databricksDATA603_FinalProject (1)

US Geospatial and Temporal Pollution Analysis

Name: Harshit Shrimali

Dataset Background

Data Source: Data was sourced from Kaggle (https://www.kaggle.com/sogun3/uspollution (https://www.kaggle.com/sogun3/uspollution))

Data Lineage: Kaggle data was sourced from US EPA Air Quality database (https://www.epa.gov/outdoor-air-quality-data/download-daily-data (https://www.epa.gov/outdoor-air-quality-data/download-daily-data))

Dataset Content

Carbon Monoxide (CO), Nitrogen Dioxide (NO2), Sulfur Dioxide (SO2), and Ozone (O3) pollution measurements taken at sites across the United States

Time series data of pollution measurements for the years 2000-2016

CO, NO2, SO2, and O3 Air Quality Index (AQI) values

Problem Statement: What are the trends in EPA pollution measurements over time and what temporal and geospatial patterns can be deduced from the data?

Notes:

mlxtend and fbprophet libraries will need to be installed in the Cluster

Use 10.1 Databricks runtime version

```
%fs ls /FileStore/tables/pollution_us_2000_2016.csv
```

	path	name	siz
1	dbfs:/FileStore/tables/pollution_us_2000_2016.csv	pollution_us_2000_2016.csv	400

Showing all 1 rows.

```
from pyspark.sql.types import *
from pyspark.sql.functions import *

data_file = '/FileStore/tables/pollution_us_2000_2016.csv'
```

```
data_schema = StructType([StructField('Index', IntegerType(), True),
StructField('State Code', IntegerType(), True),
                     StructField('County Code', IntegerType(), True),
                     StructField('SiteNum', IntegerType(), True),
                     StructField('Address', StringType(), True),
                     StructField('State', StringType(), True),
                     StructField('County', StringType(), True),
                     StructField('City', StringType(), True),
                     StructField('Date', DateType(), True),
                     StructField('NO2 Units', StringType(), True),
                     StructField('NO2 Mean', FloatType(), True),
                     StructField('NO2 Max Value', FloatType(), True),
                     StructField('NO2 Max Hour', IntegerType(),True),
                     StructField('NO2 AQI', IntegerType(), True),
                     StructField('O3 Units', StringType(), True),
                     StructField('O3 Mean', FloatType(), True),
                     StructField('O3 Max Value', FloatType(), True),
                     StructField('O3 Max Hour', IntegerType(), True),
                     StructField('O3 AQI', IntegerType(), True),
                     StructField('SO2 Units', StringType(), True),
                     StructField('SO2 Mean', FloatType(), True),
                     StructField('SO2 Max Value', FloatType(), True),
                     StructField('SO2 Max Hour', IntegerType(),True),
                     StructField('SO2 AQI', FloatType(), True),
                     StructField('CO Units', StringType(), True),
                     StructField('CO Mean', FloatType(), True),
                     StructField('CO Max Value', FloatType(), True),
                     StructField('CO Max Hour', IntegerType(),True),
                     StructField('CO AQI', FloatType(), True)])
```

```
df = spark.read.csv(data_file, header=True, schema=data_schema)
```

```
df.cache()
```

```
Out[4]: DataFrame[Index: int, State Code: int, County Code: int, SiteNum: int,
Address: string, State: string, County: string, City: string, Date: date, NO2
Units: string, NO2 Mean: float, NO2 Max Value: float, NO2 Max Hour: int, NO2 A
QI: int, O3 Units: string, O3 Mean: float, O3 Max Value: float, O3 Max Hour: i
nt, O3 AQI: int, SO2 Units: string, SO2 Mean: float, SO2 Max Value: float, SO2
Max Hour: int, SO2 AQI: float, CO Units: string, CO Mean: float, CO Max Value:
float, CO Max Hour: int, CO AQI: float]
```

```
df.printSchema()
```

```
root
 |-- Index: integer (nullable = true)
 |-- State Code: integer (nullable = true)
 |-- County Code: integer (nullable = true)
 |-- SiteNum: integer (nullable = true)
 |-- Address: string (nullable = true)
 |-- State: string (nullable = true)
 |-- County: string (nullable = true)
 |-- City: string (nullable = true)
 |-- Date: date (nullable = true)
 |-- NO2 Units: string (nullable = true)
 |-- NO2 Mean: float (nullable = true)
 |-- NO2 Max Value: float (nullable = true)
 |-- NO2 Max Hour: integer (nullable = true)
 |-- NO2 AQI: integer (nullable = true)
 |-- O3 Units: string (nullable = true)
 |-- O3 Mean: float (nullable = true)
 |-- O3 Max Value: float (nullable = true)
 |-- O3 Max Hour: integer (nullable = true)
 |-- O3 AQI: integer (nullable = true)
 |-- SO2 Units: string (nullable = true)
```

```
display(df)
```

	Index	State Code	County Code	SiteNum	Address
1	0	4	13	3002	1645 E ROOSEVELT S
2	1	4	13	3002	1645 E ROOSEVELT S
3	2	4	13	3002	1645 E ROOSEVELT S
4	3	4	13	3002	1645 E ROOSEVELT S
5	4	4	13	3002	1645 E ROOSEVELT S
6	5	4	13	3002	1645 E ROOSEVELT S
7	6	4	13	3002	1645 E ROOSEVELT S

Truncated results, showing first 1000 rows.

```
from pyspark.sql.functions import when, count, col
df.select([count(when(isnull(c), c)).alias(c) for c in df.columns]).show()
```

```
+-----+----------+-----------+-------+-------+-----+------+----+----+---------
+--------+-------------+------------+-------+--------+-------+-----------+---
--------+------+---------+---------+-------------+------------+-------+--------
+--------+------------+----------+------+
|Index|State Code|County Code|SiteNum|Address|State|County|City|Date|NO2 Units
|NO2 Mean|NO2 Max Value|NO2 Max Hour|NO2 AQI|O3 Units|O3 Mean|O3 Max Value|O3
Max Hour|O3 AQI|SO2 Units|SO2 Mean|SO2 Max Value|SO2 Max Hour|SO2 AQI|CO Units
```

```
|CO Mean|CO Max Value|CO Max Hour|CO AQI|
+-----+----------+-----------+-------+-------+-----+------+----+----+---------
+--------+-------------+------------+-------+--------+-------+------------+---
--------+------+---------+--------+-------------+------------+-------+--------
+-------+------------+-----------+------+
|    0|         0|          0|      0|      0|    0|     0|   0|   0|        0
|       0|            0|           0|      0|       0|      0|           0|
0|     0|        0|       0|            0|           0| 872907|       0|
0|           0|          0|873323|
+-----+----------+-----------+-------+-------+-----+------+----+----+---------
+--------+-------------+------------+-------+--------+-------+------------+---
--------+------+---------+--------+-------------+------------+-------+--------
+-------+------------+-----------+------+
```

```
display(df.select("City").where(col("City").isNotNull()).groupBy("City").count(
).orderBy("count", ascending=False))
```

	City	count
1	Not in a city	138411
2	New York	46887
3	Los Angeles	42241
4	Phoenix	37912
5	El Paso	36908
6	North Little Rock	35332
7	Houston	33626

Showing all 144 rows.

```
display(df.select("State").where(col("State").isNotNull()).groupBy('State').cou
nt().orderBy("count", ascending=False))
```

	State	count
1	California	576142
2	Pennsylvania	188892
3	Texas	123208
4	New York	70487
5	Arizona	69840
6	Illinois	50116
7	North Carolina	37126

Showing all 47 rows.

```
print((df.count(), len(df.columns)))
```

```
(1746661, 29)
```

Filtering Unwanted Locations

```
df = df.filter((df.City != 'Not in a city') & (df.State != 'Country Of
Mexico'))
```

```
df.agg({"Date": "max"}).collect()[0]
```

```
Out[12]: Row(max(Date)=datetime.date(2016, 5, 31))
```

Null value replacement with Median

```
from pyspark.sql.functions import when

imputeCols = [
  "SO2 AQI",
  "CO AQI",
]
```

```
from pyspark.ml.feature import Imputer

imputer = Imputer(strategy="mean", inputCols=imputeCols, outputCols=imputeCols)

df = imputer.fit(df).transform(df)
```

```
display(df.select([count(when(isnull(c), c)).alias(c) for c in df.columns]))
```

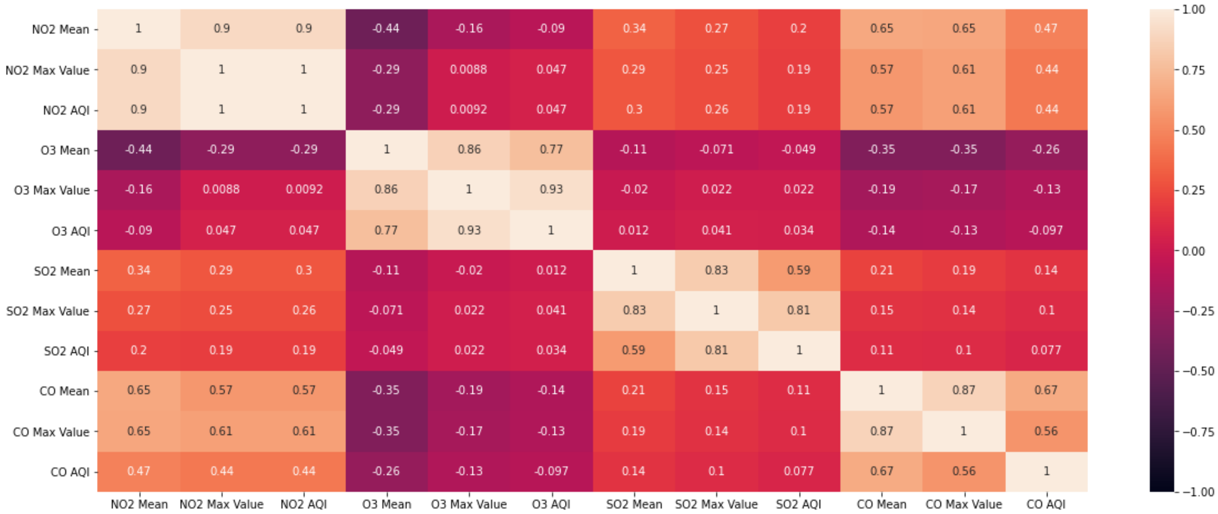
	Index	State Code	County Code	SiteNum	Address	State
1	0	0	0	0	0	0

Showing all 1 rows.

Data Exploration

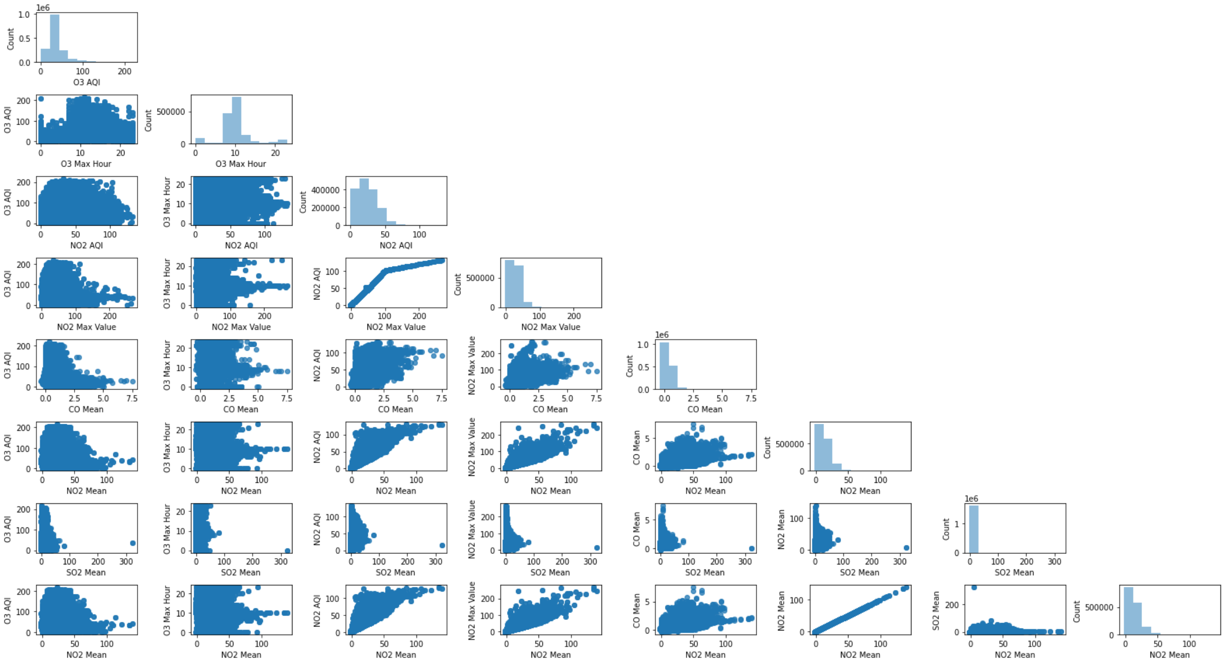
Correlation Matrix

```
# Correlation matrix displaying correlation between features in the dataset

import matplotlib.pyplot as plt
import seaborn as sns

# Source: https://sparkbyexamples.com/pyspark/convert-pyspark-dataframe-to-pandas/
pandasDF = df.toPandas()
finalDF = pandasDF[['NO2 Mean', 'NO2 Max Value', 'NO2 AQI', 'O3 Mean', 'O3 Max Value', 'O3 AQI', 'SO2 Mean', 'SO2 Max Value',
                    'SO2 AQI', 'CO Mean', 'CO Max Value', 'CO AQI']]
# Remove Index, State Code, County code, sitenum, max hour cols

# Source: https://medium.com/@szabo.bibor/how-to-create-a-seaborn-correlation-heatmap-in-python-834c0686b88e
plt.figure(figsize=(20, 8))
heatmap = sns.heatmap(finalDF.corr(), vmin=-1, vmax=1, annot=True)
```



Scatterplot Matrix

```
# Scatterplot matrix showing relationships between data features

cols = ['O3 AQI', 'O3 Max Hour', 'NO2 AQI', 'NO2 Max Value', 'CO Mean', 'NO2
Mean', 'SO2 Mean', 'NO2 Mean']
from mlxtend.plotting import scatterplotmatrix
scatterplotmatrix(pandasDF[cols].values, figsize= (22,12), names=cols,
alpha=0.5)
plt.tight_layout()
plt.show()
```

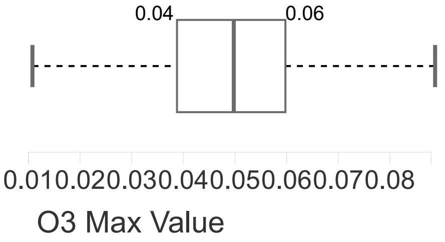


Geospacial and Temporal Analysis

```
from pyspark.sql.functions import *

# Register the DataFrame as table 'static_counts'
df.createOrReplaceTempView("pollution")
```

Distribution of Daily Max O3 Concentration (ppm)

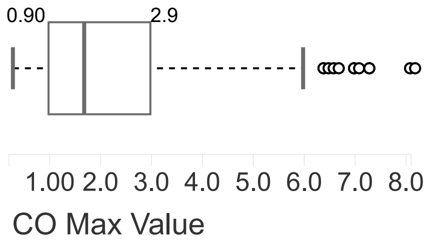
```
%sql select `O3 Max Value` from pollution
```



Showing sample based on the first 1000 rows.

Distribution of Daily Max CO Concentration (ppm)

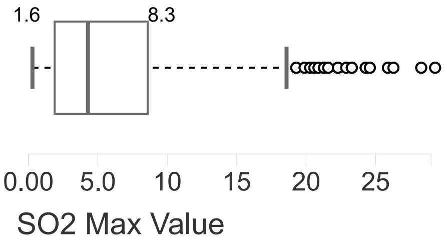
```
%sql select `CO Max Value` from pollution
```



Showing sample based on the first 1000 rows.

Distribution of Daily Max SO2 Concentration (ppm)

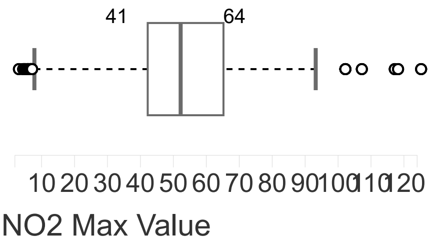
```
%sql select `SO2 Max Value` from pollution
```



Showing sample based on the first 1000 rows.

Distribution of Daily Max NO2 Concentration (ppm)

```
%sql select `NO2 Max Value` from pollution
```

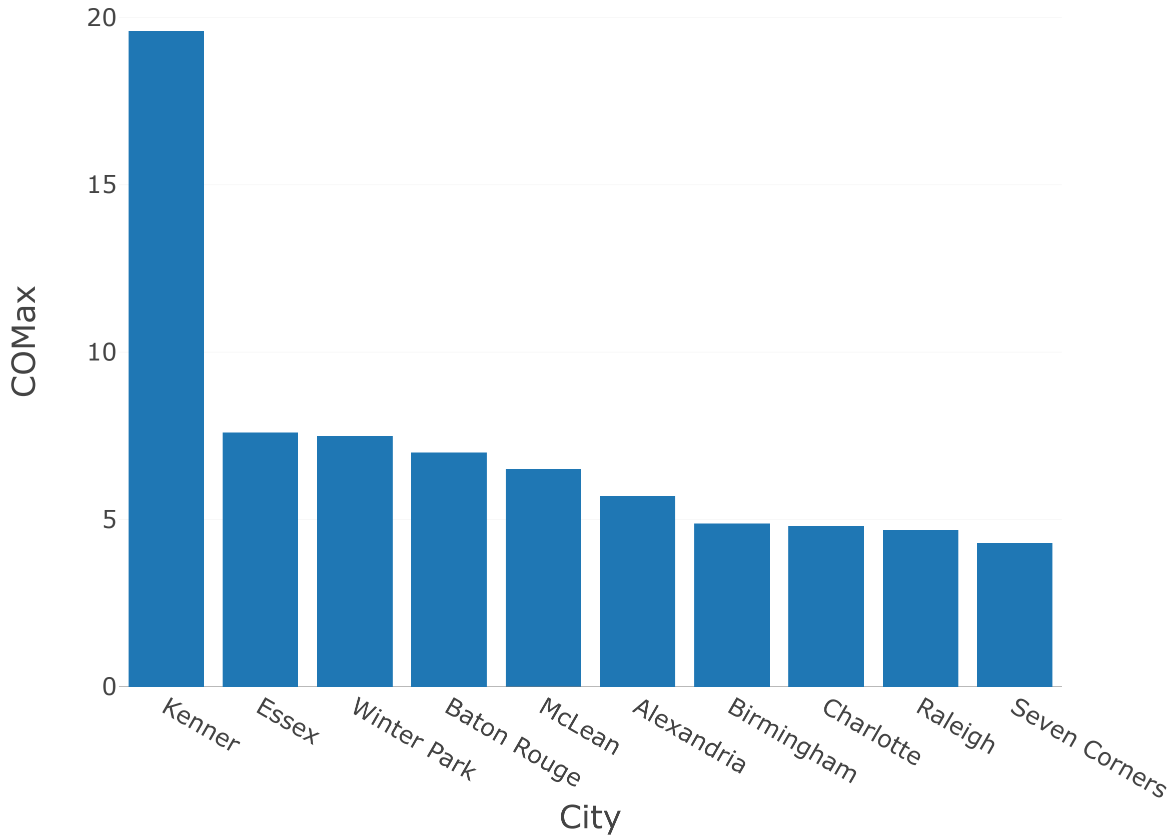


Showing sample based on the first 1000 rows.

Geospacial Analysis of Maximum Daily 8-hour CO Concentration Grouped by US Region and City

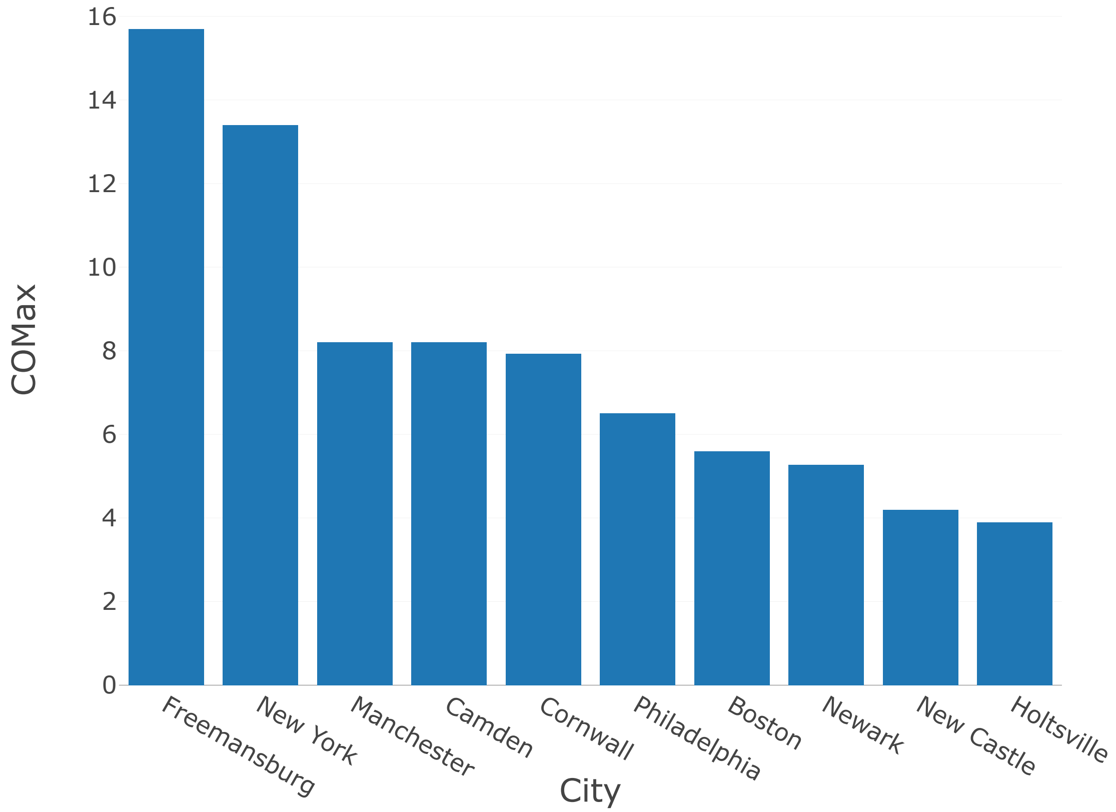
Southeast Cities with Highest CO 1st Max Valu

```
%sql select City, max(`CO Max Value`) as COMax from pollution where State IN
('Maryland', 'Virginia','North Carolina', 'South Carolina', 'Georgia',
'Alabama', 'Mississippi', 'Louisiana', 'Florida', 'Tennessee', 'Kentucky',
'West Virginia', 'Delaware')
group by City ORDER BY COMax Desc limit 10
```



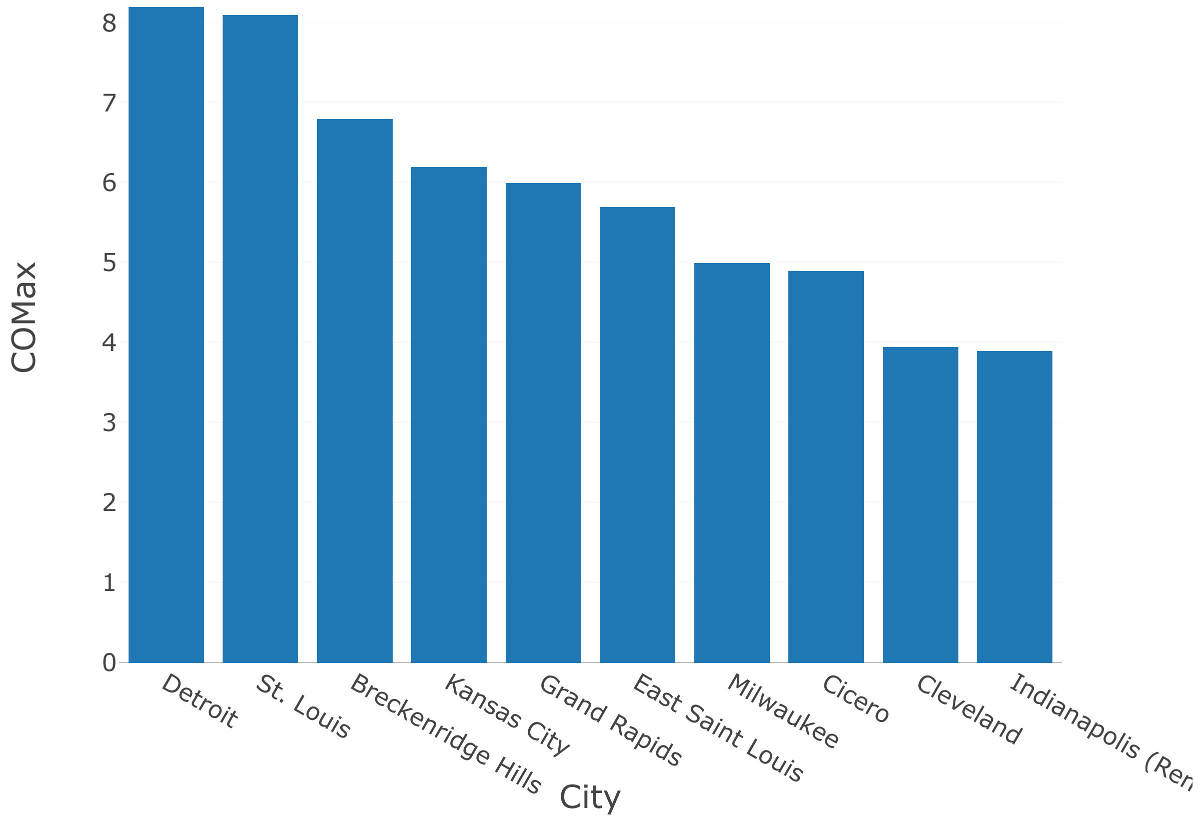
Northeast Cities with Highest CO 1st Max Valu

```
%sql select City, max(`CO Max Value`) as COMax from pollution where State IN
('New York', 'Pennsylvania','Connecticut', 'Rhode Island', 'Maine', 'New
Hampshire', 'Massachusetts', 'New Jersey', 'Vermont')
group by City ORDER BY COMax Desc limit 10
```



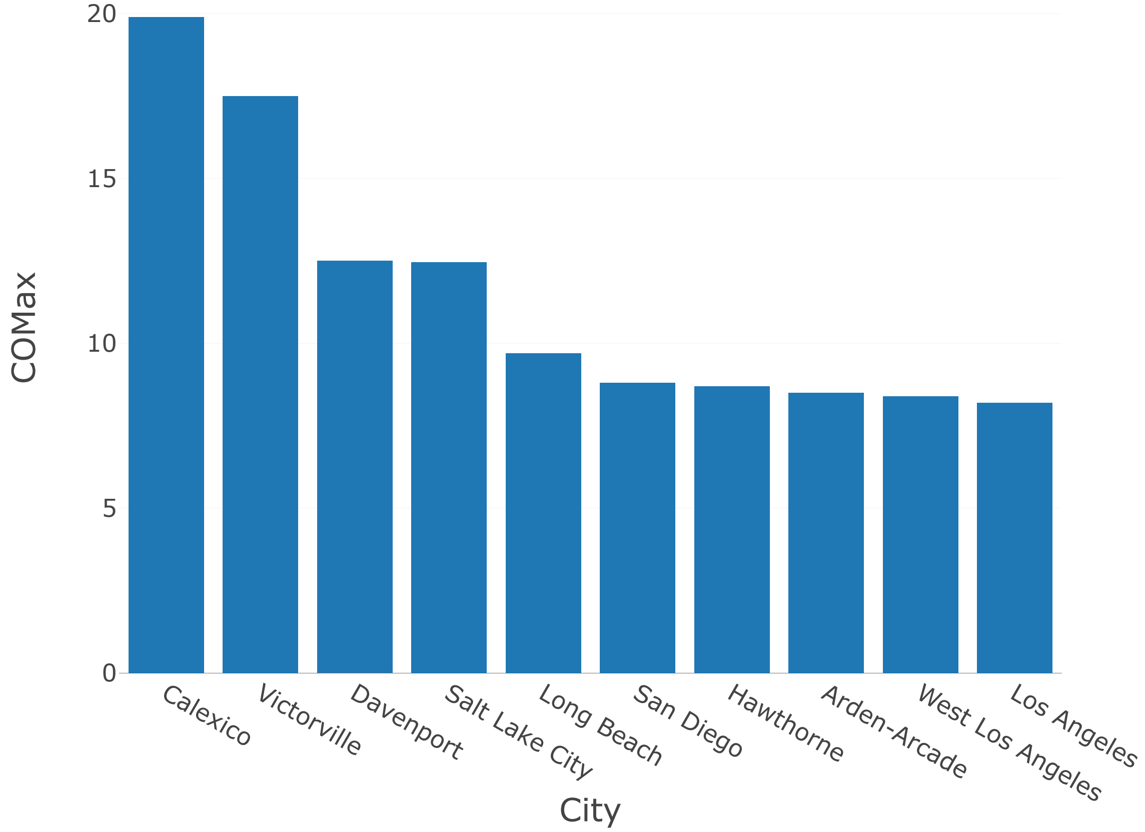
Midwest Cities with Highest CO 1st Max Value

```
%sql select City, max(`CO Max Value`) as COMax from pollution where State IN
('Minnesota', 'North Dakota','South Dakota', 'Ohio', 'Nebraska', 'Kansas',
'Iowa', 'Missouri', 'Illinois', 'Indiana', 'Michigan', 'Wisconsin')
group by City ORDER BY COMax Desc limit 10
```



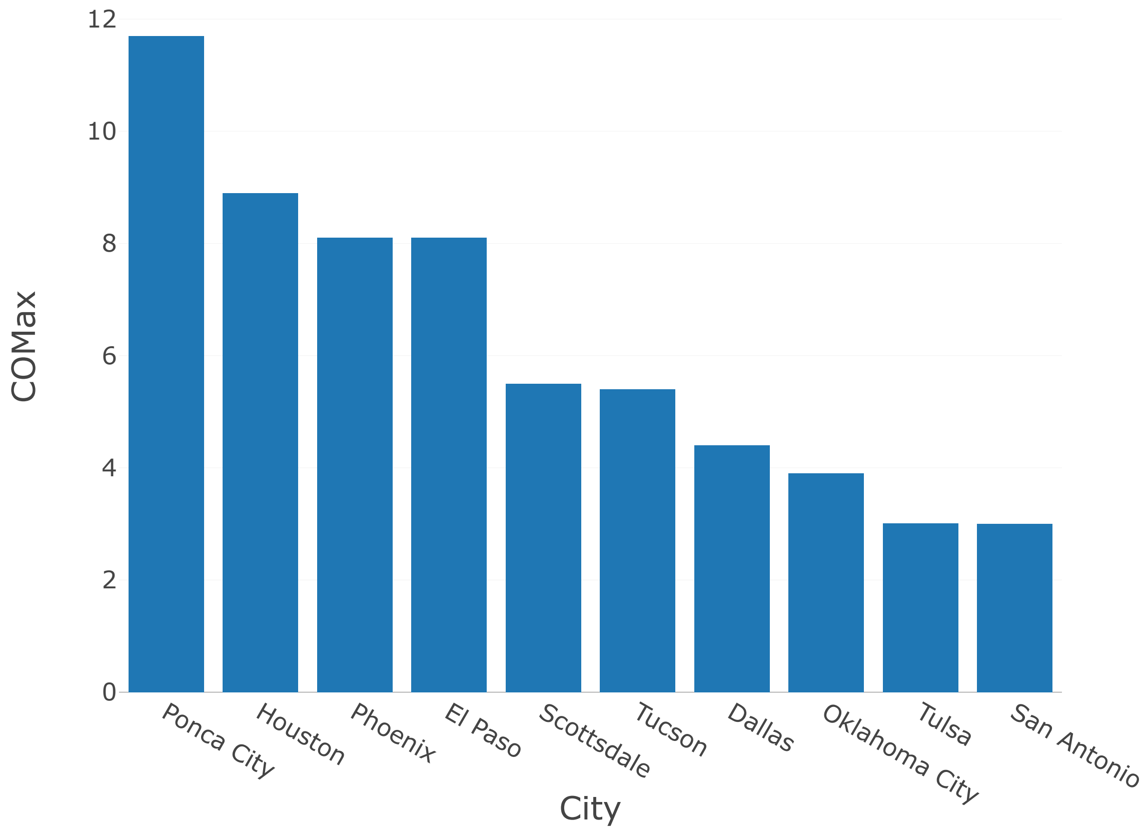
West Cities with Highest CO 1st Max Value (pp

```
%sql select City, max(`CO Max Value`) as COMax from pollution where State IN
('California', 'Oregon','Washington', 'Wyoming', 'Idaho', 'Colorado', 'Utah',
'Nevada', 'Montana', 'Alaska', 'Hawaii')
group by City ORDER BY COMax Desc limit 10
```



Southwest Cities with Highest CO 1st Max Valu

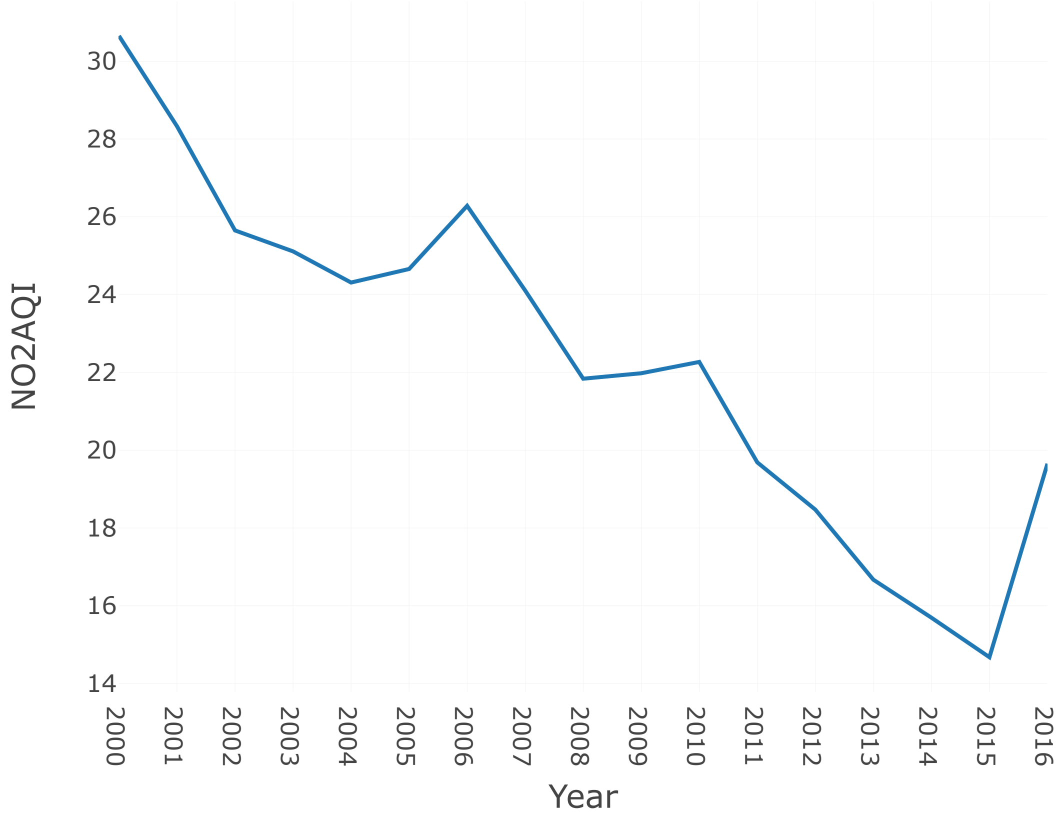
```
%sql select City, max(`CO Max Value`) as COMax from pollution where State IN
('Arizona', 'New Mexico','Texas', 'Oklahoma')
group by City ORDER BY COMax Desc limit 10
```



Temporal Analysis of NO2 AQI Yearly Trends Grouped by US Region

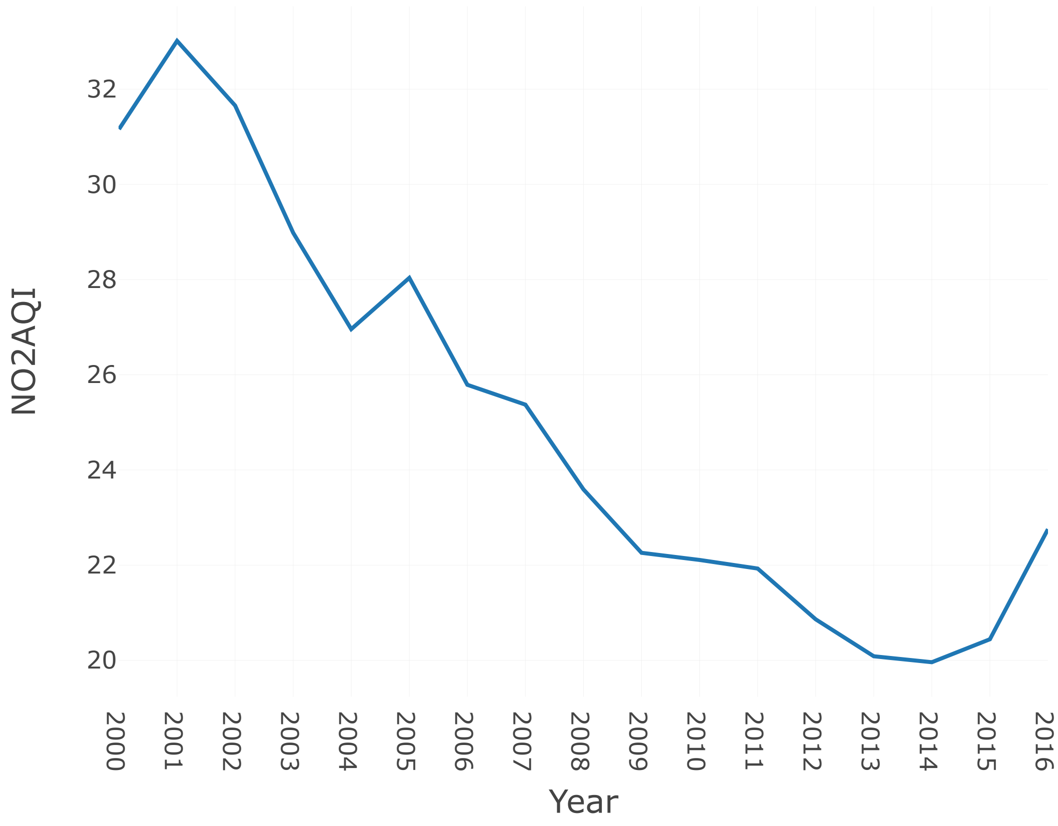
NO2 AQI Yearly Trends - Southeast Region

```
%sql select Year(`Date`) as Year, avg(`NO2 AQI`) as NO2AQI from pollution where
State IN
('Maryland', 'Virginia','North Carolina', 'South Carolina', 'Georgia',
'Alabama', 'Mississippi', 'Louisiana', 'Florida', 'Tennessee', 'Kentucky',
'West Virginia', 'Delaware')
group by Year ORDER BY Year Asc
```



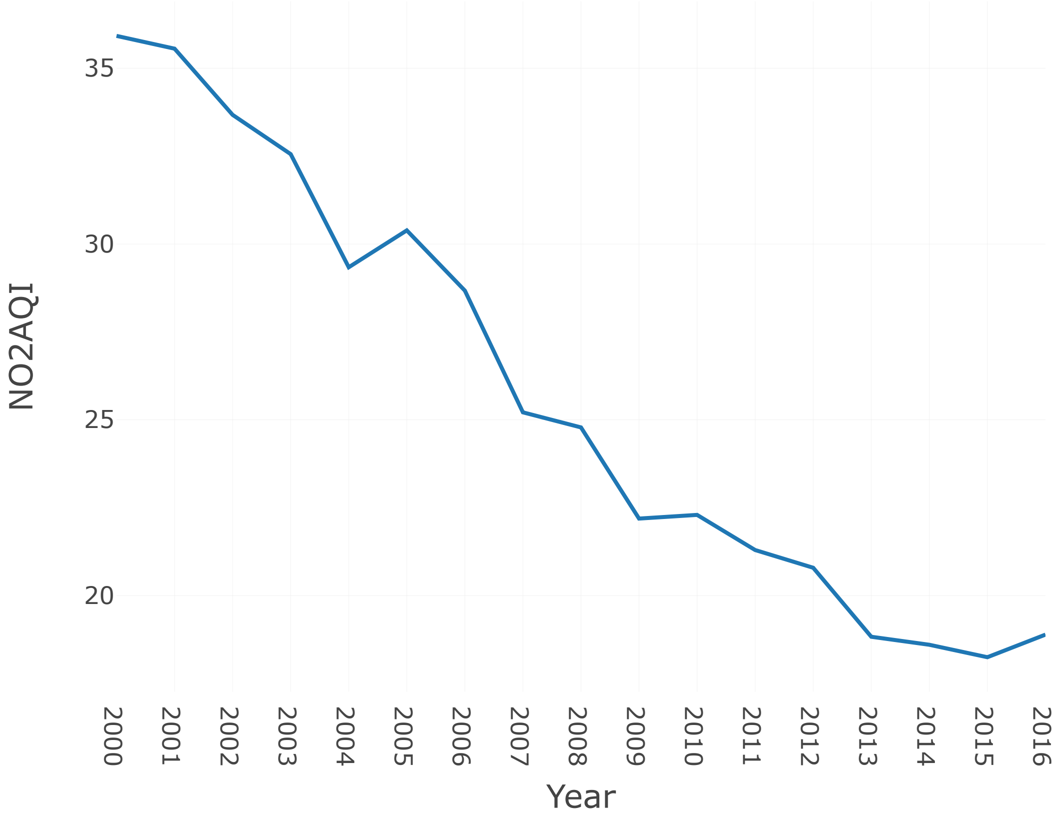
NO2 AQI Yearly Trends - Northeast Region

```
%sql select Year(`Date`) as Year, avg(`NO2 AQI`) as NO2AQI from pollution where
State IN
('New York', 'Pennsylvania','Connecticut', 'Rhode Island', 'Maine', 'New
Hampshire', 'Massachusetts', 'New Jersey', 'Vermont')
group by Year ORDER BY Year Asc
```



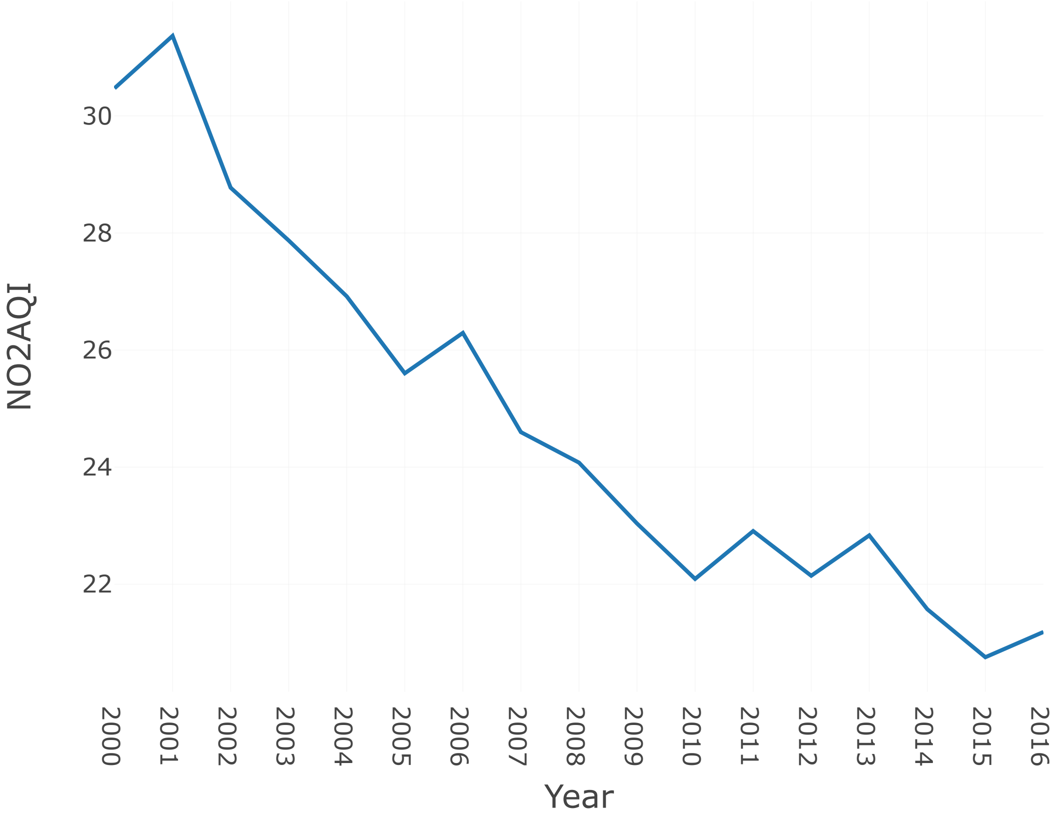
NO2 AQI Yearly Trends - Midwest Region

```
%sql select Year(`Date`) as Year, avg(`NO2 AQI`) as NO2AQI from pollution where
State IN
('Minnesota', 'North Dakota','South Dakota', 'Ohio', 'Nebraska', 'Kansas',
'Iowa', 'Missouri', 'Illinois', 'Indiana', 'Michigan', 'Wisconsin')
group by Year ORDER BY Year Asc
```



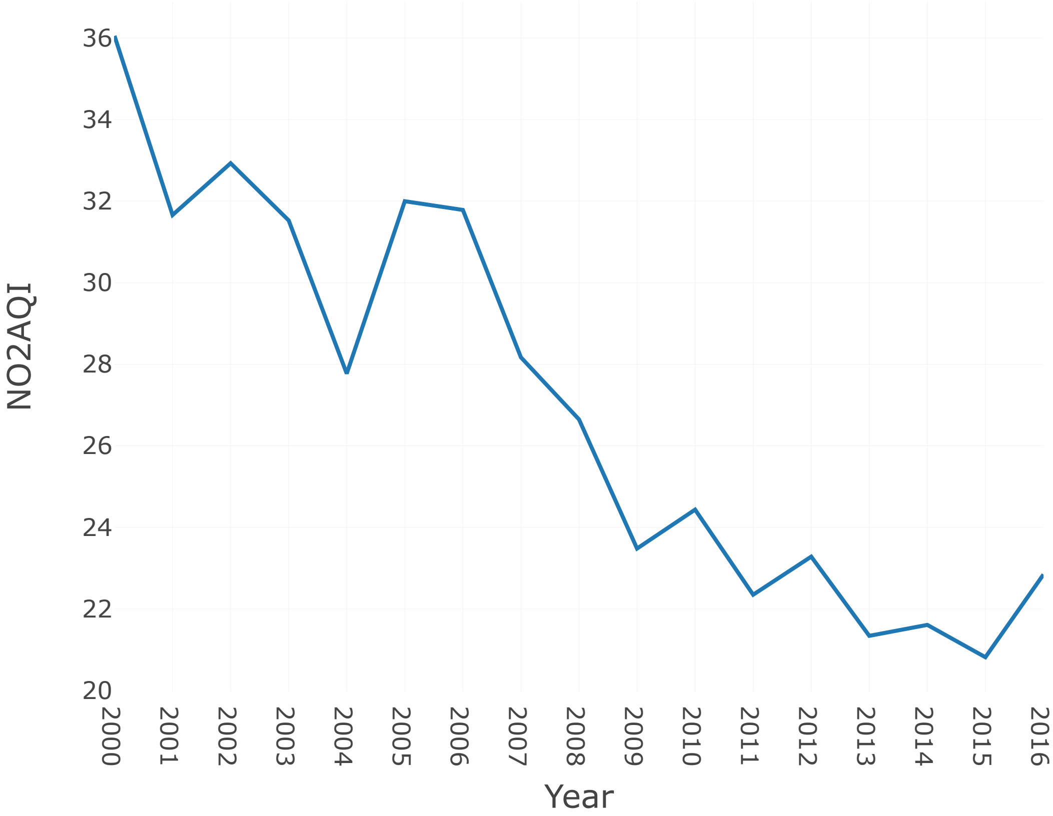
NO2 AQI Yearly Trends - West Region

```
%sql select Year(`Date`) as Year, avg(`NO2 AQI`) as NO2AQI from pollution where
State IN
('California', 'Oregon','Washington', 'Wyoming', 'Idaho', 'Colorado', 'Utah',
'Nevada', 'Montana', 'Alaska', 'Hawaii')
group by Year ORDER BY Year Asc
```



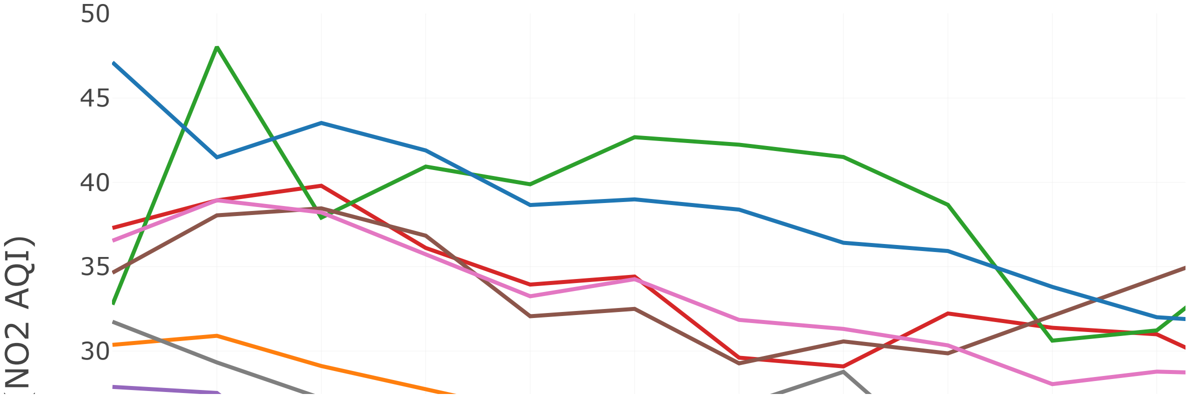
NO2 AQI Yearly Trends - Southwest Region

```
%sql select Year(`Date`) as Year, avg(`NO2 AQI`) as NO2AQI from pollution where
State IN
('Arizona', 'New Mexico','Texas', 'Oklahoma')
group by Year ORDER BY Year Asc
```



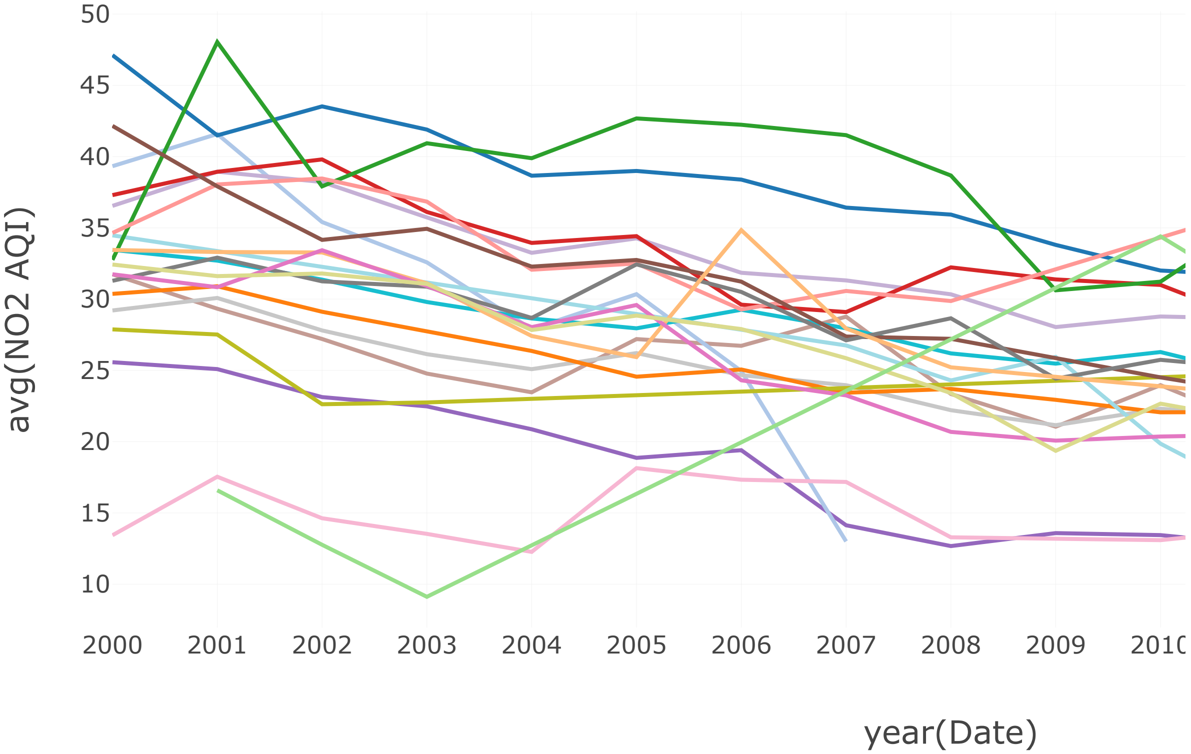
Yearly Average of NO2 for Some Selected Stat

```
display(df.select('State',year('Date'),'NO2 AQI').where((col("State")==
'District Of Columbia') | (col("State")== 'New York') | (col("State")== 'New
York') | (col("State")== 'New Jersey') | (col("State")== 'California') |
(col("State")== 'Arizona') | (col("State")== 'Oklahoma') | (col("State")==
'Colorado') | (col("State")== 'North Carolina') | (col("State")==
'Kentucky')).groupby('State','year(Date)').agg({'NO2
AQI':'avg'}).orderBy('year(Date)',ascending=True))
```



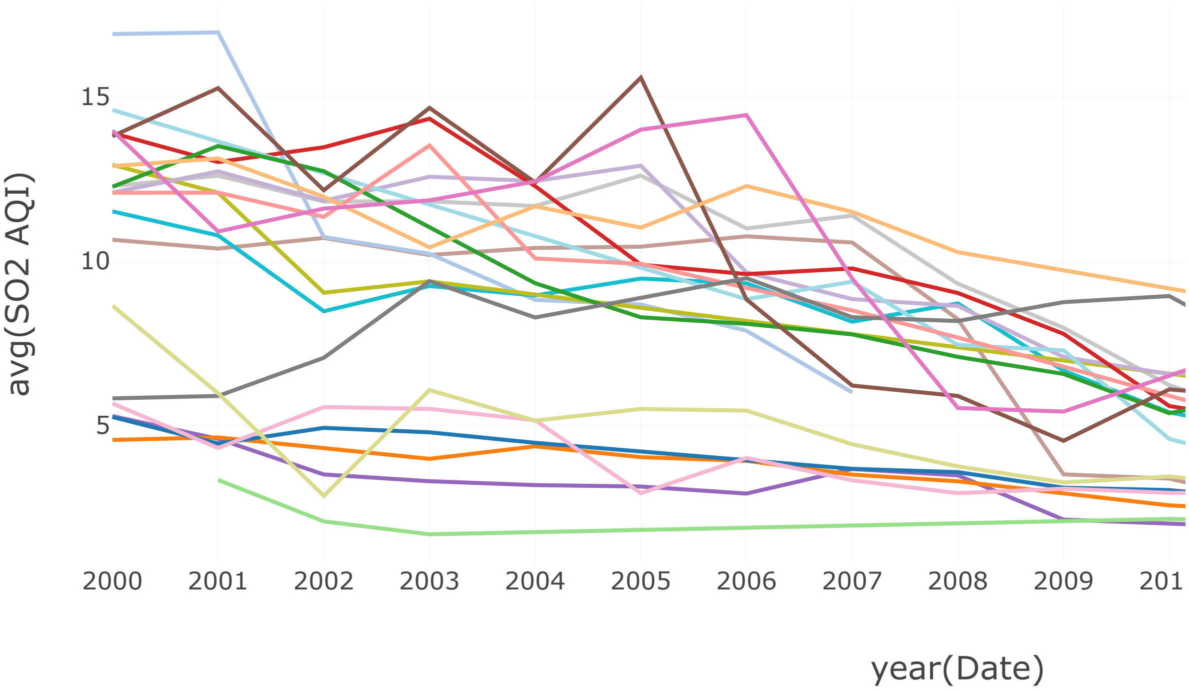
Yearly Average of NO2 for All States

```
display(df.select('State',year('Date'),'NO2
AQI').groupby('State','year(Date)').agg({'NO2
AQI':'avg'}).orderBy('year(Date)',ascending=True))
```



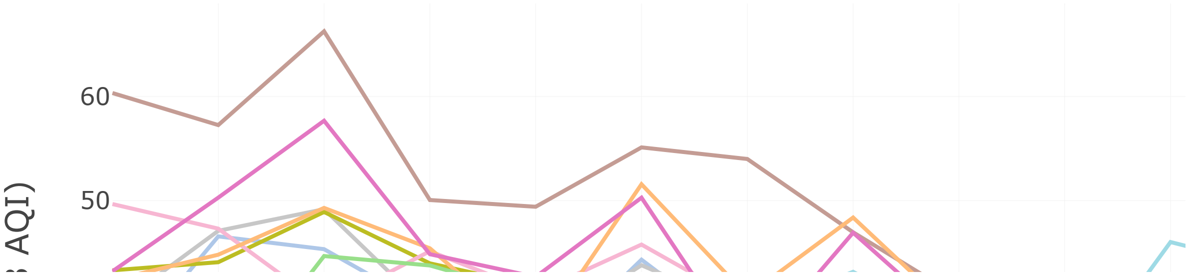
Yearly Average of SO2 for All States

```
display(df.select('State',year('Date'),'SO2
AQI').groupby('State','year(Date)').agg({'SO2
AQI':'avg'}).orderBy('year(Date)',ascending=True))
```



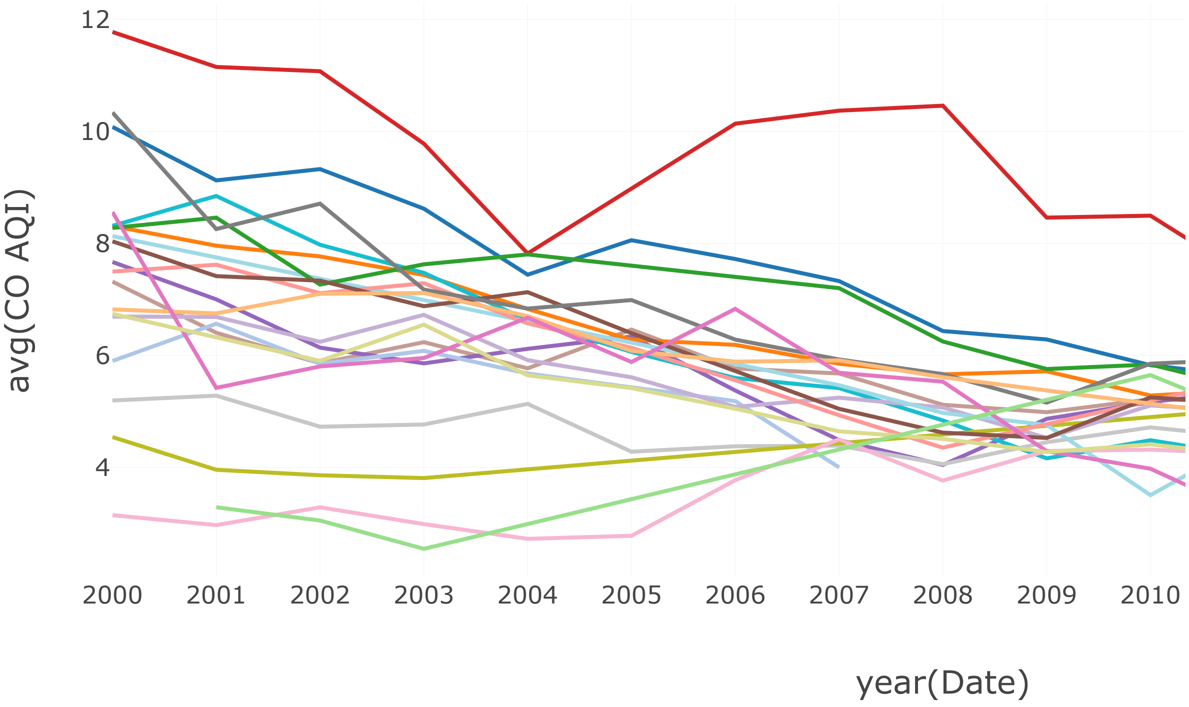
Yearly Average of O3 for All States

```
display(df.select('State',year('Date'),'O3
AQI').groupby('State','year(Date)').agg({'O3
AQI':'avg'}).orderBy('year(Date)',ascending=True))
```



Yearly Average of CO for All States

```
display(df.select('State',year('Date'),'CO
AQI').groupby('State','year(Date)').agg({'CO
AQI':'avg'}).orderBy('year(Date)',ascending=True))
```



TOP 10 biggest polluters by mean value of AQI for Nitrogen Dioxide (NO2), Ozone (O3), Sulfur Dioxide (SO2) and Carbon Monoxide (CO) for 2000-2016

```
import databricks.koalas as ks
df2 = df.to_koalas()
```

```
df_AQI = df2[['State', 'Date', 'NO2 AQI', 'O3 AQI', 'SO2 AQI', 'CO AQI']]
df_AQI_State = df_AQI.groupby('State').mean()
df_AQI_State.reset_index(inplace=True)
```

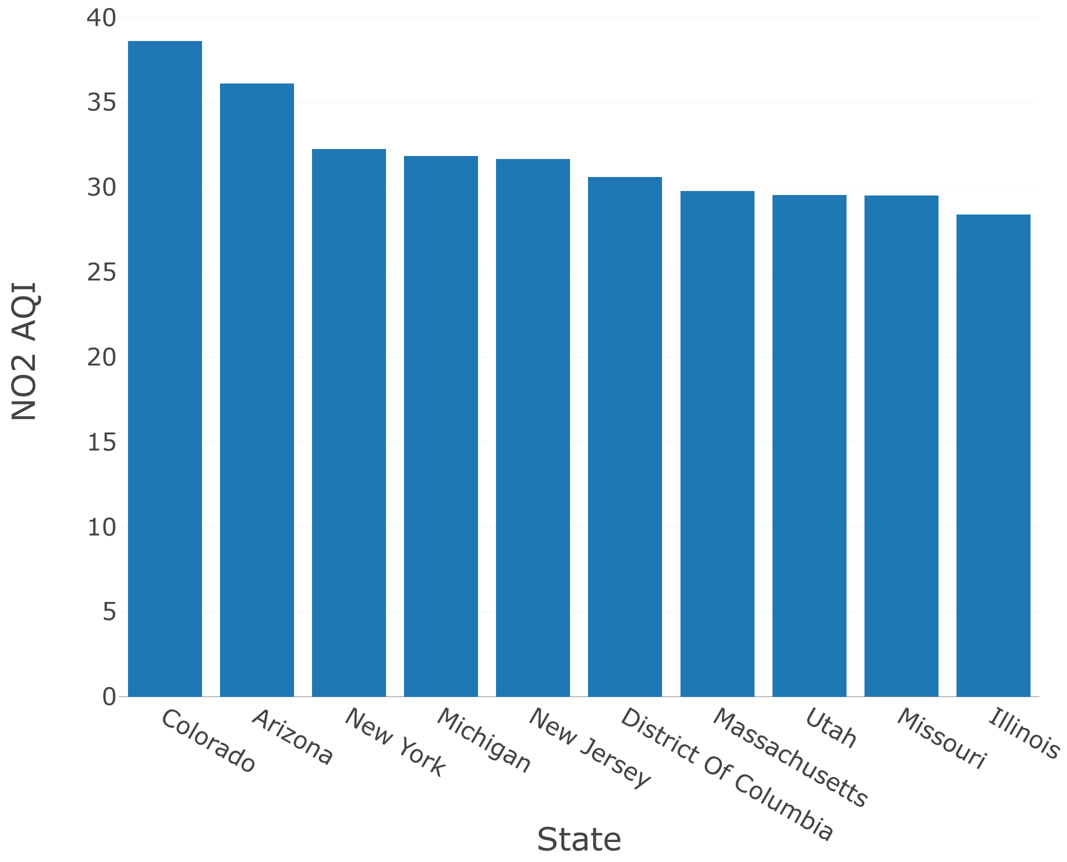
```
display(df_AQI_State)
```

	State	NO2 AQI	O3 AQI	SO2 AQI
1	District Of Columbia	30.602895392278953	33.781288916562886	7.94898038605230
2	Texas	23.868391188251	35.935864485981305	3.80315420560747
3	Pennsylvania	24.301904021103184	39.74236531502587	9.63930693180921
4	Nevada	23.94679315322747	40.50958960610435	2.05506289956692
5	Illinois	28.38151488546572	31.992417591188442	8.06806209593742
6	Oklahoma	14.902614758861127	41.02330040674027	3.36955258570598
7	Missouri	29.49598138747885	41.90524534686971	10.8387267343485

Showing all 41 rows.

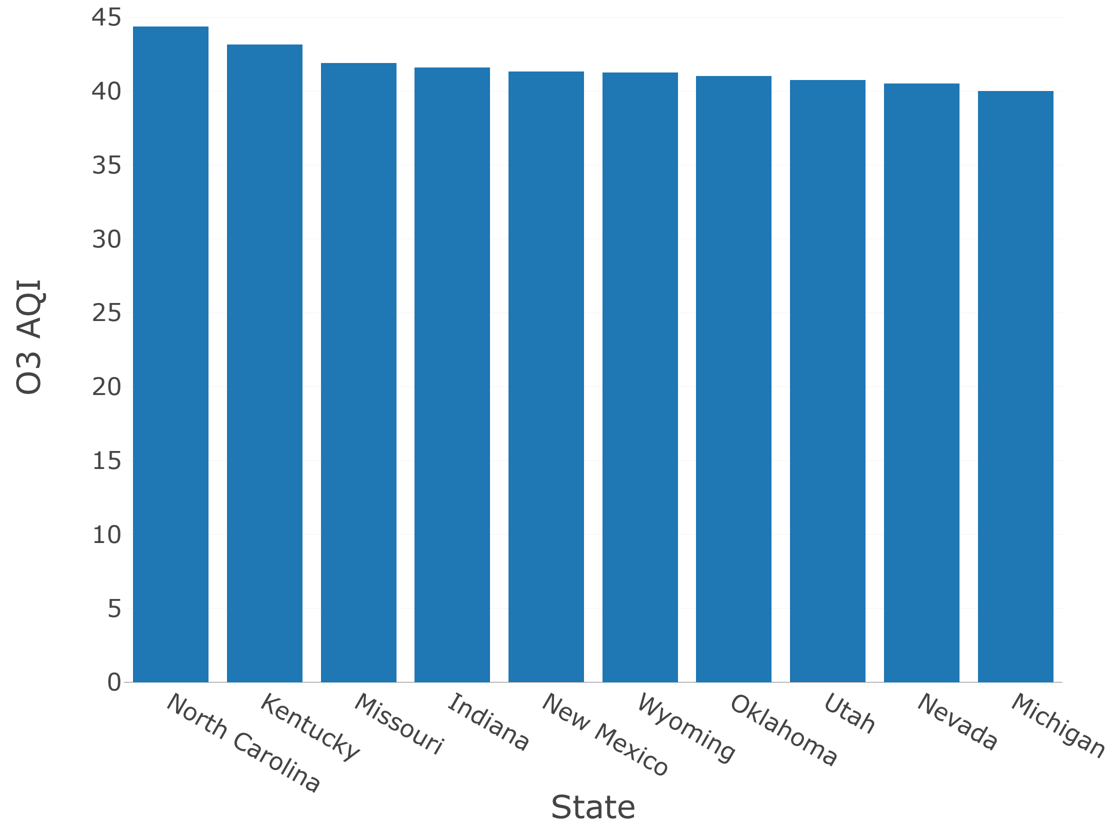
TOP 10 highest polluted states by mean value

```
df_AQI_State.sort_values(by = 'NO2 AQI', ascending = False, inplace = True)
sdf = df_AQI_State.to_spark()
display(sdf.select("State", "NO2 AQI").limit(10))
```



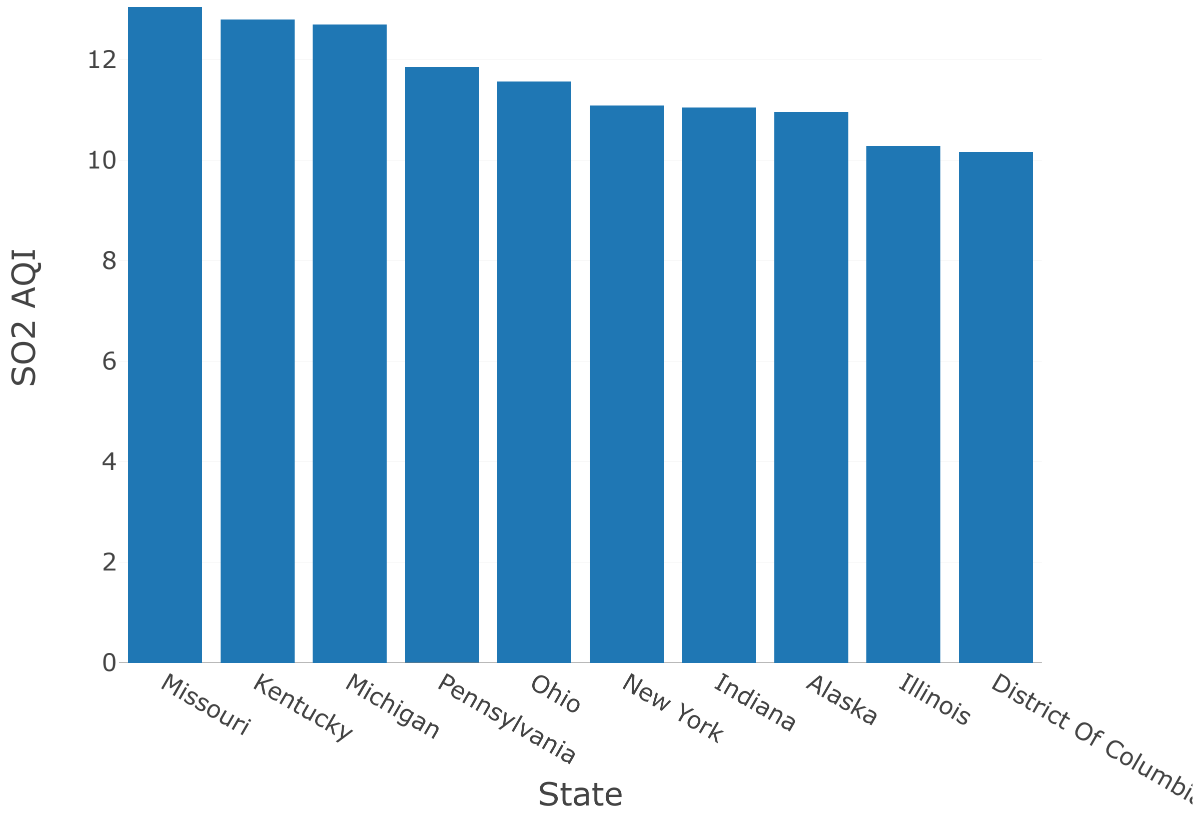
TOP 10 highest polluted states by mean value

```
df_AQI_State.sort_values(by = 'O3 AQI', ascending = False, inplace = True)
sdf = df_AQI_State.to_spark()
display(sdf.select("State", "O3 AQI").limit(10))
```



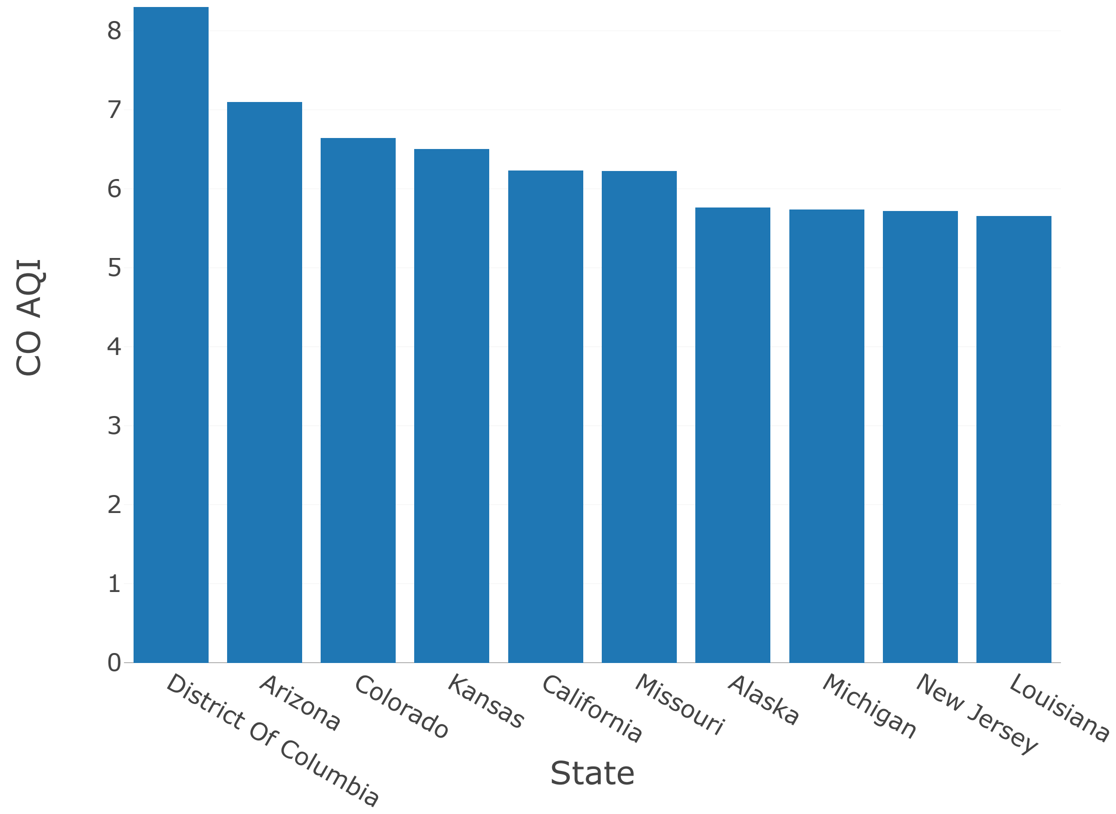
TOP 10 highest polluted states by mean value

```
df_AQI_State.sort_values(by = 'SO2 AQI', ascending = False, inplace = True)
sdf = df_AQI_State.to_spark()
display(sdf.select("State", "SO2 AQI").limit(10))
```



TOP 10 highest polluted states by mean value

```
df_AQI_State.sort_values(by = 'CO AQI', ascending = False, inplace = True)
sdf = df_AQI_State.to_spark()
display(sdf.select("State", "CO AQI").limit(10))
```



Calculating AQIs Grouped by State and Date

```
## Prepare all 4 AQIs against state and date
pollSt = df2[['State','Date','NO2 AQI','O3 AQI','SO2 AQI','CO AQI']]
pollSt['Date Local'] = ks.to_datetime(pollSt['Date'],format='%Y-%m-%d')  #
Change date from string to date value
pollSt = pollSt.groupby(['State','Date']).mean()  # Take mean values if there
are depulicated entries
```

```
/databricks/spark/python/pyspark/sql/pandas/functions.py:386: UserWarning: In
Python 3.6+ and Spark 3.0+, it is preferred to specify type hints for pandas U
DF instead of specifying pandas UDF type which will be deprecated in the futur
e releases. See SPARK-28264 for more details.
  warnings.warn(
```

```
pollSt.info
```

```
Out[67]: <bound method DataFrame.info of                                     N
O2 AQI      O3 AQI    SO2 AQI     CO AQI
State                Date
```

```
Arizona              2000-05-28   22.500000   87.000000   4.000000   6.750000
                     2000-06-29   27.500000   52.500000   3.250000   5.750000
                     2000-09-13   62.666667   39.333333   5.666667  10.166667
                     2000-11-08   44.666667   23.000000   5.333333  11.000000
                     2000-11-22   47.500000   23.500000   5.500000   7.000000
Colorado             2000-01-26   13.000000    3.000000  13.000000   9.000000
                     2000-07-26   53.000000   42.000000  13.500000   7.000000
                     2000-08-02   44.000000   42.000000  10.000000   6.000000
District Of Columbia 2000-11-05   32.000000   25.000000  11.000000  11.500000
Florida              2000-02-19   20.000000   30.000000   4.500000   7.500000
                     2000-05-29    8.000000   40.000000   1.500000   6.500000
Illinois             2000-05-19   29.000000   16.666667   8.500000   6.833333
Indiana              2000-04-01   47.000000   30.000000  20.000000  19.000000
                     2000-06-21   16.000000   39.000000  21.500000   7.500000
Kansas               2000-03-02   33.000000   23.000000  12.000000   7.500000
                     2000-04-19   18.000000   36.000000   1.500000   7.500000
Louisiana            2000-09-25   23.000000   13.000000  14.500000   7.500000
```

```
pollSt.tail(5)
```

		NO2 AQI	O3 AQI	SO2 AQI	CO AQI
State	**Date**				
Pennsylvania	**2015-07-20**	14.0	45.50	4.0	3.25
Rhode Island	**2015-01-06**	25.0	27.00	2.0	4.00
	2015-05-15	18.0	61.00	1.5	4.00
Iowa	**2016-04-20**	9.5	34.75	1.5	3.75
Missouri	**2016-01-28**	14.0	30.00	3.0	4.00

```
pollSt = pollSt.rename(columns={'NO2 AQI': 'NO2_AQI', 'O3 AQI': 'O3_AQI', 'SO2
AQI': 'SO2_AQI', 'CO AQI': 'CO_AQI' })
pollSt.head(5)
```

Creating Correlation Matrix

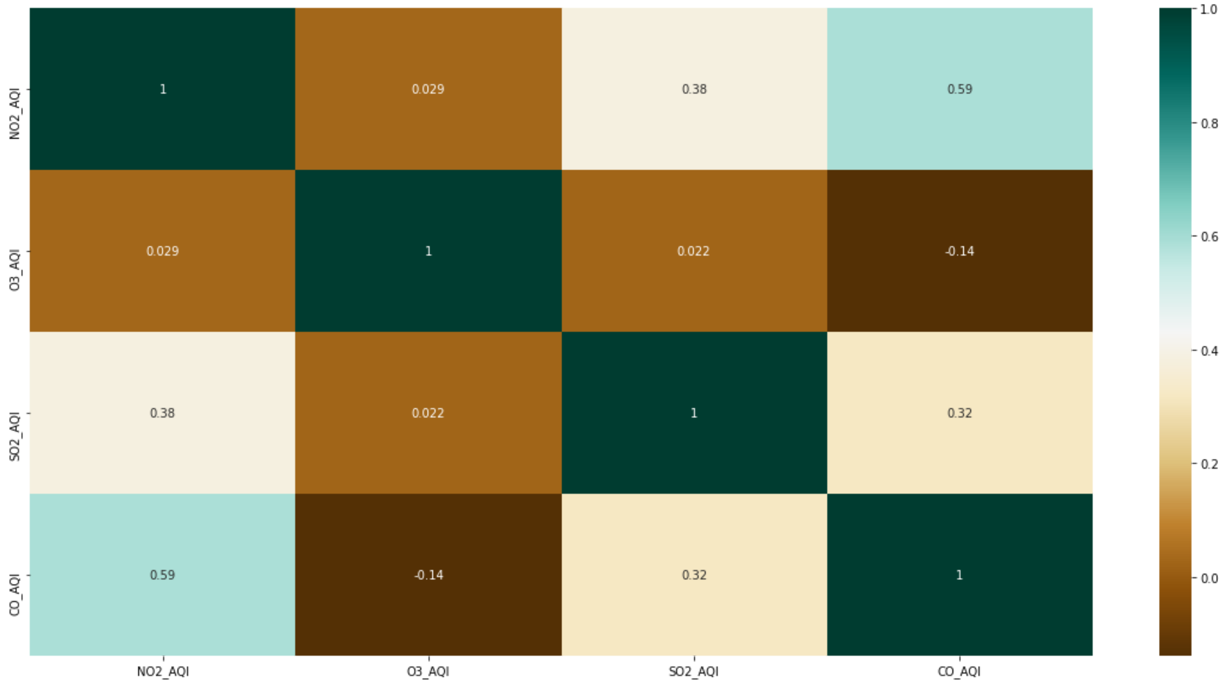
```
pollSt = pollSt.to_pandas()
pollSt.corr('pearson')
```

	NO2_AQI	O3_AQI	SO2_AQI	CO_AQI
NO2_AQI	1.000000	0.028681	0.382474	0.589804
O3_AQI	0.028681	1.000000	0.021733	-0.138445
SO2_AQI	0.382474	0.021733	1.000000	0.318885
CO_AQI	0.589804	-0.138445	0.318885	1.000000

AQI Heatmap

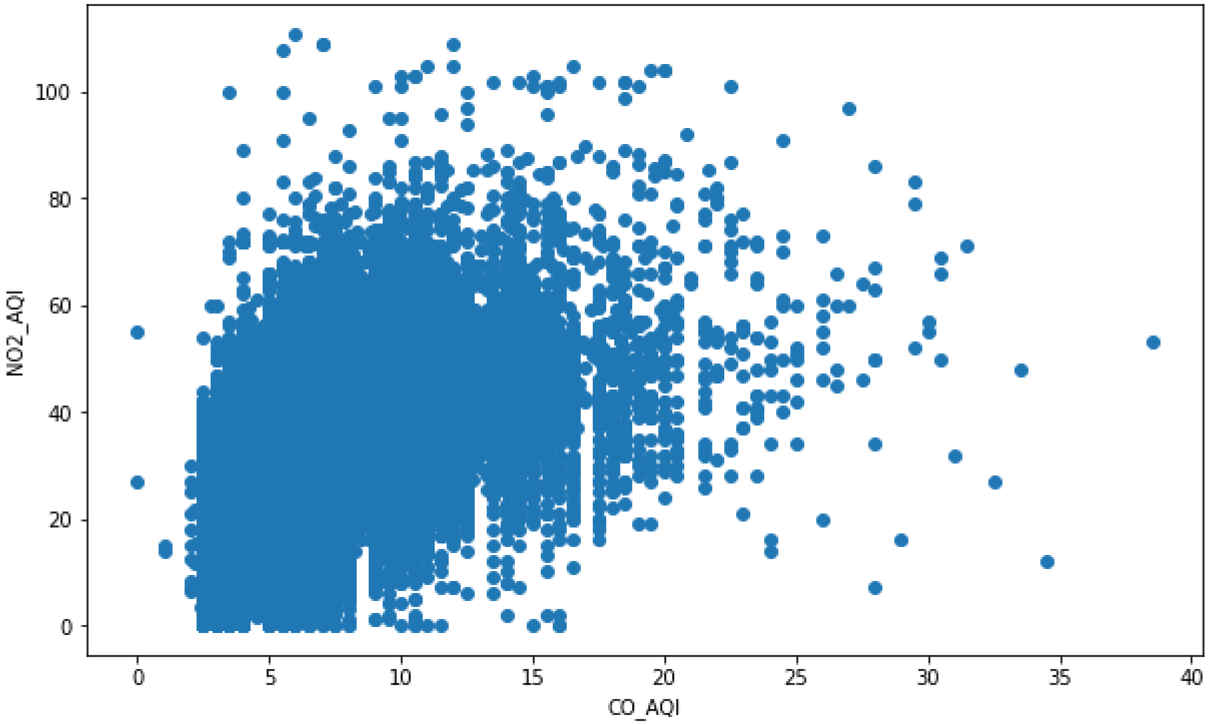
```
import matplotlib.pyplot as plt
import seaborn as sns
import statistics as stat
plt.figure(figsize=(20,10))
c= pollSt.corr()
sns.heatmap(c,cmap='BrBG',annot=True)
c
```

	NO2_AQI	O3_AQI	SO2_AQI	CO_AQI
NO2_AQI	1.000000	0.028681	0.382474	0.589804
O3_AQI	0.028681	1.000000	0.021733	-0.138445
SO2_AQI	0.382474	0.021733	1.000000	0.318885
CO_AQI	0.589804	-0.138445	0.318885	1.000000



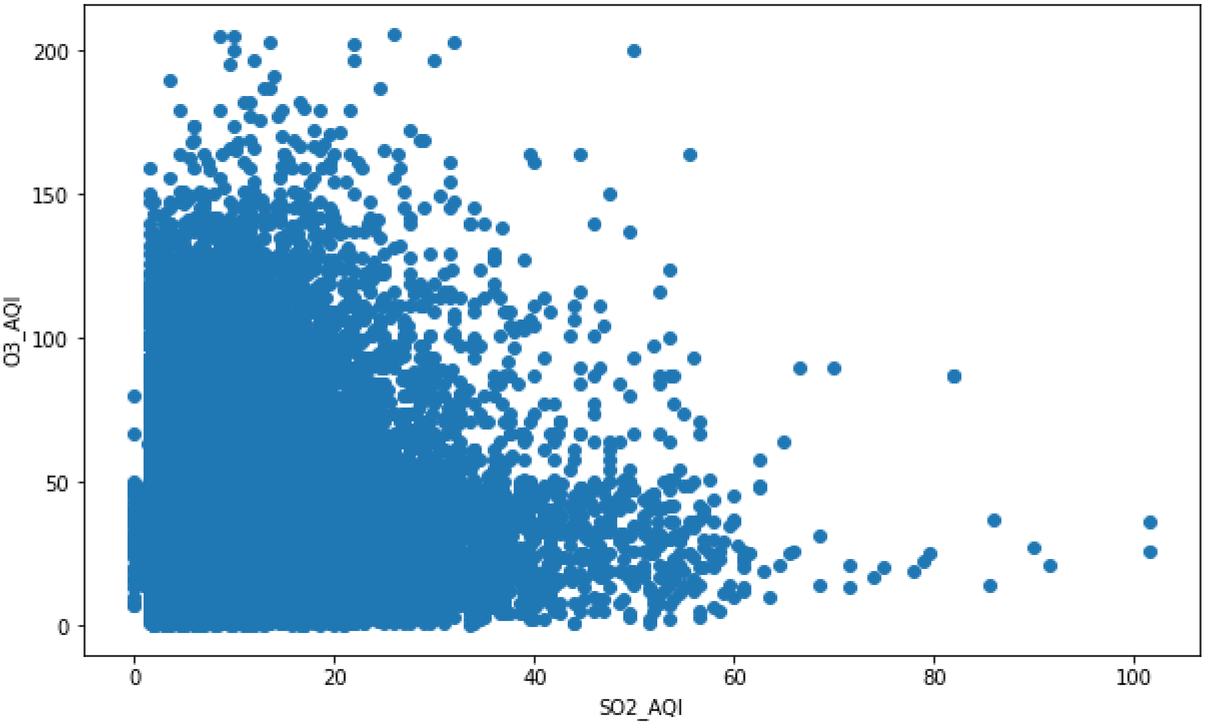
CO AQI and NO2 AQI Scatterplot

```
# Plotting a scatter plot
fig, ax = plt.subplots(figsize=(10,6))
ax.scatter(pollSt['CO_AQI'], pollSt['NO2_AQI'])
ax.set_xlabel('CO_AQI')
ax.set_ylabel('NO2_AQI')
plt.show()
```



SO2 AQI and O3 AQI Scatterplot

```
# Plotting a scatter plot
fig, ax = plt.subplots(figsize=(10,6))
ax.scatter(pollSt['SO2_AQI'], pollSt['O3_AQI'])
ax.set_xlabel('SO2_AQI')
ax.set_ylabel('O3_AQI')
plt.show()
```



Timeseries Analysis

```
from fbprophet import Prophet
```

Analysis of San Diego Pollution using Time Se

```
data1 = (df.select("City","Date", "NO2 AQI").where(col("City")== 'San
Diego').dropDuplicates(['Date'])).withColumnRenamed('Date',
'ds').withColumnRenamed('NO2 AQI', 'y')
data2 = (df.select("City","Date", "O3 AQI").where(col("City")== 'San
Diego').dropDuplicates(['Date'])).withColumnRenamed('Date',
'ds').withColumnRenamed('O3 AQI', 'y')
data3 = (df.select("City","Date", "SO2 AQI").where(col("City")== 'San
Diego').dropDuplicates(['Date'])).withColumnRenamed('Date',
'ds').withColumnRenamed('SO2 AQI', 'y')
data4 = (df.select("City","Date", "CO AQI").where(col("City")== 'San
Diego').dropDuplicates(['Date'])).withColumnRenamed('Date',
'ds').withColumnRenamed('CO AQI', 'y')
```

Last Date Before Prediction

```
data1.agg({"ds": "max"}).collect()[0]
```

```
Out[41]: Row(max(ds)=datetime.date(2011, 6, 30))
```

```
data1.explain()
```

```
== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- Project [City#18971, Date#7820 AS ds#18639, NO2 AQI#18973 AS y#18643]
   +- SortAggregate(key=[Date#7820], functions=[finalmerge_first(merge first#1
8976, valueSet#18977) AS first(City#7819)()#18970, finalmerge_first(merge firs
t#18980, valueSet#18981) AS first(NO2 AQI#7825)()#18972])
      +- Sort [Date#7820 ASC NULLS FIRST], false, 0
         +- Exchange hashpartitioning(Date#7820, 200), ENSURE_REQUIREMENTS, [i
d=#16746]
            +- SortAggregate(key=[Date#7820], functions=[partial_first(City#78
19, false) AS (first#18976, valueSet#18977), partial_first(NO2 AQI#7825, fals
e) AS (first#18980, valueSet#18981)])
               +- Sort [Date#7820 ASC NULLS FIRST], false, 0
                  +- Filter (isnotnull(City#7819) AND (City#7819 = San Diego))
                     +- InMemoryTableScan [City#7819, Date#7820, NO2 AQI#782
5], [isnotnull(City#7819), (City#7819 = San Diego)]
                           +- InMemoryRelation [Index#7812, State Code#7813, C
ounty Code#7814, SiteNum#7815, Address#7816, State#7817, County#7818, City#781
9, Date#7820, NO2 Units#7821, NO2 Mean#7822, NO2 Max Value#7823, NO2 Max Hour#
7824, NO2 AQI#7825, O3 Units#7826, O3 Mean#7827, O3 Max Value#7828, O3 Max Hou
r#7829, O3 AQI#7830, SO2 Units#7831, SO2 Mean#7832, SO2 Max Value#7833, SO2 Ma
```

```
from pyspark.sql.types import *
final_schema = StructType([StructField('ds', DateType(), True),
                           StructField('City', StringType(), True),
                           StructField('y', DoubleType(), True),
                           StructField('yhat', DoubleType(), True)])
```

```
from pyspark.sql.functions import pandas_udf, PandasUDFType

@pandas_udf(final_schema, PandasUDFType.GROUPED_MAP)
def pollution_forcast(Pre_data):
  model =
Prophet(growth='linear',seasonality_mode='multiplicative',daily_seasonality=Tru
e,weekly_seasonality=True,yearly_seasonality=True)
  model.fit(Pre_data)
  future_pd = model.make_future_dataframe(periods = 370, freq = 'd')
  forcast_pd = model.predict(future_pd)
  f_pd = forcast_pd[['ds','yhat']].set_index('ds')
  s_pd = Pre_data[['ds','City','y']].set_index('ds')

  final_df = f_pd.join(s_pd, how = 'left')
  final_df.reset_index(level=0, inplace = True)
  final_df['City'] = Pre_data['City'].iloc[0]
  return final_df[ ['ds','City','y','yhat'] ]
```

```
result =
(((data1.groupBy('City').apply(pollution_forcast)).withColumnRenamed("y","NO2")
.withColumnRenamed("yhat","NO2_Pred")).join(((data2.groupBy('City').apply(pollu
tion_forcast)).withColumnRenamed("y","O3").withColumnRenamed("yhat","O3_Pred").
drop('City')),'ds').join(((data3.groupBy('City').apply(pollution_forcast)).with
ColumnRenamed("y","SO2").withColumnRenamed("yhat","So2_Pred").drop('City')),'ds
').join(((data3.groupBy('City').apply(pollution_forcast)).withColumnRenamed("y"
,"CO").withColumnRenamed("yhat","CO_Pred").drop('City')),'ds'))
```

```
/databricks/spark/python/pyspark/sql/pandas/group_ops.py:81: UserWarning: It i
s preferred to use 'applyInPandas' over this API. This API will be deprecated
in the future releases. See SPARK-28264 for more details.
  warnings.warn(
```

```
result.cache()
```

```
Out[46]: DataFrame[ds: date, City: string, NO2: double, NO2_Pred: double, O3:
double, O3_Pred: double, SO2: double, So2_Pred: double, CO: double, CO_Pred: d
ouble]
```

```
result.count()
```

```
Out[47]: 4511
```

Last Date After Prediction

```
result.agg({"ds": "max"}).collect()[0]
```

```
Out[48]: Row(max(ds)=datetime.date(2012, 7, 4))
```

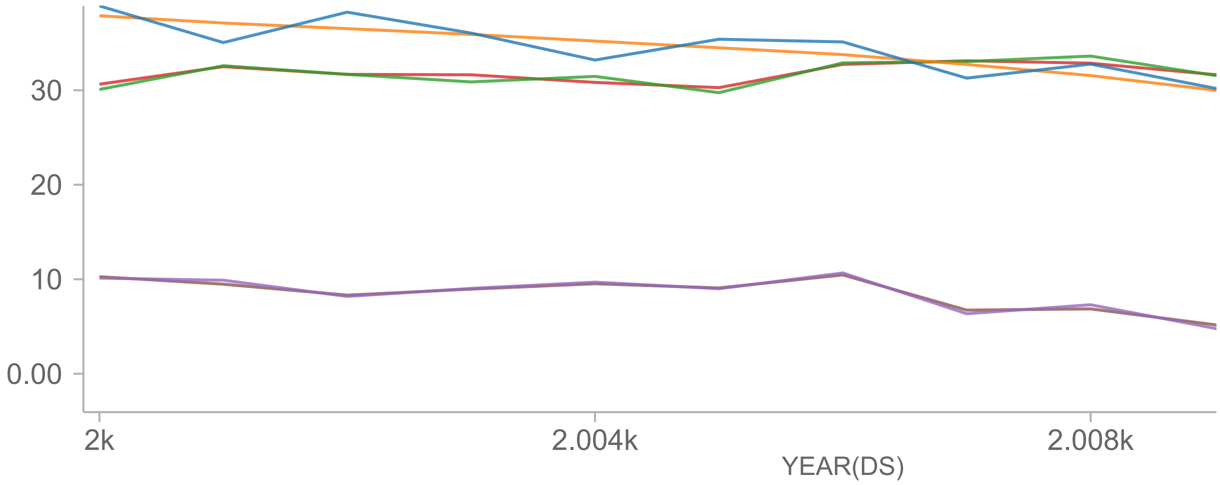
```
display(result.orderBy('ds',ascending=True).tail(400))
```

	ds	City	NO2	NO2_Pred	O3	O
1	2011-06-01	San Diego	28	17.592869112693467	38	37
2	2011-06-02	San Diego	29	17.717877675720853	36	37
3	2011-06-03	San Diego	30	17.745392564704733	36	36
4	2011-06-04	San Diego	17	14.329857280642704	37	39
5	2011-06-05	San Diego	25	12.892818187953315	34	40
6	2011-06-06	San Diego	17	17.878738043921086	33	35
7	2011-06-07	San Diego	24	18.303752768349383	36	34

Showing all 400 rows.

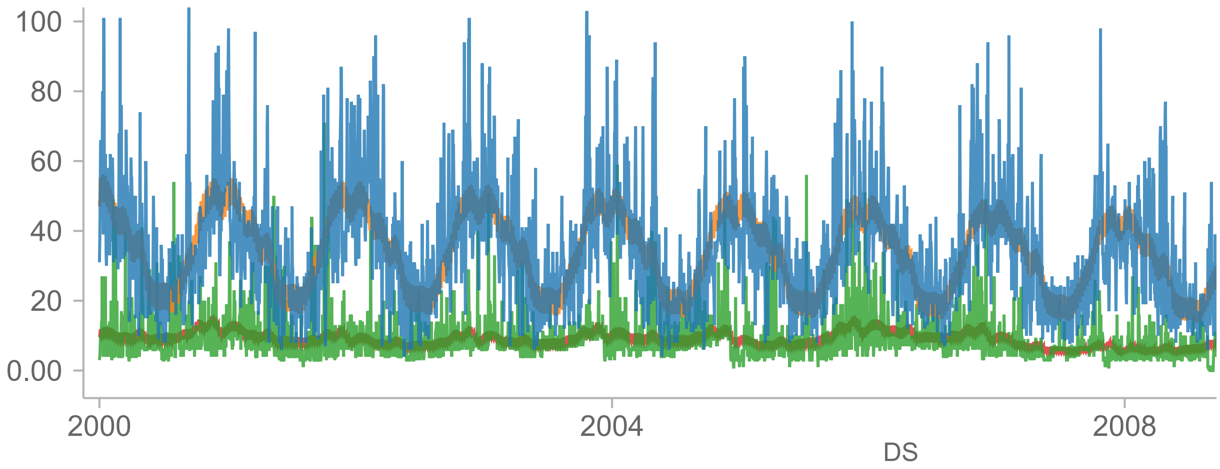
Yearly Average of all type of pollution with pred

```
display(result.select('City',year('ds'),'NO2','NO2_Pred','O3','O3_Pred','SO2','
So2_Pred','CO','CO_Pred').orderBy('year(ds)', ascending=True))
```



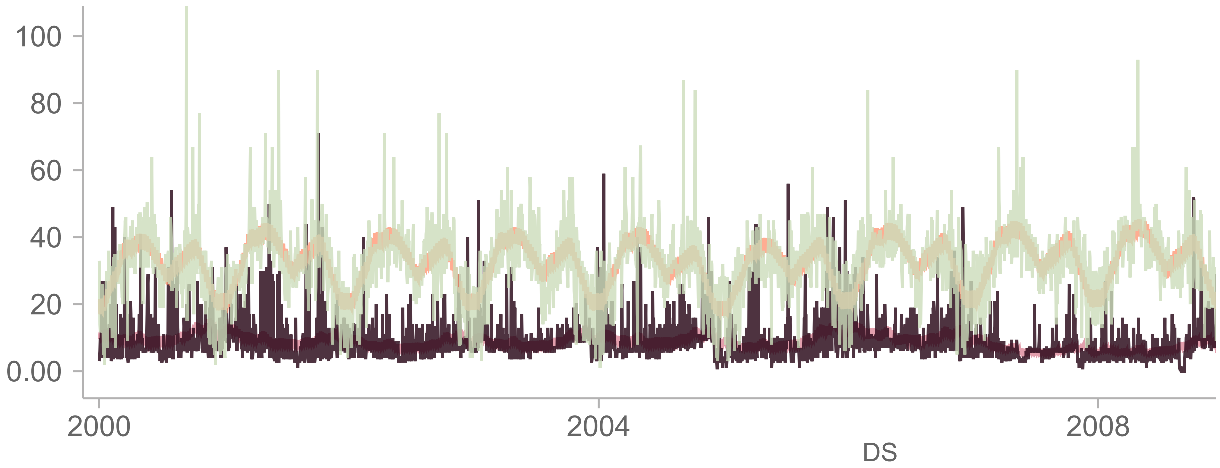
Real and predicted values of NO2 and SO2 ov

```
display(result.select('City','ds','NO2','NO2_Pred','SO2','So2_Pred').orderBy('d
s',ascending=True))
```



Real and predicted values of O3 and CO over

```
display(result.select('City','ds','O3','O3_Pred','CO','CO_Pred').orderBy('ds',a
scending=True))
```



```
new = (result.dropna())
```

```
new.count()
```

```
Out[17]: 4141
```

Accuracy on NO2

```
import pyspark.sql.functions as psf
from pyspark.ml.evaluation import RegressionEvaluator
def Accuracy(expected_col, actual_col):
  regressionEvaluator = RegressionEvaluator(predictionCol=expected_col,
labelCol=actual_col)
  r2 = (regressionEvaluator.setMetricName("r2").evaluate(new))
  rmse= new.withColumn('error', psf.pow(psf.col(actual_col)-
psf.col(expected_col),psf.lit(2))).groupBy('City').agg(psf.avg(psf.col('error')
).alias('mse')).withColumn('rmse',psf.sqrt(psf.col('mse'))).withColumn('R2
Value',lit(r2))
  return(rmse)
display(Accuracy("NO2_Pred","NO2"))
```

	City	mse	rmse	R2 Value
1	San Diego	148.79434845839467	12.198128891694605	0.43200843404262523

Showing all 1 rows.

Accuracy on O3

```
display(Accuracy("O3_Pred","O3"))
```

	City	mse	rmse	R2 Value
1	San Diego	54.40534742562648	7.375998063016725	0.45241005369046694

Showing all 1 rows.

Accuracy on So2

```
display(Accuracy("So2_Pred","SO2"))
```

	City	mse	rmse	R2 Value
1	San Diego	30.49080362643493	5.521847845281046	0.24385284328927936

Showing all 1 rows.

Accuracy on CO

	City	mse	rmse	R2 Value
1	San Diego	30.49080362643493	5.521847845281046	0.24385284328927936

Showing all 1 rows.