*Azure Machine Learning Studio is a powerfully simple browser-based, visual drag-and-drop authoring environment where no coding is necessary. Go from idea to deployment in a matter of clicks.* It is a fully-managed cloud service that enables you to easily build, deploy, and share predictive analytics solutions. And you can get your model into production as a web service in minutes—a web service that can be called from any device, anywhere, and that can use any data source. It also includes hundreds of built-in packages and support for custom code in R or Python.

**Example 1 - Income prediction based on census data:**

(Final module tree print-screen at the end of the setup instruction)

First, create a **NEW** (bottom left) experiment

List of templates to choose from, quite extensive.

There is a tutorial, we will choose the **Blank Experiment**

*Properties* on the right, *Components* on the left

This is your canvas, a drag-and-drop environment, with items on your left side to be used to build your experiment. The items may also have sub-items, and below are some examples:

* **Saved Datasets** – if you have imported datasets earlier, you will find them here
* **Trained Models** – you can bring your already trained models to canvas
* **Data Format Conversions** – convert your datasets to different formats:
  + ARFF (attribute relation file format),
  + CSV.
  + TSV (tab-delimited format),
  + etc.
* **Data Input and Output**
  + *Enter data manually*,
  + *Export Data* (to Azure Blob, Hive, Azure SQL DB etc),
  + *Import Data* (from Web, Azure Blob, Azure SQL DB, Hive, Azure DocumentDB, etc.)
  + *Etc.*
* **Data Transformation**
  + Filter data (various data filtering options),
  + *Sample and Split* (partition and sample, Split data),
  + *Scale and Reduce*,
  + etc.
* **Machine Learning** 
  + *Evaluate*,
  + *Initialize Model*
    - Anomaly Detection,
    - *Classification* (various models – multiclass decision forest, multiclass decision jungle, multiclass neural network, two-class Bayes Point Machine, two-class boosted decision tree, two-class decision forest, etc.),
    - *Clustering* – K-Means clustering),
    - Regression (Bayesian Linear, Decision Forest, Linear, Nerual Network, etc.)
* **Score**
  + *Score Model,*
  + *Apply Transformation,*
  + *Assign Data To Clusters,*
  + *etc*.
* **Python Language Modules –** execute python script
* **R Language Modules –** execute R script
* **Statistical functions –** apply various statfunctions to data (math, linear correlation, probability function, summarize data, etc.)
* **Text analysis**
  + *Detect languages*
  + *Extract key phrases from text*
  + *Feature hashing*
  + *Preprocess text*
  + *Score Vowpal Wabbit (various versions)*
  + *Train Vowpal Wabbit (various versions)*
* **Time series**
* **Web Service**

Option to Import Data

Right side, choose from various import strategies, all the information on the internet we can import into my experiment.

For the sake of time, we will use prepared data from the **Saved Datasets**.

**Adult Census Income Binary Database**

Click on the bottom node, click on **Visualize**

You will see data, we have 32000 records of data, give us lots of census data.

Each part of the data can be visualized as a curve.

The Income has been split into **binary values**.

This is because T/F, binary decisions, are easier for ML or self-trained models to predict to exact values.

We want an algorithm that trains itself so that it can predict the income of records it has never seen before.

**Adult Census Income Binary Database**

**Data Transformation**

**Manipulation**

place the **Select columns in Dataset** box below the **Adult Census…,** connect nodes.

Click on Launch column selectorunder Properties on the right-hand side. Tell it only to use age, education, marital-status, relationship, race, sex, income.Data is filtered to only that needed

**Machine Learning** algorithm

Lots of models, initialization,

**Initialize Model**

**Classification**

**Two-Class Boosted Decision Tree**

Good at predicting T/F values

**Train**

**Train Model**

Connect left node to Two-Class, right to data

**Data Transformation**

**Sample and Split**

**Split Data**

Tell it to use 80% of data in Propeties (“Fraction of rows …. 0.8”)

**Train Model**

Click on Launch column selector*.* Select/typeincome**,**  tells the model it is supposed to be able to guess the income based on all the other data it receives

**Machine Learning**

**Score**

**Score Model**

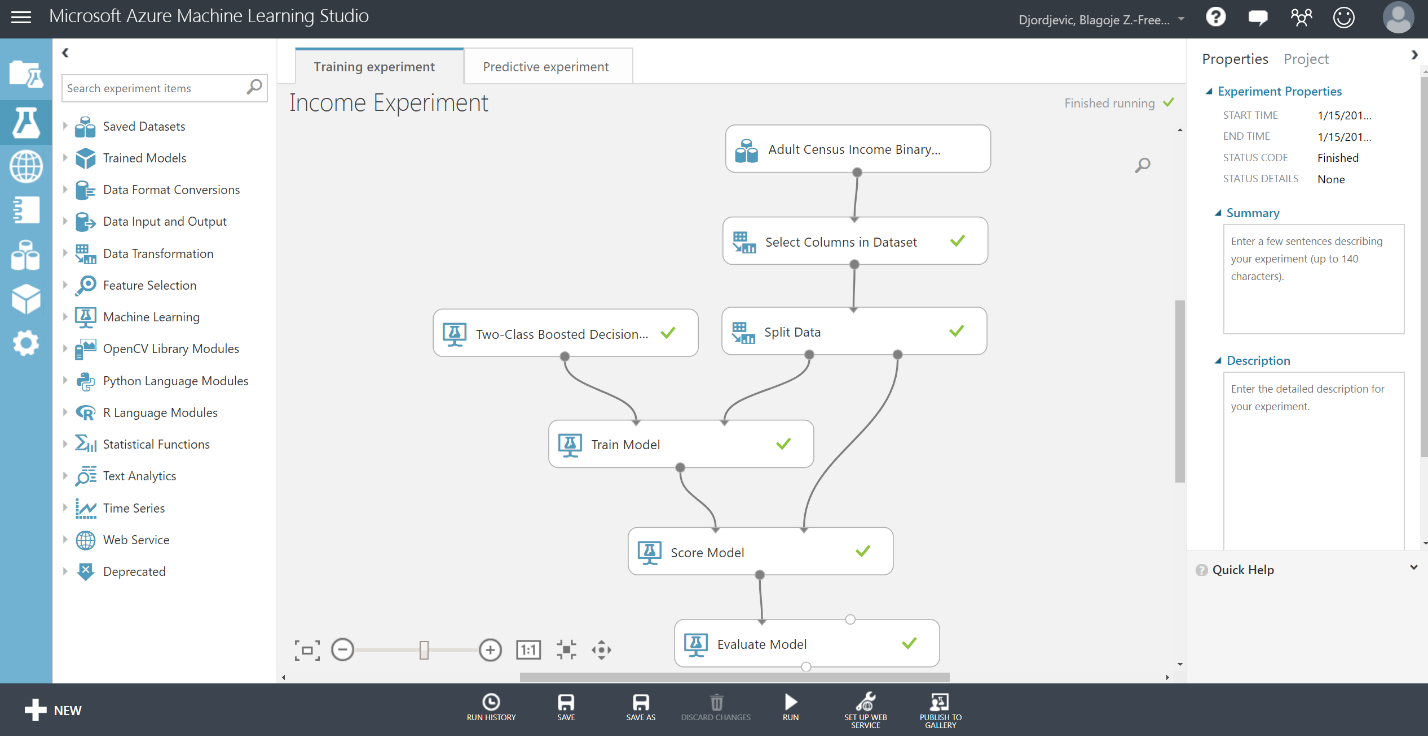
**Evaluate**

**Evaluate Model**

**Score Model** is unhappy, needs other data

Create a link between remaining 20% of the data and the **Score Model**

Click on Save on the gray bar at the bottom of the internet browser window.



Click on Run. This means all the steps in our tree will be executed in the cloud, not on your computer

Visualize the **Evaluate Model** by right clicking on the node at the bottom of the module.

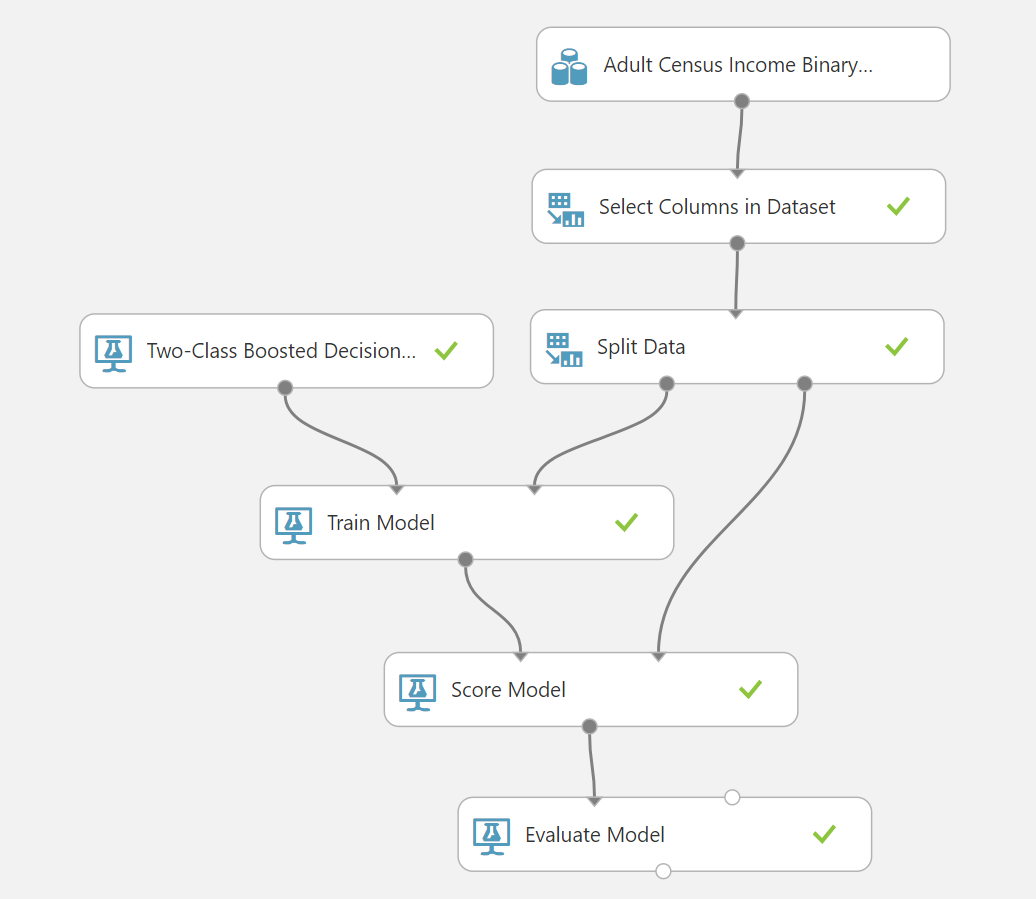
The curve means

Curved to the upper left – model is very precise in predicting the income of a person

Diagonal – the model is randomly guessing, just 50% proper guesses

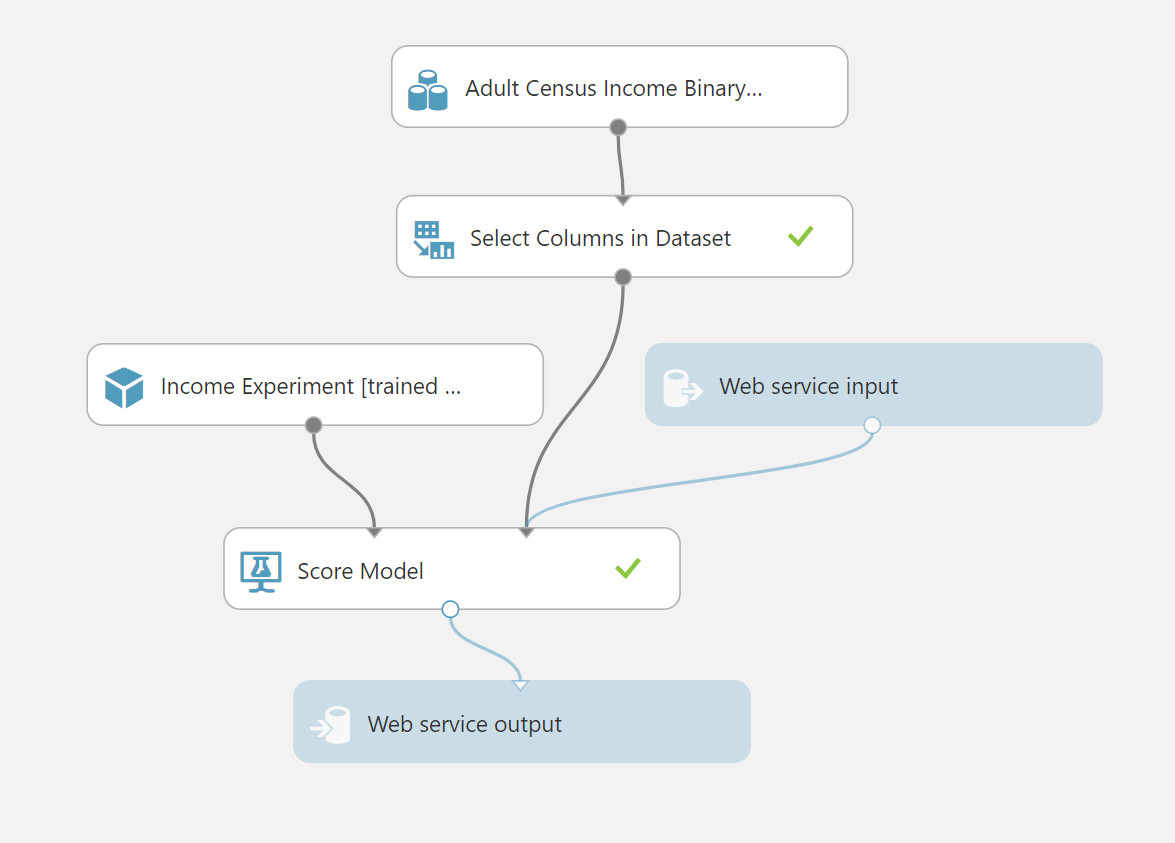
Bottom-right – it means that it is very precise at predicting the wrong answer, which does not mean that it is a bad model, you can use it to always take the opposite to get the desired information

More detailed information at the bottom of the page.



**SET UP WEB SERVICE**

Tell our system to save our model that will be reused and published as a web service by clicking on DEPLOY WEB SERVICE on the bottom, gray bar.



We want the input to go to the Right score model node

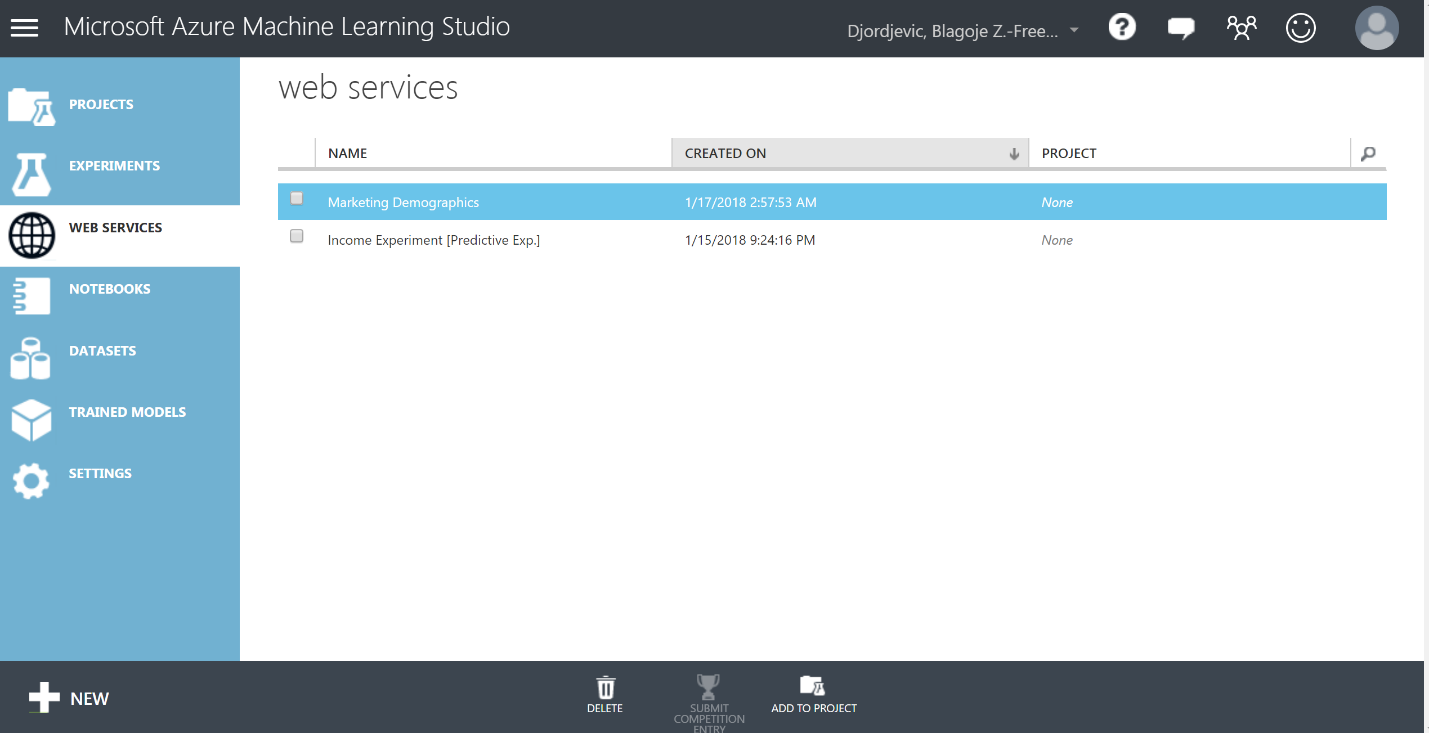
Change **Select Columns in Dataset** column selector and delete the income selection

The webservice is a function in the cloud that allows me to connect any software or data to this function and tell me what the expected income would be.

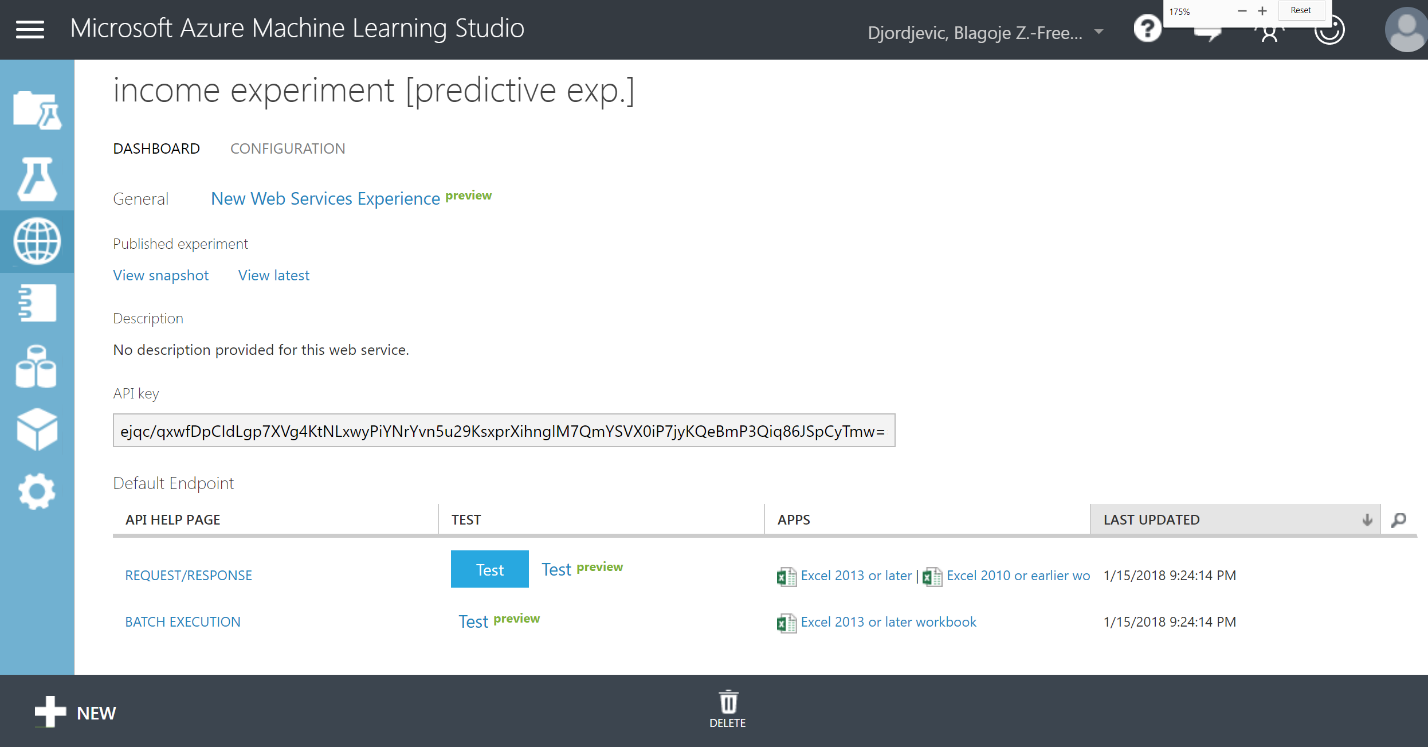
SAVE

RUN**,** has to be run once before it can be published

Left bar interface, *EMPLOY WEB SERVICES* tab (the internet Globe symbol)



Click on your experiment, click on Test for REQUEST/RESPONSE.



Test data:

44

Doctorate

Married-civ-spouse

Own-child

White

Female

This person will be evaluated by algorithm, and it tells us that this person will earn >$50K with a probability of 90%. If we change the age to 22, the probability that that person earns more than $50K falls to only 20%, as might be expected.

**Example 2 - Image Classification:**

Machine learning is especially famous for image processing, notably for processing cat images. We will apply a pre-made model for face recognition, particularly for detecting whether a face is facing forward or not (**Frontal Face**)

**OpenCV Library Modules**

**Pretrained Cascade Image Classification**

Scale factor **-**  specifies how much the image size is reduced at each image scale, allowing for easy resizing of the processing target.

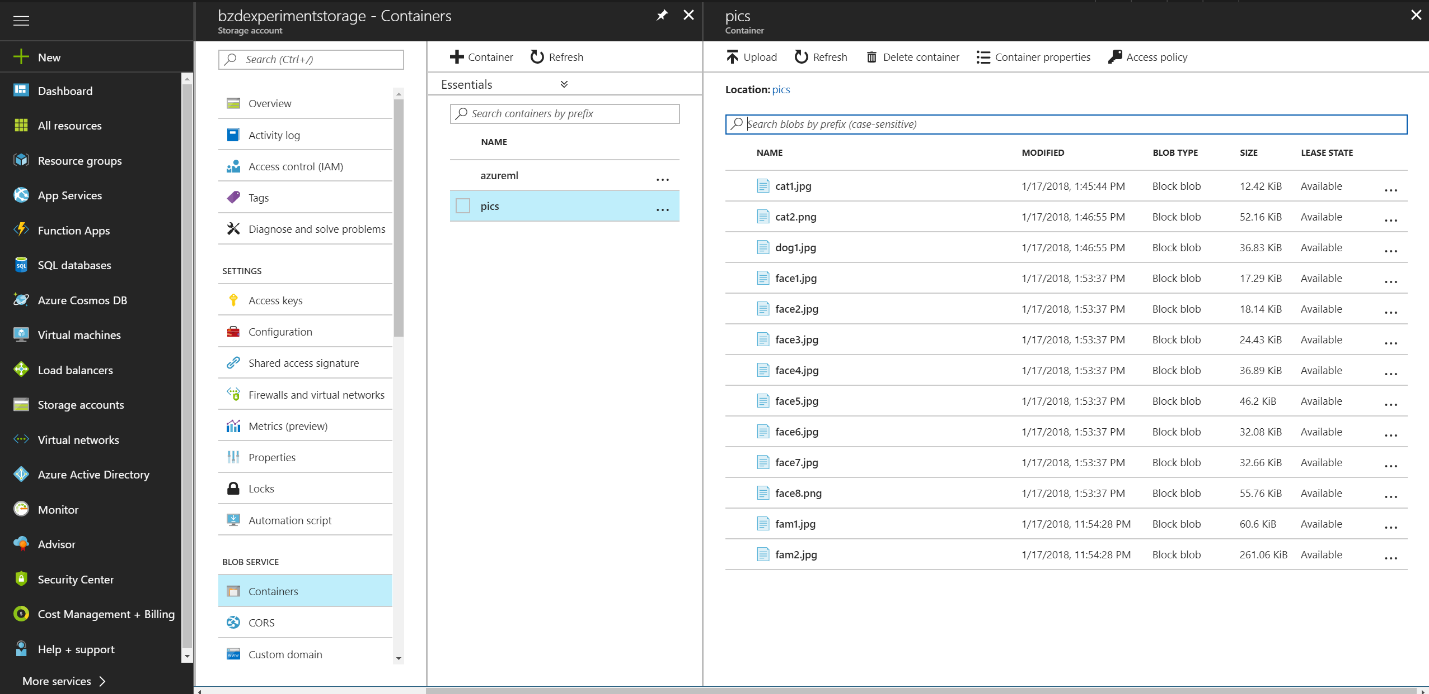
Minimum number of neighbors – minimum number of overlapping rectangles that are required to detect a face in a region.

Minimum/Maximum height – the pixel height of the corresponding search rectangle

**OpenCV Library Modules**

**Import Images** – Add your own image dataset from blob storage

In general, all images in the dataset should be the same size and relatively small, i.e. less than 409,600 pixels or smaller than 640x640.



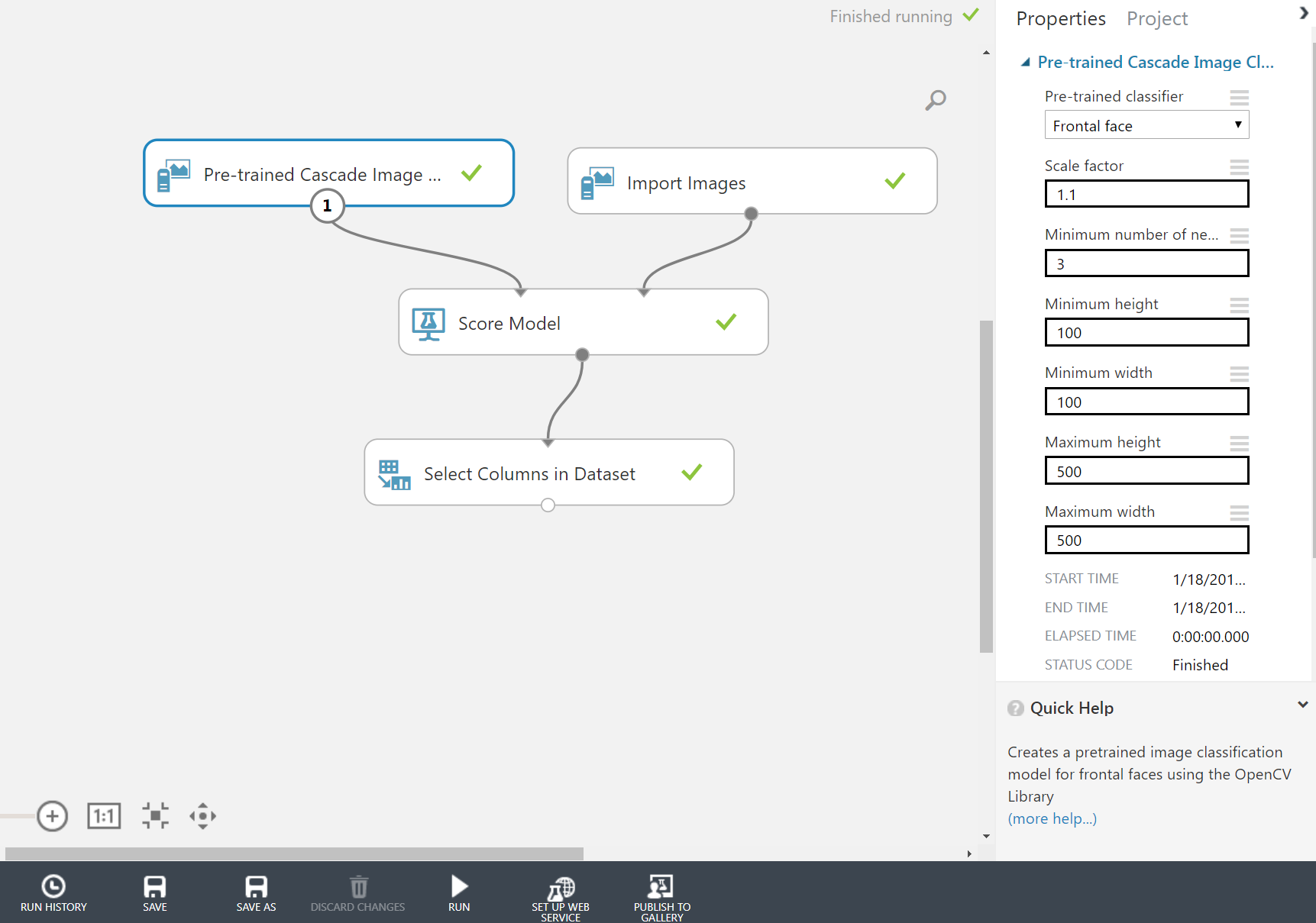
**Machine Learning**

**Score Model**

There will be a lot of outputs, primarily the RGB values for all color channels in the dataset, so we can filter these out using **Data Transformation > Manipulation > Select Columns in Dataset**

Click on Save, Run on the bottom of the browser, it takes about 5-10 minutes

After finish, right-click on bottom node of **Select Columns in Dataset** and click on Visualize to see how the trained model did on your provided images.



**Example 3 – Marketing statistics:**

Test whether marketing appeals work to get customers to buy product and what kind of customers are buying, i.e. demographics. Use predictive model to evaluate potential customers before appeals are sent to them, use demographic info, i.e. marital status, gender, age, postal code, if they were our customers in the past, etc. to get the best list of potential customers.

The working example will contain:

* Dataset with 1M rows of data
* Resampling and partitioning the data (if needed)
* Some data cleansing
* Splitting the data in train and test datasets
* Training & Evaluating the models using four ML algorithms.
* Comparing the results and choosing the best model

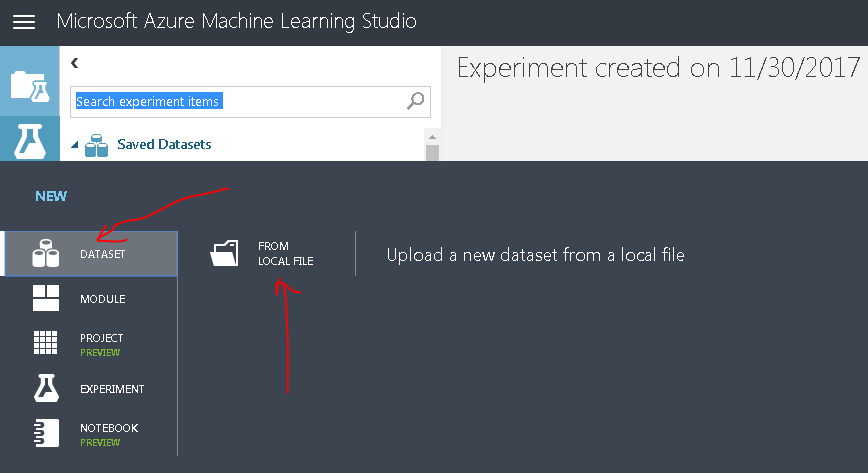
1. **Data set**

Data set we will use contains over a million data point over 4 years of sales in a CSV file:

* **appeal\_date** - Date of the appeal sent to the potential customer
* **appeal\_month** – the name of month, extracted from the appeal\_date
* **season** – the name of the year’s season, extracted from the appeal\_date
* **donor\_id** – system ID of the donor, from our DW, but randomized, to preserve the privacy of the customers
* **appeal\_id** – an ID of the appeal/marketing strategy
* **age** – actual age of the customer, if available - demographic info
* **cnt\_spouse** – boolean field – 0 if no spouse, 1 if has a spouse - demographic info
* **marital status** – current marital status of the potential customer - demographic info
* **gender** – gender of the potential customer - demographic info
* **FSA** – customer’s postal code in Canada – equivalent to zip code in US (more or less)
* **Has been patient** – boolean field – 0 if was not a patient in the hospital, 1 if yes
* **cnt\_purchased** – boolean field – 0 if not purchased within 60 days after the appeal was sent, 1 if did - **THIS FIELD WILL BE PREDICTED**

**Steps to build the experiment:**

***STEP 1.*** To get data, click on the **NEW** button at the bottom left corner, and click on **Dataset**, and choose From Local File, as per below screenshot:



Here we will choose the CSV file, **marketing\_data.csv** , and bring it to Azure ML Studio. Once uploaded, we will add it to canvas from **Saved Datasets** item on the left side, in **My Datasets**. Just drag it to the canvas.

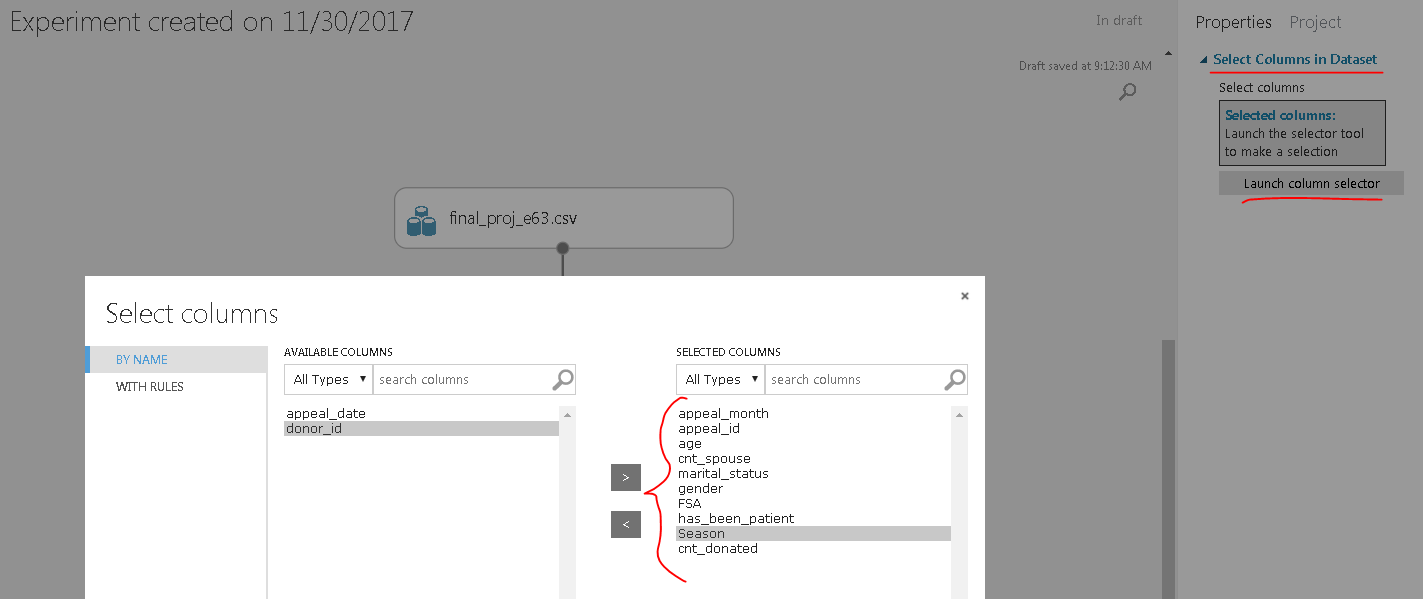
***STEP 2.***

Do not need all data, filter

**Select Columns in Dataset**, found at **Data Transformation** – **Manipulation** item on the left.

Drag and connect the two items

Now let’s select the columns we would from the dataset. Click on the **Select Columns…** item on the canvas, and on the right side you will see the properties of the select item on the canvas. Click on the Launch column selector, and select columns as per screenshot below:



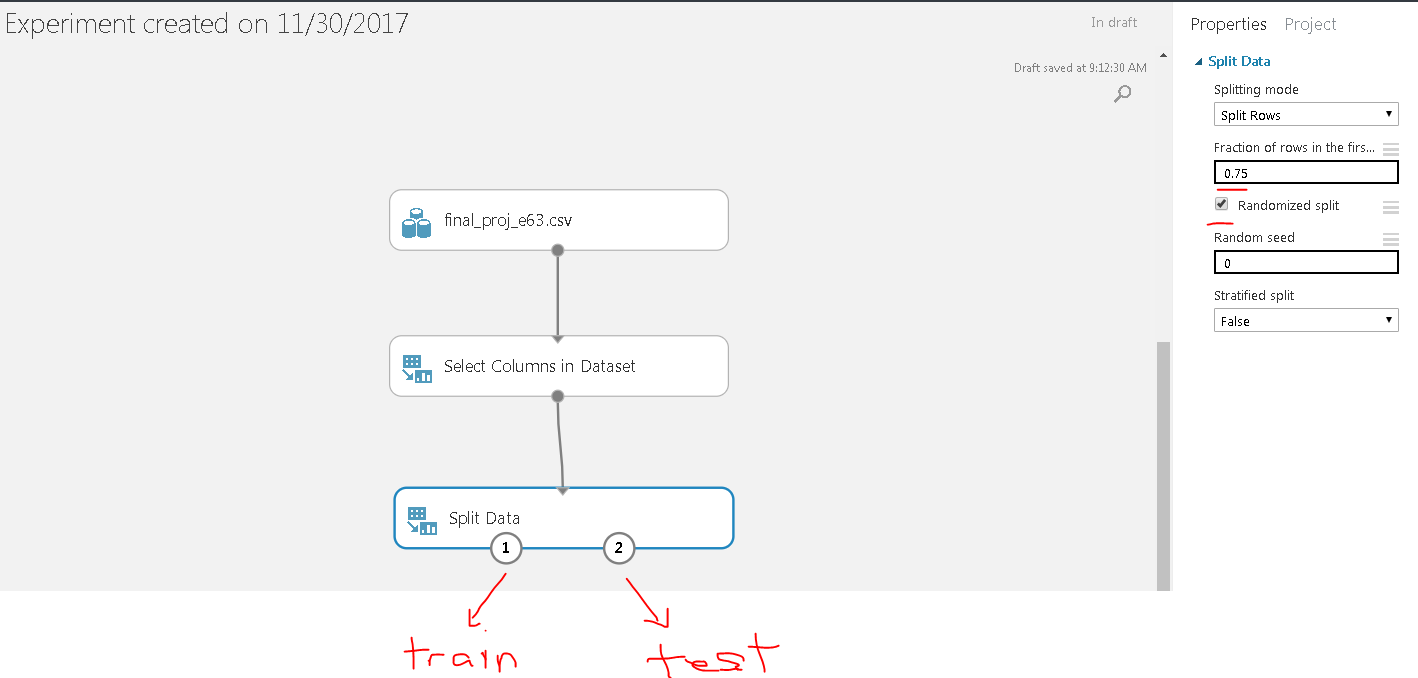
***STEP 3.***

Now let’s split data into training and testing sets.

It can be done by using the Split Data item from the **Data Transformation** – **Sample and Split** store on the left.

Drag it on the canvas, and connect **Select Columns…** output to **Sample and Split** input.

Select the item and on the properties on the right, in Fraction of rows box, enter 0.75. This means that we will create a dataset in the first output of the item with 75% of the source data, and we will use this first output as our training data, while the second dataset will be used for testing, and will be coming out of the second output (1 for first output, and 2 for second). Keep the Randomize split checkbox checked.



You can add some comments for each item on the canvas, by double-clicking on it. Let’s add a comment: Split the data: 75% for training, 25% for testing. And that concludes the data preparation for the model.

Now we will bring 4 different regression models and test each to see which one performs the best. The four different models we will use are:

* Bayesian Linear Regression
* Decision Forest Regression
* Linear Regression
* Neural Network Regression

***STEP 4.***

Let’s start with *Bayesian Linear Regression* first. Let’s add it from **Machine Learning > Initialize Model > Regression** store on the left, add subsequently the *Train Model* item from **Machine Learning > Train** and connect:

***STEP 5.***

The Train Model items requires to select which columns it will use to predict the value.

The marketing department wants to send the materials only to potential customers, i.e. to those who will potentially purchase the product, so we have selected the column **cnt\_purchased**, so the model will try to predict who made the purchase based on the other 9 columns in the dataset.

To select a column, click on the Train Model on canvas, and on the Properties on the right side, click Launch column selector and select **cnt\_purchased** column.

We did not change any settings for Bayesian Linear Regression model – we have left Regularization weight at 1 - it is used to prevent overfitting.

***STEP 6.***

And now we need to score the model – it will generate a prediction value using the above mentioned regression model. For image classification models, the score might be the class of object in the image, or a Boolean indicating whether a particular feature was found.

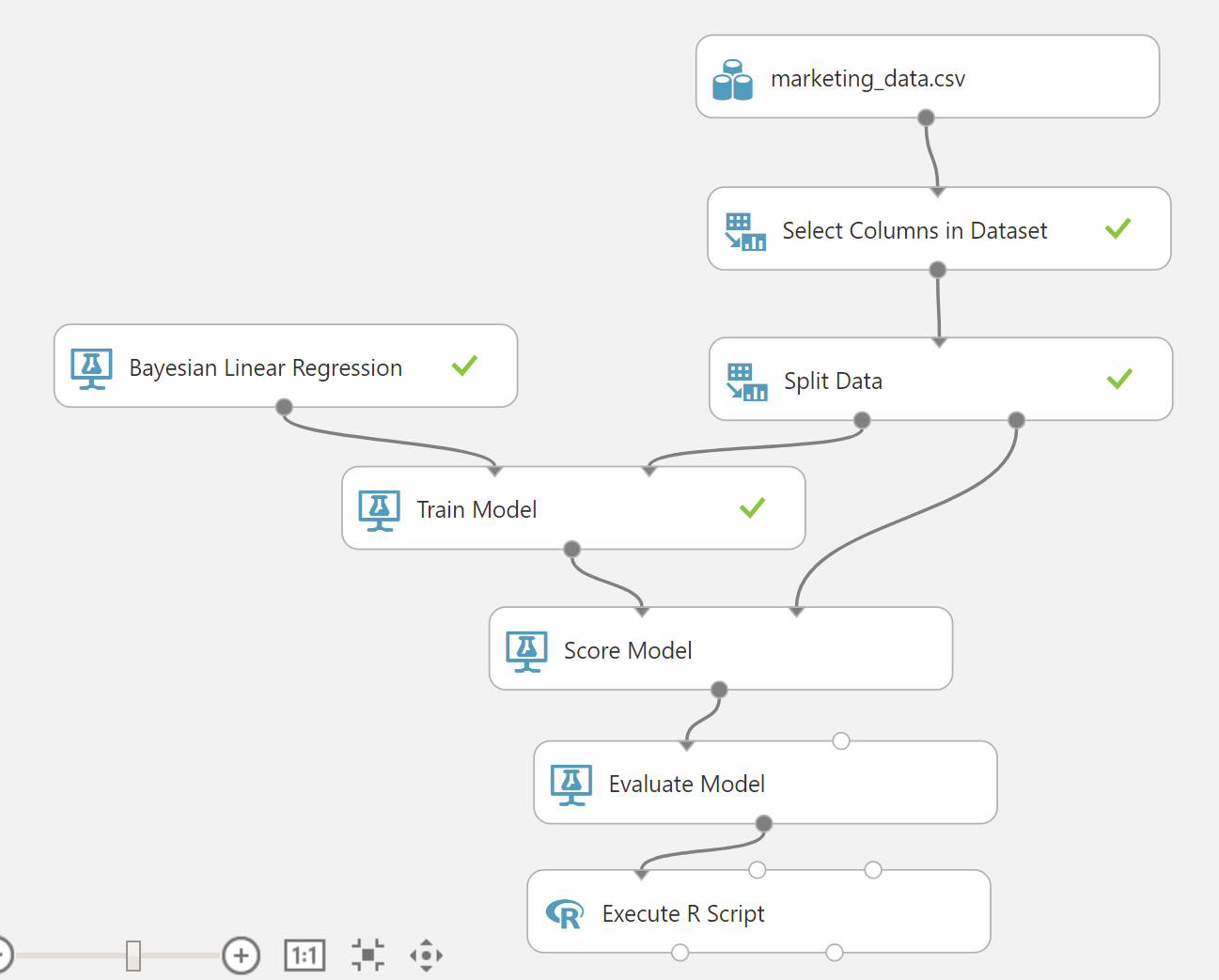
To score the model, we need a *Score Model* item from **Machine Learning – Score.** Now we need to use the testing dataset from the Split Data item and connect it to the *Score Model* item, along with the output from the *Train Model* item.

***STEP 7.***

And finally, we will use *Evaluate Model* to measure the accuracy of a trained model. It will compute a set of industry-standard evaluation metric:

* **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.
* **Mean Zero One Error (MZOE)** indicates whether the prediction was correct or not. In other words: ZeroOneLoss(x,y) = 1 when x!=y; otherwise 0
* **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.

Add the *Evaluate Model* item from **Machine Learning – Evaluate,** and connect the Score Model to it, as per below:



***STEP 8.***

The output of the Evaluate Model is a dataset of the above mentioned metrics. Since we are going to add multiple regression models to this experiment, we need to add a column to this dataset that describes the algorithm used. And to do that, we will add either an R script or Python script that will add column to the resulting dataset. For this model, let’s add an R script. Add *Execute R Script* item from **R Language Modules,** and paste this code to the Properties of R script, on the right side of your screen, to be executed:

#Get the metrics from the Evaluate model – map the input data to port 1

dataset <- maml.mapInputPort(1)

# Add algorithm name as a column to data frame

data <- data.frame(Algorithm='Bayesian Linear Regression')

data <- cbind(data, dataset[2:6])

#output

maml.mapOutputPort("data");

Add this comment to the item by double-clicking it: Get Performance Metrics – R.

Now we can run and test this first regression model. The easiest way to run it is to click **RUN** button at the bottom of the screen .Or you can test your experiment and run individual items by right-clicking on each of them, and just run the portion of the experiment, up to the item you have right-clicked. So let’s test first everything up to the Split Data item. Right-click on it, and select **Run Selected**, as per below:

It was successful, so next let’s try to run the entire model. Click Run at the bottom of screen, and once it is done, check the output of the final item – *Execute R Script*. Right-click at the output node, select **Visualize**.

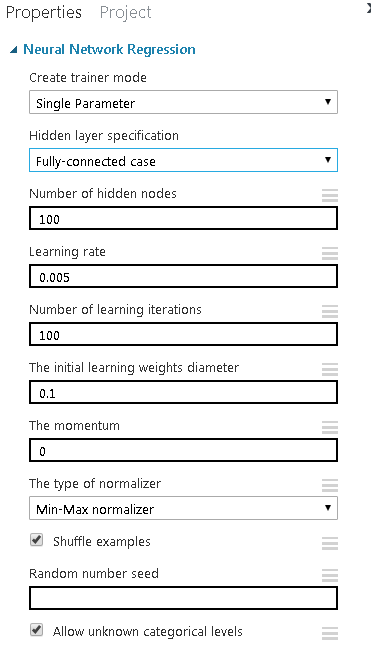
Below are the results of the training model using the Bayesian Linear Regression:



Now let’s compare it with other three regressions. To add three additional Train models, we will for the most apart repeat Steps 4 – 8, with small changes to algorithms used.

Let’s add three remaining regression types to the experiment. From **Machine Learning – Initialize Model – Regression** store on the left, drag **Decision Forest Regression**, **Linear Regression**, and **Neural Network Regression** items to the canvas, and place them one next to each other. And repeat steps 5 – 8, and add additional items as per steps above: **Train Model, Score Model, Evaluate Model**, and this time, instead of R script, let’s add **Execute Python Script**.

Each of the regression models has attributes to set, but we kept them all at default values, for example, for Neural Network, the properties are as per below:



but we have kept them all at default values, as it seemed to be fine.

***STEP 5*** - Connect all training datasets to *Train Model* items, and connect respective algorithms to *Train Models*.

***STEP 6*** – Connect all *Train Models* outputs to *Score Model* inputs 1, and connect testing datasets to *Score Model* inputs 2.

***STEP 7*** – Connect all *Score Models* outputs to *Evaluate Model* inputs.

***STEP 8*** – Drag three *Execute Python Script to canvas and c*onnect all *Evaluate Model* outputs to itsinputs.

Add this script to each python item, and change the column name based on the regression type:

# The script MUST contain a function named azureml\_main

# which is the entry point for this module.

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):

#New column, change here the name of the model for other scripts

Algorithm = ['Linear Regression']

df2 = dataframe1

#add column to dataframe

df2['Algorithm'] = Algorithm

cols = df2.columns.tolist()

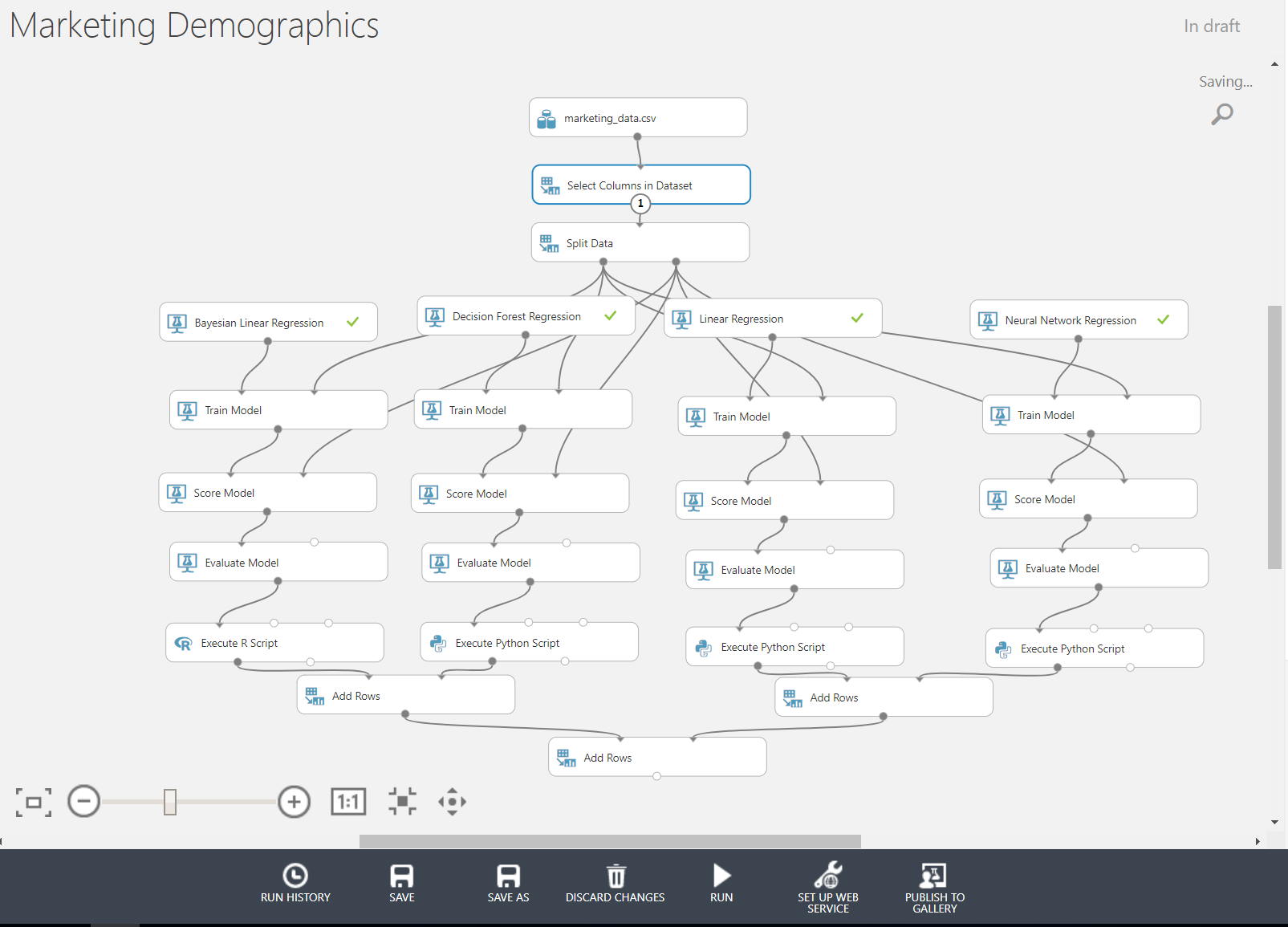
cols = cols[-1:] + cols[:-1] # cols[-1:] + cols[1:-1] for Dec Forest Reg

df = df2[cols]

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

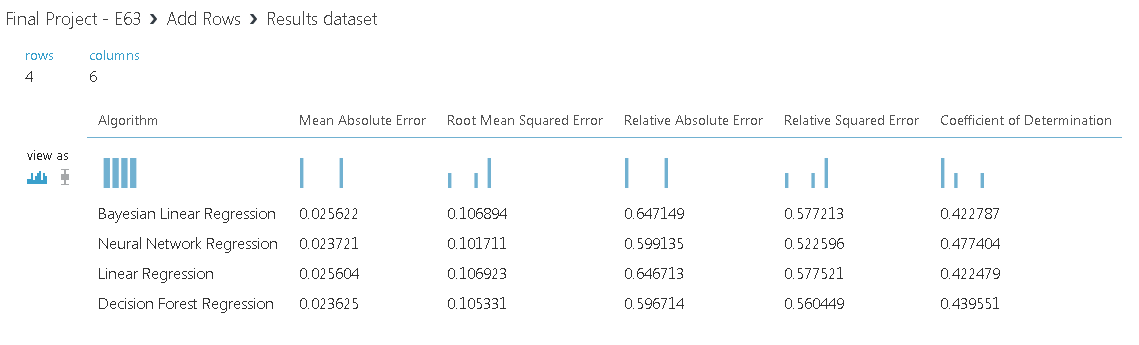
return df

Finally, we will add *Add Rows* item to collect the scores of each model and get them into one dataset. We need three *Add Rows* items as one can only combine two inputs into one, as per the screenshot below:



Now let’s run the entire experiment and check the results. Click on the *RUN* button at the bottom and wait until completed.

Once the experiment is done, you can check the results by right-clicking on the last *Add Rows* output node, and selecting *Visualize*. Below is the screenshot of the results.



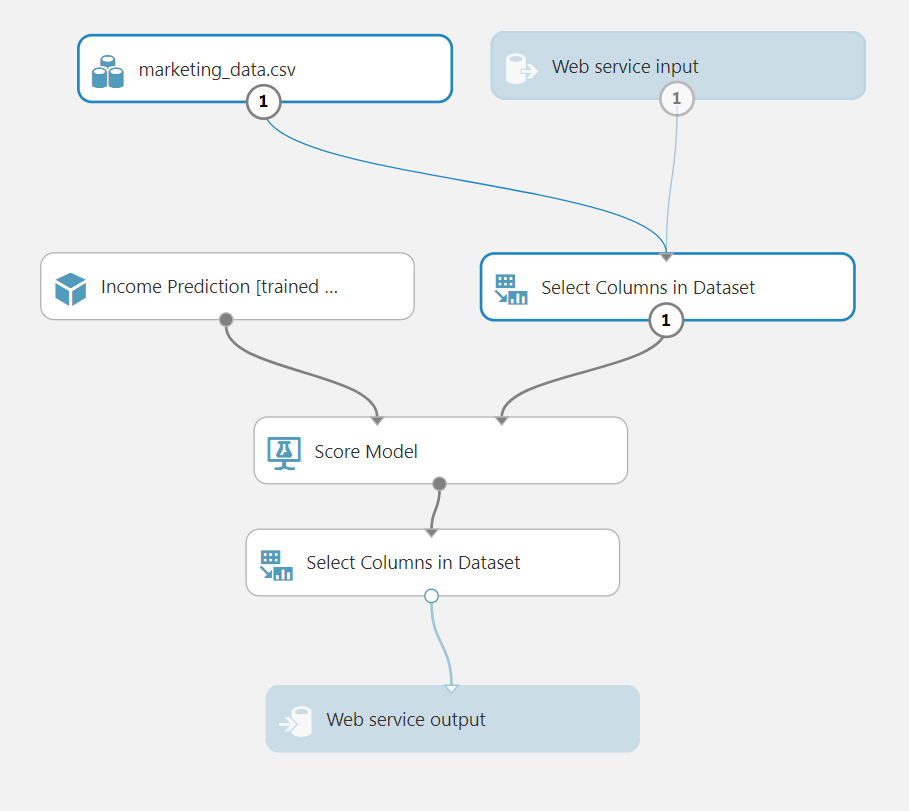
So based on the results above, each regression model did a pretty good and even job, but the best one was Neural Network: second lowest MAE, lowest RMSE, second lowest RAE, lowest RSE, and best CoD.

1. **Results**

The next step is to publish the best model and set it up as a consumable web service, to which you can pass data and it will return the results through a web service too.

Click on the Train Model item for Neural Network Regression, and then click on the *SET UP WEB SERVICE* button in the ribbon at the bottom, and select *Predictive Web Service (Recommended)*.

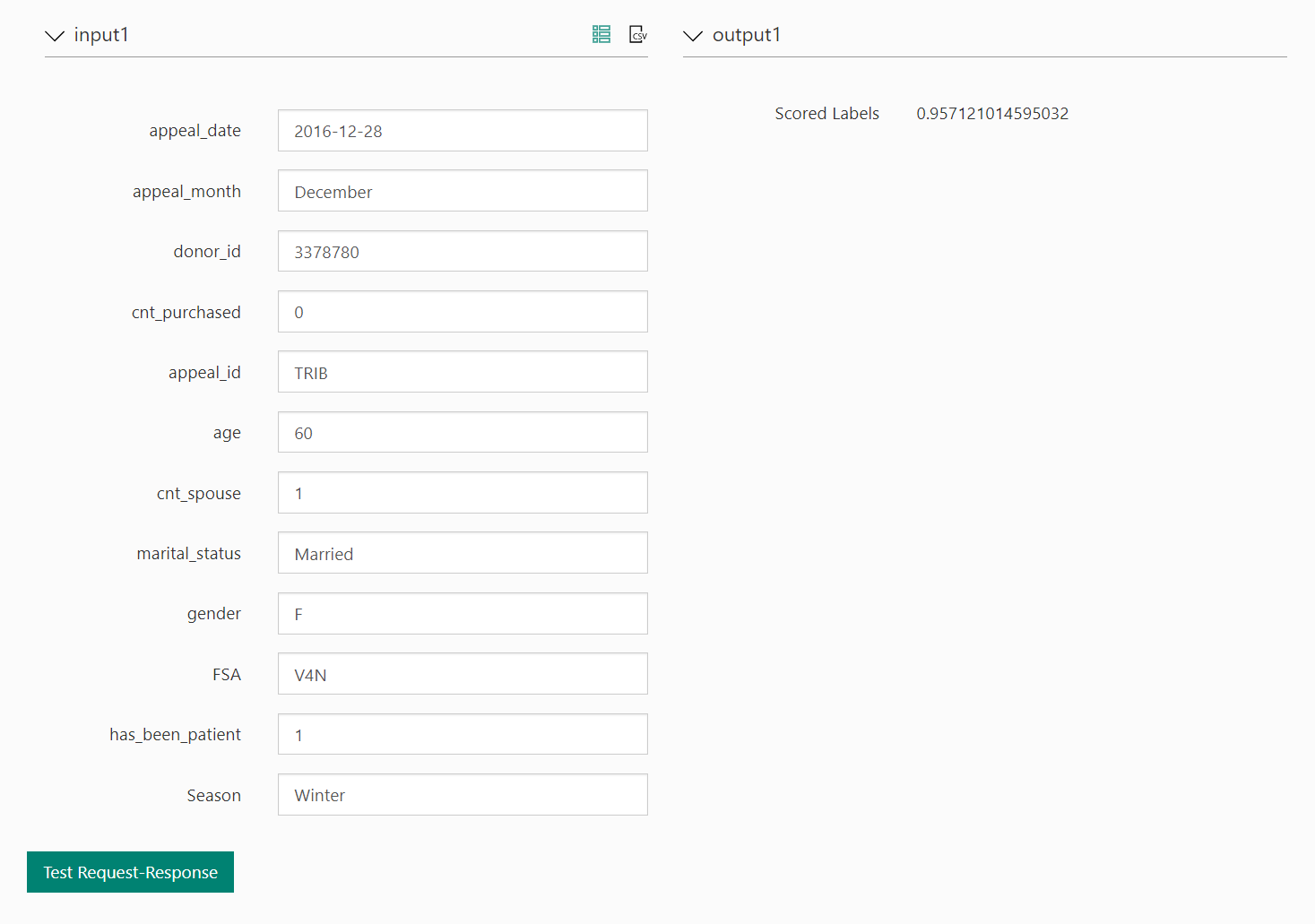
After the button is clicked, the Training Experiment will be converted to a Predictive Experiment.



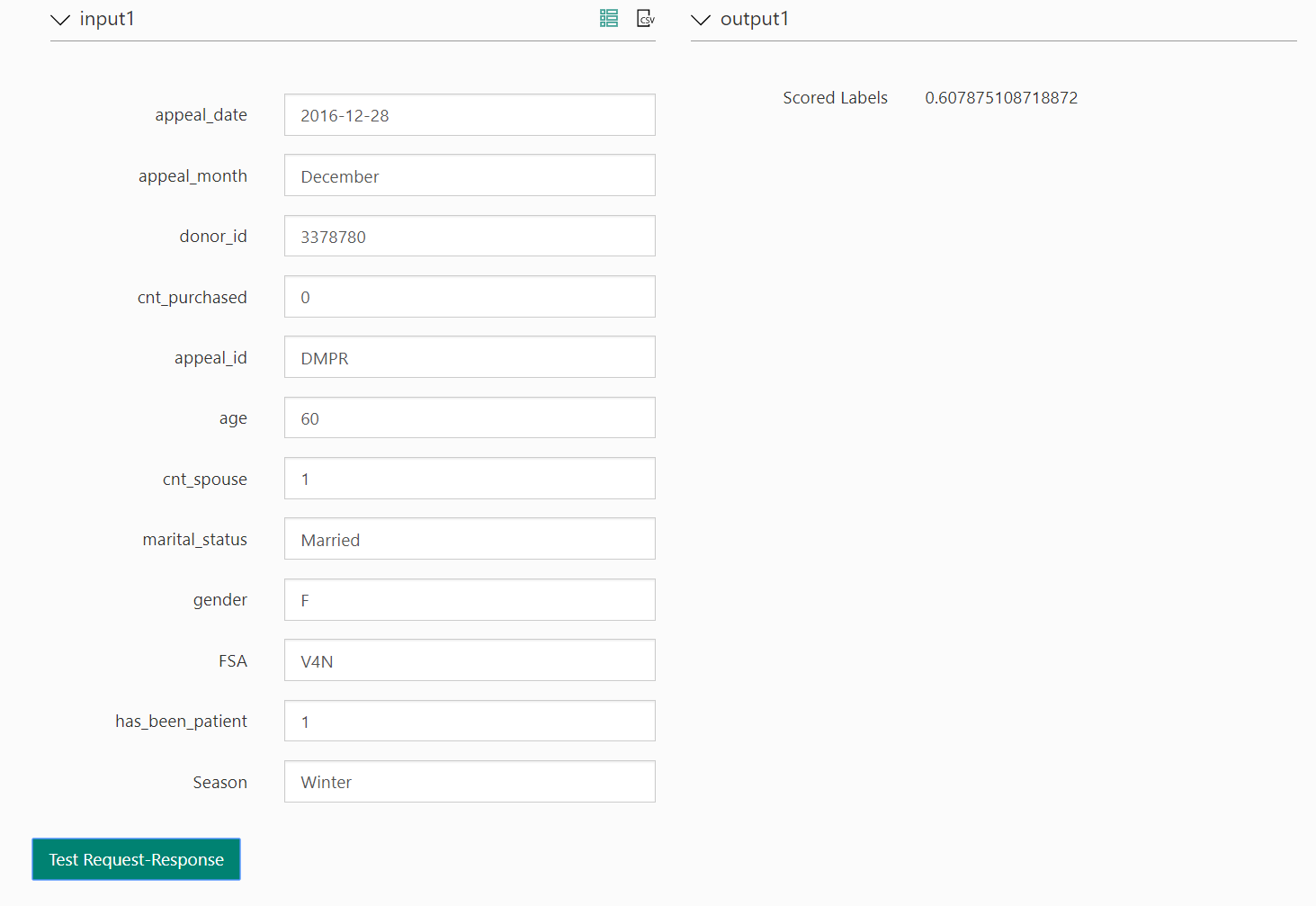
We have added one more item to the experiment, *Select Column in Dataset*, between the *Score Model* and *Web service output items*, which selects only the Score from the model and returns it, instead of the all columns.

Now we can Deploy Web Service and consume it. But first let’s test it and see what the output of it is. Click on the button Deploy Web Service, and once the processing is done:

This is your new web service, with an API key, that needs to be used in your python script to access it. Before writing a script, let’s test it. Click on the link-button **Test Preview** as per above. A new window will appear and you will enter (or select, if you enable *Sample Data*) values for columns as per CSV file. And once you enter the values, the model will calculate or score your data on how likely this person is to purchase the product, based on entered values.



So for the values entered above, the likelihood for this person that will make a purchase, with the specified values in the columns, is 96%. If you change the value in one column, for example, change the appeal\_id from TRIB to DMPR, the Score will change to 0.6078, i.e. 61%, as per below.



Let’s test the web service now using the python script. The Web service is an Azure web service that can receive and return data using REST APIs in one of two ways:

* **Request/Response** - The user sends one or more rows of credit data to the service by using an HTTP protocol, and the service responds with one or more sets of results.
* **Batch Execution** - The user stores one or more rows of credit data in an Azure blob and then sends the blob location to the service. The service scores all the rows of data in the input blob, stores the results in another blob, and returns the URL of that container.

We will do it using the Request/Response. Below is the script to use (for Python 3.+). we am using the same values to test as in previous test:

import urllib.request

import json

data = {

"Inputs": {

"input1":

[

{

'appeal\_date': "2016-12-28",

'appeal\_month': "December",

'donor\_id': "3378780",

'cnt\_purchased': "0",

'appeal\_id': "TRIB",

'age': "65",

'cnt\_spouse': "1",

'marital\_status': "Married",

'gender': "F",

'FSA': "V4N",

'has\_been\_patient': "1",

'Season': "Winter",

}

],

},

"GlobalParameters": {

}

}

body = str.encode(json.dumps(data))

url = 'https://ussouthcentral.services.azureml.net/workspaces/13cd50a296274e3abab663495278a8cd/services/90d9f9864d734620b9ed540023857f7d/execute?api-version=2.0&format=swagger'

api\_key = '9pEEb/rfIpRAw4Yvl25mDCfPoNSo+nSEtpAX2wPlvOPBG50Uwv46itNLLAV8ncvKsxDkN3W/crVbJStkdG9UpQ==' # Replace this with the API key for the web service

headers = {'Content-Type':'application/json', 'Authorization':('Bearer '+ api\_key)}

req = urllib.request.Request(url, body, headers)

try:

response = urllib.request.urlopen(req)

result = response.read()

print(result)

except urllib.error.HTTPError as error:

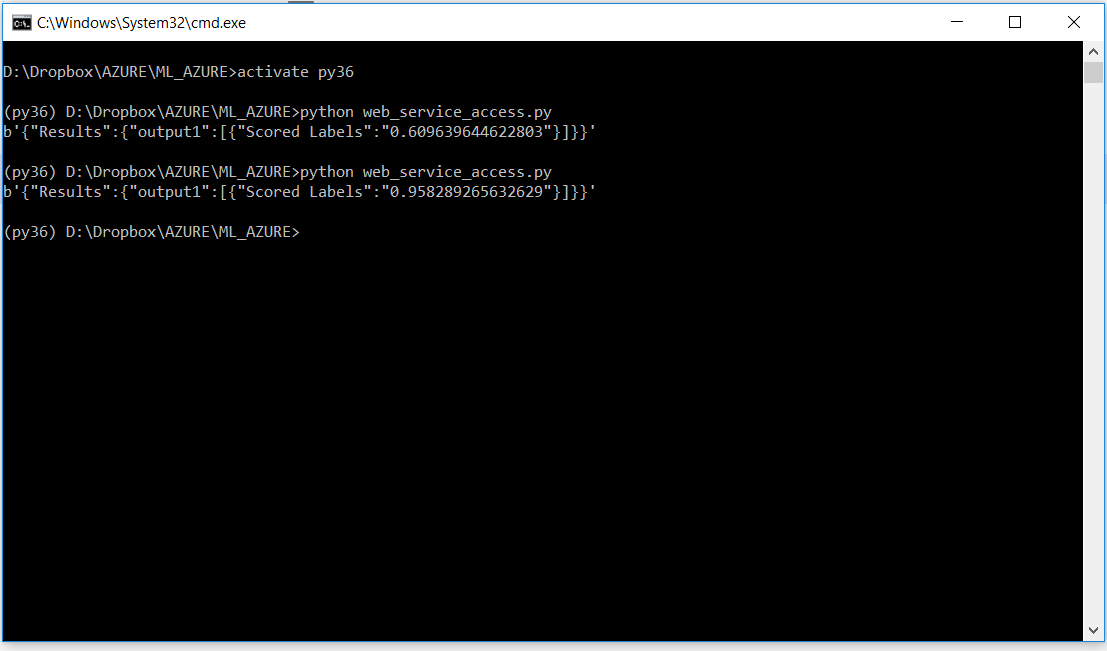
print("The request failed with status code: " + str(error.code))

# Print the headers - they include the requert ID and the timestamp, which are useful for debugging the failure

print(error.info())

print(json.loads(error.read().decode("utf8", 'ignore')))

In the script we have used the same values as in the previous test, and got the same value of prediction, as per below – 95%.



The values for URL and API key can be obtained from the Azure ML Web Service page, as per below:

