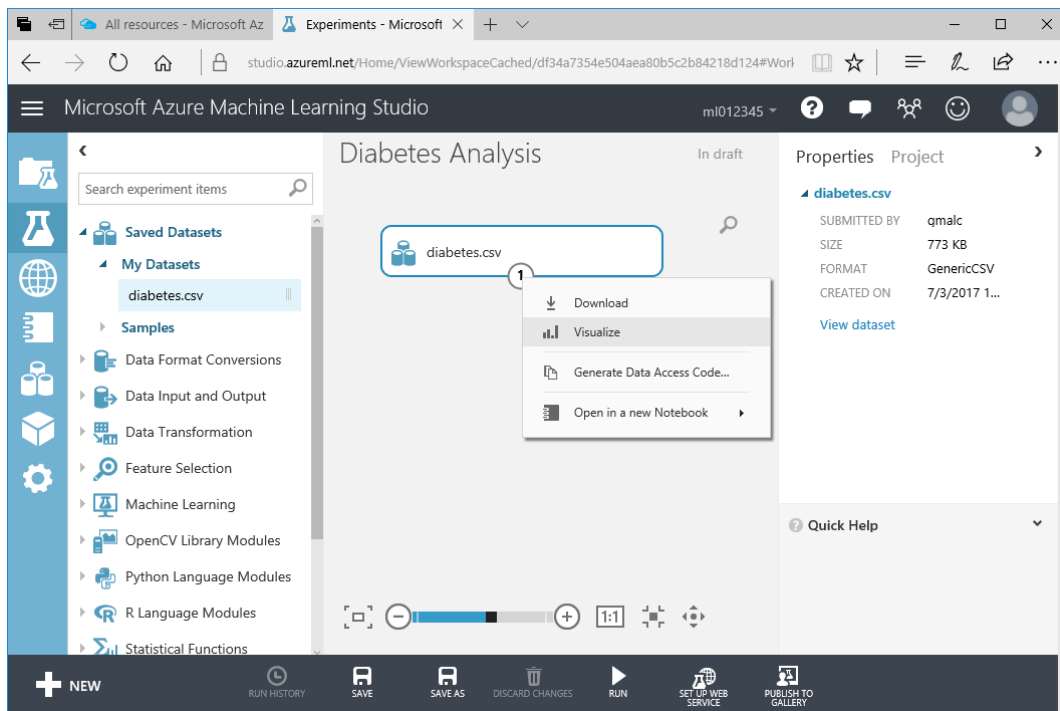
6. In the data visualization, note that the dataset includes a record, often referred to as an *observation* or *case*, for each patient (identified by a **PatientID**), and each case includes characteristics, or *features*, of the patient – in this example, clinical measurements and medical details. The dataset also includes a column named **Diabetic**, which indicates whether the patient is diabetic – this is the *label* that ultimately you must train a machine learning model to predict based on the features of new patients for whom the diabetes diagnosis is unknown.

7. Note the number of rows and columns in the dataset, and then select the column heading for the **PlasmaGlucose** column and note the statistics about that column that are displayed, as shown here:
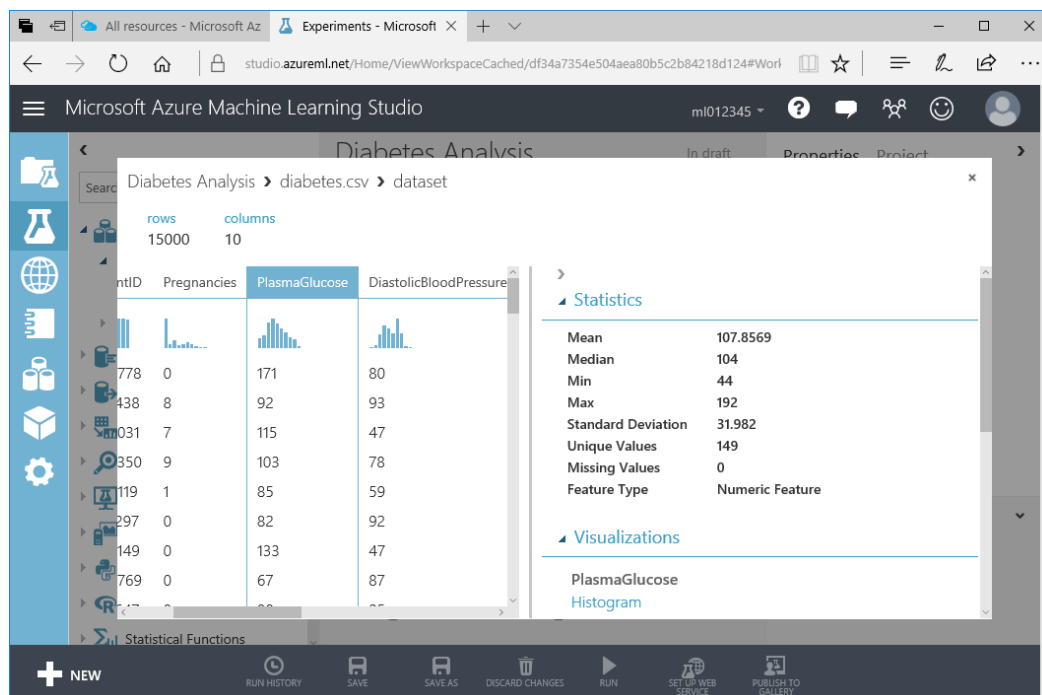
8. In the data visualization, scroll down if necessary to see the histogram for **PlasmaGlucose**. This shows the distribution of different weights within the patient records in the dataset.
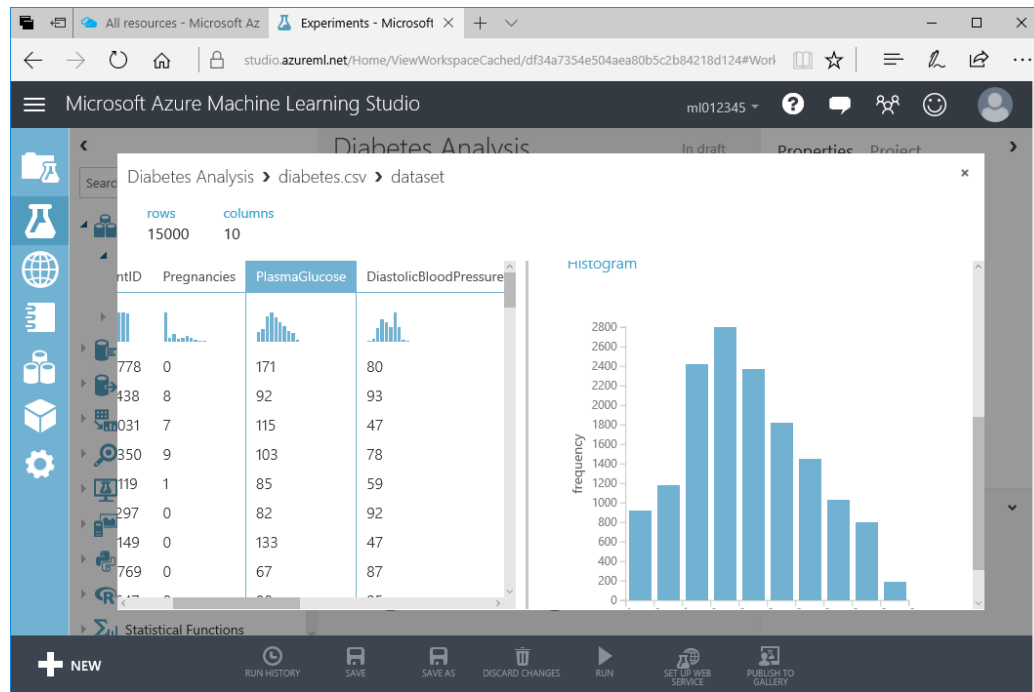
Note: This distribution is close to what data scientists call a *normal* distribution, in which the most frequently occurring values for a variable (in this case the level of glucose in the patient's blood) tend to be in the middle of the range, with approximately similar rates of drop-off as the values move towards the extreme high and low ends. In a histogram, this creates a visualization with a "bell-shaped curve" shape.

9. Close the visualization and return to the experiment canvas.

The primary care physicians for the patients are stored in a separate file, which you must upload as a dataset and add to the experiment.

5. At the bottom left, click **+ NEW**, and select the **DATASET** tab.
6. Click **FROM LOCAL FILE**. Then in the **Upload a new dataset** dialog box, browse to select the **doctors.csv** file in the **Lab01** folder where you extracted the lab files on your local computer and enter the following details as shown in the image below, and then click the **(✓)** icon.
   - **This is a new version of an existing dataset**: Unselected
   - **Enter a name for the new dataset**: doctors.csv
   - **Select a type for the new dataset**: Generic CSV file with a header (.csv)
   - **Provide an optional description**: Doctor data.
10. Wait for the upload of the **doctors.csv** dataset to be completed, then verify that it is listed in the **My Datasets** node.
11. Drag the **doctors.csv** dataset to the experiment canvas, to the right of the **diabetes.csv** dataset.
12. Visualize the output of the **doctors.csv** and note that it contains a row for each patient, along with the name of the patient's physician.
13. Close the visualization and return to the experiment canvas.

Now your experiment contains two datasets with a common **PatientID** field. You can use this field to combine the two datasets.

14. In the **Search experiment items** box, type *Join*, and then from the filtered items list, drag the **Join Data** module onto the canvas and place it below the two datasets.
15. Connect the dataset output from the **diabetes.csv** dataset to the **Dataset1** (left) input of the **Join Data** module, and connect the dataset output from the **doctors.csv** dataset to its **Dataset2** (right) input as shown here:
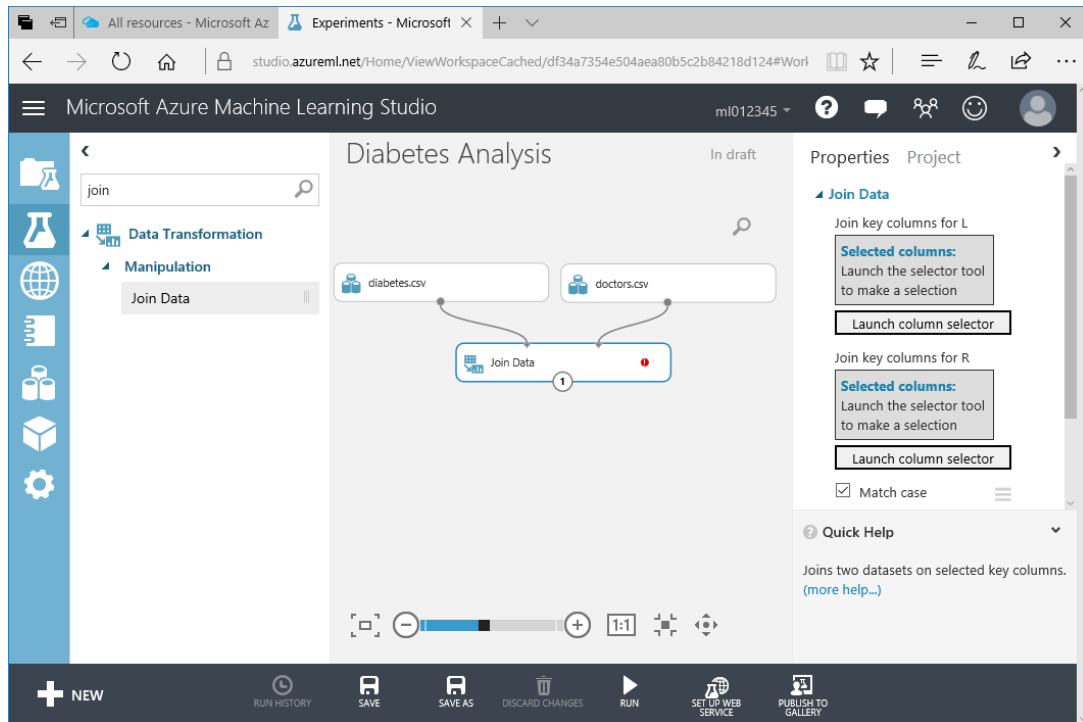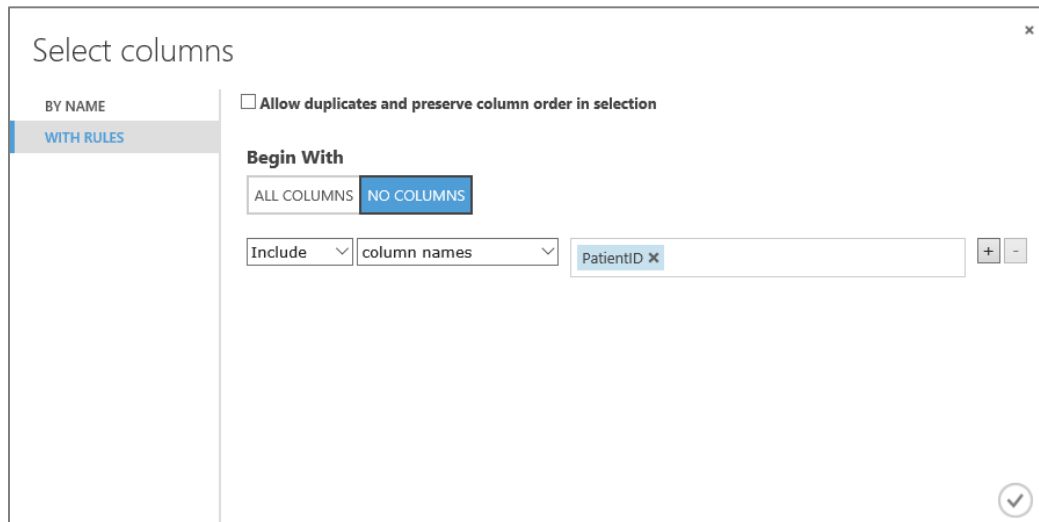

16. With the **Join Data** module selected, in the **Properties** pane, under **Join key columns for L**, click **Launch column selector**.
17. With the **BY NAME** tab selected, select the **PatientID** column and click **[>]** to add it to the **Selected Columns** list, as shown here:
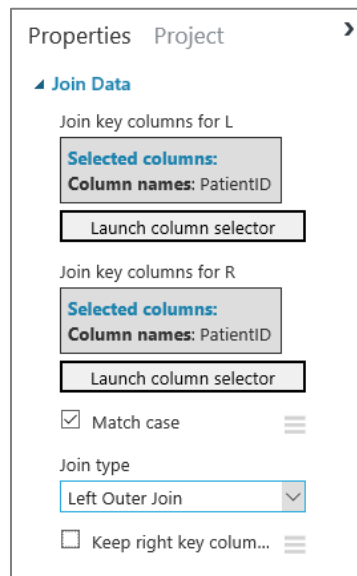

18. Click the **(✓)** icon to confirm the column selection.
19. With the **Join Data** module selected, in the **Properties** pane, under **Join key columns for R**, click **Launch column selector**.

20. Click the **WITH RULES** tab, and note that you can define rules to select columns based on their name, index, and data type. Under **Begin With**, select **No Columns**. Then select **Include**, **column names**, and **PatientID** as shown here (the column names should appear when you start to type). Then click the **(✔)** icon.

21. With the **Join Data** module selected, in the **Properties** pane, clear the **Match case** checkbox, select **Left Outer Join**, and clear the **Keep right key column in joined table** checkbox as shown here:

Note: Using a left outer join ensures that the joined table includes all users in the **diabetes.csv** dataset and their corresponding calories measurement from the **doctors.csv** dataset. If there are any observations in the **diabetes.csv** dataset with no matching **doctors.csv** record, the exercise data will be retained, and the corresponding calories value will be NULL.

22. On the toolbar at the bottom of the experiment canvas, click **Save**. Then click **Run** to run the experiment. Wait for the experiment to finish running (a green ✔ icon will be displayed in the **Join Data** module)

23. Visualize the **Results dataset** output from the **Join Data** module, and note that it contains all the **diabetes.csv** columns and the corresponding **Physician** column from the **doctors.csv** dataset.
24. Close the visualization.

> **Note**: Azure Machine Learning experiments typically include multiple modules that form a data flow in which data from a dataset is cleaned, filtered, and otherwise prepared for analysis or modeling. Azure Machine Learning includes a wide range of modules for common data operations as well as modules that enable you to implement custom logic in Python, R, or SQL.

Create a Notebook

Notebooks provide a convenient way for data scientists to explore data using R or Python code.

1. In the **Search experiment items** box, type *Convert*, and then from the filtered items list, drag the **Convert to CSV** module onto the canvas and place it below the **Join Data** module.
2. Connect the **Results dataset** output from the **Join Data** module to the **Dataset** input of the **Convert to CSV** module as shown here:
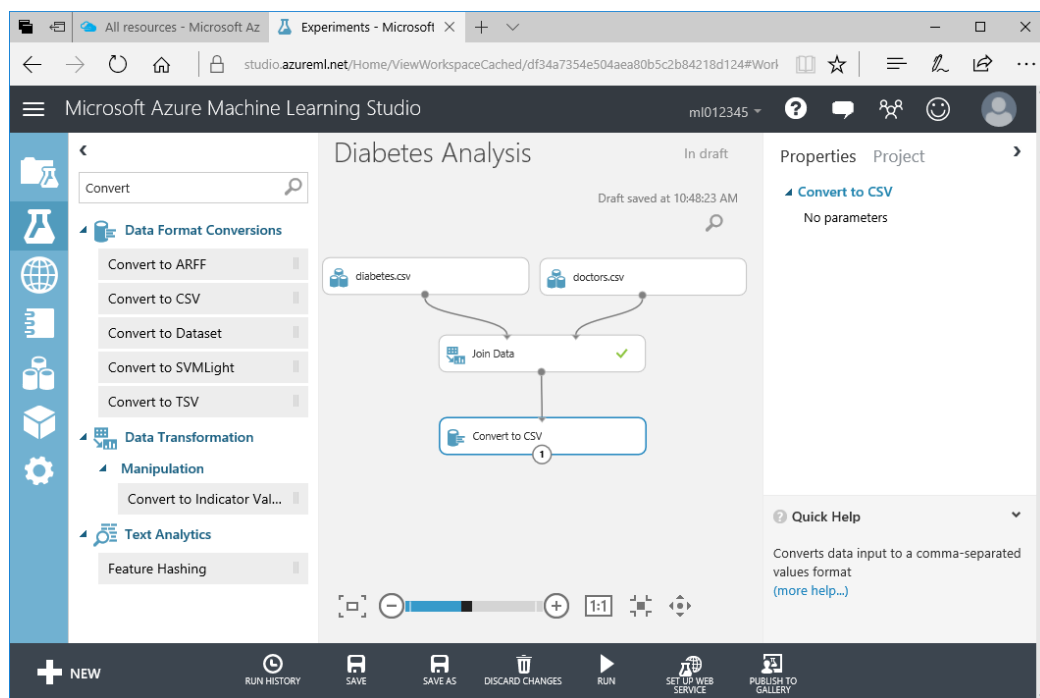
> **Note**: You can open a CSV dataset directly in a notebook. If you have applied any transformations to the data, you must convert it back to CSV before opening it in a notebook.

3. Select the **Convert to CSV** module, then on the **Run** menu, click **Run Selected**. This runs the selected module using the upstream output from the previous run. Wait until it has finished running.
4. Right-click the **Result dataset** output of the **Convert to CSV** module, and then in the **Open in a new Notebook** menu, click **Python 3**; as shown here:
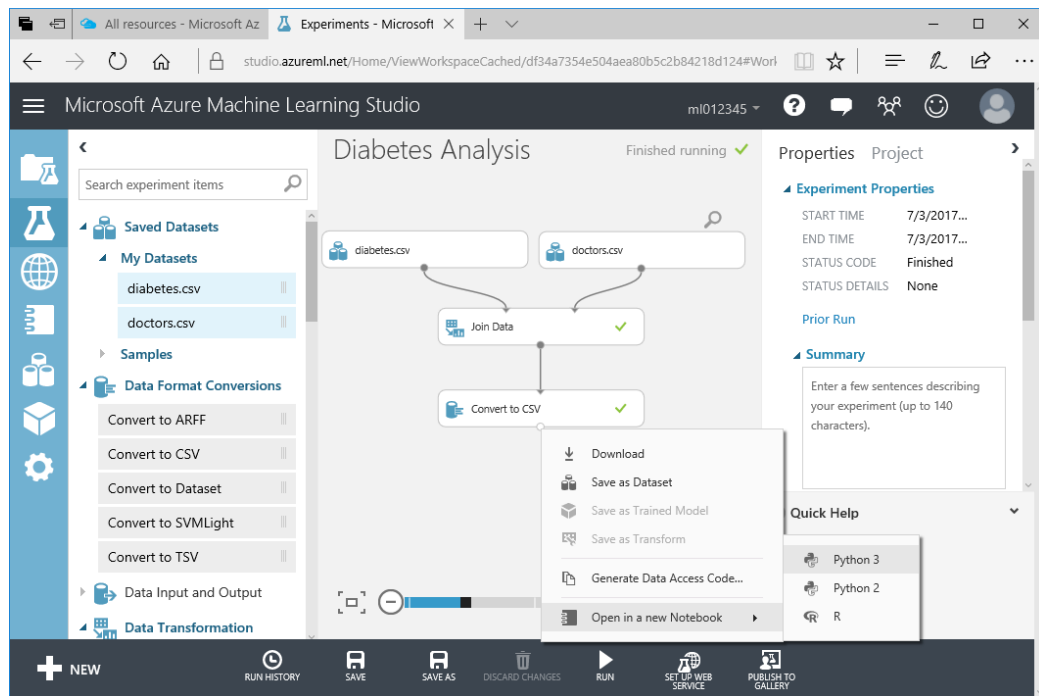
Note: Azure Machine Learning Jupyter notebooks currently support custom code written in R, Python 2, and Python 3. Data scientists can use whichever language they prefer.

5. In the new browser tab that opens, view the Jupyter notebook that has been created. Observe that the notebook contains two cells. The first cell contains code that loads the CSV dataset into a data frame named **frame**, similar to this:

```
from azureml import Workspace
ws = Workspace()
experiment = ws.experiments['335fdf9...f-id.0d127ca2e02245..']
ds = experiment.get_intermediate_dataset(
    node_id='ee772e8e-1600-49a5-a615-a6cfd9661208-21',
    port_name='Results dataset',
    data_type_id='GenericCSV'
)
frame = ds.to_dataframe()
```

The second cell contains the following code, which displays a summary of the data frame:

```
frame
```

6. On the **Cell** menu, click **Run All** to run all of the cells in the workbook. As the code runs, the O symbol next to **Python 3** at the top right of the page changes to a ● symbol, and then returns to O when the code has finished running.

7. Observe the output from the second cell, which shows the first few rows of data from the dataset, as shown here:

8. Click cell 2 (which contains the code `frame`), and then on the **Insert** menu, click **Insert Cell Below**. This adds a new cell to the notebook, under the output generated by cell 2.

9. Add the following code to the new empty cell (you can copy and paste this code from **NotebookCode.txt** in the **Lab01** folder):

```
# Create Faceted Histograms
%matplotlib inline

import seaborn as sns
sns.set_style("whitegrid")

def create_facethists(df):
    import numpy as np
    cols = df.columns.tolist()[:-1]
    for col in cols:
        if(df[col].dtype in [np.int64, np.int32, np.float64]
            and df[col].name != "Diabetic"):
            g = sns.FacetGrid(frame, col="Diabetic")
            g.map(sns.distplot, col)
    return('Done')

create_facethists(frame)
```

Note: This code creates histograms for each numeric variable, faceted by the **Diabetic** label – in other words, it enables you to compare the distribution of numeric feature values for diabetic and non-diabetic patients.

10. With the cell containing the new code selected, on the **Cell** menu, click **Run Cells and Select Below** (or click the ▶| button on the toolbar) to run the cell, creating a new cell beneath.

Note: You can ignore the warnings that are generated.

11. View the output from the code, which consists of a series of histogram pairs in a scrollable pane, as shown here (you can click the left margin of the plot output to show the plots in the main notebook window):
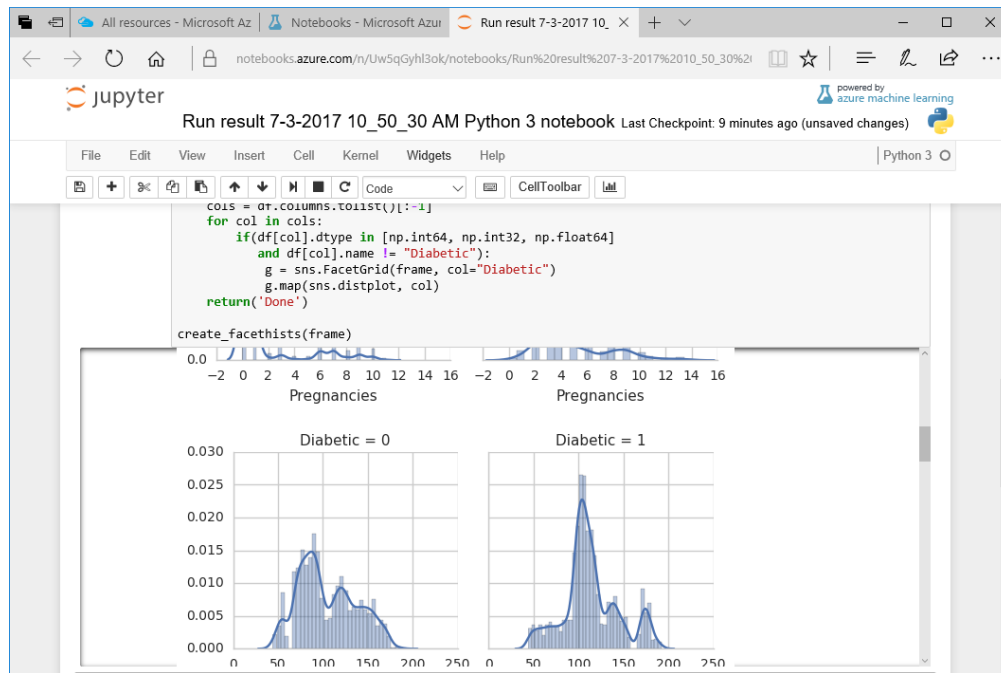
Note that from this diagram, you can see differences in the distribution of some numeric features depending on whether or not the patient is diabetic.

12. In the empty cell at the end of the notebook, add the following code (you can copy and paste this code from **NotebookCode.txt** in the **Lab01** folder):

```
# Create Boxplots

def create_boxplots(df):
    import numpy as np
    import matplotlib.pyplot as plt

    cols = df.columns.tolist()[:-1]
    for col in cols:
        if(df[col].dtype in [np.int64, np.int32, np.float64]
            and df[col].name != "Diabetic"):
            fig = plt.figure(figsize = (6,6))
            fig.clf()
            ax = fig.gca()
            df.boxplot(column = [col], ax = ax, by = ['Diabetic'])
    return('Done')

create_boxplots(frame)
```

Note: This code shows the same comparison of feature value distributions as box plots, which make it easier to see the median and quartile values in the distributions.

13. Run the cell, and view the output from the code, which consists of a series of boxplots, as shown here:
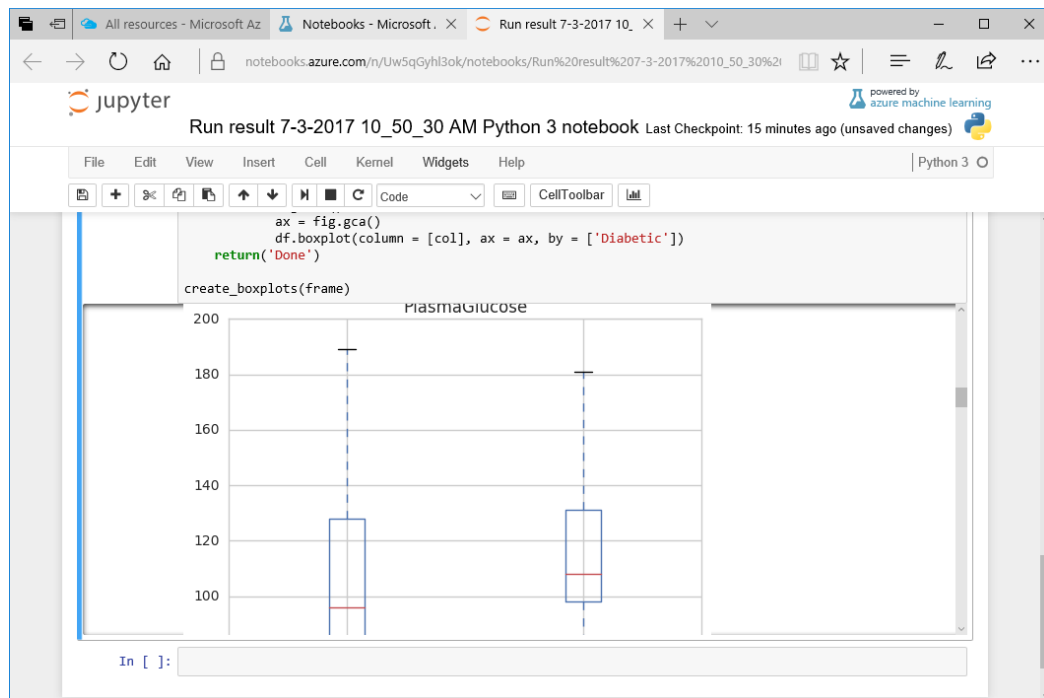
14. On the **File** menu, click **Save and Checkpoint** to save the notebook. Then on the **File** menu, click **Close and Halt** to close the notebook.

15. In Azure Machine Learning Studio, save the experiment, and then click the **Notebooks** page on the left and verify that the notebook is listed there.

16. Select the notebook and at the bottom of the screen, click **Rename**. Then rename the notebook to **Diabetes Analysis**.

Exercise 3: Training a Classification Model

In this exercise, you will use Azure Machine Learning to create a classification model. The goal of this model is to predict whether a patient is diabetic based on the diagnostic features.

Prepare the Data

As is often the case with machine learning of any kind, some data preparation is required before you can use the data to train a model.

1. In Azure Machine Learning Studio, on the **Experiments** page, open the **Diabetes Analysis** experiment you created in the previous exercise, and verify that it looks like this:
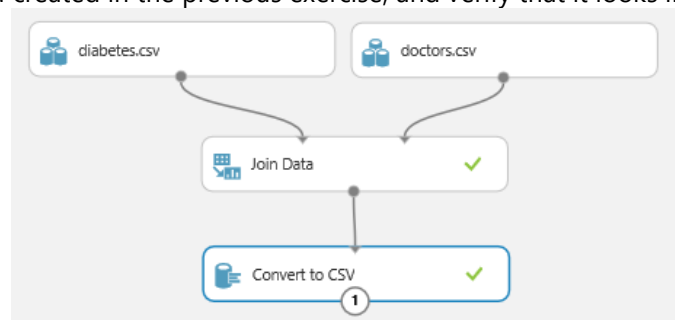

The distribution of the **Age** column in the diabetes.csv dataset is skewed because most patients are in the youngest age bracket. Creating a version of this feature that uses a natural log transformation can help create a more linear relationship between **Age** and other features, and

improve the ability to predict the **Diabetic** label. This kind of feature engineering as it's called is common in machine learning data preparation.

2. Add an **Apply Math Operation** module to the experiment, and connect the output of the **Join Data** module to its input (bypassing the **Convert to CSV** module, which was only required to open the data in a notebook). Your experiment should now look like this:
3. Configure the **Apply Math Operation** module properties as follows:
    *   **Category**: Basic
    *   **Basic math function**: Ln
    *   **Column set**: Select only the **Age** column
    *   **Output mode**: Append

    Your experiment should now look like this:
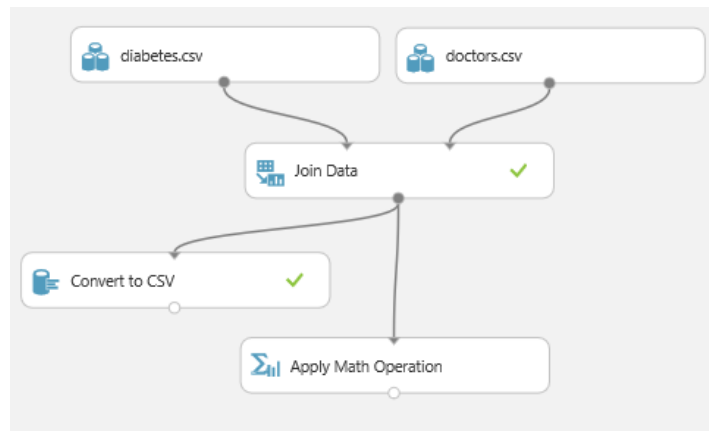
4. **Save** your experiment and **Run selected** to update the output schema of the **Apply Math Operation** module to verify that the **Ln(Age)** column has been added to the dataset.

    It's common when preparing data for model training to scale, or *normalize*, numeric features so that those with large numeric values do not dominate the training of the machine learning model, simply because the values are large. The specific scaling technique used depends on the statistical distribution of the data values - the **Normalize Data** module supports five normalization methods. In this case, the distributions of the numeric columns are quite different. Some of the features have an approximately *Normal* distribution (visualized as histogram with a bell-shaped curve). **ZScore** (mean-variance) normalization is appropriate here. Other features have distributions far from Normal, so **MinMax** normalization (forcing all values in a limited range of say {0,1}) is more appropriate for these columns. Therefore, you will use two **Normalize Data** modules to perform the normalization of all numeric features.

5. Add a **Normalize Data** module to the experiment and connect the output of the **Apply Math Operation** module to its input.
6. Configure the **Normalize Data** module properties as follows:
    *   **Transformation method**: ZScore
    *   **Use 0 for constant columns when checked**: Checked
    *   **Select columns**:
                PlasmaGlucose
                DiastolicBloodPressure
                TriceptsThickness
                SerumInsuin

BMI

7. Add a second **Normalize Data** module to the experiment, and connect the **Transformed dataset** (left) output of the first **Normalize Data** module to its input.
8. Configure the new **Normalize Data** module as follows
    - **Transformation method**: MinMax
    - **Use 0 for constant columns when checked**: Checked
    - **Select columns**:
        Pregnancies
        DiabetesPedigree
        Age
        Ln(Age)

Your experiment should now look like this:
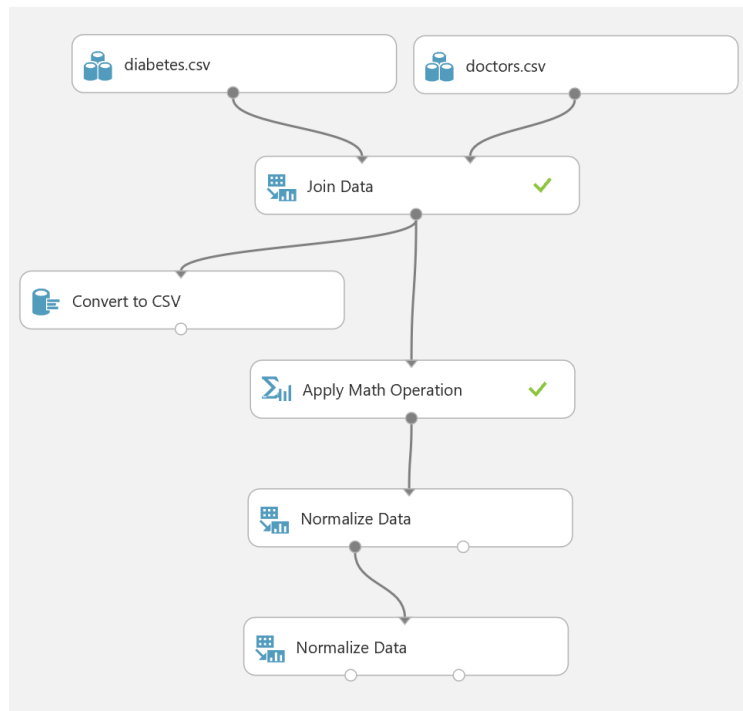
9. Save your experiment and select and run the last **Normalize Data** module. Then visualize **Transformed dataset** (left) output of the last **Normalize Data** module and observe that the values of the numeric features have been scaled.

The **PatientID** and **Physician** columns are unlikely to contribute any predictive information for diagnosing diabetes. You will therefore clear these features so that they are not used to train the model.

10. Add an **Edit Metadata** module to the experiment and connect the **Transformed dataset** (left) output of the second **Normalize Data** module to its input.
11. Configure the **Edit Metadata** module as follows:
    - **Selected columns**: *PatientID* and *Physician*
    - **Data type**: Unchanged
    - **Categorical**: Unchanged
    - **Fields**: Clear feature
    - **New column names**: *Leave blank*

Your experiment should now look like this:
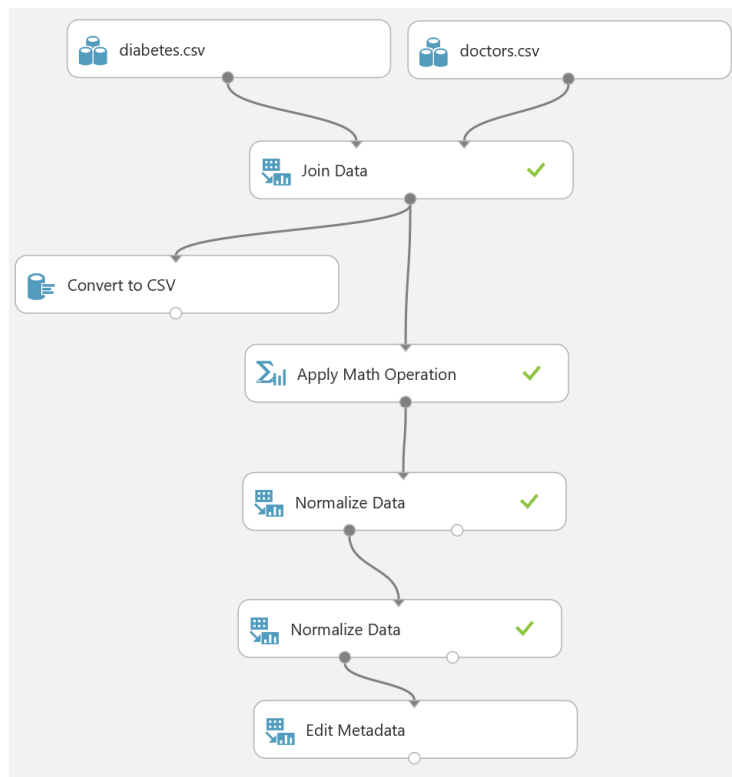
12. Save and run the experiment, and then visualize the output of the **Edit Metadata** module and verify that when you select the **PatientID** column header, its **Feature Type** property is listed as *Numeric*, but when you select the **Pregnancies** column header, its **Feature Type** is *Numeric Feature*.

## Train and Evaluate the Classification Model

Now that you have prepared the data set, you will use it to train and evaluate a classifier machine learning model. Typically, when training a *supervised learning* model, in which the training data includes known label values, you split the data into a training set with which to train the model, and a test set with which to validate predictions generated by the trained model.

1. Add a **Split Data** module to the experiment and connect the output of the **Edit Metadata** module to its input.
2. Configure the **Split Data** module as follows:
    - **Splitting mode**: Split Rows
    - **Fraction of rows in the first output dataset**: 0.7
    - **Random seed**: 1234
    - **Stratified split**: False
3. Add a **Train Model** module to the experiment and connect the **Results dataset1** (left) output of the **Split Data** module to its **Dataset** (right) input.
4. Configure the **Train Model** module properties to set the **Label column** to **Diabetic**.
5. Add a **Two Class Logistic Regression** module to the experiment, and connect its output to the **Untrained model** (left) input of the **Train Model** module.
6. Configure the **Two Class Logistic Regression** module as follows:
    - **Create trainer mode**: Single Parameter
    - **Optimization tolerance**: 1E-07
    - **L1 regularization weight**: 1
    - **L2 regularization weight**: 1

- **Memory size for L-BFGS**: 20
- **Random number seed**: 1234
- **Allow unknow categories**: Checked

7. Add a **Score Model** module to the experiment, and connect the output of the **Train Model** module to its **Trained model** (left) input. Then connect the **Result dataset2** (right) output of the **Split Data** module to its **Dataset** (right) input.
8. Add an **Evaluate Model** module to the experiment, and connect the output of the **Score Model** module to its **Scored dataset** (left) input.
9. Verify that the lower portion of your experiment resembles the figure below.
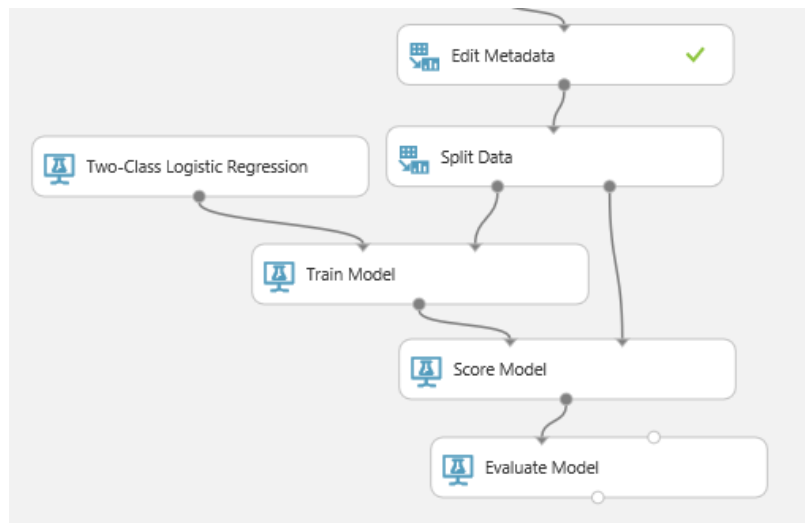
10. Save and run the experiment.
11. Visualize the output of the **Score Model** module, and examine the results. Compare the label column (**Diabetic**), which indicates is the patient is actually diabetic or not, and the **Scored Labels** column which contains the model predictions for diabetic patients. Notice that most of the cases are classified correctly, but that there are some errors.
12. To examine summary statistics of the visualize the output of the **Evaluate Model** module and examine the results. View the ROC curve, which should resemble the figure below.
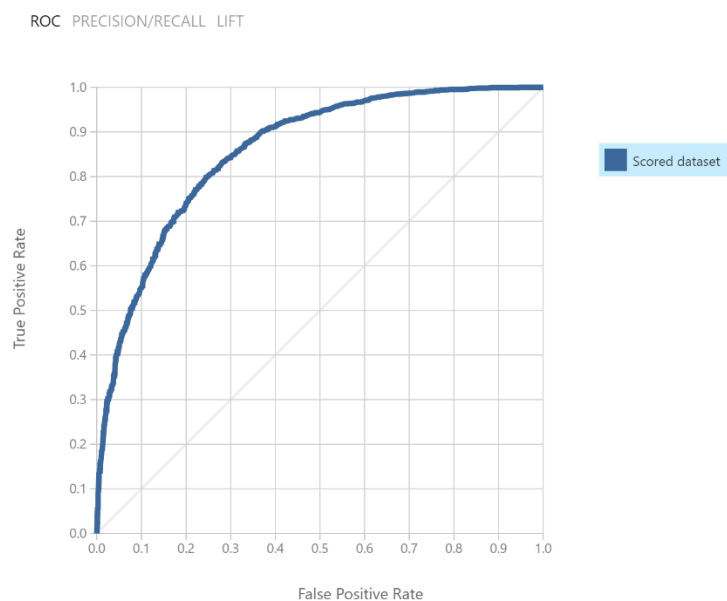
The ROC curve shows the trade-off between the *True Positive Rate* (positive cases correctly classified) and *False Positive Rate* (positive cases incorrectly classified). This curve is above and to the left of the light 45 degree line. This indicates that the classifier is more effective than simply guessing.

13. Scroll down and examine the performance metrics, which should appear as shown below.

True Positive	False Negative	Accuracy	Precision	Threshold		AUC
893	614	0.786	0.719	0.5		0.861

False Positive	True Negative	Recall	F1 Score
349	2644	0.593	0.650

Positive Label	Negative Label
1	0

Notice the following about these metrics:
- The **Confusion Matrix** shows the number of *True Positives* and *True Negatives* (cases correctly classified) and *False Negatives* and *False Positives* (cases incorrectly classified).
- The **AUC** (*Area Under the Curve*) is the area under the ROC curve. A perfect classifier would have an AUC of 1.0, indicating no trade-off between True and False Positive Rates.
- **Accuracy** is the fraction of cases correctly classified.
- **Recall**, is the fraction of positive cases correctly classified. Notice this figure is only 0.593, which means the classifier misclassifies more than 4 of 10 diabetic patients as non-diabetic.
- **Precision** is the fraction of negative cases correctly classified.

Overall, this classifier is significantly better than random guessing, but has only limited accuracy.

Try a Different Algorithm

Given the questionable performance of the classification model, a data scientist would normally work on ways to improve the result. One approach is to find an alternative model algorithm that fits the particular problem better.

1. Add a **Two Class Boosted Decision Tree** module to the experiment.
2. Configure the **Two Class Boosted Decision Tree** module properties as follows:
   - **Create trainer mode**: Single Parameter
   - **Maximum number of leaves per tree**: 20
   - **Minimum number of training instances required to form a leaf**: 10
   - **Learning rate**: 0.2
   - **Number of trees constructed**: 100
   - **Random number seed**: 1234
   - **Allow unknown categories**: Checked
3. Select the **Train Model** and **Score Model** modules, and then copy and paste them to add new copies of these modules to the experiment.
4. Connect the output of the **Two Class Boosted Decision Tree** module to the **Untrained model** (left hand) input of the new **Train Model** module.
5. Connect the **Results dataset1** (left) output of the **Split Data** module to the **Dataset** (right) input of the new **Train Model** module (in addition to the existing connection to the original **Train Model** module.)

6. Connect the **Results dataset2** (right) output of the **Split Data** module to the **Dataset** (right) input of the new **Score Model** module (in addition to the existing connection to the original **Score Model** module.)
7. Connect the output of the new **Score Model** module to the **Scored dataset to compare** (right) input of the **Evaluate Model** module.
8. Add a second **Score Model** module to the experiment, and connect the output of the **Train Model** module for the **Two Class Boosted Decision Tree** module to its **Trained model** (left) input. Then connect the **Result dataset2** (right) output of the **Split Data** module to its **Dataset** (right) input. A shown here.
9. Connect the output of the new **Score Model** module to the **Scored dataset to compare** (right) input of the **Evaluate Model** module, as shown here:
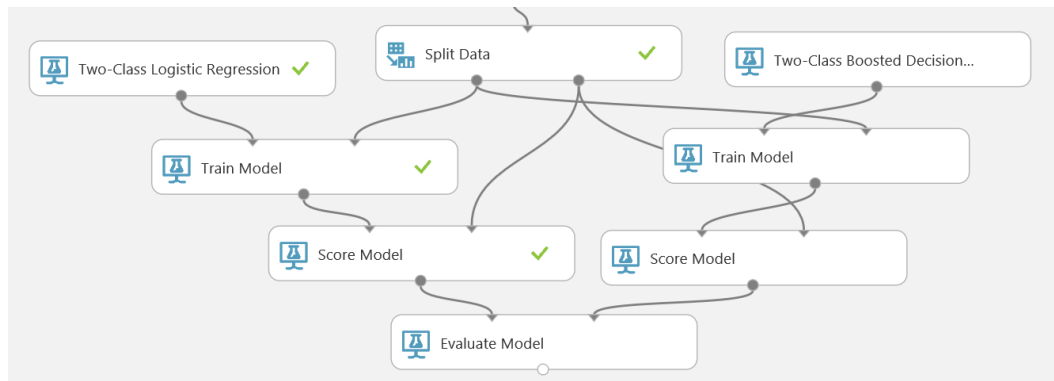
10. Save and run the experiment.
11. Visualize the output of the **Evaluate Model** module, and in the legend for the ROC chart, select **Scored dataset to compare** (which reflects the output from the **Two-Class Boosted Tree** model). Note that the curve for this model is significantly higher and more to the left than the curve for the **Two-Class Logistic Regression** model:
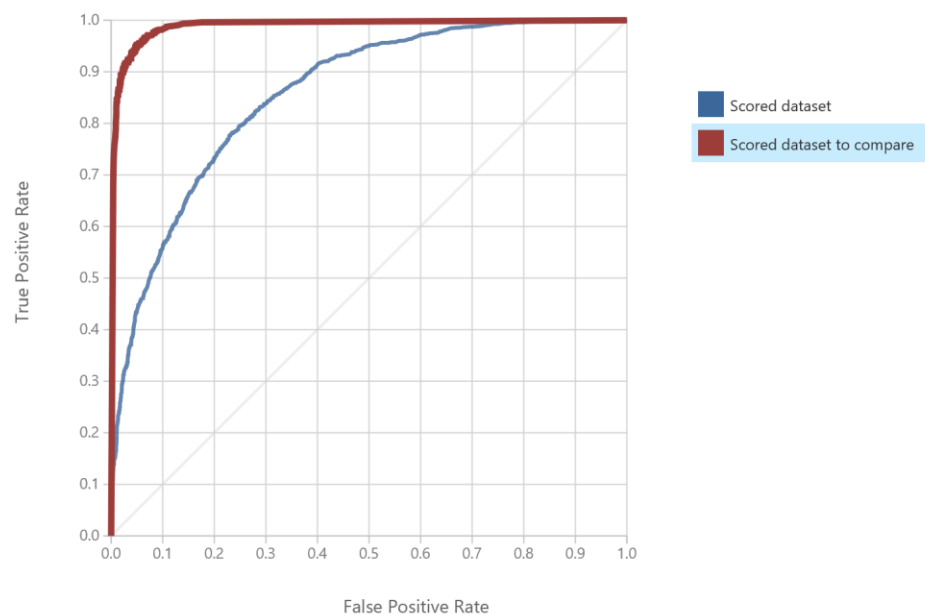
12. With **Scored dataset to compare** selected, scroll down to the performance metrics which should resemble the figure below.

True Positive	False Negative	Accuracy	Precision	Threshold		AUC
1399	108	0.954	0.934	0.5		0.991

False Positive	True Negative	Recall	F1 Score
99	2894	0.928	0.931

Positive Label	Negative Label
1	0

Compare these results to those achieved with the Two Class Logistic Regression, noticing the following:

- The **AUC** is over 0.99, indicating the classifier is nearly ideal.
- The **Accuracy** is now over 95%.
- **Recall** is now nearly 0.93.
- **False Positives** have been greatly reduced, reflected by the **Precision** of 0.934.

This classifier is much more satisfactory when compared to the previous one.

13. Delete the **Two-Class Logistic Regression** module and its **Train Model** and **Score Model** modules, and then switch the output connection from the **Scored Model** module for the **Two-Class Boosted Tree** model to the **Scored dataset** (left) input of the **Evaluate Model** module as shown here.
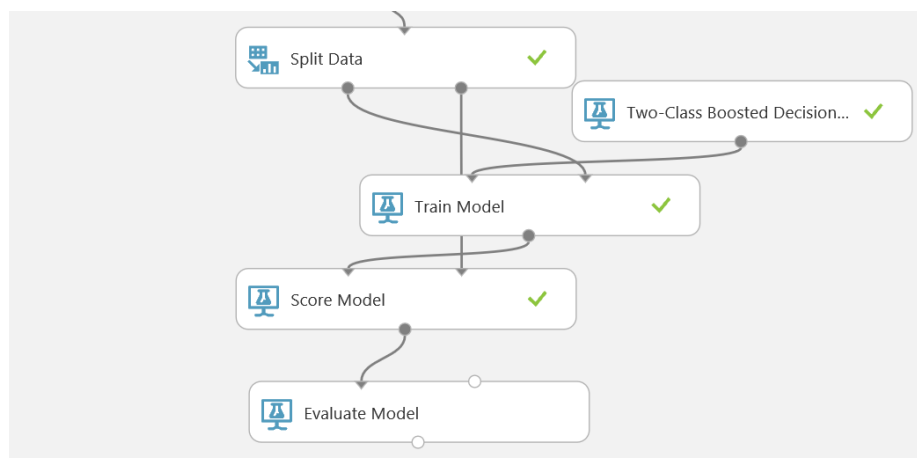
14. Save and run the experiment, and visualize the output of the **Evaluate Model** module to verify that only the **Two-Class Boosted Tree** model is now reflected in the results.

Exercise 4: Publishing and Consuming a Web Service

In this exercise, you will publish and consume the diabetes classification model as a web service.

Create a Predictive Experiment

Now that you have created and evaluated a machine learning you will publish the model as a web service. The first step of this process is to create a predictive experiment from your training experiment, and make any modifications required before operationalizing it as a web service.

1. In Azure Machine Learning Studio, open the **Diabetes Analysis** experiment you created in the previous exercises, and if it has been modified since it was last run, run it. Note that the **Set Up Web Service** icon is disabled until the experiment has been run.
2. On the **Set Up Web Service** icon at the bottom of the studio screen, click **Predictive Web Service [Recommended]**. A new tab containing a **Predictive Experiment** will appear.

3. Change the name of the predictive experiment to **Diabetes Predictor**.

   The **Web service input** in the predictive experiment is based on the training dataset in your training experiment (**diabetes.csv**) This data includes the label (**Diabetes**), which is what the web service will predict – so it does not seem sensible to require this as an input parameter. You will address this problems by defining a custom input for the predictive experiment.

4. Delete the **diabetes.csv** dataset from the experiment canvas.
5. Add an **Enter Data Manually** module to the experiment and connect its output to the left input of the **Join Data** module (to which the **Web service input** is also connected).
6. Configure the **Enter Data Manually** module as follows to provide a header and some test data to define the input schema of the web service (you can copy and paste the data from **Manual Input.txt** in the **Lab01** folder):
   - **DataFormat**: CSV
   - **HasHeader**: Checked
   - **Data**:
     PatientID,Pregnancies,PlasmaGlucose,DiastolicBloodPressure,TricepsThickness,SerumInsulin,BMI,DiabetesPedigree,Age
     1882185,9,104,51,7,24,27.36983156,1.350472047,43
     1662484,6,73,61,35,24,18.74367404,1.074147566,75
     1228510,4,115,50,29,243,34.69215364,0.741159926,59

7. Verify that the top part of your experiment now looks like this:
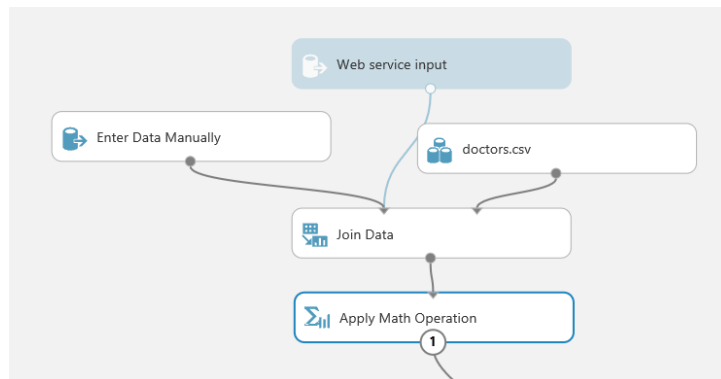
8. Save and run the experiment to verify that there are no errors as a result of changing the input.
9. Visualize the output of the **Score Model** module, which determines the output of the web service, and verify that is contains only three rows – one for each test row in the **Enter Data Manually** module.

   Note: The output includes all columns in the data flow. You will now modify the experiment so that only the patient ID, physician, diabetic prediction, and probability are returned.

10. Delete the connection between the **Score Model** module and the **Web Services Output**.
11. Add an **Apply SQL Transformation** module to the experiment and connect the output of the **Score Model** module to its **Table1** (leftmost) input.
12. Configure the **Apply SQL Transformation** module properties to update the SQL query script to the following code (which you can copy and paste from **Diabetes SQL.txt** in the **Lab03** folder):

```
SELECT PatientID,
       Physician,
       [Scored Labels] AS DiabetesPrediction,
       [Scored Probabilities] AS Probability
FROM t1;
```

13. Connect the output of the **Apply SQL Transformation** module to the input of the **Web service output**.
14. Verify that, with a bit of rearranging, your experiment resembles the figure below:

Diabetes Predictor
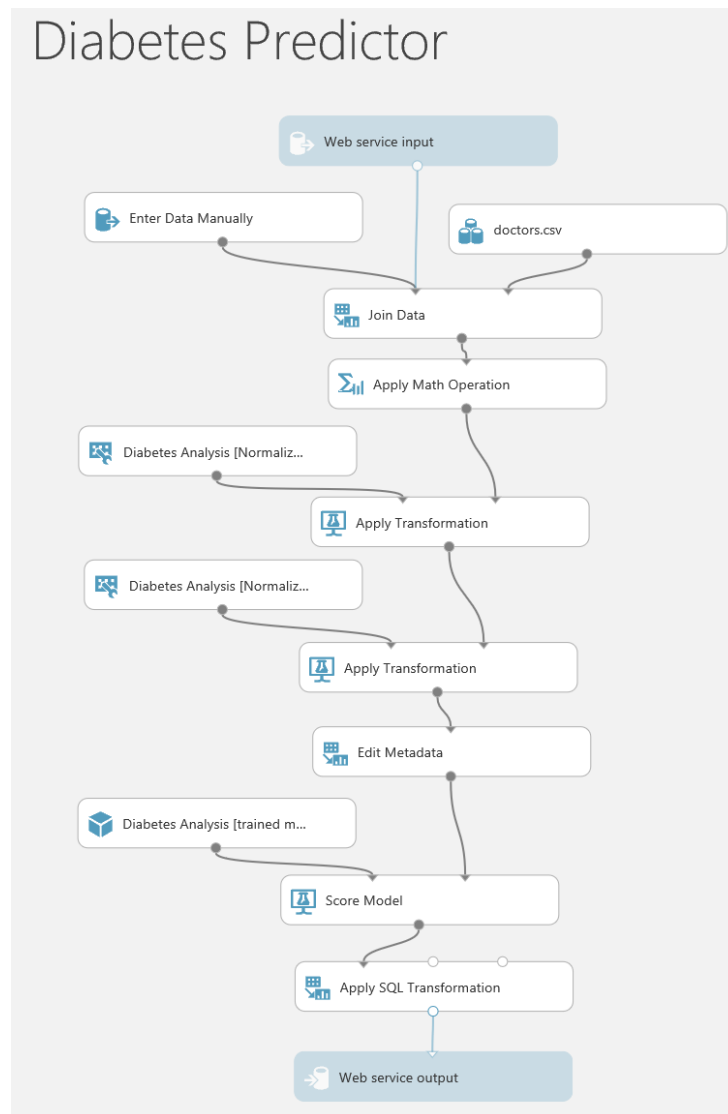
15. Save the run the experiment.
16. Visualize the output from the **Apply SQL Transformation** module and verify that only four columns are returned.

Deploy the Web Service
You are now ready to publish and test the web service.

1. With the **Diabetes Predictor** predictive experiment open, in the **Deploy Web Services** menu, click **Deploy Web Service [New] Preview**. A web page with a header **Deploy "Diabetes Predictor" experiment as a web service** will appear.
2. Set the web service configuration as follows:
   - **Web Service Name**: DiabetesPredictor
   - **Storage Account**: The Azure storage account associated with your workspace
   - **Pricing Plan**: The Azure Machine Learning pricing plan you created with your workspace.

3. Click the **Deploy** box in the lower left of the page. After some time, the **Quickstart** page for your **Diabetes Predictor** web service should appear.

Consume the Web Service

Now that you have published the predictive web service, you can consume it from client applications.

Note: In this exercise, you will consume the web service from Excel Online, which is a service included with your Microsoft account (for example, an *outlook.com* account). If you have Excel 2010 or later installed on your local computer, you can use that instead of Excel Online if you prefer.

1. Click the **Configure** page for the web service, and next to **Sample Data Enabled?**, click **Yes**. Then click **Save** and wait for the changes to be saved.

   Note: By enabling sample data, client applications can download the sample data you defined in the **Enter Data Manually** module in the predictive experiment to test the web service.

2. In the **Configure** page for your web service, click **Consume**, and note that this page lists the primary and secondary keys for your web service, and its **Request-Response** and **Batch Requests** endpoint URLs.
3. Open a new tab in your browser, navigate to https://office.live.com/start/Excel.aspx, and sign in using your Microsoft account.
4. Create a new blank workbook, as shown here:
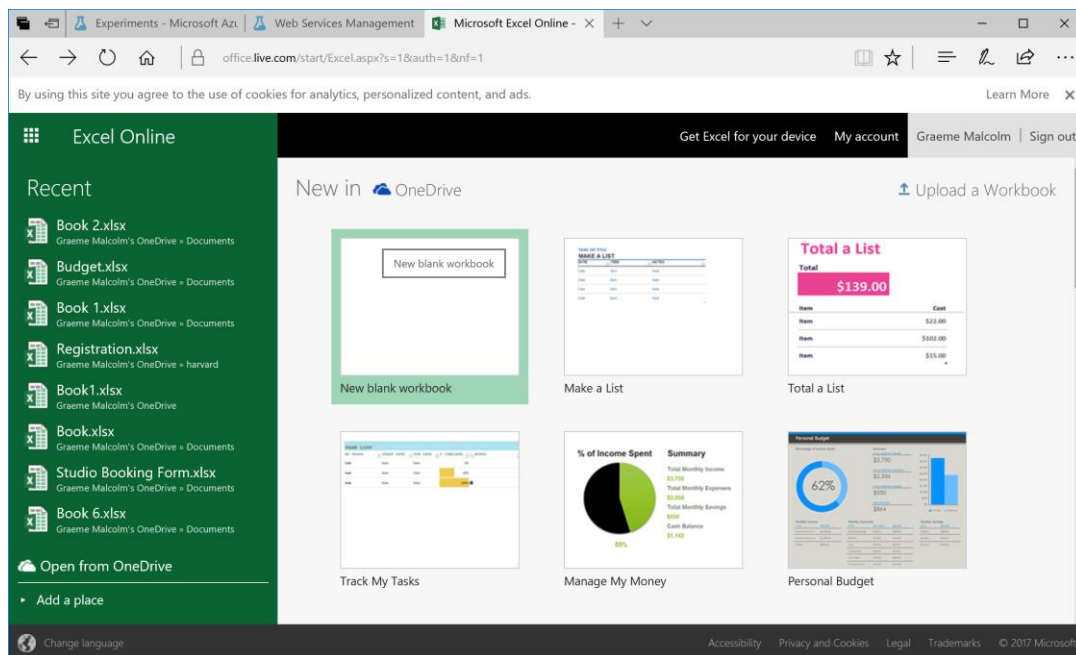
5. At the top of the blank workbook, change the name to **Diabetes Predictions** as shown here:
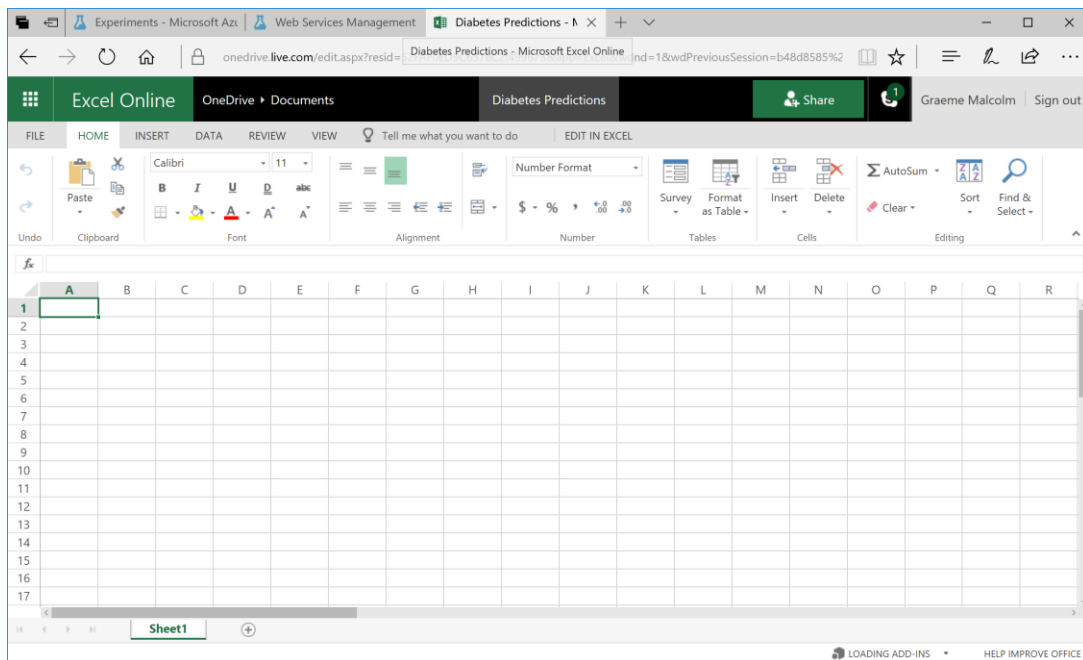
6. On the **Insert** tab, click **Office Add-ins**. Then in the **Office Add-ins** dialog box, select **Store**, search for *Azure Machine Learning*, and add the **Azure Machine Learning** add-in as shown below:
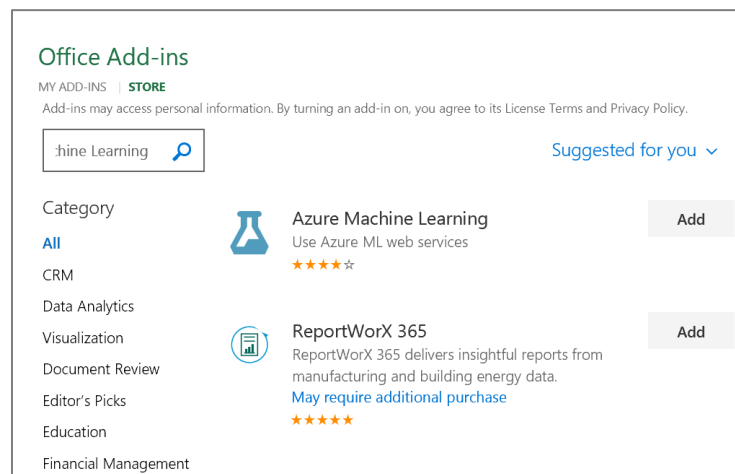
After the add-in is installed, it will appear as a pane named **Azure Machine Learning** on the right side of your Excel workbook.

7. In the **Azure Machine Learning** pane, click **Add Web Service**. Boxes for the URL and API key of the web service will appear.

8. On the browser tab containing the **Consume** page for your web service, click the **Copy** button for the **Request-Response** URL. Then return to the browser tab containing the Excel Online workbook and paste the URL into the URL box.

9. On the browser tab containing the **Consume** page for your web service, click the **Copy** button for the **Primary Key**. Then return to the browser tab containing the Excel Online workbook and paste it into the **API key** box.

10. Verify that the **Azure Machine Learning** pane in your workbook now resembles this, and click **Add**:

Azure Machine Learning

Web Services

🌐 Titanic Survivor Predictor (Excel Add-in Sa…

🌐 Text Sentiment Analysis (Excel Add-in Sam…

URL ❓

https://ussouthcentral.services.azureml.net/subsc
riptions/7a81077e659242f58dbc631b59a08f0c/se
rvices/67215c8153034eeb8d2b17f16a613c58/exec
ute?api-version=2.0&format=swagger

API key ❓

LQYNXzMBAxR+QyJVZhPl+7/tLyWbh+B/FTDw
W30BNiwaWVyl61h2JLs4rXAnY4rUF7AJlP/0REpFf
Cof19YmdQ==

Cancel   Add

☐ Auto-predict   Predict All

Help   Privacy Statement

11. After the web service has been added, in the **Azure Machine Learning** pane, click **1. View Schema** and note the *inputs* expected by the web service, the *outputs* returned by the web service, and the *global parameters* for the web service.
12. In the Excel worksheet select cell A1. Then in the **Azure Machine Learning** pane, collapse the **1. View Schema** section and in the **2. Predict** section, click **Use sample data**. this enters sample input values in the worksheet.
13. With the cells containing the sample input data (cells A1 to I4) selected, in the **Azure Machine Learning** pane, click the button to select the input range and confirm that it is **'Sheet1'!A1:I4**.
14. Ensure that the **My data has headers** box is checked.
15. In the **Output** box type **J1**, and ensure the **Include headers** box is checked.
16. Click the **Predict** button, and after a few seconds, view the predictions in cells J1:M4, as shown here:

17. Try changing a few of the input variables and predicting the diabetes classifications. You can add multiple rows to the input range and try various combinations.
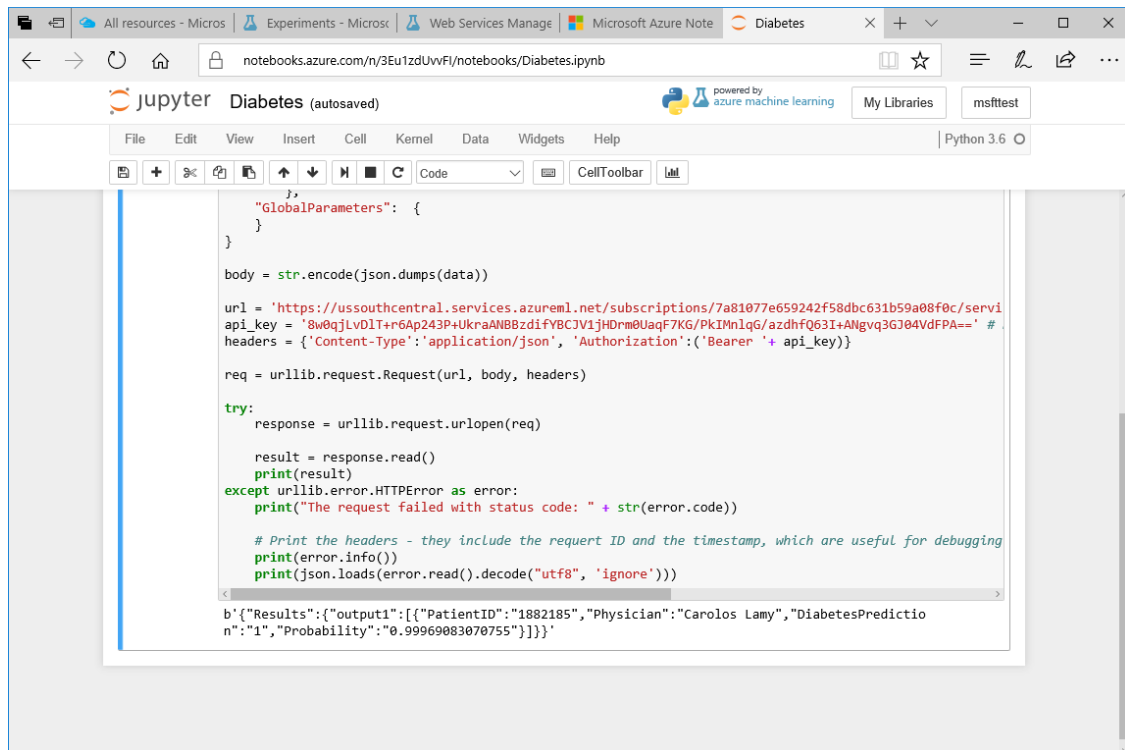
Create a Custom Client

Your Azure Machine Learning web service is available to any REST-compatible client application over HTTP; so, in addition to using the Excel add-in, you can easily implement a custom client application to consume your web service.

1. Switch back to the browser tab containing the Azure Machine Learning web services management site for your web service.
2. In the **Consume** page, under the primary and secondary keys for your web service and its **Request-Response** and **Batch Requests** endpoint URLs, note that Azure Machine Learning has generated C#, Python, and R code that you can use to call your web service.
3. Open a new tab in your browser and navigate to https://notebooks.azure.com. This is a free Azure service that provides Jupyter notebooks independently of your Azure Machine Learning workspace.
4. Sign in using your Microsoft account. If this is the first time you have signed into Azure Notebooks, you may need to grant the application some permissions.
5. After you have signed in, click **Libraries**. Then click **New Library**.
6. Create a new library with your choice of name, unique ID, and description. If you wish, you can choose to make the library public (so you could enable other users to view your notebooks).
7. In your new library, click **New** to create a new notebook. Name the new notebook **Diabetes.ipynb** and choose **Python 3.6** as the language.
8. Open the **Diabetes.ipynb** notebook, and note that the first cell is empty, awaiting your code.
9. Switch back to the Web Services Management tab, in which the **Consume** page for your web service should be visible, and view the **Python 3.x** code for the **Request-Response** endpoint of your web service.
10. Click the **Copy** button to copy the code to the clipboard, and then switch back to the **Diabetes.ipynb** notebook and paste the code into the first cell.

11. In the code you have just pasted, under the JSON definition of the request body, find the following line of code:

```
api_key = 'abc123' # Replace this with the API key for the web service
```

12. Switch back to the Web Services Management tab, and in the **Consume** page, copy the **Primary Key** for your web service to the clipboard.
13. Switch back you the **Diabetes.ipynb** notebook, and in the line of code above, replace *abc123* with the primary key you have copied.
14. On the **Cell** menu, click **Run All**, and then view the JSON output returned by the web service as shown here:

Summary

In this lab, you have used Azure Machine Learning to train and evaluate a classification model. This required you to explore and prepare the data for modeling through feature engineering and normalization, splitting the data into training and test datasets, and evaluating performance metrics for classification. You then published your predictive model as a web service and consumed it from Excel and from a custom client application.