Advances in Data Science

Assignment 2&3

Report by:

Ann Sara Sajee

Gunjan Lalwani

Rishabh Jain

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.

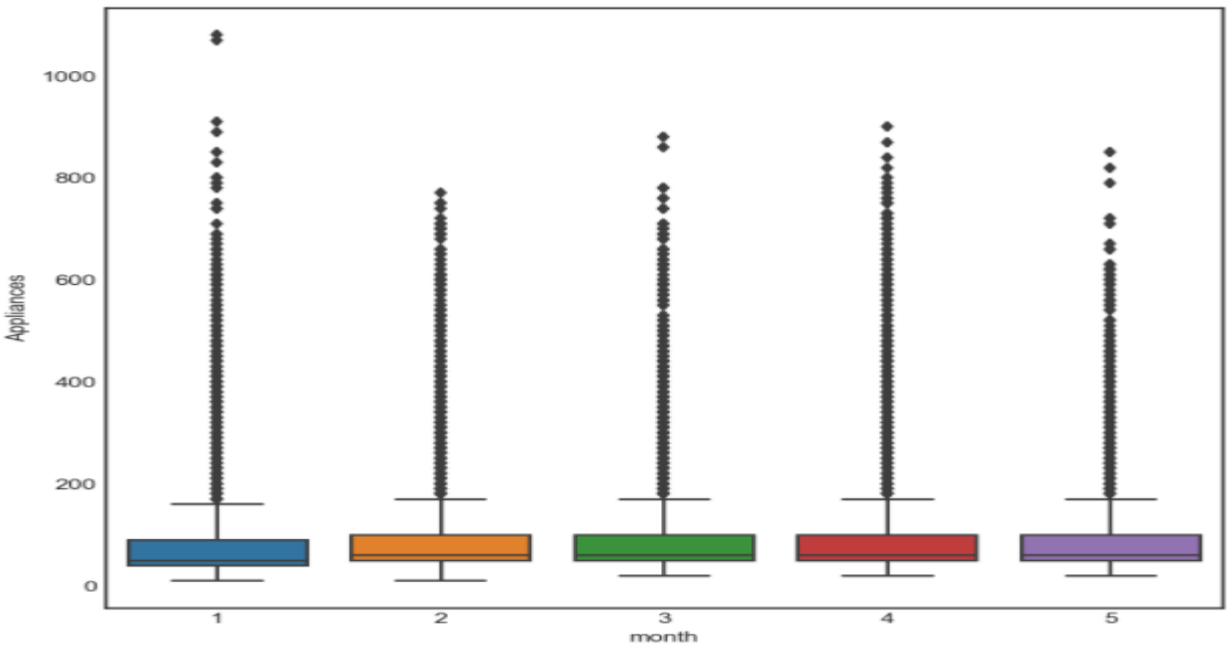

Fig1. Energy Consumption Per Month
(1: Jan 2: Feb 3: March 4: April 5: May)

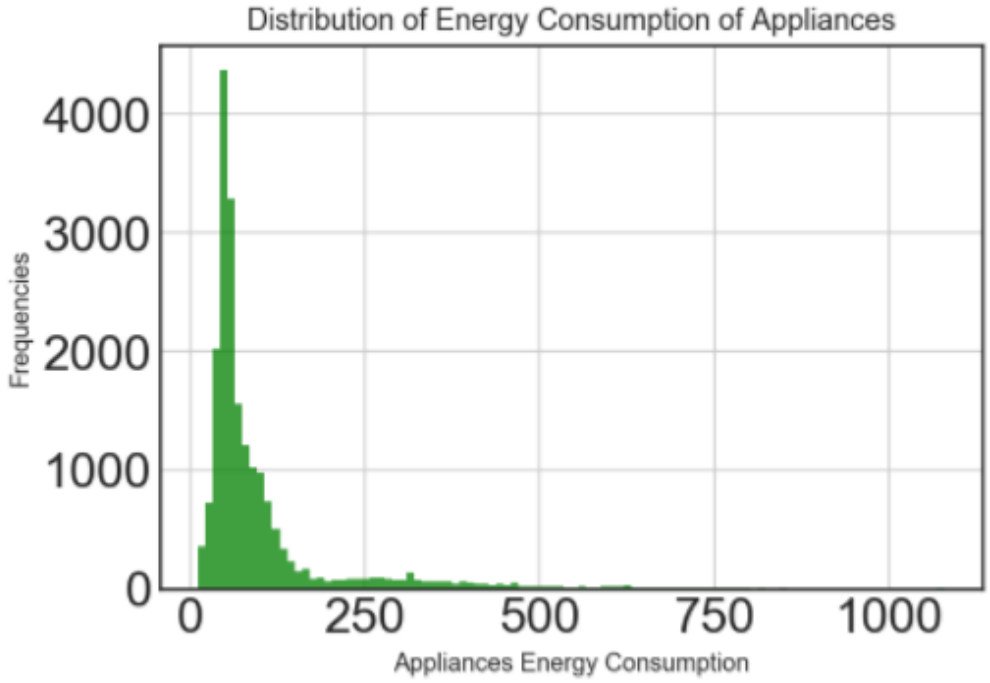

Fig2. Total Energy Consumption

As can be seen the data distribution has a long tail. In the boxplot, the median is represented with a thick black line inside the rectangles, and has a value of 60 Wh. The lower whisker has a value of 10 Wh and the upper whisker has a value of 170 Wh. It also shows that the data

above the median is more dispersed and that there are several outliers (marked with the round circles above the upper whisker).

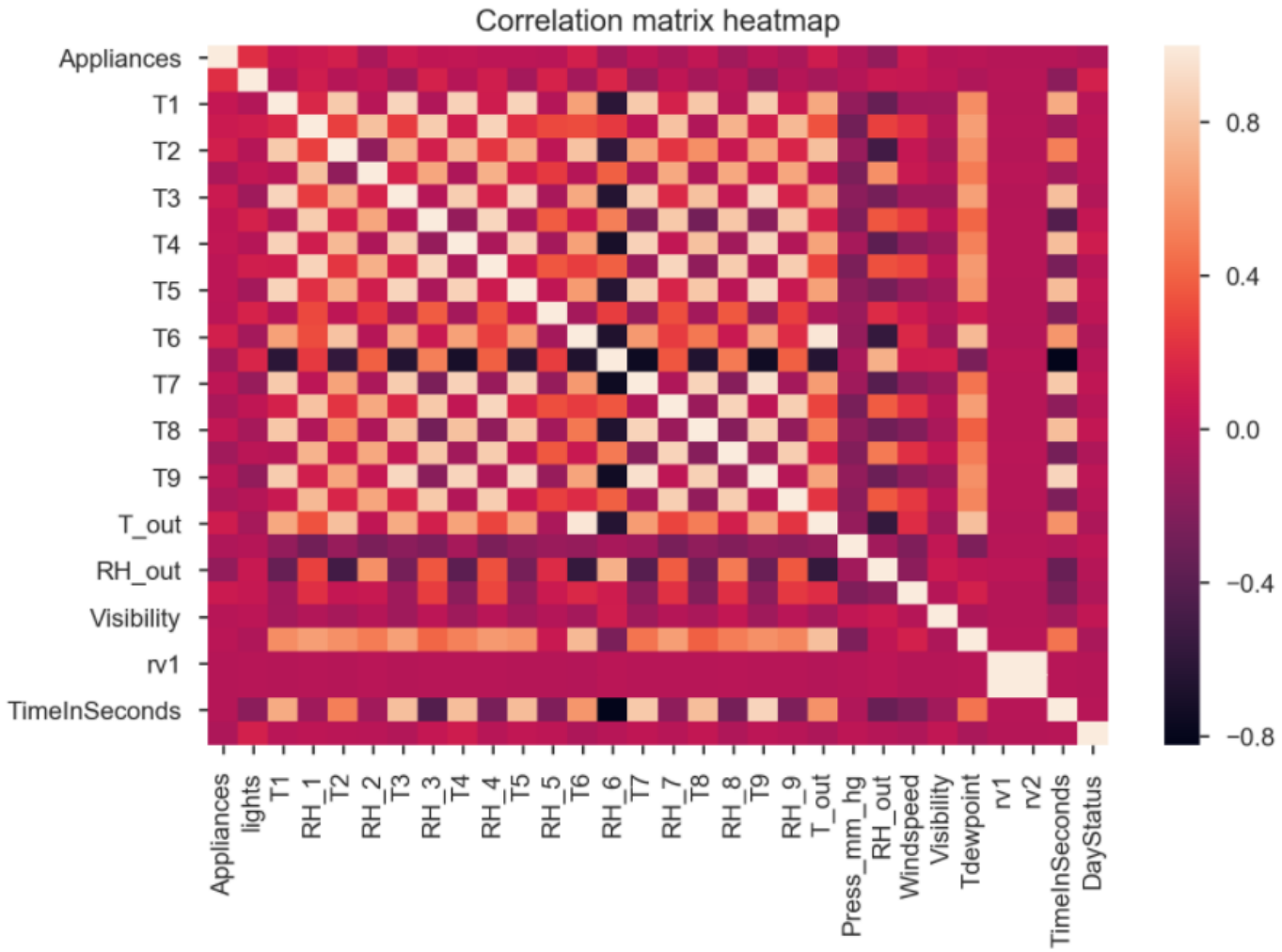

Fig 3. Correlation Matrix

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.

```
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from boruta import BorutaPy

df = pd.read_csv("energydata_complete.csv")
X = df.drop(['Appliances','date'],axis=1).values
y = df['Appliances'].values
y = y.ravel()

# load X and y
# NOTE BorutaPy accepts numpy arrays only, hence the .values attribute


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

# define Boruta feature selection method
feat_selector = BorutaPy(rf, n_estimators='auto', verbose=2, random_state=1)

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

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

# check ranking of features
feat_selector.ranking_

# call transform() on X to filter it down to selected features
X_filtered = feat_selector.transform(X)
```

```
feat_selector.get_params
```

```
<bound method BaseEstimator.get_params of BorutaPy(alpha=0.05,
     estimator=RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           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=67, n_jobs=-1,
           oob_score=False,
           random_state=<mtrand.RandomState object at 0x0000024621C64360>,
           verbose=0, warm_start=False),
     max_iter=100, n_estimators='auto', perc=100,
     random_state=<mtrand.RandomState object at 0x0000024621C64360>,
     two_step=True, verbose=2)>
```

```
feat_selector.ranking_
```

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1,
       1, 3, 1, 4, 5])
```

The feature ranking shows that r1 and r2 are the least important variables. So, we can eliminate both the features.

1. lights 2. T1 3.RH_1 4. T2 5. RH_2 6. T3 7. RH_3 8. T4 9. RH_4 10. T5 11. RH_5 12.T6 13. RH_6 14. T7	15. RH_7 16. T8 17. RH_8 18. T9 19. RH_9 20. Press_mm_hg 21. Windspeed 22. RH_out 23. Tdewpoint 24. T_out 25. Visibility 26. rv1 27. rv2

Since all the predictors can be considered relevant to minimize the RMSE, they all will be used to test three regression models (Linear Regression, Random Forest and Neural Networks).

We have also used Recursive Feature Algorithm for selecting the relevant features.

```
model = RandomForestRegressor(n_estimators=300, max_features = 11)
    # create the RFE model and select 3 attributes
rfe = RFE(model)
rfe = rfe.fit(X_train, y_train)
    # summarize the selection of the attributes
print(rfe.support_)
print(rfe.ranking_)
print(rfe.n_features_)
    #Check the accuracy of the model
rfe.score(X_train, y_train)
```

```
[ True False False  True False  True  True  True  True False False  True
  True  True  True  True  True  True False  True False  True False False
  True  True False False False False False False False False False]
[ 1  6  4  1  2  1  1  1  1  3  7  1  1  1  1  1  1  9  1  5  1 10  8
  1  1 11 13 12 16 18 19 14 15 17]
17

0.9453484492136436
```

2.2: Preprocessing using Standard Scalar

Standardization of a dataset is a common requirement for many machine learning estimators: they might behave badly if the individual feature do not more or less look like standard normally distributed data (e.g. Gaussian with 0 mean and unit variance).

Standardizing features by removing the mean and scaling to unit variance

Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set. Mean and standard deviation are then stored to be used on later data using the transform method.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
# Fit only to the training data
scaler.fit(X_train)
# Now apply the transformations to the data:
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

We have also changed some category columns to numeric and created dummy variables for some columns.

```
df['date'] = pd.to_datetime(df.date)

df['weekday'] = df['date'].dt.strftime('%A')

df['Month'] = df['date'].dt.strftime('%m').astype('int64')

df['Week_no'] = df['date'].dt.strftime('%W').astype('int64')

df['Hour_of_the_day'] = df['date'].dt.strftime('%H').astype('int64')

df['NSM'] = df['date'].dt.strftime('%H:%M:%S')
df['NSM'] = df['NSM'].str.split(':').apply(lambda x: int(x[0]) * 3600 + int(x[1]) *60 + int(x[2]))

df['WeekStatus'] = (df['date'].dt.strftime('%w').astype(int) < 5).astype('int64')

df['date'] = df['date'].dt.strftime('%Y-%m-%d %H:%M:%S')

W_Status = pd.get_dummies(df.WeekStatus,prefix='W_Status').astype('int64')
Day_W = pd.get_dummies(df.weekday, prefix = 'Dy_w').astype('int64')

df = pd.concat([df,W_Status,Day_W],axis=1)
```

2.3: TPOT

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

```
pipeline_optimizer = TPOTRegressor(generations=10, population_size=20, cv = 3,
                                   random_state=42, verbosity=2)
```

```
pipeline_optimizer.fit(X_train,y_train)
```

```
Optimization Progress:  18%|██        | 40/220 [04:58<15:25,  5.14s/pipeline]
Generation 1 - Current best internal CV score: -5462.105355916628
Optimization Progress:  27%|███       | 60/220 [16:49<31:25, 11.78s/pipeline]
Generation 2 - Current best internal CV score: -5462.105355916628
Optimization Progress:  38%|████      | 84/220 [40:09<47:38, 21.02s/pipeline]
Generation 3 - Current best internal CV score: -5462.105355916628
Optimization Progress:  47%|████▋     | 104/220 [44:18<24:14, 12.54s/pipeline]
Generation 4 - Current best internal CV score: -5462.105355916628
Optimization Progress:  56%|█████▌    | 124/220 [49:54<33:31, 20.95s/pipeline]
Generation 5 - Current best internal CV score: -5462.105355916628
Optimization Progress:  65%|██████▍   | 144/220 [56:14<27:25, 21.65s/pipeline]
Generation 6 - Current best internal CV score: -5447.387656797821
Optimization Progress:  75%|███████▍  | 164/220 [1:02:53<18:57, 20.31s/pipeline]
Generation 7 - Current best internal CV score: -5447.313903685393
Optimization Progress:  84%|████████▎ | 184/220 [1:08:37<12:10, 20.30s/pipeline]
Generation 8 - Current best internal CV score: -5447.313903685393
Optimization Progress:  93%|█████████▎| 205/220 [1:19:28<19:18, 77.26s/pipeline]
Generation 10 - Current best internal CV score: -5330.967706336275

Best pipeline: ElasticNetCV(RandomForestRegressor(input_matrix, bootstrap=False, max_features=0.15, min_samples_leaf=1, min_sam
ples_split=2, n_estimators=100), l1_ratio=0.95, tol=0.001)

TPOTRegressor(config_dict={'sklearn.linear_model.ElasticNetCV': {'l1_ratio': array([ 0.  ,  0.05,  0.1 ,  0.15,  0.2 ,  0.25,
0.3 ,  0.35,  0.4 ,
        0.45,  0.5 ,  0.55,  0.6 ,  0.65,  0.7 ,  0.75,  0.8 ,  0.85,
        0.9 ,  0.95,  1.  ]), 'tol': [1e-05, 0.0001, 0.001, 0.01, 0.1]}, 'sklearn.ensemble.ExtraT....45,
        0.5 ,  0.55,  0.6 ,  0.65,  0.7 ,  0.75,  0.8 ,  0.85,  0.9 ,
        0.95,  1.  ])}}}},
       crossover_rate=0.1, cv=3, disable_update_check=False,
       early_stop=None, generations=10, max_eval_time_mins=5,
       max_time_mins=None, memory=None, mutation_rate=0.9, n_jobs=1,
       offspring_size=20, periodic_checkpoint_folder=None,
       population_size=20, random_state=42, scoring=None, subsample=1.0,
       verbosity=2, warm_start=False)
```

```
print(pipeline_optimizer.score(X_test, y_test))
```

```
-4178.76640353
```

2.4: 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.

```
import featuretools as ft
```

```
phase_featuretools = df.groupby(['date','month','day_of_week','hour','day_number','min','period']).mean()
phase_featuretools = phase_featuretools.reset_index()
phase_featuretools['day_of_week']=phase_featuretools['day_of_week'].apply(int)
phase_featuretools['hour']=phase_featuretools['hour'].apply(int)
phase_featuretools['min']=phase_featuretools['min'].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'])
phase_featuretools = phase_featuretools.drop(['rv1','rv2'],axis=1)
```

```
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")]
```

```
feature_matrix_app,features_defs = ft.dfs(entities=entities,relationships=relationships,target_entity="appliances")
```

```
feature_matrix_app1 = feature_matrix_app
```

```
feature_matrix_app1.dtypes
```

2.4: Tsfresh

```
from tsfresh import extract_features
```

```
/Users/rishabhjain/anaconda3/lib/python3.6/site-packages/statsmodels/compat/pandas.py:56: FutureWarning: The pandas.cor
ols module is deprecated and will be removed in a future version. Please use the pandas.tseries module instead.
  from pandas.core import datetools
```

```
X_tsfresh = phase_tsfresh.drop('Appliances',axis=1)
y_tsfresh = phase_tsfresh['Appliances']
```

```
from tsfresh import extract_features
from tsfresh import select_features
from tsfresh.utilities.dataframe_functions import impute
```

```
p = phase_featuretools.drop('Appliances',axis=1)
```

```
# For Extracing Minimal Features
from tsfresh.feature_extraction import MinimalFCParameters
extracted_features = extract_features(p, column_id="date", column_sort="period",show_warnings=False, default_fc_paramet
```

```
Feature Extraction: 100%|██████████| 20/20 [00:57<00:00,  2.88s/it]
```

```
from tsfresh import select_features
from tsfresh.utilities.dataframe_functions import impute
# for extracting relevant features
impute(extracted_features)
features_filtered = select_features(extracted_features, y)
```

```
WARNING:tsfresh.feature_selection.relevance:Infered classification as machine learning task
WARNING:tsfresh.feature_selection.relevance:[test_feature_significance] Feature Press_mm_hg__length is constant
WARNING:tsfresh.feature_selection.relevance:[test_feature_significance] Feature Press_mm_hg__standard_deviation is cons
WARNING:tsfresh.feature_selection.relevance:[test_feature_significance] Feature RH_1__length is constant
WARNING:tsfresh.feature_selection.relevance:[test_feature_significance] Feature Press_mm_hg__variance is constant
WARNING:tsfresh.feature_selection.relevance:[test_feature_significance] Feature RH_2__length is constant
WARNING:tsfresh.feature_selection.relevance:[test_feature_significance] Feature RH_2__standard_deviation is constant
```

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):

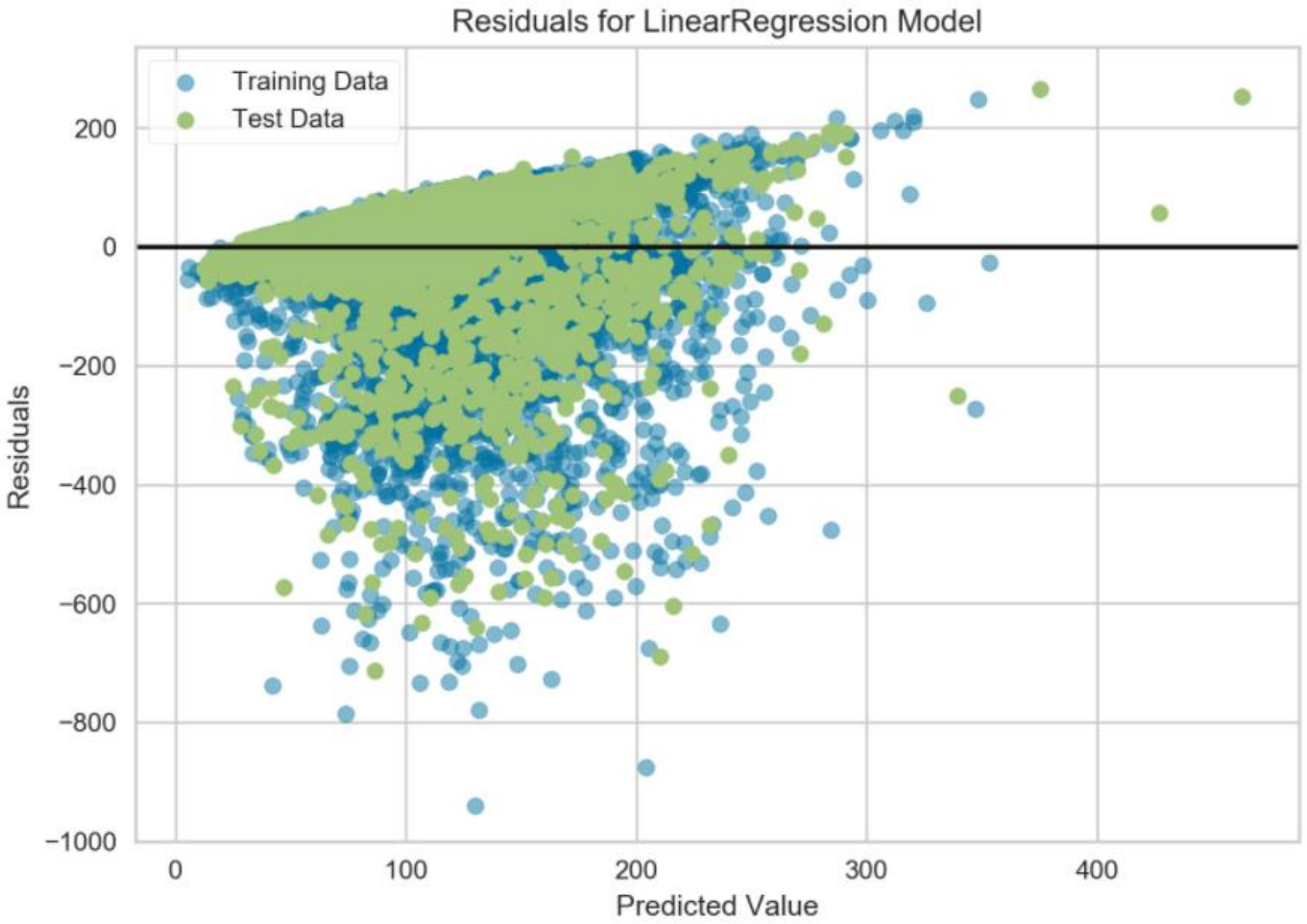

Fig 4. Residuals Plot

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. The RMSE does not appear to change after about 300 trees and the optimal number of random selected predictors is 20.

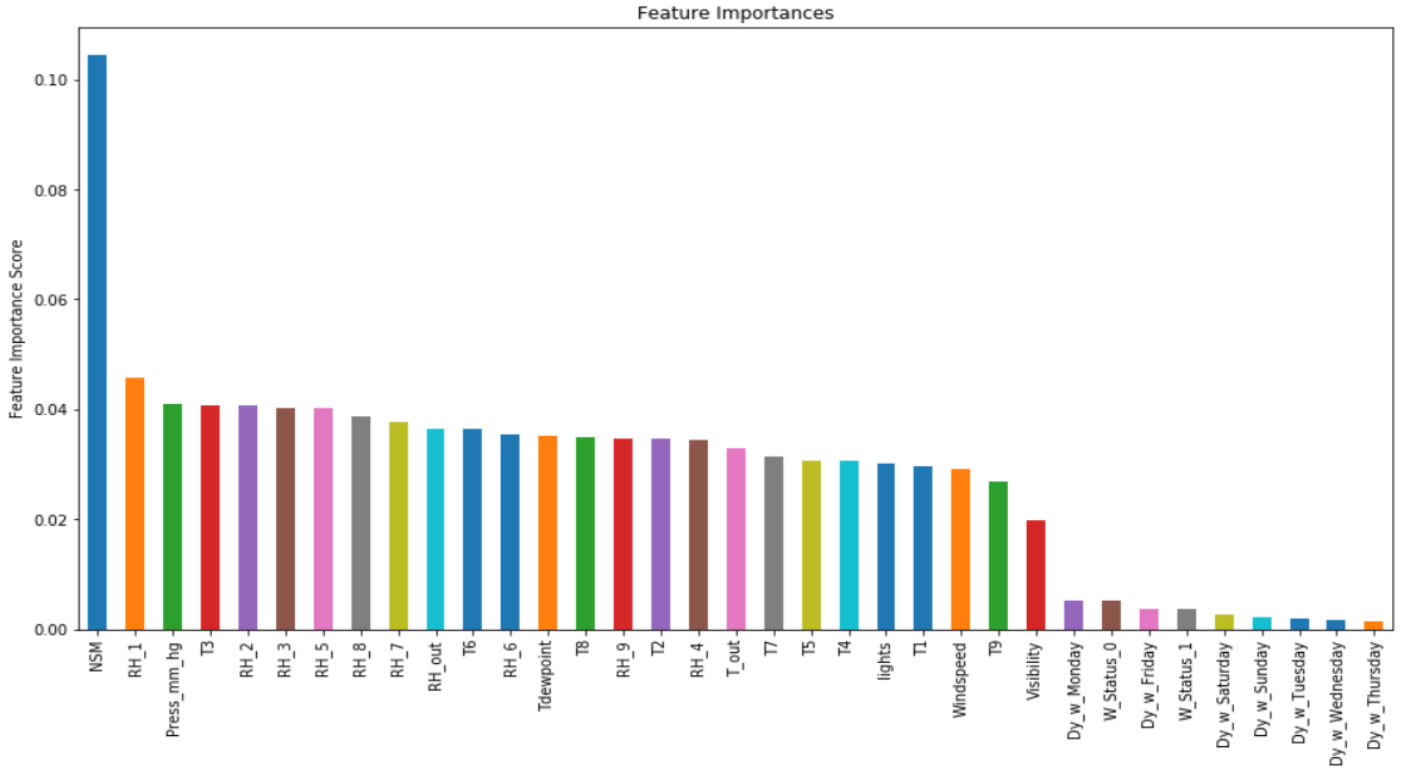

Fig 5. Random Forest Feature Importance

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:

0	Regression	52.954546	53.124709	63.023106	61.001825	0.163900	0.176196	91.135629	93.898834
0	RandomForest	29.610952	11.577408	29.320830	11.341582	0.600921	0.942823	62.963464	24.737694
0	Neural Network	52.467578	53.314667	59.486755	58.293158	0.024052	0.021077	98.462826	102.358162

The error- metrics with the features selected from Boruta and Tsfresh:

Out[15]:

	Model	ModelType	mae_test	mae_train	mape_test	mape_train	r2_test	r2_train	rms_test	rms_train
0	Boruta	LinearRegression(copy_X=True, fit_intercept=Tr...	57.710630	57.757545	70.214249	68.730210	0.058925	0.054655	98.465907	100.103478
0	Boruta	(DecisionTreeRegressor(criterion='mse', max_de...	34.833664	13.288502	35.772812	13.517528	0.497772	0.927404	71.932358	27.740307
0	Boruta	MLPRegressor(activation='relu', alpha=0.55, ba...	55.596154	55.597342	64.910656	63.420752	0.060283	0.055814	98.394836	100.042119

	Model	ModelType	mae_test	mae_train	mape_test	mape_train	r2_test	r2_train	rms_test	rms_train
0	EDA	LinearRegression(copy_X=True, fit_intercept=Tr...	54.518326	54.956444	64.909192	63.197640	0.106095	0.109963	94.233351	97.600539
0	EDA	(DecisionTreeRegressor(criterion='mse', max_de...	32.826494	12.518691	33.565539	12.433910	0.536329	0.935792	67.867805	26.214598
0	EDA	MLPRegressor(activation='relu', alpha=0.55, ba...	53.412059	53.818487	63.538577	61.588379	0.150851	0.153441	91.844065	95.186801
0	After Scaling	LinearRegression(copy_X=True, fit_intercept=Tr...	54.518326	54.956444	64.909192	63.197640	0.106095	0.109963	94.233351	97.600539
0	After Scaling	(DecisionTreeRegressor(criterion='mse', max_de...	32.894151	12.529282	33.607748	12.440774	0.533610	0.935646	68.066556	26.244289
0	After Scaling	MLPRegressor(activation='relu', alpha=0.55, ba...	53.257418	53.217531	63.330905	60.967995	0.173680	0.191225	90.601055	93.038369
0	tsfresh	LinearRegression(copy_X=True, fit_intercept=Tr...	52.871416	53.006543	62.749132	60.769515	0.159510	0.174227	91.374542	94.010984
0	tsfresh	(DecisionTreeRegressor(criterion='mse', max_de...	31.399031	12.105940	31.131029	11.842409	0.552057	0.937720	66.706842	25.817908
0	tsfresh	MLPRegressor(activation='relu', alpha=0.55, ba...	50.940304	51.569830	57.150062	55.570290	0.098663	0.095612	94.624296	98.384239
0	After Scaling_tsfresh	LinearRegression(copy_X=True, fit_intercept=Tr...	52.879459	53.050457	62.563573	60.645490	0.159194	0.173737	91.391713	94.038847
0	After Scaling_tsfresh	(DecisionTreeRegressor(criterion='mse', max_de...	31.459870	12.099251	31.165438	11.841266	0.550264	0.937809	66.840201	25.799619
0	After Scaling_tsfresh	MLPRegressor(activation='relu', alpha=0.55, ba...	48.508453	47.744067	55.495240	52.768771	0.270034	0.316146	85.155037	85.551936

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.

4.1 Evaluating the prediction with different data subsets

After finding out that the RF model provided the best RMSE and R2 in the previous analysis, this model was used to study the prediction performance with different predictors subsets: removing the light consumption, removing the light and no weather data, removing the temperature and humidity from the wireless sensor network and only using the weather and time information. See Table 1 for the considered predictors and the model's performance in the training and testing sets. From Table 1, the performance of the RF model without the lights predictor is quite accurate in comparison with the RF model in above analysis. For this model the R2 is 0.62 in the testing set. The third model in Table 1 that includes the weather data and the light predictor has an equivalent performance to the fourth model that only includes the weather data since their R2are the same.

Models	Parameters	Training				Testing			
		RMSE	R2	MAE	MAPE	RMSE	R2	MAE	MAPE
no lights	T1, RH1, T2, RH2,T3,RH3, T4,RH4,T5,RH5,T6,RH6, T7,RH7,T8,TH8,T9,RH9,To,Pressure,Rho,WindSpeed, Tdewpoint,NSM, WeekStatus, Day of Week	24.94	0.94	11.38	11.00	61.85	0.61	29.17	28.84
no lights and no weather data	T1,RH1,T2,RH2,T3,RH3, T4,RH4,T5,RH5,T6,RH6, T7,RH7,T8,TH8,T9,RH9, NSM, WeekStatus, Day of Week	24.50	0.94	11.23	10.90	61.44	0.62	28.92	28.51
no Temp and humidity inside house	Light, To,Pressure,Rho, WindSpeed,Tdewpoint,NSM, WeekStatus, Day of Week	24.28	0.94	11.47	11.56	60.40	0.63	29.26	30.07
only weather and time information	To,Pressure,Rho,WindSpeed, Tdewpoint,NSM, WeekStatus, Day of Week	24.39	0.94	11.42	11.41	60.80	0.63	29.20	29.65

Table 1.

4.2: Model Validation

All the regression models were trained with 10 fold cross validation using GridSearchCv to select the best. The below table shows all the Error metrics for testing and training data.

Models	Training				Testing			
	RMSE	R2	MAE	MAPE	RMSE	R2	MAE	MAPE
Linear Regression	93.89	0.17	53.12	61.00	91.13	0.16	52.95	63.02
Random Forest	24.94	0.94	11.61	11.36	63.54	0.59	29.87	29.66
Neural networks	96.03	0.138	53.99	61.93	92.67	0.13	53.54	63.65

Table 2.

This shows that Random forest provided the best RMSE and R2 on training dataset and testing dataset.

Section 5: Final pipeline

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

```
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_error_metric('RandomForest', gs_rf, X_train, y_train, X_test, y_test)
print('RandomForest completed')

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_error_metric('Nueral Network', gs_nn, X_train, y_train, X_test, y_test)
print('Neural Network completed')

pipe_gbm = Pipeline([('scl', StandardScaler()),('gbm', GradientBoostingRegressor(n_estimators=300,learning_rate= 0.1,max_features
grid_params_gbm =[{}]
gs_gbm = GridSearchCV(estimator=pipe_gbm, param_grid=grid_params_gbm, cv=10)
gs_gbm.fit(X_train, y_train)
calc_error_metric('GradientBoostingRegressor', gs_gbm, X_train, y_train, X_test, y_test)
print('Gradient Boosting completed')

#### Calculate best model
best_model =  min(rmse_dict.items(),key=operator.itemgetter(1))[0]
print('Best Model is ', best_model)

print('Error Metrics are:')
print(error_metric)

#### Write the error
error_metric.to_csv('Error_metrics.csv')
```