**Naan Mudhalvan Project**

**Air Quality Analysis in Tamil Nadu**

**Phase 5**

**Team Members:**

~Samiukktha Dharmalingam - 2021115089

~Santhya - 2021115093

~ Saravanasudharsan S - 2021115095

~Sathyadharini S - 2021115097

~ Vairaarasu - 2021115301

Team Members:

~Samiukktha Dharmalingam - 2021115089

~Santhya - 2021115093

~ Saravanasudharsan S - 2021115095

~Sathyadharini S - 2021115097

~ Vairaarasu - 2021115301

Team Members:

~Samiukktha Dharmalingam - 2021115089

~Santhya - 2021115093

~ Saravanasudharsan S - 2021115095

~Sathyadharini S - 2021115097

~ Vairaarasu - 2021115301

Team Members:

~Samiukktha Dharmalingam - 2021115089

~Santhya - 2021115093

~ Saravanasudharsan S - 2021115095

~Sathyadharini S - 2021115097

~ Vairaarasu - 2021115301

Team Members:

~Samiukktha Dharmalingam - 2021115089

~Santhya - 2021115093

~ Saravanasudharsan S - 2021115095

~Sathyadharini S - 2021115097

~ Vairaarasu - 2021115301

Team Members:

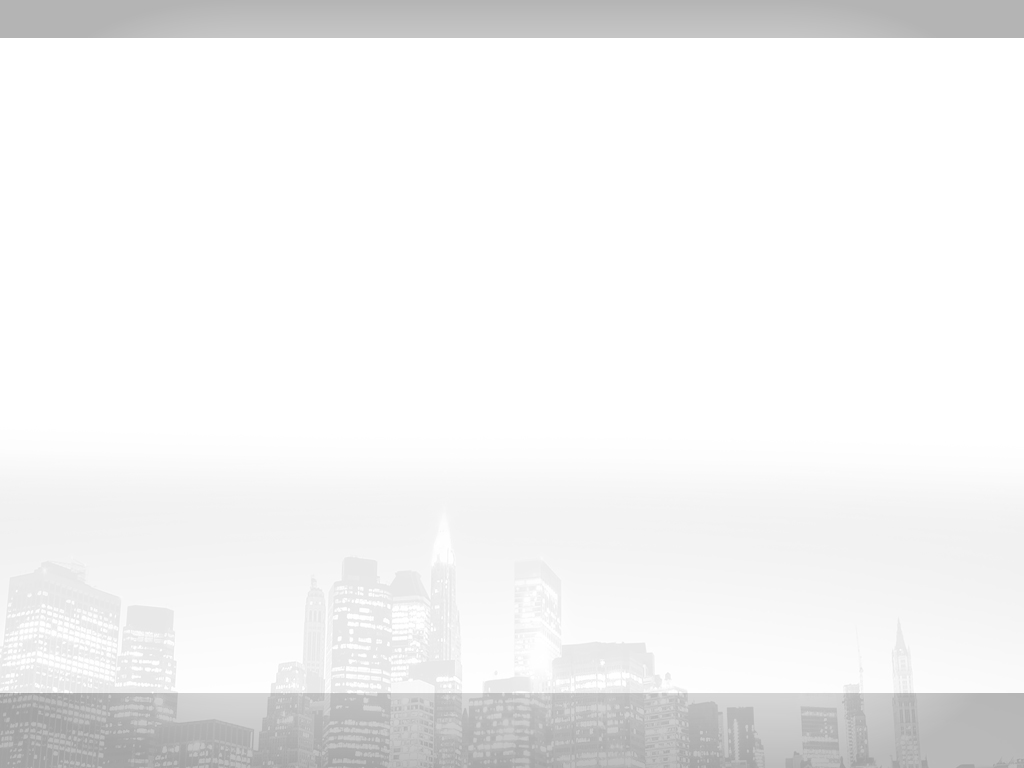
~Samiukktha Dharmalingam - 2021115089

~Santhya - 2021115093

~ Saravanasudharsan S - 2021115095

~Sathyadharini S - 2021115097

~ Vairaarasu - 2021115301



**Phase 5: Project Documentation & Submission**

**Overview:**

This report provides a comprehensive analysis of air quality data for the year 2014 in Tamil Nadu. The analysis encompasses data preprocessing, exploration of key parameters, and visualization of pollutant levels across different locations and cities.

**Indian AQI:**

The Indian AQI range differs from that of US-EPA. To calculate AQI, a minimum of three parameters should be taken out of which one must be either PM10 or PM2.5.

According to the Indian Government (CPCB), Indian AQI range is from 0-500, from 0 being good and 500 being severe. There are eight major pollutants to be taken into account for AQI calculation, viz. particulate matter (PM 10 and PM 2.5), carbon monoxide (CO), ozone (O3), nitrogen dioxide (NO2), sulfur dioxide (SO2), ammonia (NH3), and lead (Pb). To calculate AQI, data for a minimum of three pollutants must be present, of which one should be either PM10 or PM2.5, AQI ranging from 0-500 has different concentrations for each pollutant and has health effects accordingly.

Indian AQI range & probable impacts:

0-50: This range defines air quality as good as it shows minimal or no impact on health.

51-100: This is a satisfactory air quality range and it can show effects such as breathing difficulty in sensitive groups.

101-200: The range shows moderate air quality with impacts such as breathing discomfort for children and elderly people, and people already suffering from lung disorders and heart disease.

201-300: AQI falling in this range communicates that the air quality is poor and shows health effects on people when exposed for the long term. People already suffering from heart diseases can experience discomfort from short exposure.

301-400: This range shows very poor air quality and causes respiratory illness for a longer duration of exposure.

401-500: This is the severe range of AQI causing health impacts to normal and diseased people. It also causes severe health impacts on sensitive groups.

**Calculation of AQI:**

To calculate sub-indices, 16 hours of data is needed.

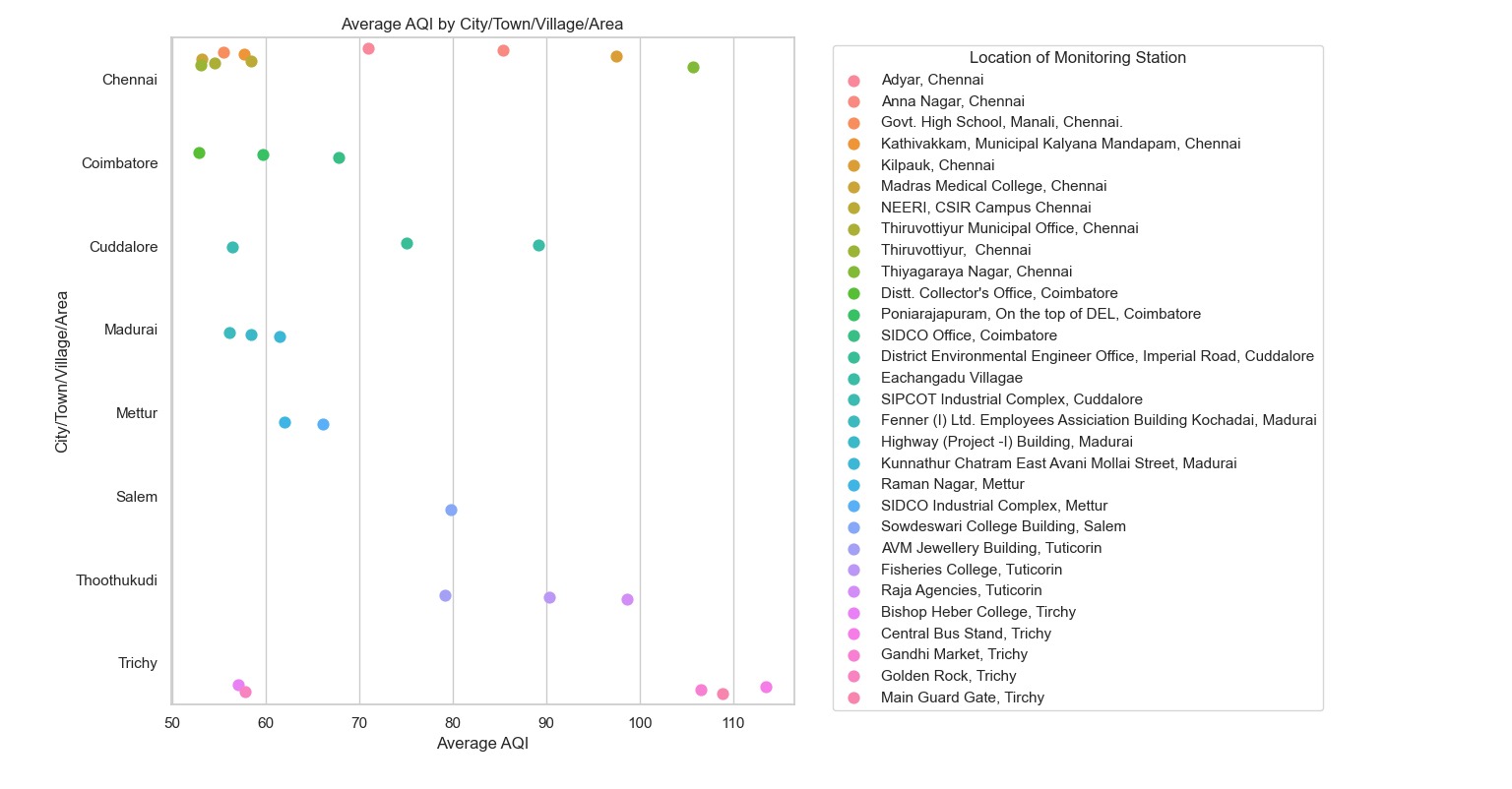
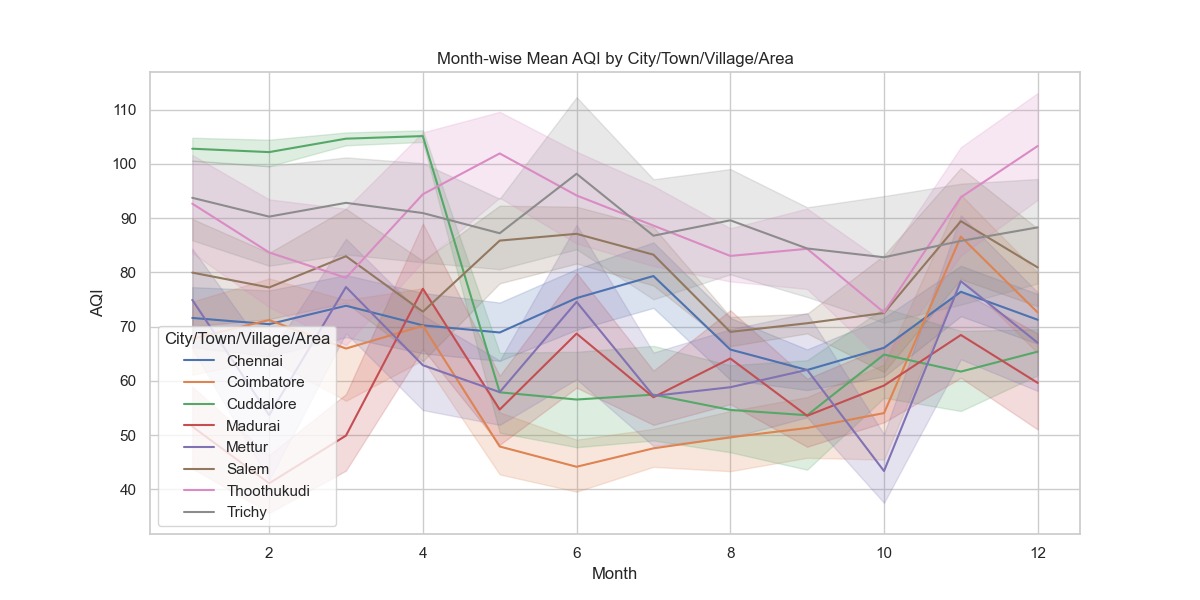
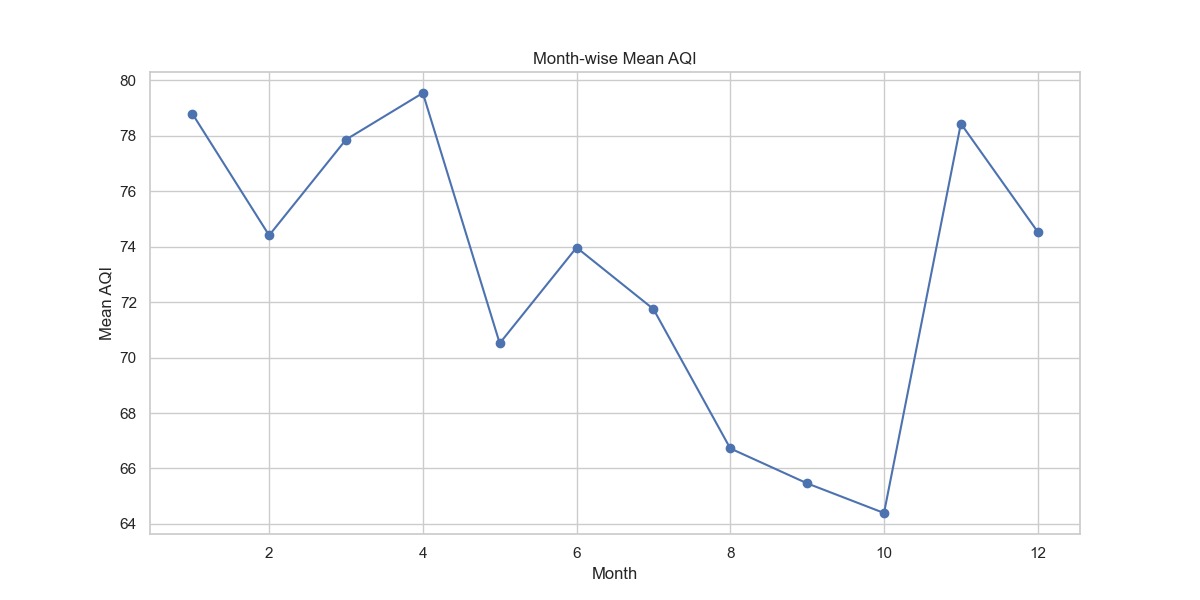
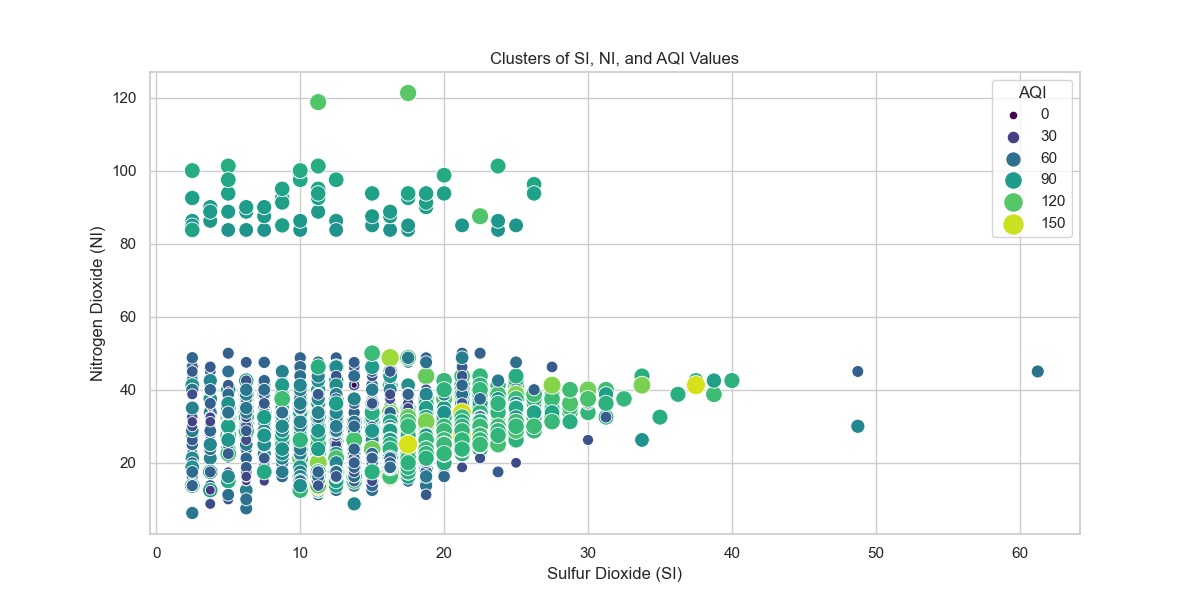
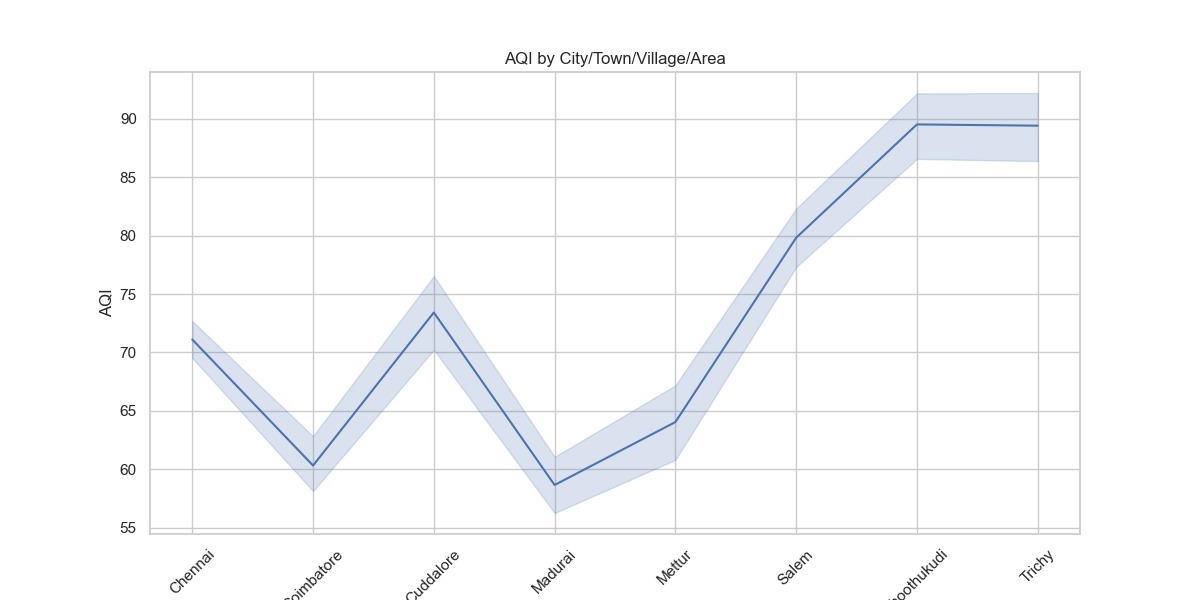
For example: If you wish to calculate AQI on the basis of PM2.5, CO, and ozone, calculate the sub-index for each parameter separately.

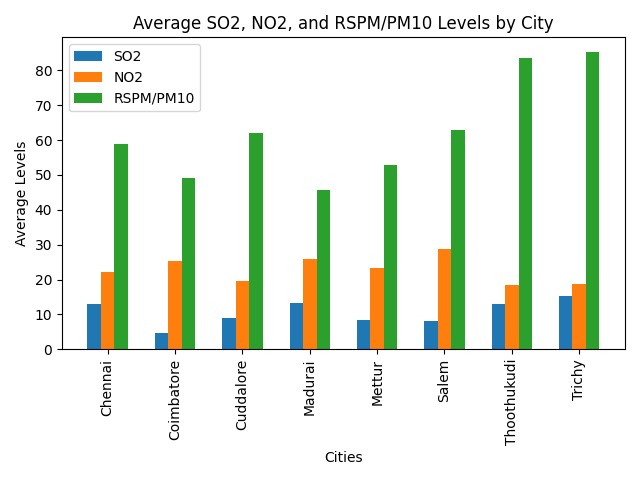
If the current concentration of PM2.5 is 110 ug/m3, then referring to AQI range as per Indian standards BPHi = 120, BPLo = 91, IHi = 300 and ILo = 201.

Putting the values in equation and solving:

Sub Index= [(300-201)/ (120-91)] (110-91) + 201 = 265.86

**RESULTS:**

****

****

**CODE:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from statsmodels.tsa.seasonal import seasonal\_decompose

df = pd.read\_csv(r'D:\nm\_dsc\cpcb\_dly\_aq\_tamil\_nadu-2014.csv')

print(df.head)

print("INFO:")

print(df.info())

print("\nDescribe:")

print(df.describe())

print("\nShape")

print(df.shape)

print("\nREMOVING COLUMNS WITH NULL VALUES\n ")

df = df.drop('PM 2.5', axis=1)

df.dropna(inplace=True)

# Drop duplicate rows

print("\nDROPPING DUPLICATE ROWS:\n")

df.drop\_duplicates(subset=None, inplace=True)

print(df.head)

print("\nCONVERTING TO DATE-TIME FORMAT\n")

# Convert 'Sampling Date' column to datetime format

df['Sampling Date'] = pd.to\_datetime(df['Sampling Date'])

# Display the first few rows after preprocessing

print("\nHead after preprocessing:")

print(df.head)

unique\_locations = df['Location of Monitoring Station'].unique()

# Display the unique locations

print("\nLocations of Monitoring Stations:")

print(unique\_locations)

# Group by 'City/Town/Village/Area' and count the number of monitoring stations in each city

city\_station\_counts = df.groupby('City/Town/Village/Area')['Location of Monitoring Station'].count().reset\_index()

# Rename the columns for clarity

city\_station\_counts.columns = ['City', 'Number of Monitoring Stations']

# Display the result

print("\nCity-wise Number of Monitoring Stations:")

print(city\_station\_counts)

# Group by both 'City/Town/Village/Area' and 'Location of Monitoring Station' and count the number of rows

location\_counts = df.groupby(['City/Town/Village/Area', 'Location of Monitoring Station']).size().reset\_index()

location\_counts.columns = ['City', 'Location', 'Number of Rows']

# Display the result

print("\nLocation-wise Number of Rows with City:")

print(location\_counts)

# Calculate the sum of 'SO2' and 'NO2' levels for each group

# Group by 'City/Town/Village/Area' and 'Location of Monitoring Station' and calculate the sum and average SO2 levels

# Group by 'City/Town/Village/Area' and 'Location of Monitoring Station' and calculate the sum and average SO2 and NO2 levels

# Group by 'City/Town/Village/Area' and 'Location of Monitoring Station' and calculate the sum and average levels

summary = df.groupby(['City/Town/Village/Area', 'Location of Monitoring Station'])[['SO2', 'NO2', 'RSPM/PM10']].agg(['sum', 'mean']).reset\_index()

# Rename columns for clarity

summary.columns = ['City', 'Location', 'SO2 Sum', 'SO2 Average', 'NO2 Sum', 'NO2 Average', 'RSPM/PM10 Sum', 'RSPM/PM10 Average']

# Display the result

print("\nSummary of SO2, NO2, and RSPM/PM10 Levels by Location:")

print(summary)

print()

# Group by 'City/Town/Village/Area' and calculate the average levels

city\_avg = df.groupby('City/Town/Village/Area')[['SO2', 'NO2', 'RSPM/PM10']].mean().reset\_index()

# Rename columns for clarity

city\_avg.columns = ['City', 'SO2 Average', 'NO2 Average', 'RSPM/PM10 Average']

# Display the result

print("\nAverage SO2, NO2, and RSPM/PM10 Levels by City:")

print(city\_avg)

cities = city\_avg['City']

so2\_avg = city\_avg['SO2 Average']

no2\_avg = city\_avg['NO2 Average']

rspm\_avg = city\_avg['RSPM/PM10 Average']

# Bar width

bar\_width = 0.2

# Positions for the bars on the x-axis

r1 = range(len(cities))

r2 = [x + bar\_width for x in r1]

r3 = [x + bar\_width for x in r2]

# Create the bar graph

plt.bar(r1, so2\_avg, width=bar\_width, label='SO2')

plt.bar(r2, no2\_avg, width=bar\_width, label='NO2')

plt.bar(r3, rspm\_avg, width=bar\_width, label='RSPM/PM10')

# X-axis labels

plt.xlabel('Cities')

plt.xticks([x + bar\_width for x in r1], cities, rotation=90)

# Y-axis label

plt.ylabel('Average Levels')

# Graph title

plt.title('Average SO2, NO2, and RSPM/PM10 Levels by City')

# Add a legend

plt.legend()

# Show the graph

plt.tight\_layout()

plt.show()

# Group by both 'City/Town/Village/Area' and 'Location of Monitoring Station' and count the number of rows

import matplotlib.pyplot as plt

# Assuming you have the 'summary' DataFrame with 'City', 'Location', 'SO2 Sum', 'SO2 Average', 'NO2 Sum', 'NO2 Average', 'RSPM/PM10 Sum', 'RSPM/PM10 Average' columns

# Get a list of unique cities

unique\_cities = summary['City'].unique()

# Iterate through each city and create a separate graph for each

"""for city in unique\_cities:

    city\_data = summary[summary['City'] == city]

    # Data for the current city

    locations = city\_data['Location']

    so2\_avg = city\_data['SO2 Average']

    no2\_avg = city\_data['NO2 Average']

    rspm\_avg = city\_data['RSPM/PM10 Average']

    # Create a bar graph for the current city

    plt.figure(figsize=(10, 5))

    plt.bar(locations, so2\_avg, width=0.2, label='SO2')

    plt.bar(locations, no2\_avg, width=0.2, label='NO2', bottom=so2\_avg)

    plt.bar(locations, rspm\_avg, width=0.2, label='RSPM/PM10', bottom=so2\_avg + no2\_avg)

    # X-axis labels

    plt.xlabel('Locations')

    plt.xticks(rotation=45, ha='right')

    # Y-axis label

    plt.ylabel('Average Levels')

    # Graph title

    plt.title(f'Average Pollutant Levels in {city}')

    # Add a legend

    plt.legend()

    # Show the graph

    plt.tight\_layout()

    plt.show()

# Group by 'Location of Monitoring Station' and count the number of rows for each location

# Merge the two DataFrames on 'Location of Monitoring Station' to combine the results

"""

#calculating individual index for so2 and no2

#so2

def calculate\_si(SO2):

    si=0

    if (SO2<=40):

     si= SO2\*(50/40)

    if (SO2>40 and SO2<=80):

     si= 50+(SO2-40)\*(50/40)

    if (SO2>80 and SO2<=380):

     si= 100+(SO2-80)\*(100/300)

    if (SO2>380 and SO2<=800):

     si= 200+(SO2-380)\*(100/800)

    if (SO2>800 and SO2<=1600):

     si= 300+(SO2-800)\*(100/800)

    if (SO2>1600):

     si= 400+(SO2-1600)\*(100/800)

    return si

df['si']=df['SO2'].apply(calculate\_si)

df[['SO2','si']]

print(df.head(20))

# Check the column names in your DataFrame

print(df.columns)

def calculate\_ni(NO2):

    ni=0

    if(NO2<=40):

     ni= NO2\*50/40

    elif(NO2>40 and NO2<=80):

     ni= 50+(NO2-14)\*(50/40)

    elif(NO2>80 and NO2<=180):

     ni= 100+(NO2-80)\*(100/100)

    elif(NO2>180 and NO2<=280):

     ni= 200+(NO2-180)\*(100/100)

    elif(NO2>280 and NO2<=400):

     ni= 300+(NO2-280)\*(100/120)

    else:

     ni= 400+(NO2-400)\*(100/120)

    return ni

df['ni']=df['NO2'].apply(calculate\_ni)

df[['NO2','ni']]

print(df.head(20))

"""1. Indian equation for AQI:

The Indian AQI range differs from that of US-EPA. To calculate AQI, a minimum of three parameters should be taken out of which one must be either PM10 or PM2.5.

To calculate sub-indices, 16 hours of data is needed.

For example: If you wish to calculate AQI on the basis of PM2.5, CO, and ozone, calculate the sub-index for each parameter separately.

If the current concentration of PM2.5 is 110 ug/m3, then referring to AQI range as per Indian standards BPHi = 120, BPLo = 91, IHi = 300 and ILo = 201.

Putting the values in equation and solving:

Sub Index= [(300-201)/ (120-91)] (110-91) + 201 = 265.86

Similarly, for other parameters, the sub-index can be calculated and the worst sub-index shows the AQI."""

df.rename(columns={'RSPM/PM10': 'rspm'}, inplace=True)

def calculate\_(rspm):

    rpi=0

    if(rspm<=30):

     rpi=rspm\*50/30

    elif(rspm>30 and rspm<=60):

     rpi=50+(rspm-30)\*50/30

    elif(rspm>60 and rspm<=90):

     rpi=100+(rspm-60)\*100/30

    elif(rspm>90 and rspm<=120):

     rpi=200+(rpi-90)\*100/30

    elif(rspm>120 and rspm<=250):

     rpi=300+(rspm-120)\*(100/130)

    else:

     rpi=400+(rspm-250)\*(100/130)

    return rpi

df['rpi']=df['rspm'].apply(calculate\_si)

df[['rspm','rpi']]

print(df.tail(20))

print(df.columns)

#function to calculate the air quality index (AQI) of every data value

#its is calculated as per indian govt standards

def calculate\_aqi(si,ni,rpi):

    aqi=0

    if(si>ni and si>rpi):

     aqi=si

    if(ni>si and ni>rpi):

     aqi=ni

    if(rpi>si and rpi>ni):

     aqi=rpi

    return aqi

df['AQI']=df.apply(lambda x:calculate\_aqi(x['si'],x['ni'],x['rpi']),axis=1)

df[['si','ni','rpi','AQI']]

print(df.head(20))

grouped\_data = df.groupby(['City/Town/Village/Area', 'Location of Monitoring Station'])

# Calculate the average AQI for each unique combination

average\_aqi = grouped\_data['AQI'].mean().reset\_index()

# Print the first few rows of the resulting DataFrame to check the data

print(average\_aqi)

sns.set(style="whitegrid")

# Create a point plot

plt.figure(figsize=(12, 8))

sns.pointplot(x="AQI", y="City/Town/Village/Area", data=average\_aqi, hue="Location of Monitoring Station", join=False, dodge=True)

plt.title("Average AQI by City/Town/Village/Area")

plt.xlabel("Average AQI")

plt.ylabel("City/Town/Village/Area")

plt.legend(title="Location of Monitoring Station", bbox\_to\_anchor=(1.05, 1), loc='upper left')

plt.tight\_layout()

plt.show()

print(df.columns)

df.to\_csv(r'D:\nm\_dsc\cpcb\_dly\_aq\_tamil\_nadu-2014-modified.csv', index=False)

print("saved")

df = pd.read\_csv(r'D:\nm\_dsc\cpcb\_dly\_aq\_tamil\_nadu-2014-modified.csv')

print(df.head)

#monthwise mean aqi which speaks on the pollution level

df['Sampling Date'] = pd.to\_datetime(df['Sampling Date'])

# Extract year and month

df['Year'] = df['Sampling Date'].dt.year

df['Month'] = df['Sampling Date'].dt.month

# Group the data by 'Stn Code,' Year, and Month

grouped\_data = df.groupby(['City/Town/Village/Area', 'Year', 'Month'])

# Calculate the mean 'AQI' for each group

month\_wise\_mean\_aqi = grouped\_data['AQI'].mean().reset\_index()

# Print the first few rows of the resulting DataFrame

print(month\_wise\_mean\_aqi.head(25))

sns.set(style="whitegrid")

# Create a figure and axis

fig, ax = plt.subplots(figsize=(12, 6))

# Plot the data for each 'City/Town/Village/Area'

sns.lineplot(x="Month", y="AQI", hue="City/Town/Village/Area", data=df, ax=ax)

# Set labels and title

ax.set\_xlabel("Month")

ax.set\_ylabel("AQI")

ax.set\_title("Month-wise Mean AQI by City/Town/Village/Area")

# Add a legend

ax.legend(title="City/Town/Village/Area")

# Show the plot

plt.show()

#overall month trends

# Convert the 'Sampling Date' column to datetime

df['Sampling Date'] = pd.to\_datetime(df['Sampling Date'])

# Extract the year and month from the 'Sampling Date'

df['Year'] = df['Sampling Date'].dt.year

df['Month'] = df['Sampling Date'].dt.month

# Calculate the mean AQI for all data (no grouping by 'City/Town/Village/Area')

month\_wise\_mean\_aqi = df.groupby(['Year', 'Month'])['AQI'].mean().reset\_index()

# Print the first few rows of the resulting DataFrame to check the data

print(month\_wise\_mean\_aqi.head(100))

df['Sampling Date'] = pd.to\_datetime(df['Sampling Date'])

# Extract the year and month from the 'Sampling Date'

df['Year'] = df['Sampling Date'].dt.year

df['Month'] = df['Sampling Date'].dt.month

# Calculate the mean AQI for all data (no grouping by 'City/Town/Village/Area')

month\_wise\_mean\_aqi = df.groupby(['Year', 'Month'])['AQI'].mean().reset\_index()

# Create a line plot for month-wise mean AQI

plt.figure(figsize=(12, 6))

plt.plot(month\_wise\_mean\_aqi['Month'], month\_wise\_mean\_aqi['AQI'], marker='o')

plt.title('Month-wise Mean AQI')

plt.xlabel('Month')

plt.ylabel('Mean AQI')

plt.grid(True)

# Show the plot

plt.show()

plt.figure(figsize=(12, 6))

sns.scatterplot(x='si', y='ni', hue='AQI', data=df, palette='viridis', size='AQI', sizes=(20, 200))

# Set labels and title

plt.xlabel('Sulfur Dioxide (SI)')

plt.ylabel('Nitrogen Dioxide (NI)')

plt.title('Clusters of SI, NI, and AQI Values')

# Show the legend

plt.legend(title='AQI')

# Show the plot

plt.show()

#In this code, we use sns.scatterplot from the Seaborn library to create a scatter plot that visualizes the clusters of SI, NI, and AQI values. The hue parameter is set to 'AQI' to color the data points based on the AQI values. The size parameter is also set to 'AQI' to represent AQI values with different point sizes. You can customize the colors, size range, and other plot aesthetics as needed.

# Your previous code to read and preprocess the data

sns.set(style="whitegrid")

# Create a figure and axis

fig, ax = plt.subplots(figsize=(12, 6))

# Plot the data for each 'City/Town/Village/Area'

sns.lineplot(x="City/Town/Village/Area", y="AQI", data=df, ax=ax)

# Set labels and title

ax.set\_xlabel("City/Town/Village/Area")

ax.set\_ylabel("AQI")

ax.set\_title("AQI by City/Town/Village/Area")

# Rotate x-axis labels for better readability

plt.xticks(rotation=45)

# Show the plot

plt.show()