ASSIGNMENT 2&3 - REPORT

Machine learning with Energy datasets

COURSE: INFO7390
Advance Data Science & Architecture

PROFESSOR:
Srikanth Krishnamurthy

SUBMITTED BY:
TEAM 9
Amit Pingale - 001898697
Himani Solanki - 001899580
Shubham Patel - 001899476

Objective:

The Report summarizes the design and implementation of machine learning performed on the Appliance Energy Consumption dataset. The report is divided into:

Part 1: Research
Part 2: Exploratory Data Analysis
Part 3: Feature engineering
Part 4: Prediction algorithms
Part 5: Feature Selection
Part 6: Final pipeline

Part 1: Research

Reviewed the following papers:

A. https://www.sciencedirect.com/science/article/pii/S0378778816308970?via%3Dihub
B. https://www.sciencedirect.com/science/article/pii/S1364032116307420
C. https://www.sciencedirect.com/science/article/pii/S0360544212002903

The respective jupyter notebook:

A.https://github.com/ADSteam9/ADS/blob/master/Assignment_2/Research_Paper/PaperSummary1.ipynb
B.https://github.com/ADSteam9/ADS/blob/master/Assignment_2/Research_Paper/PaperSummary2.ipynb
C.https://github.com/ADSteam9/ADS/blob/master/Assignment_2/Research_Paper/PaperSummary3.ipynb

Part 2: Exploratory Data Analysis

The Exploratory Data Analysis is used for the following purpose:

- Test business assumptions
- Generate hypotheses for further analysis
- Prepare the data for modeling

Step1: Data Information :

```
date time year-month-day hour:minute:second
Appliances, energy use in Wh
lights, energy use of light fixtures in the house in Wh
T1, Temperature in kitchen area, in Celsius
RH_1, Humidity in kitchen area, in %
T2, Temperature in living room area, in Celsius
RH_2, Humidity in living room area, in %
T3, Temperature in laundry room area
RH_3, Humidity in laundry room area, in %
T4, Temperature in office room, in Celsius
RH_4, Humidity in office room, in %
T5, Temperature in bathroom, in Celsius
RH_5, Humidity in bathroom, in %
T6, Temperature outside the building (north side), in Celsius
RH_6, Humidity outside the building (north side), in %
T7, Temperature in ironing room , in Celsius
RH_7, Humidity in ironing room, in %
T8, Temperature in teenager room 2, in Celsius
RH_8, Humidity in teenager room 2, in %
T9, Temperature in parents room, in Celsius
RH_9, Humidity in parents room, in %
To, Temperature outside (from Chièvres weather station), in Celsius
Pressure (from Chièvres weather station), in mm Hg
RH_out, Humidity outside (from Chièvres weather station), in %
Windspeed (from Chièvres weather station), in m/s
Visibility (from Chièvres weather station), in km
Tdewpoint (from Chièvres weather station), °C
rv1, Random variable 1, nondimensional
rv2, Rnadom variable 2, nondimensional
```

Step 2: Reading the data set

```
In [5]: energy_data = pd.read_csv('energydata_complete.csv')
        energy_data.head()
```

Out[5]:

	date	Appliances	lights	T1	RH_1	T2	RH_2	T3	RH_3	T4	...	T9	RH_9	T_out	Press_mm_hg	RH_out	Winds
0	2016-01-11 17:00:00	60	30	19.89	47.596667	19.2	44.790000	19.79	44.730000	19.000000	...	17.033333	45.53	6.600000	733.5	92.0	7.00
1	2016-01-11 17:10:00	60	30	19.89	46.693333	19.2	44.722500	19.79	44.790000	19.000000	...	17.066667	45.56	6.483333	733.6	92.0	6.66
2	2016-01-11 17:20:00	50	30	19.89	46.300000	19.2	44.626667	19.79	44.933333	18.926667	...	17.000000	45.50	6.366667	733.7	92.0	6.33
3	2016-01-11 17:30:00	50	40	19.89	46.066667	19.2	44.590000	19.79	45.000000	18.890000	...	17.000000	45.40	6.250000	733.8	92.0	6.00
4	2016-01-11 17:40:00	60	40	19.89	46.333333	19.2	44.530000	19.79	45.000000	18.890000	...	17.000000	45.40	6.133333	733.9	92.0	5.66

5 rows × 29 columns

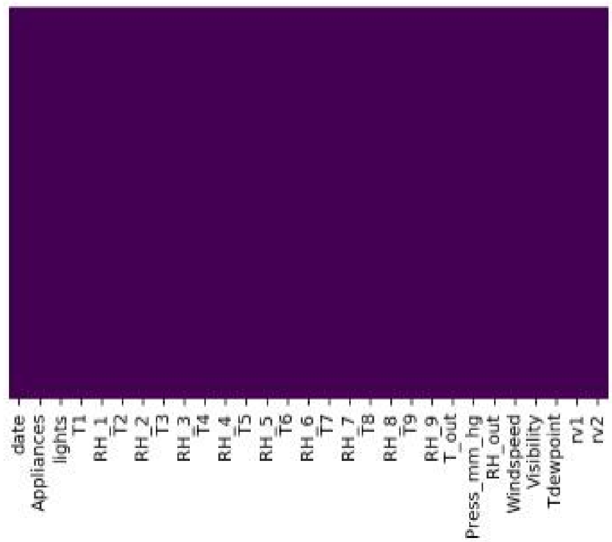
Step 3: Getting the data information

```
In [6]: energy_data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19735 entries, 0 to 19734
Data columns (total 29 columns):
date           19735 non-null object
Appliances     19735 non-null int64
lights         19735 non-null int64
T1             19735 non-null float64
RH_1           19735 non-null float64
T2             19735 non-null float64
RH_2           19735 non-null float64
T3             19735 non-null float64
RH_3           19735 non-null float64
T4             19735 non-null float64
RH_4           19735 non-null float64
T5             19735 non-null float64
RH_5           19735 non-null float64
T6             19735 non-null float64
RH_6           19735 non-null float64
T7             19735 non-null float64
RH_7           19735 non-null float64
T8             19735 non-null float64
RH_8           19735 non-null float64
T9             19735 non-null float64
RH_9           19735 non-null float64
T_out          19735 non-null float64
Press_mm_hg    19735 non-null float64
RH_out         19735 non-null float64
Windspeed      19735 non-null float64
Visibility     19735 non-null float64
Tdewpoint      19735 non-null float64
rv1            19735 non-null float64
rv2            19735 non-null float64
dtypes: float64(26), int64(2), object(1)
memory usage: 4.4+ MB
```

Step 4: Missing Value Analysis:

```
In [8]: sns.heatmap(energy_data.isnull(), yticklabels=False, cbar=False, cmap='viridis')
```

```
Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x1069baa58>
```



Step 5: Data Pre-processing

```
In [9]: energy_data['date'] = pd.to_datetime(energy_data.date)
energy_data['year'] = energy_data.date.dt.year
energy_data['month'] = energy_data.date.dt.month
energy_data['day'] = energy_data.date.dt.day
energy_data['hours'] = energy_data.date.dt.hour
energy_data['minutes'] = energy_data.date.dt.minute
energy_data['seconds'] = energy_data.date.dt.second
energy_data['week'] = energy_data.date.dt.week
energy_data['day_name'] = energy_data.date.dt.weekday_name
energy_data['day_of_week'] = energy_data.date.dt.dayofweek
energy_data['weekday'] = ((energy_data.date.dt.dayofweek // 5 == 1).astype(int))

energy_data.to_csv("energydata_complete_revised.csv", index=False)
energy_data.head()
```

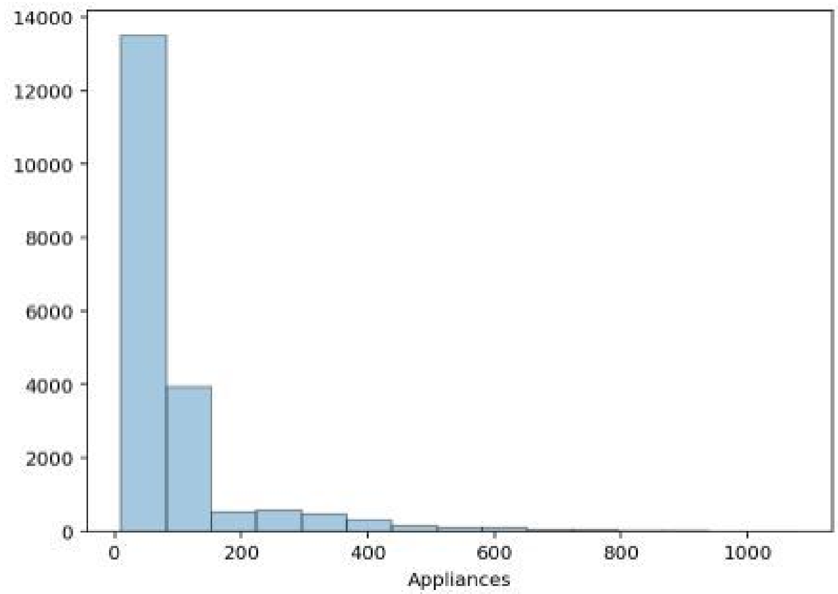
Out[9]:

	date	Appliances	lights	T1	RH_1	T2	RH_2	T3	RH_3	T4	...	year	month	day	hours	minutes	seconds	week	day_nam
0	2016-01-11 17:00:00	60	30	19.89	47.596667	19.2	44.790000	19.79	44.730000	19.000000	...	2016	1	11	17	0	0	2	Mond:
1	2016-01-11 17:10:00	60	30	19.89	46.693333	19.2	44.722500	19.79	44.790000	19.000000	...	2016	1	11	17	10	0	2	Mond:
2	2016-01-11 17:20:00	50	30	19.89	46.300000	19.2	44.626667	19.79	44.933333	18.926667	...	2016	1	11	17	20	0	2	Mond:
3	2016-01-11 17:30:00	50	40	19.89	46.066667	19.2	44.590000	19.79	45.000000	18.890000	...	2016	1	11	17	30	0	2	Mond:
4	2016-01-11 17:40:00	60	40	19.89	46.333333	19.2	44.530000	19.79	45.000000	18.890000	...	2016	1	11	17	40	0	2	Mond:

5 rows × 39 columns

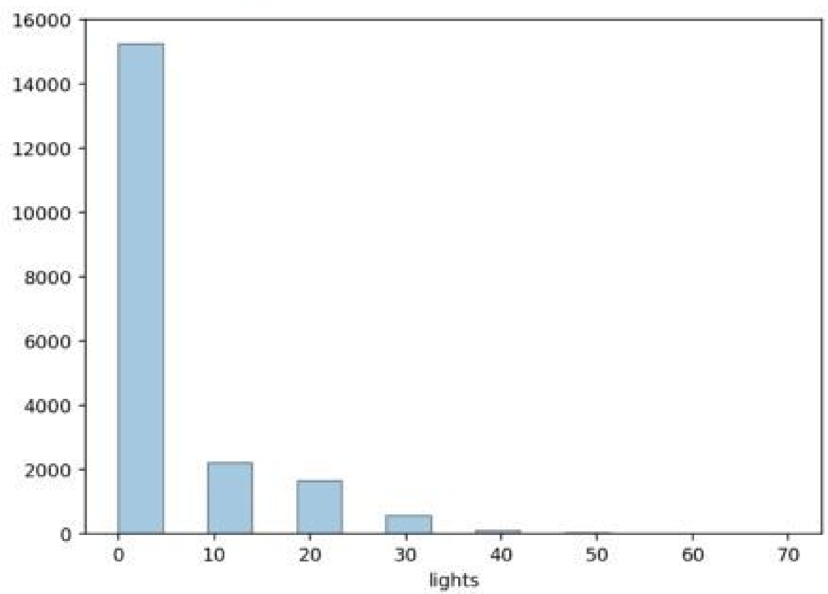
Step 6: Plot Frequency v/s Appliance

```
In [10]: plt.figure(figsize=(7,5))
         sns.distplot(energy_data['Appliances'],kde=False,bins=15)

Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x1a0d4ba320>
```

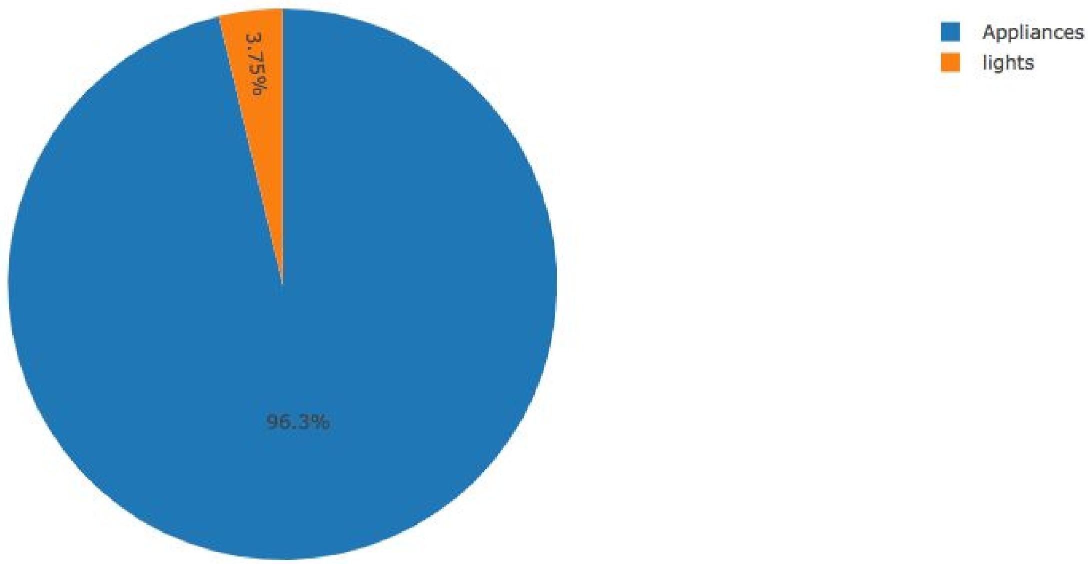


Step 7: Plot between Frequency v/s Lights

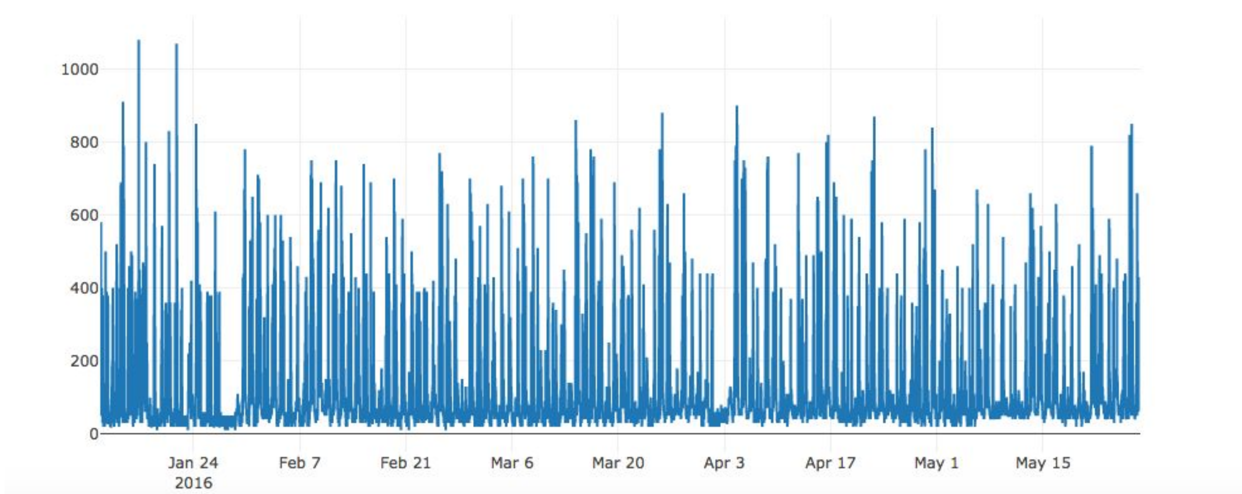
```
In [11]: plt.figure(figsize=(7,5))
         sns.distplot(energy_data['lights'],kde=False,bins=15)

Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x1a0e72e0b8>
```



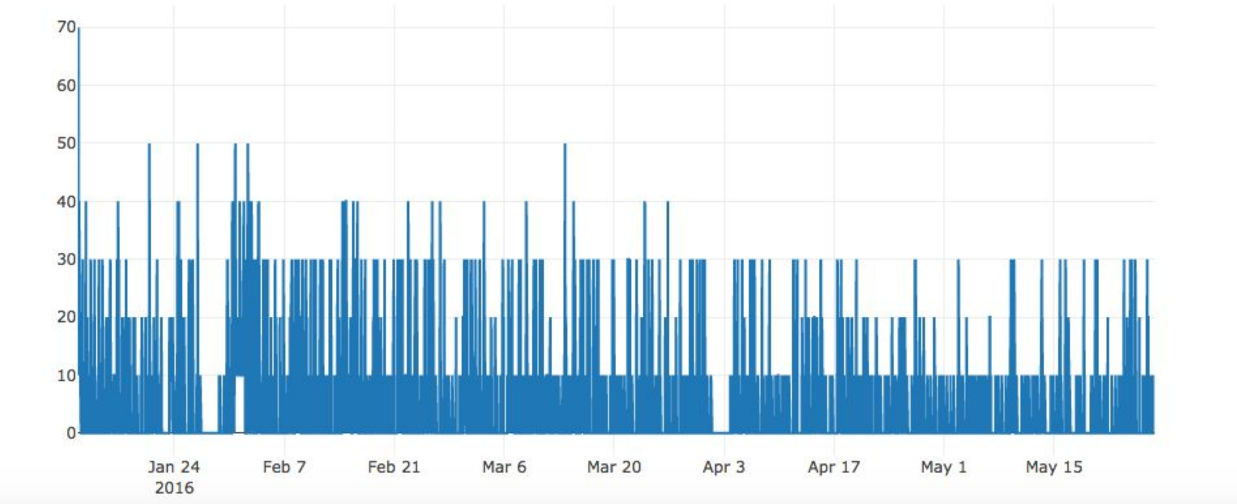
Step 8: Analysis for the target variables using PiePlot. Ratio of appliances with respect to lights



Step 9: Energy consumed by appliances in 1st year quarter



Step 10: Energy consumed by lights in 1st year quarter



Step 11: Calculating correlation

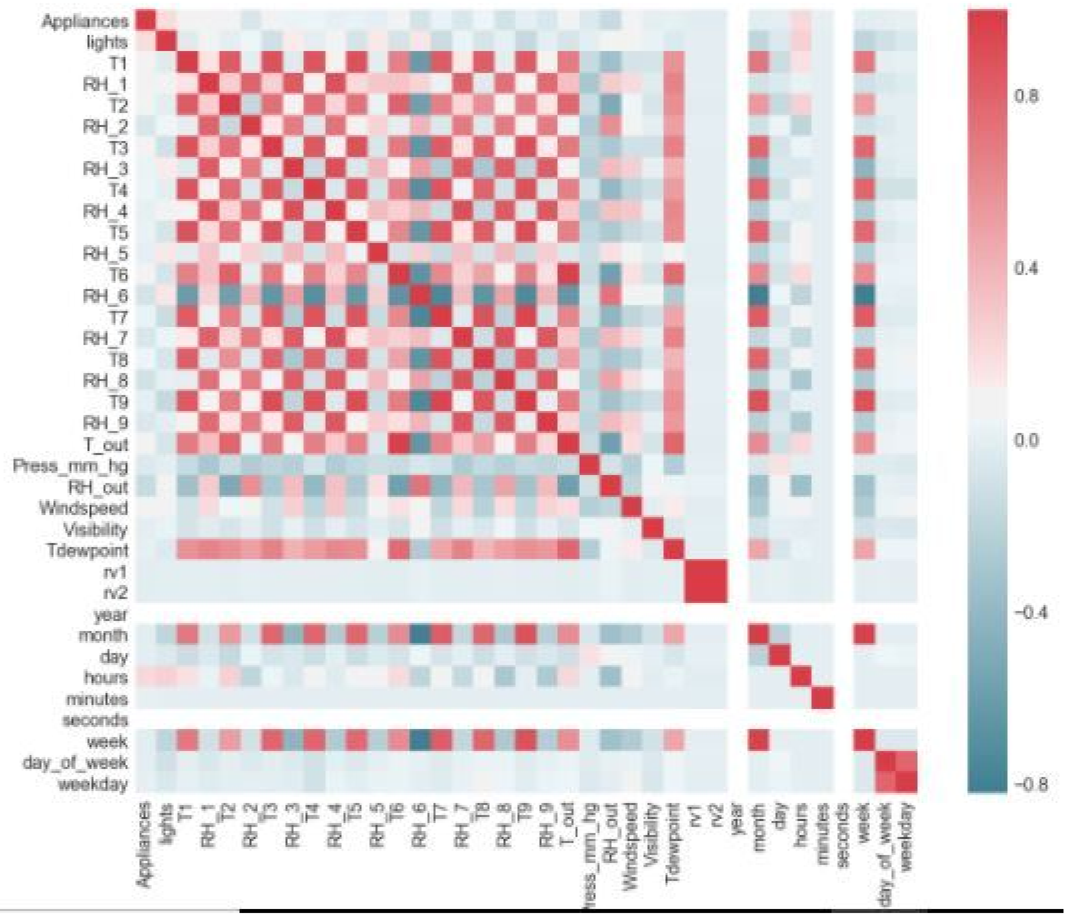
In [148]:
```
corr_data = energy_data.drop(['year', 'seconds'], axis=1)
corr_data = corr_data.corr()
corr_data.head()
```

Out[148]:

	Appliances	lights	T1	RH_1	T2	RH_2	T3	RH_3	T4	RH_4	...	Tdewpoint	rv1	rv2	m
Appliances	1.000000	0.197278	0.055447	0.086031	0.120073	-0.060465	0.085060	0.036292	0.040281	0.016965	...	0.015353	-0.011145	-0.011145	-0.01
lights	0.197278	1.000000	-0.023528	0.106968	-0.005622	0.050985	-0.097393	0.131161	-0.008859	0.114936	...	-0.036322	0.000521	0.000521	-0.17
T1	0.055447	-0.023528	1.000000	0.164006	0.836834	-0.002509	0.892402	-0.028550	0.877001	0.097861	...	0.571309	-0.006203	-0.006203	0.70
RH_1	0.086031	0.106968	0.164006	1.000000	0.269839	0.797535	0.253230	0.844677	0.106180	0.880359	...	0.639106	-0.000699	-0.000699	-0.09
T2	0.120073	-0.005622	0.836834	0.269839	1.000000	-0.165610	0.735245	0.121497	0.762066	0.231563	...	0.582602	-0.011087	-0.011087	0.53

5 rows × 35 columns

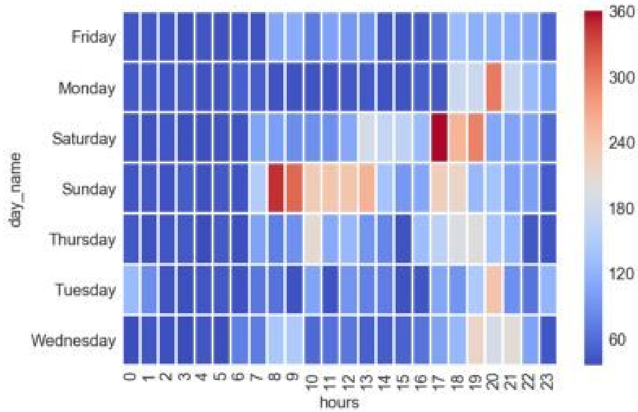
```
In [19]: f, ax = plt.subplots(figsize=(10, 8))
         corr = energy_data.corr()
         sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool),
                     cmap=sns.diverging_palette(220, 10, as_cmap=True),
                     square=True, ax=ax)

Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1cc017b8>
```



Step 12: HeatMap

In [147]:
```
month1 = energy_data[energy_data['month'] == 1]
month1_pivot =month1.pivot_table(index='day_name', columns='hours', values='Appliances')
sns.heatmap(month1_pivot, linecolor='white', linewidths=1, cmap='coolwarm')
```

Out[147]: <matplotlib.axes._subplots.AxesSubplot at 0x1a3ce18cf8>



Part 3: Feature engineering

Step 1: Reading the revised csv file

In [42]:
```
df = pd.read_csv("energydata_complete_revised.csv")
df.head()
```

Out[42]:

	Unnamed: 0	date	Appliances	lights	T1	RH_1	T2	RH_2	T3	RH_3	...	rv2	year	month	day	hours	minutes	seconds	v
0	0	2016-01-11 17:00:00	60	30	19.89	47.596667	19.2	44.790000	19.79	44.730000	...	13.275433	2016	1	11	17	0	0	
1	1	2016-01-11 17:10:00	60	30	19.89	46.693333	19.2	44.722500	19.79	44.790000	...	18.606195	2016	1	11	17	10	0	
2	2	2016-01-11 17:20:00	50	30	19.89	46.300000	19.2	44.626667	19.79	44.933333	...	28.642668	2016	1	11	17	20	0	
3	3	2016-01-11 17:30:00	50	40	19.89	46.066667	19.2	44.590000	19.79	45.000000	...	45.410389	2016	1	11	17	30	0	
4	4	2016-01-11 17:40:00	60	40	19.89	46.333333	19.2	44.530000	19.79	45.000000	...	10.084097	2016	1	11	17	40	0	

5 rows × 39 columns

Step 2: Using one-hot encoding encoded the Day_name column to individual weekdays named columns(Sunday,Monday,...)

In [43]:
```
day_name_encoding = pd.get_dummies(df['day_name'])
list(day_name_encoding.columns.values)
day_name_encoding.head()
```

Out[43]:

	Friday	Monday	Saturday	Sunday	Thursday	Tuesday	Wednesday
0	0	1	0	0	0	0	0
1	0	1	0	0	0	0	0
2	0	1	0	0	0	0	0
3	0	1	0	0	0	0	0
4	0	1	0	0	0	0	0

Step 3: Concating the encoded day_name to energy data set

```
In [44]: data2 = pd.concat([df,day_name_encoding],axis=1, )
```

```
In [45]: data2.columns.values
```

```
Out[45]: array(['Unnamed: 0', 'date', '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', 'rv1', 'rv2',
               'year', 'month', 'day', 'hours', 'minutes', 'seconds', 'week',
               'day_name', 'weekday', 'Friday', 'Monday', 'Saturday', 'Sunday',
               'Thursday', 'Tuesday', 'Wednesday'], dtype=object)
```

Step 5: Analysis of coefficient

```
In [60]: coeff_df = pd.DataFrame(lm.coef_,X.columns,columns=['Coefficient'])
         coeff_df
```

Out[60]:

	Coefficient
lights	1.996121e+00
T1	-6.786990e+00
RH_1	1.504226e+01
T2	-1.878083e+01
RH_2	-1.449467e+01
T3	2.839187e+01
RH_3	5.811535e+00
T4	1.087502e+00
RH_4	-3.406372e-01
T5	8.778356e-01
RH_5	6.285350e-02
T6	6.986168e+00
RH_6	7.088992e-02
T7	1.181055e+00
RH_7	-1.243376e+00
T8	7.587025e+00
RH_8	-4.006524e+00

Part 4: Prediction algorithms

Step 1: Linear Regression

```
In [7]: X = energy_data[['hours','T1','RH_1','RH_5','RH_6','RH_8','T_out','Windspeed','Visibility','Tdewpoint']]
        y = energy_data['Appliances']
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

```
In [8]: from sklearn.linear_model import LinearRegression
        lm = LinearRegression()
        lm.fit(X_train,y_train)

Out[8]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

```
In [9]: print(lm.intercept_)

        -97.8260529478
```

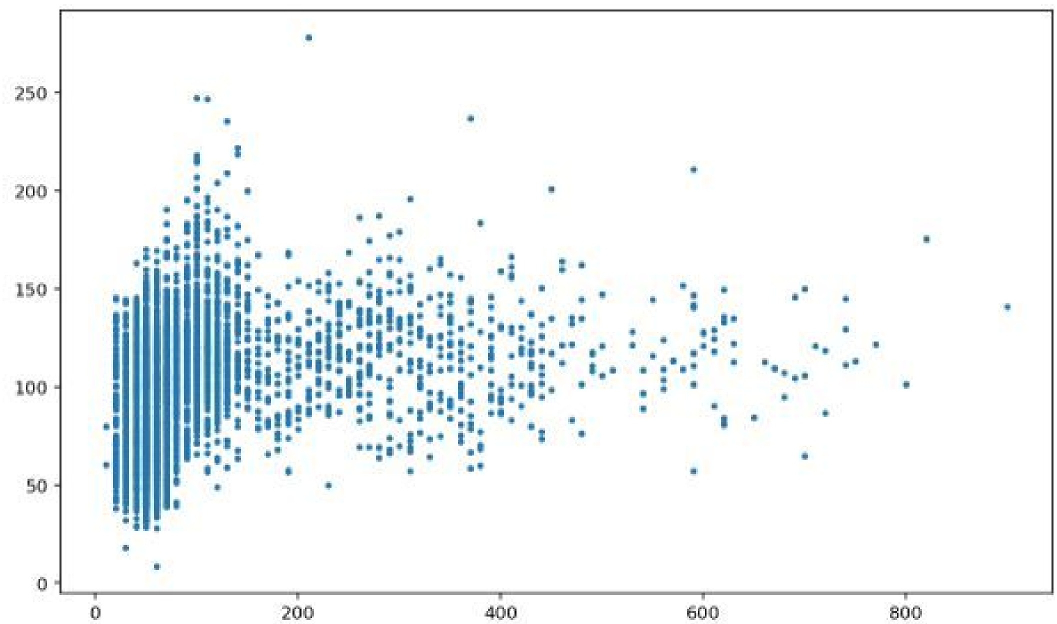
```
In [10]: coeff_df = pd.DataFrame(lm.coef_,X.columns,columns=['Coefficient'])
         print(coeff_df)

                     Coefficient
         hours          1.717167
         T1             0.870430
         RH_1           8.364219
         RH_5          -0.072304
         RH_6           0.052212
         RH_8          -4.614726
         T_out          2.539544
         Windspeed      2.431697
         Visibility     0.217996
         Tdewpoint     -4.636342
```

<u>Step 2:</u> Plotting scatter for linear regression

```
In [11]: predictions = lm.predict(X_test)
```

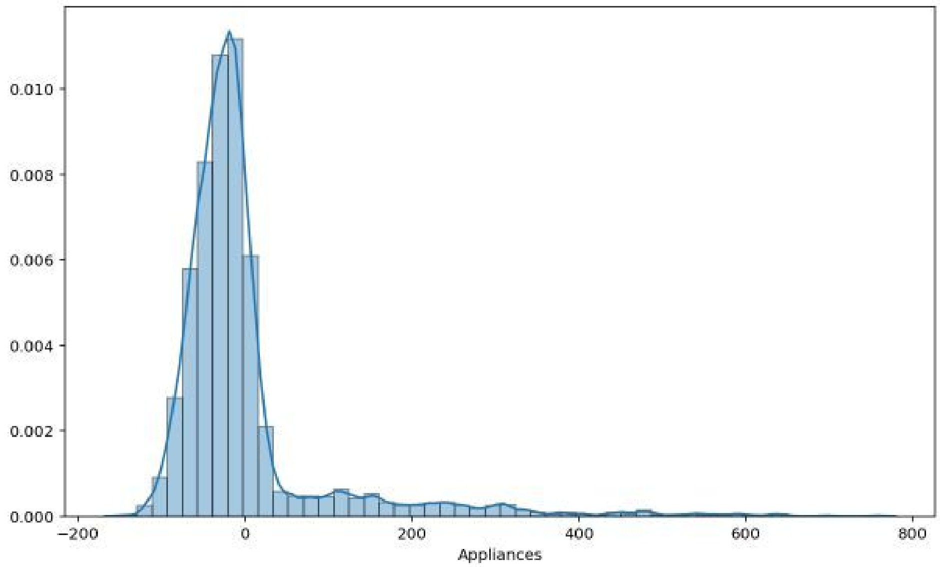
```
In [12]: plt.figure(figsize=(10,6))
         plt.scatter(y_test,predictions, s = 7)
```

```
Out[12]: <matplotlib.collections.PathCollection at 0x118bfe940>
```



Step 3: Analysis of residual

```
In [13]: plt.figure(figsize=(10,6))
         sns.distplot((y_test-predictions),bins=50);
```



Step 4: Calculating and analysing metrics

```
In [18]: from sklearn import metrics
         print('MAE:', metrics.mean_absolute_error(y_test, predictions))
         print('MSE:', metrics.mean_squared_error(y_test, predictions))
         print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))
         print('R2:', metrics.r2_score(y_test, predictions))

MAE: 55.3358885225
MSE: 9116.53059277
RMSE: 95.4805246779
R2: 0.0822771755142
```

Step 5: Random Forest: uses decision tree with boosting and bootstrapping

In [19]:
```
labels = np.array(energy_data['Appliances'])
features = energy_data[['hours','T1','RH_1','RH_5','RH_6','RH_8','T_out','Windspeed','Visibility','Tdewpoint']]
feature_list = list(features.columns)
features = np.array(features)
```

In [20]:
```
from sklearn.model_selection import train_test_split
train_features, test_features, train_labels, test_labels = train_test_split(features, labels, test_size = 0.25, random_state = 42
```

In [21]:
```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(n_estimators = 1000, random_state = 42)
rf.fit(train_features, train_labels);
```

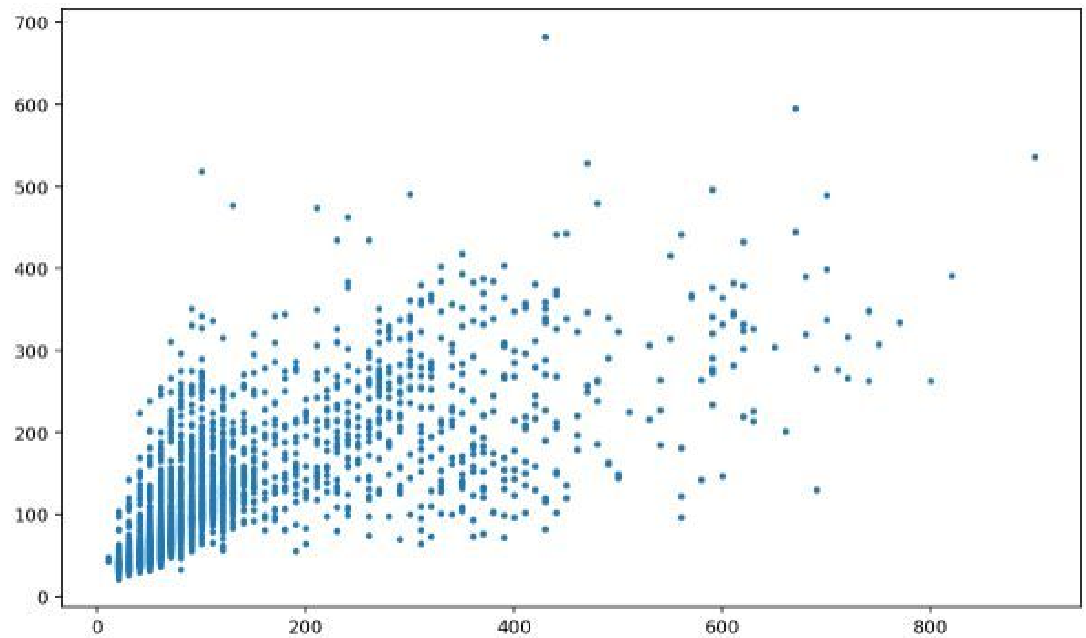
In [22]:
```
predictions = rf.predict(test_features)
```

In [24]:
```
# Calculate the absolute errors
errors = abs(predictions - test_labels)
# Print out the mean absolute error (mae)
print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')

# Calculate mean absolute percentage error (MAPE)
mape = 100 * (errors / test_labels)
# Calculate and display accuracy
accuracy = 100 - np.mean(mape)
print('Accuracy:', round(accuracy, 2), '%.')

from sklearn import metrics
print('MAE:', metrics.mean_absolute_error(test_labels, predictions))
print('MSE:', metrics.mean_squared_error(test_labels, predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(test_labels, predictions)))
print('R2:', metrics.r2_score(test_labels, predictions))
```

```
Mean Absolute Error: 31.16 degrees.
Accuracy: 68.59 %.
MAE: 31.159014998
MSE: 4232.98236749
RMSE: 65.0613738519
R2: 0.573883453276
```

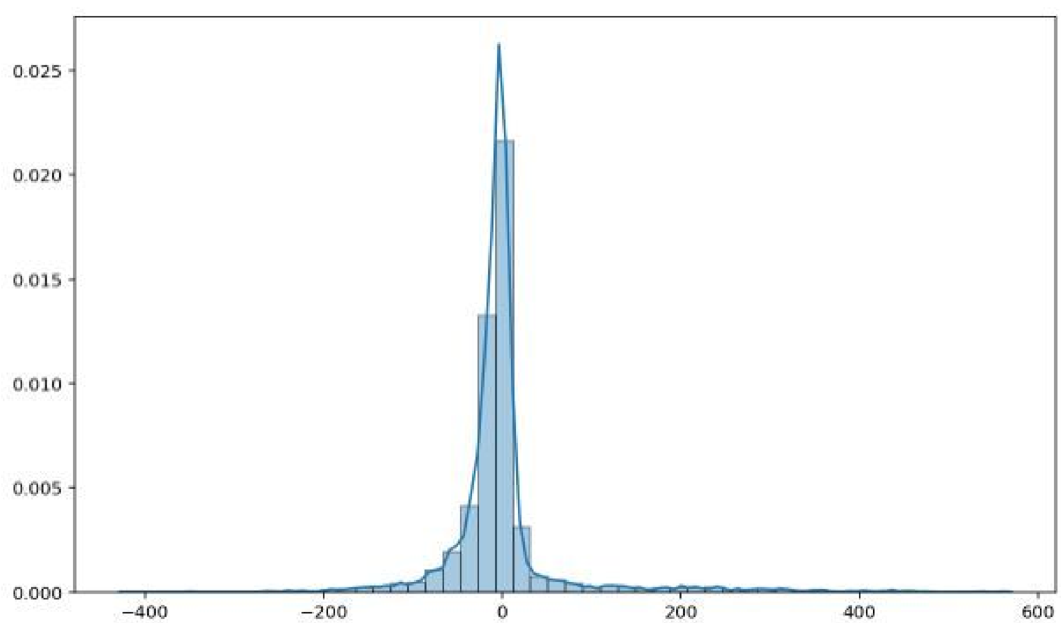
Step 6: **Scatter Plot for random forest regression**

```
In [25]: plt.figure(figsize=(10,6))
         plt.scatter(test_labels, predictions, s = 7)
Out[25]: <matplotlib.collections.PathCollection at 0x1a5049eba8>
```



Step 7: **Residual Analysis**

```
In [26]: plt.figure(figsize=(10,6))
         sns.distplot((test_labels-predictions),bins=50);
```



Step 8: **Metric calculation**

```
In [24]: # Calculate the absolute errors
errors = abs(predictions - test_labels)
# Print out the mean absolute error (mae)
print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')

# Calculate mean absolute percentage error (MAPE)
mape = 100 * (errors / test_labels)
# Calculate and display accuracy
accuracy = 100 - np.mean(mape)
print('Accuracy:', round(accuracy, 2), '%.')

from sklearn import metrics
print('MAE:', metrics.mean_absolute_error(test_labels, predictions))
print('MSE:', metrics.mean_squared_error(test_labels, predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(test_labels, predictions)))
print('R2:', metrics.r2_score(test_labels, predictions))
```

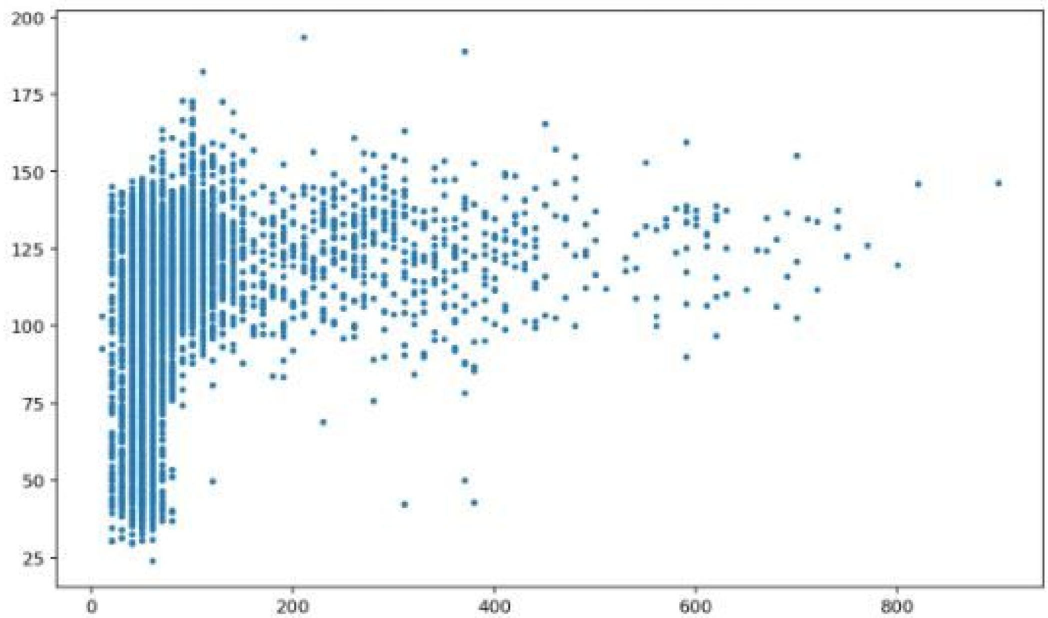
```
Mean Absolute Error: 31.16 degrees.
Accuracy: 68.59 %.
MAE: 31.159014998
MSE: 4232.98236749
RMSE: 65.0613738519
R2: 0.573883453276
```

Step 9: Neural Network

In [31]:
```
from sklearn.neural_network import MLPRegressor
nn = MLPRegressor(activation='relu',learning_rate='adaptive',alpha=0.55)
modelneuralnetwork = nn.fit(X_train, y_train)

y_train_prediction = nn.predict(X_train)
y_test_prediction = nn.predict(X_test)
```

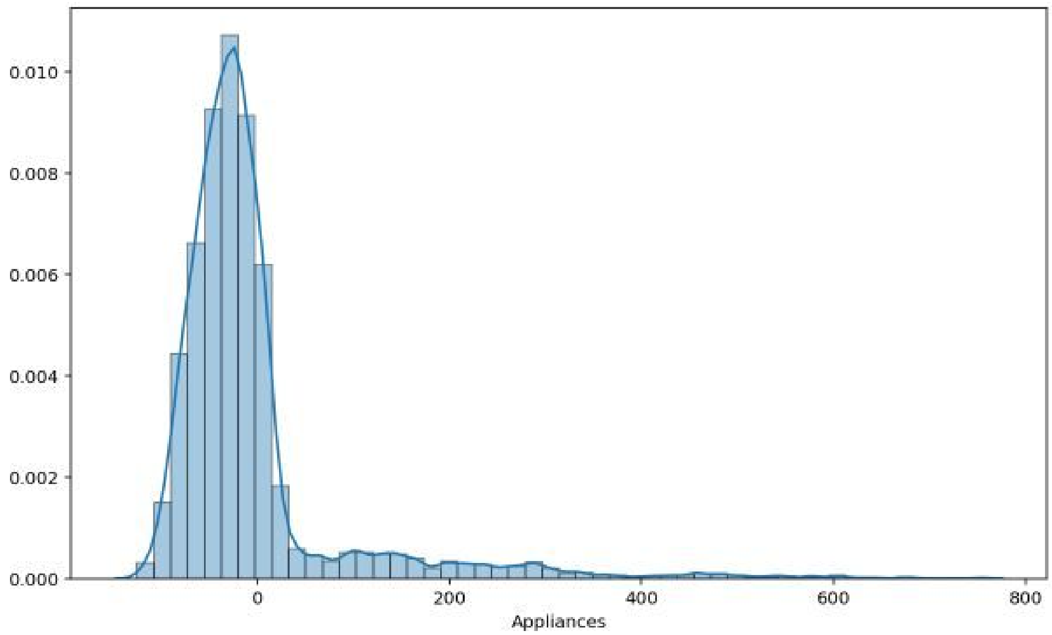
In [32]:
```
plt.figure(figsize=(10,6))
plt.scatter(y_test,y_test_prediction, s = 7)
```

Out[32]: <matplotlib.collections.PathCollection at 0x1a58467c88>



Step 10: Residual Analysis

In [33]:
```
plt.figure(figsize=(10,6))
sns.distplot((y_test - y_test_prediction),bins=50);
```



Step 11: Metric Calculation

In [36]:
```
from sklearn import metrics
from sklearn.metrics import r2_score
print('MAE:', metrics.mean_absolute_error(y_test, y_test_prediction))
print('MSE:', metrics.mean_squared_error(y_test, y_test_prediction))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_test_prediction)))
print('R2:', metrics.r2_score(y_test, y_test_prediction))
```

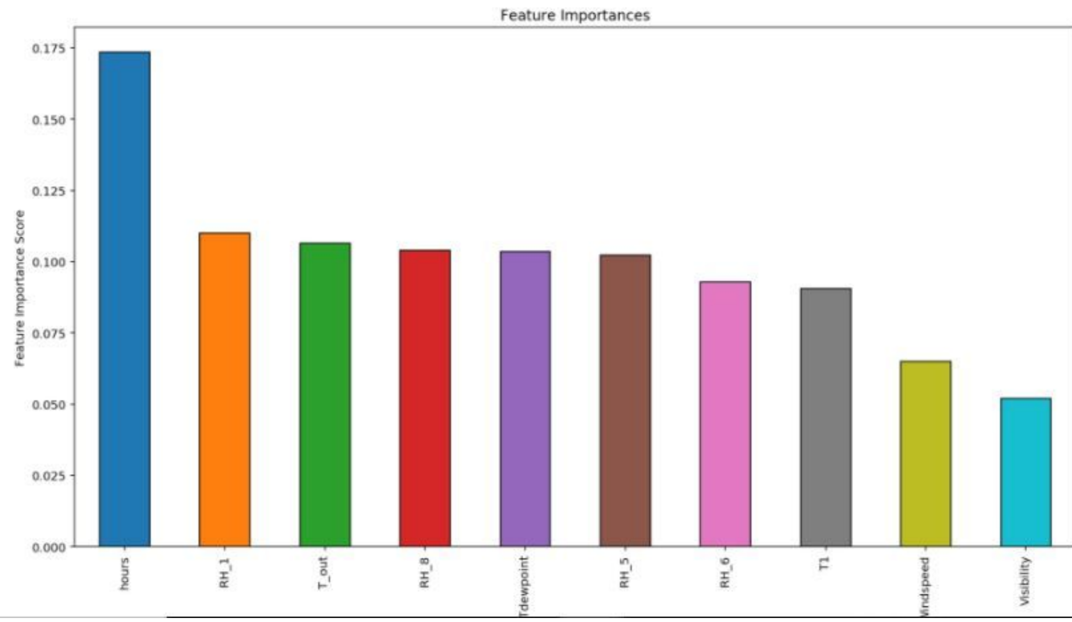
```
MAE: 56.8324185253
MSE: 8917.77015971
RMSE: 94.4339460136
R2: 0.102285553062
```

Part 5: Feature Selection

Step 1: Taking multicollinearity as factor relation with dependent variable

In [38]:
```
feat_imp = pd.Series(rf.feature_importances_,X_train.columns).sort_values(ascending=False)
feat_imp.plot(kind='bar', title='Feature Importances',figsize = (15,8))
plt.ylabel('Feature Importance Score')
```

Out[38]:
```
Text(0,0.5,'Feature Importance Score')
```



Step 2: Boruta

```
In [29]: data_1 = data.drop(['Appliances', 'date', 'day_name'],axis=1)
         X_boruta = data_1
         X_boruta = X_boruta.values

         y_boruta = data['Appliances']
         y_boruta = y_boruta.values
         y_boruta

Out[29]: array([ 60,  60,  50, ..., 270, 420, 430])
```

```
In [30]: from sklearn.ensemble import RandomForestClassifier
         rf1 = RandomForestClassifier(n_jobs=-1, class_weight='balanced', max_depth=5)
```

```
In [31]: from boruta import BorutaPy
         feat_selector = BorutaPy(rf1, n_estimators='auto', verbose=5, random_state=1)
         feat_selector.fit(X_boruta,y_boruta)

         Iteration:      1 / 100
         Confirmed:      0
         Tentative:      42
         Rejected:       0
         Iteration:      2 / 100
         Confirmed:      0
         Tentative:      42
         Rejected:       0
         Iteration:      3 / 100
         Confirmed:      0
         Tentative:      42
         Rejected:       0
         Iteration:      4 / 100
         Confirmed:      0
         Tentative:      42
         Rejected:       0
         Iteration:      5 / 100
```

Step 3: Computing ranking of each feature for prediction

```
In [32]: feat_selector.support_
         print(feat_selector.ranking_)

[29  8  6 12 18  1 19  9 17 10  2  7 22 15 24 12 26 15 25 10 14 28 23  3 26
  4  5 41 35 20 30 31 41 20 34 39 36 32 37 33 39 39]
```

```
In [73]: ranking = pd.DataFrame({'Index':data_1.columns, 'Ranking': feat_selector.ranking_})
         ranking
```

Out[73]:

	Index	Ranking
0	lights	29
1	T1	8
2	RH_1	6
3	T2	12
4	RH_2	18
5	T3	1
6	RH_3	19
7	T4	9

Step 4: TPOT

```
In [48]: X = data.drop(['Appliances', 'date', 'day_name'],axis=1)
         y = data['Appliances']
```

```
In [50]: from sklearn.model_selection import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

```
In [52]: from tpot import TPOTRegressor

         tpot = TPOTRegressor(generations=10, population_size=10,offspring_size=None, mutation_rate=0.9, verbosity=3)
         tpot.fit(X_train, y_train)
         print(tpot.score(X_test, y_test))
         tpot.export('tpot_pipeline.py')
```

```
Warning: xgboost.XGBRegressor is not available and will not be used by TPOT.
28 operators have been imported by TPOT.
Optimization Progress:  18%|██        | 20/110 [02:21<09:04,  6.05s/pipeline]
Generation 1 - Current Pareto front scores:
-1      -6160.052847414862      RandomForestRegressor(input_matrix, RandomForestRegressor__bootstrap=False, RandomForestRegre
ssor__max_features=0.25, RandomForestRegressor__min_samples_leaf=16, RandomForestRegressor__min_samples_split=6, RandomForest
Regressor__n_estimators=100)

Optimization Progress:  19%|██        | 21/110 [02:21<06:26,  4.34s/pipeline]
Pipeline encountered that has previously been evaluated during the optimization process. Using the score from the previous ev
aluation.
Optimization Progress:  27%|███       | 30/110 [04:55<23:30, 17.63s/pipeline]
```

<u>Step 5:</u> TFRESH

In [70]:
```
tsfresh = energy_data
from tsfresh import extract_features
X_tsfresh = tsfresh[['date','hours','T1','RH_1','RH_5','RH_6','RH_8','T_out','Windspeed','Visibility','Tdewpoint']]
y_tsfresh = tsfresh['Appliances']
X_tsfresh.head()
```

Out[70]:

	date	hours	T1	RH_1	RH_5	RH_6	RH_8	T_out	Windspeed	Visibility	Tdewpoint
0	2016-01-11 17:00:00	17	19.89	47.596667	55.20	84.256667	48.900000	6.600000	7.000000	63.000000	5.3
1	2016-01-11 17:10:00	17	19.89	46.693333	55.20	84.063333	48.863333	6.483333	6.666667	59.166667	5.2
2	2016-01-11 17:20:00	17	19.89	46.300000	55.09	83.156667	48.730000	6.366667	6.333333	55.333333	5.1
3	2016-01-11 17:30:00	17	19.89	46.066667	55.09	83.423333	48.590000	6.250000	6.000000	51.500000	5.0
4	2016-01-11 17:40:00	17	19.89	46.333333	55.09	84.893333	48.590000	6.133333	5.666667	47.666667	4.9

In [71]:
```
from tsfresh import extract_features
from tsfresh import select_features
from tsfresh.utilities.dataframe_functions import impute
p = tsfresh.drop('Appliances',axis=1)
```

In [72]:
```
# For Extracing Minimal Features
from tsfresh.feature_extraction import MinimalFCParameters
extracted_features = extract_features(p, column_id="date", column_sort="hours",show_warnings=False, default_fc_parameters=Minimal
```

```
Feature Extraction:   0%|          | 0/10 [00:00<?, ?it/s]
```

```
---------------------------------------------------------------------------
RemoteTraceback                           Traceback (most recent call last)
```

Part 6: Model Validation and Selection

Step 1: Selecting Random forest on basis of metrics below

```
Model Analysis:  RandomForest
METRIC INFO:
             Model   mae_test  mae_train  mape_test  mape_train   r2_test  \
0       Regression  55.335889  55.651561  67.366540   65.439100  0.082277
0       Regression  55.335889  55.651561  67.366540   65.439100  0.082277
0     RandomForest  30.922860  11.991840  30.980940   11.864655  0.580944
0   Nueral Network  55.317005  56.092054  67.682968   66.345399  0.077118

   r2_train   rms_test  rms_train
0  0.089554  95.480525  98.713194
0  0.089554  95.480525  98.713194
0  0.940003  64.520113  25.340324
0  0.075978  95.748517  99.446455
```

Part 7: Final pipeline

Created pipeline to automate the entire model from data ingestion to final model prediction

Adding estimator1: StandardScaler & LinearRegression

```
In [18]: 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)
         calc_mertic_info('Regression', gs_lr, X_train, y_train, X_test, y_test)
         print('LinearRegression completed')

         LinearRegression completed
```

Adding estimator2: StandardScaler & RandomForestRegression

```
n [22]: pipe_rf = Pipeline([('scl', StandardScaler()),('rf', RandomForestRegressor(n_estimators=115,max_features=6,random_state=42))])
        grid_params_rf = [{}]
        gs_rf = GridSearchCV(estimator=pipe_rf, param_grid=grid_params_rf, cv=10)
        gs_rf.fit(X_train, y_train)
        calc_mertic_info('RandomForest', gs_rf, X_train, y_train, X_test, y_test)
        print('RandomForest completed')

        RandomForest completed
```

Adding estimator3: Neural Network Regression

```
In [23]: pipe_nn = Pipeline([('min/max scaler', MinMaxScaler(feature_range=(0.0, 1.0))),
                             ('neural network', MLPRegressor(activation = 'logistic',learning_rate='adaptive',alpha=0.5))])
         grid_params_nn = [{}]
         gs_nn = GridSearchCV(estimator=pipe_nn, param_grid=grid_params_nn, cv=10)
         gs_nn.fit(X_train, y_train)
         calc_mertic_info('Nueral Network', gs_nn, X_train, y_train, X_test, y_test)
         print('Neural Network completed')
```

Exporting Regression metrics ¶

```
In [24]: optimum_model =  min(rmse_dict.items(),key=operator.itemgetter(1))[0]
         print('Model Analysis: ', optimum_model)

         print('METRIC INFO:')
         print(mertic_info)

         mertic_info.to_csv('Metric_Info.csv')

         Model Analysis:  RandomForest
         METRIC INFO:
                     Model   mae_test  mae_train  mape_test  mape_train   r2_test  \
         0      Regression  55.335889  55.651561  67.366540   65.439100  0.082277
         0      Regression  55.335889  55.651561  67.366540   65.439100  0.082277
         0    RandomForest  30.922860  11.991840  30.980940   11.864655  0.580944
         0  Nueral Network  55.317005  56.092054  67.682968   66.345399  0.077118

            r2_train   rms_test  rms_train
         0  0.089554  95.480525  98.713194
         0  0.089554  95.480525  98.713194
         0  0.940003  64.520113  25.340324
         0  0.075978  95.748517  99.446455
```