MACHINE LEARNING PROJECT

Air Quality Index Analysis using Python

Air Quality Index (AQI) analysis is a vital component of environmental data science, playing a key role in monitoring and assessing the quality of air in specific locations. This process provides a numerical indicator of air quality, which is crucial for public health protection and environmental management. The AQI is derived from various pollutants' concentrations and is calculated using standardized formulas provided by environmental agencies. To conduct AQI analysis, data is collected from multiple sources, such as government monitoring stations, air quality sensors, and satellite imagery. This data is then cleaned and preprocessed to remove inconsistencies and errors. After calculating the AQI, visualizations like line charts and heatmaps are created to depict changes over time or spatial differences across regions. By comparing these metrics to recommended air quality standards, authorities and individuals can make informed decisions regarding environmental policy and public health interventions. This analysis highlights the importance of continuous monitoring to maintain air quality and mitigate risks associated with pollution.

Here's a **step-by-step guide** along with key features for **Air Quality Index (AQI) Analysis**:

1. Data Collection

- **Key Point**: Gather air quality data from multiple sources.
- **Features**:
  - Use data from **government monitoring stations**, **air quality sensors**, or **satellite imagery**.
  - Include **key pollutants**: PM2.5, PM10, CO, NO2, SO2, and O3.
  - Ensure collection over a **sufficient time period** for meaningful analysis.

2. Data Cleaning and Preprocessing

- **Key Point**: Clean and preprocess the data for analysis.
- **Features**:
  - **Remove outliers** and handle **missing data**.
  - **Normalize** different datasets (different pollutants may have different units).
  - Convert timestamps and **geographical coordinates** into a standardized format.

3. AQI Calculation

- **Key Point**: Calculate the AQI using standardized formulas.

- Features:
    - Use formulas from **relevant environmental authorities** (e.g., EPA, WHO).
    - Compute **individual pollutant sub-indexes** for PM2.5, PM10, CO, etc.
    - Use the **maximum sub-index** to determine the overall AQI for a given period or location.

4. Visualization and Analysis

- **Key Point**: Visualize and analyze the AQI data.
- **Features**:
    - Create **line charts** to show AQI trends over time.
    - Develop **heatmaps** to show AQI levels across different geographical areas.
    - Use **bar charts** to compare pollutant-specific indices and their contributions to the AQI.

5. Comparison to Standards

- **Key Point**: Compare the calculated AQI values with recommended air quality standards.
- **Features**:
    - Compare the results against **national/international standards** (e.g., EPA, WHO guidelines).
    - Highlight periods or locations where AQI **exceeds safe limits**.
    - Identify the **primary pollutant contributors** in areas of concern.

6. Reporting and Insights

- **Key Point**: Generate insights and actionable reports.
- **Features**:
    - Provide **summary statistics**: average AQI, peak AQI, pollutant distributions.
    - Highlight regions or time periods with **dangerously high AQI**.
    - Offer **recommendations** for pollution control and public health advisories.
    - Predict trends using **forecasting models** for future AQI based on historical data.

7. Actionable Feedback

- **Key Point**: Suggest actions based on AQI outcomes.
- **Features**:

- Propose **policy interventions** for pollution control.
- Recommend **public health advisories** for vulnerable populations.
- Utilize the analysis for **urban planning** or **emission reduction** strategies.

This systematic approach ensures that the AQI analysis is comprehensive, actionable, and aligned with environmental and health guidelines.

Now, let's get started with the task of Air Quality Index Analysis by importing the necessary Python libraries

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.graph_objects as go
import plotly.express as px
```

```
data = pd.read_csv("delhiaqi.csv")
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
0   5.83
1   7.66
2  11.40
3  13.55
4  14.19
```

I'll convert the date column in the dataset into a datetime data type and move forward:

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

```
0     2023-01-01 00:00:00
1     2023-01-01 01:00:00
2     2023-01-01 02:00:00
3     2023-01-01 03:00:00
4     2023-01-01 04:00:00
              ...
556   2023-01-24 04:00:00
557   2023-01-24 05:00:00
558   2023-01-24 06:00:00
559   2023-01-24 07:00:00
560   2023-01-24 08:00:00
Name: date, Length: 561, dtype: datetime64[ns]
```

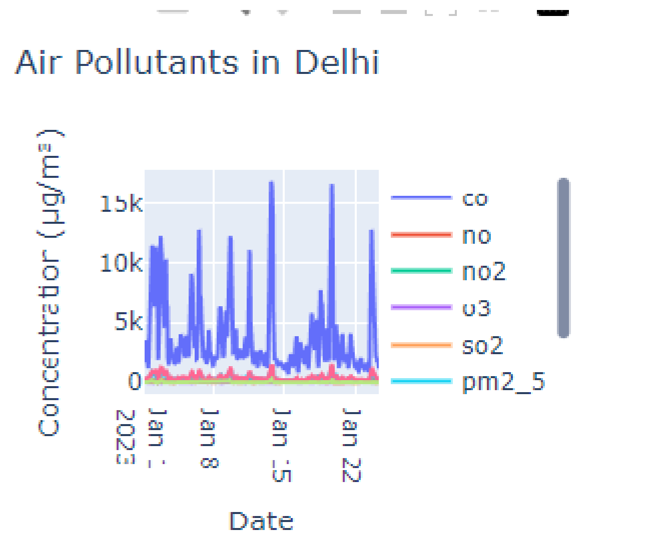
Now, let's have a look at the descriptive statistics of the data:

```python
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
```

Now let's have a look at the intensity of each pollutant over time in the air quality:

```
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='Air Pollutants in Delhi',
                  xaxis_title='Date', yaxis_title='Concentration (µg/m³)', width=300,
    height=300,)
fig.show()
```



In the above code, we are creating a air pollutant in delhi plot for each air pollutant in the dataset. It helps analyze the intensity of air pollutants over time.

Calculating Air Quality Index

Now, before moving forward, we need to calculate the air quality index and its category. AQI is typically computed based on the concentration of various pollutants, and each pollutant has its sub-index. Here's how we can calculate AQI:

```python
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)
data['AQI'] = data.apply(calculate_overall_aqi, axis=1)
aqi_categories = [
    (0, 50, 'Good'), (51, 100, 'Moderate'), (101, 150, 'Unhealthy for Sensitive Groups'),
    (151, 200, 'Unhealthy'), (201, 300, 'Very Unhealthy'), (301, 500, 'Hazardous')
]
```

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

In the above code, we are defining AQI breakpoints and corresponding AQI values for various air pollutants according to the Air Quality Index (AQI) standards. The aqi_breakpoints list defines the concentration ranges and their corresponding AQI values for different pollutants. We then define two functions:

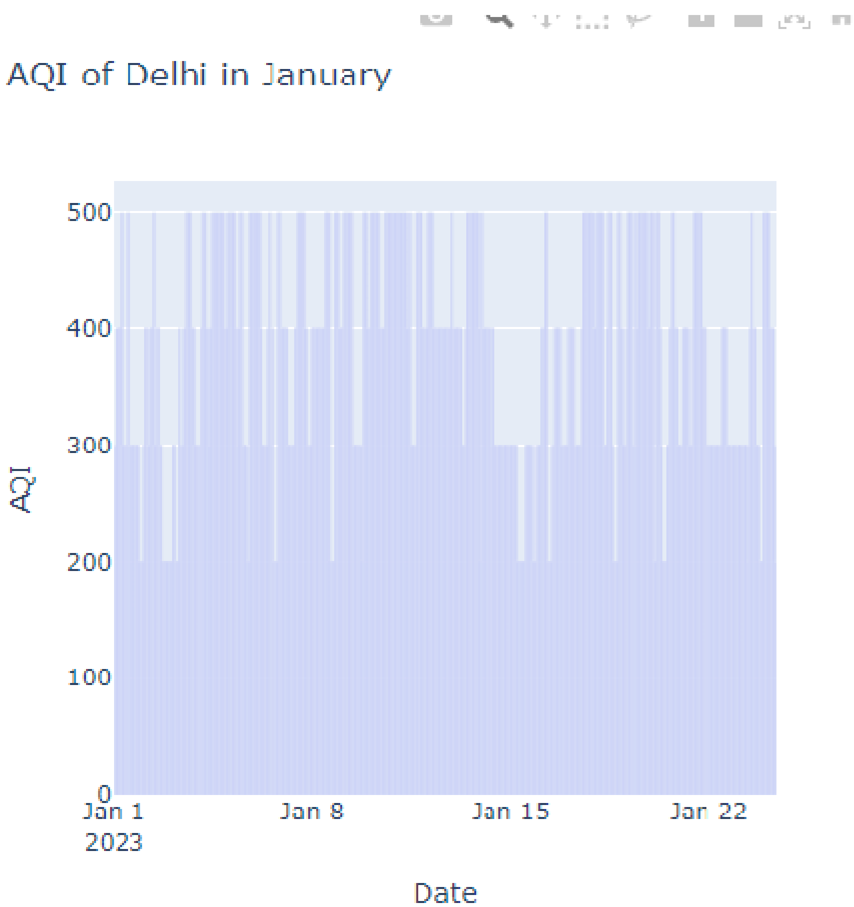
1. calculate_aqi: to calculate the AQI for a specific pollutant and concentration by finding the appropriate range in the aqi_breakpoints
2. calculate_overall_aqi: to calculate the overall AQI for a row in the dataset by considering the maximum AQI value among all pollutants

The calculated AQI values are added as a new column in the dataset. Additionally, we defined AQI categories in the aqi_categories list and used the categorize_aqi function to assign an AQI category to each AQI value. The resulting AQI categories are added as a new column as AQI Category in the dataset.

Analyzing AQI of Delhi

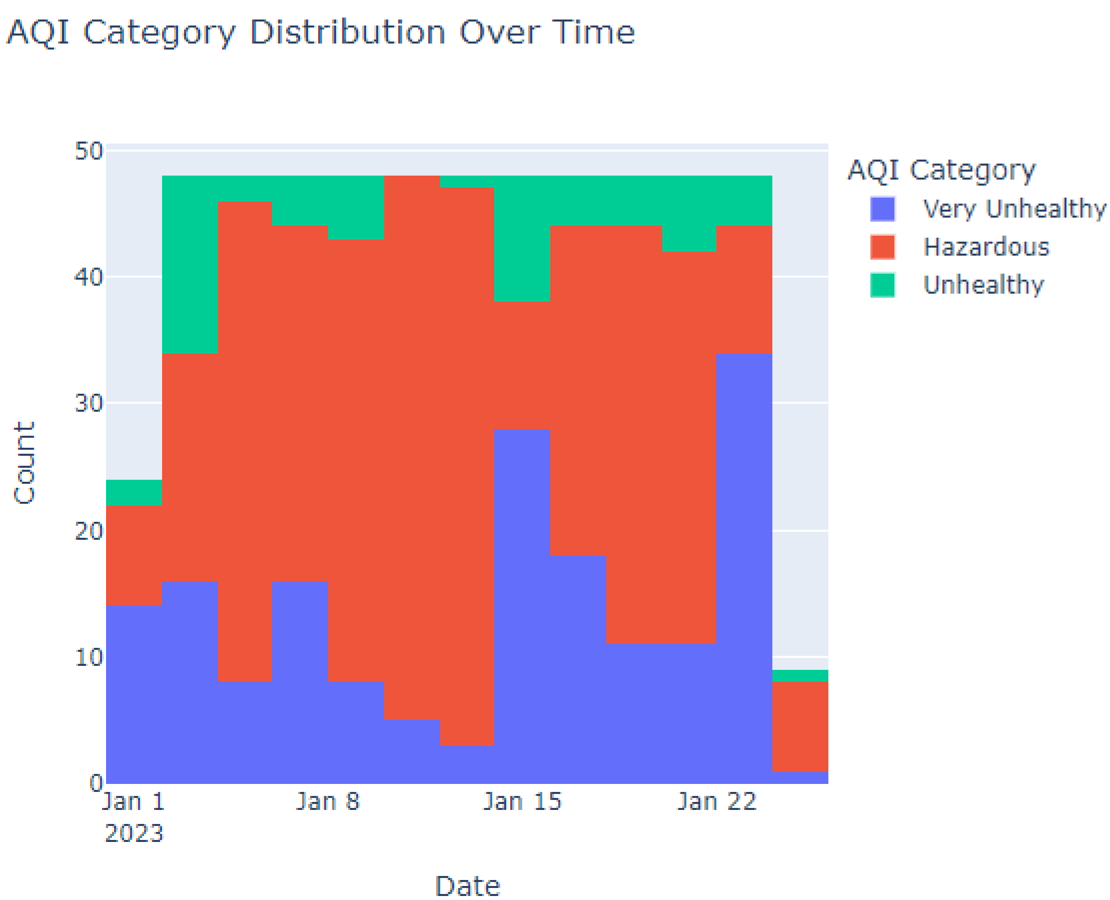
Now, let's have a look at the AQI of Delhi in January:

```
fig = px.bar(data, x="date", y="AQI",width=500,
             height=500,
             title="AQI of Delhi in January")
fig.update_xaxes(title="Date")
fig.update_yaxes(title="AQI")
fig.show()
```



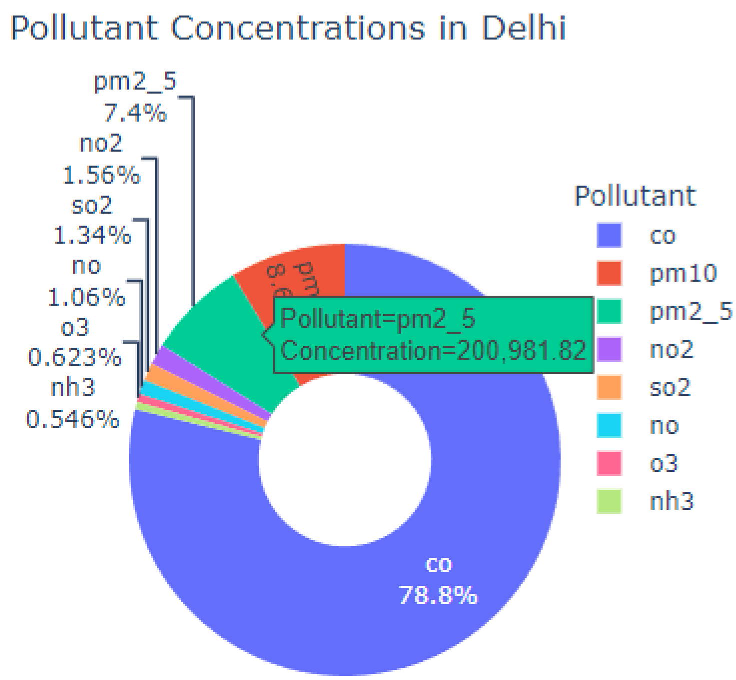
Now, let's have a look at the AQI category distribution:

```
fig = px.histogram(data, x="date",
                   color="AQI Category",width=600, height=500,
                   title="AQI Category Distribution Over Time")
fig.update_xaxes(title="Date")
fig.update_yaxes(title="Count")
fig.show()
```



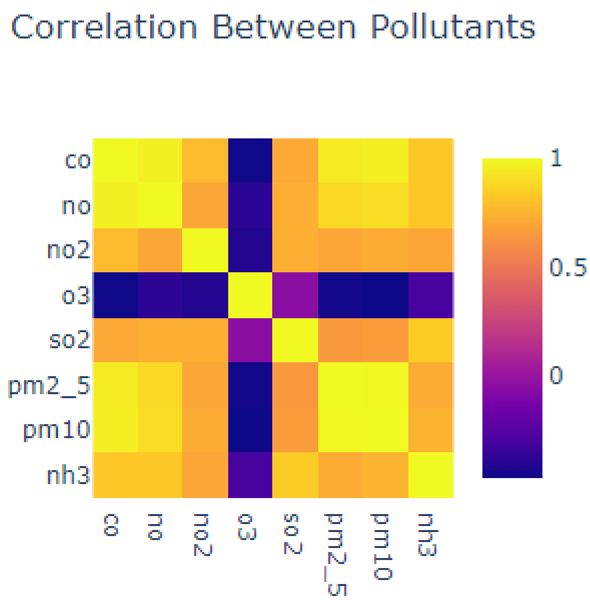
Now, let's have a look at the distribution of pollutants in the air quality of Delhi:

```
pollutants = ["co", "no", "no2", "o3", "so2", "pm2_5", "pm10", "nh3"]
pollutant_colors = px.colors.qualitative.Plotly
total_concentrations = data[pollutants].sum()
concentration_data = pd.DataFrame({
    "Pollutant": pollutants,
    "Concentration": total_concentrations
})

fig = px.pie(concentration_data, names="Pollutant", values="Concentration",
             title="Pollutant Concentrations in Delhi",
             hole=0.4, height=500,width=400,color_discrete_sequence=pollutant_colors)
fig.update_traces(textinfo="percent+label")
fig.update_layout(legend_title="Pollutant")
fig.show()
```



Now, let's have a look at the correlation between pollutants:

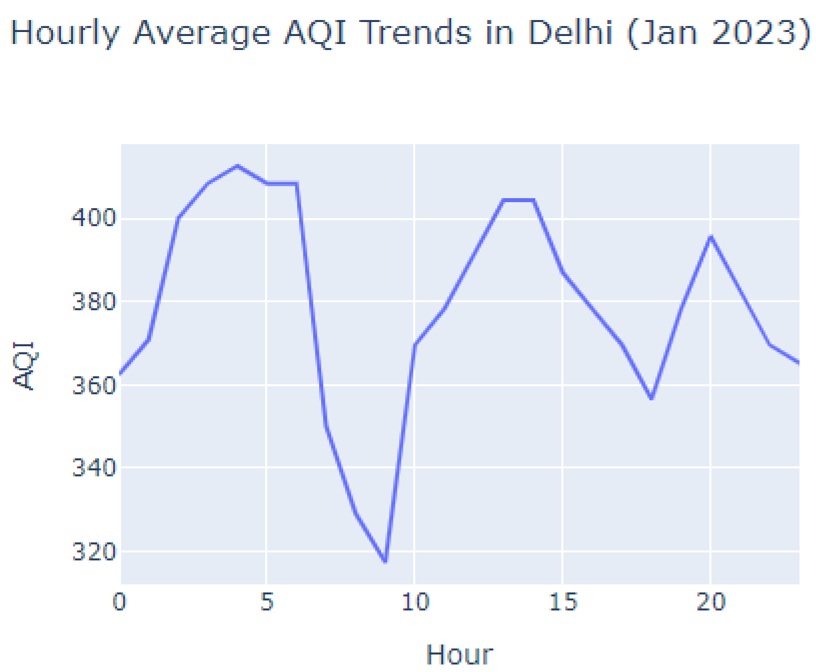
```
correlation_matrix = data[pollutants].corr()
fig = px.imshow(correlation_matrix,
                x=pollutants,
                y=pollutants,
                title="Correlation Between Pollutants")
fig.update_layout(
    title={
        'text': "Correlation Between Pollutants",
        'x': 0.5,
         }
)
fig.update_layout(
    coloraxis_colorbar=dict(
        x=0.6,
        y=0.5,)
)
fig.show()
```



The correlation matrix shows how different air pollutants are related, with correlation coefficients indicating the strength and direction of their relationships (ranging from -1 to 1). Pollutants like CO, NO, NO2, SO2, PM2.5, PM10, and NH3 have positive correlations, suggesting they likely originate from similar sources or follow comparable pollution trends. On the other hand, O3 shows a negative correlation with these pollutants, possibly because it acts as both a pollutant and a natural atmospheric cleaner.

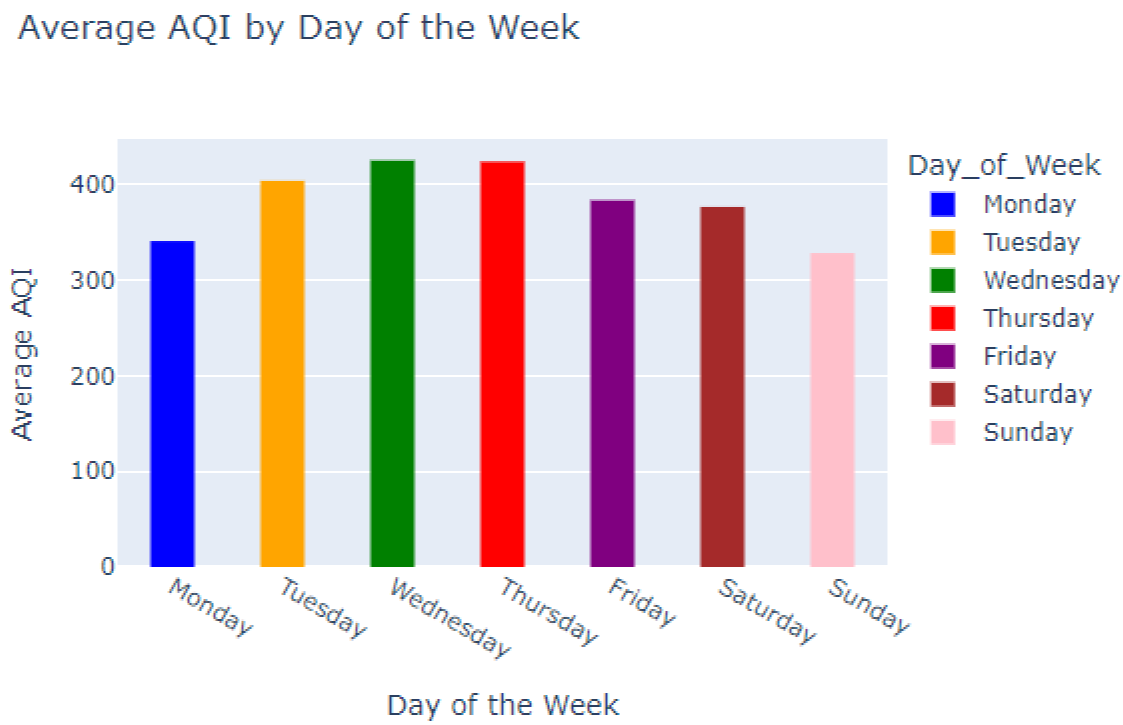
Now, let’s have a look at the hourly average trends of AQI in Delhi:

```
data['Hour'] = pd.to_datetime(data['date']).dt.hour
hourly_avg_aqi = data.groupby('Hour')['AQI'].mean().reset_index()
fig = px.line(hourly_avg_aqi, x='Hour', y='AQI',
              title='Hourly Average AQI Trends in Delhi (Jan 2023)')
fig.update_layout(width=500, height=400)
fig.show()
```



Now, let's have a look at the average AQI by day of the week in Delhi:

```
data['Day_of_Week'] = pd.to_datetime(data['date']).dt.day_name()
average_aqi_by_day = data.groupby('Day_of_Week')['AQI'].mean().reindex(
    ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
)
custom_colors = ['blue', 'orange', 'green', 'red', 'purple', 'brown', 'pink']
fig = px.bar(average_aqi_by_day, x=average_aqi_by_day.index, y='AQI',
             title='Average AQI by Day of the Week',
             color=average_aqi_by_day.index,
             color_discrete_sequence=custom_colors)
fig.update_traces(marker=dict(line=dict(width=0)), width=0.4)
fig.update_layout(width=600, height=400)
fig.update_xaxes(title="Day of the Week")
fig.update_yaxes(title="Average AQI")
fig.show()
```



Conclusion:

Air quality index (AQI) analysis is a crucial aspect of environmental data science that involves monitoring and analyzing air quality in a specific location. It aims to provide a numerical value representative of overall air quality, essential for public health and environmental management.