DataScience Project

ENERGY PRODUCTION PREDICTION

PROJECT MENTOR
MR.Varun Vennelaganti

Project Details

Team Members

- KONGARI AARTHI
  kongariaarthi22@gmail.com
- VAKA DURGA RUSHIPRIYA
  rushipriyavalmiki@gmail.com
- SHAIK MOULALI.
  moulali1322@gmail.com
- VEMISHETTY SNEHA
  vemishettysneha99@gmail.com
- MALOTH SUDHAKAR
  malothsudhakar2001@gmail.com

Table of

CONTENTS

INTRODUCTION

A combined-cycle power plant comprises gas turbines, steam turbines, and heat recovery steam generators. In this type of plant, the electricity is generated by gas and steam turbines combined in one cycle. Then, it is transferred from one turbine to another. We have to model the energy generated as a function of exhaust vacuum and ambient variables and use that model to improve the plants performance.

Project Architecture

Business understanding

Exploring and transforming raw data through statistical and computational techniques to extract meaningful patterns, enabling informed decision-making and predictive modeling.

Model Building

Constructing algorithms that learn from data to make predictions or decisions.

Collection of data

Gathering and organizing information into a structured dataset for analysis or reference.

Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) involves visually and quantitatively inspecting, summarizing, and understanding data's main characteristics to uncover patterns, anomalies, and insights for further analysis.

Deployment

Making developed models or software available and functional for users in real-world scenarios.

Data Set Details

This is a project where the variable to be predicted is energy production
The data file contains 9568 observations with five variables collected from a combined cycle power plant over six years when the power plant was set to work with a full load.

01 **Temperature (in degrees Celsius).**

02 **exhaust_vacuum, in cm Hg.**

03 **amb_pressure, in millibar. (Ambient pressure)**

04 **r_humidity, in percentage. (Relative humidity)**

05 **energy_production, in MW, net hourly electrical energy output.**

Exploratory Data Analysis (EDA)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib

#Model
import statsmodels.formula.api as smf
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import BaggingRegressor
from sklearn.ensemble import AdaBoostRegressor
from xgboost import XGBRegressor
from sklearn.svm import SVR
from sklearn import metrics
from sklearn.neighbors import KNeighborsRegressor

#AutoML TPot
from tpot import TPOTRegressor

from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV,KFold

import warnings
warnings.filterwarnings("ignore")  #--to ignore warnings
```

```
[ ]  file_path = '/content/energy_production.csv'

     df = pd.read_csv(file_path, sep=';')
```

```
[ ]  df.head()
```

	temperature	exhaust_vacuum	amb_pressure	r_humidity	energy_production
0	9.59	38.56	1017.01	60.10	481.30
1	12.04	42.34	1019.72	94.67	465.36
2	13.87	45.08	1024.42	81.69	465.48
3	13.72	54.30	1017.89	79.08	467.05
4	15.14	49.64	1023.78	75.00	463.58

EDA

```
#Checking for Null values
df.isnull().sum()
```

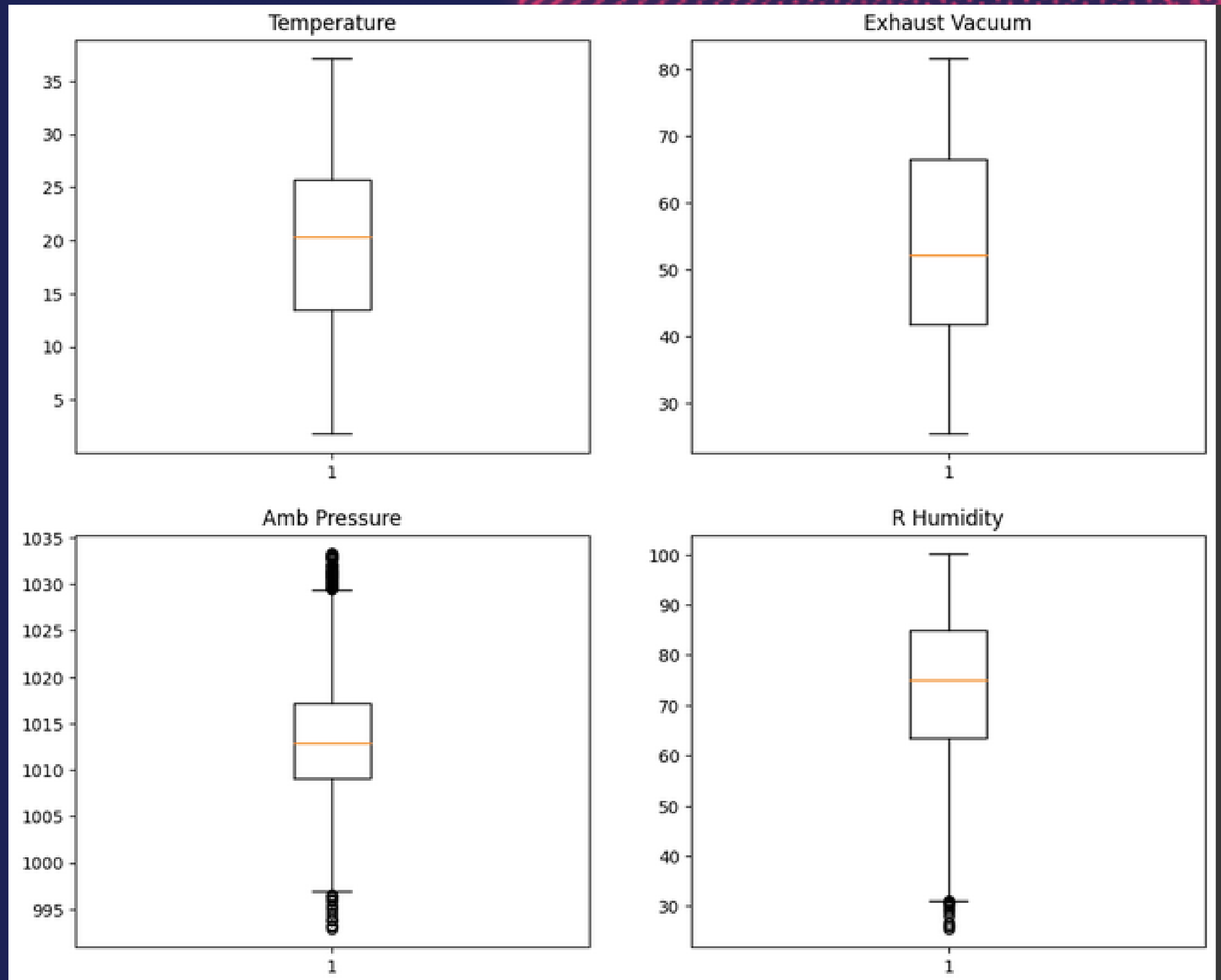
```
temperature          0
exhaust_vacuum       0
amb_pressure         0
r_humidity           0
energy_production    0
dtype: int64
```

INFERENCE :There are no NULL Values in the dataset

VISUALIZATIONS

CHECKING FOR OUTLIERS USING BOXPLOT:

VISUALIZATIONS, LIKE BOXPLOTS, PROVIDE A GRAPHICAL WAY TO IDENTIFY OUTLIERS IN DATA BY DISPLAYING THE DISTRIBUTION AND POTENTIAL EXTREME VALUES, AIDING IN DATA ANALYSIS AND ANOMALY DETECTION.



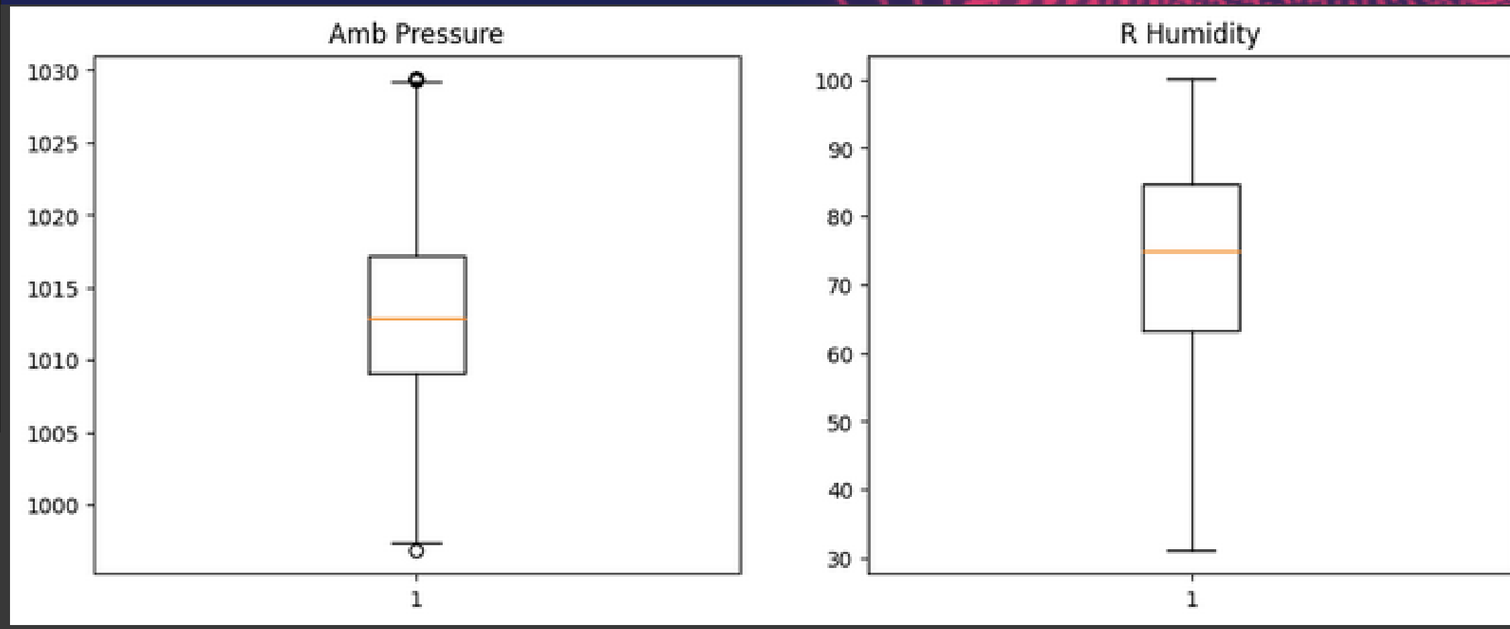
REMOVING OUTLIERS USING IQR METHOD

```
# Calculating IQR
IQR_amb = 1017.26 - 1009.10
IQR_r = 84.83 - 63.335
#printing
print("IQR for amb_pressure is: ",IQR_amb)
print("IQR for r_humidity is: ",IQR_r)

#Calculating Lower & Upper Extreme for amb_pressure
LE_amb = 1009.10 - IQR_amb * 1.5
UE_amb = 1017.26 + IQR_amb * 1.5
#printing
print("Lower Extreme of amb_pressure is: ",LE_amb)
print("Upper Extreme of amb_pressure is: ",UE_amb)

#Calculating Lower & Upper Extreme for r_humidity
LE_r = 63.335 - IQR_r * 1.5
UE_r = 81.56 + IQR_r * 1.5
#printing
print("Lower Extreme of r_humidity is: ",LE_r)
print("Upper Extreme of r_humidity is: ",UE_r)
```

```
IQR for amb_pressure is:  8.159999999999968
IQR for r_humidity is:  21.494999999999997
Lower Extreme of amb_pressure is:  996.8600000000001
Upper Extreme of amb_pressure is:  1029.5
Lower Extreme of r_humidity is:  31.09250000000001
Upper Extreme of r_humidity is:  113.8025
```

Applying the Interquartile Range (IQR) method involves detecting and subsequently removing data points that fall beyond a certain range around the median, helping to enhance the robustness of the dataset by mitigating the influence of extreme values.

AFTER REMOVING OUTLIERS



Inference : There are No more Outliers present in the dataset

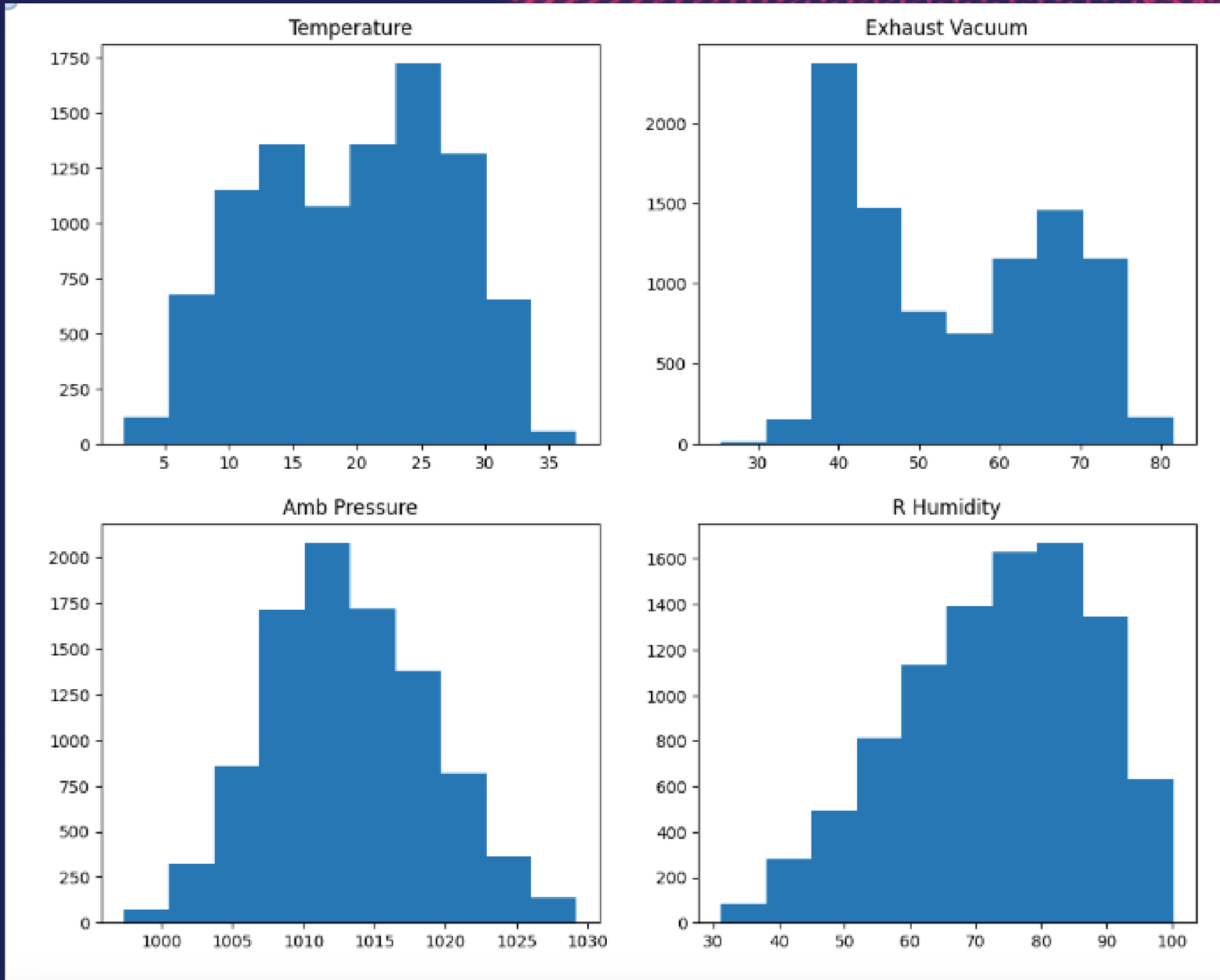
FINAL DATA SET AFTER EDA

```
# Final Dataset after EDA
df5.head()
```

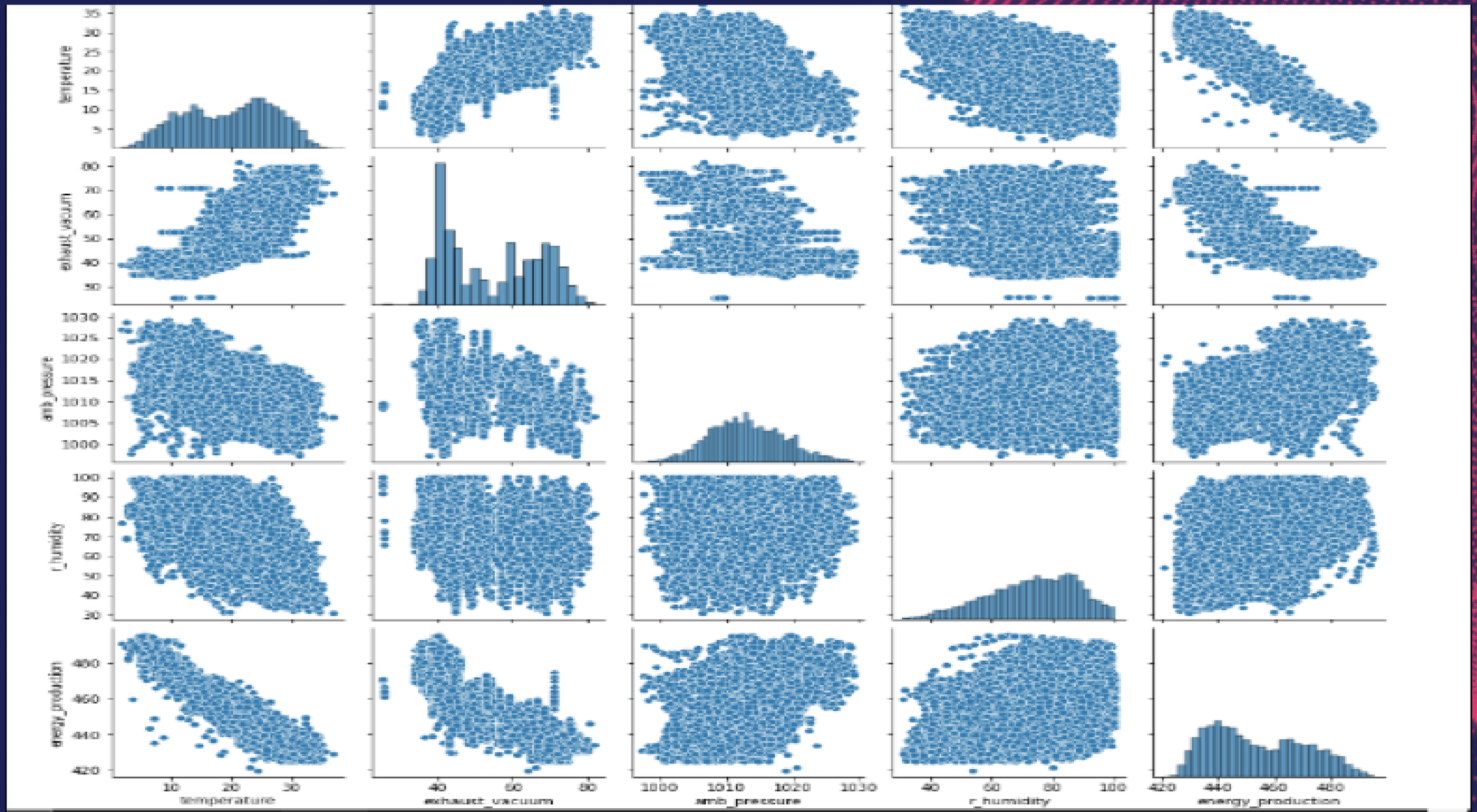
	temperature	exhaust_vacuum	amb_pressure	r_humidity	energy_production
0	9.59	38.56	1017.01	60.10	481.30
1	12.04	42.34	1019.72	94.67	465.36
2	13.87	45.08	1024.42	81.69	465.48
3	13.72	54.30	1017.89	79.08	467.05
4	15.14	49.64	1023.78	75.00	463.58

HISTOGRAM

Inference:
This inference aids in identifying trends and potential outliers, contributing to a better grasp of the data's characteristics and informing decision-making in relevant fields.



CHECKING COLLINEARITY B/W INDEPENDENT VARIABLES



CORRELATION B/W VARIABLES

```
[31] df5.corr()
```

	temperature	exhaust_vacuum	amb_pressure	r_humidity	energy_production
temperature	1.000000	0.842728	-0.508625	-0.542175	-0.947491
exhaust_vacuum	0.842728	1.000000	-0.415389	-0.310217	-0.868693
amb_pressure	-0.508625	-0.415389	1.000000	0.105210	0.521194
r_humidity	-0.542175	-0.310217	0.105210	1.000000	0.388023
energy_production	-0.947491	-0.868693	0.521194	0.388023	1.000000

- Inference: As we can see "temperature" & "exhaust_vaccum" have a strong positve Correlation, So we can say that there is a multicollinearity effect present.

CROSS CHECK WITH VIF

Cross-check with VIF

```
[32] rsq_Tem = smf.ols('temperature~exhaust_vacuum+amb_pressure+r_humidity',data=df5).fit().rsquared
     vif_Tem = 1/(1-rsq_Tem)

     rsq_ex = smf.ols('exhaust_vacuum~temperature+amb_pressure+r_humidity',data=df5).fit().rsquared
     vif_ex = 1/(1-rsq_ex)

     rsq_amb = smf.ols('amb_pressure~temperature+exhaust_vacuum+r_humidity',data=df5).fit().rsquared
     vif_amb = 1/(1-rsq_amb)

     rsq_rh = smf.ols('r_humidity~temperature+exhaust_vacuum+amb_pressure',data=df5).fit().rsquared
     vif_rh = 1/(1-rsq_rh)

     # Storing vif values in a data frame
     d1 = {'Variables':['temperature','exhaust_vacuum','amb_pressure','r_humidity'],'VIF':[vif_Tem,vif_ex,vif_amb,vif_rh]}
     Vif_frame = pd.DataFrame(d1)
     Vif_frame
```

	Variables	VIF
0	temperature	5.906452
1	exhaust_vacuum	3.907032
2	amb_pressure	1.447254
3	r_humidity	1.696703

Inference - There is no multi-collinearity between the variables , since VIF is less than 20.

OVERVIEW REPORT OF EDA

```
[34] import pandas_profiling as pp
```

```
[35] EDA_report= pp.ProfileReport(df5)
     EDA_report.to_file(output_file='report.html')
```

Summarize dataset: 100% 39/39 [00:09<00:00, 2.49it/s, Completed]

Generate report structure: 100% 1/1 [00:06<00:00, 6.42s/it]

Render HTML: 100% 1/1 [00:02<00:00, 2.60s/it]

Export report to file: 100% 1/1 [00:00<00:00, 19.83it/s]

[35]

```
df5
```

	temperature	exhaust_vacuum	amb_pressure	r_humidity	energy_production
0	9.59	38.56	1017.01	60.10	481.30
1	12.04	42.34	1019.72	94.67	465.36
2	13.87	45.08	1024.42	81.69	465.48
3	13.72	54.30	1017.89	79.08	467.05
4	15.14	49.64	1023.78	75.00	463.58
...	...	...	...	...	...
9563	17.10	49.69	1005.53	81.82	457.32
9564	24.73	65.34	1015.42	52.80	446.92
9565	30.44	56.24	1005.19	56.24	429.34
9566	23.00	66.05	1020.61	80.29	421.57
9567	17.75	49.25	1020.86	63.67	454.41

9461 rows × 5 columns

INFERENCE

- ORIGINAL DATA SET CONTAINS - 9568 OBSERVATIONS
- DUPLICATES – 41 OBSERVATIONS
- AFTER REMOVAL OF DUPLICATES - 9527 OBSERVATIONS
- AFTER REMOVAL OF OUTLIERS - 9417

Final Data set Contains – 9418 Observations

MODEL BUILDING

MODELS USED FOR MODEL BUILDING -

- LINEAR REGRESSION
- LASSO AND RIDGE REGRESSION
- DECISION TREE REGRESSION
- RANDOM FOREST REGRESSION
- ADABOOST REGRESSOR
- KNN MODEL
- XGBoost Regressor

SPLITTING X and Y

```
#Splitting X & Y Variable
X = df5.iloc[:,0:4]
Y = df5.iloc[:,4]
```

```
[38] X.head()
```

	temperature	exhaust_vacuum	amb_pressure	r_humidity
0	9.59	38.56	1017.01	60.10
1	12.04	42.34	1019.72	94.67
2	13.87	45.08	1024.42	81.69
3	13.72	54.30	1017.89	79.08
4	15.14	49.64	1023.78	75.00

```
[39] Y.head()
```

```
0    481.30
1    465.36
2    465.48
3    467.05
4    463.58
Name: energy_production, dtype: float64
```

```
[40] # Train & Test Split
     X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.2,random_state=10)
```

01.LINEAR MODEL

Linear Model

```
[41] #Model
     Lmodel = LinearRegression()
     Lmodel.fit(X_train,Y_train)
     Lmodel.score(X_test,Y_test)*100
```

```
93.06433601411435
```

Model Validation

```
[42] #Kfold of Linear model
     kfold = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)
     cross_val_score(LinearRegression(), X, Y, cv=kfold)
```

```
array([0.93055762, 0.92618122, 0.93250536, 0.92817959, 0.92822301])
```

LASSO REGRESSION

```
# Lasso Regression
from sklearn.linear_model import Lasso

# Train the model
lasso = Lasso(alpha = 1)
lasso.fit(X_train, Y_train)
y_pred1 = lasso.predict(X_test)

# Calculate Mean Squared Error
mean_squared_error = np.mean((y_pred1 - Y_test)**2)
print("Mean squared error on test set", mean_squared_error)
lasso_coeff = pd.DataFrame()
lasso_coeff["Columns"] = X_train.columns
lasso_coeff['Coefficient Estimate'] = pd.Series(lasso.coef_)

print(lasso_coeff)
```

```
Mean squared error on test set 21.021888475894894
          Columns  Coefficient Estimate
0     temperature             -1.921902
1  exhaust_vacuum             -0.248816
2    amb_pressure              0.064529
3      r_humidity             -0.144717
```

RIDGE REGRESSION

```
# Ridge regression
from sklearn.linear_model import Ridge

# Train the model
ridgeR = Ridge(alpha = 1)
ridgeR.fit(X_train, Y_train)
y_pred = ridgeR.predict(X_test)

# calculate mean square error
mean_squared_error_ridge = np.mean((y_pred - Y_test)**2)
print("Mean_Squared_error:",mean_squared_error_ridge)

# get ridge coefficient and print them
ridge_coefficient = pd.DataFrame()
ridge_coefficient["Columns"]= X_train.columns
ridge_coefficient['Coefficient Estimate'] = pd.Series(ridgeR.coef_)
print(ridge_coefficient)
```

```
Mean_Squared_error: 21.01452500349596
          Columns  Coefficient Estimate
0     temperature             -1.976951
1  exhaust_vacuum             -0.230662
2    amb_pressure              0.080297
3      r_humidity             -0.160455
```

Decision Tree REGRESSION

```
# Decision Tree Regressor
seed = 7
kfold = KFold(n_splits=10, random_state=seed,shuffle=True)
cart = DecisionTreeRegressor()
num_trees = 100
model = BaggingRegressor(base_estimator=cart, n_estimators=num_trees, random_state=seed)
results = cross_val_score(model, X_train, Y_train, cv=kfold)
print(results.mean())
```

```
0.9599131469522387
```

Random forest REGRESSION

```
# Random Forest Regressor
num_trees = 100
max_features = 3
kfold = KFold(n_splits=10, random_state=7,shuffle=True)
model = RandomForestRegressor(n_estimators=num_trees, max_features=max_features)
results = cross_val_score(model, X_train, Y_train, cv=kfold)
print(results.mean())
```

```
0.9606008110064351
```

Adaboost REGRESSION

```
# Adaboost Regressor
num_trees = 10
seed=7
kfold = KFold(n_splits=10, random_state=seed,shuffle=True)
model = AdaBoostRegressor(n_estimators=num_trees, random_state=seed)
results = cross_val_score(model, X_train, Y_train, cv=kfold)
print(results.mean())
```

```
0.9075739194628539
```

KNN Model

KNN Model

```
[50] n_neighbors = np.array(range(1,41))
     param_grid = dict(n_neighbors=n_neighbors)
     model = KNeighborsRegressor()
     grid = GridSearchCV(estimator=model,param_grid=param_grid)
     grid.fit(X_train,Y_train)
     print(grid.best_score_)
     print(grid.best_params_)
```

```
0.9436345358354734
{'n_neighbors': 5}
```

```
[51] KNN_model = KNeighborsRegressor(n_neighbors=5)
     KNN_model.fit(X_train,Y_train)
```

```
KNeighborsRegressor
KNeighborsRegressor()
```

```
[52] Y_pred = KNN_model.predict(X_test)
     KNN_1 = metrics.explained_variance_score(Y_test,Y_pred)
     print("Accuracy:" ,KNN_1)
```

```
Accuracy: 0.9443999431784652
```

Choosing the best Model

```
[53] from sklearn.linear_model import LinearRegression
     from sklearn.model_selection import GridSearchCV
```

```
import pandas as pd
from sklearn.model_selection import ShuffleSplit, GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge, Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor

def find_best_model_using_gridsearchcv(x, y):
    algos = {
        'linear_regression': {
            'model': LinearRegression(),
            'params': {}
        },
        'ridge': {
            'model': Ridge(),
            'params': {'alpha': [0.1, 0.5, 1]}
        },
        'lasso': {
            'model': Lasso(),
            'params': {'alpha': [0.1, 0.5, 1], 'selection': ['random', 'cyclic']}
        },
        'decision_tree': {
            'model': DecisionTreeRegressor(),
            'params': {'criterion': ['mse', 'friedman_mse'], 'splitter': ['best', 'random']}
        },
        'random_forest': {
            'model': RandomForestRegressor(),
            'params': {'n_estimators': [100, 125, 150, 200], 'max_features': [3, 4]}
        },
        'xgboost': {
            'model': XGBRegressor(),
            'params': {'n_estimators': [100, 125, 150, 200, 225, 250], 'max_depth': [3, 4, 5], 'learning_rate': [0.1, 0.2]}
        },
        'svm': {
            'model': SVR(),
            'params': {'kernel': ['rbf']}
        },
        'knn': {
            'model': KNeighborsRegressor(),
            'params': {'n_neighbors': [5, 10, 15, 20, 25, 30, 35, 40, 45, 50]}
        }
```

CHOOSING BEST MODEL

```
    scores = []
    cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=10)

    for algo_name, config in algos.items():
        gs = GridSearchCV(config['model'], config['params'], cv=cv)
        gs.fit(x, y)
        scores.append({
            'model': algo_name,
            'best_score': gs.best_score_,
            'best_params': gs.best_params_
        })

    return pd.DataFrame(scores, columns=['model', 'best_score', 'best_params'

# Assuming you have your data X and labels Y
result_df = find_best_model_using_gridsearchcv(X, Y)
print(result_df)
```

```
                model  best_score  \
0  linear_regression    0.929326
1              ridge    0.929326
2              lasso    0.929330
3      decision_tree    0.925217
4      random_forest    0.963065
5            xgboost    0.966585
6                svm    0.376926
7                knn    0.945204
```

Random Forest Accuracy:96.38

Deployement

Best Model For Deployment

XGBoost

```
[55] gb_model = XGBRegressor(n_estimators=250,learning_rate=0.2, max_depth=5)
     gb_model.fit(X_train,Y_train)
     gb_model.score(X_test,Y_test)*100
```

```
96.75663719774154
```

```
[56] import pickle
```

```
[57] pickle_out = open('gb_model.pkl','wb')
     pickle.dump(gb_model,pickle_out)
     pickle_out.close()
```

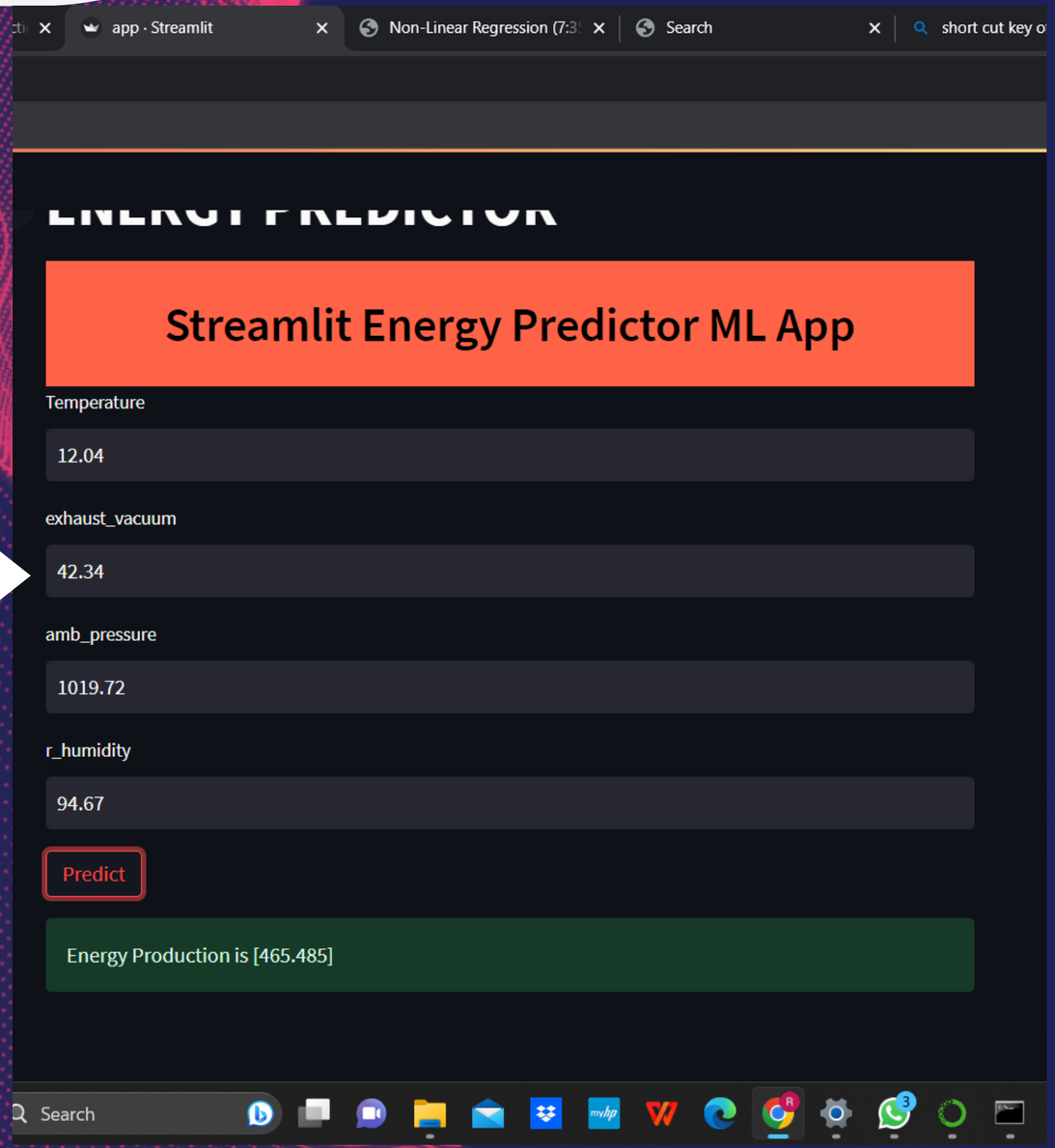
Random Forest

```
[58] rf_model = RandomForestRegressor(n_estimators=10, random_state=20)
     rf_model.fit(X_train,Y_train)
     rf_model.score(X_test,Y_test)*100
```

```
95.93349411371767
```

Pickling the File

```
[59] pickle_out = open('rf_model.pkl','wb')
     pickle.dump(rf_model,pickle_out)
     pickle_out.close()
```




Thank You