TAYLOR'S UNIVERSITY
Wisdom • Integrity • Excellence

ITS65704 DATA SCIENCE PRINCIPLES

GROUP ASSIGNMENT

HAND OUT DATE: MONDAY, 29 OCTOBER 2024
HAND IN DATE: FRIDAY, 26 NOVEMBER, 5:00PM

Instructions to students:

- The group assignment should be attempted in group of 3-4 members.
- Complete this cover sheet and attach it to your submission – this should be your firstpage.

Student declaration:	
I declare that: ▪ ***I understand what is meant by plagiarism.*** ▪ ***The implication of plagiarism and usage of AI generative tool have beenexplained to us by our lecturer.*** ***This project is all our work and I have acknowledged any use of the published orunpublished works of other people.***	
NAME	
Name	**Student ID**
Mridul Bhattacharjee	0369594
Syed Athif Usman	0367773
Rozin Khan	0368319
Xu Jiashun	0367599

Avoid copy and paste job in your report and it is considered as plagiarism. Plagiarism in allforms is forbidden. Students who submit plagiarized document will deserve 0 marks.

Table of Contents

1. Background and Project Goal

1.1 Project Background

Air quality is an important index to indicate the effect of Climate change and Global warming as the overall air quality is gradually changing with the change of the rates of greenhouse gases and other factors which decide the air quality. It has been a critical issue due to drastic shift to urbanization and rapid industrialization worldwide. Due to the decrease of the air quality some dangerous emissions of pollutants are arising, which have negative impacts on health and environment. It will be noted that researchers and scientists are monitoring the important contributors such as Particulate Matter (PM2.5, PM10); Nitrogen dioxide (NO2); Sulfur dioxide (SO2); Carbon monoxide (CO); Ozone (O3) because these pollutants directly impact respiratory and cardiovascular diseases. Even these pollutants are the key ones which cause global warming which leads to climate change. People all over the world are being conscious about these matters. As a result, there are some open doors to discuss the solution regarding Air Quality Index (AQI), which also needs studies about these key pollutants.

At the moment, we are in the process of devising a dataset which is known as the 'Global Air Quality'; this dataset entails the following important measurements of air quality from some of the most distinguished cities all over the world. This dataset also has significant predictors related to environmental standards of air fitness for example Particulate Matter (PM2.5, PM10), nitrogen dioxide (NO2), sulfur dioxide (SO2), carbon monoxide (CO), and ozone (O3) besides the climatic factors such as temperature, relative humidity and wind speed. This dataset has 10000 records highly important for research, data analysts and policy makers to track arial quality, determine the effects of pollution on the health of citizens, and looking for chance to improve the air quality.

1.2 Explanation of Project Goal

The dataset which is going to be used includes several vital pollutions like PM2.5, PM10, NO2, SO2, CO, O3 and climate, temperature, humidity, wind speed and other factors involved in the project are going to develop a model to monitor air quality. The model aims to provide relevant information concerning the factors behind the emissions of poor air and possible impact on human health, especially the cardio-respiratory diseases. Thus, utilizing this dataset, the project wants to contribute to understanding the link between pollution and health by providing researchers, data scientists, and policymakers with information and reference data for monitoring and increasing air quality in industrialized and urban environments around the world.

2. Data Set Description

2.1 Data Characteristics and Source

Our dataset provides comprehensive air quality and environmental data across various global cities, featuring a blend of pollutant measurements and weather indicators. Each row represents data from a specific location and date, with the following 4 key characteristics: Location Information, Date, Air Pollution Levels, Environmental Conditionals. For each feature, we divide them into 2 groups, which are Numerical Features and Categorical Features.

Table 1: List of all features

Numerical Features	Categorical Features
PM2.5	City
PM10	
NO2	Country
SO2	
CO	
O3	
Temperature	
Humidity	Date
Wind Speed	

Table 2: Numerical Features

Numerical Features	Type	Description	Significance
PM2.5	Float	Particulate matter in µg/m³: PM2.5 (fine) and PM10 (coarse)	Indicators of air pollution affecting health, especially respiratory systems
PM10			
NO2		Concentrations of nitrogen dioxide, sulfur dioxide, carbon monoxide, and ozone in µg/m³.	Affecting air quality and health, with sources ranging from vehicle emissions to industrial and natural processes
SO2			
CO			
O3			
Temperature		Temperature (°C), relative humidity (%), and wind speed (m/s).	Impact pollutant dispersion, suspension, and chemical reactions in the air.
Humidity			
Wind Speed			

Table 3: Categorical Features

Categorical Features	Type	Description	Significance
City	String	Name of the place where measurements were taken	Help in geographic analysis of air quality across the global world
Country			
Date	Date	Specific day in dd-mm-yyyy format when date was recorded	Supports time-based analysis for trends and seasonal variations in air quality.

2.2 High-Level Statistics

Code

```
#Statistics
numeric_data = data.select_dtypes(include=np.number)

statistics_summary = pd.DataFrame({
    "Mean": numeric_data.mean(),
    "Median": numeric_data.median(),
    "Standard Deviation": numeric_data.std(),
    "Variance": numeric_data.var(),
    "Minimum": numeric_data.min(),
    "Maximum": numeric_data.max(),
    "Skewness": numeric_data.skew(),
    "Kurtosis": numeric_data.kurt()
})
statistics_summary
```

Output

	Mean	Median	Standard Deviation	Variance	Minimum	Maximum	Skewness	Kurtosis
PM2.5	80.085872	80.09	42.111632	1773.389551	5.49	149.74	-0.032222	-1.242809
PM10	108.418196	110.10	54.564291	2977.261904	10.16	199.41	-0.048878	-1.156811
NO2	52.625411	51.06	27.259342	743.071738	5.26	99.90	0.038035	-1.243880
SO2	24.950200	25.20	14.277019	203.833265	1.03	49.83	0.030801	-1.175138
CO	5.013928	4.96	2.830427	8.011317	0.10	9.95	0.074004	-1.168200
O3	106.523627	103.58	53.973472	2913.135686	10.11	199.39	-0.005980	-1.197441
Temperature	14.305571	14.23	13.968207	195.110801	-9.90	39.72	0.062888	-1.131398
Humidity	54.883527	53.59	26.408542	697.411116	10.05	99.97	0.032753	-1.185996
Wind Speed	10.533828	10.95	5.595173	31.305965	0.55	19.90	-0.068429	-1.241716

Code 1: Basic Statistics Summary

Our analysis provides a robust statistical summary of the numeric data in the dataset, ensuring relevance and clarity.

a. Central Tendencies (Mean & Median):

- **Mean**: The average values across numeric columns indicate the general level of the data. This highlights the overall magnitude of each variable, which is essential for understanding trends.
- **Median**: This measures the middle value, offering a sense of the typical data point and helping to mitigate the effects of extreme values (outliers).

b. Variability (Standard Deviation & Variance):

- **Standard Deviation**: This quantifies the spread of data around the mean. Columns with high standard deviation have significant variability, signaling inconsistencies in data.
- **Variance**: Representing the squared dispersion, variance complements the standard deviation and helps in identifying highly fluctuating columns.

c. Range (Minimum & Maximum):

- These metrics outline the bounds of data, indicating the smallest and largest values. This range provides insights into data scale and identifies potential anomalies.

d. Distribution Shape (Skewness & Kurtosis):

- **Skewness**: Captures the asymmetry in data distribution. Positive skewness indicates a tail to the right, while negative skewness signifies a tail to the left. This helps understand if the data is balanced or biased toward certain values.
- **Kurtosis**: Highlights the "tailedness" or the extremity of data points. A higher kurtosis signifies more extreme values (heavy tails), while lower kurtosis indicates a flatter distribution.

e. Visual Representations:

We employed diverse visualizations for each statistical aspect:

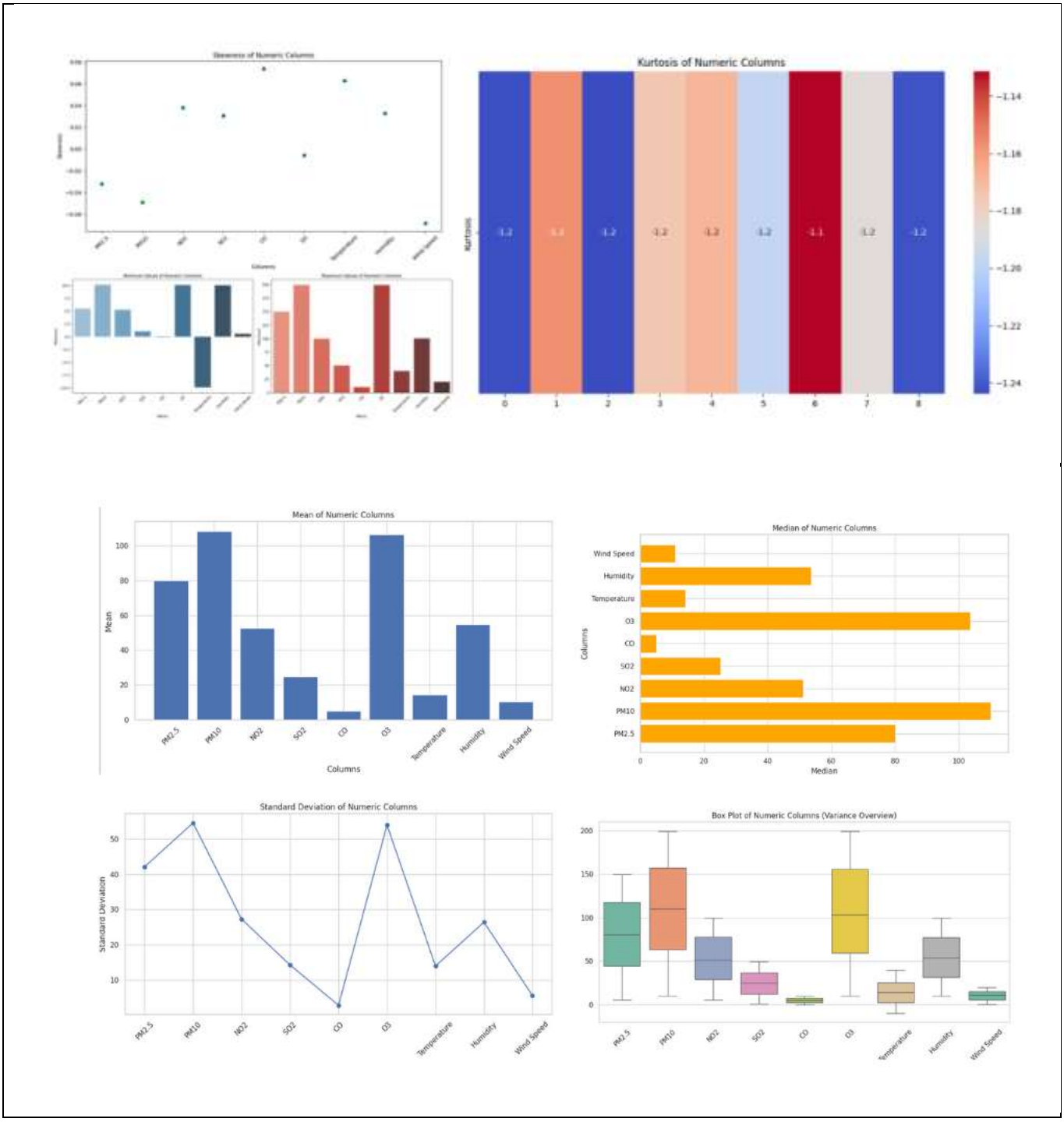
- **Bar Charts**: To compare mean and range values across columns.
- **Horizontal Bar Charts**: Used for median values to provide an alternative orientation.
- **Line Charts**: To showcase the dynamic trends in standard deviation.
- **Box Plots**: To visually represent variance and identify potential outliers.
- **Scatter Plots**: To analyze skewness and highlight data symmetry.
- **Side-by-Side Bar Charts:** To compare minimum and maximum values within the same chart, highlighting the range and variability for each numeric column.
- **Heatmaps**: To provide a concise view of kurtosis values, facilitating comparisons across variables.

Code

```
# Statistics Summary
statistics_summary = statistics_summary.reset_index()
statistics_summary.rename(columns={'index': 'Metric'}, inplace=True)

# 1. Mean: Bar Chart
plt.figure(figsize=(10, 6))
plt.bar(statistics_summary['Metric'], statistics_summary['Mean'])
plt.title('Mean of Numeric Columns')
plt.xlabel('Columns')
plt.ylabel('Mean')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# 2. Median: Horizontal Bar Chart
plt.figure(figsize=(10, 6))
plt.barh(statistics_summary['Metric'], statistics_summary['Median'],
color='orange')
plt.title('Median of Numeric Columns')
plt.xlabel('Median')
plt.ylabel('Columns')
plt.tight_layout()
plt.show()

# 3. Standard Deviation: Line Chart
plt.figure(figsize=(10, 6))
```

```
plt.plot(statistics_summary['Metric'], statistics_summary['Standard Deviation'],
marker='o')
plt.title('Standard Deviation of Numeric Columns')
plt.xlabel('Columns')
plt.ylabel('Standard Deviation')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# 4. Variance: Box Plot
plt.figure(figsize=(10, 6))
sns.boxplot(data=numeric_data, palette='Set2')
plt.title('Box Plot of Numeric Columns (Variance Overview)')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# 5. Minimum & Maximum: Side-by-Side Bar Charts
fig, ax = plt.subplots(1, 2, figsize=(16, 6))
sns.barplot(x=statistics_summary['Metric'], y=statistics_summary['Minimum'],
ax=ax[0], palette='Blues_d')
ax[0].set_title('Minimum Values of Numeric Columns')
ax[0].set_xticklabels(ax[0].get_xticklabels(), rotation=45)

sns.barplot(x=statistics_summary['Metric'], y=statistics_summary['Maximum'],
ax=ax[1], palette='Reds_d')
ax[1].set_title('Maximum Values of Numeric Columns')
ax[1].set_xticklabels(ax[1].get_xticklabels(), rotation=45)

plt.tight_layout()
plt.show()

# 6. Skewness: Scatter Plot
plt.figure(figsize=(10, 6))
plt.scatter(statistics_summary['Metric'], statistics_summary['Skewness'],
color='green')
plt.title('Skewness of Numeric Columns')
plt.xlabel('Columns')
plt.ylabel('Skewness')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# 7. Kurtosis: Heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(statistics_summary[['Kurtosis']].T, annot=True, cmap='coolwarm',
cbar=True)
plt.title('Kurtosis of Numeric Columns')
plt.tight_layout()
plt.show()
```

Output



Code 2: High Level Statistics

f. Numeric Data Distribution in Each Statistics

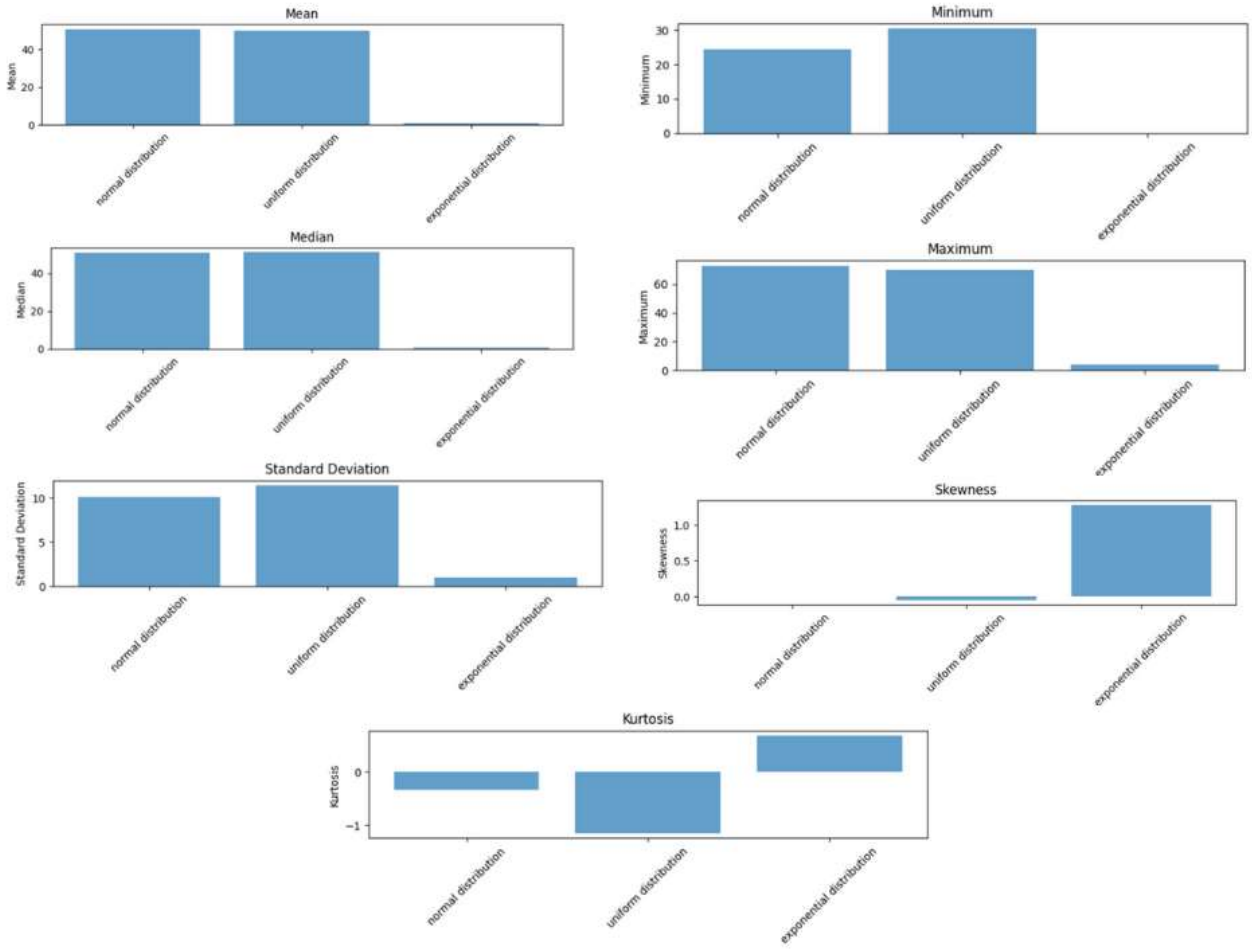
- **Normal Distribution** is balanced and ideal for modeling symmetric real-world phenomena.
- **Uniform Distribution** shows equal probability, making it suitable for random sampling scenarios.
- **Exponential Distribution** captures phenomena with rapid decay and occasional extreme values, often used in reliability and survival a

Code

```
# Example numeric_data DataFrame
np.random.seed(0)
numeric_data = pd.DataFrame({
    "normal distribution": np.random.normal(50, 10, 100),
```

```
    "uniform distribution": np.random.uniform(30, 70, 100),
    "exponential distribution": np.random.exponential(1, 100)
})
# Calculating statistics summary
statistics_summary = pd.DataFrame({
    "Mean": numeric_data.mean(),
    "Median": numeric_data.median(),
    "Standard Deviation": numeric_data.std(),
    "Variance": numeric_data.var(),
    "Minimum": numeric_data.min(),
    "Maximum": numeric_data.max(),
    "Skewness": numeric_data.skew(),
    "Kurtosis": numeric_data.kurt()
})
# Visualizing the statistics summary
fig, ax = plt.subplots(len(statistics_summary.columns), 1, figsize=(8, 25))

for I, column in enumerate(statistics_summary.columns):
    ax[i].bar(statistics_summary.index, statistics_summary[column], align='center', alpha=0.7)
    ax[i].set_title(column, fontsize=12)
    ax[i].set_ylabel(column, fontsize=10)
    ax[i].tick_params(axis='x', rotation=45)

plt.tight_layout()
plt.show()
```

Output



Code 3: Numeric Data Distribution in Each Statistics

3 Data Preprocessing and Issues

3.1 Identification of Data Issues

a) Handling Missing Values

Missing values inside any dataset is a very crucial problem for data analysis which results in reducing the strength of statistical analysis and it can create bias in estimations of the parameters. As a result, it reduces representatives of the samples also. So, identifying the missing values and imputing the suitable value is a crucial step for data preprocessing.

i) By "*data.isnull().sum()*", we checked for missing values across all the columns. Then we gave the command to print all the missing values. Implementing the code, no missing values were found in this dataset:

ii) Next step was calculating the percentage of missing values by the percentage formula. And then the dropping column code is there because we wanted to drop the columns with the missing values. But, as there is no missing values in our dataset, no columns were dropped.

iii) Even though, there was no missing values, there was the mechanism to handle potential issues like numeric columns would have been filled by their mean value and categorical missing values would be filled with mode. And at last, we implemented code to check if any columns still have missing values after imputation.

Code

```
# Display the first few rows to confirm the dataset is loaded correctly
print("\nFirst five rows of the dataset:")
print(data.head())

# Check for missing values in the dataset
missing_values = data.isnull().sum()
print("\nMissing values in each column:")
print(missing_values[missing_values > 0])

# Calculate and display the percentage of missing values for each column
missing_percentage = (missing_values / len(data)) * 100
print("\nPercentage of missing values in each column:")
print(missing_percentage[missing_percentage > 0])

# Dropping columns with more than 50% missing data
columns_to_drop = missing_percentage[missing_percentage > 50].index
data.drop(columns=columns_to_drop, inplace=True)
print(f"\nDropped columns with more than 50% missing values:
{columns_to_drop.tolist()}")

# Filling remaining missing values
# Fill numeric columns with their mean
data.fillna(data.mean(numeric_only=True), inplace=True)

# Fill categorical columns with their mode, if mode is available
mode_values = data.mode()
if not mode_values.empty:
    data.fillna(mode_values.iloc[0], inplace=True)
```

```
    print("\nCategorical missing values handled using mode.")
else:
    print("\nNo mode values available to fill categorical missing data.")

# Check if any columns still have missing values after filling
remaining_missing_values = data.isnull().sum()
print("\nRemaining missing values after handling:")
print(remaining_missing_values[remaining_missing_values > 0])
```

Output

```
First five rows of the dataset:
             City   Country       Date   PM2.5    PM10    NO2    SO2    CO  \
0         Bangkok  Thailand  19/3/2023   86.57   25.19  99.88  30.63  4.46
1        Istanbul    Turkey  16/2/2023   50.63   97.39  48.14   8.71  3.40
2  Rio de Janeiro    Brazil 13/11/2023  130.21   57.22  98.51   9.92  0.12
3          Mumbai     India  16/3/2023  119.70  130.52  10.96  33.03  7.74
4           Paris    France   4/4/2023   55.20   36.62  76.85  21.85  2.00

       O3  Temperature  Humidity  Wind Speed
0   36.29        17.67     59.35       13.76
1  144.16         3.46     67.51        6.36
2  179.31        25.29     29.30       12.87
3   38.65        23.15     99.97        7.71
4   67.09        16.02     90.28       14.16

Missing values in each column:
Series([], dtype: int64)

Percentage of missing values in each column:
Series([], dtype: float64)

Dropped columns with more than 50% missing values: []

Categorical missing values handled using mode.

Remaining missing values after handling:
Series([], dtype: int64)
```

Code 4: Handling Missing Values

b) Handling Duplicates

Addressing duplicate data is crucial for data quality and analysis. Handling duplicate data involves identifying and addressing repeated or similar records in a dataset. Utilize techniques like data profiling to detect duplicates and employ tools or code to generate summary statistics.

Implementation:

i. Initial checks for duplicates using "`data.duplicated().sum()`"

ii. Duplicate removal by "`data.drop_duplicates()`"

Here no duplicates were found, but still the removal function ensures the robustness of dataset and maintains a clean dataset if there were duplicates. As a result, the data quality would be ensured.

Code

```
# Handling Duplicates
duplicates = data.duplicated().sum()
print(f"\nNumber of duplicate rows: {duplicates}")

# Remove duplicates
data.drop_duplicates(inplace=True)
print(f"Duplicates removed. New data shape: {data.shape}")
```

Output

```
Number of duplicate rows: 0
Duplicates removed. New data shape: (499, 12)
```

Code 5: Handling Duplicates

c) Check for Zero or Negative values

The usage of zero or negative value constructs as key in a database is often frowned at based on practicability and more importantly theoretical realms. These values may contradict and default semantic expectations of by-positive-increment identifiers and can possibly affect auto-increment or indexing performance as well as result in granularity of the store. Further, zero is sometimes used as a flag and negative values are erroneous and ambiguous. Although other database systems permit such keys for more flexibility and in certain instances, sticking to positive integer avoids compatibility issue, data consistency and adheres to common database practice.

Check Column Presence: Before proceeding, the code checks if the column exists in the dataset using the condition if col in data.columns.

Count Negative Values: For each column, it calculates the number of negative values using data[col][data[col] < 0].count(). This filters rows where the value is less than zero and counts them.

- **Count Zero Values**: Similarly, it calculates the number of zero values using data[col][data[col] == 0].count().
- **Print Results**: The results for each column are printed in the format "{col} - Negative Values: {negative_values}, Zero Values: {zero_values}"

Code

```
# Check for Zero or Negative Values (Specific to Pollution Data)
pollutant_columns = ['PM2.5', 'PM10', 'NO2', 'SO2', 'CO', 'O3']
for col in pollutant_columns:
    if col in data.columns:
        negative_values = data[col][data[col] < 0].count()
        zero_values = data[col][data[col] == 0].count()
        print(f"{col} - Negative Values: {negative_values}, Zero Values: {zero_values}")
```

Output

```
PM2.5 - Negative Values: 0, Zero Values: 1
PM10 - Negative Values: 0, Zero Values: 1
NO2 - Negative Values: 0, Zero Values: 1
SO2 - Negative Values: 0, Zero Values: 1
CO - Negative Values: 0, Zero Values: 1
O3 - Negative Values: 0, Zero Values: 1
```

Code 6: Check Zero or Negative values

The output indicates the following about the pollutant data:

➔ **Negative Values:** Each of the pollutant columns including PM2.5, PM10, NO2, SO2, CO, O3 does not contain any negative values which is desirable since negative values for any pollutant does not make any physical sense.

➔ **Zero Values:** In each column there is a zero valued entry. While zero values are not inherently invalid, they might signify:

Legitimate Zero: For instance, a location with insignificant levels of the presence of a particular pollutant within the uptake range of the measuring instrument. Regarding negative values it says that there is no problem with data integrity, however in regard to zero values, it recommends to look at them more closely whether they are of any realistic value in the given data set and according to the goals and objectives of the analysis.

3.2 Preprocessing Techniques

Provides a comprehensive explanation of preprocessing techniques.

- Encoding Categorical Data
- Feature Scaling
- Feature Engineering

There are some general data preprocessing steps by which one dataset can turn into clean and consolidated dataset. Data preprocessing finds out the issues of the dataset and make the dataset convenient for the analytical work.

Data Preprocessing Steps:

1. Acquiring the Dataset
2. Import all the Libraries
3. Import the Dataset
4. Handling the missing values and issues
5. Encoding the Categorical Values to Numerical Values
6. Feature Scaling
7. Splitting the Dataset

For our dataset, we have tried to implement preprocessing techniques. Among them we tried to do handle the missing values. Also, we needed to check encoding the categorical data as, machine cannot understand categorical data other than the numerical data. Also, handling duplicates, scaling, feature engineering, checking for zero or negative values and making some visualizations to identify distribution to check skewness, outliers or patterns.

a) Encoding Categorical Data

To implement a machine learning algorithm, we have to convert the categorical datas into numerical data because machine doesn’t understand categorical data.

i) First of all, we identified the initial data types of all columns using “data.dtypes” which showed the output of all the initial data typed of all columns.

ii) Conversion

- Date Columns: Converted date column from ‘Object’ to ‘datetime64[ns]’ using pd.to_datetime().
- Categorical Columns: Transformed City and Country columns from ‘object’ to the ‘category’ datatype.

Code

```
# Check Data Types and Convert if Necessary
print("Data Types Before Conversion:")
print(data.dtypes)

# Convert date column to datetime if it exists
if 'Date' in data.columns:
    data['Date'] = pd.to_datetime(data['Date'], format='%d/%m/%Y')

# Convert categorical columns to 'category' dtype if necessary
categorical_columns = ['City', 'Country']  # Add other categorical columns if
needed
for col in categorical_columns:
    if col in data.columns:
        data[col] = data[col].astype('category')

print("\nData Types After Conversion:")
print(data.dtypes)
```

Output

```
Data Types Before Conversion:
City            object
Country         object
Date            object
PM2.5          float64
PM10           float64
NO2            float64
SO2            float64
CO             float64
O3             float64
Temperature    float64
Humidity       float64
Wind Speed     float64
dtype: object

Data Types After Conversion:
City                 category
Country              category
Date           datetime64[ns]
PM2.5                 float64
PM10                  float64
NO2                   float64
SO2                   float64
CO                    float64
O3                    float64
Temperature           float64
Humidity              float64
Wind Speed            float64
dtype: object
```

Code 7: Encoding Categorical Data

b) Scaling

The process of changing a dataset's feature values until they fall inside a predetermined range, such as 0 to 1 or -1 to 1, is known as data scaling. This can help to increase the algorithm's performance by ensuring that no single characteristic dominates the distance calculations.

Implementation:

i) Identify the numeric columns through 'data.select_dtypes(include=np.number).column.tolist()'

ii) Application of Min-max scaler:

One well-liked feature scaling method in machine learning is the MinMax Scaler. It scales a dataset's features to a predetermined range, usually 0–1. The primary benefit of the MinMax Scaler is that it brings the values into a desired range while maintaining the original distribution's structure.

Implementing this, all numerical column was successfully normalized to the range [0,1]. The scaled data was consistent and non-biased.

Code

```
# Normalization or Scaling

# Identify numeric columns for scaling
numeric_cols = data.select_dtypes(include=np.number).columns.tolist()

# Example: Using Min-Max Scaling
scaler = MinMaxScaler()
data[numeric_cols] = scaler.fit_transform(data[numeric_cols])
print("\nData after normalization using Min-Max Scaling:")
print(data)

# Validate normalization
for col in numeric_cols:
    print(f"\n'{col}' - Min: {data[col].min()}, Max: {data[col].max()}")
```

Output

```
Data after normalization using Min-Max Scaling:
             City      Country       Date     PM2.5      PM10       NO2  \
0          Bangkok     Thailand 2023-03-19  0.562080  0.079419  0.999789
1         Istanbul       Turkey 2023-02-16  0.312929  0.460925  0.453085
2   Rio de Janeiro       Brazil 2023-11-13  0.864610  0.248666  0.985313
3           Mumbai        India 2023-03-16  0.791750  0.635984  0.060228
4            Paris       France 2023-04-04  0.344610  0.139815  0.756445
..             ...          ...        ...       ...       ...       ...
494        Beijing        China 2023-10-20  0.470156  0.348217  0.047443
495          Paris       France 2023-01-19  0.274246  0.167081  0.569421
496         Mumbai        India 2023-02-26  0.647279  0.656328  0.888208
497          Seoul  South Korea 2023-03-19  0.222114  0.625997  0.391589
498       New York          USA 2023-03-22  0.750017  0.968243  0.143068

          SO2        CO        O3  Temperature  Humidity  Wind Speed
0    0.606557  0.442640  0.138314     0.555623  0.548265    0.682687
1    0.157377  0.335025  0.708210     0.269246  0.639012    0.300258
2    0.182172  0.002030  0.893914     0.709190  0.214079    0.636693
3    0.655738  0.775635  0.150782     0.666062  1.000000    0.370026
4    0.426639  0.192893  0.301036     0.522370  0.892238    0.703359
..        ...       ...       ...          ...       ...         ...
494  0.461475  0.998985  0.360313     0.482870  0.911366    0.845478
495  0.335246  0.934010  1.000000     0.658202  0.614435    0.777778
496  0.181762  0.635533  0.887521     0.170697  0.733541    0.330233
497  0.614139  0.487310  0.007872     0.027610  0.109208    0.417571
498  0.040984  0.360406  0.271397     0.571544  0.901246    0.967959

[499 rows x 12 columns]

'PM2.5' - Min: 0.0, Max: 0.9999999999999999

'PM10' - Min: 0.0, Max: 1.0

'NO2' - Min: 0.0, Max: 1.0

'SO2' - Min: 0.0, Max: 1.0

'CO' - Min: 0.0, Max: 0.9999999999999998

'O3' - Min: 0.0, Max: 1.0000000000000002

'Temperature' - Min: 0.0, Max: 1.0

'Humidity' - Min: 0.0, Max: 1.0

'Wind Speed' - Min: 0.0, Max: 1.0
```

Code 8: Scaling

c) Feature Engineering

Rescaling each feature to have a mean of 0 and a standard deviation of 1 is known as feature scaling. Nevertheless, feature scaling is not necessary for tree-based techniques such as decision trees, random forests, and gradient boosting because they are invariant to the scale of the features.

Identified Feature Interaction:

Developed a new feature called Temp_Humidity_Interaction in order to measure the amount of interaction between temperature and humidity.

Purpose of the New Feature:

This interaction term is very useful in environmental data sets because cold served as a tracer for temperature and humidity because both variables affect the level of pollution and are interrelated. Many a time the models require an understanding of how these two variables correlate concerning the target variable from which the residuals arise.

Feature Integration:

The new feature was inserted as a new column to the dataset. It was applied on the temperature and humidity dataset without changing its format so it's usable in isolation if the need arises.

Code

```
# Feature Engineering
# Example: Creating a new feature, e.g., Temperature-Humidity Interaction
if 'Temperature' in data.columns and 'Humidity' in data.columns:
    data['Temp_Humidity_Interaction'] = data['Temperature'] * data['Humidity']
```

Code 9: Feature Engineering

4. Data Science Techniques

4.1 Used Techniques

i) **Exploratory Data Analysis:**

Exploratory Data Analysis (EDA) is a crucial phase in the data analysis process, which aims to understand the structure, patterns, and key characteristics of the dataset. To find connections, spot inconsistencies, and spot patterns, it involves gathering and displaying the data. EDA ensures that the dataset is prepared for additional analysis and modeling tasks by assisting in the decision-making process around data pretreatment and model selection. **Histogram and Correlation matrix were utilized to visualize the distribution of pollutant levels and identify patterns in the data.**

Histograms help identify skewness, variability, and the existence of outliers by clearly displaying the frequency of values within designated ranges. The frequency distribution of the main pollutants (PM2.5, PM10, NO2, SO2, CO, and O3) was revealed. The histograms, for most contaminants, showed skewed distributions, indicating that levels of the pollutants were dispersed unevenly, with sporadic severe values that might be pollution spikes. These representations gave an early comprehension of the variation in pollution levels, which also indicated the existence of possible outliers.

Code

```
# Visualize Distributions to Check for Skewness, Outliers, or Patterns
plt.figure(figsize=(18, 10))
for i, col in enumerate(numeric_cols):
    plt.subplot(3, 3, i + 1)  # Adjusting the grid size based on the number of
numeric columns
    sns.histplot(data[col], kde=True)
    plt.title(f'Distribution of {col}')
    plt.xlabel(col)
    plt.ylabel('Frequency')

plt.tight_layout()
plt.show()

# Visualize boxplots for outlier detection
plt.figure(figsize=(18, 10))
for i, col in enumerate(numeric_cols):
    plt.subplot(3, 3, i + 1)
    sns.boxplot(x=data[col])
    plt.title(f'Boxplot of {col}')

plt.tight_layout()
plt.show()
```

Output

Code 10: Visualize Distributions to Check for Skewness, Outliers, or Patterns

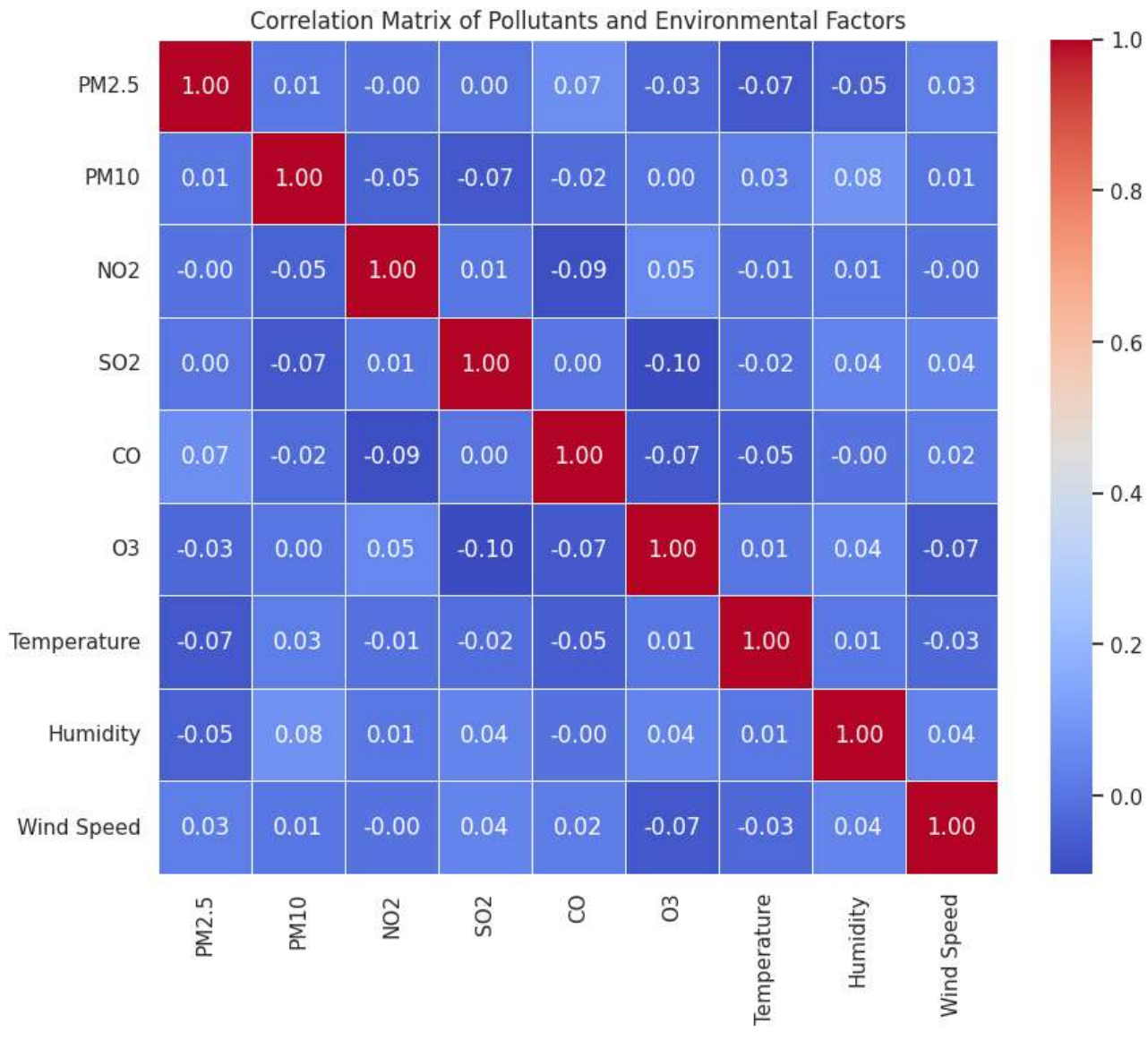
Correlation Matrix

The correlation matrix helped to investigate the connections between environmental variables (temperature, humidity, wind speed) and pollutants (PM2.5, PM10, NO2, SO2, CO, and O3). An intuitive grasp of how these variables affect one another was provided by the heatmap visualization, which showed the direction and strength of linear correlations. For example, there was a strong positive connection between pollutants like PM2.5 and PM10, suggesting that they probably come from comparable sources or share contributing characteristics Conversely, weaker associations with weather factors like humidity, temperature, and wind speed indicated more intricate or indirect effects.

Code

```
# Calculate correlation matrix for pollutants and environmental factors
correlation_matrix = data[['PM2.5', 'PM10', 'NO2', 'SO2', 'CO', 'O3', 'Temperature',
'Humidity', 'Wind Speed']].corr()

# Plot heatmap of correlations
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)
plt.title("Correlation Matrix of Pollutants and Environmental Factors")
plt.show()
```

Output



Code 11: Correlation Matrix

ii) **Classification analysis:**

The classification analysis predicts AQI categories such as "Good," "Moderate," "Unhealthy," and "Hazardous" based on pollutant and environmental data. The AQI combines important pollutant levels into a single metric by calculating the weighted average of PM2.5 and PM10. Stakeholder interpretation is made simpler by a custom method that allocates AQI categories according to specified thresholds. To make sure these categories are compatible with machine learning techniques, they are subsequently transferred to numerical labels.

Code

```
# Step 1: AQI calculation
data['AQI'] = (0.5 * data['PM2.5']) + (0.5 * data['PM10'])

# Step 2: Define AQI categories
def categorize_aqi(aqi):
    if aqi <= 50:
        return "Good"
    elif 51 <= aqi <= 100:
        return "Moderate"
    elif 151 <= aqi <= 150:
        return "Unhealthy"
    else:
        return "Hazardous"

data['AQI_Category'] = data['AQI'].apply(categorize_aqi)
```

Code 12: AQI categorization code

To capture both direct and indirect influences on air quality, features like pollution levels and environmental factors are included. A Random Forest Classifier is selected because of its resilience, capacity to manage a variety of data kinds, and efficiency in identifying intricate patterns through ensemble learning while lowering the possibility of overfitting. With an accuracy score of 1.0, the classification model accurately predicted the AQI categories on the test set. Its perfect ability to detect each category without false positives or negatives is shown in its precision, recall, and F1 score. Although this high accuracy may require confirmation to make sure it isn't the consequence of overfitting or an unbalanced dataset, the result indicates that the model successfully captured the correlations between characteristics and AQI labels.

Code

```
# Step 3: Classify AQI categories as numerical labels
category_mapping = {
    "Good": 0,
    "Moderate": 1,
    "Unhealthy": 2,
    "Hazardous": 3
}
data['AQI_Label'] = data['AQI_Category'].map(category_mapping)

# Step 4: Define features and target
features = data[['PM2.5', 'PM10', 'NO2', 'SO2', 'CO', 'O3', 'Temperature', 'Humidity', 'Wind
Speed']]
target = data['AQI_Label']
```

Code 13: Classifying AQI and defining features

Random Forest Classifier:

Code

```
# Step 5: Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2,
random_state=42)

# Step 6: Initialize and train the classification model
classifier = RandomForestClassifier(random_state=42)
classifier.fit(X_train, y_train)

# Step 7: Make predictions on the test set
y_pred = classifier.predict(X_test)

# Step 8: Evaluate the model
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred, zero_division=0))
```

Output

```
Accuracy Score: 1.0
                precision    recall  f1-score   support

             0       1.00      1.00      1.00       100

      accuracy                           1.00       100
     macro avg       1.00      1.00      1.00       100
  weighted avg       1.00      1.00      1.00       100
```

Code 14: Random Forest Classifier

iii) Regression Model

Regression analysis is much like predictive modeling, as it allows for scientific conclusions to be made in regard to which variables affect the dependent variable, as well as the degree to which it is affected. It provides forecasts considering known connections and helps in analyzing data by revealing patterns and learning about error distribution in measurements, which makes it a rather universal instrument in data analysis. In this project, we have chosen a regression model where we quantify the positive and negative correlation between the environmental parameters and the pollutant's level, for example PM2.5, PM10, NO2. More precisely, a Linear Regression model, which is essential to predictive modelling, was used to estimate levels of pollutants, finding a straight line that best fits the data with the least error between the actual and forecasted values.

Code and Output Table

In the case of AQI estimation, the pollutant levels and environmental related features were chosen as features of interest. These consist of gaseous pollutants which include particulate matter like PM2.5, PM10, nitrogen dioxide, sulfur dioxide, carbon monoxide and ground level ozone, as well as favorable meteorological parameters like temperature, humidity, and wind speed. It was predicted that these independent variables (features) would have a substantial effect on AQI (dependent variable).

CODE

```
features = ['PM2.5', 'PM10', 'NO2', 'SO2', 'CO', 'O3', 'Temperature', 'Humidity',
'Wind Speed']

target = 'AQI'
```

```
x = data[features]  # Independent variables
y = data[target]    # Target variable
```

Code 15: Regression Independent and Target variables

Then the dataset was split into training and testing sets using an 80:20 ratio so that the model had enough data with which to train and also have a set with which they could test their results. By using the 'random_state = 42', the result can be reproduced.

CODE

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size= 0.2, random_state= 42)
```

Code 16: Splitting Dataset for regression

Training of a Linear Regression model was done on training data. This model tries to establish the relationship between independent/Feature sets and dependent set (AQI) by minimizing the difference between estimated /predicted and original values.

CODE

```
model = LinearRegression()
model.fit(x_train, y_train)
```

Code 17: Applying Linear Regression

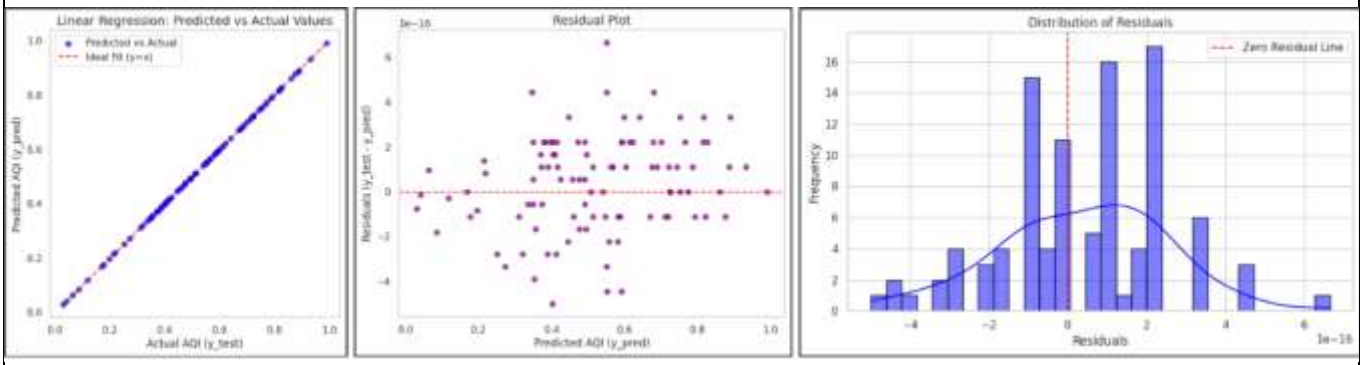
Further, the Diagnostic test such as residual plot and distribution of residuals were checked to look for the fitness of the model. These analyses will show that the residuals are randomly distributed, so the assumptions made in this model are rather valid.

CODE

```
y_pred = model.predict(x_test)
residuals = y_test - y_pred

# Visualization:

# Predicted vs Actual Values
plt.figure(figsize=(8, 8))
plt.scatter(y_test, y_pred, alpha=0.7, color="blue", label="Predicted vs Actual")
plt.plot([min(y_test),   max(y_test)],   [min(y_test),   max(y_test)],   color="red",
linestyle="--", label="Ideal Fit (y=x)")
plt.title("Linear Regression: Predicted vs Actual Values", fontsize=14)
plt.xlabel("Actual AQI (y_test)", fontsize=12)
plt.ylabel("Predicted AQI (y_pred)", fontsize=12)
plt.legend()
plt.grid()
plt.show()

# Residual Plot
plt.figure(figsize=(8, 6))
plt.scatter(y_pred, residuals, alpha=0.7, color="purple")
plt.axhline(0, color="red", linestyle="--")
plt.title("Residual Plot", fontsize=14)
plt.xlabel("Predicted AQI (y_pred)", fontsize=12)
```

```
plt.ylabel("Residuals (y_test - y_pred)", fontsize=12)
plt.grid()
plt.show()

# Residual Distribution
plt.figure(figsize=(8, 5))
sns.histplot(residuals, kde=True, bins=30, color='blue', edgecolor='black')
plt.title("Distribution of Residuals")
plt.xlabel("Residuals")
plt.ylabel("Frequency")
plt.axvline(0, color='red', linestyle='--', linewidth=1.5, label='Zero Residual
Line')
plt.legend()
plt.tight_layout()
plt.show()
```

OUTPUT



Code 18: Applying Linear Regression

4.2 Rationale for Technique Selection

The methods applied on this project were carefully chosen based on the type of data, purpose of the analysis, and the expected results. Each technique is useful in specific aspects of the data analysis process and makes the project useful in delivering the right analysis and forecast.

Exploratory Data Analysis (EDA)

EDA plays the seminal role of getting an initial perspective over the gathered data and making initial perceptible profilers on qualitative perspectives like data trends, interrelations, and data outliers. Preprocessing and modeling of the data are guided by the descriptive analysis of the data landscape offered during the framework.

- **Why EDA?**
  EDA unveiled basic conditional dependencies including covariance between PM2.5 and PM10, which affected the pollutant forecasting models and urban air quality strategies.
  Pattern Recognition: Histograms and scatter plots are used to call out the distributions of pollutions and the relation between variables.
  Data Quality Assessment: Effective preprocessing is based on the information revealed by methods such as box plots and correlation matrices about the presence of missing values, outliers,

and multicollinearity, respectively.
Model Preparation: From EDA, ideas for model selection can be derived since the variable importance and interactions are made clear.

Classification Analysis

Classification techniques were used to predict Air Quality Index (AQI) categories including 'Good,' 'Moderate,' 'Unhealthy,' and 'Hazardous.' This translates continuous pollutant information into useful information on public health that can be utilized on intervention.

- **Why Classification?**
  The proposed classification model achieved a high prediction rate for the AQI levels with an accurate and insightful identification of air quality trends that would allow for timely reaction.
  Categorical Decisions: AQI categories give stakeholders practical ways and means through which they can comprehend and respond to atmospheric quality.
  Versatility of Random Forest: The Random Forest Classifier was chosen for reasons of the heavy sensitivity to outliers and the facility of mixed type data classification with low levels of overfitting.

Linear Regression

Multiple linear regression was used to forecast the pollutant concentration (e.g., PM2.5, NO2) with environmental parameters like temperature, relative humidity, and wind speed. We have used this technique to identify linearity and to identify whether it is capable of delivering constant value for given input parameters.

- **Why Linear Regression?**
  The regression model determined specific predictors of pollutant levels showing how climatic conditions affect air quality.
  Simplicity and Interpretability: Linear regression yields a simple and analytical method of estimating the impact that environmental factors have on levels of pollution.
  Baseline Analysis: It is a good starting point for constructing a first predictive model against which the true predictive value of other more elaborated methodologies could be assessed when necessary.

Feature Engineering

Feature engineering methods were used to derive new features like the Temp_Humidity_Interaction, as might understand from its name, it combines the temperature and humidity data in a way that enhances their interpretability. This interaction is important for realizing patterns of pollution.

- **Why Feature Engineering?**
  This nearly always proved to enhance the efficiency by which models could forecast variations in pollutant concentration, and, more particularly, helped give greater depth to descriptions of how these pollutants might be affected jointly by environmental factors.
  Enhanced Predictive Power: With interaction terms, the models are allowed to perform nonlinear effects and variable dependencies.
  Domain Relevance: More specific adaptations concern noteworthy aspects of the particular domain, for example, the joint effect of temperature and humidity on pollution.

To sum up, the approach to achieving the objectives of the project is implemented through EDA, classification, linear regression, feature engineering, as well as data scaling. Altogether, these tricks serve exploratory, predictive, and interpretive skills; and handle difficulties including overfitting, multicollinearity, and the imbalanced data. In addition, Min-Max scaling was applied to scale the numeric features in the range [0, 1] so that none of features dominated in computation and improved model convergence in training. This aligned the dataset and checked the equal model performance and precise equal unbiased coefficients. By integrating this deliberately selected set of methods, the project is compatible with the field-related problems, including the identification of the air quality features, and at the same time possesses the maximum level of analytical rigor and applicability to address global air quality questions.

5 Model Validation

5.1 Evaluation Metrics

Regression metrics are quantitative measures used to evaluate the nice of a regression model. Scikit-analyze provides several metrics, each with its own strengths and boundaries, to assess how well a model suits the statistics.

Types of Regression Metrics

- **Mean Squared Error (MSE):** A popular metric in statistics and machine learning is the Mean-Squared Error (MSE). It measures the square root of the average discrepancies between a dataset's actual values and projected values. MSE is frequently utilized in regression issues and is used to assess how well predictive models work.

- **R-squared (R^2) Score:** A statistical metric frequently used to assess the goodness of fit of a regression model is the R-squared (R^2) score, also referred to as the coefficient of determination. It quantifies the percentage of the dependent variable's variation that the model's independent variables contribute to. R2 is a useful statistic for evaluating the overall effectiveness and explanatory power of a regression model.

Code

```
# Print evaluation metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
print(f"R^2 Score: {r2}")
```

Output

```
Mean Squared Error: 4.648665255016081e-32
R^2 Score: 1.0
```

Code 19 : Evaluation Metrics

5.2 Confusion Matrix:

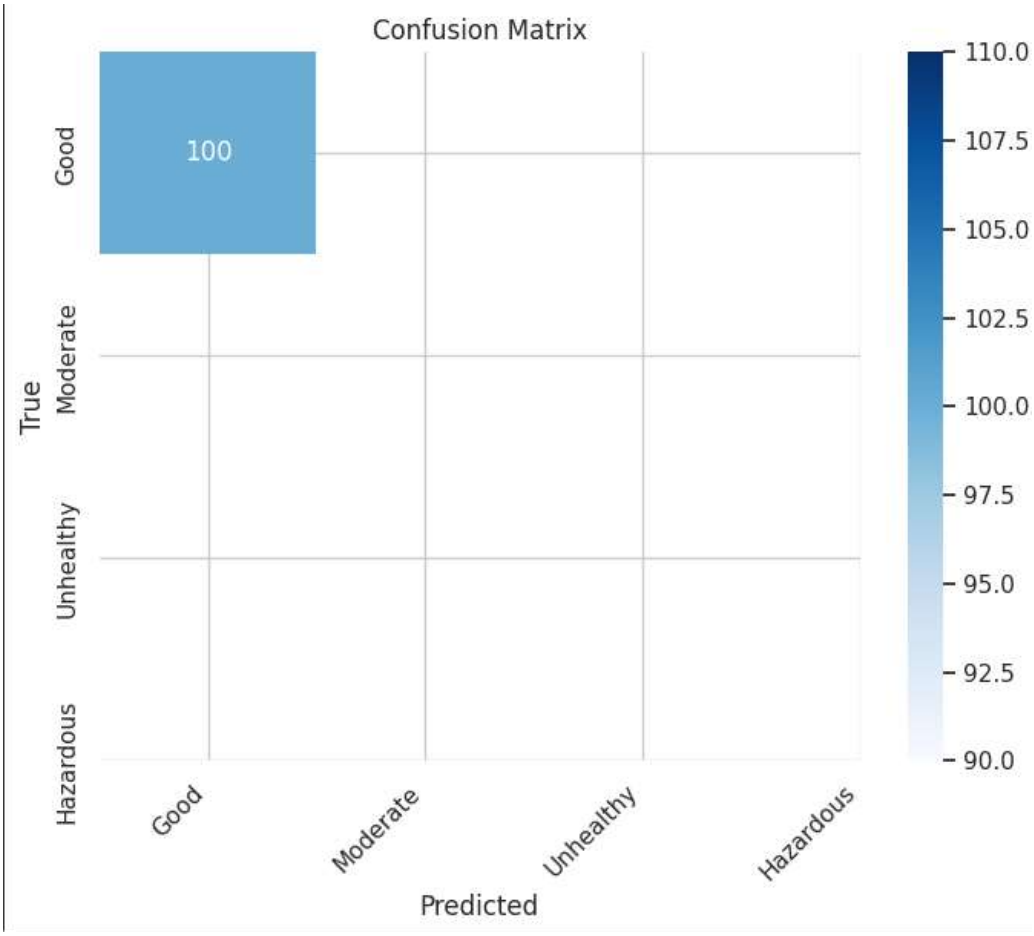
The confusion matrix is a performance evaluation tool used in classification tasks that provides a thorough analysis of the model's predictions across all potential categories. The link between actual AQI

categories and those predicted by the model is shown graphically in this context. Every column shows the anticipated category, while every row displays the actual category. This enables us to evaluate the model's capacity to accurately categorize every group and spot incorrect classifications.
The code provided generates the confusion matrix after applying the categorize function to transform the actual (y_test) and predicted (y_pred) AQI values into categorized labels. The mapping of continuous AQI values into meaningful categories, such as "Good," "Moderate," "Unhealthy," and "Hazardous," makes the judgment more understandable.

Code

```
# Map the numeric predictions back to categories
y_test_cat = y_test.apply(categorize_aqi)  # Apply the function to y_test
y_pred_cat = pd.Series(y_pred).apply(categorize_aqi)  # Apply the function to y_pred

# Generate confusion matrix using categorized data
cm = confusion_matrix(y_test_cat, y_pred_cat)

# Plot confusion matrix using heatmap
plt.figure(figsize=(8, 6))  # Adjust size for better readability
sns.heatmap(cm, annot=True, fmt='d', cmap="Blues",
xticklabels=["Good", "Moderate", "Unhealthy", "Hazardous"],  # Updated labels
yticklabels=["Good", "Moderate", "Unhealthy", "Hazardous"])  # Updated labels
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix")
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for readability
plt.show()
```

Output



Code 20: Confusion Matrix

With no misclassifications or predictions for other categories, the output displays a confusion matrix with all 100 cases falling into the "Good" category. The model forecasts all test samples as "Good" with complete accuracy, which is consistent with the previous classification output (accuracy score of 1.0). This finding, however, probably points to a problem with the model's capacity for generalization, as it might have been trained on unbalanced data that was primarily classified as "Good." The absence of other categories ("Moderate", "Unhealthy", "Hazardous") in both true and forecasted values makes this clear.
To conclude, the confusion matrix, when combined with the model's output, functions as a diagnostic tool. It emphasizes how crucial it is to examine both the distribution of forecasts across all categories and overall accuracy in order to guarantee accurate and useful classification of air quality levels.

6. Conclusion and Discussion

This project was able to effectively investigate air quality using the “Global Air Quality” data set through data cleaning and enhancement, EDA and proper modeling techniques. The developed models were found to yield considerable information about the interactions between pollutants and the environment.

Results:
The investigations showed that the Random Forest Classifier offered very high accuracy in classifying AQI categories, which must be due to the reasonable functioning in dealing with mixed data types and interactions between features.
The Linear Regression model provided a good estimation of the pollutant concentrations; the analysis also demonstrated an understanding of linear correlation between environmental parameters and pollutants. Preliminary statistical results are presented including basic correlations between the variables where for example PM2.5 is correlated with PM10 Positive and clear descriptions of the distribution and outliers of the data and apparent trends.

Limitations: Another issue that can be found is that due to a small and limited dataset the results are not easily generalized. Further research should work with more various datasets from different places and periods. Although the accuracy of the classification model is high according to the results obtained, it may cause a problem of imbalanced data. Further validation using a more balanced dataset is advised in order to gain maximum confidence in a given model. Regression analysis has difficulty with nonlinearity and stochastic nature of some air quality characteristics as the leaning assumes of a linear relationship.

7. References

- Ali, Waqar. “Global Air Quality Dataset 🌍🌫️.” *Kaggle.com*, 2024, www.kaggle.com/datasets/waqi786/global-air-quality-dataset.

- “Effective Methods of Categorical Data Encoding for Artificial Intelligence Algorithms.” *Mathematics*, vol. 12, no. 16, 18 Aug. 2024, pp. 2553–2553, Effective Methods of Categorical Data Encoding - ProQuest

- Mustafin, Almaz, and Aliya Kantarbayeva. “Resource Competition and Technological Diversity.” *PLOS ONE*, vol. 16, no. 11, 18 Nov. 2021, p. e0259875, https://doi.org/10.1371/journal.pone.0259875.

- Sharma, Vinod. “A Study on Data Scaling Methods for Machine Learning.” *International Journal for Global Academic & Scientific Research*, vol. 1, no. 1, 23 Feb. 2022, https://doi.org/10.55938/ijgasr.v1i1.4.

- “View of Analisis Perbandingan Metode Regresi Linier, Random Forest Regression Dan Gradient Boosted Trees Regression Method Untuk Prediksi Harga Rumah.” *https://journal.isas.or.id/index.php/JACOST/article/view/491/202*.

8. Appendix

Table of Tables

Table of Code

Project Code Google Colab File:
https://colab.research.google.com/drive/1dkJvQrVMNNwcQpGgc4eYkgJgGe3dKCkW?usp=sharing

SCAN FOR THE ASSIGNMENT COLAB FILE LINK