Comparison of R2_score and RMSE of different models on air quality dataset

Authors:

Shrutik Pawale
Tanay Shah
Tanvi Venkatesh

Abstract

The dataset contains 9358 instances of hourly averaged responses from an array of 5 metal oxide chemical sensors embedded in an Air Quality Chemical Multisensor Device. The device was located on the field in a significantly polluted area, at road level,within an Italian city.

Data was recorded from March 2004 to February 2005 (one year) representing the longest freely available recordings of on field deployed air quality chemical sensor devices responses. Ground Truth hourly averaged concentrations for CO, Non Metanic Hydrocarbons, Benzene, Total Nitrogen Oxides (NOx) and Nitrogen Dioxide (NO2) and were provided by a co-located reference certified analyzer.
We have implemented multiple regression models and compared their accuracy before and after Principal Component Analysis.

Introduction

Urban atmospheric pollutants are considered responsible for the increased incidence of respiratory illness in citizens, and some of them (e.g. benzene) are known to induce cancers in case of prolonged exposure. Precise estimation of pollutants distribution is hence relevant for traffic management in the municipalities and more generally for the definition of integrated mobility plans designed to face these problems.

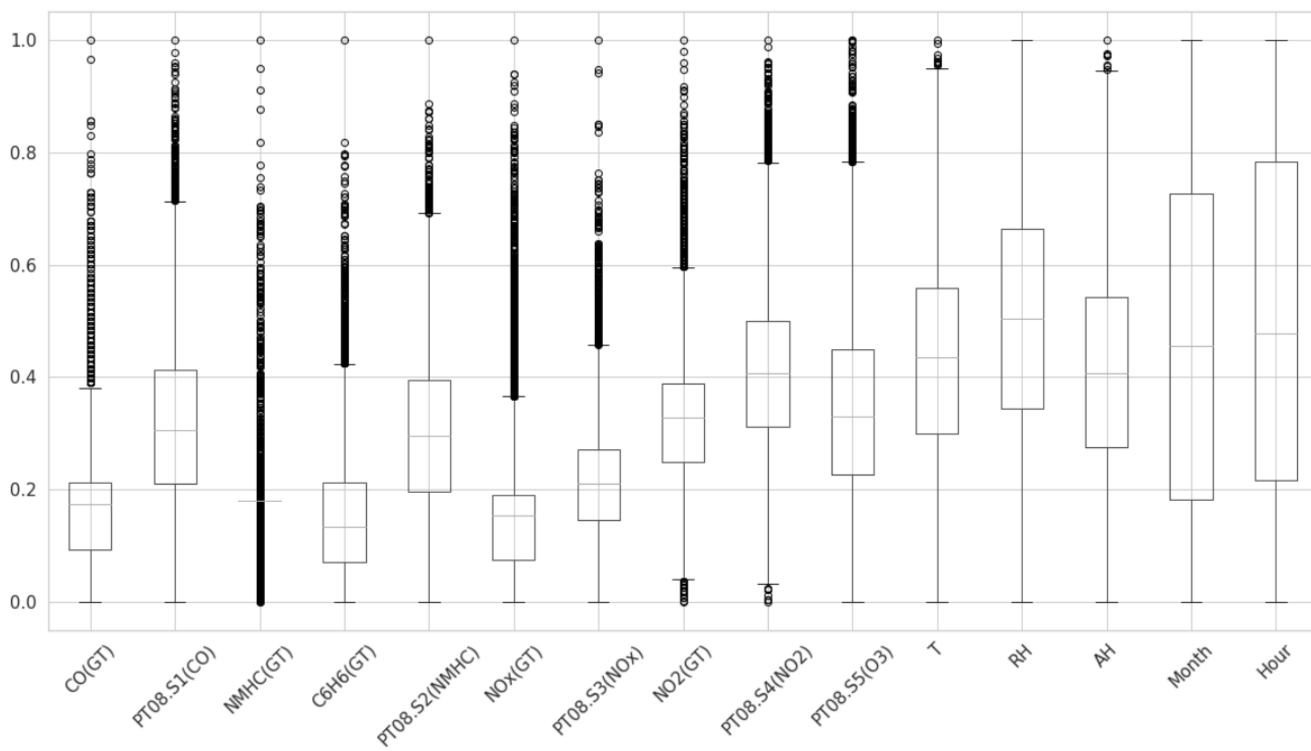
Nowadays, urban air pollution monitoring is primarily carried out by means of networks of spatially distributed fixed stations. These equipments, mostly based on industrial spectrometers, can selectively and precisely estimate the concentrations of many atmospheric pollutants, but their costs and sizes seriously hamper the deployment of adequately dense measurement networks. Unfortunately, pollutants diffusion is heavily affected by atmosphere dynamics and the availability of a limited number of measurement nodes may lead to the misevaluation of the real distribution of gases and particles concentrations in a complex and turbulent environment such as a city

Implementation

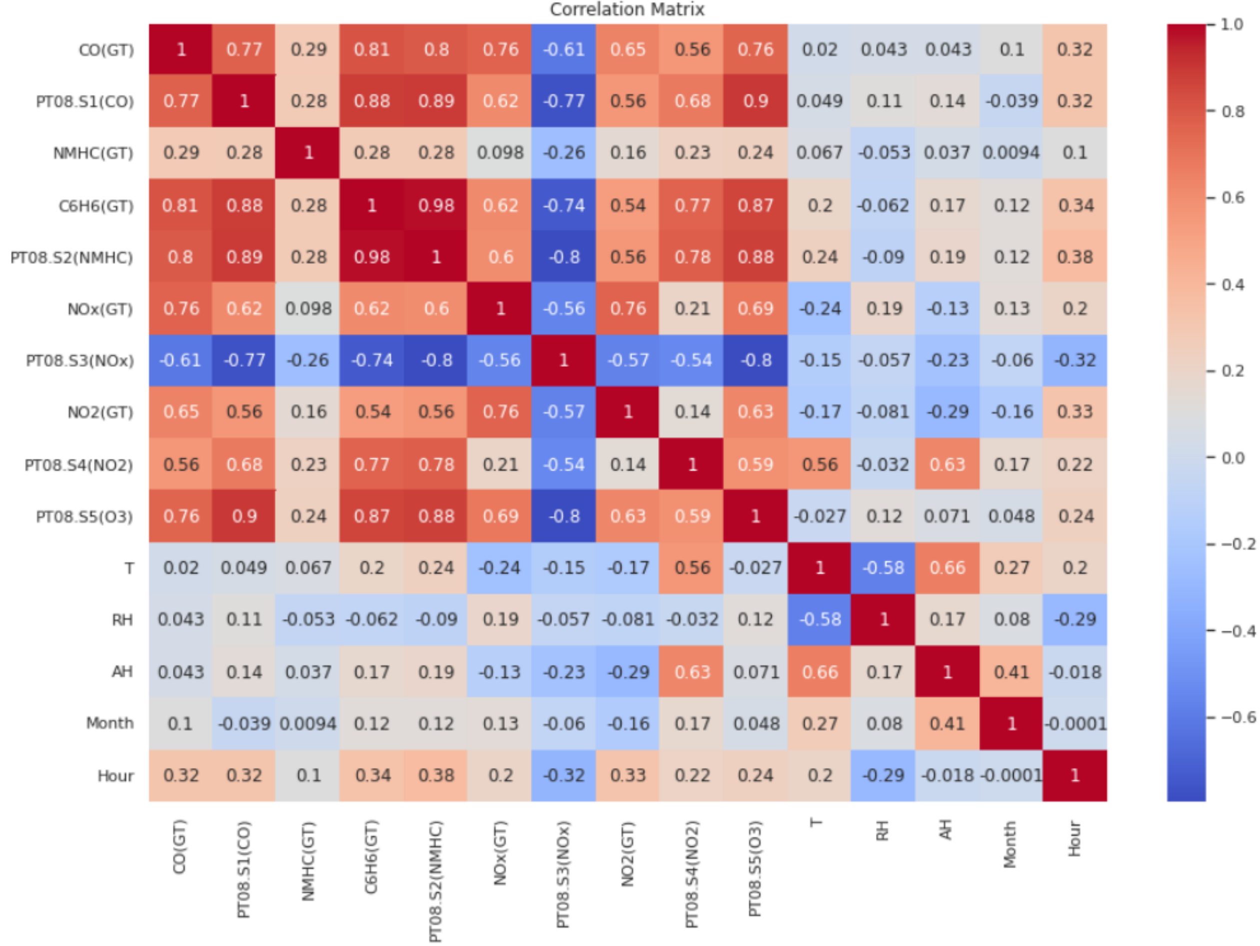
The Dataset has date and time columns, so, we have extracted month and hour.
There is missing data which is represented by NaN and -200. First, we dropped NaN values and then replaced -200 by the mean of features using SimpleImputer.
We have scaled the data using MinMaxScaler in the range (0,1) and plotted the correlation matrix.
Observing the correlation matrix we dropped RH, AH, T, Month, Hour and PT08.S3(NOx) which is NOx targeted tungsten oxdie as they have very low correlation. Having low correlated variables in the training set affects the R2_score of the model. We will also drop NMHC as a lot of the values are close to zero.
We choose PT08.S5(O3) as our target variable. The data is split as 30% test set and rest as training set.
Principal component analysis, or PCA, is a dimensionality-reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set. Reducing the number of variables of a data set naturally comes at the expense of accuracy, but the trick in dimensionality reduction is to trade a little accuracy for simplicity.
The explained variance ratio is the percentage of variance that is attributed by each of the selected components. Ideally, you would choose the number of components to include in your model by adding the explained variance ratio of each component until you reach a total of around 0.8 or 80% to avoid overfitting.
Principal Component Analysis using 2 principal components is done on X_train and total explained variance ratio is obtained as [0.76580657, 0.1961985 , 0.03799493].
LinearRegression, Lasso, Ridge, Logistic, SVD and OLS is done on original data and normalized data.

Observations

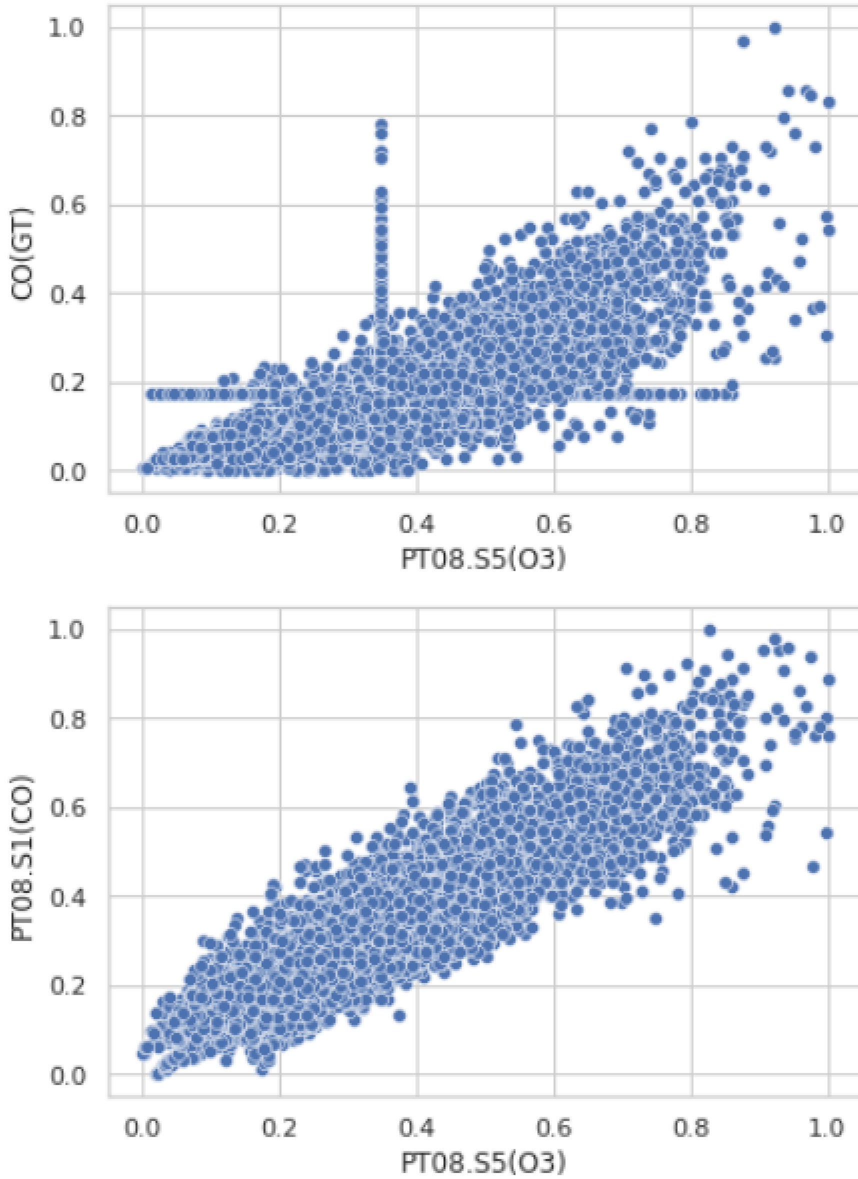
Box Plot



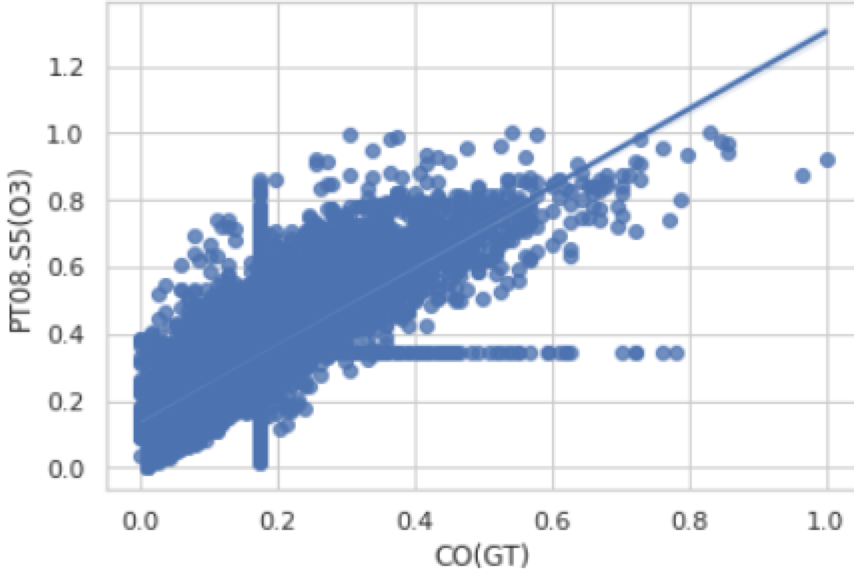
Correlation Matrix

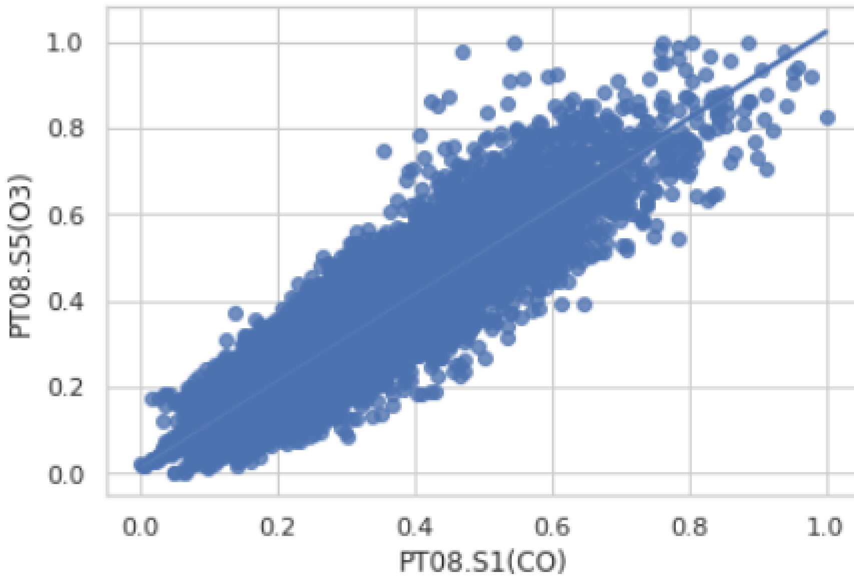


Scatter plot of S5(O3) with CO and S1(CO)



Regression plot of S5(O3) with CO and S1(CO)





RMSE and R2_score of different Models

	MSE	RMSE	R^2
Linear Regression	19895.3696	141.0509	87.2235
Linear Regression using normalization	0.0048	0.0689	83.8280
Lasso Regression	19921.0274	141.1419	87.2070
Lasso Regression using normalization	0.0294	0.1714	-0.0332
Ridge Regression	19895.3812	141.0510	87.2235
Ridge Regression using normalization	0.0048	0.0689	83.8305
Logistic Regression	114274.9316	338.0458	-18.6171
SVD Model	19607.5102	140.0268	87.2418

R2_score and RMSE for ordinary lease squares using statsmodels

	Original Data	Normalized data
R2 for OLS	0.981	0.974
RMSE for OLS	606.660	0.230

Conclusion

From the observations of R2_score and MSE, we can see that R2_score for Linear Regression without normalization is 87.2 but the MSE is 141, but R2_score and MLE after normalization of data is 83.82 and 0.0689, which means the accuracy is reduced but the error is almost 0.
Lasso gave a negative R_2 score which means the chosen model fits worse than a horizontal line. R2 compares the fit of the chosen model with that of a horizontal straight line (the null hypothesis).
Logistic Regression failed to converge as the maximum likelihood estimates can be infinite and the algorithm fails to converge.
The best accuracy and MSE was achieved by Ordinary Least Squares using statsmodels.

References

[1] S. De Vito, E. Massera, M. Piga, L. Martinotto, G. Di Francia, On field calibration of an electronic nose for benzene estimation in an urban pollution monitoring scenario, Sensors and Actuators B: Chemical, Volume 129, Issue 2, 22 February 2008, Pages 750-757, ISSN 0925-4005

[2] Saverio De Vito, Marco Piga, Luca Martinotto, Girolamo Di Francia, CO, NO2 and NOx urban pollution monitoring with on-field calibrated electronic nose by automatic bayesian regularization.

[3] De Vito, G. Fattoruso, M. Pardo, F. Tortorella and G. Di Francia, 'Semi-Supervised Learning Techniques in Artificial Olfaction: A Novel Approach to Classification Problems and Drift Counteraction,' in IEEE Sensors Nov. 2012.

[5] Predicting air pollution level in a specific city by Dan Wei

[6] A Machine Learning Approach to Predict Air Quality in California Mauro Castelli,Fabiana Martins Clemente,Aleš Popovič,2Sara Silva,and Leonardo Vanneschi

Project Code

Feature Information:

0 Date (DD/MM/YYYY)
1 Time (HH.MM.SS)
2 True hourly averaged concentration CO in mg/m^3 (reference analyzer)
3 PT08.S1 (tin oxide) hourly averaged sensor response (nominally CO targeted)

4 True hourly averaged overall Non Metanic HydroCarbons concentration in microg/m^3 (reference analyzer)
5 True hourly averaged Benzene concentration in microg/m^3 (reference analyzer)
6 PT08.S2 (titania) hourly averaged sensor response (nominally NMHC targeted)
7 True hourly averaged NOx concentration in ppb (reference analyzer)
8 PT08.S3 (tungsten oxide) hourly averaged sensor response (nominally NOx targeted)
9 True hourly averaged NO2 concentration in microg/m^3 (reference analyzer)
10 PT08.S4 (tungsten oxide) hourly averaged sensor response (nominally NO2 targeted)
11 PT08.S5 (indium oxide) hourly averaged sensor response (nominally O3 targeted)
12 Temperature in Â°C
13 Relative Humidity (%)
14 AH Absolute Humidity

IMPORT FUNCTIONS

In [230…

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score,accuracy_score, mean_squared_error
import seaborn as sns
sns.set_theme(style="whitegrid")
from sklearn.linear_model import LinearRegression,LogisticRegression,Lasso,Ridge
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler,LabelEncoder
from sklearn.model_selection import train_test_split
import plotly.express as px
from sklearn.decomposition import PCA
from scipy import stats
```

In [231…

```
df = pd.read_csv('AirQualityUCI.csv')
df.head()
```

Out[231]:

	Date	Time	CO(GT)	PT08.S1(CO)	NMHC(GT)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)	PT08.S3(NOx)	N
0	3/10/2004	18:00:00	2.6	1360.0	150.0	11.9	1046.0	166.0	1056.0	
1	3/10/2004	19:00:00	2.0	1292.0	112.0	9.4	955.0	103.0	1174.0	
2	3/10/2004	20:00:00	2.2	1402.0	88.0	9.0	939.0	131.0	1140.0	
3	3/10/2004	21:00:00	2.2	1376.0	80.0	9.2	948.0	172.0	1092.0	
4	3/10/2004	22:00:00	1.6	1272.0	51.0	6.5	836.0	131.0	1205.0	

In [232…

```
df = df.drop(columns = ['Unnamed: 15','Unnamed: 16'])
df.head()
```

Out[232]:

	Date	Time	CO(GT)	PT08.S1(CO)	NMHC(GT)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)	PT08.S3(NOx)	N
0	3/10/2004	18:00:00	2.6	1360.0	150.0	11.9	1046.0	166.0	1056.0	
1	3/10/2004	19:00:00	2.0	1292.0	112.0	9.4	955.0	103.0	1174.0	
2	3/10/2004	20:00:00	2.2	1402.0	88.0	9.0	939.0	131.0	1140.0	
3	3/10/2004	21:00:00	2.2	1376.0	80.0	9.2	948.0	172.0	1092.0	
4	3/10/2004	22:00:00	1.6	1272.0	51.0	6.5	836.0	131.0	1205.0	

In [233…
```
df.isna().sum().sum()
```
Out[233]:
```
1710
```
In [234…
```
df=df.dropna()
```
In [235…
```
df.info()
```
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9357 entries, 0 to 9356
Data columns (total 15 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Date           9357 non-null   object 
 1   Time           9357 non-null   object 
 2   CO(GT)         9357 non-null   float64
 3   PT08.S1(CO)    9357 non-null   float64
 4   NMHC(GT)       9357 non-null   float64
 5   C6H6(GT)       9357 non-null   float64
 6   PT08.S2(NMHC)  9357 non-null   float64
 7   NOx(GT)        9357 non-null   float64
 8   PT08.S3(NOx)   9357 non-null   float64
 9   NO2(GT)        9357 non-null   float64
 10  PT08.S4(NO2)   9357 non-null   float64
 11  PT08.S5(O3)    9357 non-null   float64
 12  T              9357 non-null   float64
 13  RH             9357 non-null   float64
 14  AH             9357 non-null   float64
dtypes: float64(13), object(2)
memory usage: 1.1+ MB
```
In [236…
```
list(df.columns)
```
Out[236]:
```
['Date',
 'Time',
 'CO(GT)',
 'PT08.S1(CO)',
 'NMHC(GT)',
 'C6H6(GT)',
 'PT08.S2(NMHC)',
 'NOx(GT)',
 'PT08.S3(NOx)',
 'NO2(GT)',
 'PT08.S4(NO2)',
 'PT08.S5(O3)',
 'T',
 'RH',
 'AH']
```
In [237…
```
df['Date']=pd.to_datetime(df.Date, dayfirst=False)
df['Month']= df['Date'].dt.month
df['Hour']=df['Time'].apply(lambda x: int(x.split(':')[0]))
df.dtypes
```
Out[237]:
```
Date                 datetime64[ns]
Time                         object
CO(GT)                      float64
PT08.S1(CO)                 float64
NMHC(GT)                    float64
C6H6(GT)                    float64
PT08.S2(NMHC)               float64
NOx(GT)                     float64
PT08.S3(NOx)                float64
NO2(GT)                     float64
```

```
PT08.S4(NO2)            float64
PT08.S5(O3)             float64
T                       float64
RH                      float64
AH                      float64
Month                     int64
Hour                      int64
dtype: object
```

CODE

In [238…
```
df.replace(to_replace= -200, value= np.NaN, inplace= True)
```

In [239…
```
df.isna().sum().sum()
```

Out[239]:
```
16701
```

In [240…
```
df= df.drop(columns=['Date','Time'])
df.head()
```

Out[240]:

	CO(GT)	PT08.S1(CO)	NMHC(GT)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)	PT08.S3(NOx)	NO2(GT)	PT08.S4(NO2
0	2.6	1360.0	150.0	11.9	1046.0	166.0	1056.0	113.0	1692
1	2.0	1292.0	112.0	9.4	955.0	103.0	1174.0	92.0	1559
2	2.2	1402.0	88.0	9.0	939.0	131.0	1140.0	114.0	1554
3	2.2	1376.0	80.0	9.2	948.0	172.0	1092.0	122.0	1584
4	1.6	1272.0	51.0	6.5	836.0	131.0	1205.0	116.0	1490

In [241…
```
imputer = SimpleImputer(missing_values=np.NaN, strategy = 'mean')
df = imputer.fit_transform(df)
df = pd.DataFrame(df, columns = ['CO(GT)','PT08.S1(CO)','NMHC(GT)', 'C6H6(GT)', 'PT08.S2
 'PT08.S5(O3)', 'T', 'RH', 'AH','Month','Hour'])
```

In [242…
```
df.isna().sum().sum()
```

Out[242]:
```
0
```

In [243…
```
df.head()
```

Out[243]:

	CO(GT)	PT08.S1(CO)	NMHC(GT)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)	PT08.S3(NOx)	NO2(GT)	PT08.S4(NO2
0	2.6	1360.0	150.0	11.9	1046.0	166.0	1056.0	113.0	1692
1	2.0	1292.0	112.0	9.4	955.0	103.0	1174.0	92.0	1559
2	2.2	1402.0	88.0	9.0	939.0	131.0	1140.0	114.0	1554
3	2.2	1376.0	80.0	9.2	948.0	172.0	1092.0	122.0	1584
4	1.6	1272.0	51.0	6.5	836.0	131.0	1205.0	116.0	1490

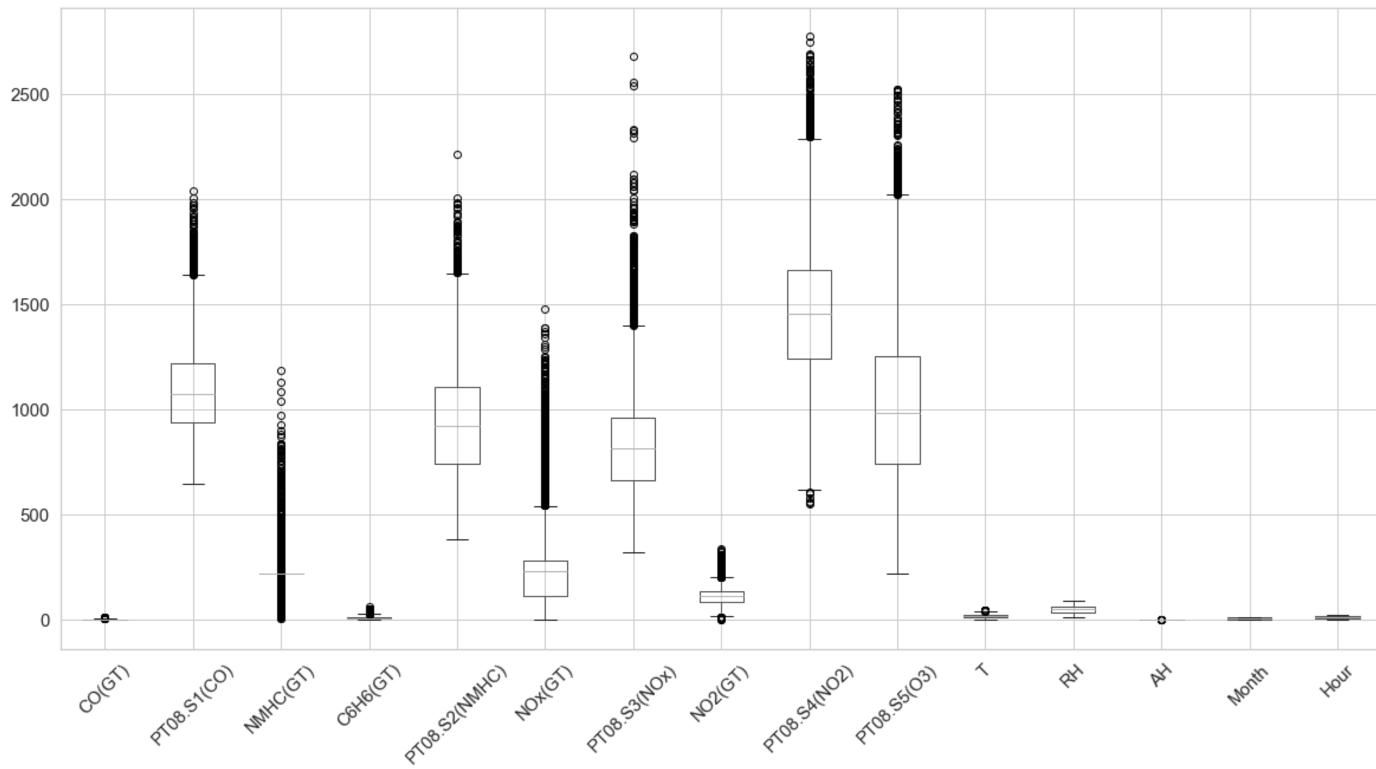
In [244…
```
df.describe()
```

Out[244]:

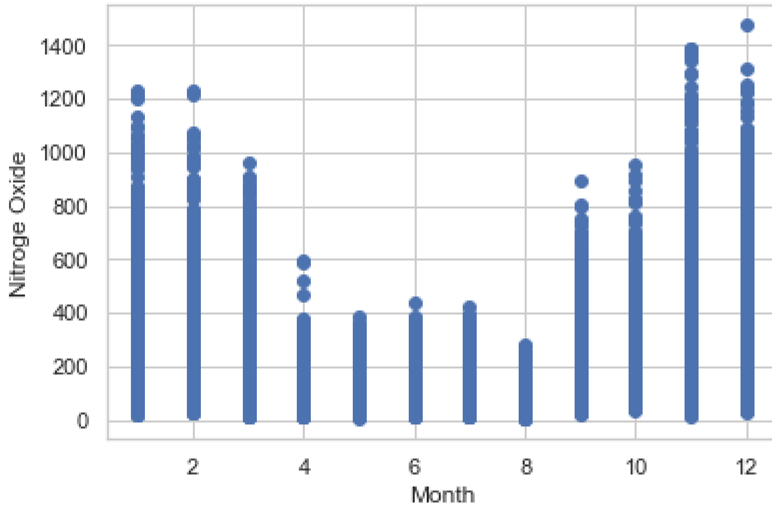
	CO(GT)	PT08.S1(CO)	NMHC(GT)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)	PT08.S3(NOx)	NO2
count	9357.000000	9357.000000	9357.000000	9357.000000	9357.000000	9357.000000	9357.000000	9357.00
mean	2.152750	1099.713158	218.811816	10.083105	939.030252	246.882871	835.370370	113.07

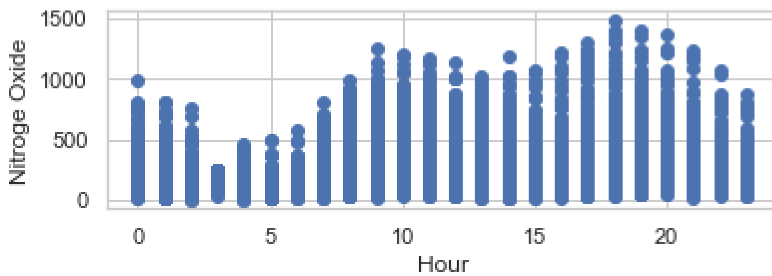
std	1.316068	212.797231	63.870229	7.302650	261.558742	193.423447	251.742814	43.91
min	0.100000	647.000000	7.000000	0.100000	383.000000	2.000000	322.000000	2.00
25%	1.200000	941.000000	218.811816	4.600000	742.000000	112.000000	666.000000	86.00
50%	2.152750	1074.000000	218.811816	8.600000	923.000000	229.000000	818.000000	113.07
75%	2.600000	1221.000000	218.811816	13.600000	1105.000000	284.000000	960.000000	133.00
max	11.900000	2040.000000	1189.000000	63.700000	2214.000000	1479.000000	2683.000000	340.00

In [245…
```
plt.figure(figsize=(20,10))
boxplot = df.boxplot(rot=45,fontsize=15)
```



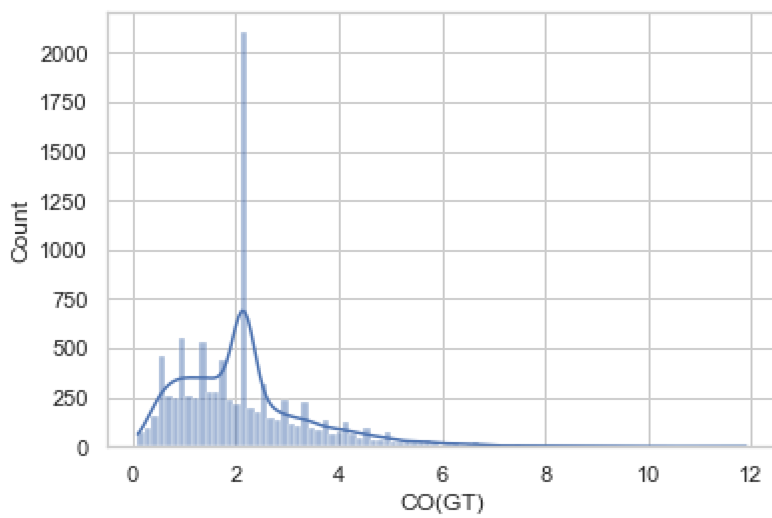
In [246…
```
plt.subplot(1,1,1)
plt.xlabel('Month')
plt.ylabel('Nitroge Oxide')
plt.scatter(df['Month'],df['NOx(GT)'])
plt.show()

plt.subplot(2,1,2)
plt.xlabel('Hour')
plt.ylabel('Nitroge Oxide')
plt.scatter(df['Hour'],df['NOx(GT)'])
plt.show()
```

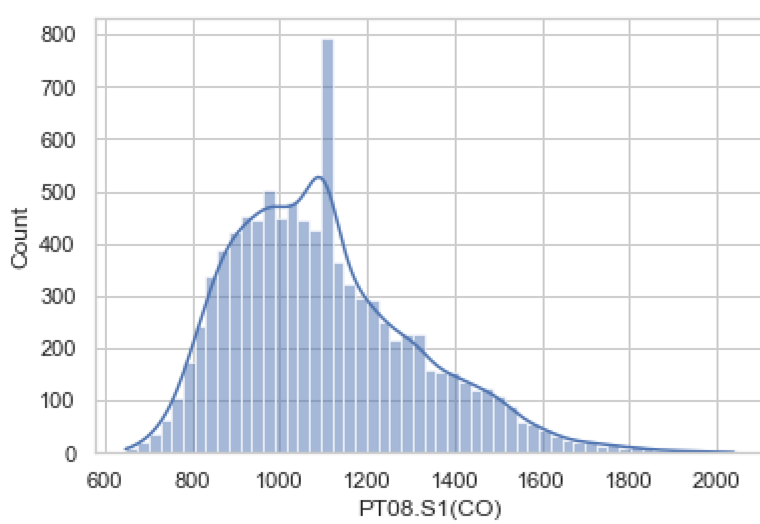


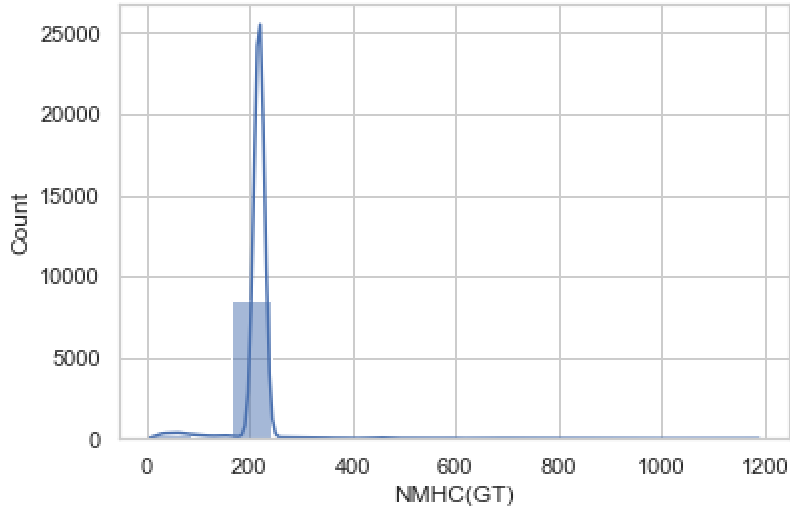


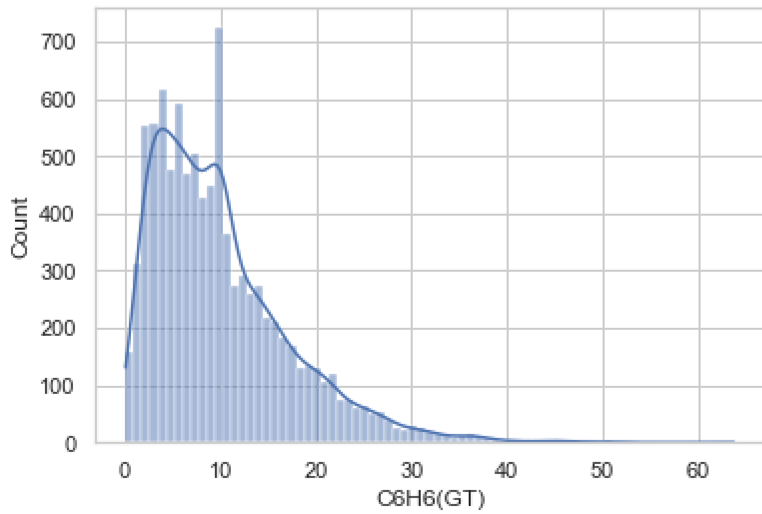
In [247…

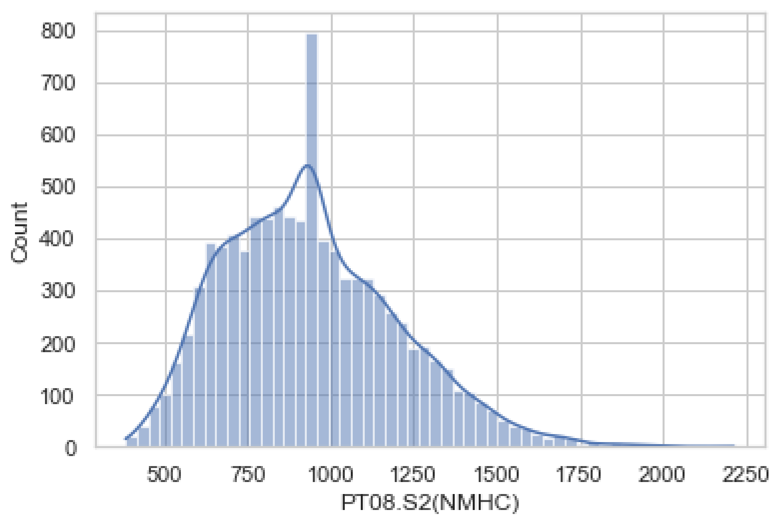
```
for i in df.columns[:13]:
    sns.histplot(df[i],kde=True)
    plt.show()
```

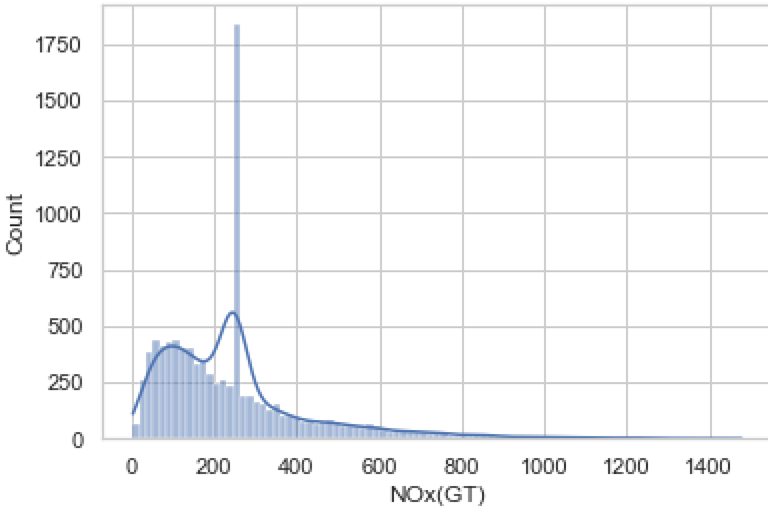


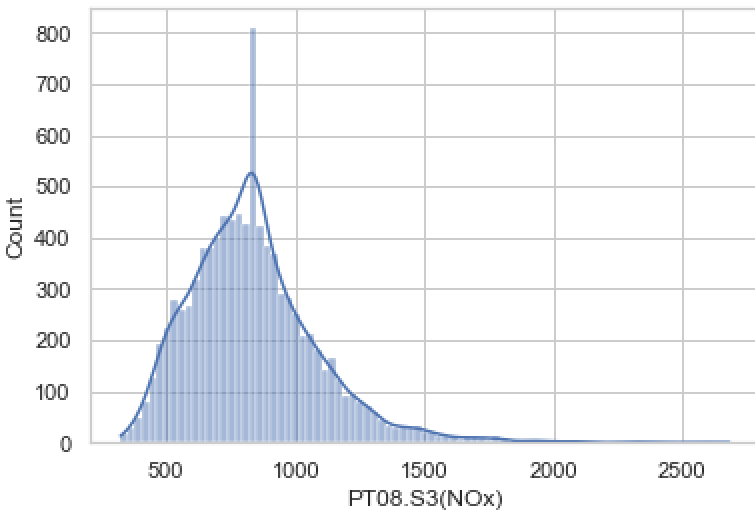


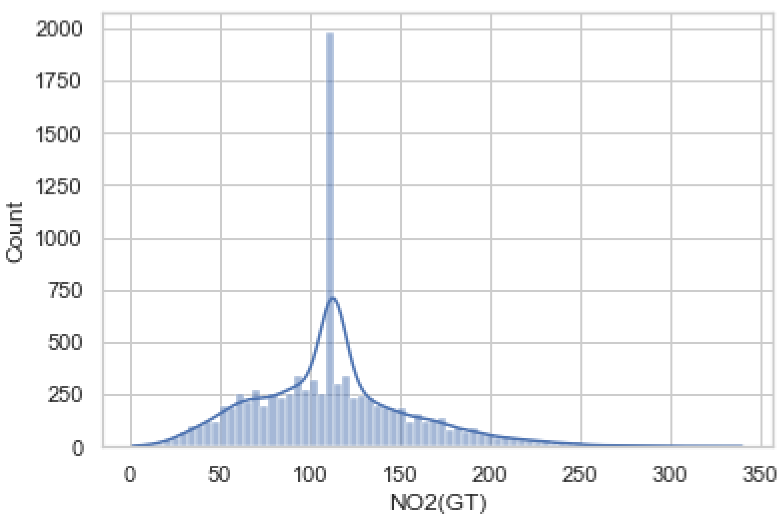

Count
25000
20000
15000
10000
5000
0
0
200
400
600
800
1000
1200
NMHC(GT)

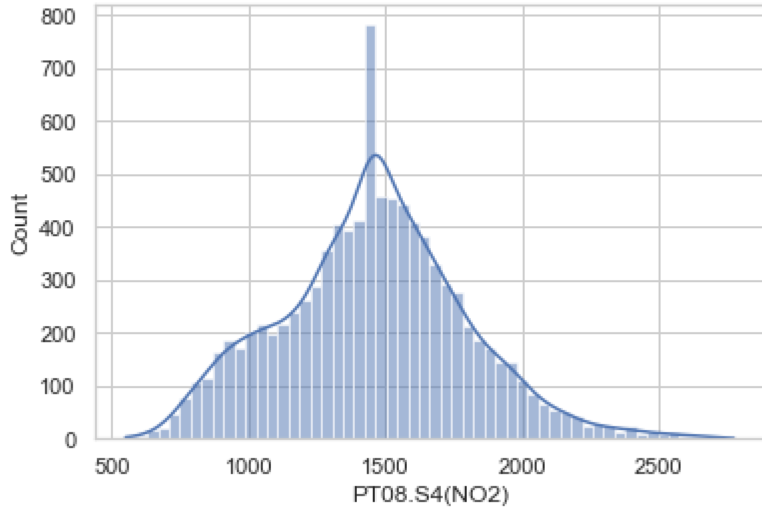

Count
700
600
500
400
300
200
100
0
0
10
20
30
40
50
60
C6H6(GT)


Count
800
700
600
500
400
300
200
100
0
500
750
1000
1250
1500
1750
2000
2250
PT08.S2(NMHC)

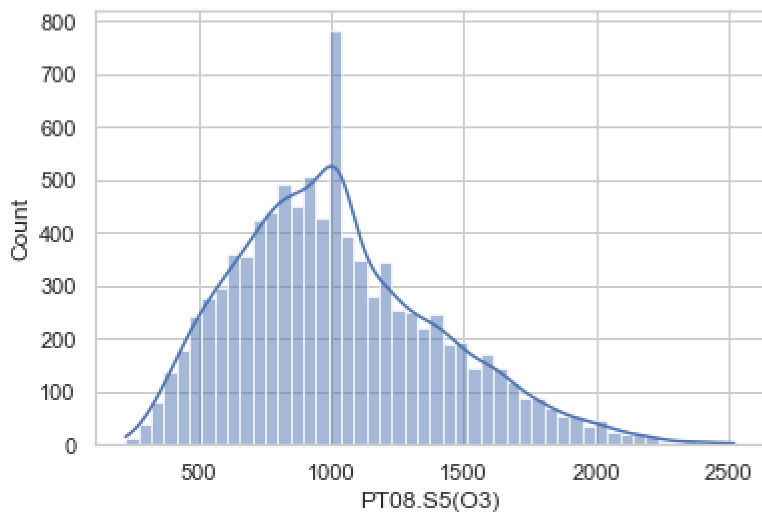

Count
NOx(GT)


Count
PT08.S3(NOx)

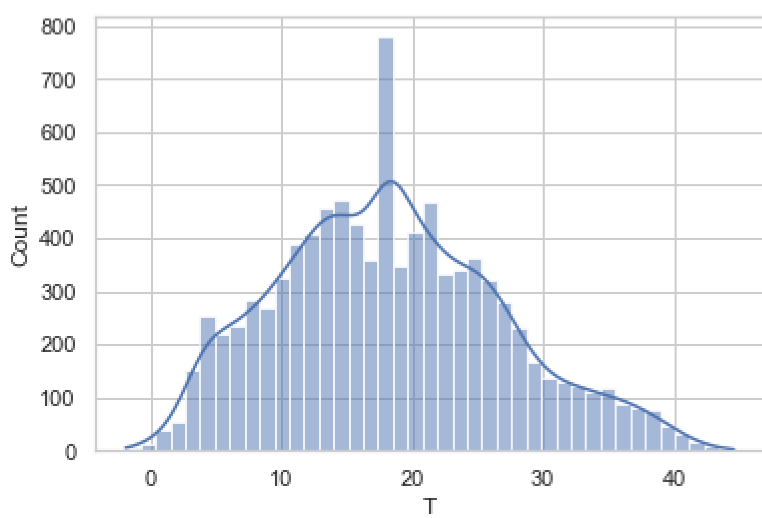

Count
NO2(GT)



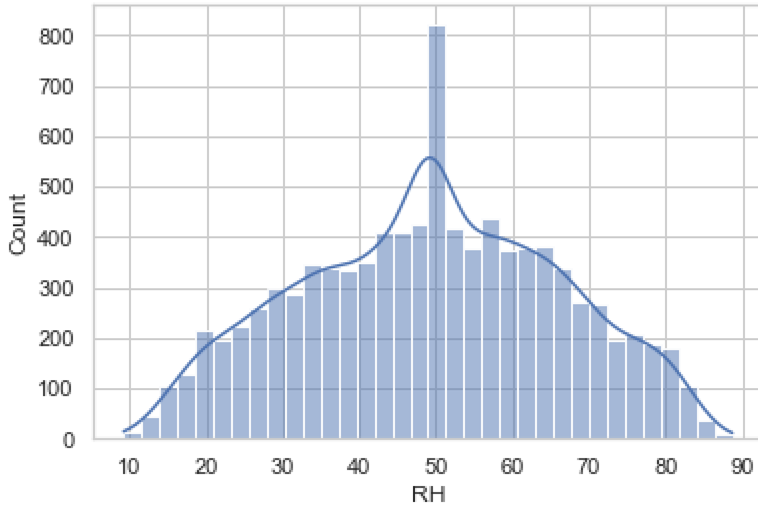
Count
PT08.S4(NO2)

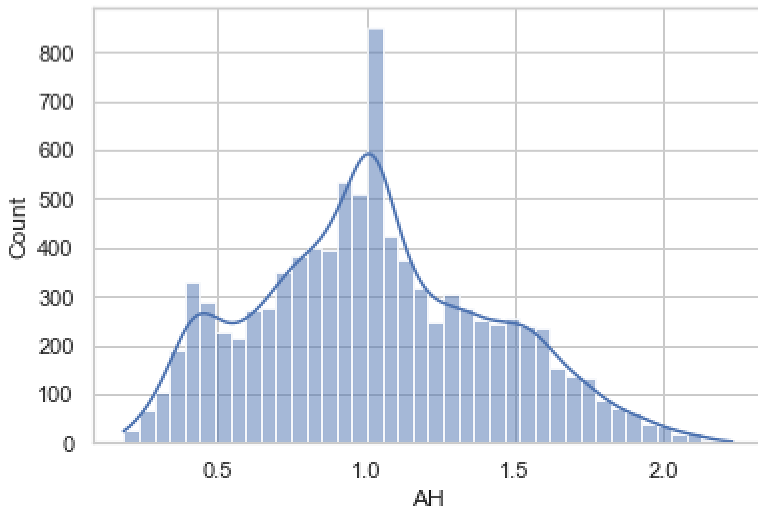


Count
PT08.S5(O3)

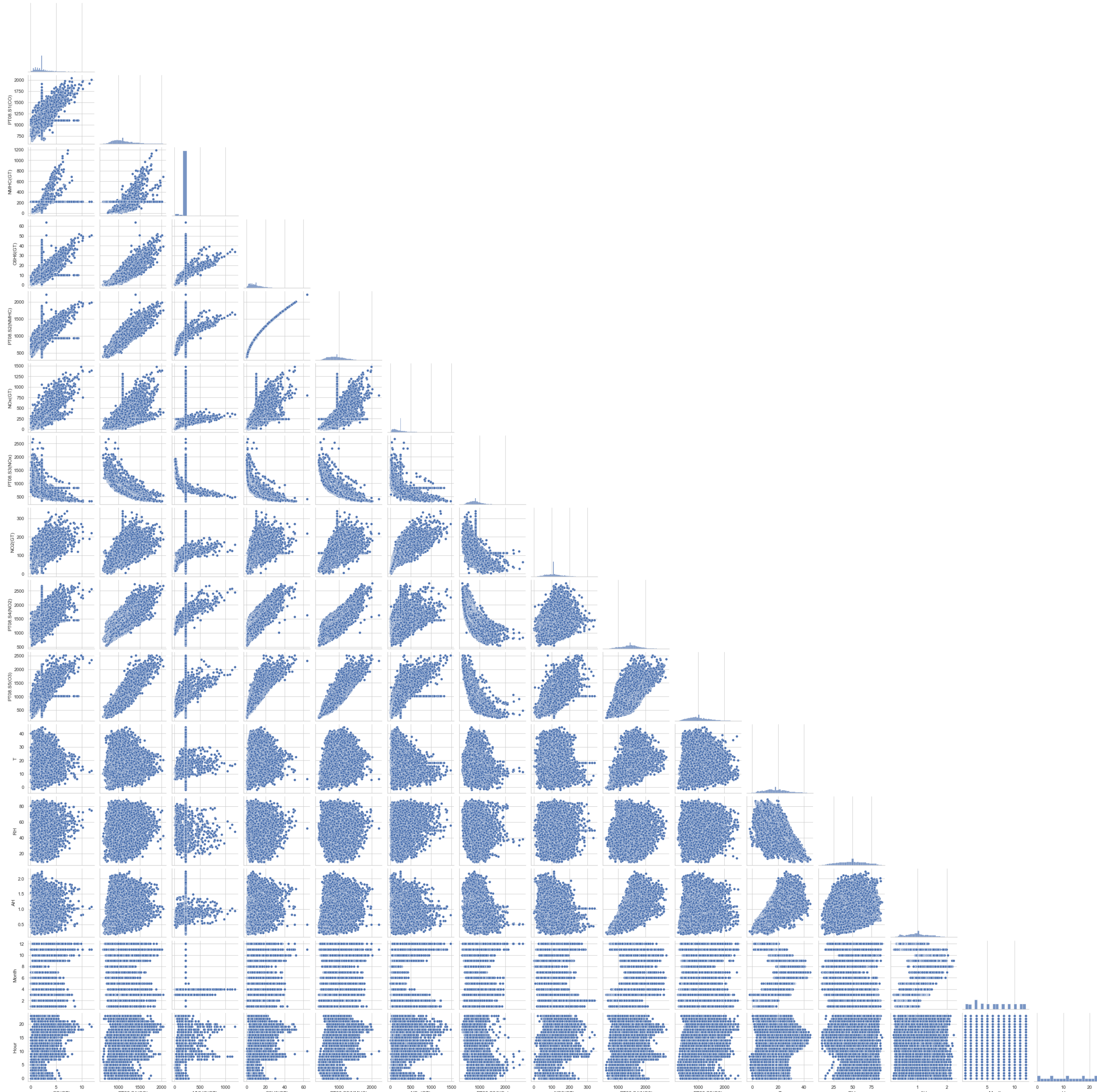


Count
T





In [248…
```
g = sns.pairplot(data=df, vars=df.columns,
                 corner=True)
```



In [249…

```
df2=df.copy()
Scaler=MinMaxScaler()
Xsd=Scaler.fit_transform(df2)    #Applying Normalization
df3 = pd.DataFrame(Xsd, columns = ['CO(GT)','PT08.S1(CO)','NMHC(GT)','C6H6(GT)', 'PT08.S
 'PT08.S5(O3)', 'T', 'RH','AH','Month','Hour'])
```

In [250…

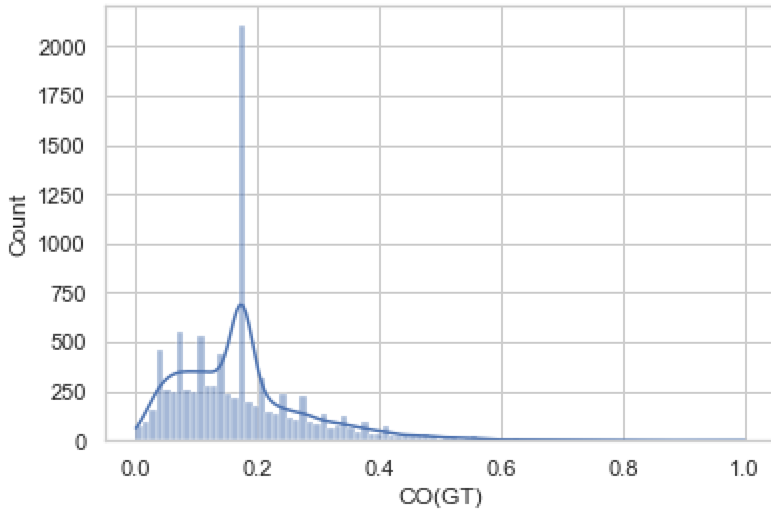
```
df3.head()
```

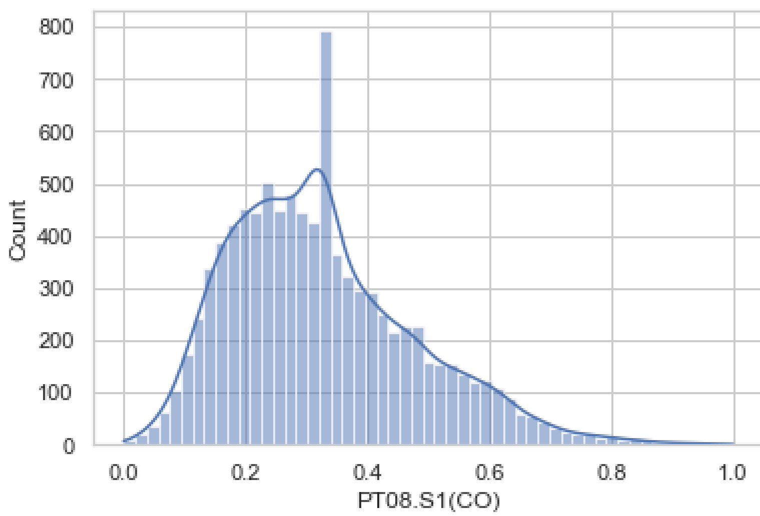
Out[250]:

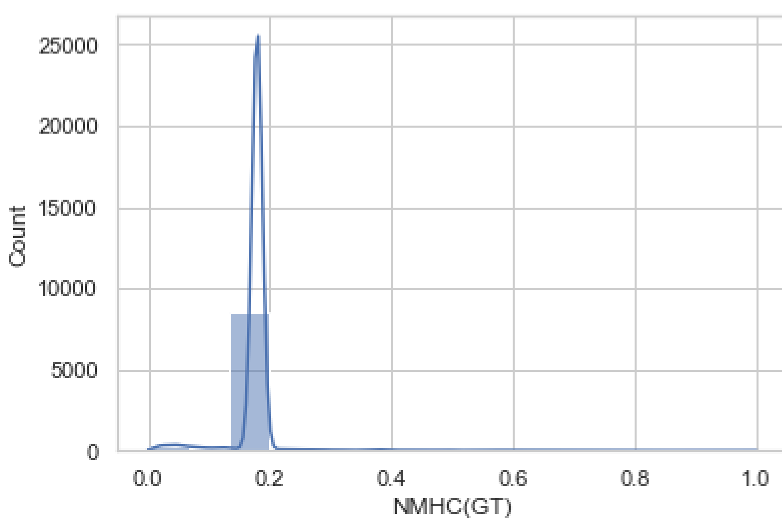
	CO(GT)	PT08.S1(CO)	NMHC(GT)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)	PT08.S3(NOx)	NO2(GT)	PT08.S4(NC
0	0.211864	0.511845	0.120981	0.185535	0.362097	0.111036	0.310885	0.328402	0.513(
1	0.161017	0.463029	0.088832	0.146226	0.312398	0.068382	0.360864	0.266272	0.4532
2	0.177966	0.541996	0.068528	0.139937	0.303659	0.087339	0.346463	0.331361	0.450(
3	0.177966	0.523331	0.061760	0.143082	0.308575	0.115098	0.326133	0.355030	0.4644
4	0.127119	0.448672	0.037225	0.100629	0.247406	0.087339	0.373994	0.337278	0.4222

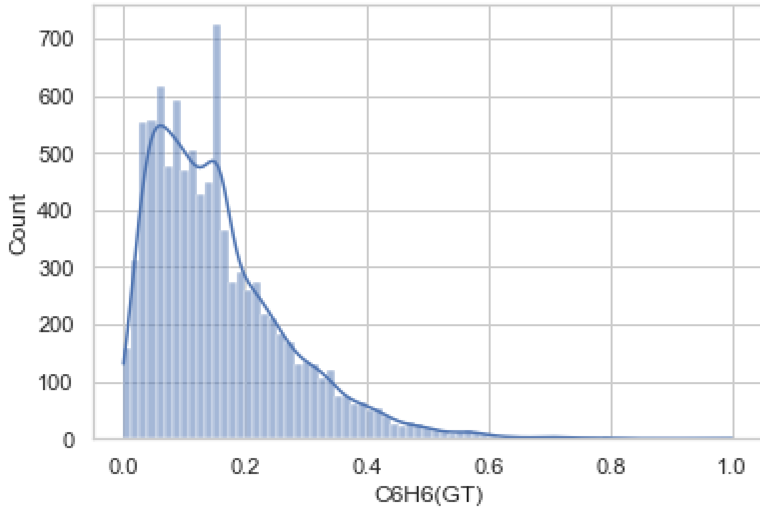
In [251…

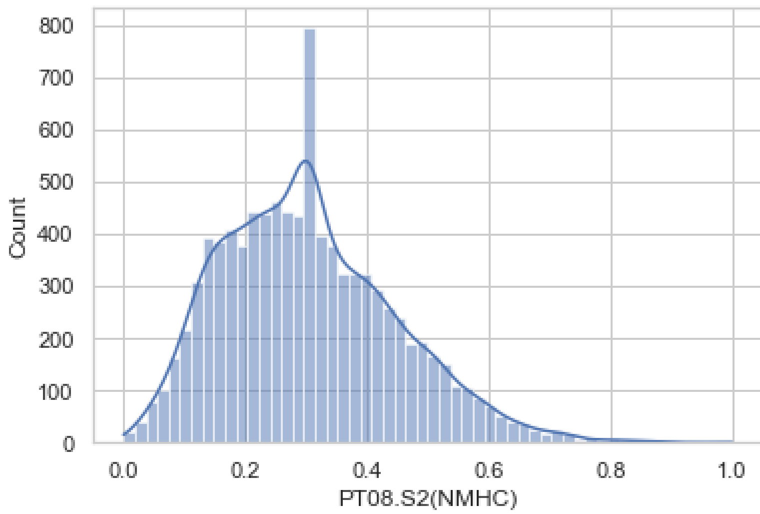
```
for i in df3.columns[:13]:
    sns.histplot(df3[i],kde=True)
    plt.show()
```

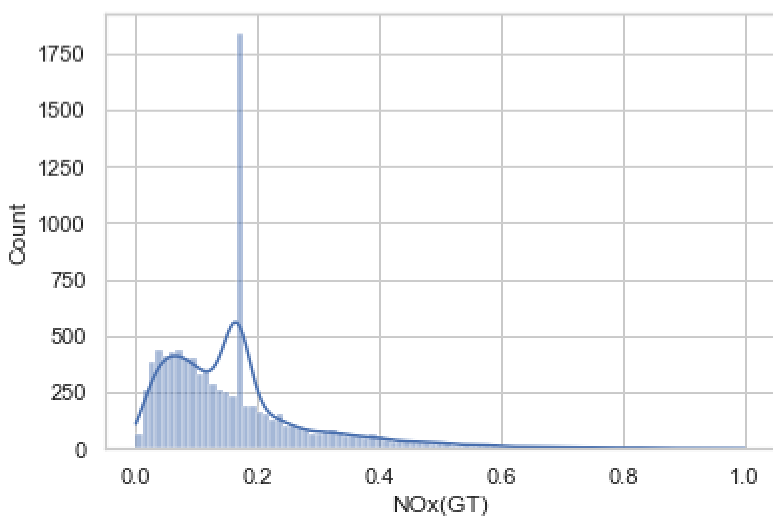


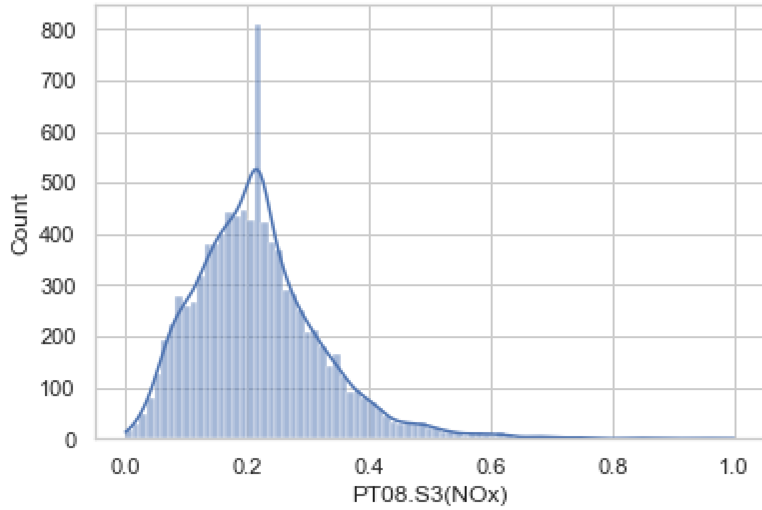


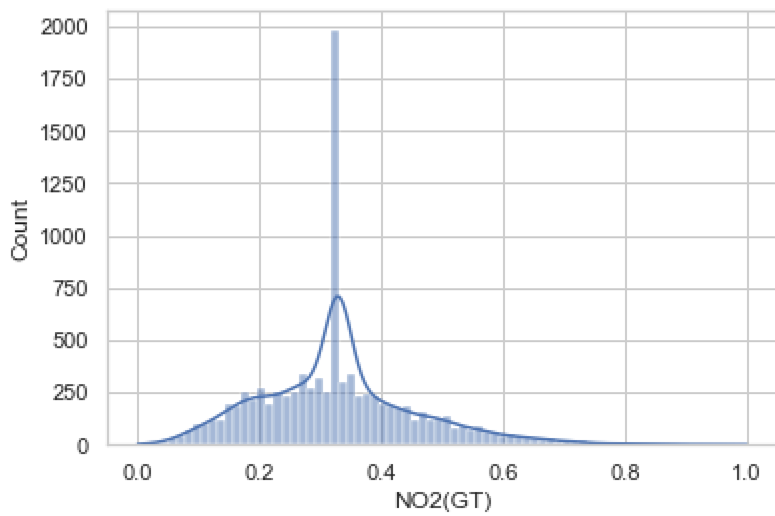


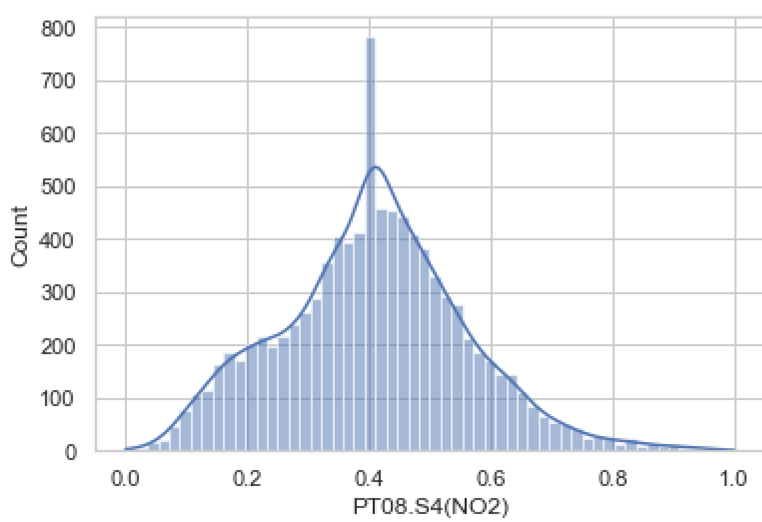

Count
C6H6(GT)

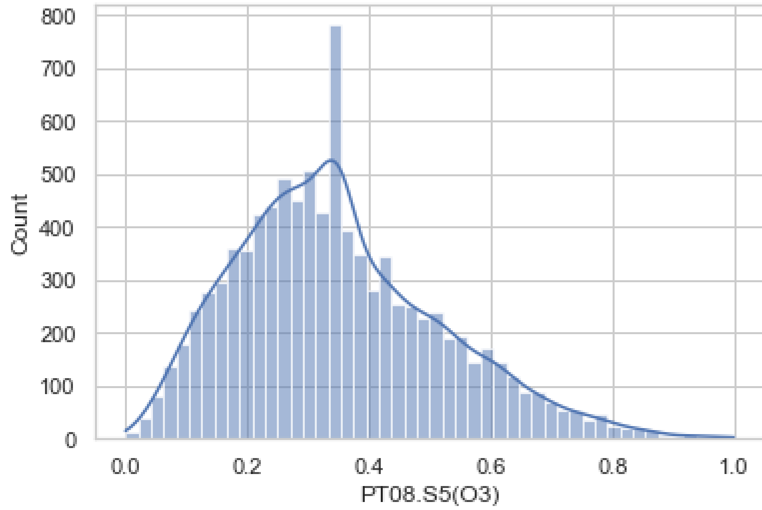

Count
PT08.S2(NMHC)

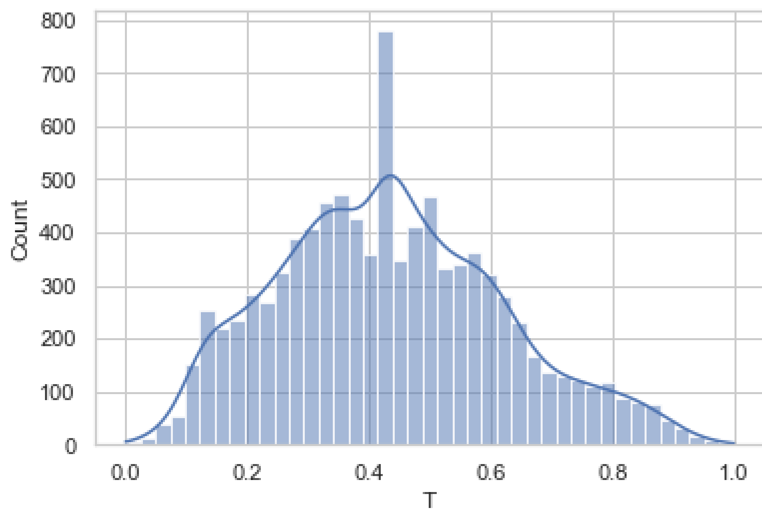

Count
NOx(GT)

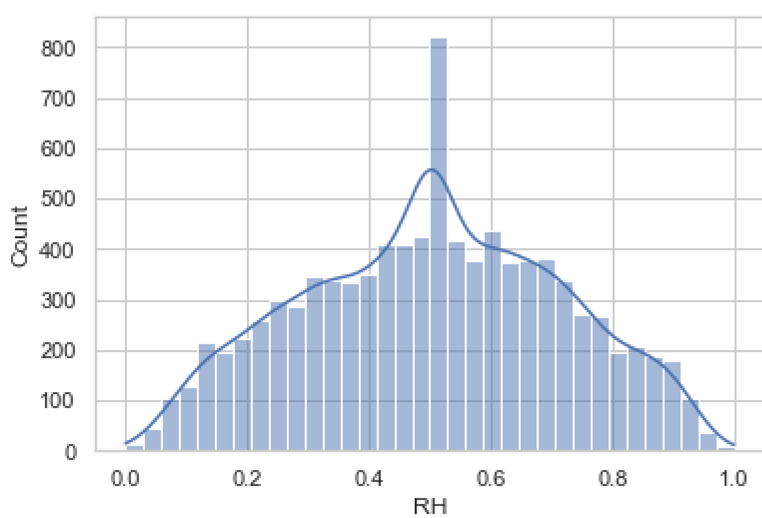

Count
PT08.S3(NOx)
0
100
200
300
400
500
600
700
800
0.0
0.2
0.4
0.6
0.8
1.0

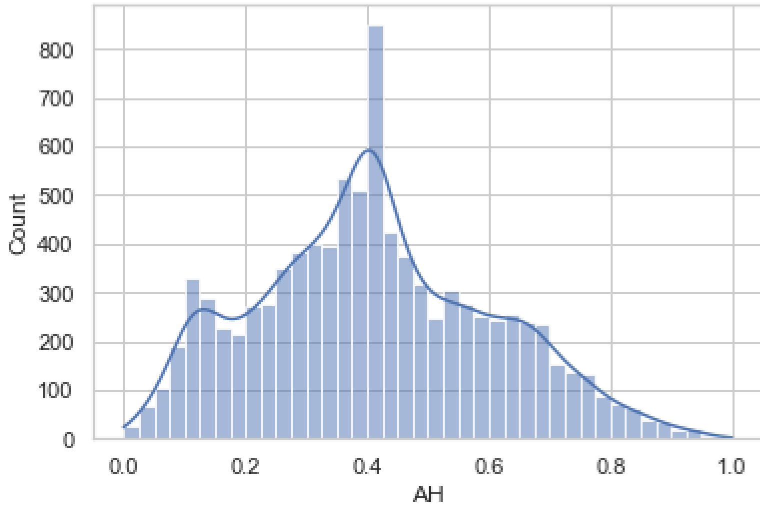

Count
NO2(GT)
0
250
500
750
1000
1250
1500
1750
2000
0.0
0.2
0.4
0.6
0.8
1.0


Count
PT08.S4(NO2)
0
100
200
300
400
500
600
700
800
0.0
0.2
0.4
0.6
0.8
1.0


Count
PT08.S5(O3)

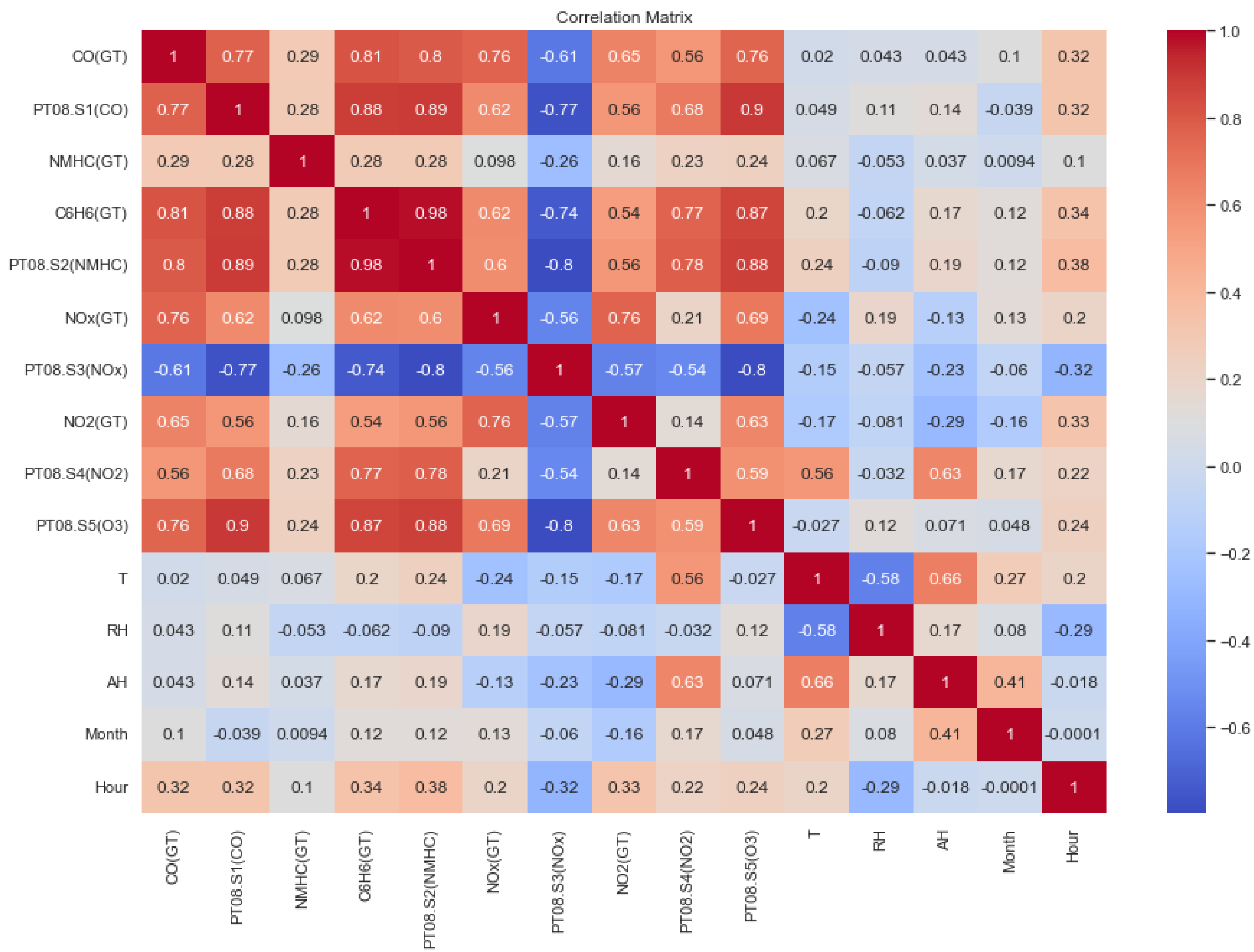

Count
T


Count
RH



In [252…

```
#Plotting correlation matrix
plt.figure(figsize=(15,10))
sns.heatmap(df3.corr(),annot=True,cmap = 'coolwarm')
plt.title('Correlation Matrix')
plt.show()
```



In [253…

```
df4=df.copy()
X=df4.drop(columns=['PT08.S5(O3)','T','RH','AH','Hour','Month','PT08.S3(NOx)','NMHC(GT)'
y=df4[['PT08.S5(O3)']]
X_n=df3.drop(columns=['PT08.S5(O3)','T','RH','AH','Hour','Month','PT08.S3(NOx)','NMHC(GT
y_n=df3[['PT08.S5(O3)']] #target variable PT08.S5(O3)
```

In [254…

```
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.3,random_state=42)
```

```
X_train_n, X_test_n, y_train_n, y_test_n=train_test_split(X_n, y_n, test_size=0.3,random
```

In [255…

```
#PCA on Normalized Data
pca_n = PCA(n_components=3)
X_train_n = pca_n.fit_transform(X_train_n)
X_test_n = pca_n.transform(X_test_n)
```

In [256…

```
explained_variance = pca_n.explained_variance_ratio_
explained_variance
```

Out[256]:

```
array([0.71286281, 0.18263439, 0.03536817])
```

LINEAR REGRESSION

In [257…

```
model_lr=LinearRegression()
model_lr=model_lr.fit(X_train, y_train)
```

In [258…

```
model_lr_n=LinearRegression()
model_lr_n=model_lr_n.fit(X_train_n, y_train_n)
```

In [259…

```
y_pred_lr_n=model_lr_n.predict(X_test_n)
y_pred_lr=model_lr.predict(X_test)
```

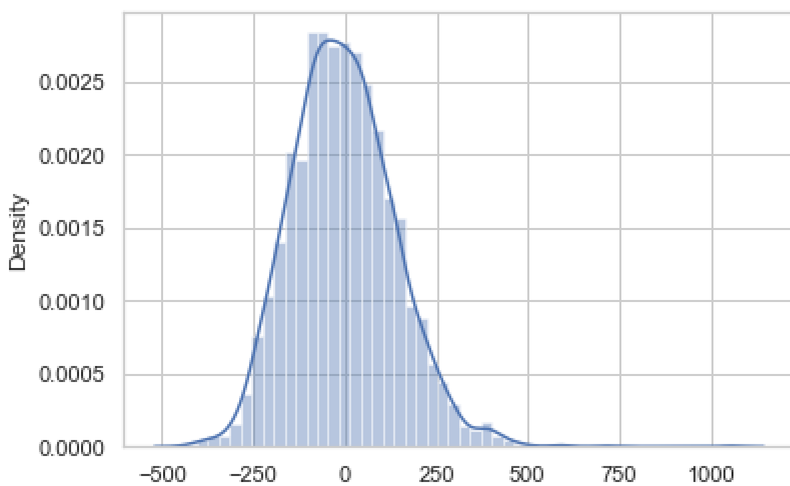
In [260…

```
sns.distplot(y_test-y_pred_lr)
```

```
C:\Users\rmpaw\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adap
t your code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[260]:

```
<AxesSubplot:ylabel='Density'>
```



In [261…

```
lrstat = [round(mean_squared_error(y_test,y_pred_lr),4), round(np.sqrt(mean_squared_erro
print('MSE value for LinearRegression model is {}'.format(lrstat[0]))
print('RMSE value for LinearRegression model is {}'.format(lrstat[1]))
print('R^2 value for LinearRegression model is {}'.format(lrstat[2]))
```

```
MSE value for LinearRegression model is 19895.3696
RMSE value for LinearRegression model is 141.0509
R^2 value for LinearRegression model is 87.2235
```

In [262…

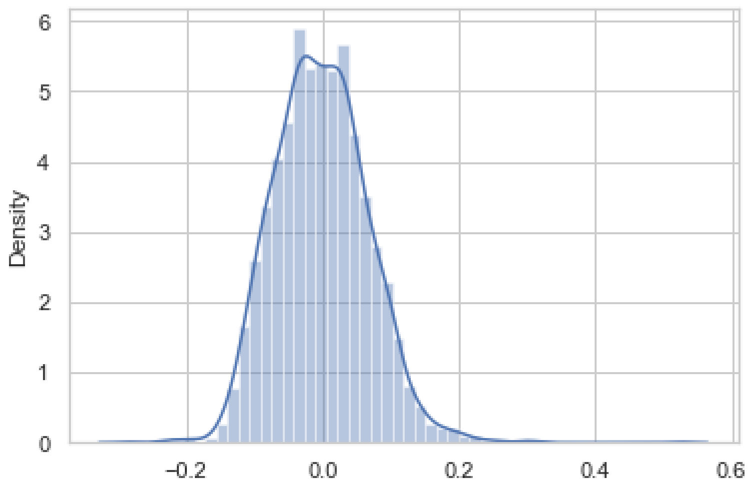
```
sns.distplot(y_test_n-y_pred_lr_n)
```

```
C:\Users\rmpaw\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning:
```

```
`distplot` is a deprecated function and will be removed in a future version. Please adap
t your code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[262]:

```
<AxesSubplot:ylabel='Density'>
```



In [263…

```
lrstat_n = [round(mean_squared_error(y_test_n,y_pred_lr_n),4), round(np.sqrt(mean_square
print('MSE value for Normalized LinearRegression model is {}'.format(lrstat_n[0]))
print('RMSE value for Normalized LinearRegression model is {}'.format(lrstat_n[1]))
print('R^2 value for Normalized LinearRegression model is {}'.format(lrstat_n[2]))
```

```
MSE value for Normalized LinearRegression model is 0.0048
RMSE value for Normalized LinearRegression model is 0.0689
R^2 value for Normalized LinearRegression model is 83.828
```

LASSO

In [264…

```
model_las=Lasso(alpha=3)
model_las=model_las.fit(X_train,y_train)
y_pred_las=model_las.predict(X_test)
```

In [265…

```
model_las_n=Lasso(alpha=3)
model_las_n=model_las_n.fit(X_train_n,y_train_n)
y_pred_n=model_las_n.predict(X_test_n)
```

In [266…

```
lasstat = [round(mean_squared_error(y_test,y_pred_las),4), round(np.sqrt(mean_squared_er
print('MSE value for LassoRegression model is {}'.format(lasstat[0]))
print('RMSE value for LassoRegression model is {}'.format(lasstat[1]))
print('R^2 value for LassoRegression model is {}'.format(lasstat[2]))
```

```
MSE value for LassoRegression model is 19921.0274
RMSE value for LassoRegression model is 141.1419
R^2 value for LassoRegression model is 87.207
```

In [267…

```
lasstat_n = [round(mean_squared_error(y_test_n,y_pred_n),4), round(np.sqrt(mean_squared_
print('MSE value for Normalized LassoRegression model is {}'.format(lasstat_n[0]))
print('RMSE value for Normalized LassoLinearRegression model is {}'.format(lasstat_n[1])
print('R^2 value for Normalized LassoLinearRegression model is {}'.format(lasstat_n[2]))
```

```
MSE value for Normalized LassoRegression model is 0.0294
RMSE value for Normalized LassoLinearRegression model is 0.1714
R^2 value for Normalized LassoLinearRegression model is -0.0332
```

RIDGE

In [268…
```
model_rid=Ridge()
model_rid=model_rid.fit(X_train,y_train)
```

In [269…
```
model_rid_n=Ridge()
model_rid_n=model_rid_n.fit(X_train_n,y_train_n)
```

In [270…
```
y_pred_rid=model_rid.predict(X_test)
y_pred_rid_n=model_rid_n.predict(X_test_n)
```

In [271…
```
ridstat = [round(mean_squared_error(y_test,y_pred_rid),4), round(np.sqrt(mean_squared_er
print('MSE value for Normalized RidgeRegression model is {}'.format(ridstat[0]))
print('RMSE value for Normalized RidgeLinearRegression model is {}'.format(ridstat[1]))
print('R^2 value for Normalized RidgeLinearRegression model is {}'.format(ridstat[2]))
```

```
MSE value for Normalized RidgeRegression model is 19895.3812
RMSE value for Normalized RidgeLinearRegression model is 141.051
R^2 value for Normalized RidgeLinearRegression model is 87.2235
```

In [272…
```
ridstat_n = [round(mean_squared_error(y_test_n,y_pred_rid_n),4), round(np.sqrt(mean_squa
print('MSE value for Normalized RidgeRegression model is {}'.format(ridstat_n[0]))
print('RMSE value for Normalized RidgeLinearRegression model is {}'.format(ridstat_n[1])
print('R^2 value for Normalized RidgeLinearRegression model is {}'.format(ridstat_n[2]))
```

```
MSE value for Normalized RidgeRegression model is 0.0048
RMSE value for Normalized RidgeLinearRegression model is 0.0689
R^2 value for Normalized RidgeLinearRegression model is 83.8305
```

LOGISTIC

In [273…
```
X_las_n=scale(X_n)
y_las_n=scale(y_n) #Center todd the mean and component wise scale to unit variance.
X_train_las_n, X_test_las_n, y_train_las_n, y_test_las_n=train_test_split(X_las_n, y_las
```

In [274…
```
df_log=df.copy()
X_log=df_log.drop(columns=['PT08.S5(O3)','T','RH','AH','Hour','Month','PT08.S3(NOx)','NM
y_log=df_log[['PT08.S5(O3)']]
X_log = np.column_stack([np.ones(X_n.shape[0]), X_n])

X_train_log, X_test_log, y_train_log, y_test_log=train_test_split(X_log, y_n, test_size=
lab_enc = LabelEncoder()
training_y = lab_enc.fit_transform(y_train_log)
y_test_enc = lab_enc.fit_transform(y_test_log)

training_y = training_y.reshape(-1, 1)
print(training_y)
model_log = LogisticRegression()
model_log=model_log.fit(X_train_log,training_y)

y_pred_log=model_log.predict(X_test_log)
```

```
[[1248]
 [ 632]
 [1126]
 ...
 [1123]
 [1086]
 [1294]]
```

```
C:\Users\rmpaw\anaconda3\lib\site-packages\sklearn\preprocessing\_label.py:115: DataConv
ersionWarning: A column-vector y was passed when a 1d array was expected. Please change
the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\rmpaw\anaconda3\lib\site-packages\sklearn\utils\validation.py:993: DataConversi
```

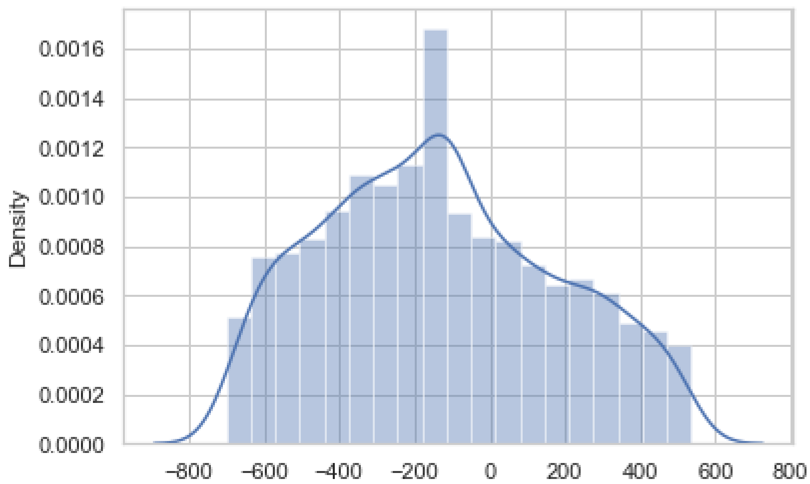
```
onWarning: A column-vector y was passed when a 1d array was expected. Please change the
shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [275…

```
sns.distplot(y_test_enc-y_pred_log)
```

```
C:\Users\rmpaw\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adap
t your code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[275]: <AxesSubplot:ylabel='Density'>



In [277…

```
logstat = [round(mean_squared_error(y_test_enc,y_pred_log),4), round(np.sqrt(mean_square
print('MSE value for LogisticRegression model is {}'.format(logstat[0]))
print('RMSE value for LogisticRegression model is {}'.format(logstat[1]))
print('R^2 value for LogisticRegression model is {}'.format(logstat[2]))
```

```
MSE value for LogisticRegression model is 114274.9316
RMSE value for LogisticRegression model is 338.0458
R^2 value for LogisticRegression model is -18.6171
```

SVD

In [278…

```
X_svd = np.column_stack([np.ones(X_n.shape[0]), X_n])
X_train_svd, X_test_svd, y_train_svd, y_test_svd=train_test_split(X_svd, y, test_size=0.

U,S,Vt = np.linalg.svd(X_train_svd, full_matrices=False)

x_hat = Vt.T @ np.linalg.inv(np.diag(S)) @ U.T @ y_train_svd

y_pred_svd_train = X_train_svd @ x_hat
y_pred_svd = X_test_svd @ x_hat

mse_svd=np.sqrt(mean_squared_error(y_test_svd,y_pred_svd))
print(mse_svd)
print(r2_score(y_test_svd,y_pred_svd))
```

```
139.03533744590558
0.8704821690812764
```

In [279…

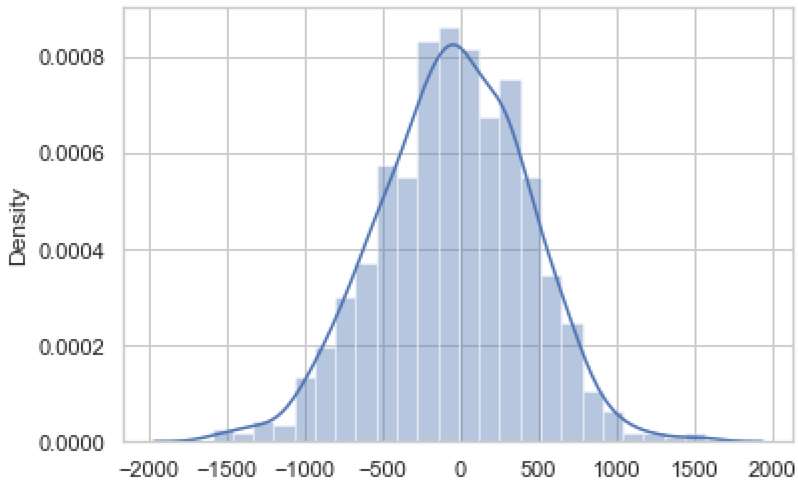
```
sns.distplot(y_test_svd-y_pred_svd)
```

```
C:\Users\rmpaw\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adap
t your code to use either `displot` (a figure-level function with similar flexibility) o
```

```
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[279]:

```
<AxesSubplot:ylabel='Density'>
```



In [280…

```
svdstat = [round(mean_squared_error(y_test_svd,y_pred_svd),4), round(np.sqrt(mean_square
print('MSE value for SVD model is {}'.format(svdstat[0]))
print('RMSE value for SVD model is {}'.format(svdstat[1]))
print('R^2 value for SVD model is {}'.format(svdstat[2]))
```

```
MSE value for SVD model is 19330.8251
RMSE value for SVD model is 139.0353
R^2 value for SVD model is 87.0482
```

LEAST SQUARES

In [281…

```
import statsmodels.api as sm

results = sm.OLS(y_train, X_train).fit()
results.summary()
```

Out[281]:

OLS Regression Results

Dep. Variable:	PT08.S5(O3)	**R-squared (uncentered):**	0.981
Model:	OLS	**Adj. R-squared (uncentered):**	0.981
Method:	Least Squares	**F-statistic:**	4.873e+04
Date:	Tue, 20 Dec 2022	**Prob (F-statistic):**	0.00
Time:	01:21:58	**Log-Likelihood:**	-42100.
No. Observations:	6549	**AIC:**	8.421e+04
Df Residuals:	6542	**BIC:**	8.426e+04
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
CO(GT)	-19.1610	3.065	-6.251	0.000	-25.170	-13.153
PT08.S1(CO)	0.7546	0.019	40.408	0.000	0.718	0.791
C6H6(GT)	23.9525	0.813	29.467	0.000	22.359	25.546
PT08.S2(NMHC)	0.1809	0.034	5.326	0.000	0.114	0.248

NOx(GT)	0.2730	0.019	14.258	0.000	0.235	0.311
NO2(GT)	0.3066	0.074	4.117	0.000	0.161	0.453
PT08.S4(NO2)	-0.1884	0.011	-17.195	0.000	-0.210	-0.167

Omnibus:	417.569	**Durbin-Watson:**	2.010
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	744.530
Skew:	0.480	**Prob(JB):**	2.12e-162
Kurtosis:	4.344	**Cond. No.**	3.52e+03

Notes:
[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[3] The condition number is large, 3.52e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

In [282…

```
X_train_n, X_test_n, y_train_n, y_test_n=train_test_split(X_n, y_n, test_size=0.3,random
results1 = sm.OLS(y_train_n, X_train_n).fit()
results1.summary()
```

Out[282]:

OLS Regression Results

Dep. Variable:	PT08.S5(O3)	**R-squared (uncentered):**	0.974
Model:	OLS	**Adj. R-squared (uncentered):**	0.974
Method:	Least Squares	**F-statistic:**	3.527e+04
Date:	Tue, 20 Dec 2022	**Prob (F-statistic):**	0.00
Time:	01:21:59	**Log-Likelihood:**	8911.9
No. Observations:	6549	**AIC:**	-1.781e+04
Df Residuals:	6542	**BIC:**	-1.776e+04
Df Model:	7		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
CO(GT)	-0.1133	0.015	-7.571	0.000	-0.143	-0.084
PT08.S1(CO)	0.5692	0.012	49.180	0.000	0.546	0.592
C6H6(GT)	-0.2325	0.031	-7.543	0.000	-0.293	-0.172
PT08.S2(NMHC)	0.7091	0.030	23.469	0.000	0.650	0.768
NOx(GT)	0.2168	0.012	18.445	0.000	0.194	0.240
NO2(GT)	0.0423	0.010	4.335	0.000	0.023	0.061
PT08.S4(NO2)	-0.1115	0.009	-12.679	0.000	-0.129	-0.094

Omnibus:	472.013	**Durbin-Watson:**	2.009
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	862.147
Skew:	0.522	**Prob(JB):**	6.13e-188

Kurtosis:	4.438	**Cond. No.**	44.2

Notes:
[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [283...
```
from statsmodels.tools.eval_measures import rmse

ypred = results.predict(X_train)

# calc rmse
rmse = rmse(y_train, ypred)
```

In [284...
```
print("RMSE along the axis :",rmse) # for Original Data
```

```
RMSE along the axis : [606.66413296 395.75400972 542.67371677 ... 722.01929078 571.12431
375
 664.75226789]
```

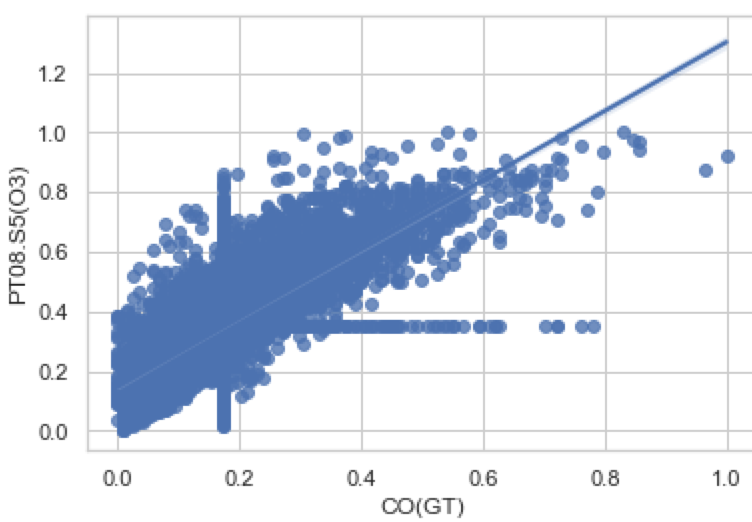
In [285...
```
from statsmodels.tools.eval_measures import rmse

ypred = results1.predict(X_train_n)

# calc rmse
rmse = rmse(y_train_n, ypred)
```

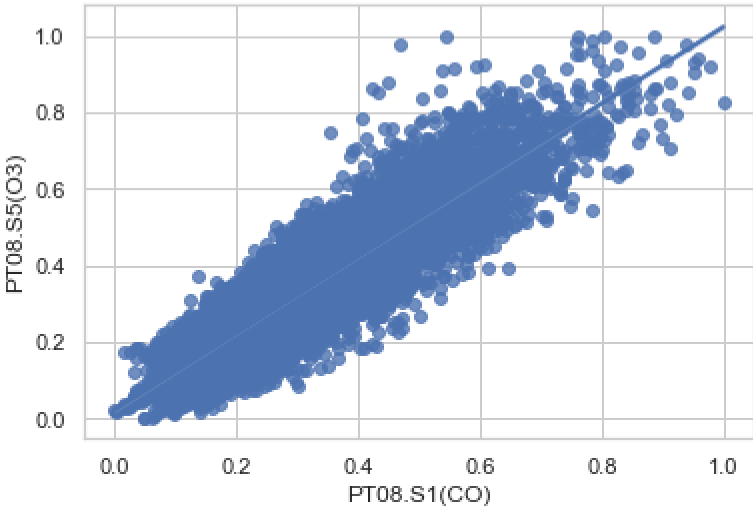
In [286...
```
print("RMSE along the axis :",rmse) # for normalized data
```

```
RMSE along the axis : [0.26861643 0.17108547 0.24613596 ... 0.30799439 0.25407641 0.3079
1442]
```

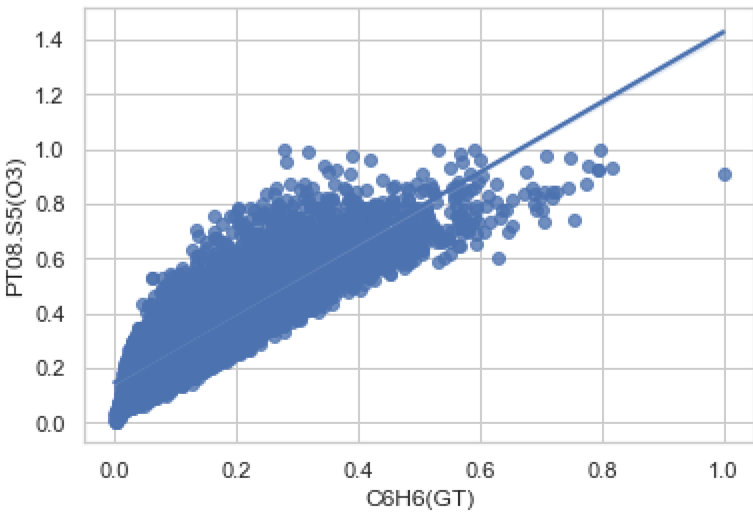
In [287...
```
OLS_R2=[0.981,0.974]
OLS_RMSE = [606.66,0.23]
```

In [288...
```
for i in X.columns[:12]:
  sns.regplot(data=df3,x=df3[i], y=df3['PT08.S5(O3)'])
  plt.show()
```

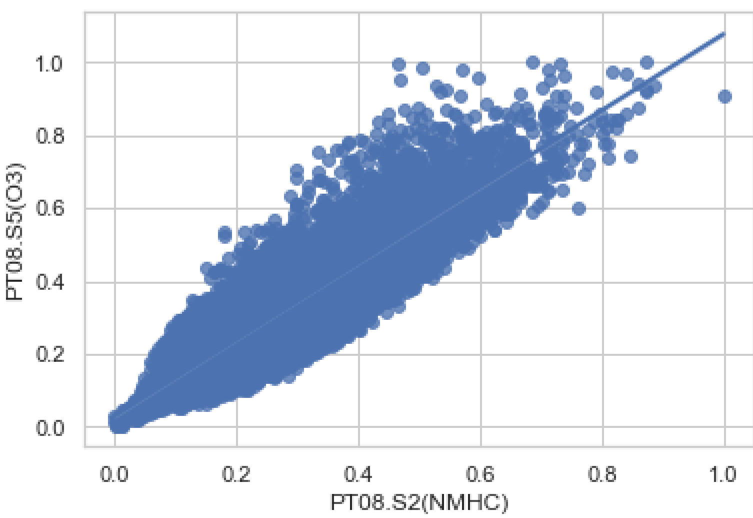




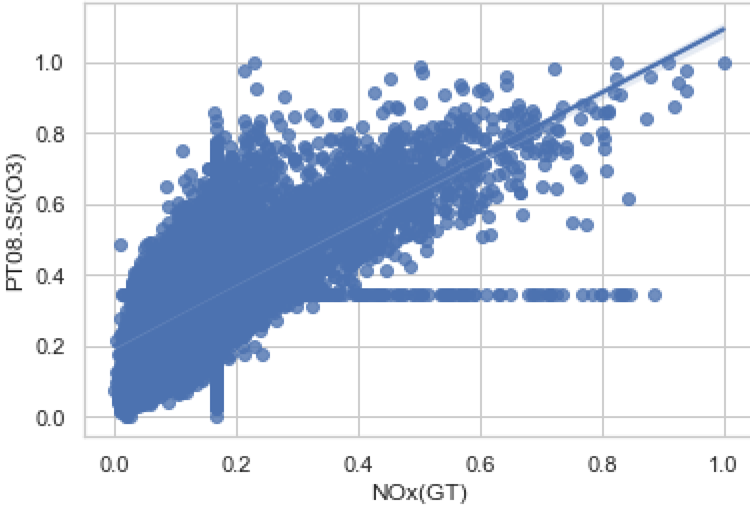
PT08.S5(O3)
0.0
0.2
0.4
0.6
0.8
1.0
PT08.S1(CO)

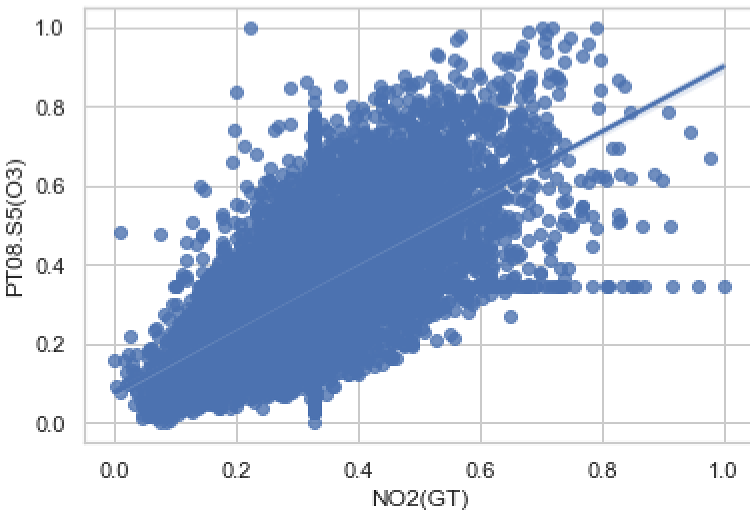


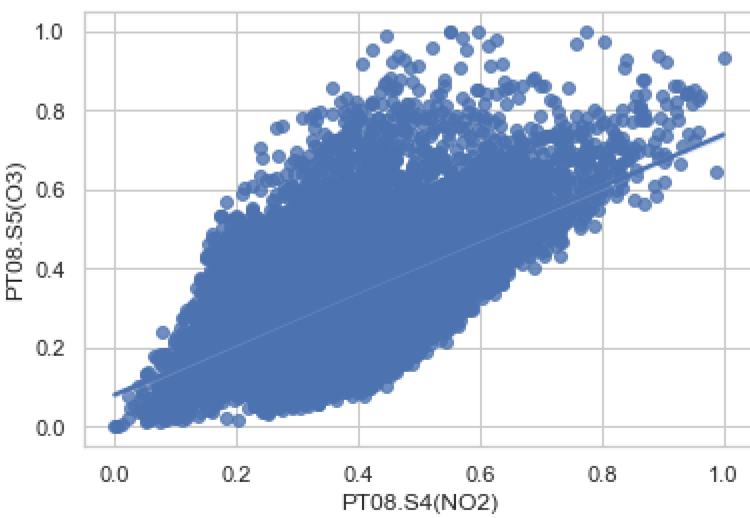
PT08.S5(O3)
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
C6H6(GT)



PT08.S5(O3)
0.0
0.2
0.4
0.6
0.8
1.0
PT08.S2(NMHC)







CONCLUSION

In [289…

```
conclusion = pd.DataFrame(data=[lrstat,lrstat_n,lasstat,lasstat_n,ridstat,ridstat_n,logs
                          index=['Linear Regression','Linear Regression using normalizat
                                 'Lasso Regression using normalization','Ridge Regressio
                         columns = ['MSE','RMSE','R^2'])
conclusion
```

Out[289]:

	MSE	RMSE	R^2
Linear Regression	19895.3696	141.0509	87.2235

Linear Regression using normalization	0.0048	0.0689	83.8280
Lasso Regression	19921.0274	141.1419	87.2070
Lasso Regression using normalization	0.0294	0.1714	-0.0332
Ridge Regression	19895.3812	141.0510	87.2235
Ridge Regression using normalization	0.0048	0.0689	83.8305
Logistic Regression	114274.9316	338.0458	-18.6171
SVD Model	19330.8251	139.0353	87.0482

In [290…

```
OLS_conclusion = pd.DataFrame(data=[OLS_R2,OLS_RMSE],index = ['R2 for OLS','RMSE for OLS
                              columns=['Original Data','Normalized data',])
OLS_conclusion
```

Out[290]:

	Original Data	Normalized data
R2 for OLS	0.981	0.974
RMSE for OLS	606.660	0.230