

Machine Learning

What is Azure Machine Learning?

Contents

The Problem Domain

Enter Workspace

Getting Started

Data Pre-Processing

Training the Model

Comparing Classifiers

Conclusion

Introduction

First Time Setup

The Data

Data Processing

Training the Model

Comparing Classifiers

Publish as a Web Service

Visualise Results

Conclusion

Extra Resources

What is Azure Machine Learning?

This lab will introduce you to machine learning capabilities available in Microsoft Azure, specifically Microsoft Azure Machine Learning (Azure ML) and the Studio options.

Azure ML is a fully managed machine learning platform that allows you to perform predictive analytics. Development of models (experiments) is achieved using the Azure ML Studio, a web based design environment that empowers both data scientists and domain specialists to build end-to-end solutions and significantly reduce the complexity of building and publishing predictive models. It has a simple drag-and-drop authoring interface and a catalogue of modules that provide functionality for an end-to-end workflow. More experienced users can also embed their own Python or R scripts inline and explore the data interactively with Jupyter Notebooks.

One of the most important features of Azure ML is its publishing service where by a finished (trained) experiment can be exposed as a web API that can be consumed by any other applications, such as a website or mobile application etc. Each experiment can also be configured to be re-trained with new data using a separate API to maintain it.

In this short lab, we’ll explore how Azure ML works by looking at a simple scenario, involving predicting the price of a car give a list of other attributes about the car. This will be a regression based experiment and we will see how built in modules alongside Python scripts and Jupyter Notebooks can be used to aid in the exploration and evaluation of the data.

In future labs, we’ll see how to publish the model from within the studio to create an API for your model. Then review the sample code for a published experiment and test the output from an Excel add-in/application.

The Problem Domain

During this lab, we will create and evaluate two models, that given past data collected about cars and their values, will try to predict the price/value of a car given other attributes associated with the car:

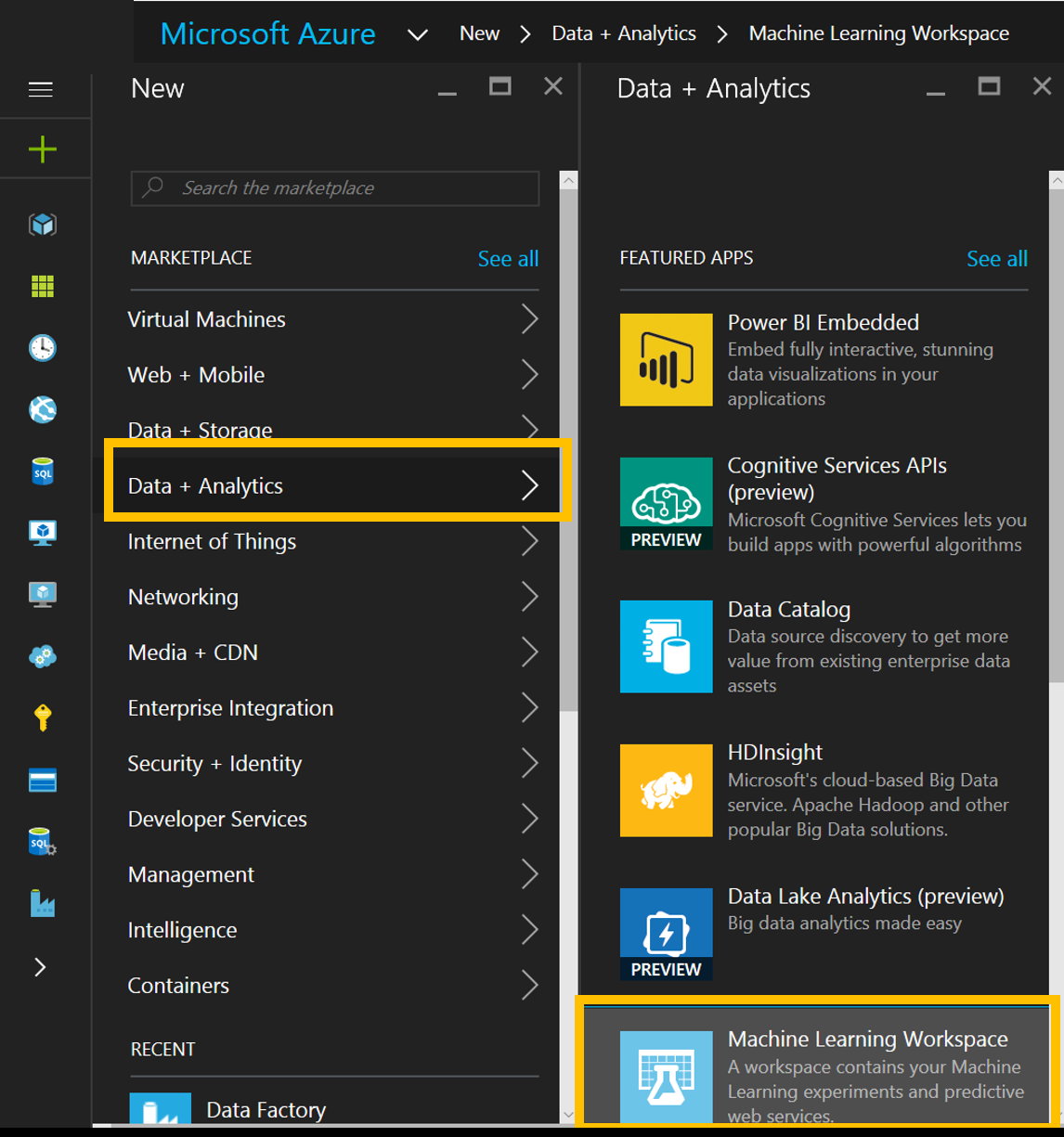
* The attribute columns in the dataset include values such as the model/make, fuel type and body style as well as performance values such as MPG, horsepower and engine type
* The value we are trying to predict is the price of the car. In this dataset, the values range from £5,000 to £45,000.

We will retrieve data from an Azure Blob Storage account and start to pre-process the dataset ready to train a machine learning model. The model we’ll create is a form of supervised learning so we will use historical car attributes and values to predict the price of future cars we might receive. This model will perform a regression algorithm to try and predict the actual price of the car with the lowest amount of error, for example £16,595. This information is in the sample data in the ‘price’ column.

Enter Workspace

In a previous lab, you should have setup your **Azure Subscription** with your university email address and created an **Azure Machine Learning Workspace** as shown in the screenshot below.

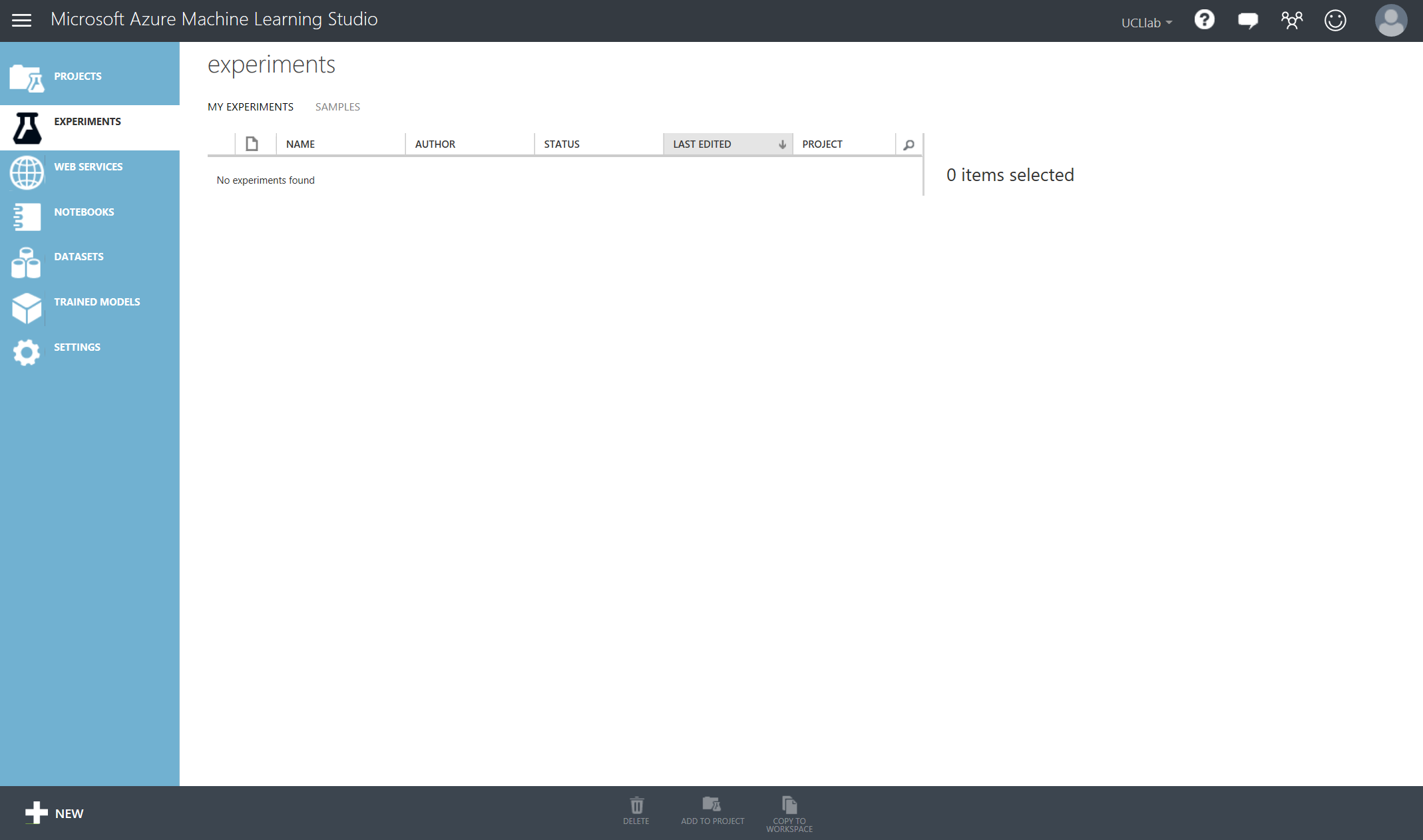
*If you have not already setup these resources, please let a member of staff know.*



Please make sure you revisit the same workspace you used previously by going to <http://studio.azureml.net> . Next choose the option to **’Sign In’** and provide your azure subscription details.

This will take you to the Azure Machine Learning Studio, by default you will end in the Experiments tab on the left but there are other tabs available and described below:

* **Projects:** Like a file-explorer, a place to keep all associated resources together for a similar project. You may look at creating a project called ‘AzureMLStudioLab’ to keep all the experiments, notebooks and datasets you create today in one place that’s well organised.
* **Experiments:** As you start creating machine learning experiments/models you will build up a list of different experiments that can be add, edited and deleted by an owner of the workspace
* **Web Services:** Once you have created a model and are ready to make it a production-ready API, this is the section you will find all your web services listed under
* **Notebooks:** You may have already explored notebooks. Although the Studio provides an easy to use, yet powerful, drag-drop style of creating experiments, you sometimes need a good old [“REPL”](https://en.wikipedia.org/wiki/Read%E2%80%93eval%E2%80%93print_loop) to have a tight loop where you enter some script code and get a response. This functionality is integrated into ML Studio through Jupyter Notebooks
* **Datasets:** A place to store datasets/folders of scripts or functions (CSV, TSV,.zip) that you upload from your local machine
* **Trained Models:** Each time you create a model and start scoring against it to create a web service, this will create a ‘Trained Model’ module that will be reusable. This will be your chosen algorithm, trained on your chosen data in order to answer a machine learning question.
* **Settings:** The settings tab allows you to add descriptions to your workspace as well as add new users so you can work in a more collaborative enviroment when designing your machine learning algorithms

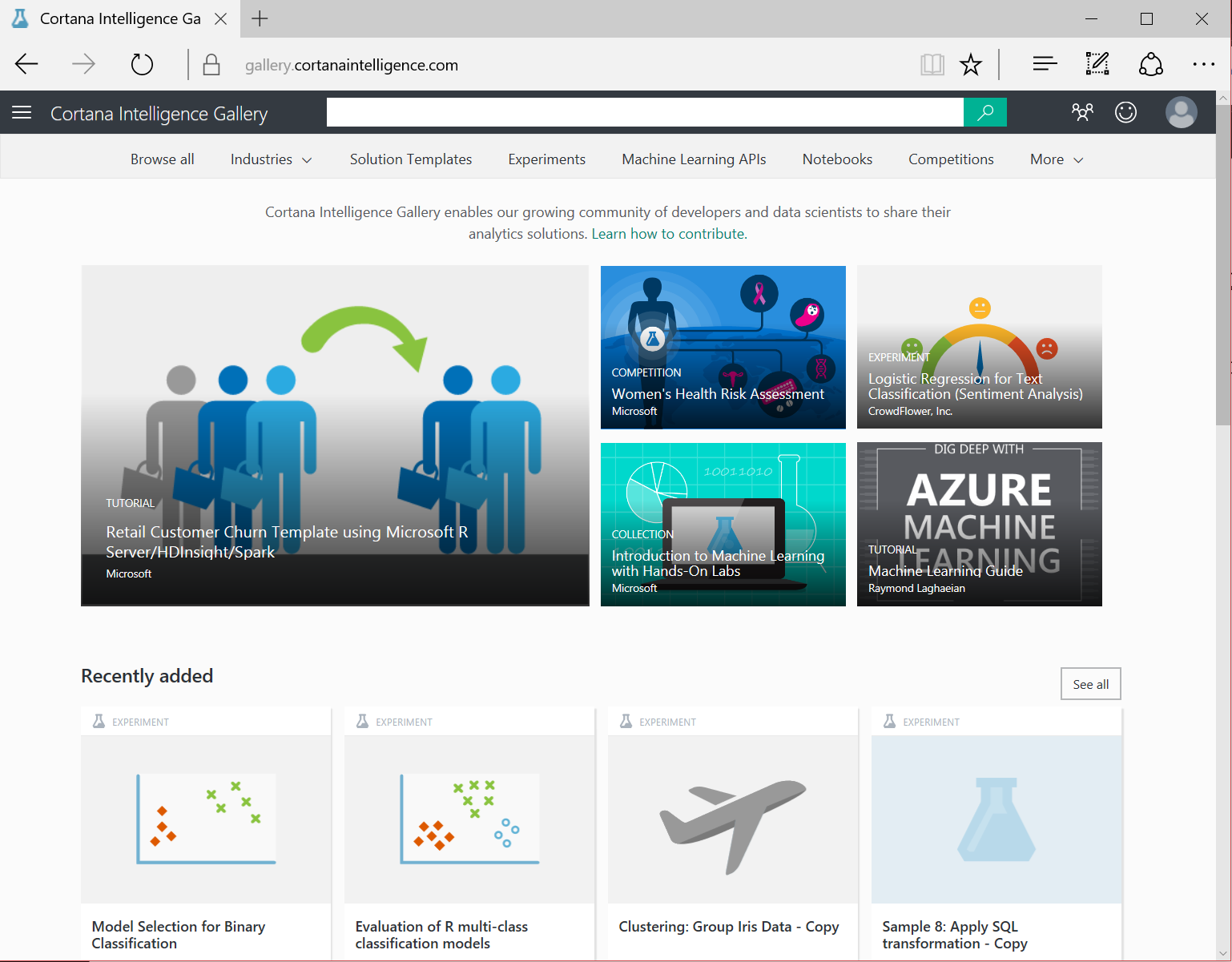


Getting Started

Now we can start building our experiment. Typically, we would get hold of a data set and either connect to its source or upload a file:

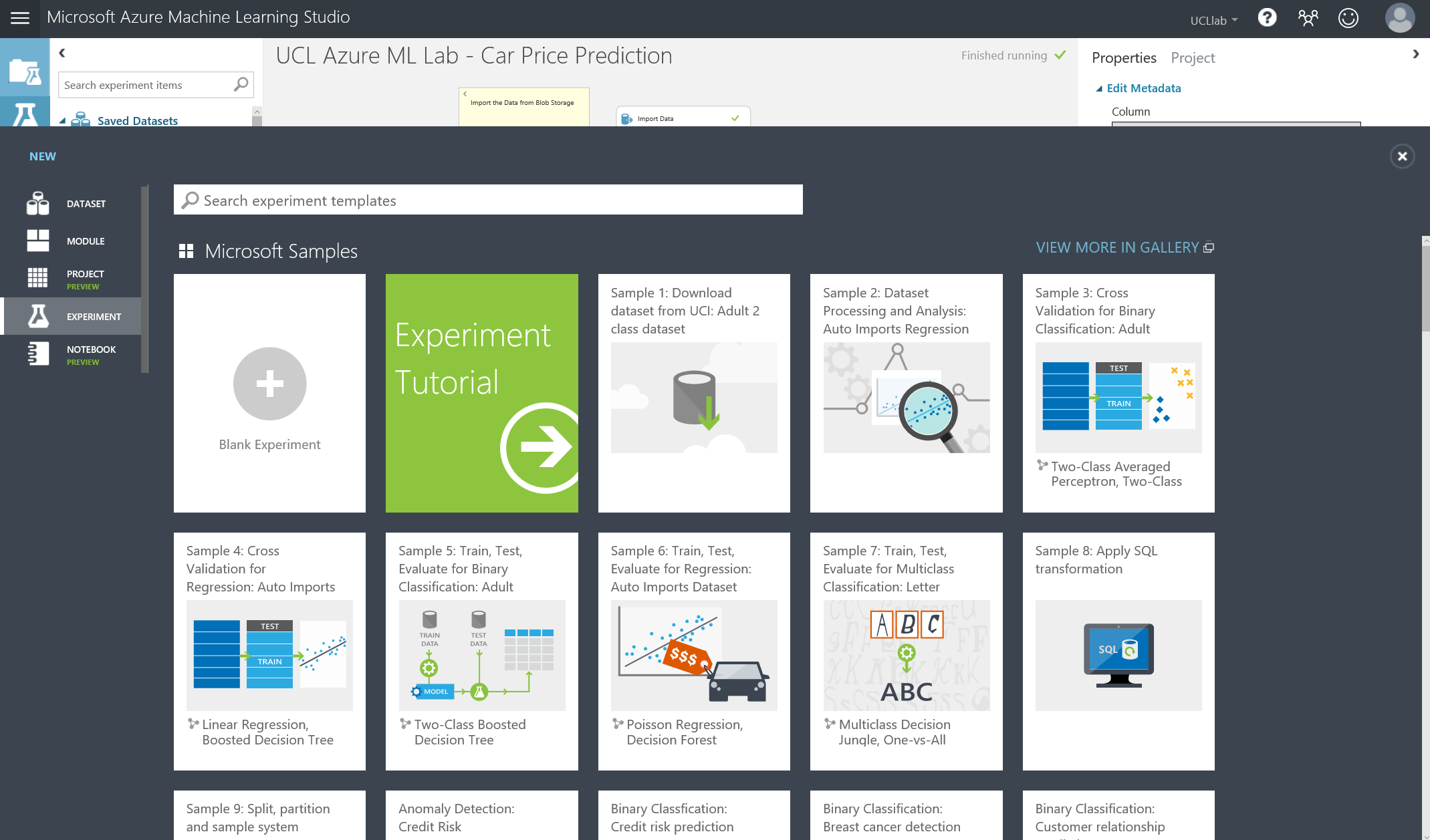
* Azure ML accepts many formats including CSV, TSV, Plain Text and Zip files via upload
* Alternatively, you can use the ‘Import Data’ Module to access your data from sources such as Azure SQL DB, Blob storage, HDInsight and industry standard OData feeds.

***[EXTRA INFORMATION:]*** *It’s also possible to adapt an existing experiment to your needs from one of the many samples in the* [*Cortana Intelligence Gallery*](https://gallery.cortanaintelligence.com/) *– you can change it to use your own data and parameters and quickly see how suitable it is for your needs:*

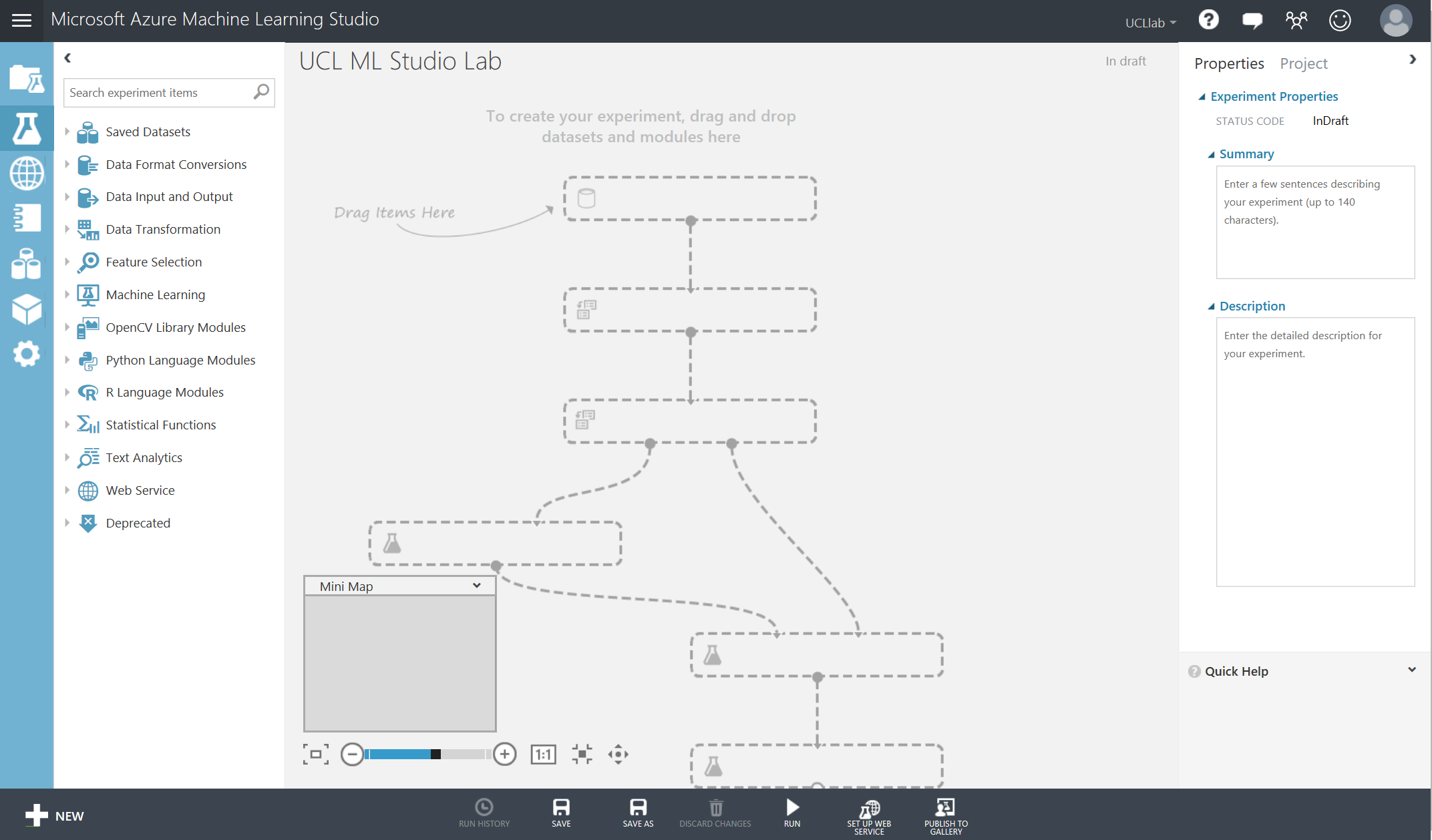


For this lab, we will be creating a new experiment and using the ‘Import Data’ module to read in data from an Azure Blob Storage account.

* Let’s start by creating a new experiment: Bottom left corner choose ‘New’ -> ‘Experiment’ -> ‘Blank Experiment’



* Next change the title of the experiment to something descriptive, for example: UCL ML Studio Lab

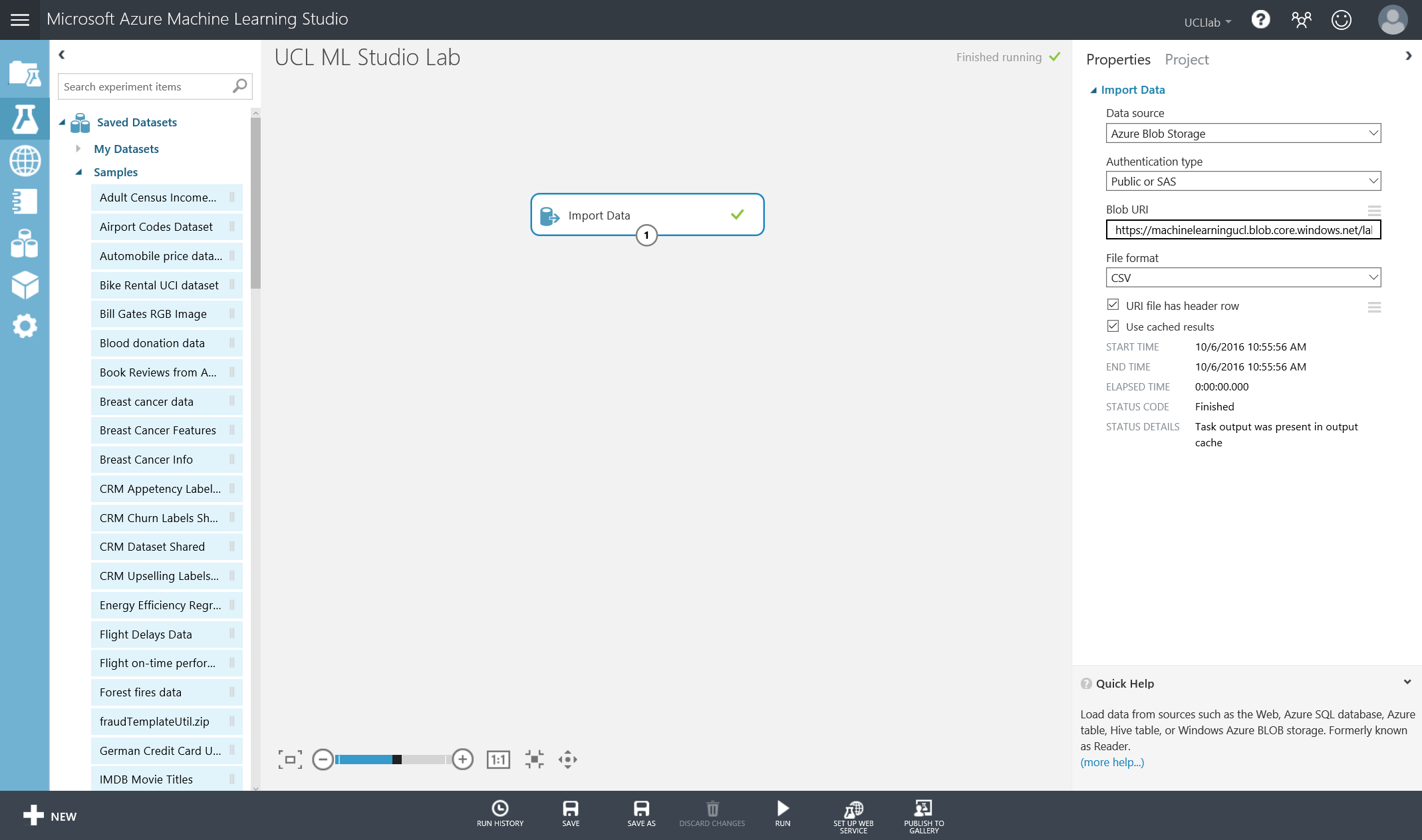


On the left in blue with white icons are the different objects we use in ML studio such as Projects, Experiments, Web services, Jupyter Notebooks, Data Sets, Trained Models and Settings. Next on the left is a nested list of all the modules we can drag on to the design surface in the middle of the screen. On the right is the properties pane, this will change dynamically for each module or properties for the whole experiment if no module is selected. There are also functions we can perform in the bottom dark grey toolbar such as Save, Run and Setup Web Service.

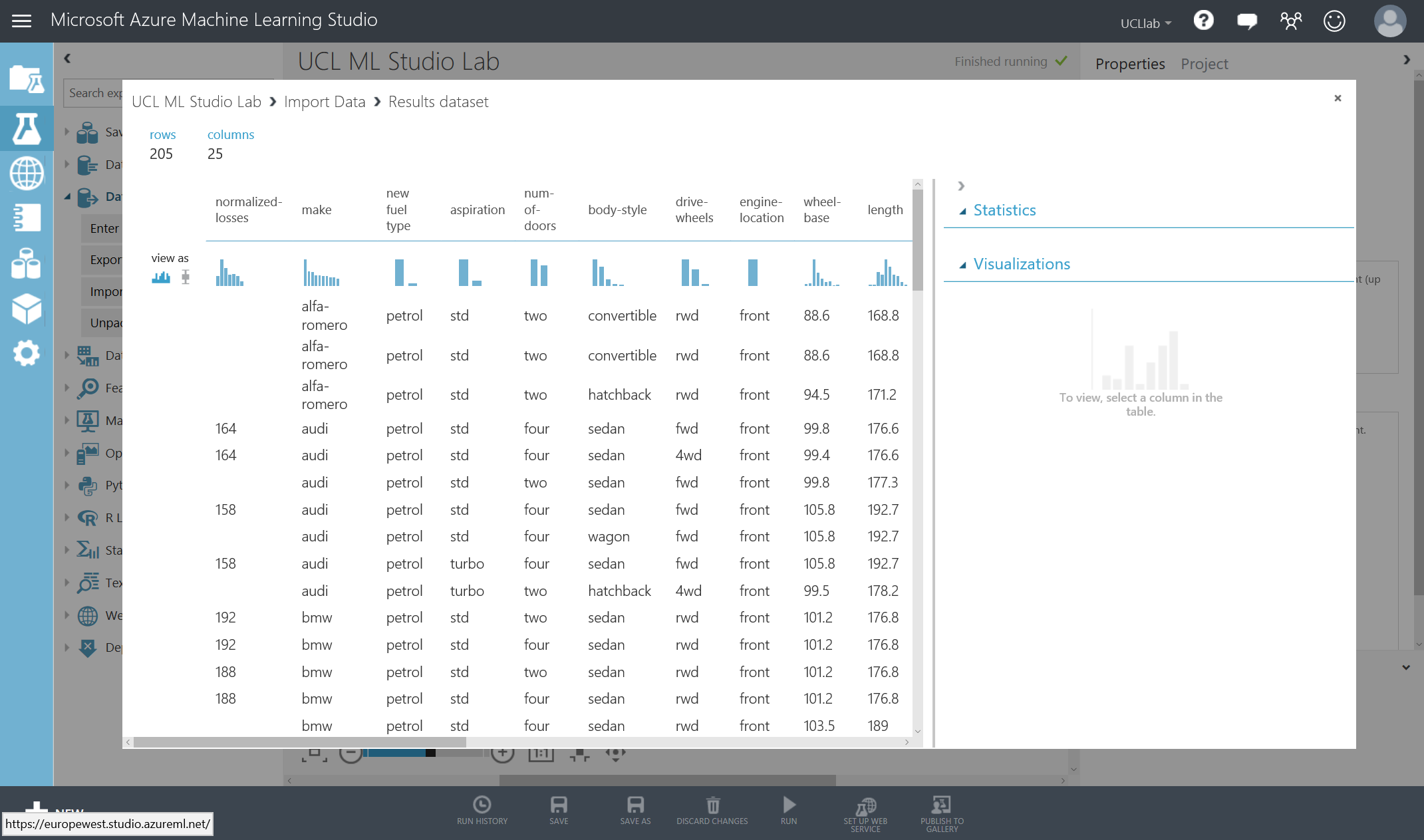
From the list of nested modules on the left, choose ‘Data Input and Output’ and then click and drag ‘Import Data’ onto the grey workspace. Over on the properties pane, enter the details below. This will allow access to a Car Price Prediction Dataset from an Azure Blob Storage account:

* **Data Source:** Azure Blob Storage
* **Authentication Type:** Public or SAS
* **Blob URI:** <https://machinelearningucl.blob.core.windows.net/lab/carPriceData.csv>
* **File Format:** CSV
* **File Has Header Row:** CHECKED
* **Use Cached Results:** CHECKED

Once these details are entered, go to the functions toolbar at the bottom of the page and click ‘Run’. Once completed processing the ‘Import Data’ module will have a green tick by it, like below



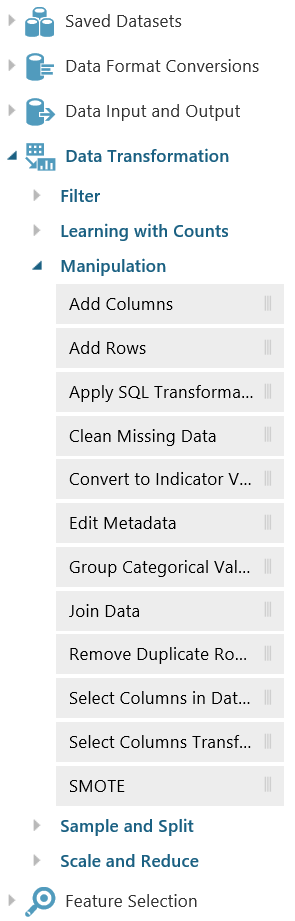
Now let’s look at the data. Hover over the output port of the Import Data Module. As you hover over it, it will say ‘Results Dataset’, right click and choose ‘Visualise’ to see a snapshot of the dataset and some summary information



The ‘Visualise Data’ option is extremely useful as you work through building your experiment. I encourage you to visualise the dataset after each module change to see how the transformation or function affected your original data. Things to note are:

* The number of rows and columns in the dataset (top left)
* The names of the columns in the dataset
* If you click on any of the columns you will see on the right of the pop up some statistics such as data type and range of values or amount of missing values
* Also, each column will show a histogram of the distribution of the data
* Note as you scroll down in this window you cannot scroll through the whole dataset, this is only showing the head of the dataset

Data Pre-Processing

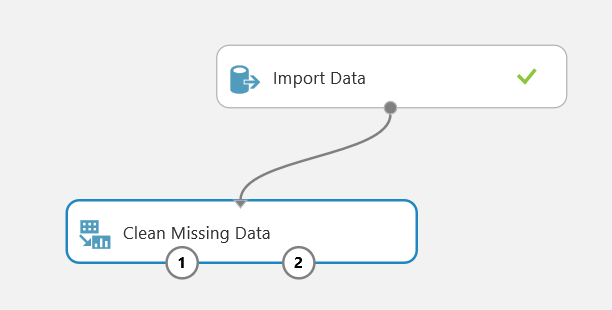
In this section, we will get the dataset into a form that is suitable to train a machine learning model.

The major data preparation tasks include data cleaning, integration, transformation, reduction, and discretization or quantization.

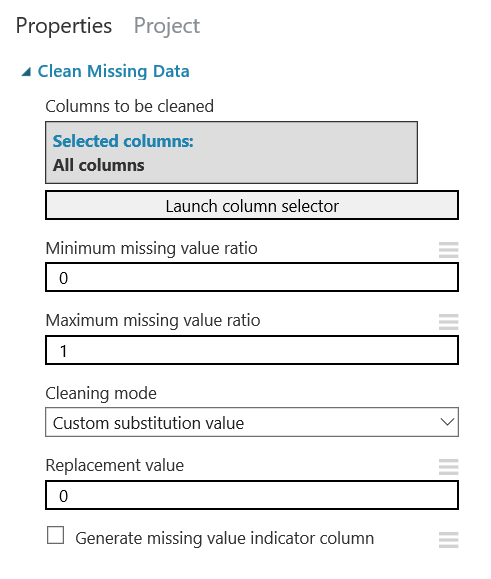
In Azure ML studio, you can find modules to perform all these operations and other data pre-processing tasks in the **Data Transformation** group in the left panel, we will look at a couple of built in modules, listed to the left. We will also see who we can integrate Jupyter Notebooks and Python code to help use cleanse and understand our data.

Let’s start by cleaning some of the missing values in the dataset. For example, have a look at the ‘normalised losses’ column – there are 41 missing values.

Click and drag the ‘Clean Missing Data’ module in the Data Transformation modules on the left of the screen into the experiment space. Then connect the output of the ‘Import Data’ module to the input port of the ‘Clean Missing Data’ module



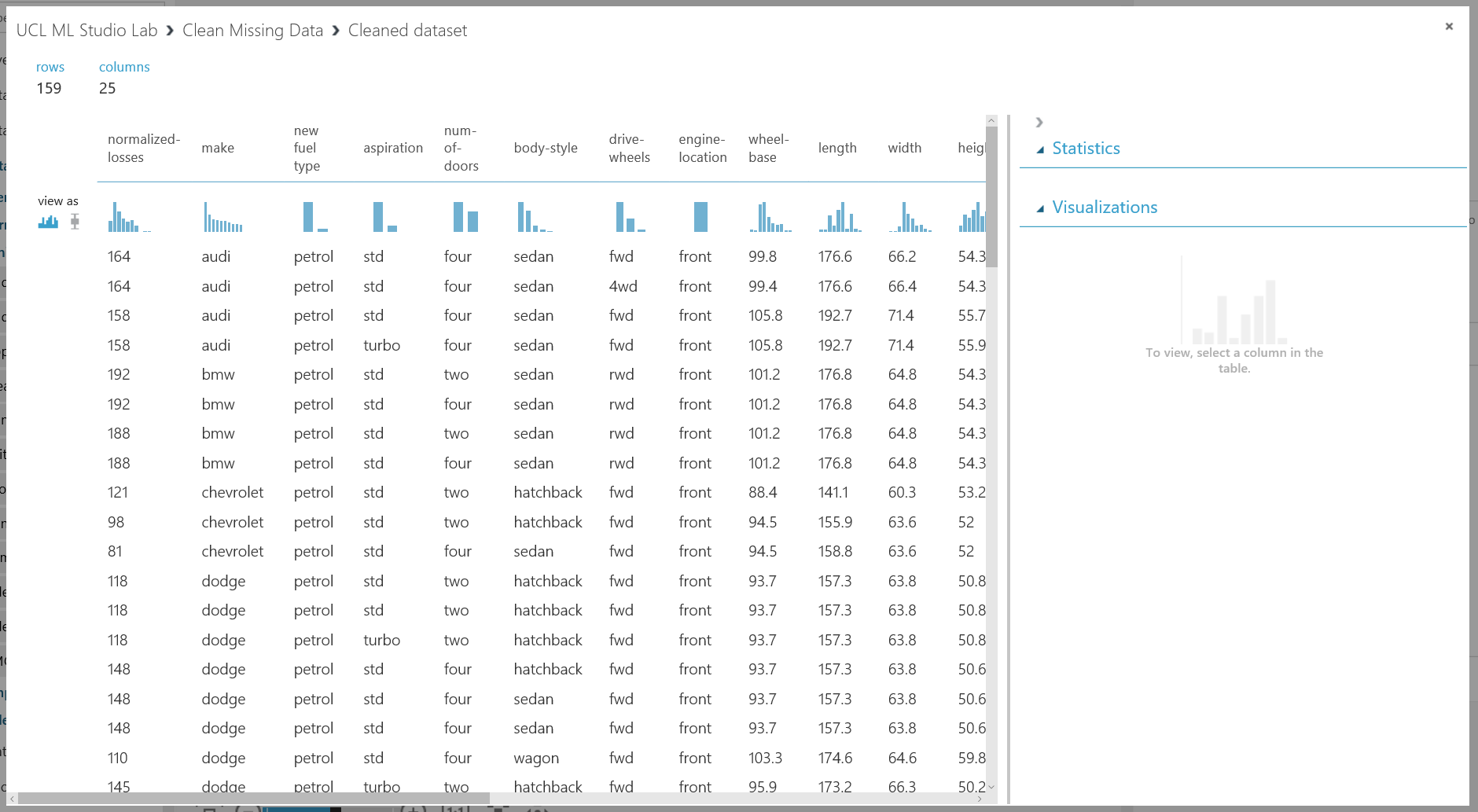
In the properties section on the right, you will see the properties below such as the columns that are selected to operate on as well as what type of cleaning mode will be used



Change the cleaning mode property to ‘Remove entire row’. This will perform a transformation where all columns in the dataset are scanned for missing values, if found, it will remove the row of data. The output of this transformation will be a slightly smaller dataset; however, the dataset records will be full, descriptive data for the machine learning algorithm to learn from.

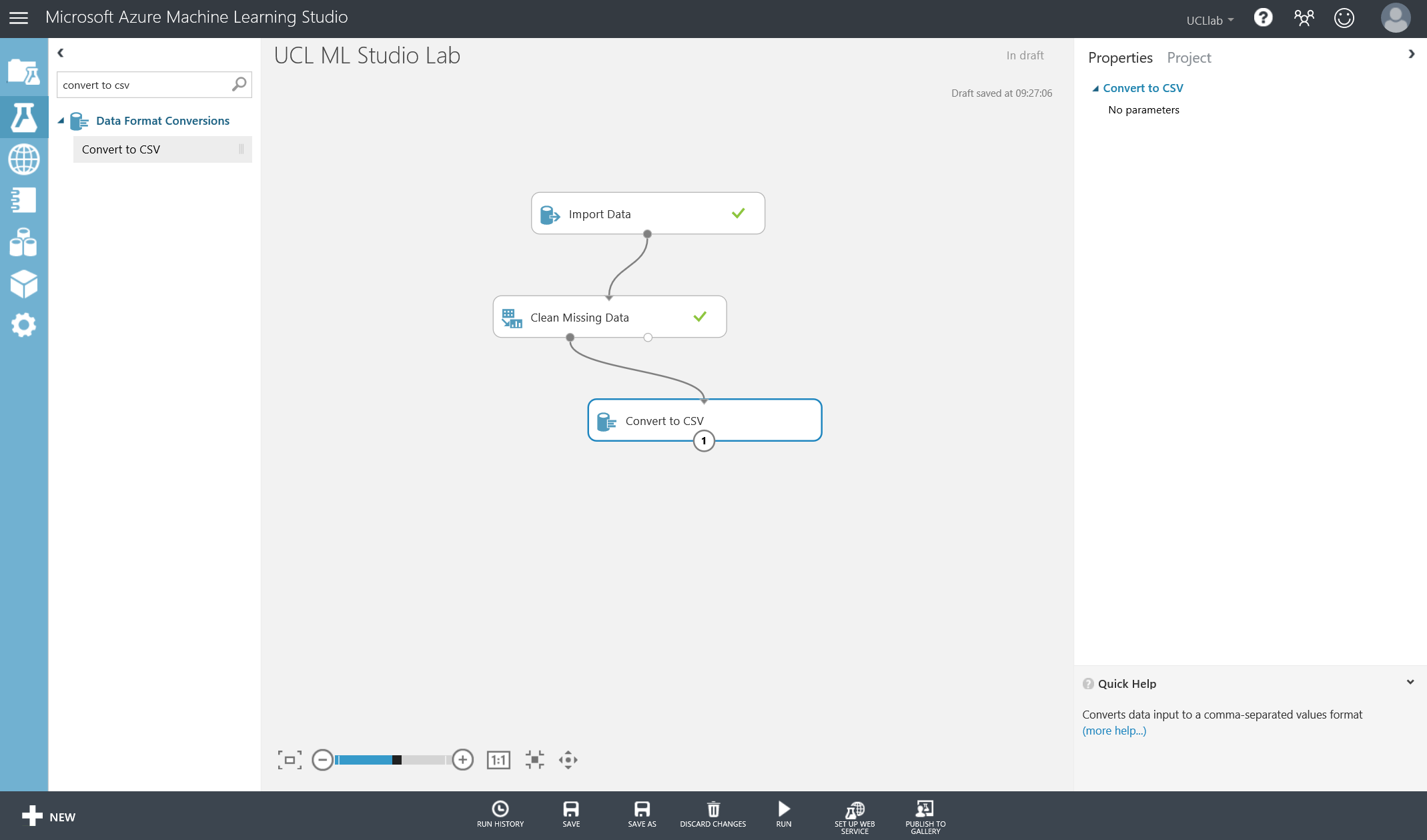
Once properties are set, run the experiment from the bottom toolbar and wait for all modules to have a green tick by them. Once complete choose the left output port of the ‘Clean Missing Data’ module and right click -> visualise.

Note the number of rows in the data has decreased to 159 and the normalised losses columns now doesn’t have sparse gaps.

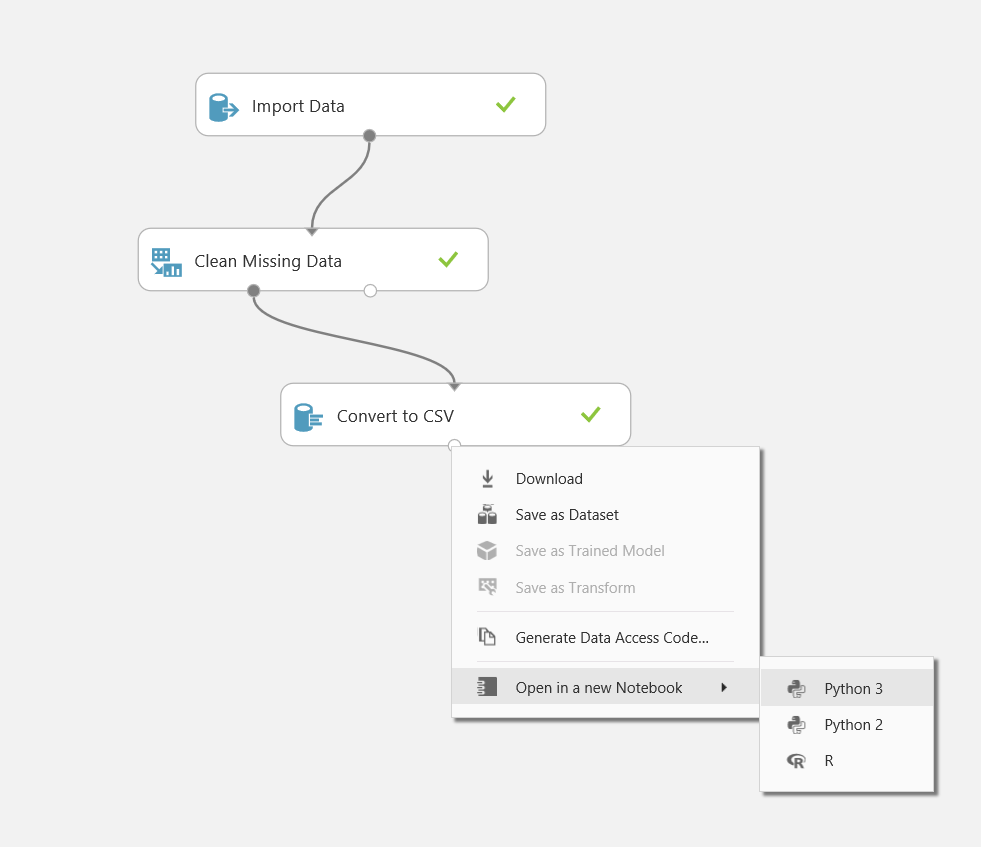


Next we will see how to interact with Jupyter Notebooks in Azure ML from the studio space. This allows us more flexibility to explore the dataset using Python.

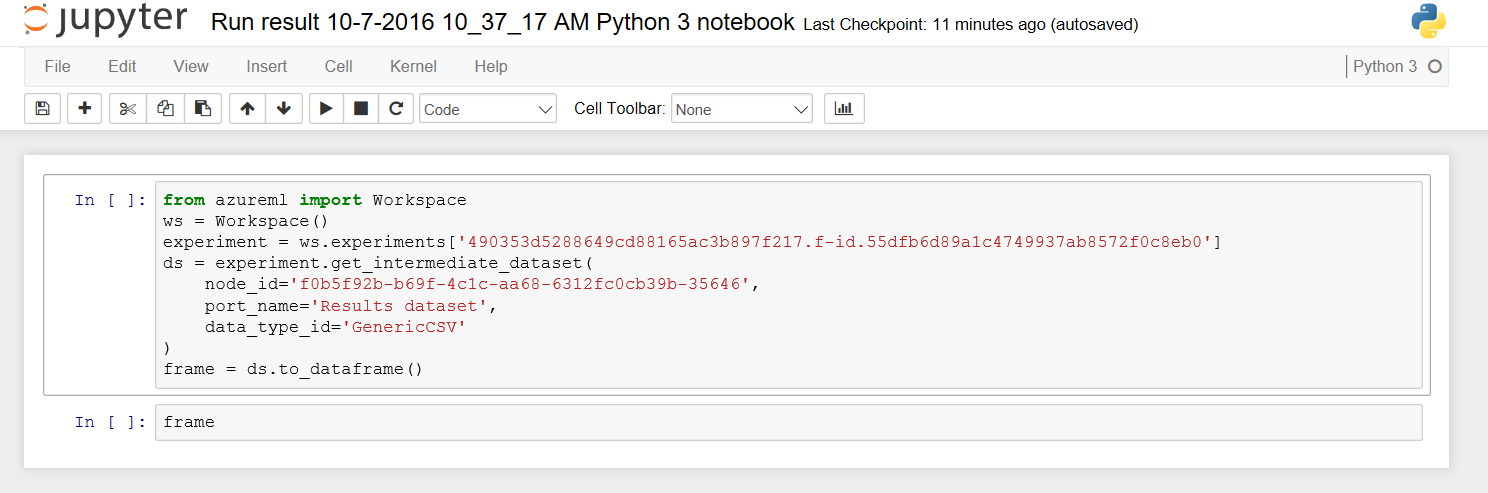
Connect the left output port of the ‘Clean Missing Data’ module to a ‘Convert to CSV’ module. Try and use the search box in the top left of the screen to find this module and connect it up, so it looks like the screenshot below. Once connected, run the experiment again.



To access the Jupyter Notebooks functionality from inside the Azure ML Studio, you must first convert the dataset into a CSV, as we have done above. Now, if you right click on the output port of the ‘Convert to CSV’ file and select ‘Open in a new notebook’ and ‘Python 3’.



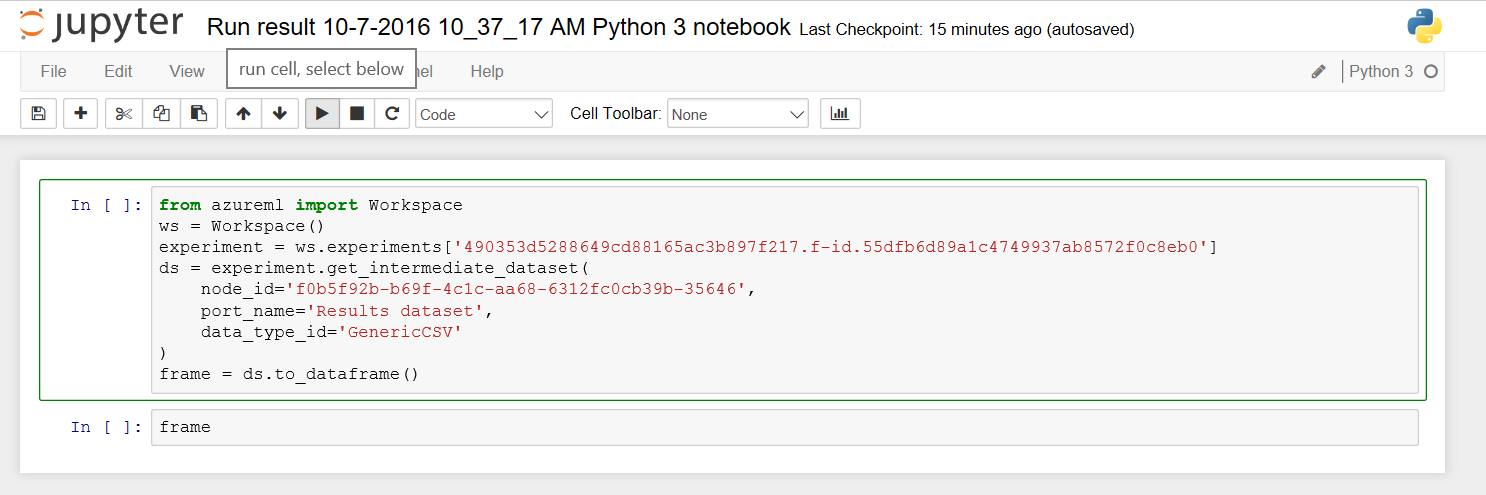
This will then take you to a new tab in the browser, which you might recognise from previous lab sessions, Jupyter Notebooks.



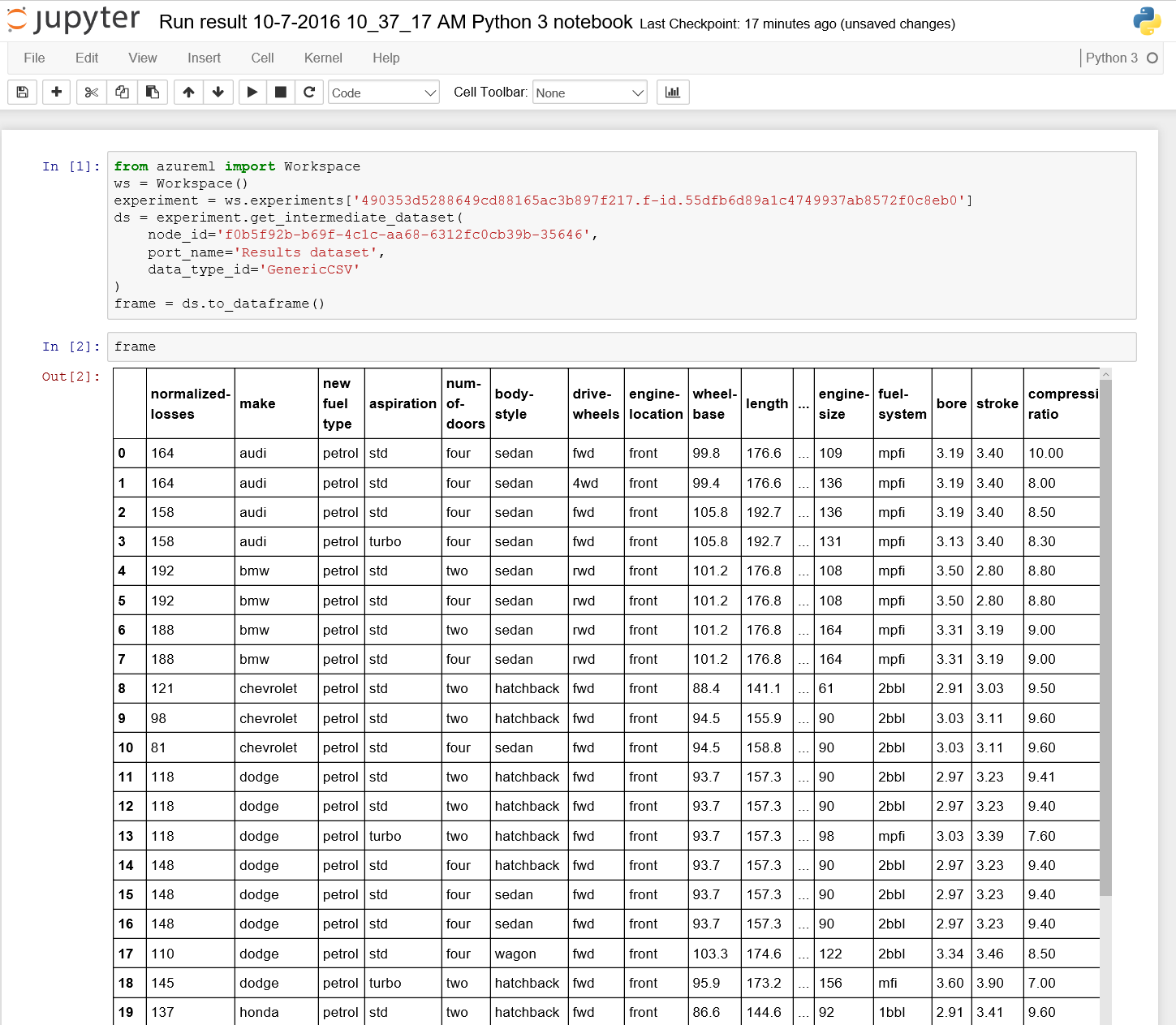
One thing to note, is the use of the ‘azureml’ package in the code snippet above that imports the dataset from the workspace/experiment we were working in, and the node ID (convert to csv) that the dataset is coming from. Now you can interact with the dataset from the car prediction experiment using Python.

In the next step, we will change a couple of columns of data in the data frame using simple python statements. This task is to get you used to the notebooks space and not to challenge your python skills.

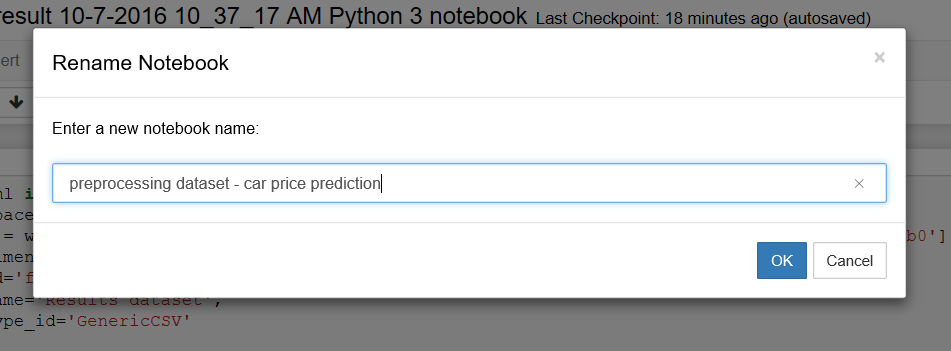
Choose the first block of interactive code and hit Run from the toolbar above. This will pull in the dataset from the experiment into a variable call ‘frame’.



Once complete then run the second interactive code block to see the dataset displayed.



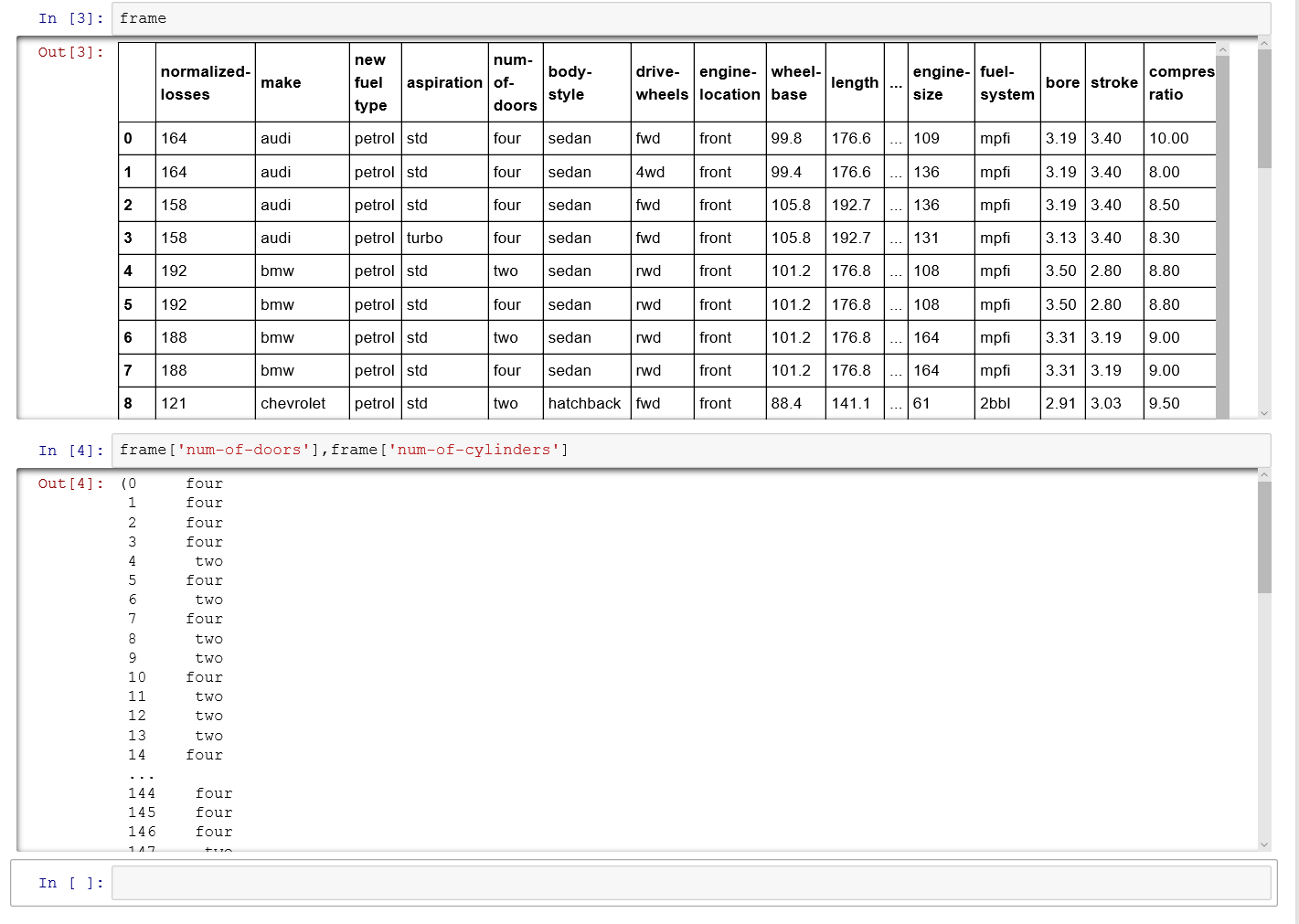
Before we interact further with the dataset, lets rename the notebook to something we recognise. Choose File -> Rename … and choose a filename such as: “preprocessing dataset – car price prediction” and click OK.



In this dataset there are a couple of columns, such as ‘num-of-doors’ and ‘num-of-cylinders’ that are string values such as “two” or “four”, lets change their values to integer types and feed back into the experiment the process to do that.

Lets have a look at the two columns mentioned above by creating a new code cell and running:

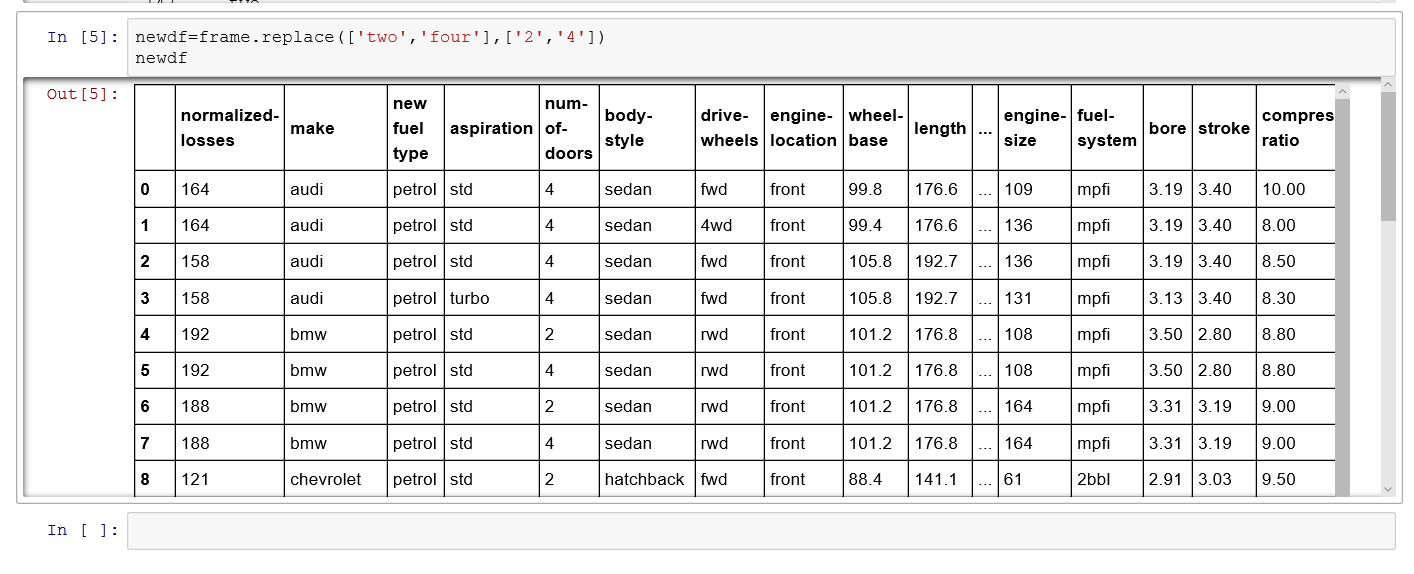
frame['num-of-doors'], frame['num-of-cylinders']



Now let’s take those values in the data frame and swap them for numeric values. We will start with the number of doors example. Create a new code cell and run:

newdf=frame.replace(['two','four'],['2','4'])

newdf



This shows the num-of-doors column now in integer values however the num-of-cylinder column has other values. To change them and check the columns are correct run the piece of python code below:

newdf = frame.replace(['four','six','five','eight','two','three','twelve'],['4','6','5','8','2','3','12'])

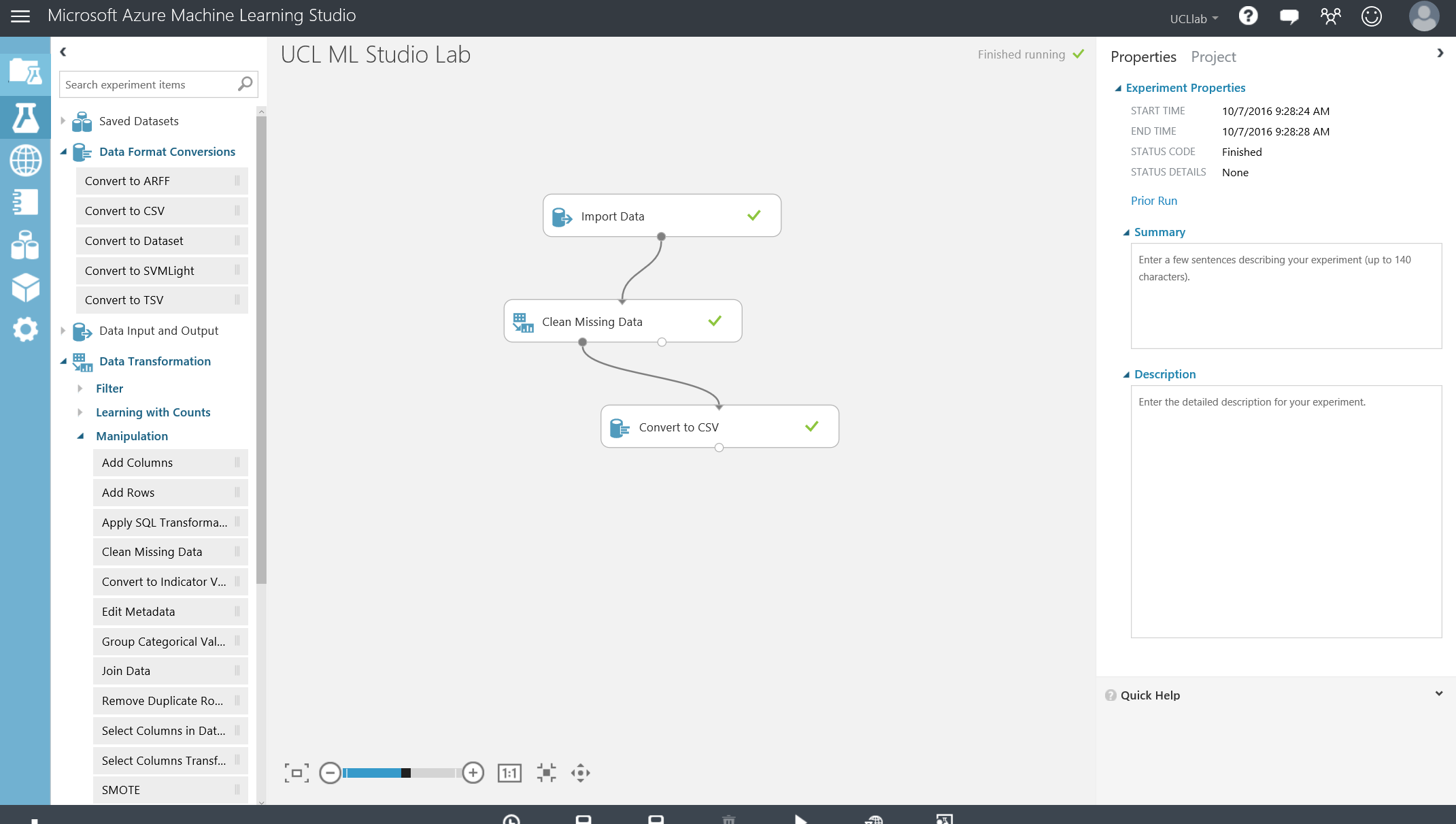
newdf['num-of-cylinders'],newdf['num-of-doors']

Now we have changed the data types and values in the dataset we want to be able to use this edited dataset back in the code, there are a couple of ways to do this:

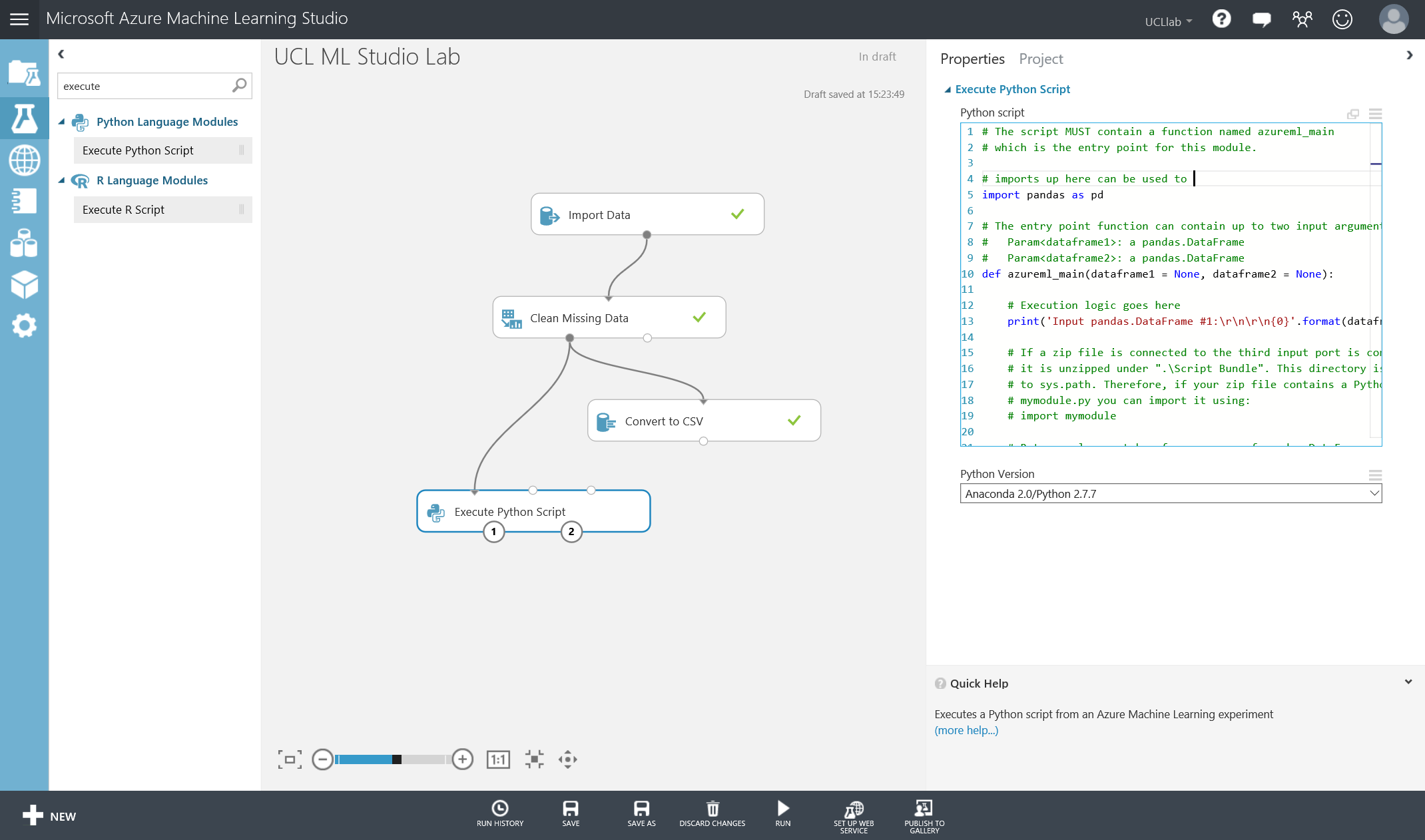
1. If it is short snippets of code (such as above) we can take this code and copy it into an ‘Execute Python’ Module back in the experiment space
2. If this is a more complex and involved function. You can tidy up the notebook to contain the function call and return a data frame. The save the notebook, by using **File -> download as -> Python (.py)** file and upload to execute python module in the .zip input port.

In the lab we will look into option 1, however an extend section at the bottom of this lab is provided to explore the opportunities with option 2.

Make sure you save the notebook and head back to the experiment tab in your browser:



Now let’s edit the dataset using the code we just tested in the Python Notebooks. Search for the module ‘Execute Python Script’ and drag onto the design surface.



Connect the Python module to the left output port of the ‘Clean Missing Data’ module, the last iteration of the dataset within the experiment space. Connect it to the first input port of the python module.

The Execute Python Script module has 3 input ports and 2 output ports. It can take in two datasets and .zip files that could contain python packages you would like to import or python functions you want to call in your script. This module also in output form allows to output the whole data frame to continue your experiment as well as a ‘Python Device’ which lets you see and plots and graphs you may have created in your python code.

*To find out more about this modules capability check out the documentation here:* <https://azure.microsoft.com/en-gb/documentation/articles/machine-learning-execute-python-scripts/>

This module gives you sample code structure where you can add your scripts inside the main function and return a data frame.

Read and copy the code below into the properties, python script box:

# imports up here can be used to

import pandas as pd

# The entry point function can contain up to two input arguments:

# Param<dataframe1>: a pandas.DataFrame

# Param<dataframe2>: a pandas.DataFrame

def azureml\_main(dataframe1 = None, dataframe2 = None):

# Execution logic goes here

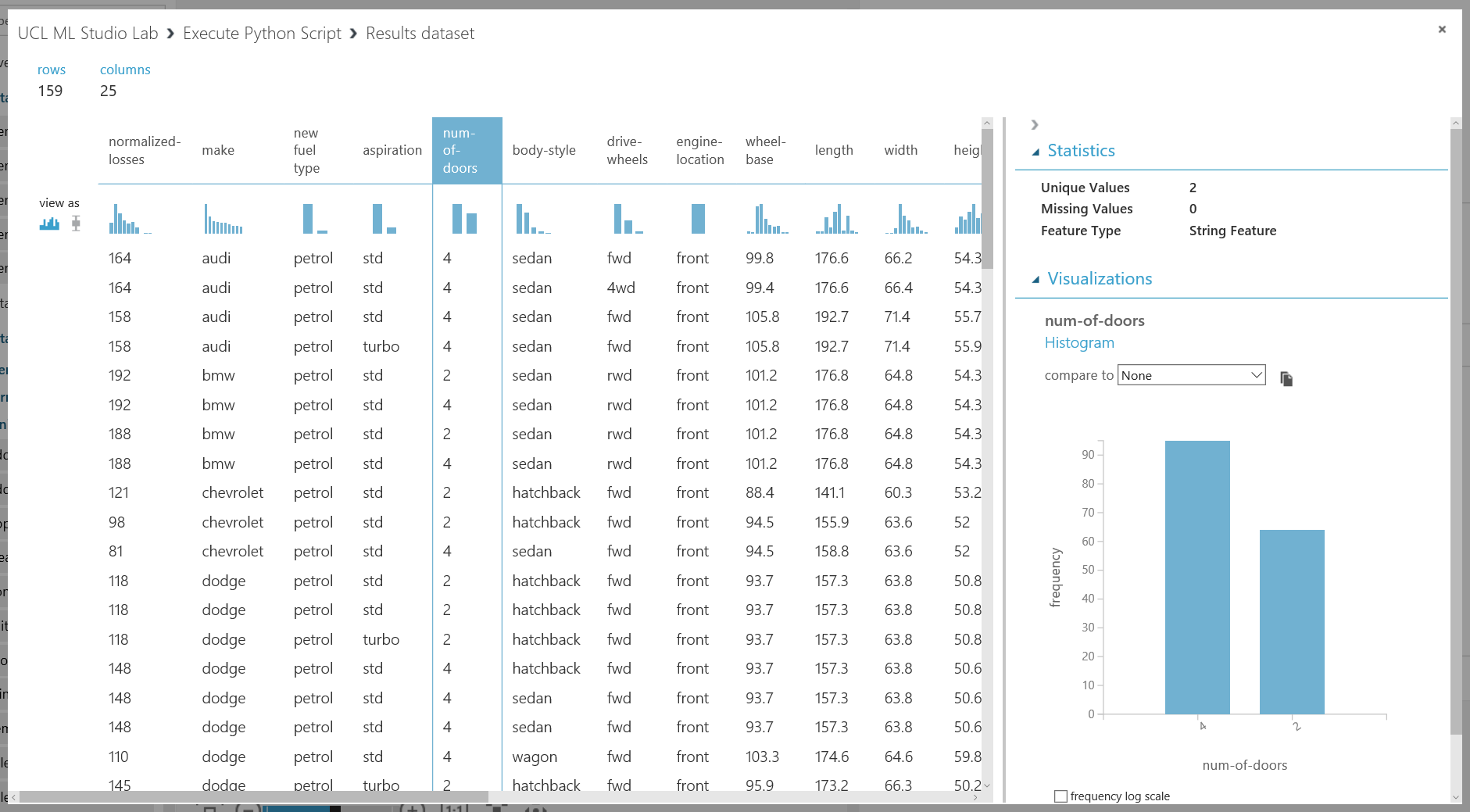
print('Input pandas.DataFrame #1:\r\n\r\n{0}'.format(dataframe1))

newdf = dataframe1.replace(['four','six','five','eight','two','three','twelve'],['4','6','5','8','2','3','12'])

# Return value must be of a sequence of pandas.DataFrame

return newdf,

Once this code is copied in, run the experiment from the bottom toolbar as we have done before. Once completed, right click and visualise the left output port of the ‘Execute Python Script’ module. You should see that ‘num-of-doors’ and ‘num-of-cylinder’ columns have been changed to digits instead of text.

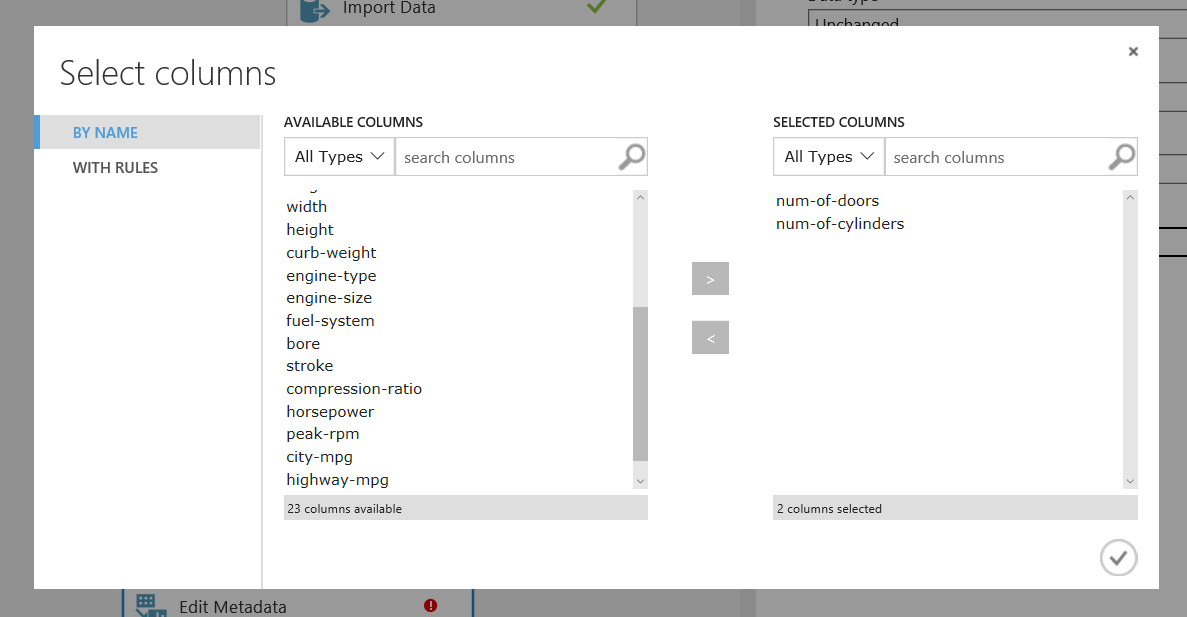


However over on the right of the visualise data pane, the feature type is still a string feature. We need to make sure the studio experiment understands the true data type of this column.

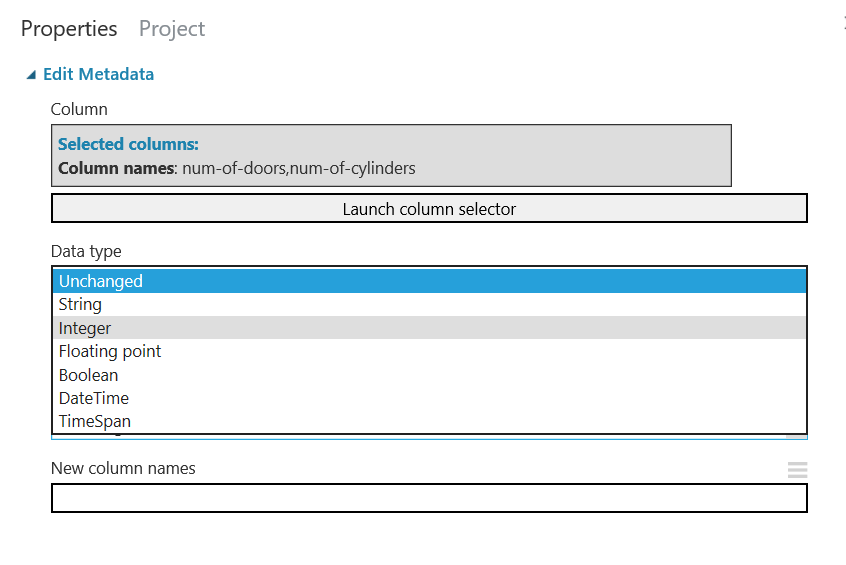
To do that we can use a ‘Edit Metadata’ module, find that module under Data Transformation, Manipulation and click and drag it onto the experiment surface. Connect the left output port (dataset) of the ‘Execute Python Script’ module to the input port of the ‘Edit Metadata’ module



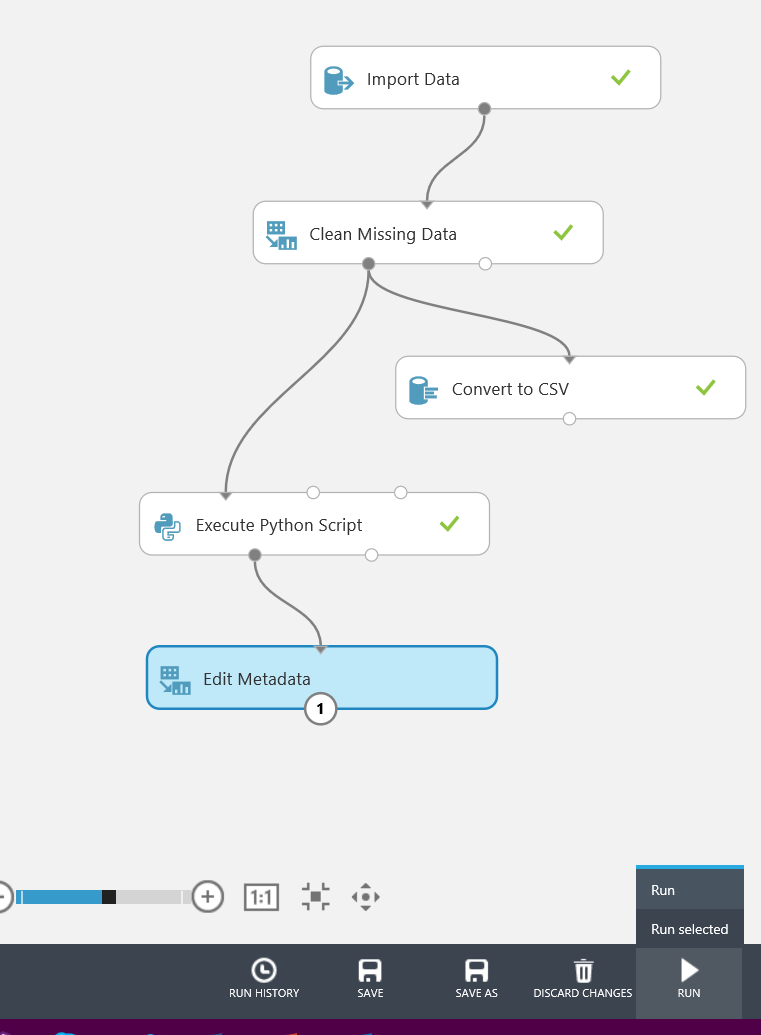
On the properties pane, select launch column selector and choose the ‘num-of-doors’ and ‘num-of-cylinders’ columns as below and click the tick button to complete



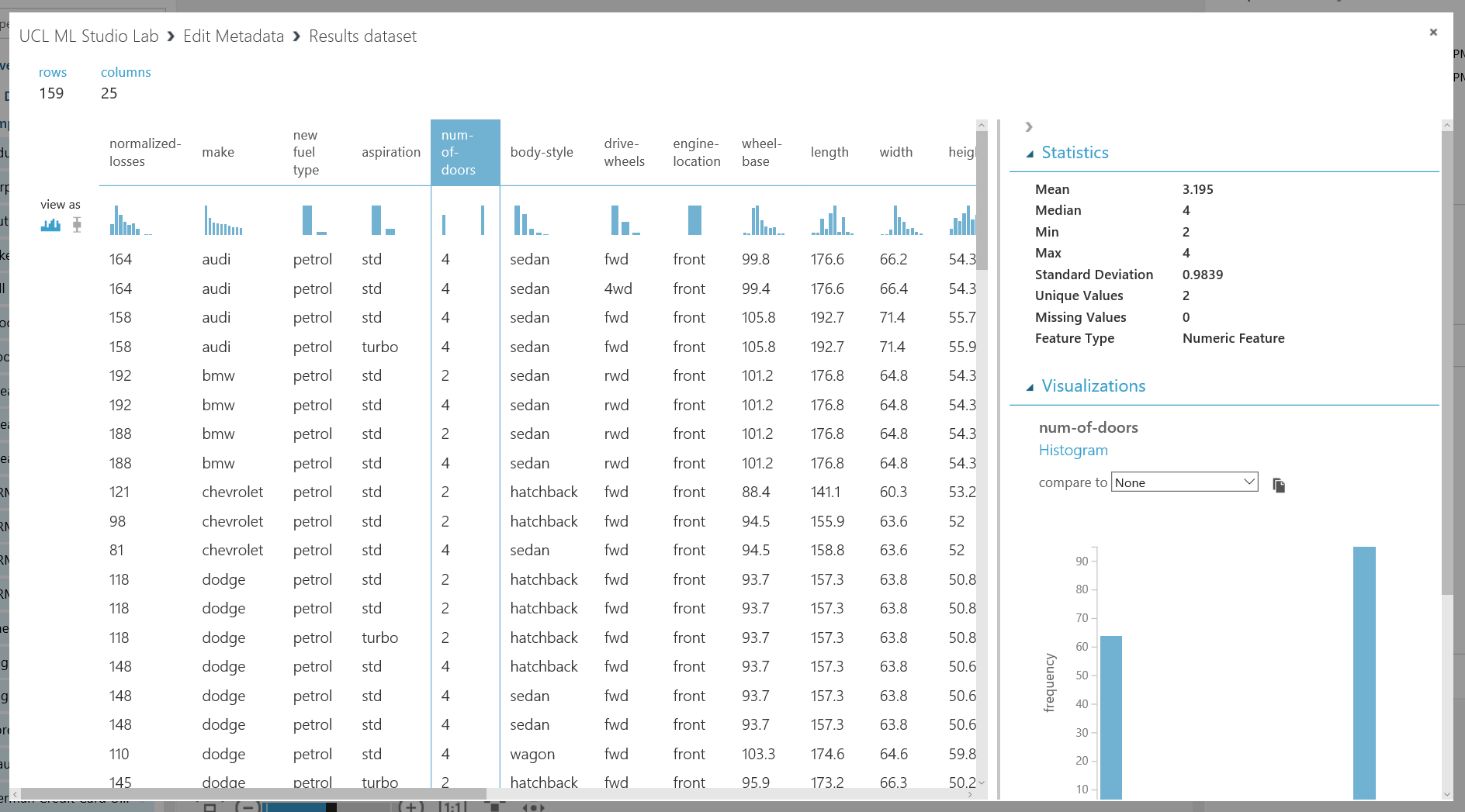
Next change the data type to integer using the drop down available. Then run the experiment.



**Note:** when only running a subsection of the experiment in Azure ML Studio, you can hover over the run button and choose ‘Run Selected’ and the selected modules will light up as below. Try this out now



Once run, visualise the data from the Edit Metadata module and you should see that the columns selected have now been changed to numeric data type:



In this section, we have cleaned missing data from the dataset, edited values in the dataset using a python script and then changed the meta data associated with the columns in the dataset to understand when a column is numeric.

Now we have our dataset in good shape after pre-processing to start training a model in the next section

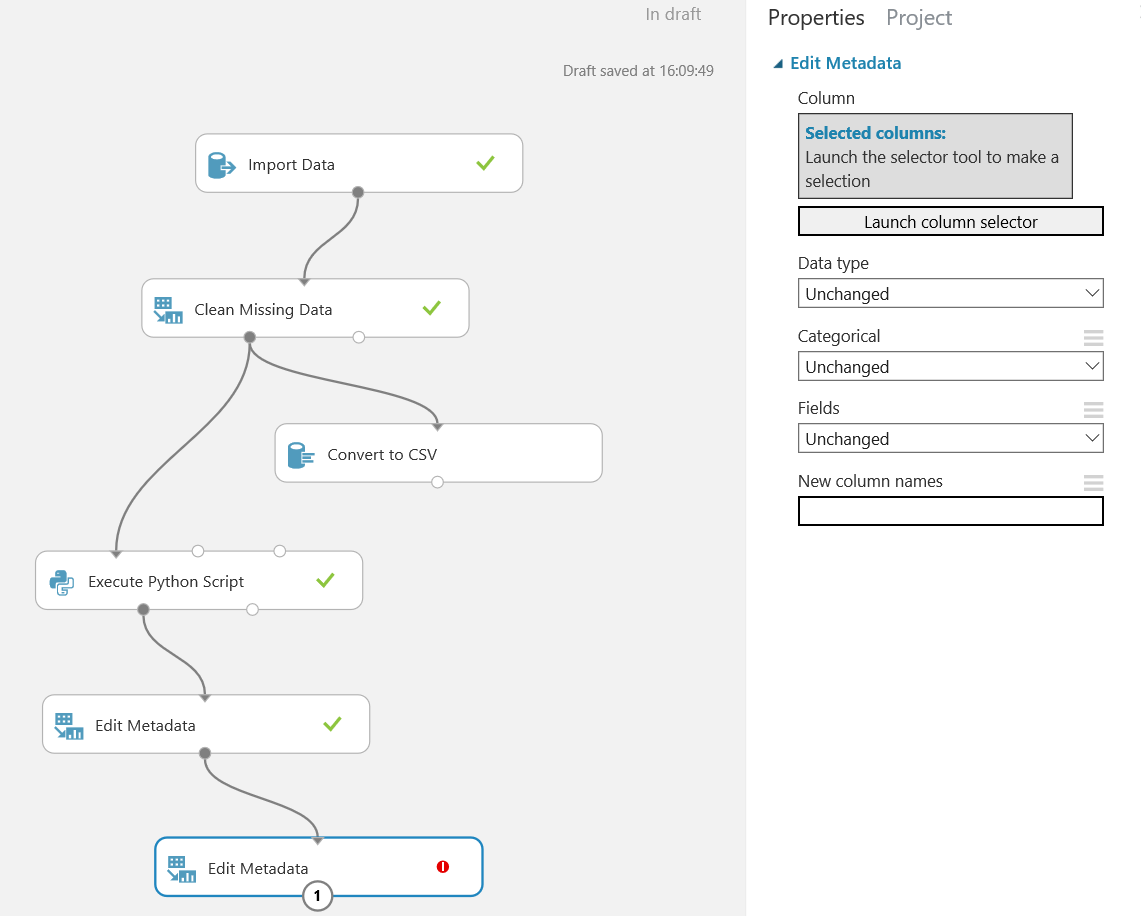
Training the Model

In this section, we will start training a regression model using an algorithm and our pre-processed data to predict the price of a car.

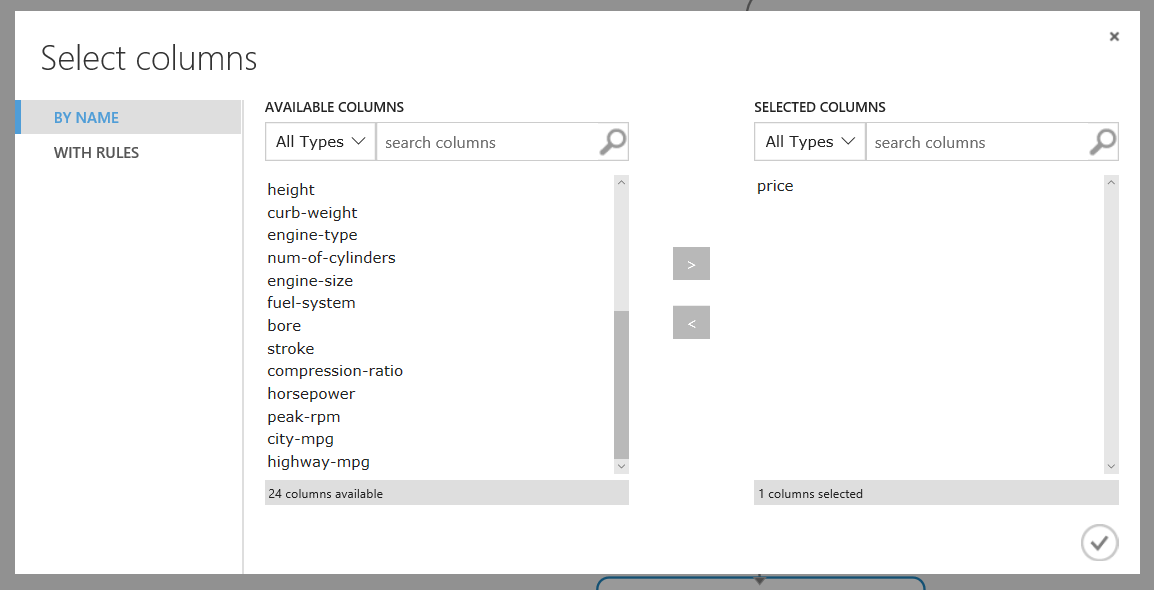
We need to tell the Azure ML studio which columns in our dataset are labels and which columns are features to be operated on.

* **Labels:** Our label is price – as we want to predict the value of a car given all its attributes
* **Features:** The features in our dataset are the rest of the columns, not including the price. Things like make, engine size, MPG, horse power etc.

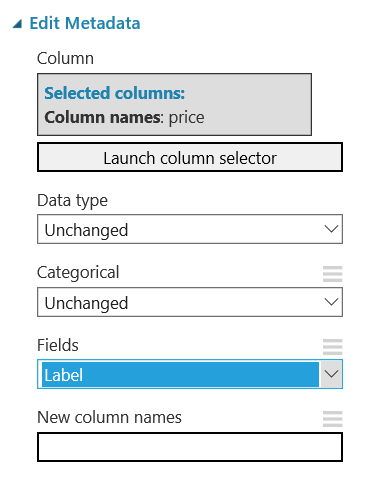
Drag another ‘Edit Metadata’ module onto the experiment space and launch the column selector from the properties pane.



Select the ‘price’ column in the dataset and click the tick:



Next choose the Field property and select ‘Label’ from the drop-down menu

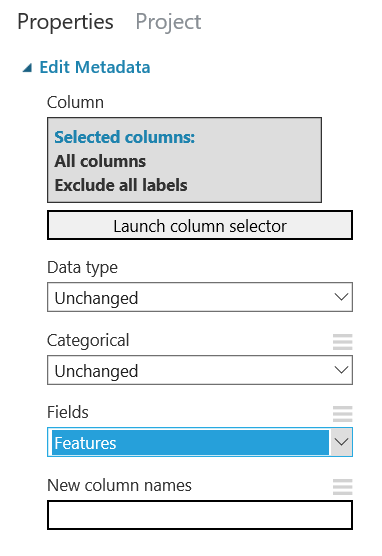
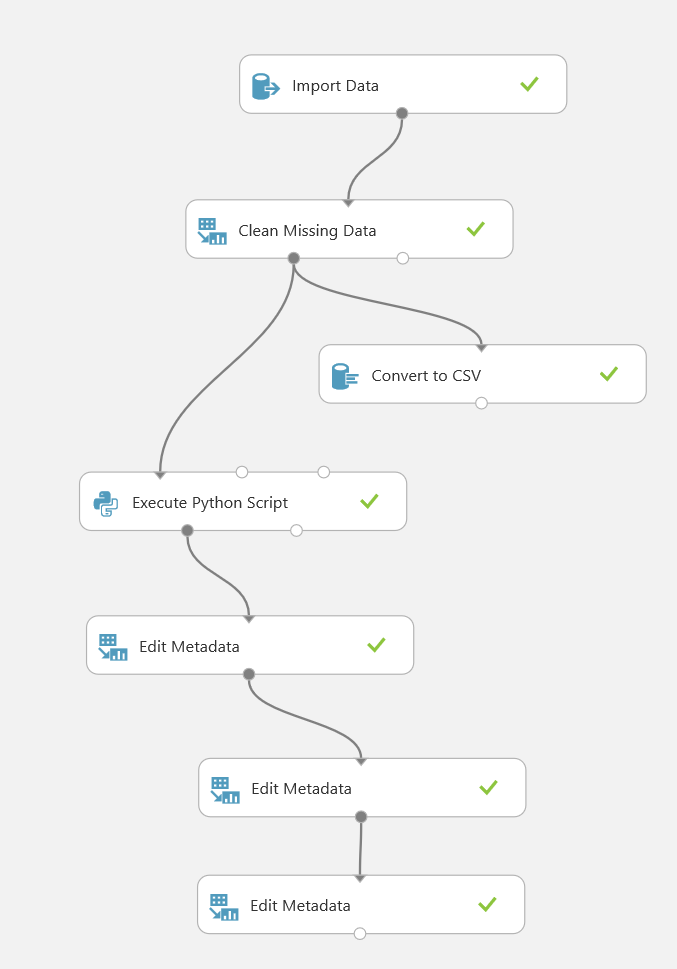


Now let’s do the same process but for features. Select another ‘Edit Metadata’ module and drag onto the experiment surface and connect to the bottom of the previous module. In the properties column open the ‘Launch Column Selector’

In the column selector, select ‘With Rules’ on the left. Then you want to ‘Begin With’ ‘All Columns’ and ‘Exclude’ ‘All Labels’. See the screenshot below for clarification:

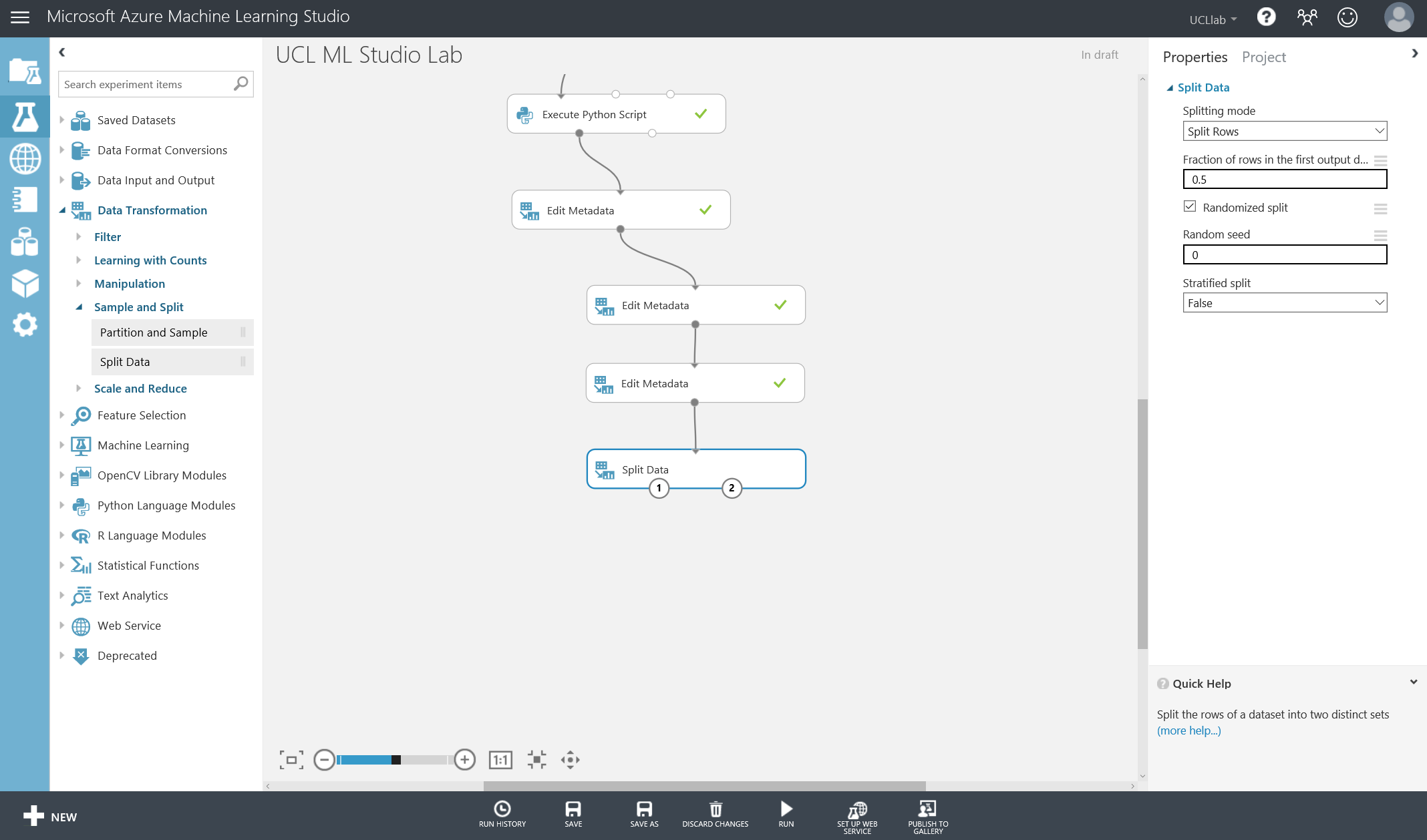


Select the tick button and then back in the properties pane set Field to Features as shown below. And run the experiment:

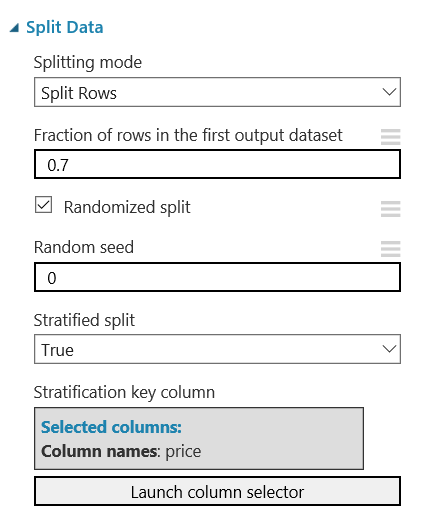


Next we need to split the dataset into training and testing datasets. One dataset will be used to train the model on the patterns and correlations in the data, the other set will be held back and not trained on, to score the accuracy of the model built from the training data.

Find the ‘Split Data’ module under the Data Transformation -> Sample and Split headings in the module list. Drag onto the experiment space and connect to the output port of the last module in the experiment.



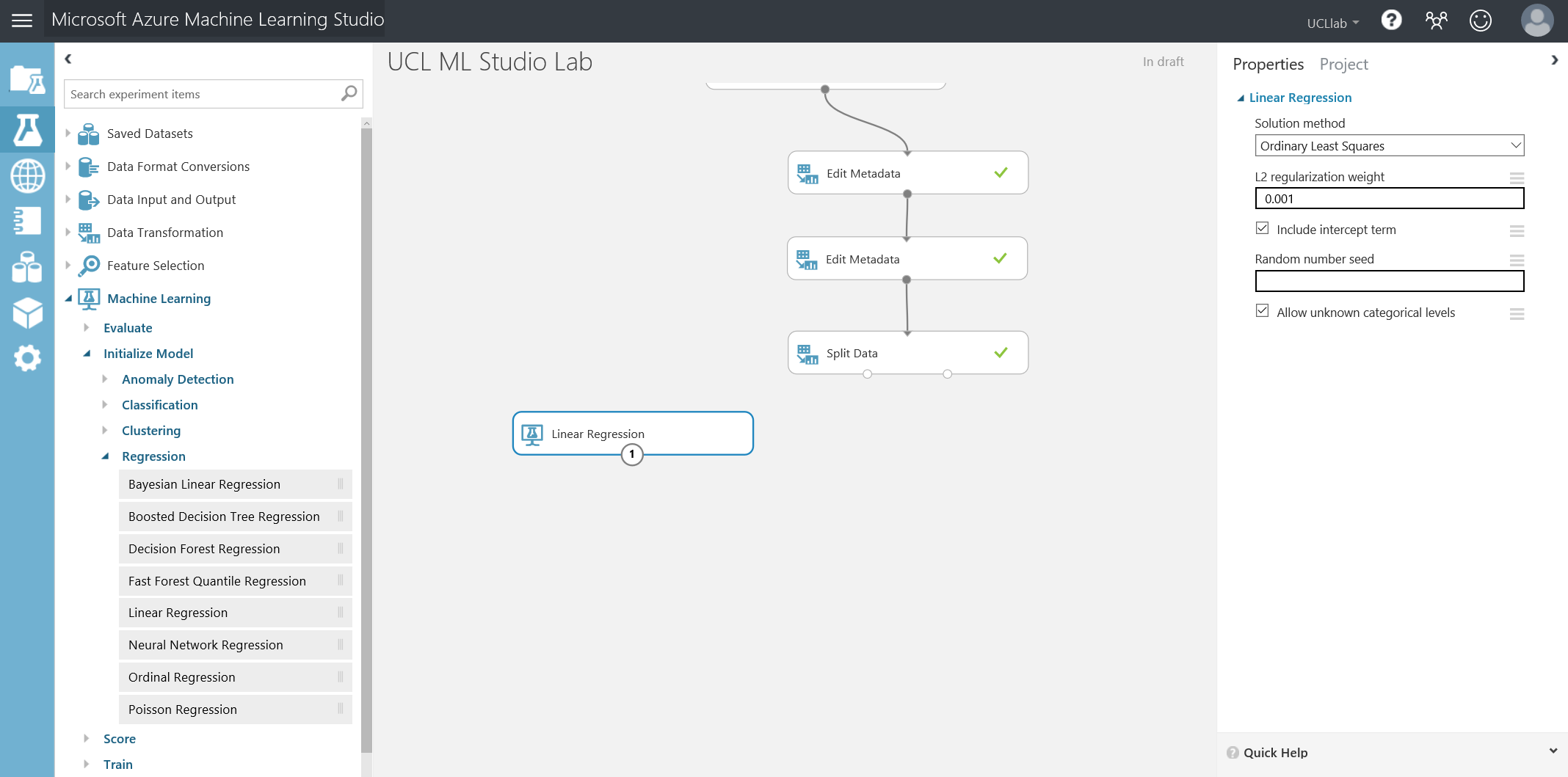
In the properties pane you want to keep the ‘splitting method’ as ‘split rows’. However we want to split the dataset 70% training data and 30% testing data. Therefore in the ‘Fraction of rows in the first output dataset’ put 0.7 for the training dataset. This will automatically mean that 30% of the data will be put into the second output port of the module for the test dataset. You don’t need to set a random seed however we will change ‘Stratified Split’ to True and select the price column in the Launch column selector.



Now select an algorithm to train using your datasets. In the modules on the left find:

Machine Learning -> Initialise Model -> Regression section and find ‘Linear Regression’.

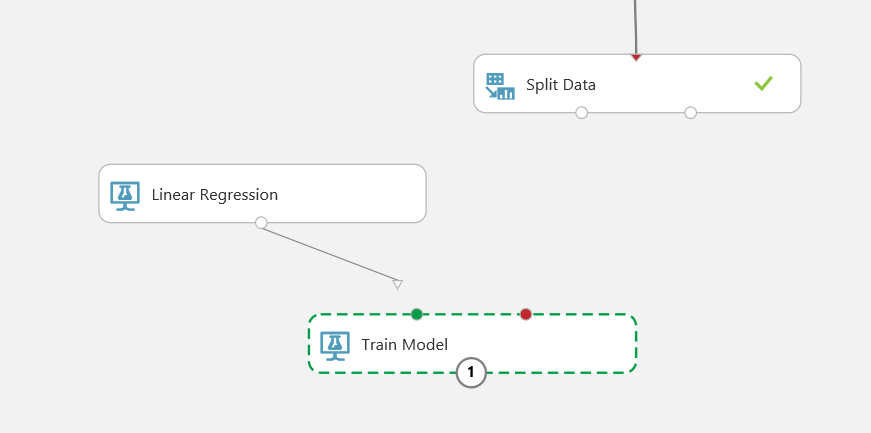
Drag this onto the experiment surface as a pre-built algorithm inside of Azure ML studio.



Each one of the pre-built algorithms provided in Azure Machine Learning Studio allows you to tweak the parameters associated. In every case default parameters are supplied for each algorithm so that you can quickly build out experiments. We will keep the parameters of the Linear Regression algorithm as default for now.

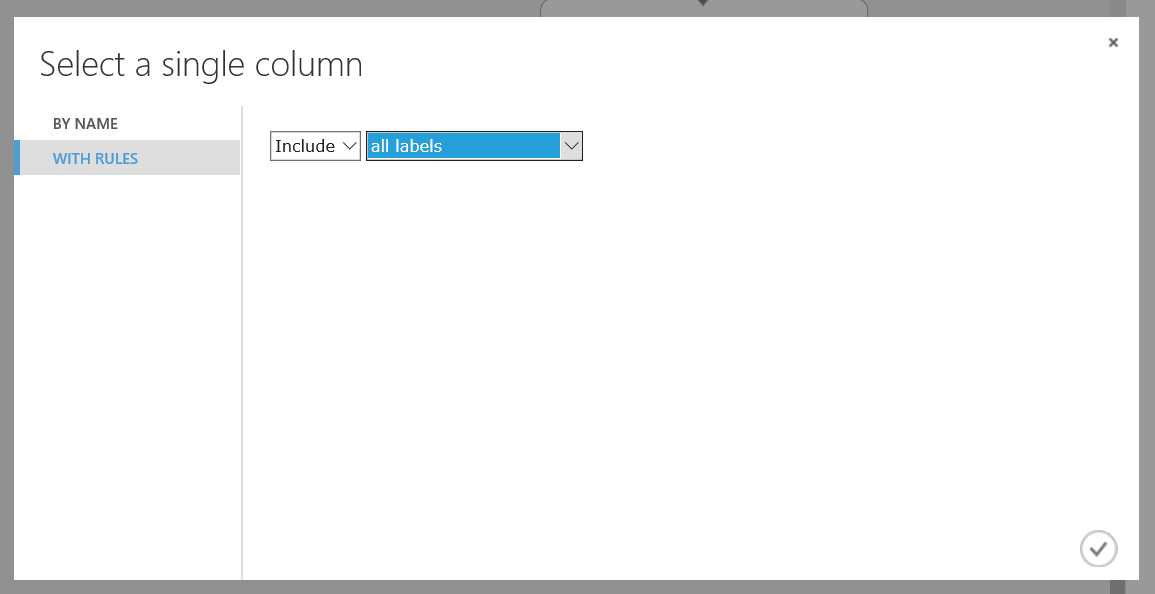
To combine the algorithm and our dataset we need to bring on a ‘Train Model’ module. This takes an untrained algorithm and dataset as inputs and produces a trained model out of the module output port.

Connect the linear regression algorithm to the left input port of the Train Model module, then connect your 70% training dataset (left output port of Split Data) to the right input port of the Train Model module. You won’t be able to wire these up the wrong way, as the model gives you guidance on the input type its expects as seen in the screenshots below:

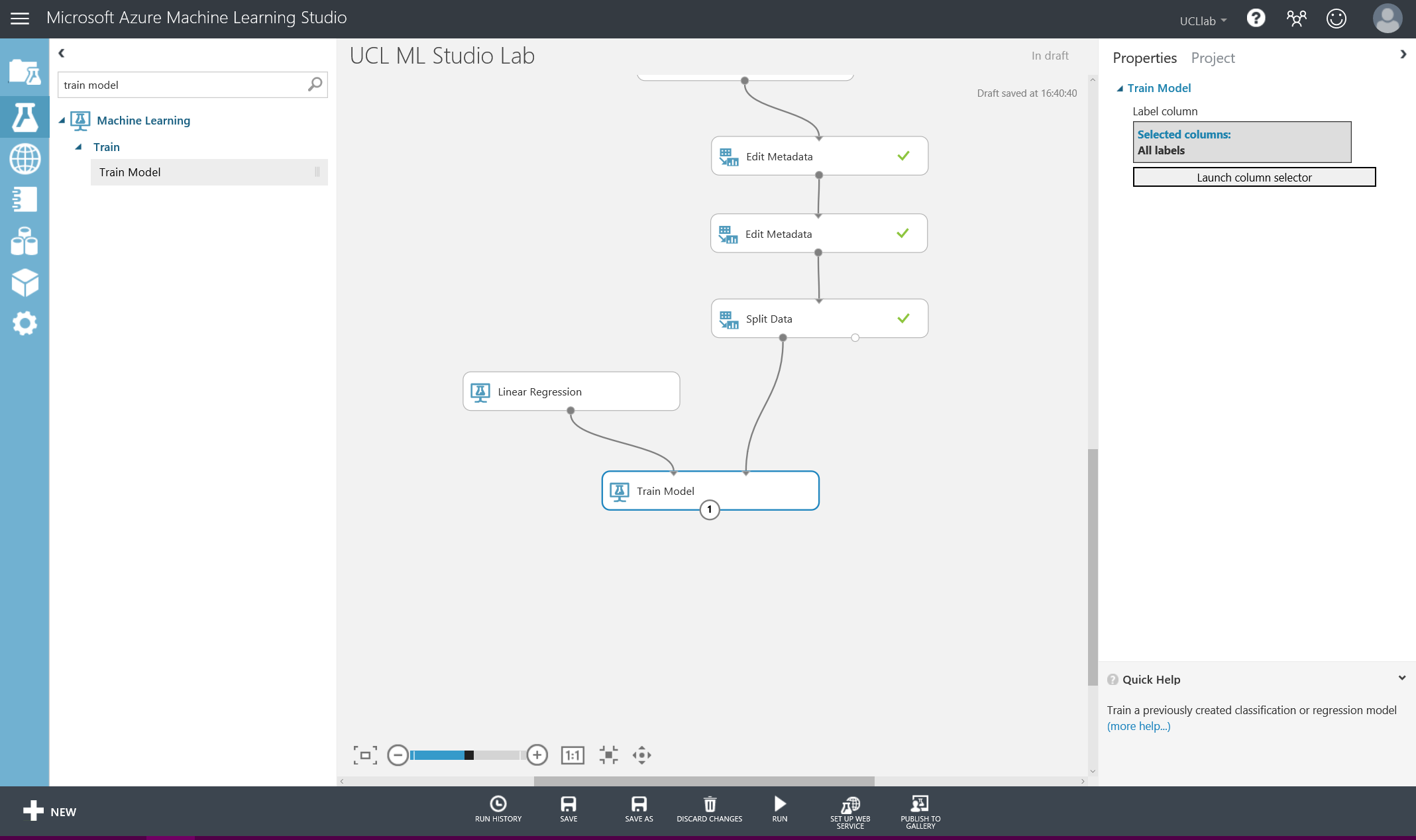


Once connected up, select the Train Model module and in the properties pane, Launch the Column Selector. Choose the ‘With Rules’ section on the left and select option ‘Include’ with ‘All Labels’.

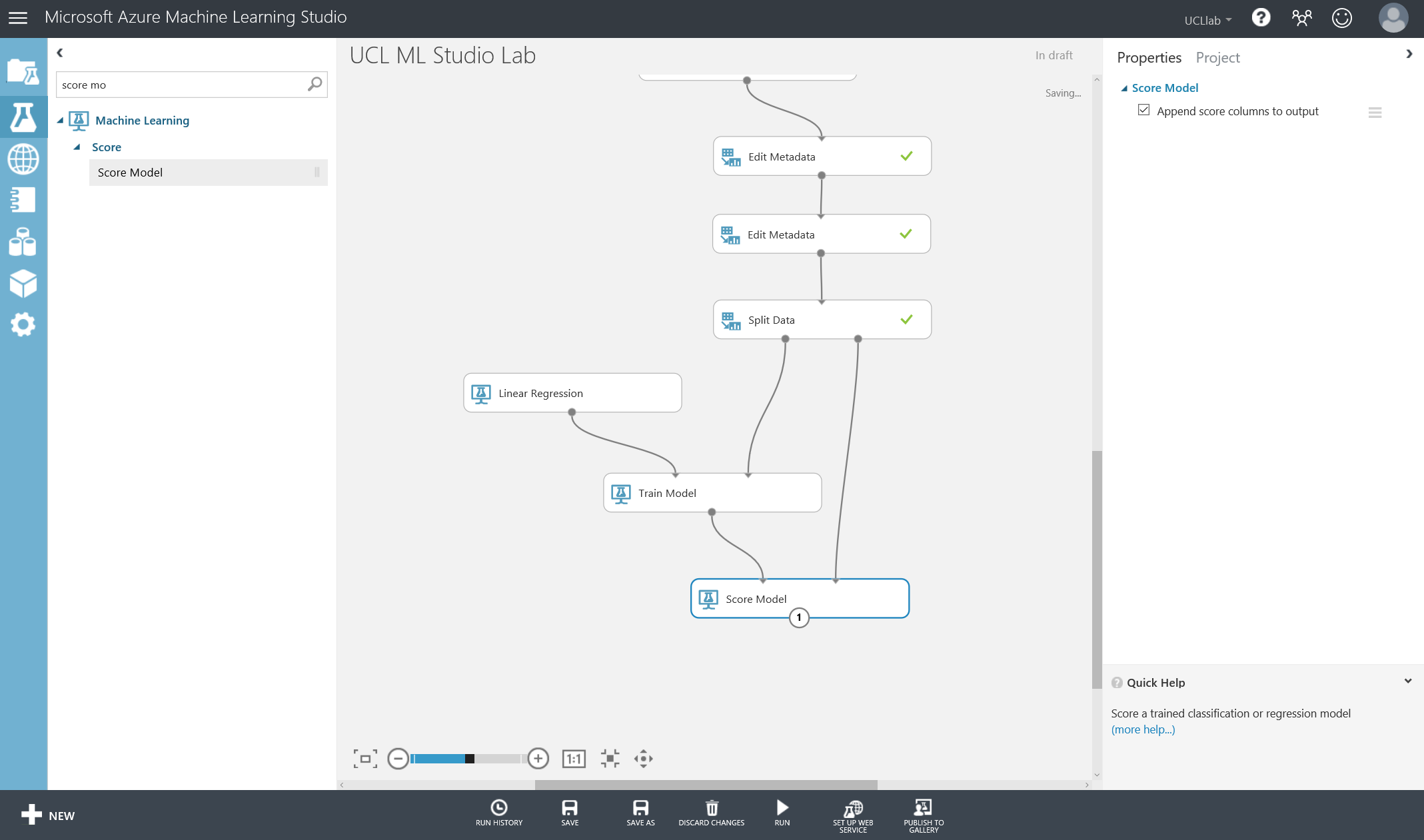
This will allow the train model module to know it needs to focus on learning the price of the car (label) given the attributes and patterns in the dataset (features)



Select the tick box



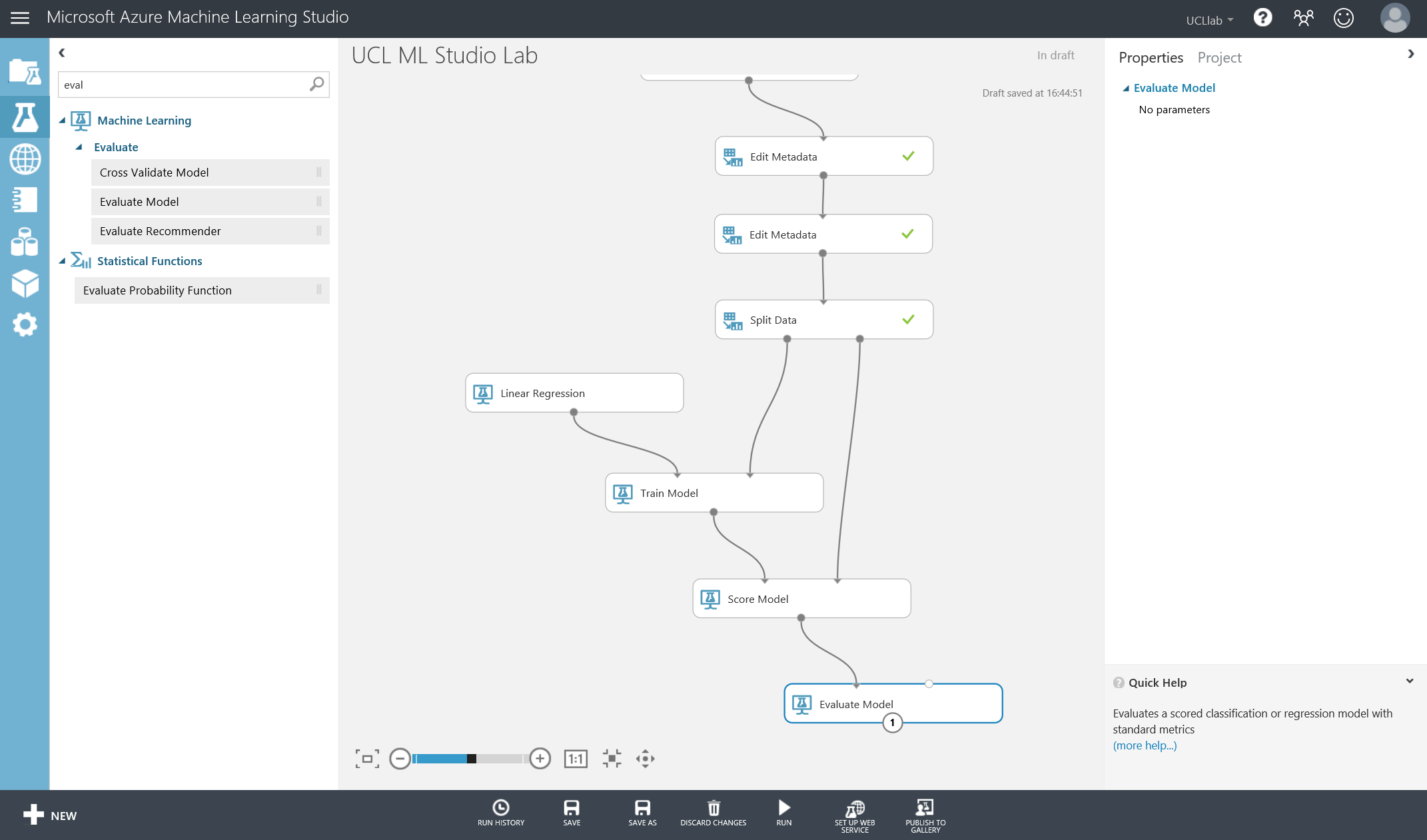
Once the model is trained (output port of the Train Model module) we need to score the model on how well it can predict the price of a car. This is where we connect the trained model to a ‘Score Model’ module and the testing dataset (the right output port of the Split Data module).



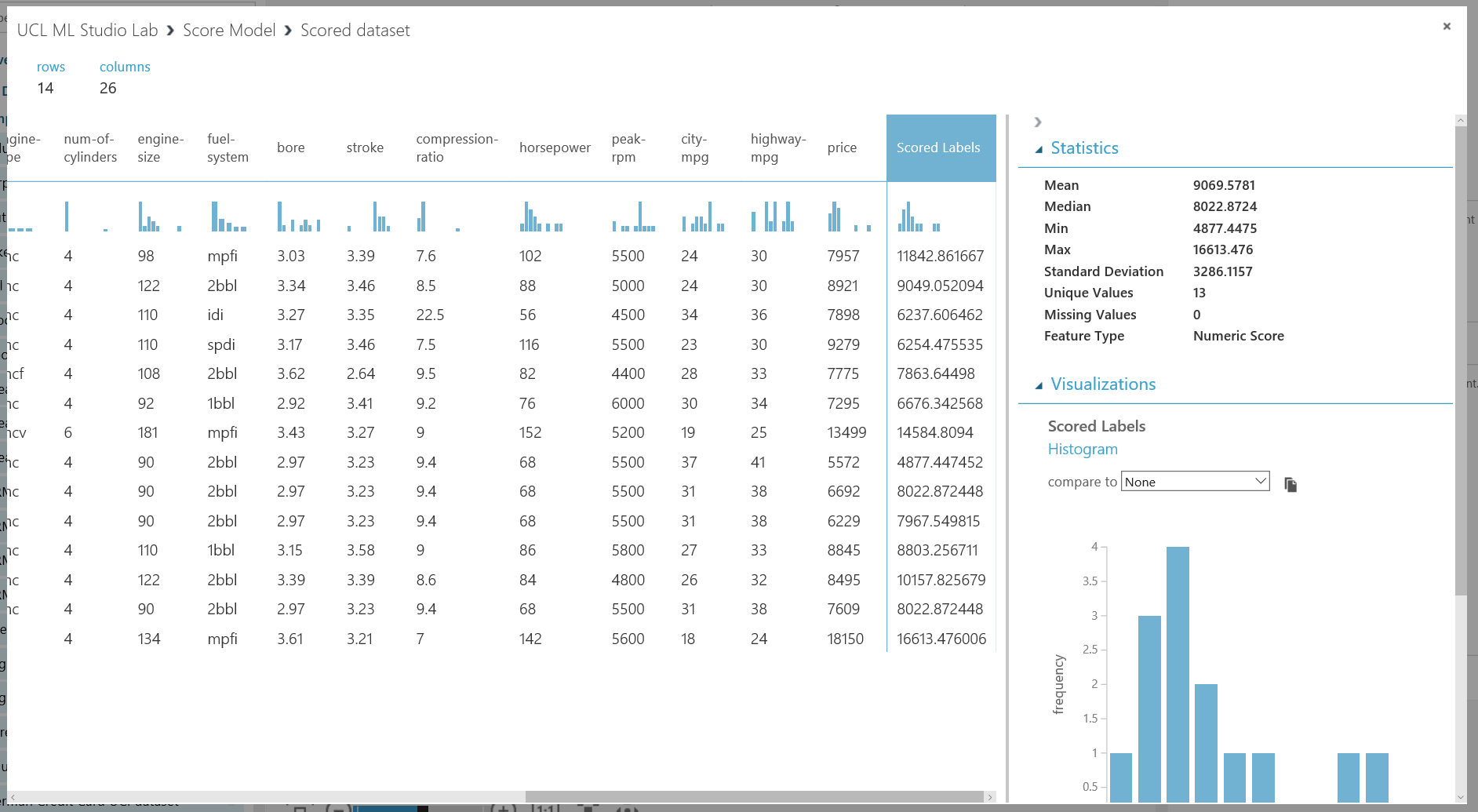
There are no parameters to set on this module as it takes the testing dataset and queries the trained model using that data, it asks it to predict the price of each car in that testing dataset. Inside the testing data we have the actual price of the car still, therefore we can start to measure the error value of the regression model – how far is the predicted value of the car price compared to the actual price of the car.

Finally, in order to view some simple statistics on this model connect an ‘Evaluate Model’ module to the output of the ‘Score Model’ module.

Notice the ‘Evaluate Model’ module has two input ports. At this point connect the score model module to the left port of the evaluate model module as shown below and Run the experiment.



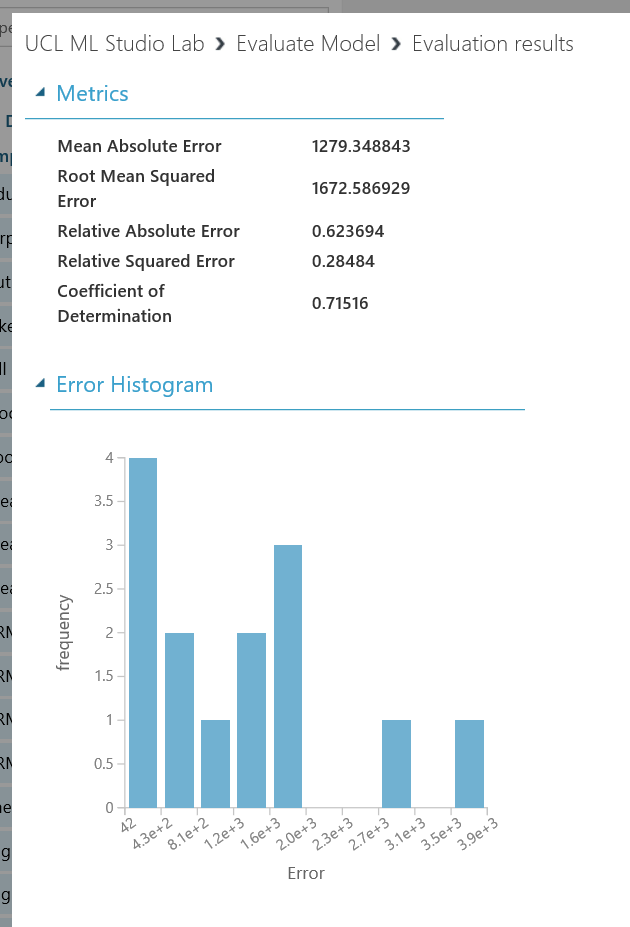
Once run, let’s look at the outcome of the model. Select the output of the Score Model module and right click -> visualise the data. Scroll the end of the dataset and find the Scored Labels column. This is what the trained regression model has predicted compared to the actual value column next to it, ‘price’.



Now if we look at the output of the Evaluate Model module, this will provide us with more statistics relating to Scored Labels. Select Evaluate Model module and right click -> visualise

The metrics returned for regression models are generally designed to estimate the amount of error. A model is considered to fit the data well if the difference between observed and predicted values is small. However, looking at the pattern of the residuals (the difference between any one predicted point and its corresponding actual value) can tell you a lot about potential bias in the model.

The following metrics are reported for evaluating regression models. All metrics are reported but the models are ranked by the metric you select for evaluation.

**Mean absolute error (MAE)** measures how close the predictions are to the actual outcomes; thus, a lower score is better.

**Root mean squared error (RMSE)** creates a single value that summarizes the error in the model. By squaring the difference, the metric disregards the difference between over-prediction and under-prediction.

**Relative absolute error (RAE)** is the relative absolute difference between expected and actual values; relative because the mean difference is divided by the arithmetic mean.

**Relative squared error (RSE)** similarly normalizes the total squared error of the predicted values by dividing by the total squared error of the actual values.

**Coefficient of determination**, often referred to as R2, represents the predictive power of the model as a value between 0 and 1. Zero means the model is random (explains nothing); 1 means there is a perfect fit. However, caution should be used in interpreting R2 values, as low values can be entirely normal and high values can be suspect.