Assignment No: 7

—----------

Title of the Assignment: Data Loading, Storage and File Formats

Problem Statement: Analyzing Sales Data from Multiple File Formats

Dataset: Sales data in multiple file formats (e.g., CSV, Excel, JSON)

Description: The goal is to load and analyze sales data from different file formats, including CSV, Excel, and JSON, and perform data cleaning, transformation, and analysis on the dataset.

Tasks to Perform:

Obtain sales data files in various formats, such as CSV, Excel, and JSON.

1. Load the sales data from each file format into the appropriate data structures or DataFrames.
2. Explore the structure and content of the loaded data, identifying any inconsistencies, missing values, or data quality issues.
3. Perform data cleaning operations, such as handling missing values, removing duplicates, or correcting inconsistencies.
4. Convert the data into a unified format, such as a common DataFrames or data structure, to enable seamless analysis.
5. Perform data transformation tasks, such as merging multiple datasets, splitting columns, or deriving new variables.
6. Analyze the sales data by performing descriptive statistics, aggregating data by specific variables, or calculating metrics such as total sales, average order value, or product category distribution.
7. Create visualizations, such as bar plots, pie charts, or box plots, to represent the sales data and gain insights into sales trends, customer behaviour, or product performance.

Objective of the Assignment: Students should be able to load data from various file formats, clean and preprocess the data, and perform exploratory data analysis (EDA) to gain insights.

Data loading, storage, and file formats play a crucial role in data analysis. Different sources and file formats require specific methods for loading and processing. In this assignment, we will explore the following steps:

Step-1: Import necessary libraries

Before starting with data loading and analysis, we need to import the required Python libraries, such as pandas, NumPy, and matplotlib, to facilitate the process.

Step-2: Load data from different file formats

In this step, we will load sales data from various file formats, including CSV, Excel, and JSON. Each format may require specific methods and libraries for loading.

Step-3: Data Cleaning and Preprocessing
Data from different sources can be messy and may contain missing values, duplicates, or inconsistent data. Data cleaning is essential to prepare the dataset for analysis. We will cover techniques for data cleaning and preprocessing.

Step-4: Exploratory Data Analysis (EDA)
EDA is a crucial step in understanding the dataset. It involves tasks like summary statistics, data visualization, and identifying patterns or trends in the data. We will use Python libraries like matplotlib for data visualization.

Step-5: Analysis and Insights
After performing EDA, we will extract valuable insights from the sales data. This may include identifying top-selling products, sales trends, or customer behaviour patterns.

Conclusion: In this assignment, we have explored the fundamentals of data loading, storage, and file formats. We have learned how to load and analyze sales data from various file formats, perform data cleaning, transformation, and conduct exploratory data analysis. This knowledge is essential for data analysts and data scientists working with diverse datasets from different sources and formats.

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import os
```

In [2]:
```
csv_data = pd.read_csv("format1.csv")
excel_data = pd.read_excel("format2.xlsx")
json_data = pd.read_json("format3.json")
```

In [3]:
```
csv_data.head()
```

Out[3]:

	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total	Date	Time	Payment	cogs	gross margin percentage	gro inco
0	A	Yangon	Member	Female	Health and beauty	74.69	7	26.1415	548.9715	1/5/2019	13:08	Ewallet	522.83	4.761905	26.14
1	C	Naypyitaw	Normal	Female	Electronic accessories	15.28	5	3.8200	80.2200	3/8/2019	10:29	Cash	76.40	4.761905	3.82
2	A	Yangon	Normal	Male	Home and lifestyle	46.33	7	16.2155	340.5255	3/3/2019	13:23	Credit card	324.31	4.761905	16.21
3	A	Yangon	Member	Male	Health and beauty	58.22	8	23.2880	489.0480	1/27/2019	20:33	Ewallet	465.76	4.761905	23.28
4	A	Yangon	Normal	Male	Sports and travel	86.31	7	30.2085	634.3785	2/8/2019	10:37	Ewallet	604.17	4.761905	30.20

In [4]:
```
excel_data.head()
```

Out[4]:

	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total	Date	Time	Payment	cogs	gross margin percentage	gro inco
0	A	Yangon	Normal	Male	Electronic accessories	51.69	7	18.0915	379.9215	1/26/2019	18:22	Cash	361.83	4.761905	18.09
1	B	Mandalay	Member	Female	Fashion accessories	54.73	7	19.1555	402.2655	3/14/2019	19:02	Credit card	383.11	4.761905	19.15
2	B	Mandalay	Member	Male	Home and lifestyle	27.00	9	12.1500	255.1500	3/2/2019	14:16	Cash	243.00	4.761905	12.15
3	C	Naypyitaw	Normal	Female	Electronic accessories	30.24	1	1.5120	31.7520	3/4/2019	15:44	Cash	30.24	4.761905	1.51
4	B	Mandalay	Member	Female	Food and beverages	89.14	4	17.8280	374.3880	1/7/2019	12:20	Credit card	356.56	4.761905	17.82

In [5]:
```
json_data.head()
```

Out[5]:

	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total	Date	Time	Payment	cogs	gross margin percentage	gross income
701	B	Mandalay	Normal	Male	Food and beverages	32.32	3	4.8480	101.8080	2019-03-27	19:11	Credit card	96.96	4.761905	4.848(
702	B	Mandalay	Member	Female	Fashion accessories	19.77	10	9.8850	207.5850	2019-02-27	18:57	Credit card	197.70	4.761905	9.885(
703	B	Mandalay	Member	Male	Health and beauty	80.47	9	36.2115	760.4415	2019-01-06	11:18	Cash	724.23	4.761905	36.211
704	B	Mandalay	Member	Female	Home and lifestyle	88.39	9	39.7755	835.2855	2019-03-02	12:40	Cash	795.51	4.761905	39.775
705	B	Mandalay	Normal	Male	Health and beauty	71.77	7	25.1195	527.5095	2019-03-29	14:06	Cash	502.39	4.761905	25.119

In [6]:
```
def merge(dataframes):
    if dataframes:
        return pd.concat(dataframes)
```

In [7]:
```
df = merge([csv_data, excel_data, json_data])
```

In [8]:
```
df.head()
```

Out[8]:

	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total	Date	Time	Payment	cogs	gross margin percentage	gro incor
0	A	Yangon	Member	Female	Health and beauty	74.69	7	26.1415	548.9715	1/5/2019	13:08	Ewallet	522.83	4.761905	26.14
1	C	Naypyitaw	Normal	Female	Electronic accessories	15.28	5	3.8200	80.2200	3/8/2019	10:29	Cash	76.40	4.761905	3.82
2	A	Yangon	Normal	Male	Home and lifestyle	46.33	7	16.2155	340.5255	3/3/2019	13:23	Credit card	324.31	4.761905	16.21
3	A	Yangon	Member	Male	Health and beauty	58.22	8	23.2880	489.0480	1/27/2019	20:33	Ewallet	465.76	4.761905	23.28
4	A	Yangon	Normal	Male	Sports and travel	86.31	7	30.2085	634.3785	2/8/2019	10:37	Ewallet	604.17	4.761905	30.20

In [9]:
```
df.describe()
```

Out[9]:

	Unit price	Quantity	Tax 5%	Total	cogs	gross margin percentage	gross income	Rating
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.00000	1000.000000	1000.000000	1000.00000
mean	55.672130	5.510000	15.379369	322.966749	307.58738	4.761905	15.379369	6.97270
std	26.494628	2.923431	11.708825	245.885335	234.17651	0.000000	11.708825	1.71858
min	10.080000	1.000000	0.508500	10.678500	10.17000	4.761905	0.508500	4.00000
25%	32.875000	3.000000	5.924875	124.422375	118.49750	4.761905	5.924875	5.50000
50%	55.230000	5.000000	12.088000	253.848000	241.76000	4.761905	12.088000	7.00000
75%	77.935000	8.000000	22.445250	471.350250	448.90500	4.761905	22.445250	8.50000
max	99.960000	10.000000	49.650000	1042.650000	993.00000	4.761905	49.650000	10.00000

In [10]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000 entries, 0 to 999
Data columns (total 16 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Branch                   1000 non-null   object 
 1   City                     1000 non-null   object 
 2   Customer type            1000 non-null   object 
 3   Gender                   1000 non-null   object 
 4   Product line             1000 non-null   object 
 5   Unit price               1000 non-null   float64
 6   Quantity                 1000 non-null   int64  
 7   Tax 5%                   1000 non-null   float64
 8   Total                    1000 non-null   float64
 9   Date                     1000 non-null   object 
 10  Time                     1000 non-null   object 
 11  Payment                  1000 non-null   object 
 12  cogs                     1000 non-null   float64
 13  gross margin percentage  1000 non-null   float64
 14  gross income             1000 non-null   float64
 15  Rating                   1000 non-null   float64
dtypes: float64(7), int64(1), object(8)
memory usage: 132.8+ KB
```

In [11]:
```
df.Date = pd.to_datetime(df.Date)
df.Time = pd.to_datetime(df.Time)
```

In [12]:
```
df.dtypes
```

Out[12]:
```
Branch                          object
City                            object
Customer type                   object
Gender                          object
Product line                    object
Unit price                     float64
Quantity                         int64
Tax 5%                         float64
Total                          float64
Date                    datetime64[ns]
Time                    datetime64[ns]
Payment                         object
cogs                           float64
gross margin percentage        float64
gross income                   float64
Rating                         float64
dtype: object
```

In [13]:
```
df.isna().sum()
```

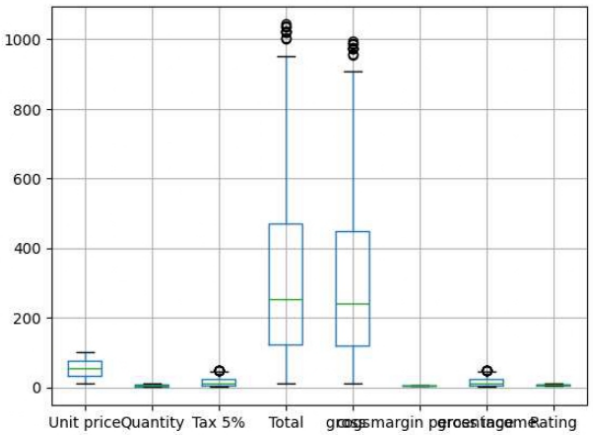
Out[13]:
```
Branch                    0
City                      0
Customer type             0
Gender                    0
Product line              0
Unit price                0
Quantity                  0
Tax 5%                    0
Total                     0
Date                      0
Time                      0
Payment                   0
cogs                      0
gross margin percentage   0
gross income              0
Rating                    0
dtype: int64
```

In [14]:
```
df.duplicated().sum()
```

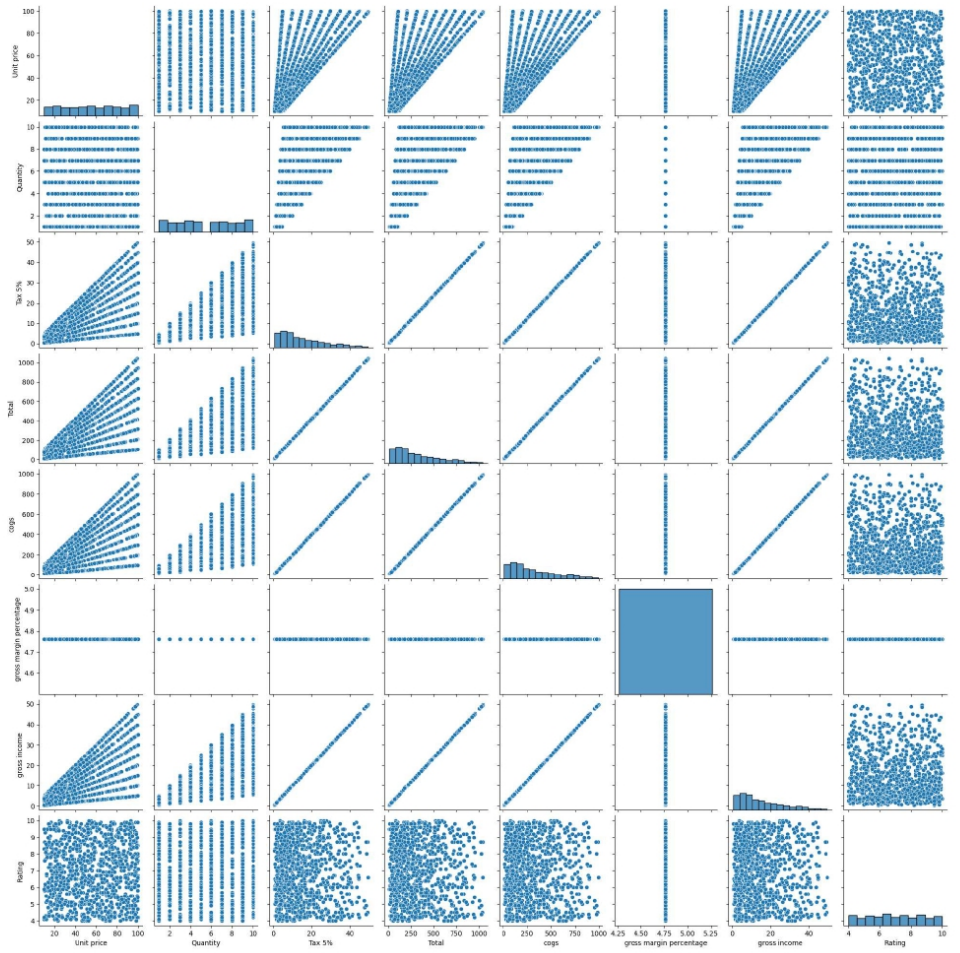
Out[14]:
```
0
```

In [15]:
```
df.boxplot()
```

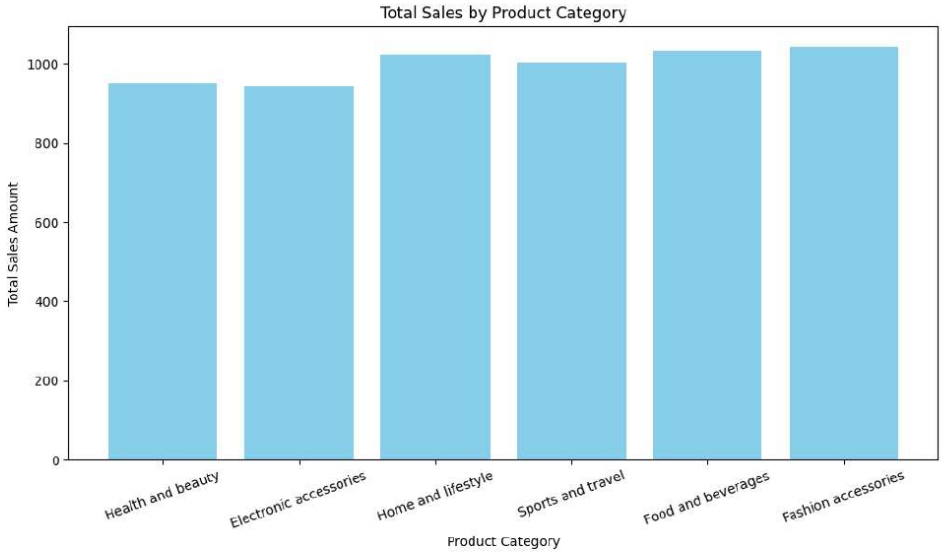
Out[15]:
```
<Axes: >
```



In [16]:
```
sns.pairplot(df)
plt.show()
```



In [17]:
```
plt.figure(figsize=(12, 6))
plt.bar(df['Product line'], df['Total'], color='skyblue')
plt.xlabel('Product Category')
plt.ylabel('Total Sales Amount')
plt.title('Total Sales by Product Category')
plt.xticks(rotation=20)
plt.show()
```



Assignment No: 8

—----------

Title of the Assignment: Interacting with Web APIsProblem Statement: Analyzing Weather Data from OpenWeatherMap API

Dataset: Weather data retrieved from OpenWeatherMap API

Description: The goal is to interact with the OpenWeatherMap API to retrieve weather data for a specific location and perform data modelling and visualization to analyze weather patterns over time.

Tasks to Perform:

1. Register and obtain API key from OpenWeatherMap.
2. Interact with the OpenWeatherMap API using the API key to retrieve weather data for a specific location.
3. Extract relevant weather attributes such as temperature, humidity, wind speed, and precipitation from the API response.
4. Clean and preprocess the retrieved data, handling missing values or inconsistent formats.
5. Perform data modelling to analyze weather patterns, such as calculating average temperature, maximum/minimum values, or trends over time.
6. Visualize the weather data using appropriate plots, such as line charts, bar plots, or scatter plots, to represent temperature changes, precipitation levels, or wind speed variations.
7. Apply data aggregation techniques to summarize weather statistics by specific time periods (e.g., daily, monthly, seasonal).
8. Incorporate geographical information, if available, to create maps or geospatial visualizations representing weather patterns across different locations.
9. Explore and visualize relationships between weather attributes, such as temperature and humidity, using correlation plots or heatmaps.

Objective of the Assignment: Students should be able to access and retrieve data from a web API, clean and preprocess the data, and perform data modelling and visualization to analyze weather patterns.

Web APIs (Application Programming Interfaces) provide a valuable way to access data from various sources, including weather data from services like OpenWeatherMap. In this assignment, we will follow these steps:

Step-1: Introduction to Web APIs

Understand the concept of web APIs and how they enable data retrieval from external sources. Explore the OpenWeatherMap API and its capabilities for accessing weather data.

Step-2: Making HTTP Requests in Python

Learn how to use Python to make HTTP requests to the OpenWeatherMap API. We'll cover different HTTP methods and how to pass parameters to retrieve specific weather data for a location.

Step-3: Data Retrieval from OpenWeatherMap API
Retrieve weather data for a specified location using the OpenWeatherMap API. This data may include temperature, humidity, wind speed, and more.

Step-4: Data Cleaning and Preprocessing
Clean and preprocess the retrieved weather data. This may involve handling missing values, data type conversions, and removing outliers.

Step-5: Data Modelling for Weather Analysis
Perform data modelling to analyze weather patterns over time. This could include tasks like calculating monthly averages, identifying temperature trends, or understanding the impact of weather conditions on local events.

Step-6: Data Visualization with Python Libraries
Visualize the analyzed weather data using Python libraries such as matplotlib. Create plots and charts to present weather patterns effectively.

Conclusion: In this assignment, we have explored the process of interacting with web APIs, specifically the OpenWeatherMap API, to retrieve and analyze weather data. Students have learned how to make HTTP requests, clean and preprocess data, perform data modelling, and create data visualizations to gain insights into local weather conditions. This knowledge is valuable for data analysts and scientists working with external data sources to derive meaningful conclusions from real-world data.

In [12]:
```
import requests
import pandas as pd
import json
import matplotlib.pyplot as plt
import seaborn as sns

API_key = '89b6535b863227f5b3200a4f58ca7650'

countries = ['Jamaica', 'Indonesia', 'United States',  "Turkey", 'Saudi Arabia' , "Egypt", 'China']

country_name_list = []
maxtemp = []
mintemp = []
humidity = []
windspeed = []

for country_names in countries:

    url = f'http://api.openweathermap.org/data/2.5/weather?q={country_names}&APPID={API_key}&units=imperial'

    r = requests.get(url)

    data = r.json()

    formatted_json = json.dumps(data, sort_keys = True, indent = 4)

    country_name_list.append(data['name'])
    maxtemp.append(data['main']['temp_max'])
    mintemp.append(data['main']['temp_min'])
    humidity.append(data['main']['humidity'])
    windspeed.append(data['wind']['speed'])


df = pd.DataFrame()
df['Names'] = country_name_list
df['Max_Temp'] = maxtemp
df['Min_Temp'] = mintemp
df['Humidity'] = humidity
df['WindSpeed'] = windspeed

df.head()
```

Out[12]:

	Names	Max_Temp	Min_Temp	Humidity	WindSpeed
0	Jamaica	89.91	89.91	52	10.78
1	Indonesia	72.34	72.34	73	4.23
2	United States of America	49.41	49.41	45	14.07
3	Turkey	56.12	56.12	84	3.31
4	Saudi Arabia	81.48	81.48	13	4.72

In [13]:
```
df.isna().sum()
```

Out[13]:
```
Names        0
Max_Temp     0
Min_Temp     0
Humidity     0
WindSpeed    0
dtype: int64
```

In [14]:
```
df.duplicated().sum()
```

Out[14]:
```
0
```

In [15]:
```
df.describe()
```

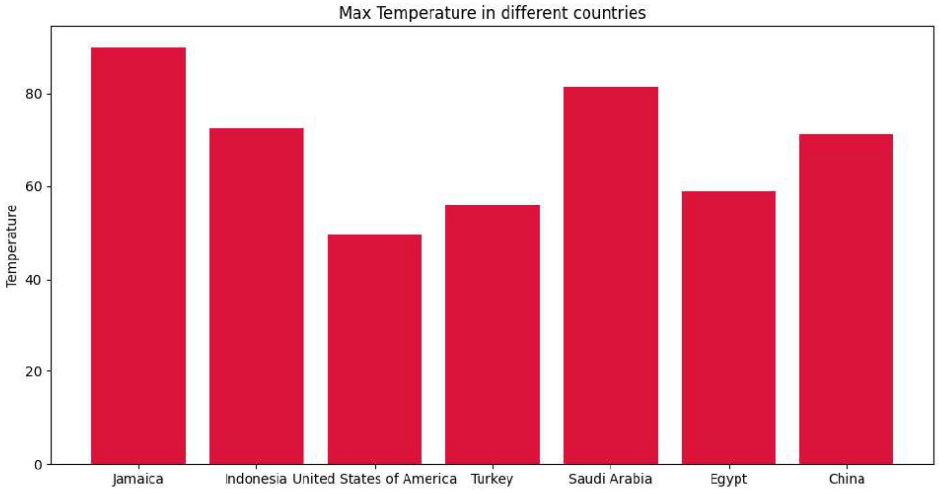
Out[15]:

	Max_Temp	Min_Temp	Humidity	WindSpeed
count	7.000000	7.000000	7.000000	7.000000
mean	68.455714	67.607143	52.714286	9.261429
std	14.490269	15.299906	24.095050	5.025695
min	49.410000	49.410000	13.000000	3.310000
25%	57.470000	54.500000	40.500000	4.475000
50%	71.110000	71.110000	52.000000	10.780000
75%	76.910000	76.910000	69.500000	13.410000
max	89.910000	89.910000	84.000000	14.970000

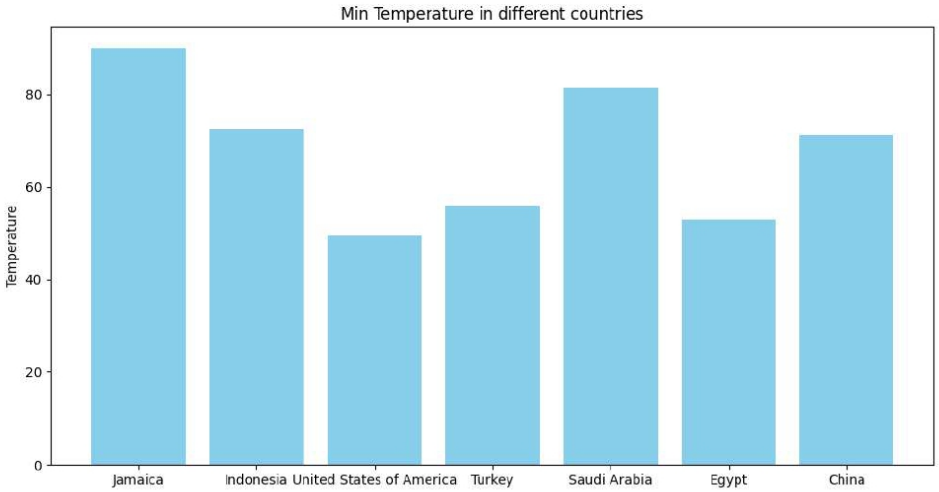
In [16]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7 entries, 0 to 6
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Names      7 non-null      object
 1   Max_Temp   7 non-null      float64
 2   Min_Temp   7 non-null      float64
 3   Humidity   7 non-null      int64
 4   WindSpeed  7 non-null      float64
dtypes: float64(3), int64(1), object(1)
memory usage: 408.0+ bytes
```

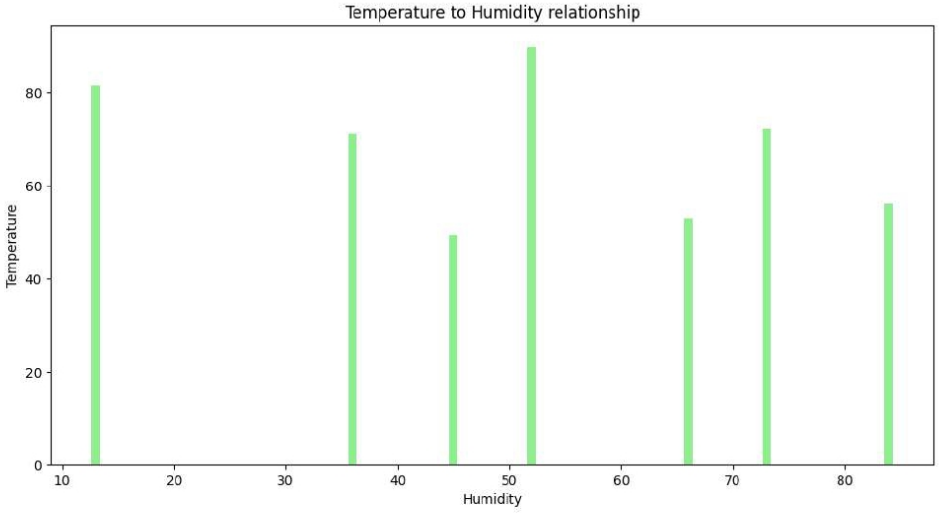
In [28]:
```
plt.figure(figsize=(12,6))
plt.bar(df['Names'], df['Max_Temp'], color='crimson')
plt.title("Max Temperature in different countries")
plt.ylabel("Temperature")
plt.show()
```



In [29]:
```
plt.figure(figsize=(12,6))
plt.bar(df['Names'], df['Min_Temp'], color='skyblue')
plt.title("Min Temperature in different countries")
plt.ylabel("Temperature")
plt.show()
```



In [42]:
```
plt.figure(figsize=(12,6))
plt.bar(df['Humidity'], df['Min_Temp'], color='lightgreen')
plt.title("Temperature to Humidity relationship")
plt.ylabel("Temperature")
plt.xlabel("Humidity")
plt.show()
```



In [ ]:

Assignment No: 9

—----------

Title of the Assignment: Data Cleaning and Preparation

Problem Statement: Analyzing Customer Churn in a Telecommunications Company

Dataset: "Telecom_Customer_Churn.csv"

Description: The dataset contains information about customers of a telecommunications company and whether they have churned (i.e., discontinued their services). The dataset includes various attributes of the customers, such as their demographics, usage patterns, and account information. The goal is to perform data cleaning and preparation to gain insights into the factors that contribute to customer churn.

Tasks to Perform:

1. Import the "Telecom_Customer_Churn.csv" dataset.
2. Explore the dataset to understand its structure and content.
3. Handle missing values in the dataset, deciding on an appropriate strategy.
4. Remove any duplicate records from the dataset.
5. Check for inconsistent data, such as inconsistent formatting or spelling variations, and standardize it.
6. Convert columns to the correct data types as needed.
7. Identify and handle outliers in the data.
8. Perform feature engineering, creating new features that may be relevant to predicting customer churn.
9. Normalize or scale the data if necessary.
10. Split the dataset into training and testing sets for further analysis.
11. Export the cleaned dataset for future analysis or modelling.

Objective of the Assignment: Students should be able to clean and prepare data for analysis, including handling missing values, handling outliers, and performing feature engineering to extract insights into customer churn.

Customer churn analysis is crucial for businesses, especially in the telecommunications industry. It helps in understanding why customers leave and what factors influence their decision. This assignment will guide students through the following steps:

Step-1: Introduction to Customer Churn Analysis

Understand the importance of analyzing customer churn and its relevance in the telecommunications industry. Define the problem and objectives.

Step-2: Data Cleaning and Preprocessing

Clean and preprocess the dataset to ensure it is suitable for analysis. This includes handling missing values, ensuring data consistency, and preparing it for further analysis.

Step-3: Handling Missing Values

Identify and address missing values in the dataset. Techniques such as imputation or removal of missing data will be discussed.

Step-4: Outlier Detection and Treatment
Identify and handle outliers in the data. Outliers can significantly impact the results of the analysis and should be treated appropriately.

Step-5: Feature Engineering
Create new features or transform existing ones to extract insights related to customer churn. Feature engineering can enhance the model's predictive power.

Step-6: Data Visualization for Churn Analysis
Visualize the cleaned and prepared data to identify patterns and relationships that may indicate why customers churn. Use data visualization libraries to create informative charts and graphs.

Step-7: Drawing Conclusions
Summarize the findings from the data analysis and draw conclusions regarding the factors that contribute to customer churn. Provide actionable insights that can be used by the telecommunications company to reduce churn.

Conclusion: In this assignment, we have focused on data cleaning and preparation for customer churn analysis in a telecommunications company. By following these steps, students have learned how to clean data, handle missing values and outliers, engineer features, and visualize the data to gain insights into customer behaviour. This knowledge is essential for businesses seeking to reduce churn and improve customer retention.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder

df = pd.read_csv("telecom_churn.csv")
df.head()
```

Out[1]:

	customer_id	telecom_partner	gender	age	state	city	pincode	date_of_registration	num_dependents	estimated_salary	calls_
0	1	Reliance Jio	F	25	Karnataka	Kolkata	755597	2020-01-01	4	124962	
1	2	Reliance Jio	F	55	Mizoram	Mumbai	125926	2020-01-01	2	130556	
2	3	Vodafone	F	57	Arunachal Pradesh	Delhi	423976	2020-01-01	0	148828	
3	4	BSNL	M	46	Tamil Nadu	Kolkata	522841	2020-01-01	1	38722	
4	5	BSNL	F	26	Tripura	Delhi	740247	2020-01-01	2	55098	

In [2]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 243553 entries, 0 to 243552
Data columns (total 14 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   customer_id           243553 non-null  int64
 1   telecom_partner       243553 non-null  object
 2   gender                243553 non-null  object
 3   age                   243553 non-null  int64
 4   state                 243553 non-null  object
 5   city                  243553 non-null  object
 6   pincode               243553 non-null  int64
 7   date_of_registration  243553 non-null  object
 8   num_dependents        243553 non-null  int64
 9   estimated_salary      243553 non-null  int64
 10  calls_made            243553 non-null  int64
 11  sms_sent              243553 non-null  int64
 12  data_used             243553 non-null  int64
 13  churn                 243553 non-null  int64
dtypes: int64(9), object(5)
memory usage: 26.0+ MB
```

In [3]:
```
df.describe()
```

Out[3]:

	customer_id	age	pincode	num_dependents	estimated_salary	calls_made	sms_sent	data_used	
count	243553.000000	243553.000000	243553.000000	243553.000000	243553.000000	243553.000000	243553.000000	243553.000000	24
mean	121777.000000	46.077609	549501.270541	1.997500	85021.137839	49.010548	23.945404	4993.186025	
std	70307.839393	16.444029	259808.860574	1.414941	37508.963233	29.453556	14.733575	2942.019547	
min	1.000000	18.000000	100006.000000	0.000000	20000.000000	-10.000000	-5.000000	-987.000000	
25%	60889.000000	32.000000	324586.000000	1.000000	52585.000000	24.000000	11.000000	2490.000000	
50%	121777.000000	46.000000	548112.000000	2.000000	84990.000000	49.000000	24.000000	4987.000000	
75%	182665.000000	60.000000	774994.000000	3.000000	117488.000000	74.000000	36.000000	7493.000000	
max	243553.000000	74.000000	999987.000000	4.000000	149999.000000	108.000000	53.000000	10991.000000	

In [4]:
```
df.shape
```

Out[4]: (243553, 14)

In [5]:
```
df.isna().sum()
```

Out[5]:
```
customer_id             0
telecom_partner         0
gender                  0
age                     0
state                   0
city                    0
pincode                 0
date_of_registration    0
num_dependents          0
estimated_salary        0
calls_made              0
sms_sent                0
data_used               0
churn                   0
dtype: int64
```

In [6]:
```
df.dropna(inplace=True)
```

In [7]:
```
df.duplicated().sum()
```

Out[7]:
```
0
```

In [8]:
```
df.drop_duplicates(inplace=True)
```

In [9]:
```
df.columns
```

Out[9]:
```
Index(['customer_id', 'telecom_partner', 'gender', 'age', 'state', 'city',
       'pincode', 'date_of_registration', 'num_dependents', 'estimated_salary',
       'calls_made', 'sms_sent', 'data_used', 'churn'],
      dtype='object')
```

In [10]:
```
df.drop(['customer_id','state','city',"pincode",'telecom_partner','date_of_registration'], inplace=True, axis=1
df.head()
```

Out[10]:

	gender	age	num_dependents	estimated_salary	calls_made	sms_sent	data_used	churn
0	F	25	4	124962	44	45	-361	0
1	F	55	2	130556	62	39	5973	0
2	F	57	0	148828	49	24	193	1
3	M	46	1	38722	80	25	9377	1
4	F	26	2	55098	78	15	1393	0

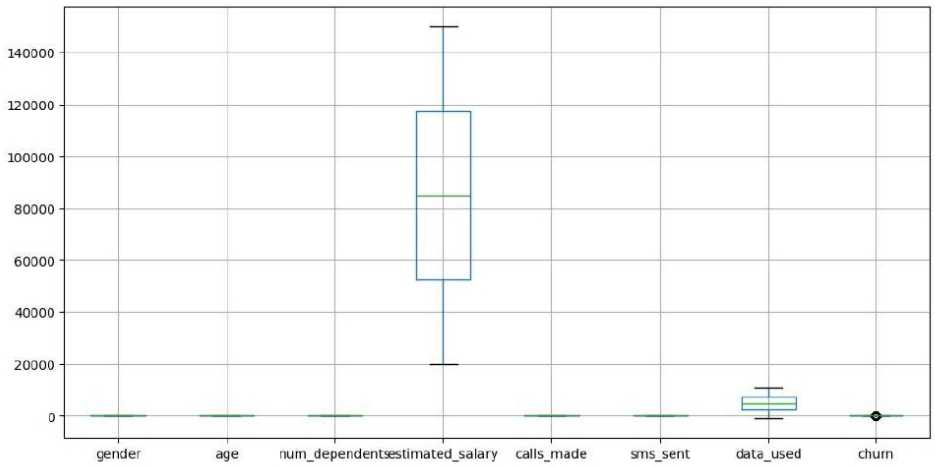
In [11]:
```
le = LabelEncoder()
df['gender'] = le.fit_transform(df['gender'])

df.head()
```

Out[11]:

	gender	age	num_dependents	estimated_salary	calls_made	sms_sent	data_used	churn
0	0	25	4	124962	44	45	-361	0
1	0	55	2	130556	62	39	5973	0
2	0	57	0	148828	49	24	193	1
3	1	46	1	38722	80	25	9377	1
4	0	26	2	55098	78	15	1393	0

In [12]:
```
plt.figure(figsize=(12,6))
df.boxplot()
plt.show()
```



In [13]:
```
df.dtypes
```

Out[13]:
```
gender              int32
age                 int64
num_dependents      int64
estimated_salary    int64
calls_made          int64
sms_sent            int64
data_used           int64
churn               int64
dtype: object
```

In [14]:
```
X = df.drop(columns=['churn'])
y = df['churn']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [15]:
```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [16]:
```
df.to_csv("Cleaned_Telecom_Customer_Churn.csv", index=False)
```

In [ ]:

Assignment No: 10

—----------

Title of the Assignment: Data Wrangling

Problem Statement: Data Wrangling on Real Estate Market

Dataset: "RealEstate_Prices.csv"

Description: The dataset contains information about housing prices in a specific real estate market. It includes various attributes such as property characteristics, location, sale prices, and other relevant features. The goal is to perform data wrangling to gain insights into the factors influencing housing prices and prepare the dataset for further analysis or modelling.

Tasks to Perform:

1. Import the "RealEstate_Prices.csv" dataset. Clean column names by removing spaces, special characters, or renaming them for clarity.
2. Handle missing values in the dataset, deciding on an appropriate strategy (e.g., imputation or removal).
3. Perform data merging if additional datasets with relevant information are available (e.g., neighbourhood demographics or nearby amenities).
4. Filter and subset the data based on specific criteria, such as a particular time period, property type, or location.
5. Handle categorical variables by encoding them appropriately (e.g., one-hot encoding or label encoding) for further analysis.
6. Aggregate the data to calculate summary statistics or derived metrics such as average sale prices by neighbourhood or property type.
7. Identify and handle outliers or extreme values in the data that may affect the analysis or modelling process.

Objective of the Assignment: Students should be able to perform data wrangling tasks such as data cleaning, data transformation, and data preparation to make the dataset suitable for analysis or modelling in the context of the real estate market.

Data wrangling, also known as data munging, is a critical step in the data analysis process. It involves cleaning, transforming, and preparing the data for analysis. In this assignment, students will follow these steps:

Step-1: Introduction to Data Wrangling

Understand the importance of data wrangling and its role in preparing data for analysis. Define the problem and objectives of the real estate market data analysis.

Step-2: Data Cleaning and Quality Assurance

Clean the dataset to ensure data quality. This includes identifying and handling missing values, dealing with duplicates, and ensuring data consistency.

Step-3: Data Transformation and Feature Engineering
Transform the data and create new features to extract insights related to housing prices. Feature engineering can enhance the analysis or modelling process.

Step-4: Handling Missing Values
Identify and address missing values in the dataset. Techniques such as imputation or removal of missing data will be discussed.

Step-5: Dealing with Duplicates
Identify and handle duplicate records in the data. Duplicates can impact the analysis and should be treated appropriately.

Step-6: Data Formatting and Standardization
Format data appropriately, including handling data types, converting categorical data, and standardizing units or scales.

Step-7: Data Preparation for Analysis
Prepare the cleaned and transformed dataset for further analysis or modelling. This may involve splitting data, scaling features, or encoding categorical variables.

Conclusion: In this assignment, we have focused on data wrangling for a real estate market dataset. By following these steps, students have learned how to clean data, transform it, handle missing values and duplicates, engineer features, and prepare the dataset for analysis or modelling. Data wrangling is a crucial step in data analysis to ensure that the data is of high quality and ready for further exploration.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder

df = pd.read_csv("Housing.csv")
df.head()
```

Out[1]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	fu
0	13300000	7420	4	2	3	yes	no	no	no	yes	2	yes	
1	12250000	8960	4	4	4	yes	no	no	no	yes	3	no	
2	12250000	9960	3	2	2	yes	no	yes	no	no	2	yes	
3	12215000	7500	4	2	2	yes	no	yes	no	yes	3	yes	
4	11410000	7420	4	1	2	yes	yes	yes	no	yes	2	no	

In [2]:
```
df.isna().sum()
```

Out[2]:
```
price             0
area              0
bedrooms          0
bathrooms         0
stories           0
mainroad          0
guestroom         0
basement          0
hotwaterheating   0
airconditioning   0
parking           0
prefarea          0
furnishingstatus  0
dtype: int64
```

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 545 entries, 0 to 544
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   price             545 non-null    int64 
 1   area              545 non-null    int64 
 2   bedrooms          545 non-null    int64 
 3   bathrooms         545 non-null    int64 
 4   stories           545 non-null    int64 
 5   mainroad          545 non-null    object
 6   guestroom         545 non-null    object
 7   basement          545 non-null    object
 8   hotwaterheating   545 non-null    object
 9   airconditioning   545 non-null    object
 10  parking           545 non-null    int64 
 11  prefarea          545 non-null    object
 12  furnishingstatus  545 non-null    object
dtypes: int64(6), object(7)
memory usage: 55.5+ KB
```

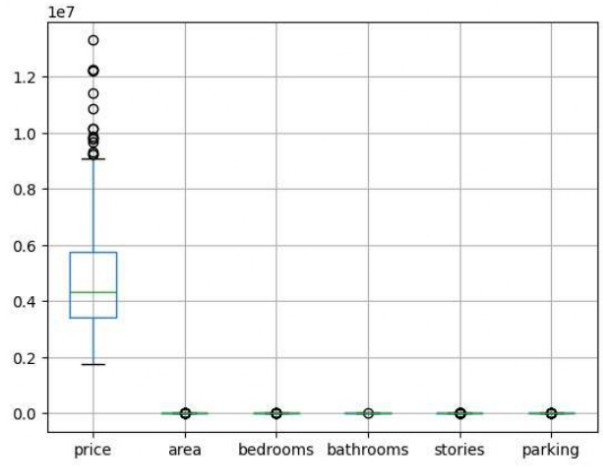
In [4]:
```
df.describe()
```

Out[4]:

	price	area	bedrooms	bathrooms	stories	parking
count	5.450000e+02	545.000000	545.000000	545.000000	545.000000	545.000000
mean	4.766729e+06	5150.541284	2.965138	1.286239	1.805505	0.693578
std	1.870440e+06	2170.141023	0.738064	0.502470	0.867492	0.861586
min	1.750000e+06	1650.000000	1.000000	1.000000	1.000000	0.000000
25%	3.430000e+06	3600.000000	2.000000	1.000000	1.000000	0.000000
50%	4.340000e+06	4600.000000	3.000000	1.000000	2.000000	0.000000
75%	5.740000e+06	6360.000000	3.000000	2.000000	2.000000	1.000000
max	1.330000e+07	16200.000000	6.000000	4.000000	4.000000	3.000000

In [5]:
```
df.boxplot()
```

Out[5]: <Axes: >



In [6]:
```
Q1 = df['price'].quantile(0.25)
Q3 = df['price'].quantile(0.75)
iqr = Q3 - Q1
minm= Q1 - (1.5*iqr)
maxm = Q3 + (1.5*iqr)
df=df[(df['price']>minm) & (df['price']<maxm)]
df.head()
```

Out[6]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	fu
15	9100000	6000	4	1	2	yes	no	yes	no	no	2	no	
16	9100000	6600	4	2	2	yes	yes	yes	no	yes	1	yes	
17	8960000	8500	3	2	4	yes	no	no	no	yes	2	no	
18	8890000	4600	3	2	2	yes	yes	no	no	yes	2	no	
19	8855000	6420	3	2	2	yes	no	no	no	yes	1	yes	

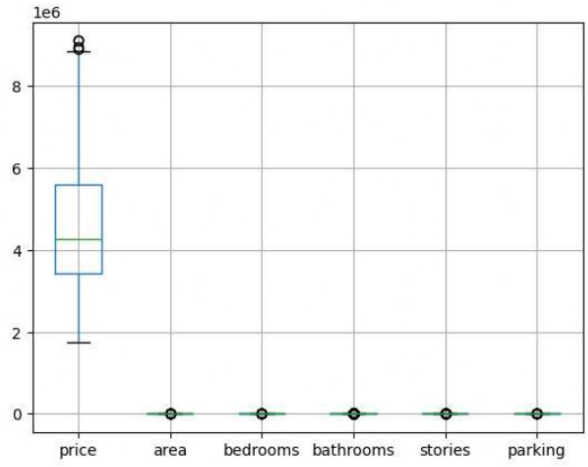
In [7]:
```
df.columns
```

Out[7]:
```
Index(['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'mainroad',
       'guestroom', 'basement', 'hotwaterheating', 'airconditioning',
       'parking', 'prefarea', 'furnishingstatus'],
      dtype='object')
```

In [8]:
```
df.dtypes
```

Out[8]:
```
price              int64
area               int64
bedrooms           int64
bathrooms          int64
stories            int64
mainroad          object
guestroom         object
basement          object
hotwaterheating   object
airconditioning   object
parking            int64
prefarea          object
furnishingstatus  object
dtype: object
```

In [9]:
```
df.boxplot()
plt.show()
```



In [10]:
```
df.head()
```

Out[10]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	fu
15	9100000	6000	4	1	2	yes	no	yes	no	no	2	no	
16	9100000	6600	4	2	2	yes	yes	yes	no	yes	1	yes	
17	8960000	8500	3	2	4	yes	no	no	no	yes	2	no	
18	8890000	4600	3	2	2	yes	yes	no	no	yes	2	no	
19	8855000	6420	3	2	2	yes	no	no	no	yes	1	yes	

In [11]:
```
le = LabelEncoder()
df['mainroad'] = le.fit_transform(df['mainroad'])
df['guestroom'] = le.fit_transform(df['guestroom'])
df['basement'] = le.fit_transform(df['basement'])
df['hotwaterheating'] = le.fit_transform(df['hotwaterheating'])
df['airconditioning'] = le.fit_transform(df['airconditioning'])
df['furnishingstatus'] = le.fit_transform(df['furnishingstatus'])
df['prefarea'] = le.fit_transform(df['prefarea'])

df.head()
```

Out[11]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	fu
15	9100000	6000	4	1	2	1	0	1	0	0	2	0	
16	9100000	6600	4	2	2	1	1	1	0	1	1	1	
17	8960000	8500	3	2	4	1	0	0	0	1	2	0	
18	8890000	4600	3	2	2	1	1	0	0	1	2	0	
19	8855000	6420	3	2	2	1	0	0	0	1	1	1	

In [ ]:
```
```

Assignment No: 11

Title of the Assignment: Data Visualization using matplotlib

Problem Statement: Analyzing Air Quality Index (AQI) Trends in a City

Dataset: "City_Air_Quality.csv"

Description: The dataset contains information about air quality measurements in a specific city over a period of time. It includes attributes such as date, time, pollutant levels (e.g., PM2.5, PM10, CO), and the Air Quality Index (AQI) values. The goal is to use the matplotlib library to create visualizations that effectively represent the AQI trends and patterns for different pollutants in the city.

Tasks to Perform:

1. Import the "City_Air_Quality.csv" dataset.
2. Explore the dataset to understand its structure and content.
3. Identify the relevant variables for visualizing AQI trends, such as date, pollutant levels, and AQI values.
4. Create line plots or time series plots to visualize the overall AQI trend over time.
5. Plot individual pollutant levels (e.g., PM2.5, PM10, CO) on separate line plots to visualize their trends over time.
6. Use bar plots or stacked bar plots to compare the AQI values across different dates or time periods.
7. Create box plots or violin plots to analyze the distribution of AQI values for different pollutant categories.
8. Use scatter plots or bubble charts to explore the relationship between AQI values and pollutant levels.
9. Customize the visualizations by adding labels, titles, legends, and appropriate colour schemes.

Objective of the Assignment: Students should be able to use Matplotlib to create various types of data visualizations that provide insights into air quality trends, allowing for the interpretation and communication of findings.

Data visualization is a powerful tool for interpreting and communicating data patterns effectively. In this assignment, students will be guided through the following steps:

Step-1: Introduction to Data Visualization with Matplotlib

Understand the importance of data visualization and how Matplotlib can be used to create a variety of charts and plots for data analysis.

Step-2: Understanding Air Quality Index (AQI)

Learn about AQI and its significance as a measure of air quality. Understand the pollutants used to calculate AQI and their health implications.

Step-3: Types of Data Visualizations for AQI Trends
Explore the different types of data visualizations suitable for analyzing AQI trends, including line plots, bar charts, and box plots.

Step-4: Line Plots for Time Series Analysis
Create line plots to visualize time series data, focusing on how AQI values change over time for specific pollutants.

Step-5: Bar Charts for Comparing Pollutants
Use bar charts to compare pollutant levels and AQI values across different time periods or locations within the city.

Step-6: Box Plots for AQI Distribution
Generate box plots to visualize the distribution of AQI values, highlighting variations and outliers in air quality data.

Step-7: Creating Legends and Labels
Enhance visualizations by adding legends, labels, and annotations to make the information more accessible and interpretable.

Conclusion: In this assignment, we have explored data visualization techniques using Matplotlib to analyze air quality trends in a specific city. By creating line plots, bar charts, and box plots, students have learned how to effectively represent AQI data, making it easier to identify patterns and communicate findings related to air quality in the city. Data visualization is a crucial skill for data analysts and scientists to derive insights from complex datasets.

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

df = pd.read_csv("AQI Data Set.csv",  parse_dates=['Mounths'])
df.head()
```

Out[1]:

	Id	Mounths	PM10 in æg/m3	SO2 in æg/m3	NOx in æg/m3	PM2.5 in æg/m3	Ammonia - NH3 in æg/m3	O3 in æg/m3	CO in mg/m3	Benzene in æg/m3	AQI
0	1	Jan-17	174.0	26.4	35.0	79	25.0	107.6	0.9	0.7	149.0
1	2	Feb-17	143.0	35.1	40.3	75	31.0	103.0	0.9	0.9	129.0
2	3	Mar-17	142.0	32.1	30.9	59	26.0	80.7	0.8	0.5	128.0
3	4	Apr-17	117.0	50.9	36.3	75	36.0	79.5	0.9	0.7	111.0
4	5	May-17	NaN	41.6	25.2	53	28.0	70.0	0.5	0.5	NaN

In [2]:
```
df.columns
```

Out[2]:
```
Index(['Id', 'Mounths', 'PM10 in æg/m3', 'SO2 in æg/m3', 'NOx  in æg/m3',
       ' PM2.5  in æg/m3', 'Ammonia - NH3  in æg/m3', 'O3   in æg/m3',
       'CO  in mg/m3', ' Benzene  in æg/m3', 'AQI'],
      dtype='object')
```

In [3]:
```
column_names = ['Id', 'Months', 'PM10', 'SO2', 'NOx',
       'PM25', 'NH3', 'O3','CO', ' Benzene', 'AQI']

df.columns = column_names
df.head()
```

Out[3]:

	Id	Months	PM10	SO2	NOx	PM25	NH3	O3	CO	Benzene	AQI
0	1	Jan-17	174.0	26.4	35.0	79	25.0	107.6	0.9	0.7	149.0
1	2	Feb-17	143.0	35.1	40.3	75	31.0	103.0	0.9	0.9	129.0
2	3	Mar-17	142.0	32.1	30.9	59	26.0	80.7	0.8	0.5	128.0
3	4	Apr-17	117.0	50.9	36.3	75	36.0	79.5	0.9	0.7	111.0
4	5	May-17	NaN	41.6	25.2	53	28.0	70.0	0.5	0.5	NaN

In [4]:
```
df.isna().sum()
```

Out[4]:
```
Id          0
Months      0
PM10        6
SO2         1
NOx         2
PM25        0
NH3         0
O3          0
CO          0
 Benzene    0
AQI         5
dtype: int64
```

In [5]:
```
df.dropna(inplace=True)
df.isna().sum()
```

Out[5]:
```
Id          0
Months      0
PM10        0
SO2         0
NOx         0
PM25        0
NH3         0
O3          0
CO          0
 Benzene    0
AQI         0
dtype: int64
```

In [6]:
```
df.describe()
```

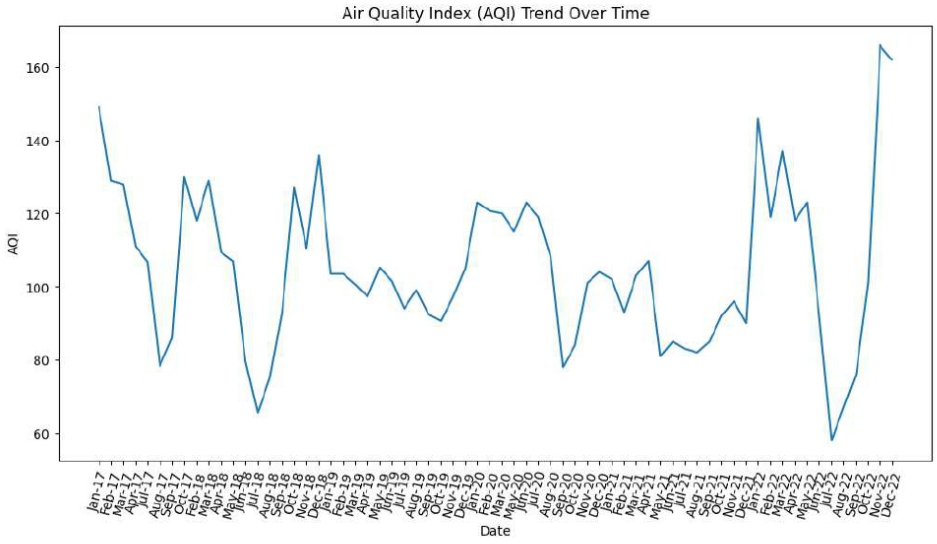
Out[6]:

	Id	PM10	SO2	NOx	PM25	NH3	O3	CO	Benzene	AQI
count	66.000000	66.000000	66.000000	66.000000	66.000000	66.000000	66.000000	66.000000	66.000000	66.000000
mean	38.500000	109.393939	16.093939	30.263636	46.393939	24.072727	25.350000	0.551212	0.213636	104.807576
std	20.417376	25.271376	9.265218	3.947838	20.261277	5.960474	21.426413	0.241550	0.190922	22.054250
min	1.000000	76.000000	4.000000	18.400000	12.000000	11.000000	2.400000	0.200000	0.000000	58.000000
25%	22.250000	90.000000	9.850000	28.125000	27.500000	20.250000	12.025000	0.400000	0.100000	90.950000
50%	38.500000	104.000000	13.700000	29.750000	46.500000	23.000000	18.750000	0.500000	0.150000	103.250000
75%	55.750000	128.000000	17.150000	32.550000	62.750000	28.000000	31.575000	0.640000	0.300000	119.000000
max	72.000000	178.000000	50.900000	40.300000	87.000000	37.000000	107.600000	1.520000	0.900000	166.000000

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 66 entries, 0 to 71
Data columns (total 11 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Id        66 non-null     int64
 1   Months    66 non-null     object
 2   PM10      66 non-null     float64
 3   SO2       66 non-null     float64
 4   NOx       66 non-null     float64
 5   PM25      66 non-null     int64
 6   NH3       66 non-null     float64
 7   O3        66 non-null     float64
 8   CO        66 non-null     float64
 9    Benzene  66 non-null     float64
 10  AQI       66 non-null     float64
dtypes: float64(8), int64(2), object(1)
memory usage: 6.2+ KB
```

In [8]:
```
plt.figure(figsize=(12, 6))
plt.plot(df['Months'], df['AQI'])
plt.xlabel('Date')
plt.ylabel('AQI')
plt.xticks(rotation=75)
plt.title('Air Quality Index (AQI) Trend Over Time')
plt.show()
```

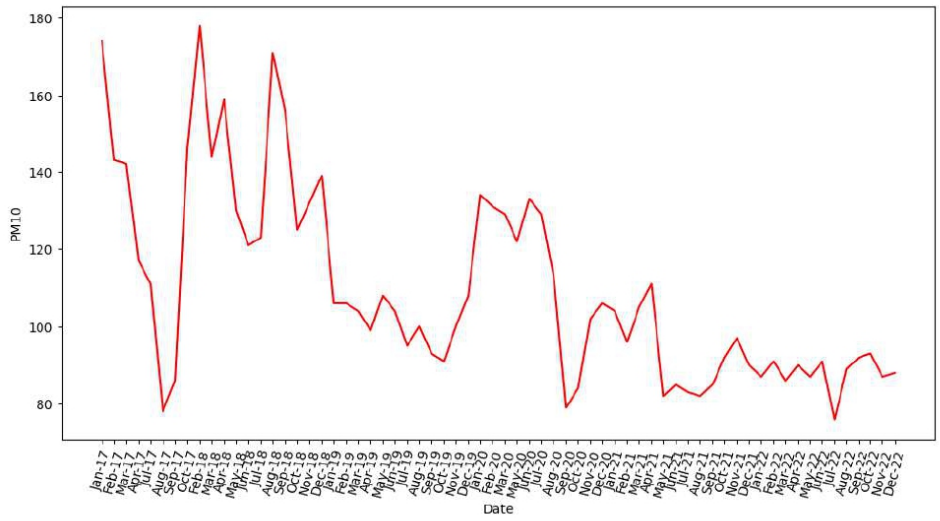


In [9]:
```
df.columns
```

Out[9]:
```
Index(['Id', 'Months', 'PM10', 'SO2', 'NOx', 'PM25', 'NH3', 'O3', 'CO',
       ' Benzene', 'AQI'],
      dtype='object')
```

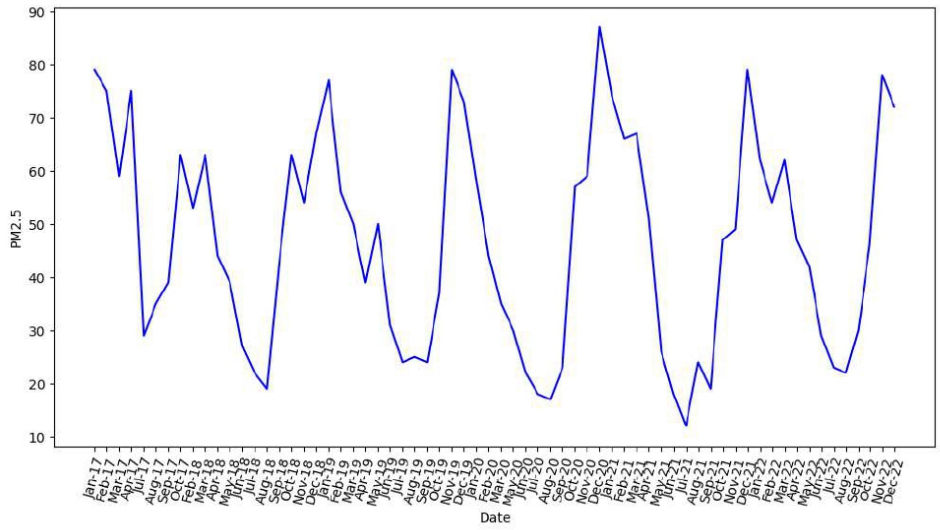
In [10]:
```
plt.figure(figsize=(12, 6))
plt.plot(df['Months'], df['PM10'], color='red')
plt.xlabel('Date')
plt.xticks(rotation=75)
plt.ylabel('PM10')
```

Out[10]: Text(0, 0.5, 'PM10')



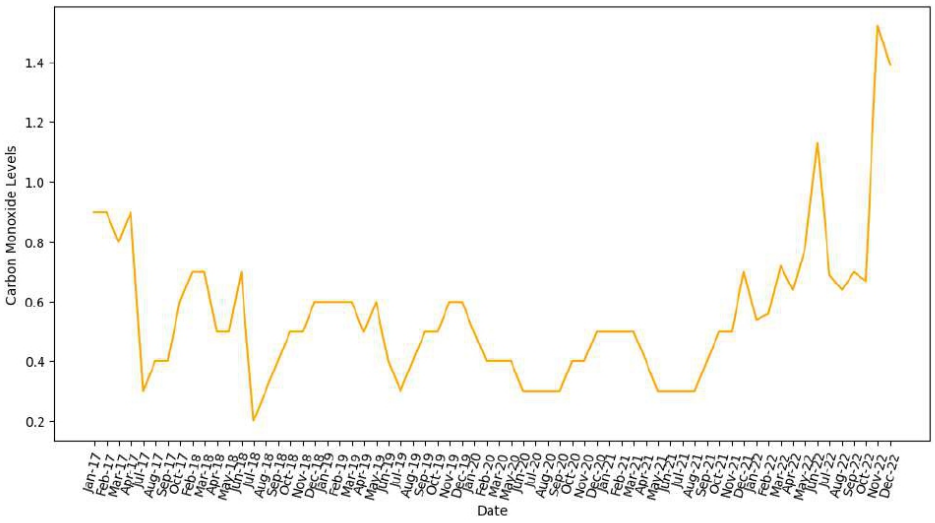
In [11]:
```
plt.figure(figsize=(12, 6))
plt.plot(df['Months'], df['PM25'], color='blue')
plt.xlabel('Date')
plt.xticks(rotation=75)
plt.ylabel('PM2.5')
```

Out[11]: Text(0, 0.5, 'PM2.5')

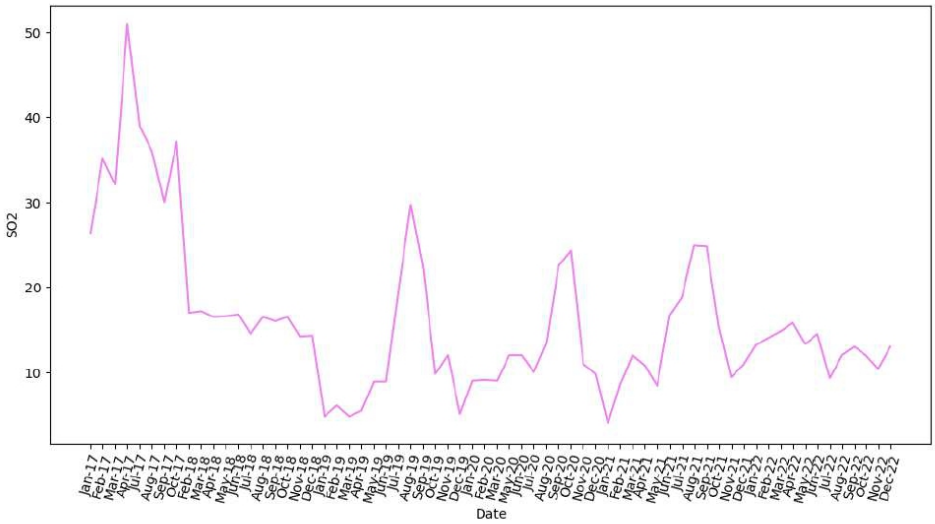


In [12]:
```
plt.figure(figsize=(12, 6))
plt.plot(df['Months'], df['CO'], label='CO', color='orange')
plt.xlabel('Date')
plt.xticks(rotation=75)
plt.ylabel('Carbon Monoxide Levels')
```

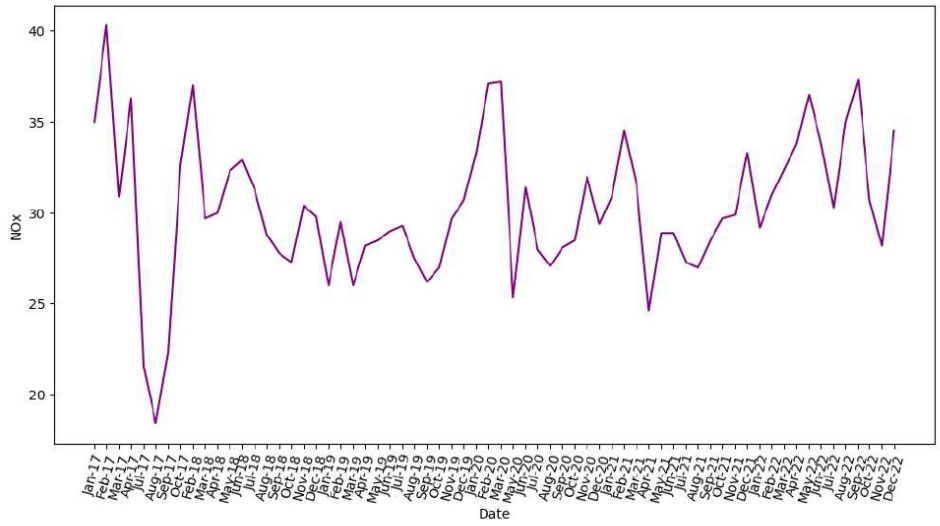
Out[12]:
```
Text(0, 0.5, 'Carbon Monoxide Levels')
```



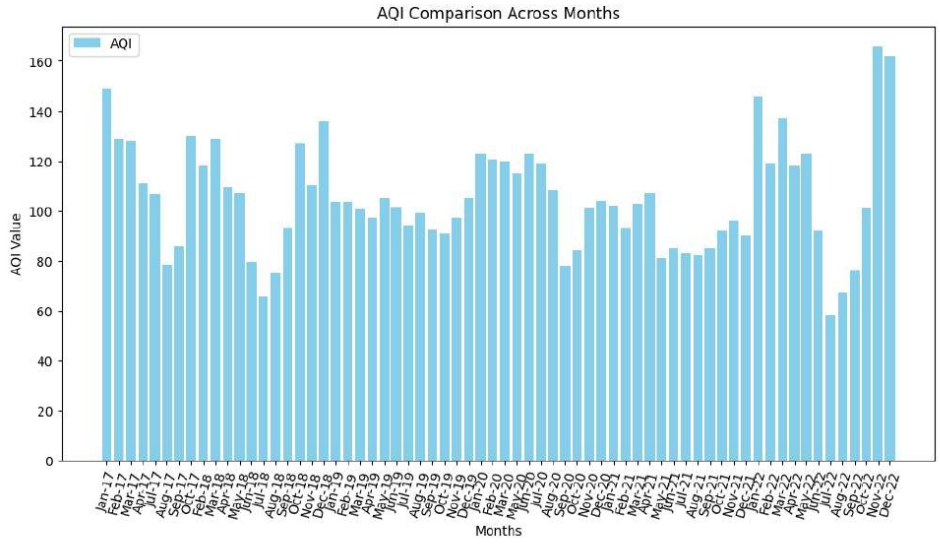
In [13]:
```
plt.figure(figsize=(12, 6))
plt.plot(df['Months'], df['SO2'], label='SO2', color='violet')
plt.xlabel('Date')
plt.ylabel('SO2')
plt.xticks(rotation=75)
plt.show()
```



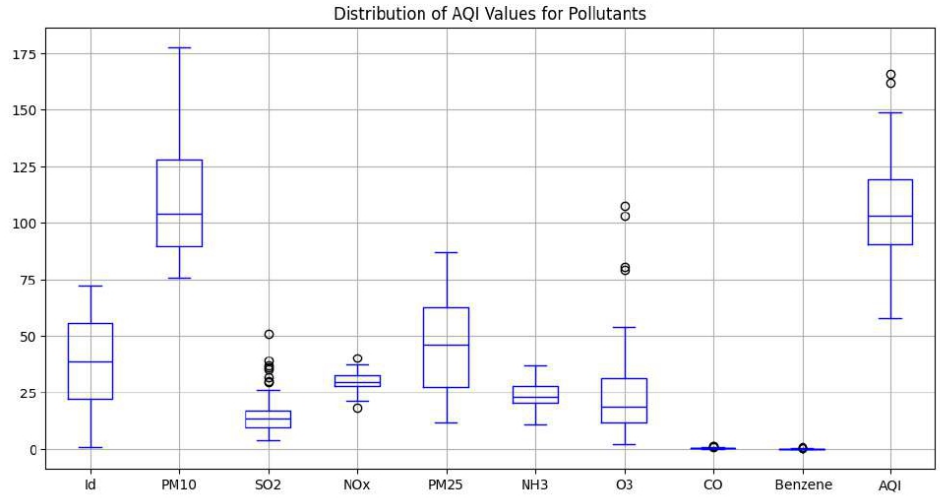
In [14]:
```
plt.figure(figsize=(12, 6))
plt.plot(df['Months'], df['NOx'], label='NOx', color='purple')
plt.xlabel('Date')
plt.ylabel('NOx')
plt.xticks(rotation=75)
plt.show()
```



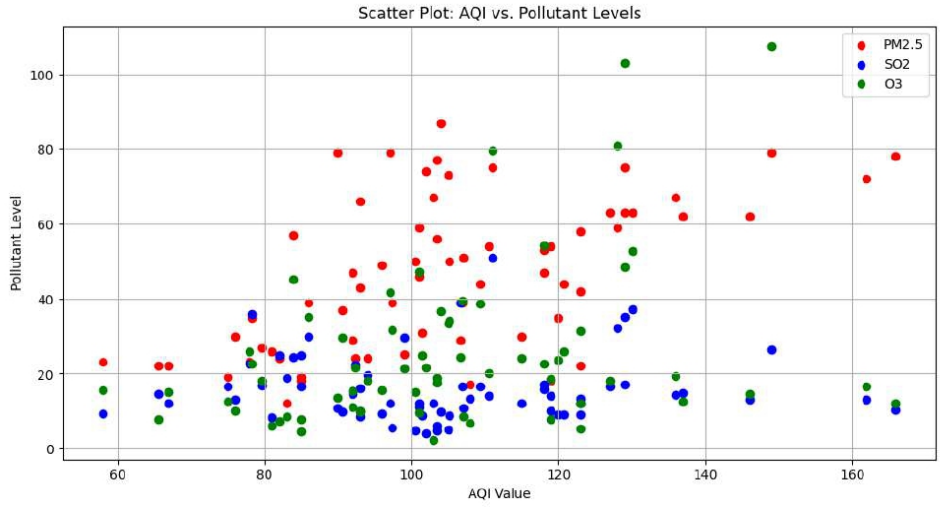
In [15]:
```
# Bar plots to compare AQI values across different dates or time periods
plt.figure(figsize=(12, 6))
plt.bar(df["Months"], df["AQI"], color="skyblue", label="AQI")
plt.title("AQI Comparison Across Months")
plt.xlabel("Months")
plt.ylabel("AQI Value")
plt.legend()
plt.xticks(rotation=75)
plt.show()
```



In [16]:
```
#Box plots to analyze the distribution of AQI values for different pollutant categories
plt.figure(figsize=(12, 6))
df.boxplot(color='blue')
plt.title("Distribution of AQI Values for Pollutants")
plt.show()
```



In [17]:
```
#Scatter plots to explore the relationship between AQI values and pollutant levels
plt.figure(figsize=(12, 6))
plt.scatter(df["AQI"], df["PM25"], c="red", label="PM2.5")
plt.scatter(df["AQI"], df["SO2"], c="blue", label="SO2")
plt.scatter(df["AQI"], df["O3"], c="green", label="O3")
plt.title("Scatter Plot: AQI vs. Pollutant Levels")
plt.xlabel("AQI Value")
plt.ylabel("Pollutant Level")
plt.legend()
plt.grid(True)
plt.show()
```



In [ ]:

Assignment No: 12

Title of the Assignment: Data Aggregation

Problem Statement: Analyzing Sales Performance by Region in a Retail Company

Dataset: "Retail_Sales_Data.csv"

Description: The dataset contains information about sales transactions in a retail company. It includes attributes such as transaction date, product category, quantity sold, and sales amount. The goal is to perform data aggregation to analyze the sales performance by region and identify the top-performing regions.

Tasks to Perform:

1. Import the "Retail_Sales_Data.csv" dataset.
2. Explore the dataset to understand its structure and content.
3. Identify the relevant variables for aggregating sales data, such as region, sales amount, and product category.
4. Group the sales data by region and calculate the total sales amount for each region.
5. Create bar plots or pie charts to visualize the sales distribution by region.
6. Identify the top-performing regions based on the highest sales amount.
7. Group the sales data by region and product category to calculate the total sales amount for each combination.
8. Create stacked bar plots or grouped bar plots to compare the sales amounts across different regions and product categories.

Objective of the Assignment: Students should be able to aggregate and summarize data to gain insights into sales performance by region, allowing for the identification of top-performing regions within the retail company.

Data aggregation is essential for summarizing and analyzing data at a higher level, providing insights into overall performance. In this assignment, students will follow these steps:

Step-1: Introduction to Data Aggregation
Understand the importance of data aggregation in analyzing sales performance. Define the problem and objectives for analyzing sales data by region.

Step-2: Identifying Key Metrics for Analysis
Identify the key metrics or attributes that are relevant to analyzing sales performance by region. These may include sales amount, quantity sold, or product categories.

Step-3: Grouping Data by Region
Group the sales data by region to create subsets of data that can be analyzed separately. Regions may be defined by geographical areas or any other relevant criteria.

Step-4: Aggregating Sales Data
Aggregate the sales data for each region, calculating metrics such as total sales amount, total quantity sold, average sales, etc. This step involves using aggregation functions like sum, mean, or count.

Step-5: Calculating Key Performance Indicators (KPIs)
Calculate key performance indicators (KPIs) that provide insights into sales performance. KPIs may include top-selling products, revenue growth, or sales trends.

Step-6: Visualizing Sales Performance
Create visualizations, such as bar charts or heatmaps, to represent the sales performance by region. Visualization helps in understanding the data and communicating findings effectively.

Conclusion: In this assignment, we have focused on data aggregation to analyze sales performance by region in a retail company. By identifying key metrics, grouping data by region, aggregating sales data, and calculating KPIs, students have learned how to gain insights into the top-performing regions. Data aggregation is a valuable technique for decision-makers in retail companies to identify strengths and areas for improvement.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df = pd.read_csv("retail_sales_data.csv", parse_dates=['invoice_date'])
df.head()
```

```
C:\Users\lenovo\AppData\Local\Temp\ipykernel_22672\2264015570.py:1: UserWarning: Parsing dates in DD/MM/YYYY f
ormat when dayfirst=False (the default) was specified. This may lead to inconsistently parsed dates! Specify a
format to ensure consistent parsing.
  df = pd.read_csv("retail_sales_data.csv", parse_dates=['invoice_date'])
```

Out[2]:

	invoice_no	customer_id	gender	age	category	quantity	price	payment_method	invoice_date	shopping_mall
0	I138884	C241288	Female	28	Clothing	5	1500.40	Credit Card	2022-05-08	Kanyon
1	I317333	C111565	Male	21	Shoes	3	1800.51	Debit Card	2021-12-12	Forum Istanbul
2	I127801	C266599	Male	20	Clothing	1	300.08	Cash	2021-09-11	Metrocity
3	I173702	C988172	Female	66	Shoes	5	3000.85	Credit Card	2021-05-16	Metropol AVM
4	I337046	C189076	Female	53	Books	4	60.60	Cash	2021-10-24	Kanyon

In [3]:
```
df.describe()
```

Out[3]:

	age	quantity	price
count	99457.000000	99457.000000	99457.000000
mean	43.427089	3.003429	689.256321
std	14.990054	1.413025	941.184567
min	18.000000	1.000000	5.230000
25%	30.000000	2.000000	45.450000
50%	43.000000	3.000000	203.300000
75%	56.000000	4.000000	1200.320000
max	69.000000	5.000000	5250.000000

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99457 entries, 0 to 99456
Data columns (total 10 columns):
 #   Column          Non-Null Count  Dtype         
---  ------          --------------  -----         
 0   invoice_no      99457 non-null  object        
 1   customer_id     99457 non-null  object        
 2   gender          99457 non-null  object        
 3   age             99457 non-null  int64         
 4   category        99457 non-null  object        
 5   quantity        99457 non-null  int64         
 6   price           99457 non-null  float64       
 7   payment_method  99457 non-null  object        
 8   invoice_date    99457 non-null  datetime64[ns]
 9   shopping_mall   99457 non-null  object        
dtypes: datetime64[ns](1), float64(1), int64(2), object(6)
memory usage: 7.6+ MB
```

In [5]:
```
df.isna().sum()
```

Out[5]:
```
invoice_no        0
customer_id       0
gender            0
age               0
category          0
quantity          0
price             0
payment_method    0
invoice_date      0
shopping_mall     0
dtype: int64
```

In [6]:
```
df.isnull().sum()
```

Out[6]:
```
invoice_no        0
customer_id       0
gender            0
age               0
category          0
quantity          0
price             0
payment_method    0
invoice_date      0
shopping_mall     0
dtype: int64
```

In [7]:
```
df.drop(["invoice_no", "customer_id", "gender", "age", "payment_method"], axis=1, inplace=True)
df.head()
```

Out[7]:

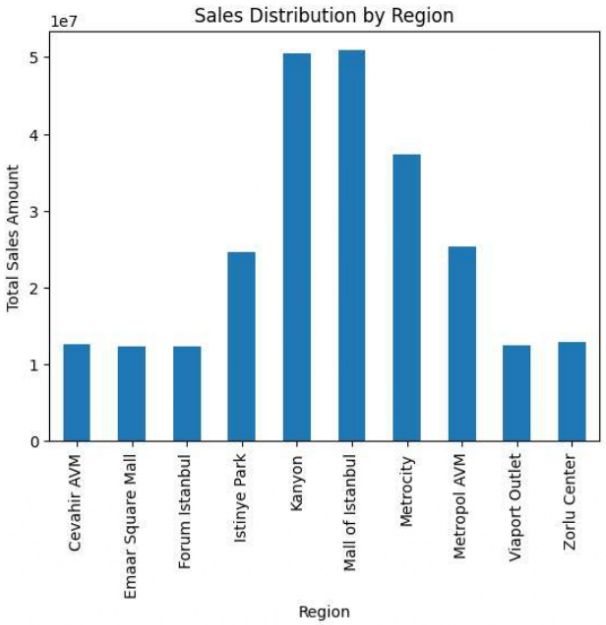
	category	quantity	price	invoice_date	shopping_mall
0	Clothing	5	1500.40	2022-05-08	Kanyon
1	Shoes	3	1800.51	2021-12-12	Forum Istanbul
2	Clothing	1	300.08	2021-09-11	Metrocity
3	Shoes	5	3000.85	2021-05-16	Metropol AVM
4	Books	4	60.60	2021-10-24	Kanyon

In [8]:
```
df['Sales'] = df['quantity']*df['price']
df.head()
```

Out[8]:

	category	quantity	price	invoice_date	shopping_mall	Sales
0	Clothing	5	1500.40	2022-05-08	Kanyon	7502.00
1	Shoes	3	1800.51	2021-12-12	Forum Istanbul	5401.53
2	Clothing	1	300.08	2021-09-11	Metrocity	300.08
3	Shoes	5	3000.85	2021-05-16	Metropol AVM	15004.25
4	Books	4	60.60	2021-10-24	Kanyon	242.40

In [9]:
```
# Group data by region and calculate total sales amount
region_sales = df.groupby("shopping_mall")["Sales"].sum()
region_sales.plot(kind="bar")
plt.title("Sales Distribution by Region")
plt.xlabel("Region")
plt.ylabel("Total Sales Amount")
plt.show()
```



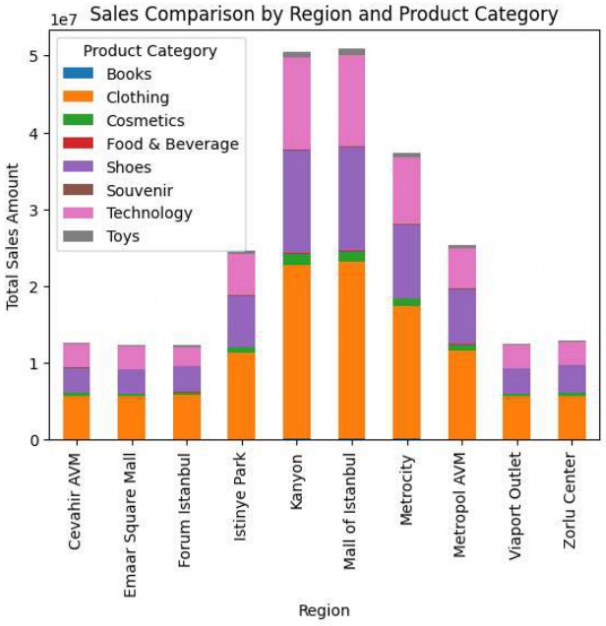
In [13]:
```
print(f"The top-performing region is: {region_sales.idxmax()}")
```

```
The top-performing region is: Mall of Istanbul
```

In [14]:
```
#Stacked bar plot to compare sales amounts across regions and categories

region_category_sales = df.groupby(["shopping_mall", "category"])["Sales"].sum().unstack()

region_category_sales.plot(kind="bar", stacked=True)
plt.title("Sales Comparison by Region and Product Category")
plt.xlabel("Region")
plt.ylabel("Total Sales Amount")
plt.legend(title="Product Category")
plt.show()
```



Assignment No: 13

Title of the Assignment: Time Series Data Analysis

Problem statement: Analysis and Visualization of Stock Market Data

Dataset: "Stock_Prices.csv"

Description: The dataset contains historical stock price data for a particular company over a period of time. It includes attributes such as date, closing price, volume, and other relevant features. The goal is to perform time series data analysis on the stock price data to identify trends, patterns, and potential predictors, as well as build models to forecast future stock prices.

Tasks to Perform:

1. Import the "Stock_Prices.csv" dataset.
2. Explore the dataset to understand its structure and content.
3. Ensure that the date column is in the appropriate format (e.g., datetime) for time series analysis.
4. Plot line charts or time series plots to visualize the historical stock price trends over time.
5. Calculate and plot moving averages or rolling averages to identify the underlying trends and smooth out noise.
6. Perform seasonality analysis to identify periodic patterns in the stock prices, such as weekly, monthly, or yearly fluctuations.
7. Analyze and plot the correlation between the stock prices and other variables, such as trading volume or market indices.
8. Use autoregressive integrated moving average (ARIMA) models or exponential smoothing models to forecast future stock prices.

Objective of the Assignment: Students should be able to analyze time series data, identify trends, patterns, and potential predictors in stock prices, and build forecasting models for future stock price prediction.

Time series data analysis is essential for understanding and predicting trends in sequential data. In this assignment, students will follow these steps:

Step-1: Introduction to Time Series Data Analysis

Understand the significance of time series data analysis, especially in the context of stock price forecasting. Define the problem and objectives for analyzing the stock price data.

Step-2: Exploratory Data Analysis (EDA) for Time Series Data

Perform EDA to understand the data, identify patterns, and visualize stock price trends over time. Techniques may include time plots, autocorrelation, and partial autocorrelation plots.

Step-3: Time Series Decomposition

Decompose the time series data into its components, such as trend, seasonality, and residuals. This step helps in understanding the underlying patterns in the data.

Step-4: Building Time Series Models
Build time series models such as Autoregressive Integrated Moving Average (ARIMA) or Exponential Smoothing to forecast future stock prices. Model selection and parameter tuning are crucial.

Step-5: Model Evaluation and Forecasting
Evaluate the performance of time series models using appropriate metrics and cross-validation techniques. Use the models to make future stock price forecasts.

Step-6: Data Visualization for Stock Price Analysis
Create visualizations, such as time series plots, forecasted trends, and prediction intervals, to represent the stock price analysis. Visualization enhances the understanding of data.

Step-7: Interpretation of Time Series Results
Interpret the results, identify significant predictors, and draw conclusions regarding stock price trends and predictions. Communicate findings effectively.

Conclusion: In this assignment, we have explored time series data analysis for historical stock price data. Students have learned how to perform EDA, decompose time series data, build forecasting models, evaluate model performance, and visualize stock price trends. Time series data analysis is essential for investors and analysts to make informed decisions in the stock market.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.seasonal import STL

df = pd.read_csv("fb.us.txt", parse_dates=['Date'], index_col=['Date'])
df.head()
```

Out[1]:

Date	Open	High	Low	Close	Volume	OpenInt
2012-05-18	42.05	45.00	38.00	38.23	580438450	0
2012-05-21	36.53	36.66	33.00	34.03	169418988	0
2012-05-22	32.61	33.59	30.94	31.00	101876406	0
2012-05-23	31.37	32.50	31.36	32.00	73678512	0
2012-05-24	32.95	33.21	31.77	33.03	42560731	0

In [2]:
```
df.describe()
```

Out[2]:

	Open	High	Low	Close	Volume	OpenInt
count	1381.000000	1381.000000	1381.000000	1381.000000	1.381000e+03	1381.0
mean	83.543667	84.384940	82.630555	83.543827	3.770716e+07	0.0
std	43.981535	44.161698	43.756570	44.015093	3.294912e+07	0.0
min	18.080000	18.270000	17.550000	17.730000	5.913000e+06	0.0
25%	46.750000	47.530000	45.960000	46.700000	1.843043e+07	0.0
50%	78.600000	79.690000	77.930000	78.790000	2.812660e+07	0.0
75%	117.710000	118.600000	116.700000	117.650000	4.601640e+07	0.0
max	182.360000	182.900000	180.570000	182.660000	5.804384e+08	0.0

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1381 entries, 2012-05-18 to 2017-11-10
Data columns (total 6 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   Open     1381 non-null   float64
 1   High     1381 non-null   float64
 2   Low      1381 non-null   float64
 3   Close    1381 non-null   float64
 4   Volume   1381 non-null   int64  
 5   OpenInt  1381 non-null   int64  
dtypes: float64(4), int64(2)
memory usage: 75.5 KB
```

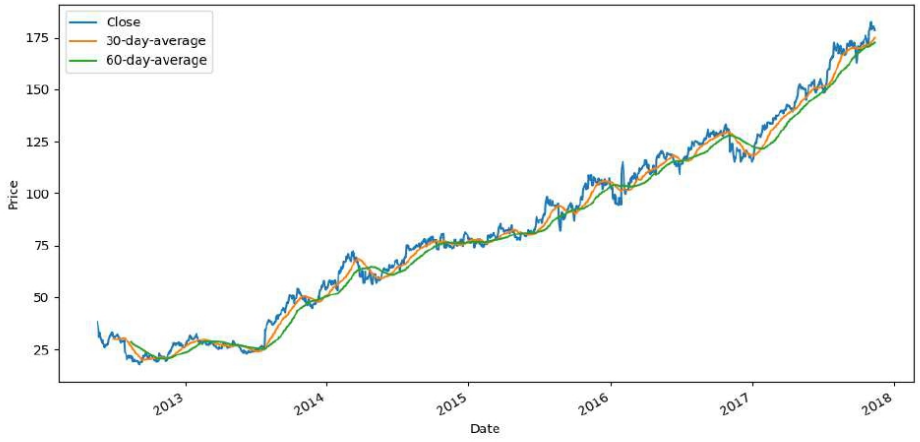
In [4]:
```
df.isna().sum()
```

Out[4]:
```
Open       0
High       0
Low        0
Close      0
Volume     0
OpenInt    0
dtype: int64
```

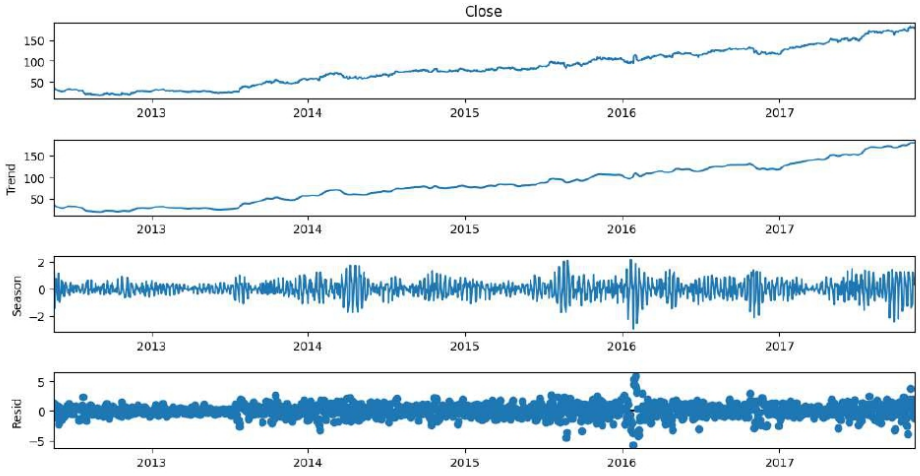
In [5]:
```
df.isnull().sum()
```

Out[5]:
```
Open       0
High       0
Low        0
Close      0
Volume     0
OpenInt    0
dtype: int64
```

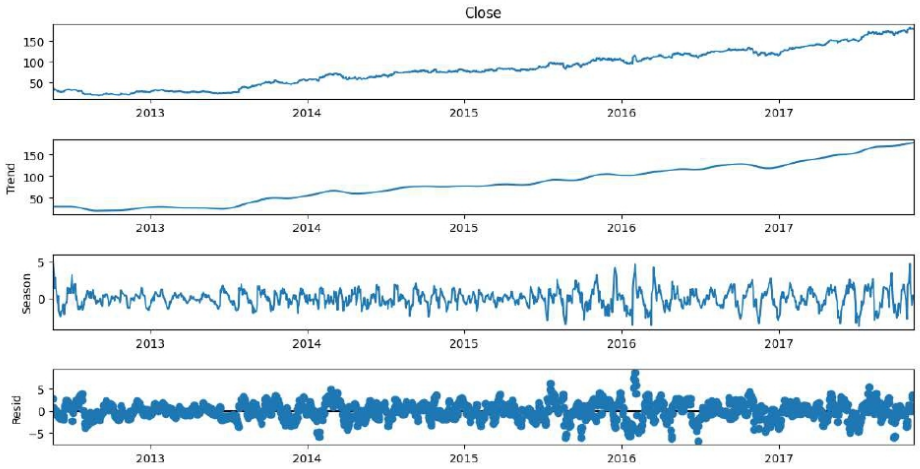
In [6]:
```
df["30-day-average"] = df['Close'].rolling(window=30).mean()
df["60-day-average"] = df['Close'].rolling(window=60).mean()
df[["Close", "30-day-average", "60-day-average"]].plot(figsize=(12,6), label="Moving averages")
plt.legend()
plt.xlabel("Date")
plt.ylabel("Price")
plt.show()
```



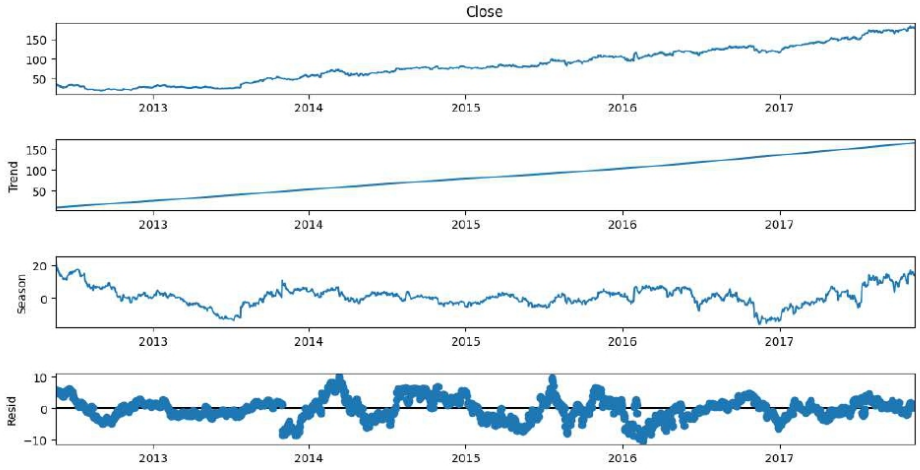
In [7]:
```
result = STL( df["Close"] , period=7).fit()
fig = result.plot()
fig.set_size_inches(12,6)
```



In [8]:
```
result = STL( df["Close"] , period=30).fit()
fig = result.plot()
fig.set_size_inches(12,6)
```

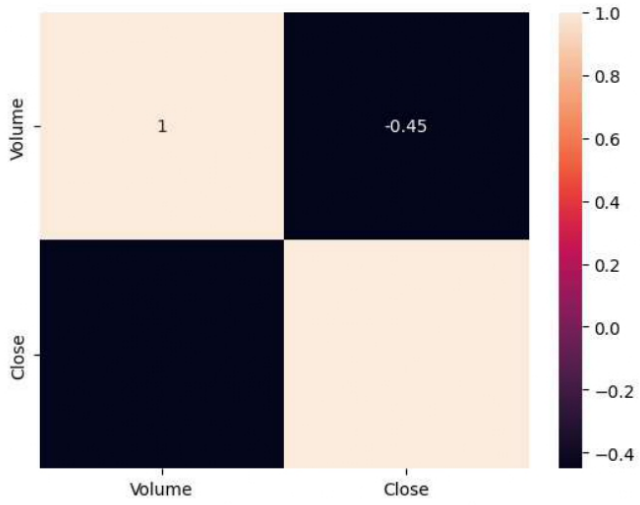


In [9]:
```
result = STL( df["Close"] , period=365).fit()
fig = result.plot()
fig.set_size_inches(12,6)
```



In [10]:
```
corrr = df[["Volume", "Close"]].corr()
sns.heatmap(corrr, annot=True)
```

Out[10]: <Axes: >



In [11]:
```
model = ARIMA( df["Close"] , order=(2,1,2))
results = model.fit()

forecast_steps = 365
forecast = results.forecast(forecast_steps)
forecast = forecast[1:]
forecast_index = pd.date_range(start=df.index[-1], periods=forecast_steps, closed='right')

plt.figure(figsize=(12,6))
plt.plot(forecast_index, forecast, label="Forecasted Value", color="Blue")
plt.plot(df.index, df['Close'], label="Actual Value", color="Red")
plt.legend()
plt.show()
```

```
C:\Users\lenovo\AppData\Local\Programs\Python\Python310\lib\site-packages\statsmodels\tsa\base\ts
a_model.py:471: ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
C:\Users\lenovo\AppData\Local\Programs\Python\Python310\lib\site-packages\statsmodels\tsa\base\ts
a_model.py:471: ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
C:\Users\lenovo\AppData\Local\Programs\Python\Python310\lib\site-packages\statsmodels\tsa\base\ts
a_model.py:471: ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
C:\Users\lenovo\AppData\Local\Programs\Python\Python310\lib\site-packages\statsmodels\tsa\base\ts
a_model.py:834: ValueWarning: No supported index is available. Prediction results will be given w
ith an integer index beginning at `start`.
  return get_prediction_index(
C:\Users\lenovo\AppData\Local\Temp\ipykernel_17364\930545061.py:6: FutureWarning: The behavior of
`series[i:j]` with an integer-dtype index is deprecated. In a future version, this will be treate
d as *label-based* indexing, consistent with e.g. `series[i]` lookups. To retain the old behavio
r, use `series.iloc[i:j]`. To get the future behavior, use `series.loc[i:j]`.
  forecast = forecast[1:]
C:\Users\lenovo\AppData\Local\Temp\ipykernel_17364\930545061.py:7: FutureWarning: Argument `close
d` is deprecated in favor of `inclusive`.
  forecast_index = pd.date_range(start=df.index[-1], periods=forecast_steps, closed='right')
```