INFO 7390 Advances in Data Sciences & Architecture

Machine Learning with Energy Dataset

Introduction

Part	Description
Part 1: Research Paper Review	Review 3 papers and provide a Jupyter Notebook for each paper.
Part 2: Exploratory Data Analysis	Conduct EDA using Python libraries (plotly, seaborn, matplotlib etc.). Provide a PowerPoint Report with graphs and key insights.
Part 3: Feature Engineering	Conduct thorough feature analysis and use pre-processing techniques to make the data usable.
Part 4: Prediction Algorithms	Try Linear Regression, Random Forest, Neural Networks to build prediction models in using sklearn in Python. Compute RMS, MAPE, R2 and MAE for Training and Testing Datasets. Recommend a model.
Part 5: Feature Selection	Understand the importance of the various features and how the features influence the output. Explore tpot, featuretools, Boruta, tsfresh.
Part 6: Model Validation and Selection	Understand hyperparameter tuning and model validation prior to model selection for production.
Part 7: Final Pipeline	Recommend a Final Model with reason. Automate the entire model from Data Ingestion to Final Model Prediction.

Project Tools:

- Language: Python
- Process: Review Research Paper, Exploratory Data Analysis, Feature Engineering, Prediction Algorithms, Feature Selection, Model Validation and Selection, Pipelining
- Tools used: Jupyter Notebook, Docker, Sublime text

Team:

Ankur Jain - 1206900

Amandeep Singh - 001271649

Eklavya Saxena - 001850025

GitHub Link:

https://github.com/eklavyasaxena/Advances-in-Data-Sciences-and-Architecture

1. Exploratory Data Analysis

Conducted an exploratory data analysis using Python packages (plotly, seaborn, matplotlib etc.) to understand the energy dataset.

1.a General Tips followed –

- Before plotting/joining/doing something, have a question or hypothesis that you want to investigate
- Draw a plot of what you want to see on paper to sketch the idea
- Write it down, then make the plan on how to get there
- How do you know you aren't fooling yourself?
- What else can I check if this is actually true?
- What evidence could there be that it's wrong?

1.b Libraries Used –

Plotting	Wrangling	System Packages	EDA Tools
Matplotlib	Numpy	Os	Missingno
Seaborn	Pandas	Sys	Pandas_profiling
folium		Warnings	Pivottablejs

1.c Description of the Data columns –

Where indicated, data from the nearest airport weather station (Chièvres Airport, Belgium) was downloaded from a public data set from Reliable Prognosis, rp5.ru. Permission was obtained from Reliable Prognosis for the distribution of the **4 months of data**.

Variables, Description	Units	Number of Features
date, Date time stamp	year-month-day hour:min:s	–
Appliances, Appliances energy consumption	Wh	1
Lights, Light energy consumption	Wh	2
T1, Temperature in kitchen area	°C	3
RH_1, Humidity in kitchen area	%	4
T2, Temperature in living room area	°C	5
RH_2, Humidity in living room area	%	6
T3, Temperature in laundry room area	°C	7
RH_3, Humidity in laundry room area	%	8
T4, Temperature in office room	°C	9
RH_4, Humidity in office room	%	10
T5, Temperature in bathroom	°C	11
RH_5, Humidity in bathroom	%	12
T6, Temperature outside the building (north side)	°C	13
RH_6, Humidity outside the building (north side)	%	14
T7, Temperature in ironing room	°C	15
RH_7, Humidity in ironing room	%	16
T8, Temperature in teenager room 2	°C	17
RH_8, Humidity in teenager room 2	%	18
T9, Temperature in parents room	°C	19
RH_9, Humidity in parents room	%	20
T_out, Temperature outside (from Chièvres weather station)	°C	21
Press_mm_hg, Pressure (from Chièvres weather station)	mm Hg	22
RH_out, Humidity outside (from Chièvres weather station)	%	23
Windspeed, Windspeed (from Chièvres weather station)	m/s	24
Visibility, Visibility (from Chièvres weather station)	km	25
Tdewpoint, Dew point temp (from Chièvres weather station)	°C	26
rv1, Random Variable 1	Non dimensional	27
rv2, Random Variable 2	Non dimensional	28

1.d Missing Data Handling –

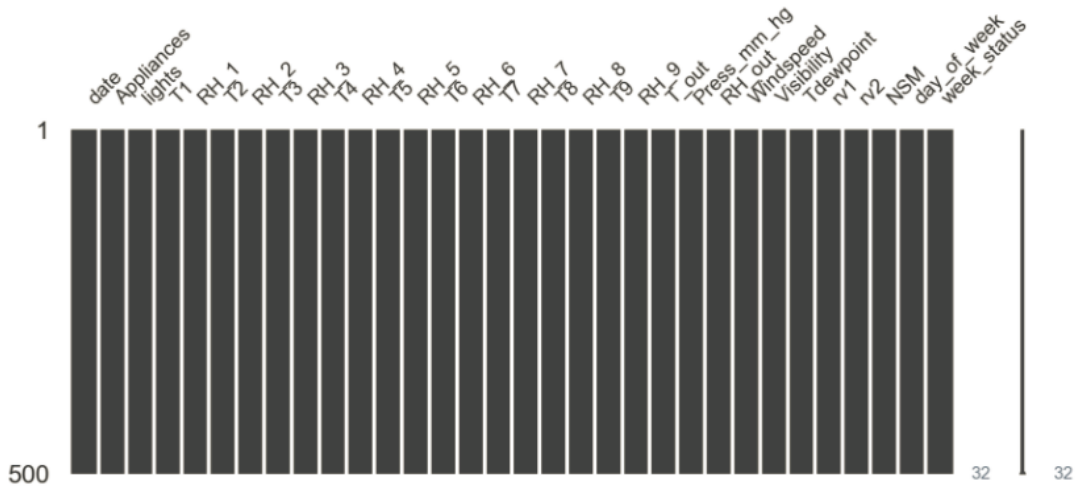
In this dataset, there is no missing data. The dataset is very clean.

```
In [347]: # Checking Null Values
          df.apply(lambda x: sum(x.isnull()), axis=0)

Out[347]: date           0
          Appliances     0
          lights         0
          T1             0
          RH_1           0
          T2             0
          RH_2           0
          T3             0
          RH_3           0
          T4             0
          RH_4           0
          T5             0
          RH_5           0
          T6             0
          RH_6           0
          T7             0
          RH_7           0
          T8             0
          RH_8           0
          T9             0
          RH_9           0
          T_out          0
          Press_mm_hg    0
          RH_out         0
          Windspeed      0
          Visibility     0
          Tdewpoint      0
          rv1            0
          rv2            0
          NSM            0
          day_of_week    0
          week_status    0
          dtype: int64
```

1.e Visualizing the missing-ness of data -

```
In [349]: # Matrix of missingness of the data
          msno.matrix(df.sample(500), figsize=(14, 5), width_ratios=(14, 1));
```



From the above matrix, it is clear that there is no data missing.

1.f Data Visualization –

1. **Univariate Distributions** – for numerical data, we looked at location, spread and shape of the columns

- *Location*: mean, median, mode, interquartile mean
- *Spread*: standard deviation, variance, range, interquartile range
- *Shape*: skewness, kurtosis

2. **Bivariate Distributions –**

   a. Categorical Vs Categorical
      - Heat Map
      - Multiple Bar Plots
   b. Categorical Vs Continuous
      - Box Plots
      - Violin Plots
      - Overlaid Histograms (if 3 or less categories)
   c. Continuous Vs Continuous
      - Scatter Plots
      - Hexbin Plots

- Joint Kernel Density Estimation Plots
- Correlation Matrix Heatmap

1.g Detecting Outliers –

Proceed with the following steps to detect outliers in the dataset:

1. Arrange all the dataset points and calculate median
2. Calculate the upper quartile
3. Calculate the lower quartile
4. Calculate the interquartile range

   Product of numeric value of 1.5 and difference of the upper quartile (75%) and lower quartile (25%)
   - 1.5 X (Upper Quartile - Lower Quartile)
5. Calculate the inner fences for the dataset

   Set of numerical boundaries which is classified as major and minor outlier:
   - Major Outlier = Upper Quartile + Interquartile Range
   - Minor Outlier = Lower Quartile - Interquartile Range

```
def remove_outlier(df, variable):
    major_o = df_describe.loc[variable,'Major Outlier']
    minor_o = df_describe.loc[variable,'Minor Outlier']
    df = df.drop(df[(df[variable]>major_o) | (df[variable]<minor_o)].index)
    return df
```

```
outlier_column_list = [x for x in all_columns
                       if x not in ('date', 'Appliances', 'lights', 'RH_6', 'RH_out', 'Windspeed', 'Visibi
lity', 'rv1', 'rv2')]
```

```
for column_name in outlier_column_list:
    df = remove_outlier(df, column_name)
```

```
# Percentage of Data Dropped
((df_loaded.shape[0] - df.shape[0])/df_loaded.shape[0])*100
```

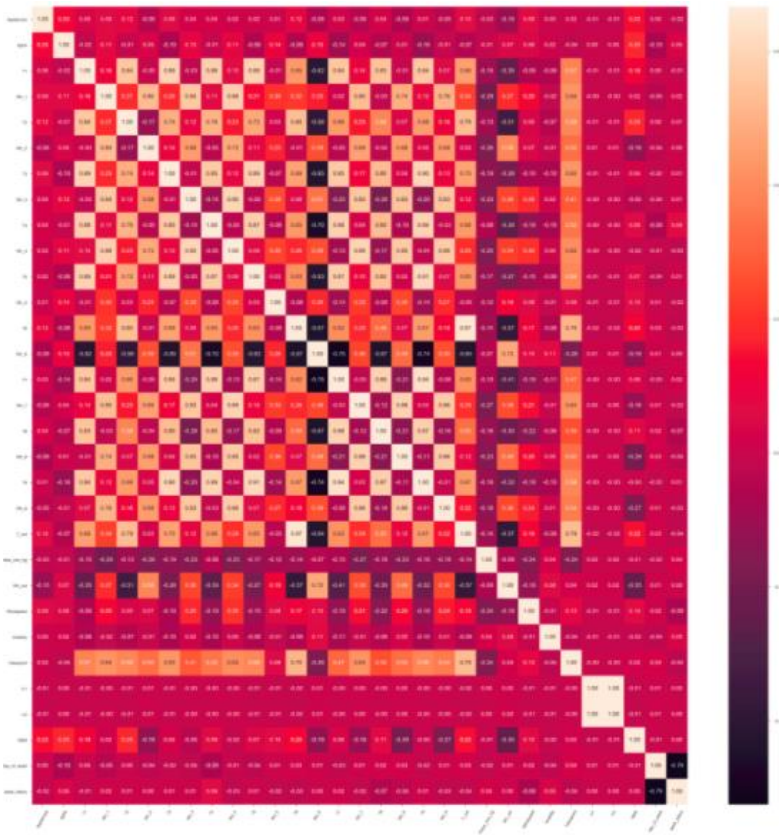
```
14.83151760831011
```

From the above formulae, we have dropped around 15% data which is acceptable.

1.h Pair Wise Relationship –

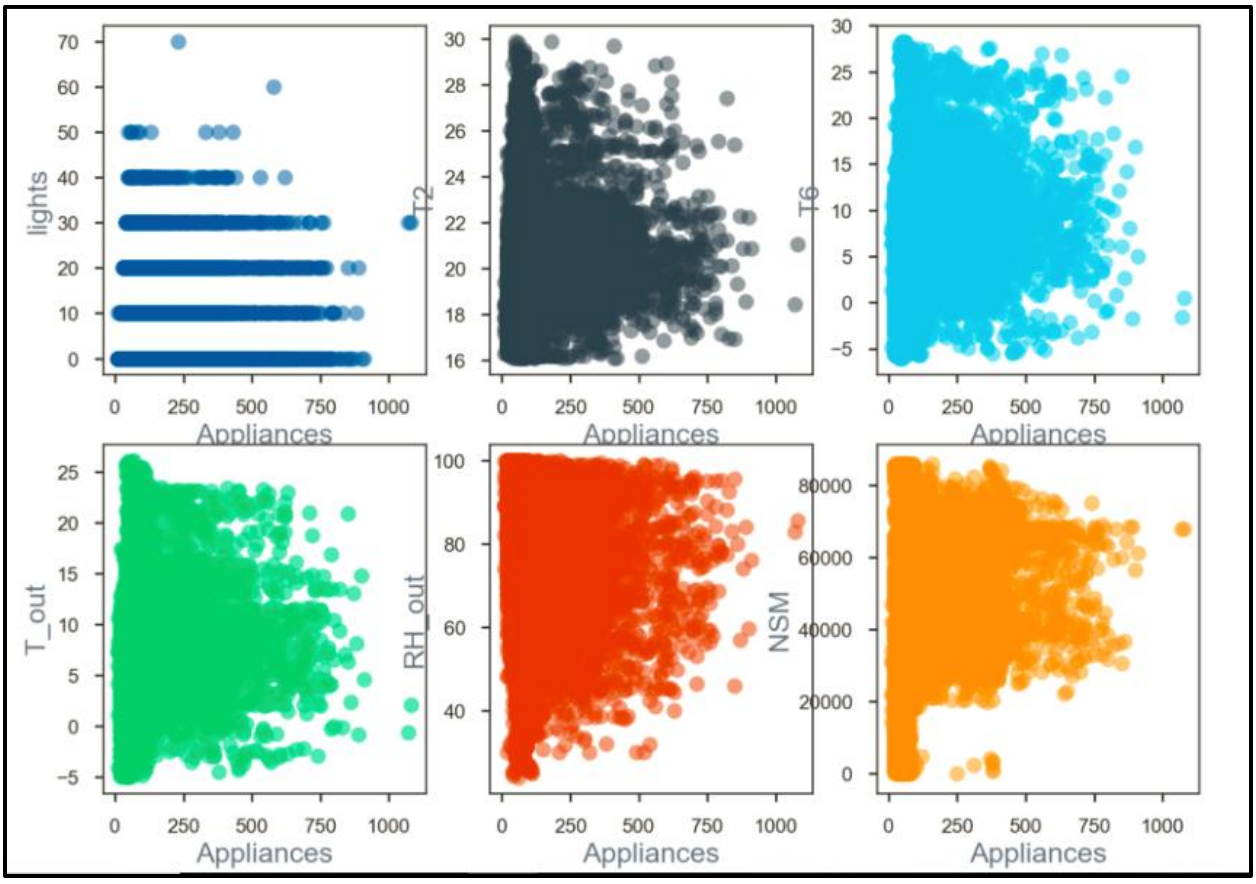
1. **Correlation Heat Map** – From the correlation map, it can be deduced that only following attributes are closely related (as compared to others) to Appliance's Energy Consumption:

Variables, Description	Correlation
Lights, Light energy consumption	0.20
T2, Temperature in living room area	0.12
T6, Temperature outside the building (north side)	0.12
T_out, Temperature outside (from Chièvres weather station)	0.10
RH_out, Humidity outside (from Chièvres weather station)	-0.15
NSM, Number of seconds from midnight	0.22



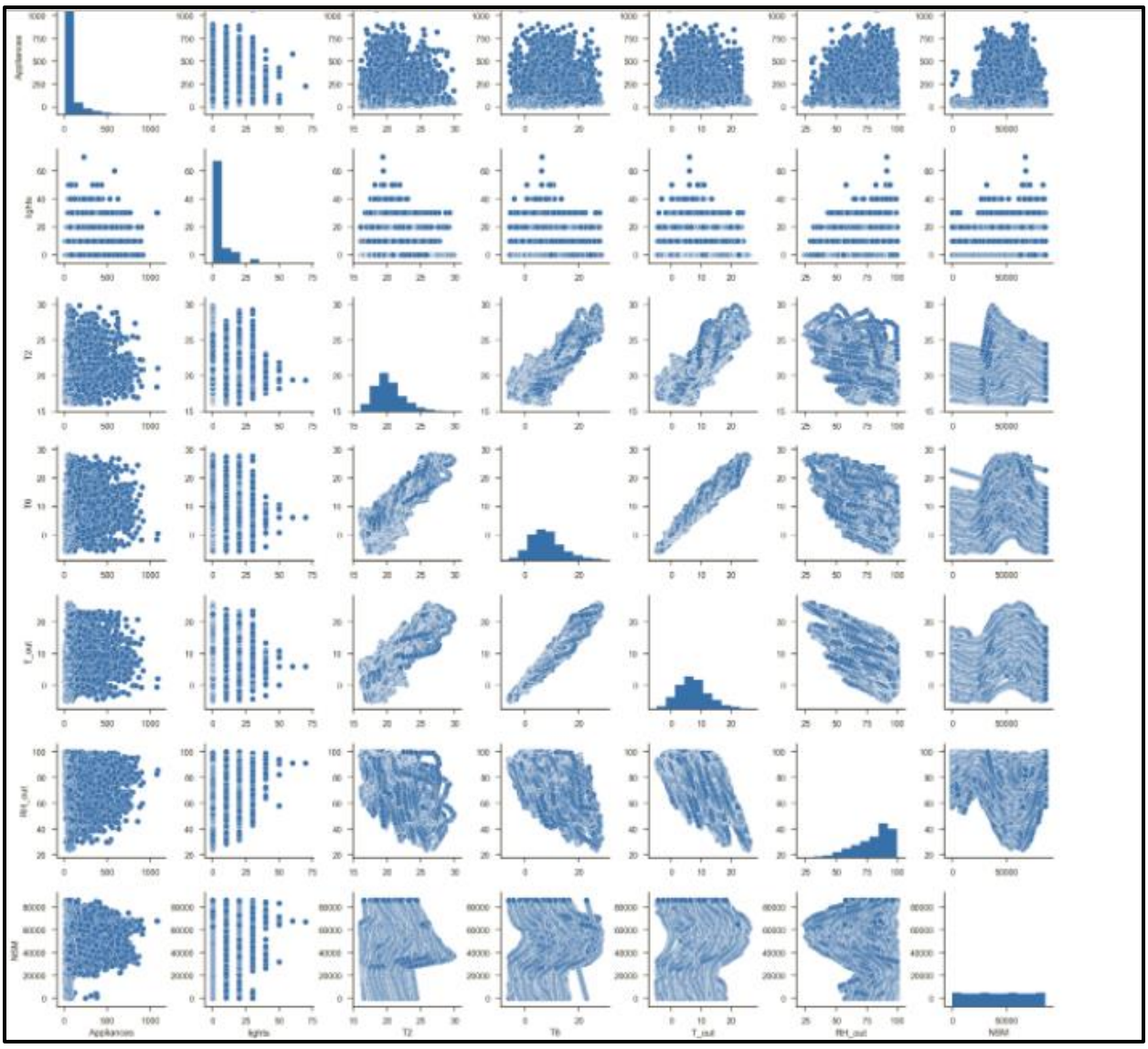
2. Scatter Plots –

```
fig, ((ax1, ax2, ax3), (ax4, ax5, ax6)) = plt.subplots(figsize=(14,10), nrows=2, ncols=3)
sns.regplot(x='Appliances', y='lights', data=df, fit_reg=False, scatter_kws={'alpha': 0.5}, ax = ax1)
sns.regplot(x='Appliances', y='T2', data=df, fit_reg=False, scatter_kws={'alpha': 0.5}, ax = ax2)
sns.regplot(x='Appliances', y='T6', data=df, fit_reg=False, scatter_kws={'alpha': 0.5}, ax = ax3)
sns.regplot(x='Appliances', y='T_out', data=df, fit_reg=False, scatter_kws={'alpha': 0.5}, ax = ax4)
sns.regplot(x='Appliances', y='RH_out', data=df, fit_reg=False, scatter_kws={'alpha': 0.5}, ax = ax5)
sns.regplot(x='Appliances', y='NSM', data=df, fit_reg=False, scatter_kws={'alpha': 0.5}, ax = ax6);
```



3. Pair Plots -

```
sns.set(style="ticks", color_codes=True);
sns.pairplot(df[['Appliances', 'lights', 'T2', 'T6', 'T_out', 'RH_out', 'NSM']]);
```



There are very few obvious outliers in these relationships. A quick check through the correlations:

- **Are there zeros**?
  - Yes, zero is a possible observation for all the attributes except 'lights' which 77% of its data as zeros
- **Are there strange correlations**?

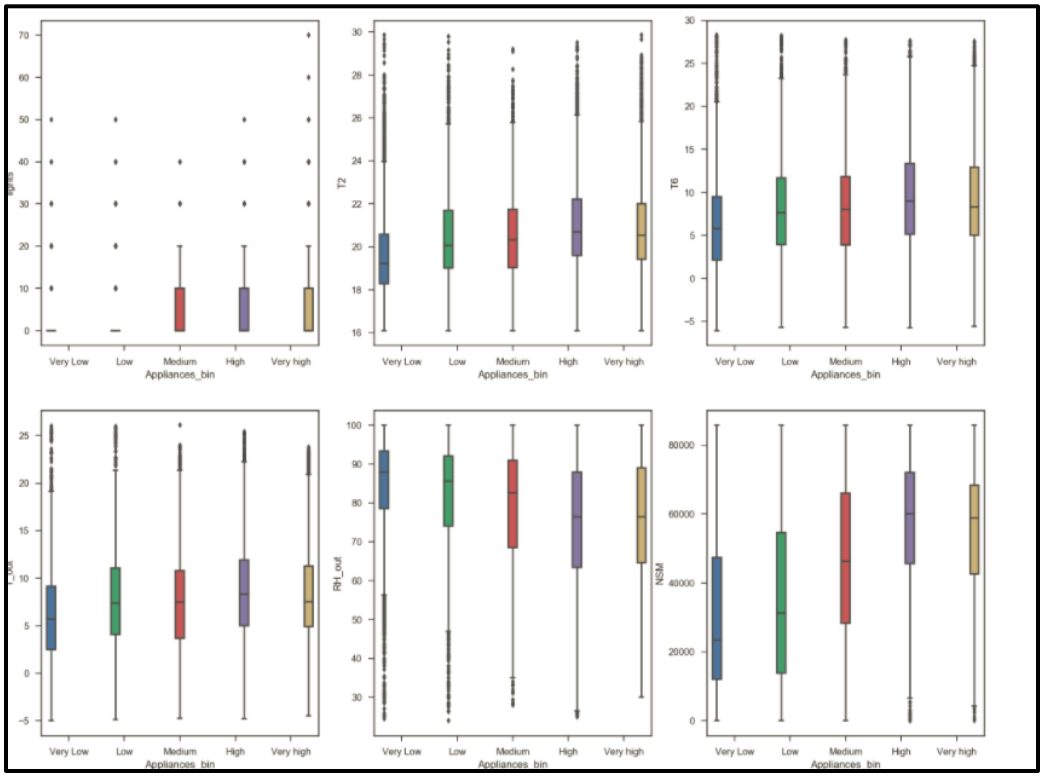
- Yes, the correlation is too scattered to figure out any pattern
- NSM shows that appliances are usually used from morning 5:30am (20,000 sec past midnight) everyday
- RH_out shows a rough interpretation that consumption increases with respect to the increase in humidity

- **Are there separate clusters (possible data recording error or anomaly)?**
  - No, but there are many outliers

There are many non-linear relationships. One way to address this is to bin variables into categories and look at the distribution of other variables for each category.

4. Box Plots –

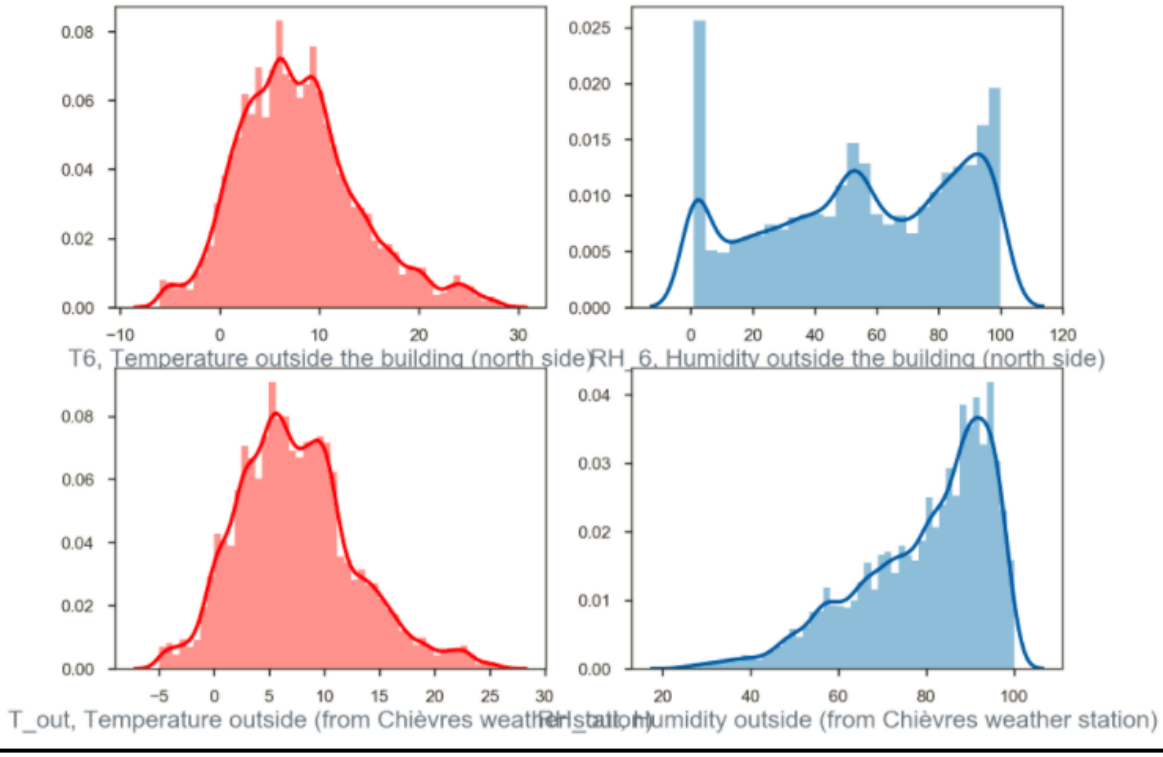
```
fig, ((ax1, ax2, ax3), (ax4, ax5, ax6)) = plt.subplots(figsize=(18,14), nrows=2, ncols=3)
sns.boxplot(x='Appliances_bin', y='lights', hue='Appliances_bin', data=df, ax = ax1)
sns.boxplot(x='Appliances_bin', y='T2', hue='Appliances_bin', data=df, ax = ax2)
sns.boxplot(x='Appliances_bin', y='T6', hue='Appliances_bin', data=df, ax = ax3)
sns.boxplot(x='Appliances_bin', y='T_out', hue='Appliances_bin', data=df, ax = ax4)
sns.boxplot(x='Appliances_bin', y='RH_out', hue='Appliances_bin', data=df, ax = ax5)
sns.boxplot(x='Appliances_bin', y='NSM', hue='Appliances_bin', data=df, ax = ax6)
ax1.legend().set_visible(False)
ax2.legend().set_visible(False)
ax3.legend().set_visible(False)
ax4.legend().set_visible(False)
ax5.legend().set_visible(False)
ax6.legend().set_visible(False);
```



The above Box plot shows that there are outliers present in the dataset.

5. Outside Temperature & Humidity contributing to aggregate Energy Consumption -

```
fig, ((ax1, ax3), (ax2, ax4)) = plt.subplots(figsize=(14,10), nrows=2, ncols=2)
sns.distplot(df.T6, color="red", ax=ax1, axlabel="T6, Temperature outside the building (north side)")
sns.distplot(df.T_out, color="red", ax=ax2, axlabel="T_out, Temperature outside (from Chièvres weather sta
tion)")
sns.distplot(df.RH_6, ax=ax3, axlabel="RH_6, Humidity outside the building (north side)")
sns.distplot(df.RH_out, ax=ax4, axlabel="RH_out, Humidity outside (from Chièvres weather station)");
```



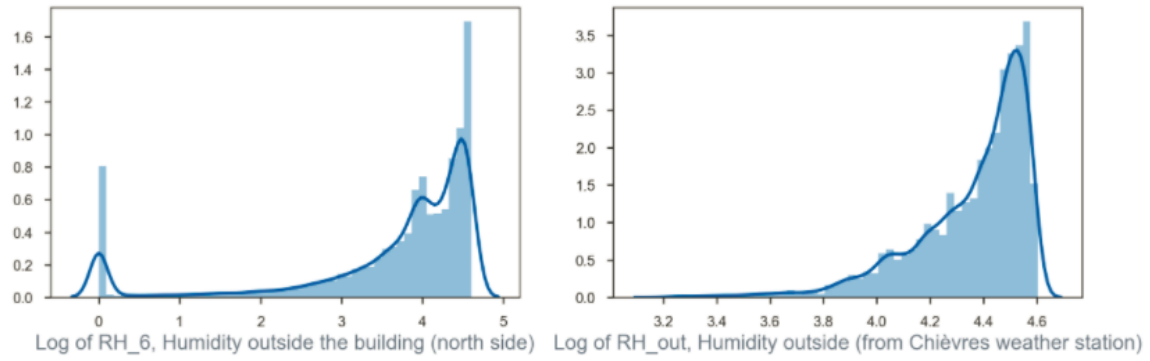
It can be roughly concluded:

- While the temperatures are related, **humidity vary significantly** between *Building Outside* and *Chièvres Weather Station*
- Temperature peaks at 5°C and are normally distributed

6. Log Transform –

Taking a log transform will make a variable more normal.

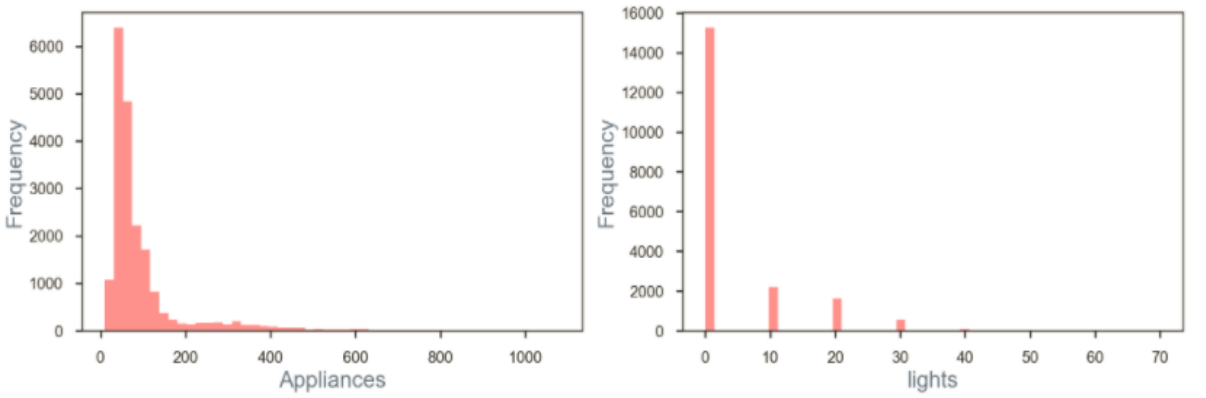
```
fig, ((ax3, ax4)) = plt.subplots(figsize=(18,5), ncols=2)
sns.distplot(df.RH_6.apply(np.log), ax=ax3, axlabel="Log of RH_6, Humidity outside the building (north sid
e)")
sns.distplot(df.RH_out.apply(np.log), ax=ax4, axlabel="Log of RH_out, Humidity outside (from Chièvres weat
her station)");
```



7. Skewness and Kurtosis –

- **Skewness**: measure of lack of symmetry.
- **Kurtosis**: measure of whether the data are heavily tailed relative to the normal distribution.

```
fig, (ax1, ax2) = plt.subplots(figsize=(18,5), ncols=2)
sns.distplot(df.Appliances, kde=False, color="red", ax=ax1)
sns.distplot(df.lights, kde=False, color="red", ax=ax2)
ax1.set_ylabel("Frequency")
ax2.set_ylabel("Frequency");
```



- Skewness for normal distribution should be zero. Negative skewness indicates skew left and positive skewness indicates skew right.
- Kurtosis is also zero for a normal distribution and can only be positive.

Part-3 Feature Engineering –

In the dataset:

- There are columns with a scope of transformation, like WeekStatus and Days_of_week
- There is no null data in the dataset
- Also, date column is not required because NSM column already got interpreted from this

3.a Transformation of WeekStatus and Days_of_week columns

```
week_status = pd.get_dummies(df['week_status'], prefix = 'week_status')
day_of_week = pd.get_dummies(df['day_of_week'], prefix = 'day_of_week')
#['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
#['Weekend', 'Weekday']

# Concat above dummies variable dataframe to the main dataframe
df = pd.concat((df,week_status),axis=1)
df = pd.concat((df,day_of_week),axis=1)

# Drop the WeekStatus and Day_of_week column
df = df.drop(['week_status','day_of_week'],axis=1)
```

```
df = df.rename(columns={'week_status_0': 'Weekend', 'week_status_1': 'Weekday',
                        'day_of_week_0': 'Monday', 'day_of_week_1': 'Tuesday', 'day_of_week_2': 'Wednesday',
                       'day_of_week_3': 'Thursday', 'day_of_week_4': 'Friday', 'day_of_week_5': 'Saturday',
                       'day_of_week_6': 'Sunday'})
```

3.b Pandas Profiling –

After Profiling following can be deduced:

- Dropping the highly correlated attributes:
  - T9, Temperature in parents room and T7, Temperature in ironing room (ρ = 0.94478)
    - T9 is matter of importance, hence considering to drop T7
  - T_out, Temperature outside (from Chièvres weather station) and T6, Temperature outside the building (north side) (ρ = 0.97479)
    - As RH_out is more in correlation with 'Appliances' as compared to RH_6, considering to drop T6
  - rv2, Random Variable 2 and rv1, Random Variable 1 (ρ = 1)
    - As both of these attributes are equal, considering to drop rv1
  - date column and NSM column provides similar information.
    - Therefore considering to drop date

3.c Redefining the Appliances column -

As the task is to understand the energy consumption, adding the consumption of lights to the appliances and then exported the data to the csv file.

```
df['Appliances'] = df['Appliances'] + df['lights']
df = df.drop(['lights'],axis=1)
```

```
# Exporting the transformed dataset
df.to_csv('energydata_complete_transformed.csv', index=False)
```

Part 4- Prediction Algorithms –

4.a Dividing dataset into Training and Testing data –

```
X = df.drop(['Appliances'],axis=1)
y = df['Appliances']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
```

4.b Function to print metrices –

```
# Function to print metrics
def print_training_metrics(**kwargs):
    for name, value in kwargs.items():
        value = float("{0:.2f}".format(value))
        print('{0} for Training Dataset is: {1}'.format(name, value))
def print_testing_metrics(**kwargs):
    for name, value in kwargs.items():
        value = float("{0:.2f}".format(value))
        print('{0} for Testing Dataset is: {1}'.format(name, value))

# Function to print and add metrics to dataframe
def print_metrics(df, model, r2_train, rms_train, mae_train, mape_train, r2_test, rms_test, mae_test, mape
_test):
    #print('R Squared for Training Data:',float("{0:.2f}".format(r2_train)))
    #print('R Squared for Testing Data:',float("{0:.2f}".format(r2_test)))
    #print('RMS for Training Data:',float("{0:.2f}".format(rms_train)))
    #print('RMS for Testing Data:',float("{0:.2f}".format(rms_test)))
    #print('MAE for Training Data:',float("{0:.2f}".format(mae_train)))
    #print('MAE for Testing Data:',float("{0:.2f}".format(mae_test)))
    #print('MAPE for Training Data:',float("{0:.2f}".format(mape_train)))
    #print('MAPE for Testing Data:',float("{0:.2f}".format(mape_test)))
    df[model] = [float("{0:.2f}".format(r2_train)), float("{0:.2f}".format(r2_test)),
                 float("{0:.2f}".format(rms_train)), float("{0:.2f}".format(rms_test)),
                 float("{0:.2f}".format(mae_train)), float("{0:.2f}".format(mae_test)),
                 float("{0:.2f}".format(mape_train)), float("{0:.2f}".format(mape_test))]
    return df
```

The above function will print the following metrices –

- Root Mean Square Error
- R Squared Error
- Mean Absolute Error
- Mean Absolute Percentage Error

4.c Linear Regression Model

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from math import sqrt
from sklearn.metrics import mean_absolute_error

lr = LinearRegression()
lr.fit(X_train,y_train)

# Predicting and Calculating the Metrices for Prediction of Testing Dataset
prediction_test_lr = lr.predict(X_test)

r2_test_lr = r2_score(y_test, prediction_test_lr)
rms_test_lr = sqrt(mean_squared_error(y_test, prediction_test_lr))
mae_test_lr = mean_absolute_error(y_test,prediction_test_lr)
mape_test_lr = np.mean(np.abs((y_test - prediction_test_lr) / y_test)) * 100

# Predicting and Calculating the Metrices for Prediction of Training Dataset
prediction_train_lr = lr.predict(X_train)

r2_train_lr = r2_score(y_train, prediction_train_lr)
rms_train_lr = sqrt(mean_squared_error(y_train, prediction_train_lr))
mae_train_lr = mean_absolute_error(y_train,prediction_train_lr)
mape_train_lr = np.mean(np.abs((y_train - prediction_train_lr) / y_train)) * 100

# Printing the training and testing metrices
metrics_df = print_metrics(metrics_df, 'LR_Model', r2_train_lr, rms_train_lr, mae_train_lr, mape_train_lr,
 r2_test_lr, rms_test_lr, mae_test_lr, mape_test_lr)
metrics_df
```

The output of above Linear Regression model –

	LR_Model
RSquared_train	0.17
RSquared_test	0.17
RMS_train	93.21
RMS_test	91.97
MAE_train	52.61
MAE_test	52.42
MAPE_train	61.90
MAPE_test	62.34

4.d Random Forest Model -

```
from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor(n_estimators = 500)
rf.fit(X_train, y_train)

# Predicting and Calculating the Metrices for Prediction of Testing Dataset
prediction_test_rf = rf.predict(X_test)
r2_test_rf = r2_score(y_test, prediction_test_rf)
rms_test_rf = sqrt(mean_squared_error(y_test, prediction_test_rf))
mae_test_rf = mean_absolute_error(y_test,prediction_test_rf)
mape_test_rf = np.mean(np.abs((y_test - prediction_test_rf) / y_test)) * 100

# Predicting and Calculating the Metrices for Prediction of Training Dataset
prediction_train_rf = rf.predict(X_train)
r2_train_rf = r2_score(y_train, prediction_train_rf)
rms_train_rf = sqrt(mean_squared_error(y_train, prediction_train_rf))
mae_train_rf = mean_absolute_error(y_train,prediction_train_rf)
mape_train_rf = np.mean(np.abs((y_train - prediction_train_rf) / y_train)) * 100

# Printing the training and testing metrices
print('Random Forest Model\n')
metrics_df = print_metrics(metrics_df, 'RF_Model', r2_train_rf, rms_train_rf, mae_train_rf, mape_train_rf,
 r2_test_rf, rms_test_rf, mae_test_rf, mape_test_rf)
metrics_df
```

The output of above Random Forest Model –

	LR_Model	RF_Model
RSquared_train	0.17	0.95
RSquared_test	0.17	0.60
RMS_train	93.21	23.41
RMS_test	91.97	63.34
MAE_train	52.61	10.94
MAE_test	52.42	29.51
MAPE_train	61.90	10.85
MAPE_test	62.34	29.37

4.e Neural Network Model

```
# Import Multi-Layer Perceptron Classifier Model
from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(hidden_layer_sizes=(37,37,37))
mlp.fit(X_train,y_train)

# Predicting and Calculating the Metrices for Prediction of Testing Dataset
prediction_test_nn = mlp.predict(X_test)
r2_test_nn = r2_score(y_test, prediction_test_nn)
rms_test_nn = sqrt(mean_squared_error(y_test, prediction_test_nn))
mae_test_nn = mean_absolute_error(y_test,prediction_test_nn)
mape_test_nn = np.mean(np.abs((y_test - prediction_test_nn) / y_test)) * 100

# Predicting and Calculating the Metrices for Prediction of Training Dataset
prediction_train_nn = mlp.predict(X_train)
r2_train_nn = r2_score(y_train, prediction_train_nn)
rms_train_nn = sqrt(mean_squared_error(y_train, prediction_train_nn))
mae_train_nn = mean_absolute_error(y_train,prediction_train_nn)
mape_train_nn = np.mean(np.abs((y_train - prediction_train_nn) / y_train)) * 100

# Printing the training and testing metrices
print('Neural Network Model')
metrics_df = print_metrics(metrics_df, 'NN_Model', r2_train_nn, rms_train_nn, mae_train_nn, mape_train_nn,
 r2_test_nn, rms_test_nn, mae_test_nn, mape_test_nn)
metrics_df
```

The output of above Neural Network Model –

	LR_Model	RF_Model	NN_Model
RSquared_train	0.17	0.95	-0.21
RSquared_test	0.17	0.60	-0.22
RMS_train	93.21	23.41	112.84
RMS_test	91.97	63.34	111.20
MAE_train	52.61	10.94	51.98
MAE_test	52.42	29.51	51.89
MAPE_train	61.90	10.85	35.19
MAPE_test	62.34	29.37	35.46

The above Neural Network Model fits worse.

- R2 compares the fit of the chosen model with that of a horizontal straight line (the null hypothesis). If the chosen model fits worse than a horizontal line, then R2 is negative. Therefore, Neural Network is not a suitable predictive model.

Part 5 Feature Selection –

5.a Recursive Feature Elimination- The Recursive Feature Elimination (RFE) method is a feature selection approach. It works by recursively removing attributes and building a model on those attributes that remain. It uses the model accuracy to identify which attributes (and combination of attributes) contribute the most to predicting the target attribute.

```
# Recursive Feature Elimination
from sklearn.feature_selection import RFE
from sklearn.linear_model import (LinearRegression, Ridge, Lasso)
from sklearn.ensemble import RandomForestRegressor

# Create a base classifier used to evaluate a subset of attributes
model_lr = LinearRegression()
model_lasso = Lasso()
model_rf = RandomForestRegressor(n_estimators=50)

# Create the RFE model and select 3 attributes
rfe_lr = RFE(model_lr, n_features_to_select = 3)
rfe_lr = rfe_lr.fit(X, y)

rfe_lasso = RFE(model_lasso, n_features_to_select = 3)
rfe_lasso = rfe_lasso.fit(X, y)

rfe_rf = RFE(model_rf, n_features_to_select = 3)
rfe_rf = rfe_rf.fit(X, y)

# Summarize the selection of the attributes
feature_df['RFE_LR_Rank'] = rfe_lr.ranking_
feature_df['RFE_Lasso_Rank'] = rfe_lasso.ranking_
feature_df['RFE_RF_Rank'] = rfe_rf.ranking_

feature_df
```

	RFE_LR_Rank	RFE_Lasso_Rank	RFE_RF_Rank
T1	19	31	20
RH_1	1	1	11
T2	2	1	14
RH_2	1	1	16
T3	3	2	6
RH_3	13	8	2
T4	11	14	5
RH_4	21	24	3
T5	12	11	18
RH_5	25	15	9
RH_6	29	17	4
RH_7	22	13	10
T8	5	4	8
RH_8	10	5	15

The above output shows the ranking of each and every predictor with different algorithms using RFE

5.b Feature Importance - Methods that use ensembles of decision trees (like Random Forest or Extra Trees) can also compute the relative importance of each attribute. These importance values can be used to inform a feature selection process.

```
# Feature Importance
from sklearn import metrics
from sklearn.ensemble import ExtraTreesClassifier

# Fit an Extra Trees model to the data
model = ExtraTreesClassifier()
model.fit(X, y)

# Display the relative importance of each attribute
feature_df['Feature_Imp_ETC'] = model.feature_importances_
feature_df
```

The above code will calculate the feature importance values for each predictor

	RFE_LR_Rank	RFE_Lasso_Rank	RFE_RF_Rank	Feature_Imp_ETC
T1	19	31	20	0.036187
RH_1	1	1	11	0.041077
T2	2	1	14	0.041253
RH_2	1	1	16	0.042027
T3	3	2	6	0.036308
RH_3	13	8	2	0.039195
T4	11	14	5	0.035974
RH_4	21	24	3	0.040497
T5	12	11	18	0.033217
RH_5	25	15	9	0.040028
RH_6	29	17	4	0.042822
RH_7	22	13	10	0.041788
T8	5	4	8	0.037749
RH_8	10	5	15	0.043768
T9	4	3	12	0.027852
RH_9	24	12	17	0.043937
T_out	20	30	1	0.043390
Press_mm_hg	26	19	1	0.040907
RH_out	28	21	13	0.043055
Windspeed	16	10	21	0.034927
Visibility	27	20	22	0.031805
Tdewpoint	18	23	7	0.044073
rv2	30	25	19	0.044856

5.c Exhaustive Search -

The below numbers are Exhaustive Search Values using pValue

```
df_exhaust_X_train = X_train[selected_features]
df_exhaust_X_test = X_test[selected_features]
calculate_rms_randomforest(df_exhaust_X_train, df_exhaust_X_test, y_train, y_test, calculate_all=True)
```

```
r2_train_rf:  0.92
r2_test_rf:  0.58
rms_train_rf:  28.94
rms_test_rf:  63.21
mae_train_rf:  12.57
mae_test_rf:  29.69
mape_train_rf:  12.32
mape_test_rf:  29.46
```

5.d – Forward Search Value –

The below values are Forward Search Values using pValue

```
r2_train_rf:  0.92
r2_test_rf:  0.57
rms_train_rf:  29.15
rms_test_rf:  63.88
mae_train_rf:  12.62
mae_test_rf:  30.15
mape_train_rf:  12.31
mape_test_rf:  29.85
```

5.e Backward Search Value –

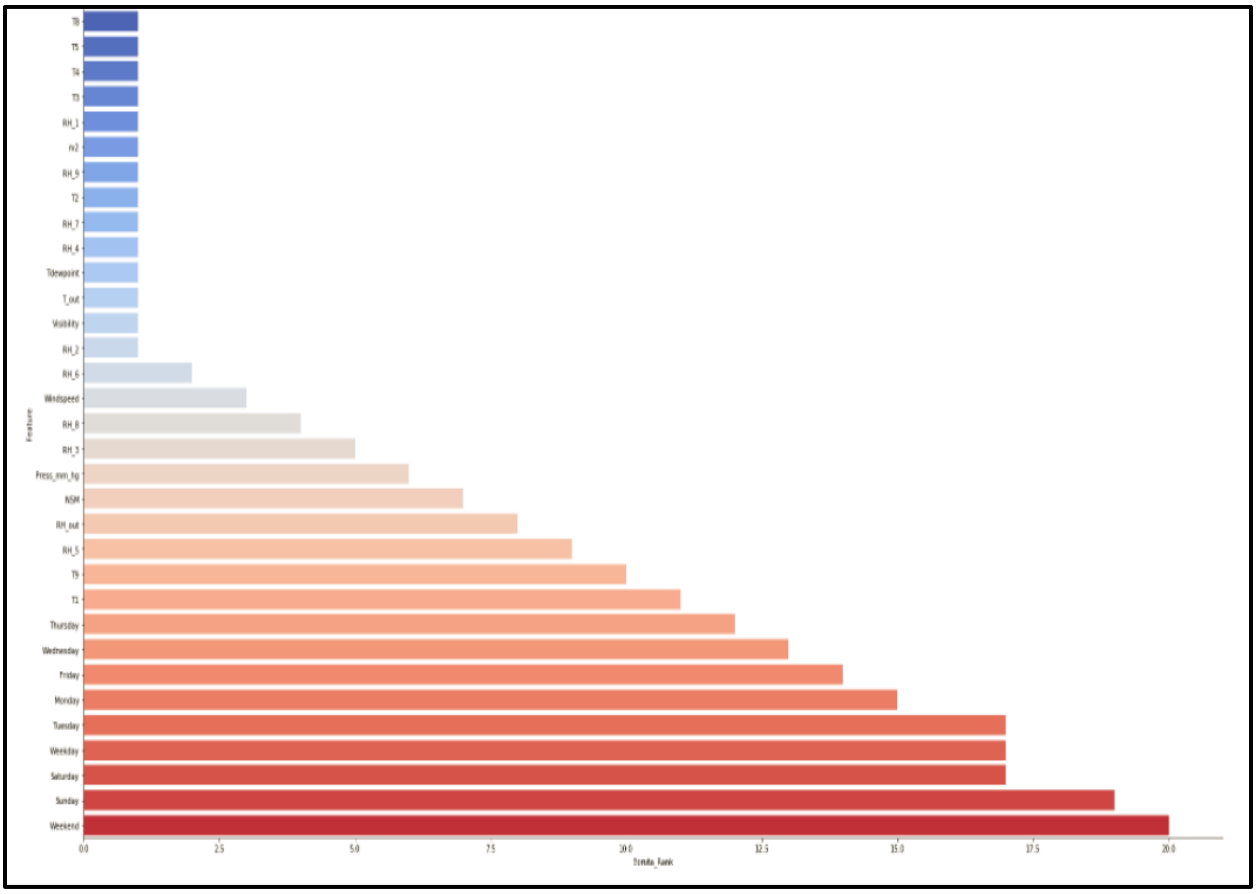
```
r2_train_rf:  0.92
r2_test_rf:  0.6
rms_train_rf:  28.44
rms_test_rf:  61.9
mae_train_rf:  12.43
mae_test_rf:  29.11
mape_train_rf:  12.08
mape_test_rf:  28.75
```

Upon multiple feature search, the feature search using Forward Search ~ Feature Selection using RFE & Feature Importance Ranks gives best results with top 5 features selected. Selecting:

- **NSM**, Number of seconds from midnight
- **T_out**, Temperature outside (from Chièvres weather station)

- **Tdewpoint**, Dew point temp (from Chièvres weather station)
- **RH_6**, Humidity outside the building (north side)
- **Press_mm_hg**, Pressure (from Chièvres weather station)

5.f Boruta Library – After implementing Boruta library, the below graph shows the rank of the predictors



5.g TPOT Library – The goal of TPOT is to automate the building of ML pipelines by combining a flexible expression tree representation of pipelines with stochastic search algorithms such as genetic programming. TPOT makes use of the Python-based scikit-learn library as its ML menu.

The below code will give the rank by using TPOT library

```
from tpot import TPOTRegressor
#from sklearn.datasets import load_digits
import numpy as np

X = df[selected_cols]
y = df['Appliances']

# Taking very small dataset due computational power constraints
X_tpot_train, X_tpot_test, y_tpot_train, y_tpot_test = train_test_split(X, y, train_size=0.05, test_size=
0.025)

tpot = TPOTClassifier(generations=10, population_size=100, verbosity=2,
                      n_jobs=-1,warm_start=True, periodic_checkpoint_folder="./optCode/")

tpot.fit(X_tpot_train, y_tpot_train)
print(tpot.score(X_tpot_test, y_tpot_test))
tpot.export('tpot_mnist_pipeline.py')
```

Part 6 Model Validation and Selection –

6.a Cross Validation - In the basic approach, called k-fold CV, the training set is split into k smaller sets (other approaches are described below, but generally follow the same principles). The following procedure is followed for each of the k "folds":

- A model is trained using k-5 of the folds as training data;
- the resulting model is validated on the remaining part of the data (i.e., it is used as a test set to compute a performance measure such as accuracy). The performance measure reported by k-fold cross-validation is then the average of the values computed in the loop. This approach can be computationally expensive, but does not waste too much data (as it is the case when fixing an arbitrary test set), which is a major advantage in problem such as inverse inference where the number of samples is very small.

```
from sklearn.cross_validation import cross_val_score

rf = RandomForestRegressor()
scores = cross_val_score(rf, X, y, cv=5, scoring='r2')
print("R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

R2: -0.18 (+/- 0.63)
Above, is the mean score and the 95% confidence interval of the score estimate

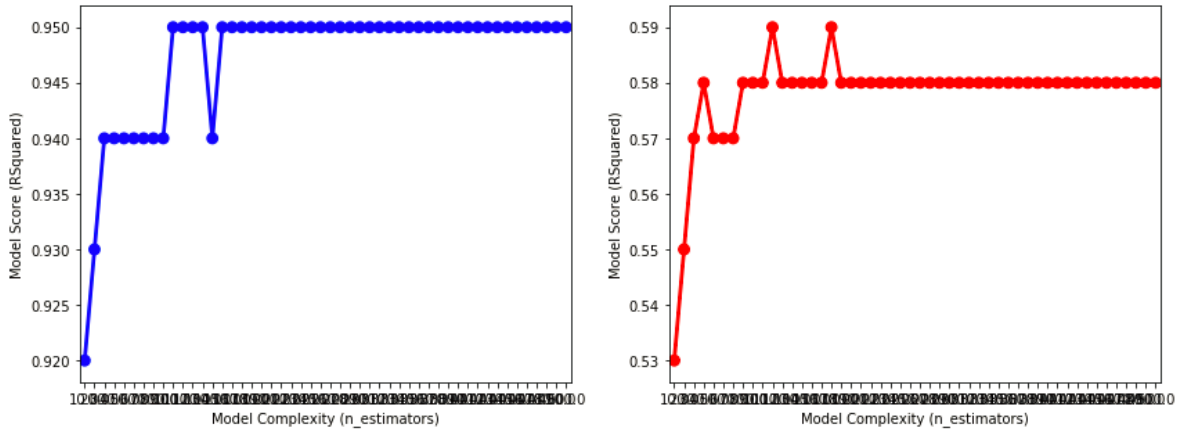
6.b Bias – Variance Trade off- Fundamentally, the question of "the best model" is about finding a sweet spot in the tradeoff between bias and variance. Consider the following figure, which presents two regression fits to the same dataset:

- The model on the left attempts to find a straight-line fit through the data.
- The model on the right attempts to fit a high-order polynomial through the data.

The score here is the R2 score, or coefficient of determination, which measures how well a model performs relative to a simple mean of the target values. R2=1 indicates a perfect match, R2=0 indicates the model does no better than simply taking the mean of the data, and negative values mean even worse models. From the scores associated with these two models, we can make an observation that holds more generally:

- For high-bias models, the performance of the model on the validation set is similar to the performance on the training set.
- For high-variance models, the performance of the model on the validation set is far worse than the performance on the training set.

```
fig, (ax1, ax2) = plt.subplots(figsize=(14,5), ncols=2)
sns.pointplot(x='n_estimators', y='RSquared_train', data=metrics_df.T, ax=ax1, color='blue', label='Train', linestyle='-')
sns.pointplot(x='n_estimators', y='RSquared_test', data=metrics_df.T, ax=ax2, color='red', label='Test', linestyle='-')
ax1.set_ylabel('Model Score (RSquared)')
ax1.set_xlabel('Model Complexity (n_estimators)')
ax2.set_ylabel('Model Score (RSquared)')
ax2.set_xlabel('Model Complexity (n_estimators)')
```



6.c Grid Search – Scikit-Learn provides automated tools to do this in the grid search module. Here is an example of using grid search to find the optimal polynomial model. We will explore a three-dimensional grid of model features; namely the polynomial degree, the flag telling us whether to fit the intercept, and the flag telling us whether to normalize the problem. This can be set up using Scikit-Learn's `GridSearchCV` meta-estimator:

```
from sklearn import linear_model
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer


param_grid = {'polynomialfeatures__degree': np.arange(21),
              'linearregression__fit_intercept': [True, False],
              'linearregression__normalize': [True, False]}


def PolynomialRegression(degree=2, **kwargs):
    return make_pipeline(PolynomialFeatures(degree),
                         LinearRegression(**kwargs))


grid = GridSearchCV(PolynomialRegression(), param_grid, cv=7)

X = df.drop(['Appliances'],axis=1)
y = df['Appliances']


grid.fit(X_train, y_train)
grid.best_params_
model = grid.best_estimator_

plt.scatter(X.ravel(), y)
lim = plt.axis()
y_test = model.fit(X, y).predict(X_test)
plt.plot(X_test.ravel(), y_test, hold=True);
plt.axis(lim);
```

6.d Regularization -Regularization is a very important technique in machine learning to prevent overfitting. Mathematically speaking, it adds a regularization term in order to prevent the coefficients to fit so perfectly to overfit. The difference between the L1 and L2 is just that L2 is the sum of the square of the weights, while L1 is just the sum of the weights.