DATA SCIENCE DOJO

2205 152ND AVE NE, REDMOND, WA 98052

The Library of Congress has catalogued this edition as follows:
Data Science Dojo LLC
Data Science and Data Engineering Handbook : 2.3
Registration Number: TX0008030133
2015-05-22

Printed in the United States of America

1. GitHub Tutorial

2. Introduction to R

3. Data Exploration Using R

4. Data Mining with Azure ML Studio

4.1 Getting Started with Azure Machine Learning Studio

In this section we will be familiarizing ourselves with Azure Machine Learning Studio (Azure ML for short). We will create a Azure account and a workspace within our account, learn how to access the workspace from the Azure Portal, and finally create our first experiment.

4.1.1 Exercise: Creating an Azure Account

1. In order to get started and begin your first exercise with Azure ML, you must sign up for a free trial. You can register at: `http://azure.microsoft.com/en-us/pricing/free-trial/`. You may skip this step if you already have an Azure Account.
2. To access a previously created account go to: `https://portal.azure.com`

4.1.2 Exercise: Creating an Azure Machine Learning Studio Workspace

Once you have a dedicated Azure account, we will create an Azure ML Studio workspace.

1. Click on the **+New** button
2. Search for **"Machine Learning"** (Figure: 4.1).

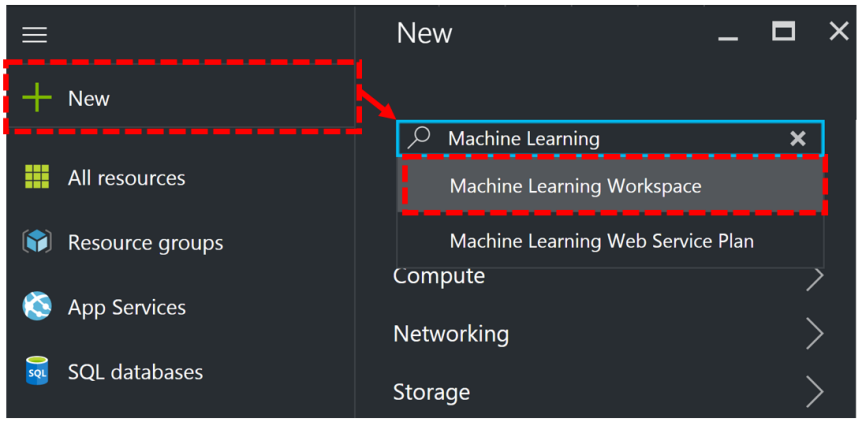


Figure 4.1: Search for "Machine Learning"

3. Select **Machine Learning Workspace** by Microsoft and click **Create** (Figure: 4.2).

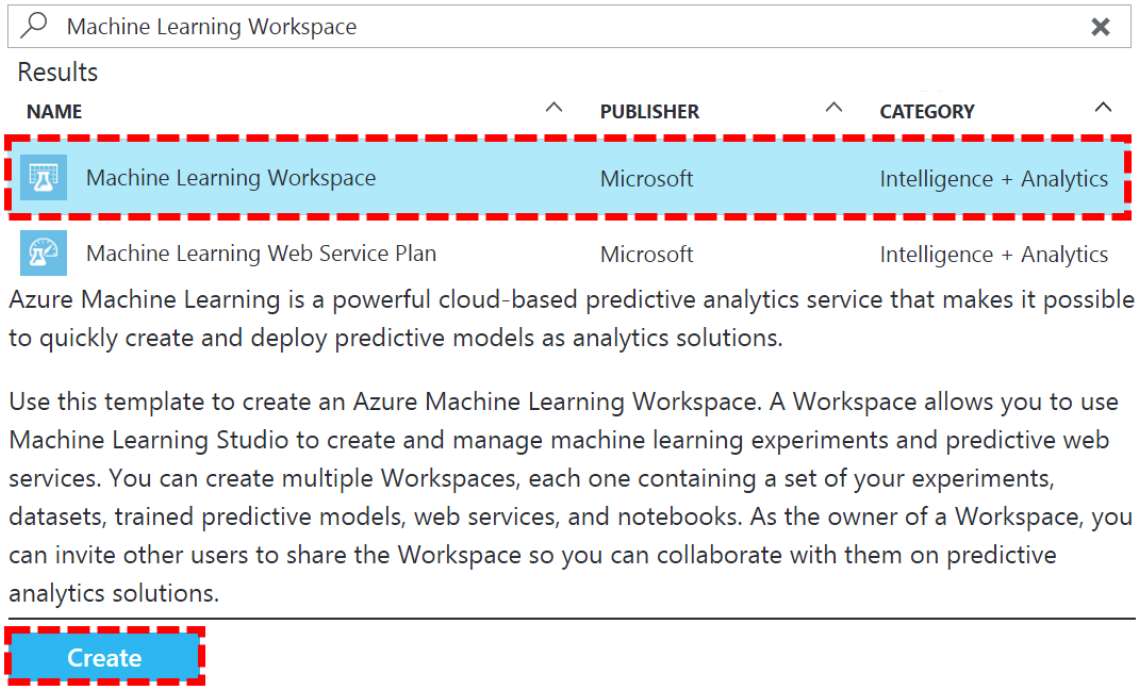


Figure 4.2: Create a new workspace

4. In the **Workspace Name** box, assign a globally unique name.
5. In the **Subscription** dropdown, select a subscription if you have more than one.
6. In the **Resource Group** box, create a new resource group.

7. In the **Storage Account** box, create a new globally unique storage account. It is recommended that you dedicate a storage account to only Azure ML.
8. In the **Web Service Plan** box, create a new web service plan.
9. In the **Web Service Pricing Plan Tier** choices, choose the $1 Standard tier (Figure: 4.3).

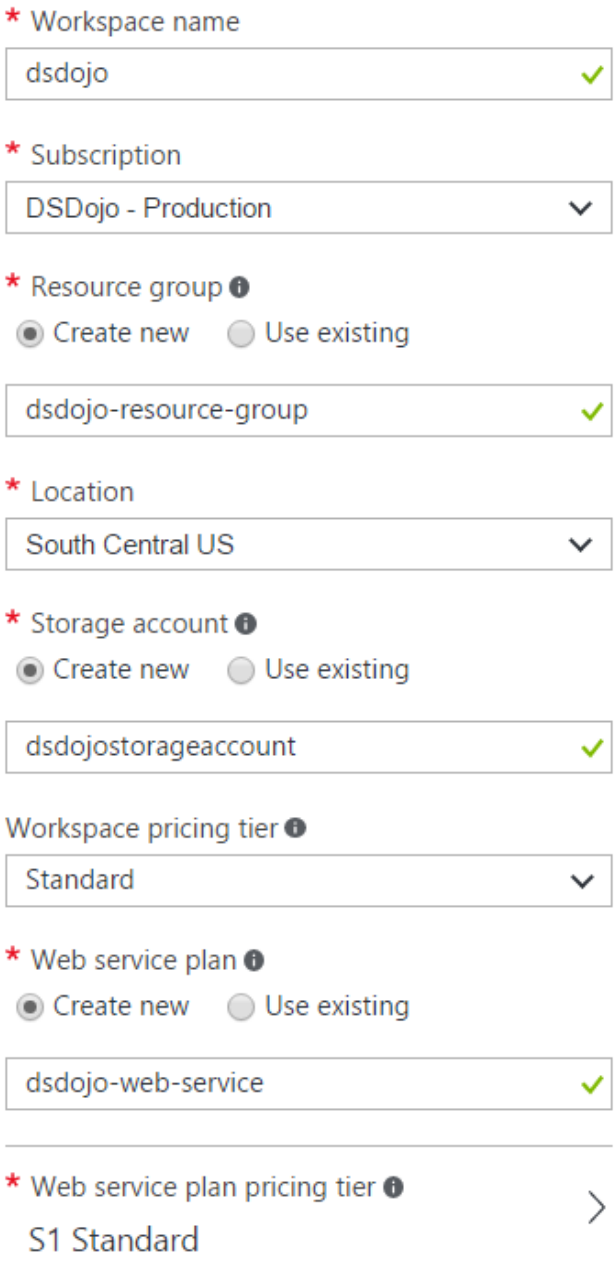


Figure 4.3: Fill out credentials for a new workspace

10. Click the **Create** button once the credentials have been populated to send off a workspace request to Azure. The workspace will take at least two minutes to setup. Accidently deleting the resource group, storage account, or web service plan associated with your Azure ML workspace will corrupt the workspace.

4.1.3 Exercise: Creating your First Experiment

Data Science is an interdisciplinary art and science. It borrows terms from other disciplines, especially the sciences. In this tradition, a project in data science is called an "experiment".

1. You may now access your Azure ML workspace at: `https://studio.azureml.net/` and signing into your account
   If you have multiple workspaces, you can change your workspace by clicking on the dropdown menu near your user name in the top right corner. (Figure: 4.4)

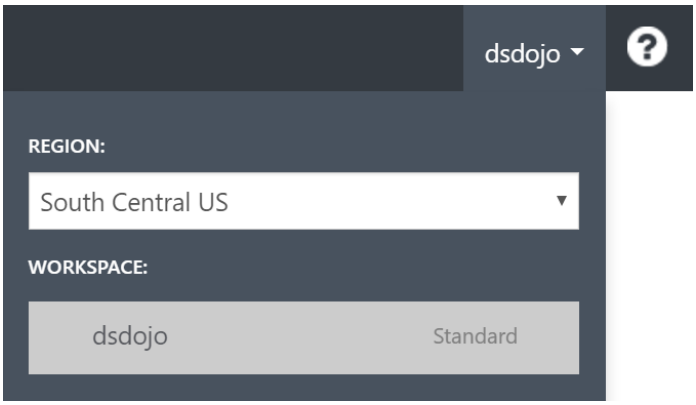


Figure 4.4: Change your workspace

2. To create a new experiment, clicking **+New** button (Figure: 4.5)

Figure 4.5: New experiment button

3. Then, click on **Experiment** > "Blank Experiment" (Figure: 4.6).

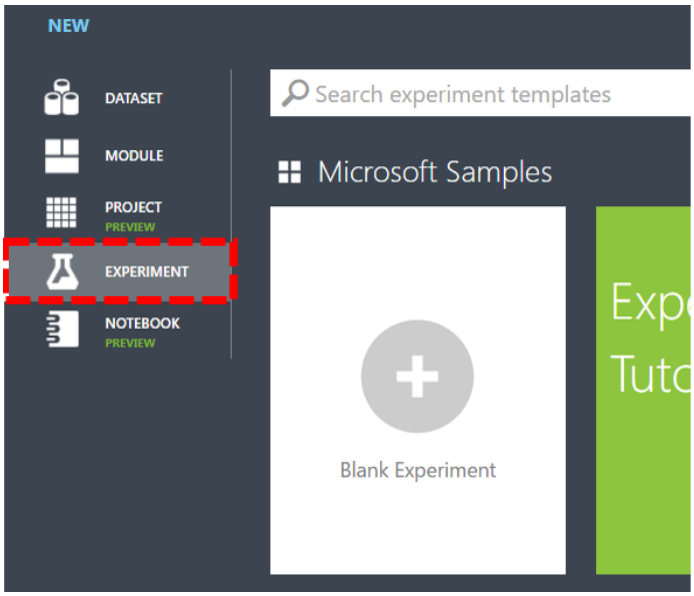


Figure 4.6: Create a blank experiment

4. Name your experiment in the "Experiment Name" field (Figure: 4.7).

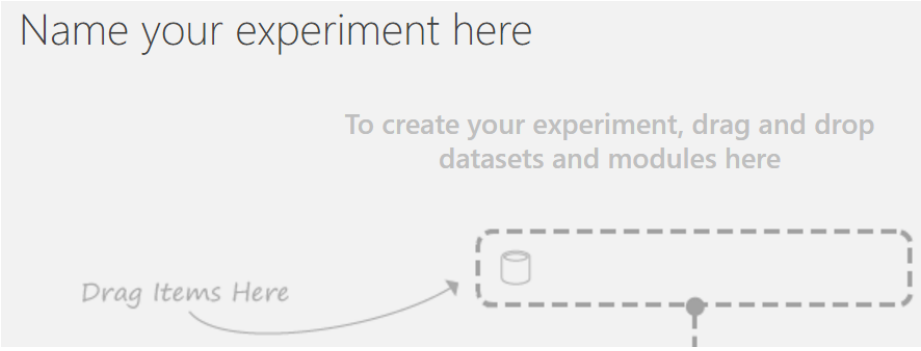


Figure 4.7: Naming a new experiment

4.2 Methods of Ingress and Egress with Azure Machine Learning Studio

4.2.1 Exercise: Reading a Dataset from a Local File

The first dataset we will be using is the "hello world" dataset when getting started with data science. The data describes features of an iris plant in an attempt to predict its species.
To retrieve the dataset, Google "UCI Iris Data" or go to:
`http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data`
Notice how commas differentiate each value. This allows us to know that the elements can be read as a comma separated values ("CSV") file. Excel files and delimited text files can be read as CSV as well. Also notice that the data does not have headers. Headers can be defined later on.

1. Download and save the text as a CSV file. For example "filename.csv".

2. In Azure ML Studio, select **+New > Dataset > From Local File** (Figure: 4.8).

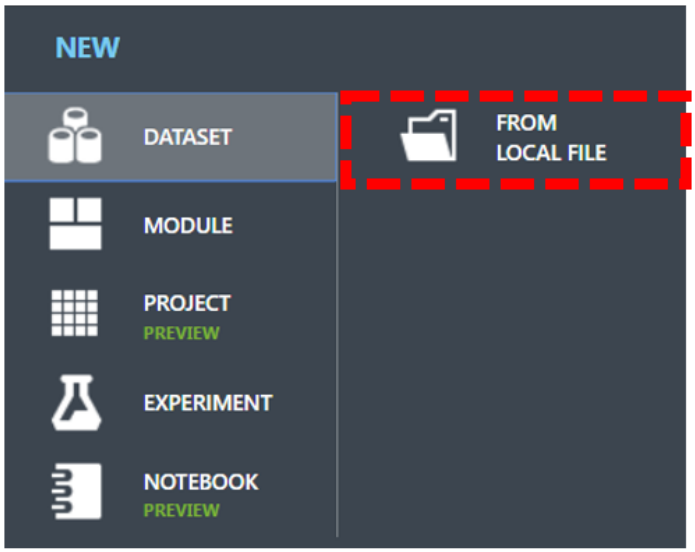


Figure 4.8: Upload dataset from local file

3. Please note that by default, Azure ML ships with a dataset called "Iris Two Class Data". To avoid confusion, give your dataset a unique name, then import.
4. To verify that your data has been imported, go into any experiment and look under the directory **Saved Datasets > My Datasets** (Figure: 4.9). You should see the name you chose for your data listed.

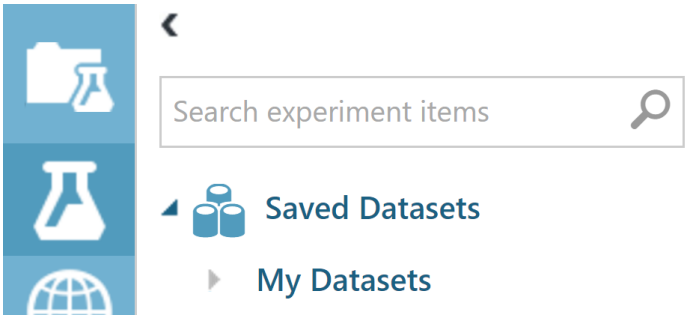


Figure 4.9: Saved dataset directory

4.2.2 Exercise: Reading a Dataset from the Web via a URL

1. To begin, use the search bar to find the **Import Data** module within your experiment. Drag and drop the module from the menu on the left (Figure: 4.10).

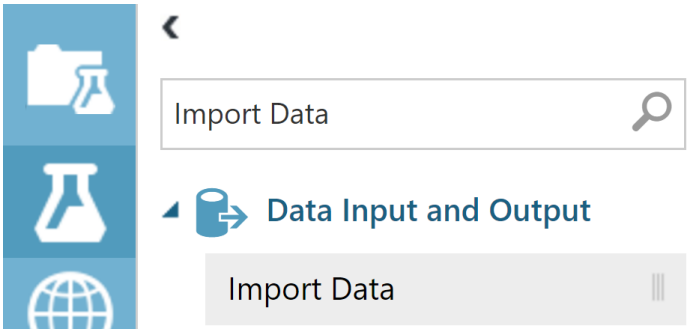


Figure 4.10: Search for the Import Data module

2. In the **Import Data** module settings for "Data source" select "Web URL via HTTP".
3. In the "URL" box, enter the URL of the iris data set:
   `http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data`
4. In the "Data format" drop down, select "CSV".
5. Leave "CSV or TSV has header row" unchecked (Figure: 4.11).

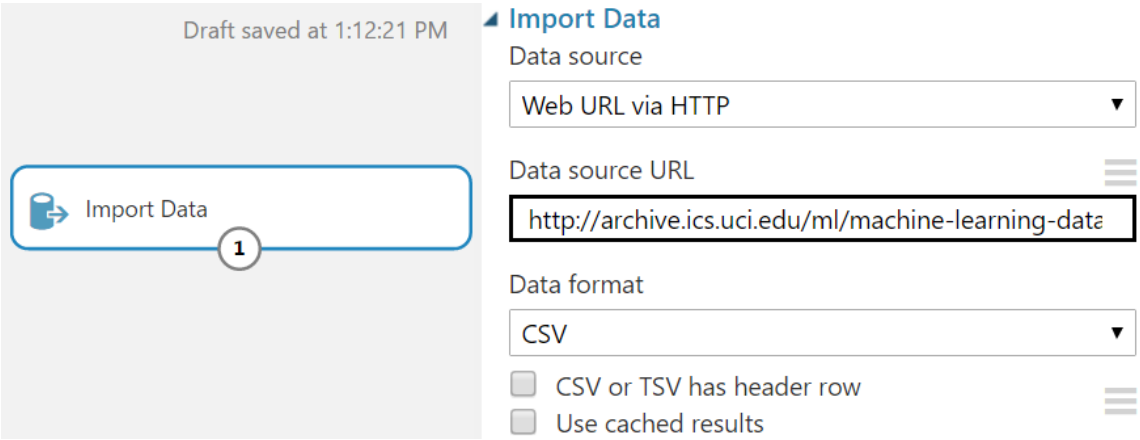


Figure 4.11: Import Data module settings

6. Select **Run** to import and parse the data into the experiment (Figure: 4.12).

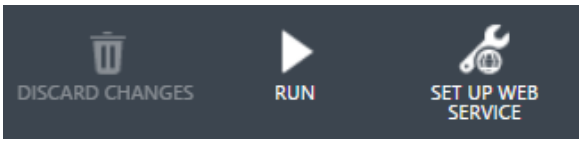


Figure 4.12: Run the experiment

7. In order to preserve the dataset, we must save it as a dataset. Save the output of your experiment by right-clicking the bottom middle node of the **Import Data** module again. Select "Save as Dataset" (Figure: 4.13). Please note that by default, Azure ML ships with a dataset called "Iris Two Class Data". To avoid confusion, give your dataset a unique name, then import.

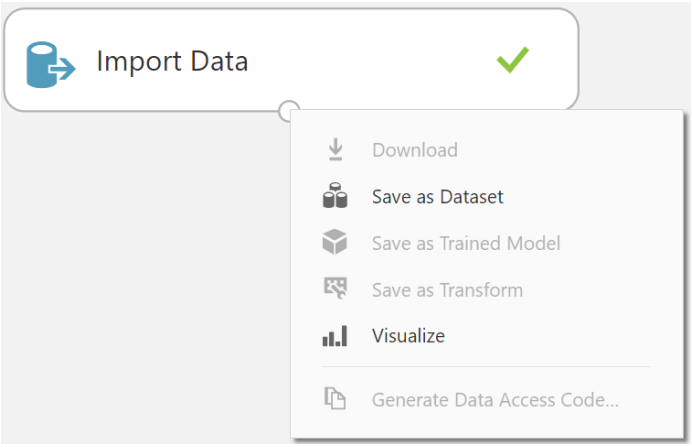


Figure 4.13: Save as Dataset

8. To verify that your dataset has been successfully imported, go into any experiment and look under the directory **Saved Datasets** (Figure: 4.14). You should see the name you chose for your data listed.

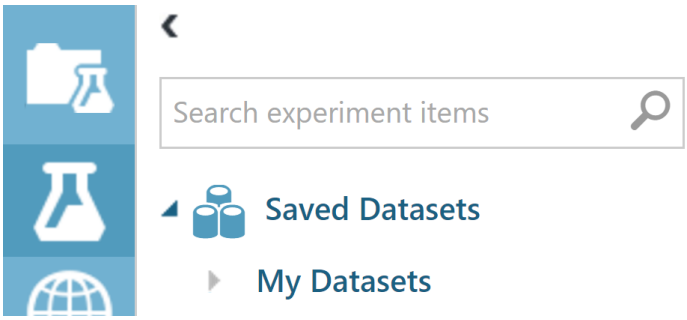


Figure 4.14: Save as Dataset

4.3 Visualizing, Exploring, Cleaning, and Manipulating Data

4.3.1 About the Data

For most of this chapter we will be working with the Titanic dataset. The Titanic data is a good beginner's dataset to start learning how to data mine. The sinking of the RMS Titanic occurred in 1912 and is one of the most infamous shipwrecks in history. 1,502 out of 2,224 passengers were killed in the tragedy and the incident caused an international backlash for ship safety reform. Although the mass loss of life is mainly attributed to the lack of life vessels and other elements of chance, some groups were more likely to survive than others. We will use the power of machine learning to uncover which types of individuals were more likely to survive.

Figure 4.15: Route of the Titanic's first and last maiden voyage. Source: Prioryman

The predictive model that we are going to build throughout Chapter 4.3-4.5 can be cloned at: `https://gallery.cortanaintelligence.com/Experiment/Building-an-Ensemble-Classifier-in-Azure-ML-1`.

There is also a condensed eVersion of Chapter 4.3-4.5 available at: `http://datasciencedojo.com/building-classification-model-azure-machine-learning-studio/`.
However, it is recommended that you follow along the book to build your predictive model step-by-step. To begin, we will first procure the dataset.

4.3.2 Exercise: Obtaining the Titanic sample data

You will be reading in the dataset by using the **Import Data** module.

1. Drag and drop a **Import Data** module from the menu on left (Figure: 4.16).

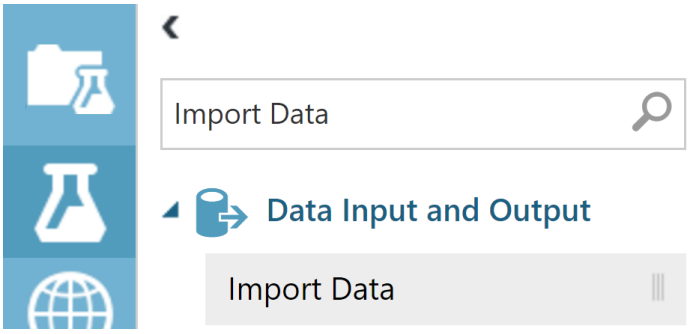


Figure 4.16: Search for the Import Data module

2. Set the **Import Data** module with the settings found in Table 4.1.

Field	Setting and Inputs
Data Source	Web URL via HTTP
URL	`http://download.datasciencedojo.com/titanic.csv`
Data format	CSV
CSV or TSV has header row	Checked
Use cached results	Checked

Table 4.1: Import Data module fields

Figure: 4.17 depicts a sample of what your **Import Data** module settings will look like after all of the fields are entered into the inputs.

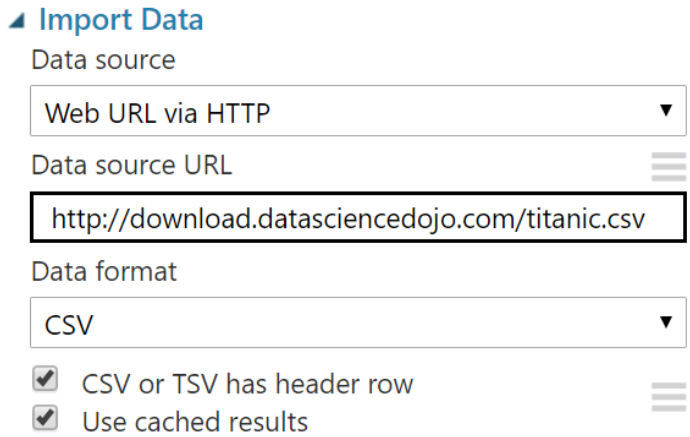


Figure 4.17: Import Data module settings

3. Select **Run** to execute the import and parse.

4. In order to preserve the dataset, we must save our work. Save the output of your experiment by right-clicking the bottom middle node of the **Import Data** module again. Select "Save as Dataset" (Figure: 4.18).

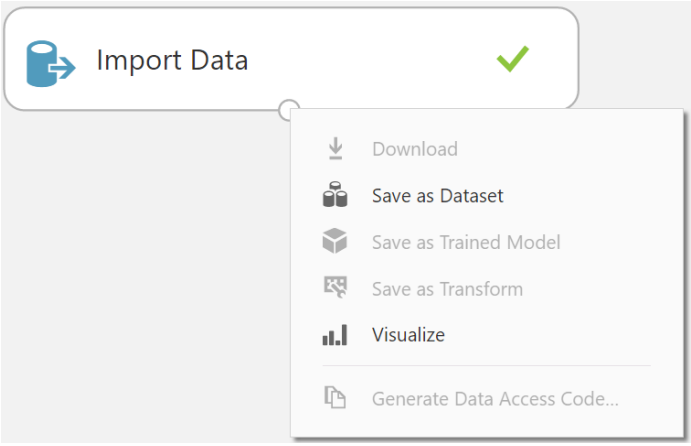


Figure 4.18: Save as Dataset

5. Save your dataset with the name "Titanic Dataset" (Figure: 4.19).

Figure 4.19: Naming the Import Data module dataset

6. To verify that your dataset has been successfully imported, go into any experiment and look under the directory **Saved Datasets**. You should see the "Titanic Dataset" listed (Figure: 4.20).

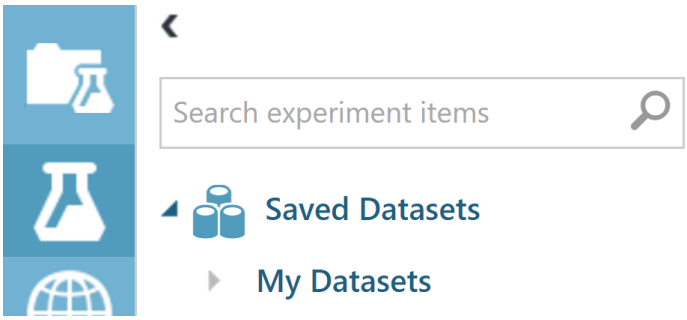


Figure 4.20: View saved Datasets

7. Drag and drop a new **Titanic Dataset** module from the left menu (Figure: 4.21).

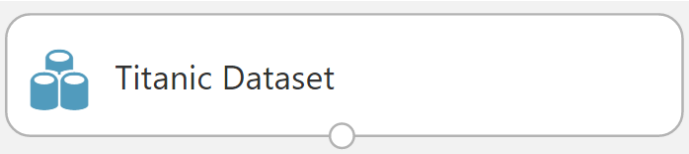


Figure 4.21: **Titanic Dataset** module

8. Right-click on the **Titanic Dataset** module and select “Visualize" (Figure: 4.22).

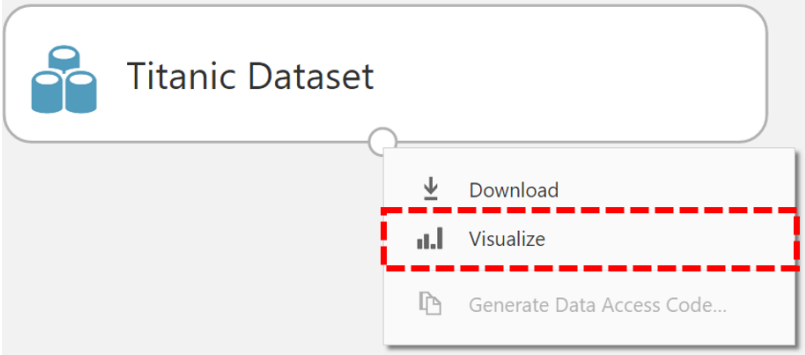


Figure 4.22: Visualize the **Titanic Dataset** module

rows 891 columns 12

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
view as									
	1	0	3	Braund, Mr. Owen Harris	male	22	1	0	A/5 21171
	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Thayer)	female	38	1	0	PC 17599
	3	1	3	Heikkinen, Miss. Laina	female	26	0	0	STON/O2. 3101282

Figure 4.23: Data from the **Titanic Dataset** module visualization

4.3.3 Titanic Dataset Key

The Titanic dataset is made up of qualitative and quantitative information. The Titanic key describes the features and their corresponding values.
The key can also be found at:
`https://www.kaggle.com/c/titanic/data`.

Variable Descriptions

Column Name	Meaning	Notes
Survived	Survival	(0 = No; 1 = Yes)
Pclass	Passenger Class	(1 = 1st; 2 = 2nd; 3 = 3rd)
Name	Name	
Sex	Gender	
Age	Age	
SibSp	Number of Siblings/Spouses Aboard	
Parch	Number of Parents/Children Aboard	
Ticket	Ticket Number	
Fare	Passenger Fare	In 1910 USD
Cabin	Cabin	
Embarked	Port of Embarkation	(C = Cherbourg; Q = Queenstown; S = Southampton)

Special Notes

- Pclass is a way to infer socio-economic status
  1st Upper; 2nd Middle; 3rd Lower
- Age is in Years; age is fractional if the passenger age is less than one
  If the age is Estimated it is in the form "xx.5"
- With respect to the family relation variables (i.e. sibsp and parch) some relations were ignored. The following are the definitions used for sibsp and parch.
  - Sibling: Brother, sister, stepbrother, or stepsister of the passenger
  - Spouse: Husband or wife of the passenger (mistresses and fiances Ignored)
  - Parent: Mother or father of the passenger
  - Child: Son, daughter, stepson, or stepdaughter of the passenger
- Other family relatives excluded from this study include cousins, nephews, nieces, aunts, uncles, and in-laws. Some children travelled only with a nanny, therefore parch=0 for them. In addition, some passengers travelled with close friends or neighbors in a village, however, the definitions do not support such relations.

4.3.4 Exercise: Removing Your Columns

Some of the columns in the dataset hold little data mining value in its current form. For example, "PassengerId" seems like a database primary key, holding no value other than being a unique identifier. The same applies for "Name", and "Ticket", both holding little value in this context. "Cabin" contains too many missing values and will also be dropped.

1. We will continue building off of the other modules already in your workspace. Drag in the **Titanic Dataset** module from the previous chapter.

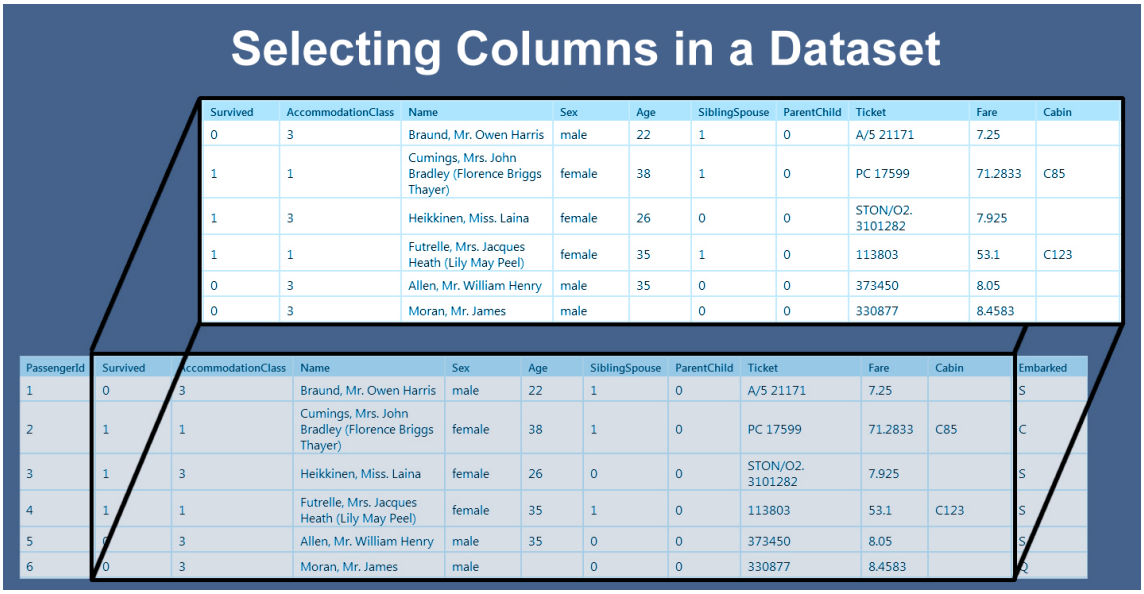


Figure 4.24: In this example of selecting columns, all columns except for "Embarked" and "PassengerID" are being selected, essentially dropping these columns.

2. Connect the output node of the **Titanic Dataset** module to the input node of the **Select Columns in Dataset** module (Figure: 4.25).

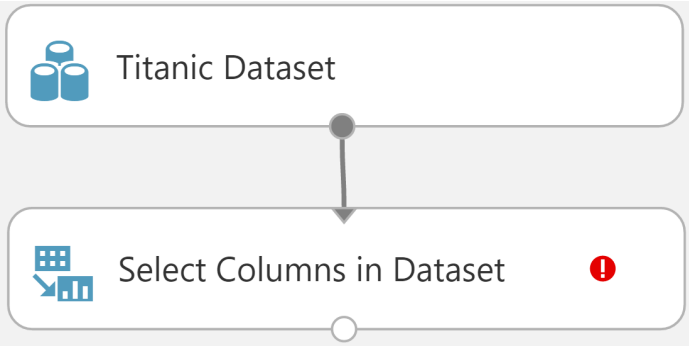


Figure 4.25: Connecting the **Titanic Dataset** module to the **Select Columns in Dataset** module

3. Click on the **Select Columns in Dataset** module and a menu on the right of your screen will show up (Figure: 4.26).

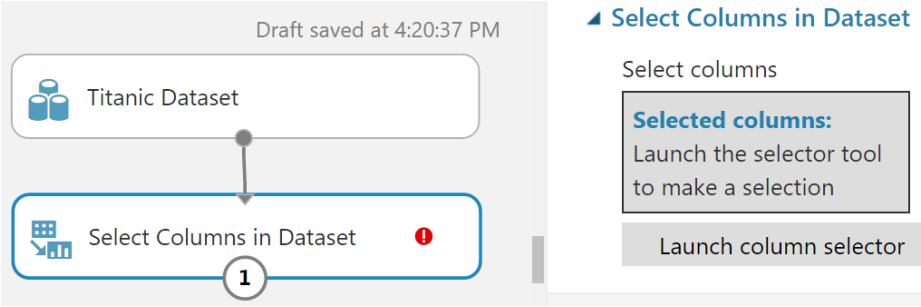


Figure 4.26: **Select Columns in Dataset** module menu

4. Select "Launch column selector", with rules, begin with all, and exclude "PassengerId", "Ticket", "Name", and "Cabin" (Figure: 4.27, 4.28).

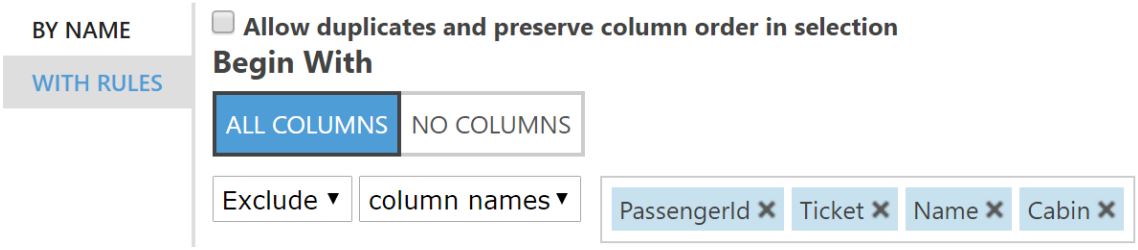


Figure 4.27: Column selector settings

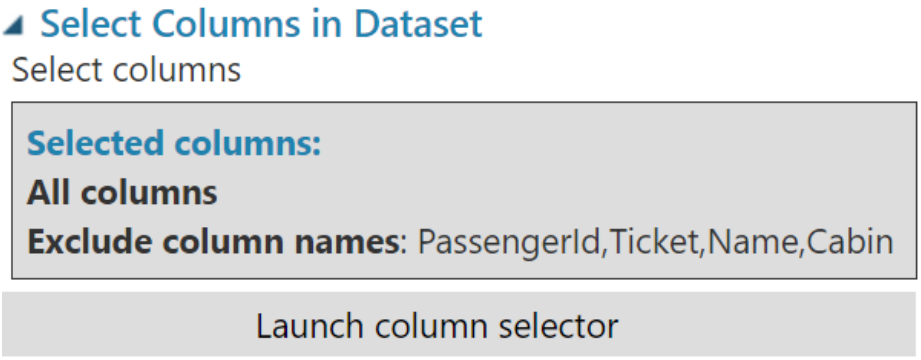


Figure 4.28: Included columns in data set

5. For clarity you can label your **Select Columns in Dataset** module "Dropping PassengerId, Name, Ticket, Cabin" (Figure: 4.29).

6. **Run** the workspace then right-click on the bottom middle node of the **Select Columns in Dataset** module and select "Visualize" (Figure: 4.29).

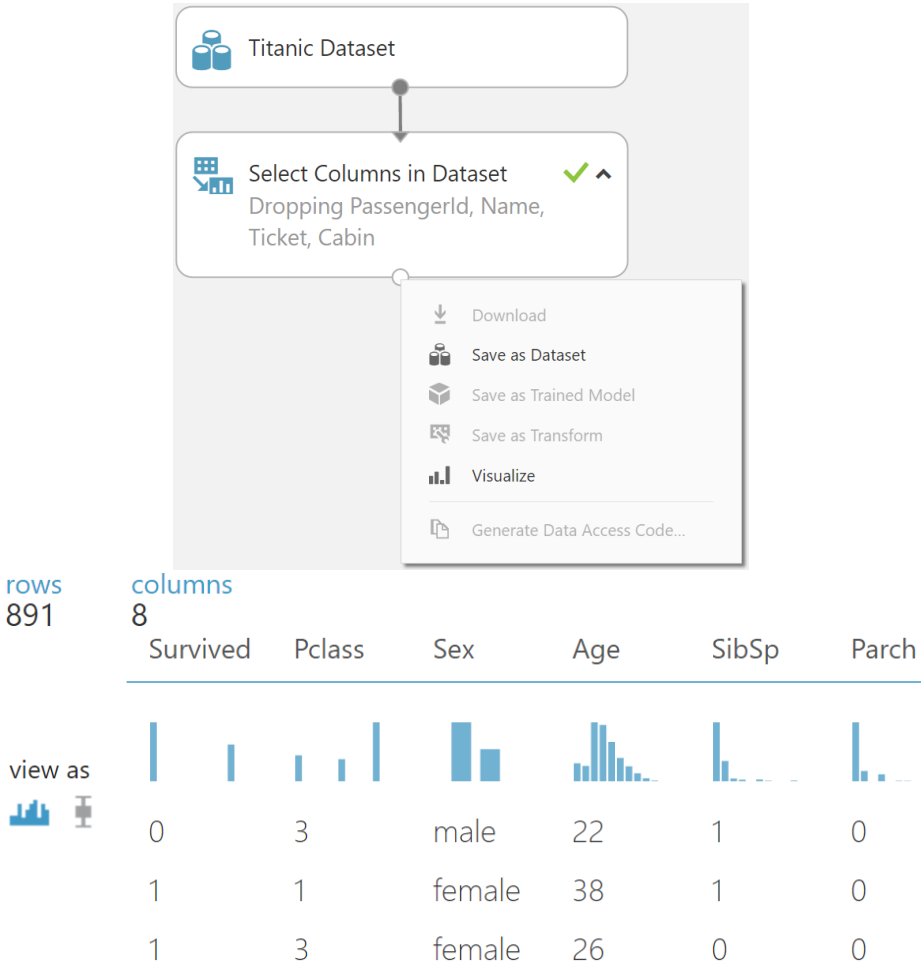


Figure 4.29: **Select Columns in Dataset** module visualization

4.3.5 Exercise: Casting Columns

We must now define which features are non-continuous by casting them as categorical. Mathematical approaches for continuous and non-continuous values differ greatly. To ensure proper mathematical treatment by the machine learning algorithm, identify and cast those features into categories.
The Titanic dataset contains categorical data types, by default Azure will treat them as sequential numbers. Therefore we must tell Azure which columns are categorical.

1. Right-click on the bottom center node of the Select Columns in Dataset module. Select "Visualize" to see the output. Verify that it looks like Figure: 4.18.
2. Select the "Survived" column and toggle the "Statistics and Visualizations" menu on the right. Notice that the "Feature Type" under the "Statistics" dropdown is "Numeric Feature", this will have to be changed to categorical (Figure: 4.30).

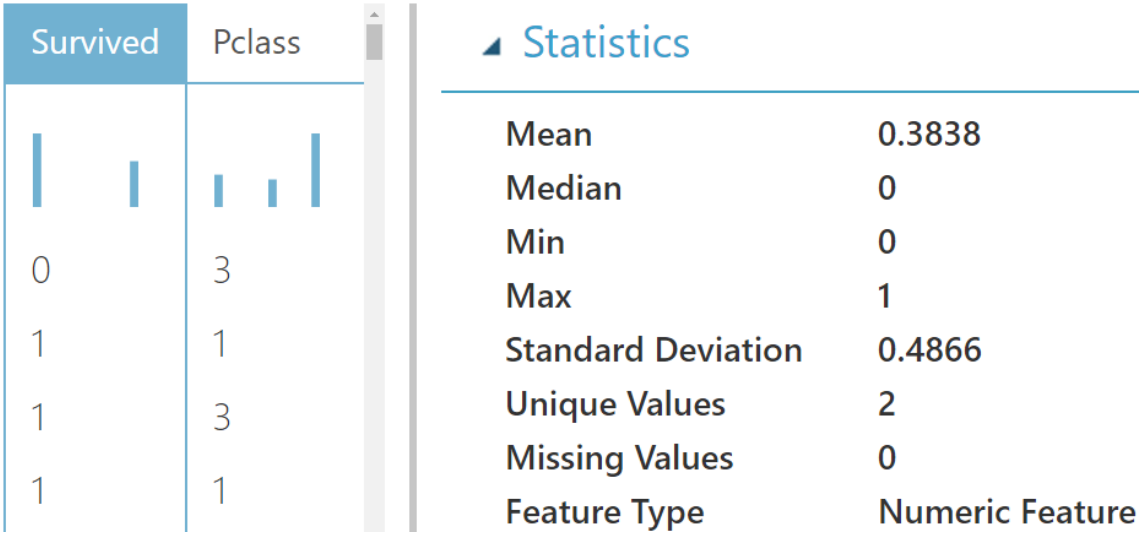


Figure 4.30: Statistics of the Survived column

3. To begin, search for the **Edit Metadata** module in the left menu and drag the module into the workspace. Connect it to the **Select Columns in Dataset** module. (Figure: 4.31).

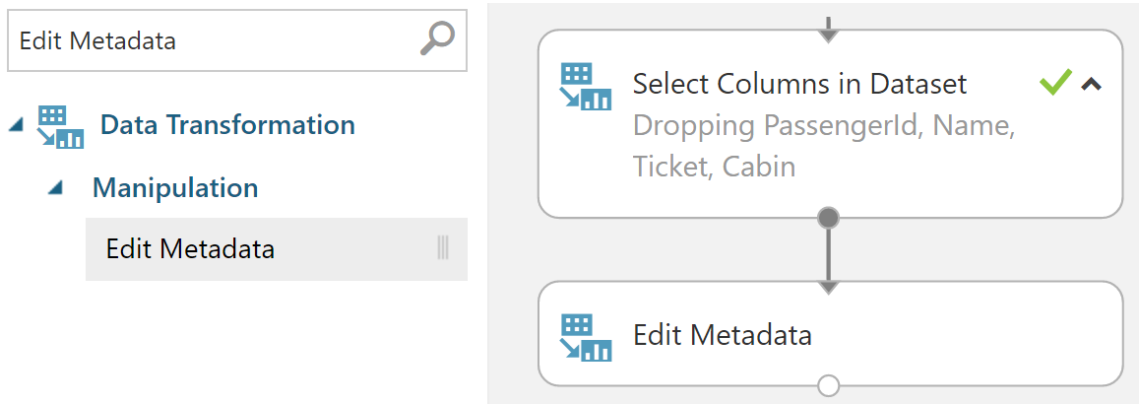


Figure 4.31: Drag the **Edit Metadata** module into the workspace and connect it to Select Columns in Dataset module

4. Select "Launch column selector", begin with no columns, include the columns: "Survived", "Sex", "Pclass", "Embarked", and "PassengerId" to be edited (Figure: 4.32).

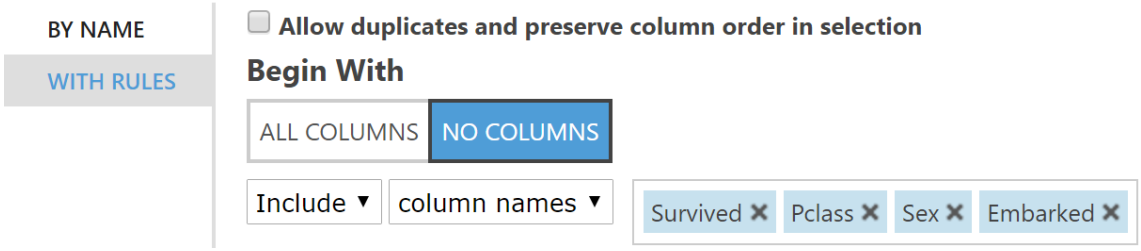


Figure 4.32: Included columns in column selector settings

5. After selecting the columns to cast in the "column selector", change the "Categorical" field to "Make Categorical" in the **Edit Metadata** module. Keep all other fields as they are (Figure: 4.33) and **Run** the workspace.

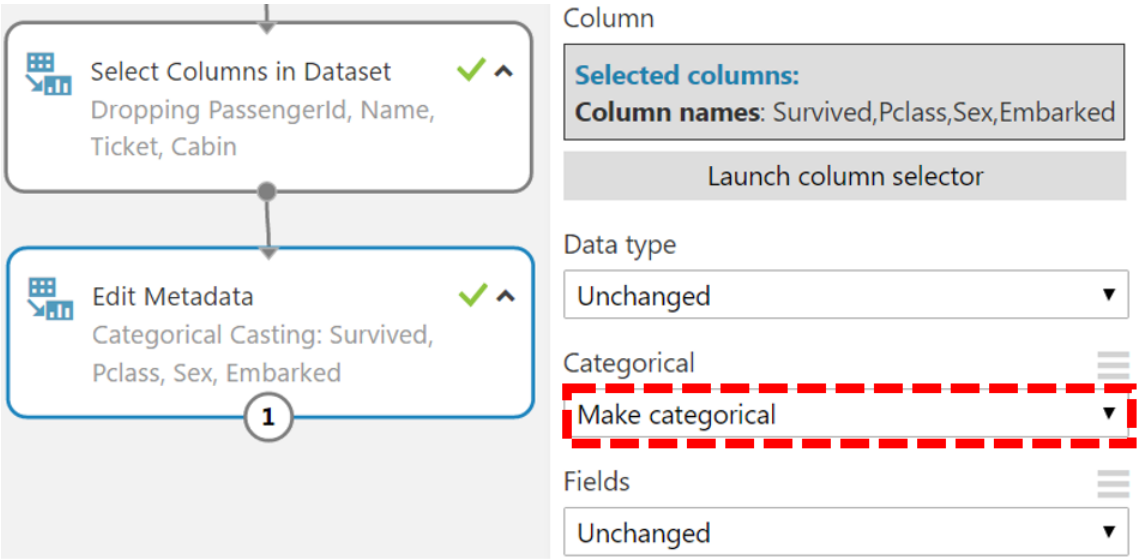


Figure 4.33: **Edit Metadata** module settings

6. It is a good habit to label the different modules so you can remember their function as more modules get added to the workspace. To do this, double click on the module and type the label. For this **Edit Metadata** module, label it "Categorical Casting: Survived, Pclass, Sex, Embarked".

4.3.6 Exercise: Data Visualization and Exploration

1. One way to view the distribution of your data is with a histogram. To do this, right-click on the **Edit Metadata** module and select "Visualize". Select the "Survived" column and make sure the "view as" is set to the picture of a histogram (Figure: 4.34)

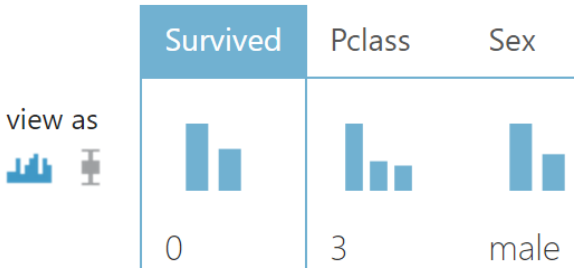


Figure 4.34: "View as" settings

2. The "Statistics and Visualizations" menu will populate the right hand side of the screen. A histogram like that in Figure: 4.35 will now appear in the window.
How is the data distributed in terms of survived and deceased passengers? Did more people survive or perish? (Figure: 4.36).

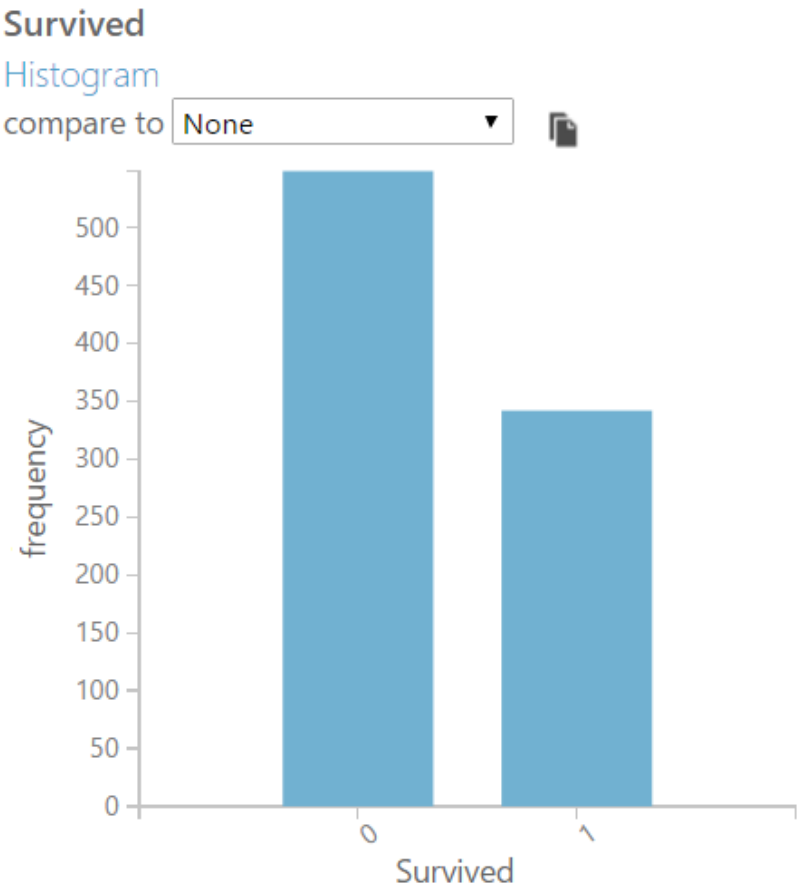


Figure 4.35: Histogram of the frequency of survival

3. Select the "Embarked" column from the left hand side. What do you notice about this histogram? Which port of embarkation did most of the passengers come from? The least?

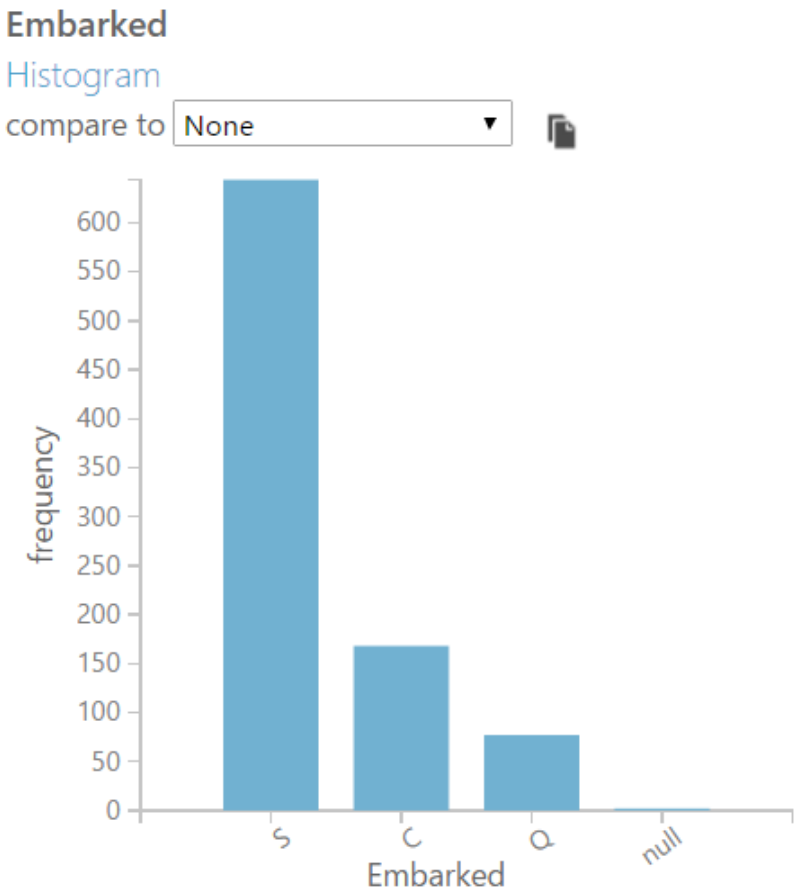


Figure 4.36: Histogram of the frequency each location was embarked from

4. It's possible to compare different variables together in the same visualization. Let's begin with comparing gender and survival. Select the "Survival" column again. On the top of the histogram under "compare to" select "Sex" (Figure: 4.37). Based on the graph, did gender play a part in the chances of survival?
5. Change "compare to" to "age" (Figure: 4.37). Is there a relationship between age and survival? What was the oldest age of survival? The oldest age of death?

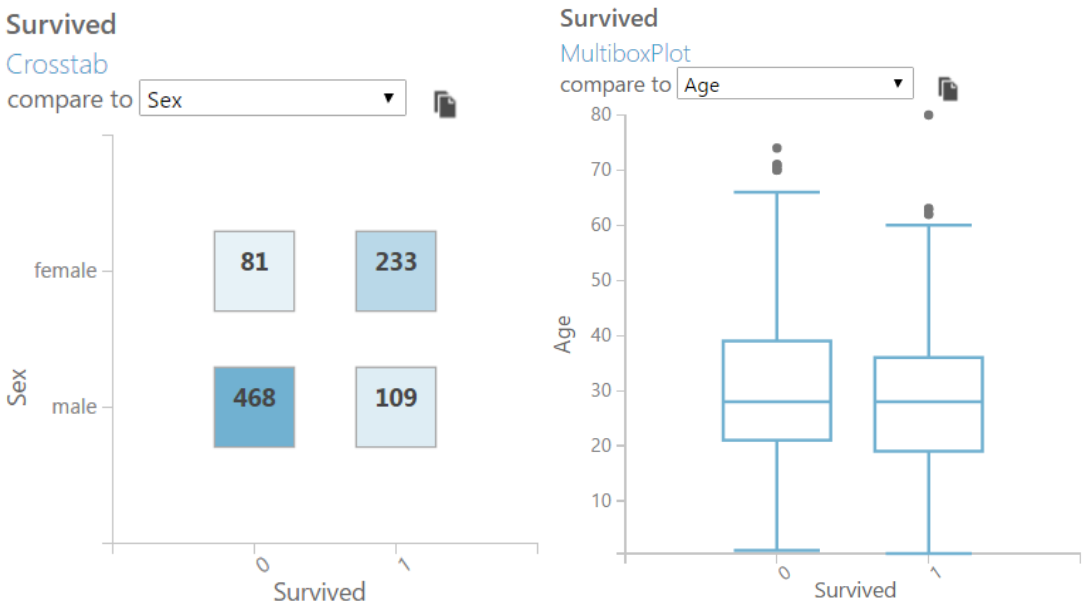


Figure 4.37: Crosstab of “Survived" compared to “Sex" and MultiboxPlot of “Survived" compared to “Age"

6. Select the “Sex” column from the left hand side. Set the “compare to” dropdown to “Age” (Figure: 4.38). What was the distribution of males and females among age groups? Which group had an older distribution of passengers?
7. Set the “compare to” dropdown to “Fare”. Make sure “log scale” is checked (Figure: 4.38). What is the relationship between gender and ticket price? Which category generally paid more?

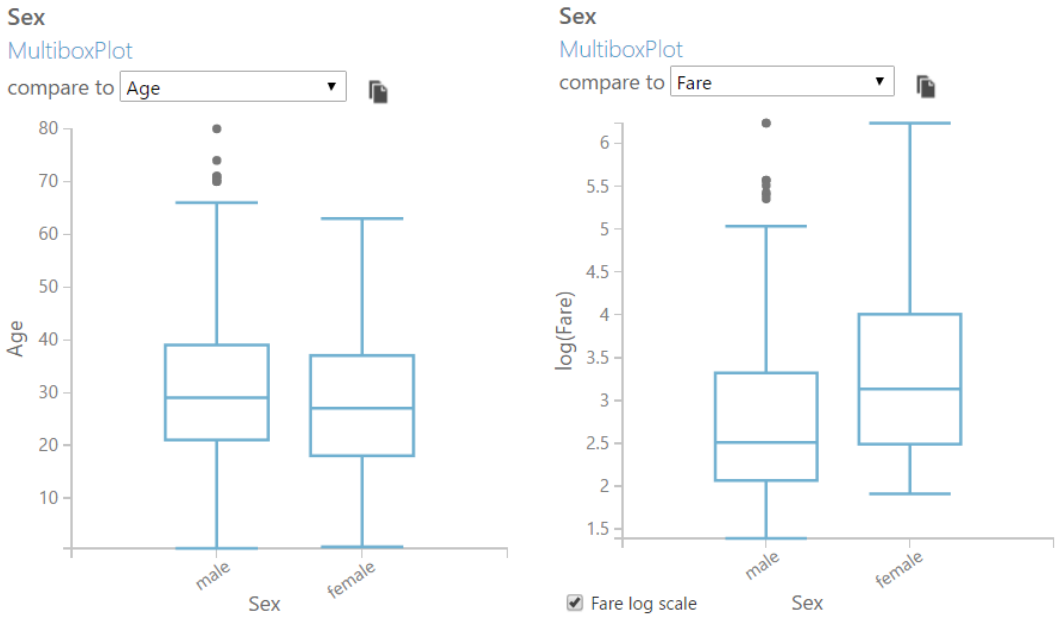


Figure 4.38: MultiboxPlot of “Sex" compared to “Age" and MultiboxPlot of “Sex" compared to “Fare"

4.3.7 Exercise: Using Summarize Data and Cleaning Missing Data

Most algorithms are unable to account for missing values and some treat it inconsistently from others. To address this, we must make sure our dataset contains no missing, blank, "null", or "NA" values. First we must identify the columns that process missing values. We will use the **Summarize Data** module to provide us with this information.

1. From the left menu, search for the **Summarize Data** module and connect it to the bottom of the existing **Edit Metadata** module (Figure: 4.39).

Figure 4.39: **Summarize Data** module

2. **Run** the experiment, right-click on the **Summarize Data** module and select "Visualize" to view the output (Figure: 4.40).

Feature	Count	Unique Value Count	Missing Value Count
Survived	891	2	0
Pclass	891	3	0
Sex	891	2	0
Age	714	88	177
SibSp	891	7	0
Parch	891	7	0
Fare	891	248	0
Embarked	889	4	2

Figure 4.40: **Summarize Data** module visualization

3. Exit the **Summarize Data** module visualization and visualize the output of the **Edit Metadata** module.
   What do you notice about the information provided? What was the mean age of travellers on the Titanic? What was the age of the oldest person? The youngest? How do you interpret the youngest age? What was the lowest fare price? The median?
   Next, look at the "Missing Value Count" column. Which columns have missing values? For example "Age" has 177 missing values.

4. "Age" and "Embarked" both have missing values. For each of these columns we want to learn the feature type. To do this, select the corresponding column name in the visualization output. Expand the "Statistics" dropdown on the left and take note of the "Feature Type" (e.g. "Age" is Numeric). Repeat this step for each column (Figure: 4.41).

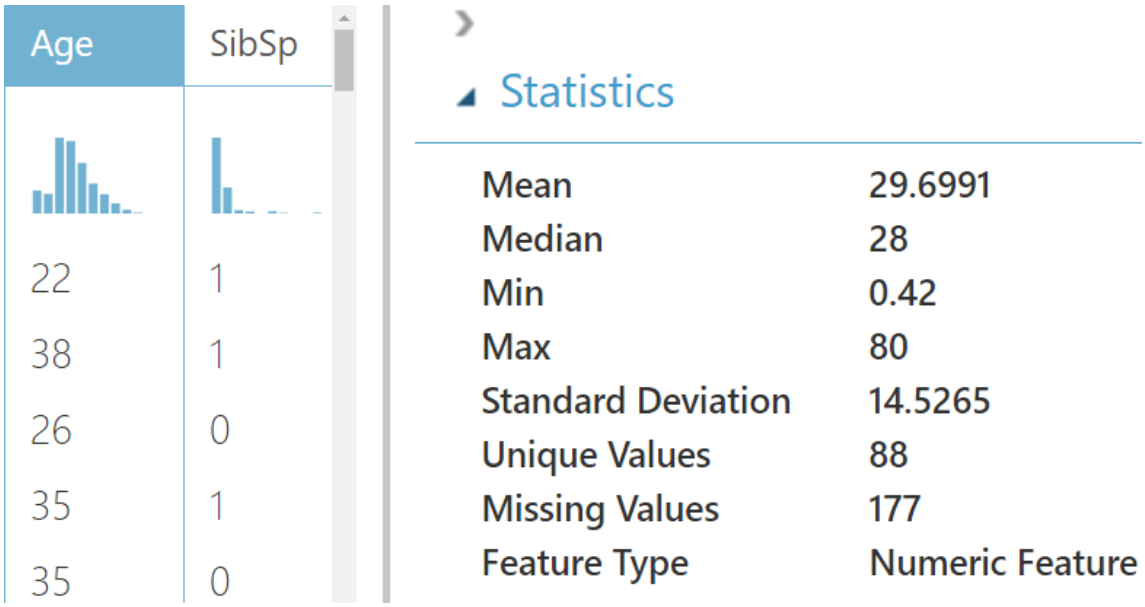


Figure 4.41: Statistics and Feature Type

Now that we have performed an analysis on the missing values and identified some of their features, we can develop a cleaning strategy.

- "Age" has 177 missing values and is numeric. Dropping this column would result in a significant loss of data. However, since we know that all "Age" values are numeric, we can easily replace these missing values with the median of the dataset without too much information loss.
- "Embarked" is only missing two values and is categorical. Because there aren't many missing values, dropping the rows of missing values will not have a great impact on the data.

5. To clean all missing values for numeric columns, we will use a **Clean Missing Data** module. Search for the **Clean Missing Data** module in the left menu, drag it into the workspace, and connect it to the **Edit Metadata** module. (Figure: 4.42). The clean missing data module can clean individual columns.

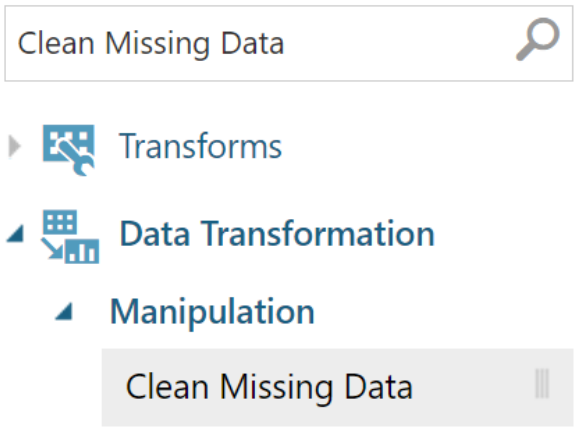


Figure 4.42: **Clean Missing Data** module

6. (a) To select all numeric columns, "Launch column selector" in the **Clean Missing Data** module.
   (b) Select the column "With rules" then select "No columns"
   (c) Choose to "Include" from the dropdown box, "column type" from the dropdown box and the "Numeric" column type. (Figure: 4.43).

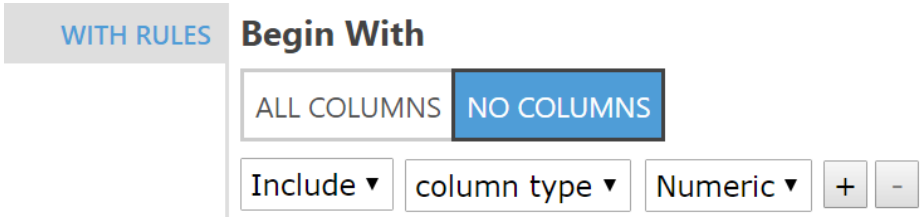


Figure 4.43: Include numeric columns

7. Select "Replace with median" from the "Cleaning Mode" dropdown. (Figure: 4.44).

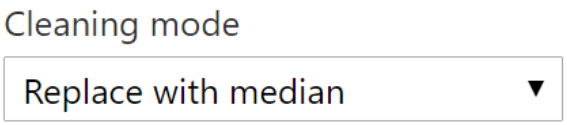


Figure 4.44: Replace with median

8. Label the **Clean Missing Data** module with "Replace all numerics with median" and **Run** your workspace.

   Tip You can always check what the median value will be by using to the **Summarize Data** module.

9. Now we will replace all categorical missing values with the mode. Drag in a second **Clean Missing Data** module again. Search for this module in the left menu, drag it into the workspace, and connect it to the previous **Clean Missing Data** module (Figure: 4.45).

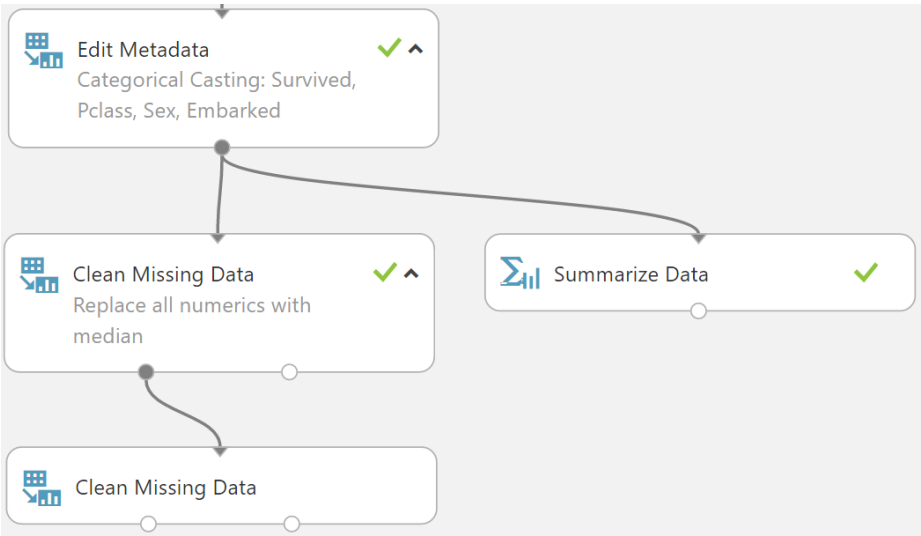


Figure 4.45: Connect the **Clean Missing Data** module

10. (a) Select the "Launch column selector" in the **Clean Missing Data** module.
    (b) Select the "With rules" then select "No columns".
    (c) Choose to "Include" from the dropdown box, "column type" from the dropdown box, and the "Categorical" column type.
11. Select "Replace with mode" from the "Cleaning Mode" dropdown and label the **Clean Missing Data** module with "Replace all categorical with mode" (Figure: 4.46).

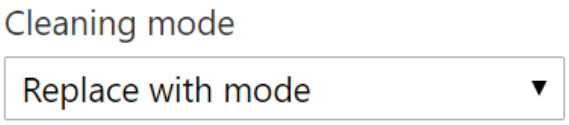


Figure 4.46: Replace with mode

12. **Run** the experiment and verify that you're workspace looks like Figure 4.47.

The data is now ready for a machine learning model.

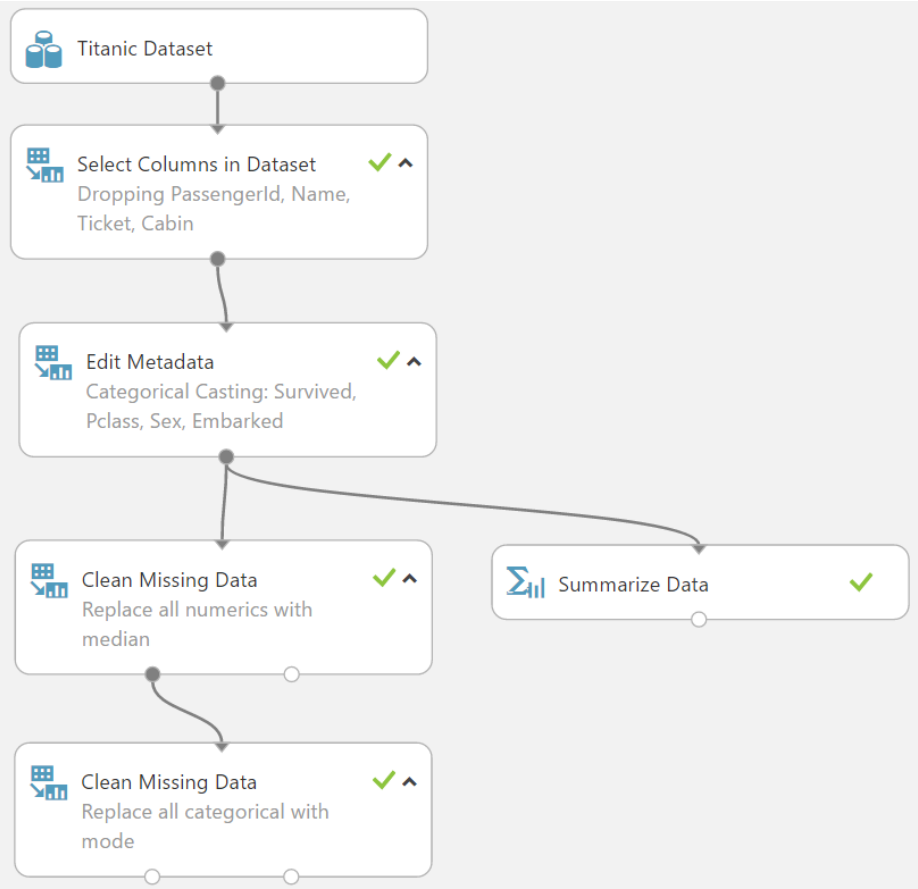


Figure 4.47: Final Result

4.4 Building a Classification Model in Azure ML

Before we start building our classification model we need to identify what kind of machine learning problem we are dealing with. We want our model to predict the chance of survival using the "Survived" column as our response class where the result is either "0" or "1", dead or alive. Because of these possible values we can tell that our problem falls into what is called a binary classification problem. Now that we have this information we will now build an Azure ML Studio classification model to predict the "Survived" column.

4.4.1 Exercise: Response Class and Partitioning Datasets

We will have to partition our data into two parts. One partition will be used to build the model while the other partition will be hidden away to evaluate the performance of the model. The holdout dataset will be data that the model has not yet seen. The assumption is if the model can do well on this holdout set that it will also do well on future unknown datasets. These partitions are generally called the train and test set.

1. In the left menu search for the **Split Data** module, drag the module into your workspace, and connect it to the **Clean Missing Data** module.
2. In the **Split Data** module set the "Fraction of rows in the first output" to 0.7. This means that 70% of the data will be partitioned to the left node and 30% will be partitioned to the right. 4.48
3. Check the "Randomized split" checkbx. This split is visualized in Figure: 4.49.

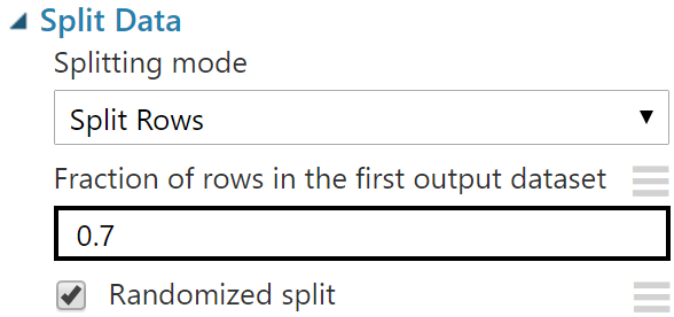


Figure 4.48: Partitioning of data in **Split Data** module settings

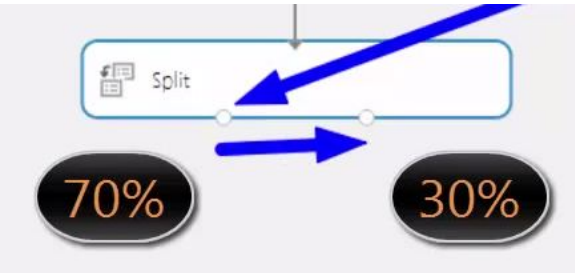
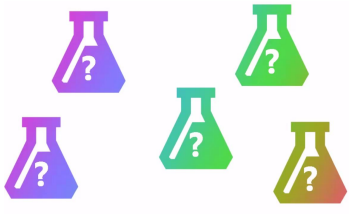


Figure 4.49: Partitioning of data visualization

4.4.2 Exercise: Algorithm Selection and Training



1. In the left menu under **Machine Learning > Initialize Model > Classification** you will see various classification models. For our experiment any of the "Two-Class" algorithms will work (Figure: 4.50). For the purpose of this exercise, drag in the classification algorithm module labeled **Two-Class Decision Forest** (Figure: 4.51).

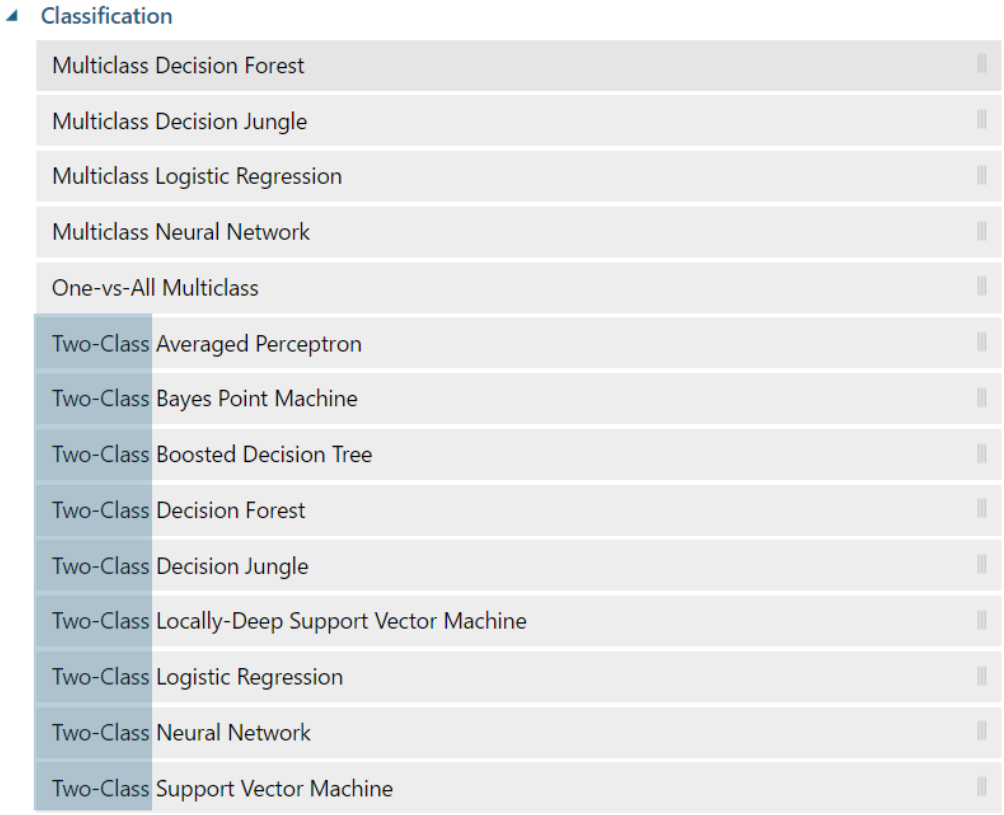


Figure 4.50: List of two-class classification algorithms.

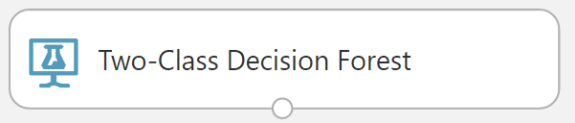


Figure 4.51: **Two-Class Decision Forest** module

2. In the left menu search for the **Train Model** module and drag it into your workspace. (Figure: 4.52). 70% of the data from the **Split Data** module must be exposed to the learning algorithm to train the model. The output of the train model module will be a predictive model.

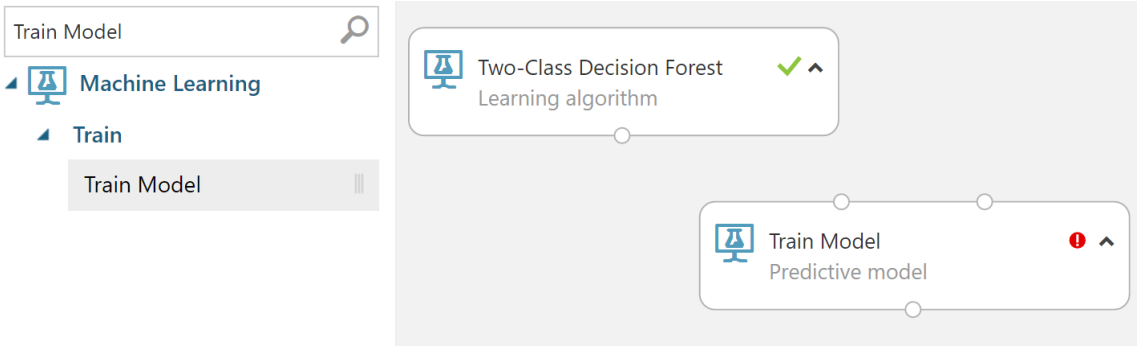


Figure 4.52: **Train Model** module

3. Connect the **Two-Class Decision Tree** module to the top left node of the **Train Model** module (Figure: 4.53).

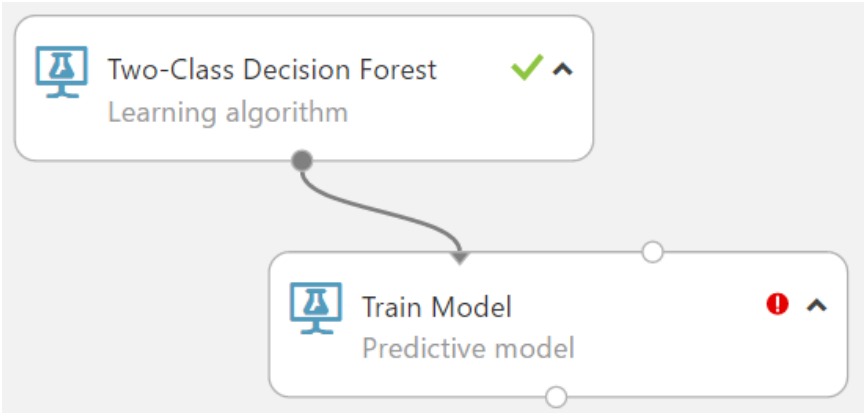


Figure 4.53: **Two-Class Decision Forest** module connected to the **Train Model** module

4. Connect the bottom-left-output node of the **Split Data** module (70%) to the top-right-input node of the **Train Model** module (Figure: 4.54).

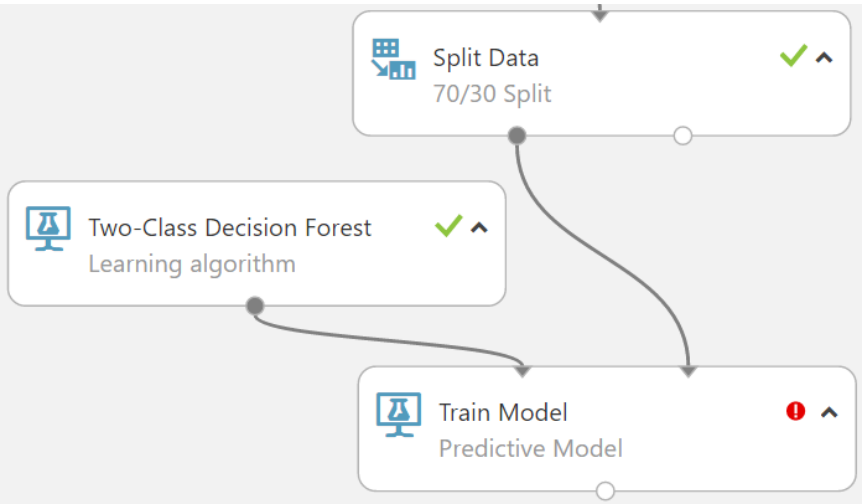


Figure 4.54: The left-output node of the **Split Data** module (70%) connected to the right-input node of the **Train Model** module

5. Now that that 70% of the data from the **Split Data** module is connected to the **Train Model** module, we must tell the algorithm which feature it is trying to predict. Click on the **Train Model** module in your workspace, select "Launch column selector", select the column with rules by changing the dropdown settings to "Include" and "column names", and enter "Survived" in the textbox (Figure: 4.55). **Run** your Workspace and now you will have a trained predictive model.

Figure 4.55: Column selector

6. Now that we have a predictive model, we will use the **Scored Model** module to generate predictions using 30% of the partitioned data from the **Split Data** module. The 30% dataset also happens to have the original "Survived" column, the ground truth of what happened, in addition to the predicted values. We can easily compare "predicted" vs "actual" ("scored labels" vs "survived" in this case).
7. In the left menu search for the **Score Model** module and drag it into your workspace (Figure: 4.56).

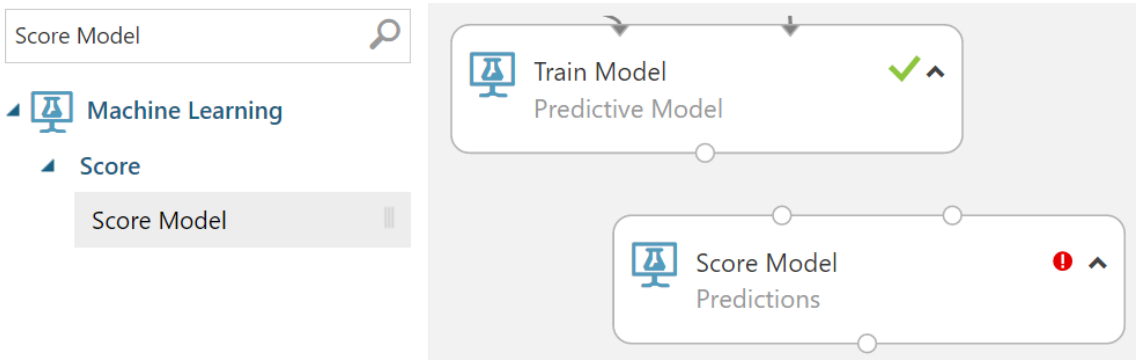


Figure 4.56: **Score Model** module

8. Connect the bottom node of the **Train Model** module to the top left node of the **Score Model** module (Figure: 4.57).

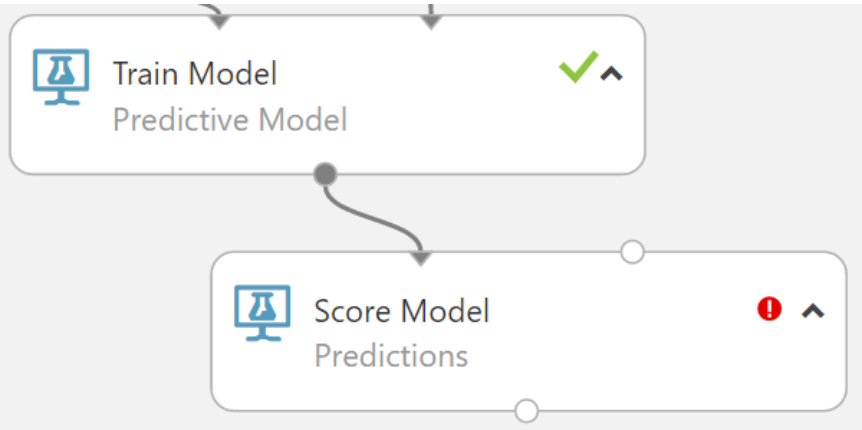


Figure 4.57: **Train Model** module connected to the **Score Model** module

9. Connect the bottom-right-output node of the **Split Data** module (30%) to the top-right-input node of the **Score Model** module (Figure: 4.58) and **Run** your workspace.

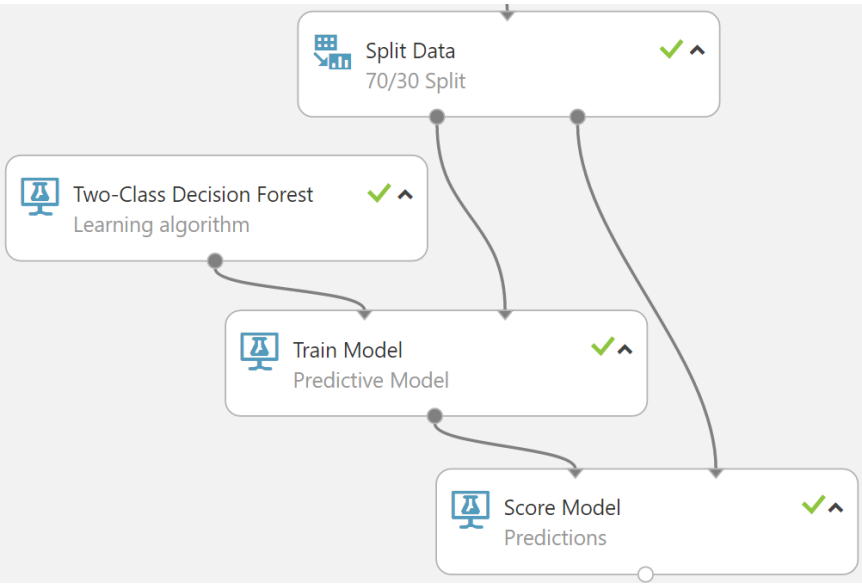


Figure 4.58: The right-output-node of the **Split Data** module (30%) connected to the right-input node of the **Score Model** module

10. We want to see if the model accurately predicted the withheld split data. Right-click on the **Score Model** module and visualize the output. Again your values will be different but your results should be similar to Figure: 4.59. Notice that we have two additional columns, “Scored Labels" and “Scored Probabilities".

Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Scored Labels	Scored Probabilities
0	3	male	33	0	0	7.775	S	0	0.098392
1	1	female	38	0	0	227.525	C	1	0.984375
1	1	male	52	0	0	30.5	S	0	0.455423
1	3	male	32	0	0	7.925	S	0	0.244067

Figure 4.59: Visualization of the **Score Model** module

The “Scored probability" column represents what the model thinks the probability of this particular observation being class “1" based upon the other predictors. In this case it is the prediction that this person will “survive" the Titanic. The “Scored label" represents derived results from the scored probability column. Probabilities greater than 50% will be cast to 1 (being alive) while probabilities less than 50% will be cast to 0 (being dead).

From Figure: 4.59, you can see that a 38-year-old, female traveling in 1st class has about an 98% chance of survival. The scored model casted her as a “1” meaning our model thinks she survived. In the “Survived" she was also marked as a “1” so it can be inferred that the model predicted her survival correctly.

However, the model won’t always be correct. In Figure: 4.59, you can see that a 32-year old male traveling in 3rd class has about a 24% chance of survival. The model casted him as a “0” meaning

our model thinks he died. In actuality, you can see from the dataset that he is marked with a "1" in the "Survived" column. This means he actually survived, making the model's prediction incorrect.

We can go down, line by line, and tally up the instances where the model was right or wrong but that will take a long time. Luckily, the **Evaluate Model** module tells us the aggregates of predicted vs actual for us. This aggregate comparison of predicted vs actual is done using the **Evaluate Model** module, which will build a confusion matrix to summarize results of each observation in the holdout set.

11. In the left menu search for the **Evaluate Model** module and drag it into your workspace (Figure: 4.60).

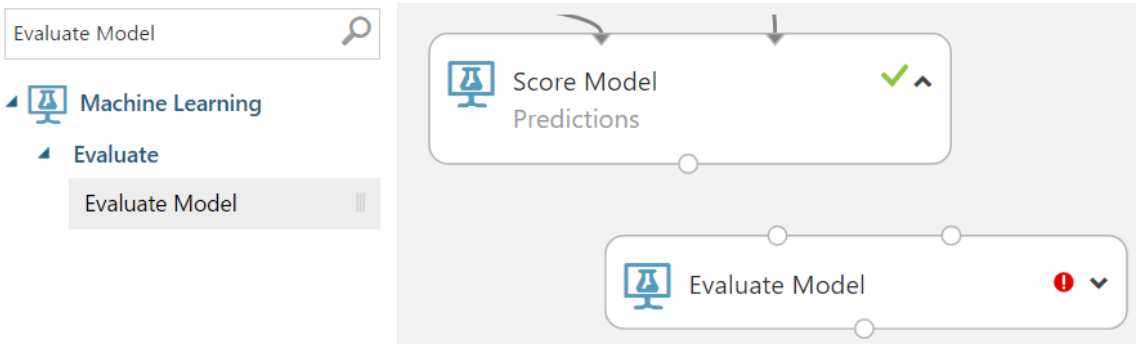


Figure 4.60: **Evaluate Model** module

12. Connect the bottom node of the **Score Model** module to the top left node of the **Evaluate Model** module (Figure: 4.61).

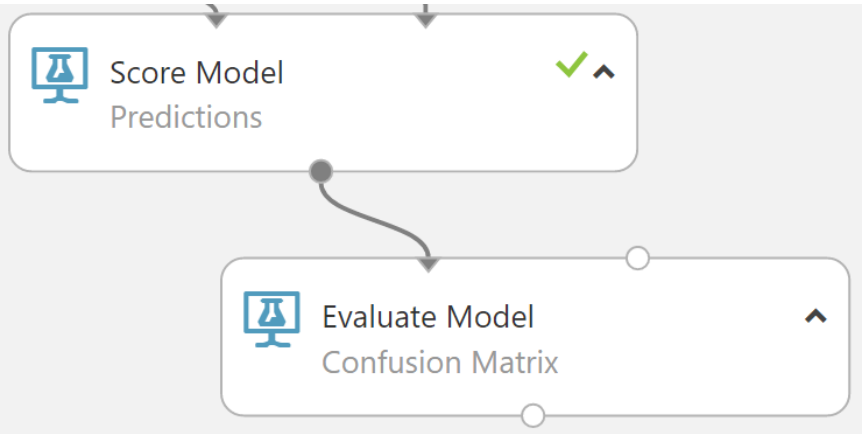


Figure 4.61: **Score Model** module connected to the **Evaluate Model** module

13. When you are finished connecting all of the modules, **Run** your workspace. Your workspace should look like Figure: 4.62

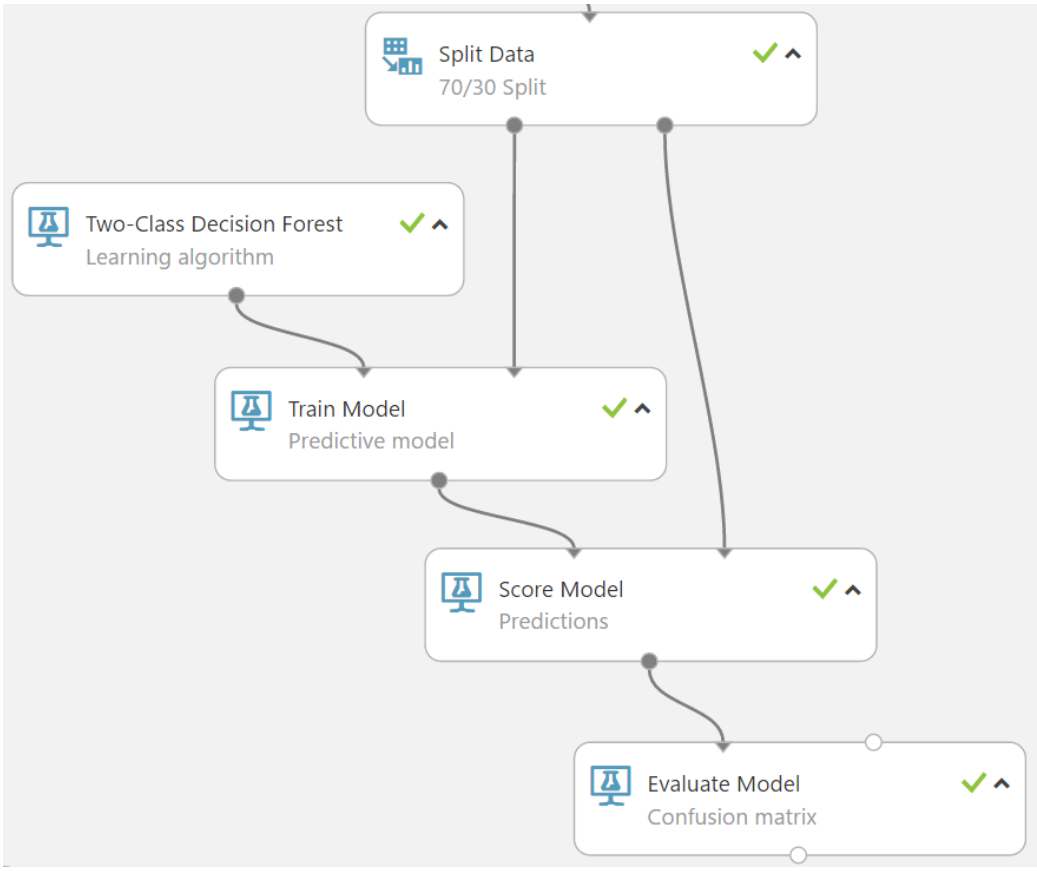


Figure 4.62: Output of connected modules

14. Right-click the bottom node of the **Evaluate Model** module and select "Visualize" to view the output of your experiment. Although your model will have different values your visualization should be similar to Figure: 4.63

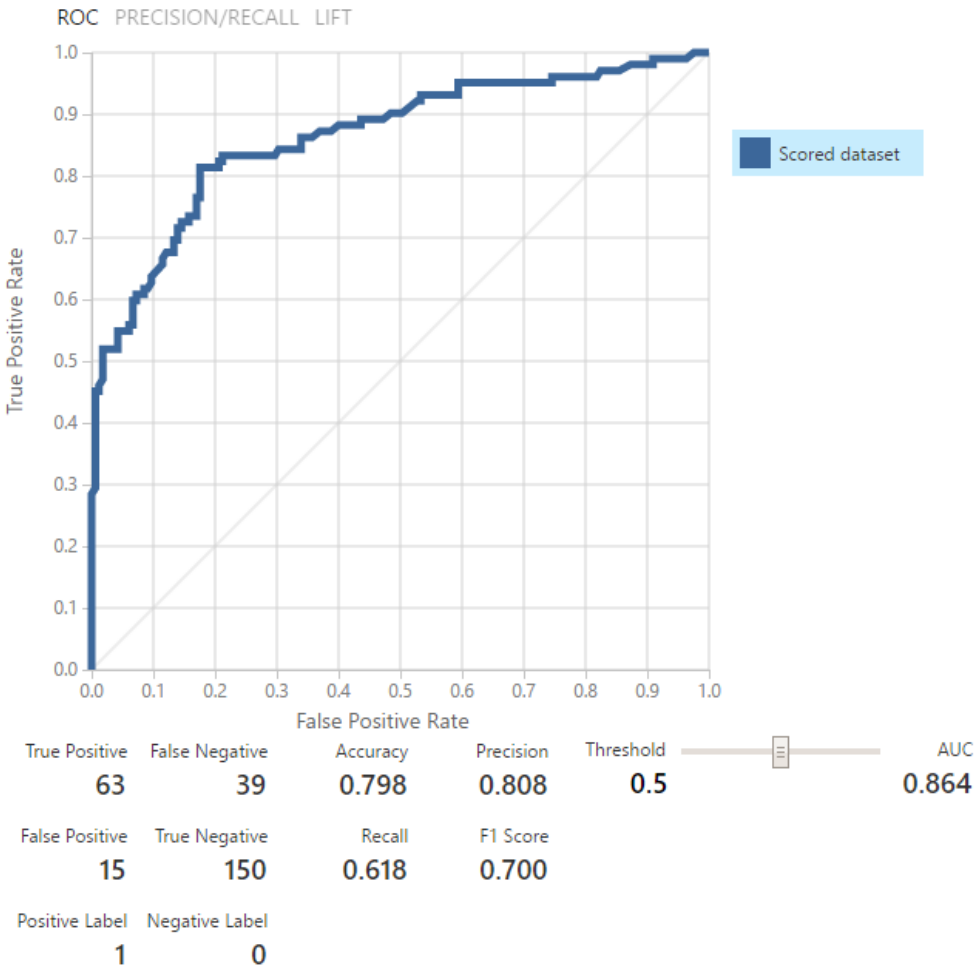


Figure 4.63: Visualization of **Evaluate Model** module

4.5 Metrics for Evaluating Models

When you visualized the **Evaluate Model** module in the previous exercise the metric information provided might have seemed overwhelming. Before completing more exercises we will introduce some of the metrics we will use.

Each machine learning problem has its own set of unique goals. This means that different priorities will be taken into account when evaluating the effectiveness of a model. As a result each problem will utilize different metrics to conduct analysis. There are four general metrics that are commonly used: RoC AuC, Precision, Recall, and Accuracy (Figure: 4.64). Maximizing some of these will metrics will degrage the other.

```
Which metric to optimize?
1. RoC AuC: Overall Performance
2. Precision: Relevance
3. Recall: Thoroughness
4. Accuracy: Correctness
```

Figure 4.64: Summary of popular machine learning performance metrics.

4.5.1 Exercise: Evaluating and Interpreting Your Model

1. We will begin evaluating the model created in the previous exercise. Right-click the **Evaluate Model** module and select "Visualize".
2. Look at the "recall" in the visualization. What is this value? Is this a successful model based on the value?
3. Look under "False Positive" and "False Negative". How many of each did the model predict? What does having a false positive or negative mean?
4. What values are displayed under "Accuracy", "Precision", and "Recall"?

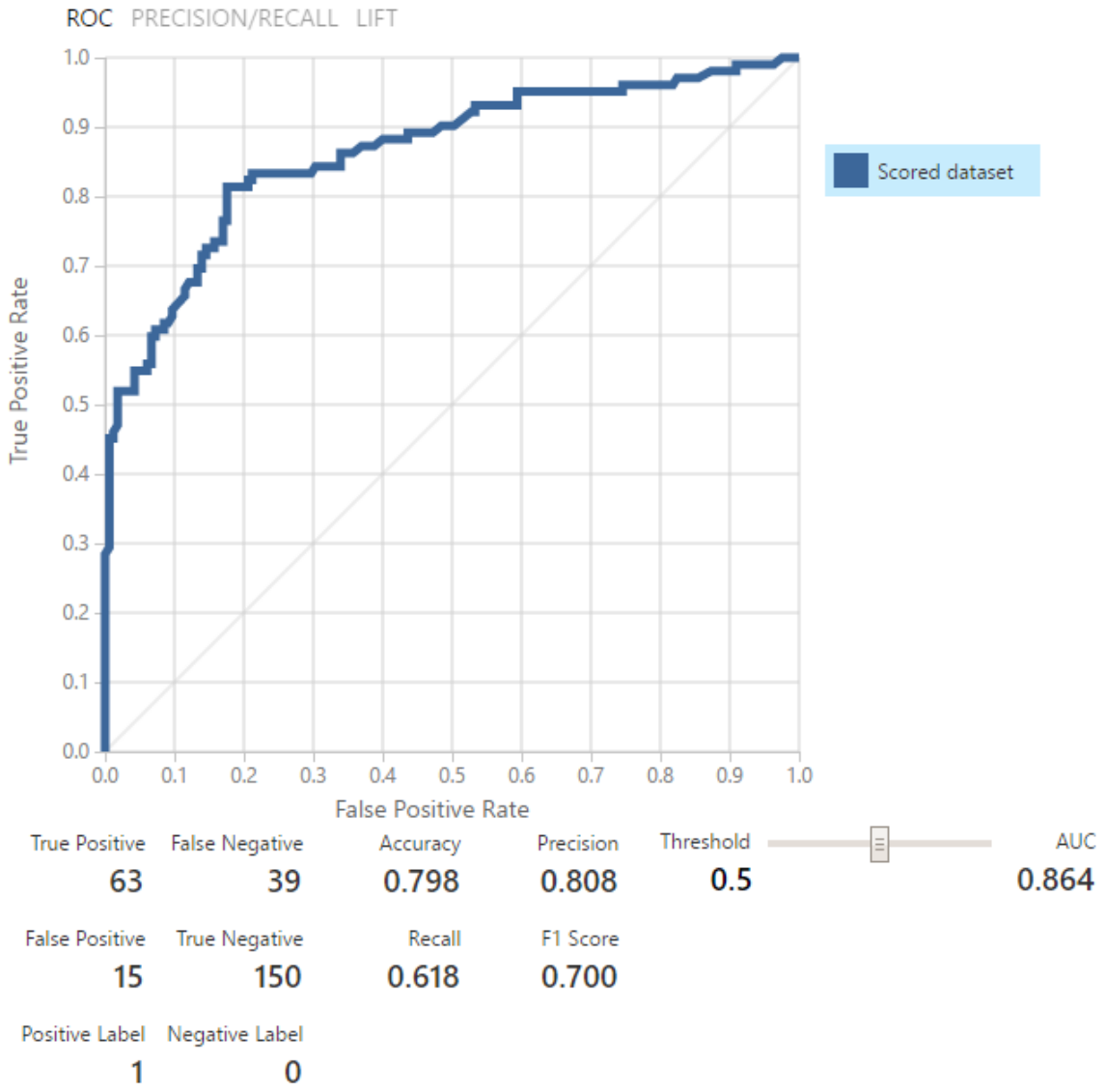


Figure 4.65: Example model output. Your model may look different.

For this lab, we will pretend that our model will be employed and used by an autonomous search and rescue robot directly after the sinking to find survivors. We want to maximize the amount of survivors we find, even if it means pulling a few extra cadavers onboard. However keep in mind that the time spent rescuing these cadavers could also have been spent finding other survivors. For this exercise we will try to build and tune a model that maximizes recall. Recall tries to maximize finding of all the true class “1" labels, so in this case, among all the survivors how many of the survivors did you truly find? In our example, our recall is 0.618 which means that among all of those who were survivors, our model was only able to find 62% of them.

4.5.2 Exercise: Fine-Tuning and Optimizing Your Model

1. For this example, we will be replacing the **Two-Class Boosted Decision Tree** in our workspace. Delete the **Two-Class Boosted Decision Tree**, search in the left menu for **Two-Class Decision Forest**, and drag this module into the workspace to replace the previous module.
2. Compress your modules together so they are lined up vertically (Figure: 4.66).

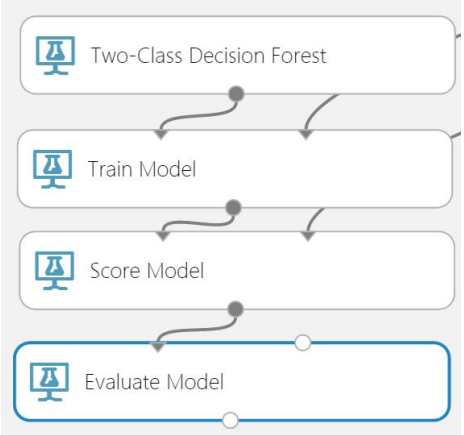


Figure 4.66: Connected modules

3. To select a group of modules, you can either drag a highlighted box over the desired modules or hold down the control key and click on each module individually. Select the first three modules in vertical alignment (**Two-Class Boosted Decision Forest**, **Train Model**, and **Score Model**), then copy (ctrl+c) and paste (ctrl+v). Align the pasted modules to the right of the original vertical set.
4. Connect the new **Score Model** module to the top right node of the existing **Evaluate Model** module (Figure: 4.67).

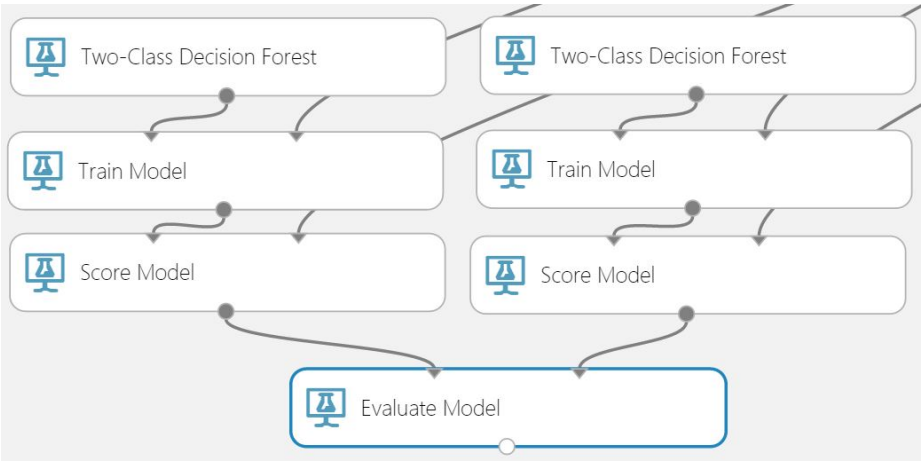


Figure 4.67: Connect the **Evaluate Model** module

For this exercise we will evaluate two different types of decision forests. One with many small trees or one with free large trees. The left two-class decision forest will be our decision forest with many small trees. The right two-class decision forest will be our decision forest with a few number of large trees.

Figure 4.68: Decision forest with a few large trees and a decision forest with many small trees

5. Select the **Two-Class Decision Forest** that we just added to the left. In the options on the right, change "number of trees" to 100 and "max depth" to 4.
6. Select the original **Two-Class Decision Forest** on the right. In the options on the right, change "number of trees" to 8 and "max depth" to 32.

After setting up these parameters, we have one algorithm of 100 decision trees with a shallower depth and another algorithm of 8 decision trees with a deeper depth. In your opinion which one will run better?

7. **Run** the model, right-click on the **Evaluate Model** module, and select "Visualize" to view the output.

An example output is displayed in Figure: 4.69.However your output may look different.

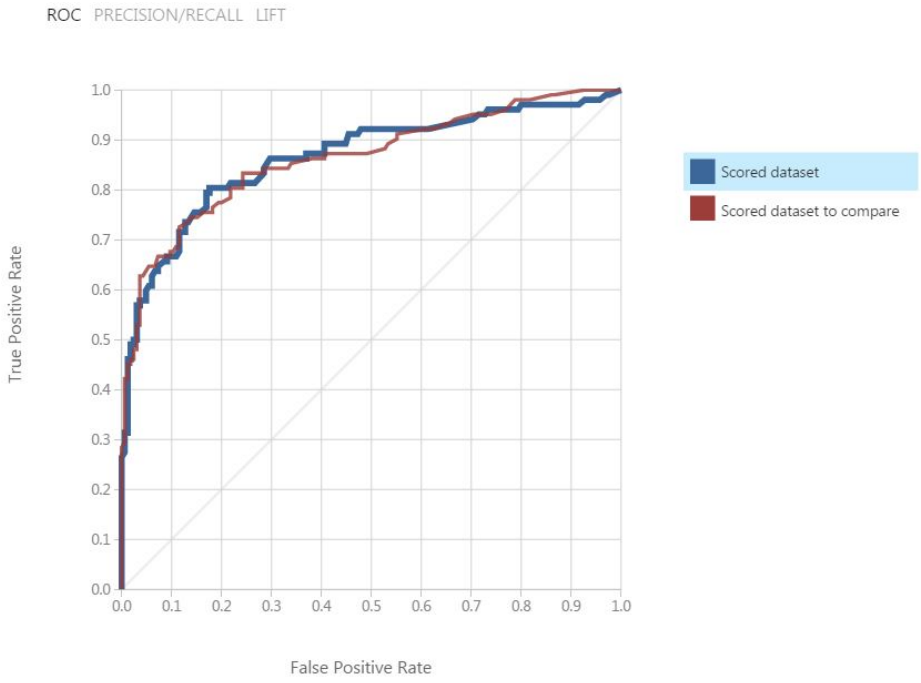


Figure 4.69: Example output

Notice that the blue curve (the output of the 100 decision tree algorithm) is higher on the graph than the red curve (the output of the 8 decision tree algorithm). The graphic visualization tells us that the 100 small decision trees outperformed the 8 large decision trees in terms of the RoC AuC.

From here you can continue optimizing your model by tweaking the parameters of the algorithm until you are happy with the model's performance.

4.5.3 Exercise: Fine-Tuning Across Different Algorithms

In this exercise we are going to compare the different outputs of the various "Two-Class" classification models, to determine the most suitable algorithm for our experiment.

1. Begin by looking in the left menu under **Machine Learning > Initialize Model > Classification**. Drag each algorithm module that begins with "Two-Class", other than **Two-Class Decision Forest**, into the workspace side-by-side (Figure: 4.70)

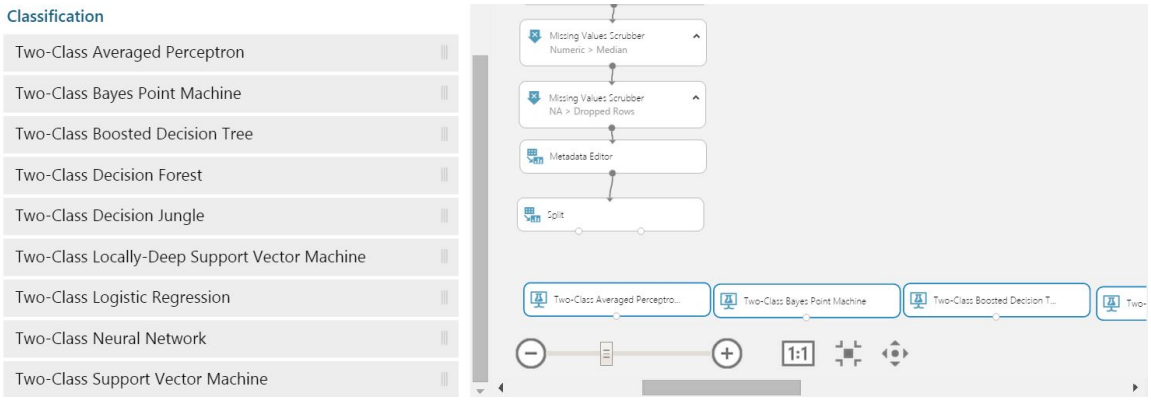


Figure 4.70: Drag two-class algorithms into the workspace

2. Select the **Train Model** module and **Score Model** module simultaneously (by highlighting or holding down the control key) from either of the **Two-Class Decision Forest** modules we previously ran. Copy and paste the two modules into the workspace and connect them to the first "Two-Class" algorithm in the chain (Figure: 4.71).

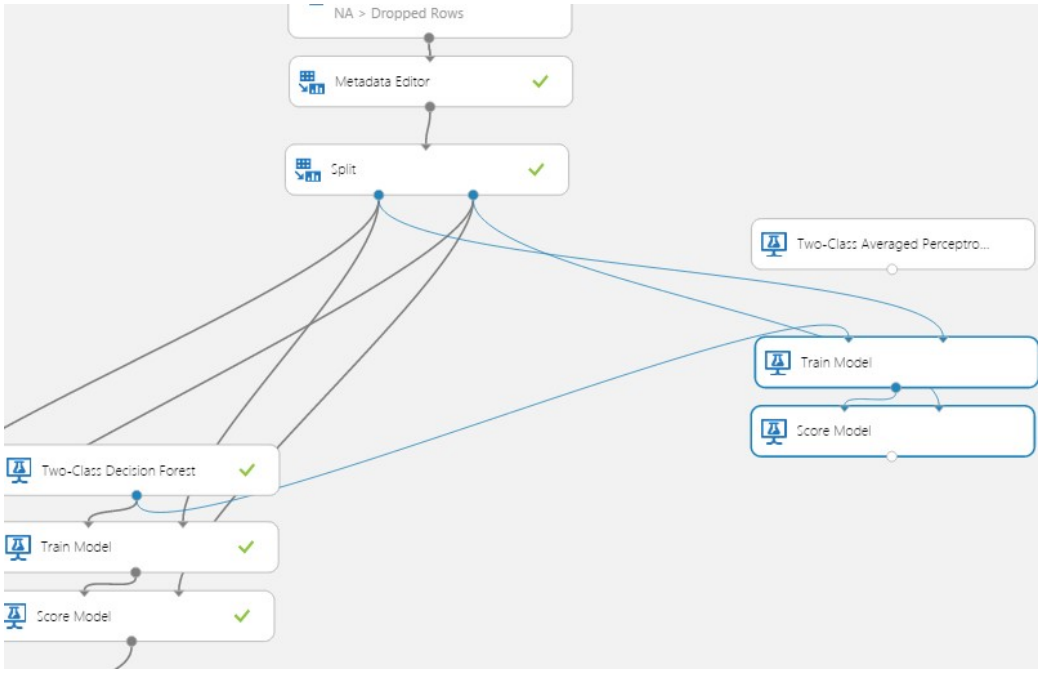


Figure 4.71: Copy and paste the train and score model modules

3. Redirect the top left node of the new **Train Model** module to the bottom middle node of the first algorithm in the chain (Figure: 4.72).

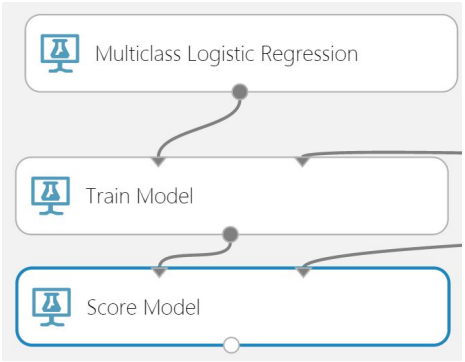


Figure 4.72: Redirect the Train Model module

4. Repeat this process for all of the "Two-Class" algorithms we dragged into the workspace (Figure: 4.73).

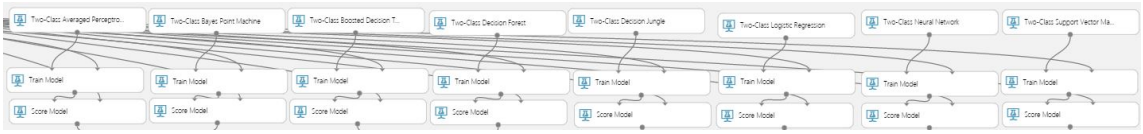


Figure 4.73: Repeat this process for two-class algorithms

5. Search in the left menu for an **Evaluate Model** module and drag it into the workspace. Connect the first two algorithms together by connecting them both to the top two nodes of the same **Evaluate Model** module (Figure: 4.74)

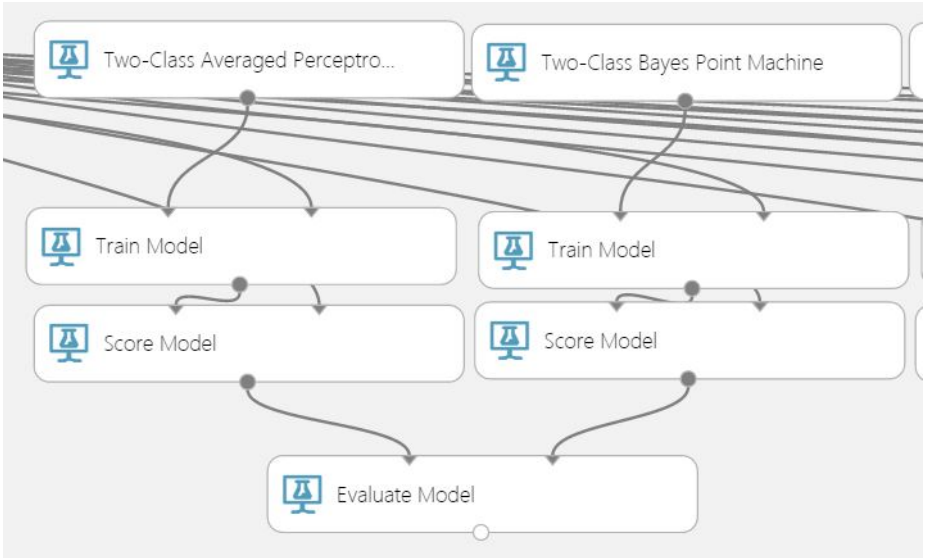


Figure 4.74: Connect algorithms to Evaluate Model module

6. Repeat this process until all of the two-class algorithms are connected and output to an **Evaluate Model** module (Figure: 4.75)

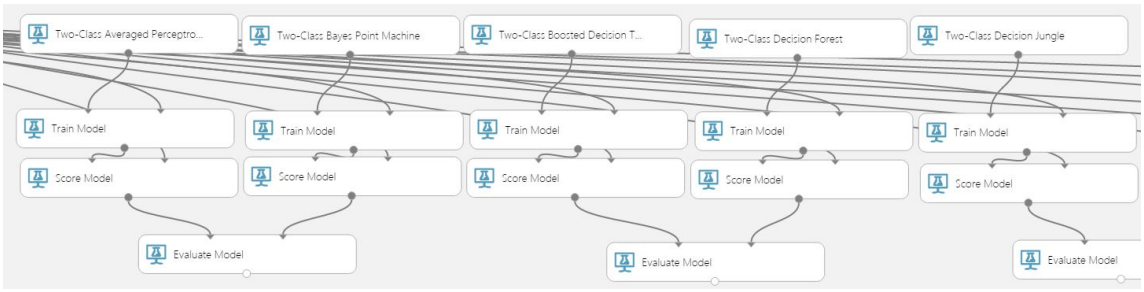


Figure 4.75: Repeat the process

7. Right-click and select "Visualize" for each of the **Evaluate Model** modules. Look at the "Cumulative AuC" for each visualization and rank the performance.

Figure: 4.76 is an example of the ranked performance of the output for our model. Keep in mind your values will likely be different.

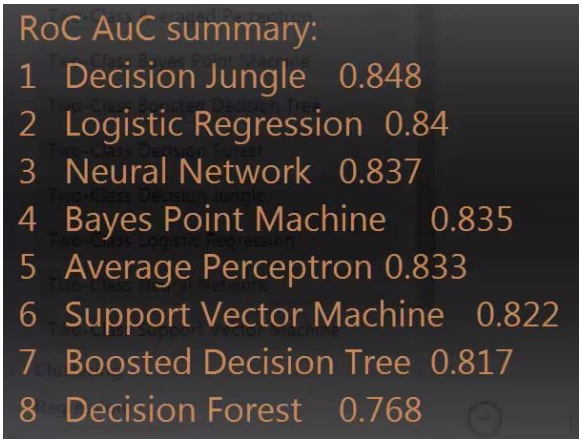


Figure 4.76: Example of ranked algorithms

The **Two-Class Decision Jungle** had the best performance in its default setting. Now that we have this information we can move forward and optimize the algorithm to create an improved model.

4.6 Creating and Publishing a Predictive Model as a Web Service

4.6.1 Exercise: Exporting and Saving Your Optimized Trained Model

Before we begin learning how to export and save your model, make sure you are happy with the algorithm you chose and the model you trained and built.

1. **Run** the model and make sure all the modules in your workspace have a green check mark displayed, with no errors (Figure: 4.77)

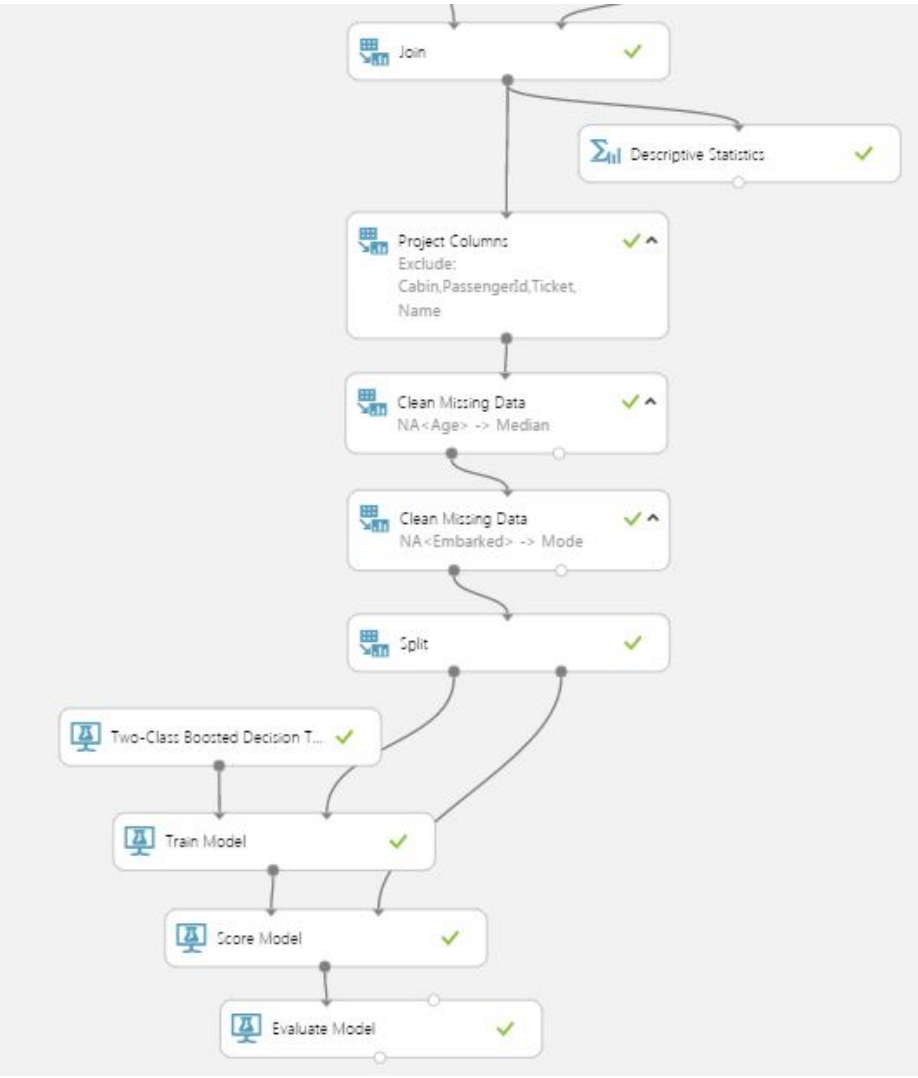


Figure 4.77: Run the trained model. Notice that each module has a check mark

2. When you are ready to export and save your model, right-click on the output node of the **Train Model** module, and select "Save as Trained Model" (Figure: 4.78)

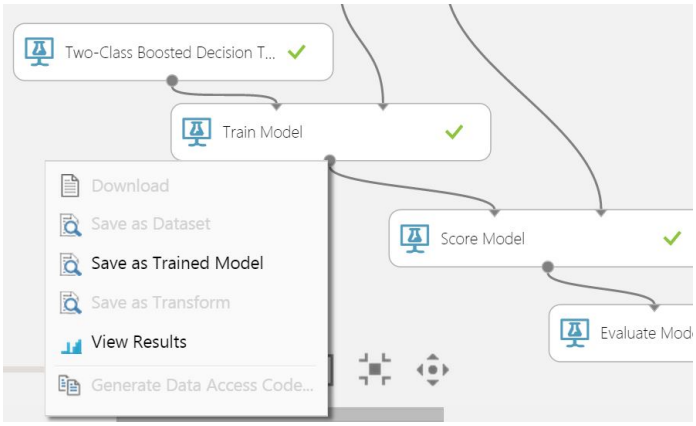


Figure 4.78: Save as a trained model

3. When saving your model, make sure to choose a descriptive and succinct name as you will not be able to rename a trained model once it has been created (Figure: 4.79)

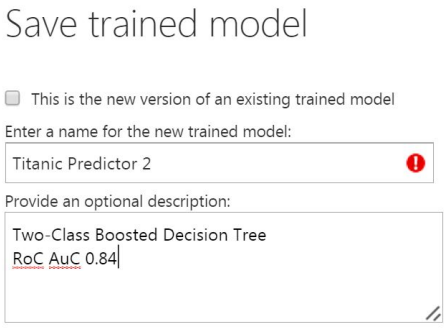


Figure 4.79: Comment on the metrics used

Multiple people often collaborate on the same Azure ML experiment. It is a good habit to avoid cluttering the trained models folder with undescriptive names. In addition, it is a good idea to provide an optional description that describes the type of algorithm used and the metric used for optimization.

4. To verify that the model was exported select the **Trained Models** dropdown from the left menu. You should see the name you chose for your model listed (Figure: 4.80).

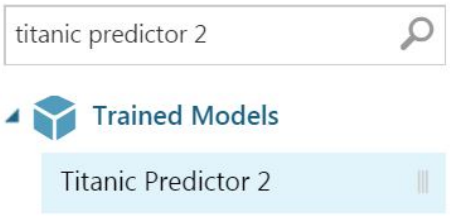


Figure 4.80: Find your model listed under Trained Models

4.6.2 Exercise: Deploying Your Model

The **Score Model** module we used in our workspace makes predictions for the experiment. Therefore we want to create a web service that sends all of the request calls to the **Score Model** module as an input so that the requests are outputted from the **Score Model** module as a prediction. This flow is visualized in Figure: 4.81.

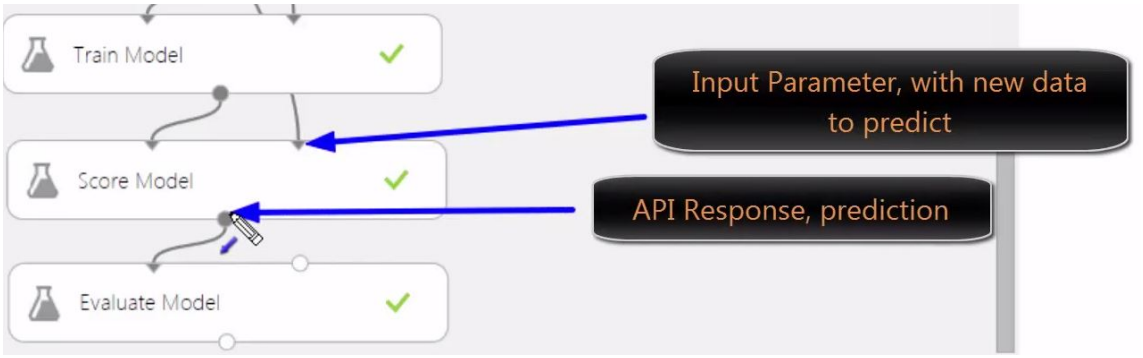


Figure 4.81: Flow of requests

1. To begin deploying your model, clone the deployment experiment at: `https://gallery.cortanaintelligence.com/Experiment/Ready-to-Deploy-Predictive-Model-1`
2. Click on the “Open in Studio" button (Figure: 4.82). A new window will open prompting you to sign-in if you are not signed-in.

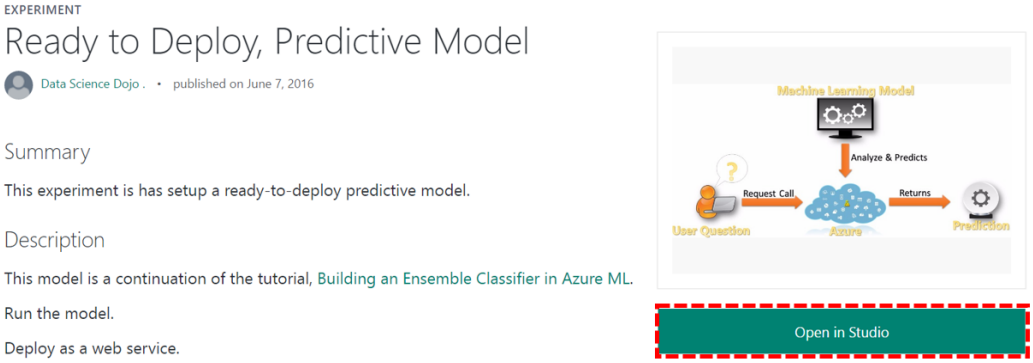


Figure 4.82: Copy a experiment to Azure ML

3. Select your Region and Workspace to copy the experiment into (Figure: 4.83).

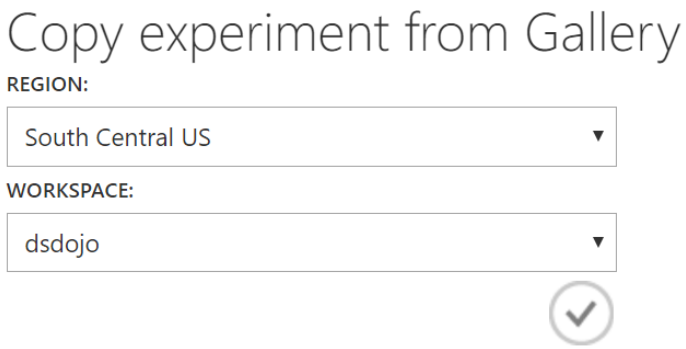


Figure 4.83: Selecting a Region and Workspace to copy a expriment

4. Once cloned, the “Ready to Deploy, Predictive Model" experiment will show up under your experiments (Figure: 4.84). Click on the experiment to view and edit it.

Figure 4.84: View cloned experiments

5. Notice that the experiment looks similar to the one that we completed in the previous section. (Figure: 4.85). However, the **Split Data** module and the **Evaluate Model** module has been removed from the experiment. We removed the **Split Data** module because we have already completed evaluating the predictive model. The **Split Data** module was originally intended to partition 30% of the data to make predictions and build a confusion matrix in the **Score Model** and **Evaluate Model** modules. We also partitioned 70% of the data to build the predictive model using the **Train Model** module. However, now that we have seen the results from using 70% of the data to train the predictive model, the **Evaluate Model** module can be removed. We know that the algorithm of the predictive model will do better with 100% of the data. So, the **Split Data** module and the **Evaluate Model** module is no longer required in the deployment of the experiment. The **Score Model** module instead, will make predictions using 100% of the data.

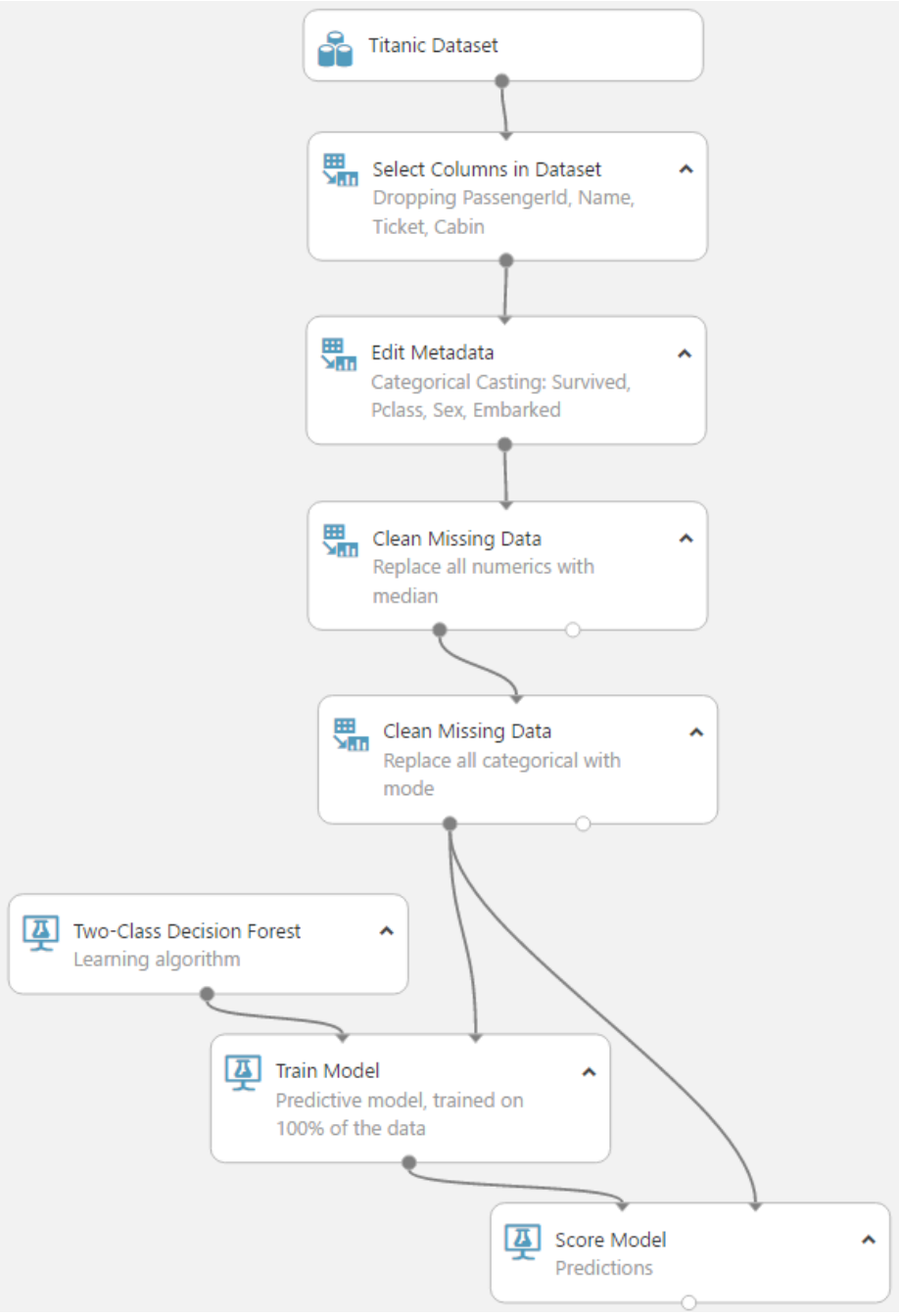
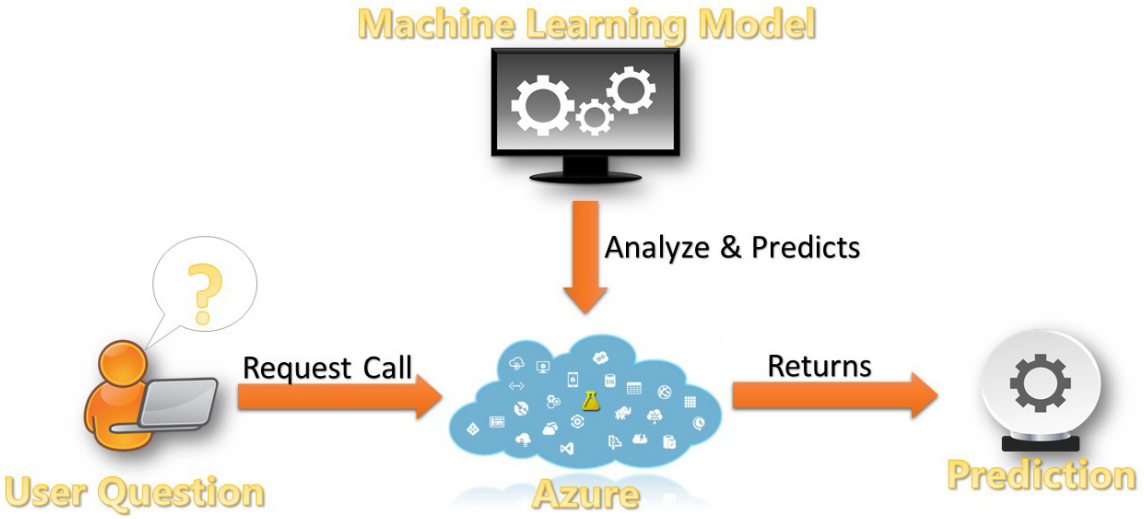


Figure 4.85: Ready to deploy predictive model of the Titanic dataset

6. **Run** the workspace to finalize the model. Make sure that the top-right corner displays "Finished running" before moving on.



7. **Hover** over the **Set up Web Service** button on the bottom of your screen and **select** the **"Predictive Web Service"** option that pops up (Figure: 4.86). If this button is still greyed out, try running the model again. All modules must have green check marks with specified output and input nodes or the button will be disabled.

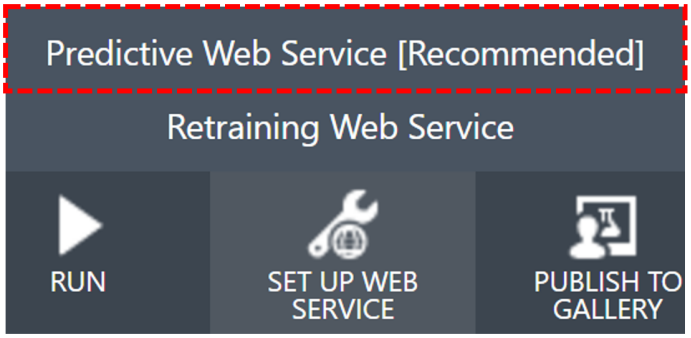


Figure 4.86: Publish as a predictive web service

8. Once deployed, you'll notice that your workspace will have two tabs at the top. "Training experiment" and "Predictive experiment" (Figure: 4.87).

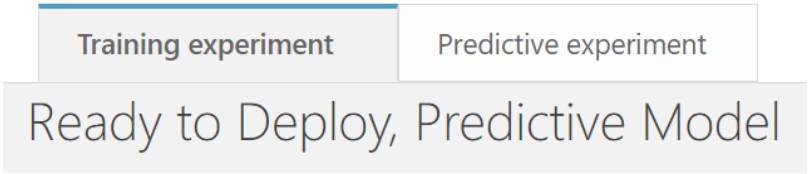
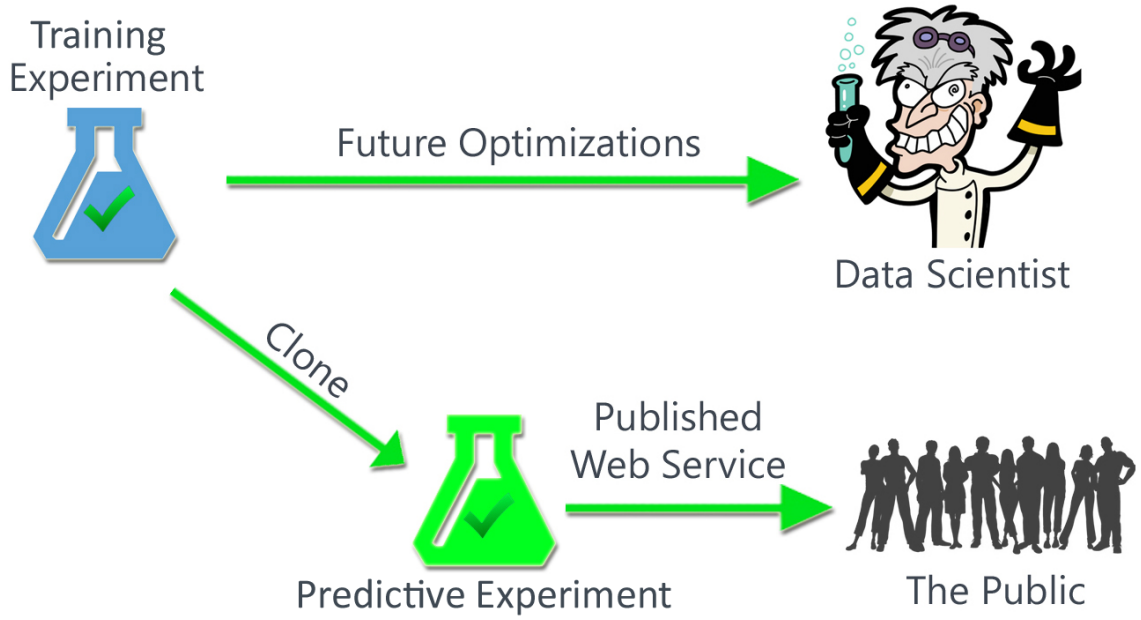


Figure 4.87: Training experiment and Predictive experiment tab options in workspace

The reason for two experiment options in your workspace is that the experiment has been consolidated into two branches, the "Training experiment" contains the training logic while the "Predictive experiment" contains the deployment logic. Model deployment is an iterative process.

The same model may always be revisited in the future. In this case, it would be the “Training experiment". A new employee may come up with a better approach to fine tuning the model, real world data might return different results than expected, new information becomes available, or the environment in which the model is operating may slightly changed. This is why you would want to have access to the “Training experiment", also called branching in the programming world. If you later decide to re-optimize your model, you will still have your original “Training experiment" preserved, without having to worry about disrupting any apps, dependent on the deployed model. The “Predictive experiment" model is what we will use to publish as a Web Service.



9. **Select** the “Predictive experiment" tab and notice that the **Web service input** module is connected to the **Select Columns in Dataset** module. This means that the **Web service input** requires a raw data schema similar to that of the Titanic Dataset. If we input the data at the current **Web service input** module location, it would go through the process of cleaning up the data. However, since this is a predictive model and not a cleaning model, we can enter clean data from a different location. To do this, drag the **Web service input** module and re-connect it to the **Score Model** module (Figure: 4.88).

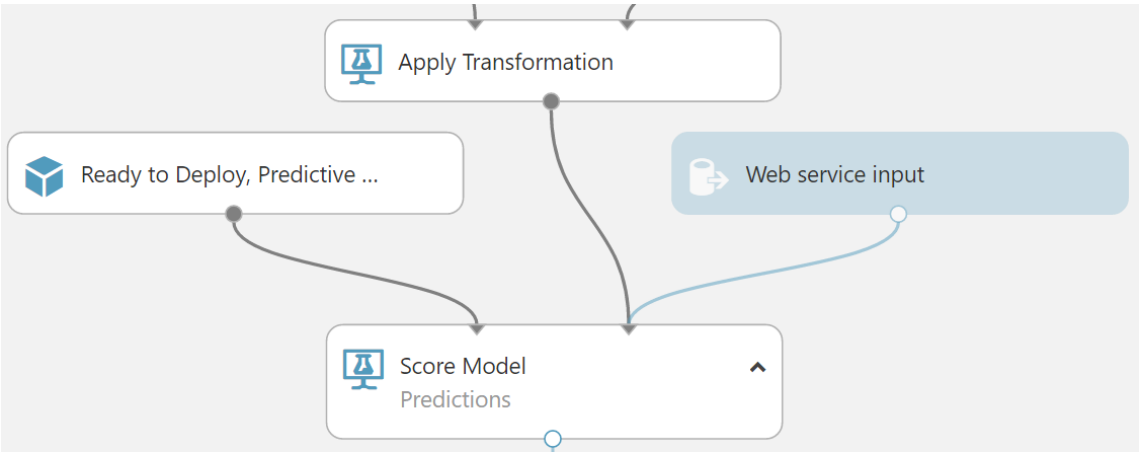


Figure 4.88: Re-connecting the **Web service input** module to input data into the **Score Model** module

10. **Run** the workspace. Make sure that the top-right corner displays “Finished running" before moving on.
11. We are now ready for deployment. **Hover** over the “Deploy Web Service" button at the bottom of your screen and **select** “Deploy Web Service [Classic]" (Figure: 4.89).

Figure 4.89: Deploying the model as a web service.

After selecting the **Deploy Web Service [Classic]** button and deploying your model, your screen should look like Figure: 4.90.

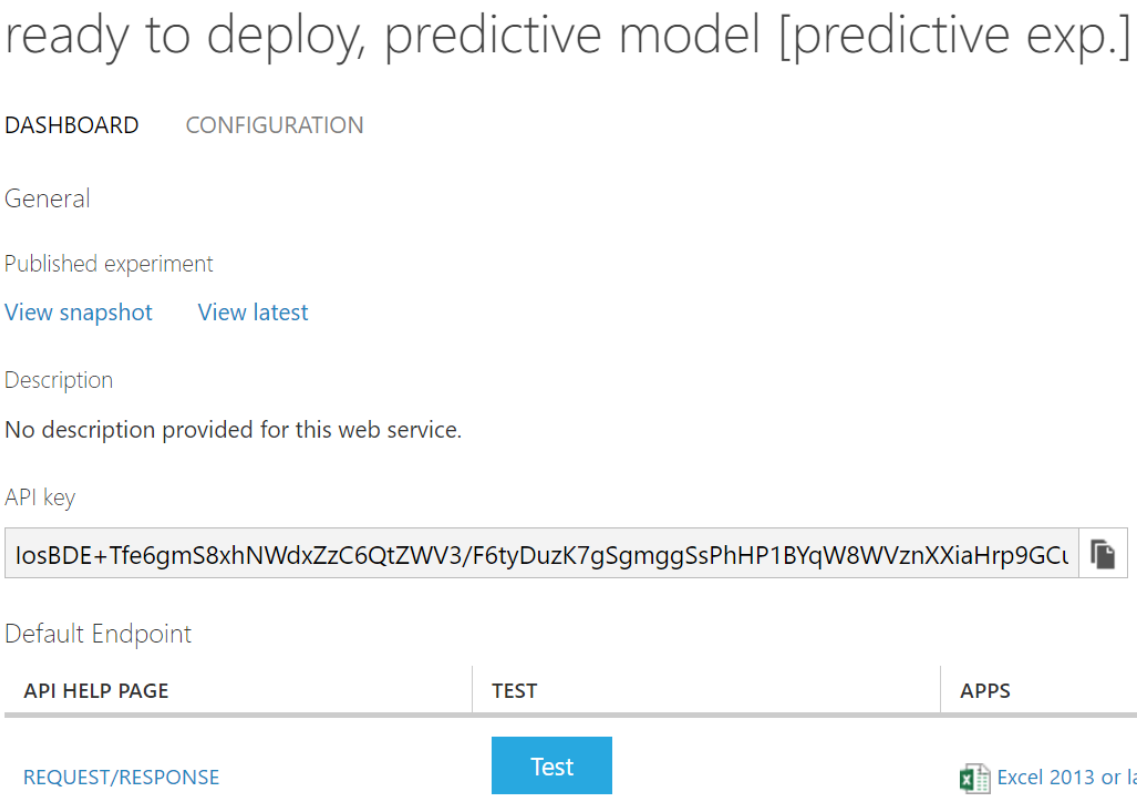


Figure 4.90: After deploying the model as a web service

12. To test the model, look under the "TEST" column and select the "Test" button to the right of your listed model (Figure: 4.90).

Figure 4.91: Test button

13. A form window should pop up. The input fields will have the same names as the columns in the schema (Figure: 4.92).

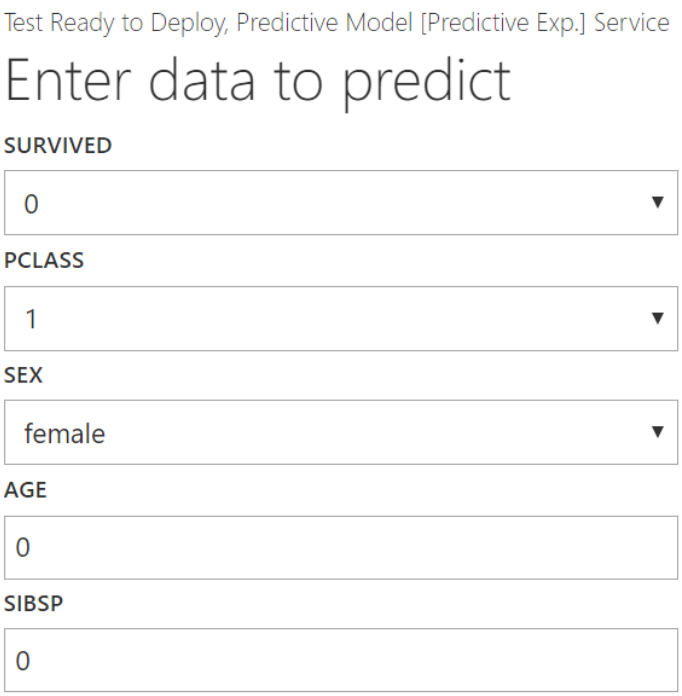


Figure 4.92: The API test form

14. Input your own variables into the fields. Keep in mind that the form is case sensitive. For example for "Sex" use "male" instead of "Male" and for "Embarked" use uppercase "Q", "C" and "S".
15. After you've input your variables, submit your responses. On the bottom right-hand side of the screen you will see a loading box. Select this box when your input has finished loading. Select "Details" and look at the result (Figure: 4.93).

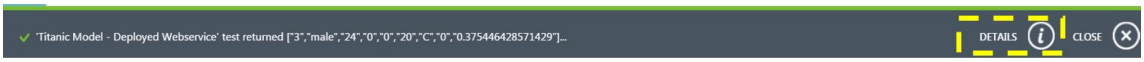


Figure 4.93: API details box

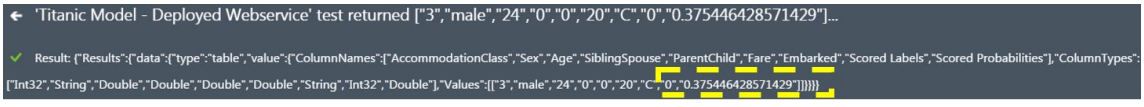


Figure 4.94: This model thinks a male of 24 years of age would have a 37% chance of survival.

The results shows a "0" followed by ".37544....". The "0" means that the passenger is predicted to not survive and has a 0.37% chance of survival.

Try out different parameters to see how the model performs with different input variables. Would you have survived the Titanic?

Data Science Dojo has launched the model you just created as a Django web application (Figure: 4.95). Take a look: `http://demos.datasciencedojo.com/demo/titanic/`.

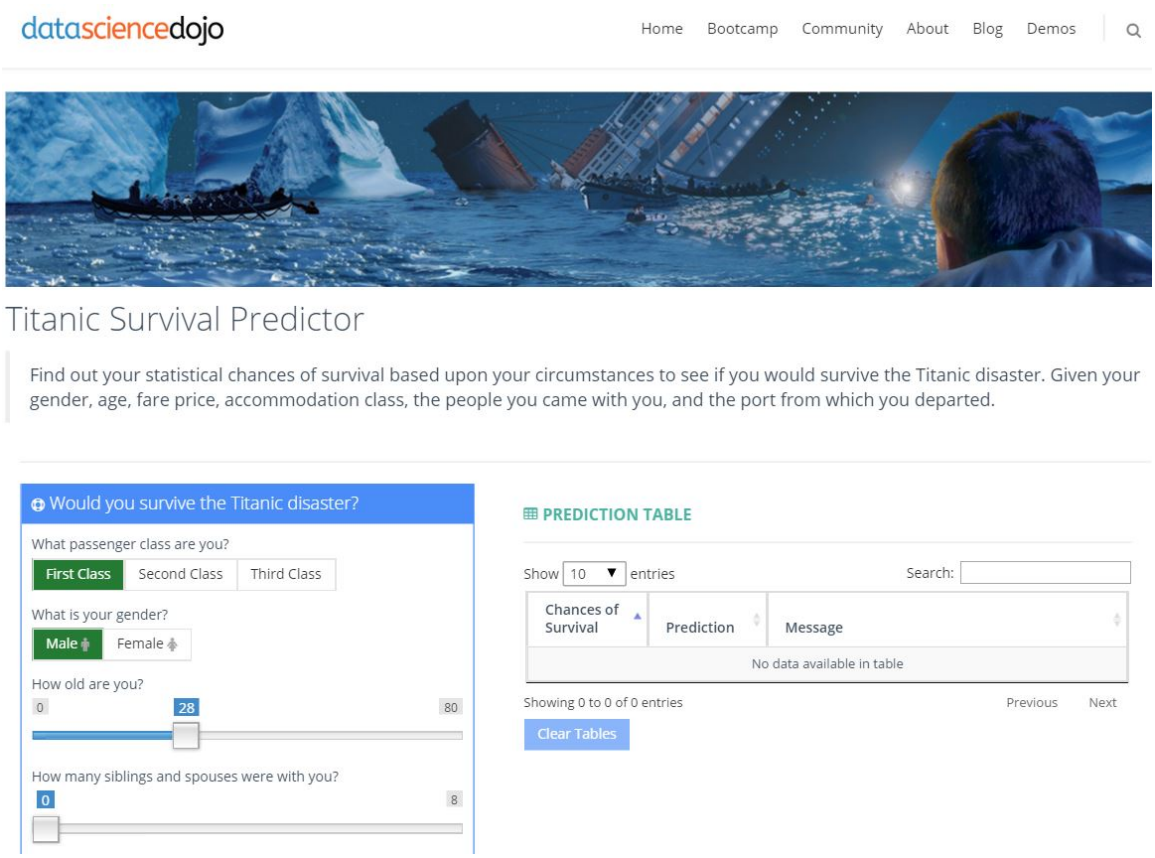


Figure 4.95: Titanic survival predictor

4.6.3 Exercise: Exploring and Testing Your API

Before we move further, it might be a good idea to provide some documentation for your API. To do this, go into **Configuration** at the top of the screen (Figure: 4.96) to input this optional information (Figure: 4.97) and save your descriptions.

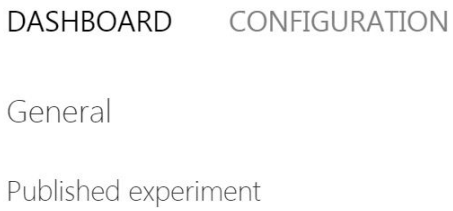


Figure 4.96: Configuration documentation

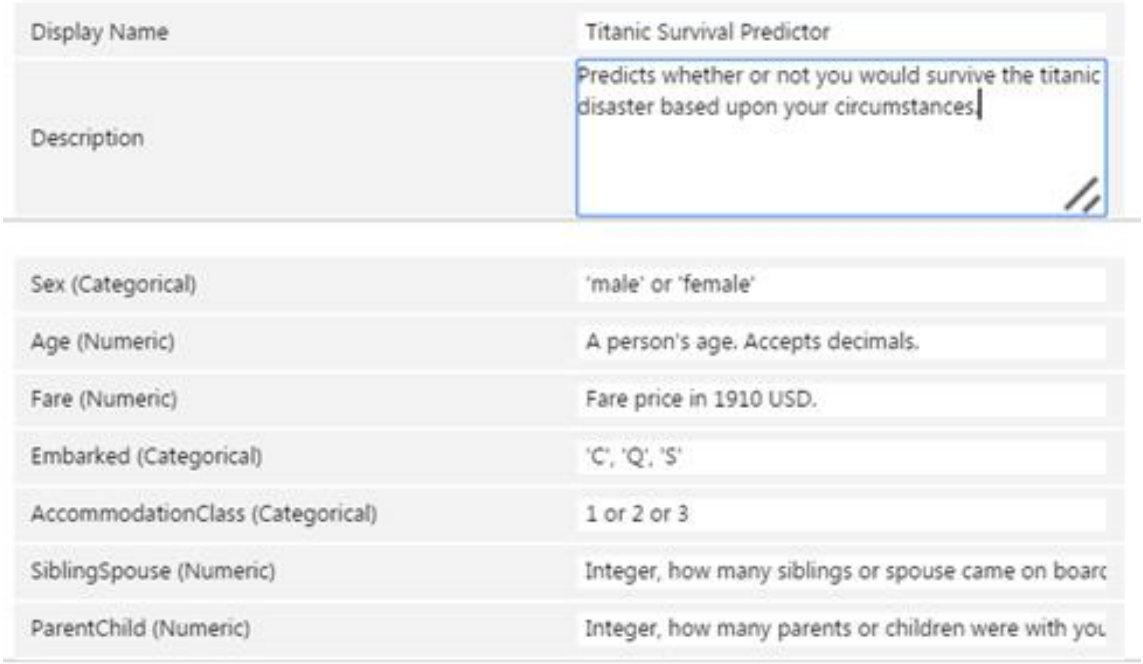


Figure 4.97: Optional information input

1. Start by going to **Dashboard** at the top of the portal page (Figure: 4.98).

Figure 4.98: Dashboard

2. View the API help page for "request/response"
   - Note the sample request. Azure will want a JSON in the exact same manner.
   - View the sample response. If this is blank, then go back and check your model because something is wrong (Figure: 4.99).

Figure 4.99: Sample response output

3. View the pre-generated code for C#, Python, and R
   - C# sample code requires .NET 4.5, therefore visual studio > 2010+
   - R sample code will only work for the 32-bit R console. It may not work with the 64-bit R console or RStudio (Figure: 4.100).

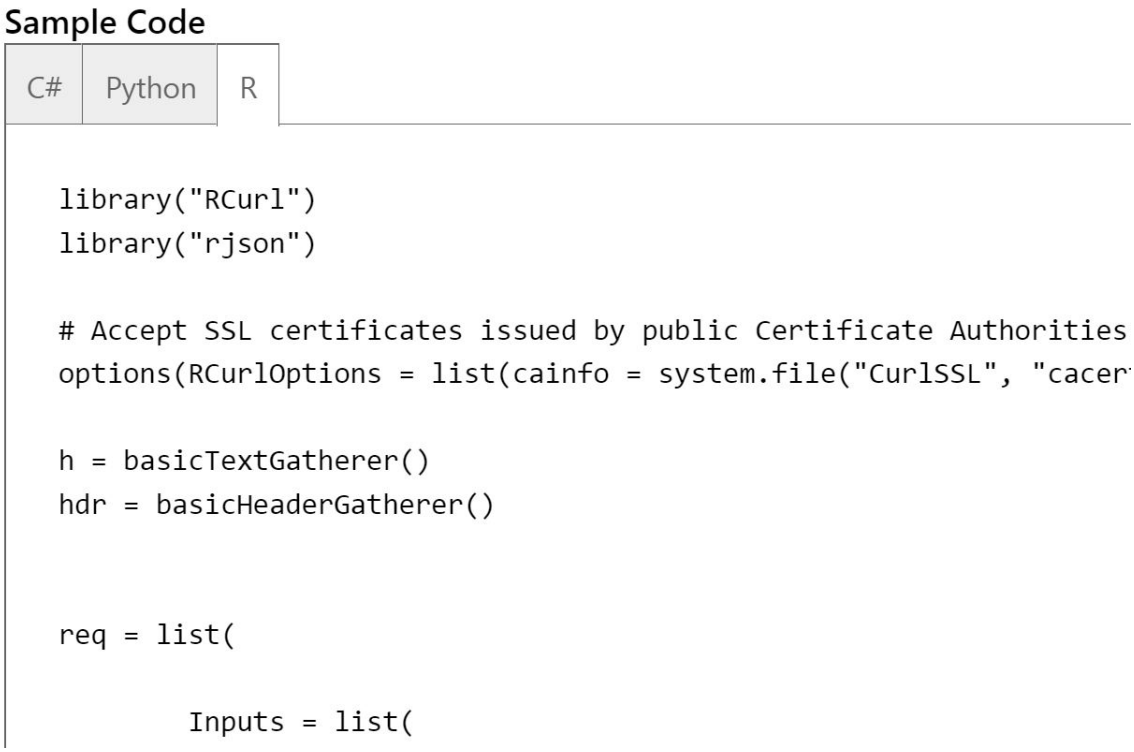


Figure 4.100: Left: sample input payload. Right: sample response payload

4. Notice how all the codes require your API key. The API key can be found on the previous page (Figure: 4.101).

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

Figure 4.101: Sample API key

4.7 R Script Module

4.7.1 Exercise: Exploring the R Module

1. In your Azure ML workspace, select +**New** / "Blank Experiment" to start a new project.
2. From the left menu, search for the **Execute R Script** module and drag it into your workspace (Figure: 4.102)

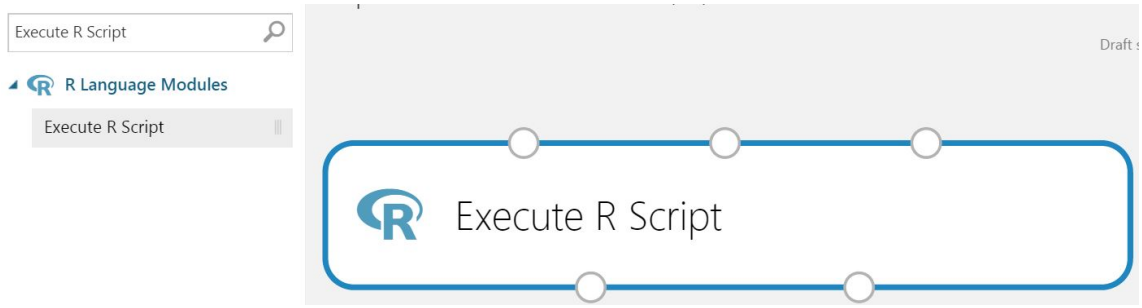


Figure 4.102: Execute R Script module

You will notice that the **Execute R Script** module has five nodes (Figure: 4.103). Use the key in Exercise: Exploring the R Script Sample Code to understand the purpose of each node.

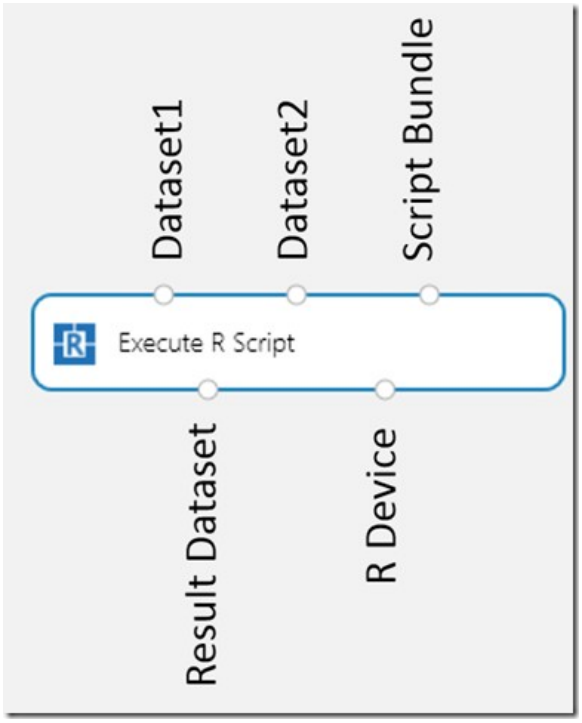


Figure 4.103: Node Key

4.7.2 Exercise: Exploring the R Script Sample Code

1. Select the **Execute R Script** module. A window will populate the right side, allowing you to type in raw R code (Figure: 4.104)

R Script

```
# Map 1-based optional input ports to variables
dataset1 <- maml.mapInputPort(1) # class: data.frame
dataset2 <- maml.mapInputPort(2) # class: data.frame

# Contents of optional Zip port are in ./src/
# source("src/yourfile.R");
# load("src/yourData.rdata");

# Sample operation
data.set = rbind(dataset1, dataset2);

# You'll see this output in the R Device port.
# It'll have your stdout, stderr and PNG graphics device(s).
plot(data.set);

# Select data.frame to be sent to the output Dataset port
maml.mapOutputPort("data.set");
```

Figure 4.104: Window for R code

In the new window, everything in green is a comment. Comments are created by using the hashtag symbol "#".The R module compiler will then skip over these lines without executing them. Comments can either be at the start of a new line or within a line (Figure: 4.105).

```
# Map 1-based optional input ports to variables
dataset1 <- maml.mapInputPort(1) # class: data.frame
dataset2 <- maml.mapInputPort(2) # class: data.frame
```

Figure 4.105: Commenting in R

The key in Table: 4.2 provides more information about the nodes and their specific functions in relation to the R script.

Node	Parameter	R Script Code
Left Input	Input Dataset1	mam1.mapInputPort(1);
Middle Input	Input Dataset2	mam1.mapInputPort(2);
Right Input	R Script Bundle	Source("src/yourFile.R");
Left Output	Output Dataset	mam1.mapOutputPort("<SomeObject>")
Right Output	R Device (graphs/charts)	plot(<SomeObject>);

Table 4.2: Node descriptions in R

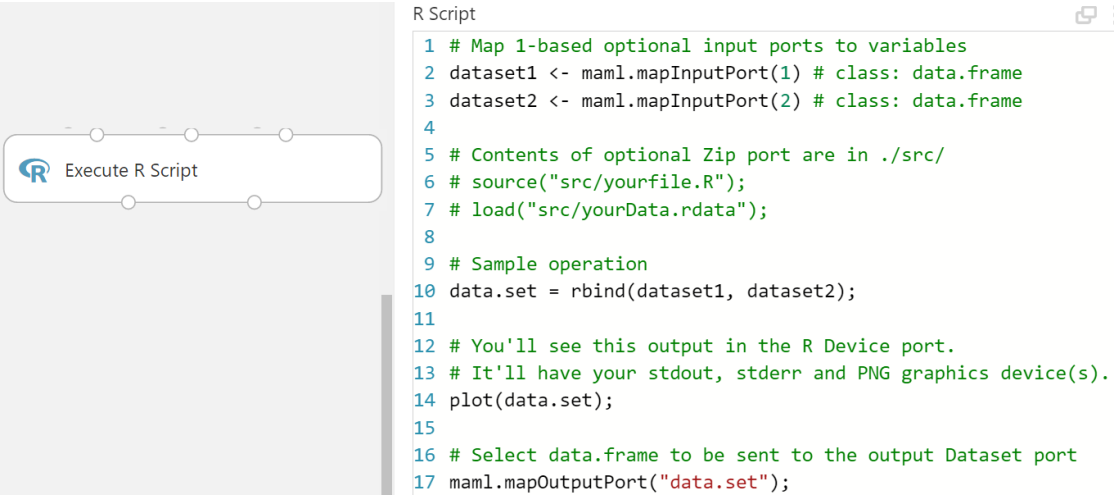


Figure 4.106: Inputted R script

4.7.3 Exercise: Troubleshooting R Bugs in R Module

Although Azure currently supports R in the R script module, there are still many bugs as it is in the beta release. Going forward it is good to keep in mind some known issues:

- Copying and pasting code will result in runtime errors within the module. Try to type out everything when possible into the R module when this is the case.
- The debugger will only tell you that an error in the script exists without providing any further insights into why the script failed. To look for errors you must find them yourself in the trace-back log.

For the purpose of this exercise you will intentionally try to execute a bug into the R script module so you can learn how to troubleshoot.

1. If you do not already have an **Execute R Script** module in your workspace use the left menu to search for this module and drag it in.

2. In the window on the right side insert the following code into the **Execute R Script** module and **Run** the experiment.

```
x <- 1.0;
z <- x + y;
print(z);
```

After the experiment has ran and finished loading, what do you see? Why do you think the module returned an error?

3. In the **Execute R Script** module window, select the "View output log" button (Figure: 4.107) to find the error.

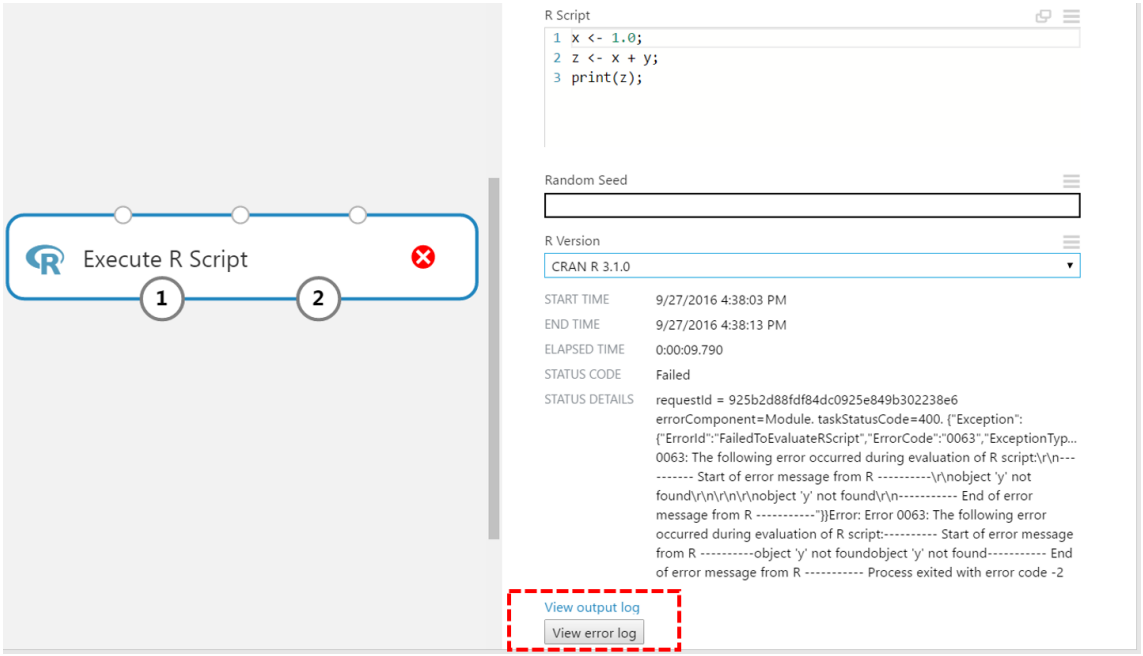


Figure 4.107: View the output log from the Execute R Script settings

After examining the log, you will find the error towards the bottom of the traceback window (Figure: 4.108). The error states that an object called "y" was called in the script but never defined.

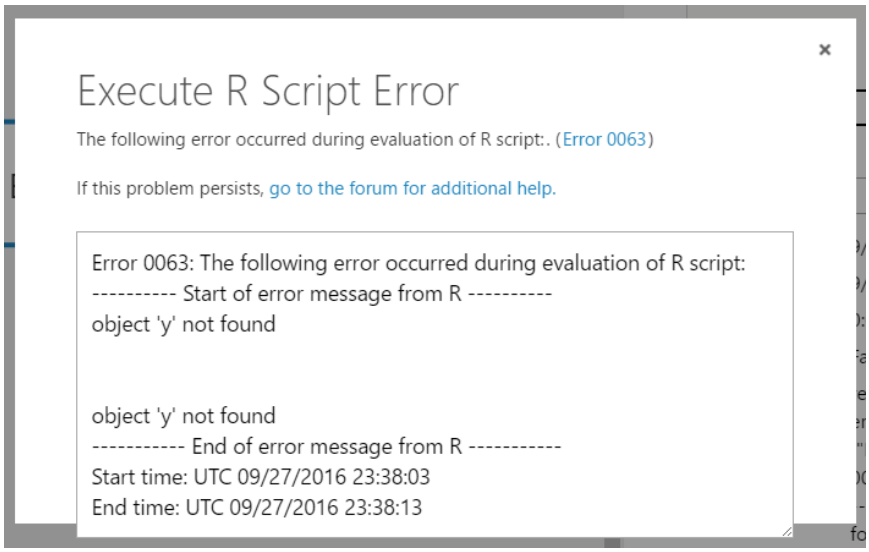


Figure 4.108: Traceback window

4. To fix the error, insert the following code into the **Execute R Script** module.

```
x <- 1.0;
y <- 3.0;
z <- x + y;
print(z);
```

Tip Notice the "print(z)" line included in the code. The value "z" is called at the end of the script in order to output the "z" value into the console.

5. Right-click the bottom right output node of the **Execute R Script** module and select "Visualize". You should see an output value of "4" (Figure: 4.109). Notice that the R device acts in place of the R console, as well as the R plot and graphs.

Figure 4.109: Output node of R module

4.7.4 Exercise: Enumerating R Packages

Within the **Execute R Script** module there are packages that ship natively called R plugins. These packages can be added to your library as a powerful way to increase your data science potential and

productivity without having to worry about implementation. Have an algorithm that you want to implement? Chances are there is a package that can do this for you.

1. In the left menu search for the **Execute R Script** module and drag it into your workspace.
2. In the right menu of the **Execute R Script** module, insert the following code (Figure: 4.110)

```r
#An object which lists all the packages installed within
#the module.
installedPackages <- installed.packages();

#Converts the object of packages into a data frame.
packageDF <- as.data.frame(installedPackages)

#Outputs the data.frame to the result dataset output node.
maml.mapOutputPort("packageDF");
```

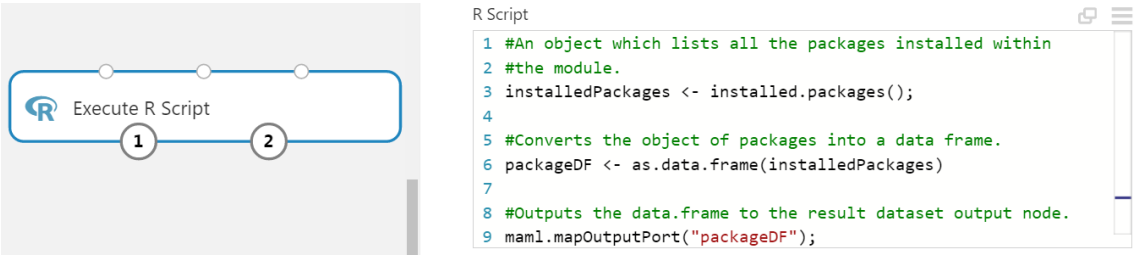
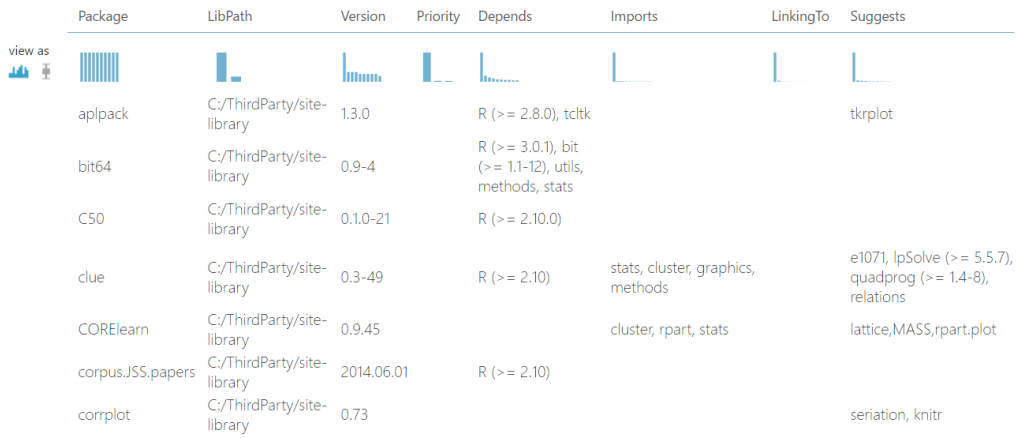


Figure 4.110: Inserted R code

Tip It can be seen in Figure: 4.110 that on line 5, "installedPackages" was converted to a "data.frame" because "mapOutputPort" expects the "data.frame" as an input parameter.



view as

Package	LibPath	Version	Priority	Depends	Imports	LinkingTo	Suggests
aplpack	C:/ThirdParty/site-library	1.3.0		R (>= 2.8.0), tcltk			tkrplot
bit64	C:/ThirdParty/site-library	0.9-4		R (>= 3.0.1), bit (>= 1.1-12), utils, methods, stats			
C50	C:/ThirdParty/site-library	0.1.0-21		R (>= 2.10.0)			
clue	C:/ThirdParty/site-library	0.3-49		R (>= 2.10)	stats, cluster, graphics, methods		e1071, lpSolve (>= 5.5.7), quadprog (>= 1.4-8), relations
COREIearn	C:/ThirdParty/site-library	0.9.45			cluster, rpart, stats		lattice,MASS,rpart.plot
corpus.JSS.papers	C:/ThirdParty/site-library	2014.06.01		R (>= 2.10)			
corrplot	C:/ThirdParty/site-library	0.73					seriation, knitr

Figure 4.111: Left-node output of R code

4.7.5 Exercise: Using ggplots for Advanced Visualizations Within the R Module

Ggplot is a powerful data visualization R plugin, extending the visualization power of R beyond its normal capacity.

For the purpose of this exercise we will be working with the "Adult Census Income Binary Classification" dataset. This dataset ships as a default within your workspace (Figure: 4.112).

1. In the left menu search for the "Adult Census Income Binary Classification" dataset and drag it into your workspace.
2. Again in the left menu search for the **Execute R Script** module and drag it into your workspace. (Figure: 4.112).
3. In the right menu of the **Execute R Script** module insert the following code:

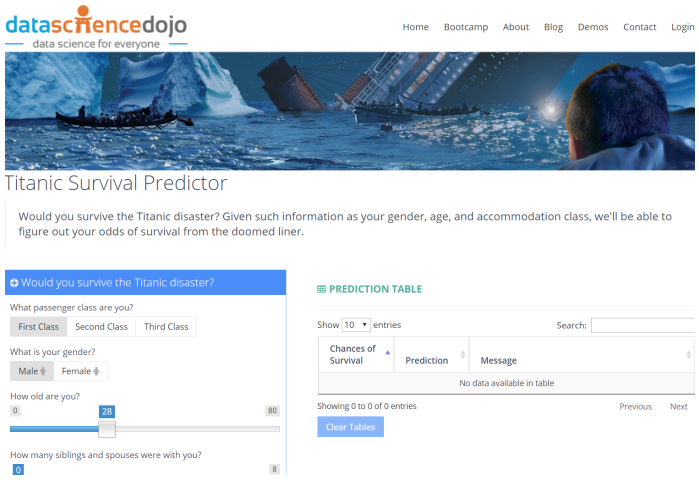


Figure 4.112: **Adult Census Income Binary Classification** module and **Execute R Script** module

```
# Import adult dataset
census.Raw <- maml.mapInputPort(1)

# Reference ggplot2 library.
library(ggplot2)

# Cleanup unfortunate variable names (r does not like '-' in variable names)
names(census.Raw) <- sub(pattern='-', replacement='.',x=names(census.Raw))

# Plots multiple distributions as they related to income levels.
census.genderDistribution <- qplot(x=census.Raw$sex, data=census.Raw, geom="histogram", fill=census.Raw$income, position="dodge")

# Prints each plot to to the R Device output.
```

```
14 print(census.genderDistribution)
```

Tip Please note that the quotation marks must be vertical, not slanted.

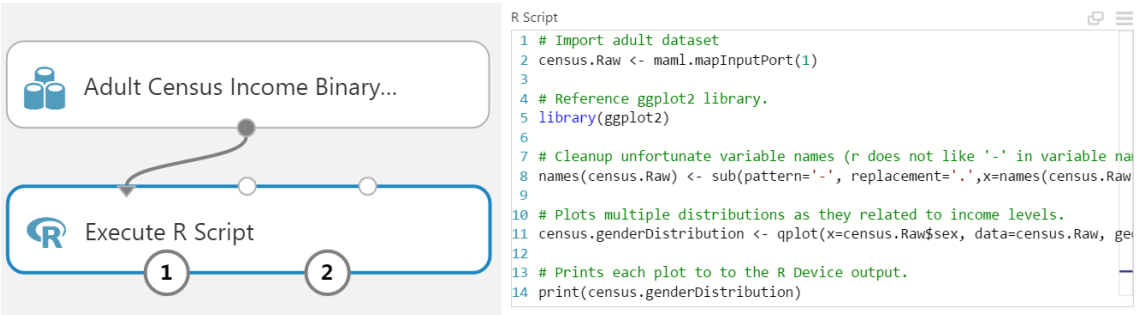


Figure 4.113: Inserted R code

4. Right click on the bottom right node of the **Execute R Script** module and select "Visualize" to view your plot (Figure: 4.114)

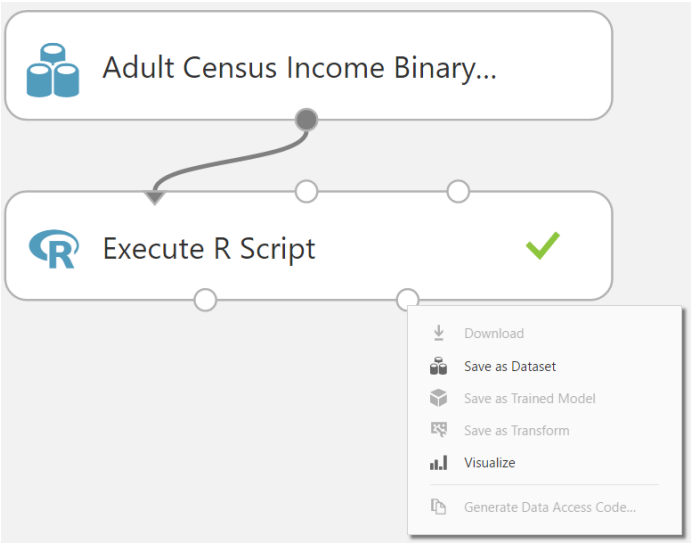


Figure 4.114: Visualization of Execute R Script module

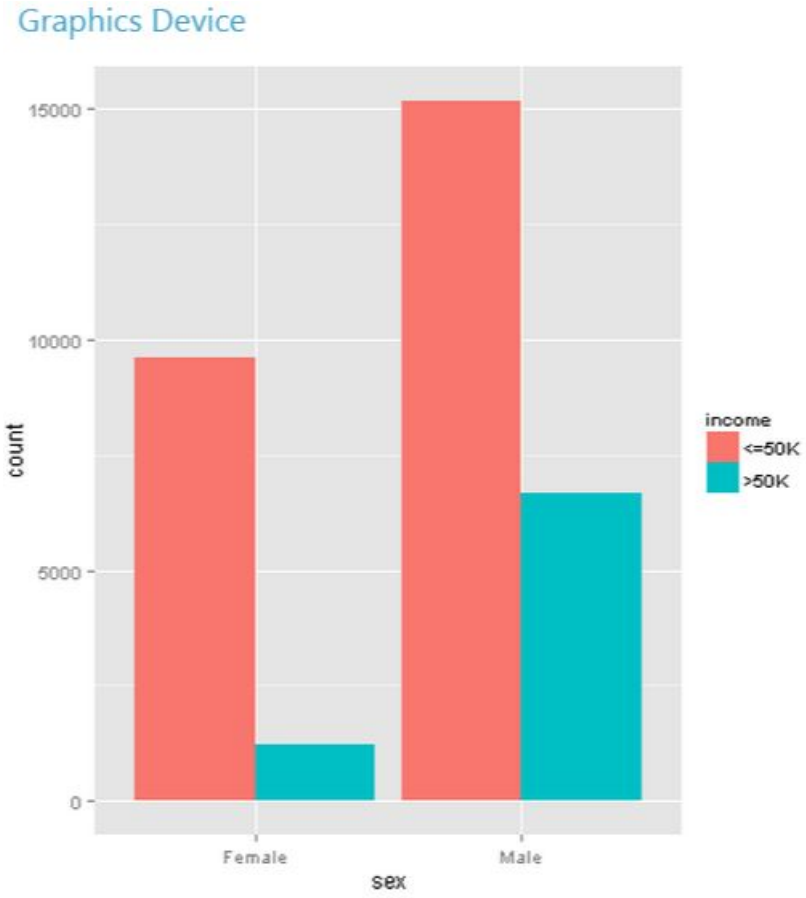


Figure 4.115: Visualization of Execute R Script module

4.7.6 Exercise: Multiple plots in R

Within the **Execute R Script** module, each graphical instance will be rendered and automatically added to the R Device output of the **Execute R Script** module. This is beneficial as in RStudio or R Console, you can only have one graphical plot at any given time. This means that calling multiple plots will overwrite the existing plots.

1. In the right menu of the **Execute R Script** module, insert the following code:

```
# Import adult dataset
census.Raw <- maml.mapInputPort(1)

# Reference ggplot2 library.
library(ggplot2)

# Cleanup unfortunate variable names (r does not like '-' in variable names)
names(census.Raw) <- sub(pattern='-', replacement='.',x=names(census.Raw))

# Plots multiple distributions as they related to income levels.
census.genderDistribution <- qplot(x=census.Raw$sex, data=census.Raw, geom="histogram", fill=census.Raw$income, position="dodge")
census.relationshipDistribution <- qplot(x=census.Raw$relationship, data=census.Raw, geom="histogram", fill=census.Raw$income, position="dodge");
census.ageDistribution <- qplot(x=census.Raw$age, data=census.Raw, geom="density", alpha=0.5, fill=census.Raw$income);
census.educationDistribution <- qplot(x=census.Raw$education.num, data=census.Raw, geom="density", alpha=0.5, fill=census.Raw$income);

# Prints each plot to to the R Device output.
print(census.genderDistribution);
print(census.relationshipDistribution)
print(census.ageDistribution)
print(census.educationDistribution)
```

After submitting your code, four visualizations will appear. Your output should appear as the visualizations in Figure: 4.116

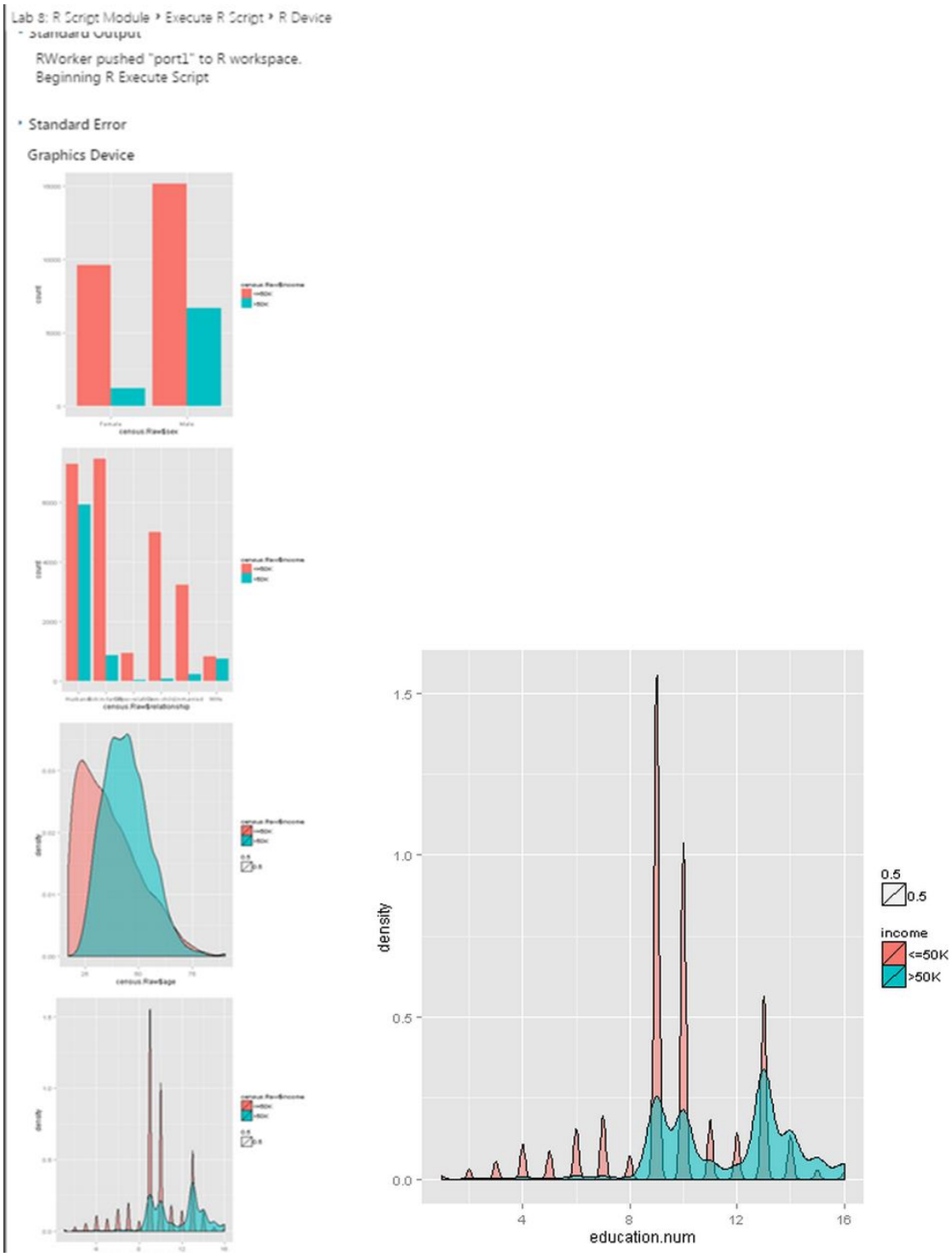


Figure 4.116: Visualizations of R Script module

4.7.7 Exercise: Reading R Scripts from the R Module

We will now be creating our own R script to be used within the R module.

1. In notepad, or any other IDE you prefer, create a new text file.
2. Insert the following code into the text file:

```
1 myPrt <- function(x){summary(x)}
```

"summary()" provides the summarized statistics of the data such as median, interquartile ranges, min, max, etc. The summary function is wrapped around a function and can be called with "myPrt(x)" where "x" is the input parameter of the data we are trying to summarize.

3. Save the text file as an R file named "myRscript.R".
4. Zip the R file we just created into "myRscript.zip".
5. Import the zip file as a new dataset, with the option to select "ZIP". To review how to do this, read over Exercise: Reading a Dataset from a Local File (Figure: 4.117).

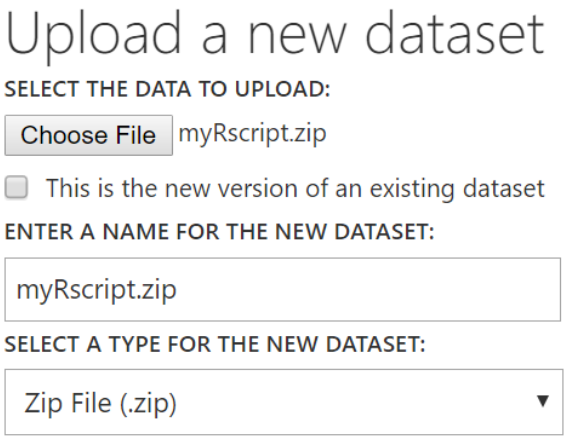


Figure 4.117: Import "myRscript.zip settings"

6. After successfully importing the data, search from the left menu under **Saved Datasets** for "myRscript.zip". Drag the dataset into your workspace (Figure: 4.118).

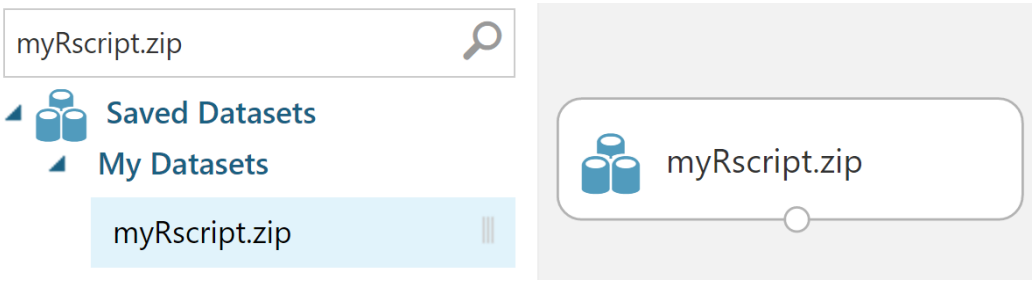


Figure 4.118: R script file under Saved Datasets

7. From the left menu, search for the **Execute R Script** module and drag it into the workspace.

8. In the left menu again, search for the **Iris Two-Class Data** module and drag it into the workspace (Figure: 4.119).

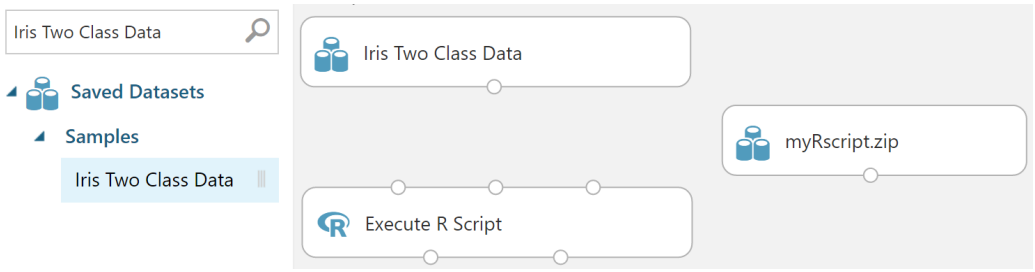


Figure 4.119: Search and drag the **Iris Two-Class Data** module

9. Connect the bottom node of the **Iris Two-Class Data** module to the top-left input node of the **Execute R Script** module (Figure: 4.120).
10. Connect the myRScript.zip data to the right input node of the **Execute R Script** module (Figure: 4.120).

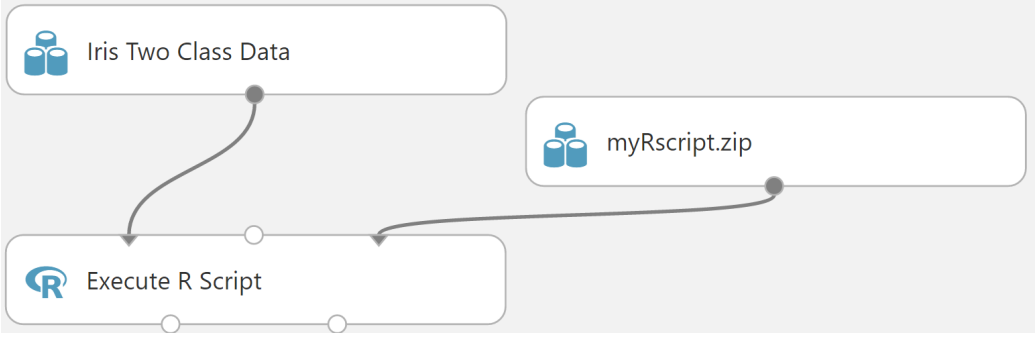


Figure 4.120: Connect the modules

11. Select **Execute R Script**. From the module menu, insert the following code:

```
# Importing & referencing the iris 2 class dataset.
iris.data <- maml.mapInputPort(1)

# Refering the zip file as a script. This will now act
   just like a library.
source("src/myRscript.R")

# Executes a function within the script file, in this case
   , being "myPrt()".
# myPrt()takes in a dataset and return a table of
   descriptive statistics.
iris.summaryTable <- myPrt(iris.data)

# Converts the table to a data.frame for output.
```

```
12 iris.SummaryDF <- as.data.frame.matrix(iris.summaryTable)
13
14 maml.mapOutputPort("iris.SummaryDF");
```

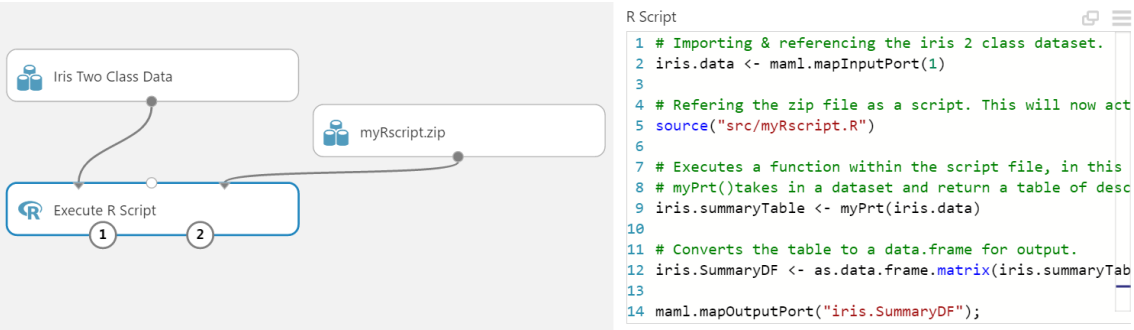


Figure 4.121: Inputted R code

12. Right-click on the bottom right node of the **Execute R Script** module and select "Visualize" to view the summarized statistics (Figure: 4.122)

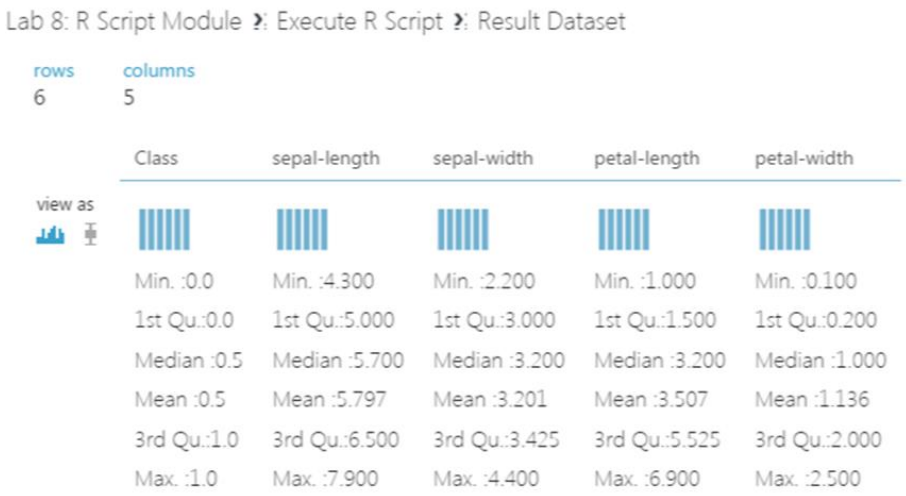


Figure 4.122: Visualization of R script module

4.8 Advanced Methods –Simultaneously Train, Score, and Evaluate Model

4.8.1 Exercise: Tune Model Hyperparameters

As we explored in Exercise: Fine-Tuning Across Different Algorithms choosing the best parameters to optimize the evaluation metrics for a specific model is an important task. It is time consuming to test the many possible combinations of parameters when training the model. Rather than manually doing all of this on our own, we can use the **Tune Model Hyperparameters** module to assist us with the task. However, it is important to note that this module should only be used as a starting point and guideline to get a sense of the behavior of the dataset. For this exercise, we will be continuing with the Titanic dataset, after we used the **Split Data** module in Exercise: Algorithm Selection and Training.

1. From the left menu, search for the **Tune Model Hyperparameters** module and drag it into your workspace.
2. For the purpose of this exercise, also search for the **Two-Class Boosted-Decision Tree** and drag it into your workspace.
3. Connect **Two-Class Boosted-Decision Tree** algorithm to the top-left node of the **Tune Model Hyperparameters** module. Connect the bottom-left node of the **Split Data** module to the top-left node of the **Tune Model Hyperparameters** module.

The top right node of the **Tune Model Hyperparameters** module is optional as it is for an optional validation dataset. In this exercise, we are using the test dataset generated from the **Split Data** module, so it may not be a good idea to use the same dataset to optimize the parameters and test the model. This may result in an unreliable inflation of evaluation metrics.

4. Highlight **Two-Class Boosted-Decision Tree** by clicking on the module.

5. In the "Properties" of the **Two-Class Boosted-Decision Tree**, input the parameters displayed in Figure: 4.123

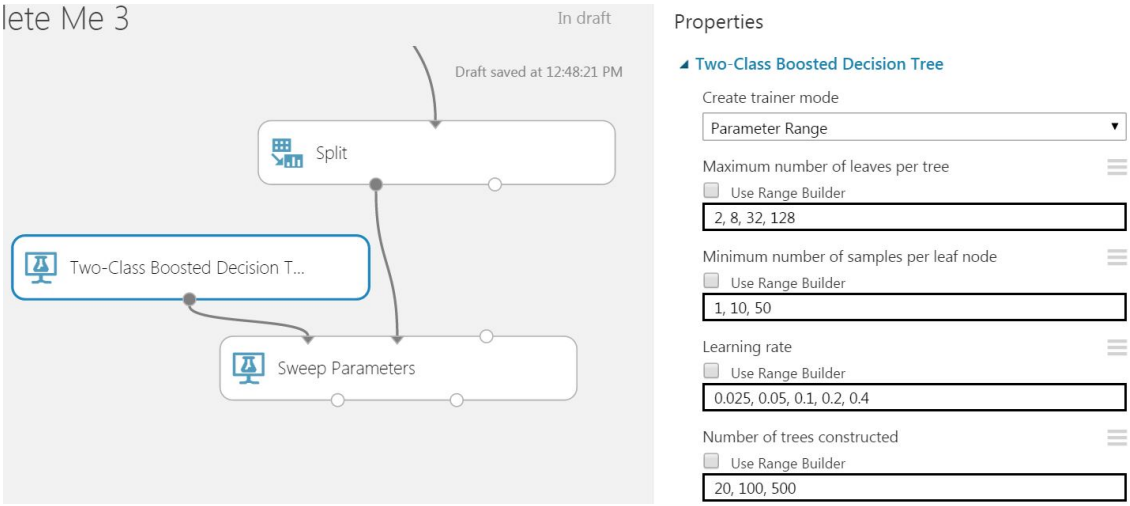


Figure 4.123: Two-Class Boosted Decision Tree parameters

Tip In "Properties", the "Create trainer mode" dropdown could be set to "Single Parameter". However under this setting, each parameter change would require a separate **Run** to view the result. By setting the "Create trainer mode" to "Parameter Range", a set of default values are assigned to each parameter optimization. These values can then be changed manually.

6. Highlight the **Tune Model Hyperparameterss** module by selecting the module.
7. Under "Properties", in "Specify parameter sweeping mode", select "Entire grid".

Tip The "Specify parameter sweeping mode" could also be set to "Random sweep". This option is useful when sweeping the entire grid for a big dataset is too time consuming.

8. Under "Label column", launch "selected columns" and select "Survived".
9. Set the "Metric for measuring performance for classification" to "Accuracy".

10. Leave the other parameters as they are and select **Run** (Figure: 4.124)

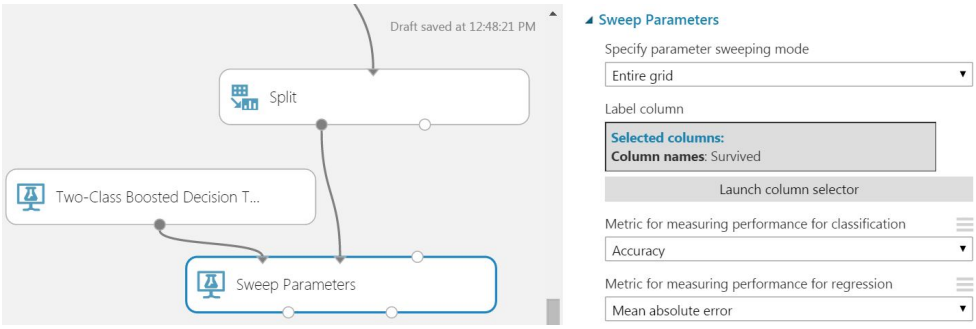


Figure 4.124: Tune Model Hyperparameters module settings

11. Right-click on the bottom left node of the **Tune Model Hyperparameters** module and select "Visualize" to view the tune results. Your visualized metrics should look like Figure: 4.125, sorted in descending order.

Number of leaves	Minimum leaf instances	Learning rate	Number of trees	Accuracy	Precision	Recall	F-Score	AUC	Average Log Loss	Training Log Loss
8	10	0.4	20	0.831461	0.788889	0.793296	0.791086	0.867591	0.544634	19.183212
8	50	0.2	100	0.831461	0.802326	0.77095	0.786325	0.873031	0.482608	28.387181
32	50	0.2	100	0.831461	0.802326	0.77095	0.786325	0.873031	0.482608	28.387181
128	50	0.2	100	0.831461	0.802326	0.77095	0.786325	0.873031	0.482608	28.387181
8	1	0.1	500	0.826966	0.796512	0.765363	0.780627	0.864903	0.707194	-4.938539
8	1	0.2	20	0.826966	0.793103	0.77095	0.78187	0.864557	0.488075	27.575877

Figure 4.125: Visualized metrics

12. From the left menu, search for the **Score Model** module and drag it into your workspace. This module is used to utilize the best trained model for scoring.

The **Tune Model Hyperparameters** module replaces the **Train Model** module. The **Tune Model Hyperparameters** module, outputs the best trained model with the best scoring parameters. To view the selected parameters, you can right-click the bottom right node of the **Tune Model Hyperparameters** module and select "Visualize". The bottom right node is also called the "best trained model".

13. Connect the bottom right node of the **Tune Model Hyperparameters** module to the top left node of the **Score Model** module, also known as the "trained model node".
14. Connect the bottom right node of the **Split Data** module, also known as the "testing dataset node", to the top right node of the **Score Model** module.
15. To evaluate the model, search in the left menu for an **Evaluate Model** module and drag it into the workspace.

16. Connect the bottom node of the **Score Model** module to the top left node of the **Evaluate Model** module (Figure: 4.126)
17. Select **Run**.

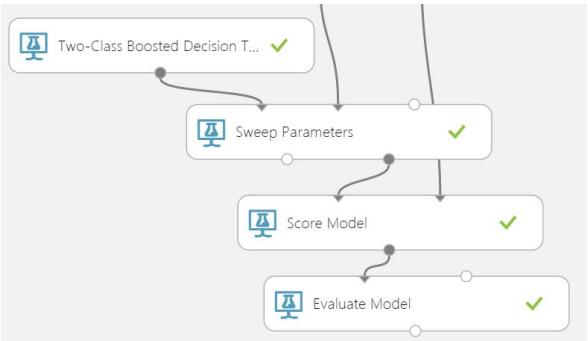


Figure 4.126: Connect the modules

4.8.2 Exercise: Cross-Validation

Cross-validation is a technique used to evaluate how a model's prediction will apply and generalize to another independent datasets. The goal of cross-validation is to define a dataset. When defined the model can be tested in the training phase, to limit problems that might arise, such as overfitting. The process of cross-validation involves partitioning a sample of data into complementary subsets. Then, on selected subsets, the model is trained and validated with the other subsets from the sample. There are multiple rounds of cross-validation using different partitions. The validation results are then averaged from the different rounds.

1. We will start experimenting with cross-validation in the Titanic experiment. We will begin after the stage of data preprocessing and algorithm selection (Exercise: Algorithm Selection and Training). We will not be using the **Split Data** module for this procedure.
2. From the left menu, search for the **Cross Validate Model** module and drag it into the workspace.
3. Connect the bottom node of the "Two-Class" algorithm you previously selected to the top left node of the **Cross Validate Model** module, or the untrained model node.
4. Connect the bottom node of the **Metadata Editor**, the result of the pre-processed data, to the top right node of the **Cross Validate Model** module, or the dataset node.
5. Highlight the **Cross Validate Model** module by clicking the module. In the right menu, under "Label column", choose "Selected columns", and select "Survived" as the label.

6. Select **Run** (Figure: 4.127).

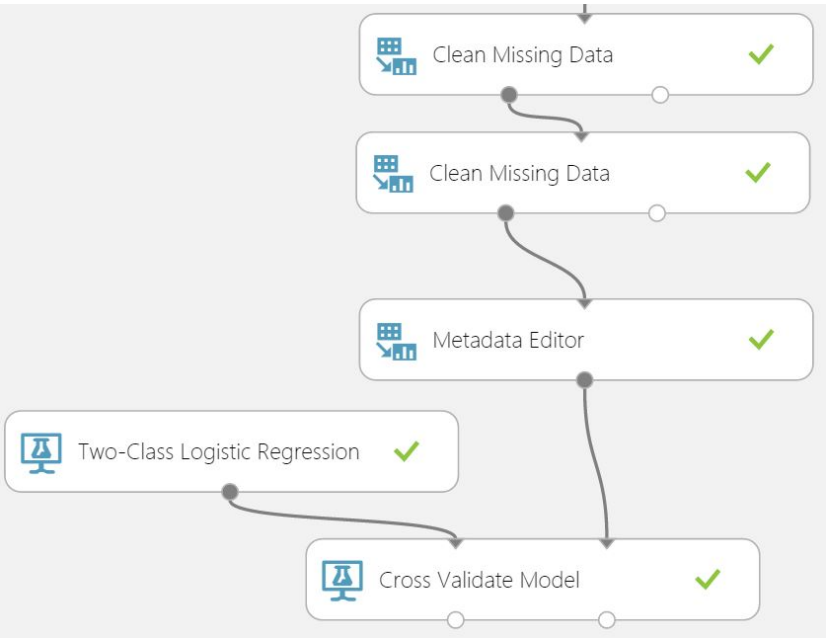


Figure 4.127: Run the modules

7. Right-click the bottom right node of the **Cross Validate Model** module and select "Visualize" to visualize the evaluation result. The output should look like Figure: 4.128

Fold Number	Number of examples in fold	Model	Accuracy	Precision	Recall	F-Score	AUC	Average Log Loss	Training Log Loss
0	88	Logistic Regression	0.806818	0.846154	0.628571	0.721311	0.846361	0.462863	31.129682
1	89	Logistic Regression	0.820225	0.774194	0.727273	0.75	0.917208	0.369453	43.968971
2	89	Logistic Regression	0.797753	0.709677	0.709677	0.709677	0.848443	0.448781	30.571892
3	89	Logistic Regression	0.786517	0.736842	0.756757	0.746667	0.835239	0.478442	29.524462
4	89	Logistic Regression	0.719101	0.6	0.727273	0.657534	0.80763	0.514999	21.895699
5	89	Logistic Regression	0.842697	0.764706	0.8125	0.787879	0.902412	0.382354	41.460861
6	89	Logistic Regression	0.820225	0.733333	0.733333	0.733333	0.890678	0.393252	38.46574
7	89	Logistic Regression	0.797753	0.896552	0.634146	0.742857	0.856453	0.471311	31.699075
8	89	Logistic Regression	0.786517	0.735294	0.714286	0.724638	0.832011	0.48489	27.648169
9	89	Logistic Regression	0.775281	0.782609	0.545455	0.642857	0.768939	0.528806	19.801688
Mean	889	Logistic Regression	0.795289	0.757936	0.698927	0.721675	0.850537	0.453515	31.616624
Standard Deviation	889	Logistic Regression	0.033325	0.079408	0.076074	0.043299	0.044573	0.055018	7.831206

Figure 4.128: Visualize the Cross Validate Model module

Tip By default, the **Cross Validate Model** module uses the k-fold cross-validation method, partitioning the dataset into 10 equal sized subsamples (k=10). For each step in the cross-validation process, a single subsample is retained as validation data to test the model. The remaining k-1 subsamples are used as training data. This process of training and validation is repeated 10 times. To customize the number of partitions that occur, the **Partition and Sample** module can be added into the workspace, immediately before the **Cross Validate Model** module.

8. Select the bottom left node of the **Cross Validate Model** module, the scored result node, and select "Visualize" to view the results.

4.9 Entity Extractor

The Azure Machine Learning Studio has more capabilities beyond creating machine learning models. One feature of Azure ML is the entity extractor. This tool is capable of extracting people, places, or organizations from any given string. For example, when the contents of *Harry Potter and the Sorcerer's Stone* are input into the entity extractor, the most frequent names are extracted as in Figure: 4.129

Entity	Type	Frequency
Harry	Person	1234
Ron	Person	385
Hermione	Person	206
Snape	Person	131
Dudley	Person	126
Neville	Person	102
McGonagall	Person	98
Hagrid	Person	89
Vernon	Person	75
Dursley	Person	52

Figure 4.129: Entity extractor frequent names

4.9.1 Exercise: Creating the Entity Extractor Model

1. To begin experimenting with the entity extractor, create a new project by selecting **+New** / "Blank Experiment".
2. From the left menu, search for the **Named Entity Recognition** module and drag it into the workspace.

3. Use the menu again to search for the **Named Entity Recognition Sample Articles**. Right-click on the bottom node of the **Named Entity Recognition Sample Articles** and select "Visualize" to view the output. Your visualized data should look like Figure: 4.130

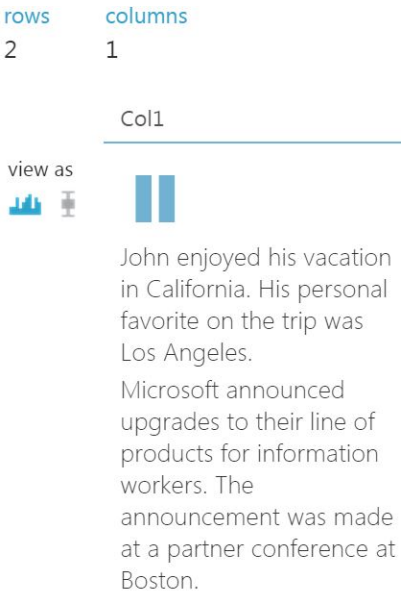


Figure 4.130: Visualization of Named Entity Recognition Sample

When visualizing the output, you will see the sample sentence being used to test the entity extractor. We are looking to see if it can correctly identity "John", "California", "Los Angeles", "Microsoft", and "Boston" as entities.

4. Connect the bottom node of the **Named Entity Recognition Sample Articles** to the top left input node of the **Named Entity Recognition** module
5. Select **Run**.
6. Right-click on the bottom output node of the **Named Entity Recognition** module and select "Visualize" to view the data. Your visualization should look like Figure: 4.131

Figure 4.131: Visualization of Named Entity Recognition module

You will notice from you visualization that the entity recognition feature not only correctly identified the entities, but quantified their "Offest", "Length", and "Type".

7. Deploy your model (Exercise: Deploying Your Model) and try the exercise again, inputting a few pages from *Harry Potter and the Sorcerer's Stone* into the entity extractor. An example of this process can be seen in Figure: 4.132. You can find the text to input at: `http://www2.sdfi.edu.cn/netclass/jiaoan/englit/download/Harry%20Potter%20and%20the%20Sorcerer%27s%20Stone.pdf`

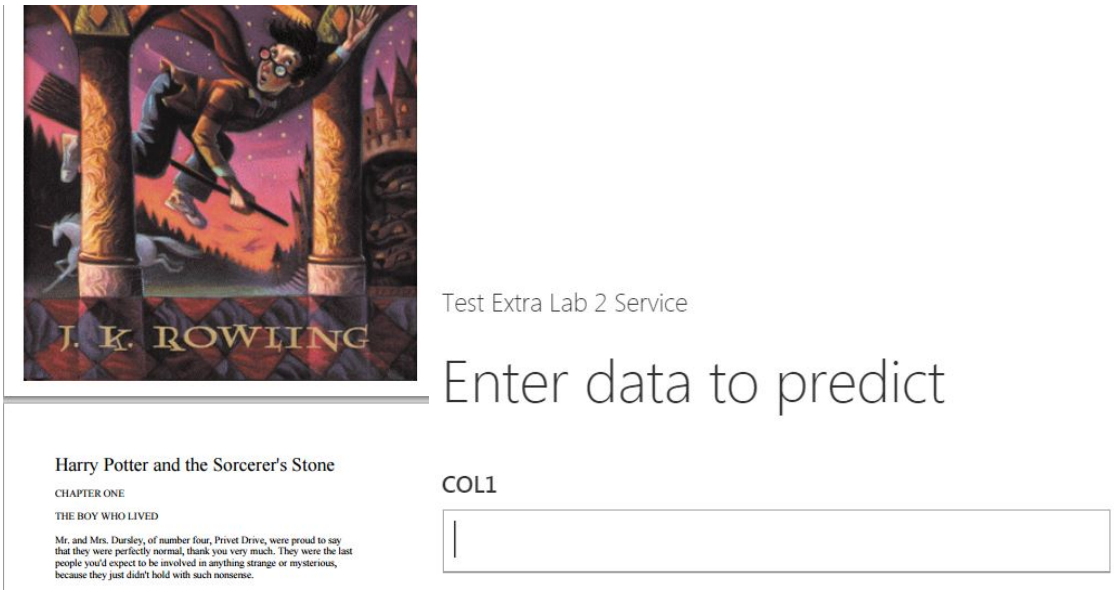


Figure 4.132: Inputting sample book text

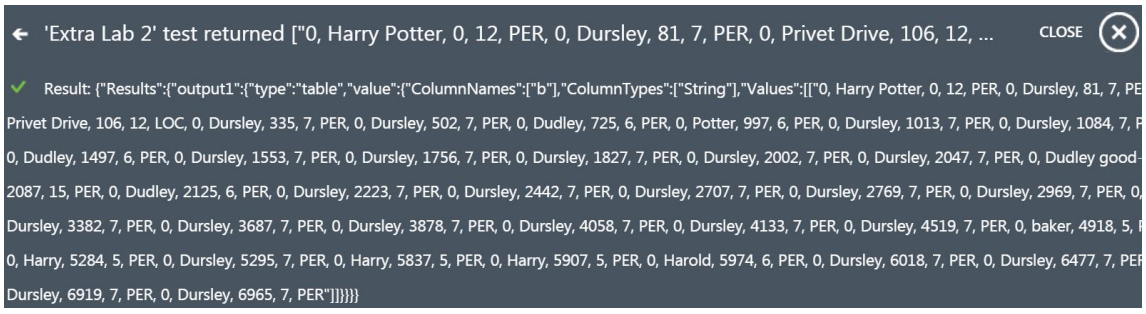


Figure 4.133: Entity extractor output

The exercise we have just completed has been exported and wrapped into a Django web-app on DataScienceDojo demos website (Figure: 4.134). In our app, you can paste the rest of *Harry Potter and the Sorcerer's Stone* or any other text you would like to extract entities from. You can check out our app at: `http://demos.datasciencedojo.com/demo/entityextractor`.

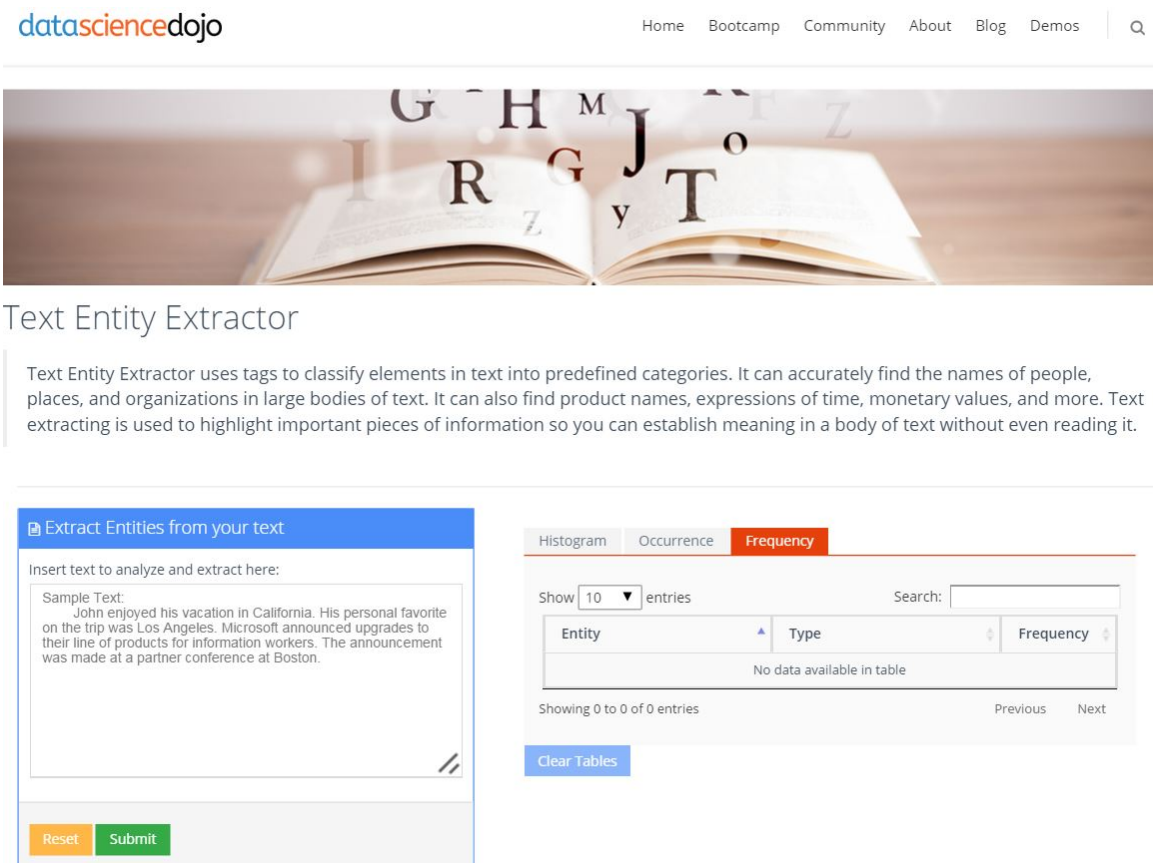


Figure 4.134: Text Entity Extractor Demo

4.10 More R in Azure Machine Learning Studio

In this section, we will build off of the previous section on the R Script Module, exploring more features of R in Azure ML.

4.10.1 Exercise: Hello World

1. From the left menu, under **Saved Datasets**, find the "Titanic Dataset" and drag it into your workspace.
2. Search in the menu again for the **Execute R Script** module and drag it into your workspace.
3. Connect the bottom output node of the "Titanic Dataset" to the top left node of the **Execute R Script** module (Figure: 4.135)

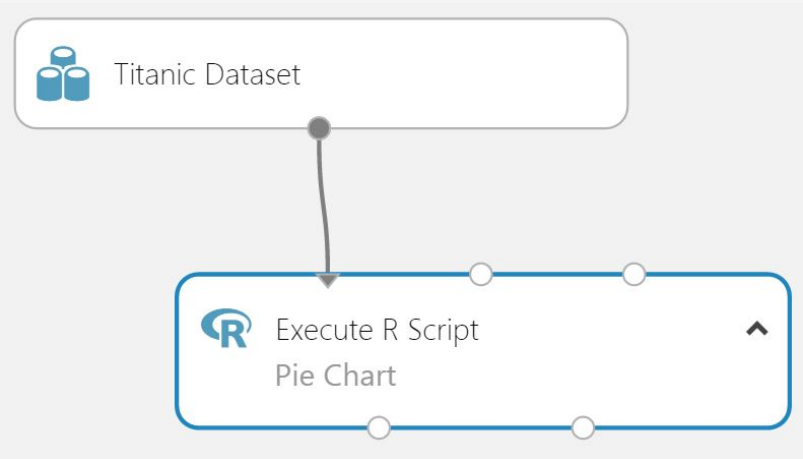


Figure 4.135: Connecting the Titanic Dataset

4. In the **Execute R Script** module menu, insert the following code:

```
dataset1 <- maml.mapInputPort(1)
data.set <- data.frame(response=paste0("Hello ",dataset1$
   Name,"!"))
maml.mapOutputPort("data.set");
```

Tip Notice that "dataset1" refers to the Titanic dataset because we assigned the dataset to a variable called "dataset1". Also keep in mind that calling "dataset1$Name" calls the "Name" column from the Titanic dataset. Finally, "paste0()" is a form of string concatenation in R, meaning this function merges "Hello" and "<PersonName>" together.

5. **Run** the experiment, right-click on the bottom right output node of the **Execute R Script** module and select "Visualize" to view the output.

Your visualization should look like Figure: 4.136. You will see that the table now says "Hello" to every individual name.

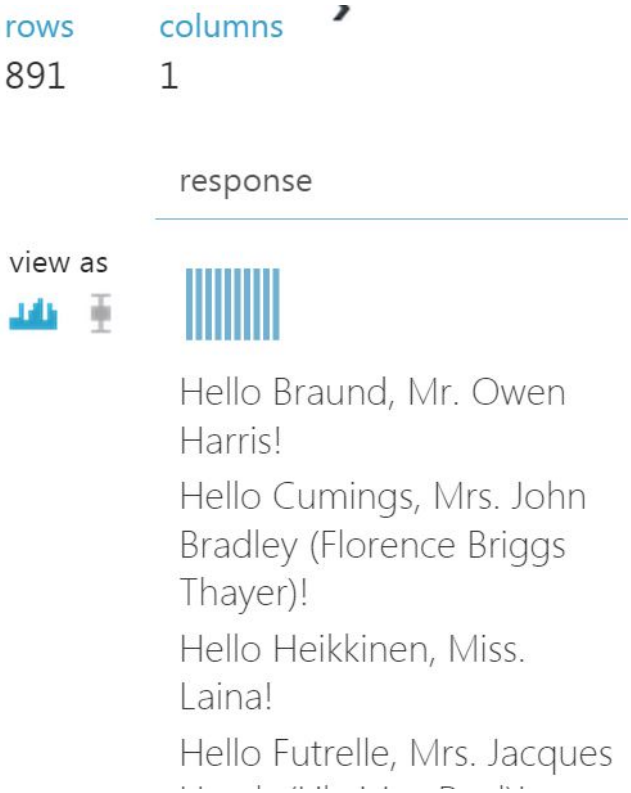


Figure 4.136: Titanic datset text visualization

6. Deploy this model as a web service (Exercise: Deploying Your Model). Then test the model in the API deployment screen with different strings to receive various "Hello" "<string>" outputs.

4.10.2 Exercise: Pie Charts

1. From the left menu, under **Saved Datasets**, find the "Titanic Dataset" and drag it into your workspace.
2. Search in the menu again for the **Execute R Script** module and drag it into your workspace.
3. Connect the bottom output node of the "Titanic Dataset" to the top left node of the **Execute R Script** module.
4. In the **Execute R Script** module menu, insert the following code:

```
dataset1 <- maml.mapInputPort(1)

gender <- dataset1$Sex
slices = table(gender)
pie(slices)

maml.mapOutputPort("dataset1");
```

Tip "dataset1$Sex" retrieves the "Sex" column from the data.frame. It then assigns the single column row to a variable called "gender". The variable "gender" is a one-column wide data.frame, while "pie()" takes in the table parameter and not the data.frame. This code allows us to convert "gender" from a data.frame into a table. The data is then plotted into a pie chart with the "pie()".

5. **Run** the experiment, right-click on the bottom right output node of the **Execute R Script** module and select "Visualize" to view the output. Your visualization should look like Figure: 4.137, showing the male to female distribution.

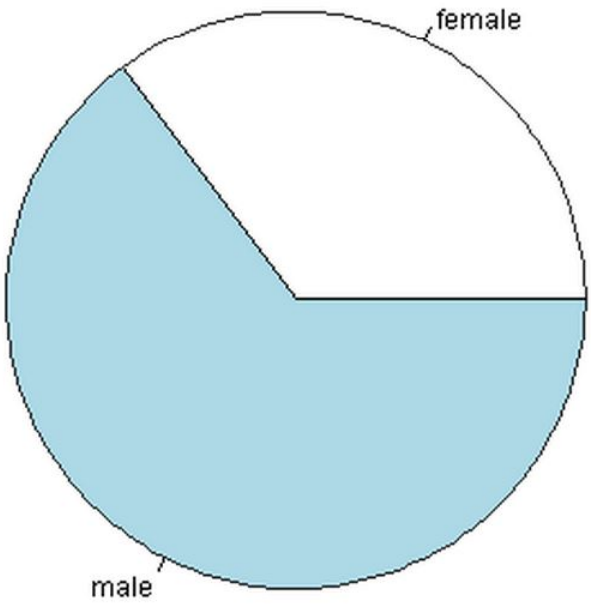


Figure 4.137: Visualization of gender

Now that we have a basic visualization, we can add features to the pie chart to make it more communicative and engaging.

6. Rename the labels, add color, and add a title to the pie chart by inputting the following code in the **Execute R Script** module:

```
dataset1 <- maml.mapInputPort(1)

gender <- dataset1$Sex
slices = table(gender)
lbs = c("Male", "Female")
color = c("red","blue")
title = "Gender Distribution"
pie(slices, label=lbs, col=color, main=title)

maml.mapOutputPort("dataset1");
```

7. **Run** the experiment, right-click on the bottom right output node of the **Execute R Script** module and select "Visualize" to view the output. Your visualization should look like Figure: 4.138, reflecting the visual changes we made.

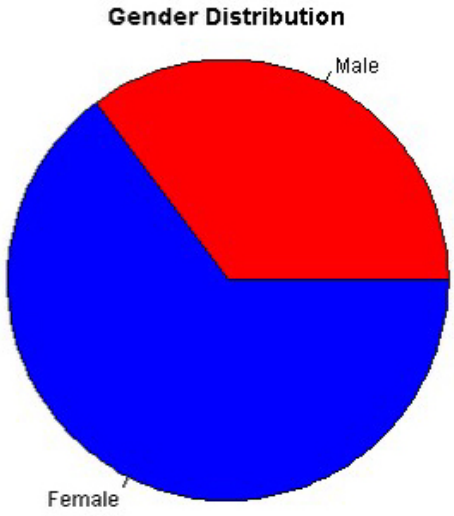


Figure 4.138: New gender visualization