Advances in Data Science

Assignment 2

Report by:

Abhinav Tiwari

Nilesh Nerkar

Dharit Shah

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

Introduction:

As interest in IOT and sensors pick up steam, companies are trying to build algorithms and systems to understand consumer behavior to help them make better decisions. One such application is energy modeling. Though, most consumers are aware of their aggregate consumption of energy, few are aware of how and where energy is consumed. With increasing sensors in equipment, it is becoming easier to find out which equipment/instruments consume the most power.

The energy (Wh) data logged every 10 min for the appliances is the focus of this analysis. The 10 min reporting interval was chosen to be able to capture quick changes in energy consumption.

Data used include measurements of temperature and humidity sensors from a wireless network, weather from a nearby airport station and recorded energy use of lighting fixtures.

The wire-less sensor network's temperature and humidity recordings were averaged for the corresponding 10 min periods and merged with the energy data set by date and time.

The time span of the data set is 137 days (4.5 months). Fig. 1 shows the energy consumption profile for the period. The energy consumption profile shows a high variability. Fig. 2 shows a histogram of the data.

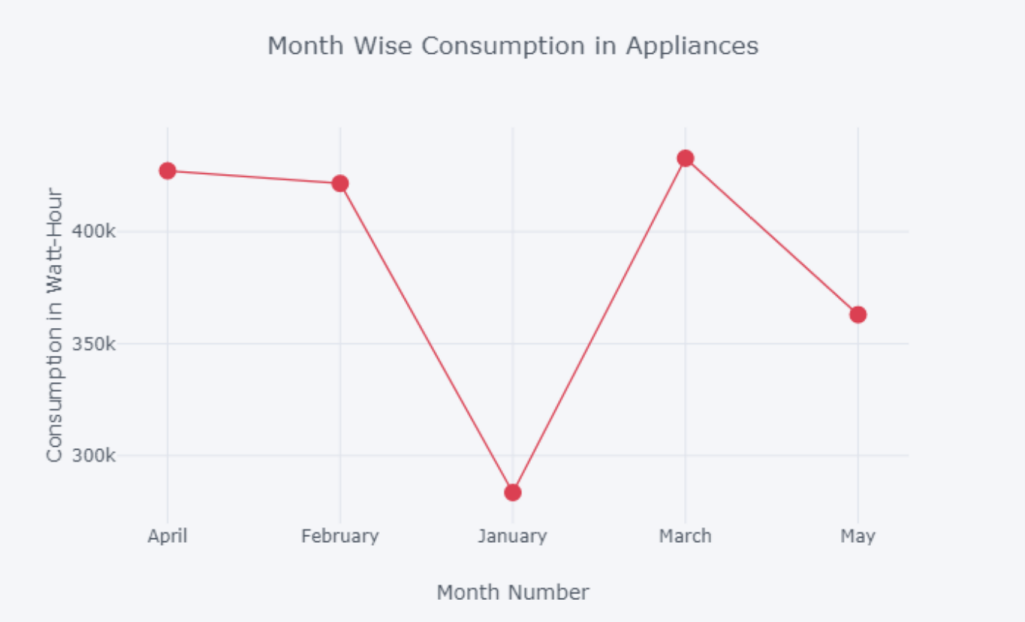


Fig1. Energy Consumption Per Month

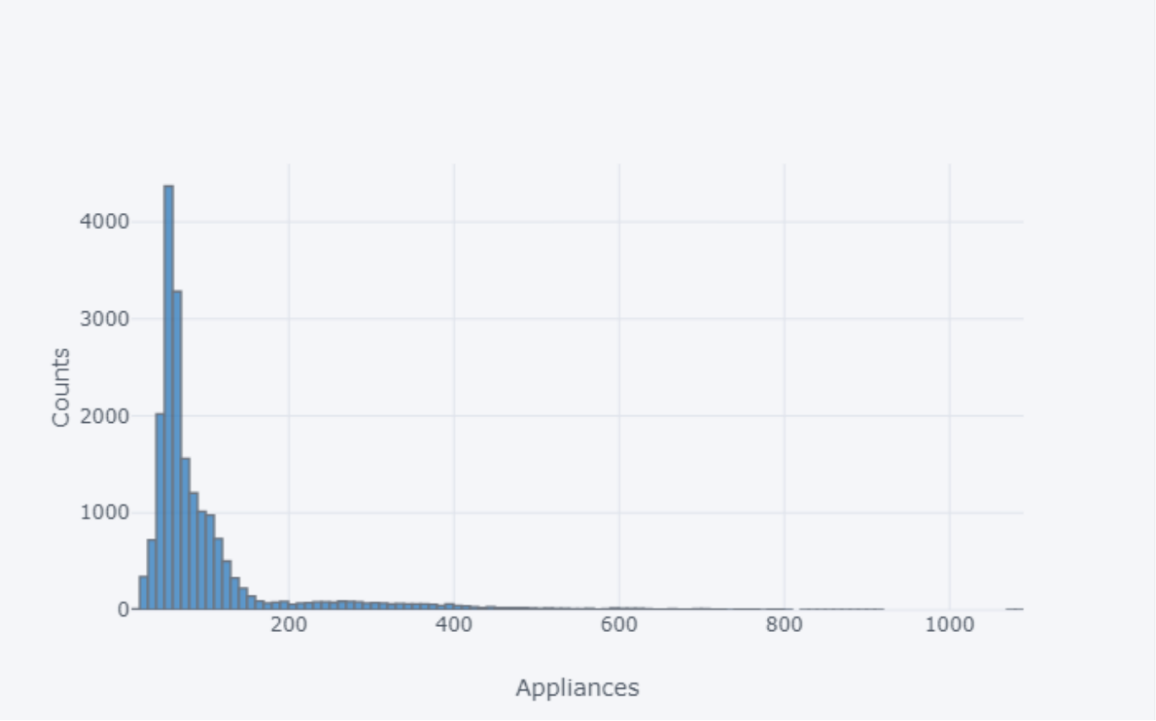


Fig2. Total Energy Consumption

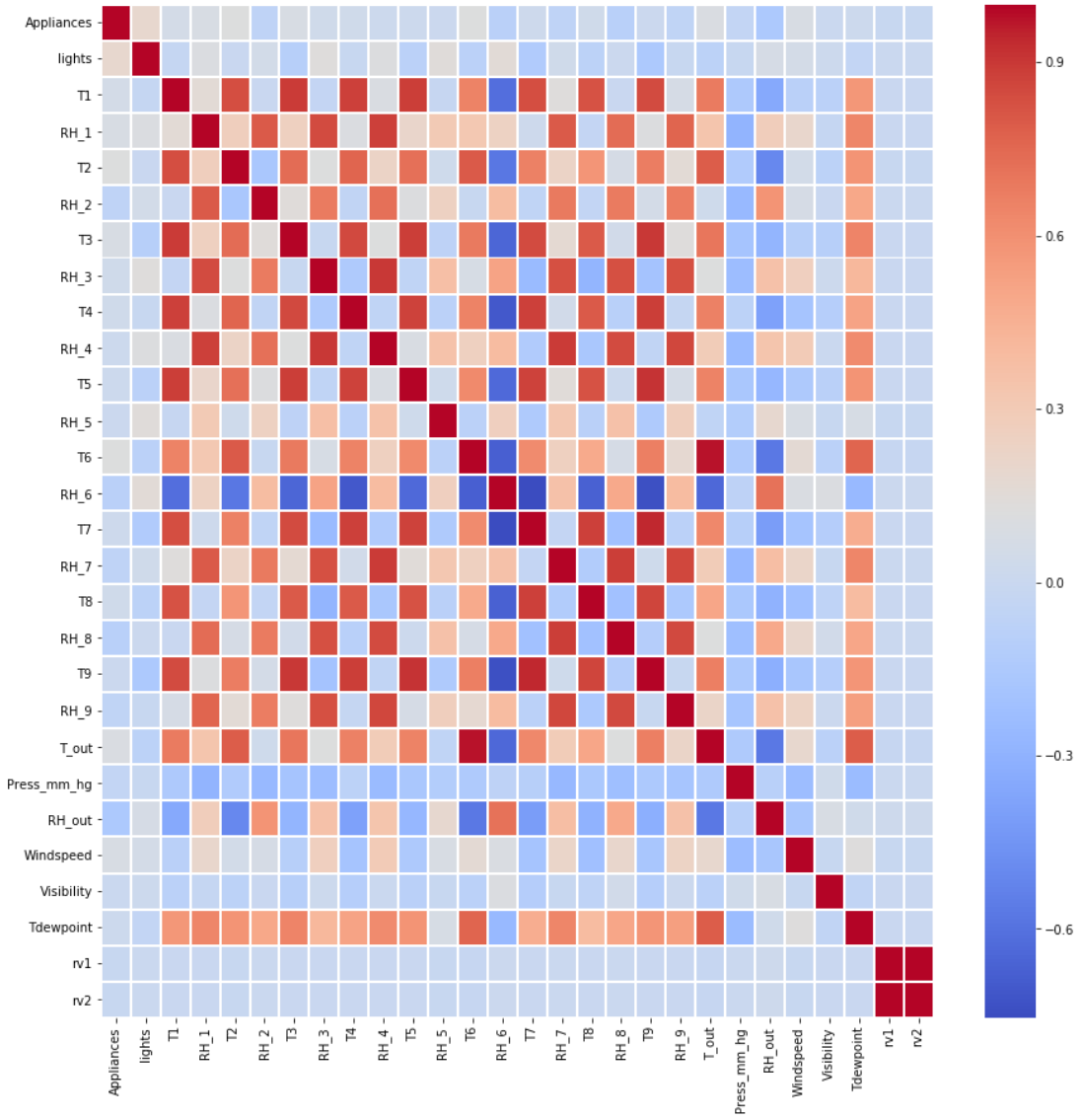


Fig 3. Correlation Matrix

Above correlation matrix heatmap shows that there is a positive correlation between the energy consumption of appliances and lights. The second largest correlation is between appliances and T2. For the indoor temperatures, the correlations are high as expected, since the ventilation is driven by the HRV unit and minimizes air temperature differences between rooms. For example, a positive correlation is found with T1 and T3. The plot shows that the highest correlation with the appliances is between the outdoor temperature. There is also a negative correlation between the appliances and outdoor humidity/RH6. It also shows positive correlations between the consumption of appliances and T7, T8 and T9 very low.

Section 2: Feature engineering and Feature Selection

2.1: Feature Selection Using Boruta

Since the dataset contains several features or parameters and considering that the airport weather station is not at the same location as the house, it is desirable to find out which parameters are the most important and which ones do not improve the prediction of the appliances' energy consumption. For this task the Boruta package is used to select all the relevant variables.

1. BORUTA

In [10]:
```
X_boruta = df_1
X_boruta = X_boruta.drop(['Appliances'],axis=1)
X_boruta = X_boruta.values
```

In [11]:
```
y_boruta = df_1
y_boruta = y_boruta['Appliances']
y_boruta = y.values
```

In [12]:
```
# NOTE BorutaPy accepts numpy arrays only, hence the .values attribute
y_boruta = y_boruta.ravel()

# define random forest classifier, with utilising all cores and
# sampling in proportion to y labels
rf1 = RandomForestRegressor(n_jobs=-1, max_depth=5)

# define Boruta feature selection method
feat_selector = BorutaPy(rf1, n_estimators='auto', verbose=5, random_state=1,max_iter=50)

# find all relevant features - 5 features should be selected
feat_selector.fit(X_boruta,y_boruta)

# check selected features - first 5 features are selected
feat_selector.support_

# check ranking of features
feat_selector.ranking_
```

```
Iteration:      1 / 50
Confirmed:      0
Tentative:      42
Rejected:       0
```

In [16]:
```
rf_boruta = RandomForestRegressor(bootstrap=True, max_depth=30, n_estimators=30,n_jobs=-1)
rf_boruta.fit(X_train_boruta,y_train_boruta)
calc_error_metric('Random Forrest_Boruta',rf_boruta, X_train_boruta, y_train_boruta, X_test_boruta, y_test
_boruta)
```

Out[16]:

	Model	r2_train	r2_test	rms_train	rms_test	mae_train	mae_test	mape_train	mape_test
0	Random Forrest_Boruta	0.927821	0.531899	27.660386	69.445427	12.773232	32.990377	12.546246	32.933694

We have also used Sequential Feature Selector for selecting the relevant features using Forward, Backward and Exhaustive Search.

```
In [13]: rf_fwd = RandomForestRegressor(n_estimators=30, max_depth=30)
         rf_fwd.fit(X_train.iloc[:, feat_cols], y_train)
```

```
Out[13]: RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=30,
                    max_features='auto', max_leaf_nodes=None,
                    min_impurity_decrease=0.0, min_impurity_split=None,
                    min_samples_leaf=1, min_samples_split=2,
                    min_weight_fraction_leaf=0.0, n_estimators=30, n_jobs=1,
                    oob_score=False, random_state=None, verbose=0, warm_start=False)
```

```
In [21]: calc_error_metric('Random Forrest FWD Selection',rf_fwd,X_train.iloc[:, feat_cols],y_train,X_test.iloc[:,f
         eat_cols],y_test)
```

Out[21]:

	Model	r2_train	r2_test	rms_train	rms_test	mae_train	mae_test	mape_train	mape_test
0	Random Forrest FWD Selection	0.935977	0.575913	25.871247	67.179993	11.95273	31.081771	11.704076	30.192527

```
In [33]: rf_bw = RandomForestRegressor(n_estimators=30, max_depth=30)
         rf_bw.fit(X_train.iloc[:, feat_cols_bw], y_train)
```

```
Out[33]: RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=30,
                    max_features='auto', max_leaf_nodes=None,
                    min_impurity_decrease=0.0, min_impurity_split=None,
                    min_samples_leaf=1, min_samples_split=2,
                    min_weight_fraction_leaf=0.0, n_estimators=30, n_jobs=1,
                    oob_score=False, random_state=None, verbose=0, warm_start=False)
```

```
In [34]: calc_error_metric('Random Forrest BWD Selection',
                           rf_bw,X_train.iloc[:, feat_cols_bw],y_train,
                           X_test.iloc[:,feat_cols_bw],y_test)
```

Out[34]:

	Model	r2_train	r2_test	rms_train	rms_test	mae_train	mae_test	mape_train	mape_test
0	Random Forrest BWD Selection	0.935673	0.561712	25.932621	68.295498	12.082505	31.644982	11.917106	30.582572

Linear Regressor with Exhaustive Search

```
In [37]: lr1 = LinearRegression()
```

```
In [15]: # Build RF classifier to use in feature selection
         #rf1 = LinearRegression( n_jobs=-1)
```

```
In [38]: #Build ExhaustiveFeatureSelector
         efs = EFS(lr1,
                   min_features=3,
                   max_features=4,
                   scoring='neg_mean_squared_error',
                   cv=0)
```

```
In [39]: efs = efs.fit(X_train.values, y_train)
         Features: 58905/58905
```

```
In [13]: #print('Selected features:', efs.best_idx_)
         Selected features: (0, 2, 4, 16)
```

```
In [43]: feat_cols = list(efs.best_idx_)
         print(feat_cols)
         [0, 2, 4, 16]
```

```
In [45]: lr1.fit(X_train.iloc[:, feat_cols], y_train)
Out[45]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

```
In [46]: pred = lr1.predict(X_test.iloc[:,feat_cols])
```

```
In [47]: r2_score(y_test,pred)
Out[47]: 0.11544846707140755
```

2.2: TPOT

We also tried implementing TPOT for getting the relevant features and optimized model.

3. TPOT

In [25]:
```
tpot = TPOTRegressor(generations=2, population_size=50,
                     offspring_size=None,
                     mutation_rate=0.9,
                     verbosity=3,cv=2,n_jobs=-1)

tpot.fit(X_train, y_train)
tpot.export('tpot_energy_pipeline.py')
```

```
Warning: xgboost.XGBRegressor is not available and will not be used by TPOT.
28 operators have been imported by TPOT.
HBox(children=(IntProgress(value=0, description='Optimization Progress', max=150), HTML(value='')))
_pre_test decorator: _random_mutation_operator: num_test=0 X contains negative values.
_pre_test decorator: _random_mutation_operator: num_test=0 __init__() got an unexpected keyword argument
'max_depth'
_pre_test decorator: _random_mutation_operator: num_test=0 __init__() got an unexpected keyword argument
'max_depth'
_pre_test decorator: _random_mutation_operator: num_test=0 Unsupported set of arguments: The combination o
f penalty='l2' and loss='epsilon_insensitive' are not supported when dual=False, Parameters: penalty='l2',
loss='epsilon_insensitive', dual=False
Generation 1 - Current Pareto front scores:
-1      -5847.517134304666      RandomForestRegressor(input_matrix, RandomForestRegressor__bootstrap=Fals
e, RandomForestRegressor__max_features=0.3, RandomForestRegressor__min_samples_leaf=1, RandomForestRegress
or__min_samples_split=11, RandomForestRegressor__n_estimators=100)

_pre_test decorator: _random_mutation_operator: num_test=0 Found array with 0 feature(s) (shape=(50, 0)) w
hile a minimum of 1 is required.
_pre_test decorator: _random_mutation_operator: num_test=0 Expected n_neighbors <= n_samples,  but n_sampl
es = 50, n_neighbors = 73
_pre_test decorator: _random_mutation_operator: num_test=0 __init__() got an unexpected keyword argument
'max_depth'
Pipeline encountered that has previously been evaluated during the optimization process. Using the score f
rom the previous evaluation.
Generation 2 - Current Pareto front scores:
-1      -5847.517134304666      RandomForestRegressor(input_matrix, RandomForestRegressor__bootstrap=Fals
e, RandomForestRegressor__max_features=0.3, RandomForestRegressor__min_samples_leaf=1, RandomForestRegress
or__min_samples_split=11, RandomForestRegressor__n_estimators=100)
```

Out[25]:
```
True
```

In [26]:
```
print('The RMSE of TPOT Regressor is {}'.format(tpot.score(X_test,y_test)))
```

```
The RMSE of TPOT Regressor is -4338.882742575104
```

2.3: Using Featuretools

It is a framework to perform automated feature engineering. It excels at transforming transactional and relational datasets into feature matrices for machine learning.

In [30]:
```
phase_featuretools = df_feature.groupby(['date','month','dayInWeek','hourOfDay','dayOfMonth','minute','per
iod']).mean()
phase_featuretools = phase_featuretools.reset_index()
phase_featuretools['dayInWeek']=phase_featuretools['dayInWeek'].apply(int)
phase_featuretools['hourOfDay']=phase_featuretools['hourOfDay'].apply(int)
phase_featuretools['minute']=phase_featuretools['minute'].apply(int)
phase_featuretools['period']=phase_featuretools['period'].apply(int)
phase_featuretools['date'] = pd.to_datetime(phase_featuretools['date'])
phase_featuretools['Press_mm_hg'] = np.log(phase_featuretools['Press_mm_hg'])
phase_featuretools['Visibility'] = np.log(phase_featuretools['Visibility'])
```

In [31]:
```
y_featuretools = phase_featuretools[['date','Appliances']]
X_featuretools = phase_featuretools.drop(['Appliances'],axis=1)
entities ={"appliances" :(y_featuretools,"date"),
           "rest" :(X_featuretools,"date")}
relationships = [("appliances","date","rest","date")]
```

In [32]:
```
feature_matrix_app,features_defs = ft.dfs(entities=entities,relationships=relationships,target_entity="app
liances")
```

In [33]:
```
features_defs
```

Out[33]:
```
[<Feature: Appliances>,
 <Feature: SUM(rest.month)>,
 <Feature: SUM(rest.dayInWeek)>,
 <Feature: SUM(rest.hourOfDay)>,
 <Feature: SUM(rest.dayOfMonth)>,
 <Feature: SUM(rest.minute)>,
 <Feature: SUM(rest.period)>,
 <Feature: SUM(rest.lights)>,
 <Feature: SUM(rest.T1)>,
 <Feature: SUM(rest.RH_1)>,
 <Feature: SUM(rest.T2)>,
 <Feature: SUM(rest.RH_2)>,
 <Feature: SUM(rest.T3)>,
 <Feature: SUM(rest.RH_3)>,
 <Feature: SUM(rest.T4)>,
 <Feature: SUM(rest.RH_4)>,
 <Feature: SUM(rest.T5)>,
 <Feature: SUM(rest.RH_5)>,
 <Feature: SUM(rest.T6)>,
 <Feature: SUM(rest.RH_6)>,
 <Feature: SUM(rest.T7)>,
 <Feature: SUM(rest.RH_7)>,
 <Feature: SUM(rest.T8)>,
 <Feature: SUM(rest.RH_8)>,
 <Feature: SUM(rest.T9)>,
 <Feature: SUM(rest.RH_9)>,
 <Feature: SUM(rest.T_out)>,
```

2.4: Tsfresh

2. TSFRESH

In [17]:
```
display(HTML(df_1.head().to_html()))
```

	Appliances	lights	T1	RH_1	T2	RH_2	T3	RH_3	T4	RH_4	T5	RH_5	T6
0	60	30	19.89	47.596667	19.2	44.790000	19.79	44.730000	19.000000	45.566667	17.166667	55.20	7.02666
1	60	30	19.89	46.693333	19.2	44.722500	19.79	44.790000	19.000000	45.992500	17.166667	55.20	6.83333
2	50	30	19.89	46.300000	19.2	44.626667	19.79	44.933333	18.926667	45.890000	17.166667	55.09	6.56000
3	50	40	19.89	46.066667	19.2	44.590000	19.79	45.000000	18.890000	45.723333	17.166667	55.09	6.43333
4	60	40	19.89	46.333333	19.2	44.530000	19.79	45.000000	18.890000	45.530000	17.200000	55.09	6.36666

In [18]:
```
ts_df = df_1
```

In [19]:
```
ts_df = ts_df.reset_index()
ts_df['date'] = df['date']
```

In [20]:
```
X_tsfresh = ts_df.drop('Appliances',axis=1)
y_tsfresh = ts_df['Appliances']
y_tsfresh.index = ts_df['date']
```

In [21]:
```
extracted_features = extract_features(ts_df, column_id="date",
                                      show_warnings=False, default_fc_parameters=MinimalFCParameters())
```

```
Feature Extraction: 100%|██████████| 10/10 [02:24<00:00, 11.96s/it]
```

In [22]:
```
impute(extracted_features)
features_filtered = select_features(extracted_features,y_tsfresh)
```

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

In [23]:
```
X_from_tsfresh = features_filtered

X_train_ts, X_test_ts, y_train_ts, y_test_ts = train_test_split(X_from_tsfresh, y_tsfresh,
                                                                test_size=0.30)
```

In [24]:
```
rf_ts = RandomForestRegressor(bootstrap=True, max_depth=30, n_estimators=30,n_jobs=-1)
rf_ts.fit(X_train_ts,y_train_ts)
calc_error_metric('Random Forrest_TSFresh',rf_ts, X_train_ts, y_train_ts, X_test_ts, y_test_ts)
```

Out[24]:

	Model	r2_train	r2_test	rms_train	rms_test	mae_train	mae_test	mape_train	mape_test
0	Random Forrest_TSFresh	0.999944	0.999325	0.769781	2.669877	0.018508	0.069414	0.002205	0.007665

Section 3: Prediction algorithms

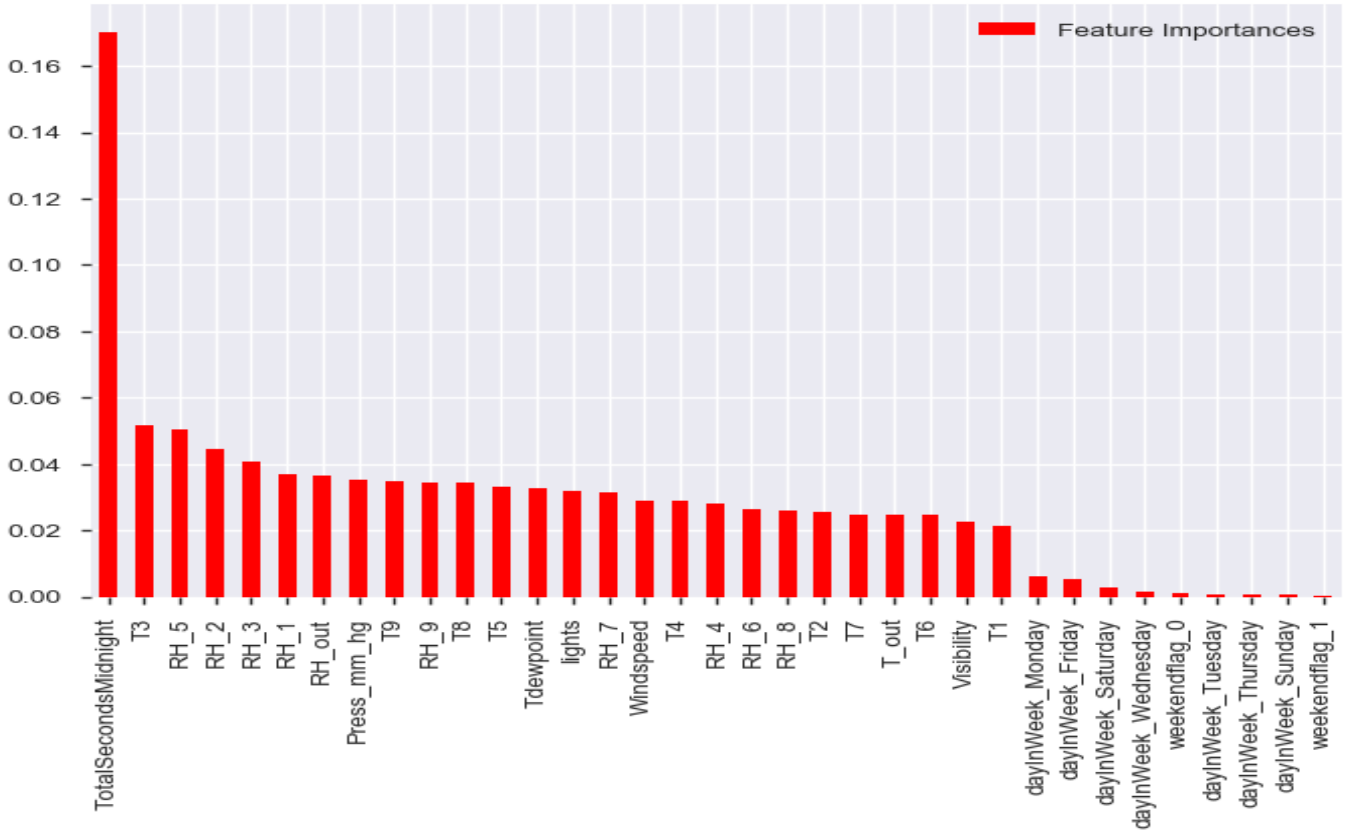
3.1: Linear Regression

The linear regression uses all the available predictors and finds the appropriate slope quantifying the effect of each predictor and the response.

Fig4. shows a residual plot for the linear regression model. The residuals were computed as the difference between the real values and the predicted values. It is obvious that the relationship between the variables and the energy consumption of appliances is not well represented by the linear model since the residuals are not normally distributed around the horizontal axis. To compare the performance of each of the regression models, different performance evaluation indices are used here: the root mean squared error (RMSE), the coefficient of determination or R-squared/R2, the mean absolute error (MAE) and the mean absolute percentage error (MAPE):

3.2: Random Forest

The random forest model is based on the output of several regression trees. However, each tree is built with a random sample of selected predictors. The idea behind this is to decorrelate the trees and improve the prediction. The random forest model requires finding the optimum number of trees and the number of randomly selected predictors.



3.3 Neural Networks

Class MLPRegressor implements a multi-layer perceptron (MLP) that trains using backpropagation with no activation function in the output layer, which can also be seen as using the identity function as activation function. Therefore, it uses the square error as the loss function, and the output is a set of continuous values. **MLPRegressor** uses parameter alpha for regularization (L2 regularization) term which helps in avoiding overfitting by penalizing weights with large magnitudes.

The error metrics for all the regression models are as follows:

	Model	mae_test	mae_train	mape_test	mape_train	r2_test	r2_train	rms_test	rms_train
0	LR	0.077071019	0.048970402	inf	inf	0.112461485	0.171897631	0.113007557	0.086957566
0	RF	0.054201379	0.01275159	inf	inf	0.283057511	0.910284879	0.101567799	0.028621882
0	NN	0.072912812	0.055380462	inf	inf	0.144665576	0.185775067	0.110938389	0.086225865
0	Tuned NN Regressor	0.062338774	0.048888068	inf	inf	0.195842966	0.218646403	0.107568302	0.084467408
0	Tuned RF Regressor	0.049123164	0.011973416	inf	inf	0.380414297	0.928983112	0.094420126	0.025465147
0	No Light Linear Model	0.077437148	0.049702081	inf	inf	0.106226732	0.151344225	0.113403788	0.08803009
0	No Light RF Model	0.04584902	0.011179313	inf	inf	0.443007171	0.93649858	0.089523836	0.024080033
0	No Light NN Model	0.072537119	0.058292962	inf	inf	0.167269561	0.16877004	0.109462686	0.087121622
0	No Weather & Light Linear Model	0.078170596	0.04992674	inf	inf	0.098191822	0.143801357	0.113912391	0.088420431
0	No Weather & Light RF Model	0.048158691	0.011390559	inf	inf	0.420727221	0.933985843	0.091296776	0.024551832
0	No Weather & Light NN Model	0.05889385	0.043600341	inf	inf	0.190399038	0.184720731	0.107931792	0.086281674
0	No Outside T & RH Linear Model	0.059645949	0.050956359	inf	inf	0.062130984	0.094869269	0.116167589	0.090911959
0	No Outside T & RH RF Model	0.041476741	0.011970333	inf	inf	0.444723975	0.92800481	0.089385761	0.025639946
0	No Outside T & RH NN Model	0.063217884	0.058825214	inf	inf	0.150010925	0.142218447	0.110591196	0.088502128
0	Weather & Time Only Linear Model	0.058298065	0.051950196	inf	inf	0.034357141	0.064386819	0.11787512	0.092430122
0	Weather & Time Only RF Model	0.038425807	0.011365101	inf	inf	0.496434599	0.935334351	0.085122001	0.024299771
0	Weather & Time Only NN Model	0.055923372	0.04679305	inf	inf	0.072266361	0.141606346	0.115538183	0.088533699

Section 4: Model Validation and Selection

After training the models, we have calculated RMSE and R2. The best models are the ones that provide the lower RMSE and highest R2 values. As can be seen the Neural Networks and Linear Regression have very similar performance based on their RMSE and R-squared values. The Random Forest model shows a significant reduction of the RMSE compared to the linear regression model. Above snippet presents the performance of the trained models.

	Model	mae_test	mae_train	mape_test	mape_train	r2_test	r2_train	rms_test	rms_train
0	LR	0.077071019	0.048970402	inf	inf	0.112461485	0.171897631	0.113007557	0.086957566
0	RF	0.054201379	0.01275159	inf	inf	0.283057511	0.910284879	0.101567799	0.028621882
0	NN	0.072912812	0.055380462	inf	inf	0.144665576	0.185775067	0.110938389	0.086225865
0	Tuned NN Regressor	0.062338774	0.048888068	inf	inf	0.195842966	0.218646403	0.107568302	0.084467408
0	Tuned RF Regressor	0.049123164	0.011973416	inf	inf	0.380414297	0.928983112	0.094420126	0.025465147
0	No Light Linear Model	0.077437148	0.049702081	inf	inf	0.106226732	0.151344225	0.113403788	0.08803009
0	No Light RF Model	0.04584902	0.011179313	inf	inf	0.443007171	0.93649858	0.089523836	0.024080033
0	No Light NN Model	0.072537119	0.058292962	inf	inf	0.167269561	0.16877004	0.109462686	0.087121622
0	No Weather & Light Linear Model	0.078170596	0.04992674	inf	inf	0.098191822	0.143801357	0.113912391	0.088420431
0	No Weather & Light RF Model	0.048158691	0.011390559	inf	inf	0.420727221	0.933985843	0.091296776	0.024551832
0	No Weather & Light NN Model	0.05889385	0.043600341	inf	inf	0.190399038	0.184720731	0.107931792	0.086281674
0	No Outside T & RH Linear Model	0.059645949	0.050956359	inf	inf	0.062130984	0.094869269	0.116167589	0.090911959
0	No Outside T & RH RF Model	0.041476741	0.011970333	inf	inf	0.444723975	0.92800481	0.089385761	0.025639946
0	No Outside T & RH NN Model	0.063217884	0.058825214	inf	inf	0.150010925	0.142218447	0.110591196	0.088502128
0	Weather & Time Only Linear Model	0.058298065	0.051950196	inf	inf	0.034357141	0.064386819	0.11787512	0.092430122
0	Weather & Time Only RF Model	0.038425807	0.011365101	inf	inf	0.496434599	0.935334351	0.085122001	0.024299771
0	Weather & Time Only NN Model	0.055923372	0.04679305	inf	inf	0.072266361	0.141606346	0.115538183	0.088533699

Section 5: Final pipeline

We have created a pipeline to automate the entire model from data ingestion to final model prediction.

In [12]:
```
pipe_lr = Pipeline([('scl', StandardScaler()),
                    ('clf', LinearRegression(normalize=True))])
grid_params_lr = [{}]
gs_lr = GridSearchCV(estimator=pipe_lr, param_grid=grid_params_lr, cv=5)
gs_lr.fit(X_train, y_train)
calc_error_metric('Regression', gs_lr, X_train, y_train, X_test, y_test)
print('Regression Pipeline Finished')


pipe_rf = Pipeline([('scl', StandardScaler()),
                    ('rf', RandomForestRegressor(n_estimators=30,max_depth=30))])
grid_params_rf = [{}]
gs_rf = GridSearchCV(estimator=pipe_rf, param_grid=grid_params_rf, cv=5)
gs_rf.fit(X_train, y_train)
calc_error_metric('RandomForest', gs_rf, X_train, y_train, X_test, y_test)
print('Random Forrest Pipeline Finished')


pipe_nn = Pipeline([('min/max scaler', MinMaxScaler()),
                    ('neural network', MLPRegressor(activation='relu',
                                                    alpha=0.05, learning_rate='constant',solver='adam'))])
grid_params_nn = [{}]
gs_nn = GridSearchCV(estimator=pipe_nn, param_grid=grid_params_nn, cv=5)
gs_nn.fit(X_train, y_train)
calc_error_metric('Nueral Network', gs_nn, X_train, y_train, X_test, y_test)
print('Neural Network Pipeline Finished')
```

```
Regression Pipeline Finished
Random Forrest Pipeline Finished
Neural Network Pipeline Finished
```