Machine Learning With Energy Dataset

OBJECTIVE

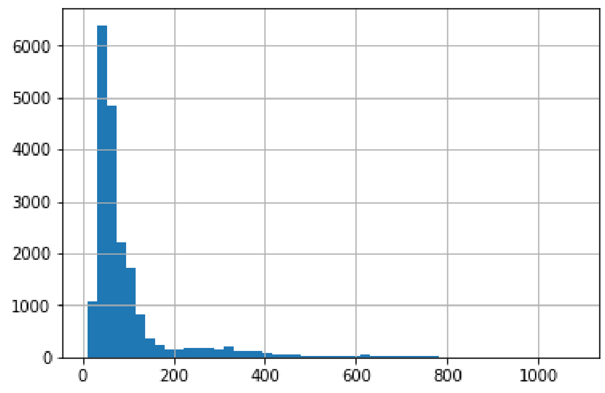
The report summarizes the design and implementation of the data wrangling performed on the Appliances Energy Consumption data set.
This report is divided into 5 sections.
Section 1: Exploratory Data Analysis
Section 2: Feature engineering and Feature Selection
Section 3: Prediction algorithms
Section 4: Model Validation and Selection
Section 5: Final pipeline

Section 1: Exploratory Data Analysis

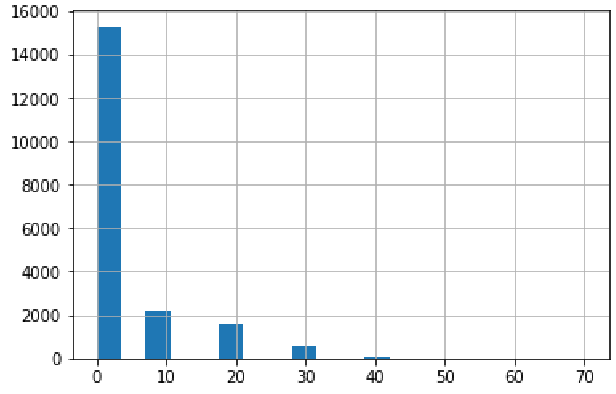
Observations made from the dataset:

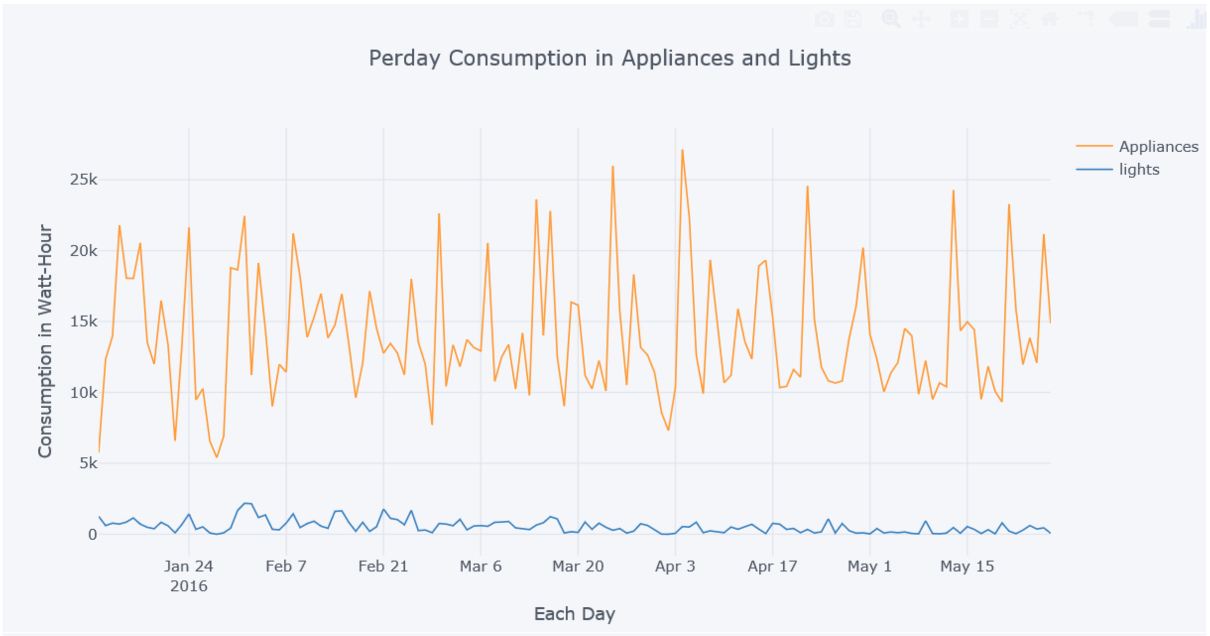
1. The date column is useful only if the date column information is divided into month, year columns. So we need to create new columns (month and year in our dataframe)for better analysis
2. We do not have any missing values in the dataset

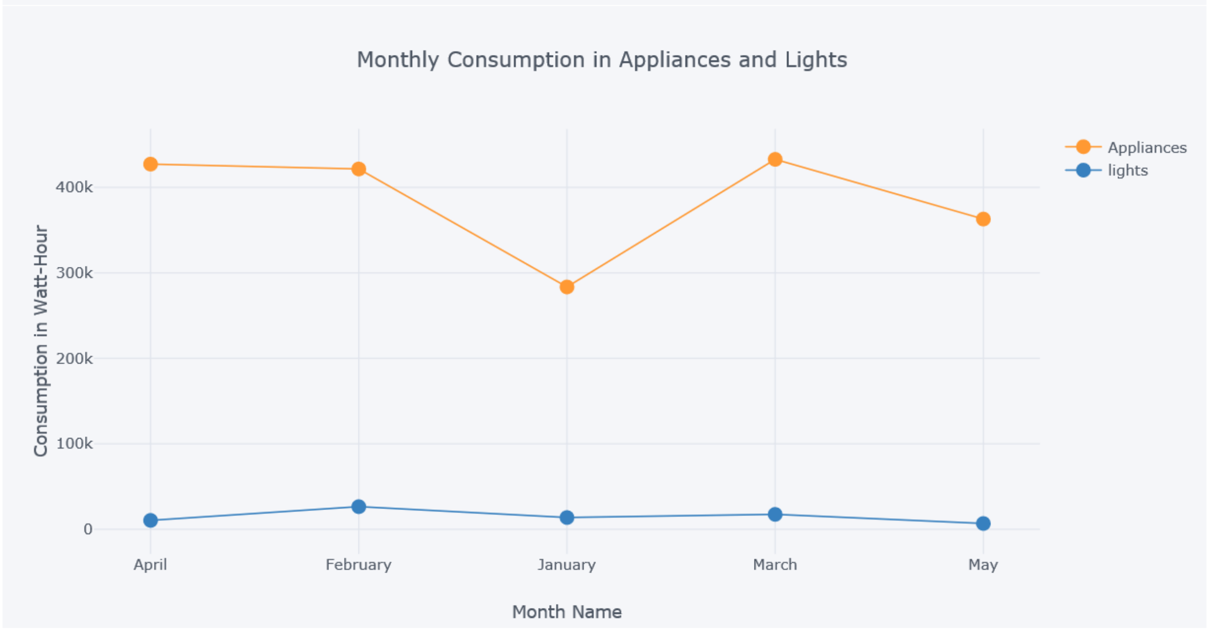
Histograms of appliances to get the frequency

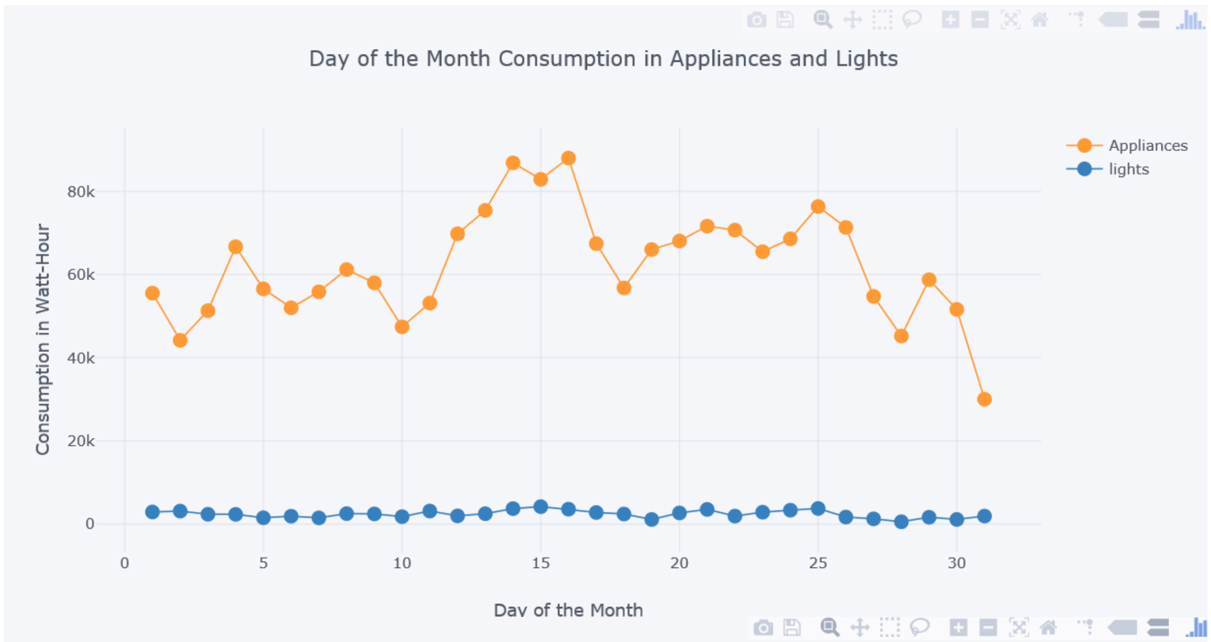


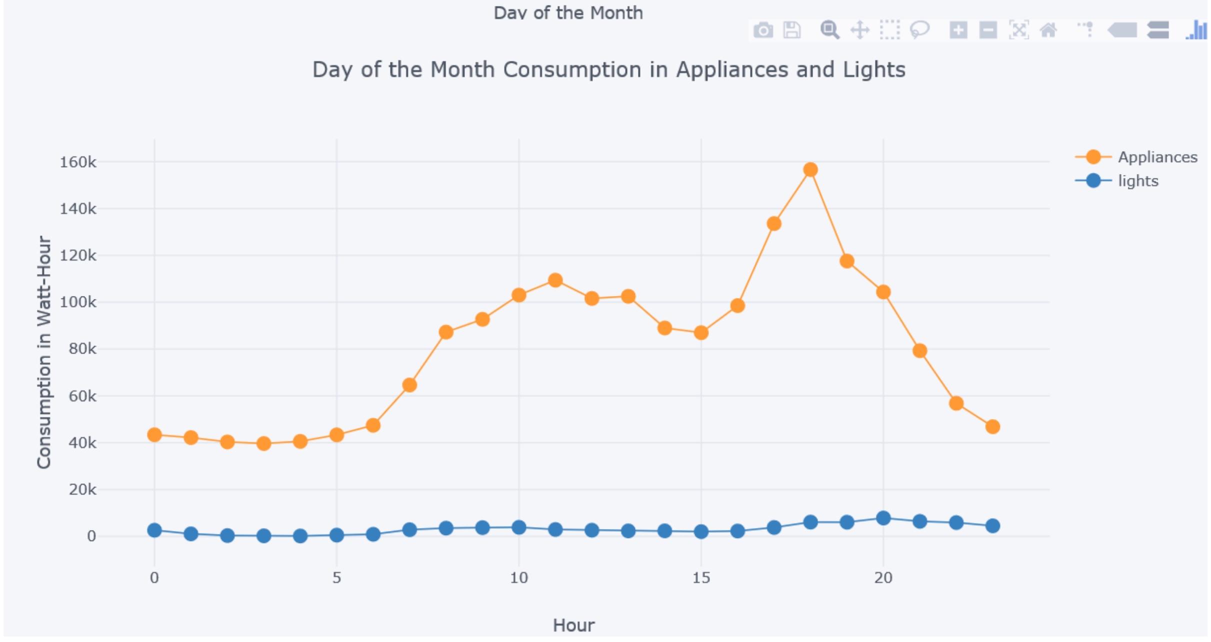
Histograms of appliances to get the frequency



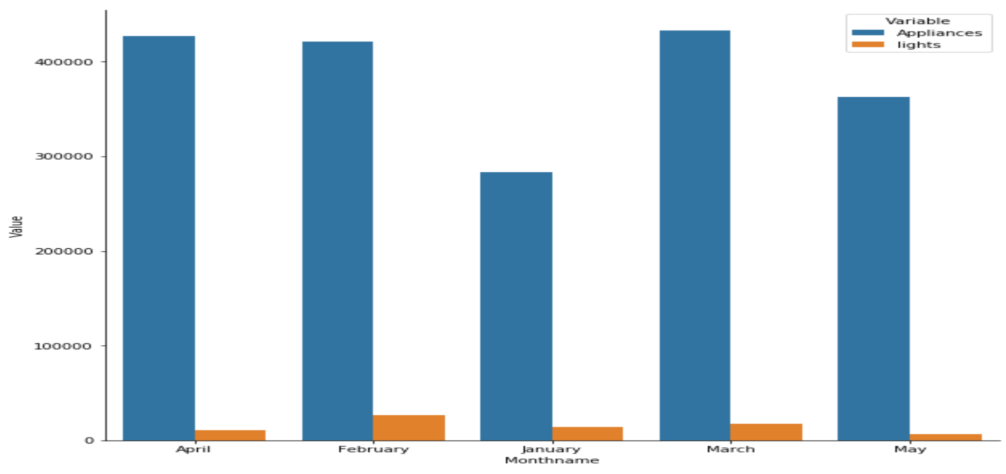

Perday Consumption in Appliances and Lights
Appliances
lights
Consumption in Watt-Hour
25k
20k
15k
10k
5k
0
Jan 24
2016
Feb 7
Feb 21
Mar 6
Mar 20
Apr 3
Apr 17
May 1
May 15
Each Day


Monthly Consumption in Appliances and Lights
Appliances
lights
Consumption in Watt-Hour
400k
300k
200k
100k
0
April
February
January
March
May
Month Name





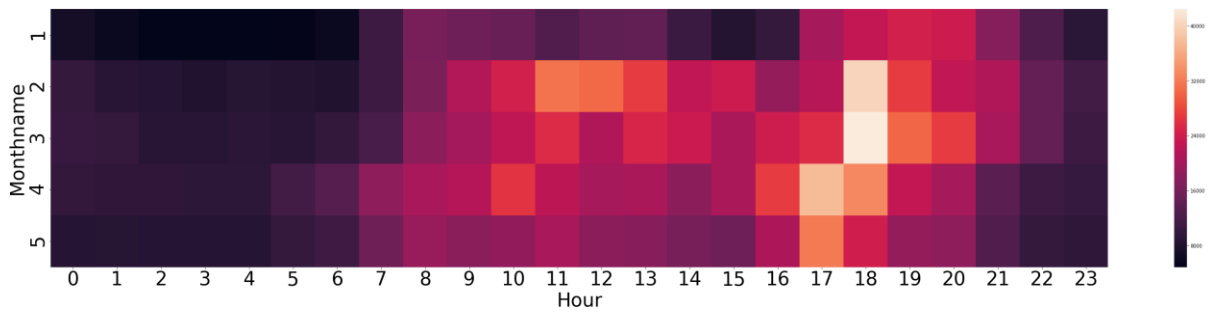
Energy consumption by appliances vs energy consumption by light



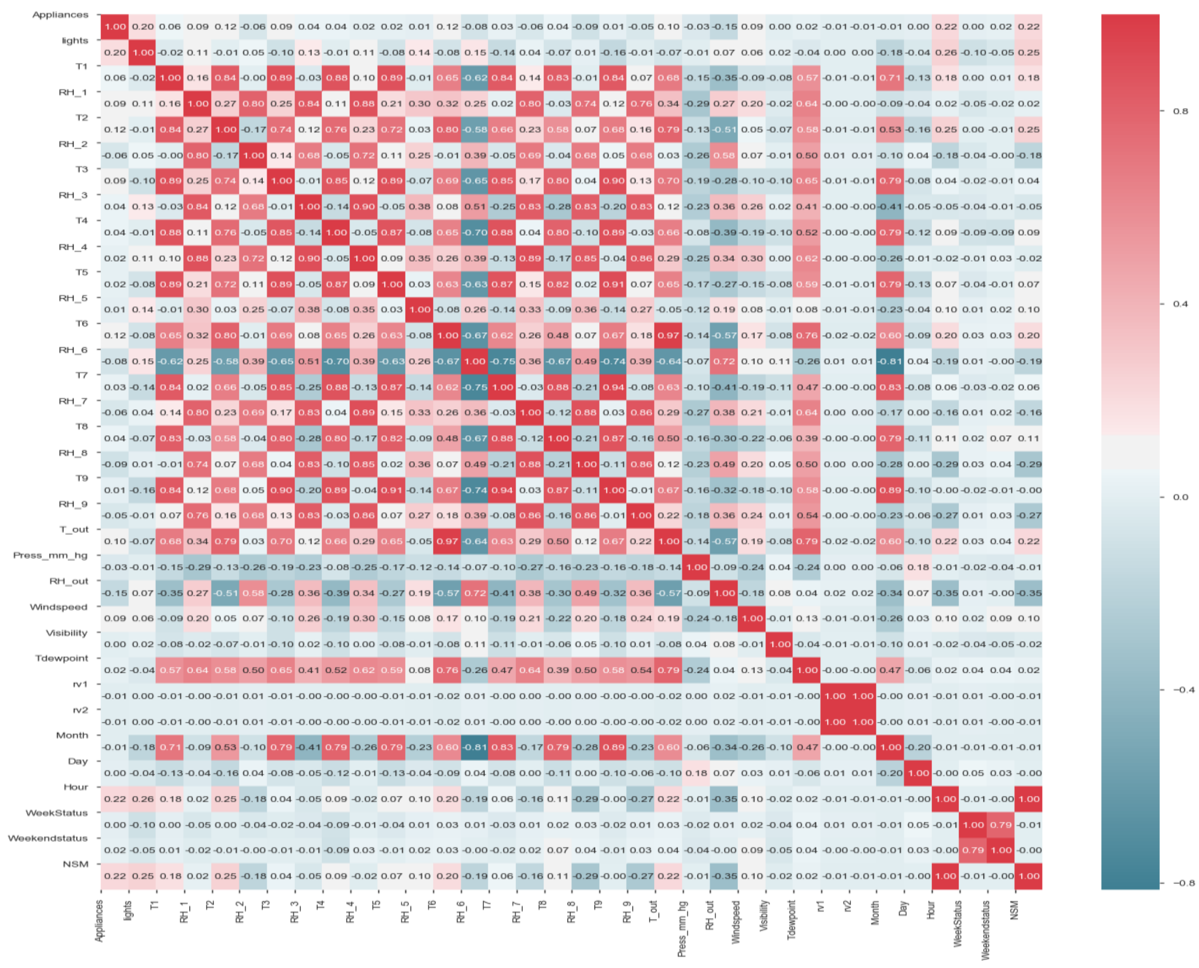
Observation:

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.

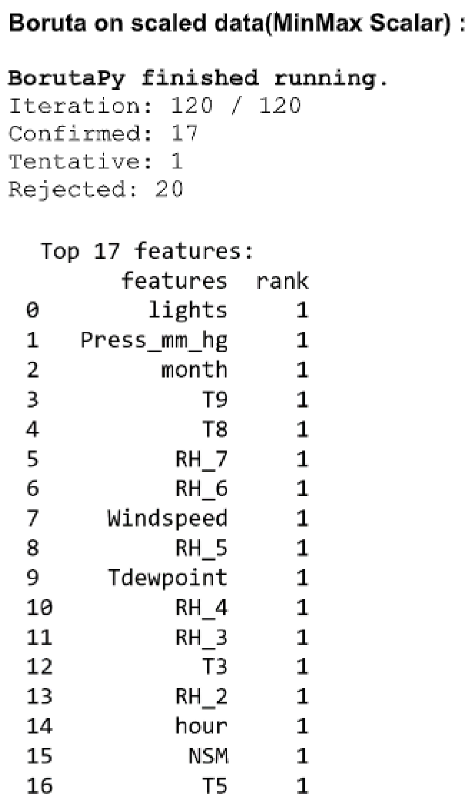
Boruta on Energy Datasets :

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.

Output:

```
BorutaPy finished running.
Iteration: 100 / 100
Confirmed: 17
Tentative: 1
Rejected: 20
19
Ranking of all the features features rank
1 lights 1                   22 T2   6
2 Press_mm_hg 1              23 RH_8 7
3 month 1                    24 RH_9 8
4 T9 1                       25 T1 9
4 T8 1                       26 RH_1 10
5 RH_7 1                     27 Friday 11
6 RH_6 1                     28 rv2 12
7 Windspeed 1                29 Visibility 12
8 RH_5 1                     30 rv1 14
9 Tdewpoint 1                31 day 14
10 RH_4 1                    32 weekendstatus 16
11 RH_3 1                    33 Sunday 17
12 T3 1                      34 Wednesday 18
13 RH_2 1                    35 Tuesday  19
14 hour 1                    36 Saturday 20
15 NSM 1
16 T5 1
17 T_out 2
18 T4 3
19 T7 3
20 RH_out 3
21 T6 5
```



Boruta on scaled data(MinMax Scalar) :

```
BorutaPy finished running.
Iteration: 120 / 120
Confirmed: 17
Tentative: 1
Rejected: 20

  Top 17 features:
        features  rank
0         lights     1
1    Press_mm_hg     1
2          month     1
3             T9     1
4             T8     1
5           RH_7     1
6           RH_6     1
7      Windspeed     1
8           RH_5     1
9      Tdewpoint     1
10          RH_4     1
11          RH_3     1
12            T3     1
13          RH_2     1
14          hour     1
15           NSM     1
16            T5     1
```

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:

```
y_featuretools = df[['date','Appliances']]
x_featuretools = df.drop(["Appliances"],axis=1)
```

```
entities1 = {
    "energy" : (x_featuretools, "date"),"appliances" :(y_featuretools,"date")
}
```

```
rel = [("appliances","date","energy","date")]
```

```
feature_matrix, feature_defs = ft.dfs(entities=entities1,relationships=rel,target_entity="appliances",verbose=1)
```

```
Built 163 features
Elapsed: 03:51 | Remaining: 00:00 | Progress: 100%|██████████████████████████████| Calculated: 11/11 chunks
```

	Appliances	SUM (energy.lights)	SUM (energy.T1)	SUM (energy.RH_1)	SUM (energy.T2)	SUM (energy.RH_2)	SUM (energy.T3)	SUM (energy.RH_3)	SUM (energy.T4)	SUM (energy.RH_4)	...
date											
2016-01-11 17:00:00	60	30	19.890000	47.596667	19.200000	44.790000	19.790000	44.730000	19.000000	45.566667	...
2016-01-11 17:10:00	60	30	19.890000	46.693333	19.200000	44.722500	19.790000	44.790000	19.000000	45.992500	...
2016-01-11 17:20:00	50	30	19.890000	46.300000	19.200000	44.626667	19.790000	44.933333	18.926667	45.890000	...
2016-01-											

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

```
extracted_features = extract_features(data1, column_id="id", column_sort="date",show_warnings=False, default_fc_parameters=Minima
extracted_features
```

```
Feature Extraction: 100%|██████████████████████████████| 20/20 [02:16<00:00,  6.43s/it]
```

Relevant Features filtered out with respect to target Appliances

```
impute(extracted_features)
features_filtered = select_features(extracted_features, y)
```

```
WARNING:tsfresh.feature_selection.relevance:Infered classification as machine learning task
```

Features

```
features_filtered
```

variable	T9__mean	T9__median	T9__minimum	T9__maximum	T9__sum_values	month__sum_values	month__minimum	month__median	month__mean	month
id										
1	17.033333	17.033333	17.033333	17.033333	17.033333	1.0	1.0	1.0	1.0	
2	17.066667	17.066667	17.066667	17.066667	17.066667	1.0	1.0	1.0	1.0	
3	17.000000	17.000000	17.000000	17.000000	17.000000	1.0	1.0	1.0	1.0	
4	17.000000	17.000000	17.000000	17.000000	17.000000	1.0	1.0	1.0	1.0	
5	17.000000	17.000000	17.000000	17.000000	17.000000	1.0	1.0	1.0	1.0	
6	17.000000	17.000000	17.000000	17.000000	17.000000	1.0	1.0	1.0	1.0	
7	17.000000	17.000000	17.000000	17.000000	17.000000	1.0	1.0	1.0	1.0	
8	17.000000	17.000000	17.000000	17.000000	17.000000	1.0	1.0	1.0	1.0	
9	17.000000	17.000000	17.000000	17.000000	17.000000	1.0	1.0	1.0	1.0	
10	17.000000	17.000000	17.000000	17.000000	17.000000	1.0	1.0	1.0	1.0	
11	17.000000	17.000000	17.000000	17.000000	17.000000	1.0	1.0	1.0	1.0	

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:

```
tpot = TPOTRegressor(generations=1, verbosity=2)
tpot.fit(X_train, Y_train)
print(tpot.score(X_test, Y_test))
```

```
Warning: xgboost.XGBRegressor is not available and will not be used by TPOT.

Generation 1 - Current best internal CV score: -61.05764752413322

Best pipeline: ExtraTreesRegressor(RobustScaler(input_matrix), bootstrap=False, max_features=0.2, min_samples_leaf=2, min_sampl
es_split=6, n_estimators=100)
-48.24926451774624
```

Sequential Forward Selection

```
In [496]: sfs = SFS(lm2,
                    forward=True,
                    floating=False,
                    k_features=25,
                    scoring='neg_mean_squared_error',
                    cv=0)

          sfs.fit(X, y)

Out[496]: SequentialFeatureSelector(clone_estimator=True, cv=0,
                         estimator=LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False),
                         floating=False, forward=True, k_features=25, n_jobs=1,
                         pre_dispatch='2*n_jobs', scoring='neg_mean_squared_error',
                         verbose=0)
```

Sequential Backward Selection

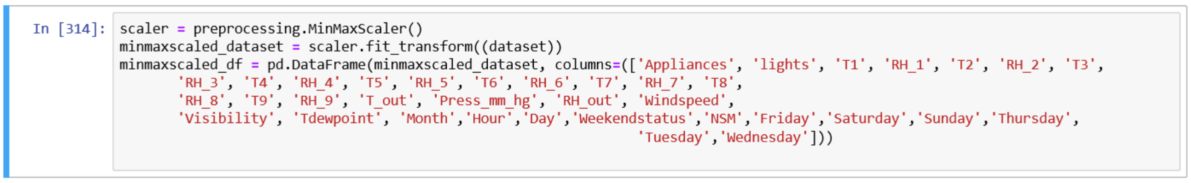
```
In [507]: sbs = SFS(lm3,
                    forward=False,
                    floating=False,
                    k_features=15,
                    scoring='neg_mean_squared_error',
                    cv=0)

          sbs.fit(X.values, y.values)

Out[507]: SequentialFeatureSelector(clone_estimator=True, cv=0,
                         estimator=LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False),
                         floating=False, forward=False, k_features=15, n_jobs=1,
                         pre_dispatch='2*n_jobs', scoring='neg_mean_squared_error',
                         verbose=0)
```

Exhaustive Selection

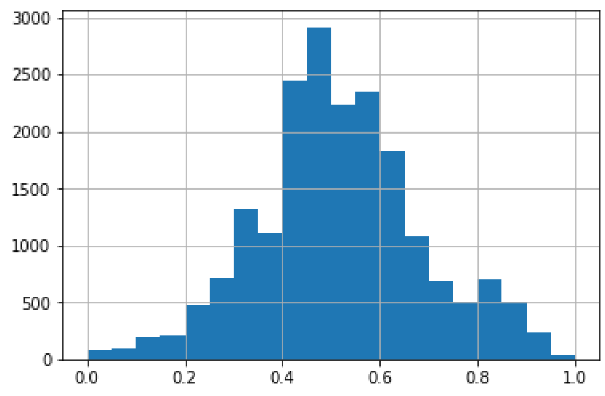
```
In [517]: lm4 = LinearRegression()

          efs = EFS(lm4,
                    min_features=3,
                    max_features=4,
                    scoring='neg_mean_squared_error',
                    cv=0)

          efs.fit(X, y)

          Features: 66045/66045

Out[517]: ExhaustiveFeatureSelector(clone_estimator=True, cv=0,
                         estimator=LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False),
                         max_features=4, min_features=3, n_jobs=1,
                         pre_dispatch='2*n_jobs', print_progress=True,
                         scoring='neg_mean_squared_error')
```

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



```
In [314]: scaler = preprocessing.MinMaxScaler()
          minmaxscaled_dataset = scaler.fit_transform((dataset))
          minmaxscaled_df = pd.DataFrame(minmaxscaled_dataset, columns=(['Appliances', 'lights', 'T1', 'RH_1', 'T2', 'RH_2', 'T3',
                 'RH_3', 'T4', 'RH_4', 'T5', 'RH_5', 'T6', 'RH_6', 'T7', 'RH_7', 'T8',
                 'RH_8', 'T9', 'RH_9', 'T_out', 'Press_mm_hg', 'RH_out', 'Windspeed',
                 'Visibility', 'Tdewpoint', 'Month','Hour','Day','Weekendstatus','NSM','Friday','Saturday','Sunday','Thursday',
                                                          'Tuesday','Wednesday']))
```



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.

```
MLPRegressor(activation='tanh', alpha=0.15, batch_size=180, beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(38, 38, 10), learning_rate='constant',
       learning_rate_init=0.01, max_iter=150, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=None,
       shuffle=True, solver='sgd', tol=0.0001, validation_fraction=0.1,
       verbose=False, warm_start=False)
```

Output:

For Scaled Data

```
Scores for test
The scores for the model :  MLP - Scaled
MAE     :      0.046838494426055415
RMSE    :      0.0816844647705519
R2      :      0.08906230082594346
MAPE    :      89.05102667269009
```

```
Scores for train
The scores for the model :  MLP - Scaled
MAE     :      0.053188756266624906
RMSE    :      0.09313050036384826
R2      :      0.13028984527973553
MAPE    :      inf
```

Using Boruta Features :
Columns with Rank = 1

```
array(['lights', 'RH_2', 'T3', 'RH_3', 'RH_4', 'T5', 'RH_5', 'RH_6',
       'RH_7', 'T8', 'T9', 'Press_mm_hg', 'Windspeed', 'Tdewpoint',
       'month', 'hour', 'NSM'], dtype=object)
```

```
Scores for test with Boruta Features
The scores for the model :  MLP - Scaled
MAE     :      0.04505609007153024
RMSE    :      0.08184386644342663
R2      :      0.08550356606844967
MAPE    :      82.67522329983609
```

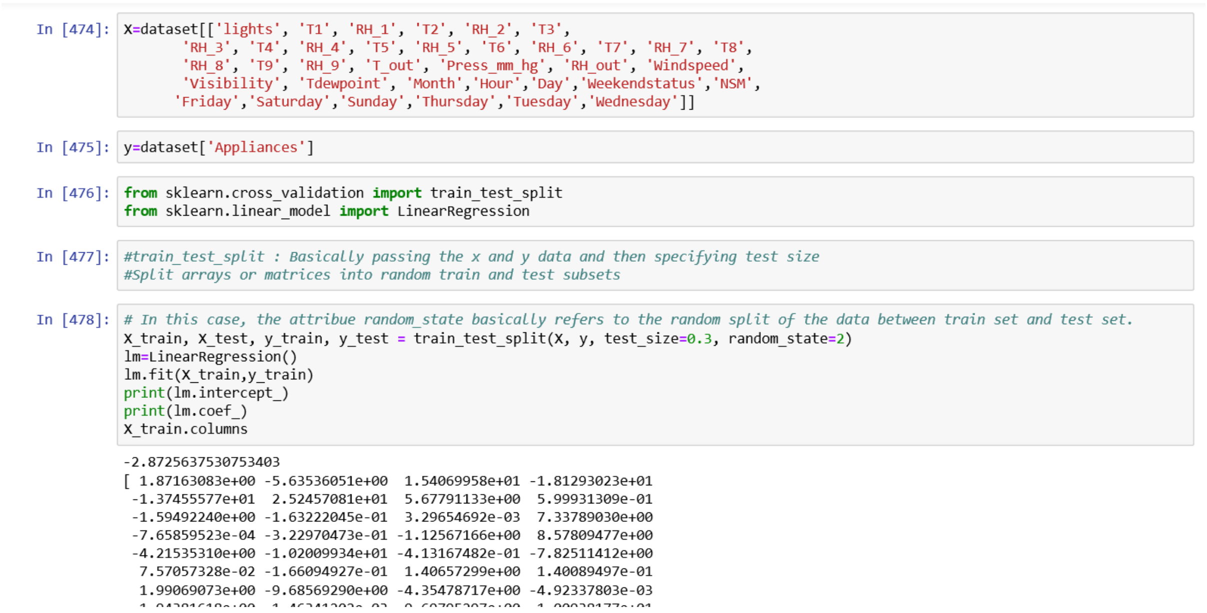
```
Scores for train with Boruta Features
The scores for the model :  MLP - Scaled
MAE     :      0.05275645221641778
RMSE    :      0.09401173682997147
R2      :      0.11375291309266078
MAPE    :      inf
```

Boruta After Hyperparameter Tuning :

```
Scores for test with Boruta Features
The scores for the model :  MLP - Scaled
MAE     :      0.04002498446365815
RMSE    :      0.08168043472311416
R2      :      0.08915218405135616
MAPE    :      69.17064426267237
```

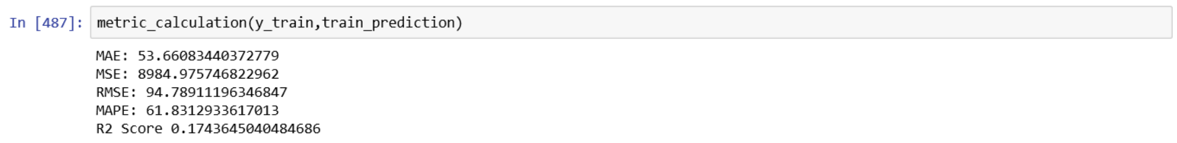
```
Scores for train with Boruta Features
The scores for the model :  MLP - Scaled
MAE     :      0.04628927278846125
RMSE    :      0.09402470146180283
R2      :      0.11350846149901017
MAPE    :      inf
```

Linear Regression:

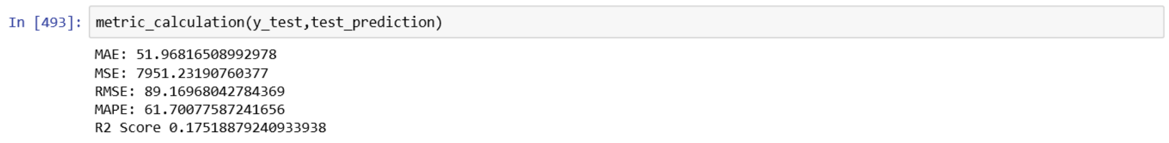


```
In [474]: X=dataset[['lights', 'T1', 'RH_1', 'T2', 'RH_2', 'T3',
                 'RH_3', 'T4', 'RH_4', 'T5', 'RH_5', 'T6', 'RH_6', 'T7', 'RH_7', 'T8',
                 'RH_8', 'T9', 'RH_9', 'T_out', 'Press_mm_hg', 'RH_out', 'Windspeed',
                 'Visibility', 'Tdewpoint', 'Month','Hour','Day','Weekendstatus','NSM',
                'Friday','Saturday','Sunday','Thursday','Tuesday','Wednesday']]

In [475]: y=dataset['Appliances']

In [476]: from sklearn.cross_validation import train_test_split
          from sklearn.linear_model import LinearRegression

In [477]: #train_test_split : Basically passing the x and y data and then specifying test size
          #Split arrays or matrices into random train and test subsets

In [478]: # In this case, the attribue random_state basically refers to the random split of the data between train set and test set.
          X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=2)
          lm=LinearRegression()
          lm.fit(X_train,y_train)
          print(lm.intercept_)
          print(lm.coef_)
          X_train.columns

          -2.8725637530753403
          [ 1.87163083e+00 -5.63536051e+00  1.54069958e+01 -1.81293023e+01
           -1.37455577e+01  2.52457081e+01  5.67791133e+00  5.99931309e-01
           -1.59492240e+00 -1.63222045e-01  3.29654692e-03  7.33789030e+00
           -7.65859523e-04 -3.22970473e-01 -1.12567166e+00  8.57809477e+00
           -4.21535310e+00 -1.02009934e+01 -4.13167482e-01 -7.82511412e+00
            7.57057328e-02 -1.66094927e-01  1.40657299e+00  1.40089497e-01
            1.99069073e+00 -9.68569290e+00 -4.35478717e+00 -4.92337803e-03
```

Output for training and testing dataset:



```
In [487]: metric_calculation(y_train,train_prediction)

          MAE: 53.66083440372779
          MSE: 8984.975746822962
          RMSE: 94.78911196346847
          MAPE: 61.8312933617013
          R2 Score 0.1743645040484686
```



```
In [493]: metric_calculation(y_test,test_prediction)

          MAE: 51.96816508992978
          MSE: 7951.23190760377
          RMSE: 89.16968042784369
          MAPE: 61.70077587241656
          R2 Score 0.17518879240933938
```

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

<u>Scaling:</u>
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]])
```

<u>Observation</u>:
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

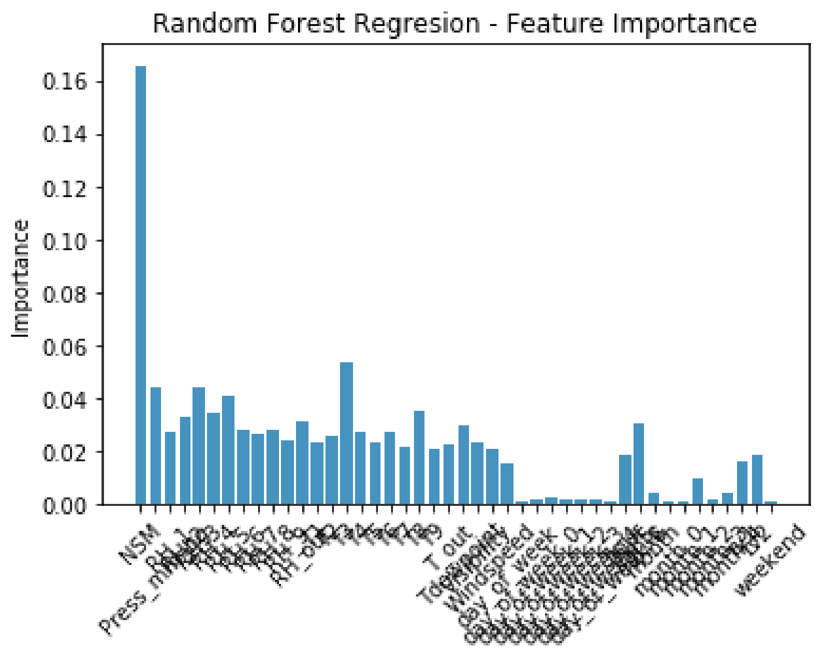
Regression parameters:

```
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
```



```
<matplotlib.figure.Figure at 0x1a1b69d4e0>
```

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

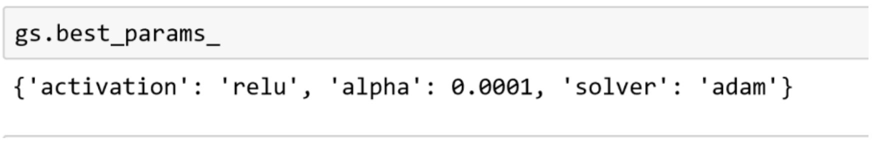
3. Fit the model to the training data
4. Use the model to predict labels for new data <br>
we need a way to validate that our model and our hyperparameters are a good fit to the data

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:

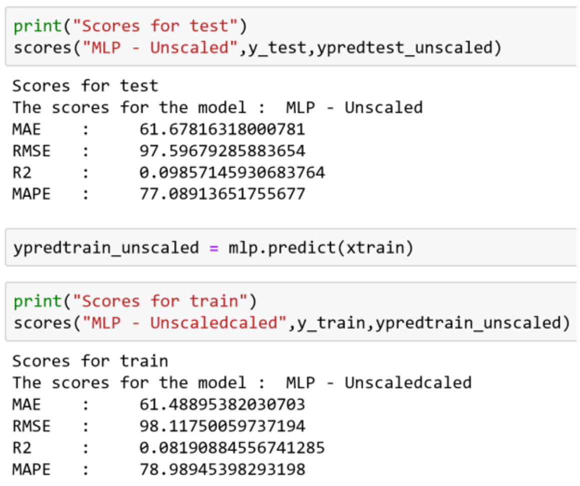
Best parameters with Unscaled Data



```
gs.best_params_
```

```
{'activation': 'relu', 'alpha': 0.0001, 'solver': 'adam'}
```

Scores :



```
print("Scores for test")
scores("MLP - Unscaled",y_test,ypredtest_unscaled)
```

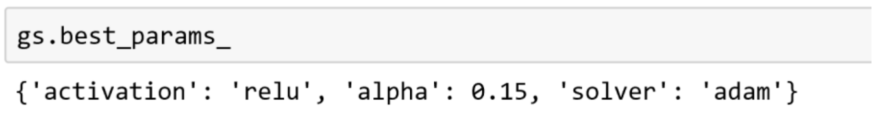
```
Scores for test
The scores for the model :  MLP - Unscaled
MAE    :      61.67816318000781
RMSE   :      97.59679285883654
R2     :      0.09857145930683764
MAPE   :      77.08913651755677
```

```
ypredtrain_unscaled = mlp.predict(xtrain)
```

```
print("Scores for train")
scores("MLP - Unscaledcaled",y_train,ypredtrain_unscaled)
```

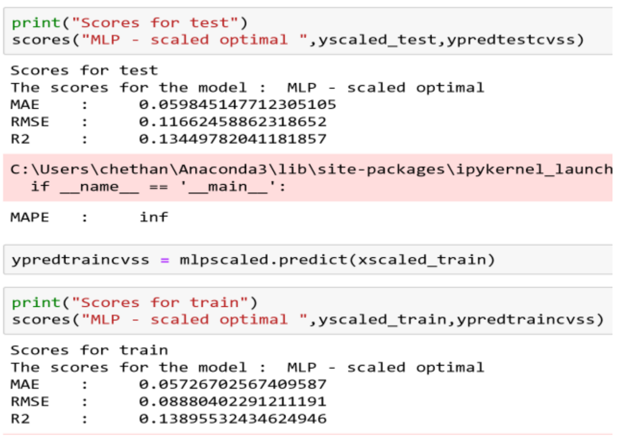
```
Scores for train
The scores for the model :  MLP - Unscaledcaled
MAE    :      61.48895382030703
RMSE   :      98.11750059737194
R2     :      0.08190884556741285
MAPE   :      78.98945398293198
```

Best parameters with Data



```
gs.best_params_
```

```
{'activation': 'relu', 'alpha': 0.15, 'solver': 'adam'}
```



```
print("Scores for test")
scores("MLP - scaled optimal ",yscaled_test,ypredtestcvss)
```

```
Scores for test
The scores for the model :  MLP - scaled optimal
MAE    :     0.059845147712305105
RMSE   :     0.11662458862318652
R2     :     0.13449782041181857
```

```
C:\Users\chethan\Anaconda3\lib\site-packages\ipykernel_launch
  if __name__ == '__main__':
```

```
MAPE   :     inf
```

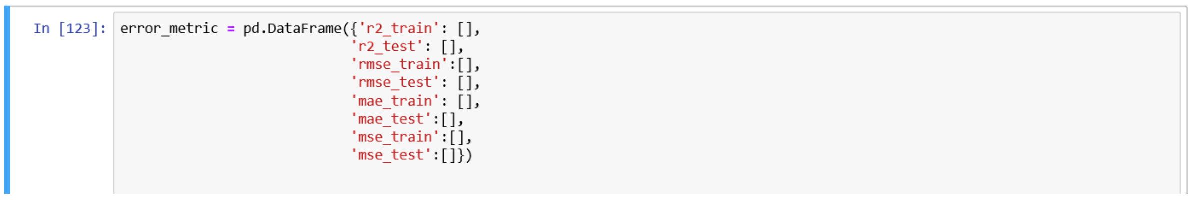
```
ypredtraincvss = mlpscaled.predict(xscaled_train)
```

```
print("Scores for train")
scores("MLP - scaled optimal ",yscaled_train,ypredtraincvss)
```

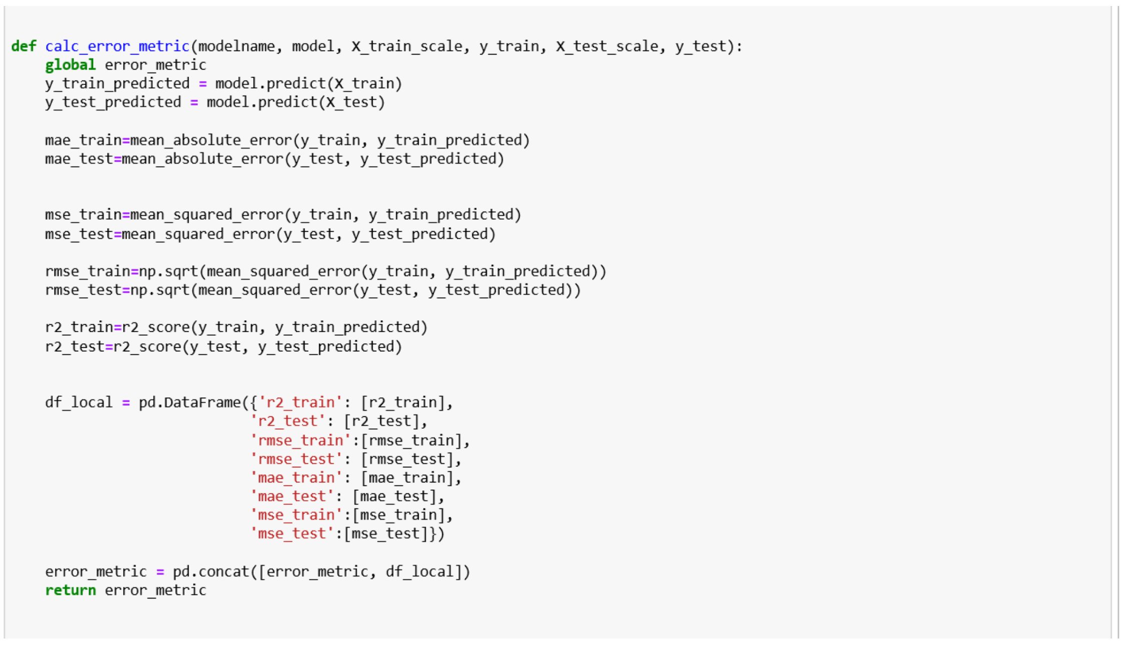
```
Scores for train
The scores for the model :  MLP - scaled optimal
MAE    :     0.05726702567409587
RMSE   :     0.08880402291211191
R2     :     0.13895532434624946
```

Section 5 : Final Pipeline

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



```
In [123]: error_metric = pd.DataFrame({'r2_train': [],
                                      'r2_test': [],
                                      'rmse_train':[],
                                      'rmse_test': [],
                                      'mae_train': [],
                                      'mae_test':[],
                                      'mse_train':[],
                                      'mse_test':[]})
```



```
def calc_error_metric(modelname, model, X_train_scale, y_train, X_test_scale, y_test):
    global error_metric
    y_train_predicted = model.predict(X_train)
    y_test_predicted = model.predict(X_test)

    mae_train=mean_absolute_error(y_train, y_train_predicted)
    mae_test=mean_absolute_error(y_test, y_test_predicted)


    mse_train=mean_squared_error(y_train, y_train_predicted)
    mse_test=mean_squared_error(y_test, y_test_predicted)

    rmse_train=np.sqrt(mean_squared_error(y_train, y_train_predicted))
    rmse_test=np.sqrt(mean_squared_error(y_test, y_test_predicted))

    r2_train=r2_score(y_train, y_train_predicted)
    r2_test=r2_score(y_test, y_test_predicted)


    df_local = pd.DataFrame({'r2_train': [r2_train],
                             'r2_test': [r2_test],
                             'rmse_train':[rmse_train],
                             'rmse_test': [rmse_test],
                             'mae_train': [mae_train],
                             'mae_test': [mae_test],
                             'mse_train':[mse_train],
                             'mse_test':[mse_test]})

    error_metric = pd.concat([error_metric, df_local])
    return error_metric
```

```
In [124]: pipe_lr = Pipeline([('scl', StandardScaler()),('clf', LinearRegression(normalize=True))])
          grid_params_lr =[{}]
          gs_lr = GridSearchCV(estimator=pipe_lr, param_grid=grid_params_lr, cv=10)
          gs_lr.fit(X_train, y_train)
          em=calc_error_metric('Regression', gs_lr, X_train, y_train, X_test, y_test)
          print('Regression completed')

          Regression completed
```

```
In [125]: em
```

Out[125]:

	r2_train	r2_test	rmse_train	rmse_test	mae_train	mae_test	mse_train	mse_test
0	0.174365	0.175189	94.789112	89.16968	53.660834	51.968165	8984.975747	7951.231908