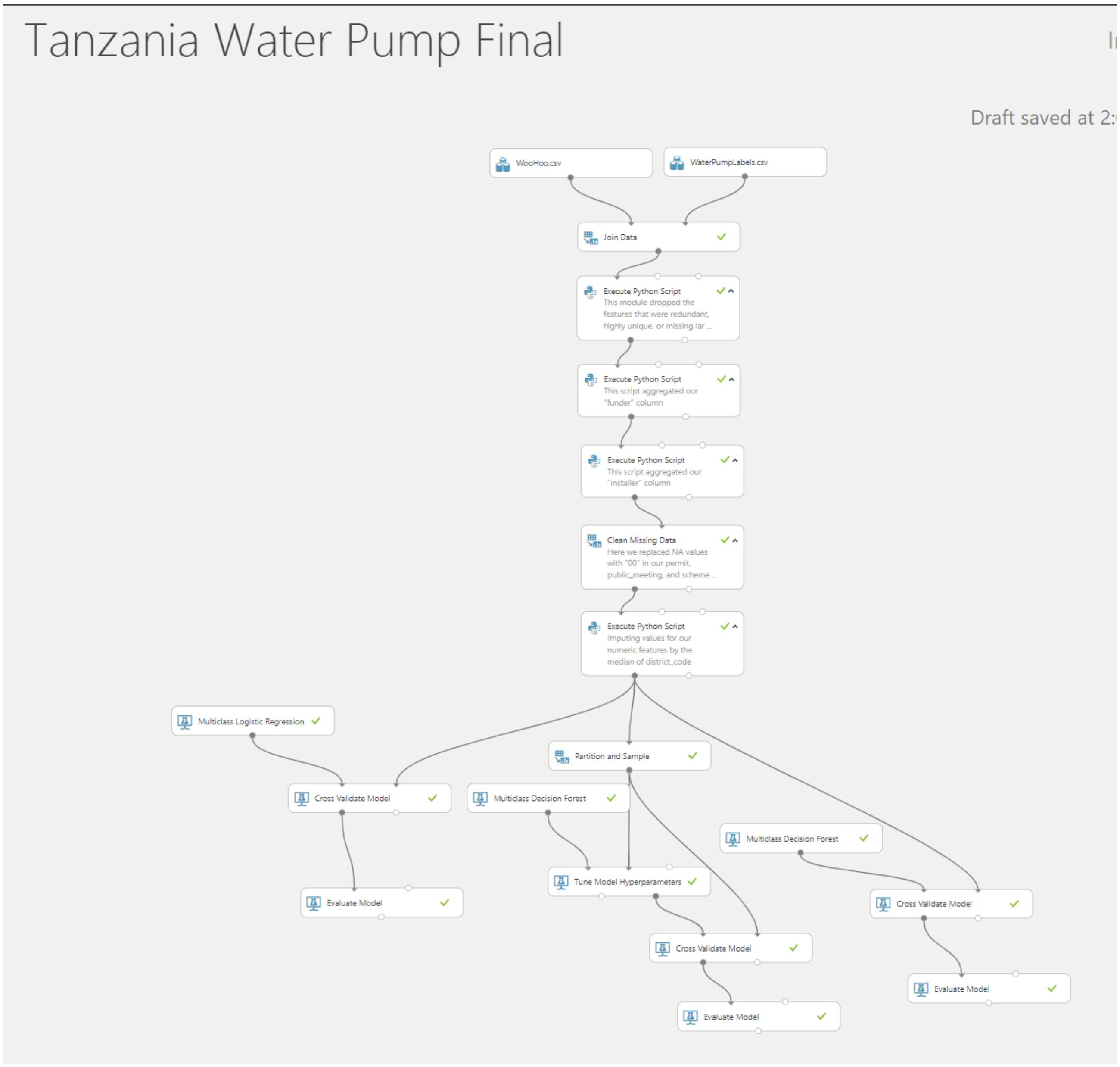
Sam Cannon



This module dropped the features that were redundant, highly unique, or missing lar ...

This script aggregated our "funder" column

This script aggregated our "installer" column

Here we replaced NA values with "00" in our permit, public_meeting, and scheme ...

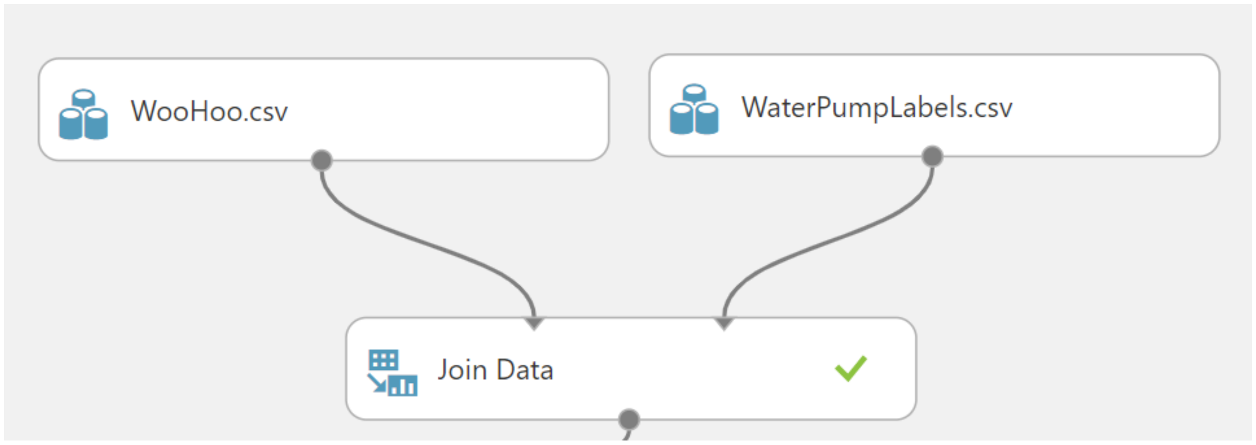
Imputing values for our numeric features by the median of district_code

Summary

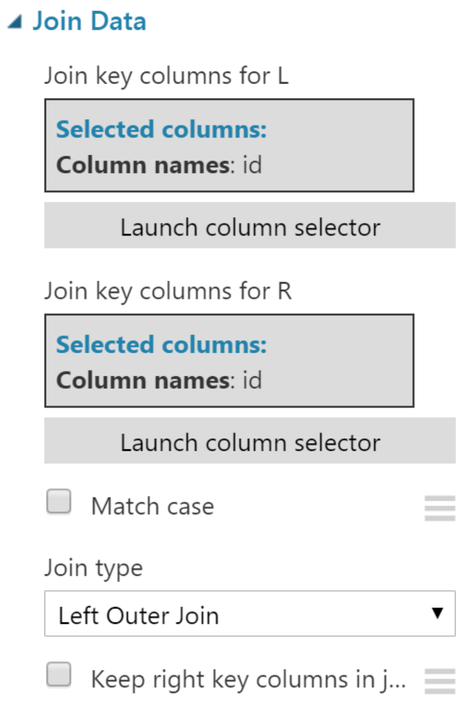
This was a project centered around the "Pump it up" DrivenData data science competition. The aim of the project was to predict the functionality of water wells in Tanzania based on a number of features provided by DrivenData.

Data Import

The project began by importing the data into Azure Machine Learning Studio. I imported the training set without the labels and the label set. I then uploaded the data into my Azure workspace from my local machine. From there, I had to join the label data to the training data, which we accomplished through using the Join Data module within Azure ML.



Within the Join Data module, I had to specify a few parameters so that my data would join properly.


Join Data

Join key columns for L

Selected columns:
Column names: id

Launch column selector

Join key columns for R

Selected columns:
Column names: id

Launch column selector

☐ Match case

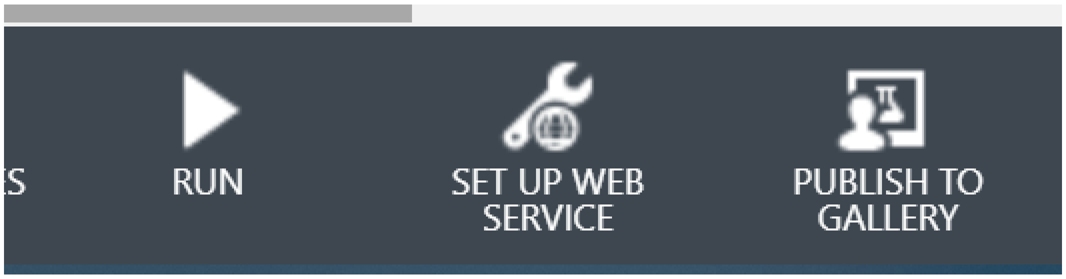
Join type

Left Outer Join

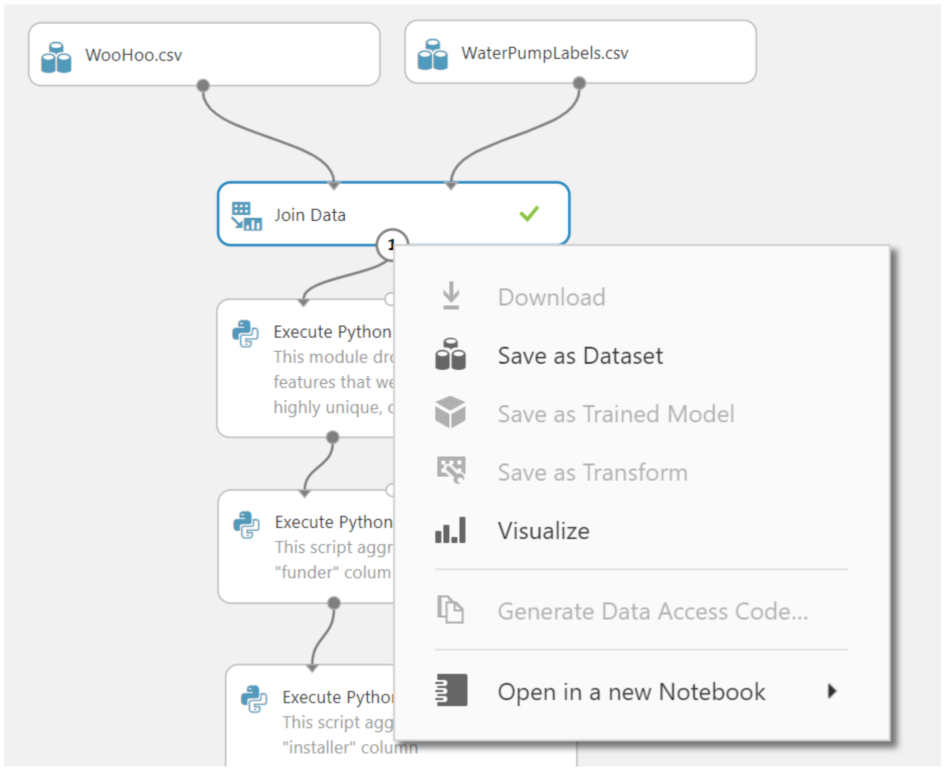
☐ Keep right key columns in j...

Here we are specifying the column that we want to join my data on. Because both my training set and my label set include an "id" column, I joined my data on this feature. Notice, I had to launch the column selector within each dataset (L and R) and specify the "id" column as my join column. I then had to clear the "matched case" and "keep right key columns" boxes so that it would be a clean join with the id column not appearing again.

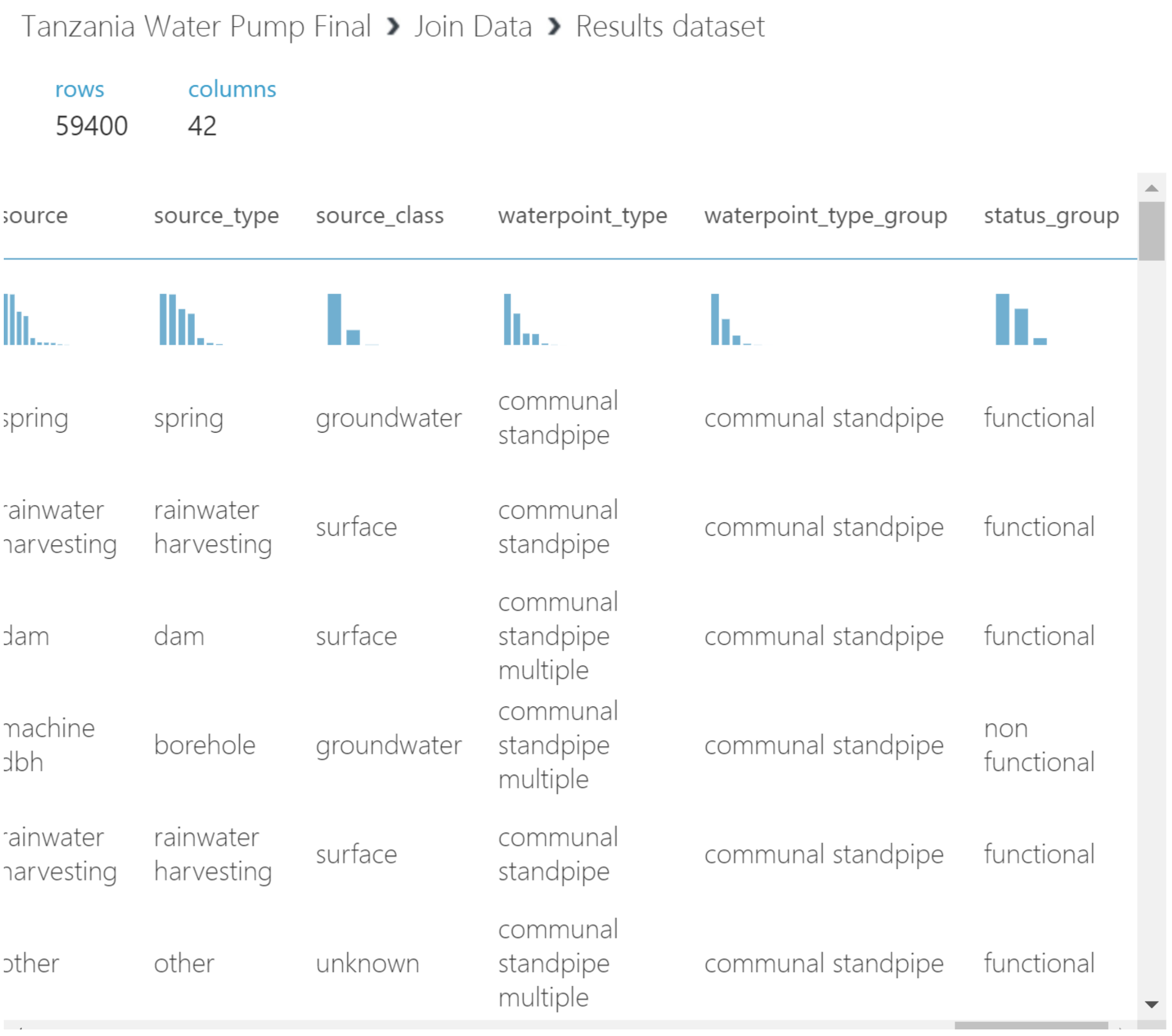
Now I must run the experiment (which I must do every time I add a new module) by clicking on "Run" at the bottom of my screen, if my module runs properly, then a green check mark appears within the module, as I can see in my Join Data module above.



In order to ensure that my import and join worked the way that I wanted it to, I can right click on the output node of the Join Data module and select the Visualize button to visualize my data.



And I see this…


Tanzania Water Pump Final › Join Data › Results dataset

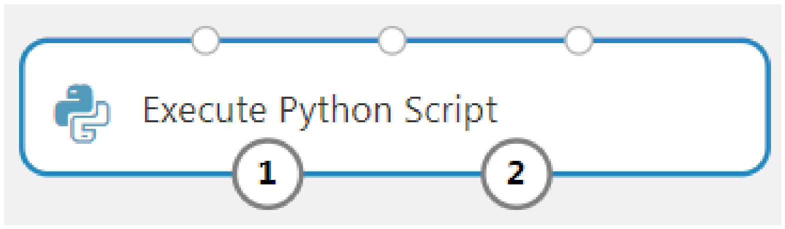
rows 59400 columns 42

ource	source_type	source_class	waterpoint_type	waterpoint_type_group	status_group
pring	spring	groundwater	communal standpipe	communal standpipe	functional
ainwater harvesting	rainwater harvesting	surface	communal standpipe	communal standpipe	functional
lam	dam	surface	communal standpipe multiple	communal standpipe	functional
nachine lbh	borehole	groundwater	communal standpipe multiple	communal standpipe	non functional
ainwater harvesting	rainwater harvesting	surface	communal standpipe	communal standpipe	functional
other	other	unknown	communal standpipe multiple	communal standpipe	functional

Which shows my label column “status_group”, and the notation that I have 42 columns with 59400 rows, which is exactly the shape of my data. This information shows us that my import and join Ire successful.

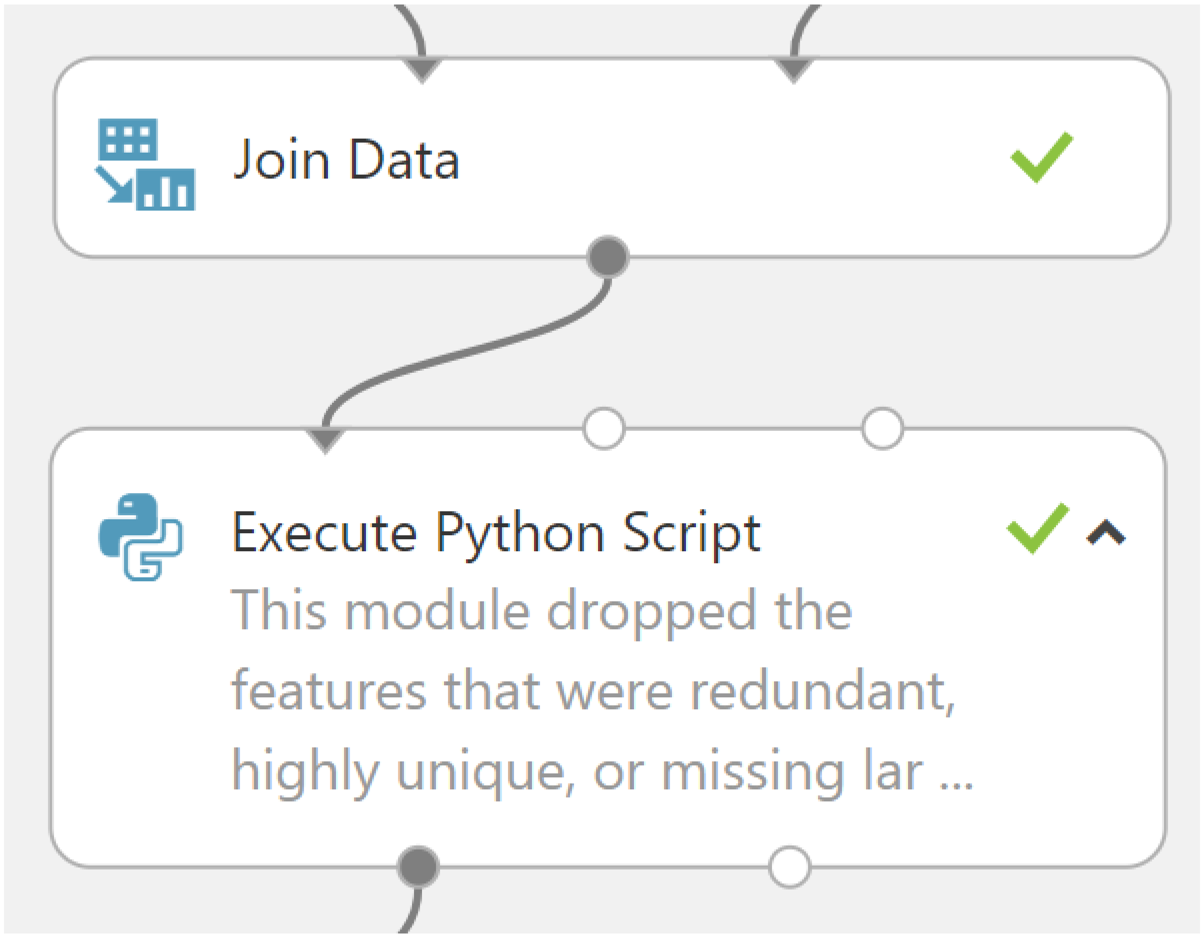
Data Preparation

Now that I have my data imported, I are ready to begin preparing the data. The first thing that I want to do is to drop the columns that are redundant, highly unique, or missing too much data. One of the most useful modules to use in Azure Machine Learning Studio for prepping data, if you do not want to join a bunch of modules together, is to use the Execute Python Script module.



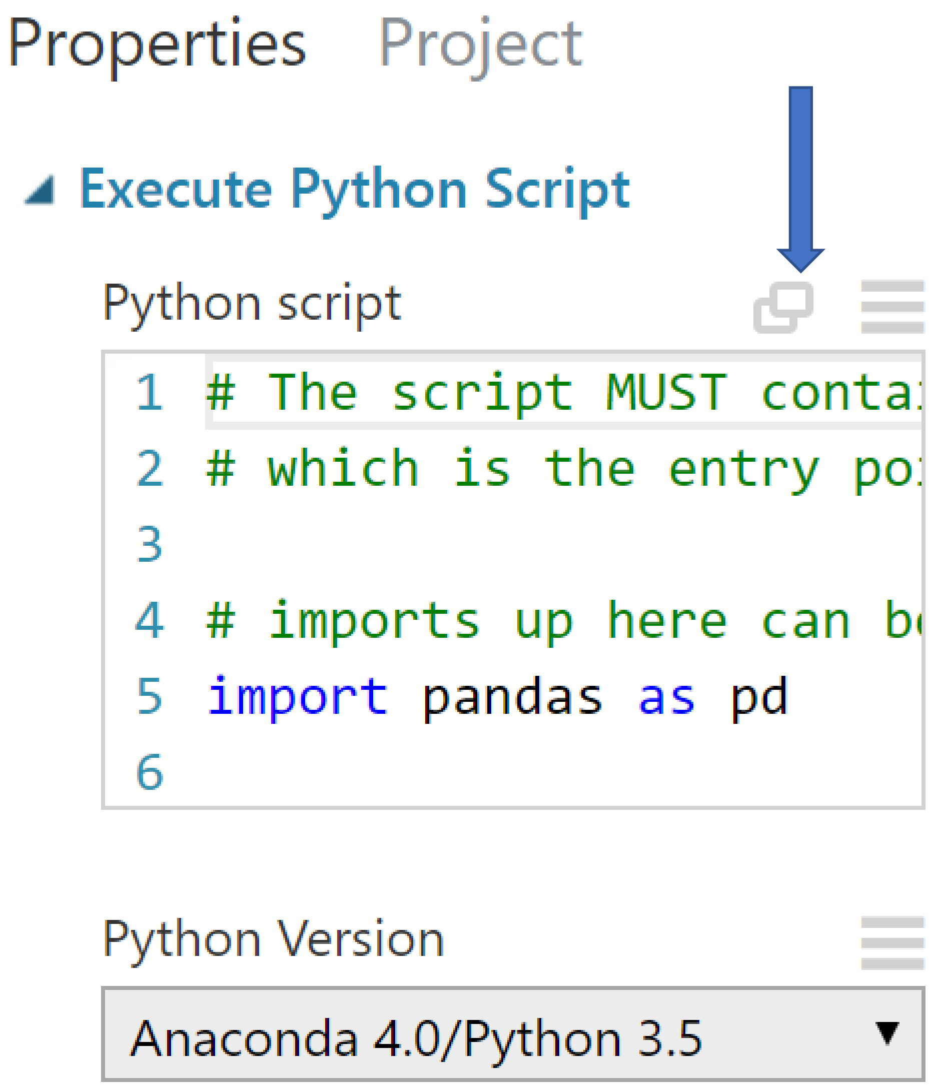
This module allows the user to enter python script into an environment within Azure ML Studio, which can cut down greatly on the number of modules used within an experiment. In addition to streamlining the process, this module allows us to transform my data in more complex ways than may be provided out of the box in Azure ML Studio.

I began by using the Execute Python Script module to drop the columns mentioned for the above reasons.


Join Data
Execute Python Script
This module dropped the features that were redundant, highly unique, or missing lar ...

I connected the output of my Join Data module to the left-most input for Execute Python Script. Once the modules Ire connected, I entered in the Python script into the Azure ML Studio scripting box.

To access the Azure ML script window, I click on my Execute Python Script module and click on the python script button indicated by the blue arrow

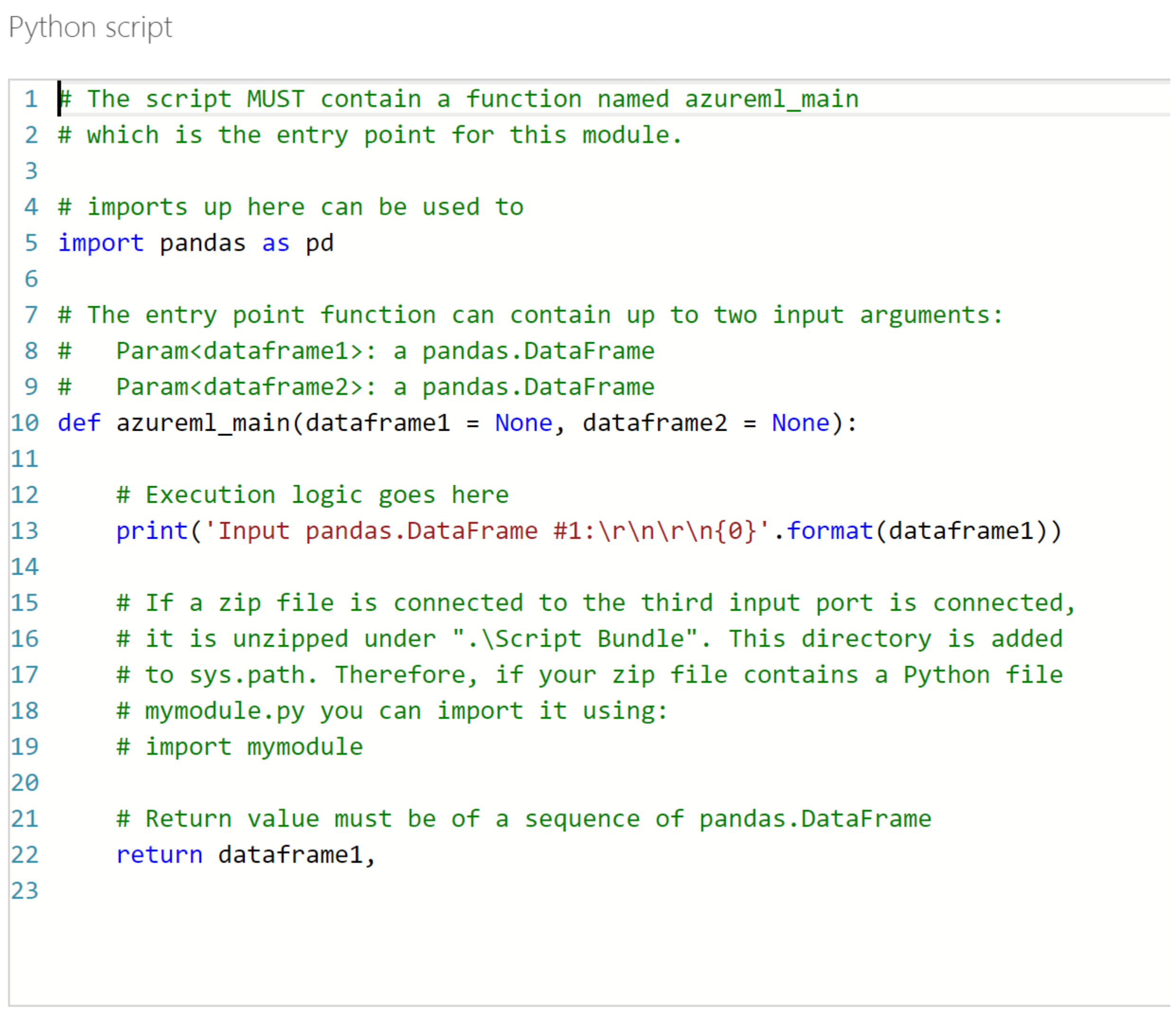


```
# The script MUST contai
# which is the entry poi

# imports up here can be
import pandas as pd

```

Anaconda 4.0/Python 3.5

Note: you must specify the version of Python that you want to use as ymy script, I want to use the most up-to-date version, so I specify Python 3.5

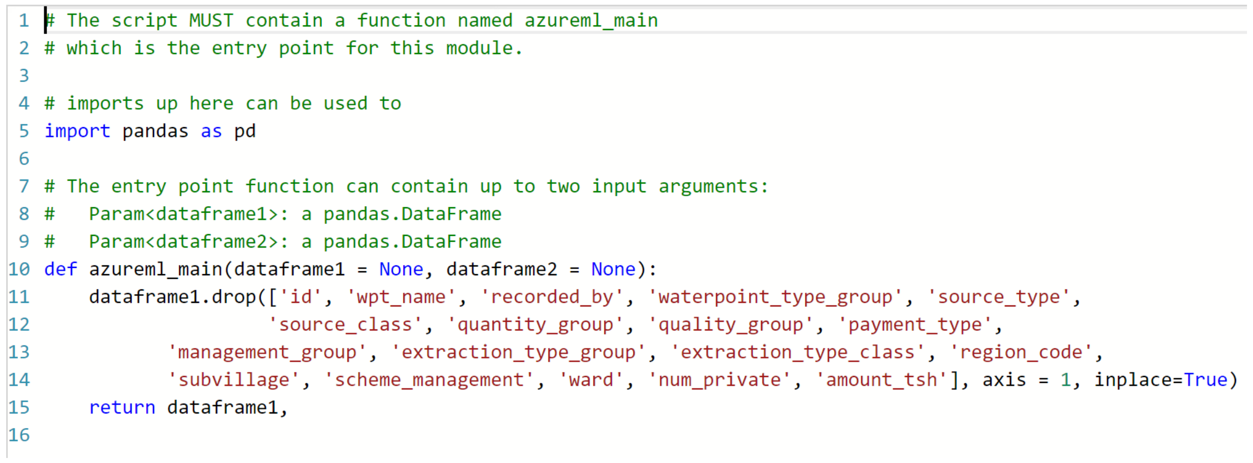
When I click on this box, I see this


Python script

```python
# The script MUST contain a function named azureml_main
# which is the entry point for this module.

# 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))

    # If a zip file is connected to the third input port is connected,
    # it is unzipped under ".\Script Bundle". This directory is added
    # to sys.path. Therefore, if your zip file contains a Python file
    # mymodule.py you can import it using:
    # import mymodule

    # Return value must be of a sequence of pandas.DataFrame
    return dataframe1,

```

What you need to know about this script box, for my purposes, is that you can import packages in the space provided, then you can run ymy script after the :

My script looks like this

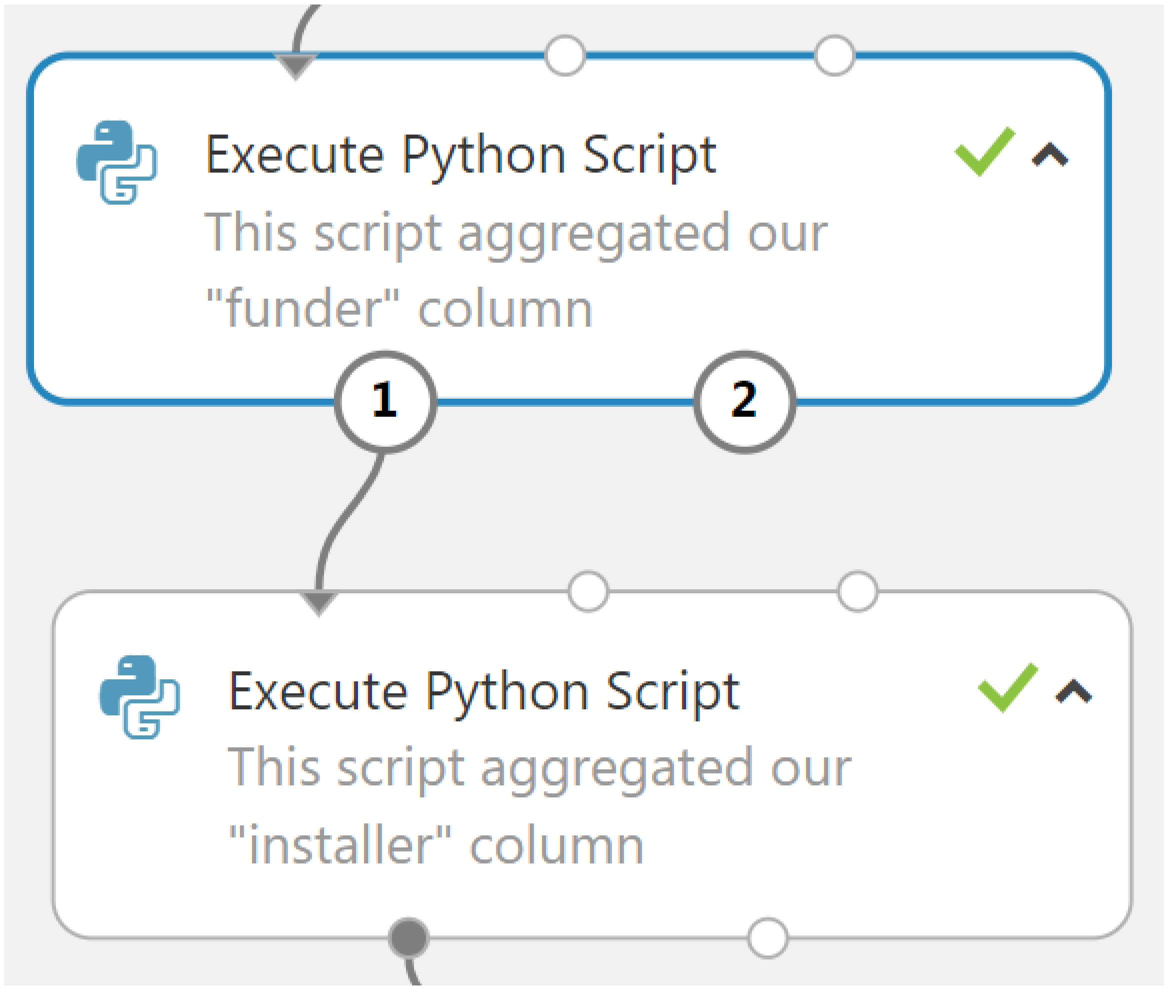


```python
# The script MUST contain a function named azureml_main
# which is the entry point for this module.

# 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):
    dataframe1.drop(['id', 'wpt_name', 'recorded_by', 'waterpoint_type_group', 'source_type',
                    'source_class', 'quantity_group', 'quality_group', 'payment_type',
           'management_group', 'extraction_type_group', 'extraction_type_class', 'region_code',
           'subvillage', 'scheme_management', 'ward', 'num_private', 'amount_tsh'], axis = 1, inplace=True)
    return dataframe1,

```

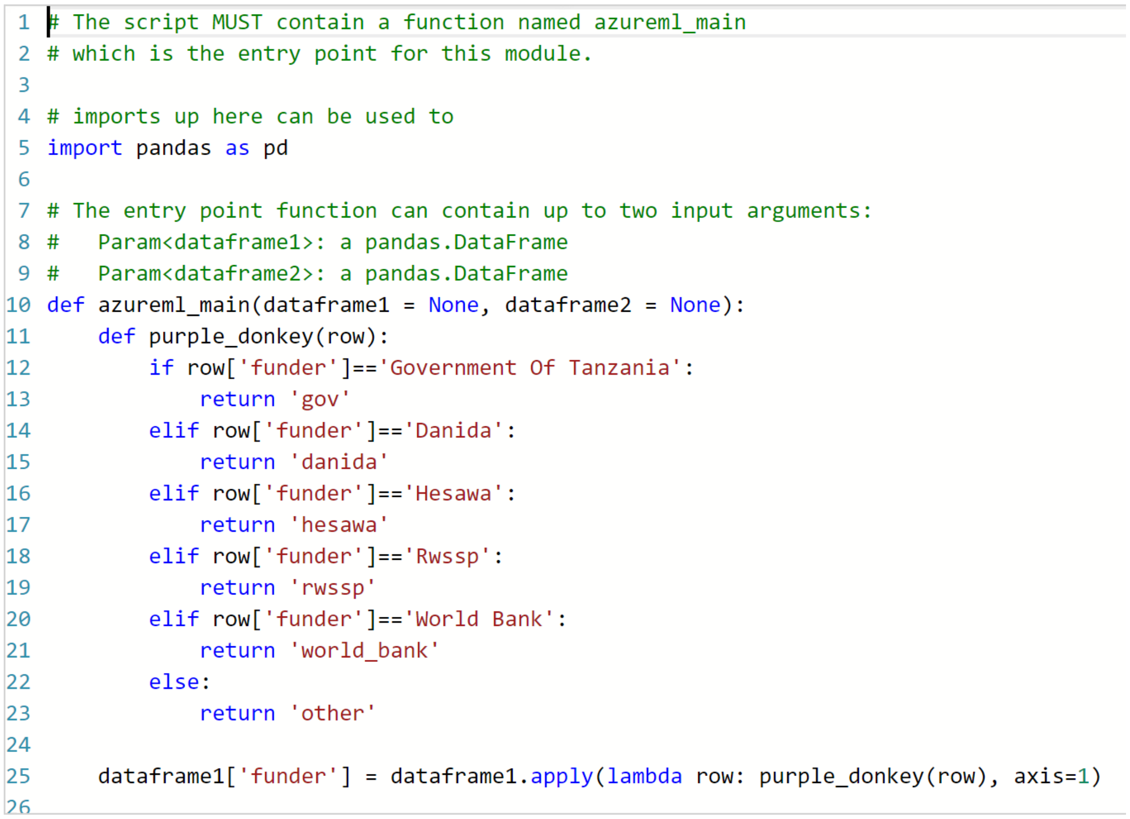
I simply wrote my python code under the azureml_main function, where my dataset was renamed as “dataframe1”, you can choose to rename this if you wish, but any data put into the Execute Python Script module will be renamed. An important note to mention here is that the indentation of the script is VERY important, ymy script will not run if it is not properly indented, underneath the azureml_main function.

Once my script is input into this window, I can close it by clicking on the check box and run the module.

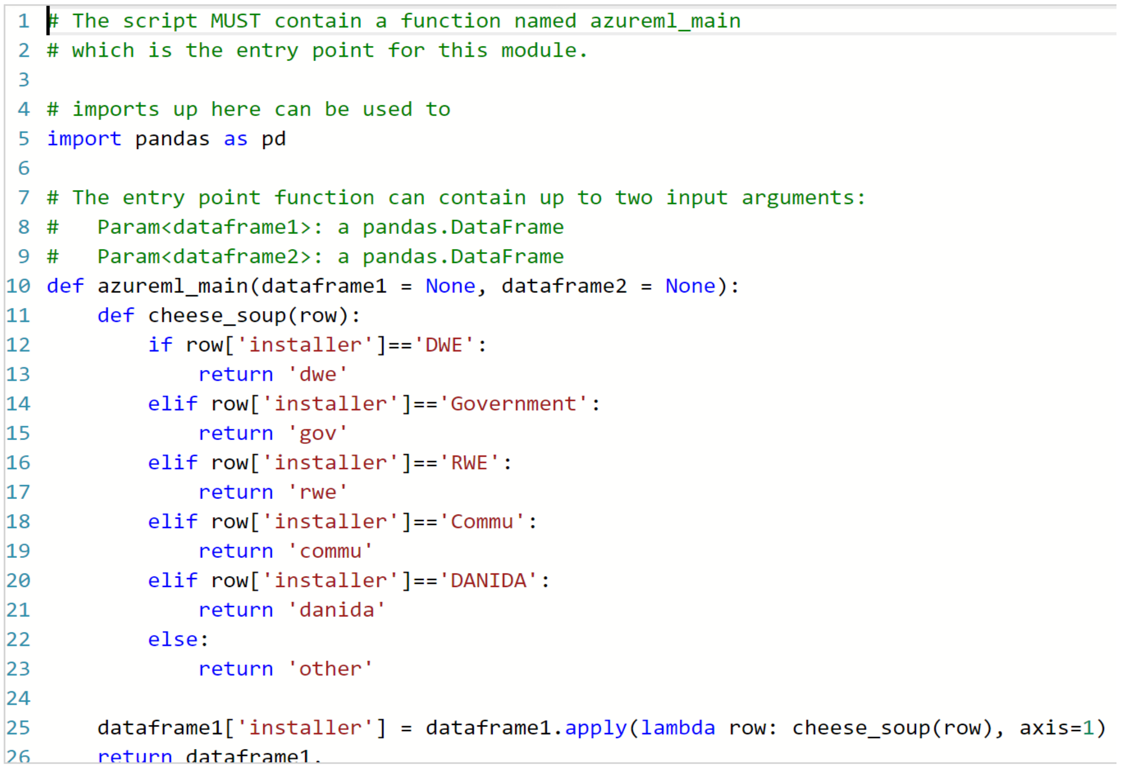
My next module that I are going to run is another Execute Python Script module, which will be aggregating my “funder” feature, the following module will do the same with my “installer” feature, exactly as it was done in my Python project.


Execute Python Script
This script aggregated our "funder" column
1
2
Execute Python Script
This script aggregated our "installer" column

My code for these modules looks like this



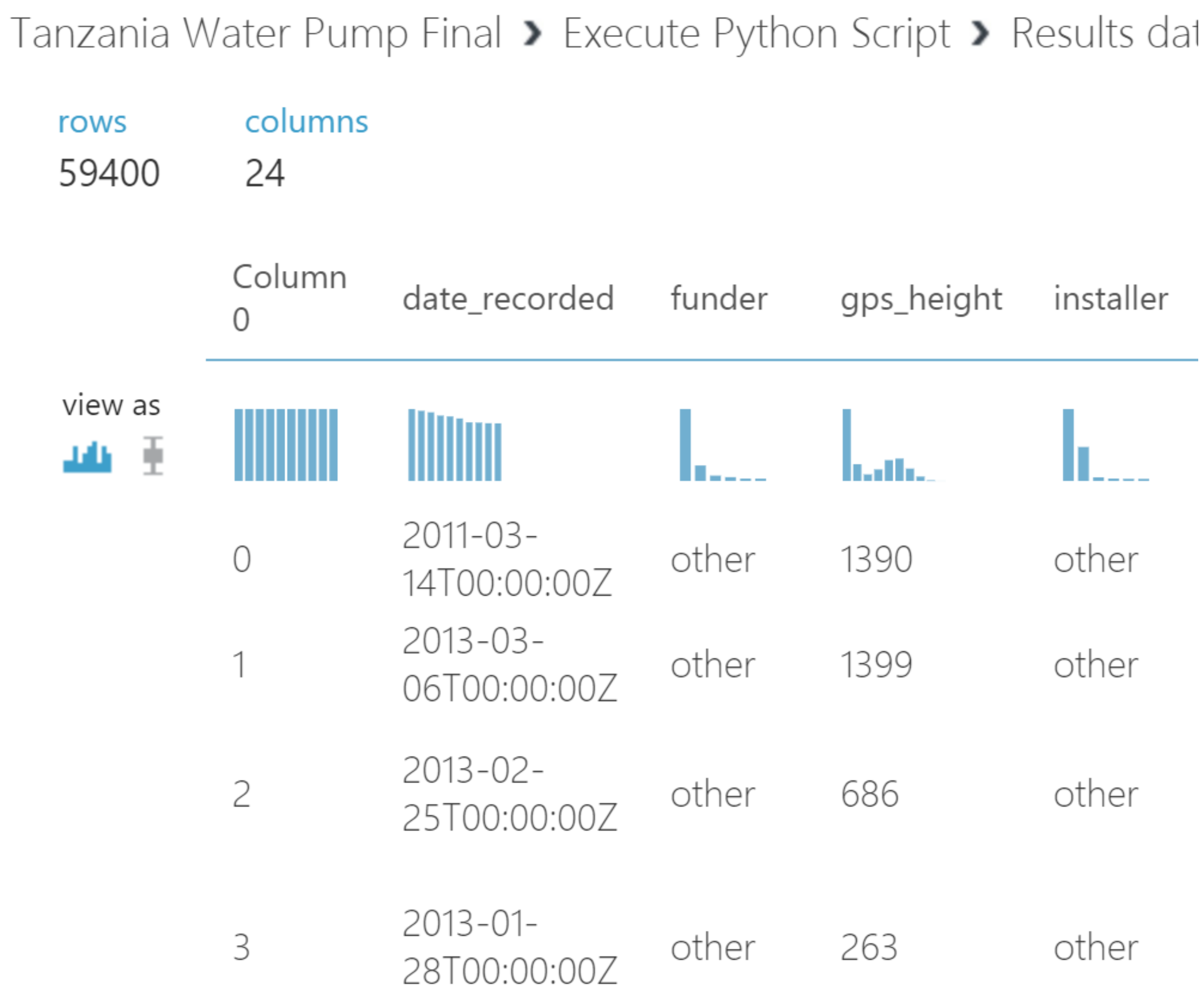
```python
# The script MUST contain a function named azureml_main
# which is the entry point for this module.

# 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):
    def purple_donkey(row):
        if row['funder']=='Government Of Tanzania':
            return 'gov'
        elif row['funder']=='Danida':
            return 'danida'
        elif row['funder']=='Hesawa':
            return 'hesawa'
        elif row['funder']=='Rwssp':
            return 'rwssp'
        elif row['funder']=='World Bank':
            return 'world_bank'
        else:
            return 'other'

    dataframe1['funder'] = dataframe1.apply(lambda row: purple_donkey(row), axis=1)
```



```python
# The script MUST contain a function named azureml_main
# which is the entry point for this module.

# 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):
    def cheese_soup(row):
        if row['installer']=='DWE':
            return 'dwe'
        elif row['installer']=='Government':
            return 'gov'
        elif row['installer']=='RWE':
            return 'rwe'
        elif row['installer']=='Commu':
            return 'commu'
        elif row['installer']=='DANIDA':
            return 'danida'
        else:
            return 'other'

    dataframe1['installer'] = dataframe1.apply(lambda row: cheese_soup(row), axis=1)
    return dataframe1,
```

Again, I must click on the “Run” button to run these modules, and I know that it was successful (after almost a Iek of agonizing error messages mind you) after the green check mark appears in my module.

I then checked to see if my code had transformed the data in the ways that I wanted it to by visualizing the data in the same way that I did at the beginning of this experiment

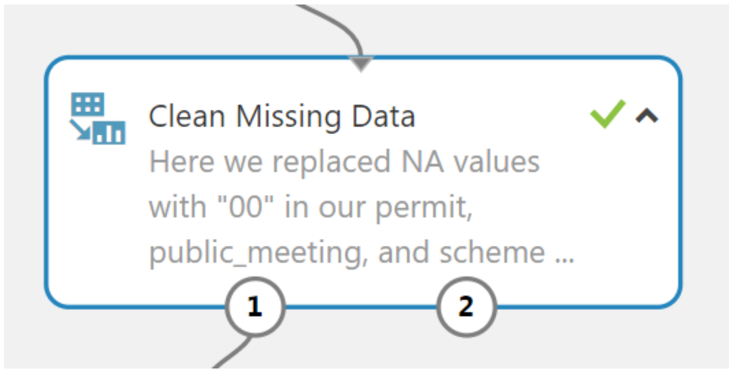

Tanzania Water Pump Final › Execute Python Script › Results dat

rows	columns
59400	24

Column 0	date_recorded	funder	gps_height	installer
0	2011-03-14T00:00:00Z	other	1390	other
1	2013-03-06T00:00:00Z	other	1399	other
2	2013-02-25T00:00:00Z	other	686	other
3	2013-01-28T00:00:00Z	other	263	other

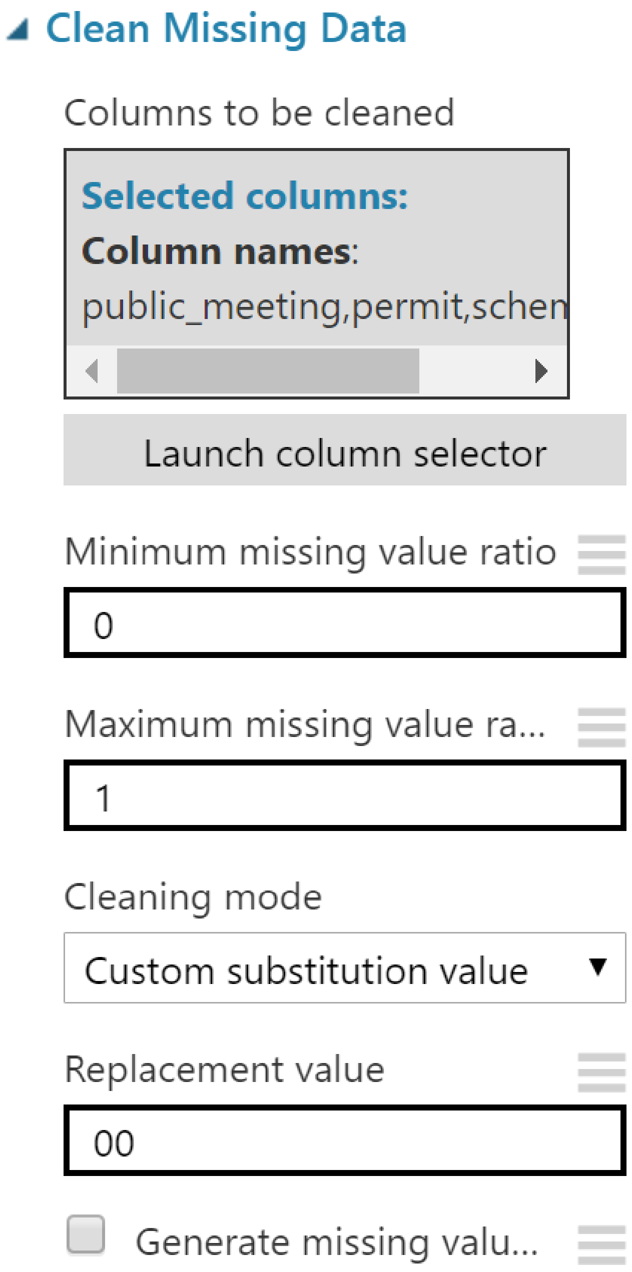
I can see here that my columns that I wanted dropped Ire dropped and that my “funder” and “installer” features have been aggregated how I wanted them, SIET!

Next, I will be cleaning some missing data. Since I cannot remove rows of data, due to the guidelines of the DrivenData competition, I simply recoded my missing data as “00” in my “permit”, “public_meeting”, and “scheme_name” features using the Clean Missing Data module



Clean Missing Data

Here we replaced NA values with "00" in our permit, public_meeting, and scheme ...



Clean Missing Data

Columns to be cleaned

Selected columns:
Column names: public_meeting,permit,schen

Minimum missing value ratio: 0

Maximum missing value ra...: 1

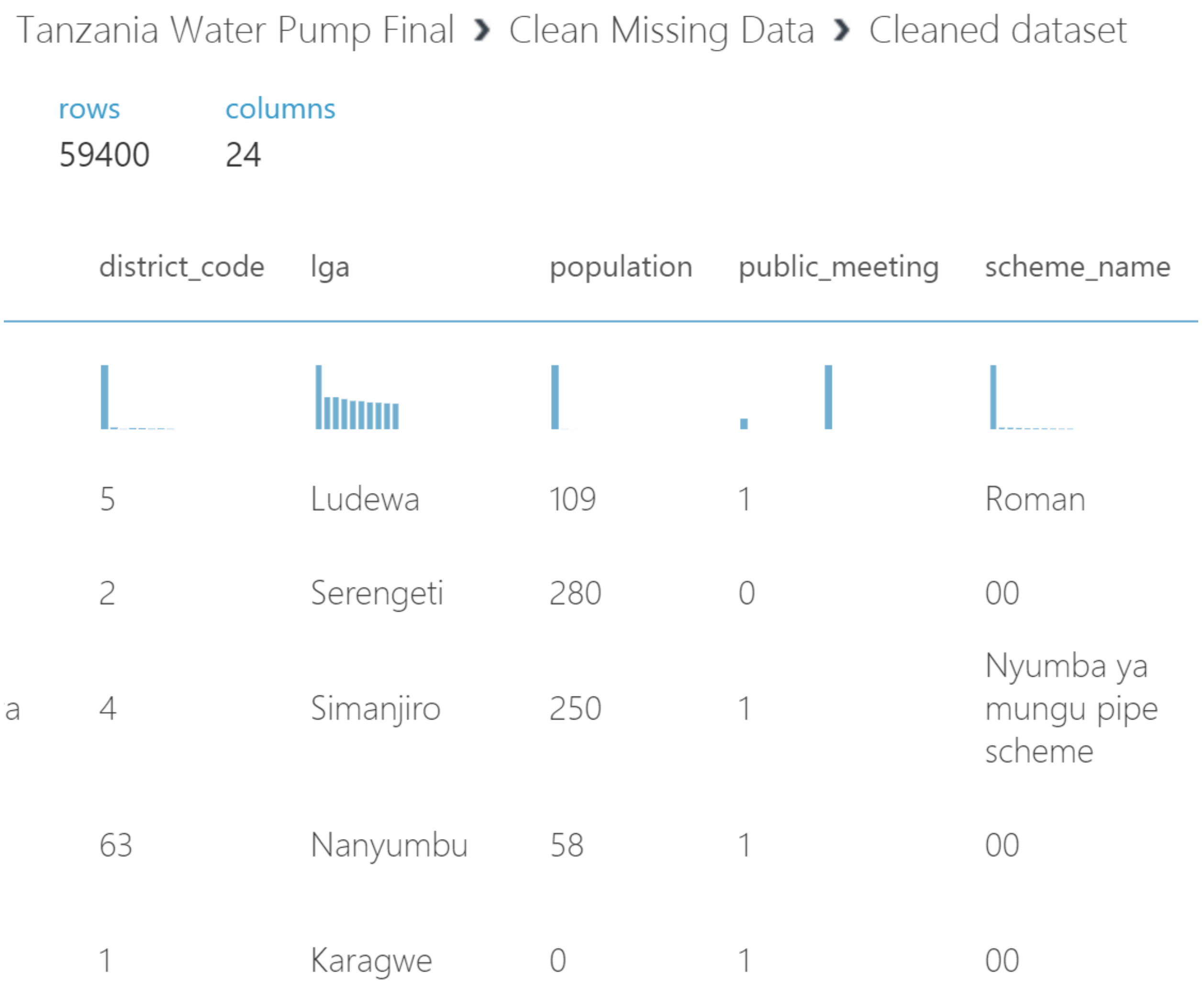
Cleaning mode: Custom substitution value

Replacement value: 00

☐ Generate missing valu...

I simply had to specify the columns that I wanted to clean the missing data in, and then choose "custom substitution value" under "cleaning mode" to impute my custom "00" for the missing data.

I then ran my experiment and visualized the results to see if things had worked properly



Tanzania Water Pump Final › Clean Missing Data › Cleaned dataset

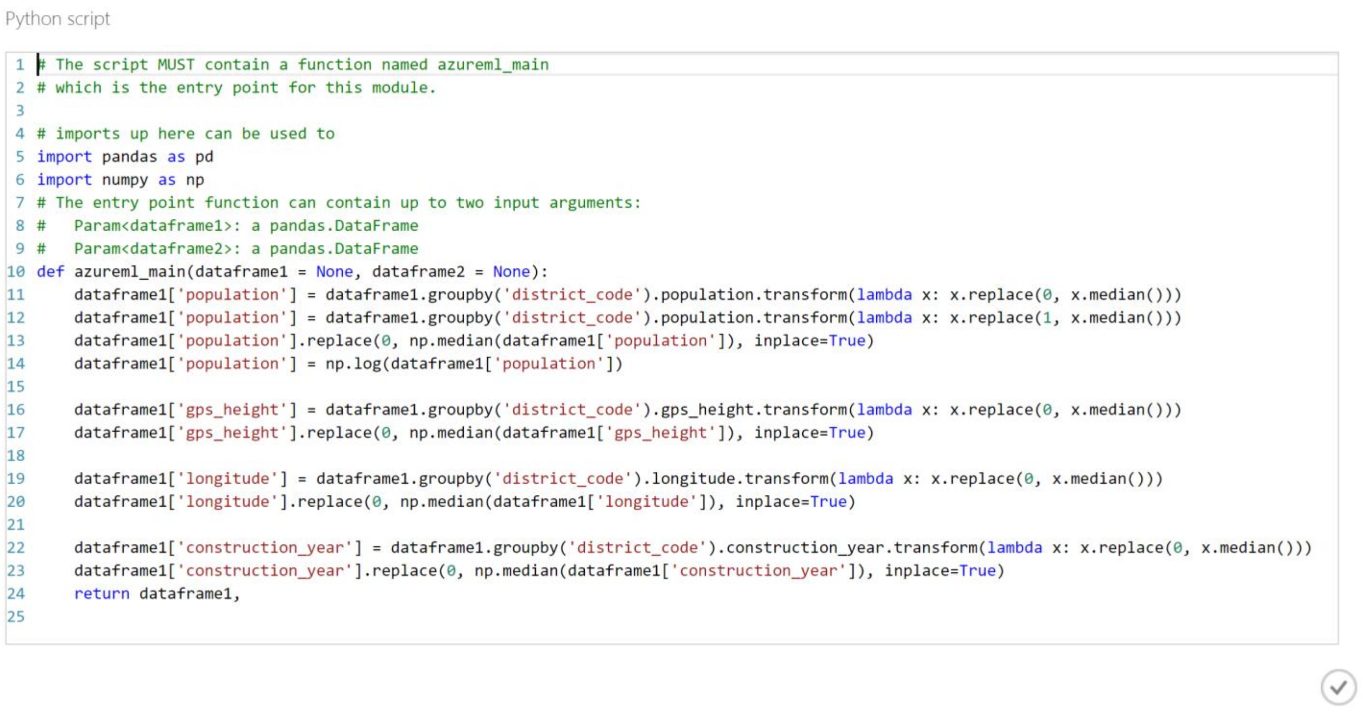
rows	columns
59400	24

	district_code	lga	population	public_meeting	scheme_name
	5	Ludewa	109	1	Roman
	2	Serengeti	280	0	00
a	4	Simanjiro	250	1	Nyumba ya mungu pipe scheme
	63	Nanyumbu	58	1	00
	1	Karagwe	0	1	00

As I can see, my "00" have replaced the NA values in my "scheme_name" feature

Now I can move into imputing missing values in my numeric feature columns. I decided to impute based on the median of my "district_code" feature since it was the most fine-grained location information and would, theoretically, provide the most accurate median value for not skewing my data. I accomplished this through another Execute Python Script module

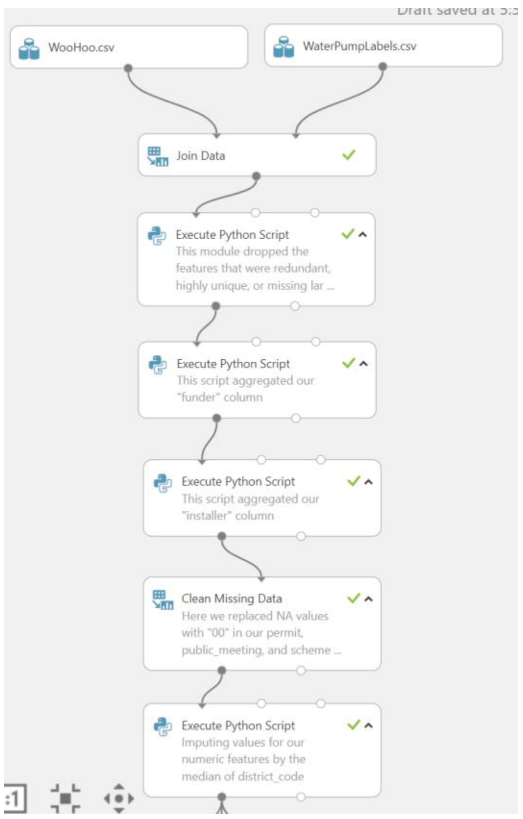
My Execute Python Script module for the imputation code looks like this


Python script

```python
# The script MUST contain a function named azureml_main
# which is the entry point for this module.

# imports up here can be used to
import pandas as pd
import numpy as np
# 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):
    dataframe1['population'] = dataframe1.groupby('district_code').population.transform(lambda x: x.replace(0, x.median()))
    dataframe1['population'] = dataframe1.groupby('district_code').population.transform(lambda x: x.replace(1, x.median()))
    dataframe1['population'].replace(0, np.median(dataframe1['population']), inplace=True)
    dataframe1['population'] = np.log(dataframe1['population'])

    dataframe1['gps_height'] = dataframe1.groupby('district_code').gps_height.transform(lambda x: x.replace(0, x.median()))
    dataframe1['gps_height'].replace(0, np.median(dataframe1['gps_height']), inplace=True)

    dataframe1['longitude'] = dataframe1.groupby('district_code').longitude.transform(lambda x: x.replace(0, x.median()))
    dataframe1['longitude'].replace(0, np.median(dataframe1['longitude']), inplace=True)

    dataframe1['construction_year'] = dataframe1.groupby('district_code').construction_year.transform(lambda x: x.replace(0, x.median()))
    dataframe1['construction_year'].replace(0, np.median(dataframe1['construction_year']), inplace=True)
    return dataframe1,

```

Once I put that script in, I ran my experiment

My experiment up to this point looks like this, I can begin modeling now


Draft saved at 5:3
WooHoo.csv
WaterPumpLabels.csv
Join Data
Execute Python Script
This module dropped the features that were redundant, highly unique, or missing lar ...
Execute Python Script
This script aggregated our "funder" column
Execute Python Script
This script aggregated our "installer" column
Clean Missing Data
Here we replaced NA values with "00" in our permit, public_meeting, and scheme ...
Execute Python Script
Imputing values for our numeric features by the median of district_code

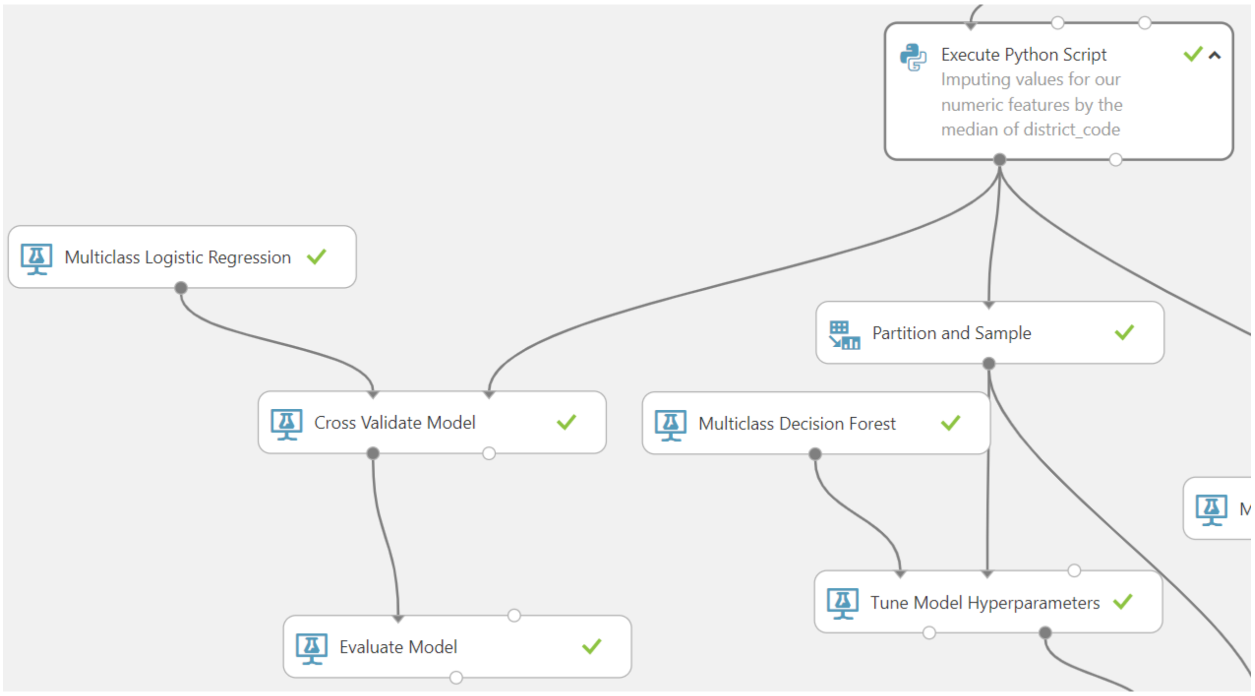
Modeling

Logistic Regression

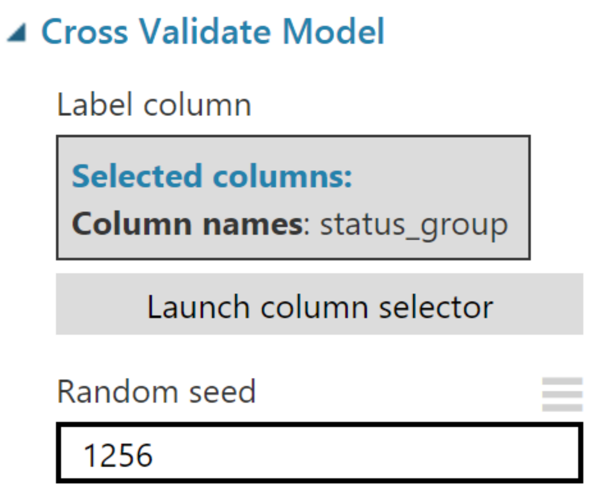
In my Python project, I used random forest and logistic regression CV algorithms, but neither of these were available in Azure ML, so Iused the Multiclass Logistic Regression and Multiclass Decision Forest modules instead. My results were similar, but score a little lower than my algorithms did in my Python project

Instead of using a train_test_split here, I used cross validation to train my models, I began with Multiclass Logistic Regression

Iconnected my dataset to the Cross Validate Model module first


Execute Python Script
Imputing values for our numeric features by the median of district_code
Multiclass Logistic Regression
Partition and Sample
Cross Validate Model
Multiclass Decision Forest
Tune Model Hyperparameters
Evaluate Model

I needed to specify my labels column and a random seed within my Cross Validate Model module


◢ Cross Validate Model

Label column

Selected columns:
Column names: status_group

Launch column selector

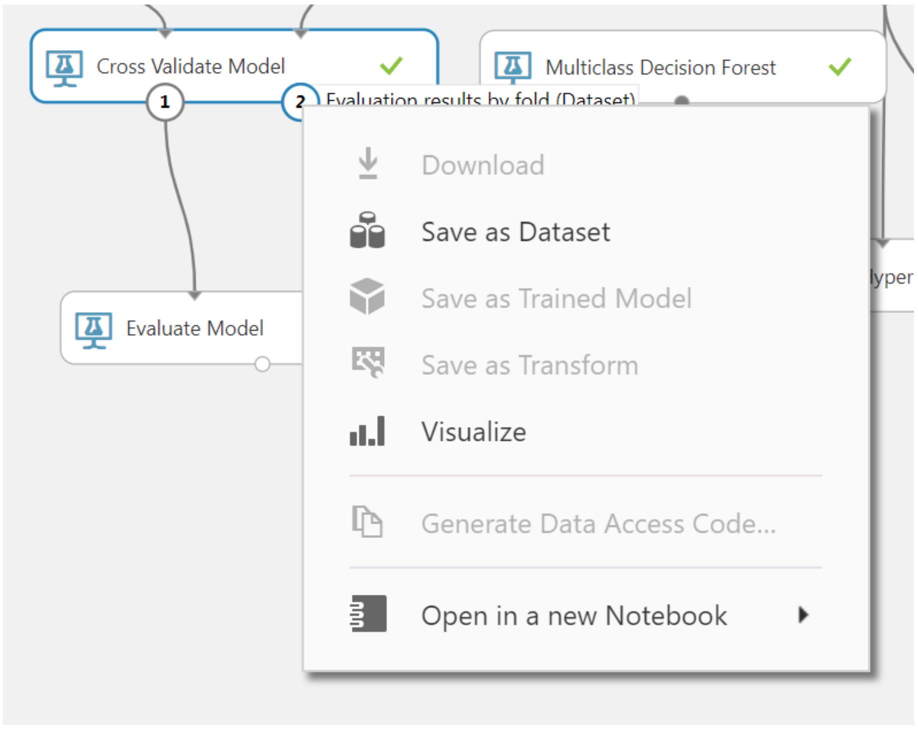
Random seed

1256

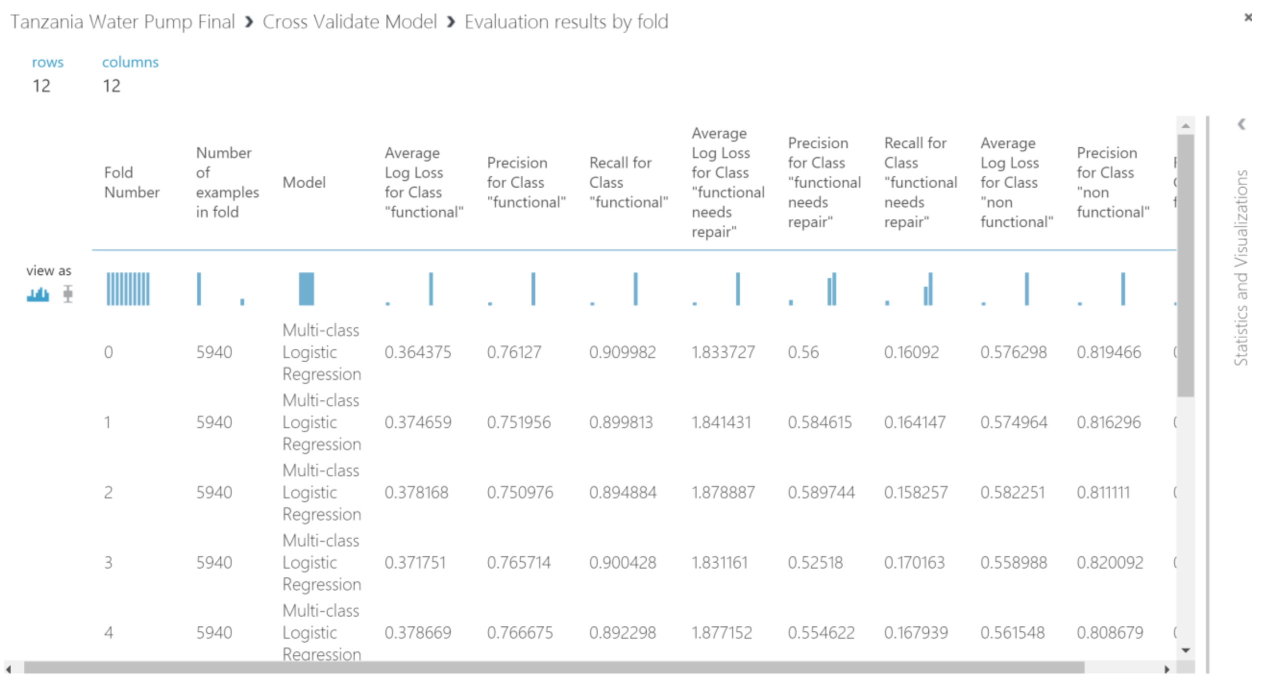
Once I had this module connected to my dataset, I connected my untrained model to the same port of my Cross Validate Model

I used all of the default parameters within the Multiclass Logistic Regression module

Once the cross validation was complete, I was able to visualize how the model scored on each fold by right clicking the Evaluate Results port on the output of Cross Validate Model



Which looked like this



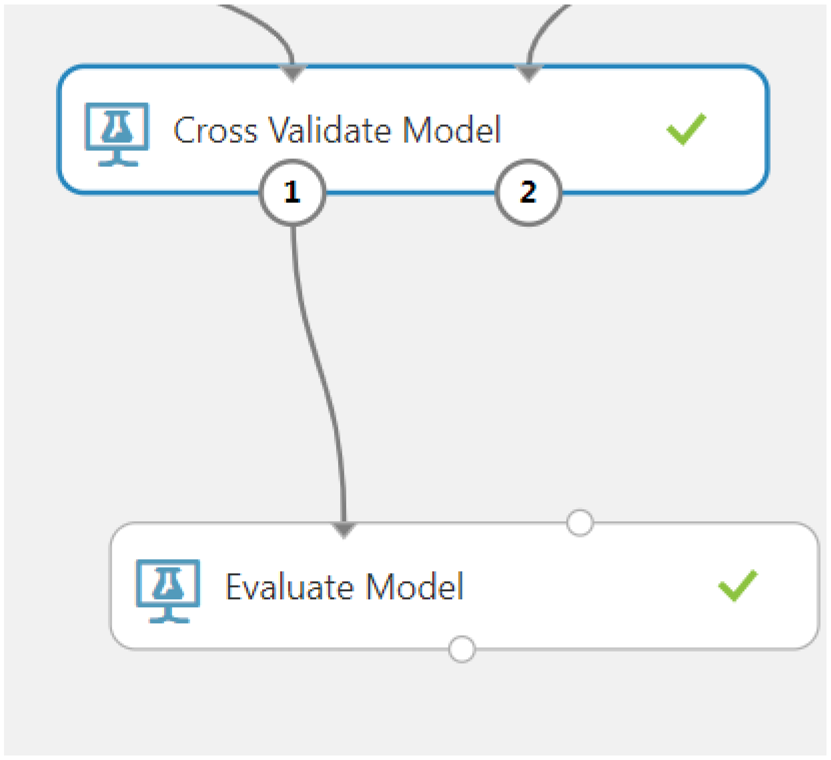
Tanzania Water Pump Final › Cross Validate Model › Evaluation results by fold

rows 12 columns 12

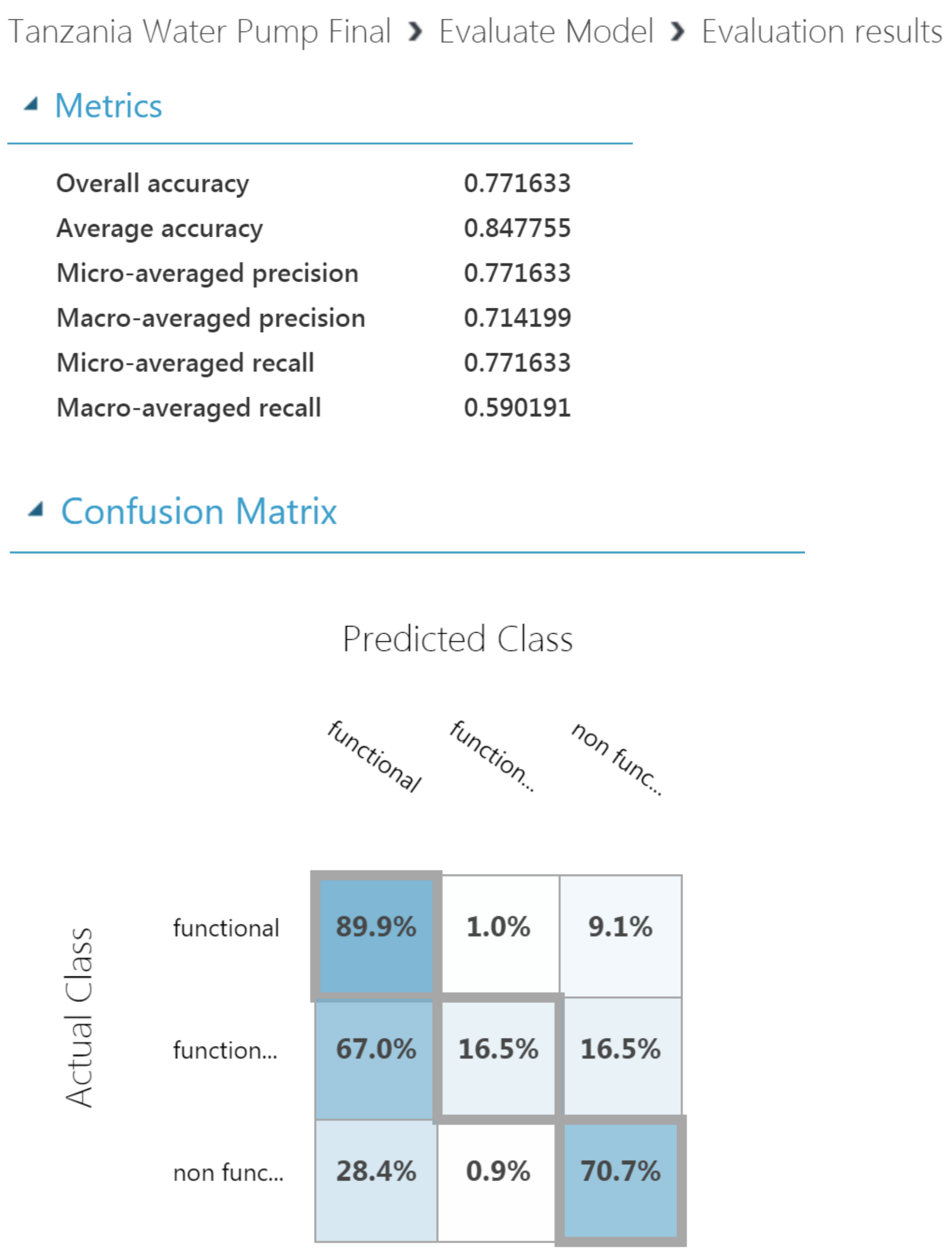
Fold Number	Number of examples in fold	Model	Average Log Loss for Class "functional"	Precision for Class "functional"	Recall for Class "functional"	Average Log Loss for Class "functional needs repair"	Precision for Class "functional needs repair"	Recall for Class "functional needs repair"	Average Log Loss for Class "non functional"	Precision for Class "non functional"
0	5940	Multi-class Logistic Regression	0.364375	0.76127	0.909982	1.833727	0.56	0.16092	0.576298	0.819466
1	5940	Multi-class Logistic Regression	0.374659	0.751956	0.899813	1.841431	0.584615	0.164147	0.574964	0.816296
2	5940	Multi-class Logistic Regression	0.378168	0.750976	0.894884	1.878887	0.589744	0.158257	0.582251	0.811111
3	5940	Multi-class Logistic Regression	0.371751	0.765714	0.900428	1.831161	0.52518	0.170163	0.558988	0.820092
4	5940	Multi-class Logistic Regression	0.378669	0.766675	0.892298	1.877152	0.554622	0.167939	0.561548	0.808679

I can see my Precision, Recall, and Average Loss for each fold under each label

In order to evaluate my model after training and testing, I connected an Evaluate Model module to my Cross Validate Model output



The Evaluate Model output shows us my overall results and a confusion matrix


Tanzania Water Pump Final › Evaluate Model › Evaluation results

Metrics

Overall accuracy	0.771633
Average accuracy	0.847755
Micro-averaged precision	0.771633
Macro-averaged precision	0.714199
Micro-averaged recall	0.771633
Macro-averaged recall	0.590191

Confusion Matrix

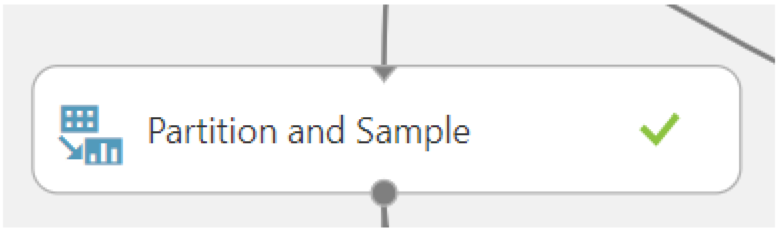
Actual Class \ Predicted Class	functional	function...	non func...
functional	89.9%	1.0%	9.1%
function...	67.0%	16.5%	16.5%
non func...	28.4%	0.9%	70.7%

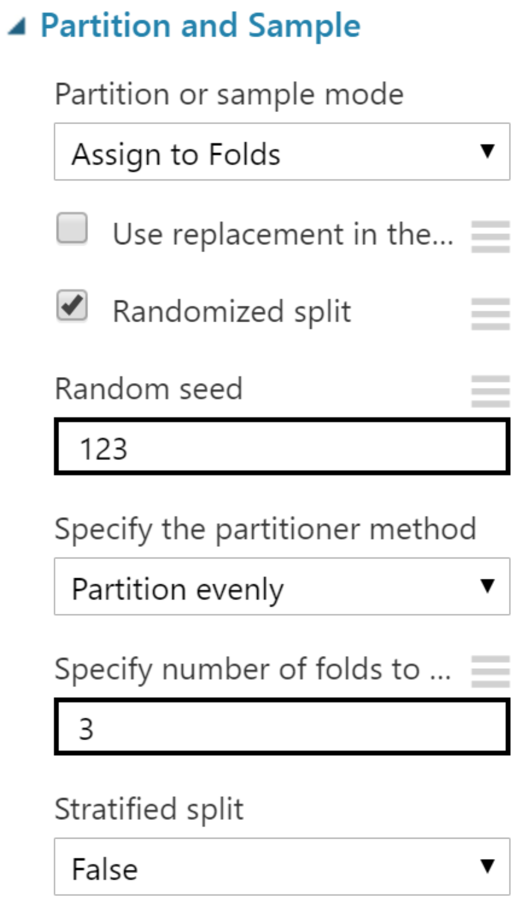
I can see that my worst class, possibly due to the amount of data available for this label, was the "functional needs repair" class

Multiclass Decision Forest

My next model I trained using nested cross validation, which is really interesting to do in Azure ML Studio, and actually quite a bit faster than using Python depending on the parameters you are using, I wonder if this is due to the code running on the cloud?

I first used the Partition and Sample module to partition my data into 3 folds, exactly how I did it in my Python Project





Partition and Sample

Partition or sample mode: Assign to Folds

[ ] Use replacement in the...

[✓] Randomized split

Random seed: 123

Specify the partitioner method: Partition evenly

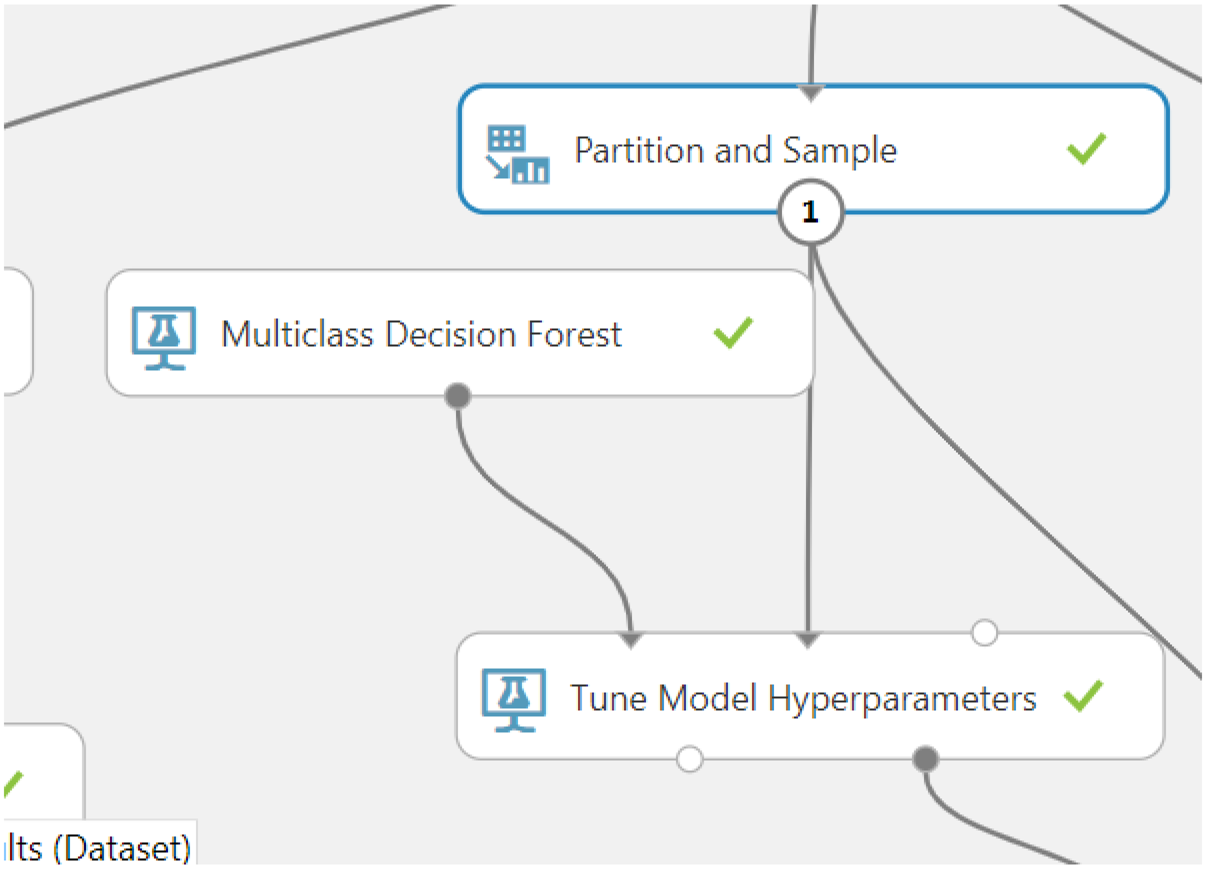
Specify number of folds to ...: 3

Stratified split: False

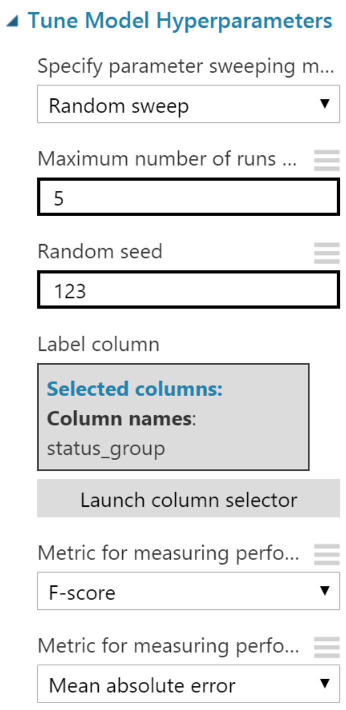
I chose the “assign to folds” option and then specified 3 folds

I chose to use a randomized split and to partition the data evenly into each fold, I did not use stratified split to sample more data heavily than others

This module was connected to the cleaned dataset and then the output port was connected to the Tune Model Hyperparameters module



Within the Tune Model Hyperparameters module, I had to specify several things



Tune Model Hyperparameters

Specify parameter sweeping m...
Random sweep

Maximum number of runs ...
5

Random seed
123

Label column
Selected columns:
Column names:
status_group

Metric for measuring perfo...
F-score

Metric for measuring perfo...
Mean absolute error

1) I wanted to do a random sweep, similar to how I did it in my Python project

2) specify the label colum

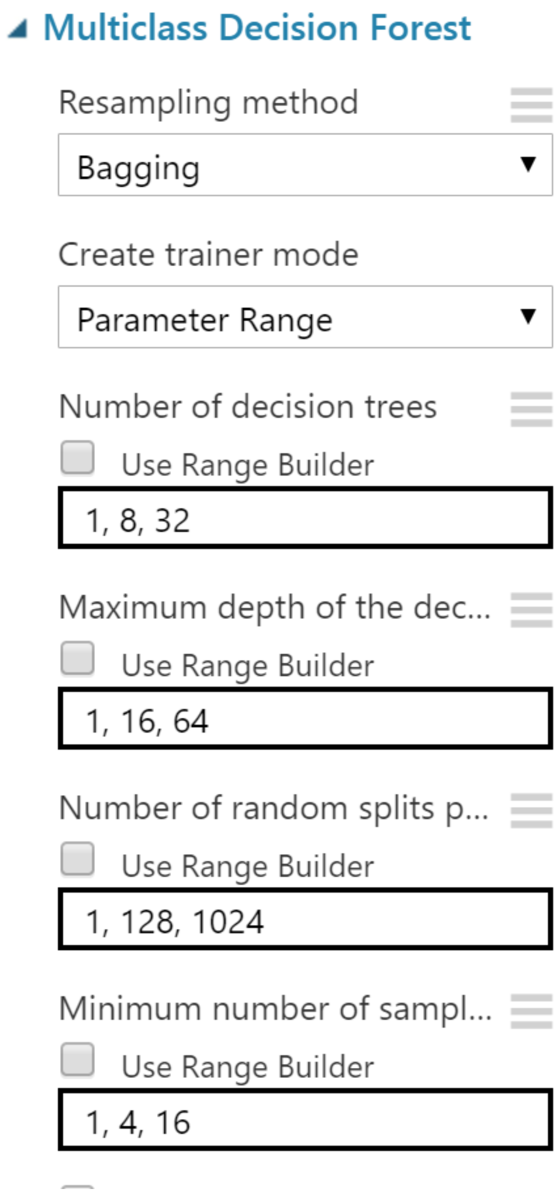
3) specify the metric I wanted my model trained on

4) specify overall measure for performance

I wanted to use F-score as my metric due to the class imbalance of my labels, I was hoping that the model would predict more of the “functional_needs_repair” cases, and it seems as though that did happen after evaluation

From there I connected an untrained Multiclass Decision Forest module to the Tune Hyperparameters module as shown above

My Multiclass Decision Forest module allowed me to specify the range of hyperparameters that we wanted to sweep


Multiclass Decision Forest

Resampling method

Bagging

Create trainer mode

Parameter Range

Number of decision trees

☐ Use Range Builder

1, 8, 32

Maximum depth of the dec...

☐ Use Range Builder

1, 16, 64

Number of random splits p...

☐ Use Range Builder

1, 128, 1024

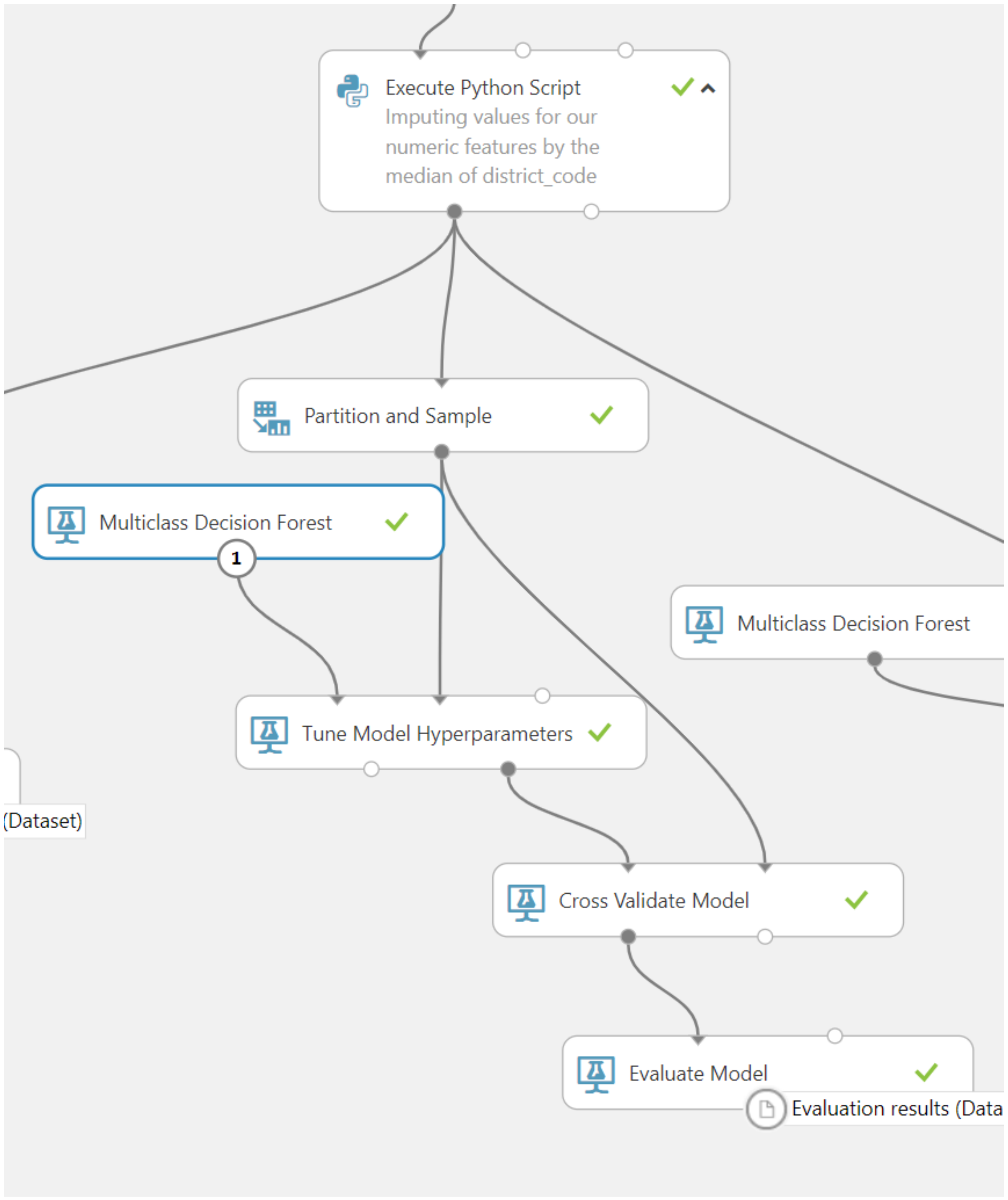
Minimum number of sampl...

☐ Use Range Builder

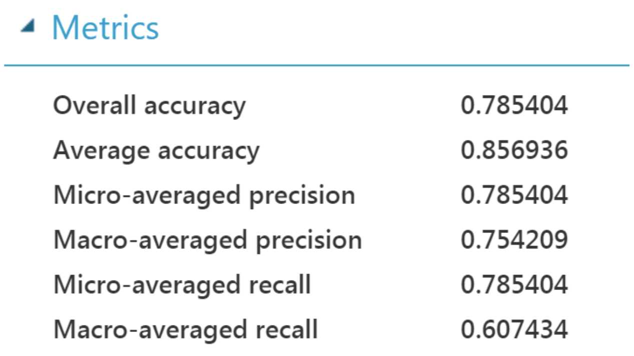
1, 4, 16

We allowed the range to remain default for sweeping due to the fact that adding decision trees to this model did not seem to increase its accuracy, which is very strange compared to my random forest model in the Python project, which when the trees were increased performed significantly better

We then added in my Cross Validate Model and connected it to the input dataset and the Tune Model Hyperparameters module before evaluating it, we simply had to specify the labels column for the Cross Validate module


Execute Python Script
Imputing values for our numeric features by the median of district_code
Partition and Sample
Multiclass Decision Forest
1
Multiclass Decision Forest
Tune Model Hyperparameters
(Dataset)
Cross Validate Model
Evaluate Model
Evaluation results (Data

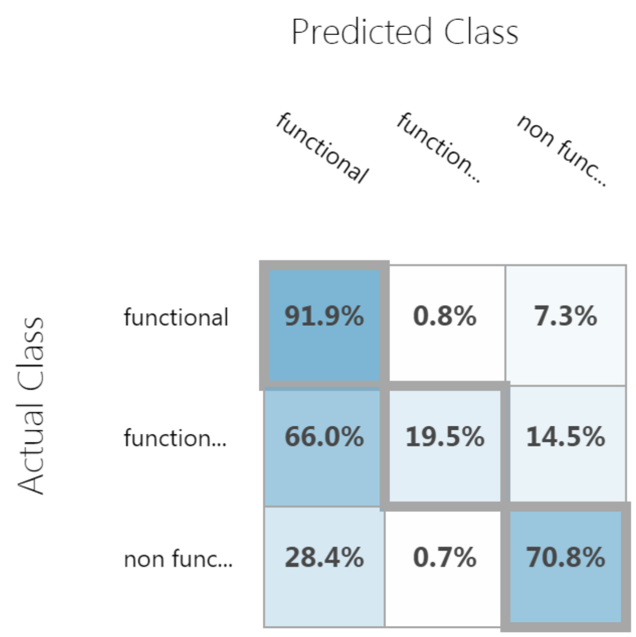
My evaluation looked like this



◢ Metrics

Overall accuracy	0.785404
Average accuracy	0.856936
Micro-averaged precision	0.785404
Macro-averaged precision	0.754209
Micro-averaged recall	0.785404
Macro-averaged recall	0.607434

This was only slightly better than my Logistic Regression model, but the predictions for classes seemed to improve, especially for the “functional_needs_repair” class

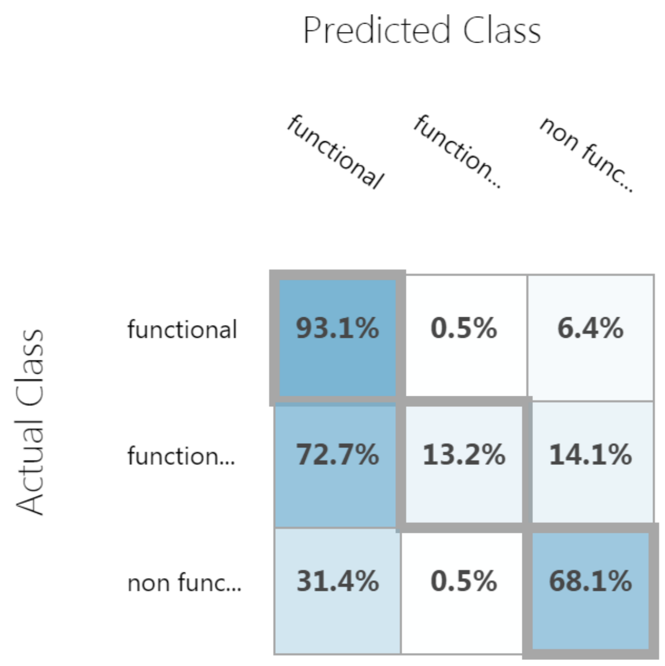


Predicted Class

Actual Class

	functional	function...	non func...
functional	91.9%	0.8%	7.3%
function...	66.0%	19.5%	14.5%
non func...	28.4%	0.7%	70.8%

It seems as though my F-score criteria for my tuning worked out, that class is being predicted at a higher, albeit crappy, level when compared to a Multiclass Decision Forest that we ran without nested cross validation



	functional	function...	non func...
functional	93.1%	0.5%	6.4%
function...	72.7%	13.2%	14.1%
non func...	31.4%	0.5%	68.1%

Predicted Class
Actual Class

And That’s My Project