In [2]:

```
import pandas as  pd
import numpy as np
```

In [3]:

```
data=pd.read_excel("C:/Users/Furkan/Desktop/FinalData.xlsx")
df=data.copy()
```

In [4]:

```
 #Dropping missing values
# For wind turbines production generally starts for wind speed values bigger than 3.5 m/s and smal
ler than 25 m/s
# So if wind speed is bigger than 3.5 but the production is 0 this means that it is a missing valu
e.
# To be able to build a model which guess for all wind speeds we did not drop the values for wind
speed is smaller than 3.5 m/s
# because our model should be able to predict if there will be no production

for items in df.index:
    if df.loc[items,"LV ActivePower (kW)"]==0 and df.loc[items,"Wind Speed (m/s)"]>=3.5:
        df=df.drop(items)
```

In [5]:

```
df=df.drop(["Sunrise","Sunset","Moonrise","Moonset","Date","Time"],axis=1)
```

In [6]:

```
from sklearn.preprocessing import scale
```

In [7]:

```
y=df["LV ActivePower (kW)"]
```

In [8]:

```
X=df.drop("LV ActivePower (kW)",axis=1)
```

In [9]:

```
X_scaled=pd.DataFrame(scale(X))
```

In [10]:

```
X_scaled.columns=X.columns
```

In [11]:

```
X_scaled
```

Out[11]:

	Wind Speed (m/s)	Theoretical_Power_Curve (KWh)	Wind Direction (°)	Month	Day/Night	Temp	Sun Hour	Moon Illumination	DewPoint	WindChillC	Wi
0	-0.531873	-0.791033	1.454940	-1.666537	-1.008335	-1.694744	-0.585395	1.606797	-1.352353	-1.427323	-1.
1	-0.447458	-0.715697	1.547640	-1.666537	-1.008335	-1.694744	-0.585395	1.606797	-1.352353	-1.427323	-1.

	Wind Speed (m/s)	Theoretical_Power_Curve (KWh)	Wind Direction (°)	Month	Day/Night	Temp	Sun Hour	Moon Illumination	DewPoint	WindChillC	Wi
2	-0.554168	-0.809527	1.589707	-1.666537	-1.008335	-1.694744	-0.585395	1.606797	-1.352353	-1.427323	-1.
3	-0.450381	-0.718453	1.575698	-1.666537	-1.008335	-1.694744	-0.585395	1.606797	-1.352353	-1.427323	-1.
4	-0.469502	-0.736216	1.515831	-1.666537	-1.008335	-1.694744	-0.585395	1.606797	-1.352353	-1.427323	-1.
...	...	...	...	...	...	...	...	...	...	...	
48308	0.893498	1.376845	-0.469476	1.628707	0.991734	-1.282735	-0.585395	-0.320615	-1.870099	-1.302749	-0.
48309	-0.058992	-0.240692	-0.431309	1.628707	0.991734	-1.282735	-0.585395	-0.320615	-1.870099	-1.302749	-0.
48310	0.198984	0.206743	-0.424019	1.628707	0.991734	-1.282735	-0.585395	-0.320615	-1.870099	-1.302749	-0.
48311	0.429658	0.664992	-0.428786	1.628707	0.991734	-1.282735	-0.585395	-0.320615	-1.870099	-1.302749	-0.
48312	0.560193	0.927390	-0.450479	1.628707	0.991734	-1.282735	-0.585395	-0.320615	-1.870099	-1.302749	-0.

48313 rows × 16 columns

In [12]:

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_validate,cross_val_predict,cross_val_score
```

In [13]:

```
#Selected 3 features according to mutual info
```

In [97]:

```
X_mi=X_scaled[["Wind Speed (m/s)","Theoretical_Power_Curve (KWh)","Density"]]
```

In [98]:

```
from sklearn.metrics import mean_squared_error,mean_absolute_error,median_absolute_error
from sklearn.model_selection import train_test_split
```

In [99]:

```
X_mi_train,X_mi_test,y_mi_train,y_mi_test=train_test_split(X_mi,y,test_size=0.33,random_state=15)
```

In [100]:

```
lin_reg_mi=LinearRegression()
```

In [101]:

```
lin_reg_mi.fit(X_mi_train,y_mi_train)
```

Out[101]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [102]:

```
lin_reg_mi.score(X_mi_test,y_mi_test)
```

Out[102]:

```
0.9590062914463233
```

In [103]:

```
mse_mi=mean_squared_error(y_mi_test,lin_reg_mi.predict(X_mi_test))
```

```
mse_mi=mean_squared_error(y_mi_test,lin_reg_mi.predict(X_mi_test))
```

In [104]:

```
mse_mi
```

Out[104]:

```
70815.0905740331
```

In [105]:

```
mae_mi=mean_absolute_error(y_mi_test,lin_reg_mi.predict(X_mi_test))
```

In [106]:

```
mae_mi
```

Out[106]:

```
130.82038354183214
```

In [107]:

```
cv_lin_mi=cross_val_score(LinearRegression(),X_mi,y,cv=10)
```

In [108]:

```
np.mean(cv_lin_mi)
```

Out[108]:

```
0.964642246833155
```

In [109]:

```
cv_lin_mi_mse=cross_val_score(LinearRegression(),X_mi,y,cv=10,scoring="neg_mean_squared_error")
```

In [110]:

```
np.mean(cv_lin_mi_mse)
```

Out[110]:

```
-65174.2927926603
```

In [111]:

```
cv_lin_mi_mae=cross_val_score(LinearRegression(),X_mi,y,cv=10,scoring="neg_mean_absolute_error")
```

In [112]:

```
np.mean(cv_lin_mi_mae)
```

Out[112]:

```
-128.48190745501242
```

In [113]:

```
cv_lin_mi_hybrid=cross_validate(LinearRegression(),X_mi_train,y_mi_train,cv=10,return_estimator=True)
```

In [114]:

```
cv_lin_mi_hybrid
```

Out[114]:

```
{'fit_time': array([0.01097131, 0.00797844, 0.00698161, 0.00897789, 0.00598526,
        0.00797844, 0.01196837, 0.01196861, 0.0209446 , 0.00598407]),
 'score_time': array([0.00299215, 0.00299144, 0.00199437, 0.00199318, 0.00299382,
        0.00299144, 0.00398827, 0.00498652, 0.00299287, 0.00299096]),
 'estimator': (LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False),
  LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False),
  LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False),
  LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False),
  LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False),
  LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False),
  LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False),
  LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False),
  LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False),
  LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)),
 'test_score': array([0.96985241, 0.95956937, 0.96854994, 0.96472092, 0.95741799,
        0.96635394, 0.96247456, 0.96422614, 0.95847937, 0.97176158])}
```

In [115]:

```
r_scores_mi_hybrid=[]
mse_mi_hybrid=[]
mae_mi_hybrid=[]
for items in cv_lin_mi_hybrid["estimator"]:
    r_scores_mi_hybrid.append(items.score(X_mi_test,y_mi_test))
    mse_mi_hybrid.append(mean_squared_error(y_mi_test,items.predict(X_mi_test)))
    mae_mi_hybrid.append(mean_absolute_error(y_mi_test,items.predict(X_mi_test)))
```

In [116]:

```
np.mean(r_scores_mi_hybrid)
```

Out[116]:

```
0.9590055718379192
```

In [117]:

```
np.mean(mse_mi_hybrid)
```

Out[117]:

```
70816.33367050115
```

In [118]:

```
np.mean(mae_mi_hybrid)
```

Out[118]:

```
130.82267121113446
```

In [43]:

```
#Linear regression with only wind speed and others not with theoretical --wind speed wind directio
n density
```

In [119]:

```
X_ext=X_scaled[["Wind Speed (m/s)","Wind Direction (°)","Density"]]
```

In [120]:

```
X_ext_train,X_ext_test,y_train,y_test=train_test_split(X_ext,y,test_size=0.33,random_state=15)
```

In [121]:

```
lin_ext=LinearRegression()
```

In [122]:

```
lin_ext.fit(X_ext_train,y_train)
```

Out[122]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [123]:

```
lin_ext.score(X_ext_test,y_test)
```

Out[123]:

```
0.8770128241245824
```

In [124]:

```
mse_ext=mean_squared_error(y_test,lin_ext.predict(X_ext_test))
```

In [125]:

```
mse_ext
```

Out[125]:

```
212455.7232400268
```

In [126]:

```
mae_ext=mean_absolute_error(y_test,lin_ext.predict(X_ext_test))
```

In [127]:

```
mae_ext
```

Out[127]:

```
353.928271207786
```

In [128]:

```
#Linear regression with only wind speed and others not with theoretical --theoretical power wind d
irection density
```

In [129]:

```
X_ext_2=X[["Theoretical_Power_Curve (KWh)","Wind Direction (°)","Density"]]
```

In [130]:

```
X_ext2_train,X_ext2_test,y_train2,y_test2=train_test_split(X_ext_2,y,test_size=0.33,random_state=1
5)
```

In [131]:

```
lin_ext_2=LinearRegression()
```

In [132]:

```
lin_ext_2.fit(X_ext2_train,y_train2)
```

Out[132]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [133]:

```
lin_ext_2.score(X_ext2_test,y_test2)
```

Out[133]:

```
0.9579897061683531
```

In [134]:

```
mse_ext2=mean_squared_error(y_test2,lin_ext_2.predict(X_ext2_test))
```

In [135]:

```
mse_ext2
```

Out[135]:

```
72571.20342831244
```

In [136]:

```
mae_ext2=mean_absolute_error(y_test2,lin_ext_2.predict(X_ext2_test))
```

In [137]:

```
mae_ext2
```

Out[137]:

```
131.8367612846859
```

In [138]:

```
#Ridge Regression with hpyperparameter Tuning
```

In [139]:

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
```

In [157]:

```
parameters={"alpha":[0.00001,0.0001,0.001,0.01,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1,5,10,50,100,10
00]}
```

In [158]:

```
ridge=Ridge()
```

In [159]:

```
grid_search=GridSearchCV(ridge,parameters,cv=10)
```

In [160]:

```
grid_search.fit(X_scaled,y)
```

Out[160]:

```
GridSearchCV(cv=10, error_score=nan,
             estimator=Ridge(alpha=1.0, copy_X=True, fit_intercept=True,
                             max_iter=None, normalize=False, random_state=None,
                             solver='auto', tol=0.001),
             iid='deprecated', n_jobs=None,
             param_grid={'alpha': [1e-05, 0.0001, 0.001, 0.01, 0.1, 0.2, 0.3,
                                   0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 5, 10, 50,
                                   100, 1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

In [161]:

```
grid_search.best_params_
```

Out[161]:

```
{'alpha': 10}
```

In [162]:

```
grid_search.best_score_
```

Out[162]:

```
0.9643034763827412
```

In [163]:

```
grid_search=GridSearchCV(ridge,parameters,cv=10,scoring="neg_mean_squared_error")
```

In [164]:

```
grid_search.fit(X_scaled,y)
```

Out[164]:

```
GridSearchCV(cv=10, error_score=nan,
             estimator=Ridge(alpha=1.0, copy_X=True, fit_intercept=True,
                             max_iter=None, normalize=False, random_state=None,
                             solver='auto', tol=0.001),
             iid='deprecated', n_jobs=None,
             param_grid={'alpha': [1e-05, 0.0001, 0.001, 0.01, 0.1, 0.2, 0.3,
                                   0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 5, 10, 50,
                                   100, 1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_mean_squared_error', verbose=0)
```

In [165]:

```
grid_search.best_score_
```

Out[165]:

```
-64891.00210530765
```

In [166]:

```
grid_search=GridSearchCV(ridge,parameters,cv=10,scoring="neg_mean_absolute_error")
```

In [167]:

```
grid_search.fit(X_scaled,y)
```

Out[167]:

```
GridSearchCV(cv=10, error_score=nan,
             estimator=Ridge(alpha=1.0, copy_X=True, fit_intercept=True,
                             max_iter=None, normalize=False, random_state=None,
```

```
                      solver='auto', tol=0.001),
             iid='deprecated', n_jobs=None,
             param_grid={'alpha': [1e-05, 0.0001, 0.001, 0.01, 0.1, 0.2, 0.3,
                                   0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 5, 10, 50,
                                   100, 1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_mean_absolute_error', verbose=0)
```

In [168]:

```
grid_search.best_score_
```

Out[168]:

```
-131.25423525060438
```

In [402]:

```
X_5f_rfe_ridge=X_scaled[["Wind Speed (m/s)","Theoretical_Power_Curve (KWh)","Temp","Pressure","Density"]]
```

In [403]:

```
X_3f_Lasso=X_scaled[["Wind Speed (m/s)","Theoretical_Power_Curve (KWh)","Wind Direction (°)"]]
```

In [404]:

```
X_3f_mi=X_scaled[["Wind Speed (m/s)","Theoretical_Power_Curve (KWh)","Density"]]
```

In [405]:

```
grid_rfe=GridSearchCV(Ridge(),parameters,cv=10)
```

In [406]:

```
grid_rfe.fit(X_5f_rfe_ridge,y)
```

Out[406]:

```
GridSearchCV(cv=10, error_score=nan,
             estimator=Ridge(alpha=1.0, copy_X=True, fit_intercept=True,
                             max_iter=None, normalize=False, random_state=None,
                             solver='auto', tol=0.001),
             iid='deprecated', n_jobs=None,
             param_grid={'alpha': [1e-05, 0.0001, 0.001, 0.01, 0.1, 0.2, 0.3,
                                   0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 5, 10, 50,
                                   100, 1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

In [407]:

```
grid_rfe.best_params_
```

Out[407]:

```
{'alpha': 10}
```

In [409]:

```
cv_grid_rfe=cross_val_score(grid_rfe.best_estimator_,X_5f_rfe_ridge,y,cv=10)
```

In [413]:

```
pd.DataFrame(cv_grid_rfe)
```

Out[413]:

	0
0	0.894577
1	0.957809
2	0.982592
3	0.984350
4	0.984824
5	0.974892
6	0.982893
7	0.988926
8	0.982654
9	0.917315

In [411]:

```
np.mean(cv_grid_rfe)
```

Out[411]:

```
0.9650830111724783
```

In [178]:

```
grid_rfe.best_score_
```

Out[178]:

```
0.9650830111724783
```

In [179]:

```
grid_rfe=GridSearchCV(Ridge(),parameters,cv=10,scoring="neg_mean_squared_error")
```

In [180]:

```
grid_rfe.fit(X_5f_rfe_ridge,y)
```

Out[180]:

```
GridSearchCV(cv=10, error_score=nan,
             estimator=Ridge(alpha=1.0, copy_X=True, fit_intercept=True,
                             max_iter=None, normalize=False, random_state=None,
                             solver='auto', tol=0.001),
             iid='deprecated', n_jobs=None,
             param_grid={'alpha': [1e-05, 0.0001, 0.001, 0.01, 0.1, 0.2, 0.3,
                                   0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 5, 10, 50,
                                   100, 1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_mean_squared_error', verbose=0)
```

In [181]:

```
grid_rfe.best_score_
```

Out[181]:

```
-64101.83799371244
```

In [184]:

```
grid_rfe=grid_rfe=GridSearchCV(Ridge(),parameters,cv=10,scoring="neg_mean_absolute_error")
```

In [185]:

```
grid_rfe.fit(X_5f_rfe_ridge,y)
```

Out[185]:

```
GridSearchCV(cv=10, error_score=nan,
             estimator=Ridge(alpha=1.0, copy_X=True, fit_intercept=True,
                             max_iter=None, normalize=False, random_state=None,
                             solver='auto', tol=0.001),
             iid='deprecated', n_jobs=None,
             param_grid={'alpha': [1e-05, 0.0001, 0.001, 0.01, 0.1, 0.2, 0.3,
                                   0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 5, 10, 50,
                                   100, 1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_mean_absolute_error', verbose=0)
```

In [186]:

```
grid_rfe.best_score_
```

Out[186]:

```
-129.45030738915142
```

In [188]:

```
grid_3f_Lasso=GridSearchCV(Ridge(),parameters,cv=10)
```

In [189]:

```
grid_3f_Lasso.fit(X_3f_Lasso,y)
```

Out[189]:

```
GridSearchCV(cv=10, error_score=nan,
             estimator=Ridge(alpha=1.0, copy_X=True, fit_intercept=True,
                             max_iter=None, normalize=False, random_state=None,
                             solver='auto', tol=0.001),
             iid='deprecated', n_jobs=None,
             param_grid={'alpha': [1e-05, 0.0001, 0.001, 0.01, 0.1, 0.2, 0.3,
                                   0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 5, 10, 50,
                                   100, 1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

In [190]:

```
grid_3f_Lasso.best_score_
```

Out[190]:

```
0.9650049864969178
```

In [191]:

```
grid_3f_Lasso=GridSearchCV(Ridge(),parameters,cv=10,scoring="neg_mean_squared_error")
```

In [192]:

```
grid_3f_Lasso.fit(X_3f_Lasso,y)
```

Out[192]:

```
GridSearchCV(cv=10, error_score=nan,
             estimator=Ridge(alpha=1.0, copy_X=True, fit_intercept=True,
                             max_iter=None, normalize=False, random_state=None,
                             solver='auto', tol=0.001),
             iid='deprecated', n_jobs=None,
             param_grid={'alpha': [1e-05, 0.0001, 0.001, 0.01, 0.1, 0.2, 0.3,
```

```
                                   0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 5, 10, 50,
                                   100, 1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_mean_squared_error', verbose=0)
```

In [193]:

```
grid_3f_Lasso.best_score_
```

Out[193]:

```
-64235.109536515956
```

In [194]:

```
grid_3f_Lasso=GridSearchCV(Ridge(),parameters,cv=10,scoring="neg_mean_absolute_error")
```

In [195]:

```
grid_3f_Lasso.fit(X_3f_Lasso,y)
```

Out[195]:

```
GridSearchCV(cv=10, error_score=nan,
             estimator=Ridge(alpha=1.0, copy_X=True, fit_intercept=True,
                             max_iter=None, normalize=False, random_state=None,
                             solver='auto', tol=0.001),
             iid='deprecated', n_jobs=None,
             param_grid={'alpha': [1e-05, 0.0001, 0.001, 0.01, 0.1, 0.2, 0.3,
                                   0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 5, 10, 50,
                                   100, 1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_mean_absolute_error', verbose=0)
```

In [196]:

```
grid_3f_Lasso.best_score_
```

Out[196]:

```
-128.7919147444924
```

In [197]:

```
grid_3f_mi=GridSearchCV(Ridge(),parameters,cv=10)
```

In [198]:

```
grid_3f_mi.fit(X_3f_mi,y)
```

Out[198]:

```
GridSearchCV(cv=10, error_score=nan,
             estimator=Ridge(alpha=1.0, copy_X=True, fit_intercept=True,
                             max_iter=None, normalize=False, random_state=None,
                             solver='auto', tol=0.001),
             iid='deprecated', n_jobs=None,
             param_grid={'alpha': [1e-05, 0.0001, 0.001, 0.01, 0.1, 0.2, 0.3,
                                   0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 5, 10, 50,
                                   100, 1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

In [199]:

```
grid_3f_mi.best_score_
```

Out[199]:

```
0.9646422468318715
```

In [200]:

```
grid_3f_mi=GridSearchCV(Ridge(),parameters,cv=10,scoring="neg_mean_squared_error")
```

In [201]:

```
grid_3f_mi.fit(X_3f_mi,y)
```

Out[201]:

```
GridSearchCV(cv=10, error_score=nan,
             estimator=Ridge(alpha=1.0, copy_X=True, fit_intercept=True,
                             max_iter=None, normalize=False, random_state=None,
                             solver='auto', tol=0.001),
             iid='deprecated', n_jobs=None,
             param_grid={'alpha': [1e-05, 0.0001, 0.001, 0.01, 0.1, 0.2, 0.3,
                                   0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 5, 10, 50,
                                   100, 1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_mean_squared_error', verbose=0)
```

In [202]:

```
grid_3f_mi.best_score_
```

Out[202]:

```
-65174.2791734955
```

In [203]:

```
grid_3f_mi=GridSearchCV(Ridge(),parameters,cv=10,scoring="neg_mean_absolute_error")
```

In [204]:

```
grid_3f_mi.fit(X_3f_mi,y)
```

Out[204]:

```
GridSearchCV(cv=10, error_score=nan,
             estimator=Ridge(alpha=1.0, copy_X=True, fit_intercept=True,
                             max_iter=None, normalize=False, random_state=None,
                             solver='auto', tol=0.001),
             iid='deprecated', n_jobs=None,
             param_grid={'alpha': [1e-05, 0.0001, 0.001, 0.01, 0.1, 0.2, 0.3,
                                   0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 5, 10, 50,
                                   100, 1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_mean_absolute_error', verbose=0)
```

In [205]:

```
grid_3f_mi.best_score_
```

Out[205]:

```
-128.4819075467492
```

In [207]:

```
from sklearn.linear_model import ElasticNet
#Hyperparameter tuning Elastic Net
```

In [240]:

```
from sklearn.model_selection import RandomizedSearchCV
```

In [241]:

```
parameters_l1={"l1_ratio":[0.01,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1],
               "alpha":[0.00001,0.0001,0.001,0.01,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1,5,10,50,100,
1000]}
```

In [263]:

```
rand_elastic=RandomizedSearchCV(ElasticNet(max_iter=10000),parameters_l1,cv=10,random_state=15)
```

In [264]:

```
rand_elastic.fit(X,y)
```

Out[264]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=ElasticNet(alpha=1.0, copy_X=True,
                                        fit_intercept=True, l1_ratio=0.5,
                                        max_iter=10000, normalize=False,
                                        positive=False, precompute=False,
                                        random_state=None, selection='cyclic',
                                        tol=0.0001, warm_start=False),
                   iid='deprecated', n_iter=10, n_jobs=None,
                   param_distributions={'alpha': [1e-05, 0.0001, 0.001, 0.01,
                                                  0.1, 0.2, 0.3, 0.4, 0.5, 0.6,
                                                  0.7, 0.8, 0.9, 1, 5, 10, 50,
                                                  100, 1000],
                                        'l1_ratio': [0.01, 0.1, 0.2, 0.3, 0.4,
                                                     0.5, 0.6, 0.7, 0.8, 0.9,
                                                     1]},
                   pre_dispatch='2*n_jobs', random_state=15, refit=True,
                   return_train_score=False, scoring=None, verbose=0)
```

In [265]:

```
rand_elastic.best_params_
```

Out[265]:

```
{'l1_ratio': 0.4, 'alpha': 0.8}
```

In [266]:

```
rand_elastic.best_score_
```

Out[266]:

```
0.9640910075130564
```

In [269]:

```
rand_elastic=RandomizedSearchCV(ElasticNet(max_iter=10000),parameters_l1,cv=10,random_state=15,sco
ring="neg_mean_squared_error")
```

In [270]:

```
rand_elastic.fit(X,y)
```

Out[270]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=ElasticNet(alpha=1.0, copy_X=True,
                                        fit_intercept=True, l1_ratio=0.5,
                                        max_iter=10000, normalize=False,
                                        positive=False, precompute=False,
                                        random_state=None, selection='cyclic',
                                        tol=0.0001, warm_start=False),
```

```
                   iid='deprecated', n_iter=10, n_jobs=None,
                   param_distributions={'alpha': [1e-05, 0.0001, 0.001, 0.01,
                                                  0.1, 0.2, 0.3, 0.4, 0.5, 0.6,
                                                  0.7, 0.8, 0.9, 1, 5, 10, 50,
                                                  100, 1000],
                                        'l1_ratio': [0.01, 0.1, 0.2, 0.3, 0.4,
                                                     0.5, 0.6, 0.7, 0.8, 0.9,
                                                     1]},
                   pre_dispatch='2*n_jobs', random_state=15, refit=True,
                   return_train_score=False, scoring='neg_mean_squared_error',
                   verbose=0)
```

In [271]:

```
rand_elastic.best_score_
```

Out[271]:

```
-65576.68444982209
```

In [272]:

```
rand_elastic=RandomizedSearchCV(ElasticNet(max_iter=10000),parameters_l1,cv=10,random_state=15,sco
ring="neg_mean_absolute_error")
```

In [273]:

```
rand_elastic.fit(X,y)
```

Out[273]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=ElasticNet(alpha=1.0, copy_X=True,
                                        fit_intercept=True, l1_ratio=0.5,
                                        max_iter=10000, normalize=False,
                                        positive=False, precompute=False,
                                        random_state=None, selection='cyclic',
                                        tol=0.0001, warm_start=False),
                   iid='deprecated', n_iter=10, n_jobs=None,
                   param_distributions={'alpha': [1e-05, 0.0001, 0.001, 0.01,
                                                  0.1, 0.2, 0.3, 0.4, 0.5, 0.6,
                                                  0.7, 0.8, 0.9, 1, 5, 10, 50,
                                                  100, 1000],
                                        'l1_ratio': [0.01, 0.1, 0.2, 0.3, 0.4,
                                                     0.5, 0.6, 0.7, 0.8, 0.9,
                                                     1]},
                   pre_dispatch='2*n_jobs', random_state=15, refit=True,
                   return_train_score=False, scoring='neg_mean_absolute_error',
                   verbose=0)
```

In [274]:

```
rand_elastic.best_score_
```

Out[274]:

```
-129.41269546286463
```

In [276]:

```
X_rfe_elastic=X[["Wind Speed (m/s)","Theoretical_Power_Curve
(KWh)","Pressure","Visibility","Density"]]
```

In [277]:

```
rand_elastic_rfe=RandomizedSearchCV(ElasticNet(max_iter=10000),parameters_l1,cv=10,random_state=15
)
```

In [278]:

```python
rand_elastic_rfe.fit(X_rfe_elastic,y)
```

Out[278]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=ElasticNet(alpha=1.0, copy_X=True,
                                        fit_intercept=True, l1_ratio=0.5,
                                        max_iter=10000, normalize=False,
                                        positive=False, precompute=False,
                                        random_state=None, selection='cyclic',
                                        tol=0.0001, warm_start=False),
                   iid='deprecated', n_iter=10, n_jobs=None,
                   param_distributions={'alpha': [1e-05, 0.0001, 0.001, 0.01,
                                                  0.1, 0.2, 0.3, 0.4, 0.5, 0.6,
                                                  0.7, 0.8, 0.9, 1, 5, 10, 50,
                                                  100, 1000],
                                        'l1_ratio': [0.01, 0.1, 0.2, 0.3, 0.4,
                                                     0.5, 0.6, 0.7, 0.8, 0.9,
                                                     1]},
                   pre_dispatch='2*n_jobs', random_state=15, refit=True,
                   return_train_score=False, scoring=None, verbose=0)
```

In [279]:

```python
rand_elastic_rfe.best_params_
```

Out[279]:

```
{'l1_ratio': 1, 'alpha': 0.7}
```

In [280]:

```python
rand_elastic_rfe.best_score_
```

Out[280]:

```
0.9647524546100115
```

In [281]:

```python
rand_elastic_rfe=RandomizedSearchCV(ElasticNet(max_iter=10000),parameters_l1,cv=10,random_state=15
,scoring="neg_mean_squared_error")
```

In [282]:

```python
rand_elastic_rfe.fit(X_rfe_elastic,y)
```

Out[282]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=ElasticNet(alpha=1.0, copy_X=True,
                                        fit_intercept=True, l1_ratio=0.5,
                                        max_iter=10000, normalize=False,
                                        positive=False, precompute=False,
                                        random_state=None, selection='cyclic',
                                        tol=0.0001, warm_start=False),
                   iid='deprecated', n_iter=10, n_jobs=None,
                   param_distributions={'alpha': [1e-05, 0.0001, 0.001, 0.01,
                                                  0.1, 0.2, 0.3, 0.4, 0.5, 0.6,
                                                  0.7, 0.8, 0.9, 1, 5, 10, 50,
                                                  100, 1000],
                                        'l1_ratio': [0.01, 0.1, 0.2, 0.3, 0.4,
                                                     0.5, 0.6, 0.7, 0.8, 0.9,
                                                     1]},
                   pre_dispatch='2*n_jobs', random_state=15, refit=True,
                   return_train_score=False, scoring='neg_mean_squared_error',
                   verbose=0)
```

In [283]:

```
rand_elastic_rfe.best_score_
```

Out[283]:

```
-64920.41091175573
```

In [284]:

```
rand_elastic_rfe=RandomizedSearchCV(ElasticNet(max_iter=10000),parameters_l1,cv=10,random_state=15
,scoring="neg_mean_absolute_error")
```

In [286]:

```
rand_elastic_rfe.fit(X_rfe_elastic,y)
```

Out[286]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=ElasticNet(alpha=1.0, copy_X=True,
                                        fit_intercept=True, l1_ratio=0.5,
                                        max_iter=10000, normalize=False,
                                        positive=False, precompute=False,
                                        random_state=None, selection='cyclic',
                                        tol=0.0001, warm_start=False),
                   iid='deprecated', n_iter=10, n_jobs=None,
                   param_distributions={'alpha': [1e-05, 0.0001, 0.001, 0.01,
                                                  0.1, 0.2, 0.3, 0.4, 0.5, 0.6,
                                                  0.7, 0.8, 0.9, 1, 5, 10, 50,
                                                  100, 1000],
                                        'l1_ratio': [0.01, 0.1, 0.2, 0.3, 0.4,
                                                     0.5, 0.6, 0.7, 0.8, 0.9,
                                                     1]},
                   pre_dispatch='2*n_jobs', random_state=15, refit=True,
                   return_train_score=False, scoring='neg_mean_absolute_error',
                   verbose=0)
```

In [287]:

```
rand_elastic_rfe.best_score_
```

Out[287]:

```
-128.46492490509758
```

In [288]:

```
rand_elastic_lasso=RandomizedSearchCV(ElasticNet(max_iter=10000),parameters_l1,cv=10,random_state=
15)
```

In [289]:

```
rand_elastic_lasso.fit(X_3f_Lasso,y)
```

Out[289]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=ElasticNet(alpha=1.0, copy_X=True,
                                        fit_intercept=True, l1_ratio=0.5,
                                        max_iter=10000, normalize=False,
                                        positive=False, precompute=False,
                                        random_state=None, selection='cyclic',
                                        tol=0.0001, warm_start=False),
                   iid='deprecated', n_iter=10, n_jobs=None,
                   param_distributions={'alpha': [1e-05, 0.0001, 0.001, 0.01,
                                                  0.1, 0.2, 0.3, 0.4, 0.5, 0.6,
                                                  0.7, 0.8, 0.9, 1, 5, 10, 50,
                                                  100, 1000],
                                        'l1_ratio': [0.01, 0.1, 0.2, 0.3, 0.4,
                                                     0.5, 0.6, 0.7, 0.8, 0.9,
                                                     1]},
                   pre_dispatch='2*n_jobs', random_state=15, refit=True,
```

```
                   return_train_score=False, scoring=None, verbose=0)
```

In [290]:

```
rand_elastic_lasso.best_params_
```

Out[290]:

```
{'l1_ratio': 1, 'alpha': 0.7}
```

In [291]:

```
rand_elastic_lasso.best_score_
```

Out[291]:

```
0.9650124101216214
```

In [292]:

```
rand_elastic_lasso=RandomizedSearchCV(ElasticNet(max_iter=10000),parameters_l1,cv=10,random_state=
15,scoring="neg_mean_squared_error")
```

In [293]:

```
rand_elastic_lasso.fit(X_3f_Lasso,y)
```

Out[293]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=ElasticNet(alpha=1.0, copy_X=True,
                                        fit_intercept=True, l1_ratio=0.5,
                                        max_iter=10000, normalize=False,
                                        positive=False, precompute=False,
                                        random_state=None, selection='cyclic',
                                        tol=0.0001, warm_start=False),
                   iid='deprecated', n_iter=10, n_jobs=None,
                   param_distributions={'alpha': [1e-05, 0.0001, 0.001, 0.01,
                                                  0.1, 0.2, 0.3, 0.4, 0.5, 0.6,
                                                  0.7, 0.8, 0.9, 1, 5, 10, 50,
                                                  100, 1000],
                                        'l1_ratio': [0.01, 0.1, 0.2, 0.3, 0.4,
                                                     0.5, 0.6, 0.7, 0.8, 0.9,
                                                     1]},
                   pre_dispatch='2*n_jobs', random_state=15, refit=True,
                   return_train_score=False, scoring='neg_mean_squared_error',
                   verbose=0)
```

In [294]:

```
rand_elastic_lasso.best_score_
```

Out[294]:

```
-64236.757394415756
```

In [295]:

```
rand_elastic_lasso=RandomizedSearchCV(ElasticNet(max_iter=10000),parameters_l1,cv=10,random_state=
15,scoring="neg_mean_absolute_error")
```

In [296]:

```
rand_elastic_lasso.fit(X_3f_Lasso,y)
```

Out[296]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=ElasticNet(alpha=1.0, copy_X=True,
```

```
                                       fit_intercept=True, l1_ratio=0.5,
                                       max_iter=10000, normalize=False,
                                       positive=False, precompute=False,
                                       random_state=None, selection='cyclic',
                                       tol=0.0001, warm_start=False),
                   iid='deprecated', n_iter=10, n_jobs=None,
                   param_distributions={'alpha': [1e-05, 0.0001, 0.001, 0.01,
                                                  0.1, 0.2, 0.3, 0.4, 0.5, 0.6,
                                                  0.7, 0.8, 0.9, 1, 5, 10, 50,
                                                  100, 1000],
                                        'l1_ratio': [0.01, 0.1, 0.2, 0.3, 0.4,
                                                     0.5, 0.6, 0.7, 0.8, 0.9,
                                                     1]},
                   pre_dispatch='2*n_jobs', random_state=15, refit=True,
                   return_train_score=False, scoring='neg_mean_absolute_error',
                   verbose=0)
```

In [297]:

```
rand_elastic_lasso.best_score_
```

Out[297]:

```
-128.7764721524034
```

In [299]:

```
rand_elastic_mi=RandomizedSearchCV(ElasticNet(max_iter=10000),parameters_l1,cv=10,random_state=15)
```

In [400]:

```
rand_elastic_mi.fit(X_3f_mi,y)
```

Out[400]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=ElasticNet(alpha=1.0, copy_X=True,
                                       fit_intercept=True, l1_ratio=0.5,
                                       max_iter=10000, normalize=False,
                                       positive=False, precompute=False,
                                       random_state=None, selection='cyclic',
                                       tol=0.0001, warm_start=False),
                   iid='deprecated', n_iter=10, n_jobs=None,
                   param_distributions={'alpha': [1e-05, 0.0001, 0.001, 0.01,
                                                  0.1, 0.2, 0.3, 0.4, 0.5, 0.6,
                                                  0.7, 0.8, 0.9, 1, 5, 10, 50,
                                                  100, 1000],
                                        'l1_ratio': [0.01, 0.1, 0.2, 0.3, 0.4,
                                                     0.5, 0.6, 0.7, 0.8, 0.9,
                                                     1]},
                   pre_dispatch='2*n_jobs', random_state=15, refit=True,
                   return_train_score=False, scoring='neg_mean_absolute_error',
                   verbose=0)
```

In [399]:

```
rand_elastic_mi.best_params_
```

Out[399]:

```
{'l1_ratio': 0.8, 'alpha': 0.001}
```

In [302]:

```
rand_elastic_mi.best_score_
```

Out[302]:

```
0.964648168767452
```

In [303]:

```
rand_elastic_mi=RandomizedSearchCV(ElasticNet(max_iter=10000),parameters_l1,cv=10,random_state=15,
scoring="neg_mean_squared_error")
```

In [304]:

```
rand_elastic_mi.fit(X_3f_mi,y)
```

Out[304]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=ElasticNet(alpha=1.0, copy_X=True,
                                        fit_intercept=True, l1_ratio=0.5,
                                        max_iter=10000, normalize=False,
                                        positive=False, precompute=False,
                                        random_state=None, selection='cyclic',
                                        tol=0.0001, warm_start=False),
                   iid='deprecated', n_iter=10, n_jobs=None,
                   param_distributions={'alpha': [1e-05, 0.0001, 0.001, 0.01,
                                                  0.1, 0.2, 0.3, 0.4, 0.5, 0.6,
                                                  0.7, 0.8, 0.9, 1, 5, 10, 50,
                                                  100, 1000],
                                        'l1_ratio': [0.01, 0.1, 0.2, 0.3, 0.4,
                                                     0.5, 0.6, 0.7, 0.8, 0.9,
                                                     1]},
                   pre_dispatch='2*n_jobs', random_state=15, refit=True,
                   return_train_score=False, scoring='neg_mean_squared_error',
                   verbose=0)
```

In [305]:

```
rand_elastic_mi.best_score_
```

Out[305]:

```
-65161.84003918494
```

In [312]:

```
rand_elastic_mi=RandomizedSearchCV(ElasticNet(max_iter=10000),parameters_l1,cv=10,random_state=15,
scoring="neg_mean_absolute_error")
```

In [313]:

```
rand_elastic_mi.fit(X_3f_mi,y)
```

Out[313]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=ElasticNet(alpha=1.0, copy_X=True,
                                        fit_intercept=True, l1_ratio=0.5,
                                        max_iter=10000, normalize=False,
                                        positive=False, precompute=False,
                                        random_state=None, selection='cyclic',
                                        tol=0.0001, warm_start=False),
                   iid='deprecated', n_iter=10, n_jobs=None,
                   param_distributions={'alpha': [1e-05, 0.0001, 0.001, 0.01,
                                                  0.1, 0.2, 0.3, 0.4, 0.5, 0.6,
                                                  0.7, 0.8, 0.9, 1, 5, 10, 50,
                                                  100, 1000],
                                        'l1_ratio': [0.01, 0.1, 0.2, 0.3, 0.4,
                                                     0.5, 0.6, 0.7, 0.8, 0.9,
                                                     1]},
                   pre_dispatch='2*n_jobs', random_state=15, refit=True,
                   return_train_score=False, scoring='neg_mean_absolute_error',
                   verbose=0)
```

In [314]:

```
rand_elastic_mi.best_score_
```

Out[314]:

```
-128.5631910290474
```

In [315]:

```
from sklearn.neural_network import MLPRegressor
```

In [330]:

```
mlp_all=MLPRegressor(hidden_layer_sizes=250,activation="relu",learning_rate_init=0.001,max_iter=500
,random_state=15)
```

In [363]:

```
?MLPRegressor
```

In [332]:

```
X_mlp_train,X_mlp_test,y_mlp_train,y_mlp_test=train_test_split(X_scaled,y,test_size=0.33,random_sta
te=21)
```

In [334]:

```
mlp_all.fit(X_mlp_train,y_mlp_train)
```

```
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

Out[334]:

```
MLPRegressor(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
             beta_2=0.999, early_stopping=False, epsilon=1e-08,
             hidden_layer_sizes=250, learning_rate='constant',
             learning_rate_init=0.001, max_fun=15000, max_iter=500,
             momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
             power_t=0.5, random_state=15, shuffle=True, solver='adam',
             tol=0.0001, validation_fraction=0.1, verbose=False,
             warm_start=False)
```

In [335]:

```
mlp_all.score(X_mlp_test,y_mlp_test)
```

Out[335]:

```
0.9866539661716076
```

In [337]:

```
mse_mlp=mean_squared_error(y_mlp_test,mlp_all.predict(X_mlp_test))
```

In [339]:

```
mse_mlp
```

Out[339]:

```
23020.690785433842
```

In [341]:

```
mae_mlp=mean_absolute_error(y_mlp_test,mlp_all.predict(X_mlp_test))
```

In [342]:

```
mae_mlp
```

Out[342]:

```
79.37164760221401
```

In [371]:

```
mlp_he=MLPRegressor(hidden_layer_sizes=250,activation="relu",learning_rate_init=0.001,max_iter=5000
,random_state=15)
```

In [372]:

```
X_mlp_train2,X_mlp_test2,y_mlp_train2,y_mlp_test2=train_test_split(X,y,test_size=0.33,random_state
=21)
```

In [373]:

```
mlp_all.fit(X_mlp_train2,y_mlp_train2)
```

Out[373]:

```
MLPRegressor(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
             beta_2=0.999, early_stopping=False, epsilon=1e-08,
             hidden_layer_sizes=250, learning_rate='constant',
             learning_rate_init=0.001, max_fun=15000, max_iter=500,
             momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
             power_t=0.5, random_state=15, shuffle=True, solver='adam',
             tol=0.0001, validation_fraction=0.1, verbose=False,
             warm_start=False)
```

In [396]:

```
mlp_all.score(X_mlp_test2,y_mlp_test2)
```

Out[396]:

```
0.9876226716827897
```

In [389]:

```
X_mlp_he=X.drop(["Temp","WindGust","Theoretical_Power_Curve (KWh)"],axis=1)
```

In [392]:

```
X_mlp_he_train,X_mlp_he_test,y_mlp_he_train,y_mlp_he_test=train_test_split(X_mlp_he,y,test_size=0.3
3,random_state=21)
```

In [394]:

```
mlp_he.fit(X_mlp_he_train,y_mlp_train)
```

Out[394]:

```
MLPRegressor(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
             beta_2=0.999, early_stopping=False, epsilon=1e-08,
             hidden_layer_sizes=250, learning_rate='constant',
             learning_rate_init=0.001, max_fun=15000, max_iter=5000,
             momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
             power_t=0.5, random_state=15, shuffle=True, solver='adam',
             tol=0.0001, validation_fraction=0.1, verbose=False,
             warm_start=False)
```

In [ ]:

```
mlp_he.score(X_mlp_he_test,y_mlp_he_test)
```

In [ ]:

```
mean_mlp_he_2=mean_squared_error(y_mlp_he_test,mlp_he.predict(X_mlp_he_test))
```

In [ ]:

```
mean_mlp_he_2
```

In [ ]:

```
mae_mlp_he_2=mean_absolute_error(y_mlp_he_test,mlp_he.predict(X_mlp_he_test))
```

In [ ]:

```
mae_mlp_he_2
```

In [378]:

```
cv_mlp=cross_validate(MLPRegressor(hidden_layer_sizes=250,random_state=15),X_mlp_he,y,cv=10)
```

```
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization
hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization
hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization
hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization
hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization
hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization
hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization
hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization
hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization
hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization
hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

In [378]:

```
cv_mlp=cross_validate(MLPRegressor(hidden_layer_sizes=250,random_state=15),X_mlp_he,y,cv=10)
```

```
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization
```

```
hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization
hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization
hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization
hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization
hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization
hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization
hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization
hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization
hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization
hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

In [379]:

```
cv_mlp
```

Out[379]:

```
{'fit_time': array([161.59515023, 161.83225441, 164.09397316, 166.27589059,
        162.96442223, 164.75759196, 164.78133607, 163.6471324 ,
        166.47193408, 178.66421413]),
 'score_time': array([0.05385685, 0.05602527, 0.05385518, 0.05485225, 0.05285859,
        0.05499482, 0.05301547, 0.05801702, 0.05299735, 0.05385518]),
 'test_score': array([ 0.88551149,  0.9541271 ,  0.95760364,  0.9848664 ,  0.99057033,
         0.97895389,  0.99183676,  0.99109201,  0.92922731, -1.97103369])}
```

In [381]:

```
np.median(cv_mlp["test_score"])
```

Out[381]:

```
0.9682787688366805
```

In [382]:

```
np.mean(cv_mlp["test_score"])
```

Out[382]:

```
0.6692755258135641
```

In [383]:

```
cv_mlp=cross_validate(MLPRegressor(hidden_layer_sizes=100,random_state=15),X_mlp_he,y,cv=10)
```

```
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization
hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization
hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization
hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization
hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization
hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization
hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization
hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization
hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization
hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization
hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

In [384]:

```
cv_mlp
```

Out[384]:

```
{'fit_time': array([87.16841936, 85.99464703, 88.20895815, 85.82003641, 85.29909849,
        85.45317626, 85.11338401, 88.92411399, 87.56587577, 91.32421875]),
 'score_time': array([0.0239687 , 0.02396631, 0.03195834, 0.02396822, 0.02396679,
        0.02796054, 0.02396846, 0.0199728 , 0.02396774, 0.02396798]),
 'test_score': array([0.88270883, 0.94420471, 0.95669055, 0.98011865, 0.98477115,
        0.97621047, 0.99022997, 0.98961552, 0.94367601, 0.00215159])}
```

In [385]:

```
np.mean(cv_mlp["test_score"])
```

Out[385]:

```
0.8650377449187963
```

In [395]:

```
cv_mlp=cross_validate(MLPRegressor(hidden_layer_sizes=100,activation="relu",learning_rate_init=0.00
1,max_iter=1000,random_state=15),
                     X_mlp_he,y,cv=10)
```

```
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:573:
UserWarning: Training interrupted by user.
  warnings.warn("Training interrupted by user.")
```

```
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:573:
UserWarning: Training interrupted by user.
  warnings.warn("Training interrupted by user.")
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:573:
UserWarning: Training interrupted by user.
  warnings.warn("Training interrupted by user.")
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:573:
UserWarning: Training interrupted by user.
  warnings.warn("Training interrupted by user.")
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:573:
UserWarning: Training interrupted by user.
  warnings.warn("Training interrupted by user.")
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:573:
UserWarning: Training interrupted by user.
  warnings.warn("Training interrupted by user.")
```

In [398]:

```
cv_mlp=cross_validate(MLPRegressor(hidden_layer_sizes=100,activation="relu",learning_rate_init=0.00
1,max_iter=500,random_state=15),
                      X_mlp_he,y,cv=10,scoring=["neg_mean_squared_error","neg_mean_absolute_error","
r2"])
```

```
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:573:
UserWarning: Training interrupted by user.
  warnings.warn("Training interrupted by user.")
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:573:
UserWarning: Training interrupted by user.
  warnings.warn("Training interrupted by user.")
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:573:
UserWarning: Training interrupted by user.
  warnings.warn("Training interrupted by user.")
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:573:
UserWarning: Training interrupted by user.
  warnings.warn("Training interrupted by user.")
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:573:
UserWarning: Training interrupted by user.
  warnings.warn("Training interrupted by user.")
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:573:
UserWarning: Training interrupted by user.
  warnings.warn("Training interrupted by user.")
C:\Users\Furkan\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:573:
UserWarning: Training interrupted by user.
  warnings.warn("Training interrupted by user.")
```

In [ ]:

In [72]:

```
import pandas as pd
import numpy as np
from sklearn.feature_selection import mutual_info_regression
from sklearn.feature_selection import f_regression
from sklearn.feature_selection import variance_threshold
from sklearn.feature_selection import RFE
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error,mean_absolute_error,median_absolute_error
from sklearn.impute import SimpleImputer
```

In [73]:

```
data=pd.read_excel("C:/Users/Furkan/Desktop/FinalData.xlsx")
df=data.copy()
```

In [74]:

```
 #Dropping missing values
# For wind turbines production generally starts for wind speed values bigger than 3.5 m/s and smal
ler than 25 m/s
# So if wind speed is bigger than 3.5 but the production is 0 this means that it is a missing valu
e.
# To be able to build a model which guess for all wind speeds we did not drop the values for wind
speed is smaller than 3.5 m/s
# because our model should be able to predict if there will be no production

for items in df.index:
    if df.loc[items,"LV ActivePower (kW)"]==0 and df.loc[items,"Wind Speed (m/s)"]>=3.5:
        df=df.drop(items)
```

In [75]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 48313 entries, 0 to 50529
Data columns (total 23 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Date                            48313 non-null  object
 1   Time                            48313 non-null  object
 2   LV ActivePower (kW)             48313 non-null  float64
 3   Wind Speed (m/s)                48313 non-null  float64
 4   Theoretical_Power_Curve (KWh)   48313 non-null  float64
 5   Wind Direction (°)              48313 non-null  float64
 6   Month                           48313 non-null  int64
 7   Day/Night                       48313 non-null  int64
 8   Temp                            48313 non-null  int64
 9   Sun Hour                        48313 non-null  float64
 10  Moon Illumunation               48313 non-null  int64
 11  Moonrise                        48313 non-null  object
 12  Moonset                         48313 non-null  object
 13  Sunrise                         48313 non-null  object
 14  Sunset                          48313 non-null  object
 15  DewPoint                        48313 non-null  int64
 16  WindChillC                      48313 non-null  int64
 17  WindGust                        48313 non-null  int64
 18  Humidity                        48313 non-null  int64
 19  RainMM                          48313 non-null  float64
 20  Pressure                        48313 non-null  int64
 21  Visibility                      48313 non-null  int64
 22  Density                         48313 non-null  float64
dtypes: float64(7), int64(10), object(6)
memory usage: 10.1+ MB
```

In [76]:

```
#After this we will drop unnecessary columns or non attribute columns such as Data,Time and Sunris
e Sunset Moonrise MoonSet
```

In [77]:

```
df=df.drop(["Sunrise","Sunset","Moonrise","Moonset","Date","Time"],axis=1)
```

In [78]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 48313 entries, 0 to 50529
Data columns (total 17 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   LV ActivePower (kW)             48313 non-null  float64
 1   Wind Speed (m/s)                48313 non-null  float64
 2   Theoretical_Power_Curve (KWh)   48313 non-null  float64
 3   Wind Direction (°)              48313 non-null  float64
 4   Month                           48313 non-null  int64  
 5   Day/Night                       48313 non-null  int64  
 6   Temp                            48313 non-null  int64  
 7   Sun Hour                        48313 non-null  float64
 8   Moon Illumunation               48313 non-null  int64  
 9   DewPoint                        48313 non-null  int64  
 10  WindChillC                      48313 non-null  int64  
 11  WindGust                        48313 non-null  int64  
 12  Humidity                        48313 non-null  int64  
 13  RainMM                          48313 non-null  float64
 14  Pressure                        48313 non-null  int64  
 15  Visibility                      48313 non-null  int64  
 16  Density                         48313 non-null  float64
dtypes: float64(7), int64(10)
memory usage: 7.9 MB
```

In [148]:

```
#Now we should standardize our variables because they are distributed in different ranges
#We will standardize only X
```

In [80]:

```
from sklearn.preprocessing import scale
```

In [154]:

```
y=df["LV ActivePower (kW)"]
```

In [155]:

```
X=df.drop("LV ActivePower (kW)",axis=1)
```

In [156]:

```
X_scaled=pd.DataFrame(scale(X))
```

In [157]:

```
X_scaled.columns=X.columns
```

In [158]:

```
X_scaled
```

Out[158]:

	Wind Speed (m/s)	Theoretical_Power_Curve (KWh)	Wind Direction (°)	Month	Day/Night	Temp	Sun Hour	Moon Illumunation	DewPoint	WindChillC	Wi
0	-0.531873	-0.791033	1.454940	-1.666537	-1.008335	-1.694744	-0.585395	1.606797	-1.352353	-1.427323	-1.
1	-0.447458	-0.715697	1.547640	-1.666537	-1.008335	-1.694744	-0.585395	1.606797	-1.352353	-1.427323	-1.
2	-0.554168	-0.809527	1.589707	-1.666537	-1.008335	-1.694744	-0.585395	1.606797	-1.352353	-1.427323	-1.
3	-0.450381	-0.718453	1.575698	-1.666537	-1.008335	-1.694744	-0.585395	1.606797	-1.352353	-1.427323	-1.
4	-0.469502	-0.736216	1.515831	-1.666537	-1.008335	-1.694744	-0.585395	1.606797	-1.352353	-1.427323	-1.
...	...	...	...	...	...	...	...	...	...	...	
48308	0.893498	1.376845	-0.469476	1.628707	0.991734	-1.282735	-0.585395	-0.320615	-1.870099	-1.302749	-0.
48309	-0.058992	-0.240692	-0.431309	1.628707	0.991734	-1.282735	-0.585395	-0.320615	-1.870099	-1.302749	-0.
48310	0.198984	0.206743	-0.424019	1.628707	0.991734	-1.282735	-0.585395	-0.320615	-1.870099	-1.302749	-0.
48311	0.429658	0.664992	-0.428786	1.628707	0.991734	-1.282735	-0.585395	-0.320615	-1.870099	-1.302749	-0.
48312	0.560193	0.927390	-0.450479	1.628707	0.991734	-1.282735	-0.585395	-0.320615	-1.870099	-1.302749	-0.

48313 rows × 16 columns

In [159]:

```
X=X_scaled
```

In [160]:

```
#Now we can work on our data
```

In [88]:

```
#Feature Selection
```

In [89]:

```
#F Regression
```

In [90]:

```
f_reg=f_regression(X,y)
```

In [91]:

```
f_reg_table=pd.DataFrame(f_reg)
f_reg_table=f_reg_table
```

In [92]:

```
f_reg_table.columns=X.columns
```

In [94]:

```
f_reg_table.T.info
```

Out[94]:

```
<bound method DataFrame.info of                                          0              1
```

```
Wind Speed (m/s)                  3.560684e+05    0.000000e+00
Theoretical_Power_Curve (KWh)    1.192345e+06    0.000000e+00
Wind Direction (°)               2.553324e+02    2.506781e-57
Month                            1.223384e+01    4.697278e-04
Day/Night                        9.532381e+01    1.695958e-22
Temp                             2.180131e+02    3.142335e-49
Sun Hour                         2.204174e+03    0.000000e+00
Moon Illumunation                1.142351e-02    9.148840e-01
DewPoint                         1.401897e+03   1.776312e-302
WindChillC                       5.918670e+02   5.975151e-130
WindGust                         1.665208e+04    0.000000e+00
Humidity                         2.984692e+02    1.127214e-66
RainMM                           2.177748e+00    1.400258e-01
Pressure                         1.589587e+02    2.180371e-36
Visibility                       2.689548e+02    2.791806e-60
Density                          2.444209e+02    5.830252e-55>
```

In [96]:

```
#Feature Selection via Lasso Regression
```

In [97]:

```
lasso_reg=Lasso(max_iter=10000)
```

In [98]:

```
from sklearn.model_selection import GridSearchCV
```

In [99]:

```
parameters={"alpha":[0.01,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1]}
```

In [100]:

```
grid=GridSearchCV(lasso_reg,parameters,cv=10)
```

In [101]:

```
grid.fit(X,y)
```

Out[101]:

```
GridSearchCV(cv=10, error_score=nan,
             estimator=Lasso(alpha=1.0, copy_X=True, fit_intercept=True,
                             max_iter=10000, normalize=False, positive=False,
                             precompute=False, random_state=None,
                             selection='cyclic', tol=0.0001, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'alpha': [0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8,
                                   0.9, 1]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

In [102]:

```
grid.best_params_
```

Out[102]:

```
{'alpha': 0.01}
```

In [103]:

```
grid.best_estimator_
```

Out[103]:

```
Lasso(alpha=0.01, copy_X=True, fit_intercept=True, max_iter=10000,
```

```
Lasso(alpha=0.01, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```

In [104]:

```
grid.best_estimator_.coef_
```

Out[104]:

```
array([ 0.11265648,  0.86535698,  0.01181929,  0.        , -0.        ,
       -0.        ,  0.        , -0.        , -0.        ,  0.        ,
        0.        , -0.        , -0.00722612,  0.        ,  0.        ,
        0.        ])
```

In [105]:

```
lasso_table=pd.DataFrame(grid.best_estimator_.coef_)
```

In [106]:

```
lasso_table=lasso_table.T
```

In [107]:

```
lasso_table.columns=X.columns
```

In [110]:

```
lasso_table.T
```

Out[110]:

	0
Wind Speed (m/s)	0.112656
Theoretical_Power_Curve (KWh)	0.865357
Wind Direction (°)	0.011819
Month	0.000000
Day/Night	-0.000000
Temp	-0.000000
Sun Hour	0.000000
Moon Illumunation	-0.000000
DewPoint	-0.000000
WindChillC	0.000000
WindGust	0.000000
Humidity	-0.000000
RainMM	-0.007226
Pressure	0.000000
Visibility	0.000000
Density	0.000000

In [36]:

```
#Mutual Info
```

In [81]:

```
mutual_info_regression(X["Wind Speed (m/s)"].values.reshape(-1,1),y)
```

Out[81]:

```
array([2.14358172])
```

In [82]:

```
mutual_info_regression(X["Theoretical_Power_Curve (KWh)"].values.reshape(-1,1),y,copy=False)
```

Out[82]:

```
array([2.10675857])
```

In [37]:

```
mutual_info_regression(X["Wind Direction (°)"].values.reshape(-1,1),y,copy=False)
```

Out[37]:

```
array([0.20771214])
```

In [38]:

```
mutual_info_regression(X["Month"].values.reshape(-1,1),y,copy=False)
```

Out[38]:

```
array([0.1490252])
```

In [39]:

```
mutual_info_regression(X["Day/Night"].values.reshape(-1,1),y,copy=False)
```

Out[39]:

```
array([0.00453503])
```

In [40]:

```
mutual_info_regression(X["Temp"].values.reshape(-1,1),y,copy=False)
```

Out[40]:

```
array([0.11342294])
```

In [41]:

```
mutual_info_regression(X["Sun Hour"].values.reshape(-1,1),y,copy=False)
```

Out[41]:

```
array([0.22143433])
```

In [42]:

```
mutual_info_regression(X["Moon Illumunation"].values.reshape(-1,1),y,copy=False)
```

Out[42]:

```
array([0.28797688])
```

In [43]:

```
mutual_info_regression(X["DewPoint"].values.reshape(-1,1),y,copy=False)
```

Out[43]:

```
array([0.11709773])
```

In [44]:

```
mutual_info_regression(X["WindChillC"].values.reshape(-1,1),y,copy=False)
```

Out[44]:

```
array([0.13094431])
```

In [45]:

```
mutual_info_regression(X["WindGust"].values.reshape(-1,1),y,copy=False)
```

Out[45]:

```
array([0.26374641])
```

In [46]:

```
mutual_info_regression(X["Humidity"].values.reshape(-1,1),y,copy=False)
```

Out[46]:

```
array([0.17060383])
```

In [48]:

```
mutual_info_regression(X["Pressure"].values.reshape(-1,1),y,copy=False)
```

Out[48]:

```
array([0.11174303])
```

In [69]:

```
mutual_info_regression(X["Density"].values.reshape(-1,1),y,copy=False)
```

Out[69]:

```
array([0.58079689])
```

In [1]:

```
#For RainMM ve Visiblity the code did not work because there is too much repeated values.
```

In [2]:

```
#RFE
```

In [111]:

```
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import ElasticNet
from sklearn.feature_selection import RFE
```

In [112]:

```
lin=LinearRegression()
```

In [113]:

```
ridge_reg=Ridge()
```

In [114]:

```
elastic=ElasticNet()
```

In [115]:

```
rfe_lin=RFE(lin,n_features_to_select=5)
```

In [116]:

```
rfe_ridge=RFE(ridge_reg,n_features_to_select=5)
```

In [117]:

```
rfe_elastic=RFE(elastic,n_features_to_select=5)
```

In [118]:

```
rfe_lin.fit(X,y)
```

Out[118]:

```
RFE(estimator=LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
                               normalize=False),
    n_features_to_select=5, step=1, verbose=0)
```

In [119]:

```
rfe_lin_table=pd.DataFrame(rfe_lin.support_).T
```

In [120]:

```
rfe_lin_table.columns=X.columns
```

In [122]:

```
rfe_lin_table.T
```

Out[122]:

	0
Wind Speed (m/s)	True
Theoretical_Power_Curve (KWh)	True
Wind Direction (°)	False
Month	False
Day/Night	False
Temp	True
Sun Hour	False
Moon Illumunation	False
DewPoint	False
WindChillC	False
WindGust	False
Humidity	False
RainMM	False
Pressure	True
Visibility	False
Density	True

In [124]:

```
rfe_ridge.fit(X,y)
```

Out[124]:

```
RFE(estimator=Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
                    normalize=False, random_state=None, solver='auto',
                    tol=0.001),
    n_features_to_select=5, step=1, verbose=0)
```

In [125]:

```
rfe_ridge_table=pd.DataFrame(rfe_ridge.support_).T
```

In [126]:

```
rfe_ridge_table.columns=X.columns
```

In [127]:

```
rfe_ridge_table.T
```

Out[127]:

	0
Wind Speed (m/s)	True
Theoretical_Power_Curve (KWh)	True
Wind Direction (°)	False
Month	False
Day/Night	False
Temp	True
Sun Hour	False
Moon Illumunation	False
DewPoint	False
WindChillC	False
WindGust	False
Humidity	False
RainMM	False
Pressure	True
Visibility	False
Density	True

In [128]:

```
rfe_elastic.fit(X,y)
```

Out[128]:

```
RFE(estimator=ElasticNet(alpha=1.0, copy_X=True, fit_intercept=True,
                         l1_ratio=0.5, max_iter=1000, normalize=False,
                         positive=False, precompute=False, random_state=None,
                         selection='cyclic', tol=0.0001, warm_start=False),
    n_features_to_select=5, step=1, verbose=0)
```

In [129]:

```
rfe_elastic_table=pd.DataFrame(rfe_elastic.support_).T
```

In [130]:

```
rfe_elastic_table.columns=X.columns
```

In [131]:

```
rfe_elastic_table.T
```

Out[131]:

	0
Wind Speed (m/s)	True
Theoretical_Power_Curve (KWh)	True
Wind Direction (°)	False
Month	False
Day/Night	False
Temp	False
Sun Hour	False
Moon Illumunation	False
DewPoint	False
WindChillC	False
WindGust	False
Humidity	False
RainMM	False
Pressure	True
Visibility	True
Density	True

In [58]:

```
#Linear Regression Experiments
```

In [161]:

```
#Experiments with train-test split with 0.33
from sklearn.model_selection import train_test_split
```

In [162]:

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.33,random_state=15)
```

In [163]:

```
linear_reg=LinearRegression()
```

In [164]:

```
#Experiment with all features
```

In [165]:

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

Out[165]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [166]:

```
lin1_pred=linear_reg.predict(X_test)
```

In [167]:

```
linear_reg.score(X_test,y_test)
```

Out[167]:

```
0.9608788496965542
```

In [168]:

```
mse_lin_1=mean_squared_error(y_test,lin1_pred)
```

In [169]:

```
mse_lin_1
```

Out[169]:

```
67580.3165861757
```

In [170]:

```
mae_lin_1=mean_absolute_error(y_test,lin1_pred)
```

In [171]:

```
mae_lin_1
```

Out[171]:

```
133.84482314549754
```

In [234]:

```
from sklearn.model_selection import cross_validate
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict
import numpy as np
```

In [174]:

```
#Cross Validate with all features
?cross_validate
```

In [236]:

```
cv_lin_1=cross_val_score(linear_reg,X,y,cv=10)
```

In [237]:

```
np.mean(cv_lin_1)
```

Out[237]:

```
0.9641296611442135
```

In [246]:

```
cv_lin_1=cross_val_score(linear_reg,X,y,cv=10,scoring="neg_mean_squared_error")
```

In [247]:

```
np.mean(cv_lin_1)
```

Out[247]:

```
-64950.62681669132
```

In [243]:

```
cv_lin_1=cross_val_score(linear_reg,X,y,cv=10,scoring="neg_mean_absolute_error")
```

In [244]:

```
np.mean(cv_lin_1)
```

Out[244]:

```
-134.51185084712873
```

In [209]:

```
#Hold out with all features
```

In [215]:

```
cv_lin_1=cross_validate(linear_reg,X_train,y_train,cv=10,return_estimator=True)
```

In [216]:

```
cv_lin_1
```

Out[216]:

```
{'fit_time': array([0.01905084, 0.01689649, 0.01606917, 0.01198673, 0.01349425,
        0.01359916, 0.0121994 , 0.01717567, 0.01496029, 0.01273346]),
 'score_time': array([0.00175118, 0.0009954 , 0.00382066, 0.00099683, 0.0009973 ,
        0.0009973 , 0.00197506, 0.00222349, 0.00185966, 0.00117159]),
 'estimator': (LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False),
  LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False),
  LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False),
  LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False),
  LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False),
  LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False),
  LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False),
  LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False),
  LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False),
  LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)),
 'test_score': array([0.9716157 , 0.96183573, 0.96983781, 0.96650251, 0.9600952 ,
        0.96784038, 0.96544156, 0.96641929, 0.96170151, 0.97288547])}
```

In [230]:

```
r_scores_sep_test=[]
mse_sep_test=[]
mae_sep_test=[]
for items in cv_lin_1["estimator"]:
    print("R scores for seperated test set",items.score(X_test,y_test))
    print("Mse for seperated test set",mean_squared_error(y_test,items.predict(X_test)))
    print("Mae for seperated test set",mean_absolute_error(y_test,items.predict(X_test)))
    r_scores_sep_test.append(items.score(X_test,y_test))
    mse_sep_test.append(mean_squared_error(y_test,items.predict(X_test)))
    mae_sep_test.append(mean_absolute_error(y_test,items.predict(X_test)))
```

```
R scores for seperated test set 0.9608650668857357
Mse for seperated test set 67604.12587376895
Mae for seperated test set 133.89707125261208
R scores for seperated test set 0.9608303756404215
Mse for seperated test set 67664.05369600632
Mae for seperated test set 133.27397290189884
R scores for seperated test set 0.960855708314368
Mse for seperated test set 67620.29245401832
Mae for seperated test set 134.3680188465835
R scores for seperated test set 0.9608827593420465
```

```
R scores for seperated test set 0.9608827593420465
Mse for seperated test set 67573.562820551
Mae for seperated test set 134.081916286209
R scores for seperated test set 0.9608248419512024
Mse for seperated test set 67673.612936121
Mae for seperated test set 133.2804997257625
R scores for seperated test set 0.9608851108414803
Mse for seperated test set 67569.50069367191
Mae for seperated test set 134.0596765540021
R scores for seperated test set 0.9609189834877491
Mse for seperated test set 67510.9869705146
Mae for seperated test set 133.88206360345663
R scores for seperated test set 0.960905122971038
Mse for seperated test set 67534.93049211644
Mae for seperated test set 133.97578171942405
R scores for seperated test set 0.9608725532321073
Mse for seperated test set 67591.19349182337
Mae for seperated test set 133.25732700309268
R scores for seperated test set 0.9609096354840477
Mse for seperated test set 67527.1352955173
Mae for seperated test set 134.49283120547997
```

In [231]:

```
r_scores_sep_test
```

Out[231]:

```
[0.9608650668857357,
 0.9608303756404215,
 0.960855708314368,
 0.9608827593420465,
 0.9608248419512024,
 0.9608851108414803,
 0.9609189834877491,
 0.960905122971038,
 0.9608725532321073,
 0.9609096354840477]
```

In [232]:

```
mse_sep_test
```

Out[232]:

```
[67604.12587376895,
 67664.05369600632,
 67620.29245401832,
 67573.562820551,
 67673.612936121,
 67569.50069367191,
 67510.9869705146,
 67534.93049211644,
 67591.19349182337,
 67527.1352955173]
```

In [233]:

```
mae_sep_test
```

Out[233]:

```
[133.89707125261208,
 133.27397290189884,
 134.3680188465835,
 134.081916286209,
 133.2804997257625,
 134.0596765540021,
 133.88206360345663,
 133.97578171942405,
 133.25732700309268,
 134.49283120547997]
```

In [248]:

```
np.mean(r_scores_sep_test)
```

Out[248]:

```
0.9608750158150198
```

In [249]:

```
np.mean(mae_sep_test)
```

Out[249]:

```
133.85691590985215
```

In [250]:

```
np.mean(mse_sep_test)
```

Out[250]:

```
67586.93947241092
```

In [266]:

```
#Linear Regression with 5 Features from RFE a)Split
```

In [252]:

```
X_rfe_lin=X[["Wind Speed (m/s)","Theoretical_Power_Curve (KWh)","Temp","Pressure","Density"]]
```

In [256]:

```
X_rfe_lin_train,X_rfe_lin_test,y_rfe_lin_train,y_rfe_lin_test=train_test_split(X_rfe_lin,y,test_siz
e=0.33,random_state=15)
```

In [259]:

```
rfe_lin_reg=LinearRegression()
```

In [260]:

```
rfe_lin_reg.fit(X_rfe_lin_train,y_rfe_lin_train)
```

Out[260]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [261]:

```
rfe_lin_reg.score(X_rfe_lin_test,y_rfe_lin_test)
```

Out[261]:

```
0.9599217794020437
```

In [262]:

```
mse_rfe_lin=mean_squared_error(y_rfe_lin_test,rfe_lin_reg.predict(X_rfe_lin_test))
```

In [263]:

```
mse_rfe_lin
```

Out[263]:

```
69233.61954369482
```

In [264]:

```
mae_rfe_lin=mean_absolute_error(y_rfe_lin_test,rfe_lin_reg.predict(X_rfe_lin_test))
```

In [265]:

```
mae_rfe_lin
```

Out[265]:

```
134.15692767321013
```

In [267]:

```
#b) Cross Validation
```

In [ ]:

```
```

In [270]:

```
cv_rfe_lin_reg=cross_val_score(LinearRegression(),X_rfe_lin,y,cv=10)
```

In [272]:

```
np.mean(cv_rfe_lin_reg)
```

Out[272]:

```
0.9649137686473788
```

In [273]:

```
cv_rfe_lin_reg=cross_val_score(LinearRegression(),X_rfe_lin,y,cv=10,scoring="neg_mean_squared_error
")
```

In [274]:

```
np.mean(cv_rfe_lin_reg)
```

Out[274]:

```
-64146.29012080049
```

In [275]:

```
cv_rfe_lin_reg=cross_val_score(LinearRegression(),X_rfe_lin,y,cv=10,scoring="neg_mean_absolute_erro
r")
```

In [276]:

```
np.mean(cv_rfe_lin_reg)
```

Out[276]:

```
-132.79314956381745
```

In [277]:

```
#c) RFE hybrid (hold out set method)
```

In [282]:

```
cv_rfe_lin_hybrid=cross_validate(LinearRegression(),X_rfe_lin_train,y_rfe_lin_train,return_estimato
r=True,cv=10)
```

In [283]:

```
cv_rfe_lin_hybrid
```

Out[283]:

```
{'fit_time': array([0.00698113, 0.00598431, 0.00598383, 0.00782895, 0.00598431,
        0.00598288, 0.00559735, 0.00784564, 0.00513792, 0.004987  ]),
 'score_time': array([0.00099683, 0.00118828, 0.00114751, 0.00117874, 0.00119114,
        0.00180173, 0.00280595, 0.00199604, 0.00084615, 0.00199413]),
 'estimator': (LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False),
  LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False),
  LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False),
  LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False),
  LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False),
  LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False),
  LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False),
  LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False),
  LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False),
  LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)),
 'test_score': array([0.97042718, 0.96057805, 0.96906978, 0.96523387, 0.95849946,
        0.96704747, 0.96322945, 0.96505575, 0.95951949, 0.9720676 ])}
```

In [284]:

```
r_scores_rfe_hybrid=[]
mse_rfe_hybrid=[]
mae_rfe_hybrid=[]
for items in cv_rfe_lin_hybrid["estimator"]:
    r_scores_rfe_hybrid.append(items.score(X_rfe_lin_test,y_rfe_lin_test))
    mse_rfe_hybrid.append(mean_squared_error(y_rfe_lin_test,items.predict(X_rfe_lin_test)))
    mae_rfe_hybrid.append(mean_absolute_error(y_rfe_lin_test,items.predict(X_rfe_lin_test)))
```

In [285]:

```
np.mean(r_scores_rfe_hybrid)
```

Out[285]:

```
0.9599207690350944
```

In [286]:

```
np.mean(mse_rfe_hybrid)
```

Out[286]:

```
69235.36491461968
```

In [287]:

```
np.mean(mae_rfe_hybrid)
```

Out[287]:

```
134.16011772608158
```

In [288]:

```
#Linear Regression with 3 Features (Wind Speed Theoretical Power Wind Direction)
```

In [289]:

```
lin_reg_3f=LinearRegression()
```

In [290]:

```
X_3f=X[["Wind Speed (m/s)","Wind Direction (°)","Theoretical_Power_Curve (KWh)"]]
```

In [291]:

```
X_3f_train,X_3f_test,y_3f_train,y_3f_test=train_test_split(X_3f,y,test_size=0.33,random_state=15)
```

In [292]:

```
lin_reg_3f.fit(X_3f_train,y_3f_train)
```

Out[292]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [295]:

```
lin_reg_3f.score(X_3f_test,y_3f_test)
```

Out[295]:

```
0.9595664505864704
```

In [296]:

```
mse_lin_3f=mean_squared_error(y_3f_test,lin_reg_3f.predict(X_3f_test))
```

In [297]:

```
mse_lin_3f
```

Out[297]:

```
69847.43671579675
```

In [298]:

```
mae_lin_3f=mean_absolute_error(y_3f_test,lin_reg_3f.predict(X_3f_test))
```

In [300]:

```
mae_lin_3f
```

Out[300]:

```
129.9940697057869
```

In [ ]:

```
cv_3f_lin=cross_val_score(LinearRegression(),X_3f,y,cv=10)
```

In [ ]:

```
cv_3f_lin
```

In [305]:

```
np.mean(cv_3f_lin)
```

Out[305]:

0.9650049864981358

In [306]:

```
cv_3f_lin=cross_val_score(LinearRegression(),X_3f,y,cv=10,scoring="neg_mean_squared_error")
```

In [307]:

```
np.mean(cv_3f_lin)
```

Out[307]:

-64235.109535094794

In [308]:

```
cv_3f_lin=cross_val_score(LinearRegression(),X_3f,y,cv=10,scoring="neg_mean_absolute_error")
```

In [309]:

```
np.mean(cv_3f_lin)
```

Out[309]:

-128.79191467315366

In [315]:

```
cv_3f_hybrid=cross_validate(LinearRegression(),X_3f_train,y_3f_train,return_estimator=True,cv=10)
```

In [316]:

```
cv_3f_hybrid
```

Out[316]:

```
{'fit_time': array([0.00705028, 0.00658441, 0.00476384, 0.00299215, 0.0032115 ,
        0.00399017, 0.00299382, 0.00299335, 0.00398827, 0.00282741]),
 'score_time': array([0.00099778, 0.00120234, 0.00080872, 0.00118494, 0.00082755,
        0.0009973 , 0.00119996, 0.00199389, 0.00100613, 0.0014472 ]),
 'estimator': (LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False),
  LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False),
  LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False),
  LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False),
  LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False),
  LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False),
  LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False),
  LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False),
  LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False),
  LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)),
 'test_score': array([0.97048562, 0.96031756, 0.96897358, 0.96519456, 0.95807521,
        0.96668922, 0.96299346, 0.96461855, 0.95912712, 0.97191798])}
```

In [321]:

```
r_scores_3f_hybrid=[]
mse_3f_hybrid=[]
mae_3f_hybrid=[]
for items in cv_3f_hybrid["estimator"]:
    r_scores_3f_hybrid.append(items.score(X_3f_test,y_3f_test))
    mse_3f_hybrid.append(mean_squared_error(y_3f_test,items.predict(X_3f_test)))
    mae_3f_hybrid.append(mean_absolute_error(y_3f_test,items.predict(X_3f_test)))
```

In [322]:

```
np.mean(r_scores_3f_hybrid)
```

Out[322]:

0.9595659773258449

In [323]:

```
np.mean(mse_3f_hybrid)
```

Out[323]:

69848.25425573745

In [324]:

```
np.mean(mae_3f_hybrid)
```

Out[324]:

129.99607150460488

In [ ]:

```

```