```
# Programmer Name:  Jacob Yim
# Class: CIS4321
# Programming Assignment : Final Project
# Date:12/1/21
##############################################
```

Import Libraries

```
#computational
import pandas as pd
import numpy as np

#Visualizations
import seaborn as sns
import matplotlib.pyplot as plt

#sets graph design
sns.set_style("white")

#filters warnings
import warnings

#ignore warnings
warnings.filterwarnings("ignore")

#allow matplotlib to allow graph in notebook
%matplotlib inline

#normalization and preprocessing
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.model_selection import train_test_split

#modeling
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.feature_selection import RFE

#import metrics
from sklearn import metrics
```

Import Data

```
energy = pd.read_csv('/content/energy_dataset.csv')
```

```
energy.head()
```

	time	generation biomass	generation fossil brown coal/lignite	generati foss coa derive ga
2	2015-01-01			
3	2015-01-01	438.0	254.0	0

Exploratory Data Analysis

Data Exploration

```
#number of rows and cols
energy.shape
```

```
(35064, 29)
```

```
#check for duplicates
energy[energy.duplicated()]
```

time	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	gener fossi

Notes:

- There are no duplicated rows

```
#describe the data
energy.describe()
```

	generation biomass	generation fossil brown coal/lignite	generation fossil coal-derived gas	gener fossi
count	35045.000000	35046.000000	35046.0	35046.(
mean	383.513540	448.059208	0.0	5622.7
std	85.353943	354.568590	0.0	2201.8
min	0.000000	0.000000	0.0	0.(
25%	333.000000	0.000000	0.0	4126.(
50%	367.000000	509.000000	0.0	4969.(
75%	433.000000	757.000000	0.0	6429.(
max	592.000000	999.000000	0.0	20034.(

Notes:

- There are a lot of nan values and 0s.

```
#since target varibale will be 'price actual'
energy[['price actual']].describe()
```

	price actual
count	35064.000000
mean	57.884023
std	14.204083
min	9.330000
25%	49.347500
50%	58.020000
75%	68.010000
max	116.800000

Notes:

- There are no negative prices, which is good.

```
#check out dtypes
energy.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35064 entries, 0 to 35063
Data columns (total 29 columns):
 #   Column                                       N
---  ------                                       -
 0   time                                         3
 1   generation biomass                           3
 2   generation fossil brown coal/lignite         3
 3   generation fossil coal-derived gas           3
 4   generation fossil gas                        3
 5   generation fossil hard coal                  3
```

```
 6   generation fossil oil                            3
 7   generation fossil oil shale                      3
 8   generation fossil peat                           3
 9   generation geothermal                            3
 10  generation hydro pumped storage aggregated       0
 11  generation hydro pumped storage consumption      3
 12  generation hydro run-of-river and poundage       3
 13  generation hydro water reservoir                 3
 14  generation marine                                3
 15  generation nuclear                               3
 16  generation other                                 3
 17  generation other renewable                       3
 18  generation solar                                 3
 19  generation waste                                 3
 20  generation wind offshore                         3
 21  generation wind onshore                          3
 22  forecast solar day ahead                         3
 23  forecast wind offshore eday ahead                0
 24  forecast wind onshore day ahead                  3
 25  total load forecast                              3
 26  total load actual                                3
 27  price day ahead                                  3
 28  price actual                                     3
dtypes: float64(28), object(1)
memory usage: 7.8+ MB
```

Notes:

- Dtypes look accurate.

```
#total number of null values
energy.isnull().sum()
```

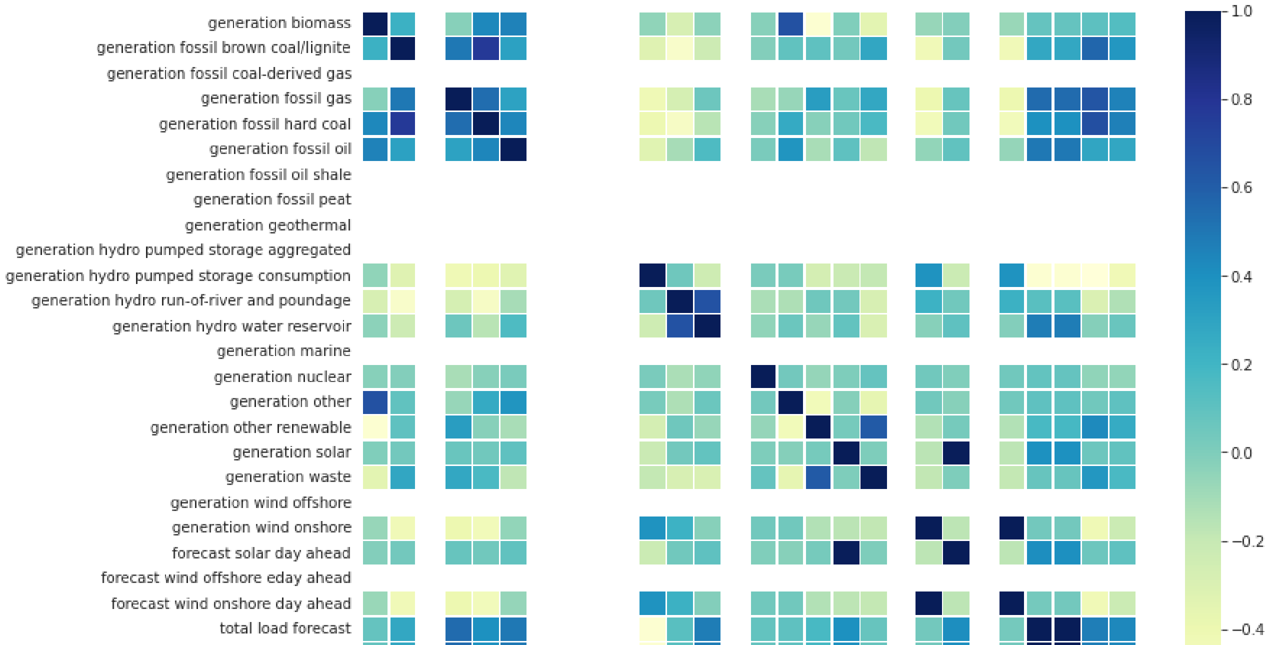
```
time
generation biomass                                    1
```

```
generation fossil brown coal/lignite                1
generation fossil coal-derived gas                  1
generation fossil gas                               1
generation fossil hard coal                         1
generation fossil oil                               1
generation fossil oil shale                         1
generation fossil peat                              1
generation geothermal                               1
generation hydro pumped storage aggregated       3506
generation hydro pumped storage consumption         1
generation hydro run-of-river and poundage          1
generation hydro water reservoir                    1
generation marine                                   1
generation nuclear                                  1
generation other                                    1
generation other renewable                          1
generation solar                                    1
generation waste                                    1
generation wind offshore                            1
generation wind onshore                             1
forecast solar day ahead
forecast wind offshore eday ahead                3506
forecast wind onshore day ahead
total load forecast
total load actual                                   3
price day ahead
price actual
dtype: int64
```

Notes:

- Some columns have the same amount of null values as rows.
- some rows are fluctuating

```
#correlation matrix
corr = energy.corr(method = 'pearson')
fig, ax = plt.subplots(figsize=(12, 9))
sns.heatmap(corr, linewidths=.5, cmap='YlGnBu');
```

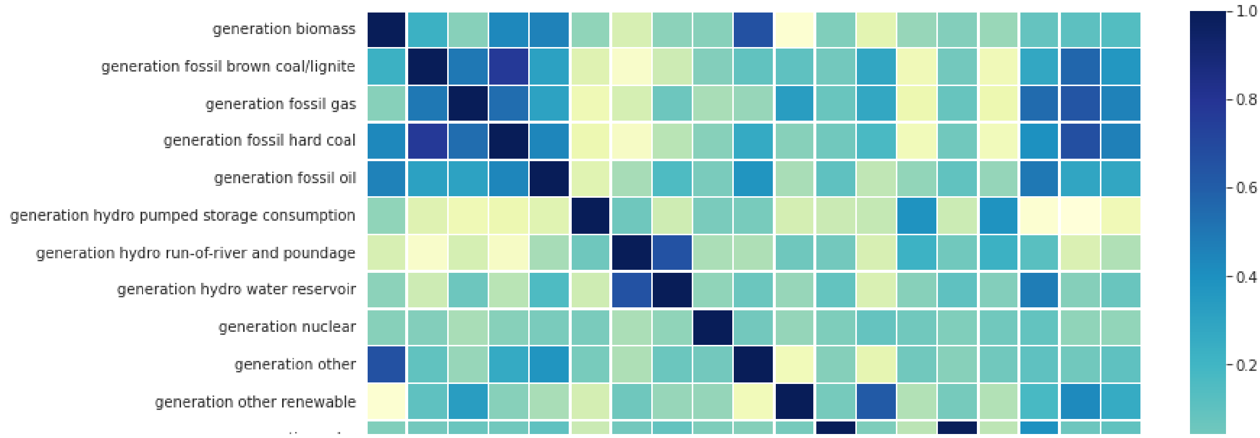


Notes:

- Clean up will include the removal of blank rows or items with an index of 0

```
#these columns are essentially useless since they have 1
drop_cols = ['generation hydro pumped storage aggregated
          'forecast wind offshore eday ahead',
          'generation fossil coal-derived gas',
          'generation fossil oil shale',
          'generation fossil peat',
          'generation geothermal',
          'generation marine',
          'generation wind offshore',
          'total load actual',
          'time']


#correlation matrix
corr = energy.drop(drop_cols, axis = 1).corr(method = 'p
```

```
fig, ax = plt.subplots(figsize=(12, 9))
sns.heatmap(corr, linewidths=.5, cmap='YlGnBu');
```




```
#top 6 correlations to price actual
corr.nlargest(6, 'price actual')['price actual']
```

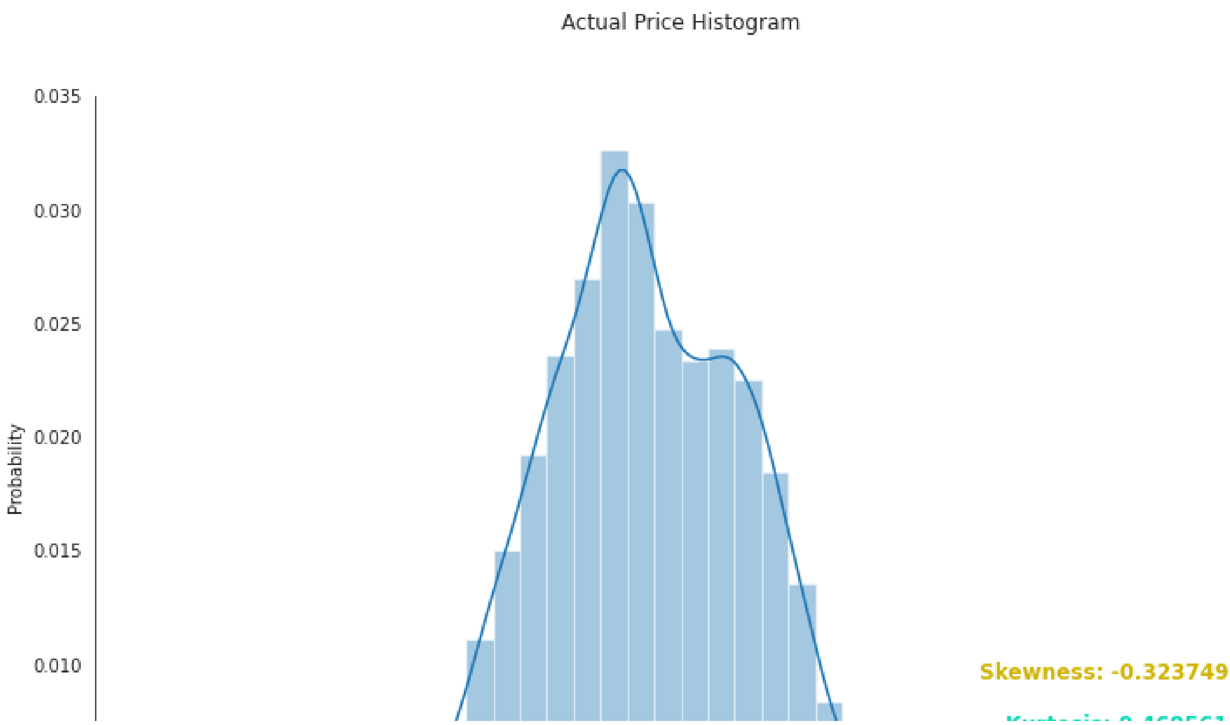
```
price actual                          1.000000
price day ahead                       0.732155
generation fossil hard coal           0.465641
generation fossil gas                 0.461706
total load forecast                   0.435864
generation fossil brown coal/lignite  0.364088
Name: price actual, dtype: float64
```

Notes

- only price day ahead has a relatively positive correlation, all other values are closer to no correlation at all

```
#Sturges Rule number of bins
bins = round(1+3.322*np.log(energy.shape[0]))

#Histogram
fig, ax = plt.subplots(figsize=(12, 9))
#Bins Calculated from the Square Root method
sns.distplot(energy['price actual'], rug=True, rug_kws={

#set labels
```

```
ax.set_title('Actual Price Histogram')
ax.set_xlabel(f'Actual Price (bins = {39})')
ax.set_ylabel('Probability')

#set legends
ax.text(x=0.97, y=0.27, transform=ax.transAxes, s="Skewn
  fontweight='demibold', fontsize=12, verticalalignment=
  color='xkcd:mustard')
ax.text(x=0.97, y=0.21, transform=ax.transAxes, s="Kurto
  fontweight='demibold', fontsize=12, verticalalignment=
  color='xkcd:aquamarine')



sns.despine(offset=2, trim=True)
```



Notes:

- Judging from the skewness, the histogram is fairly symetrical with a relatively normal distribution
- Kurtosis show that the prices are Platykurtic, which is a good indicator of less outliers
- Rugplot shows the distribution of values within histogram, and the values are rather centralized.

Data Cleaning

```
#these orginial columns are dropped
energy.drop(columns = drop_cols, inplace = True)


#shows all rows with nan values
energy[energy.isna().any(axis=1)]
```

	generation biomass	generation fossil brown coal/lignite	generation fossil gas	genera fo hard
99	NaN	NaN	NaN	
108	NaN	NaN	NaN	
109	NaN	NaN	NaN	
110	NaN	NaN	NaN	

Notes:

- threshold of 14 would be sufficient to clean the majority of the dataset

451	NaN	NaN	NaN

```
#shows all rows with nan values after a thresh of 14
energy.dropna(thresh =14, axis = 0)[energy.dropna(thresh
```

generation generation generation genera

Notes:

- There wouldn't be too much data loss if we removed these rows.

```
#These rows with na values more than 19 will be removed.
energy.dropna(thresh =19, axis = 0).isnull().sum()
```

```
generation biomass                              0
generation fossil brown coal/lignite            0
generation fossil gas                           0
generation fossil hard coal                     0
generation fossil oil                           0
generation hydro pumped storage consumption     0
generation hydro run-of-river and poundage      0
generation hydro water reservoir                0
generation nuclear                              0
generation other                                0
generation other renewable                      0
generation solar                                0
generation waste                                0
generation wind onshore                         0
forecast solar day ahead                        0
forecast wind onshore day ahead                 0
total load forecast                             0
price day ahead                                 0
price actual                                    0
dtype: int64
```

```
#show number remaining after shape
energy.dropna(thresh =19, axis = 0).shape
```

```
(35041, 19)
```

```
#confirm dropping the rows and columns
energy.dropna(thresh =19, axis = 0, inplace = True)
```

Notes:

- After dropping certain columns and cleaning up rows, the shape shows that we still have a significant amount of data to work with.

Modeling and Machine Learning

Split Features and Target Data

```
#define X and y

#feature matrix
X_defined = energy.drop(columns= ['price actual'])

#target variable
y_defined = energy['price actual']
```

Normalize the feature matrix using Standard Scaler

```
#normalize data
X = StandardScaler().fit_transform(X_defined)
```

```
#split train test data
X_train, X_test, y_train, y_test = train_test_split(X_de
```

Train on Linear Regression Model

```
#build linear model classifier
lr_model = LinearRegression()

#fit the data
lr_model.fit(X_train,y_train)

#make prediction using X_test data
lr_y_pred = lr_model.predict(X_test)
```

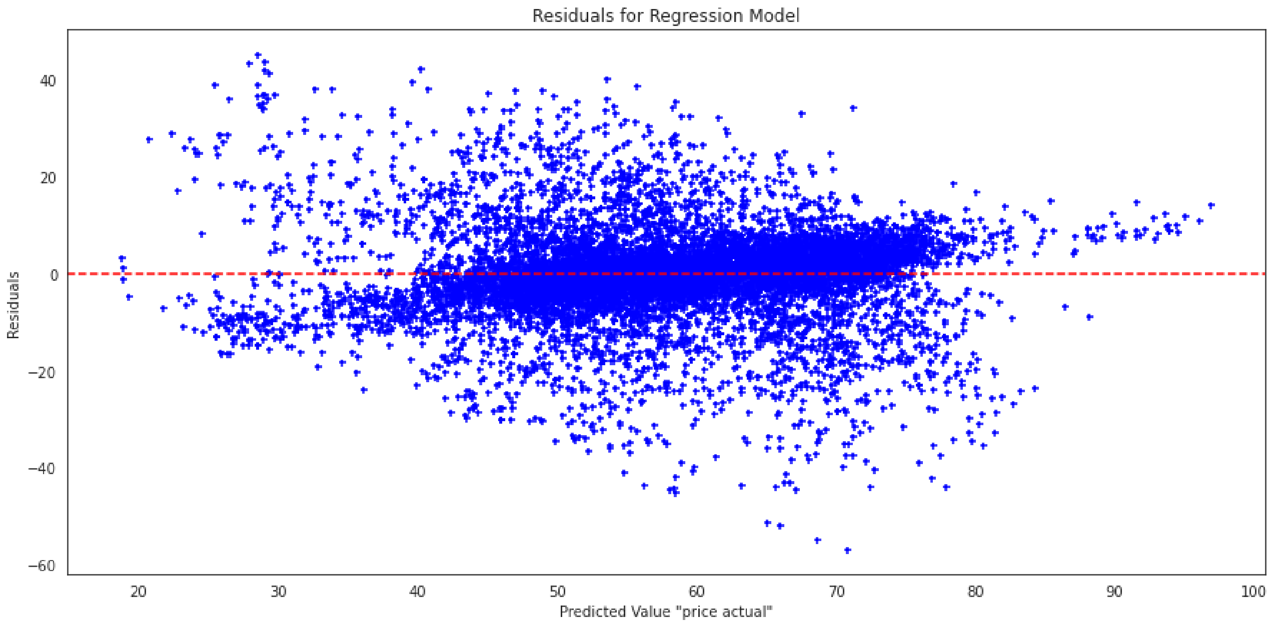
Residuals

```
#create dataframe or results of preddicted, actual, and
energy_result_df = pd.DataFrame({
    'Predicted': lr_y_pred,
    'Actual': y_test,
    'Residual': y_test - lr_y_pred})
```

```
plt.figure(figsize=[15,7])
```

```
plt.scatter(energy_result_df.Predicted, energy_result_df
plt.title('Residuals for Regression Model')
plt.xlabel('Predicted Value "price actual"')
plt.ylabel('Residuals')
plt.axhline(y = 0, color = 'r', ls= '--')
```

```
<matplotlib.lines.Line2D at 0x7f3fc28f0890>
```



Notes:

- Most of the residuals lie between 40 and 80 of the predicted value.
- due to the shape of the model, this indicates that there isn't really any form of trending pattern in the algorithm. Linear regression is most likely not a good fit to predict the independent variable.

GridSearch

```
#gridsearch Hyper Parameters
lr_parameter_space = {
    'fit_intercept': [True, False],
    'normalize': [True, False],
    'positive' : [True, False],
    'n_jobs' : [-1,1]
}


#gridsearch object
clf_lr = GridSearchCV(lr_model, lr_parameter_space)

#fit the data
clf_lr.fit(X_train,y_train)

#make prediction using X_test data
clf_lr_y_pred = clf_lr.predict(X_test)
```

Feature Selection

```
### Feature Selection
#construct RFE, model select top 6
rfe = RFE(lr_model,n_features_to_select = 6)
#fit the RFE model
rfe.fit(X_train,y_train)
```

```
RFE(estimator=LinearRegression(), n_features_to_sel
```

```
#tells us what column was selected .. (first column is n
rfe.support_
```

```
array([False, False, False, False,  True, False,  T
        True,  True, False,  True, False, False, Fa
```

```
#visualize the columns with all columns
selected = pd.DataFrame(data= rfe.support_, index = X_de

#shows only RFE True Selection
selected[selected[0]==True]
```

```
#shows the top 6 most correlated to price actual
corr.nlargest(6, 'price actual')['price actual']
```

```
price actual                                  1.000000
price day ahead                               0.732155
generation fossil hard coal                   0.465641
generation fossil gas                         0.461706
total load forecast                           0.435864
generation fossil brown coal/lignite          0.364088
Name: price actual, dtype: float64
```

Notes:

- Compared to the correlation, only Price Day Ahead was selected in RFE selection

```
#selects only the columns that are true
X_defined.columns[rfe.support_]
```

```
Index(['generation fossil oil', 'generation hydro r
       'generation other', 'generation other renewa
       'price day ahead'],
      dtype='object')
```

```
#original X_test col
X_test.head()
```

	generation biomass	generation fossil brown coal/lignite	generation fossil gas	genera fo hard
28504	388.0	625.0	4635.0	44
13685	392.0	0.0	3812.0	2
32520	368.0	479.0	4332.0	4
34705	294.0	188.0	4459.0	2

```
#updated X_test
X_test = X_test[X_defined.columns[rfe.support_]]
X_test.head()
```

	generation fossil oil	generation hydro run-of-river and poundage	generation other	generati oth renewab
28504	248.0	909.0	61.0	96
13685	242.0	668.0	56.0	8
32520	262.0	669.0	57.0	8

```
#updated X_train
X_train = X_train[X_defined.columns[rfe.support_]]
X_train.head()
```

	generation fossil oil	generation hydro run-of-river and poundage	generation other	generati oth renewab
3790	415.0	711.0	86.0	7
17131	320.0	712.0	56.0	10

```
#rebuld MLR model using the subset columns
lr_model.fit(X_train, y_train)

#make prediction using X_test data
rfe_lr_y_pred = lr_model.predict(X_test)
```

Accuracy Reports

```
#explained variance no gridsearch
print("R2: ", metrics.r2_score(y_test, lr_y_pred).round(
print("MSE: ", metrics.mean_squared_error(y_test, lr_y_p
print("RMSE: ", np.sqrt(metrics.mean_squared_error(y_tes
```

```
R2:  0.56
MSE:  88.79
RMSE:  9.42
```

```
#explained variance with gridsearch
print("R2: ", metrics.r2_score(y_test, clf_lr_y_pred).ro
print("MSE: ", metrics.mean_squared_error(y_test, clf_lr
print("RMSE: ", np.sqrt(metrics.mean_squared_error(y_tes
```

```
R2:  0.56
MSE:  88.79
RMSE:  9.42
```

```
#explained variance with RFE Feature Selection
print("R2: ", metrics.r2_score(y_test, rfe_lr_y_pred).ro
print("MSE: ", metrics.mean_squared_error(y_test, rfe_lr
print("RMSE: ", np.sqrt(metrics.mean_squared_error(y_tes
```

```
R2:  0.54
MSE:  92.33
RMSE:  9.61
```

```
#data
a_lr = [metrics.r2_score(y_test, lr_y_pred),
        metrics.mean_squared_error(y_test, lr_y_pred),
        np.sqrt(metrics.mean_squared_error(y_test, lr_y_
        ]

gs_lr = [metrics.r2_score(y_test, clf_lr_y_pred),
        metrics.mean_squared_error(y_test, clf_lr_y_pred
        np.sqrt(metrics.mean_squared_error(y_test, clf_l
        ]

rfe_lr = [metrics.r2_score(y_test, rfe_lr_y_pred),
        metrics.mean_squared_error(y_test, rfe_lr_y_pred
        np.sqrt(metrics.mean_squared_error(y_test, rfe_l
        ]

#dataframe df
accuracy_df = pd.DataFrame(np.array([a_lr, gs_lr, rfe_lr
                                    index = ['Linear Regressio
```

```
                          columns=['R2', 'MSE', 'RMS
accuracy_df
```

	R2	MSE	RMSE
Linear Regression Model	0.560213	88.788416	9.422761
GridSearch Linear Regression Model	0.560213	88.788416	9.422761
RFE Linear Regression	0.542677	92.328898	9.608793

Notes:

- Linear Regression Model showed low accuracy on all metrics.
- Gridsearch showed the same accuracy on all metrics. The linear regression model without hyperparameters is already the best result.
- With Recursive Feature Elimination (RFE), there is a significant amount of data removal, which should stem with higher accuracy. Since this metric shows lower accuracy than without feature importance, there must be other columns that are important within the dataset that justifies its accuracy.