Analyzing the Air Quality Index

Analyzing the Air Quality Index (AQI) is a fundamental aspect of environmental data science, vital for monitoring and managing air quality in specific areas. This article delves into the process of conducting AQI analysis using Python, offering insights into how to effectively assess air quality levels.

The process of Air Quality Index Analysis involves several key steps aimed at providing a numerical representation of overall air quality, crucial for both public health and environmental management. Let's outline the steps involved:

1. **Data Collection**: Begin by gathering air quality data from diverse sources, including government monitoring stations, sensor networks, or satellite imagery. Accessing a wide range of data sources ensures comprehensive coverage and accuracy in the analysis.
2. **Data Cleaning and Preprocessing**: Once the data is collected, it needs to be cleaned and preprocessed to address any inconsistencies, missing values, or outliers. Data cleaning ensures that the subsequent analysis is based on reliable and accurate information.
3. **AQI Calculation**: Utilize standardized formulas and guidelines provided by environmental agencies to calculate the Air Quality Index. These formulas typically consider various pollutants such as particulate matter, ozone, carbon monoxide, sulfur dioxide, and nitrogen dioxide to derive a comprehensive AQI value.
4. **Visualization**: Create visual representations of the AQI data using techniques such as line charts, heatmaps, or geographic visualizations. Visualizations help in understanding trends and patterns in air quality over time or across different geographical regions, aiding in effective communication of findings.
5. **Comparison and Evaluation**: Compare the AQI metrics of the analyzed location with recommended air quality standards and guidelines. This comparison provides valuable insights into the level of air pollution and its potential impacts on public health and the environment.

By following these steps, analysts can gain a comprehensive understanding of air quality conditions in specific areas, facilitating informed decision-making and targeted interventions to improve air quality and safeguard public health.

```
# Mounting to you own Google Colab drive
from google.colab import drive
try:
  drive.mount('/gdrive')
except:
  drive.mount('/content/gdrive', force_remount=True)
%cd '/gdrive/MyDrive/projects'
import pandas as pd
import plotly.express as px
import plotly.io as pio
import plotly.graph_objects as go
pio.templates.default = "plotly_white"

data = pd.read_csv("/gdrive/My Drive/projects/delhiaqi.csv")
print(data.head())
```

```
Mounted at /gdrive
/gdrive/MyDrive/projects
                  date       co     no    no2    o3    so2   pm2_5    pm10  \
0  2023-01-01 00:00:00  1655.58   1.66  39.41  5.90  17.88  169.29  194.64
1  2023-01-01 01:00:00  1869.20   6.82  42.16  1.99  22.17  182.84  211.08
2  2023-01-01 02:00:00  2510.07  27.72  43.87  0.02  30.04  220.25  260.68
3  2023-01-01 03:00:00  3150.94  55.43  44.55  0.85  35.76  252.90  304.12
4  2023-01-01 04:00:00  3471.37  68.84  45.24  5.45  39.10  266.36  322.80

     nh3
0   5.83
1   7.66
2  11.40
3  13.55
4  14.19
```

Converting the date column in the dataset into a datetime data type is a crucial step in time-series analysis. This conversion allows you to treat dates as meaningful entities rather than mere strings, enabling various time-based operations and analysis

```
data['date'] = pd.to_datetime(data['date'])
```

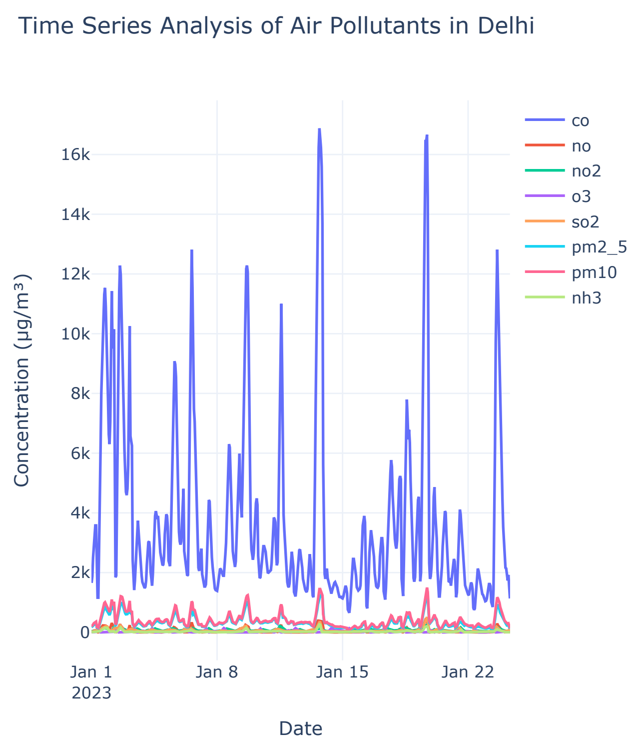
```
print(data.describe())
```

```
                 co          no         no2          o3         so2  \
count    561.000000  561.000000  561.000000  561.000000  561.000000
mean    3814.942210   51.181979   75.292496   30.141943   64.655936
std     3227.744681   83.904476   42.473791   39.979405   61.073080
min      654.220000    0.000000   13.370000    0.000000    5.250000
25%     1708.980000    3.380000   44.550000    0.070000   28.130000
50%     2590.180000   13.300000   63.750000   11.800000   47.210000
75%     4432.680000   59.010000   97.330000   47.210000   77.250000
max    16876.220000  425.580000  263.210000  164.510000  511.170000

             pm2_5         pm10         nh3
count   561.000000   561.000000  561.000000
mean    358.256364   420.988414   26.425062
std     227.359117   271.287026   36.563094
min      60.100000    69.080000    0.630000
25%     204.450000   240.900000    8.230000
50%     301.170000   340.900000   14.820000
75%     416.650000   482.570000   26.350000
max    1310.200000  1499.270000  267.510000
```

Let's examine the trend of pollutant concentrations over time in the air quality dataset.

Plotting time series for each air pollutant

```
fig = go.Figure()

for pollutant in ['co', 'no', 'no2', 'o3', 'so2', 'pm2_5', 'pm10', 'nh3']:
    fig.add_trace(go.Scatter(x=data['date'], y=data[pollutant], mode='lines',
                             name=pollutant))
fig.update_layout(title='Time Series Analysis of Air Pollutants in Delhi',
                  xaxis_title='Date', yaxis_title='Concentration (µg/m³)',
                  height=600, width=500)  # Adjust height and width as needed

fig.show()
```



In the above code, we are generating individual time series plots for each air pollutant present in the dataset. This visualization aids in understanding the variation in pollutant concentrations over time, facilitating analysis of air quality trends.

Moving forward, our next step involves calculating the Air Quality Index (AQI) and categorizing it accordingly. The AQI computation is based on the concentrations of various pollutants, with each pollutant contributing to its respective sub-index. Let's outline the process for calculating AQI:

Calculating Air Quality Index

In the provided code, we establish AQI breakpoints and their corresponding values for various air pollutants in accordance with Air Quality Index (AQI) standards. The list named aqi_breakpoints delineates concentration ranges alongside their associated AQI values for different pollutants.

```
# Define AQI breakpoints and corresponding AQI values
aqi_breakpoints = [
    (0, 12.0, 50), (12.1, 35.4, 100), (35.5, 55.4, 150),
    (55.5, 150.4, 200), (150.5, 250.4, 300), (250.5, 350.4, 400),
    (350.5, 500.4, 500)
]

def calculate_aqi(pollutant_name, concentration):
    for low, high, aqi in aqi_breakpoints:
        if low <= concentration <= high:
            return aqi
    return None

def calculate_overall_aqi(row):
    aqi_values = []
    pollutants = ['co', 'no', 'no2', 'o3', 'so2', 'pm2_5', 'pm10', 'nh3']
    for pollutant in pollutants:
        aqi = calculate_aqi(pollutant, row[pollutant])
        if aqi is not None:
            aqi_values.append(aqi)
    return max(aqi_values)

# Calculate AQI for each row
data['AQI'] = data.apply(calculate_overall_aqi, axis=1)

# Define AQI categories
aqi_categories = [
    (0, 50, 'Good'), (51, 100, 'Moderate'), (101, 150, 'Unhealthy for Sensitive Groups'),
    (151, 200, 'Unhealthy'), (201, 300, 'Very Unhealthy'), (301, 500, 'Hazardous')
]

def categorize_aqi(aqi_value):
    for low, high, category in aqi_categories:
        if low <= aqi_value <= high:
            return category
    return None

# Categorize AQI
data['AQI Category'] = data['AQI'].apply(categorize_aqi)
print(data.head())
```

```
                  date       co     no    no2    o3    so2   pm2_5    pm10  \
0 2023-01-01 00:00:00  1655.58   1.66  39.41  5.90  17.88  169.29  194.64
1 2023-01-01 01:00:00  1869.20   6.82  42.16  1.99  22.17  182.84  211.08
2 2023-01-01 02:00:00  2510.07  27.72  43.87  0.02  30.04  220.25  260.68
3 2023-01-01 03:00:00  3150.94  55.43  44.55  0.85  35.76  252.90  304.12
4 2023-01-01 04:00:00  3471.37  68.84  45.24  5.45  39.10  266.36  322.80

     nh3  AQI    AQI Category
0   5.83  300  Very Unhealthy
1   7.66  300  Very Unhealthy
2  11.40  400       Hazardous
3  13.55  400       Hazardous
4  14.19  400       Hazardous
```

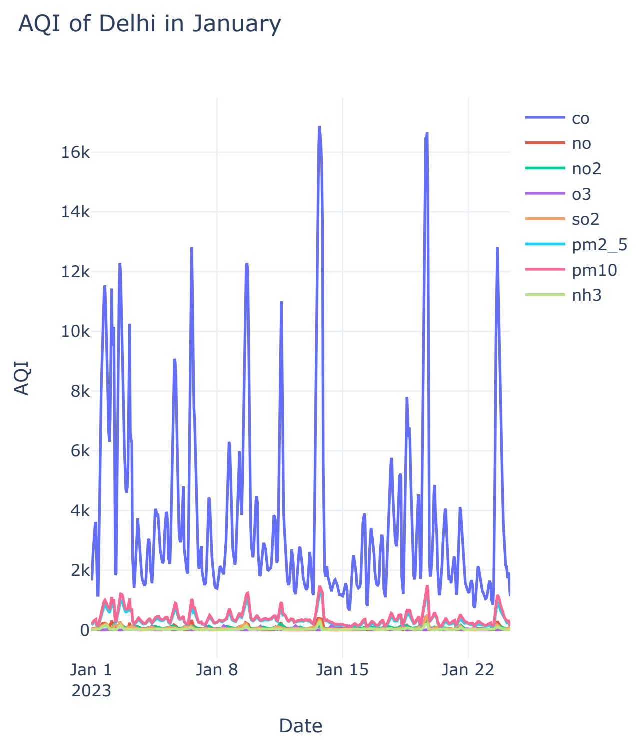
Define two key functions:

1. `calculate_aqi` : This function computes the AQI for a specific pollutant and concentration by identifying the suitable range within the `aqi_breakpoints` .
2. `calculate_overall_aqi` : This function determines the overall AQI for a given row in the dataset by selecting the maximum AQI value among all pollutants.

The computed AQI values are integrated as a new column within the dataset. Additionally, we establish AQI categories within the `aqi_categories` list and employ the `categorize_aqi` function to assign an AQI category to each AQI value. These resulting AQI categories are subsequently appended as a new column labeled "AQI Category" in the dataset.

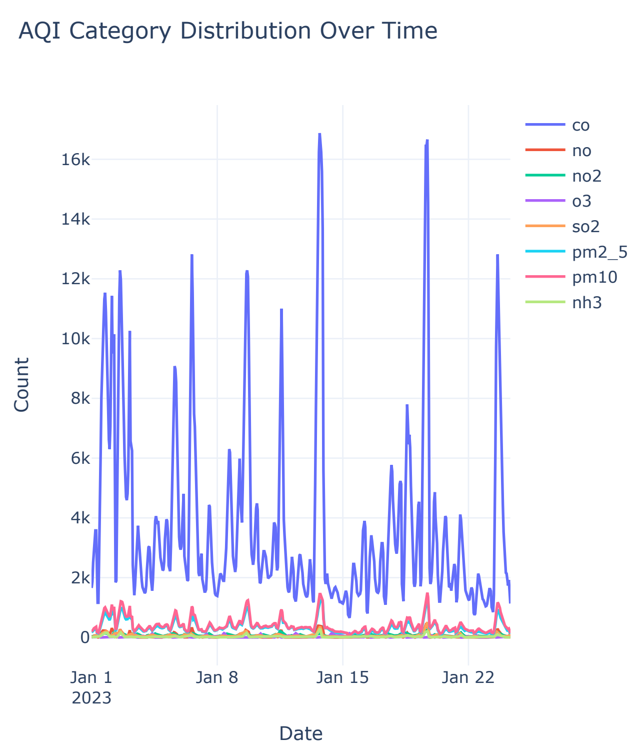
Analysis of Delhi's AQI for the month of January

```
fig.update_layout(title="AQI of Delhi in January",
                  xaxis_title="Date",
                  yaxis_title="AQI",
                  height=600,  # Adjust height as needed
                  width=500)  # Adjust width as needed
fig.show()
```



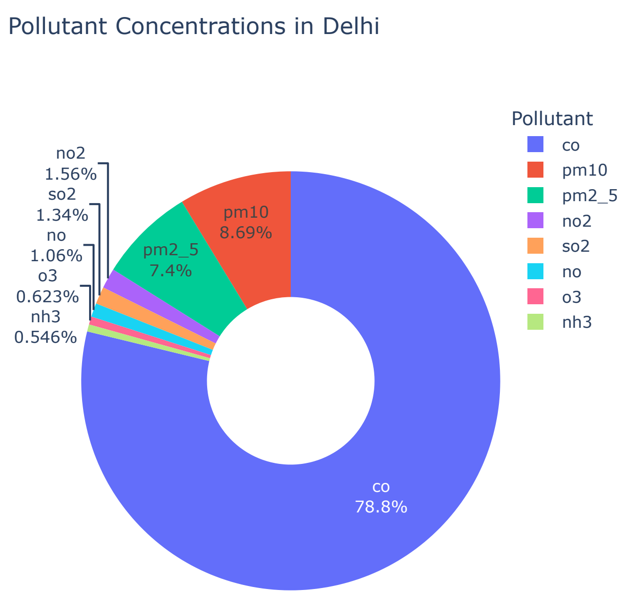
The provided code utilizes Plotly to generate a histogram depicting the distribution of AQI categories over time. Each bar represents the count of occurrences of AQI categories for respective dates. The histogram is created using the px.histogram() function, with the x-axis representing dates and the bars colored according to AQI categories. The title of the plot is set to "AQI Category Distribution Over Time," and the x-axis and y-axis titles are specified as "Date" and "Count," respectively. To adjust the size of the plot, you can utilize the update_layout() method and set the height and width parameters accordingly. Here's how you can modify the code:

```
fig.update_layout(title="AQI Category Distribution Over Time",
                  xaxis_title="Date",
                  yaxis_title="Count",
                  height=600,  # Adjust height as needed
                  width=500)  # Adjust width as needed
```



To visualize the distribution of pollutants in the air quality of Delhi, you can create a bar chart or a box plot showing the concentration of each pollutant.

```
# Define pollutants and their colors
pollutants = ["co", "no", "no2", "o3", "so2", "pm2_5", "pm10", "nh3"]
pollutant_colors = px.colors.qualitative.Plotly

# Calculate the sum of pollutant concentrations
total_concentrations = data[pollutants].sum()

# Create a DataFrame for the concentrations
concentration_data = pd.DataFrame({
    "Pollutant": pollutants,
    "Concentration": total_concentrations
})

# Create a donut plot for pollutant concentrations
fig = px.pie(concentration_data, names="Pollutant", values="Concentration",
             title="Pollutant Concentrations in Delhi",
              height=600,  # Adjust height as needed
              width=500,
             hole=0.4, color_discrete_sequence=pollutant_colors)

# Update layout for the donut plot
fig.update_traces(textinfo="percent+label")
fig.update_layout(legend_title="Pollutant")

# Show the donut plot
fig.show()
```

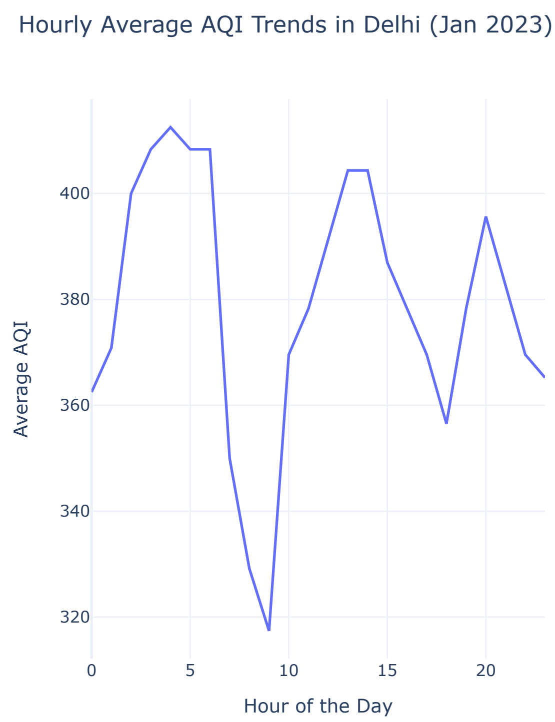



```
# Correlation Between Pollutants
correlation_matrix = data[pollutants].corr()
fig = px.imshow(correlation_matrix, x=pollutants,
                 y=pollutants, title="Correlation Between Pollutants",
                height=600,  # Adjust height as needed
              width=500)
fig.show()
```

The correlation matrix presented here illustrates the correlation coefficients between various air pollutants within the dataset. Correlation coefficients quantify the strength and direction of linear relationships between pairs of variables, with values ranging from -1 to 1. The positive correlations observed among CO, NO, NO2, SO2, PM2.5, PM10, and NH3 imply potential shared sources or similar pollution patterns among these pollutants. Conversely, O3 displays an inverse relationship with the other pollutants, likely attributed to its dual role as both a pollutant and a natural atmospheric oxidant.

Moving forward, let's explore the hourly average trends of AQI in Delhi:

```
# Extract the hour from the date
data['Hour'] = pd.to_datetime(data['date']).dt.hour

# Calculate hourly average AQI
hourly_avg_aqi = data.groupby('Hour')['AQI'].mean().reset_index()

# Create a line plot for hourly trends in AQI
fig = px.line(hourly_avg_aqi, x='Hour', y='AQI',
              title='Hourly Average AQI Trends in Delhi (Jan 2023)',
                height=600,  # Adjust height as needed
              width=500)
fig.update_xaxes(title="Hour of the Day")
fig.update_yaxes(title="Average AQI")
fig.show()
```



The average AQI by day of the week in Delhi:

```
# Average AQI by Day of the Week
data['Day_of_Week'] = data['date'].dt.day_name()
average_aqi_by_day = data.groupby('Day_of_Week')['AQI'].mean().reindex(['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday',
fig = px.bar(average_aqi_by_day, x=average_aqi_by_day.index, y='AQI',
              title='Average AQI by Day of the Week',
             height=600,  # Adjust height as needed
              width=500)
fig.update_xaxes(title="Day of the Week")
fig.update_yaxes(title="Average AQI")
fig.show()
```