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

```
df=pd.read_csv('/content/archive (35).zip')
df.head(20)
```

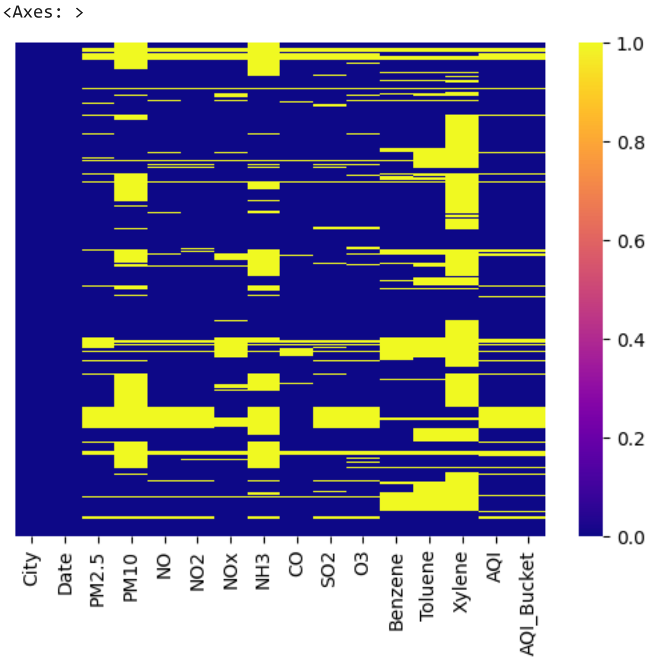
	City	Date	PM2.5	PM10	NO	NO2	NOx	NH3	CO	SO2	O3	Benz
0	Ahmedabad	2015-01-01	NaN	NaN	0.92	18.22	17.15	NaN	0.92	27.64	133.36	0
1	Ahmedabad	2015-01-02	NaN	NaN	0.97	15.69	16.46	NaN	0.97	24.55	34.06	3
2	Ahmedabad	2015-01-03	NaN	NaN	17.40	19.30	29.70	NaN	17.40	29.07	30.70	6
3	Ahmedabad	2015-01-04	NaN	NaN	1.70	18.48	17.97	NaN	1.70	18.59	36.08	4
4	Ahmedabad	2015-01-05	NaN	NaN	22.10	21.42	37.76	NaN	22.10	39.33	39.31	7
5	Ahmedabad	2015-01-06	NaN	NaN	45.41	38.48	81.50	NaN	45.41	45.76	46.51	5
6	Ahmedabad	2015-01-07	NaN	NaN	112.16	40.62	130.77	NaN	112.16	32.28	33.47	0
7	Ahmedabad	2015-01-08	NaN	NaN	80.87	36.74	96.75	NaN	80.87	38.54	31.89	0
8	Ahmedabad	2015-01-09	NaN	NaN	29.16	31.00	48.00	NaN	29.16	58.68	25.75	0
9	Ahmedabad	2015-01-10	NaN	NaN	NaN	7.04	0.00	NaN	NaN	8.29	4.55	0
10	Ahmedabad	2015-01-11	NaN	NaN	132.07	55.80	24.53	NaN	132.07	25.03	6.79	0
11	Ahmedabad	2015-01-12	NaN	NaN	52.04	40.67	90.24	NaN	52.04	51.84	45.89	2
12	Ahmedabad	2015-01-13	NaN	NaN	48.82	44.20	87.09	NaN	48.82	68.21	35.16	9

```
#check for null values
df.isna().sum()
```

```
City             0
Date             0
PM2.5         4598
PM10         11140
NO            3582
NO2           3585
NOx           4185
NH3          10328
CO            2059
SO2           3854
O3            4022
Benzene       5623
Toluene       8041
Xylene       18109
AQI           4681
AQI_Bucket    4681
dtype: int64
```

```
#plot heatmap for null values
sns.heatmap(df.isnull(), yticklabels=False, cmap='plasma')
```

```
<Axes: >
```



```
df.shape
```

```
(29531, 16)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 29531 entries, 0 to 29530
Data columns (total 16 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   City        29531 non-null  object 
 1   Date        29531 non-null  object 
 2   PM2.5       24933 non-null  float64
 3   PM10        18391 non-null  float64
 4   NO          25949 non-null  float64
 5   NO2         25946 non-null  float64
 6   NOx         25346 non-null  float64
 7   NH3         19203 non-null  float64
 8   CO          27472 non-null  float64
 9   SO2         25677 non-null  float64
 10  O3          25509 non-null  float64
 11  Benzene     23908 non-null  float64
 12  Toluene     21490 non-null  float64
 13  Xylene      11422 non-null  float64
 14  AQI         24850 non-null  float64
 15  AQI_Bucket  24850 non-null  object 
dtypes: float64(13), object(3)
memory usage: 3.6+ MB
```

```
#unique values in each column]
cols=df.columns
for cols in (df.columns):
    print ("Unique columns of", cols, "\n", df[cols].unique())
    print ("--------------")
```

```
Unique columns of City
 ['Ahmedabad' 'Aizawl' 'Amaravati' 'Amritsar' 'Bengaluru' 'Bhopal'
 'Brajrajnagar' 'Chandigarh' 'Chennai' 'Coimbatore' 'Delhi' 'Ernakulam'
 'Gurugram' 'Guwahati' 'Hyderabad' 'Jaipur' 'Jorapokhar' 'Kochi' 'Kolkata'
 'Lucknow' 'Mumbai' 'Patna' 'Shillong' 'Talcher' 'Thiruvananthapuram'
 'Visakhapatnam']
--------------
Unique columns of Date
 ['2015-01-01' '2015-01-02' '2015-01-03' ... '2020-06-29' '2020-06-30'
 '2020-07-01']
--------------
Unique columns of PM2.5
 [  nan 73.24 83.13 ... 33.17 25.4  24.38]
--------------
Unique columns of PM10
 [   nan 141.54 122.41 ...  58.54  32.27  66.  ]
--------------
Unique columns of NO
 [ 0.92  0.97 17.4  ... 29.35 30.16 18.55]
--------------
Unique columns of NO2
 [18.22 15.69 19.3  ... 58.99 52.1  53.59]
--------------
Unique columns of NOx
 [17.15 16.46 29.7  ... 42.33 45.87  7.07]
--------------
Unique columns of NH3
 [  nan 26.64 25.63 ...  4.1  28.34 42.86]
--------------
Unique columns of CO
 [ 0.92  0.97 17.4  ...  4.85  5.59  4.56]
--------------
Unique columns of SO2
 [27.64 24.55 29.07 ... 26.63 31.16 21.67]
--------------
Unique columns of O3
 [133.36  34.06  30.7  ...  70.53  60.29  34.85]
--------------
Unique columns of Benzene
 [ 0.    3.68  6.8  ...  8.1   8.85 10.32]
--------------
Unique columns of Toluene
 [2.000e-02 5.500e+00 1.640e+01 ... 2.006e+01 1.154e+01 1.026e+01]
--------------
Unique columns of Xylene
 [ 0.    3.77  2.25 ...  8.06  5.04 12.41]
--------------
Unique columns of AQI
 [  nan  209.  328.  514.  782.  914.  660.  294.  149.  190.  247.  379.
  341.  256.  388.  288.  510.  761.  475.  536.  479.  592.  427.  588.
 1141.  669. 1247.  411.  292.  189.  408.  383.  780.  233.  297.  330.
  252.  244.  234.  219.  118.  231.  286.  883.  720.  570.  589.  818.
  737.  585.  616.  437.  321.  372.  339.  324.  222.  169.  220.  303.
  314.  378.  415.  126.  175.  226.  315.  678.  774.  285.  358.  357.
  214.  253.  239.  300.  317.  240.  344.  503.  308.  227.  120.  158.
  177.  201.  211.  221.  481.  352.  327.  577.  212.  280.  162.  569.
  128.  129.  152.  176.  167.  106.  174.  185.  179.  192.  310.  223.
  191.  184.  215.  251.  216.  237.  198.  195.  187.  181.  188.  196.
```

```
#value counts of each column
for cols in (df.columns):
    print ("Value counts of", cols, "\n", df[cols].value_counts())
    print ("--------------")
```

```
Value counts of City
 Ahmedabad             2009
Delhi                 2009
```

```
Mumbai                2009
Bengaluru             2009
Lucknow               2009
Chennai               2009
Hyderabad             2006
Patna                 1858
Gurugram              1679
Visakhapatnam         1462
Amritsar              1221
Jorapokhar            1169
Jaipur                1114
Thiruvananthapuram    1112
Amaravati              951
Brajrajnagar           938
Talcher                925
Kolkata                814
Guwahati               502
Coimbatore             386
Shillong               310
Chandigarh             304
Bhopal                 289
Ernakulam              162
Kochi                  162
Aizawl                 113
Name: City, dtype: int64
--------------
Value counts of Date
 2020-07-01    26
2020-04-08    26
2020-04-10    26
2020-04-11    26
2020-04-12    26
              ..
2015-04-10     7
2015-01-04     7
2015-01-03     6
2015-01-02     6
2015-01-01     6
Name: Date, Length: 2009, dtype: int64
--------------
Value counts of PM2.5
 11.00     19
20.75     12
27.82     11
29.75     10
18.81     10
          ..
130.41     1
217.70     1
175.19     1
126.92     1
24.38      1
Name: PM2.5, Length: 11716, dtype: int64
--------------
Value counts of PM10
```

```
#fill the null values by mean()
pm2_mean=df["PM2.5"].mean()
df["PM2.5"].fillna(pm2_mean,inplace=True)
pm10_mean=df["PM10"].mean()
df["PM10"].fillna(pm10_mean,inplace=True)
no_mean=df["NO"].mean()
df["NO"].fillna(no_mean,inplace=True)
no2_mean=df["NO2"].mean()
df["NO2"].fillna(no2_mean,inplace=True)
nox_mean=df["NOx"].mean()
df["NOx"].fillna(nox_mean,inplace=True)
nh_mean=df["NH3"].mean()
df["NH3"].fillna(nh_mean,inplace=True)
co_mean=df["CO"].mean()
df["CO"].fillna(co_mean,inplace=True)
so2_mean=df["SO2"].mean()
df["SO2"].fillna(so2_mean,inplace=True)
o3_mean=df["O3"].mean()
df["O3"].fillna(o3_mean,inplace=True)
benz_mean=df["Benzene"].mean()
df["Benzene"].fillna(benz_mean,inplace=True)
tol_mean=df["Toluene"].mean()
df["Toluene"].fillna(tol_mean,inplace=True)
xy_mean=df["Xylene"].mean()
df["Xylene"].fillna(xy_mean,inplace=True)
aqi_mean=df["AQI"].mean()
df["AQI"].fillna(aqi_mean,inplace=True)
```

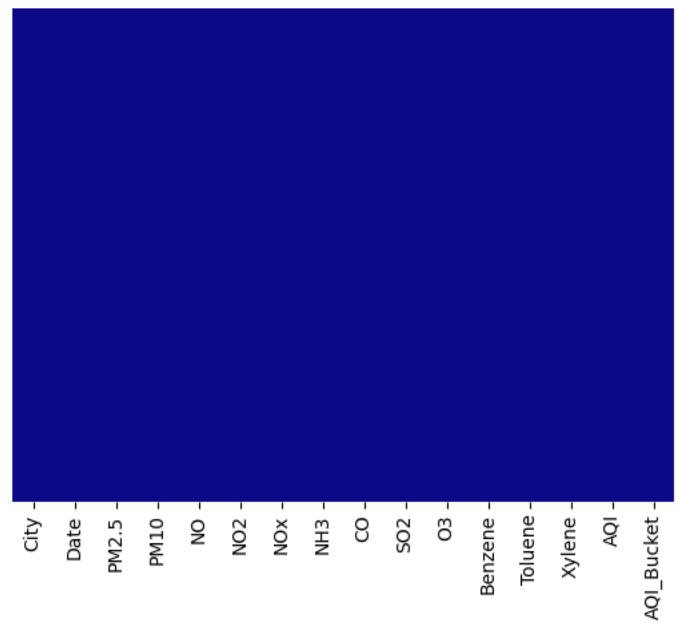
```
df.isna().sum()
```

```
City             0
Date             0
PM2.5            0
PM10             0
NO               0
NO2              0
NOx              0
NH3              0
CO               0
SO2              0
O3               0
Benzene          0
Toluene          0
Xylene           0
AQI              0
AQI_Bucket    4681
dtype: int64
```

```
#Fill the null values in the 'AQI_Bucket' column with 'Unknown'
df['AQI_Bucket'].fillna('Unknown', inplace=True)
```

```
#heat map for null values
sns.heatmap(df.isnull(), yticklabels=False, cbar=False, cmap='plasma')
```

<Axes: >




```
df
```

	City	Date	PM2.5	PM10	NO	NO2	NOx	NH3	CO	SO2	O3	Ber
0	Ahmedabad	2015-01-01	67.450578	118.127103	0.92	18.22	17.15	23.483476	0.92	27.64	133.36	0.(
1	Ahmedabad	2015-01-02	67.450578	118.127103	0.97	15.69	16.46	23.483476	0.97	24.55	34.06	3.6
2	Ahmedabad	2015-01-03	67.450578	118.127103	17.40	19.30	29.70	23.483476	17.40	29.07	30.70	6.8
3	Ahmedabad	2015-01-04	67.450578	118.127103	1.70	18.48	17.97	23.483476	1.70	18.59	36.08	4.4
4	Ahmedabad	2015-01-05	67.450578	118.127103	22.10	21.42	37.76	23.483476	22.10	39.33	39.31	7.(
...	...	...	...	...	...	...	...	...	...	...	...	
29526	Visakhapatnam	2020-06-27	15.020000	50.940000	7.68	25.06	19.54	12.470000	0.47	8.55	23.30	2.2
29527	Visakhapatnam	2020-06-28	24.380000	74.090000	3.42	26.06	16.53	11.990000	0.52	12.72	30.14	0.7

```
#find the correlation between elements
corr=df.corr()
corr
```

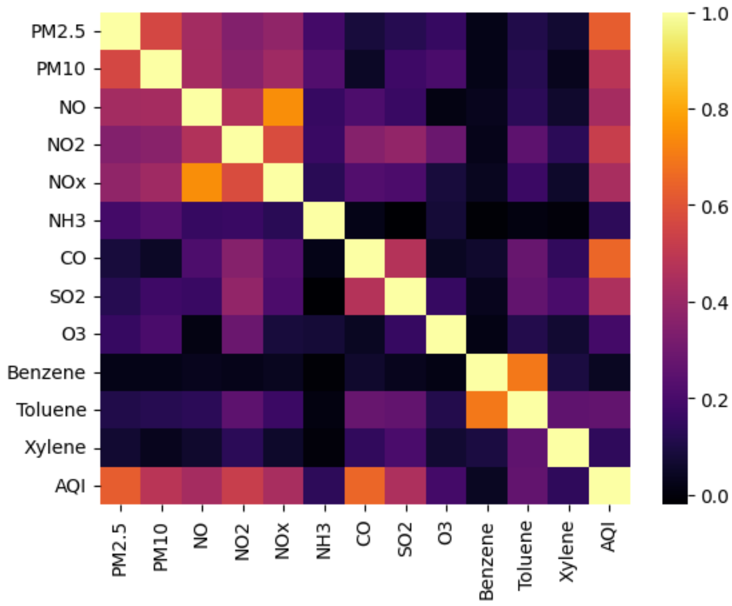
```
<ipython-input-21-5ff5af9e1e2c>:2: FutureWarning: The default value of numeric_only in DataFrame.corr i
  corr=df.corr()
```

	PM2.5	PM10	NO	NO2	NOx	NH3	CO	SO2	O3	Benze
PM2.5	1.000000	0.558079	0.426375	0.344341	0.380725	0.189227	0.086663	0.119512	0.155330	0.0219
PM10	0.558079	1.000000	0.431006	0.359165	0.415133	0.223025	0.047517	0.176188	0.203595	0.0192
NO	0.426375	0.431006	1.000000	0.462402	0.746223	0.156394	0.211639	0.166190	0.014218	0.0339
NO2	0.344341	0.359165	0.462402	1.000000	0.574190	0.165984	0.353237	0.382758	0.285448	0.0250
NOx	0.380725	0.415133	0.746223	0.574190	1.000000	0.128051	0.225097	0.208355	0.083063	0.0373
NH3	0.189227	0.223025	0.156394	0.165984	0.128051	1.000000	0.020029	-0.021005	0.078688	-0.0118
CO	0.086663	0.047517	0.211639	0.353237	0.225097	0.020029	1.000000	0.472583	0.039787	0.0613
SO2	0.119512	0.176188	0.166190	0.382758	0.208355	-0.021005	0.472583	1.000000	0.156610	0.0330
O3	0.155330	0.203595	0.014218	0.285448	0.083063	0.078688	0.039787	0.156610	1.000000	0.0187
Benzene	0.021934	0.019215	0.033901	0.025082	0.037383	-0.011864	0.061351	0.033059	0.018748	1.0000
Toluene	0.107788	0.121983	0.134201	0.254074	0.168780	0.007442	0.274882	0.265522	0.113683	0.6946
Xylene	0.070459	0.031256	0.059494	0.133037	0.056920	-0.002215	0.145190	0.203766	0.068016	0.0929
AQI	0.628860	0.484497	0.430600	0.522994	0.438363	0.137436	0.649679	0.452768	0.188590	0.0415

Visualization

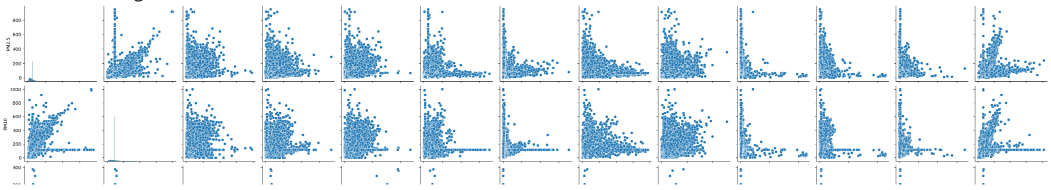
```
#heatmap of correlation
sns.heatmap(corr, cmap='inferno')
```

```
<Axes: >
```

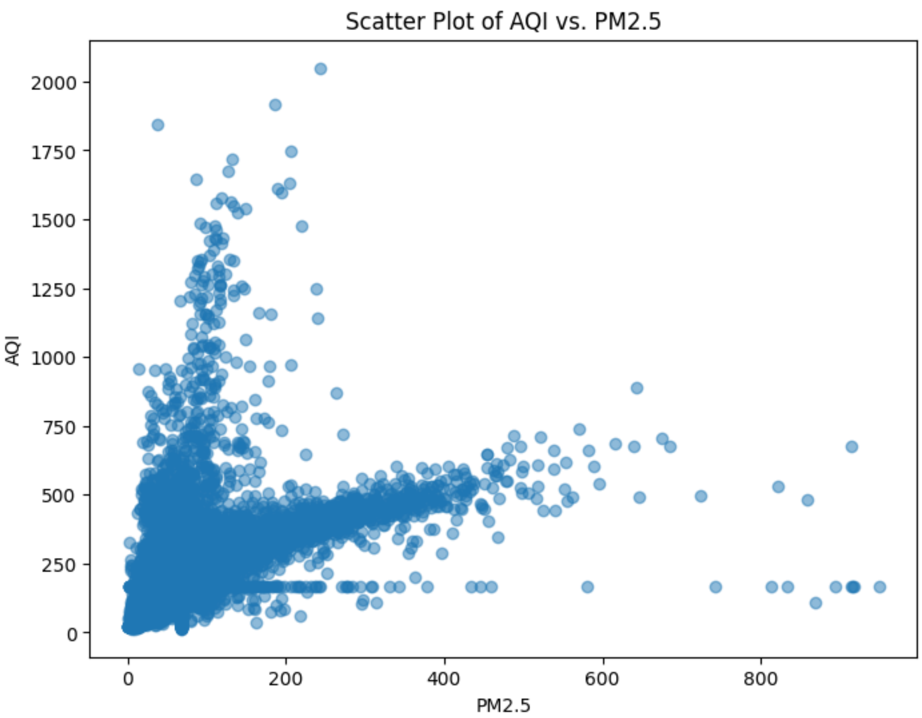



```
#pairplot of df
sns.pairplot(data=df)
```

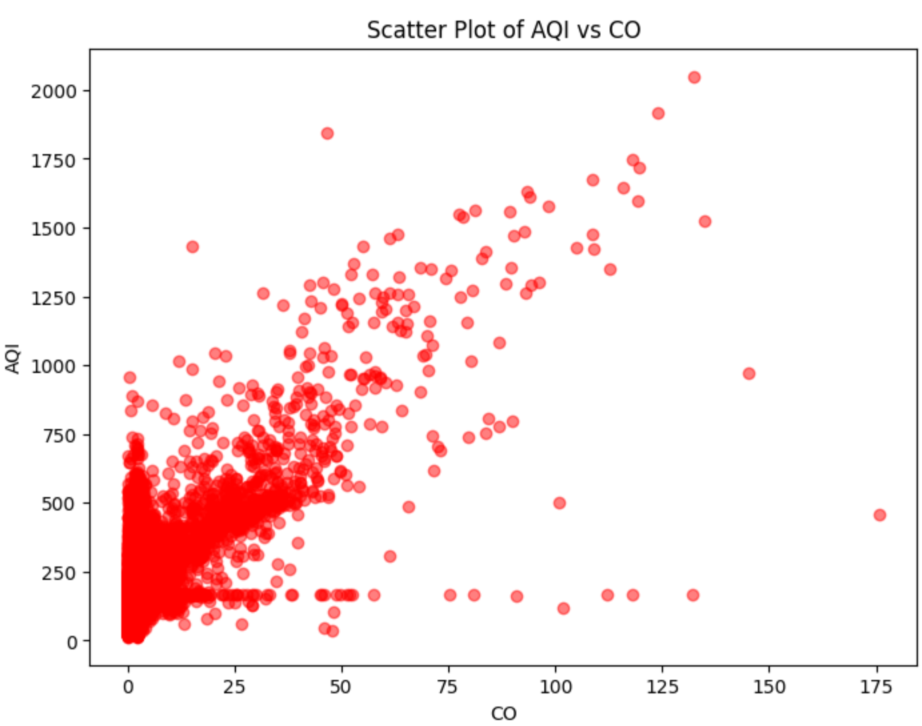
```
<seaborn.axisgrid.PairGrid at 0x7dccb8613670>
```



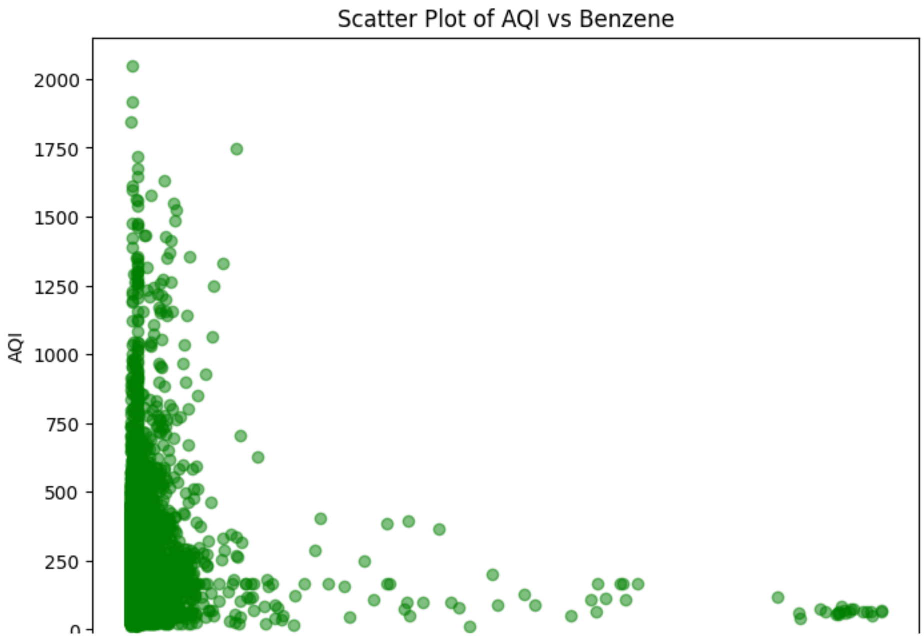
```
#Scatter Plot of AQI vs PM2.5
plt.figure(figsize=(8, 6))
plt.scatter(df['PM2.5'], df['AQI'], alpha=0.5)
plt.xlabel('PM2.5')
plt.ylabel('AQI')
plt.title('Scatter Plot of AQI vs. PM2.5')
plt.show()
```



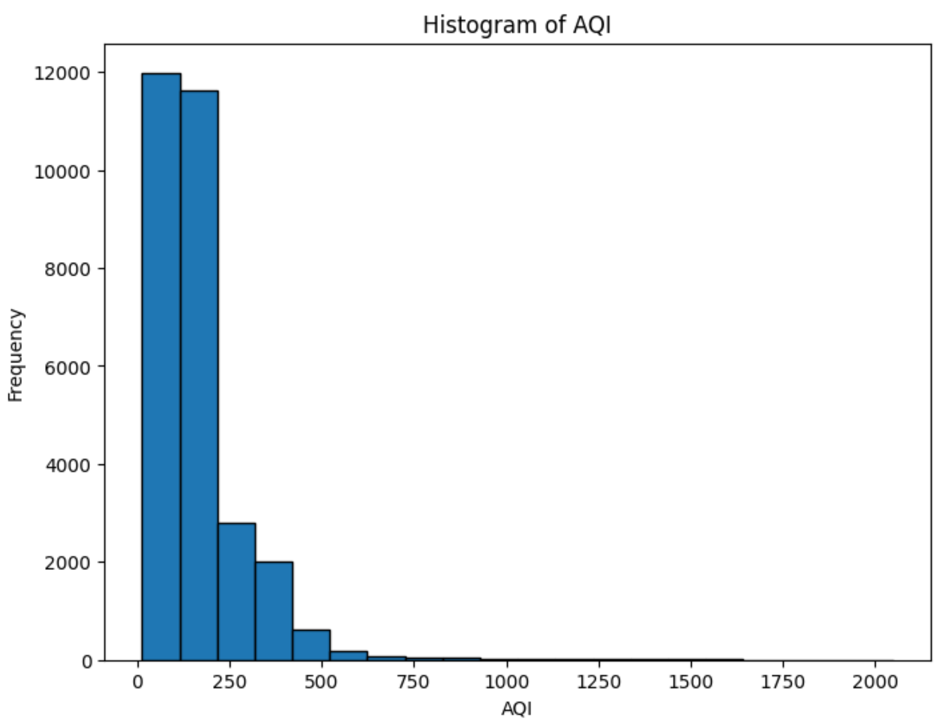

```
#Scatter Plot of AQI vs CO
plt.figure(figsize=(8, 6))
plt.scatter(df['CO'], df['AQI'], color='red', alpha=0.5)
plt.xlabel('CO')
plt.ylabel('AQI')
plt.title('Scatter Plot of AQI vs CO')
plt.show()
```



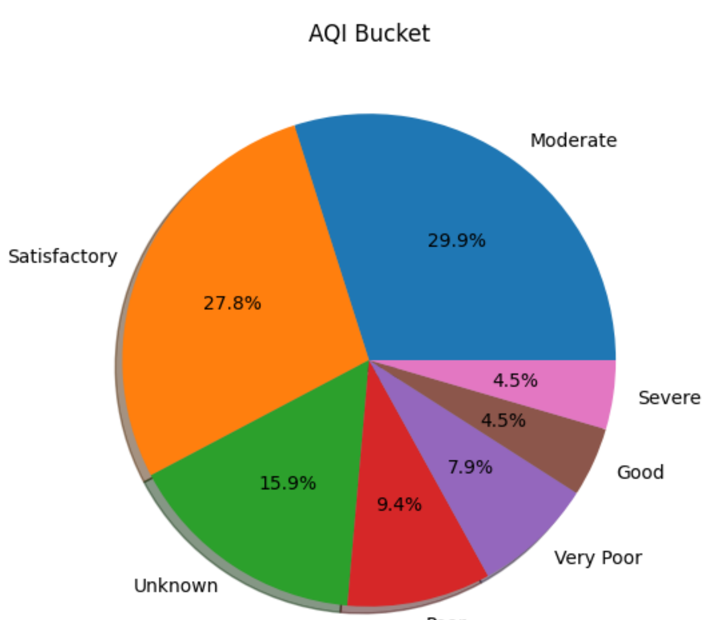

```
#Scatter Plot of AQI vs Benzene
plt.figure(figsize=(8, 6))
plt.scatter(df['Benzene'], df['AQI'], color='green', alpha=0.5)
plt.xlabel('Benzene')
plt.ylabel('AQI')
plt.title('Scatter Plot of AQI vs Benzene')
plt.show()
```



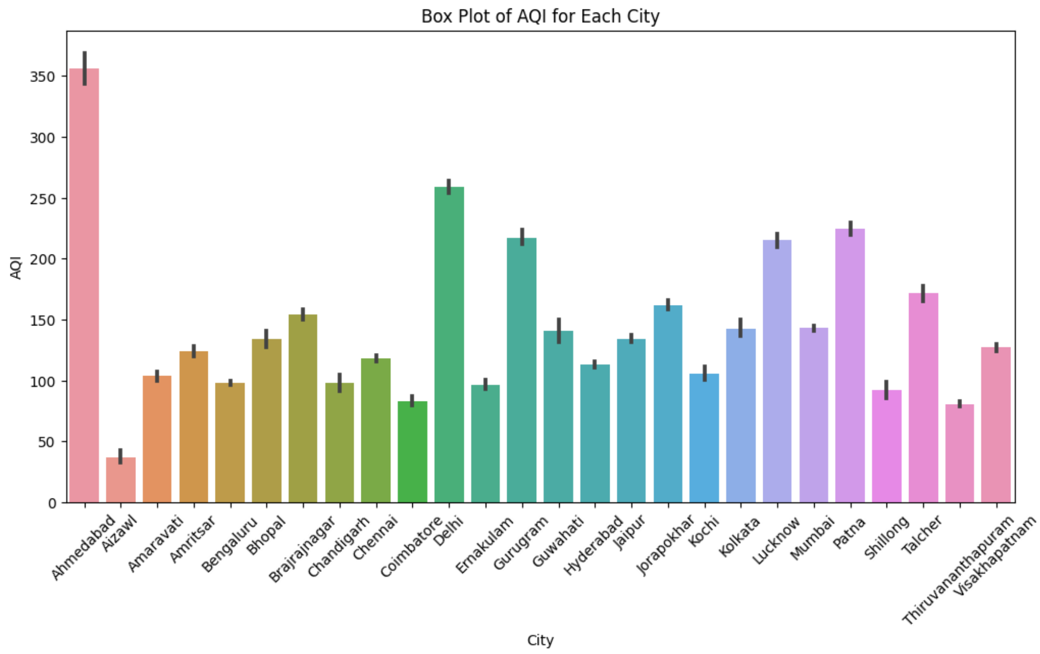

```
#Histogram of AQI
plt.figure(figsize=(8, 6))
plt.hist(df['AQI'], bins=20, edgecolor='black')
plt.xlabel('AQI')
plt.ylabel('Frequency')
plt.title('Histogram of AQI')
plt.show()
```



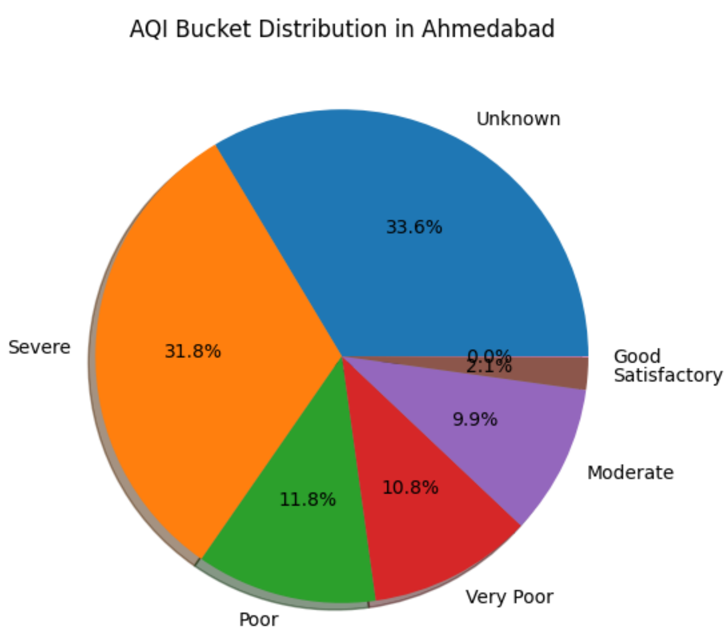

```
#Count the number of occurrences for each City
aqi_bucket_counts=df['AQI_Bucket'].value_counts()
plt.figure(figsize=(10, 6))
plt.pie(aqi_bucket_counts, labels=aqi_bucket_counts.index, autopct='%1.1f%%', shadow=True)
plt.title('AQI Bucket')
plt.show()
```



```
#Box Plot of AQI for each city
plt.figure(figsize=(12, 6))
sns.barplot(x='City', y='AQI', data=df)
plt.xlabel('City')
plt.ylabel('AQI')
plt.title('Box Plot of AQI for Each City')
plt.xticks(rotation=45)
plt.show()
```




```
#pie chart of AQI Bucket distribution for specific city (Ahmedabad)
ahmedabad_data=df[df['City'] == 'Ahmedabad']
ahmedabad_aqi_bucket_counts=ahmedabad_data['AQI_Bucket'].value_counts()
plt.figure(figsize=(8, 6))
plt.pie(ahmedabad_aqi_bucket_counts, labels=ahmedabad_aqi_bucket_counts.index, autopct='%1.1f%%', shadow=True)
plt.title('AQI Bucket Distribution in Ahmedabad')
plt.show()
```




```
#seperate the feature column as x and target column as y
x=df.iloc[:,2:13].values
y=df.iloc[:,-2].values
```

```
x.shape
```

```
(29531, 11)
```

```
y.shape
```

```
(29531,)
```

```
#train test split
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
```

```
x_train.shape
```

```
(23624, 11)
```

```
x_test.shape
```

```
(5907, 11)
```

```
#normalize
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_train=sc.fit_transform(x_train)
x_test=sc.fit_transform(x_test)
```

```
import pandas as pd
from sklearn.linear_model import LinearRegression, Lasso
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score,mean_squared_error

#Linear Regression
linear_model=LinearRegression()
linear_model.fit(x_train, y_train)
linear_y_pred=linear_model.predict(x_test)
linear_mse=mean_squared_error(y_test, linear_y_pred)
linear_r2=r2_score(y_test, linear_y_pred)
#Lasso Regression
lasso_model=Lasso(alpha=0.1) # You can adjust the alpha value as needed
lasso_model.fit(x_train, y_train)
lasso_y_pred=lasso_model.predict(x_test)
lasso_mse=mean_squared_error(y_test, lasso_y_pred)
lasso_r2=r2_score(y_test, lasso_y_pred)
#Random Forest Regressor
rf_model=RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(x_train, y_train)
rf_y_pred=rf_model.predict(x_test)
rf_mse=mean_squared_error(y_test, rf_y_pred)
rf_r2=r2_score(y_test, rf_y_pred)
```

```
#print the r2 score results
print("Linear Regression:")
print("R-squared (R2) Score:", linear_r2)
print("\nLasso Regression:")
print("R-squared (R2) Score:", lasso_r2)
print("\nRandom Forest Regressor:")
print("R-squared (R2) Score:", rf_r2)
```

```
Linear Regression:
R-squared (R2) Score: 0.8002508817468608

Lasso Regression:
R-squared (R2) Score: 0.8004094234743535

Random Forest Regressor:
R-squared (R2) Score: 0.8708695092053735
```

```
#print the mean scored error results
print("Logistic Regression:")
print("Mean Squared Error:", linear_mse)
print("\nLasso Regression:")
print("Mean Squared Error:", lasso_mse)
print("\nRandom Forest Regressor:")
print("Mean Squared Error:", rf_mse)
```

```
Logistic Regression:
Mean Squared Error: 3011.326101830205

Lasso Regression:
Mean Squared Error: 3008.935999453766

Random Forest Regressor:
Mean Squared Error: 1946.712059971221
```

```
#compare the MSE values to see which regressor has the lowest MSE
min_mse=min(linear_mse, lasso_mse, rf_mse)
best_model=None
if min_mse==linear_mse:
   best_model="Logistic Regression"
elif min_mse==lasso_mse:
   best_model="Lasso Regression"
else:
   best_model="Random Forest Regressor"
print("\nThe best performing model is:", best_model, "with mse", rf_mse)
```

```
The best performing model is: Random Forest Regressor with mse 1946.712059971221
```

```
#compare the R2 scores to see which regressor has the highest R2 score.
max_r2=max(linear_r2, lasso_r2, rf_r2)
best_model=None
if max_r2==linear_r2:
   best_model="Linear Regression"
elif max_r2==lasso_r2:
   best_model="Lasso Regression"
else:
   best_model="Random Forest Regressor"
print("\nThe best performing model is:", best_model, "with r2 score", rf_r2)
```

```
The best performing model is: Random Forest Regressor with r2 score 0.8708695092053735
```

```
#finding the difference between y test and y pred
results=pd.DataFrame()
results['Actual']=y_test
results['Predicted']=rf_y_pred
results['Difference']=y_test-rf_y_pred
results.sort_index()
```

	Actual	Predicted	Difference
0	166.463581	133.905167	32.558414
1	166.463581	133.905167	32.558414
2	137.000000	116.539272	20.460728
3	190.000000	180.330000	9.670000
4	339.000000	351.730000	-12.730000
...	...	...	...
5902	79.000000	97.934636	-18.934636
5903	91.000000	100.739272	-9.739272
5904	106.000000	104.178543	1.821457
5905	68.000000	69.054636	-1.054636
5906	136.000000	150.490994	-14.490994

5907 rows × 3 columns

Random forest classifier is performing a high accuracy.

✓ 0s completed at 5:17 AM