AIR QUALITY INDEX PREDICTION

PROBLEM STATEMENT :- Given dataset, calculate the responses of a gas multisensor device deployed on the field in an Italian city. Hourly responses averages are recorded along with gas concentrations references from a certified analyzer.

INTRODUCTION :- 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 were 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.Missing values are tagged with -200 value. Hence, responses of the gas multisensor device is calculated through **Linear Regression, KNN classification, Time Series Analysis**. We aim to provide users with an acknowledgment of the indexes associated with all pollutants generated by gases, enabling them to take necessary actions accordingly.

DATASET INFORMATION :-

https://drive.google.com/file/d/1iH8waMfK3x8SRIY4WAvGmEhUa4ZcKbWz/view?usp=drive_link

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

```
# loading the dataset
aq_data = pd.read_csv("/content/AirQualityUCI.csv")
```

```
aq_data.head()
```

	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)
0	10/3/2004	18.00.00	2.6	1360	150.0	11.9	1046	166	1056	113	1692	1268.0
1	10/3/2004	19.00.00	2.0	1292	112.0	9.4	955	103	1174	92	1559	972.0
2	10/3/2004	20.00.00	2.2	1402	88.0	9.0	939	131	1140	114	1555	1074.0
3	10/3/2004	21.00.00	2.2	1376	80.0	9.2	948	172	1092	122	1584	1203.0
4	10/3/2004	22.00.00	1.6	1272	51.0	6.5	836	131	1205	116	1490	1110.0

```
# printing last 5 rows
aq_data.tail()
```

	Date	Time	CO(GT)	PT08.S1(CO)	NMHC(GT)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)
9352	4/4/2005	10.00.00	3.1	1314	-200.0	13.5	1101	472
9353	4/4/2005	11.00.00	2.4	1163	-200.0	11.4	1027	353
9354	4/4/2005	12.00.00	2.4	1142	-200.0	12.4	1063	293
9355	4/4/2005	13.00.00	2.1	1003	-200.0	9.5	961	235
9356	4/4/2005	14.00.00	2.2	1071	-200.0	11.9	1047	265

```
aq_data.shape
```

```
(9357, 15)
```

```
# getting some info about the data
aq_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 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 int64
 4 NMHC(GT) 9357 non-null float64
 5 C6H6(GT) 9357 non-null float64
 6 PT08.S2(NMHC) 9357 non-null int64
 7 NOx(GT) 9357 non-null int64
 8 PT08.S3(NOx) 9357 non-null int64
```

```
 9 NO2(GT) 9357 non-null int64 10
PT08.S4(NO2) 9357 non-null int64 11
PT08.S5(O3) 9357 non-null float64 12 T
9357 non-null float64 13 RH 9357 non-null
float64 14 AH 9356 non-null float64
dtypes: float64(7), int64(6), object(2)
memory usage: 1.1+ MB
```

```
aq_data.isnull().sum()
```

```
Date 0
Time 0
CO(GT) 0
PT08.S1(CO) 0
NMHC(GT) 0
C6H6(GT) 0
PT08.S2(NMHC) 0
NOx(GT) 0
PT08.S3(NOx) 0
NO2(GT) 0
PT08.S4(NO2) 0
PT08.S5(O3) 0
T 0
RH 0
AH 1
dtype: int64
```

```
# Fill with mean
mean_AH = aq_data['AH'].mean()
aq_data['AH'] = aq_data['AH'].fillna(mean_AH)

# Fill with median
median_AH = aq_data['AH'].median()
aq_data['AH'] = aq_data['AH'].fillna(median_AH)

# Fill with a constant (e.g., 0)
aq_data['AH'] = aq_data['AH'].fillna(0)
```

```
aq_data.isnull().sum()
```

```
Date 0
Time 0
CO(GT) 0
PT08.S1(CO) 0
NMHC(GT) 0
C6H6(GT) 0
PT08.S2(NMHC) 0
NOx(GT) 0
PT08.S3(NOx) 0
NO2(GT) 0
PT08.S4(NO2) 0
PT08.S5(O3) 0
T 0
RH 0
AH 0
dtype: int64
```

```
aq_data.isin([-200]).sum(axis=0)
```

```
Date 0
Time 0
CO(GT) 1682
PT08.S1(CO) 367
NMHC(GT) 8442
C6H6(GT) 366
PT08.S2(NMHC) 367
NOx(GT) 1638
PT08.S3(NOx) 367
NO2(GT) 1641
PT08.S4(NO2) 366
PT08.S5(O3) 366
T 366
RH 366
AH 366
dtype: int64
```

```
aq_data.tail()
```

```
Date Time CO(GT) PT08.S1(CO) NMHC(GT) C6H6(GT) PT08.S2(NMHC) NOx(GT)
```

9352 4/4/2005 10.00.00 3.1 1314 -200.0 13.5 1101 472 **9353** 4/4/2005 11.00.00 2.4 1163 -200.0 11.4 1027

353 **9354** 4/4/2005 12.00.00 2.4 1142 -200.0 12.4 1063 293 **9355** 4/4/2005 13.00.00 2.1 1003 -200.0 9.5

961 235 **9356** 4/4/2005 14.00.00 2.2 1071 -200.0 11.9 1047 265

```
aq_data = aq_data.replace(to_replace= -200, value = np.NaN)
```

```
aq_data.isnull().sum()
```

```
Date 0
Time 0
CO(GT) 1682
PT08.S1(CO) 367
NMHC(GT) 8442
C6H6(GT) 366
PT08.S2(NMHC) 367
NOx(GT) 1638
PT08.S3(NOx) 367
NO2(GT) 1641
PT08.S4(NO2) 366
PT08.S5(O3) 366
T 366
RH 366
AH 366
dtype: int64
```

```
aq_data.tail()
```

```
Date Time CO(GT) PT08.S1(CO) NMHC(GT) C6H6(GT) PT08.S2(NMHC) NOx(GT) 9352
```

4/4/2005 10.00.00 3.1 1314.0 NaN 13.5 1101.0 472.0 **9353** 4/4/2005 11.00.00 2.4 1163.0 NaN 11.4 1027.0

353.0 **9354** 4/4/2005 12.00.00 2.4 1142.0 NaN 12.4 1063.0 293.0 **9355** 4/4/2005 13.00.00 2.1 1003.0 NaN

9.5 961.0 235.0 **9356** 4/4/2005 14.00.00 2.2 1071.0 NaN 11.9 1047.0 265.0

```
# Calculate the mean of the 'NMHC(GT)' column
mean_NMHC = aq_data['NMHC(GT)'].mean()

# Replace NaN values in the 'NMHC(GT)' column with the mean
aq_data['NMHC(GT)'] = aq_data['NMHC(GT)'].fillna(mean_NMHC)
```

```
aq_data.tail()
```

```
Date Time CO(GT) PT08.S1(CO) NMHC(GT) C6H6(GT) PT08.S2(NMHC) NOx(GT 9352
```

4/4/2005 10.00.00 3.1 1314.0 218.595519 13.5 1101.0 472.0 **9353** 4/4/2005 11.00.00 2.4 1163.0 218.595519

11.4 1027.0 353.0 **9354** 4/4/2005 12.00.00 2.4 1142.0 218.595519 12.4 1063.0 293.0 **9355** 4/4/2005 13.00.00

2.1 1003.0 218.595519 9.5 961.0 235.0 **9356** 4/4/2005 14.00.00 2.2 1071.0 218.595519 11.9 1047.0 265.0

```
# Calculate the mean of the 'CO(GT)' column (excluding NaN values)
mean_CO_GT = aq_data['CO(GT)'].mean()

# Replace NaN values in the 'CO(GT)' column with the mean
aq_data['CO(GT)'] = aq_data['CO(GT)'].fillna(mean_CO_GT)
```



```
aq_data.tail()
```

```
Date Time CO(GT) PT08.S1(CO) NMHC(GT) C6H6(GT) PT08.S2(NMHC) NOx(GT
```

9352 4/4/2005 10.00.00 3.1 1314.0 218.595519 13.5 1101.0 472.0 **9353** 4/4/2005 11.00.00 2.4 1163.0

218.595519 11.4 1027.0 353.0 **9354** 4/4/2005 12.00.00 2.4 1142.0 218.595519 12.4 1063.0 293.0 **9355**

4/4/2005 13.00.00 2.1 1003.0 218.595519 9.5 961.0 235.0 **9356** 4/4/2005 14.00.00 2.2 1071.0 218.595519

11.9 1047.0 265.0

```
aq_data.head()
```

Date Time CO(GT) PT08.S1(CO) NMHC(GT) C6H6(GT) PT08.S2(NMHC) NOx(GT) P 0 10/3/2004

18.00.00 2.6 1360.0 150.0 11.9 1046.0 166.0 **1** 10/3/2004 19.00.00 2.0 1292.0 112.0 9.4 955.0 103.0 **2**

10/3/2004 20.00.00 2.2 1402.0 88.0 9.0 939.0 131.0 **3** 10/3/2004 21.00.00 2.2 1376.0 80.0 9.2 948.0 172.0 **4**

10/3/2004 22.00.00 1.6 1272.0 51.0 6.5 836.0 131.0

```
aq_data.isnull().sum()
```

```
Date 0
Time 0
CO(GT) 0
PT08.S1(CO) 367
NMHC(GT) 0
C6H6(GT) 366
PT08.S2(NMHC) 367
NOx(GT) 1638
PT08.S3(NOx) 367
NO2(GT) 1641
PT08.S4(NO2) 366
PT08.S5(O3) 366
T 366
RH 366
AH 366
dtype: int64
```

```
# Define the list of columns to handle missing values for
columns_to_impute = ['PT08.S1(CO)', 'C6H6(GT)', 'PT08.S2(NMHC)', 'NOx(GT)',
                     'PT08.S3(NOx)', 'NO2(GT)', 'PT08.S4(NO2)', 'PT08.S5(O3)',
                     'T', 'RH', 'AH']

# Iterate over each column and handle missing values
for column in columns_to_impute:
    # Calculate the mean of the column
    mean_value = aq_data[column].mean()
    # Replace NaN values in the column with the mean
    aq_data[column] = aq_data[column].fillna(mean_value)
```

```
aq_data.isnull().sum()
```

```
Date 0
Time 0
CO(GT) 0
PT08.S1(CO) 0
NMHC(GT) 0
C6H6(GT) 0
PT08.S2(NMHC) 0
NOx(GT) 0
PT08.S3(NOx) 0
NO2(GT) 0
PT08.S4(NO2) 0
PT08.S5(O3) 0
T 0
RH 0
AH 0
dtype: int64
```


DONE WITH DATA PROCESSING

IN TIME SERIES ANALYSIS, one column should be date and time info and other one as the desired gas

```
date_info = pd.to_datetime(aq_data['Date'], format='%d/%m/%Y')
print(date_info)
```

```
0 2004-03-10
1 2004-03-10
2 2004-03-10
3 2004-03-10
4 2004-03-10
 ...
9352 2005-04-04
9353 2005-04-04
9354 2005-04-04
9355 2005-04-04
9356 2005-04-04
Name: Date, Length: 9357, dtype: datetime64[ns]
```

```
time_info = aq_data['Time']
print(time_info)
```

```
0 18.00.00
1 19.00.00
2 20.00.00
3 21.00.00
4 22.00.00
 ...
9352 10.00.00
9353 11.00.00
9354 12.00.00
9355 13.00.00
9356 14.00.00
Name: Time, Length: 9357, dtype: object
```

```
time_info = time_info.apply(lambda x : x.replace('.', ':'))
```

```
print(type(date_info))
print(type(time_info))
```

```
<class 'pandas.core.series.Series'>
<class 'pandas.core.series.Series'>
```

```
date_time = pd.concat([date_info, time_info], axis = 1)
```

```
date_time['ds'] = date_time['Date'].astype(str)+' '+date_time['Time'].astype(str)
```

```
date_time.head()
```

	Date	Time	ds
0	2004-03-10 18:00:00	2004-03-10 18:00:00	
1	2004-03-10 19:00:00	2004-03-10 19:00:00	
2	2004-03-10 20:00:00	2004-03-10 20:00:00	
3	2004-03-10 21:00:00	2004-03-10 21:00:00	
4	2004-03-10 22:00:00	2004-03-10 22:00:00	

```
date_time.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9357 entries, 0 to 9356
Data columns (total 3 columns):
 # Column Non-Null Count Dtype
--- ------ -------------- -----
 0 Date 9357 non-null datetime64[ns]
 1 Time 9357 non-null object
 2 ds 9357 non-null object
dtypes: datetime64[ns](1), object(2)
```



```
memory usage: 219.4+ KB
data = pd.DataFrame()
```

```
# Check if the 'ds' column contains the value "-200"
```

```
has_negative_200 = date_time['ds'].str.contains("-200")

# Count the number of occurrences of "-200"
count_negative_200 = has_negative_200.sum()

# Display the count and the rows where "-200" is present
print("Number of occurrences of '-200':", count_negative_200)
print("Rows where '-200' is present:")
print(date_time[has_negative_200])
```

```
Number of occurrences of '-200': 1
Rows where '-200' is present:
 Date Time ds
4887 2004-09-30 09:00:00:-200 2004-09-30 09:00:00:-200
```

```
# Filter out the row where '-200' is present in the 'ds' column
date_time = date_time[~has_negative_200]

# Reset the index after removing the row
date_time.reset_index(drop=True, inplace=True)

# Verify that '-200' is removed
print(date_time)
```

```
 Date Time ds
0 2004-03-10 18:00:00 2004-03-10 18:00:00
1 2004-03-10 19:00:00 2004-03-10 19:00:00
2 2004-03-10 20:00:00 2004-03-10 20:00:00
3 2004-03-10 21:00:00 2004-03-10 21:00:00
4 2004-03-10 22:00:00 2004-03-10 22:00:00
... ... ... ...
9351 2004-04-04 10:00:00 2004-04-04 10:00:00
9352 2004-04-04 11:00:00 2004-04-04 11:00:00
9353 2004-04-04 12:00:00 2004-04-04 12:00:00
9354 2004-04-04 13:00:00 2004-04-04 13:00:00
9355 2004-04-04 14:00:00 2004-04-04 14:00:00

[9356 rows x 3 columns]
```

```
data['ds'] = pd.to_datetime(date_time['ds'])
```

```
data['y'] = aq_data['RH']
data.head()
```

	ds	y
0	2004-03-10 18:00:00	48.9
1	2004-03-10 19:00:00	47.7
2	2004-03-10 20:00:00	54.0
3	2004-03-10 21:00:00	60.0
4	2004-03-10 22:00:00	59.6

```
pip install prophet
```

```
Requirement already satisfied: prophet in /usr/local/lib/python3.10/dist-packages (1.1.5)
Requirement already satisfied: cmdstanpy>=1.0.4 in /usr/local/lib/python3.10/dist-packages (from prophet) (1.2.2) Requirement already
satisfied: numpy>=1.15.4 in /usr/local/lib/python3.10/dist-packages (from prophet) (1.25.2) Requirement already satisfied:
matplotlib>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from prophet) (3.7.1) Requirement already satisfied: pandas>=1.0.4 in
/usr/local/lib/python3.10/dist-packages (from prophet) (2.0.3) Requirement already satisfied: holidays>=0.25 in
/usr/local/lib/python3.10/dist-packages (from prophet) (0.46) Requirement already satisfied: tqdm>=4.36.1 in
/usr/local/lib/python3.10/dist-packages (from prophet) (4.66.2) Requirement already satisfied: importlib-resources in
/usr/local/lib/python3.10/dist-packages (from prophet) (6.4.0) Requirement already satisfied: stanio<2.0.0,>=0.4.0 in
/usr/local/lib/python3.10/dist-packages (from cmdstanpy>=1.0.4->prophet) (0.5.0) Requirement already satisfied: python-dateutil in
/usr/local/lib/python3.10/dist-packages (from holidays>=0.25->prophet) (2.8.2) Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet) (1.2.1) Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet) (0.12.1)
```



```
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet) (4.51.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet) (24.0)
```

```
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet) (3.1.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.4->prophet) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.4->prophet) (2024.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil->holidays>=0.25->prophet) (1.16
```

```
from prophet import Prophet
# Python
m = Prophet()
m.fit(data)
```

```
INFO:prophet:Disabling yearly seasonality. Run prophet with yearly_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmp_xu8ph8c/0egn8v76.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmp_xu8ph8c/xs6z8615.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.10/dist-packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=925
15:02:18 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
15:02:19 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
<prophet.forecaster.Prophet at 0x7db96565b3a0>
```

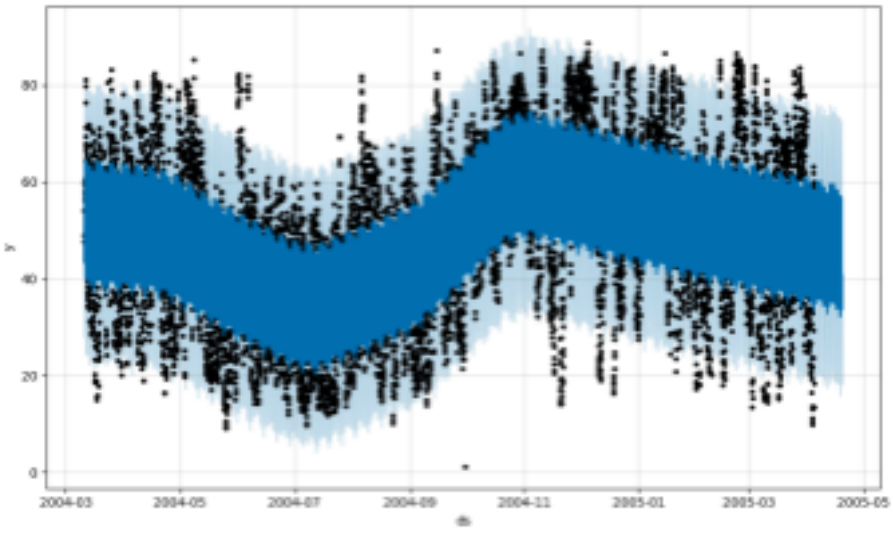
```
# Python
future = m.make_future_dataframe(periods=365, freq='H')
future.tail()
```

	ds
9716	2005-04-19 15:00:00
9717	2005-04-19 16:00:00
9718	2005-04-19 17:00:00
9719	2005-04-19 18:00:00
9720	2005-04-19 19:00:00

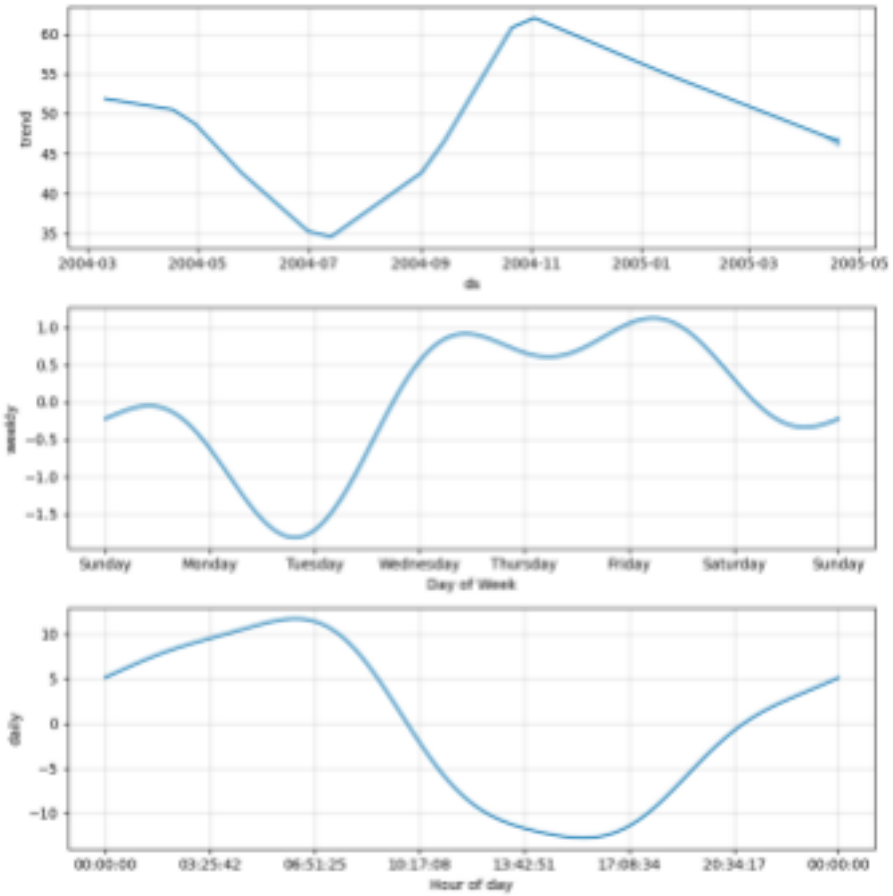
```
# Python
forecast = m.predict(future)
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

	ds	yhat	yhat_lower	yhat_upper
9716	2005-04-19 15:00:00	33.574348	19.214691	50.366622
9717	2005-04-19 16:00:00	33.586214	18.146560	49.022646
9718	2005-04-19 17:00:00	34.701250	18.733571	49.747510
9719	2005-04-19 18:00:00	37.207684	21.743954	52.404482
9720	2005-04-19 19:00:00	40.688308	25.041193	56.847734

```
fig1 = m.plot(forecast)
```

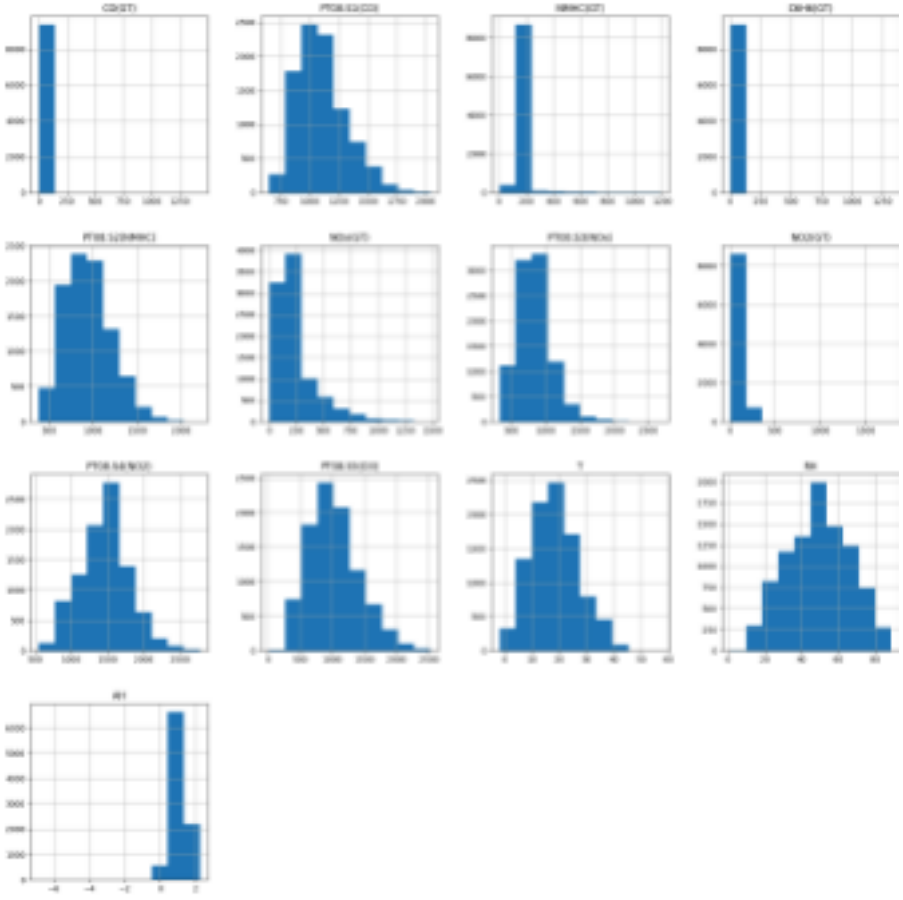



```
fig2 = m.plot_components(forecast)
```

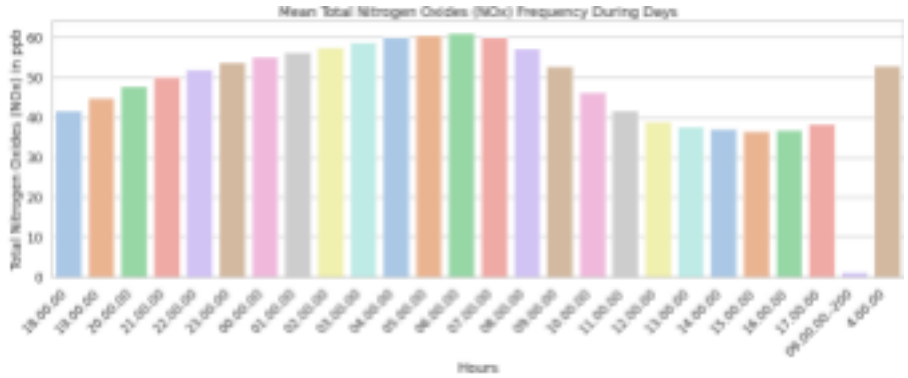
TIME SERIES ANALYSIS DONE

```
import matplotlib.pyplot as plt
aq_data.hist(figsize = (20,20))
plt.show()
```

```
import seaborn as sns
import matplotlib.pyplot as plt

# Define a custom pastel color palette
pastel_palette = sns.color_palette("pastel")

# Set the context and style
sns.set_context("talk")
sns.set_style("whitegrid")

# Plotting the bar plot with pastel colors
plt.figure(figsize=(20,6))
ax = sns.barplot(x='Time', y='RH', data=aq_data, ci=False, palette=pastel_palette)
plt.xlabel('Hours')
plt.ylabel('Total Nitrogen Oxides (NOx) in ppb') # Parts per billion (ppb)
plt.title("Mean Total Nitrogen Oxides (NOx) Frequency During Days")

# Rotate x-axis labels for better readability
ax.set_xticklabels(ax.get_xticklabels(), rotation=45, ha='right')

# Show plot
plt.show()
```

```
<ipython-input-133-ef8bbba6dc90>:13: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=('ci', False)` for the same effect.

 ax = sns.barplot(x='Time', y='RH', data=aq_data, ci=False, palette=pastel_palette)
<ipython-input-133-ef8bbba6dc90>:13: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.

 ax = sns.barplot(x='Time', y='RH', data=aq_data, ci=False, palette=pastel_palette)
<ipython-input-133-ef8bbba6dc90>:13: UserWarning:
The palette list has fewer values (10) than needed (26) and will cycle, which may produc
ax = sns.barplot(x='Time', y='RH', data=aq_data, ci=False, palette=pastel_palette)
<ipython-input-133-ef8bbba6dc90>:19: UserWarning: FixedFormatter should only be used tog
ax.set_xticklabels(ax.get_xticklabels(), rotation=45, ha='right')
```




```
X = aq_data.drop(['CO(GT)','Time'], axis=1)

y= aq_data['CO(GT)']
```



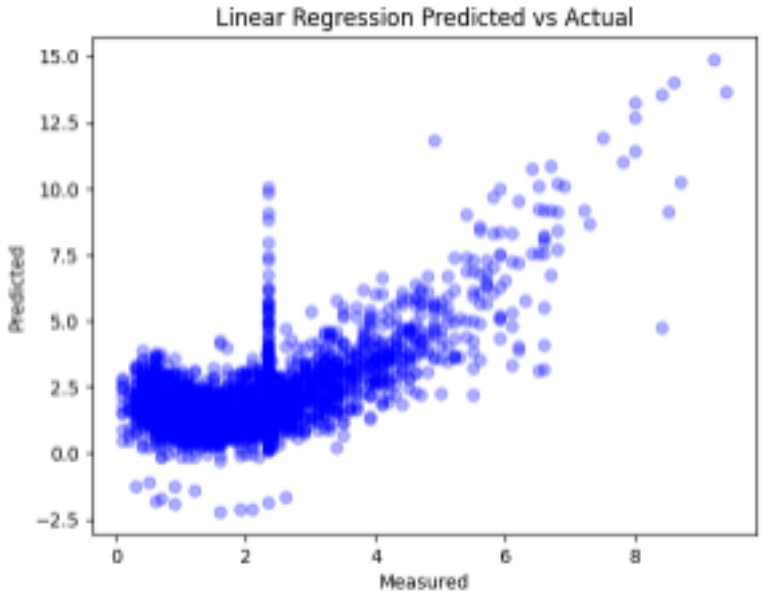
```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
from sklearn.preprocessing import RobustScaler
# Assuming x_train and x_test are DataFrames containing your training and testing data
# Drop non-numeric columns, such as date columns
x_train_numeric = x_train.select_dtypes(include=[np.number])
x_test_numeric = x_test.select_dtypes(include=[np.number])


# Scale the numeric features using RobustScaler
scaler = RobustScaler()
x_train_scaled = scaler.fit_transform(x_train_numeric)
x_test_scaled = scaler.transform(x_test_numeric)



from sklearn.linear_model import LinearRegression
lm = LinearRegression()
lm.fit(x_train_scaled, y_train)
```

▾ LinearRegression

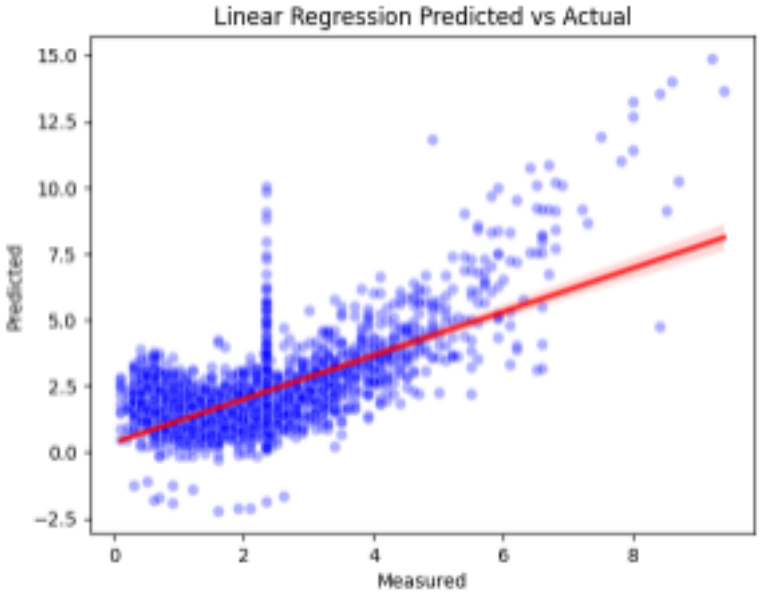
LinearRegression()

```
print(lm.intercept_)
prediction = lm.predict(x_test_scaled)
plt.scatter(y_test, prediction, c="blue", alpha=0.3)
plt.xlabel('Measured')
plt.ylabel('Predicted')
plt.title('Linear Regression Predicted vs Actual')
```

```
1.2386693603478671
Text(0.5, 1.0, 'Linear Regression Predicted vs Actual')
```




```
import seaborn as sns

# Scatter plot of predicted vs. actual values
sns.scatterplot(x=y_test, y=prediction, color='blue', alpha=0.3)
```




```
# Add a regression line
sns.regplot(x=y_test, y=prediction, scatter=False, color='red')

# Set labels and title
plt.xlabel('Measured')
plt.ylabel('Predicted')
plt.title('Linear Regression Predicted vs Actual')

# Show plot
plt.show()
```

```
score_train = lm.score(x_train_scaled, y_train)
print(score_train)
```

```
0.9945665301698585
```

```
score_test = lm.score(x_test_scaled, y_test)
score_test
```

```
0.14662097280515551
```

```
from sklearn import metrics
print('MAE:',metrics.mean_absolute_error(y_test, prediction))
print('MSE:',metrics.mean_squared_error(y_test, prediction))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test, prediction)))
```

```
MAE: 0.9106435381041353
MSE: 1.4381492617131857
RMSE: 1.199228611113488
```

REGRESSION DONE

```
X1 = aq_data.drop(['RH','Time'], axis=1)
y1 = aq_data['RH']
from sklearn.model_selection import train_test_split
x_train1, x_test1, y_train1, y_test1 = train_test_split(X1, y1, test_size=0.3,
random_state=0)
from sklearn.preprocessing import RobustScaler
# Assuming x_train and x_test are DataFrames containing your training and testing data
# Drop non-numeric columns, such as date columns
x_train_numeric1 = x_train1.select_dtypes(include=[np.number])
x_test_numeric1 = x_test1.select_dtypes(include=[np.number])
# Scale the numeric features using RobustScaler
scaler1 = RobustScaler()
x_train_scaled1 = scaler1.fit_transform(x_train_numeric1)
x_test_scaled1 = scaler1.transform(x_test_numeric1)

from sklearn.neighbors import KNeighborsRegressor
knn=KNeighborsRegressor(n_neighbors=5)
knn.fit(x_train_scaled1,y_train1)
prediction = knn.predict(x_test_scaled1)

from sklearn.metrics import confusion_matrix
# Calculate the mean of the target variable
threshold_value = y_train1.mean()

# Convert the regression problem into a classification one using the mean value as the
threshold
y_test_class = (y_test1 > threshold_value).astype(int)
prediction_class = (prediction > threshold_value).astype(int)

# Calculate confusion matrix
cm = confusion_matrix(y_test_class, prediction_class)

print("Confusion Matrix:")
        print(cm)
```

```
Confusion Matrix:
[[1282 179]
[ 140 1207]]
```

```
knn_train = knn.score(x_train_scaled1,y_train1)
print(knn_train)

kreg_test = knn.score(x_test_scaled1,y_test1)
print(kreg_test)
```

```
0.8931120086719705
0.8423852118617776
```

KNN DONE

CONCLUSION :- After conducting a comprehensive analysis utilizing various regression techniques, including linear regression, and k-nearest neighbors (KNN), Forecasting we have gained valuable insights into the factors influencing pollutant levels. Through our exploration, we identified significant predictors and their impacts on pollution levels, aiding in the development of effective mitigation strategies. By leveraging regression analysis, we can now make informed predictions and recommendations to mitigate

pollution, thereby contributing to environmental sustainability and public health.

REFERENCES :-

https://www.kaggle.com/code/parimalbhoyar25/air-quality-uci

FB PROPHET DOCUMENTATION:

https://facebook.github.io/prophet/docs/quick_start.html