aka.ms/ImperialML

Machine Learning

What is Azure Machine Learning?

Contents

The Problem Domain

First Time Setup

The Data

Data Pre-Processing

Training the Model

Optimising Classifiers

Creating a Scoring Experiment

Publish as a Web Service

Testing the Web Service

Publish to the Gallery

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).

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 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 in line in experiments and explore the data interactively with Jupyter Notebooks.

One of the most important features of Azure ML is its publishing service whereby a finished (trained) experiment can be exposed as a web API that can be consumed by any other application such as a website, 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 the incidence of diabetes in the Pima Indian tribe from Arizona in the USA. We’ll also see how Python scripts and Jupyter Notebooks can be used to aid in the exploration and evaluation of the data. We’ll see how to publish the model by reviewing the sample code for a published experiment and testing the output from an Excel add-in.

The Problem Domain

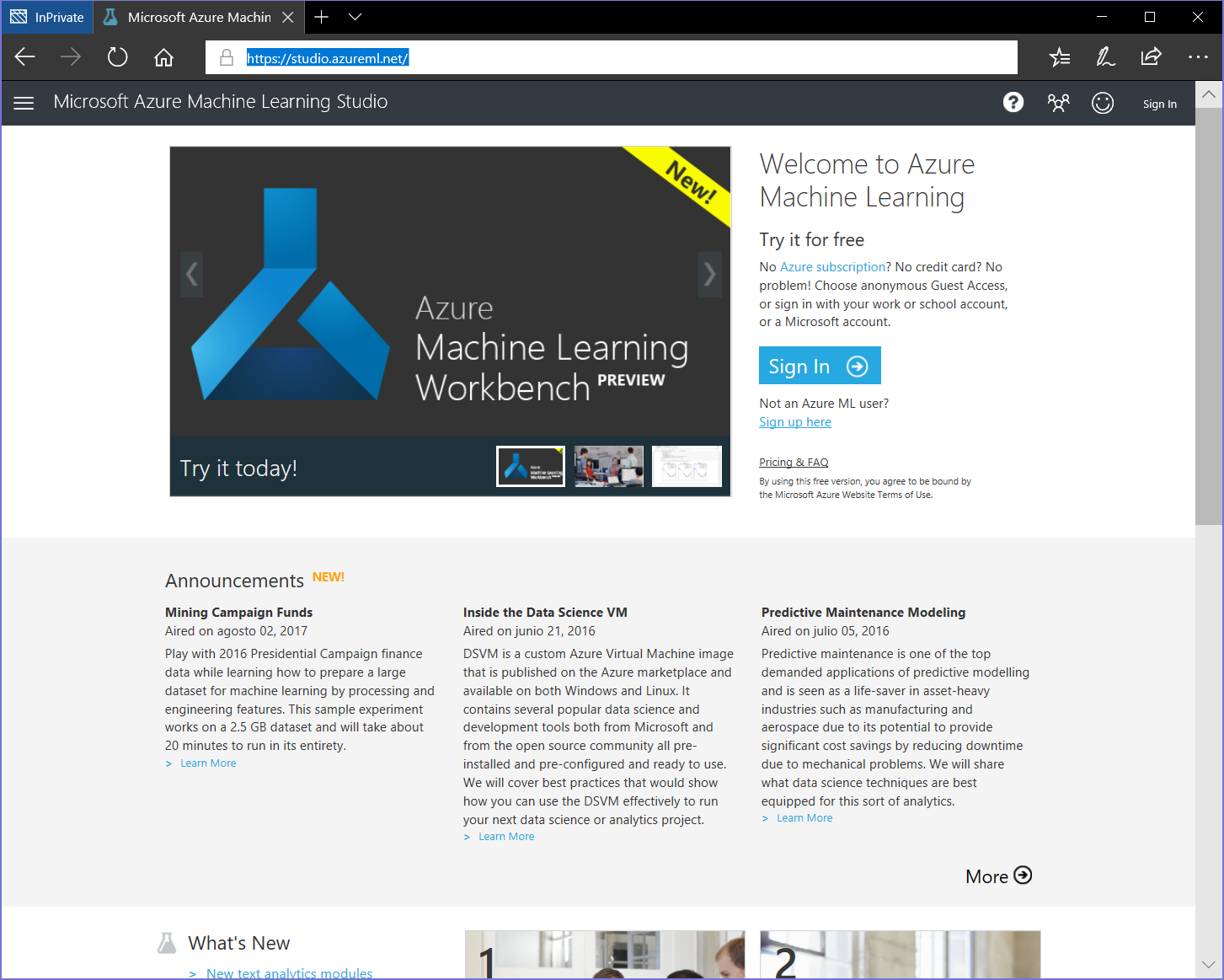
During this lab, we will create and publish a model that predicts how likely a Pima Indian is to develop diabetes. Pima Indians have been found to have a high incidence of diabetes and this has been extensively [researched](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2603306/). There is also a publicly available data set of 768 members of this tribe which has various features included and whether or not they are diabetic:

* Number of times pregnant
* Plasma glucose concentration - a 2 hours in an oral glucose tolerance test
* Diastolic blood pressure (mm Hg)
* Triceps skin fold thickness (mm)
* 2-Hour serum insulin (mu U/ml)
* Body mass index (weight in kg/(height in m)^2)
* Diabetes pedigree function
* Age (years)
* Class variable (0 or 1) to indicate whether they are diabetic or not

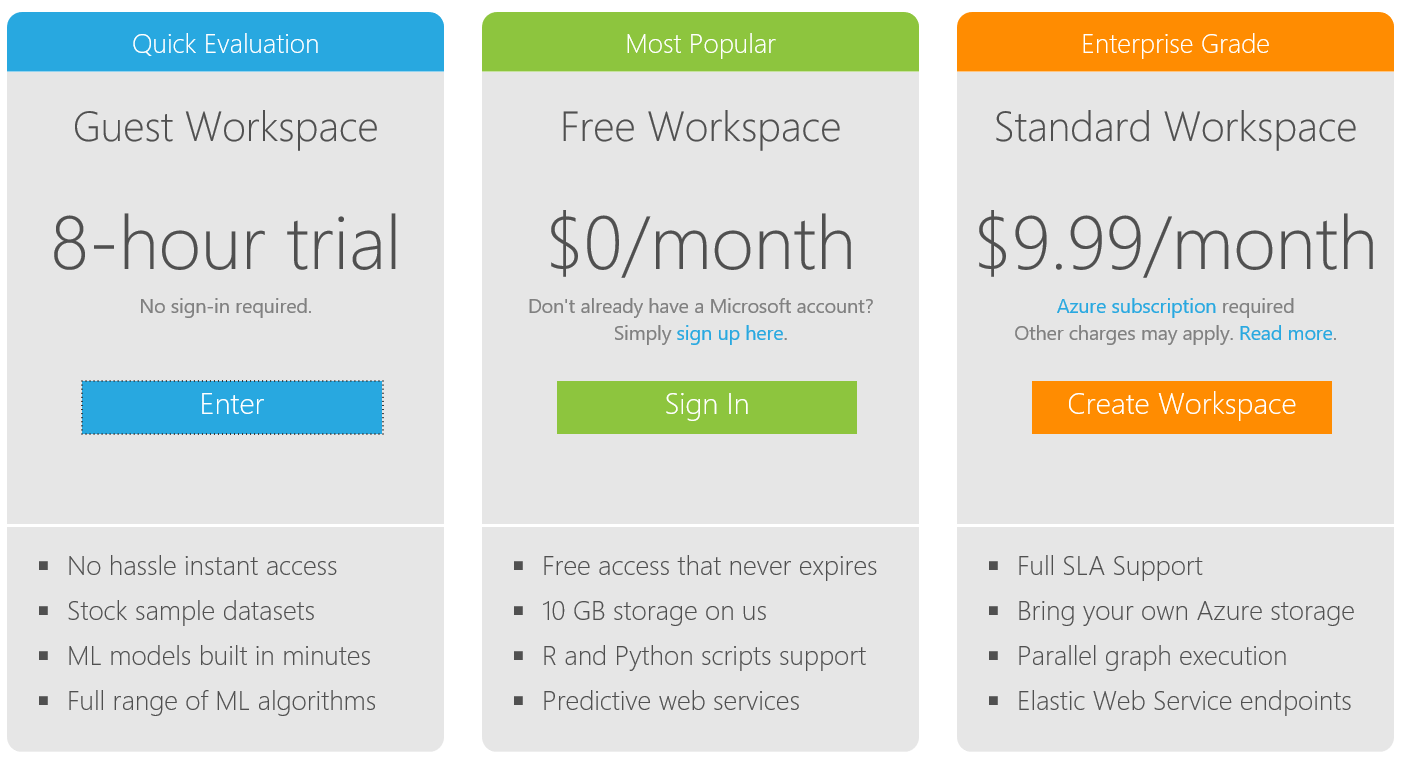
The model we’ll create is a form of supervised learning, which means the data we’ll use to build the model has the outcome or label included in the data. This model will use a binary (aka two class) classification technique by *classifying* the Pima Indians into two groups, those with diabetes and those who don’t have the disease.

First Time Setup

Microsoft Azure Machine Learning offers a free-tier service and a standard tier for which you need an Azure subscription. In order to access the free workspace, go to <https://studio.azureml.net/>, click on the sign up here link and sign in with a Microsoft account (@hotmail.com, @outlook.com etc.) and a free workspace is automatically created for you.



Microsoft Azure Machine Learning offers an 8 hour anonymous trial, a free-tier service and a standard tier (for which you need an Azure subscription). In order to access the free workspace, go to <https://studio.azureml.net/>



Select the Free workspace option and sign in with a Microsoft account (@hotmail.com, @outlook.com etc.) and a free workspace is automatically created for you.

If you don’t have a Microsoft account, create one here: <https://signup.live.com/signup.aspx>

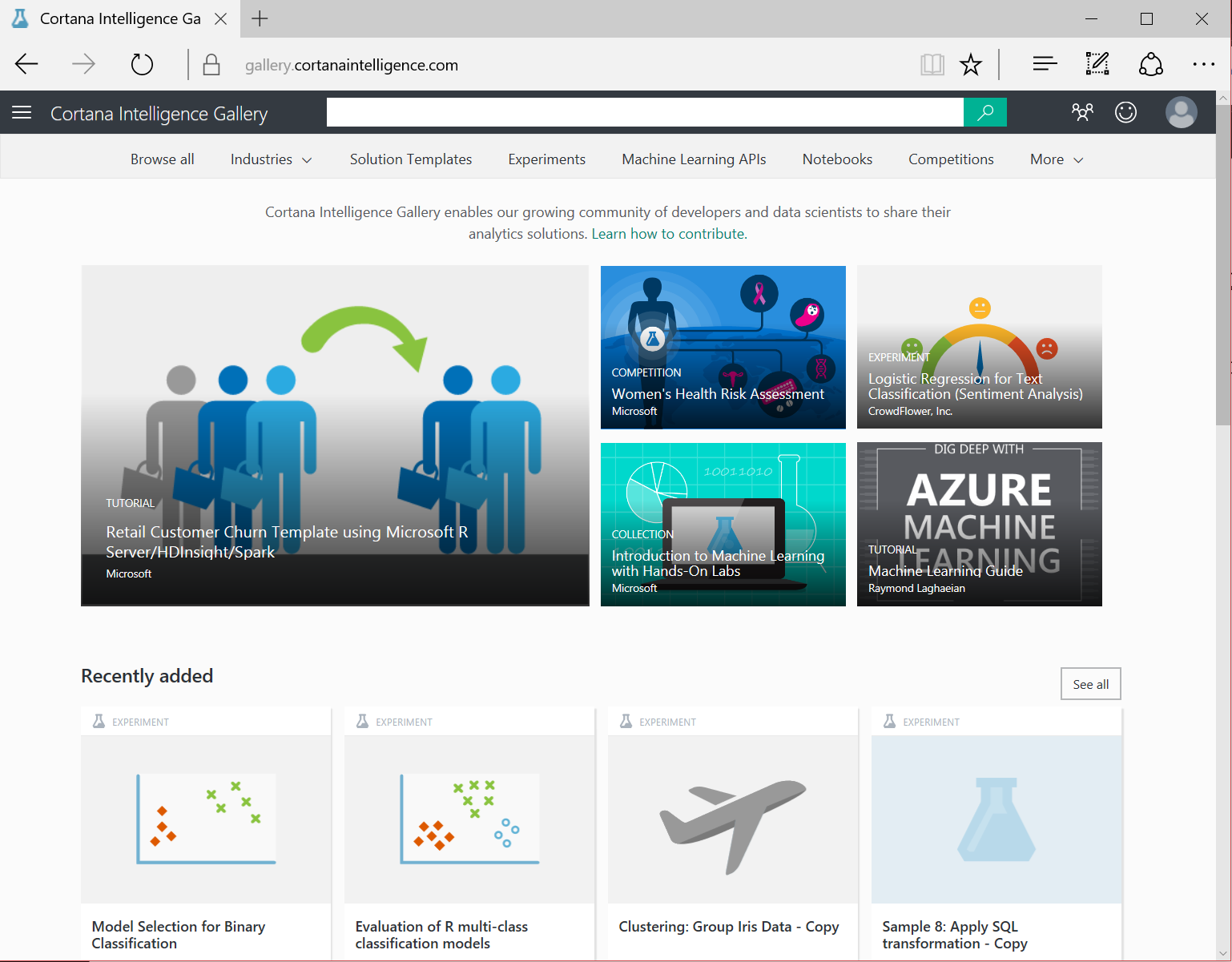
Once on the Workspace homepage click on the link ‘Sign-in to ML Studio’ and you should land on an experiments page. From there find the ‘New’ button in the bottom left corner and choose ‘Experiment’ then ‘Blank Experiment’. Notice there are sample experiments you can also use, edit and build on top of.

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:

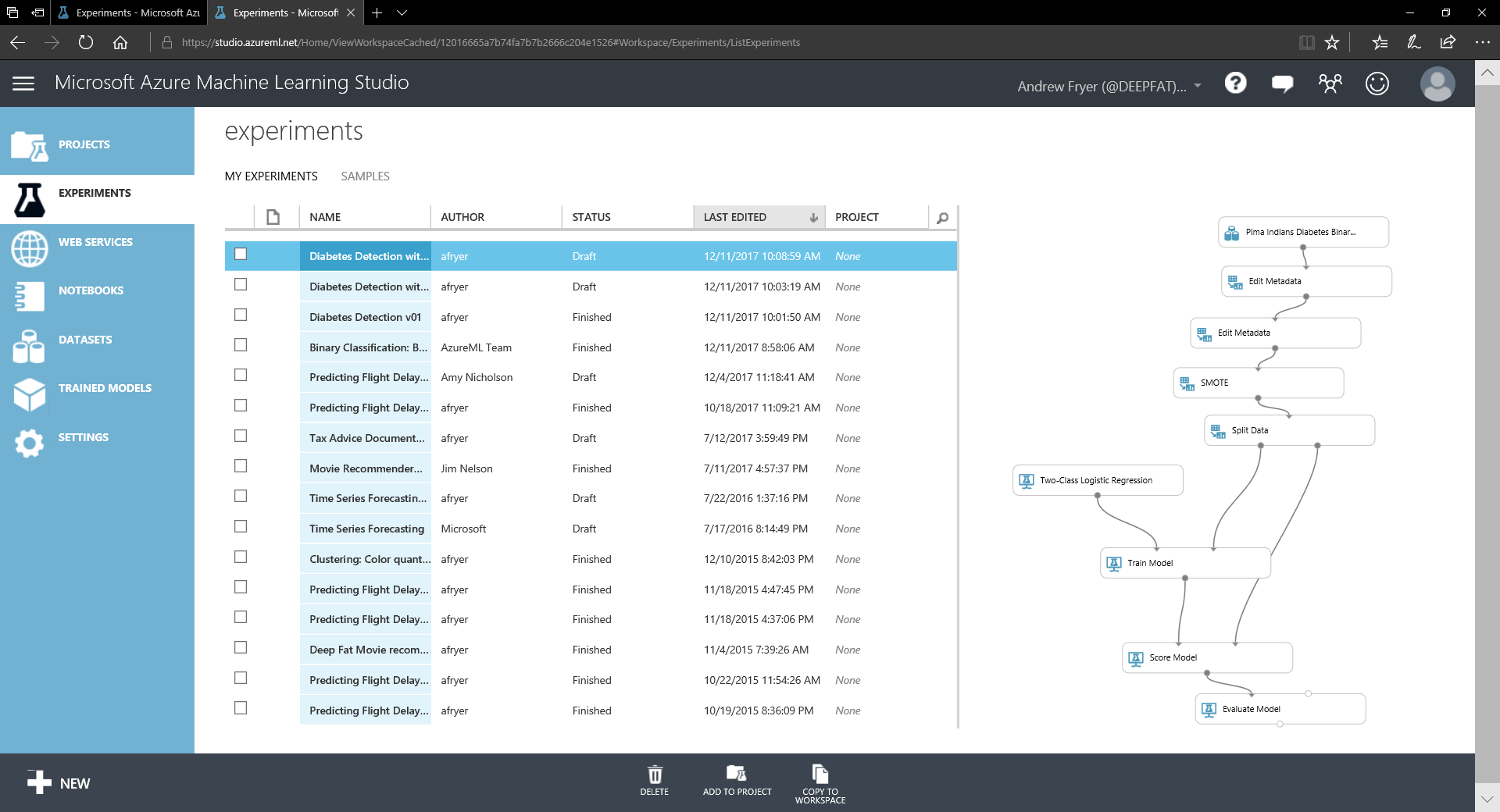
* Azure ML accepts many formats including CSV, TSV, Plain Text and Zip files
* Alternatively, you can use the ‘Import Data’ Module to access your data from sources in Azure such as databases like SQL Azure , Blob storage , HD insight (Hadoop and Spark) and industry standard OData feeds.

It’s also possible to adapt an existing experiment to our needs from one of the many samples in the [Cortana Intelligence Gallery](https://gallery.cortanaintelligence.com/) . If we open a new tab in our browser we can see the sort of samples that exist there which have either been created by Microsoft or the community of experts using the tool.



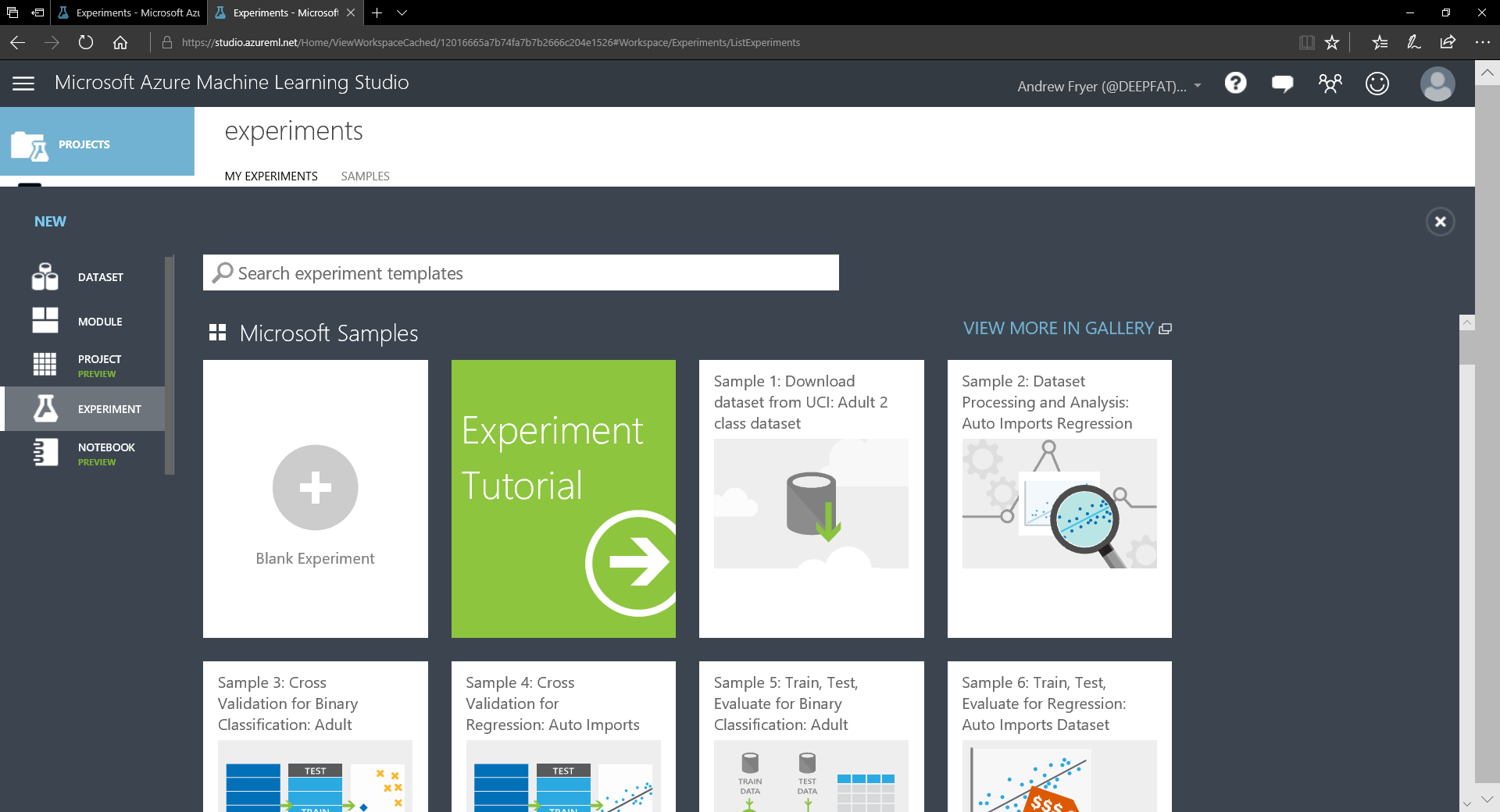
For this lab we will start with a blank experiment. Go back to to <https://studio.azureml.net/>

And sign in again if needed. We’ll get a screen like this except that as a new user there won’t be any experiments to display. Experiments are essentially models complete with the data associated with them

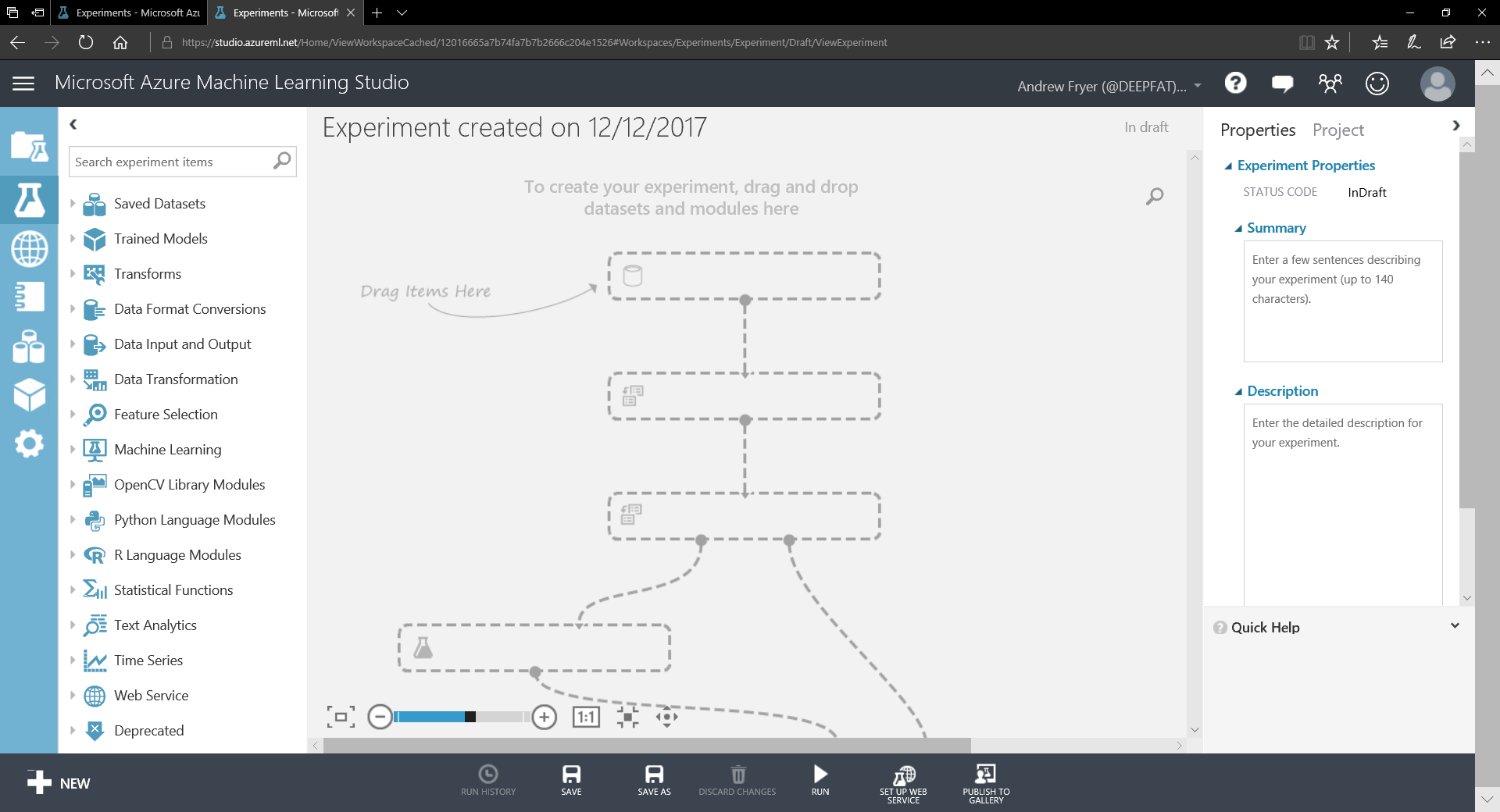


On the left in blue with white icons are the different objects we use in ML studio such as Projects, Experiments, Web services, Jupyter Notebooks, Data Sets, Trained Models and Settings.

To start this tutorial we’ll begin with a blank experiment. Click on the white plus sign on the bottom toolbar and select blank experiment:



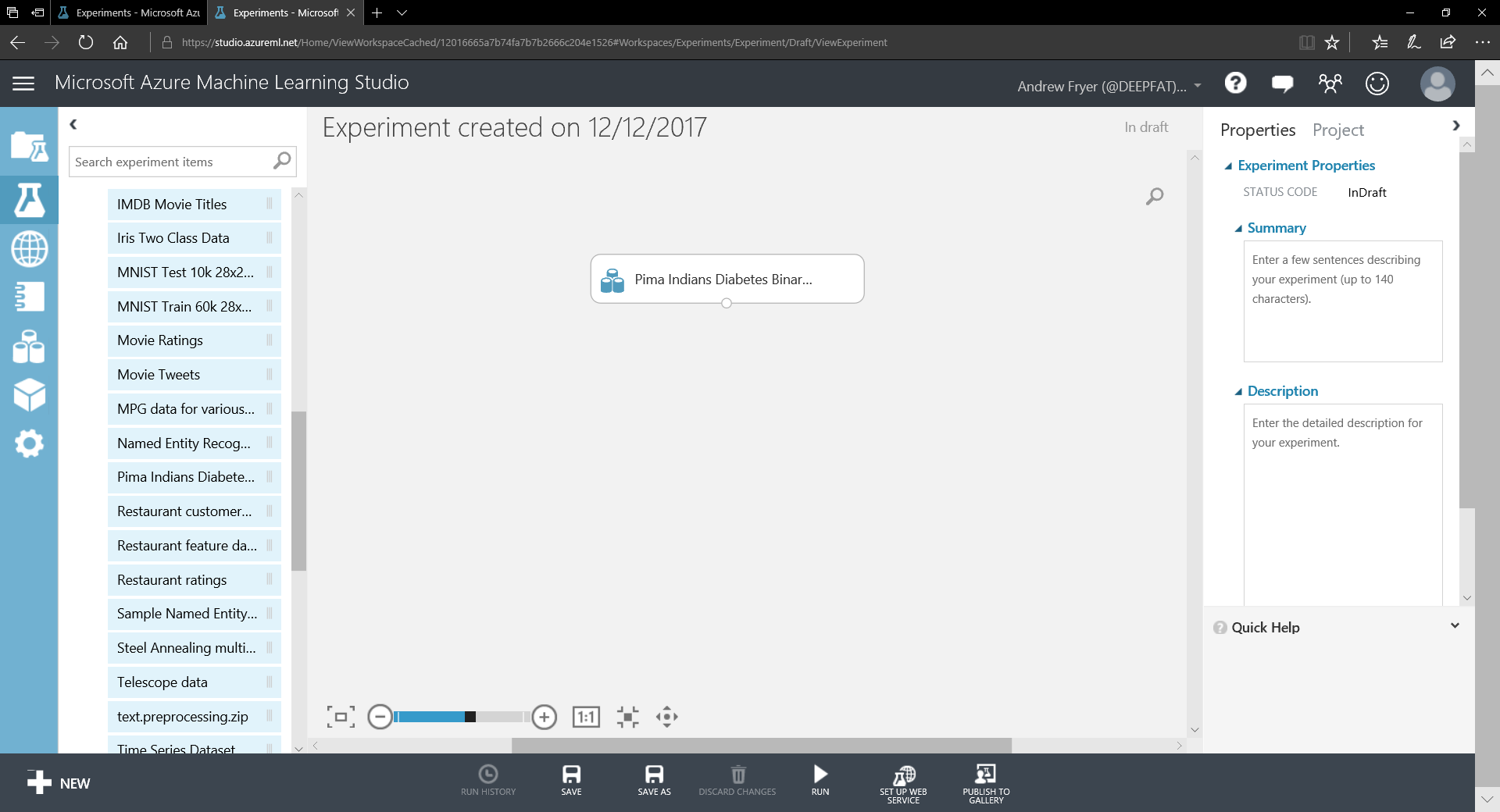
We are now presented with the design surface where we can create our own experiments:



On the far right we have all the objects in ML studio as before, but now we have a list of modules next to that which we can drag on to the design surface in the middle of the screen. On the right are the properties of the module that has focused or the whole experiment if no module is selected.

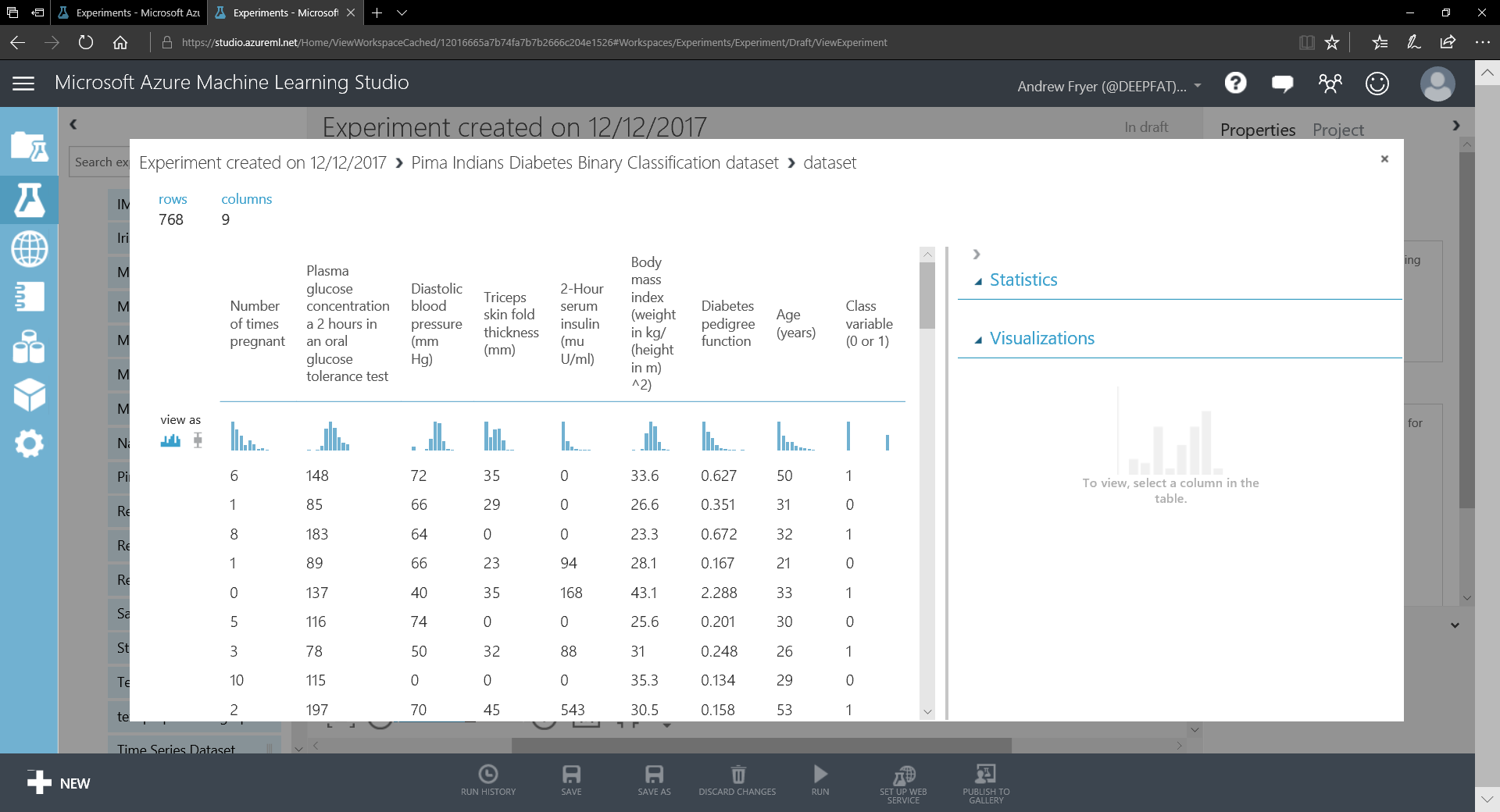
Initially the design surface just has a dotted representation of an experiment until we start to create our own experiment.

Typically we’d start work by connecting to a data source or just simply uploading a file to ML studio, but in this case we are going to use a built in sample dataset representing the Pima diabetes data mentioned earlier. Expand the Saved Datasets on the left, expand samples and scroll down to the Pima Indians Diabetes Binary Classification Dataset. Drag that onto the top centre of the design surface like so...

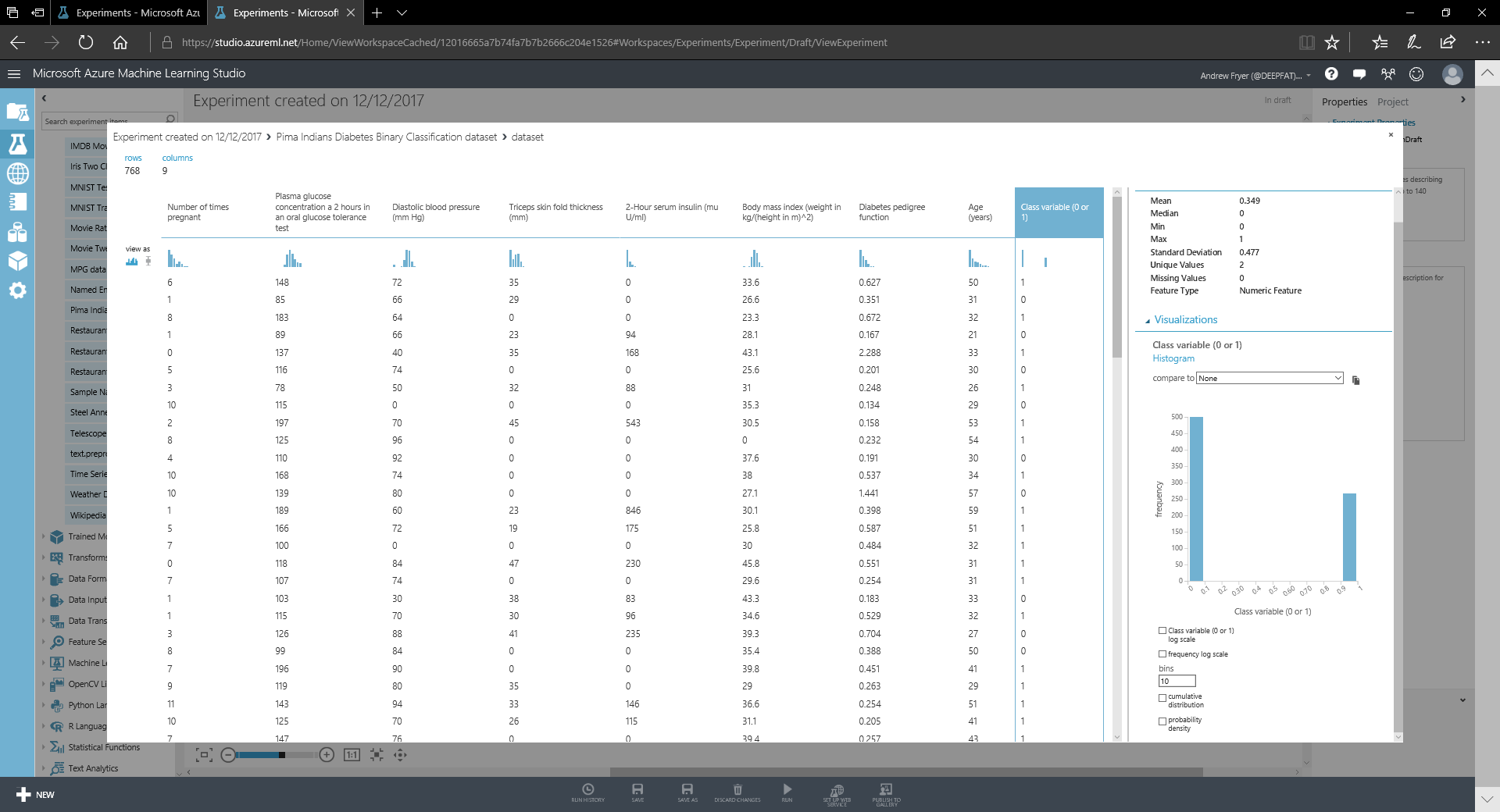


The first thing we should do is give our experiment a name – simply over type the “Experiment Created on …” and enter something like Pima Indians Diabetes Classification. Note there is no version control in ML studio; changes are saved for us but we can 'save as' and give our experiments meaningful names like V01 as we go along.

Now we have data we can immediately look at it, Right click on the circle at the bottom of the dataset and select visualise:



We can quickly see key information about the data – there are only 768 rows in 9 columns. We can see stats about a numeric column by selecting it. Select the Class variable (0 or 1) and we’ll see data about that column on the right hand side:



We can immediately see that there are only two values (0 or 1😊) and that there are about 35% of these people with diabetes (in this case the mean of 0.349 gives us that empirically as well).

Ordinarily we’d want to clean and check this data before doing anything else with it, so what might that entail?

* Typically we’d want to check for null values and duplicates and have techniques to eliminate these.
* We may also want to adapt the data into new types of data - for example we might put numerical values into bins (aka quantisation) In this data set we might do that for any of the numeric data e.g. create five bins for different BMI values and call them very low, low, medium, high, very high.
* If we don’t do that we could regularise the data so that all numeric values are scaled to be between 0 and 1 as this can help with certain types of algorithm and give better faster results.

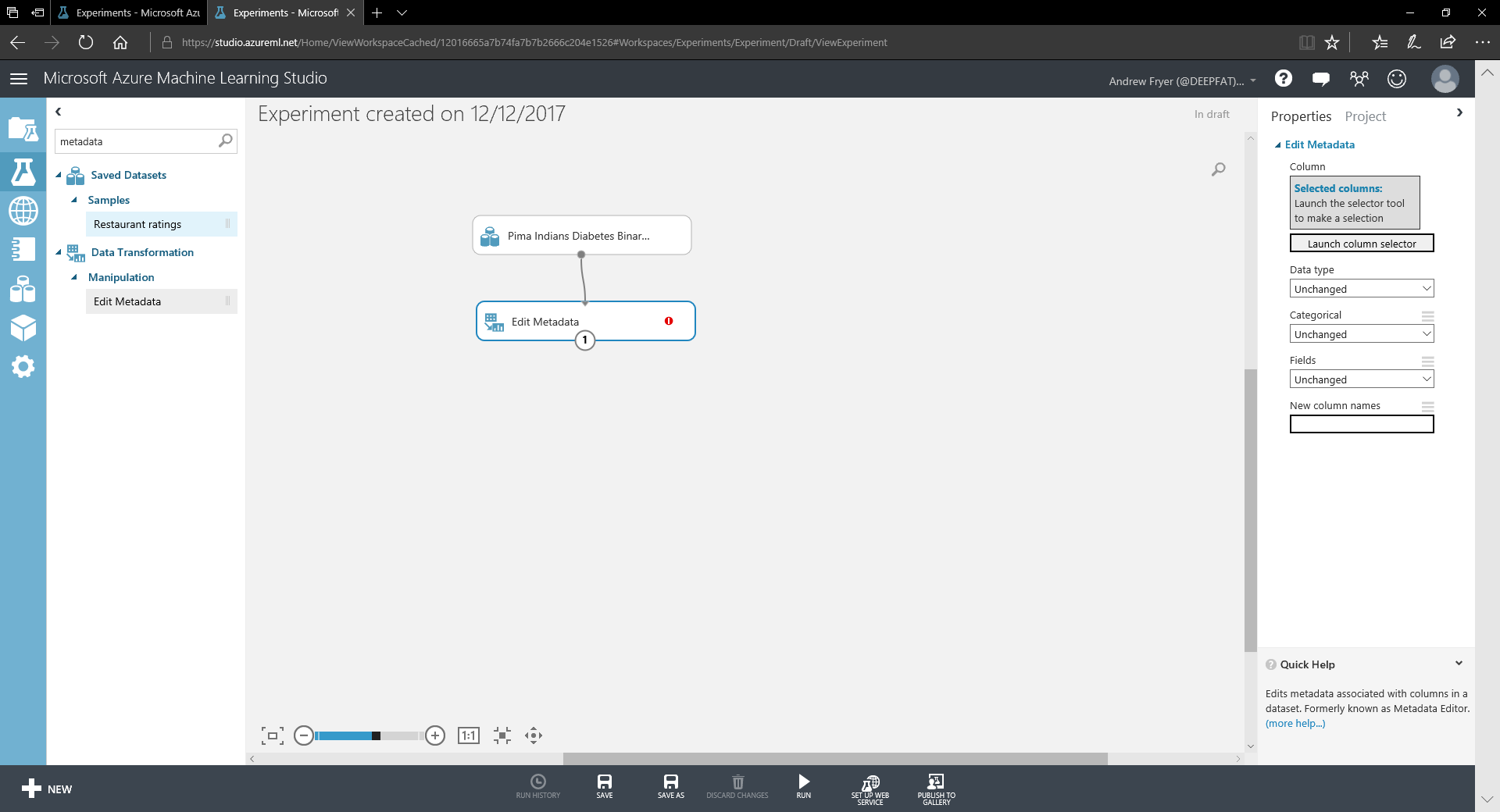
**What else can you think on based on what you have been taught so far?**

One thing we should do before we think about building a model is editing the metadata in two ways.

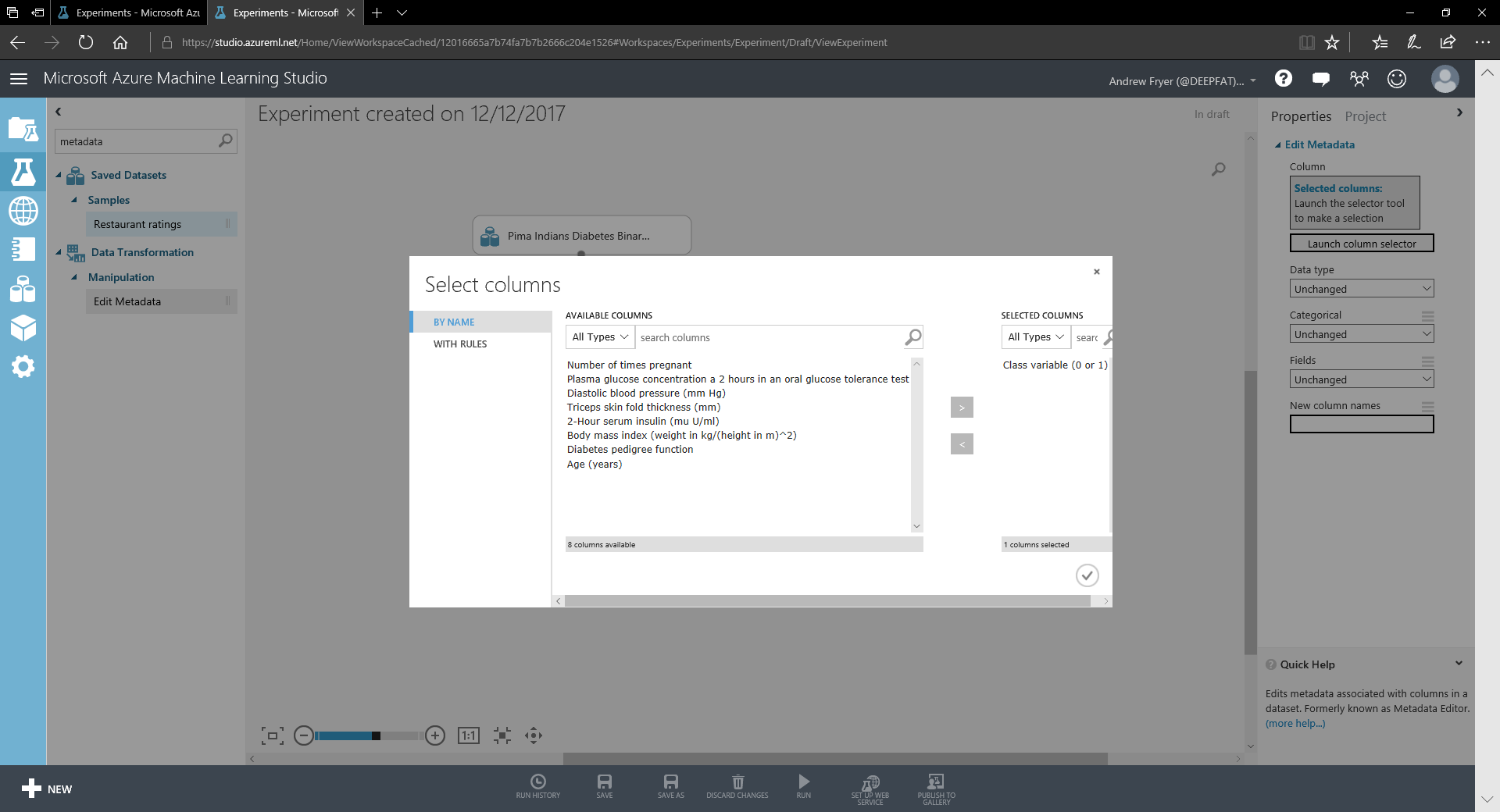
We should declare what we are trying to predict, the label and data used to make that prediction, the features. Also the field names are very easy to understand but are not great to use in programming – they have special characters in them such as brackets and spaces which can make this difficult.

There is a module in ML studio called Edit Metadata which we can use to fix these issues.

Close the visualise window, and in the search window above the list of modules type metadata (this is the quickest way to locate a given module). Drag this module underneath the dataset module. Then connect the two modules by pointing to the small circle under the dataset module and dragging to the small circle above the edit metadata module:

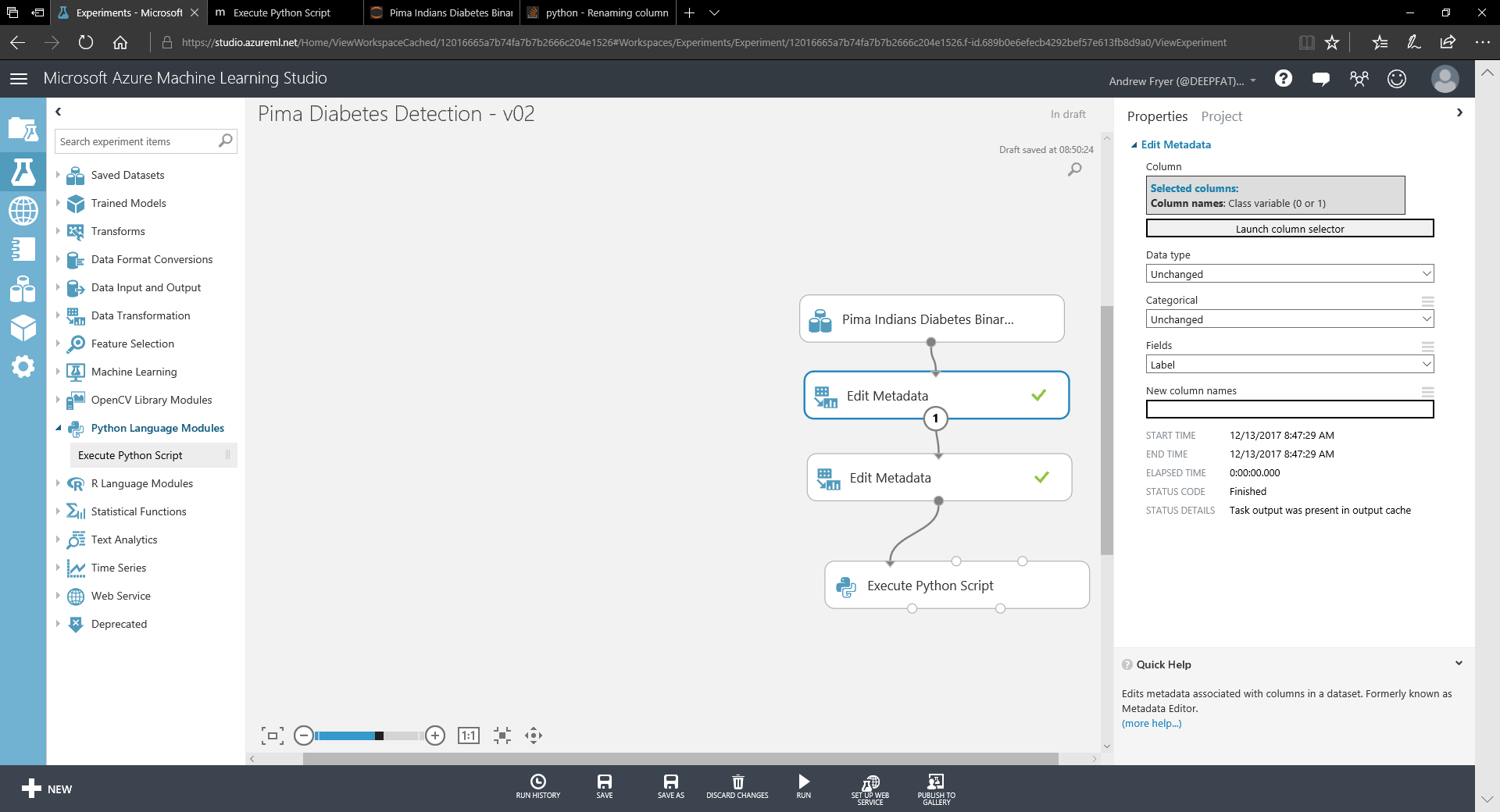


Note the metadata module has a warning against it because its properties are not set. We can see those properties above because that module is selected. Click on the box on the right of the screen, Launch Column Selector:

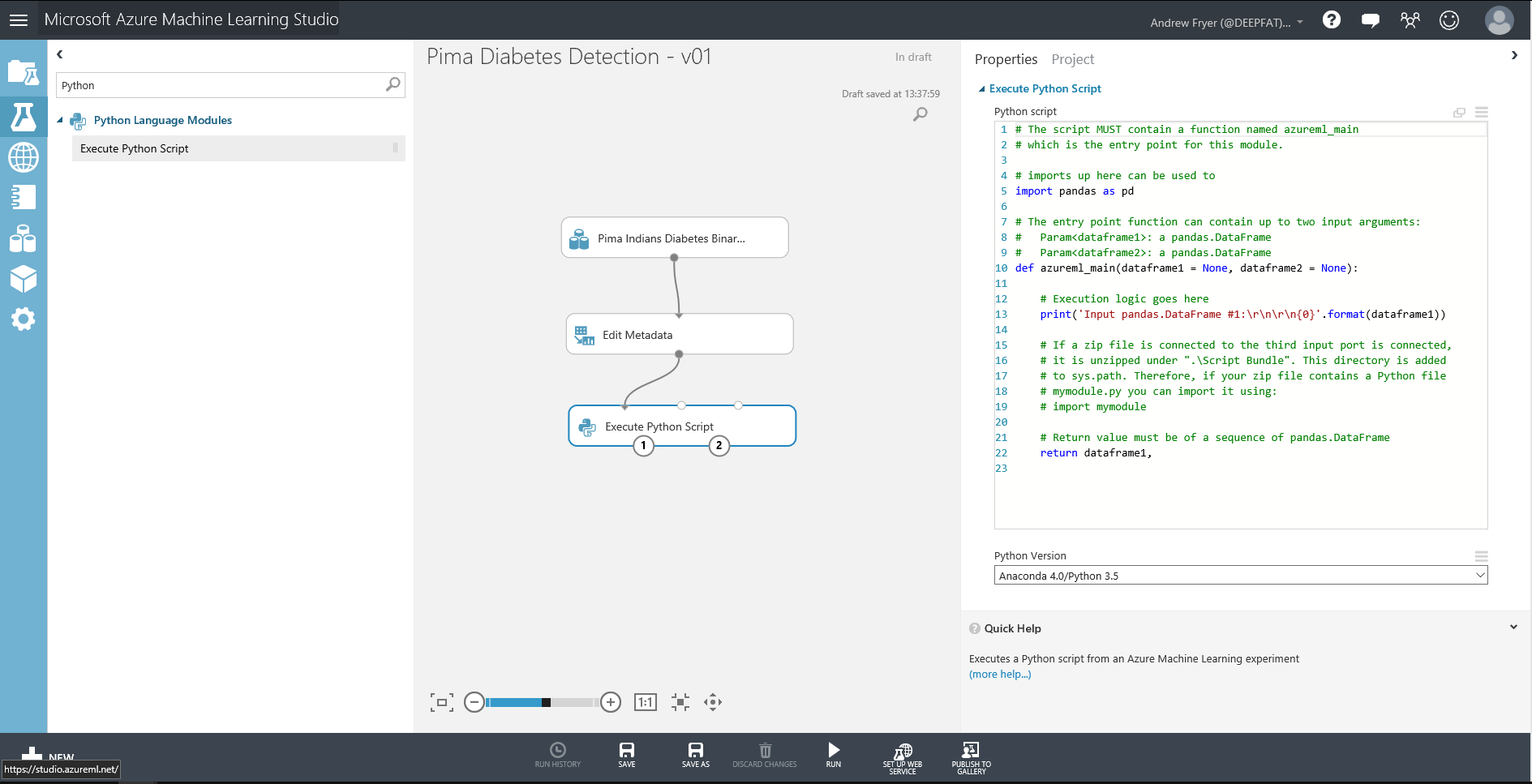


Select Class Variable and the > sign to move it to the SELECTED COLUMNS. Click on the tick box bottom right (if we just close the windows our selection is not saved).

In the edit metadata properties change the settings as shown by making the 'Fields' property to be 'Label':



All the other columns are still defined as features but if we want to change all of those feature names to computer friendly names then we’d need one Edit Metadata module for each column - so is there a better way? Yes - we can just write the code in R or Python. As we have already used Python on the course we’ll use that. In the module search 'Python'. Drag the Python module onto the design surface and connect it as shown.

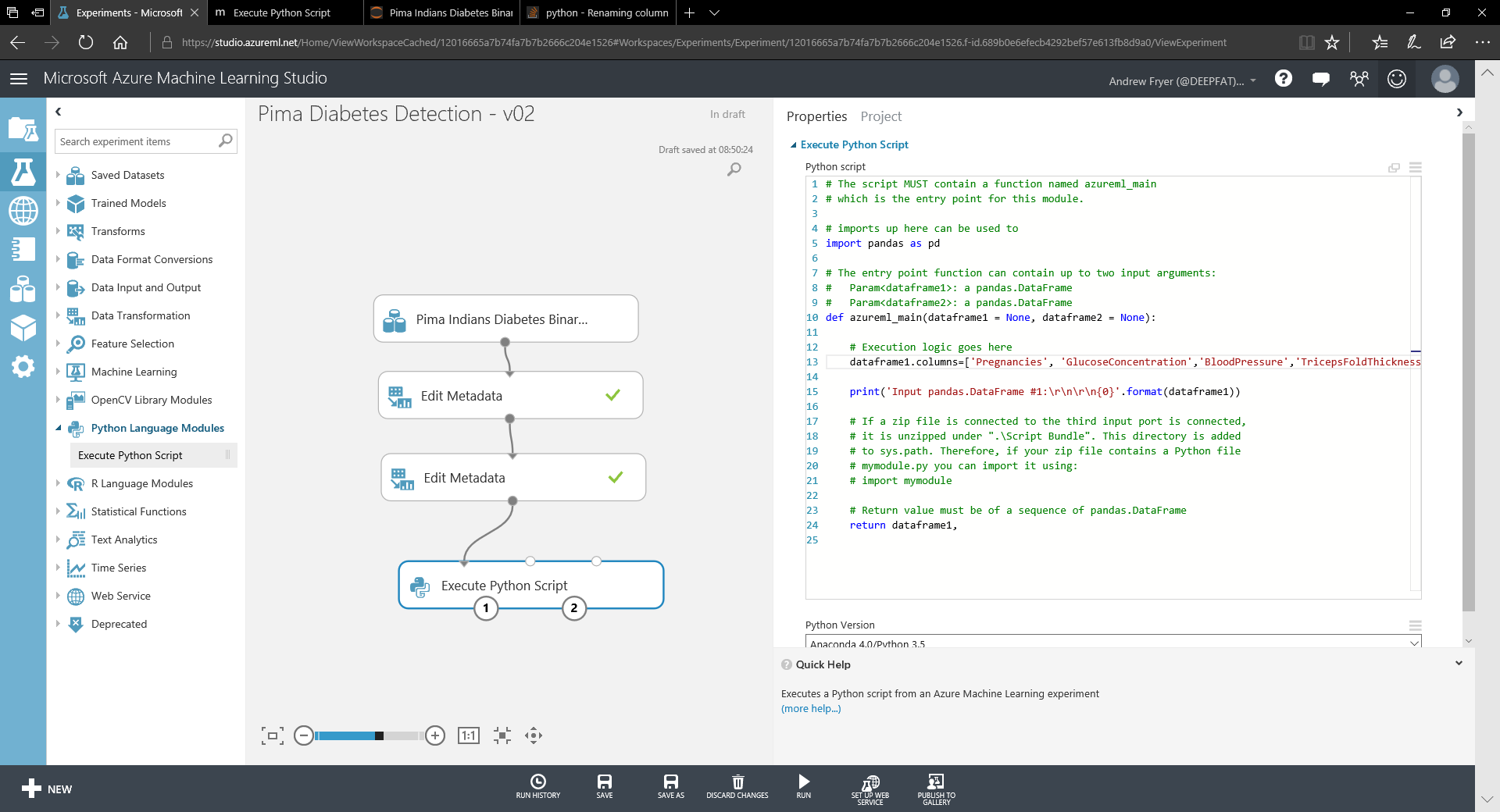


The Python module has three inputs. It can accept two datasets to enable us to code a join across them and we can add our own library on the right-hand input port by zipping it and uploading it. There are also two output ports, one for the data and one for the console output including any graphics. On the right there is a basic code editor which can be configured to work on various versions of Python. If we look at the stub code, dataframe1 is already setup with our data flowing in port 1. Clicking on help on any module will give us more information. The environment already has a lot of python libraries included and the script can import more as needed. The data comes in as Pandas dataframes.

In our case all we need to do is add this line of code

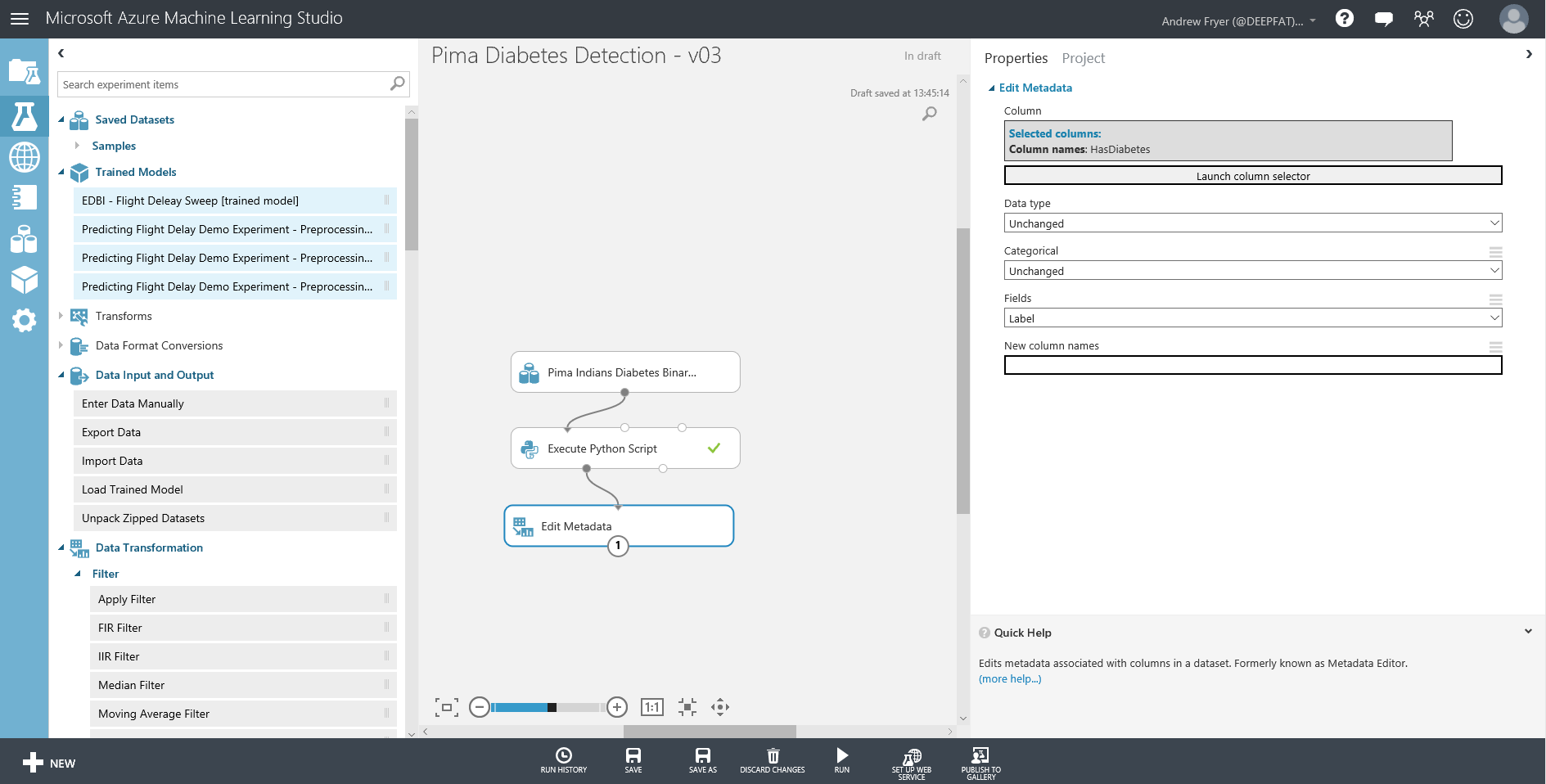
dataframe1.columns=['Pregnancies', 'GlucoseConcentration','BloodPressure','TricepsFoldThickness','SerumInsulin','BMI','DiabetesPedigree','Age','HasDiabetes']

as line 13 just before the print statement



To check the code is working we can run the experiment. Click on the Run icon at the bottom of the screen. If all is well we’ll see ticks for each step and if we right click on the left output of the Python module we’ll see that our columns have been renamed.

Unfortunately, one of the outputs from the Python module loses the fact that we declared the Field as 'Label' earlier, as the data is passed into and out of a pandas dataframe. To fix that we need to re order the modules as shown. You can right-click the arrow connections to delete them, move the modules, and then re-add the connections:



Right click on the Execute Python module and select Run Selected to rerun just this module. This is necessary to get ML studio to recognise the new column names.

Now we’ll need to change the Edit Metadata module to recognise the new column name for the label (HasDiabetes). Select the Edit Metadata module and launch the column selector, select 'By Name' on the left-hand side and select HasDiabetes.

Now we are ready to start to build a model.

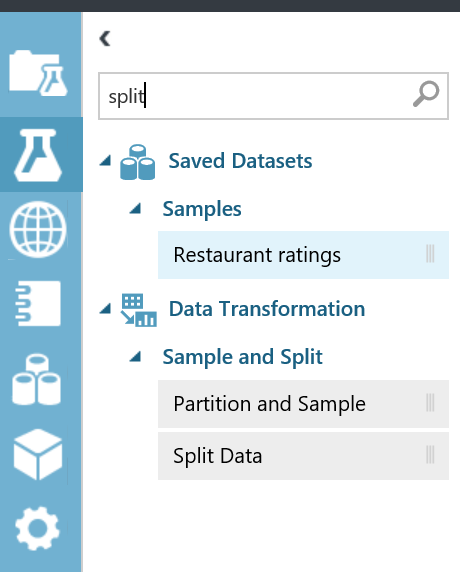
Training the Model

At this point in the experimentation process we are assuming we have a clean set of data that can be used to train and evaluate a model. We are moving from the business knowledge domain to the machine learning domain. We have identified that we are trying to predict the label (now renamed to be HasDiabetes) which can be 1 or 0 (two class classification) against a set of data about Pima Indians.

We will train a model against our data using one of the built-in algorithms in Azure ML. Once the model has been trained it can be used to make predictions about whether a given person is likely to have diabetes that the model has not seen. In order to check how good the model is, we must evaluate the predicted answer against the original value (in the HasDiabetes label column). This is just like any good science experiment - we want a control group to check the accuracy of the model.

To do this we split the data into two random sets and we’ll want to use most of the data (typically 80%) for training and reserve 20% for scoring. We’ll also want to stratify the split over the label to ensure that each of these separate groups of data have the same ration of values in the label i.e. the train score and evaluate sets of data will have the same ratio of people with and without diabetes.

In Azure ML we do this by using the built-in split module. The easiest way to find this is to type 'split' into the search box at the top of the list modules:

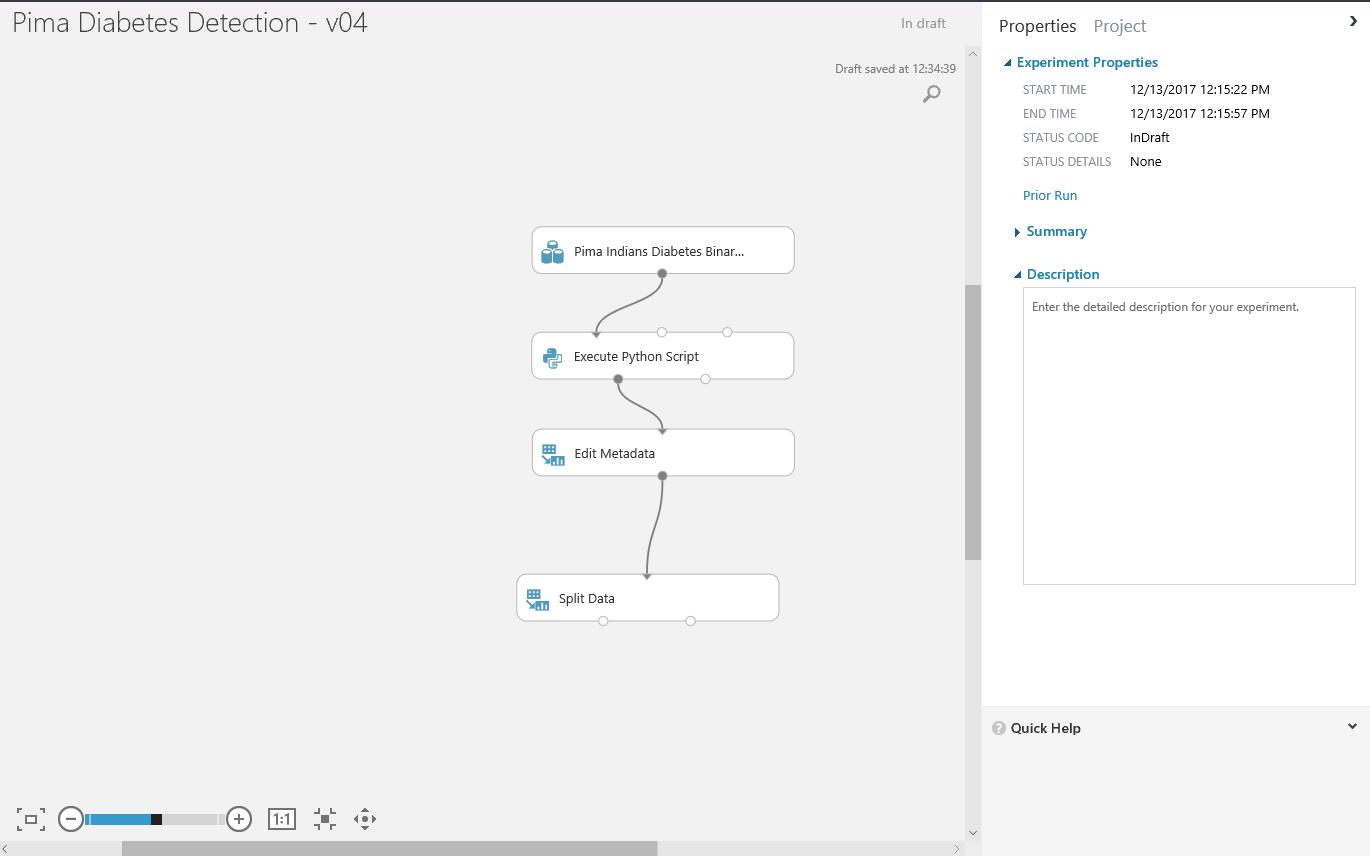


Now drag and drop the 'Split Data' module onto the workspace. Connect this to the output of the 'Edit metadata' module and set the ‘fraction of rows in the first output dataset’ field to 0.8 (80%).

Set the 'Stratified Split' option to True and launch the column selector. Select with rules, and then set the dropdown boxes to 'Include' and 'all labels'. Click the checkbox to close the dialog to save changes. This step should appear as shown below:



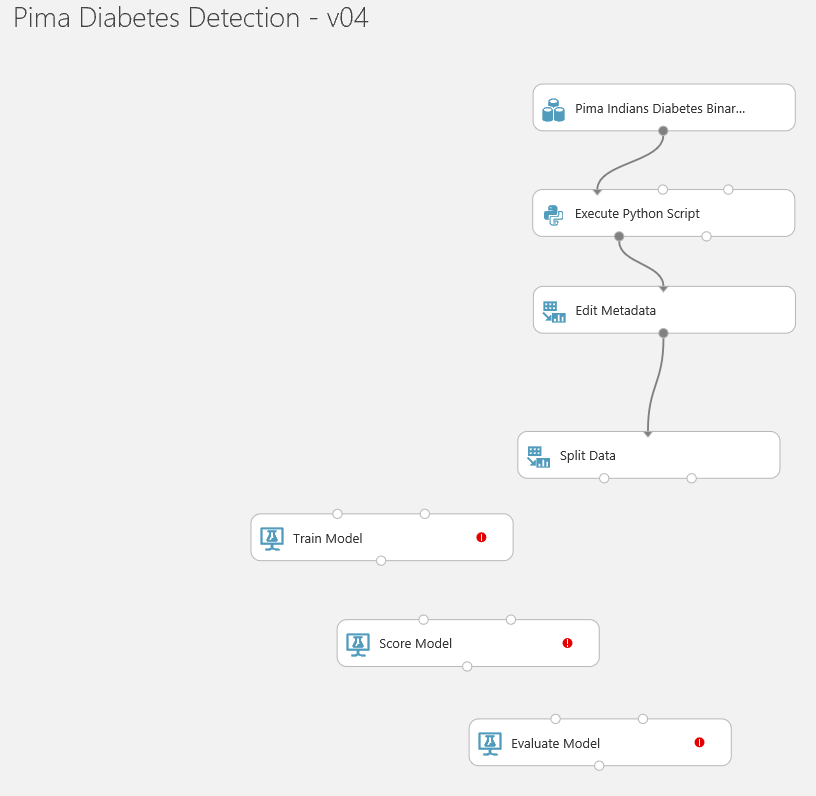
The properties of the stratified split should now look like this:



We can check the split is working by rerunning the experiment (run is at the bottom of the screen) and then visualising the output of each of the Split Data ports. Right-click the first output port and select 'Visualize' - observe the number of rows in the data, and so the same for the second output port. We can now check that the mean of each set of data is roughly the same – remember we just need to highlight the HasDiabetes column to see basic stats about it.

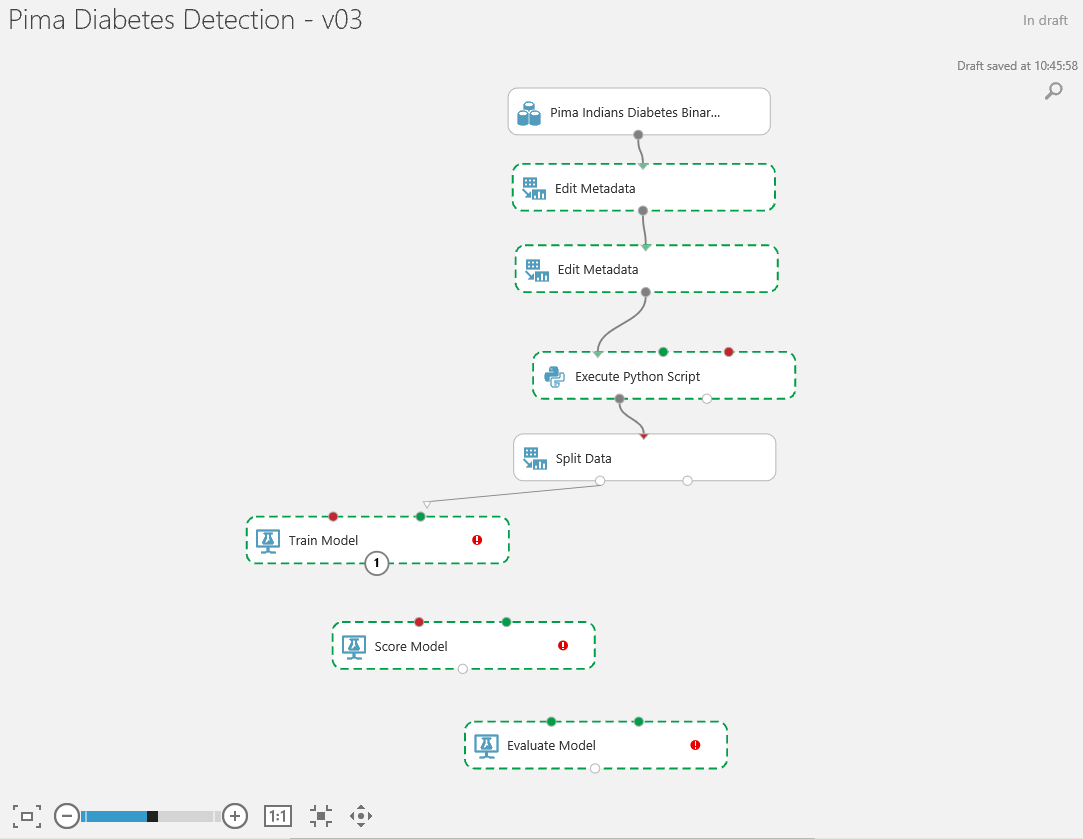
Now we can setup the process of training scoring and evaluating a model based on the [Cross Industry Standard Process for Data Mining (CRIP-DM)](https://en.wikipedia.org/wiki/Cross-industry_standard_process_for_data_mining).

We’ll need three modules for this Train Model, Score Model, and Evaluate Model. Find each of these and drag them onto the design surface like this..



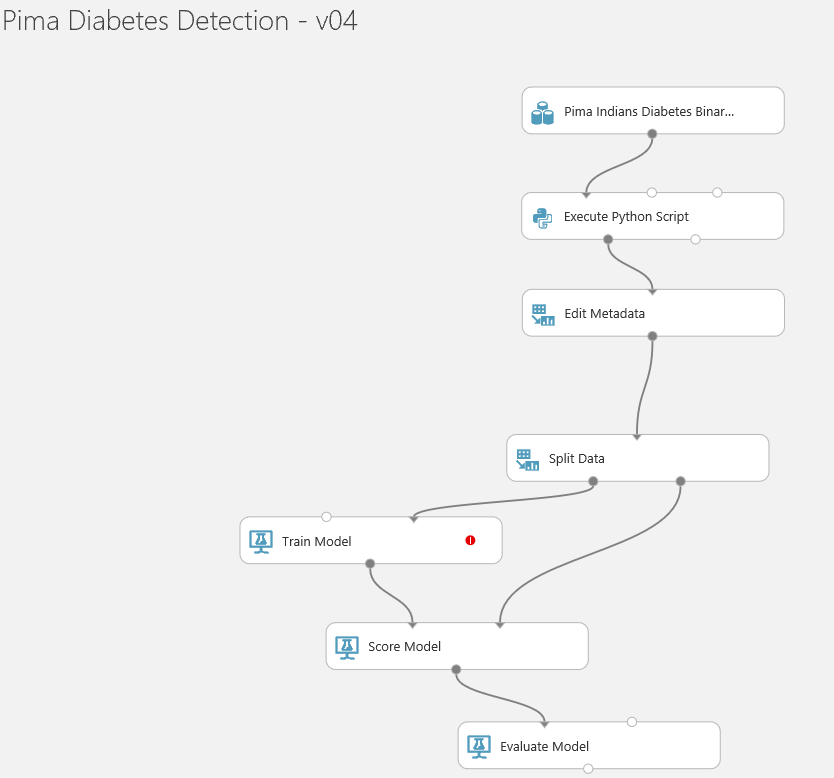
Now we can wire these together as follows:

* We’ll use 80% of the data for training the model so the left output of Split Data goes to the Train Model. Notice it can only be connected to one of the input ports marked in green



* We’ll want to score the output of the trained model so connect the output of Trained Model to the input of Score model – again we can only select one port of Score Model for this
* We’ll use the other 20% of the split data to **score** the model by comparing it with the output of the trained model so connect the other output of split to the remaining port of Score Model module.
* Finally we can use the built in Evaluate Model to show the output of Score Model visually and as a set of statistics. Connect the Output of Score Model to the Left input of Evaluate Model

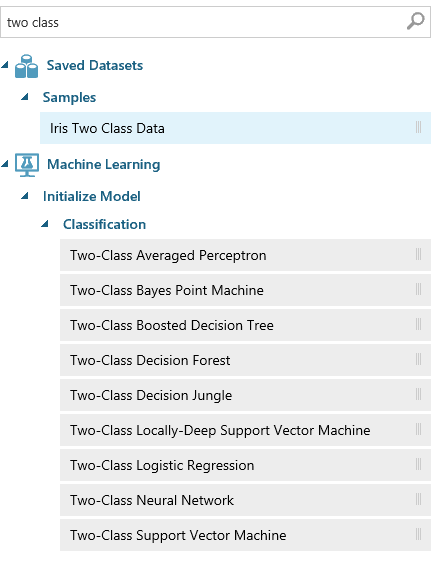
Your diagram should now look like this and there will be a warning on the Train Model module.



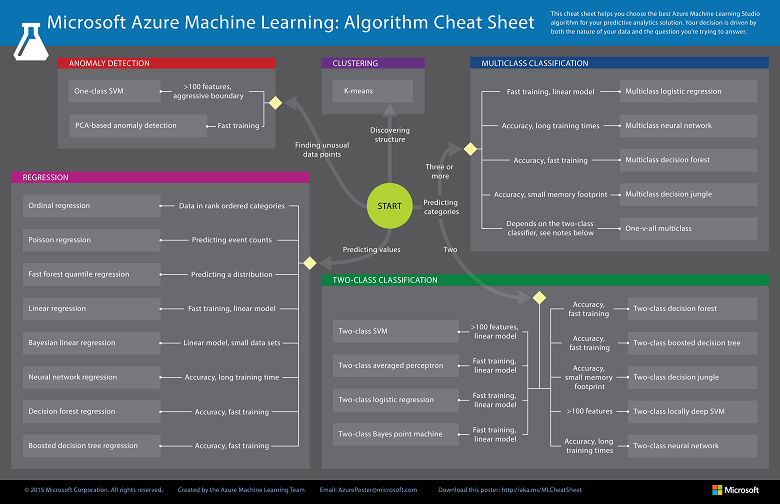
There are two reasons for this warning:

* We haven’t specified what we are trying to predict – unfortunately ML Studio doesn’t automatically select the label. So on the properties for Train Model launch the column select, go to 'With Rules' and select 'Include all labels'. Don't worry that there is still a warning...
* Because we haven’t specified how to train the model!

This model is a two class classification problem and if we type 'two class' into the module search we can see that there are several algorithms we can use:

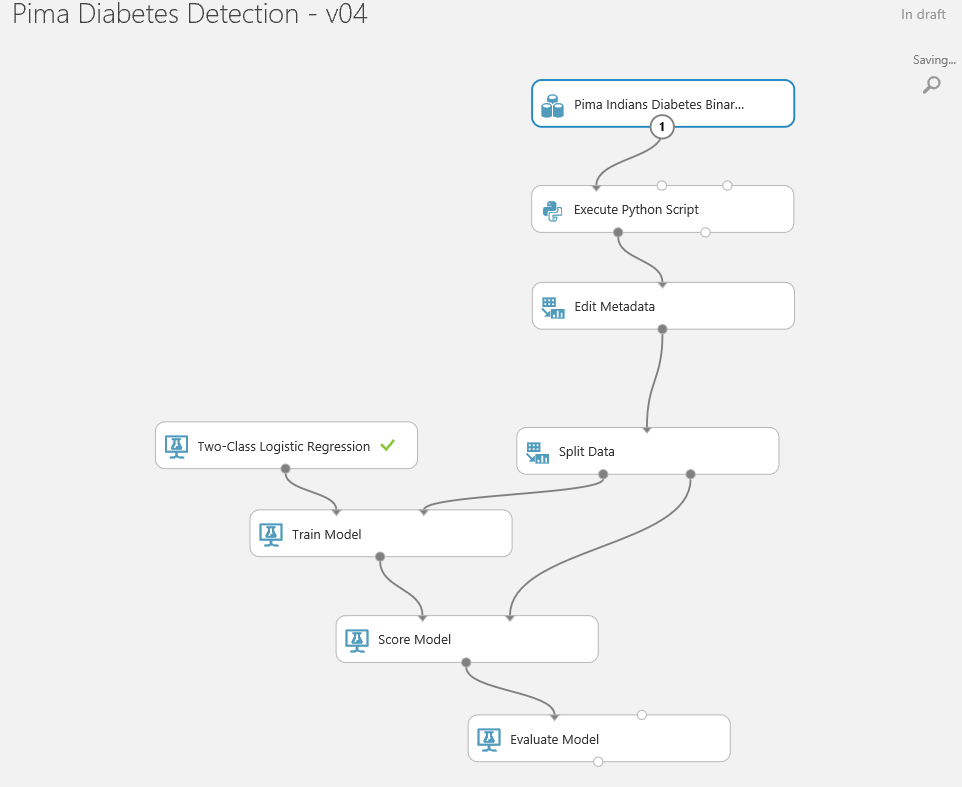


Any of these will work but some are better than others in certain circumstances, so which one should we use in this case? There is a [cheat sheet](https://docs.microsoft.com/en-us/azure/machine-learning/studio/algorithm-cheat-sheet) in the Azure ML documentation and below is the part of that for two class classification:



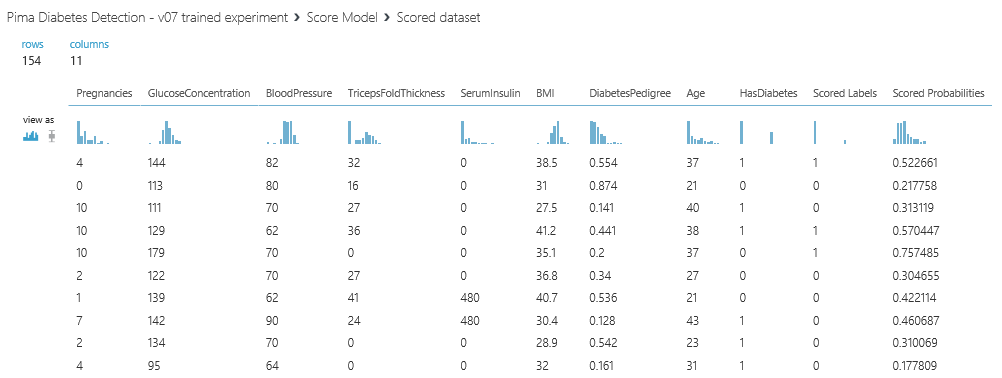
As we can see some models are fast, some are designed for accuracy, and others can handle lots of features such as pixels in an image.

For speed we’ll start with [Two-Class Logistic Regression](https://msdn.microsoft.com/en-us/library/azure/dn905994.aspx). Drag this module onto the design surface above Train Model and connect its output to Train Model..

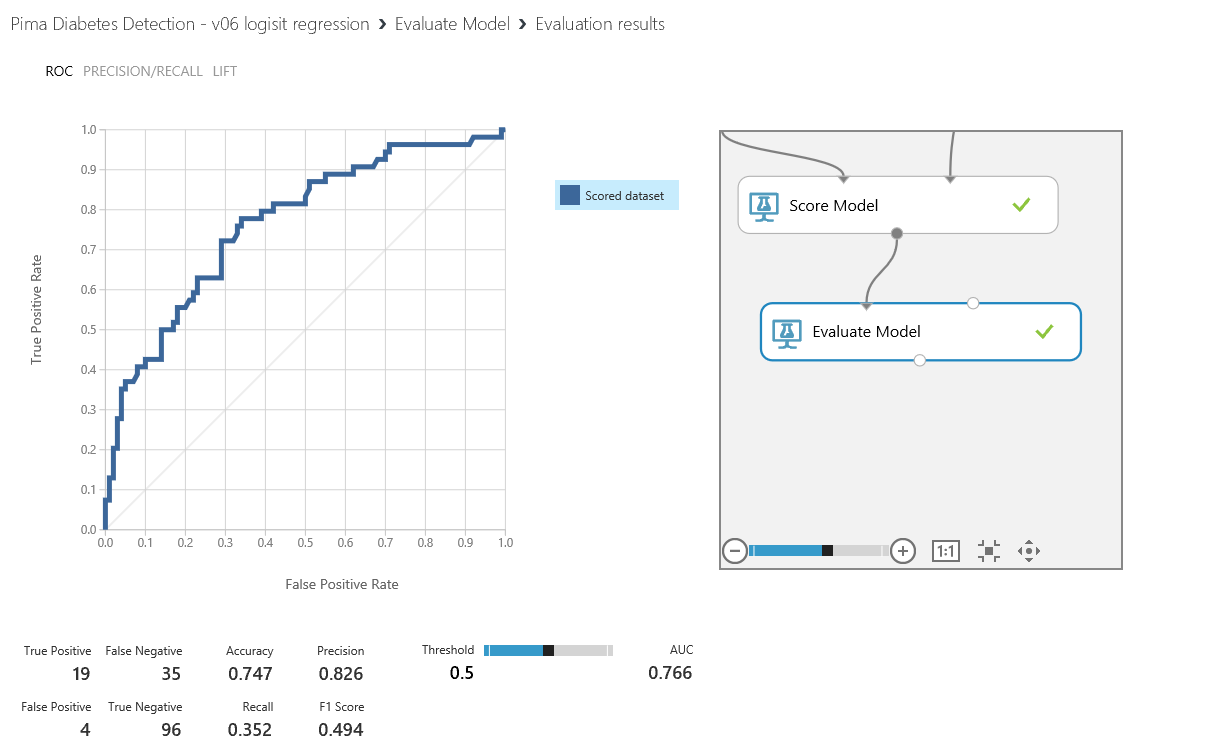


As there are no longer any red warnings we can run the experiment – this should take a few minutes. (Ask a lab supervisor if this isn't the case!)

Once it has completed we can quickly see how well the model performed. Firstly if we look at the output from the Score Model (right click on the small circle and select Visualise) we can see we have two new columns Scored Label and Scored Probability. We can then see how the Scored Label compares to the HasDiabetes column:



However if we want understand the trend across all of the data then we need to visualise the output from the Evaluate Model module:



The Receiver Operator Curve above shows us how well the model performs across the data and the greater area under the curve the better the model is – up to a point that is. If the curve is very close to being an upside down ‘L’ then the model is overfitted or there may be a feature in there that is actually a function of the label. Underneath the curve is the confusion matrix. This grid of numbers is controlled by the threshold setting on the right - if the scored probability of a record is over the threshold value (in this case 0.5) then it is scored as 1.

The calculations for accuracy etc. are as follows:

**Accuracy** = True Positives + True negatives over the overall total = 0.747

**Recall** = true positives divided by true positives + false negatives = 0.352

**Precision** = true positives divided by true positives + false positives = 0.826

**F1 score** = .2 \* (precision\* recall)/ (precision + recall) = 0.494

For extra credit you can do the evaluation in Python. Connect another Python module and set it to the SAME VERSION as the earlier Python module (only one version can be used in any one experiment). The code you’ll need is this:

import matplotlib

matplotlib.use(‘TkAgg’)

import sklearn.metrics as m

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.metrics import confusion\_matrix as cm

def azureml\_main(dataframe1 = None):

dataframe1 = dataframe1.dropna()

# pick the label out of the dataframe and the positive label and plot the RoC curve

r1 = m.roc\_curve(dataframe1[‘HasDiabetes’], dataframe1[‘Scored Probabilities’], pos\_label= 1)

plt.plot(r1[0], r1[1], 'r-', label=‘Boosted Trees’);

plt.grid(‘on’);

plt.legend(loc=‘best’)

plt.savefig(‘roc.png’)

#Derive test statistics to show the accuracy of the model and output on POrt1

cmarray = cm(dataframe1['HasDiabetes'], dataframe1['Scored Labels'])

TrueNeg, FalsePos = cmarray[0]

FalseNeg, TruePos = cmarray[1 TruePos = float(TruePos)

FalseNeg = float(FalseNeg)

FalsePos= float(FalsePos)

TrueNeg= float(TrueNeg)

Accuracy = (TruePos + TrueNeg) /(TruePos+ TrueNeg + FalsePos + FalseNeg)

Recall = TruePos /(TruePos + FalseNeg)

Precision = TruePos /(TruePos+ FalsePos)

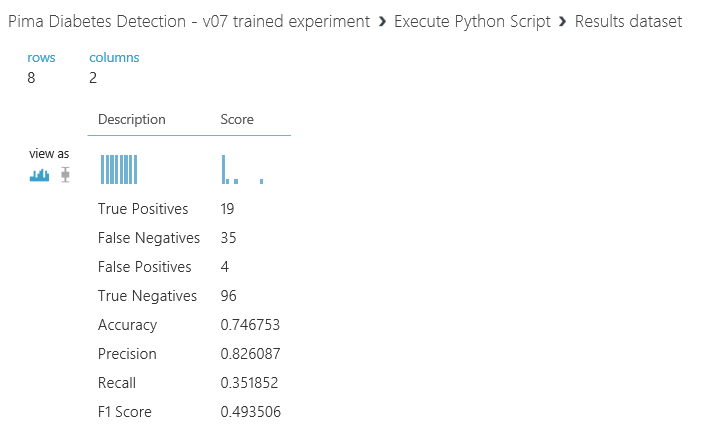
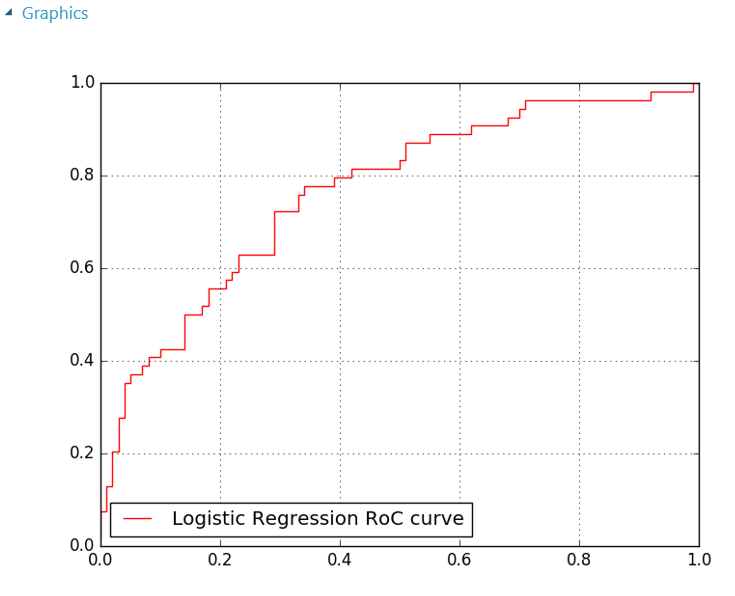
F1Score = 2 \* (Precision \* Recall)/(Precision + Recall)

data = {'Description': ['True Positives','False Negatives','False Positives','True Negatives','Accuracy','Precision','Recall','F1 Score'],'Score': [TruePos,FalseNeg,FalsePos,TrueNeg,Accuracy,Precision,Recall,F1Score]}

dataframe1 =pd.DataFrame(data,columns=['Description','Score'])

return dataframe1,

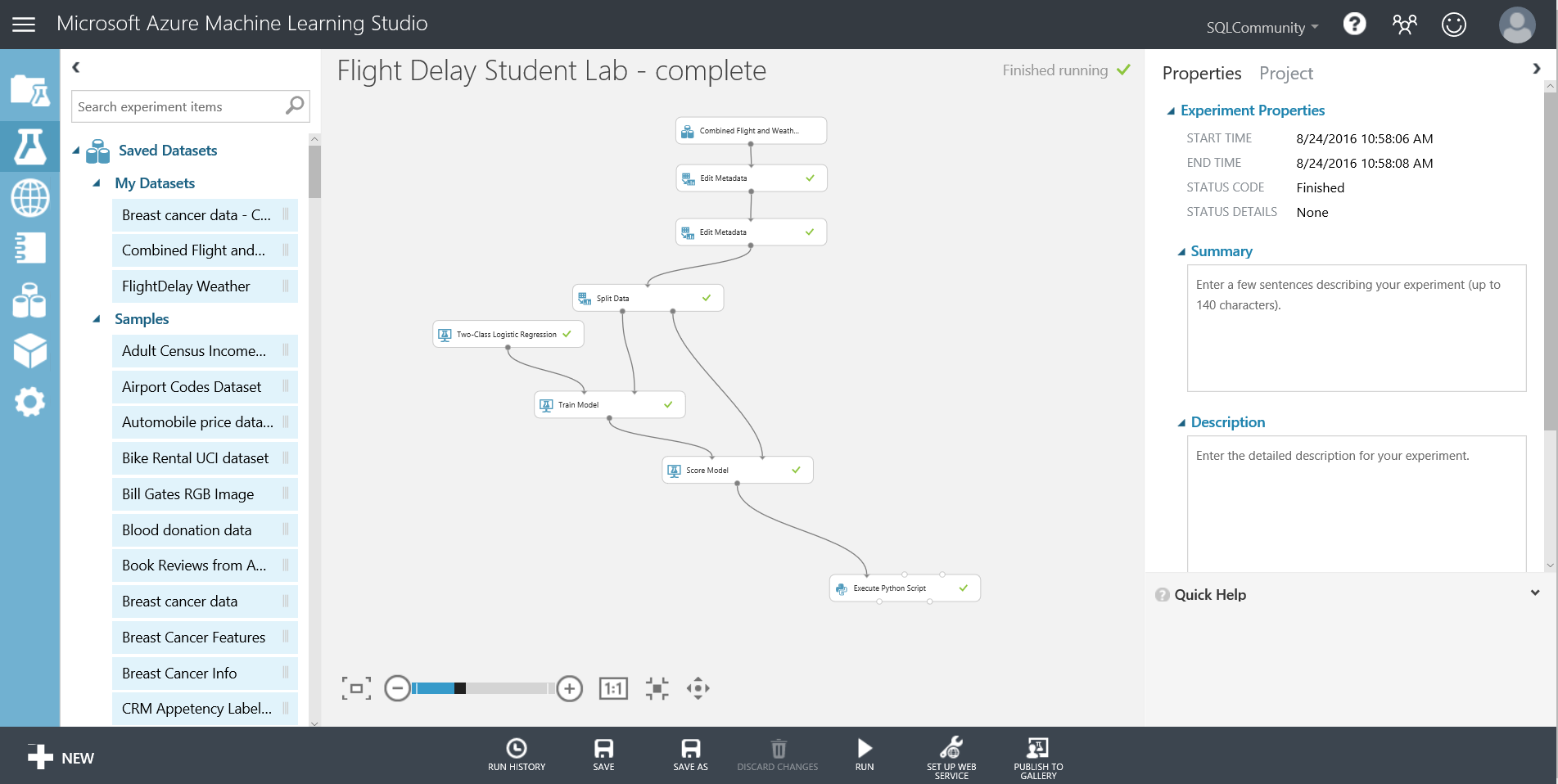
The confusion matrix will show up as the output data of the module and the plot will appear on port 2:

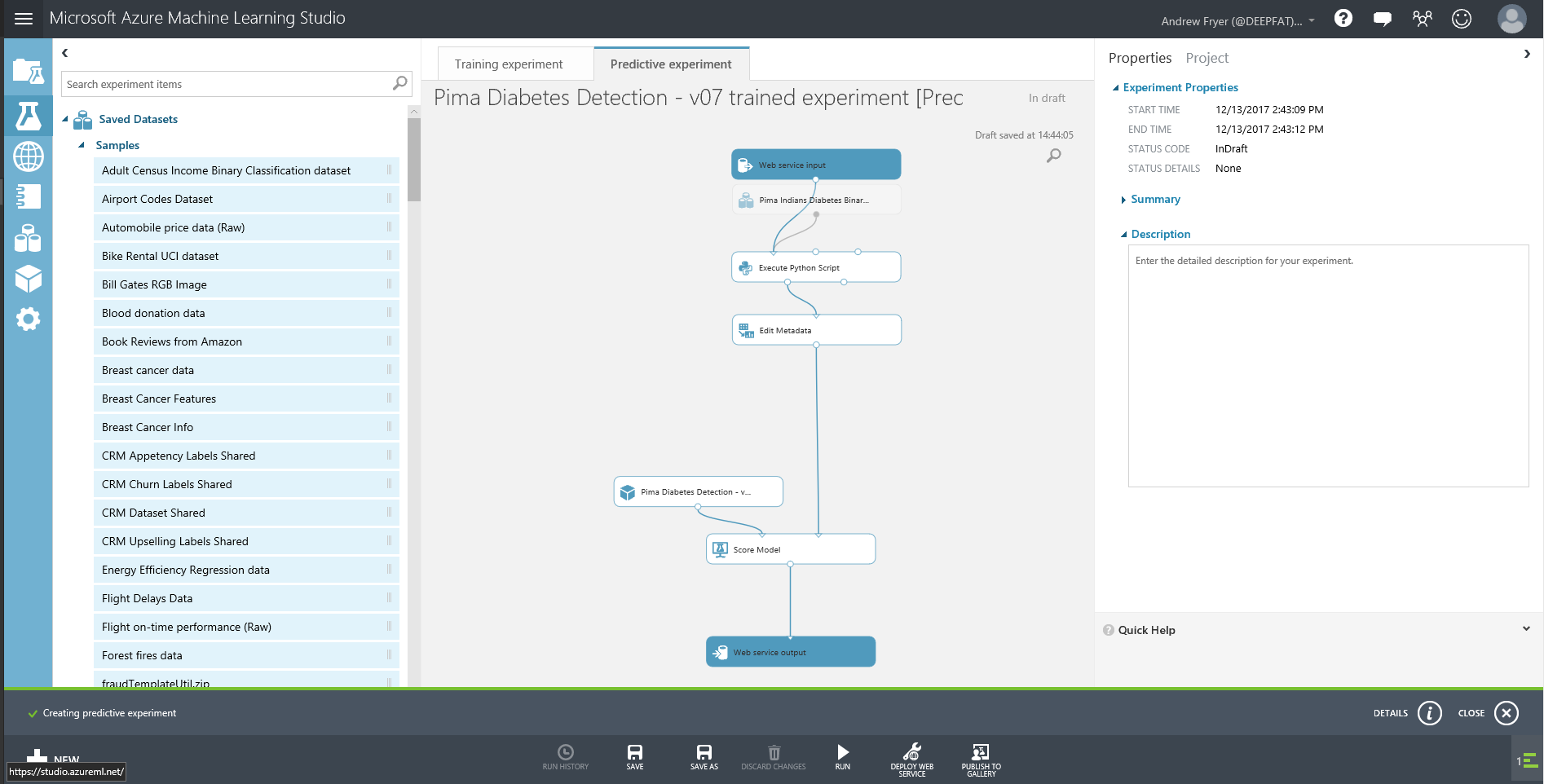
Publishing a Web Service

We have trained a model to identify diabetes in Pima Indians and checked to see how it is working. Now we want to use one of Azure ML’s key features – operationalising the classifier, by publishing an API to expose the model we have created. In order to do this, we first need to create a Training Experiment. This can then be published to the Azure ML web API service to make it available for other users or applications to use as a web service or a REST endpoint.

This may sound like a hard task, however once your experiment has been run successfully - there are green ticks by each module you will see a button on the toolbar at the bottom of the screen become active:



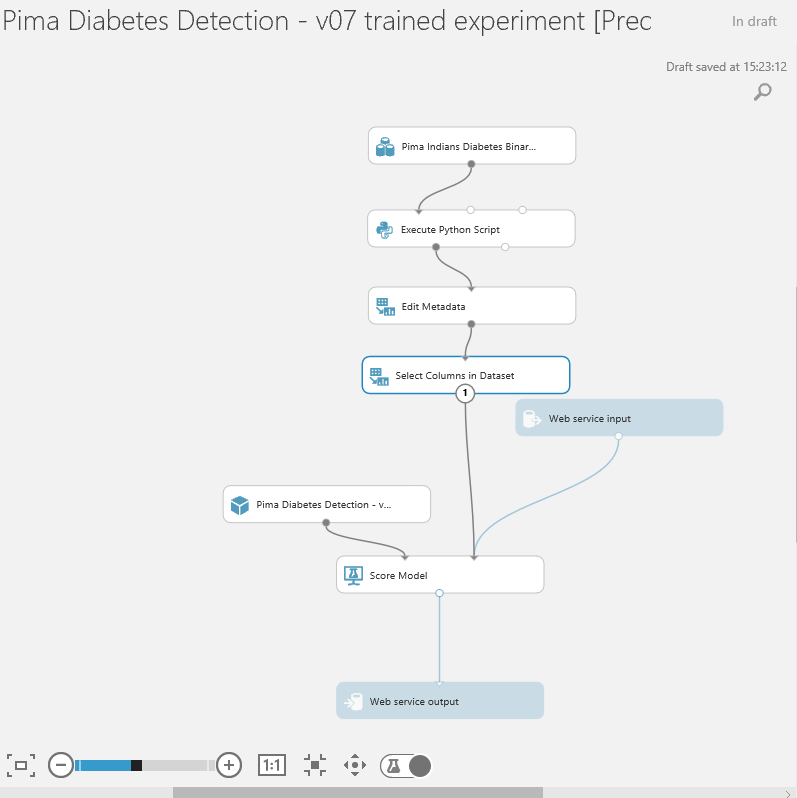
After clicking the ‘Predictive Web Service (Recommended)’ option our experiment appears to get redrawn and consolidated. What has really happened, is that ML has created a new Predictive experiment tab at the top of the screen.



and this is what we are looking at. Our original experiment is still there as it was on the Training experiment tab. Note there are two new modules in blue, “Web service input” and “Web service output” which represent the data format that will flow into and out of the web service we are creating. The Web service input will use the same fields as the input to the module it is connected to, in this case what flows into the top metadata module.

While the wizard has done a basic job of placing where the input and outputs on our predictive experiment are, it is not perfect. The data flowing through 'feed 1' in the above diagram contains our label (HasDiabetes) which is what we are trying to predict. While this is valid for training we shouldn’t have it here.

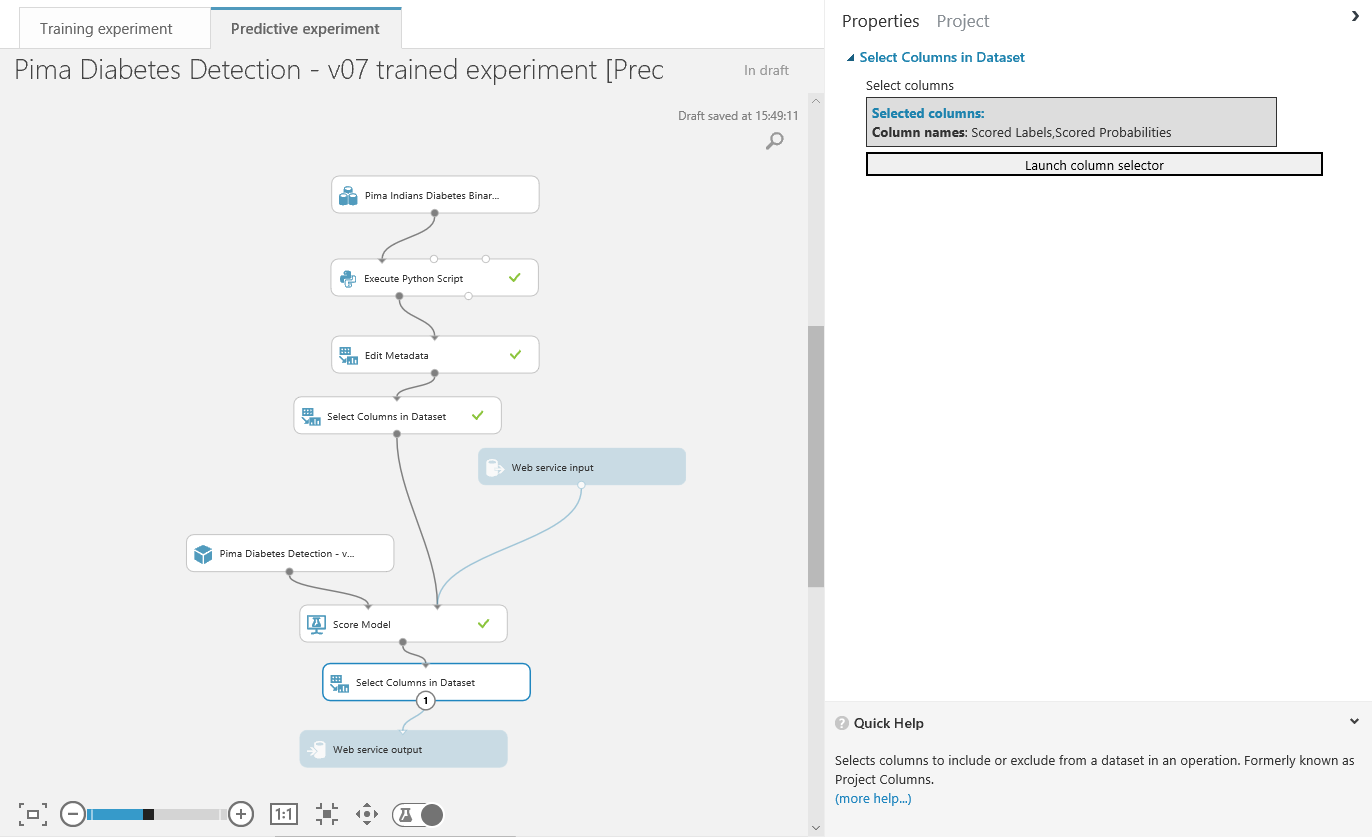
First we flip the toggle switch to experiment mode at the bottom of the experiment. Then we can more clearly see what is going on. Now we can reconnect the Web service input to connect with the score model module and adding a ‘Select columns in a Dataset’ module as shown:



set this module to exclude all labels:



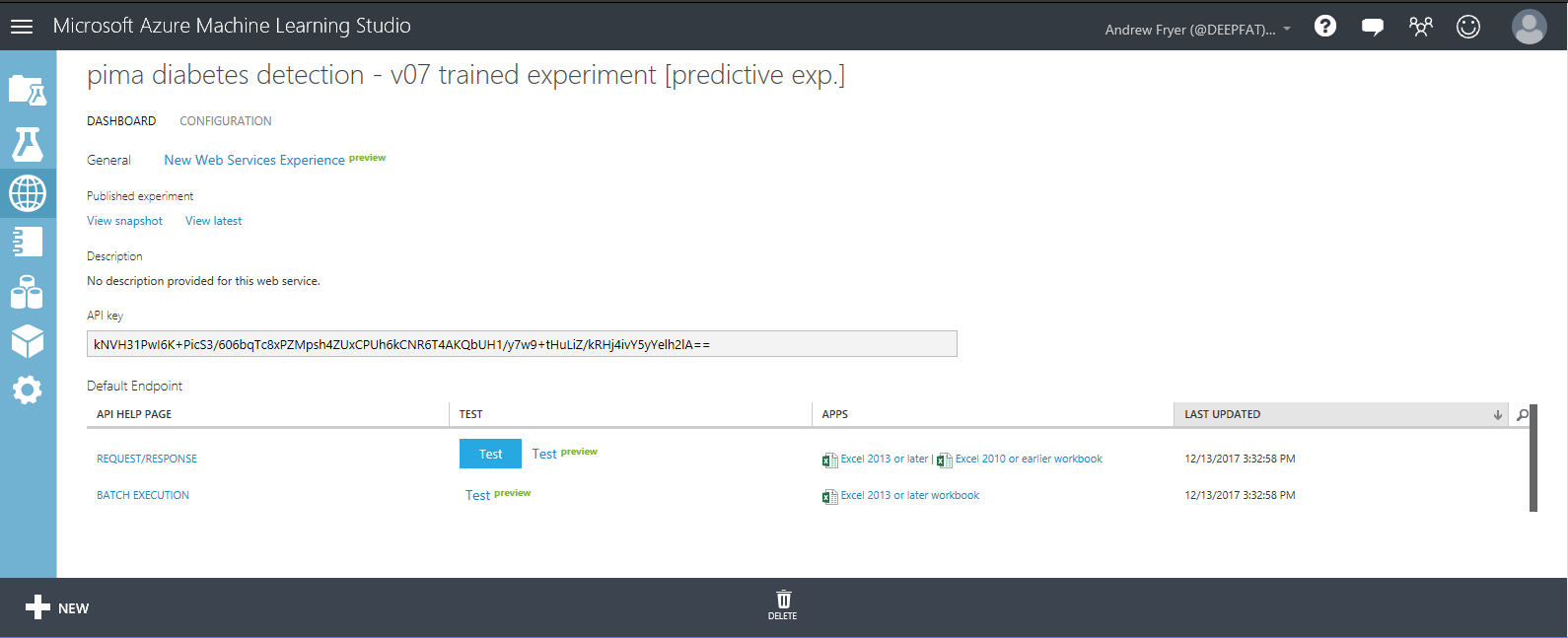
If we now think about what fields we want to return to the application or web site that will call our web service we are creating, then all we need are the scored labels (the prediction) and scored probabilities (the probability of the prediction being true). So we should add another ‘Select columns in a Dataset’ module as shown to just return those fields (in the columns selector) :



We must now run this predictive experiment as this allows ML to validate the predictive experiment before we can publish it as web service. After a successful run we’ll see that the deploy web service icon is available on the bottom toolbar



All we need to do is click on it and select deploy web service (classic). After a few seconds we’ll see we are taken to the web service section of ML studio and our new web service is displayed:



We can see some general information about the WEB API like the API key which is will be used for authorization purposes. There are also hyperlinks to help pages for the two endpoints of the service, a Request/Response endpoint and a Batch Execution endpoint. Notice we are also given some Excel spreadsheets where we can test the service as well.

On the Dashboard tab click on the button alongside the REQUEST/RESPONSE API as a quick sense check to see if our web service is working as expected.

Enter the following values in the Enter data to predict dialog:

* **Pregnancies =** 4
* **Glucose Concentration =** 95
* **BloodPressure =** 64
* **TricepFoldThickness =** 0
* **SerumInsulin = 0**
* **BMI =** 32
* **DiabetesPedigree =** 0.2
* **Age** = 40

The result you receive back after processing returns a JSON like output.

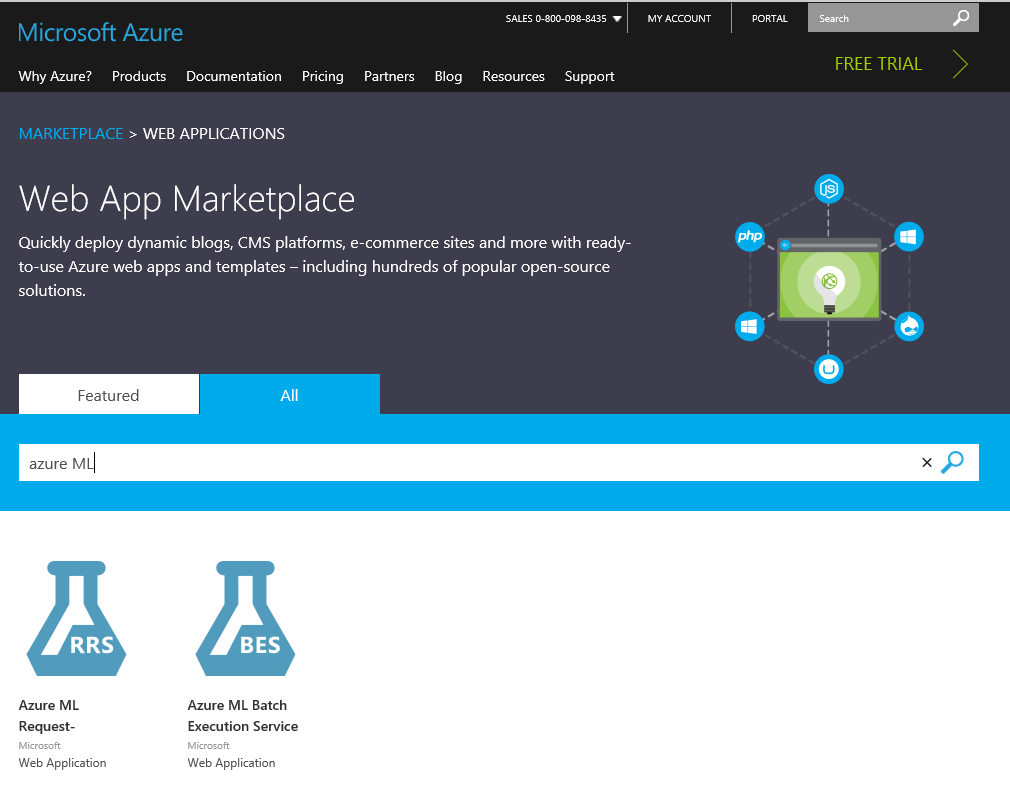


Notice we just get back the Scored Label and the Scored Probabilities. So in the case below we can say that given the parameters above the classifier has predicted the flight will be more than 15 mins delayed (1) and it is very confident in its decision (0.997861325740814).

Optional Exercise – create a Web Site using the api

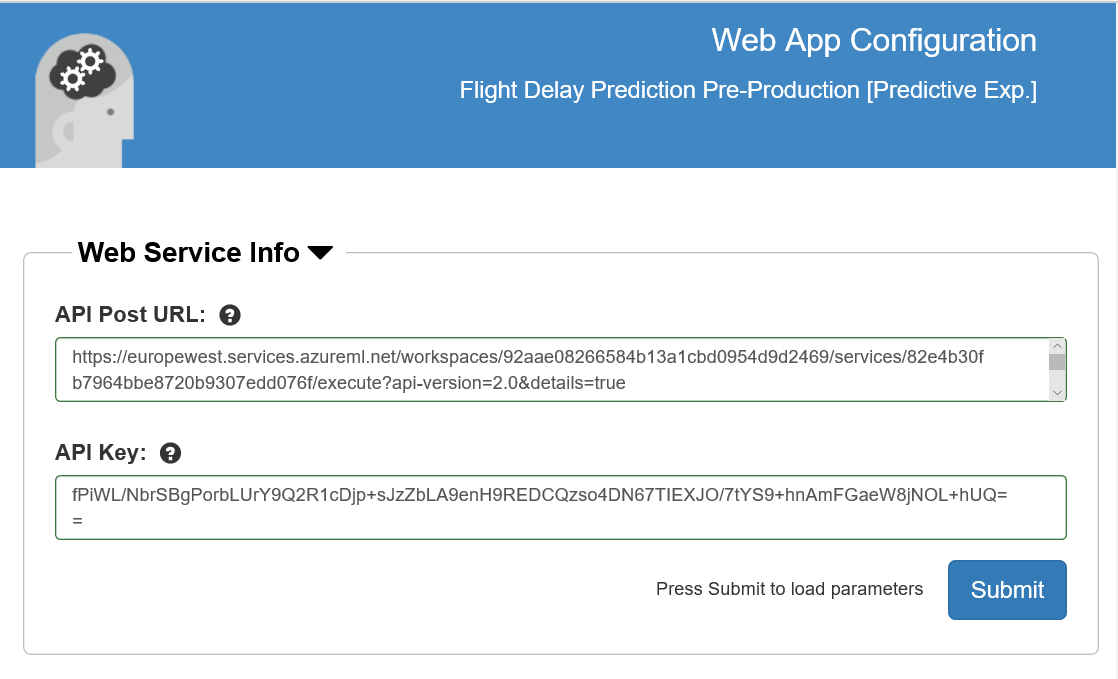
We can also test our new API using a partially configured web site in the [Azure Web App Marketplace](https://azure.microsoft.com/marketplace/web-applications/all/). Simply go to the site and look for Azure ML.

This will create a web app in our Azure subscription. You will need an Azure subscription for this.

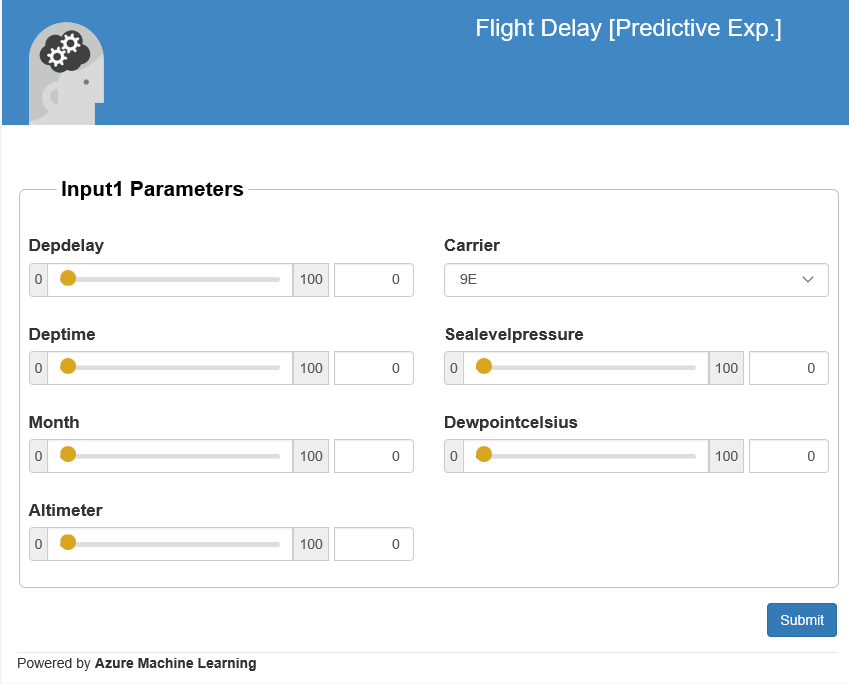




If we then click on the URL of the new site we’ll be presented with a simple page where we can enter our API URL and API key.



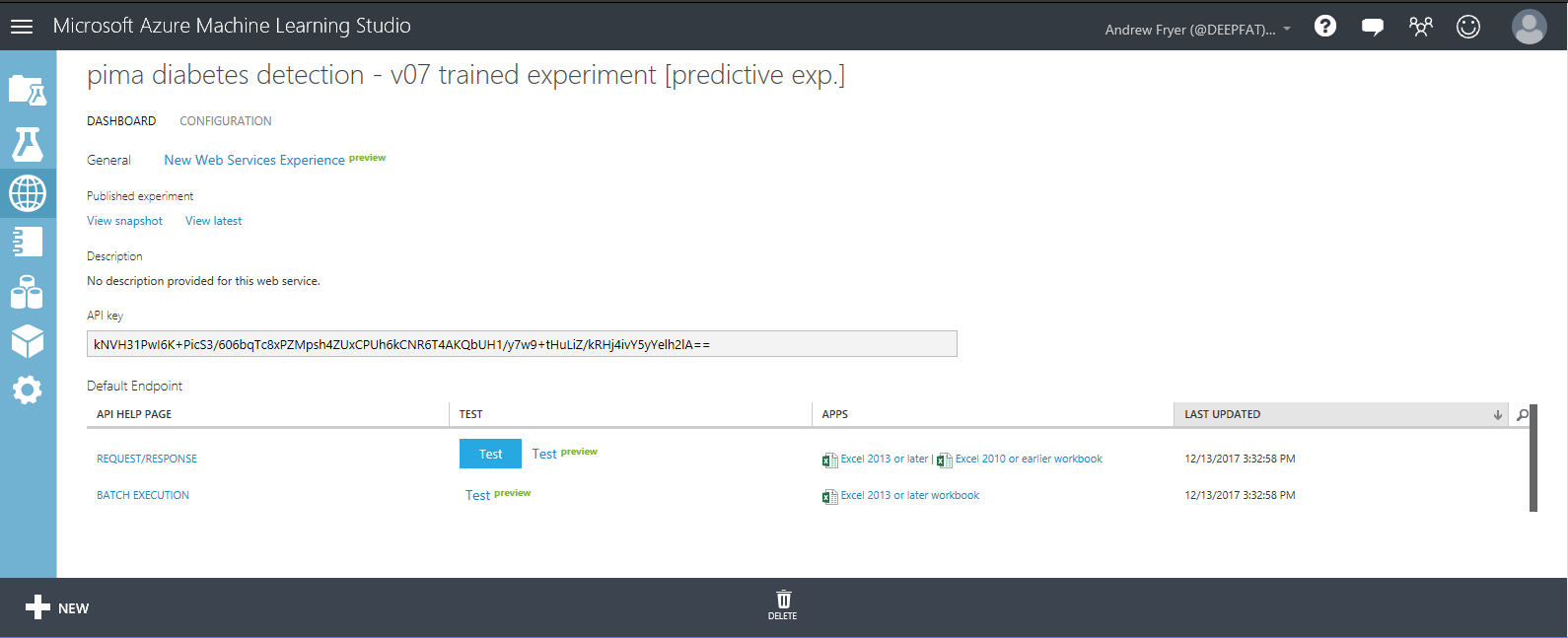
The key is on the API dashboard and the URL is at the top of the Request response page. Click Submit and close the page. Go back to the Azure Portal and open the site again and enter some trial values..



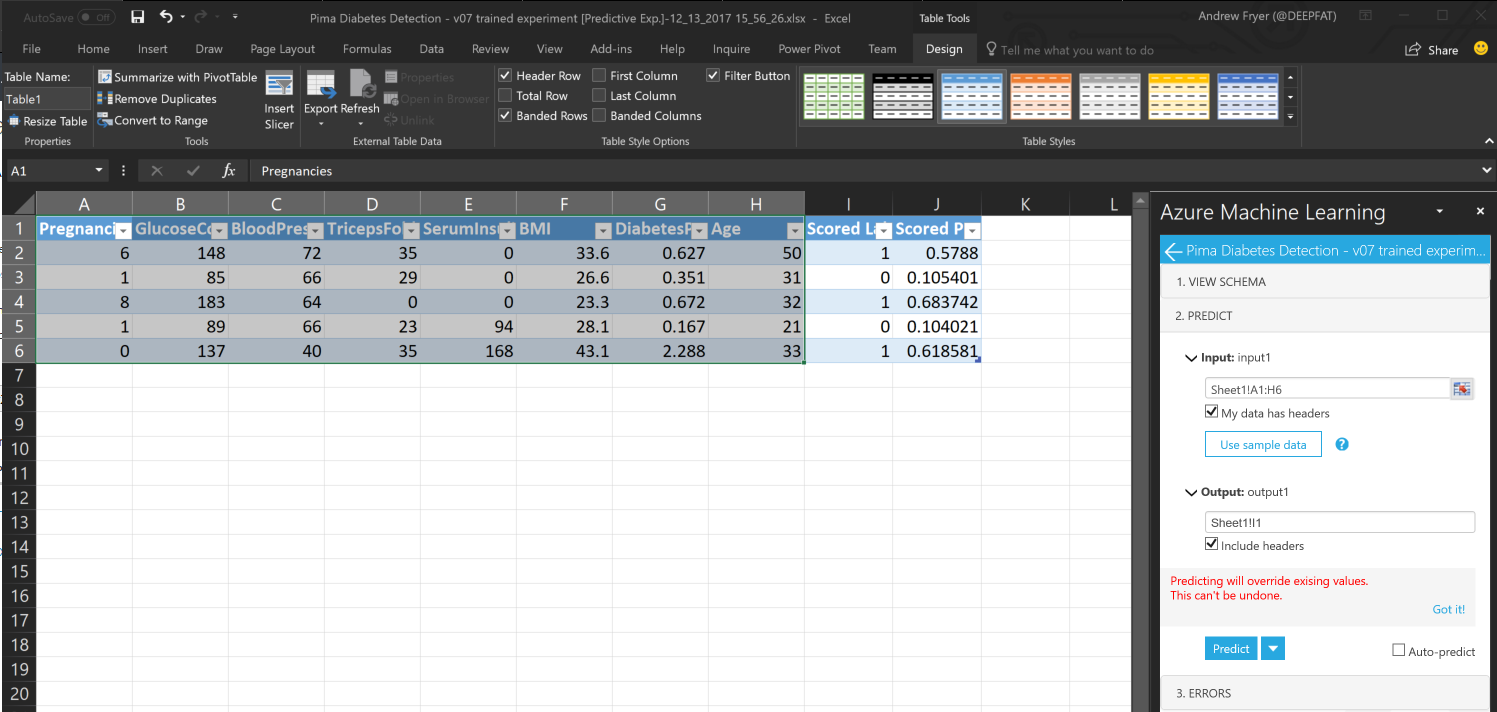
Optional Exercise – using Excel to call the API

We can also see how we can interact with the new API from Excel, if you have Excel on your machine. Below is what the Excel 2013 version looks like which uses the new Excel add-ins to automatically setup a connection to the API we have and also allows us to use sample data to test.

From the web services page click on the web service and select the right Excel version for your laptop



Accept the warning and open the spreadsheet once it’s downloaded. We need to enable content to load the Azure ML add. Once that appears we can click on the Use sample data box in the Azure Learning pane on the right. Select the sample data as Input rows and I1 as the Output and click Predict. You should see the new columns for Scored Labels and Scored Probabilities like this:



Now our sample data has the scored labels and scored probabilities against it and we could now enter our own totally unseen data to test the model.

TruePos = float(TruePos)

FalseNeg = float(FalseNeg)

FalsePos= float(FalsePos)

TrueNeg= float(TrueNeg)

Accuracy = (TruePos + TrueNeg) /(TruePos+ TrueNeg + FalsePos + FalseNeg)

Recall = TruePos /(TruePos + FalseNeg)

Precision = TruePos /(TruePos+ FalsePos)

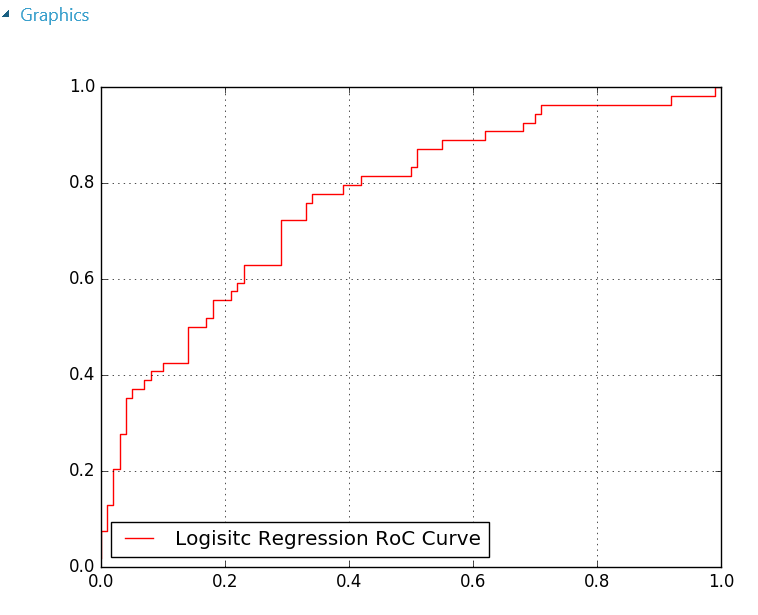
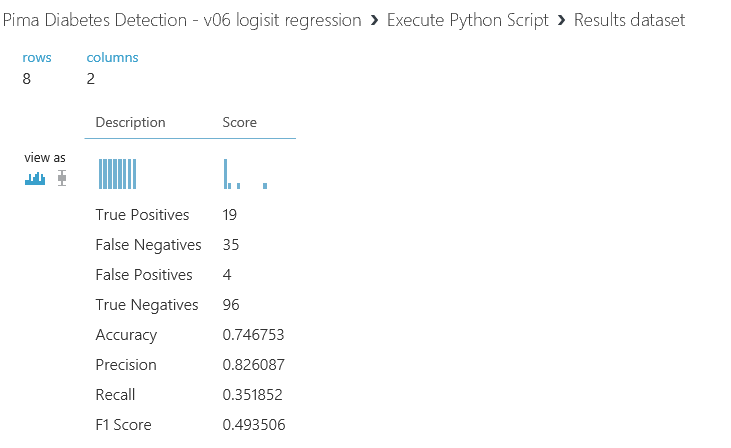
F1Score = 2 \* (Precision \* Recall)/(Precision + Recall)

data = {'Description': ['True Positives','False Negatives','False Positives','True Negatives','Accuracy','Precision','Recall','F1 Score'],'Score': [TruePos,FalseNeg,FalsePos,TrueNeg,Accuracy,Precision,Recall,F1Score]}

dataframe1 =pd.DataFrame(data,columns=['Description','Score'])

return dataframe1,

where the confusion matrix will come out of port 1 of the Python module and the plot will be in port 2:



This is not a very accurate model in that the true positives are very low compared to the errors – where the model predicts a Pima Indian has diabetes where they don’t (false positives) and has wrongly identified some Indians as not having diabetes when they do.

Why is the model not doing well and what could we do to improve it.

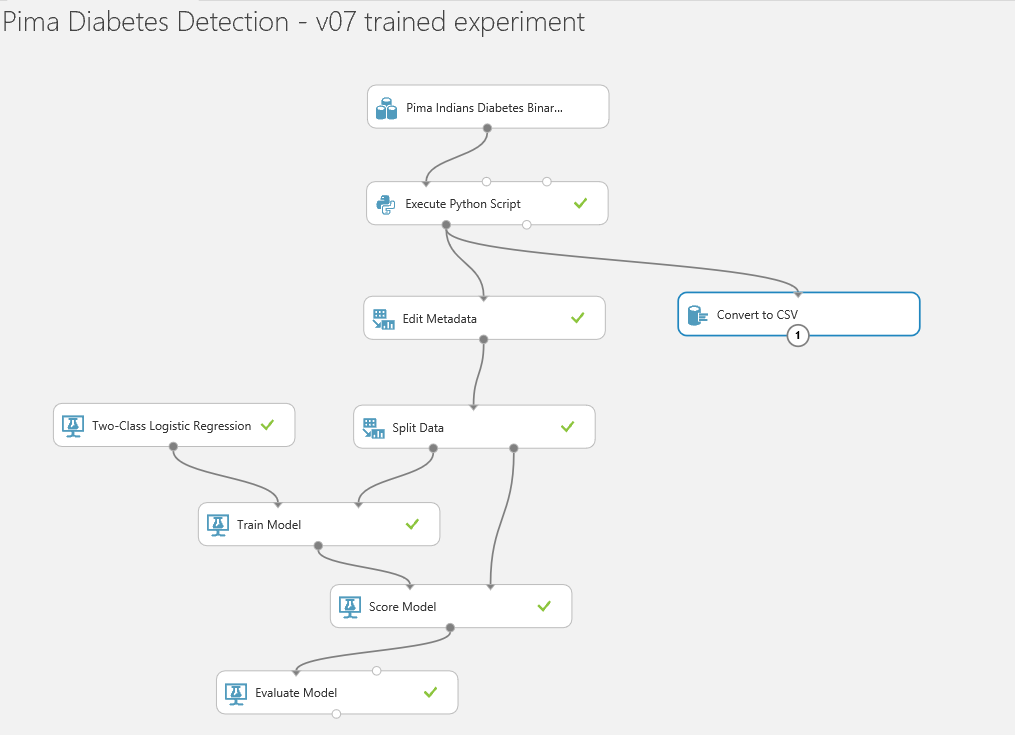
Before we review that let’s see how to publish the model assuming it is fit for production.

I

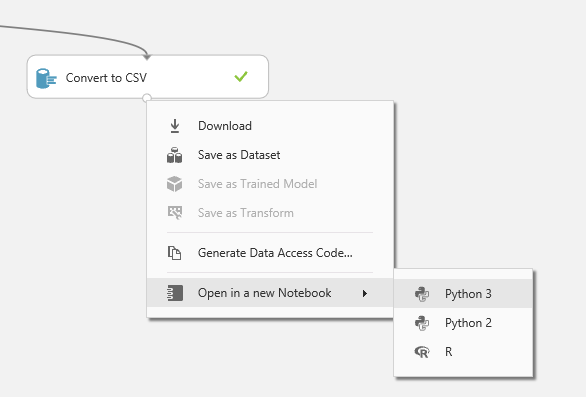
Jupyter Notebooks

Our experiment seems to be quite inaccurate, so should start to investigate the data more closely (in fact we should really have started by doing this!). We saw at the start of this lab that we can get basic statistics just by visualising the data and there are also statistical modules in ML Studio to do this. However, we might wish to bring our own tools to bear and a great way for those with Python or R skills to do this is with Jupyter Notebooks. These are ad hoc scripts which we can attach to our experiments or use on their own.

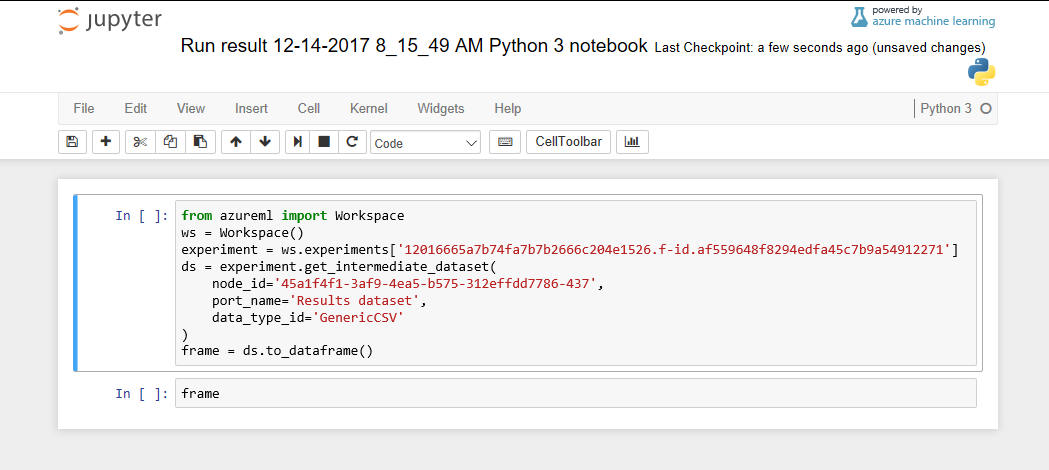
To use these notebooks, we need to extract the data to a format (.csv) that the notebooks can use and this can quickly be done with the Convert to CSV module which we’ll connect to the output of the Execute Python Script module as shown:

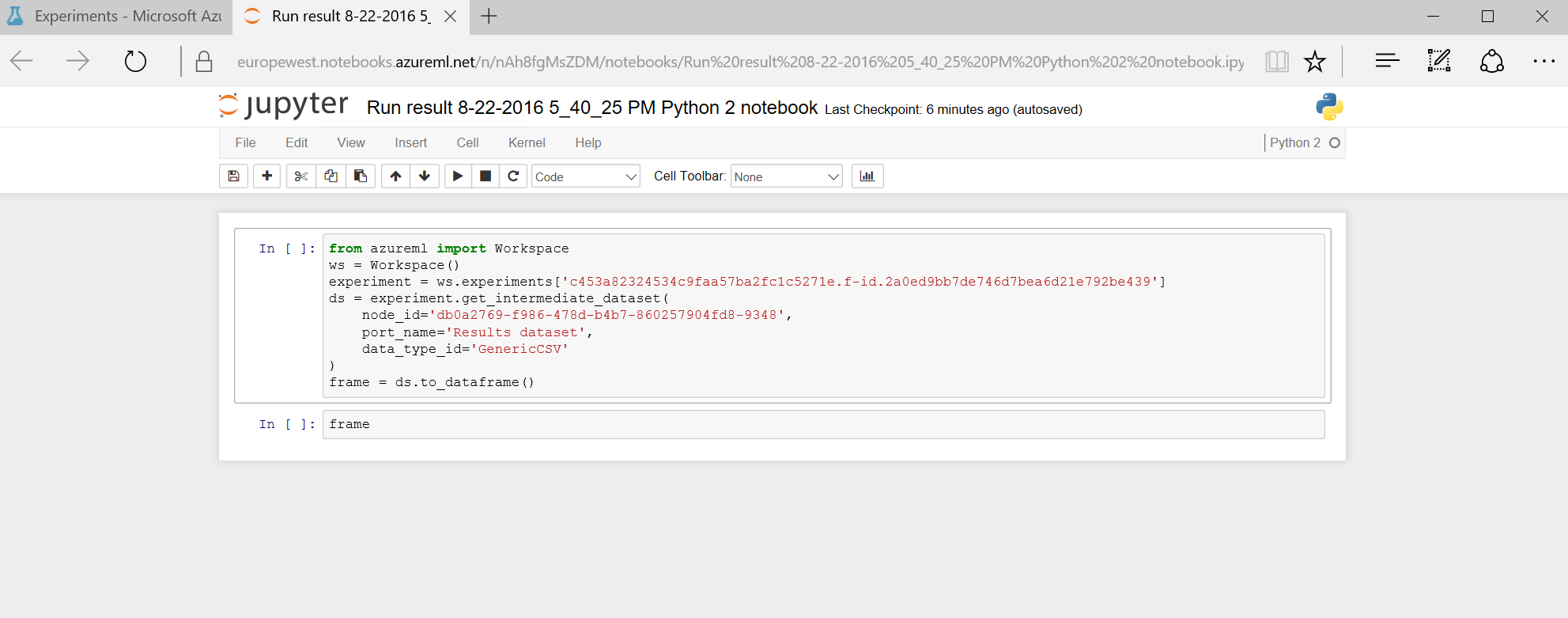
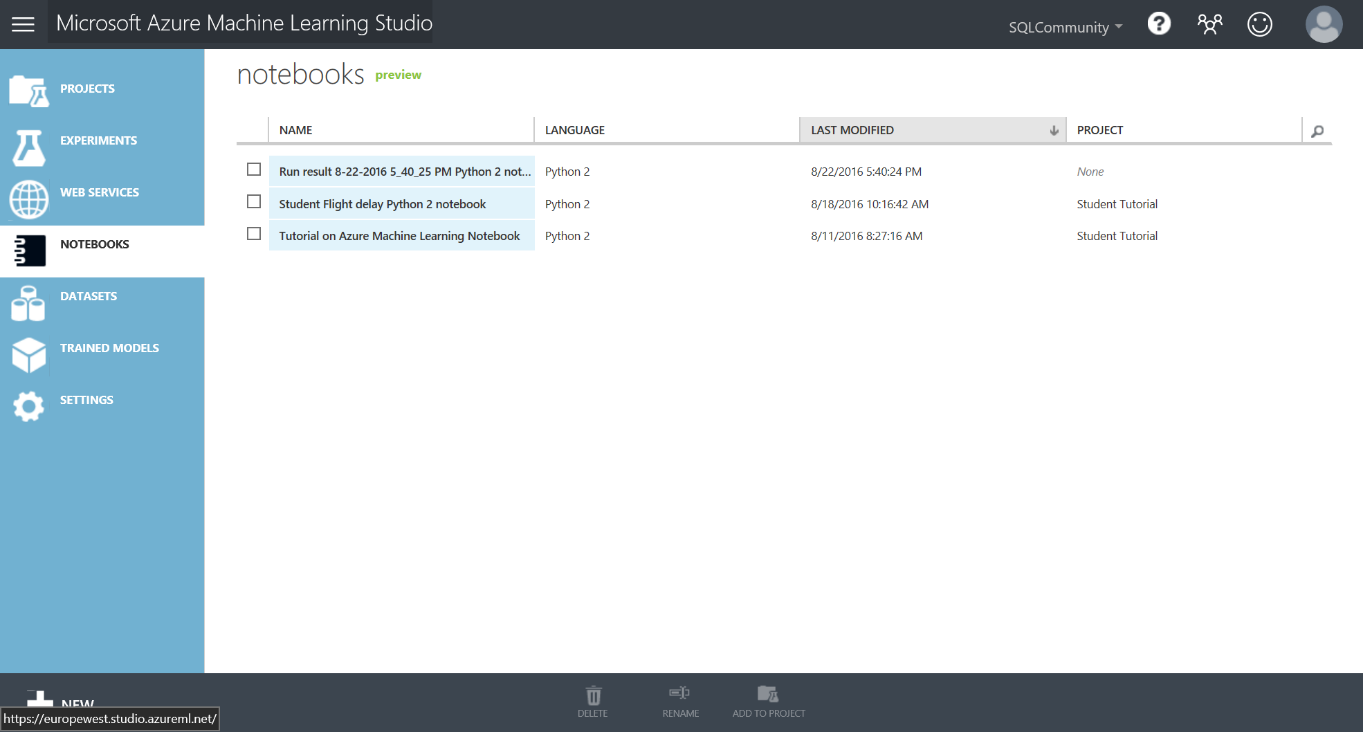


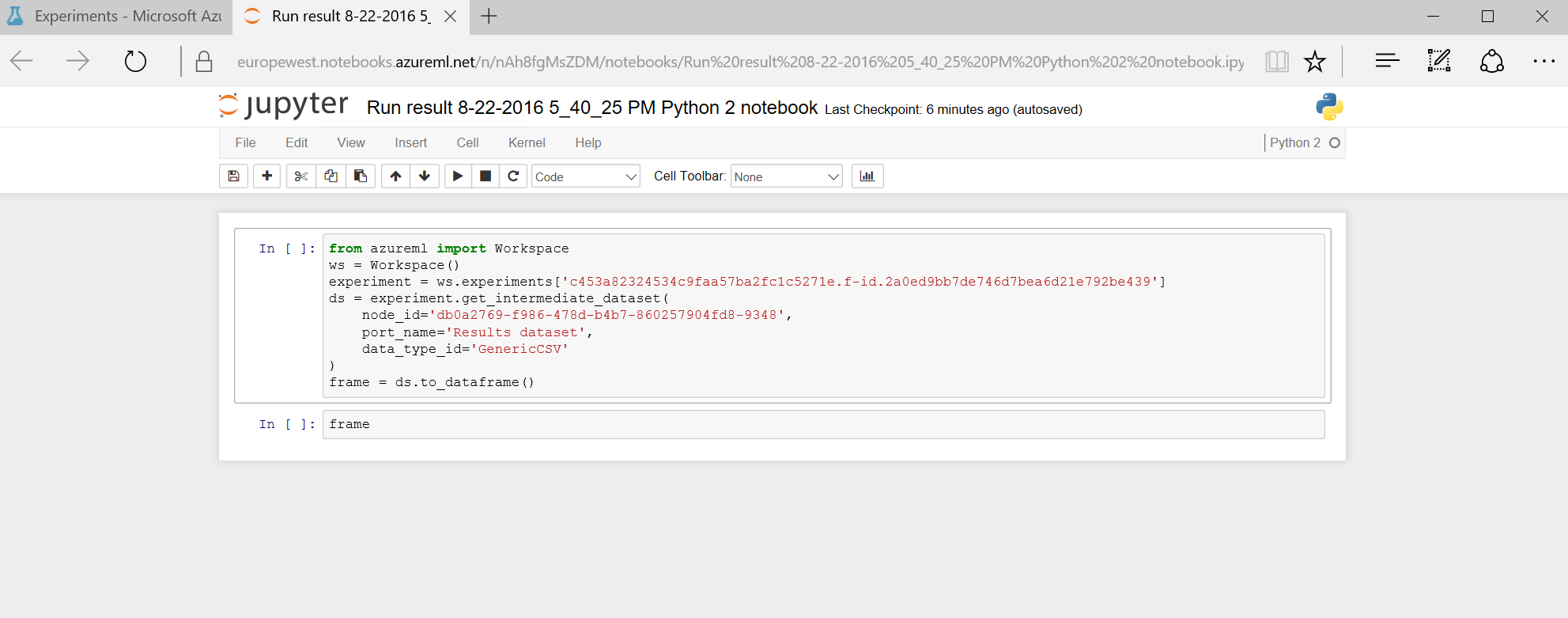
Right click on the Export to CSV module and click Run Selected to just run that module. Once it has finished right click again and select Open in a New Notebook -> Python 3



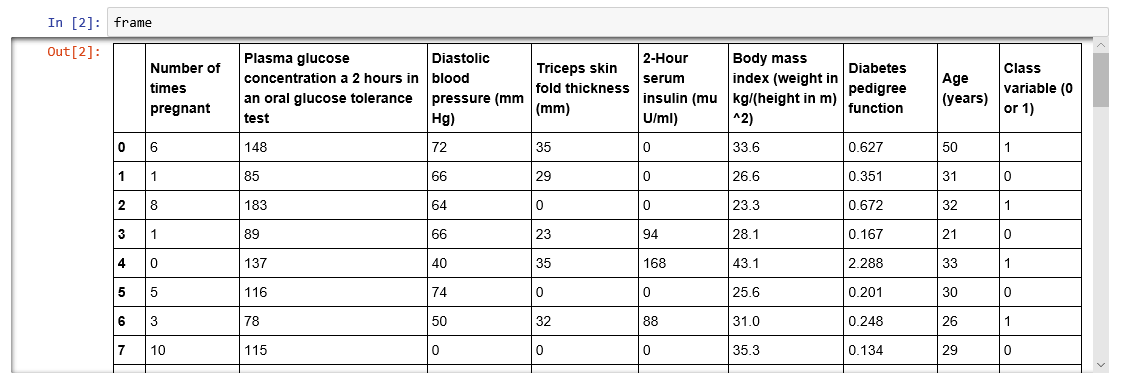
We’ll now be taken to a new browser window with our Notebook in:



Notice we already have a stub script in it which takes our data and loads it into a dataframe called frame. The first thing we should do is rename our notebook to something meaningful as this notebook will persist in our ML workspace. So rename it click on save and confirm this by going back to the browser tab for ML studio and clicking on the Notebook icon on the left (it’s not yet renamed in the screenshot below):

We can now go back to the Notebook tab and add some of our own code. Firstly, we need to run the cell with the existing code in to load the dataframe with data from our experiment which we do by setting focus on the cell and clicking on the run icon in the top toolbar.

In the second cell there is just the command frame and if we run this cell we’ll see a sample of our data:



What we can do now is use any of the many libraries and function in Python to analyse this data without any further effort. In this case let’s see how the various features correlate with each other. This plot only works against numerical data and fortunately all our data is numeric. Add a new cell to the notebook (with the **+** sign) and enter this:

%matplotlib inline

import pip

import pandas

import numpy as np

import pandas as pd

import matplotlib

matplotlib.use("agg")

import matplotlib.pyplot as plt

#Now we have these libraries we can create the plot itself:

cm=correlationFrame.corr()

fig=plt.figure()

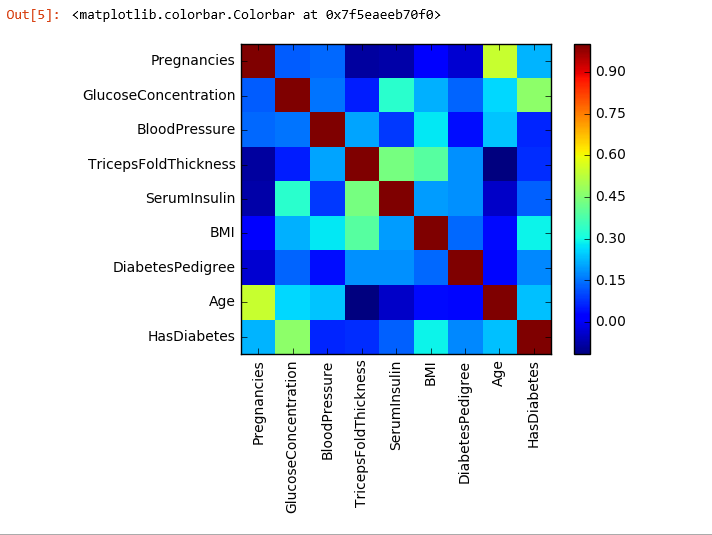
plt.imshow(cm,interpolation='nearest')

plt.xticks(list(range(0,len(cm.columns))),list(cm.columns.values), rotation=90)

plt.yticks(list(range(0,len(cm.columns))),list(cm.columns.values))

plt.colorbar()

If we run this cell we’ll get a plot like this:



This shows us is how the various columns in our dataset are correlated. For example the correlation on the leading diagonal is always one because this simply a comparison of a column with itself.

What we are interested is how each of these columns correlates with HasDiabetes (the yellow box) and here we can see there isn’t a strong correlation between any of the features and the label, the strongest being GlucoseConcentration and the weakest BloodPressure and TricepFoldThickness.

What this does show is that is a simply matter to use Jupyter notebooks against any step in our experiments to gain a deeper insight into our data

Conclusion

This lab was intended to introduce you to the basic concepts of Machine Learning such as binary classification, feature selection, training and testing a model and using Azure Machine Learning. A web service was created to operationalize and deploy the model for production.

**Next Steps:**

* Check out the UK Data Developer page on MSDN here:
* Check out the Azure ML Gallery and download and edit/run other types of machine learning experiments: <https://gallery.azureml.net/>
* Also check out other Azure Services that can be used with Machine Learning such as HDInsight, Data Factory and Stream Analytics: <http://azure.microsoft.com/en-us/services/>