BOSTON ENERGY FORECAST

USING

Report By:

Lalit Jain

Lipsa Panda

Sameer Goel

TABLE OF CONTENTS

OBJECTIVE

Our goal is to deploy the machine learning models as a service. To accomplish our goal we will use Microsoft Azure Studio where we will create web service. We will also create User Interface which can invoke Azure web service.

INTRODUCTION OF AZURE ML

Azure Machine Learning was designed for applied machine learning. We can use algorithms in simple drag-and-drop interface and go from idea to deployment in a matter of clicks.

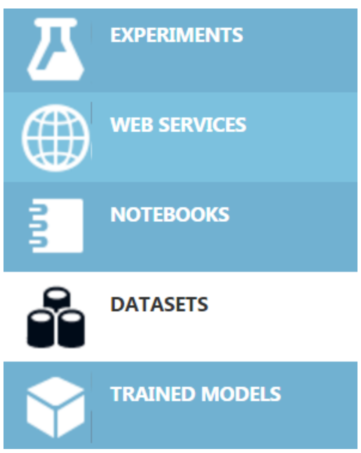
We can deploy our model into production as a web service, ml gives us a web service that can be called from any device, anywhere and that can use any data source, using REST service.

DESIGN AND IMPLEMENTATION

We have to import our csv file to Azure ML as follows:

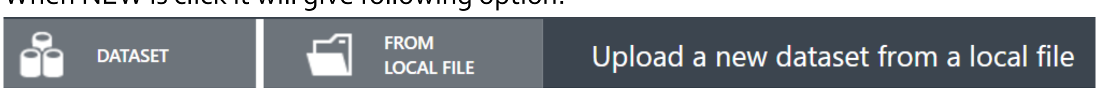
IMPORTING DATASET

1. Click 'Datasets' on the left menu of the studio.

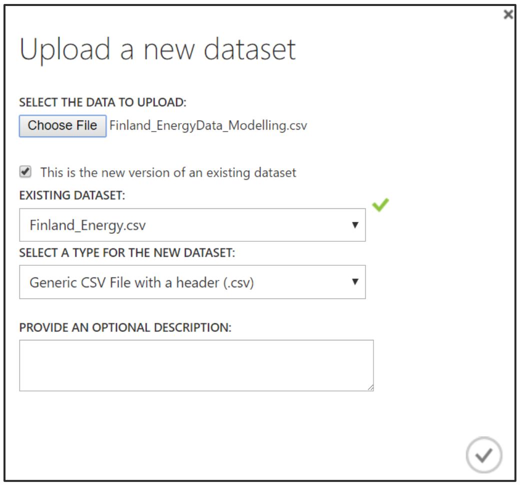


2. Now click on the '+' to import the dataset from the local storage.

3. When NEW is click it will give following option:



4. Please follow the instructions to finish import.



REGRESSION

Regression is most commonly used algorithm when there is a need a for Predicting a value. Regression tries to predict a real valued output (numerical value) of some variable for that individual. An example regression problem would be: "What will be the cost of a given house?" . The variable to be predicted here is housing price, and a model could be produced by looking at other, similar houses in the population and their historical prices.

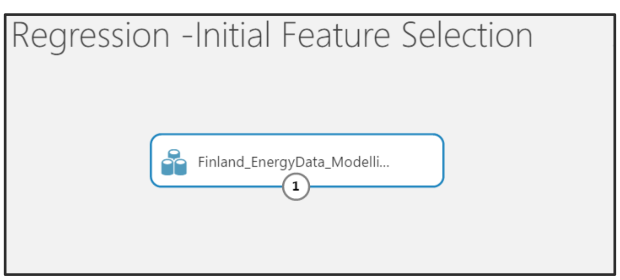
COMMON STEPS FOR ALL REGRESSION

Initial Feature Selection, Categorical Casting, split data, Train Model, Score Model, Evaluate Model are steps in all the models. We will go through the steps as follows.

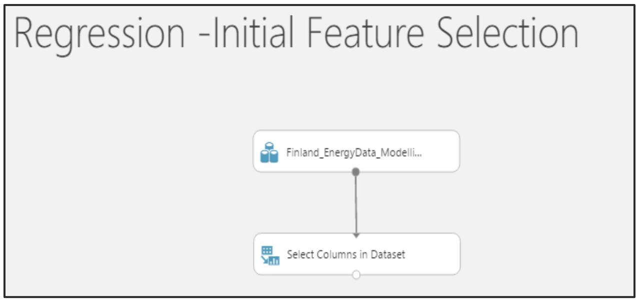
INITIAL FEATURE SELECTION EXPERIEMNT

Not every feature in its current form is expected to contain predictive value to the model, and may mislead or add noise to the model. Some low quality features were removed to improve the model's performance. Low quality includes lack of representative categories, too many missing values, or noisy features.

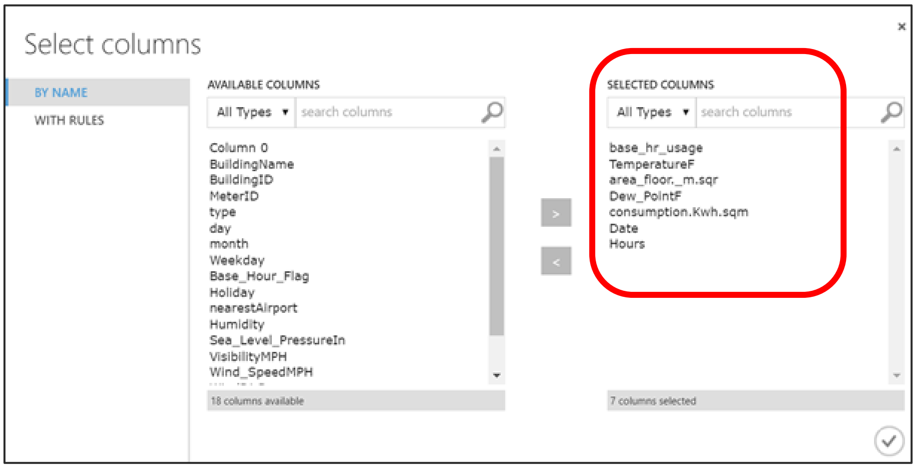
1. We start the experiment by dragging the dataset.



2. Begin by identifying columns that add little-to-no value for predictive modeling. These columns will be dropped.The columns which will be input to the model.

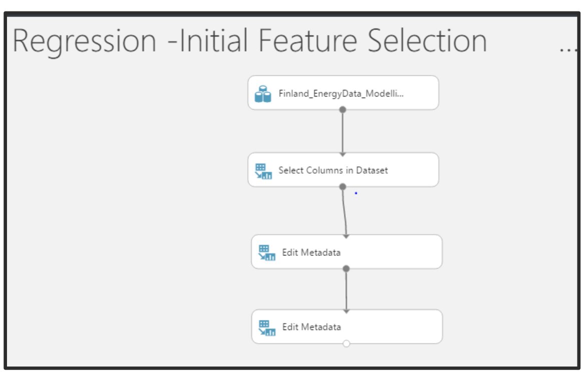


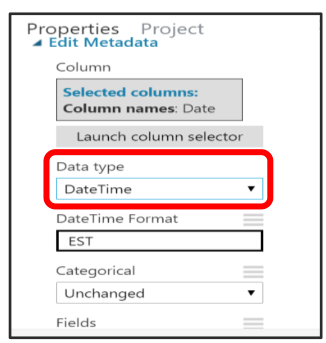
We slelect the column names from the original dataset as below:



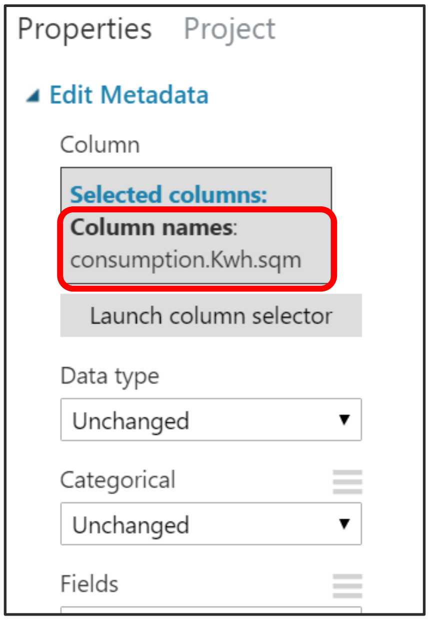
3. Categorical casting

Nominal categorical features were identified and cast to categorical data types using the meta data editor to ensure proper mathematical treatment by the machine learning algorithm.To cast these columns, drag in the metadata editor. Specify the columns to be cast, then change the "Categorical"parameter to "Make categorical".We have changed the Date column to data type "**DateTime**".





Tell Azure ML what it is trying to predict by casting the response class into a label using the metadata editor module.

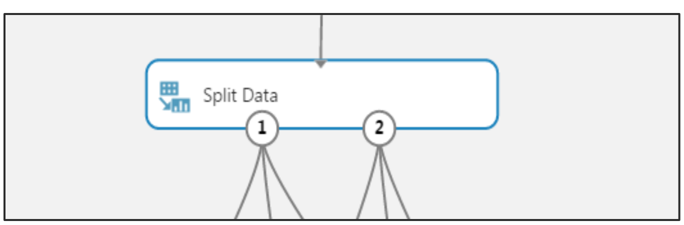


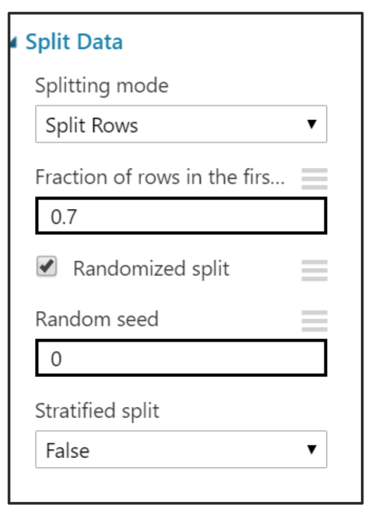
We have chosen Consumption.Kwh.sqm. as the response class to predict the value.

SPLIT THE DATA

It is extremely important to randomly partition your data prior to training an algorithm to test the validity and performance of your model. A predictive model is worthless to us if it can only accurately predict known values. Withhold data represents data that the model never saw when it was training its algorithm. This will allow you to score the performance of your model later to evaluate how well the model can predict future or unknown values.Randomly split and partition the data into 70% training and 30% scoring using the split module.

Drag in a “Split” module. It is usually industry practice to set a 70/30 split. To do this, set “fraction of rows in the first output dataset” to be 0.7. 70% of the data will be randomly shuffled into the left output node, while the remaining 30% will be shuffled into the right output node.

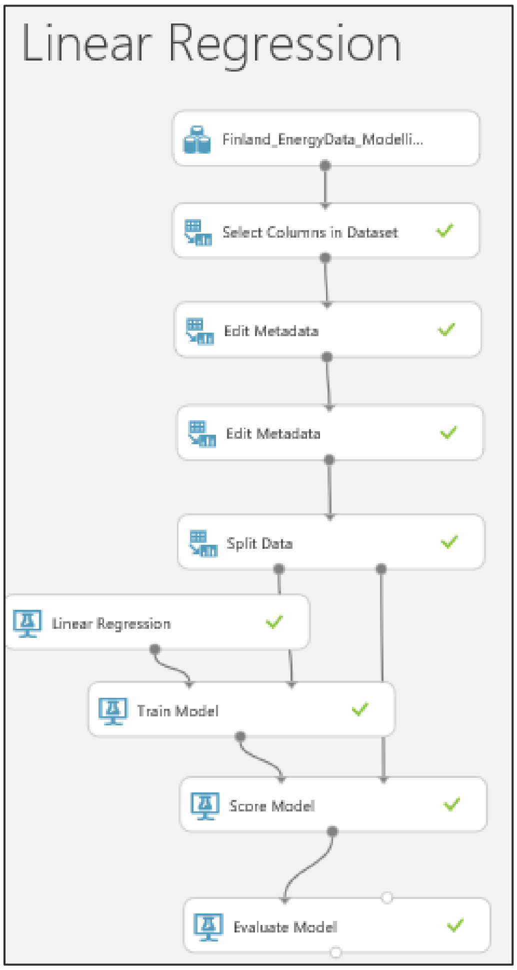


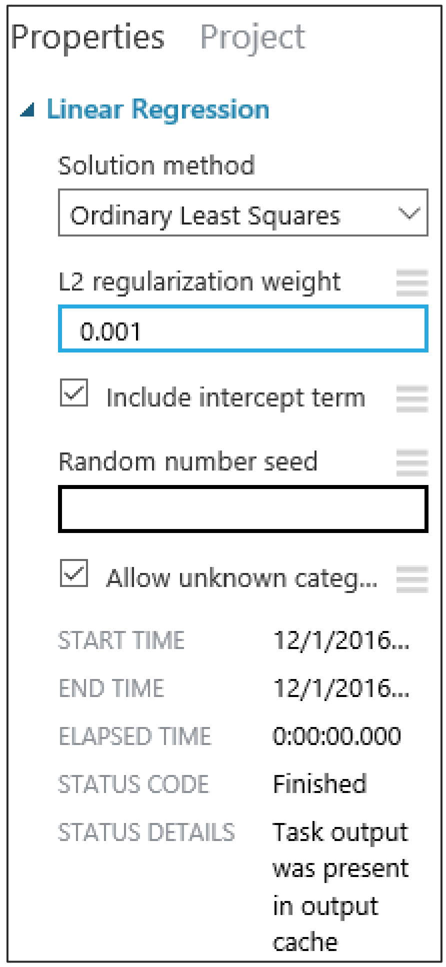


Now that the data is ready, constructing a predictive model consists of training and testing. We'll use our data to train the model, and then we'll test the model to see how closely it's able to predict.

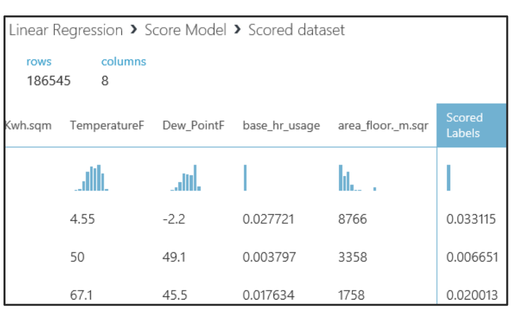
LINEAR REGRESSION

Regression is a machine learning used to predict a numeric outcome. Linear regression attempts to establish a linear relationship between independent variables and an outcome variable, or *dependent variable*, which is also numeric.





The 'Score Model' output will fetch us scored labels, which is a predicted outcome on 30% of the test data.

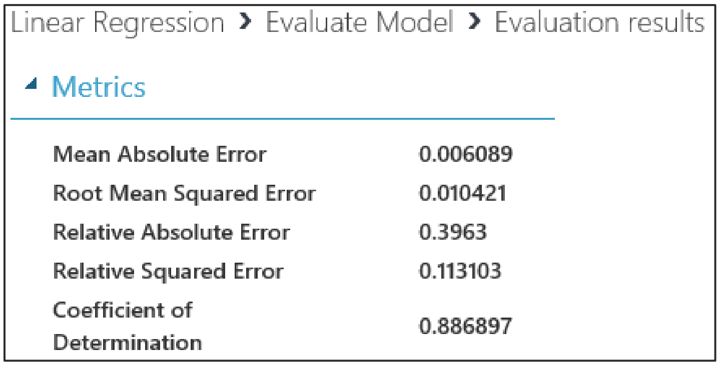


Linear Regression › Score Model › Scored dataset

rows 186545 columns 8

Kwh.sqm	TemperatureF	Dew_PointF	base_hr_usage	area_floor._m.sqr	Scored Labels
	4.55	-2.2	0.027721	8766	0.033115
	50	49.1	0.003797	3358	0.006651
	67.1	45.5	0.017634	1758	0.020013

Evaluation model gives us the matrices like MAE, RMSE, RSS. Relative squared error is 0.113103

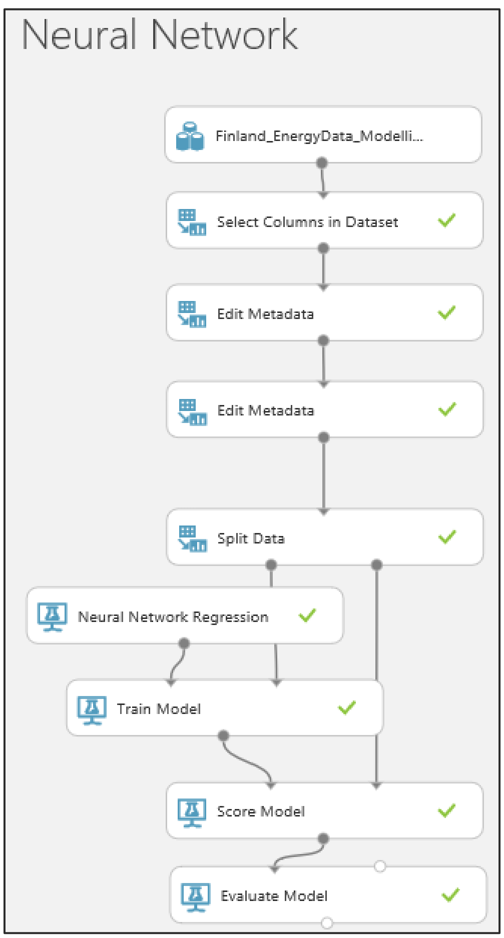

Linear Regression › Evaluate Model › Evaluation results

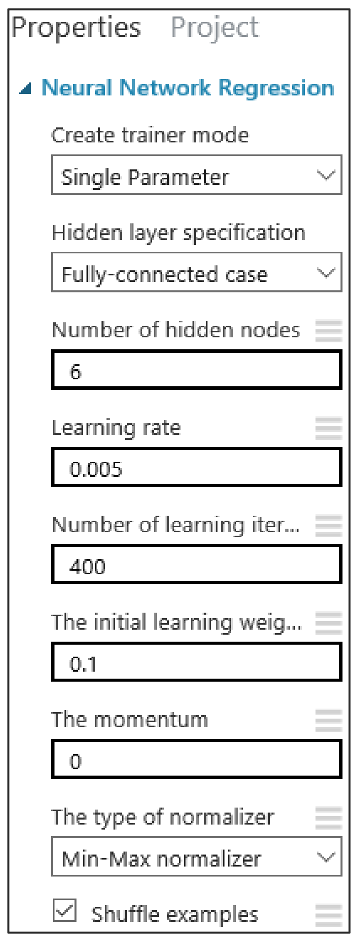
Metrics

Mean Absolute Error	0.006089
Root Mean Squared Error	0.010421
Relative Absolute Error	0.3963
Relative Squared Error	0.113103
Coefficient of Determination	0.886897

NEURAL NETWORK REGRESSION

Neural networks are widely known for use in deep learning and modeling complex problems such as image recognition, they are easily adapted to regression problems. Any class of statistical models can be termed a neural network if they use adaptive weights and can approximate non-linear functions of their inputs. Thus neural network regression is suited to problems where a more traditional regression model cannot fit a solution.





The ‘Score Model’ output will fetch us scored labels, which is a predicted outcome on 30% of the test data.

Neural Network › Score Model › Scored dataset

rows	columns
186545	8

Kwh.sqm	TemperatureF	Dew_PointF	base_hr_usage	area_floor._m.sqr	Scored Labels
	4.55	-2.2	0.027721	8766	0.029901
	50	49.1	0.003797	3358	0.00428
	67.1	45.5	0.017634	1758	0.023133

Evaluation model gives us the matrices like MAE, RMSE, RSS. Relative squared error is 0.105004.

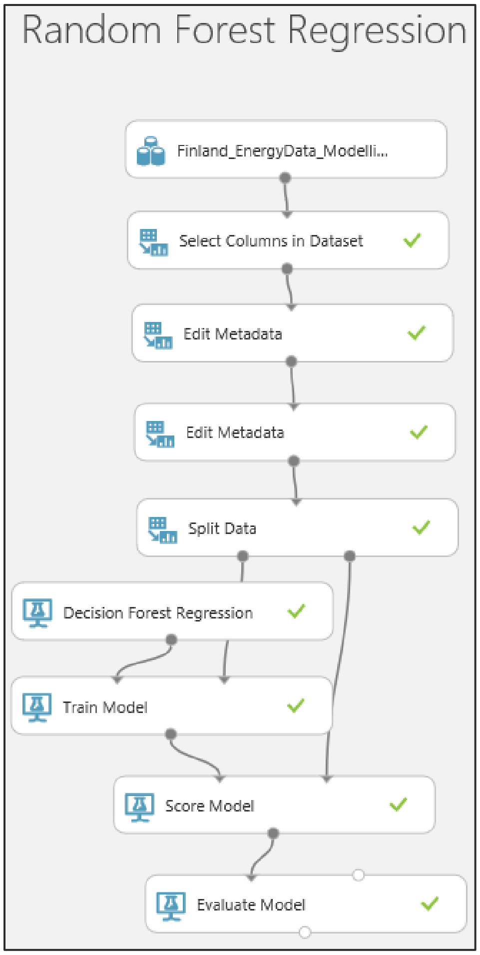
Neural Network › Evaluate Model › Evaluation results

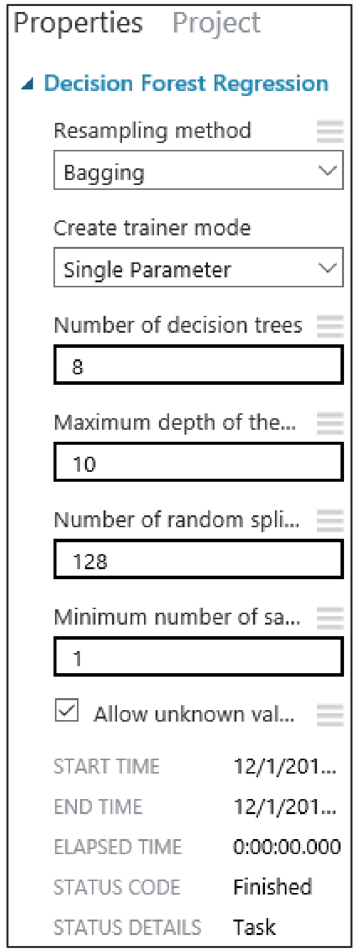
Metrics

Mean Absolute Error	0.005939
Root Mean Squared Error	0.010041
Relative Absolute Error	0.386529
Relative Squared Error	0.105004
Coefficient of Determination	0.894996

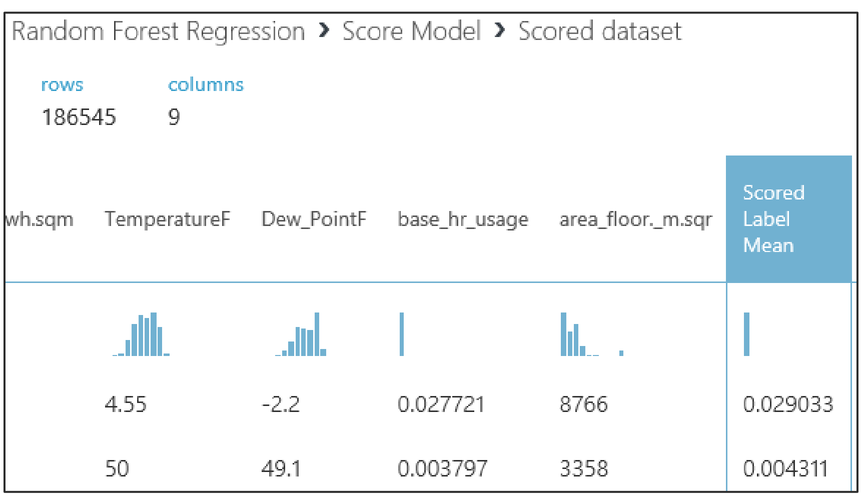
RANDOM FOREST REGRESSION

Decision trees are non-parametric models that perform a sequence of simple tests for each instance, traversing a binary tree data structure until a leaf node (decision) is reached.

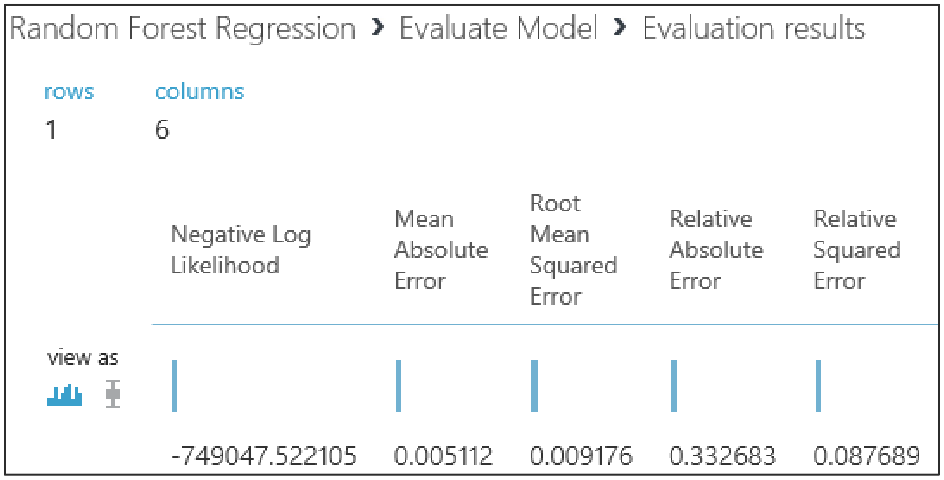




The 'Score Model' output will fetch us score labels, which is a predicted outcome on 30% of the test data.



Evaluation model gives us the matrices like MAE, RMSE, RSS. Root squared error is 0.087689.



Random Forest Regression › Evaluate Model › Evaluation results

rows 1 columns 6

Negative Log Likelihood	Mean Absolute Error	Root Mean Squared Error	Relative Absolute Error	Relative Squared Error
-749047.522105	0.005112	0.009176	0.332683	0.087689

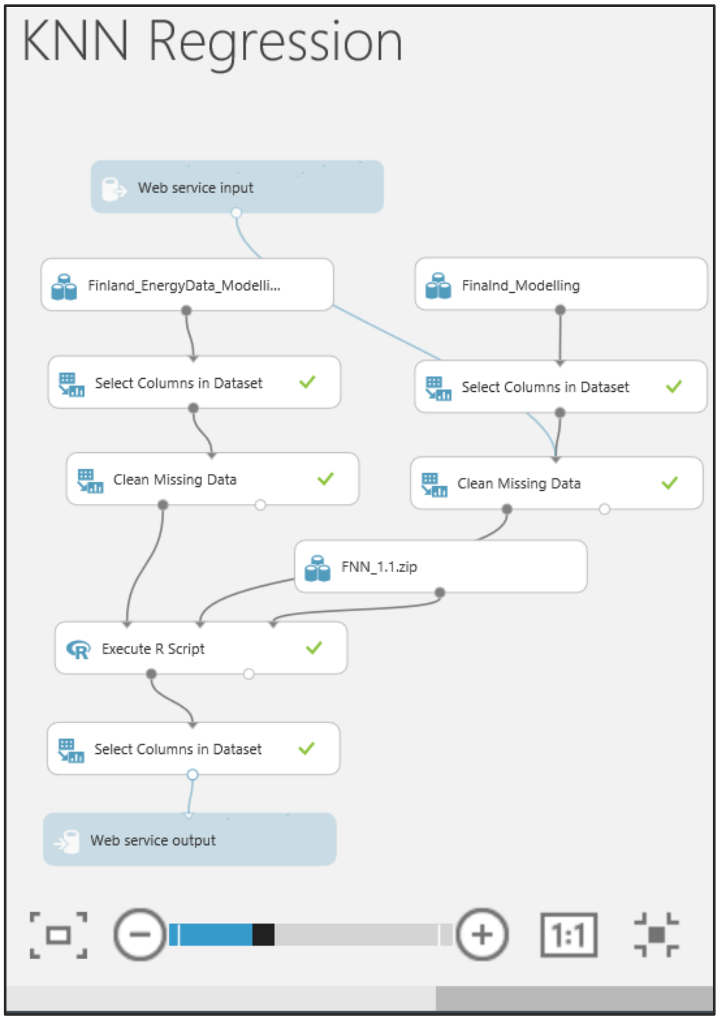
KNN REGRESSION

There is no in-built component for KNN regression provided my Azure ML studio so we will use custom function to proceed.

We will use two copies of the same dataset to make this work. The first dataset is used for training and other copy for testing.

In the training path we will use all the required features along with the predictor which is Consumption_kwh_sqm.

In the testing path we will use all the required features excluding predictor which is Consumption_kwh_sqm.

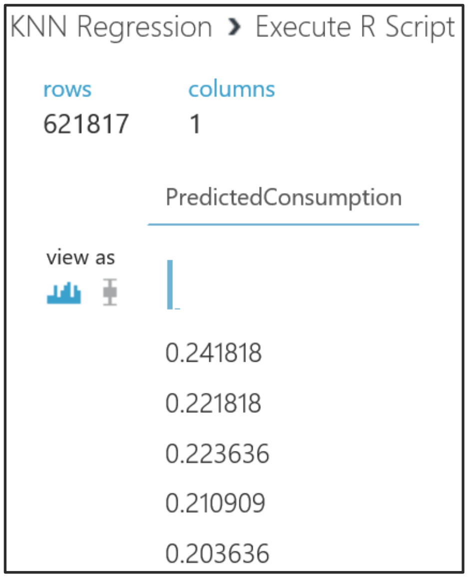


We are using R script, here we using FNN library to train the KNN model. Since we have a different route for testing we can use it as our web service input in the predictive experiment.

R Script

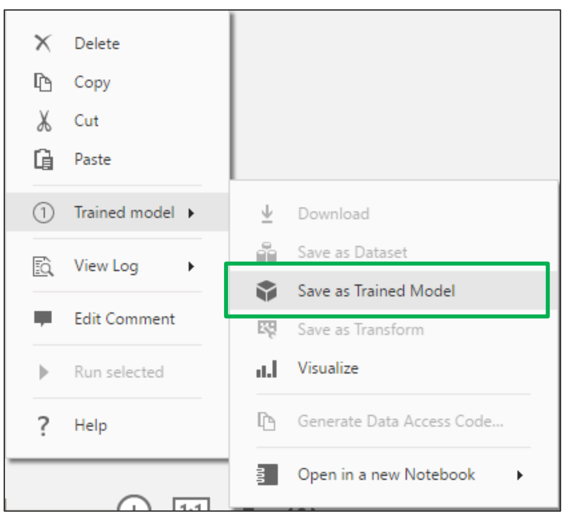
```
# Map 1-based optional input ports to variables
train <- maml.mapInputPort(1) # class: data.frame
test <- maml.mapInputPort(2) # class: data.frame

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

install.packages("src/FNN_1.1.zip", lib = ".", repos = NULL, verbose = TRUE)
library(FNN, lib.loc=".", verbose=TRUE)

## Taking columns required
train_c <- train[,!names(train) %in% c("Date","consumption.Kwh.sqm")];
test_c <- test[,!names(test) %in% c("Date")];

## Training the Model
knn.model <- FNN::knn.reg(train=train_c,test = test_c,y=train$consumption.Kwh.sqm,k=5,algorithm = c("brute"))
```

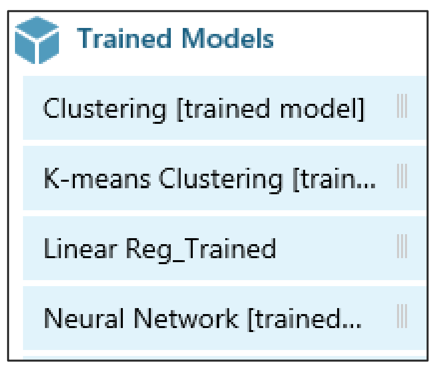
The Execute output will fetch us the predicted outcome.



Saving TRAINED MODEL

After we run all the models and is satisfied with the Scored Model output, we proceed to save the model by right clicking it.





SETUP WEB SERVICE: REGRESSION

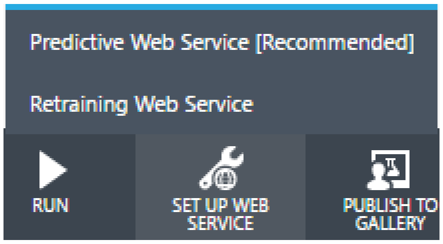
CONVERT THE TRAINING EXPERIMENT TO A PREDICTIVE EXPERIMENT

Converting to a predictive experiment involves three steps:

1. Save the model we've trained and then replace our training modules
2. Trim the experiment to remove modules that were only needed for training

3. Define where the Web service will accept input and where it generates the output

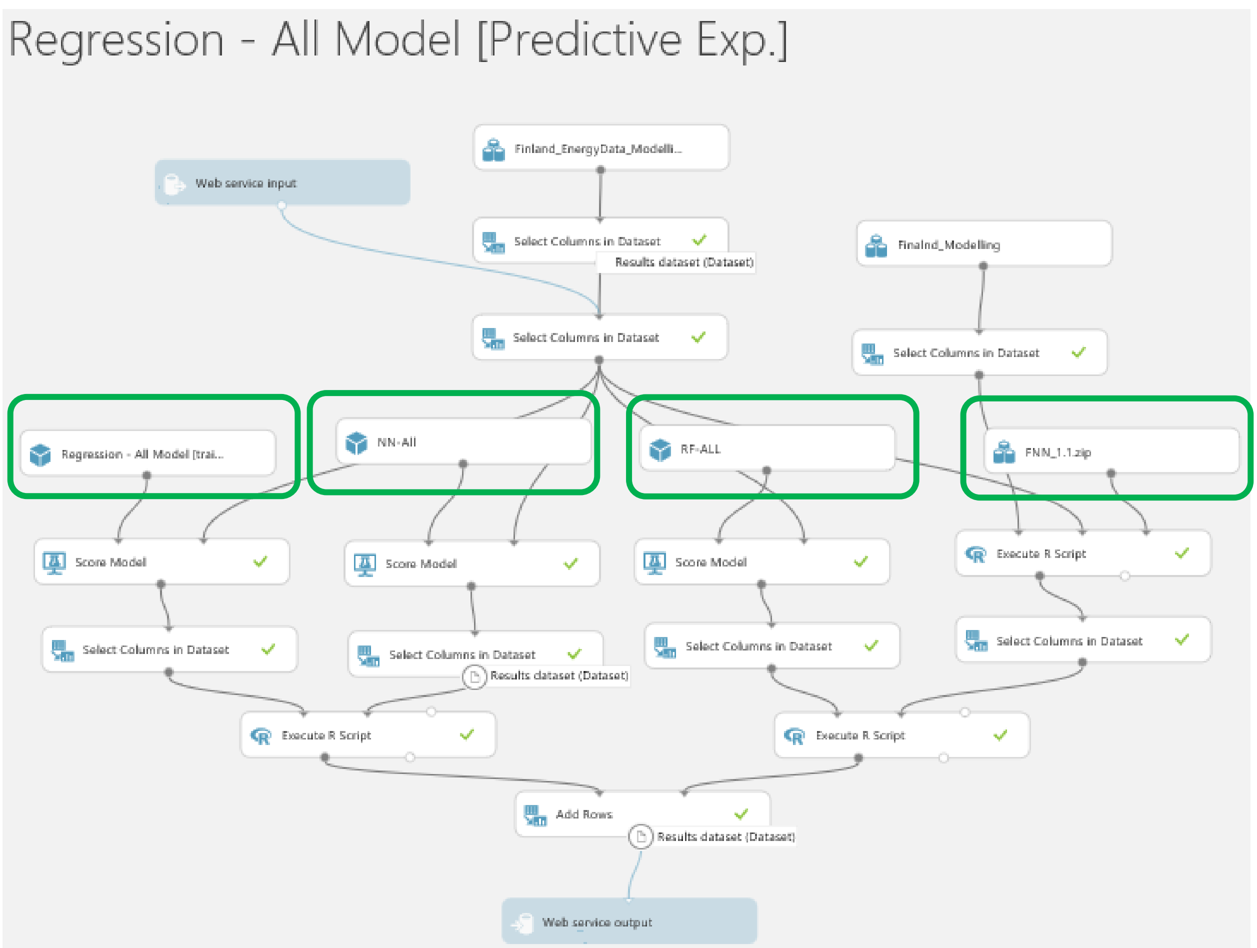
All three steps can be accomplished by clicking "Set Up Web service" at the bottom of the experiment.



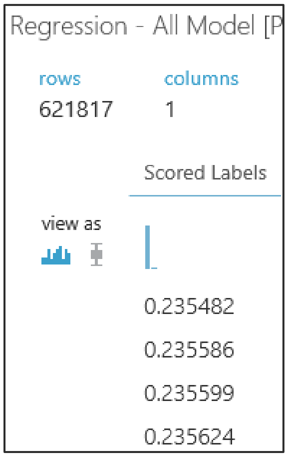
When we click Set Up Web service, several things happen:

- The trained model is saved as a single Trained Model module into the module palette to the left of the experiment canvas (we can find it under Trained Models).
- Modules that were used for training are removed.

Upon deployment our Regression model looks like below, here we have dragged our trained and saved models.



The 'scored model' component will provide Scored Probabilities which is filter in 'Select Columns Dataset' component as show below.



Since we want output to be a table with all the related methods used with related output so we will use R functionality to achieve the results. Execute R Script component does 2 things

a. Add new column 'Name' to each dataset to define the method used.
b. Row bind the 2 scored labels from the models.

R Script

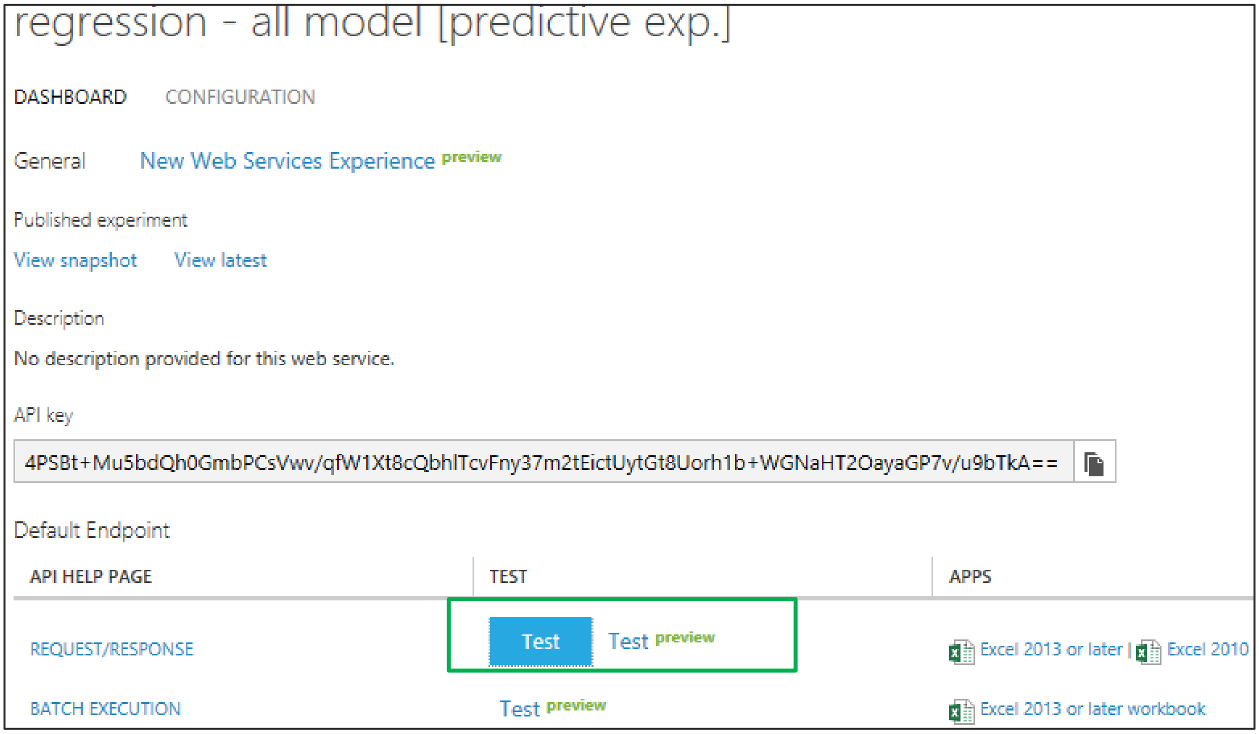
```
# Map 1-based optional input ports to variables
dataset1 <- maml.mapInputPort(1)
dataset2 <- maml.mapInputPort(2) # class: data.frame

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

# Sample operation
dataset1$Name = "Linear Regression"
dataset2$Name = "Neural Network Regression"

data.set = rbind(dataset1,dataset2)

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

This will now be tested further when deployed as a web service.

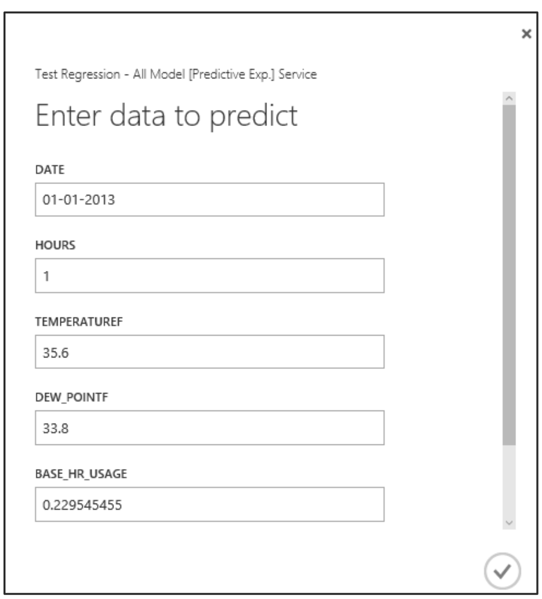
DEPLOY THE WEB SERVICE

We can deploy the experiment as either a classic Web service or a new Web service based on Azure Resource Manager.

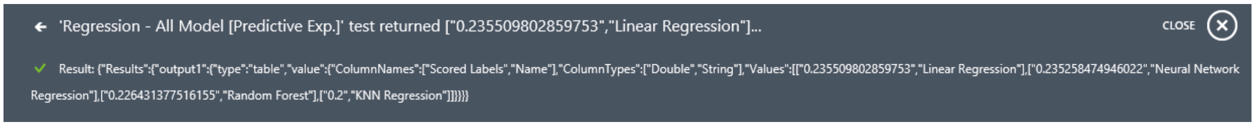
To deploy a classic Web service derived from our experiment, click **Deploy Web Service** below the canvas and select **Deploy Web Service [Classic]**. Machine Learning Studio deploys the experiment as a Web service and takes you to the dashboard for that Web service.



From here, you can return to the experiment (**View snapshot** or **View latest**) and run a simple test of the Web service (See **Test the Web service** below).



Th results will be a JSON format output.



This output shows the output from all the regression models, it has predicted conusmption for respective models.

CLASSIFICATION

When the data are being used to predict a category, supervised learning is also called classification. This is the case when assigning an image as a picture of either a 'cat' or a 'dog'. When there are only two choices, this is called **two-class** or **binomial classification**.

In our case the Boston Energy 'Base_Hour_Class' classified as "High" and "Low".

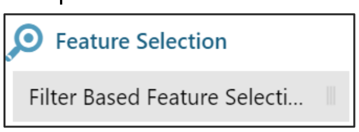
COMMON STEPS FOR ALL CLASSIFICATION

Feature selection, Normalization, Split data, Train Model, Score Model, Evaluate Model are steps in all the models. We will go through the steps as follows.

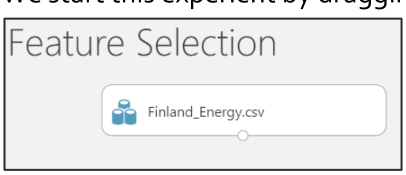
FEATURE SELECTION EXPERIEMNT

Before we proceed with our experiment, we have done feature selection in a separate experiment.

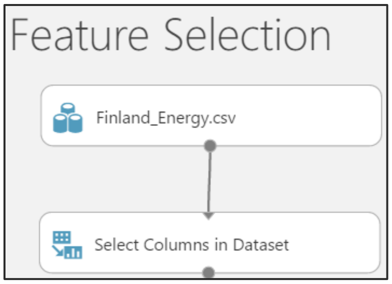
1. It is a process that is commonly applied to the construction of training data sets for predictive modeling tasks such as classification or regression tasks. The goal is to select a subset of the features from the original data set that reduces its dimensions by using a minimal set of features to represent the maximum amount of variance in the data.



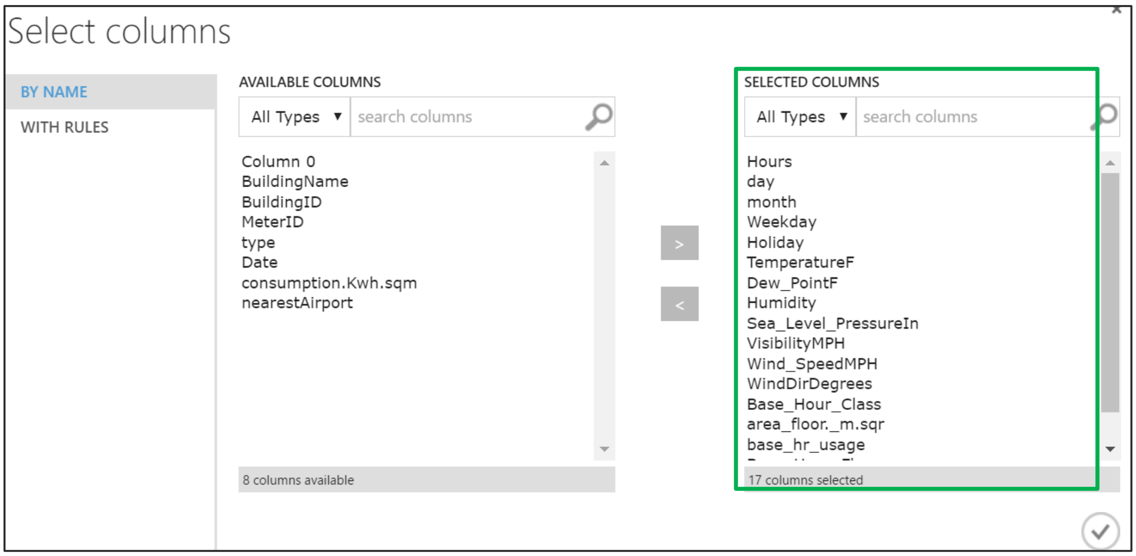
2. This subset of features contains the only features to be included to train the model. Feature selection serves two main purposes:
    - Feature selection often increases classification accuracy by eliminating irrelevant, redundant, or highly correlated features.
    - Feature selection decreases the number of features, which makes the model training process more efficient. This is particularly important for learners that are expensive to train such as support vector machines.
3. We start this experient by dragging the dataset to our experiment.



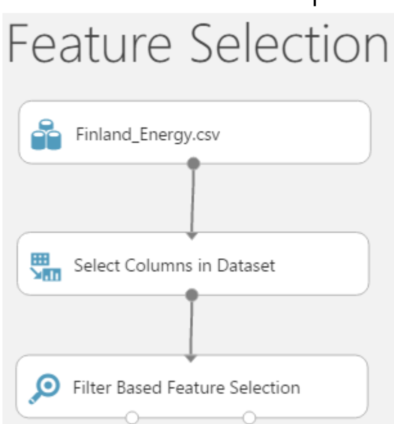
4. We then select the coulms that can be input to our feature selection



We slelect the column names from the original dataset as below:

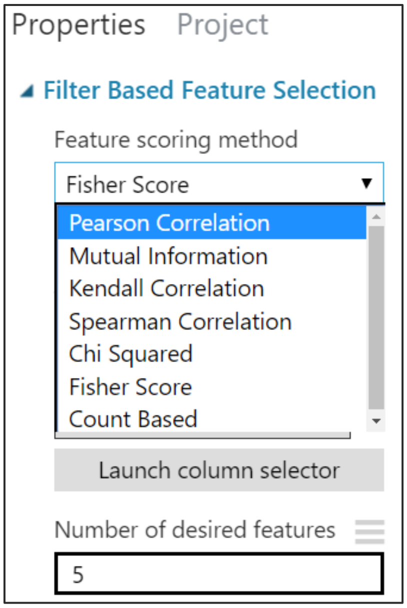


5. Now when we have the input coulumns we connect 'Feature Selection' compnent.

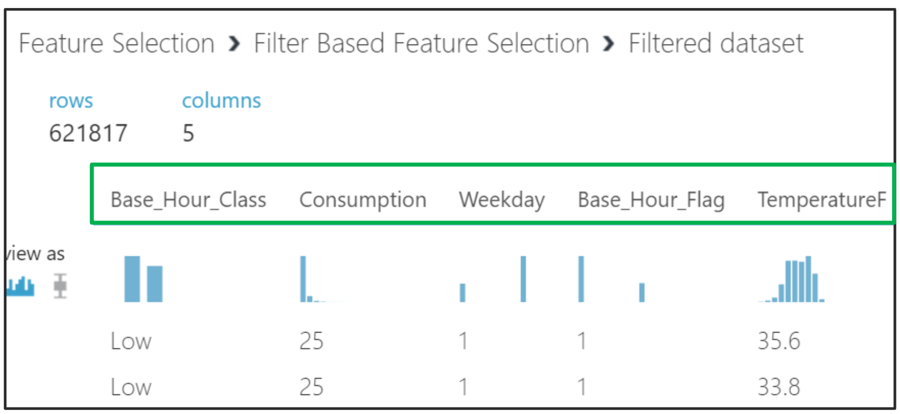


When we double click the 'Filter based feature selection' right side we see properties.

6. These properties gives us several option such as method of feature selection, number of desired features as below.



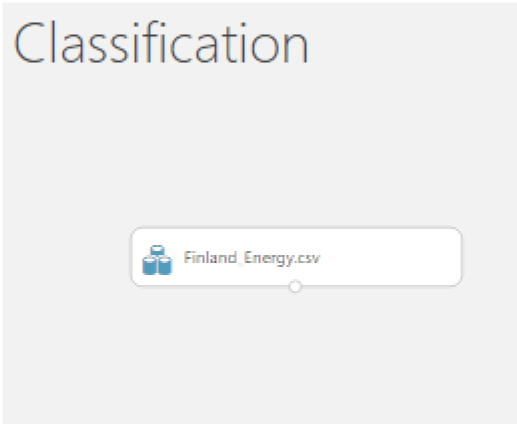
7. We can right click the 1st outfut to see the selected features as:



8. So we shortlist the above features to use it in our clustering experiment.

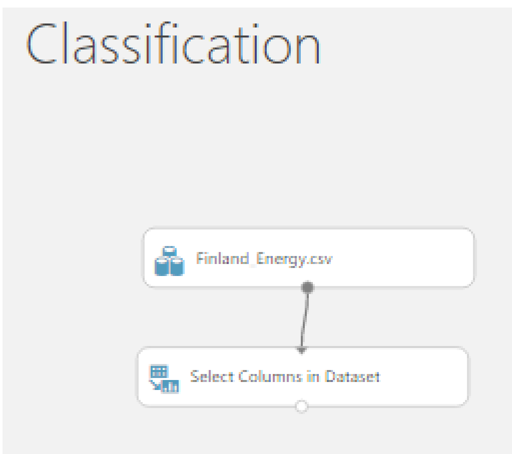
DRAG DATASET TO CLASSFICATION EXPERIMENT

We will start out our models by dragging "Finland_Energy" Dataset.



SELECT COLUMNS IN DATASET

We will select only the columns required for the modeling as we decided by Feature Selection experiment shown above.



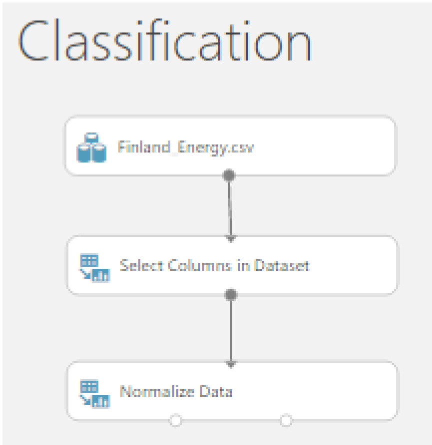
PRE-PROCESSING DATA: NORMALIZATION

In this approach, the data is scaled to a fixed range - usually 0 to 1. The cost of having this bounded range in contrast to standardization - is that we will end up with smaller standard deviations, which can suppress the effect of outliers.

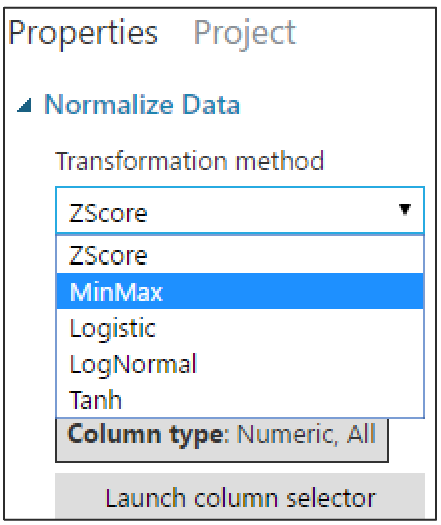
A Min-Max scaling is typically done via the following equation:

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

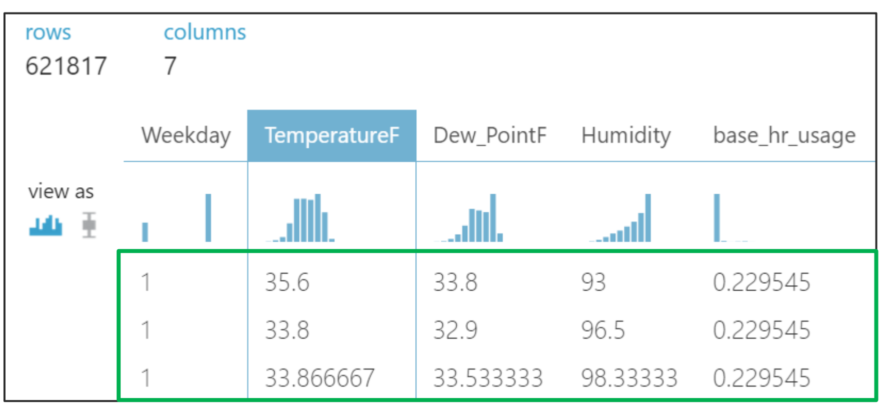
1. Azure ML provides this component where we can normalize data using various methods



We are using MinMax method for normalization as shown below:



We can visulize the columns before normalization as:



And after the normalize we van visualize and see that our fields are now distributed from 0 and 1

Weekday	TemperatureF	Dew_PointF	Humidity	base_hr_usage
1	0.529412	0.632461	0.91954	0.463303
1	0.512605	0.623037	0.95977	0.463303
1	0.513228	0.629668	0.980843	0.463303

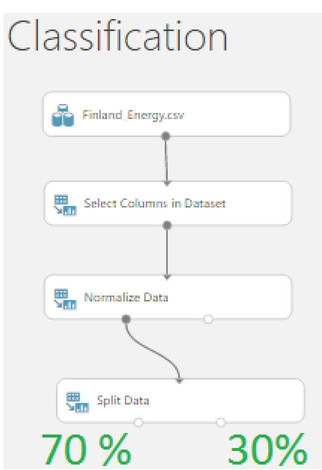
our fields are now distributed from 0 and 1

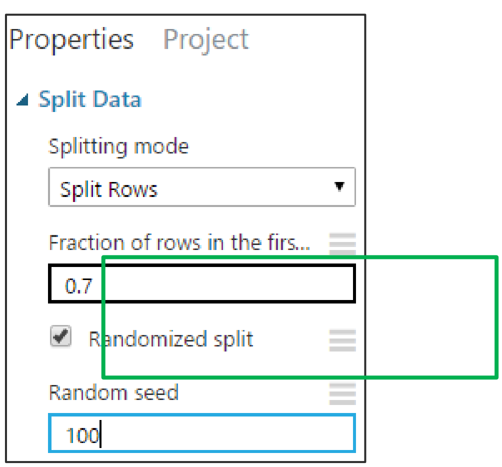
Statistics

Mean	0.6106
Median	0.6134
Min	0
Max	1
Standard Deviation	0.1658
Unique Values	653
Missing Values	0
Feature Type	Numeric Feature

SPLIT DATA: TESTING AND VALIDATION

A common strategy is to take all available labeled data, and split it into training and evaluation subsets, usually with a ratio of 70-80 percent for training and 20-30 percent for evaluation. The ML system uses the training data to train models to see patterns, and uses the evaluation data to evaluate the predictive quality of the trained model.

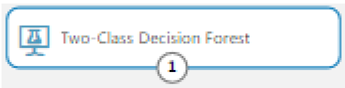




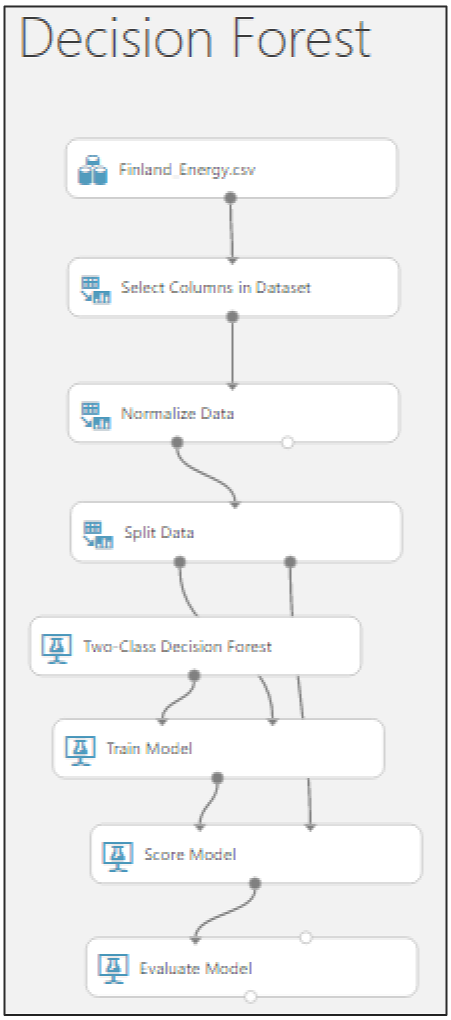
We use 70% - 30% for training and testing.

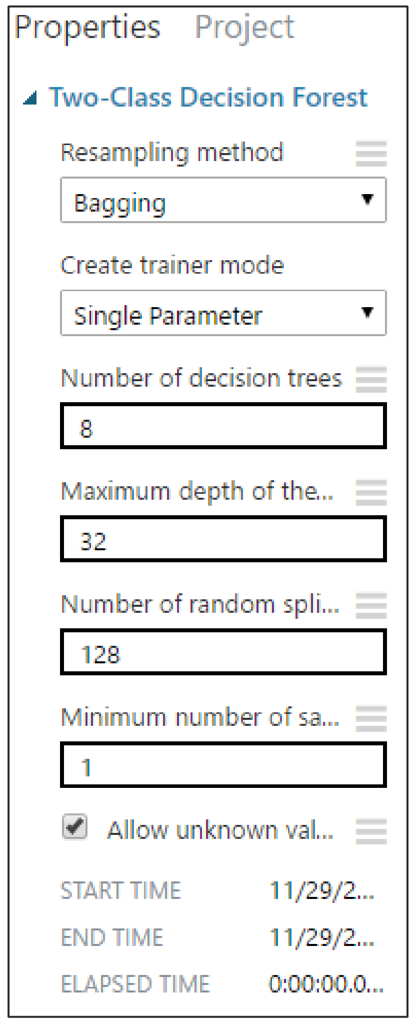
RANDOM FOREST

We will train the Random Forest model given by Azure ML.

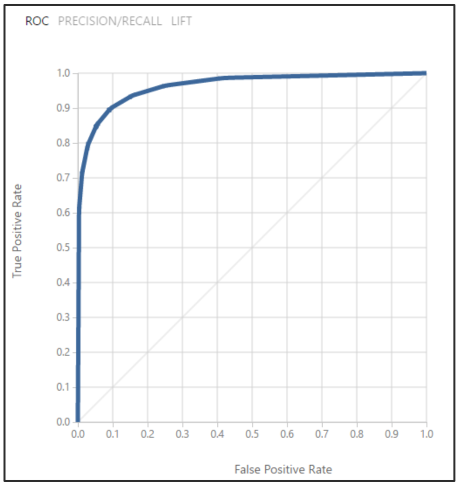


The 70% of the split data is feed into Train Model component along with Random Forest component.



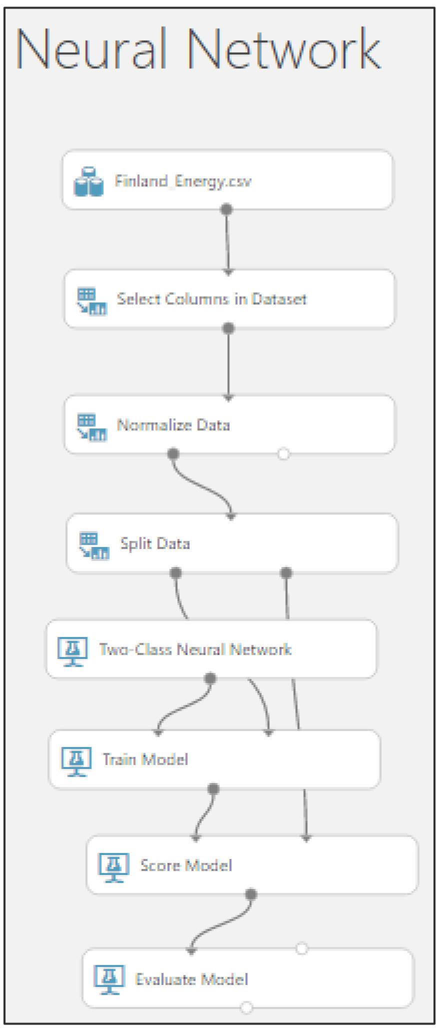


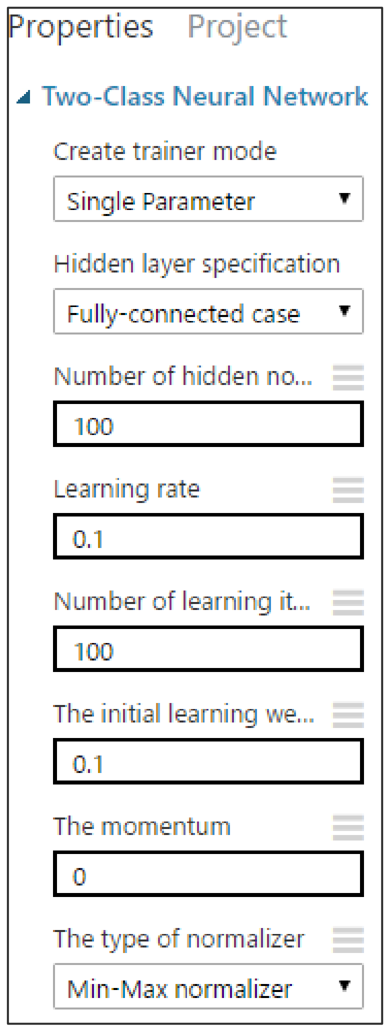
The ROC curve and accuracy is pretty decent with accuracy of 90%



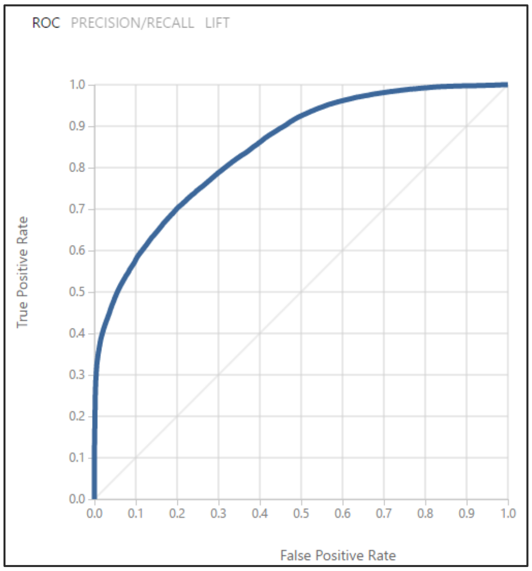
True Positive	False Negative	Accuracy	Precision
46806	7897	0.904	0.921
False Positive	**True Negative**	**Recall**	**F1 Score**
4005	65655	0.856	0.887
Positive Label	**Negative Label**		
Low	High		

NEURAL NETWORK





The ROC curve is shown is visualized as below, having a good accuracy iof 76%.

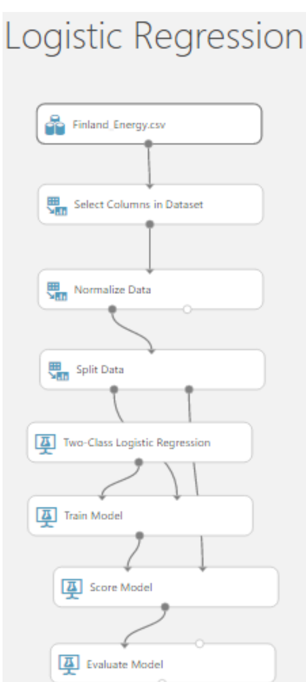


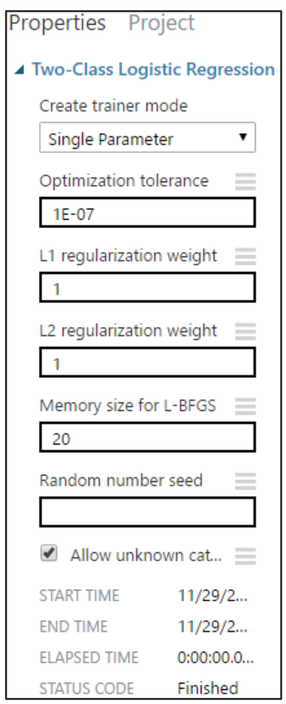
True Positive	False Negative	Accuracy	Precision
34760	19943	0.760	0.777
False Positive	True Negative	Recall	F1 Score
9950	59710	0.635	0.699
Positive Label	Negative Label		
Low	High		

LOGISTIC REGRESSION

We will be using the Logistic Regression component given by Azure ML studio.

The 70% of the split data is feed into Train Model component along with Random Forest component.



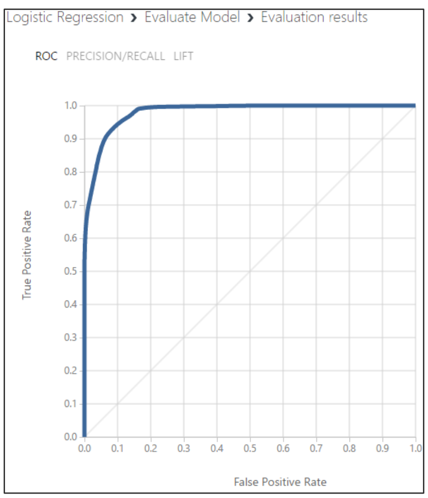


The visualization of the Scored Model is shown below, it show the scored labels which is predicted outcome along with Scored porbabilities, which shows what are

Base_Hour_Class	Scored Labels	Scored Probabilities
High	High	0.018944
High	Low	0.558313
High	High	0.001081
Low	Low	0.515536
Low	Low	0.761178
Low	Low	0.595481

"Scored Labels" is calculated result which indicates what the algorithm has calculated. In an ideal case this would always be same as value of column "Base_Hour_Class".

The visualization of the Evaluate Model is shown below:

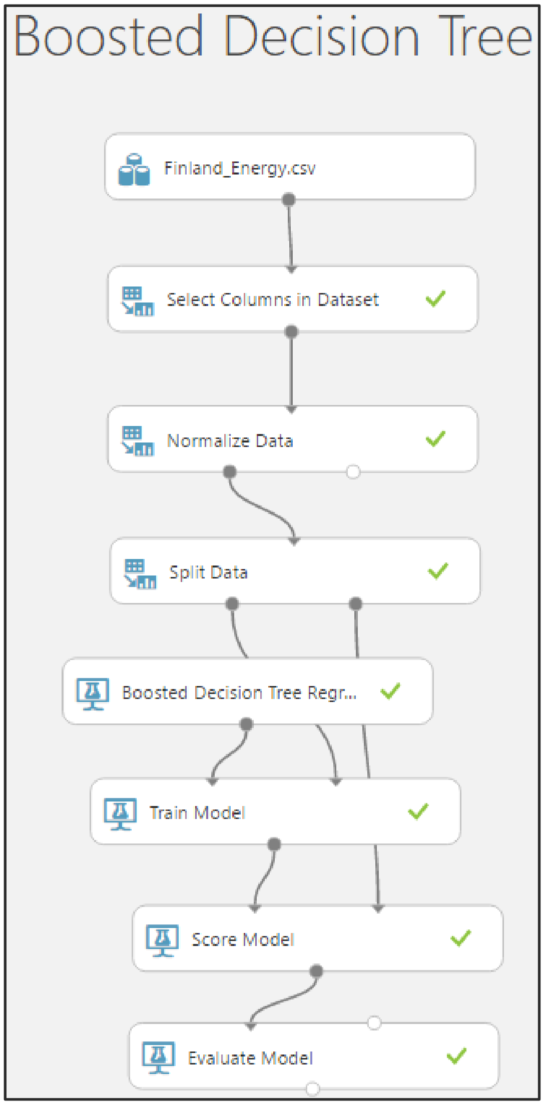


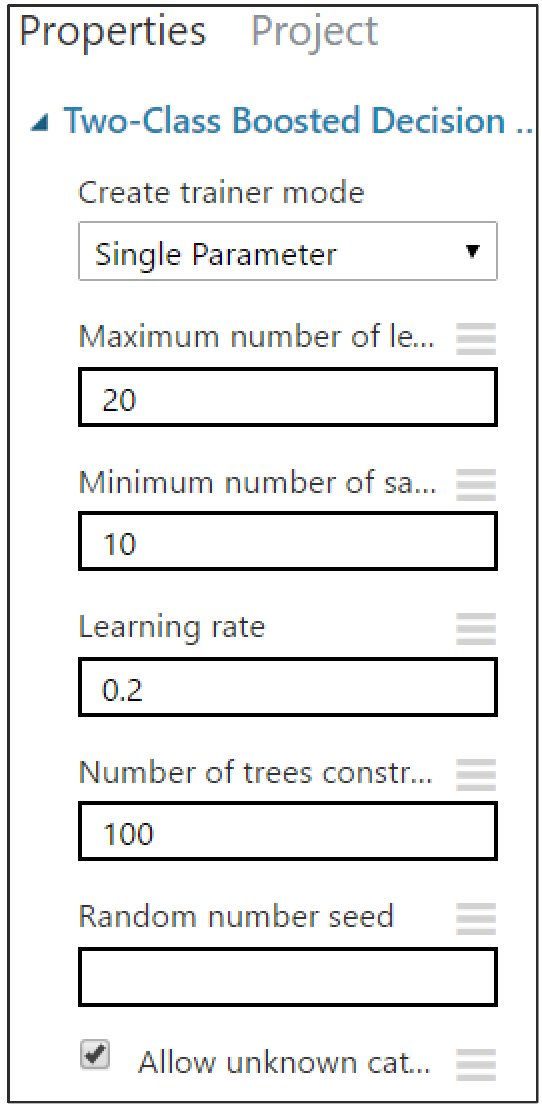
True Positive	False Negative	Accuracy	Precision
52414	2121	0.913	0.858
False Positive	**True Negative**	**Recall**	**F1 Score**
8687	61141	0.961	0.907
Positive Label	**Negative Label**		
Low	High		

The Accuracy is pretty good so we can finalize the settings of this trained model, otherwise we could have tuned the parameters of the algorithm component.

BOOSTED DECISION TREE REGRESSION

You can use the **Two-Class Boosted Decision Tree** module to create a machine learning model that is based on the boosted decision trees algorithm. A boosted decision tree is an ensemble learning method in which the second tree corrects for the errors of the first tree, the third tree corrects for the errors of the first and second trees, and so forth.





The results of are shown below

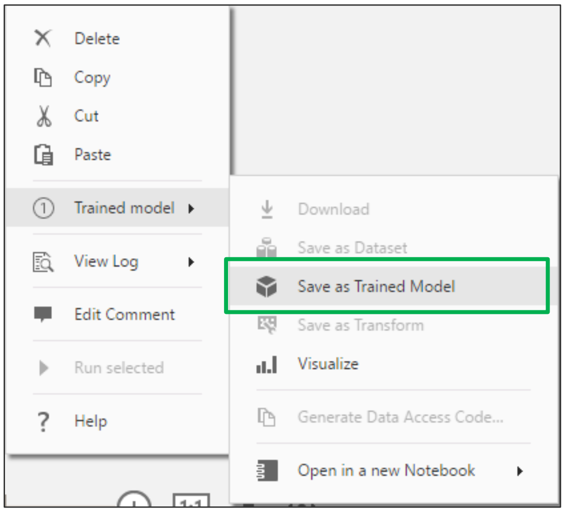
Boosted Decision Tree › Score Model › Scored dataset

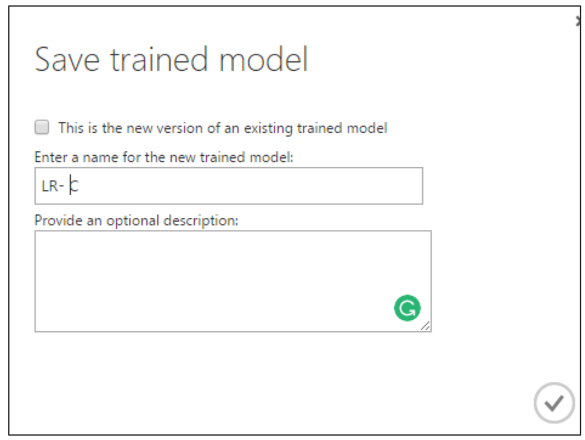
rows	columns
124363	9

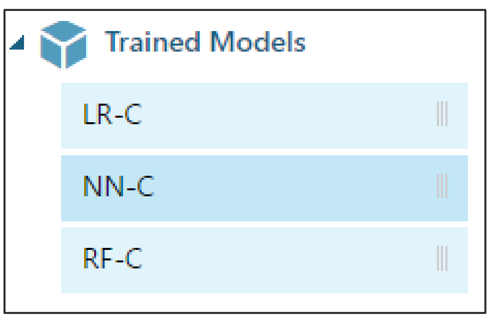
Weekday	Base_Hour_Flag	TemperatureF	Dew_PointF	base_hr_usage	Base_Hour_Class	Scored Labels	Scored Probabilities
1	0	0.462185	0.575916	0.020987	High	High	0
0	0	0.798319	0.877487	0.012561	High	High	0.007446
1	0	0.487395	0.28377	0.019448	High	High	0
1	0	0.798319	0.877487	0	Low	Low	0.999999

Saving TRAINED MODEL

After we run all the models and is satisfied with the Scored Model output, we proceed to save the model by right clicking it.



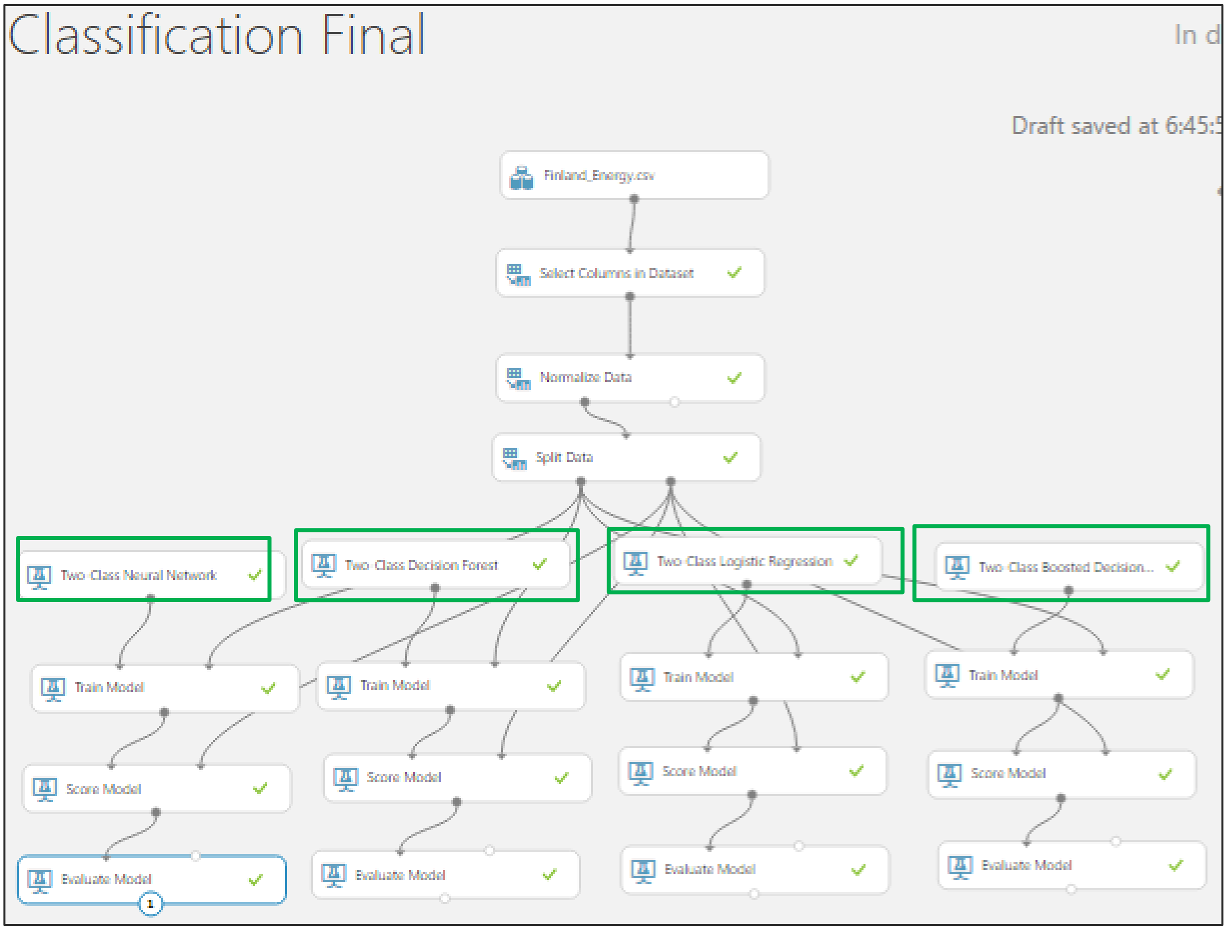




These saved will be used when we will setup our web service.

SETUP WEB SERVICE: CLASSIFICATION

Our model upon setting up as web service looks like below, in this we have dragged our saved models.

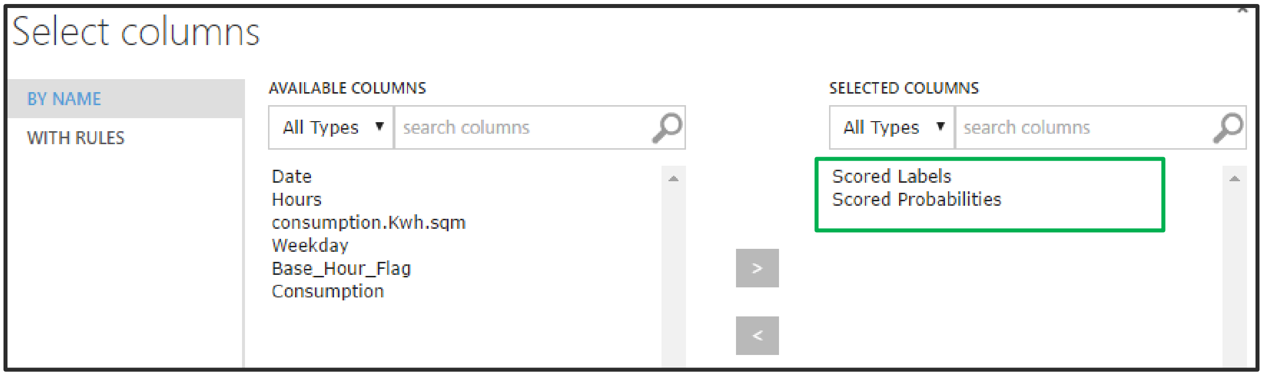


When we visualize the Score Model component we get scored labels and scored probabilities along with other features.

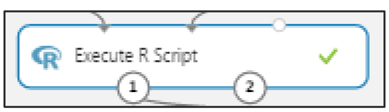
umption.Kwh.sqm	Weekday	Base_Hour_Flag	Consumption	Scored Labels	Scored Probabilities
3788	1	1	0.010081	High	0.187973
3788	1	1	0.010081	High	0.187441
3939	1	1	0.010484	High	0.175177

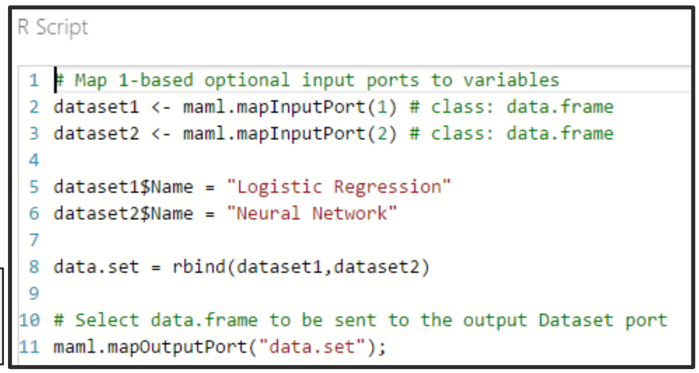
Scored Probabilities: We will consider taking this value as an output of web service to allow user to choose a model based on accuracy instead of using his own judgment.

We are using 'Select Column in Dataset' component to extract just these 2 columns.



We will also be require to use 'Execute R Script' component where we are row binding the Scored labels and Scored Probabilities from the 'Scored Models output', using this only we will be able to output all the values by 'Web Servive output' component.

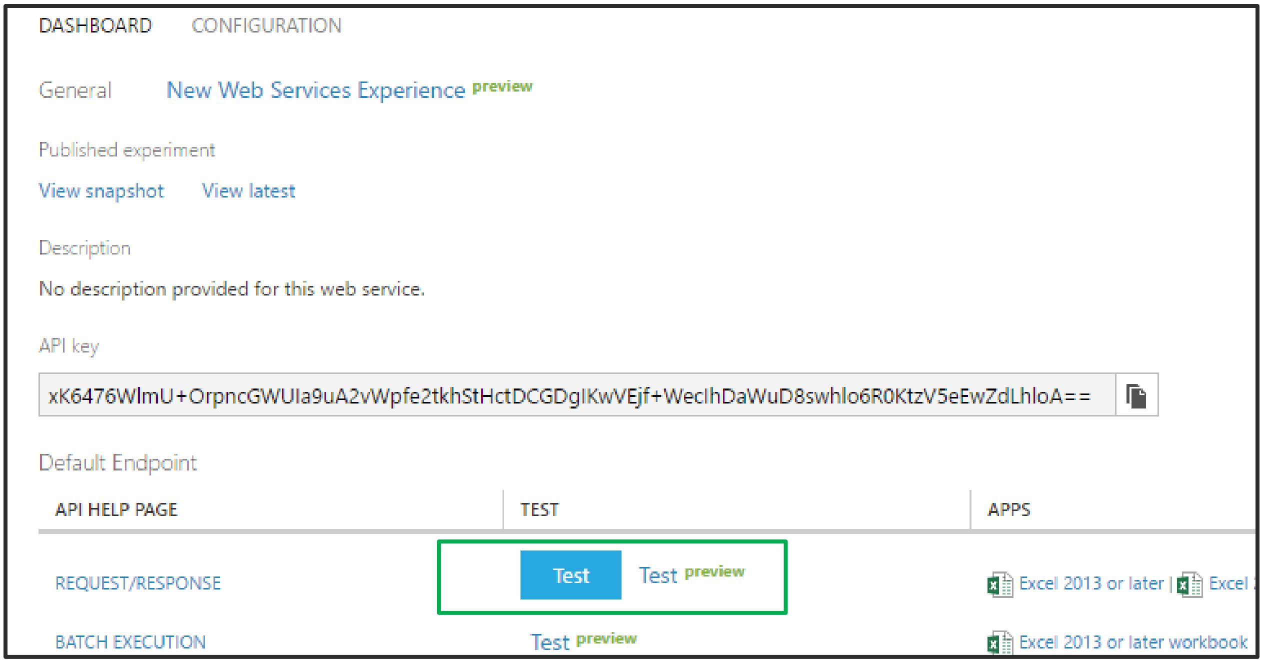




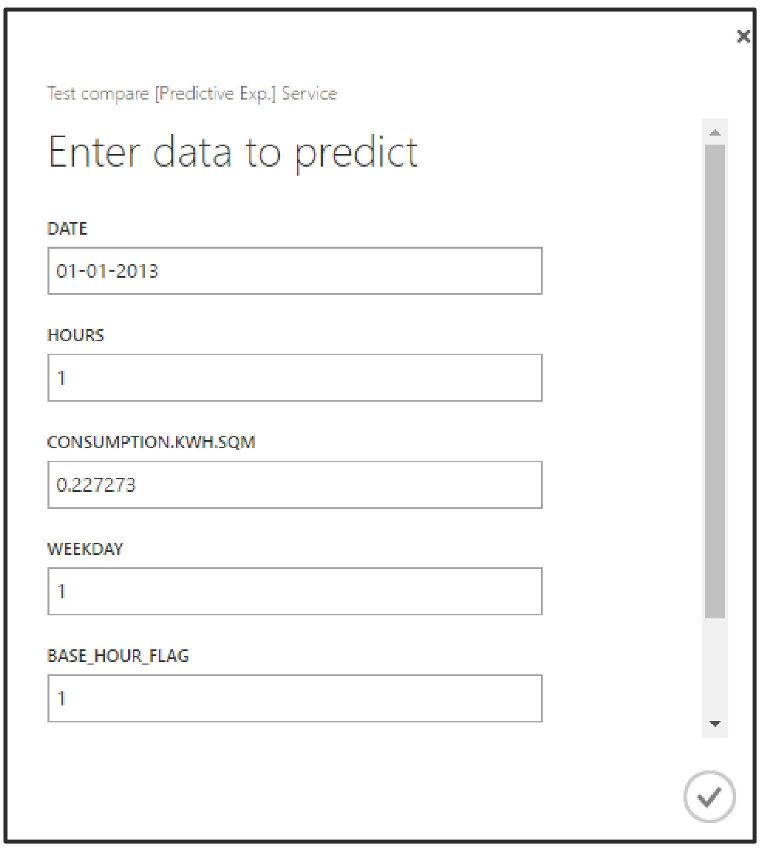
DEPLOY THE WEB SERVICE

We can deploy the experiment as either a classic Web service or a new Web service based on Azure Resource Manager.

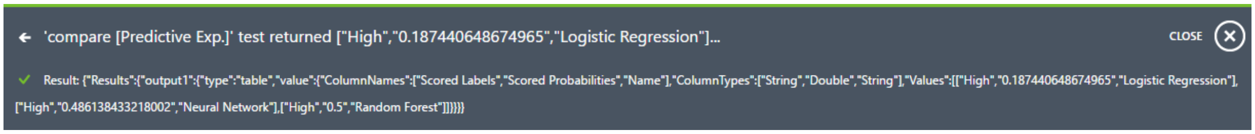
To deploy a classic Web service derived from our experiment, click **Deploy Web Service** below the canvas and select **Deploy Web Service [Classic]**. Machine Learning Studio deploys the experiment as a Web service and takes you to the dashboard for that Web service.



From here, you can return to the experiment (**View snapshot** or **View latest**) and run a simple test of the Web service (See **Test the Web service** below).



Th results will be a JSON format output



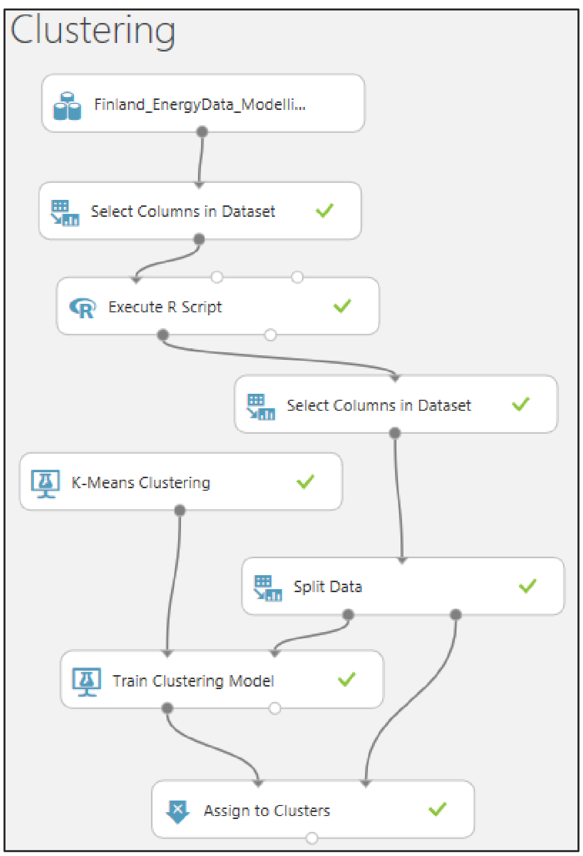
This output shows the output from all the classification models, it has scored probabilities and scored labels.

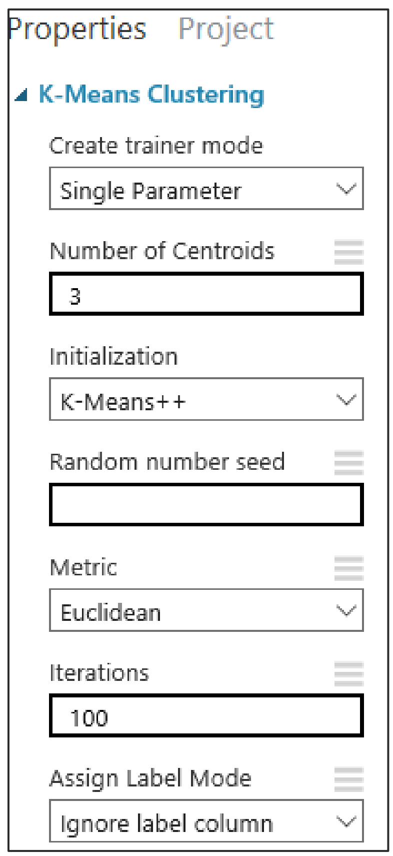
CLUSTERING

Cluster analysis classifies a set of observations into two or more mutually exclusive unknown groups based on combinations of variables. The purpose of cluster analysis is to discover a system of organizing observations, their characteristics, into groups, where members of the groups share properties in common.

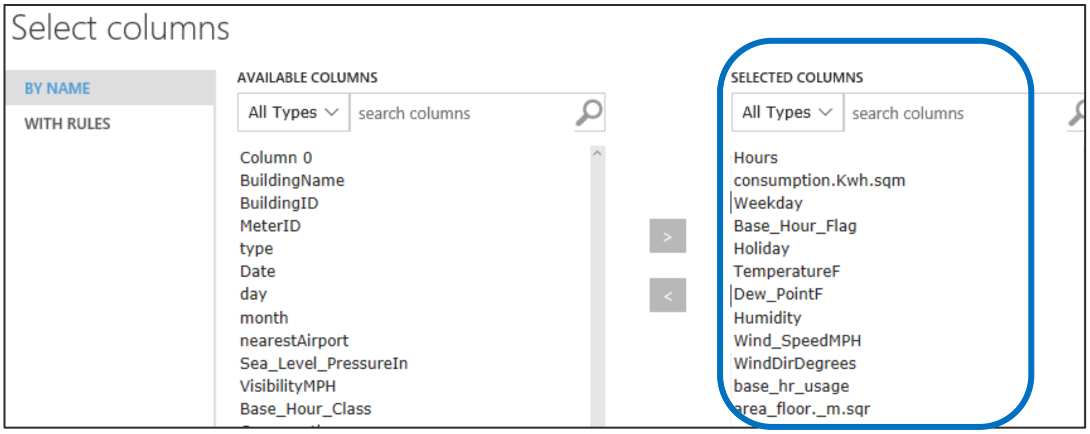
K-MEANS CLUSTERING

We can use the **K-Means Clustering** module to create an untrained K-means clustering model. K-means is one of the simplest and the best known *unsupervised* learning algorithms, and can be used for a variety of machine learning tasks, such as detecting abnormal data, clustering of text documents, and analysis of a dataset prior to using other classification or regression methods.





1. We have selected columns as below for clustering

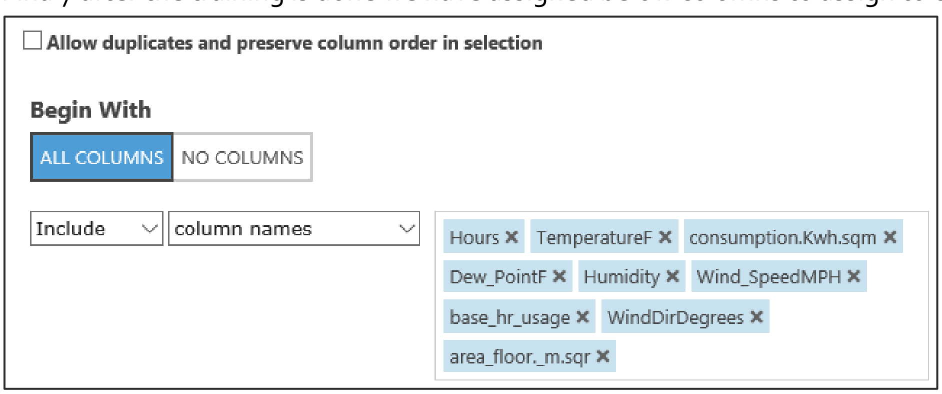


2. We have used R script do basic data mnipulation like removing NA, changing data types and to store numric values in a dataframe.

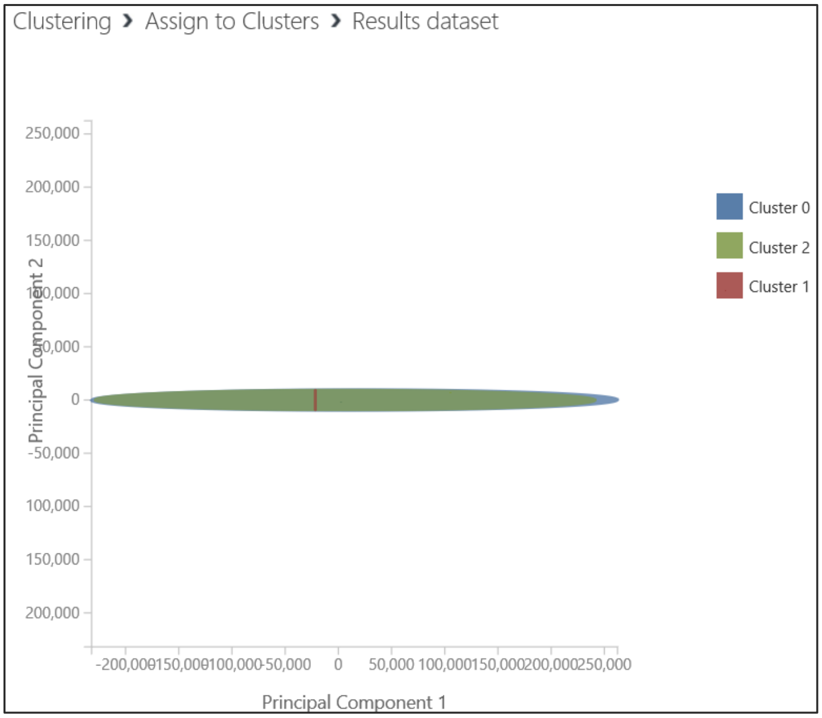
R Script

```

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

data <- Filter(function(x)!all(is.na(x)), data)
data <- Filter(function(x)!all(x=="N/A"), data)

data$Weekday=as.factor(data$Weekday)
data$Base_Hour_Flag=as.factor(data$Base_Hour_Flag)
data$Holiday=as.factor(data$Holiday)

#storing numeric values

nums<- sapply(data, is.numeric)
df<- data[,nums]


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

3. Splitting the data follow what we did in previous experiemnts 70% 30% split for test and train.
4. Finaly after the training is done we have assigned below columns to assign to cluster.



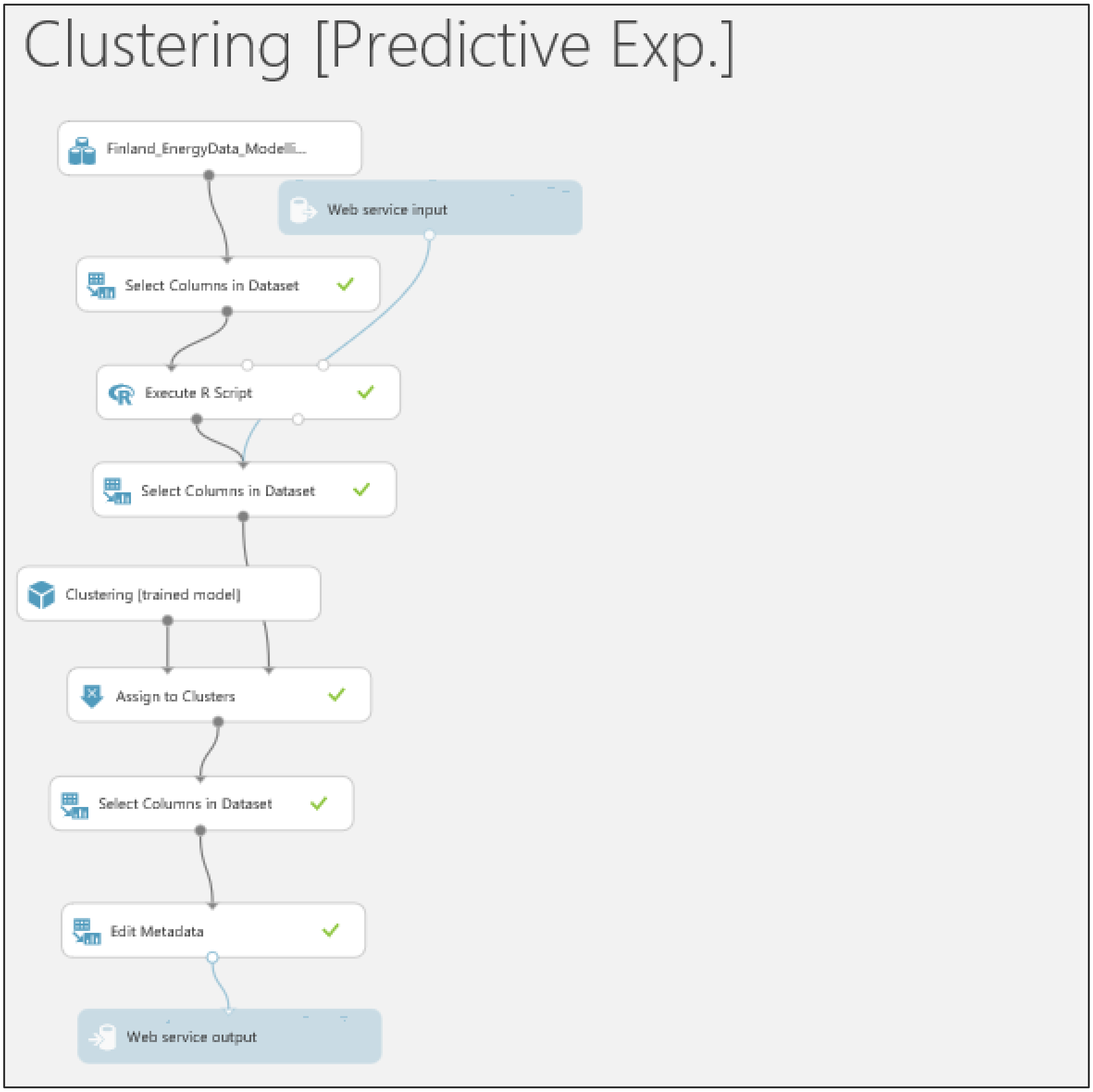
The Result dataset visulization



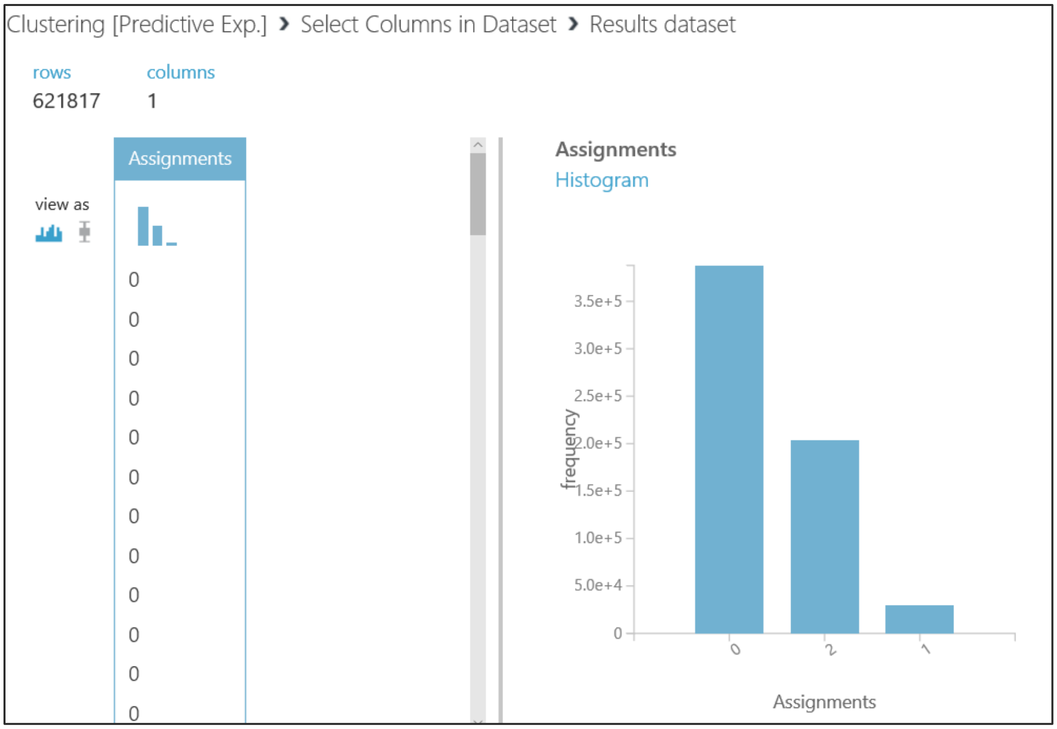
We can see the the Cluster 1 and 3 are very small compared to cluster 3.

SETUP WEB SERVICE: CLUSTERING

Our model upon setting up as web service looks like below, in this we have dragged our saved models.

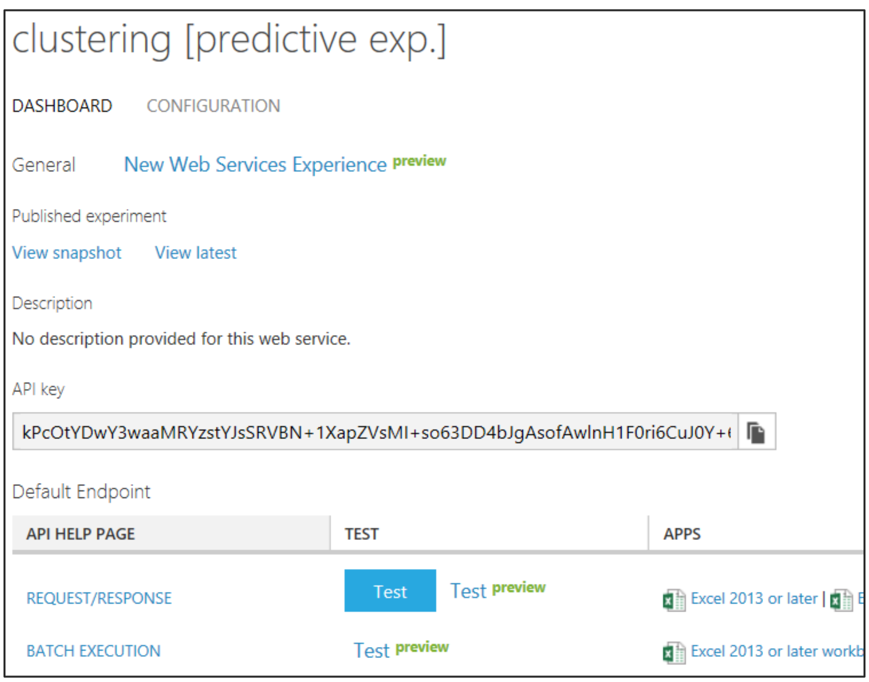


The Output is shown as:

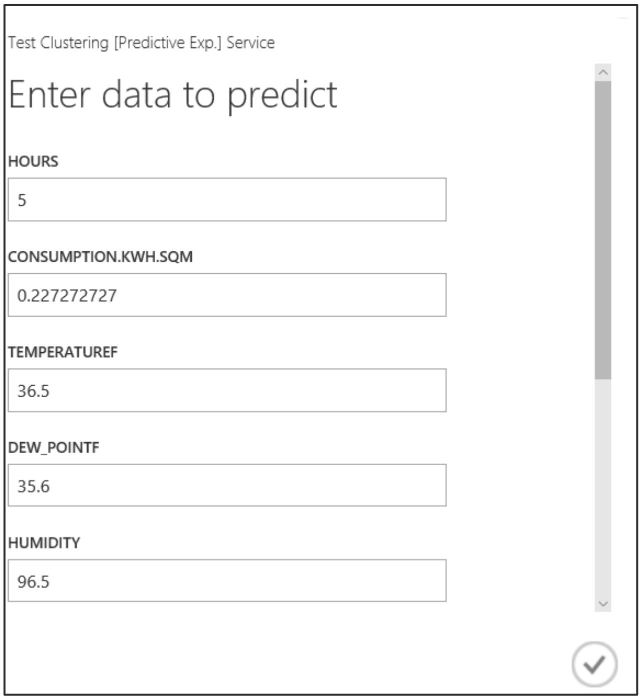


DEPLOY THE WEB SERVICE

Similar to what we did for Clusterinf and regression we can Test out web service.



We will fill out the data to the experiment



The output of this experiment is JSON format, which mentions different clusters.

'Clustering [Predictive Exp.]' test returned ["0"]...

Result: {"Results":{"output1":{"type":"table","value":{"ColumnNames":["Cluster"],"ColumnTypes":["Nullable`1"],"Values":[["0"]]}}}}

WEB IMPLEMENTATION

We will be using Angular.js as our frontend framework to call the Rest API for each part. Angular.js has following main component's.

APP.JS

We will define out Application name as shown below

```
test.js    app.js    config.js    server.js
var myApp = angular.module("productCustomizer",["ngRoute", "rzModule"]);
```

CONFIG.JS

A Configuration file will need to be made for the routing of the web page. We will define our main page as index.html and a controller which is required by Angular as Main Controller.

```
/**
 * Created by Lalit on 6/18/2016.
 */
(function(){
    angular
        .module("productCustomizer")
        .config(Config);

    function Config($routeProvider){
        $routeProvider
            .when("/", {
                templateUrl: "index.html",
                controller: "MainController"
            })
            .otherwise({
                redirectTo: "/"
            });
    }
})();
```

SERVER.JS

This will be default location for the server control. Application port's, redirection and function's to be called for REST api will be handled here

First, we will load all the required libraries for the HTTP functionality as shown below

```
var express = require('express');
var app = express();
var http = require("http");

var https = require("https");

var querystring = require("querystring");

var fs = require('fs');

var bodyParser = require('body-parser');
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));

// configure a public directory to host static content
app.use(express.static(__dirname + '/public'));


var ipaddress = process.env.OPENSHIFT_NODEJS_IP;
var port      = process.env.OPENSHIFT_NODEJS_PORT || 3000;

```

Then, we will define our application's posting url. This is essentially the routing post, where based on the type of prediction we will create different functions

```
app.listen(port, ipaddress);
// app.post("/prediction/neural-network", neuralNetwor
// app.post("/prediction/linear-regression", linearReg
// app.post("/prediction/random-regression",randomFore
app.post("/prediction/regression",regression);
app.post("/classification",classification);
app.post("/clustering",clustering);
//maml-server.js
```

So we will use three different post for three different type of predictions that we will perform

Now we can define our getPred function which we will use to call our api call and get the result back. This function will take data of the user input, api key and url for the call to work.

```
function getPred(data, path, api_key) {

    var dataString = JSON.stringify(data);
    var host = 'ussouthcentral.services.azureml.net';
    var headers = {'Content-Type': 'application/json', 'Authorization': 'Bearer ' + api_key};

    var options = {
        host: host,
        port: 443,
        path: path,
        method: 'POST',
        headers: headers
    };

    result = '';
    var reqPost = https.request(options, function (res) {
        res.on('data', function (d) {
            setTimeout(function() {
                process.stdout.write(d);
                result += d;
            }, 300);
            //return d;
        });
    });

// Would need more parsing out of prediction from the result
    reqPost.write(dataString);
    reqPost.end();
    reqPost.on('error', function (e) {
        console.error(e);

    });
    console.log(result);
    //return result;
}
```

Output of the function will be stored in a result variable and can be consumed later.

Now since we have three different calls to make, we will define 3 different functions which will basically provide respective data, api and url to the getPred function defined above as shown below

```
function classification(req, res){
    var data = req.body;
    var path = '/workspaces/45b46042032542099deeace0661fa453/services/5f98e39eacf94210bd07a15cccd93f05/execute?api-version=2.0&details=true';
    var key = 'xK6476WlmU+OrpncGWUIa9uA2vWpfe2tkhStHctDCGDgIKwVEjf+WecIhDaWuD8swhlo6R0KtzV5eEwZdLhloA==';
    getPred(data, path, key);
    setTimeout(function() {
        res.json(result);
    }, 1000);

}

function regression(req, res){
    var data = req.body;
    var path = '/workspaces/0e6e3268518847ab90cb1087c291e541/services/55103cd672ca48daa7d8f2cdc6038e8d/execute?api-version=2.0&details=true';
    var key = '4PSBt+Mu5bdQh0GmbPCsVwv/qfW1Xt8cQbhlTcvFny37m2tEictUytGt8Uorh1b+WGNaHT2OayaGP7v/u9bTkA==';
    getPred(data, path, key);
    setTimeout(function() {
        res.json(result);
    }, 1000);
}

function clustering(req, res){
    var data = req.body;
    var path = '/workspaces/0e6e3268518847ab90cb1087c291e541/services/55103cd672ca48daa7d8f2cdc6038e8d/execute?api-version=2.0&details=true';
    var key = '4PSBt+Mu5bdQh0GmbPCsVwv/qfW1Xt8cQbhlTcvFny37m2tEictUytGt8Uorh1b+WGNaHT2OayaGP7v/u9bTkA==';
    getPred(data, path, key);
    setTimeout(function() {
        res.json(result);
    }, 1000);
}
```

Now before we jump to MainContoller, we will design our web page. Since, MainController require elements fetch from the front end, we will define the home page and use those element's in the MainController later

INDEX.HTML

We will say the home page to use our app defined by ng-app. in the main html tag. Similarly, we will add scripting components to use required angular.js libraries.

We will provide ng-controller "Main Controller" to the main body section, which we will define later.

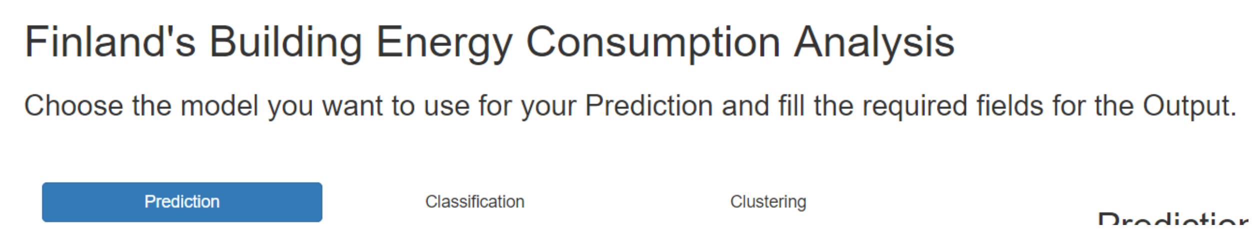
We will also add custom script's we defined earlier (app.js,config.js and main.controller.js)

```
<!DOCTYPE html>
<html lang="en" ng-app="productCustomizer">
<head>
    <title> Data Science </title>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.5.0-beta.0/angular.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.5.0-beta.0/angular-route.js"></script>

</head>
<body ng-controller="MainController" style="padding: 20px;">
<h1>Finland's Building Energy Consumption Analysis</h1>
<h3> Choose the model you want to use for your Prediction and fill the required fields for the Output. </h3>
<br/><br/>
<script src="app.js"></script>
<script src="config.js"></script>
<script src="main.controller.js"></script>
```

For the user control on the type of Prediction to be used, will provide 3 button's and link those button's to ng-method's to distinguish and control the flow.

```
<div class="col-sm-6">
    <ul class="nav nav-pills nav-justified">
        <li class="btn" ng-click="method = 'prediction'" ng-class="{'btn-primary': method == 'prediction'}">Prediction</li>
        <li class="btn" ng-click="method = 'classification'" ng-class="{'btn-primary': method == 'classification'}">Classification</li>
        <li class="btn" ng-click="method = 'clustering'" ng-class="{'btn-primary': method == 'clustering'}">Clustering</li>
    </ul>
```

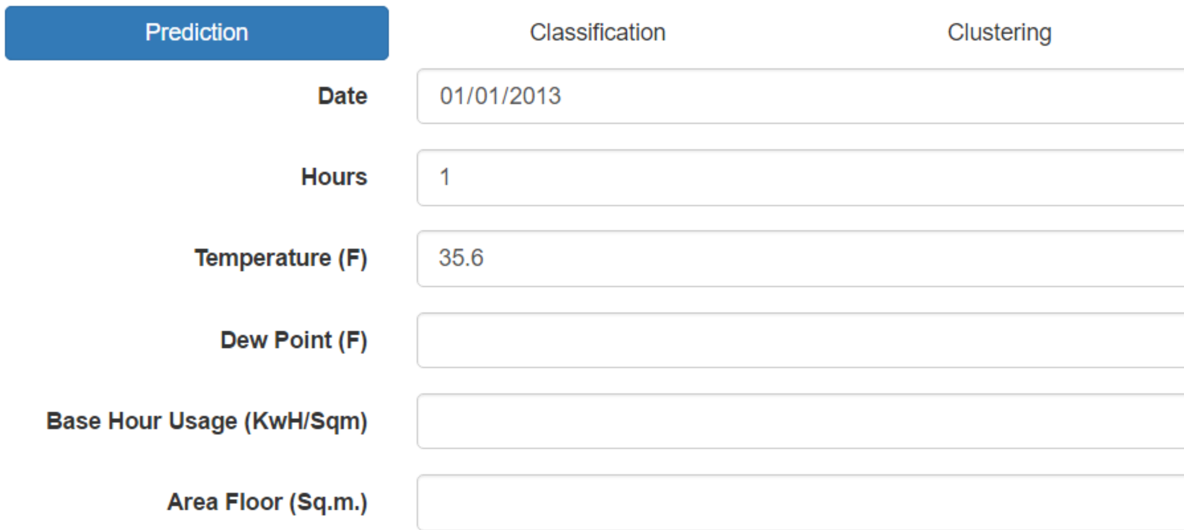
Output of the HTML page can be shown as below:

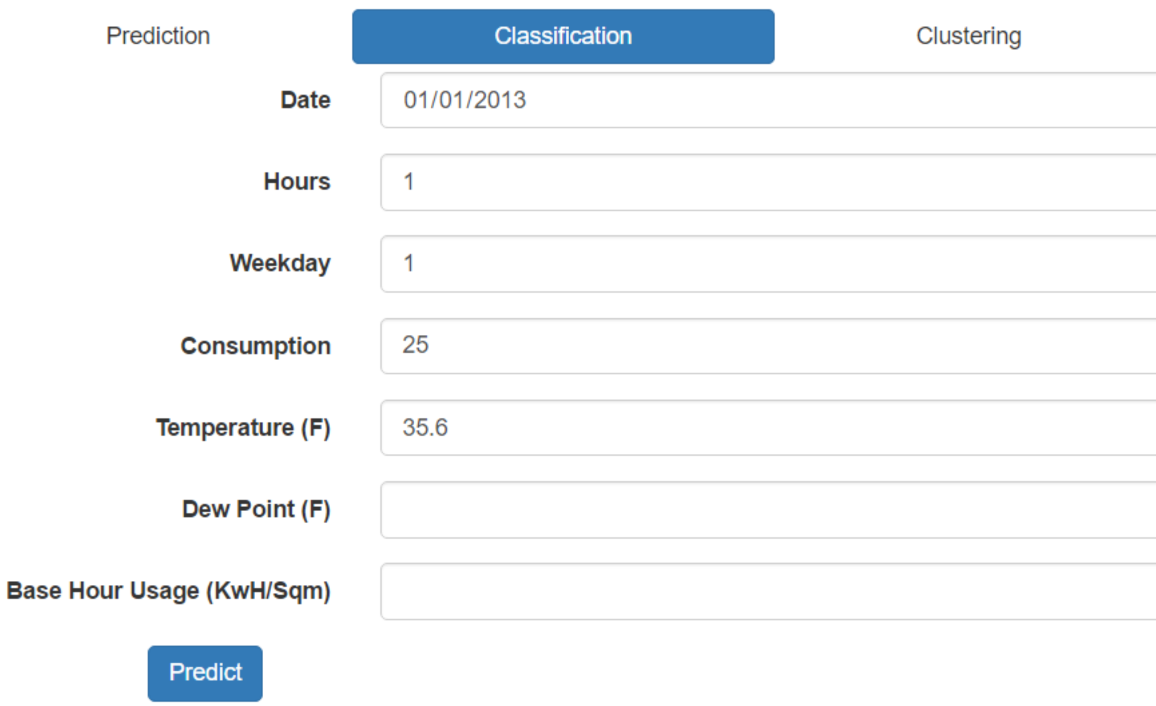


For the type of features we decided for each type of Prediction, we will use div components and tag them with related ng-method. For example, if we want to use Temperature as a component for Regression, we will mark the div component as ng-method == "Prediction" which we defined earlier.

```
<div class="form-group">
    <label for="date" class="col-sm-4 control-label"> Date </label>
    <div class="col-sm-8">
        <input class="form-control" id="date" type="Date" ng-model="date">
    </div>
</div>

<div class="form-group" ng-show="method == 'prediction' or method == 'classification'">
    <label for="hours" class="col-sm-4 control-label"> Hours </label>
    <div class="col-sm-8">
        <input class="form-control" id="hours" type="number" ng-model="hours">
    </div>
</div>
<div class="form-group" ng-show="method == 'classification'">
    <label for="weekday" class="col-sm-4 control-label">Weekday </label>
    <div class="col-sm-8">
        <input class="form-control" id="weekday" type="text" ng-model="weekday">
    </div>
</div>
<div class="form-group" ng-show="method == 'classification'">
    <label for="consumption" class="col-sm-4 control-label">Consumption </label>
    <div class="col-sm-8">
        <input class="form-control" id="consumption" type="text" ng-model="consumption">
    </div>
</div>
<div class="form-group" ng-show="1==0">
    <label for="Base_Hour_Flag" class="col-sm-4 control-label">Base Hour Flag </label>
    <div class="col-sm-8">
        <input class="form-control" id="Base_Hour_Flag" type="text" ng-model="Base_Hour_Flag">
    </div>
</div>
```

So on and so forth respectively for each type of Prediction method. We can see the result of the HTML page below



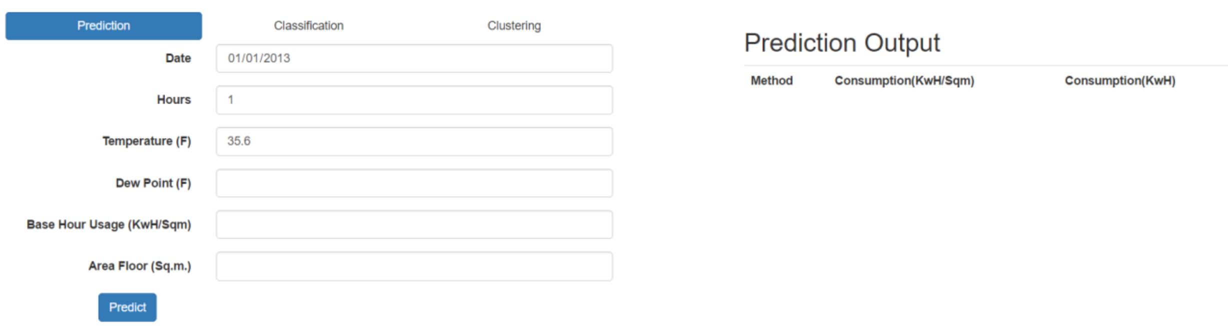


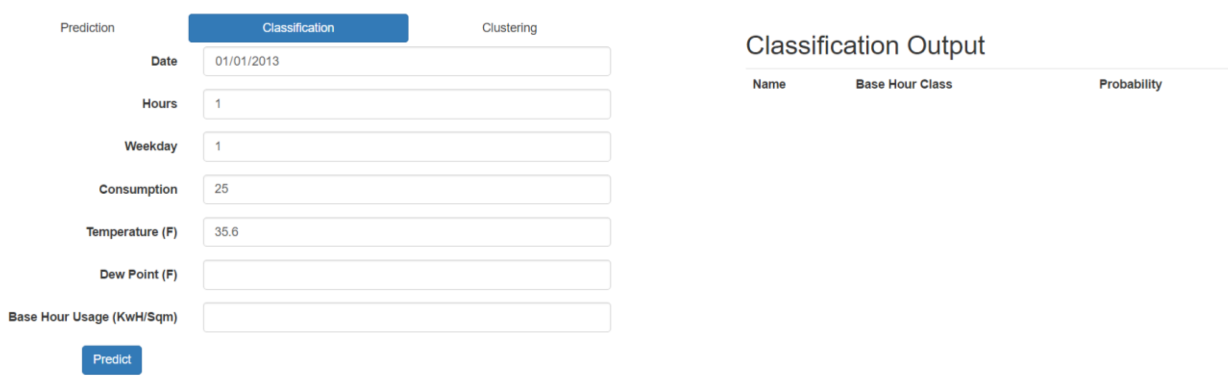
As we can see that, defending on the type of method selected and our features required for those prediction method's we will show the user those input's only.

Similarly, on the right hand side, will display the result. Again depending on the type of method selected, we will change the output table

```
<div class="col-sm-5 col-sm-offset-1">
    <div ng-show="method == 'prediction'">
        <h2>Prediction Output</h2>
        <table class="table table-hover">
            <tr>
                <th>Method</th>
                <th>Consumption(KwH/Sqm)</th>
                <th>Consumption(KwH)</th>
            </tr>
            <tr ng-repeat="result in results">
                <td> {{result.type}}</td>
                <td> {{result.persqm}}</td>
                <td> {{result.total}}</td>
            </tr>
        </table>
    </div>
    <div ng-show="method == 'classification'">
        <h2>Classification Output</h2>
        <table class="table table-hover">
            <tr>
                <th>Name</th>
                <th>Base Hour Class</th>
                <th>Probability</th>
            </tr>
            <tr ng-repeat="result in classification_results">
                <td> {{result.name}}</td>
                <td> {{result.base_hr_class}}</td>
                <td> {{result.probability}}</td>
            </tr>
        </table>
    </div>
</div>
```

Output table can be seen as shown





MAIN CONTROLLER

Now since we have defined the ng-tag of the input's in the HTML page, we can use those tag's in the Main controller to fetch the data and pass to the server.js to get the output.

```
(function(){
    angular
        .module("productCustomizer")
        .controller("MainController", MainController);

    function MainController($scope, $http){
        $scope.method = 'prediction';
        $scope.show = "teat";
        $scope.slider = {
            value: 10,
            options: {
                showSelectionBar: true
            }
        };
        var output = [];
        var classification_result = [];
```

Here we are creating different variables like Output for Regression Output, Classification_result for classification results and so on.

Now depending on the method used by the user we will use different functions, format for the data input style can be seen from the Request-Response API documentation in the Azure studio. Scope will be used to fetch the elements form the front end.

```
        var output = [];
        var classification_result = [];
        $scope.predict = function() {
            if ($scope.method == 'prediction'){
                var d = {
                    "Inputs": {
                        "input1": {
                            "ColumnNames": [
                                "Date",
                                "Hours",
                                "TemperatureF",
                                "Dew_PointF",
                                "base_hr_usage",
                                "area_floor._m.sqr"
                            ],
                            "Values": [
                                [   $scope.date,
                                    $scope.hours,
                                    $scope.TemperatureF,
                                    $scope.Dew_PointF,
                                    $scope.base_hr_usage,
                                    $scope.area_floor
                                ]
                            ]
                        }
                    },
                    "GlobalParameters": {}
                };
```

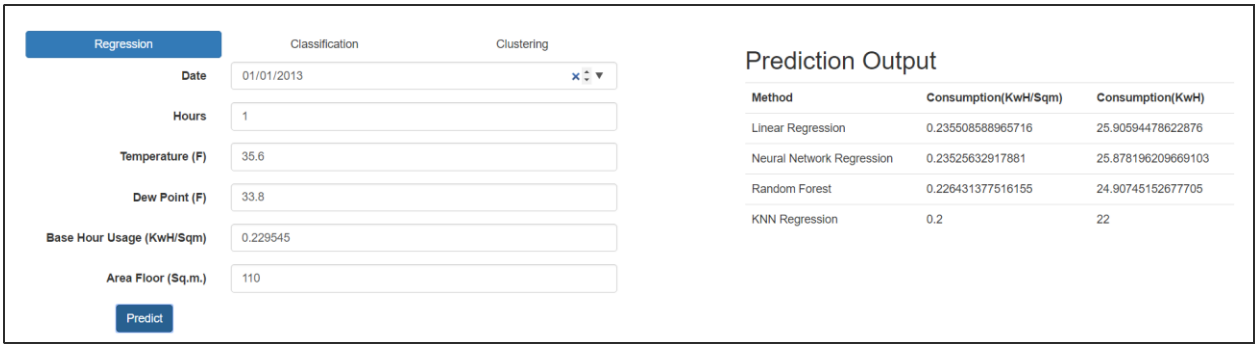
Now since we have everything we need, we can call our server.js with all the data that we have and using the appropriate app post. Output of the result will be saved and will be displayed in the HTML page. Similarly, we can add additional functions for Classification and clustering model's based on the user selection and using its respective data format.

```
                "GlobalParameters": {}
            };
            $http.post('/prediction/regression', d)
                .then(function (response) {
                    var result = JSON.parse(response.data);
                    result = result.Results.output1.value.Values;
                    for (var i = 0; i < result.length; i++) {
                        output.push({
                            'type': result[i][1],
                            'persqm': parseFloat(result[i][0]),
                            'total': parseFloat(result[i][0]) * d.Inputs.input1.Values[0][5]
                        });
                    }
                    $scope.results = output;

            });
        } else if($scope.method == 'classification'){
```

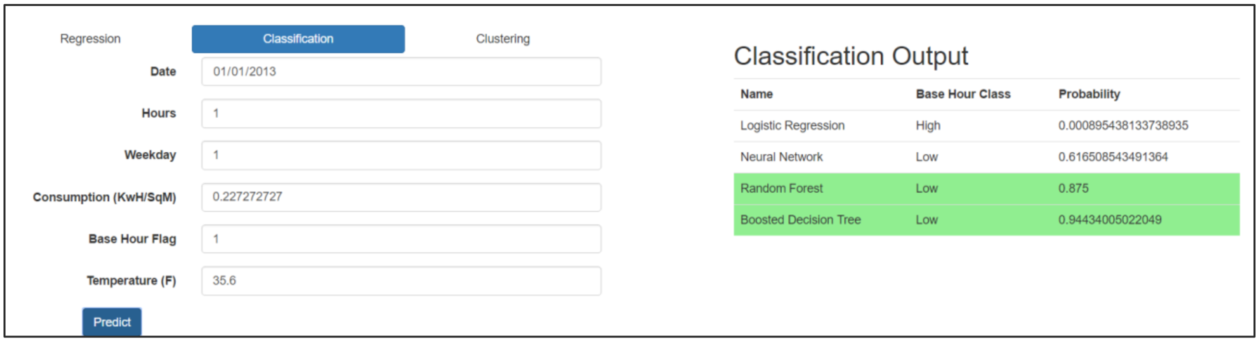
FINAL RESULTS

We can check out frontend results for different type of predictions

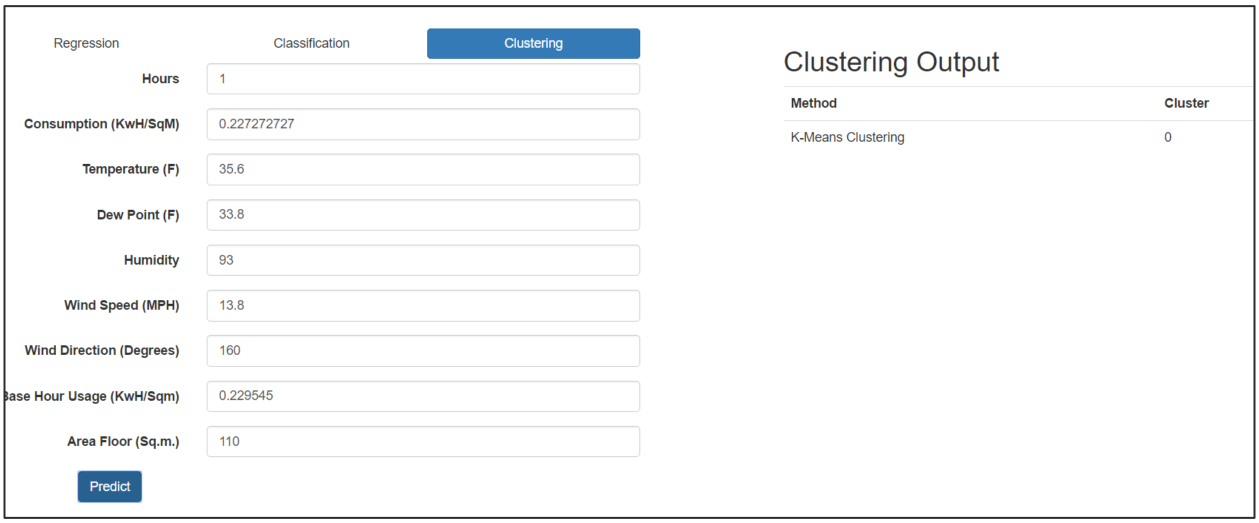
REGRESSION



CLASSIFICATION


Regression
Classification
Clustering
Date 01/01/2013
Hours 1
Weekday 1
Consumption (KwH/SqM) 0.227272727
Base Hour Flag 1
Temperature (F) 35.6
Predict
Classification Output
Name | Base Hour Class | Probability
Logistic Regression | High | 0.000895438133738935
Neural Network | Low | 0.616508543491364
Random Forest | Low | 0.875
Boosted Decision Tree | Low | 0.94434005022049

CLUSTERING


Regression
Classification
Clustering
Hours 1
Consumption (KwH/SqM) 0.227272727
Temperature (F) 35.6
Dew Point (F) 33.8
Humidity 93
Wind Speed (MPH) 13.8
Wind Direction (Degrees) 160
Base Hour Usage (KwH/Sqm) 0.229545
Area Floor (Sq.m.) 110
Predict
Clustering Output
Method | Cluster
K-Means Clustering | 0