**Machine Learning With Energy Dataset**

**OBJECTIVE**

The report summarizes the design and implementation of the data wrangling performed on the Appliances Energy Consumption data set.

This report is divided into 5 sections.

Section 1: Exploratory Data Analysis

Section 2: Feature engineering and Feature Selection

Section 3: Prediction algorithms

Section 4: Model Validation and Selection

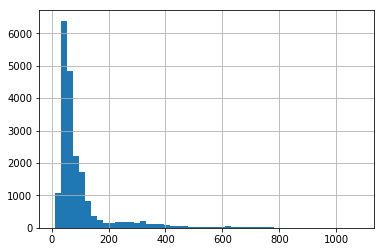
Section 5: Final pipeline

**Section 1: Exploratory Data Analysis**

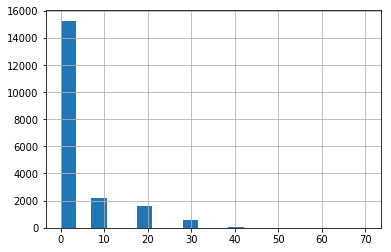
**Observations made from the dataset:**

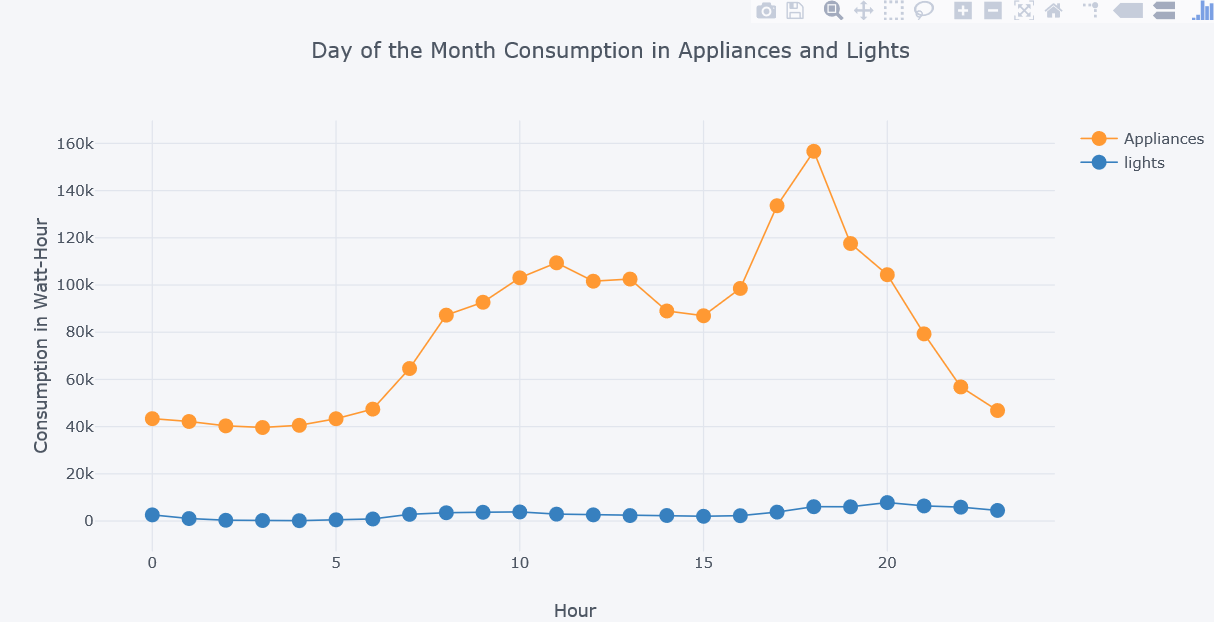
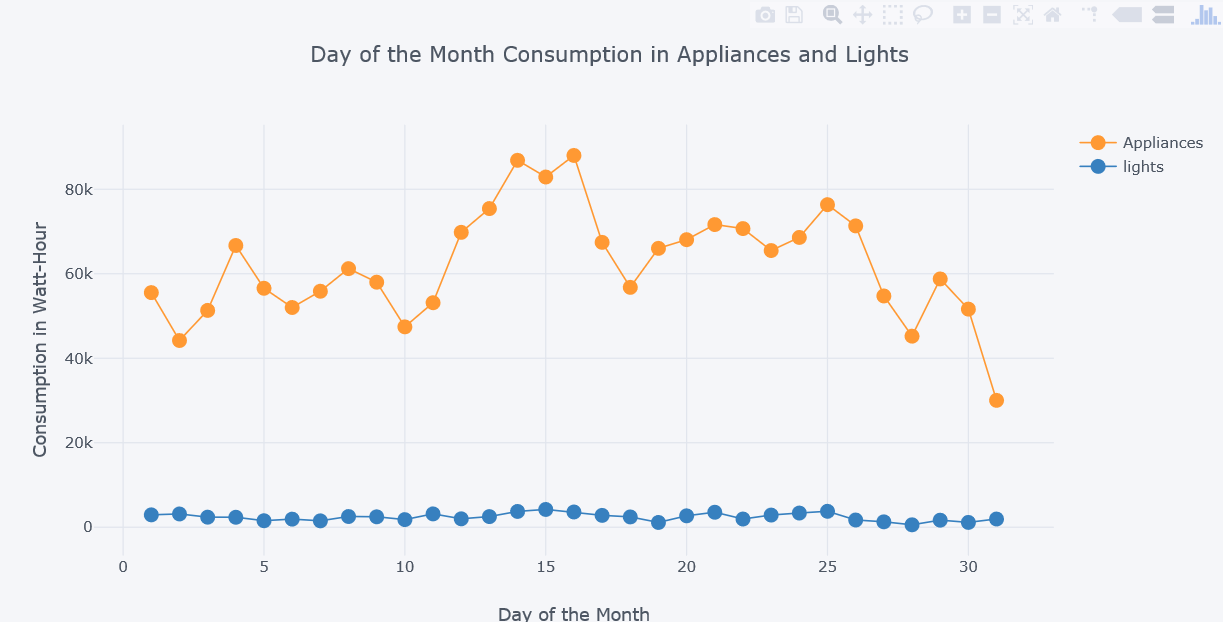
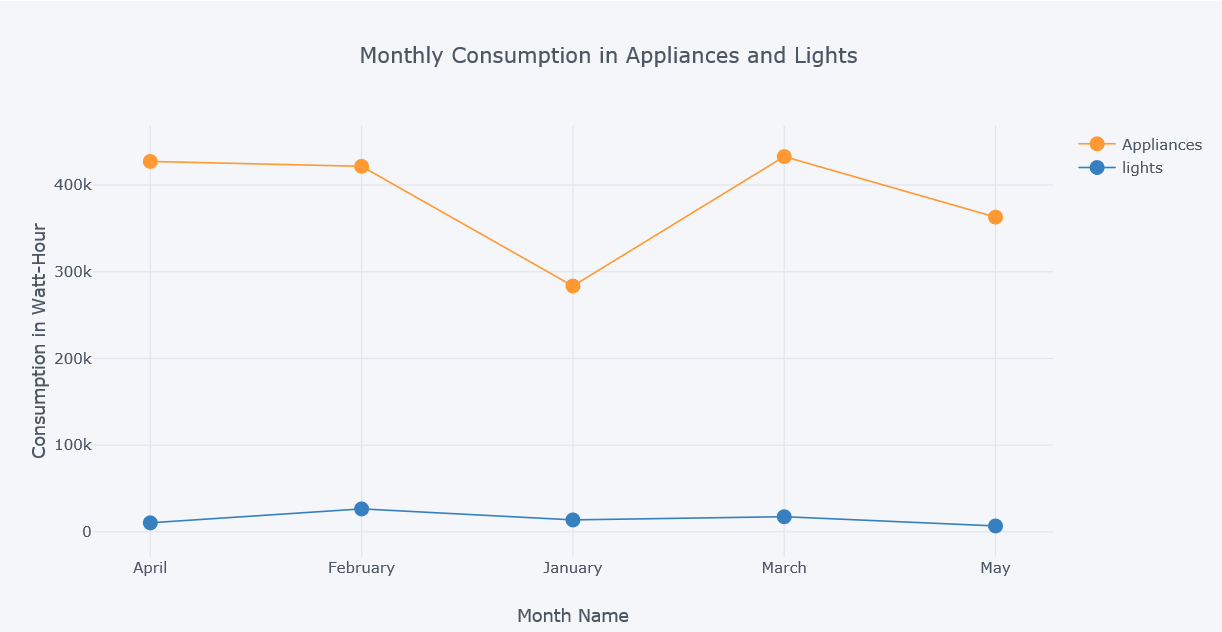
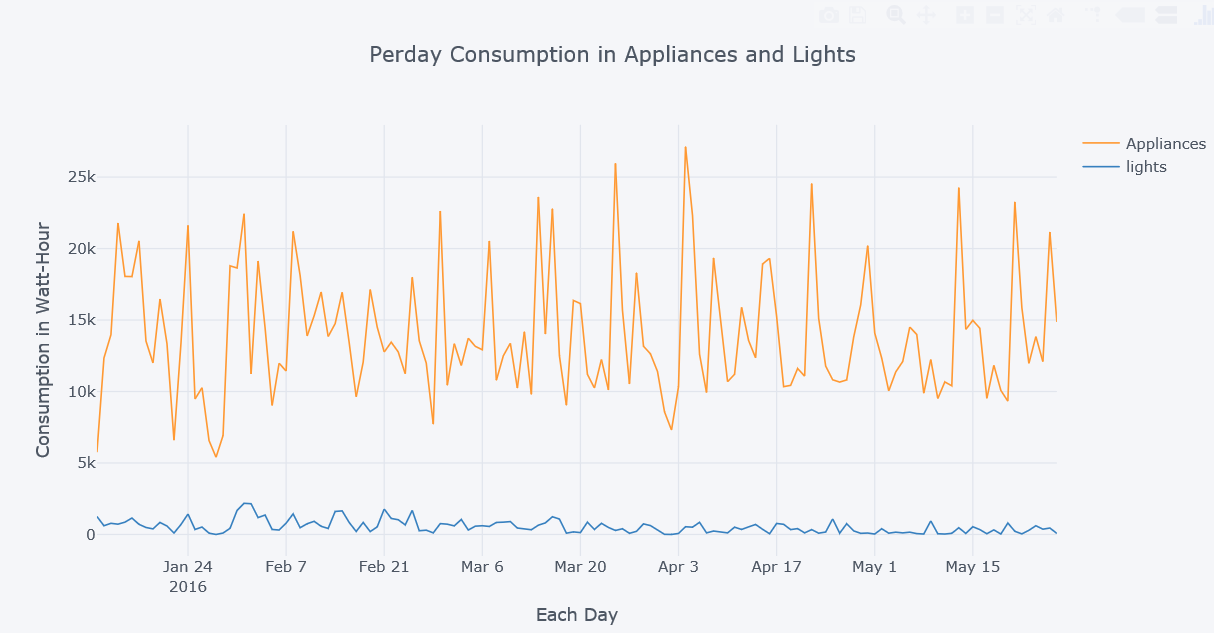
1. The date column is useful only if the date column information is divided into month, year columns. So we need to create new columns (month and year in our dataframe)for better analysis
2. We do not have any missing values in the dataset

Histograms of appliances to get the frequency

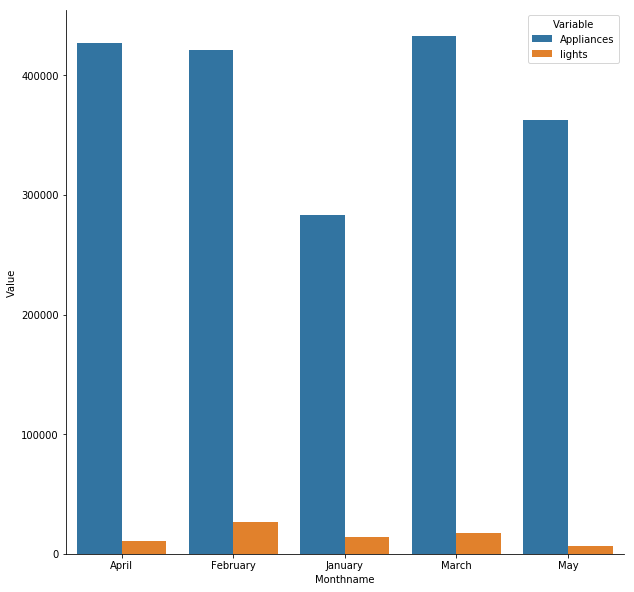


Histograms of appliances to get the frequency





Energy consumption by appliances vs energy consumption by light



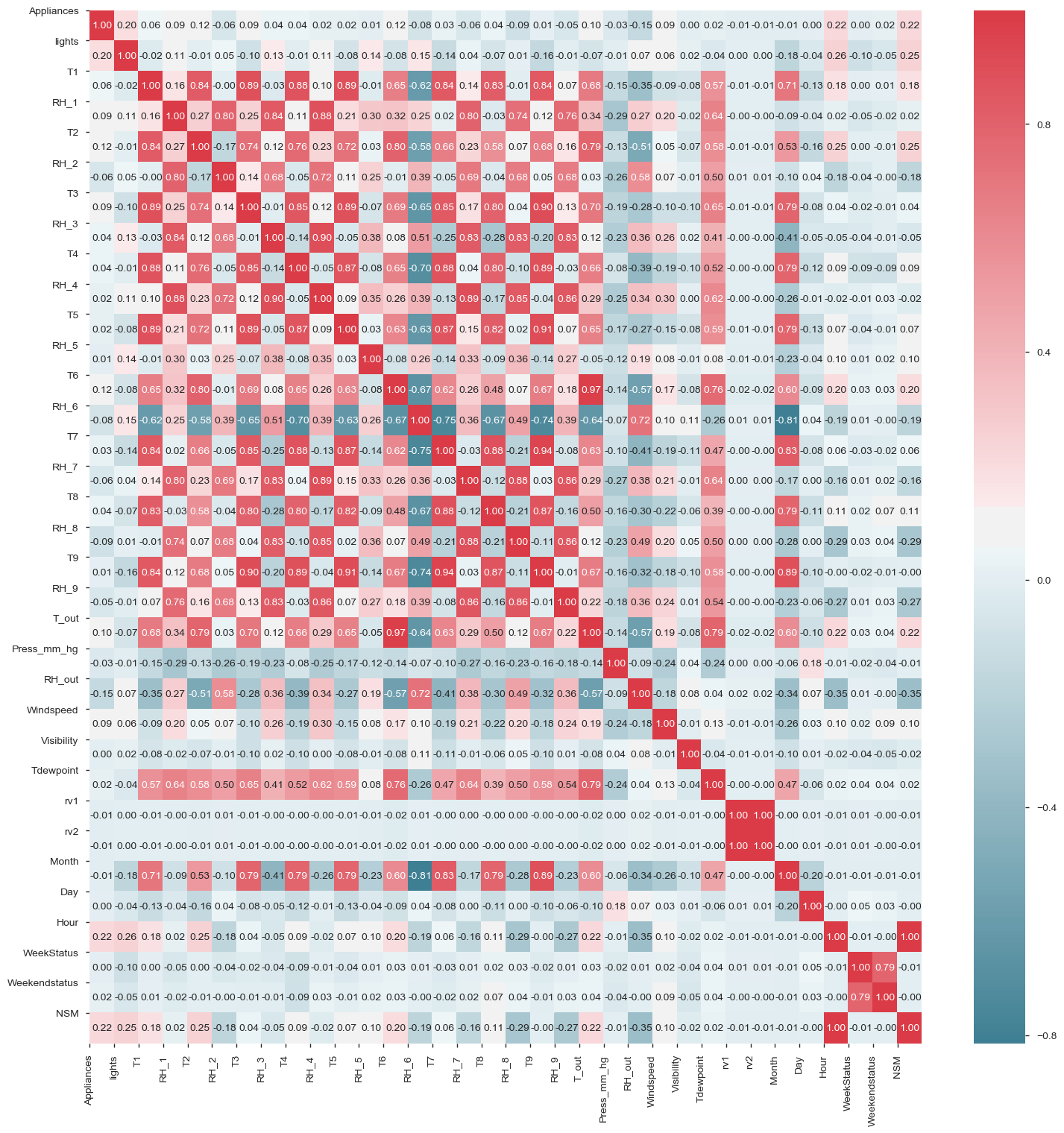
**Observation:**

1. The maximum appliance energy consumption was on 4th April, 2016. The maximum light energy consumption was on 1st February, 2016
2. The maximum appliance energy consumption was on March, 2016. The maximum light energy consumption was on February, 2016
3. The maximum appliance energy consumption was on 16th Hour of the Day. The maximum light energy consumption was on 15th Hour of the Day
4. The maximum appliance energy consumption was on 18th Day of the Month. The maximum light energy consumption was on 20th Day of the Month

Busiest Hour



Correlation Matrix:



**Section 2: Feature engineering and Feature Selection**

We have made changes to the dataset in the following ways:

1. added new columns derived from the existing columns.
2. deleted some of the existing columns in the dataframe as they are no longer needed.
3. some of the column values have been changed to convert the categorical value to a numerical value

#### Data Preprocessing

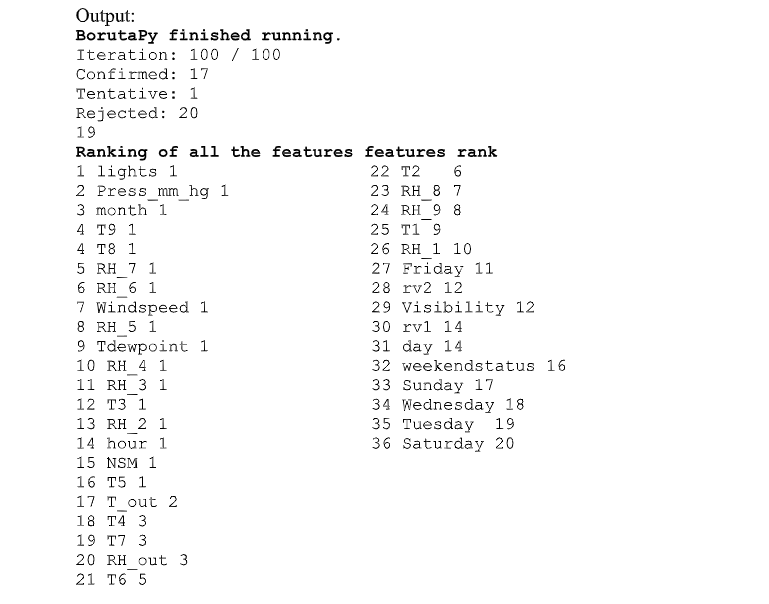
If some outliers are present in the set, robust scalers or transformers are more appropriate.

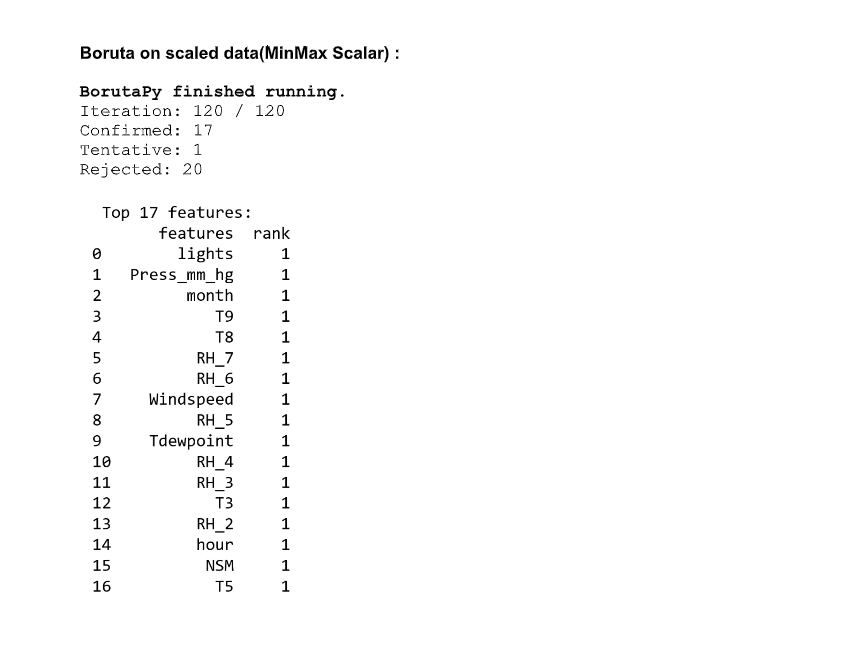
Standardization of datasets is a common requirement for many machine learning estimators implemented in scikit-learn; they might behave badly if the individual features do not more or less look like standard normally distributed data: Gaussian with zero mean and unit variance.

**Boruta on Energy Datasets :**

Boruta is an all relevant feature selection method. It is a wrapper built around the random forest classification algorithm. It tries to capture all the important, interesting features you might have in your dataset with respect to an outcome variable.

1. First, it duplicates the dataset, and shuffle the values in each column. These values are called shadow features. Then, it trains a classifier, such as a Random Forest Classifier, on the dataset. By doing this, you ensure that you can an idea of the importance -via the Mean Decrease Accuracy or Mean Decrease Impurity- for each of the features of your data set. The higher the score, the better or more important.
2. Then, the algorithm checks for each of your real features if they have higher importance. That is, whether the feature has a higher Z-score than the maximum Z-score of its shadow features than the best of the shadow features. If they do, it records this in a vector. These are called a hit. Next, it will continue with another iteration. After a predefined set of iterations, you will end up with a table of these hits
3. At every iteration, the algorithm compares the Z-scores of the shuffled copies of the features and the original features to see if the latter performed better than the former.





**Feature tools on Energy Datasets**:

It is automated feature tools Transforms Transactional and relational datasets into feature matrices for machine learning. Deep Feature Synthesis (DFS) to perform automated feature engineering. DFS is used to create the “Data Science Machine” to automatically build predictive models for complex, multi-table datasets

Each table is called an entity in Featuretools. When 2 two entities have a one-to-many relationship, then “one” enitity, is called the “parent entity”. A relationship between a parent and child is defined like this:

(parent\_entity, parent\_variable, child\_entity, child\_variable)

A minimal input to DFS is a set of entities, a list of relationships, and the “target\_entity” to calculate features for. The ouput of DFS is a feature matrix and the corresponding list of feature defintions.

Example:

feature\_matrix\_customers, features\_defs = ft.dfs(entities=entities,

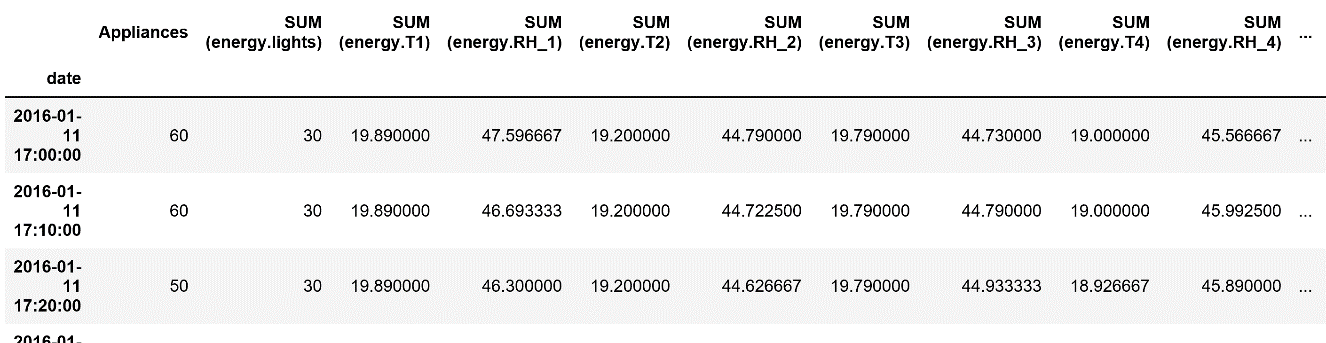
relationships=relationships,

target\_entity="customers"

We can change target entity and get feature matrix for any entities of our choice.

**Output:**

****



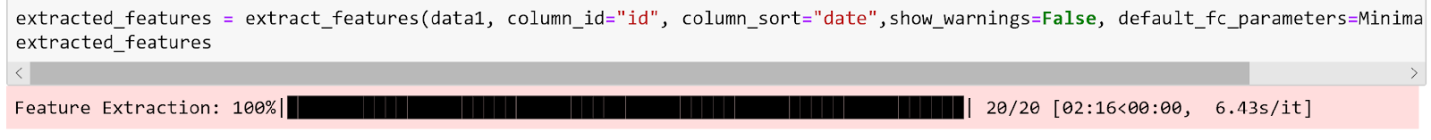
**TSFresh on Energy Datasets:**

• Tsfresh is used to extract characteristics from time series. Time series often contain noise, redundancies or irrelevant information. As a result most of the extracted features will not be useful for the machine learning task at hand.

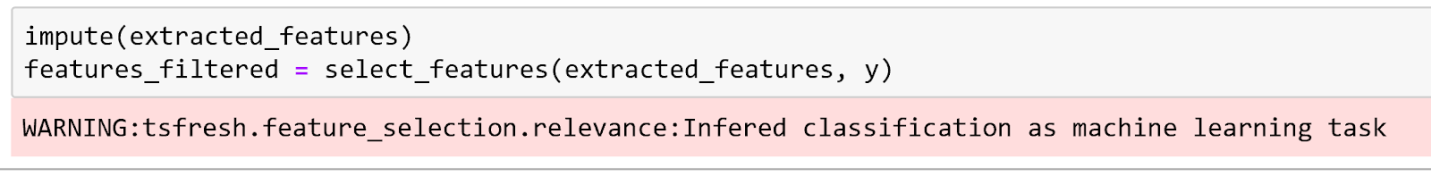
• To avoid extracting irrelevant features, the TSFRESH package has a built-in filtering procedure. This filtering procedure evaluates the explaining power and importance of each characteristic for the regression or classification tasks at hand.

**Output:**

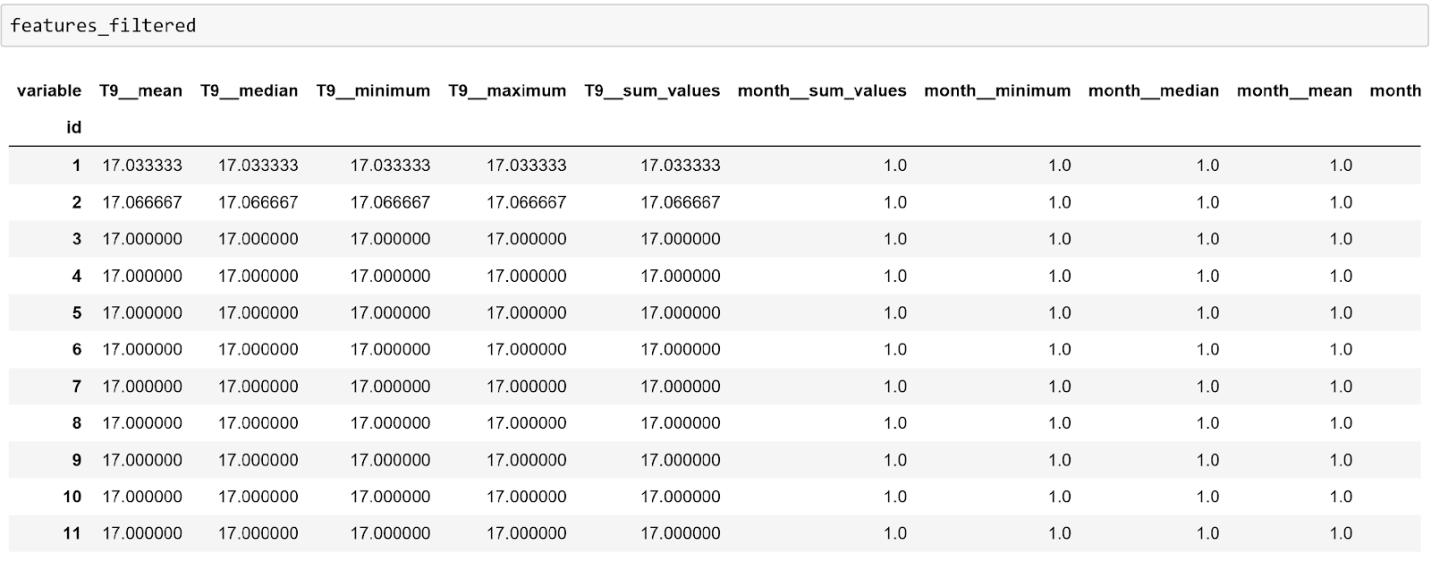
**Feature Extraction**



**Relevant Features filtered out with respect to target Appliances**



**Features**



**TPOT on Energy Datasets:**

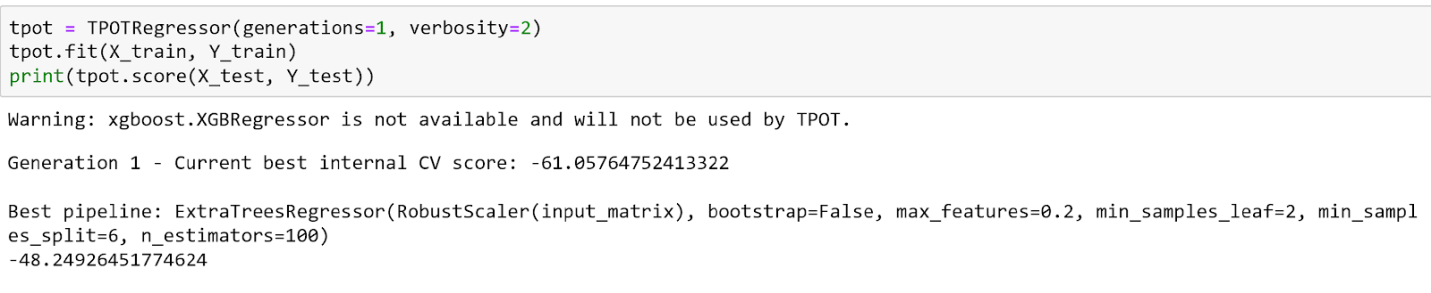
• TPOT is a Python Automated Machine Learning tool that optimizes machine learning pipelines using genetic programming. TPOT will automate the most tedious part of machine learning by intelligently exploring thousands of possible pipelines to find the best one for your data.

• Once TPOT is finished searching (or you get tired of waiting), it provides you with the Python code for the best pipeline it found so you can tinker with the pipeline from there.

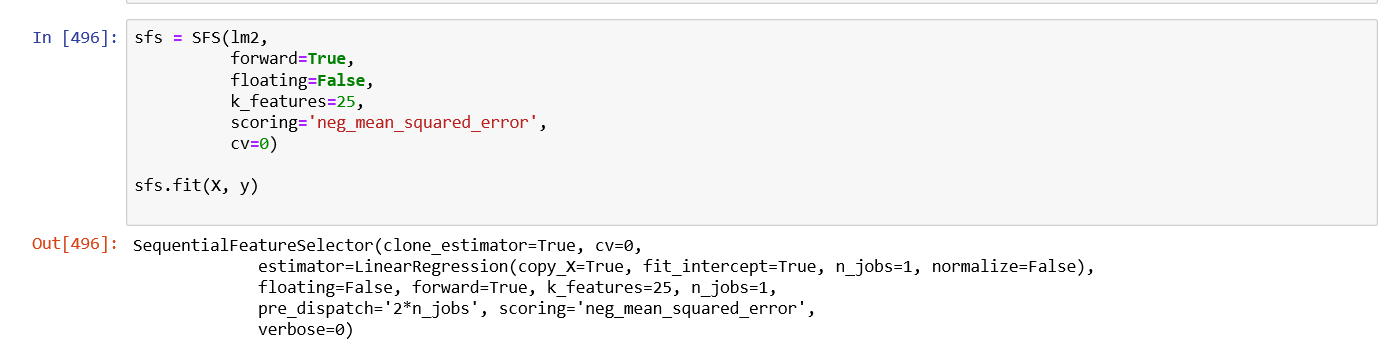
• AutoML algorithms aren't as simple as fitting one model on the dataset; they are considering multiple machine learning algorithms (random forests, linear models, SVMs, etc.) in a pipeline with multiple preprocessing steps (missing value imputation, scaling, PCA, feature selection, etc.), the hyperparameters for all of the models and preprocessing steps, as well as multiple ways to ensemble or stack the algorithms within the pipeline.

• TPOT is meant to be an assistant that gives you ideas on how to solve a particular machine learning problem by exploring pipeline configurations that you might have never considered, then leaves the fine-tuning to more constrained parameter tuning techniques such as grid search.

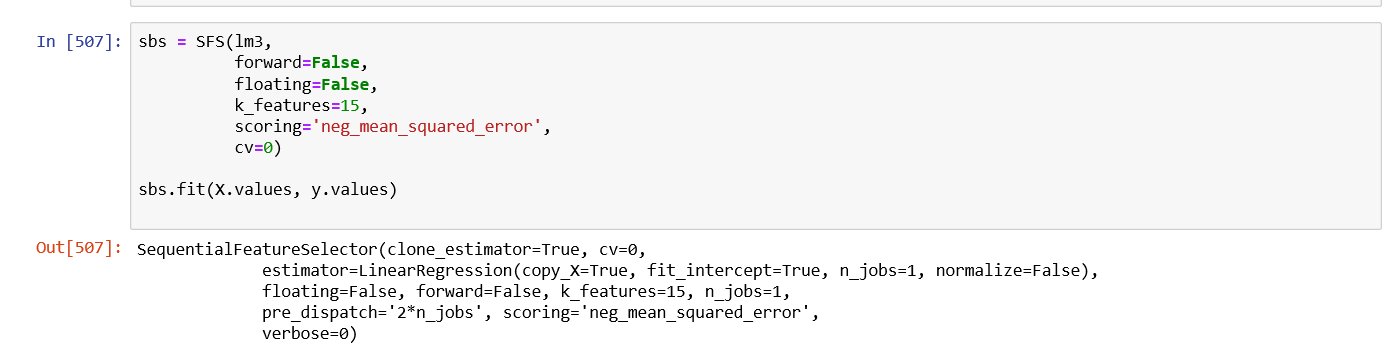
**Output:**



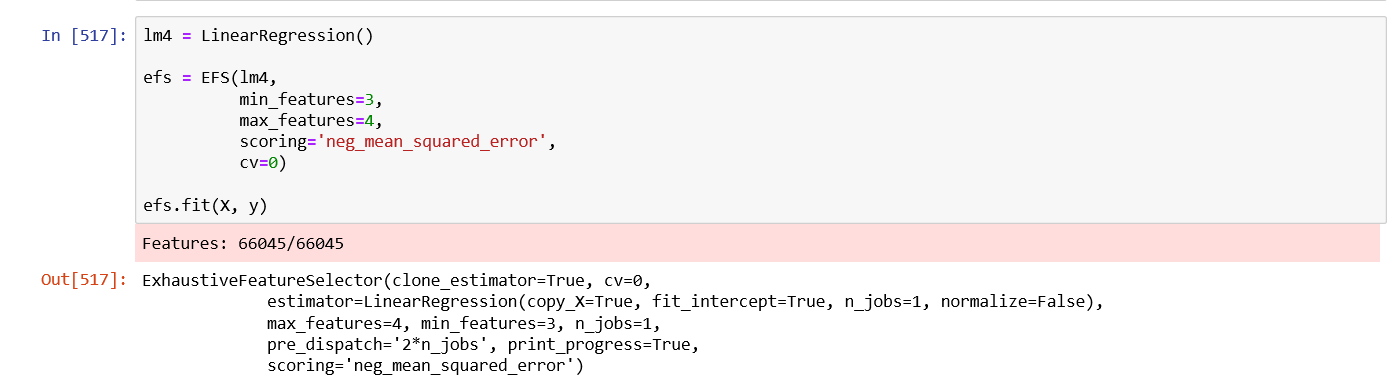
**Sequential Forward Selection**



**Sequential Backward Selection**



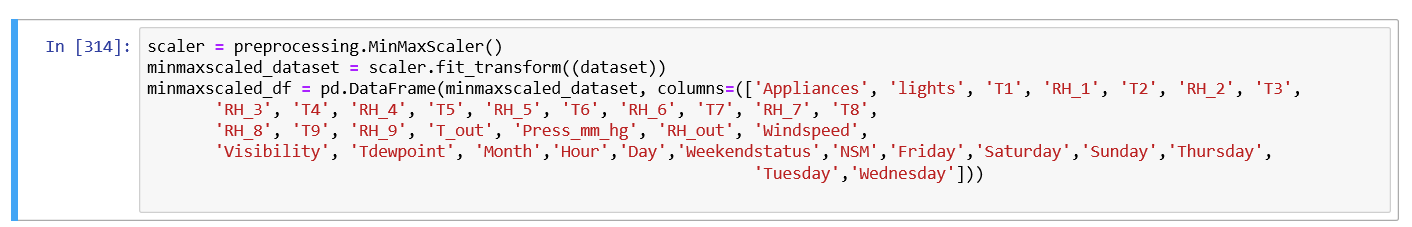
**Exhaustive Selection**

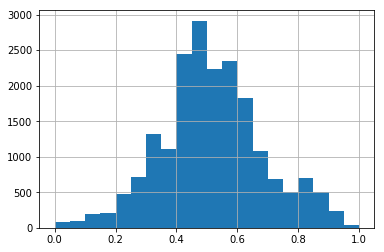


**Scaling Features**

An alternative standardization is scaling features to lie between a given minimum and maximum value, often between zero and one, or so that the maximum absolute value of each feature is scaled to unit size. This can be achieved using MinMaxScaler or MaxAbsScaler, respectively.

1.a MinMaxScaler

****

****

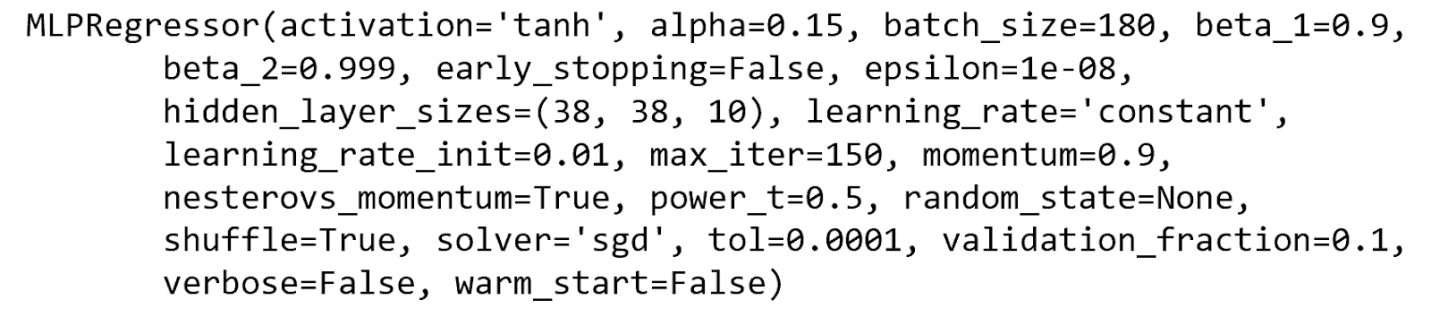
**Section 3: Prediction algorithms**

**NEURAL NETWORKS :**

**Multi Layer Perceptron (MLP)** :

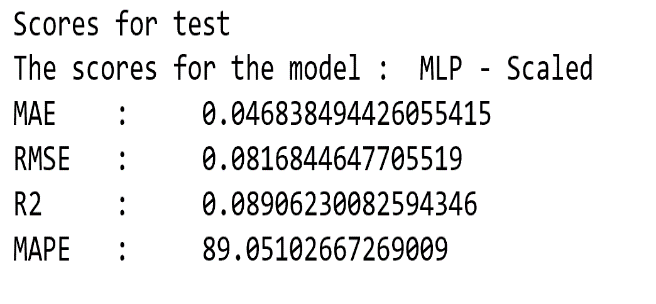
This is a supervised learning algorithm in which we give n dimension input and n dimension output. The model will train itself accordingly, depending on the weights that is given to each input and give us the proper output. The number of hidden layers between the input and the output can be tuned by us.

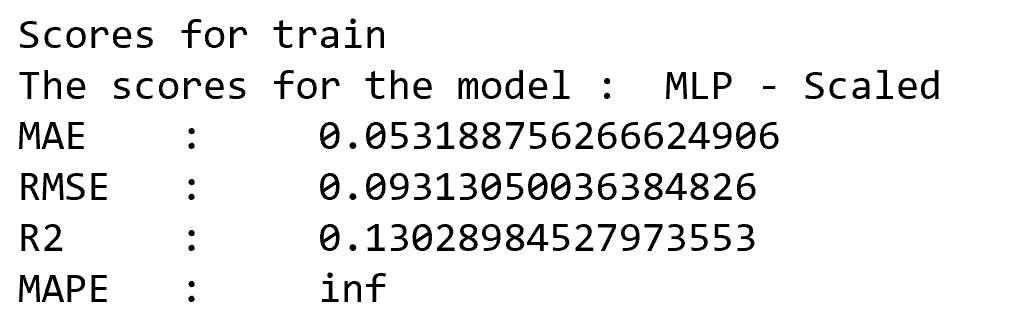
* In this project , MLP regressor with 'sgd' solver is applied
* Input is stored in X which contains all features except for Appliances that are scaled using MinMaxscaler
* Target which is Appliances is stored in Y after scaling
* Test and Train is divided into 30 ,70 ratio respectively.
* Since the optimizer used is 'sgd' we need to provide the learning rate.
* The hidden\_layer\_sizes , which is the number of neurons which doesn't include the input and output sizes.



**Output:**

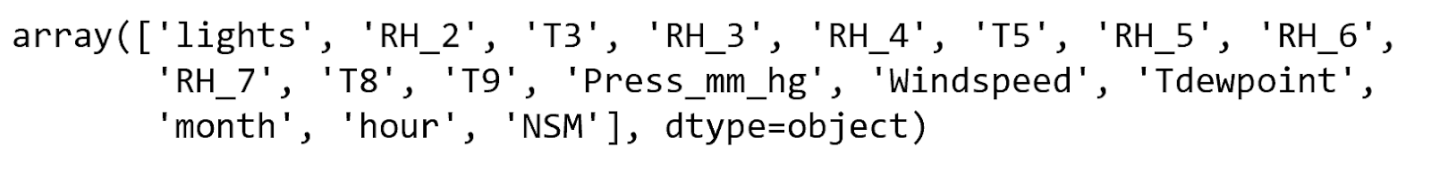
**For Scaled Data**

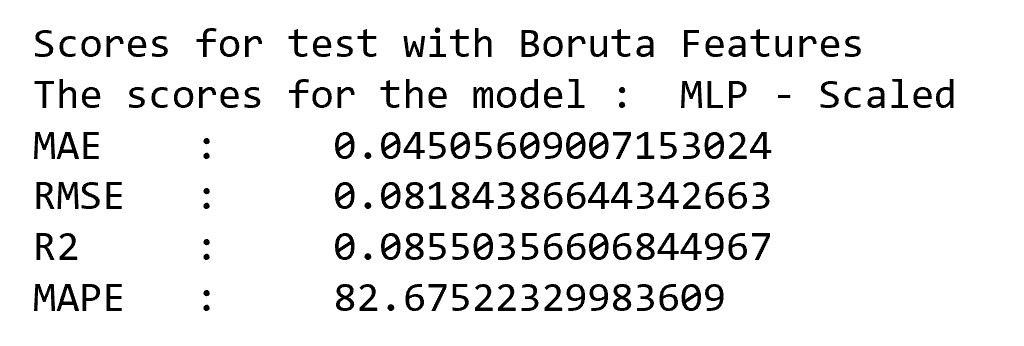
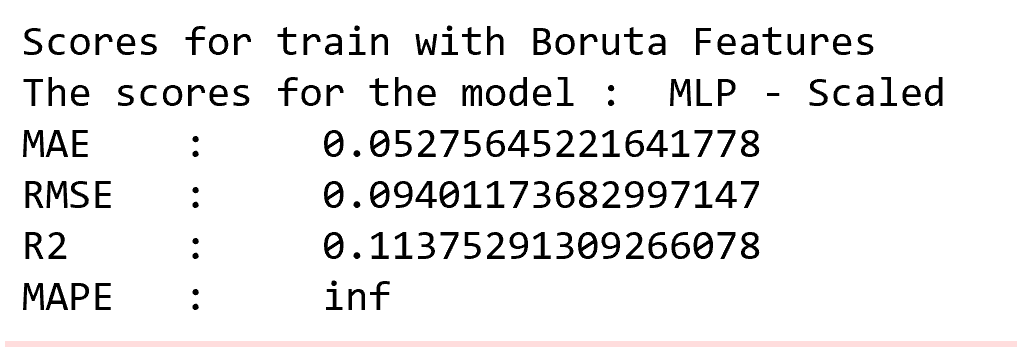




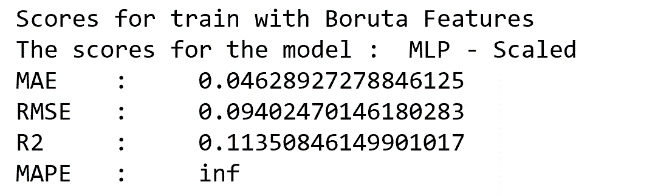
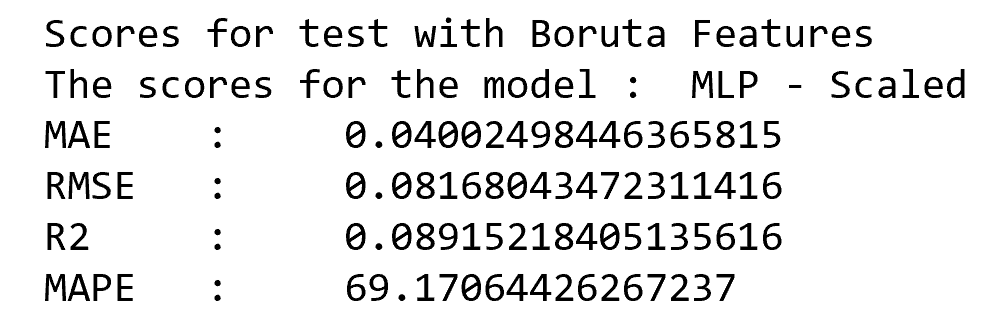
**Using Boruta Features :**

**Columns with Rank = 1**

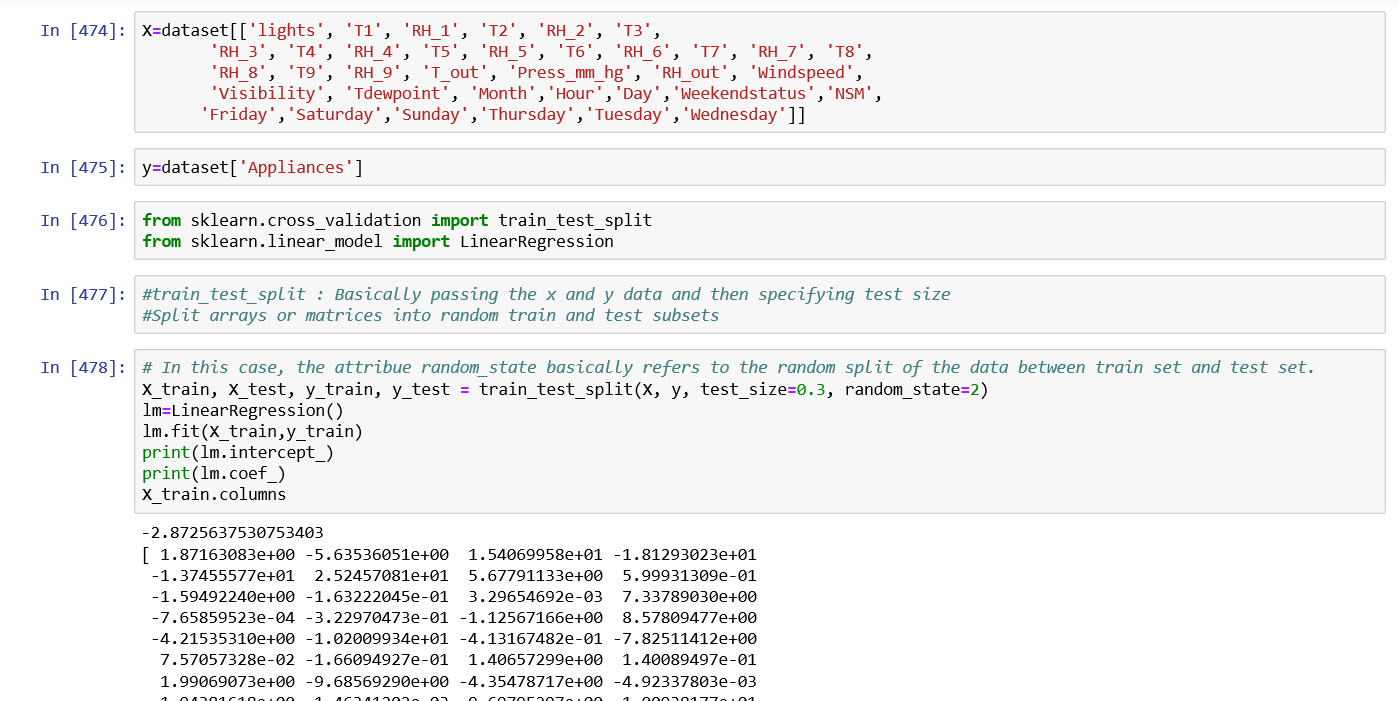


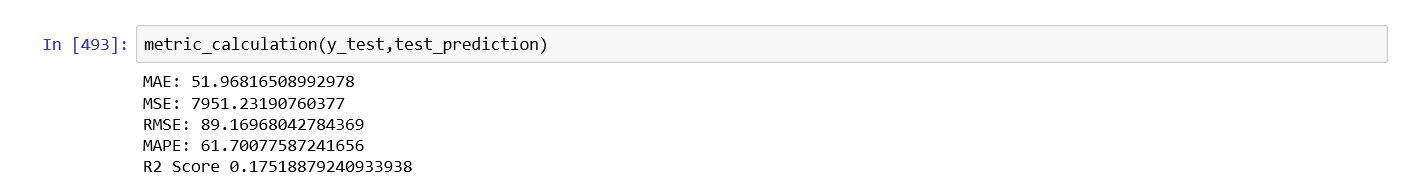
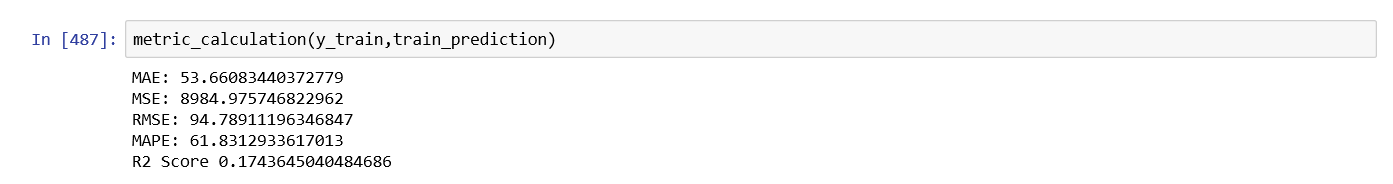
**Boruta After Hyperparameter Tuning :**



**Linear Regression:**

****

**Output for training and testing dataset:**

****

**Support Vector Regression**

The Support Vector Regression (SVR) uses the same principles as the SVM for classification, with only a few minor differences. First of all, because output is a real number it becomes very difficult to predict the information at hand, which has infinite possibilities.

Feature Engineering:

New columns derived from the ‘date’ feature:

day\_of\_week

month

hour

weekend

NSM

Since the dataset had categorical columns, scikit learn’s label encoding followed by one hot encoding applied to the dataset to convert the data into numerical.

Scaling:

Scikit learn’s ‘MinMax’ scaler used.

Performance Metrics (SVR with MinMax scaling):

|  |  |
| --- | --- |
| **Metric** | **Value** |
| Training score | -0.0612 |
| Testing score | -0.0420 |
| R2 | -0.0420 |
| RMSE | 0.1018 |
| MAE | 0.08 |
|  |  |

**SVR using Feature selection:**

Feature selector used: ‘VarianceThreshold’ with default parameters

Scaling:

Scikit learn’s ‘MinMax’ scaler used.

|  |  |
| --- | --- |
| **Metric** | **Value** |
| Training score | 0.3160 |
| Testing score | 0.2504 |
| R2 | 0.2504 |
| RMSE | 0.9110 |
| MAE | 0.38 |
| Accuracy % | 82.22 |

Predicted values (transformed):

array([[ 0.52330909],

      [-0.63417613],

      [ 1.11197285],

      ...,

      [-0.49447382],

      [ 0.17844627],

      [-0.03334546]])

Observation:

Overfitting decreased to a great extent whereas accuracy also decreased few points.

**Random forest regression**

**a) Without Scaling**

The random forest model is a type of additive model that makes predictions by combining decisions from a sequence of base models. More formally we can write this class of models as:

g(x)=f0(x)+f1(x)+f2(x)+...

where the final model g is the sum of simple base models fi. Here, each base classifier is a simple decision tree.

*regressor = RandomForestRegressor(n\_estimators = 20, random\_state = 0)*

*regressor.fit(X\_train, y\_train)*

*y\_pred = regressor.predict(X\_test)*

*y\_pred=y\_pred.reshape(-1,1)*

Scaling used: No scaling used

|  |  |
| --- | --- |
| **Metric** | **Value** |
| Training score | 0.9270 |
| Testing score | 0.5167 |
| R2 | 0.5167 |
| RMSE | 0.7315 |
| MAE | 0.35 |
|  |  |

b) **With Scaling**

Scaling type: Scikit learn’s MinMax

Regression parameters:

*from sklearn.ensemble import RandomForestRegressor*

*regressor = RandomForestRegressor(n\_estimators = 20, random\_state = 0)*

*regressor.fit(X\_train, y\_train)*

Feature Importance:

Feature Importance:

              importance

NSM              0.165534

T3               0.053716

Press\_mm\_hg      0.044042

RH\_3             0.043836

RH\_5             0.040863

T8               0.034747

RH\_4             0.034165

RH\_2             0.032413

RH\_out           0.030802

lights           0.030069

Tdewpoint        0.029008

RH\_8             0.027890

RH\_6             0.027746

RH\_1             0.026894

T4               0.026723

T6               0.026621

RH\_7             0.026105

T2               0.025486

RH\_9             0.023874

Visibility       0.023187

T1               0.022989

T5               0.022828

T\_out            0.021950

T7               0.021697

Windspeed        0.020606

T9               0.020384

rv2              0.018368

hour             0.018050

rv1              0.015655

day\_of\_week      0.014735

month\_2          0.009224

month\_4          0.004125

month            0.004062

day\_of\_week\_2    0.001913

month\_3          0.001661

day\_of\_week\_1    0.001267

day\_of\_week\_3    0.001212

day\_of\_week\_4    0.001075

day\_of\_week\_5    0.001060

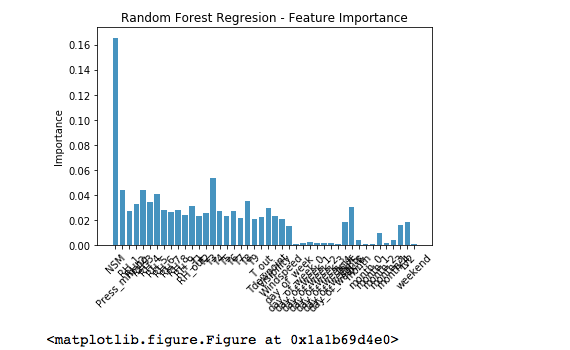
day\_of\_week\_6    0.000940

month\_0          0.000897

weekend          0.000800

day\_of\_week\_0    0.000432

month\_1          0.000350



As we can see the ‘NSM’ feature has the most impact on the model prediction.

Performance Metrics (Random Forest regression with scaling):

|  |  |
| --- | --- |
| **Metric** | **Value** |
| Training score | 0.9261 |
| Testing score | 0.5081 |
| R2 | 0.5081 |
| RMSE | 0.0700 |
| MAE | 0.03 |
|  |  |

**c) With feature selection:**

Feature selection:  RFE

Feature selection parameters: Default

*selector = RFE(regressor, step=1)*

Features selected:

*selector.support\_*

*selector.ranking\_*

array([ 1,  5, 1, 1, 1,  1, 1, 1, 1, 1,  1, 1, 1, 3, 2, 1,  1,

       1, 1, 1,  1, 1, 4, 6,  1, 1, 8, 7, 11,  9, 20, 1, 21, 14,

      13, 16, 19, 17, 18, 22, 23, 10, 12, 15])

|  |  |
| --- | --- |
| **Metric** | **Value** |
| Training score | 0.9263 |
| Testing score | 0.5036 |
| R2 | 0.5036 |
| RMSE | 0.00493941373562 |
| MAE | 0.0333398843082 |

d) **Hyper parameter tuning**

Type: GridSearchCV

*param\_grid1 = {'n\_estimators' : [20, 40, 60],'random\_state' : [0, 2]}*

Parameters suggested:

{'n\_estimators': 40, 'random\_state': 2}

|  |  |
| --- | --- |
| **Metric** | **Value** |
| Training score | 0.9342 |
| Testing score | 0.5174 |
| R2 | 0.5054 |
| RMSE | 0.0702 |
| MAE | 0.03 |

**Observation**:

Random Forest Regression model performed well in comparison with the Support Vector Regression model. Performance of the random forest regression model increased 97% with the introduction of scaling but there was no significant increase in the metrics after the introduction of hyperparameter tuning.

**Section 4: Model Validation and Selection**

**MLP Model Validation and Hyperparameter Tuning**

**Hyperparameters and Model Validation**

Basic recipe for applying a supervised machine learning model:

1. Choose a class of model

2. Choose model hyperparameters

3. Fit the model to the training data

4. Use the model to predict labels for new data <br>

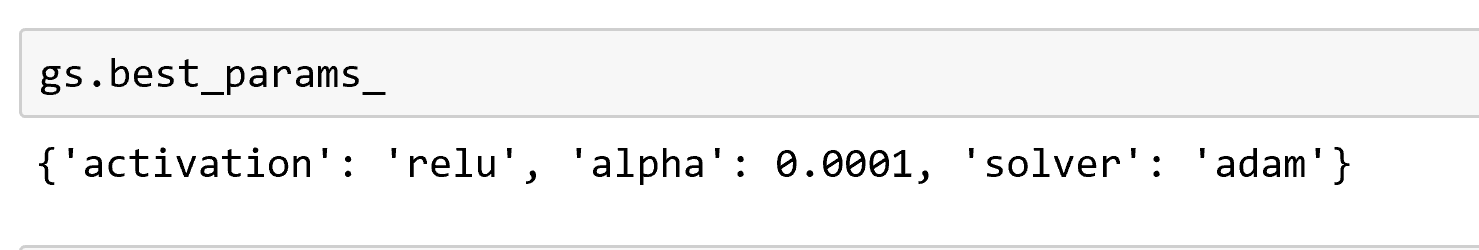
we need a way to validate that our model and our hyperparameters are a good fit to the data

**Model validation via cross-validation**

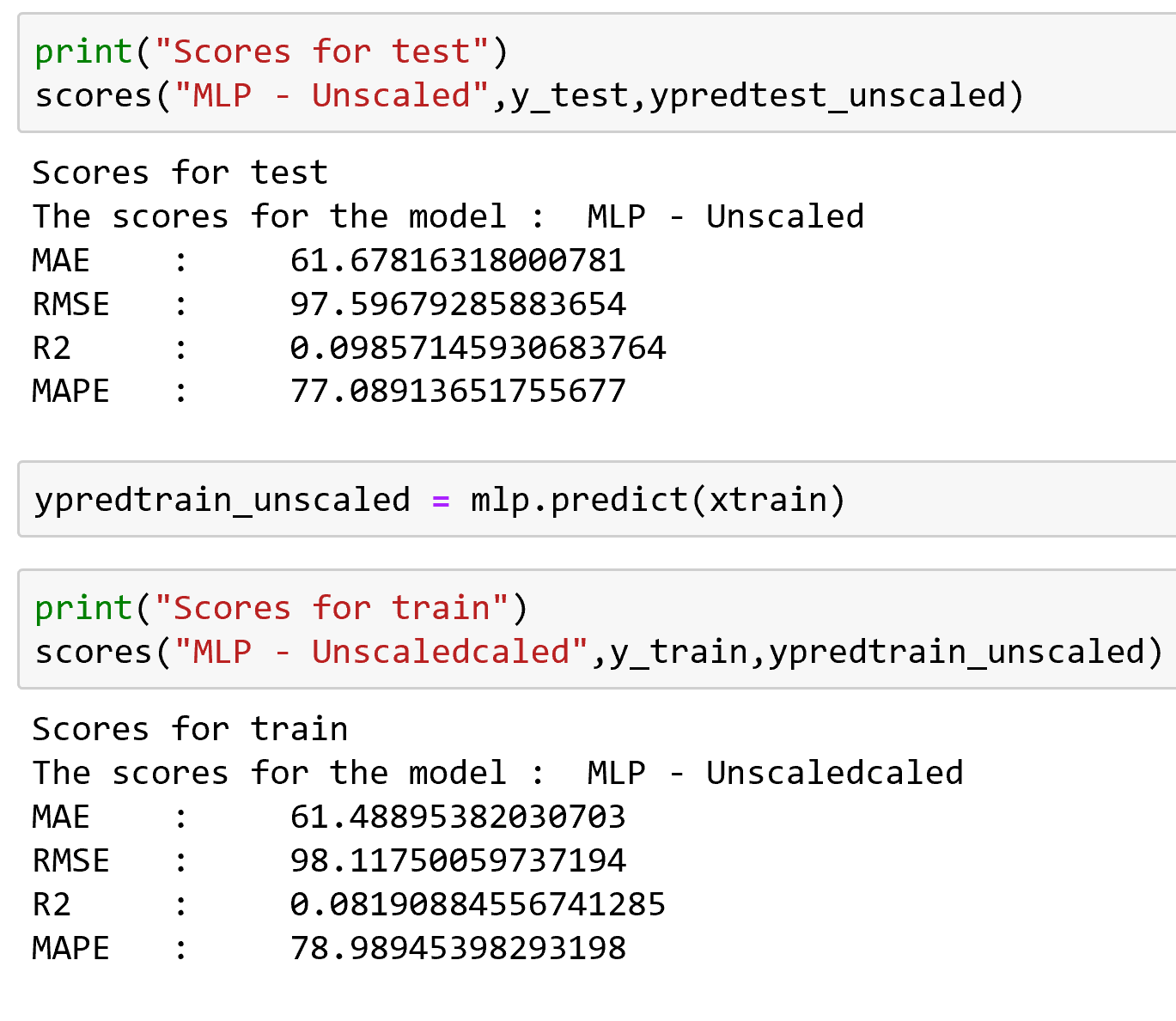
* One disadvantage of using a holdout set for model validation is that we have lost a portion of our data to the model training. This is not optimal, and can cause problems – especially if the initial set of training data is small.
* One way to address this is to use cross-validation; that is, to do a sequence of fits where each subset of the data is used both as a training set and as a validation set.
* Form of cross-validation is a two-fold cross-validation—that is, one in which we have split the data into two sets and used each in turn as a validation set. We could expand on this idea to use even more trials, and more folds in the data. Split the data into five groups, and use each of them in turn to evaluate the model fit on the other 4/5 of the data. This would be rather tedious to do by hand, and so we can use Scikit-Learn's

**Output:**

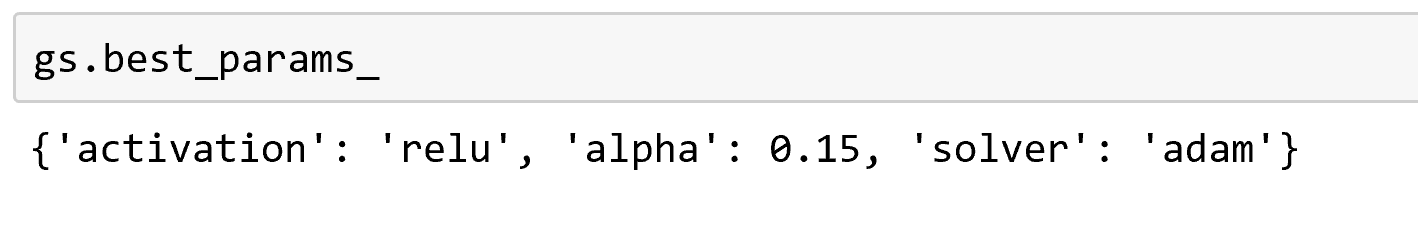
**Best parameters with Unscaled Data**

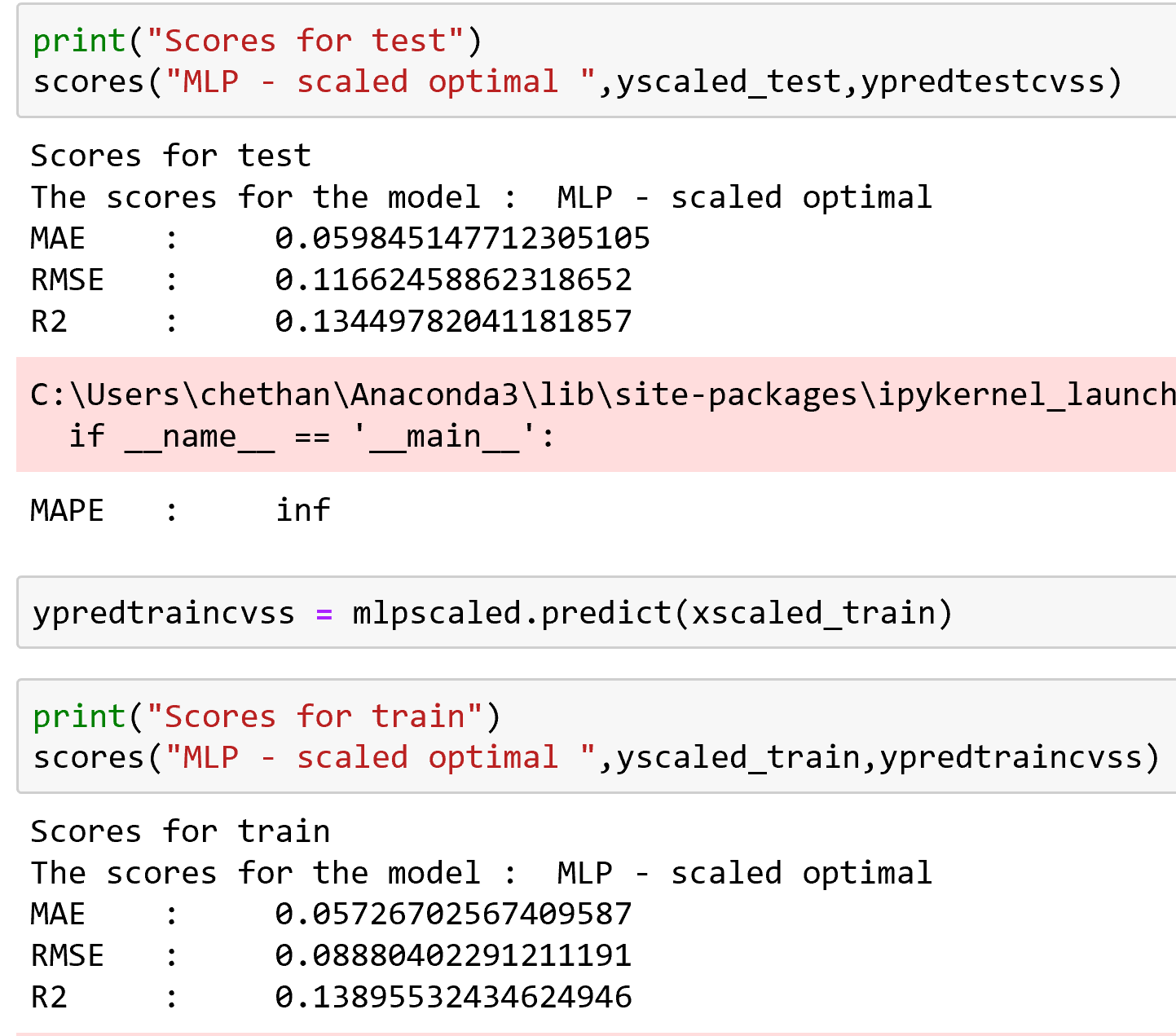


**Scores :**



**Best parameters with Data**





**Section 5 : Final Pipeline**

The pipeline has been created to automate the entire model from data ingestion to final model prediction

