JCEI's

JAIHIND COLLEGE OF ENGINEERING, KURAN.

Department Of

Artificial Intelligence and Data Science

LAB MANUAL

Data Modelling and Visualization (DMV), BE

Semester I

Prepared by:

Prof. Munde B. B.

Computer Laboratory – I

Course Code	Course Name	Teaching Scheme(Hrs./Week)	Credits
417525	Computer Laboratory I – Data Modeling and Visualization[DMV]	4	2

Course Objectives:

● Creating an emerging data model for the data to be stored in a database.

● Conceptualized representation of Data objects.

● Create associations between different data objects, and the rules.

● Organize data description, data semantics, and consistency constraints of data.

● Identifying data trends.

● Incorporate data visualization tools and reap transformative benefits in their critical areas of operations.

Course Outcomes:

After completion of the course, learners should be able to-

CO1: Summarize data analysis and visualization in the field of exploratory data science.

CO2: Analyze the characteristics and requirements of data and select an appropriate data model

CO3: Describe to load, clean, transform, merge and reshape data.

CO4: Design a probabilistic data modeling, interpretation, and analysis.

CO5: Evaluate time series data.

CO6: Integrate real world data analysis problems.

Table Of Contents

	Description: The goal is to interact with the OpenWeatherMap API to retrieve weather data for a specific location and perform data modeling 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 modeling 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.		
09.	**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	CO 3	32

	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 modeling.		
10.	**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 modeling. **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).	CO 4	48

	3. Perform data merging if additional datasets with relevant information are available (e.g., neighborhood 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 neighborhood or property type. 7. Identify and handle outliers or extreme values in the data that may affect the analysis or modeling process.		
11.	**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.	CO 5	68

	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 color schemes.		
12.	**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.	CO 6	85
13.	**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	CO 7	95

	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.		

Lab Assignment No.	07
Title	**Data Loading, Storage and File Formats .** **Problem Statement:** Analyzing Sales Data from MultipleFile Formats **Dataset:** Sales data in multiple file formats (e.g., CSV, Excel, JSON) **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 dataframe 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 behavior, or product performance.
Roll No.	
Class	BE AI & DS
Date Of Completion	
Subject	Computer Laboratory I[417525]
Assessment Marks	
Assessor's Sign	

Experiment No. 07

Aim : Data Loading, Storage and File Formats .

Problem Statement : Analyzing Sales Data from MultipleFile Formats. Perform Following tasks : 1.Load the Dataset. 2. Explore the structure and content of the loaded data. 3. Perform data cleaning operations. 4. Convert the data into a unified format. 5. Perform data transformation tasks. 6. Analyze the sales data by performing descriptive statistics. 7. Create visualizations.

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

Software Requirements : Jupyter Notebook.

Hardware Requirements : 6GB free disk space, 2GB RAM plus additional RAM for virtual machines, Intel 64 and AMD 64 Architectures.

Objectives : Efficiently load diverse sales data, Ensuring integrity, scalability, security, and analysis readiness across multiple file formats for seamless insights extraction.

Theory : Analysing sales data from multiple file formats can be a common task in business and data analysis. Sales data can come in various formats, including spreedsheets [Excel, CSV], databases and even text files.

Here's a step by step guide on how to analyze sales data from multiple file formats :

1. **Gather the Data -** Collect all the sales data files you need from various sources and formats.
2. **File format Identification –** Determine the formats of the data files you have common formats include Excel(.xlsx), CSV(.CSV), JSON(JSON), SQL databases and text files(.txt).
3. **Data Prepration –** If the data is not a format that a you can work with directly, you might need to clean and preprocess it. This includes removing duplicates, handling missing values, and standardizing data formats. It's often beneficial to have a consistent data structure across all files. This means having the same coloumns and data types in each model.
4. **Choose Analysis Tools –** Depending on your data format and the analysis you want to perform, choose appropriate analysis tools.
   For Example – 1. For Excel files, you can use Microsoft excel or google sheets.

2. For CSV files or databases, you might use python or R with libraries like pandas or SQL for database querying.
3. For text files or other custom formats, you may need to write custom parsing scripts.

5. Load Data – Import the data into your choosen analysis tool. This often involves reading the data from the file and loading it into data structures like dataframes or database tables.

6. Data Consolidation – Data consolidation refers to the process of combining data from different sources for or formats into a unified dataset. Data consolidation is specifically used for analysing sales for the integration of data sources, creating a single source of truth , data cleaning and transformation, Enhanced analysis capabilities, scalabilty and efficiency.

7. Visualization and Reporting - Create a visualizations such as charts, graphs, and dashboards to communicate key findings effectively. Design reports that summarizes the analysis results, including insigts, trends and actionable recommendations.

8. Interpretation and Actionable Insights – Interpret the analysis results in the context of business goals and objectives. Identify actionable insights and recommendations based on the findings to optimize sales strategies, improve efficiency, or address challenges.

9. Documentation and Iteration – Document the analysis process, methodologies used and assumptions made for transperancy and reproducibility. Iterate on the analysis as nedded, incorporating feedback new data, or challenging business requirements to refine insights and strategies over time.

By following this steps, you can effectively analyse sales data from multiple file formats and derive valuable insights to drive business decisions.

Implementation :

Step 1 - Load the Sales Dataset.

```
import pandas as pd

df = pd.read_csv(r"C:\Users\saira\Downloads\supermarket_sales - Sheet1.csv")

df.head()
```

```
Invoice ID          Branch   City       Customer type       Gender   Product line         Unit
price       Quantity Tax 5%  Total      Date       Time     Payment  cogs      gross margin
percentage          gross income        Rating
```

0 750-67-8428 A Yangon Member Female Health and beauty 74.69
7 26.1415 548.9715 1/5/2019 13:08 Ewallet 522.83 4.761905 26.1415
9.1
1 226-31-3081 C Naypyitaw Normal Female Electronic
accessories 15.28 5 3.8200 80.2200 3/8/2019 10:29 Cash 76.40
4.761905 3.8200 9.6
2 631-41-3108 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 16.2155 7.4
3 123-19-1176 A Yangon Member Male Health and beauty 58.22
8 23.2880 489.0480 1/27/2019 20:33 Ewallet 465.76
4.761905 23.2880 8.4
4 373-73-7910 A Yangon Normal Male Sports and travel 86.31
7 30.2085 634.3785 2/8/2019 10:37 Ewallet 604.17 4.761905 30.2085
5.3

df.tail()

Invoice ID Branch City Customer type Gender Product line Unit price Quantity
Tax 5% Total Date Time Payment cogs gross margin percentage
gross income Rating

995 233-67-5758 C Naypyitaw Normal Male Health and beauty
40.35 1 2.0175 42.3675 1/29/2019 13:46 Ewallet 40.35
4.761905 2.0175 6.2

996 303-96-2227 B Mandalay Normal Female Home and lifestyle
97.38 10 48.6900 1022.4900 3/2/2019 17:16 Ewallet
973.80 4.761905 48.6900 4.4

997 727-02-1313 A Yangon Member Male Food and beverages
31.84 1 1.5920 33.4320 2/9/2019 13:22 Cash 31.84 4.761905
1.5920 7.7

998 347-56-2442 A Yangon Normal Male Home and lifestyle
65.82 1 3.2910 69.1110 2/22/2019 15:33 Cash 65.82 4.761905
3.2910 4.1

999 849-09-3807 A Yangon Member Female Fashion accessories
88.34 7 30.9190 649.2990 2/18/2019 13:28 Cash 618.38
4.761905 30.9190 6.6

Step 2- Explore the Structure and Content.

df.describe()

	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

df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 17 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Invoice ID      1000 non-null   object
 1   Branch          1000 non-null   object
```

```
 2   City                    1000 non-null   object
 3   Customer type           1000 non-null   object
 4   Gender                  1000 non-null   object
 5   Product line            1000 non-null   object
 6   Unit price              1000 non-null   float64
 7   Quantity                1000 non-null   int64
 8   Tax 5%                  1000 non-null   float64
 9   Total                   1000 non-null   float64
 10  Date                    1000 non-null   object
 11  Time                    1000 non-null   object
 12  Payment                 1000 non-null   object
 13  cogs                    1000 non-null   float64
 14  gross margin percentage 1000 non-null   float64
 15  gross income            1000 non-null   float64
 16  Rating                  1000 non-null   float64
dtypes: float64(7), int64(1), object(9)
memory usage: 132.9+ KB
df.isnull().sum()
Invoice ID        0
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
```

```
df.duplicated().sum()
```

```
0
```

Step 4 - Convert the Data into a Unified Format.

```
df.columns = [col.lower() for col in df.columns]
df['date'] = pd.to_datetime(df['date'])
df['unit price'] = df['unit price'].astype(float)
df['quantity'] = df['quantity'].astype(int)
df['total'] = df['total'].astype(float)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 18 columns):
```

```
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   invoice id               1000 non-null   object
 1   branch                   1000 non-null   object
 2   city                     1000 non-null   object
 3   customer type            1000 non-null   object
 4   gender                   1000 non-null   object
 5   product line             1000 non-null   object
 6   unit price               1000 non-null   float64
 7   quantity                 1000 non-null   int32
 8   tax 5%                   1000 non-null   float64
 9   total                    1000 non-null   float64
 10  date                     1000 non-null   datetime64[ns]
 11  time                     1000 non-null   object
 12  payment                  1000 non-null   object
 13  cogs                     1000 non-null   float64
 14  gross margin percentage  1000 non-null   float64
 15  gross income             1000 non-null   float64
 16  rating                   1000 non-null   float64
 17  total sales              1000 non-null   float64
dtypes: datetime64[ns](1), float64(8), int32(1), object(8)
memory usage: 136.8+ KB
```

Step 5 - Perform Data Transformation and Analyze the Data.

```
df['total sales'] = df['unit price'] * df['quantity']

df
```

invoice id branch city customer type gender product line unit price quantity tax 5% total date time payment cogs gross margin percentage gross income rating total sales

0 750-67-8428 A Yangon Member Female Health and beauty 74.69 7 26.1415 548.9715 2019-01-05 13:08 Ewallet 522.83 4.761905 26.1415 9.1 522.83

1 226-31-3081 C Naypyitaw Normal Female Electronic accessories 15.28 5 3.8200 80.2200 2019-03-08 10:29 Cash 76.40 4.761905 3.8200 9.6 76.40

2 631-41-3108 A Yangon Normal Male Home and lifestyle 46.33 7 16.2155 340.5255 2019-03-03 13:23 Credit card 324.31 4.761905 16.2155 7.4 324.31

3 123-19-1176 A Yangon Member Male Health and beauty 58.22 8 23.2880 489.0480 2019-01-27 20:33 Ewallet 465.76 4.761905 23.2880 8.4 465.76

4 373-73-7910 A Yangon Normal Male Sports and travel 86.31 7 30.2085 634.3785 2019-02-08 10:37 Ewallet 604.17 4.761905 30.2085 5.3 604.17

... ...

995 233-67-5758 C Naypyitaw Normal Male Health and beauty 40.35 1 2.0175 42.3675 2019-01-29 13:46 Ewallet 40.35 4.761905 2.0175 6.2 40.35

996 303-96-2227 B Mandalay Normal Female Home and lifestyle 97.38 10 48.6900 1022.4900 2019-03-02 17:16 Ewallet 973.80 4.761905 48.6900 4.4 973.80

997 727-02-1313 A Yangon Member Male Food and beverages 31.84 1 1.5920 33.4320 2019-02-09 13:22 Cash 31.84 4.761905 1.5920 7.7 31.84

998 347-56-2442 A Yangon Normal Male Home and lifestyle 65.82 1 3.2910 69.1110 2019-02-22 15:33 Cash 65.82 4.761905 3.2910 4.1 65.82

999 849-09-3807 A Yangon Member Female Fashion accessories 88.34 7 30.9190 649.2990 2019-02-18 13:28 Cash 618.38 4.761905 30.9190 6.6 618.38

1000 rows × 18 columns

total_sales.sum()

307587.38

average_order_value = df['total sales'].mean()

average_order_value

307.58738

category_sales = df.groupby('product line')['total sales'].sum()

category_sales

product line

Electronic accessories 51750.03

Fashion accessories 51719.90

Food and beverages 53471.28

Health and beauty 46851.18

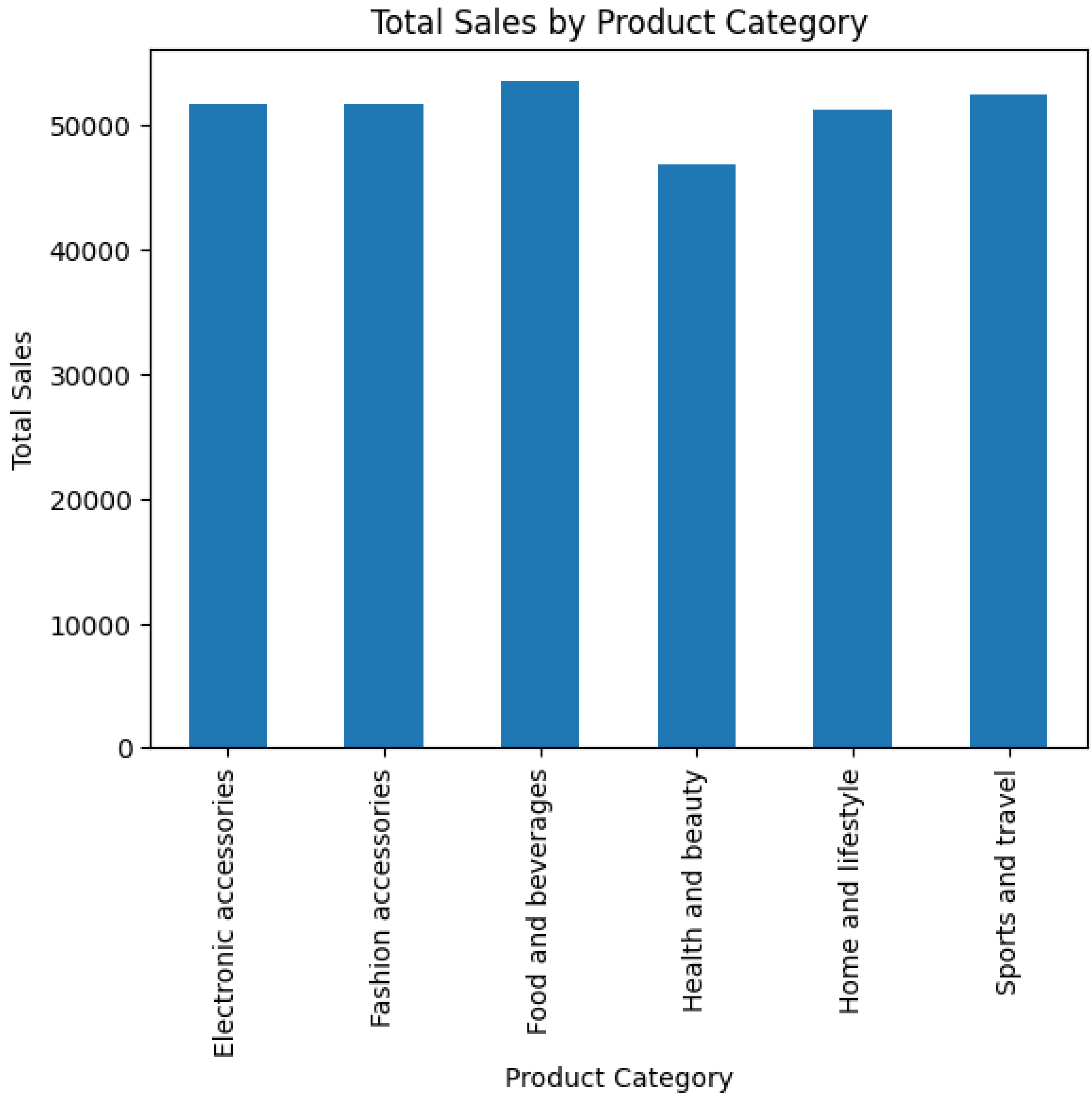
Home and lifestyle 51297.06

Sports and travel 52497.93

Name: total sales, dtype: float64

Step 6 -Create Visualizations.

```
import matplotlib.pyplot as plt
category_sales.plot(kind='bar', title='Total Sales by Product Category')
plt.xlabel('Product Category')
plt.ylabel('Total Sales')
plt.show()
```

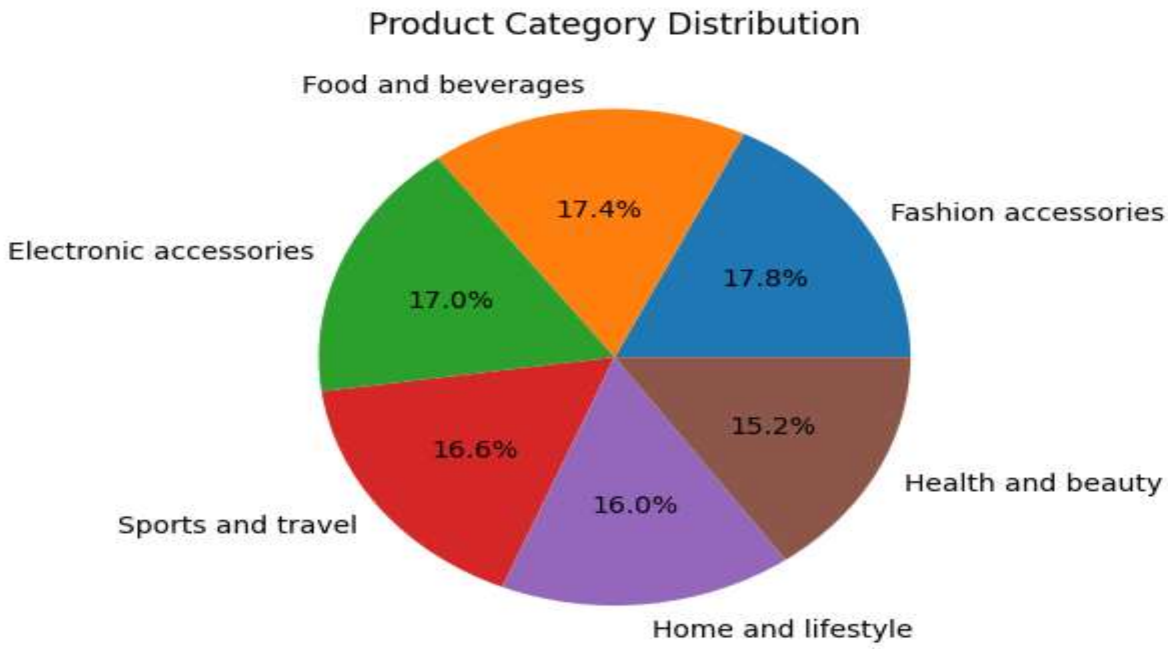



```
category_distribution = df['product line'].value_counts()
```

category_distribution.plot(kind='pie', title='Product Category Distribution', autopct='%1.1f%%')

plt.ylabel('')

plt.show()



df.boxplot(column='total sales', by='product line', grid=False, rot=45)

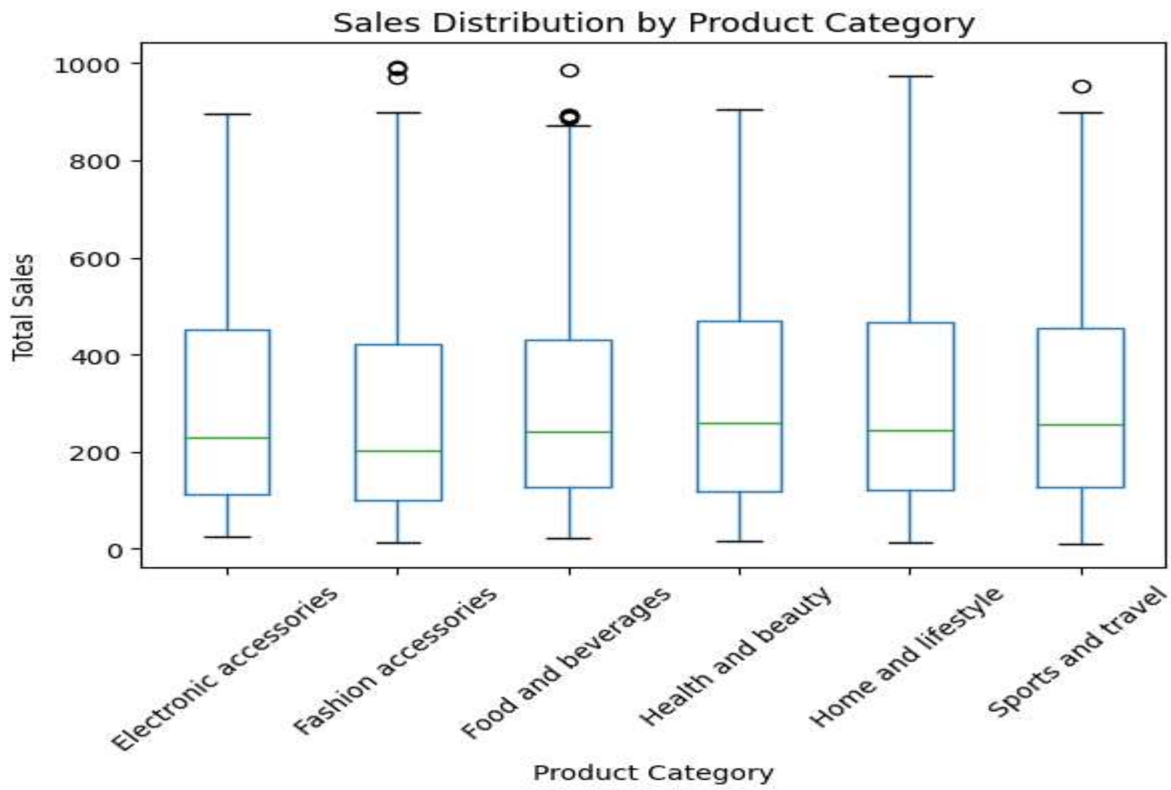
plt.title('Sales Distribution by Product Category')

plt.suptitle('')

plt.xlabel('Product Category')

plt.ylabel('Total Sales')

plt.show()



Conclusion – We can successfully step by step analyze sales data from CSV File formats.

Lab Assignment No.	08
Title	**Interacting with Web APIs** **Problem Statement:** Analyzing Weather Data from OpenWeatherMap API **Dataset:** Weather data retrieved from OpenWeatherMap API **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 modeling 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.
Roll No.	
Class	BE AI & DS
Date Of Completion	
Subject	Computer Laboratory I[417525]
Assessment Marks	
Assessor’s Sign	

Experiment No. 08.

Aim : Interacting with Web APIs.

Problem Statement: Analyzing Weather Data from OpenWeatherMap API, and perform following tasks – 1. Register and obtain API key from OpenWeatherMap. 2. Interact with the OpenWeatherMap API. 3. Extract relevant weather attributes. 4. Clean and preprocess the retrieved data. 5. Perform data modeling to analyze weather patterns. 6. Visualize the weather data. 7. Apply data aggregation techniques. 8. Incorporate geographical information. 9. Explore and visualize relationships between weather attributes.

Dataset: Weather data retrieved from OpenWeatherMap API.

Software Requirements : Operating System, Python, Python Libraries.

Hardware Requirements : Processor, RAM, Storage, and Internet Connection.

Objective : i) Understand the fundamentals of web API's and how to interact with them. ii) Retrive and parse JSON data from the open weather map API. iii) Extract and display key weather metrics. iv) Visualize weather trends using a plotting library.

Theory : What is an API's ?

An API, or Application Programming Interface, is a set of rules and tools that allows different software applications to communicate with each other. Think of it like a menu in a restaurant: it lists the dishes you can order and tells the kitchen how to prepare them. Similarly, an API defines the methods and data formats that applications can use to request and exchange information. APIs are used in many contexts, from web services to operating systems, and they play a crucial role in enabling different software systems to work together seamlessly.

RESTFUL API's

A RESTful API (Representational State Transfer) is a specific type of API that adheres to the principles of REST architecture. REST is a set of constraints and guidelines for creating web services that are scalable, stateless, and easily maintainable. REST is a design pattern for networked applications that promotes scalability, stateless communication, and simplicity. RESTful APIs are popular because they are simple, flexible, and use standard HTTP methods, making them easy to implement and consume.

Key Features of RESTFUL API's

1. **Statelessness**: Each request from a client to a server must contain all the information needed to understand and process the request. The server does not store any state between requests. This makes the API more scalable because each request is independent.
2. **Client-Server Architecture**: The client and server operate independently of each other. The client makes requests to the server, and the server responds with the requested data. This separation allows for scalability and flexibility.
3. **Cacheability**: Responses from the server can be explicitly marked as cacheable or non-cacheable. This helps improve performance by reducing the need for repeated requests to the server.

Open weather Map API's

OpenWeatherMap is a service that provides weather data through APIs. These APIs allow developers to integrate weather information into their applications or websites. OpenWeatherMap offers several types of APIs to access different kinds of weather data.

Endpoints –

Current Weather Data API: https://api.openweathermap.org/data/2.5/weather

Forecast API: https://api.openweathermap.org/data/2.5/forecast

Historical Weather Data API: https://api.openweathermap.org/data/2.5/onecall/timemachine

Response Format

The response format for the OpenWeatherMap API is typically in JSON (JavaScript Object Notation), which is a lightweight data-interchange format that's easy for humans to read and write and easy for machines to parse and generate.

Implementation

Step No. 1 – Regiester and obtain API Key

Sign up on the open weather map website.

Navigate to the API’s section and generate an API’s key.

Step No. 02 - Interact with the OpenWeatherMap API using the API key to retrieve weather data for a specific location.

```
import requests
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime
import seaborn as sns

api_key = 'c15c676d2f9d0dada63d7fa10c76ce01'
location = 'India'
url =
f'http://api.openweathermap.org/data/2.5/forecast?q=India&appid=c15c676d2f9d0dada63d7f
a10c76ce01&units=metric'

response = requests.get(url)
data = response.json()
if response.status_code == 200:
    "Data retrieved successfully for India"
else:
    data = response.json()
    f"Error: {data.get('message', 'Failed to retrieve data')}"

data
```

Step No. 03 - Extract Relevant Weather Attributes.

```
weather_list = data['list']
weather_data = {'datetime': [], 'teampreature': [], 'humidity': [], 'wind_speed': [],
'precipitation': []}
for entry in weather_list:
    weather_data['datetime'].append(datetime.fromtimestamp(entry['dt']))
    weather_data['teampreature'].append(entry['main']['temp'])
    weather_data['humidity'].append(entry['main']['humidity'])
    weather_data['wind_speed'].append(entry['wind']['speed'])
    precipitation = entry['rain'].get('3h', 0) if 'rain' in entry else 0
    weather_data['precipitation'].append(precipitation)

import pandas as pd
df = pd.DataFrame(weather_data)
df.head()
```

	datetime	teampreature	humidity	wind_speed	precipitation
0	2024-07-12 14:30:00	15.82	95	1.35	2.38
1	2024-07-12 17:30:00	18.21	90	1.74	3.68
2	2024-07-12 20:30:00	20.88	83	2.04	3.51
3	2024-07-12 23:30:00	18.24	95	1.89	1.91
4	2024-07-13 02:30:00	14.46	99	1.71	4.05

Step No. 04 - Clean and Preprocess the Data.

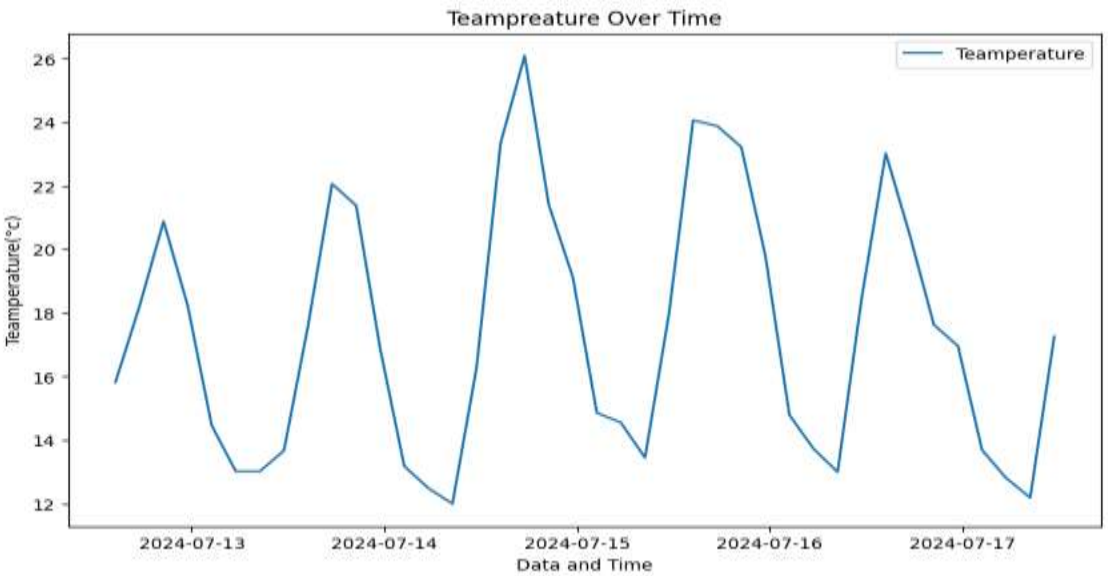
```
df.isnull().sum()

datetime        0
teampreature    0
humidity        0
wind_speed      0
precipitation   0
dtype: int64
```

Step No. 05 - Perform Data Modelling.

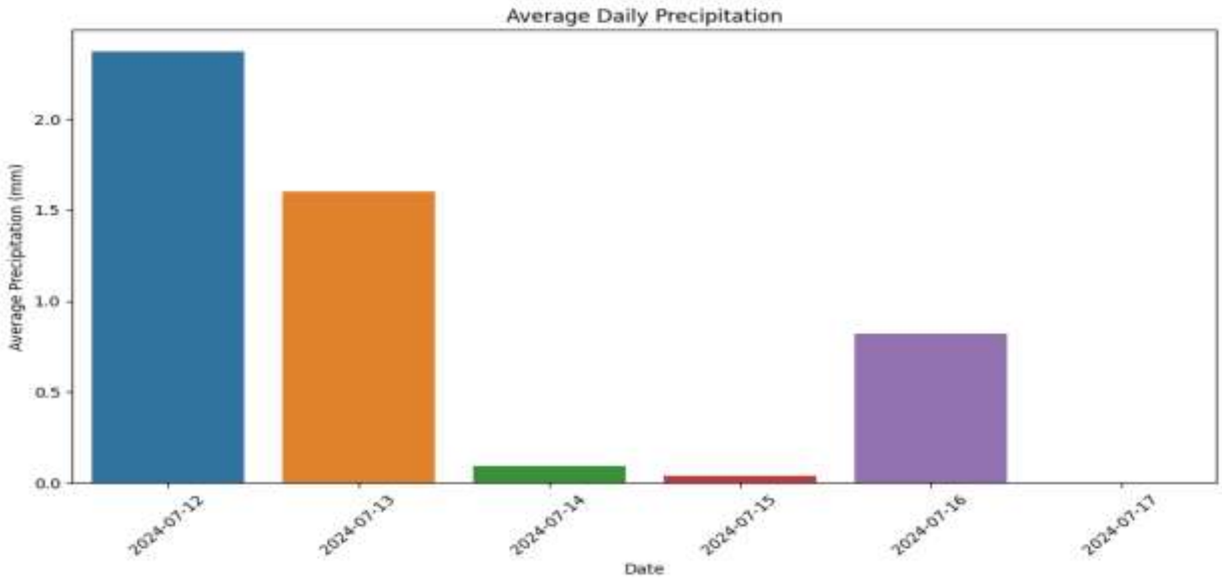
```
avg_temp = df['teampreature'].mean()
avg_temp
17.82775
max_temp = df['teampreature'].max()
min_temp = df['teampreature'].min()
max_temp
25.93
min_temp
11.81
```

Step no. 06 - Visualize the Weather Data.

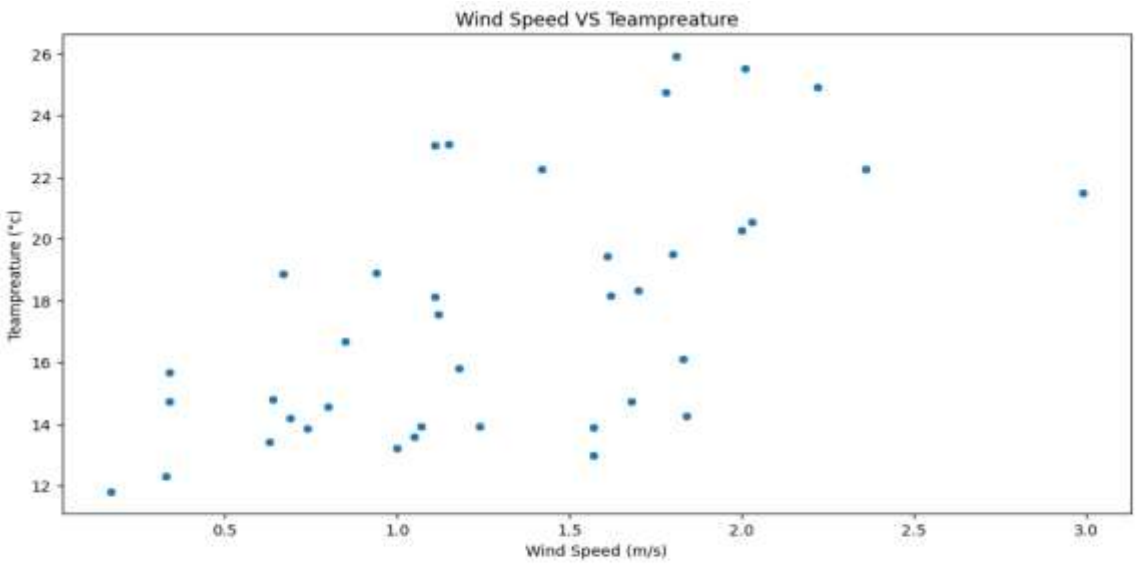
```
# 1. Line Chart
plt.figure(figsize=(10, 6))
plt.plot(df['datetime'], df['teampreature'], label='Teamperature')
plt.xlabel('Data and Time')
plt.ylabel('Teamperature(°c)')
plt.title('Teampreature Over Time')
plt.legend()
plt.show()
```



```
# 2. Bar Plot
plt.figure(figsize=(12, 6))
sns.barplot(data=daily_precipitation, x='date', y='precipitation')
plt.xlabel('Date')
plt.ylabel('Average Precipitation (mm)')
plt.title('Average Daily Precipitation')
plt.xticks(rotation=45)
plt.show()
```




```
# 3. Scatter Plot.
plt.figure(figsize=(12, 6))
sns.scatterplot(data=df, x='wind_speed', y='teampreature')
plt.xlabel('Wind Speed (m/s)')
plt.ylabel('Teampreature (°c)')
plt.title('Wind Speed VS Teampreature')
plt.show()
```



Step No. 07 - Apply Data Aggregation Techniques.

```
# 1. Daily Aggregation
daily_weather = df.resample('D').agg({'teampreature': 'mean', 'humidity': 'mean', 'wind_speed': 'max'})
daily_weather.head()
```

	teampreature	humidity	wind_speed
datetime			
2024-07-12	18.17000	90.600	2.03
2024-07-13	16.73500	83.000	2.99
2024-07-14	18.30750	70.125	2.22
2024-07-15	19.69875	68.875	2.01
2024-07-16	17.78000	81.250	1.62

```
# 2. Monthly Aggregation.
monthly_weather = df.resample('M').agg({'teampreature': 'mean', 'humidity': 'mean', 'wind_speed': 'max'})
monthly_weather.head()
```

teampreature	humidity	wind_speed	
datetime			
2024-07-31	17.82775	78.725	2.99

```
# 3. Seasnoal Aggregation.
def get_season(month):
    if month in [12, 1, 2]:
        return 'Winter'
    elif month in [3, 4, 5]:
        return 'Spring'
    elif month in [6, 7, 8]:
        return 'Summer'
    else:
        return 'Autumn'

df['season'] = df.index.month.map(get_season)

seasonal_weather = df.groupby('season').agg({'temperature': 'mean', 'humidity': 'mean', 'wind_speed': 'max'})
seasonal_weather
```

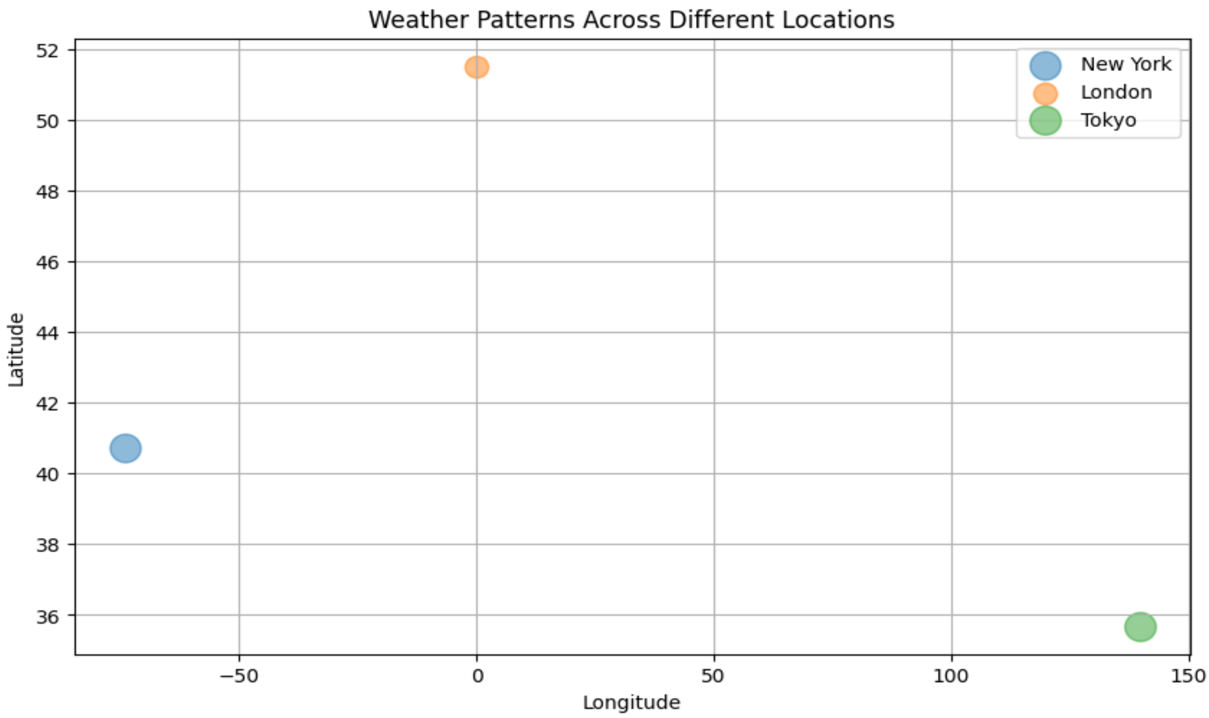
	temperature	humidity	wind_speed
season			
Summer	26.0	84.2	9

Step No. 08 - Incorporate Geographical Information.

```
pip install folium
```

```
# 1. Fetch Weather Data for Multiple Locations.
# Replace 'your_api_key' with your actual API key
api_key = 'c15c676d2f9d0dada63d7fa10c76ce01'
locations = [
    {'name': 'New York', 'lat': 40.7128, 'lon': -74.0060},
    {'name': 'London', 'lat': 51.5074, 'lon': -0.1278},
    {'name': 'Tokyo', 'lat': 35.6895, 'lon': 139.6917},
    # Add more locations as needed
]
weather_data = []
for loc in locations:
    url =
f'http://api.openweathermap.org/data/2.5/weather?lat={loc["lat"]}&lon={loc["lon"]}&appid=
{api_key}&units=metric'
    response = requests.get(url)
    data = response.json()
    if response.status_code == 200:
        weather_data.append({
            'name': loc['name'],
            'temperature': data['main']['temp'],
            'humidity': data['main']['humidity'],
            'wind_speed': data['wind']['speed'],
            'latitude': loc['lat'],
            'longitude': loc['lon']
        })
    else:
        print(f"Error fetching data for {loc['name']}: {data.get('message', 'Unknown error')}")

weather_data

[{'name': 'New York',
  'temperature': 23.3,
  'humidity': 73,
  'wind_speed': 1.54,
  'latitude': 40.7128,
  'longitude': -74.006},
 {'name': 'London',
  'temperature': 13.5,
  'humidity': 87,
  'wind_speed': 4.12,
  'latitude': 51.5074,
  'longitude': -0.1278},
 {'name': 'Tokyo',
  'temperature': 24.13,
  'humidity': 91,
  'wind_speed': 1.03,
  'latitude': 35.6895,
  'longitude': 139.6917}]

# 2. Create a Geospatial visualization using Folium.
```

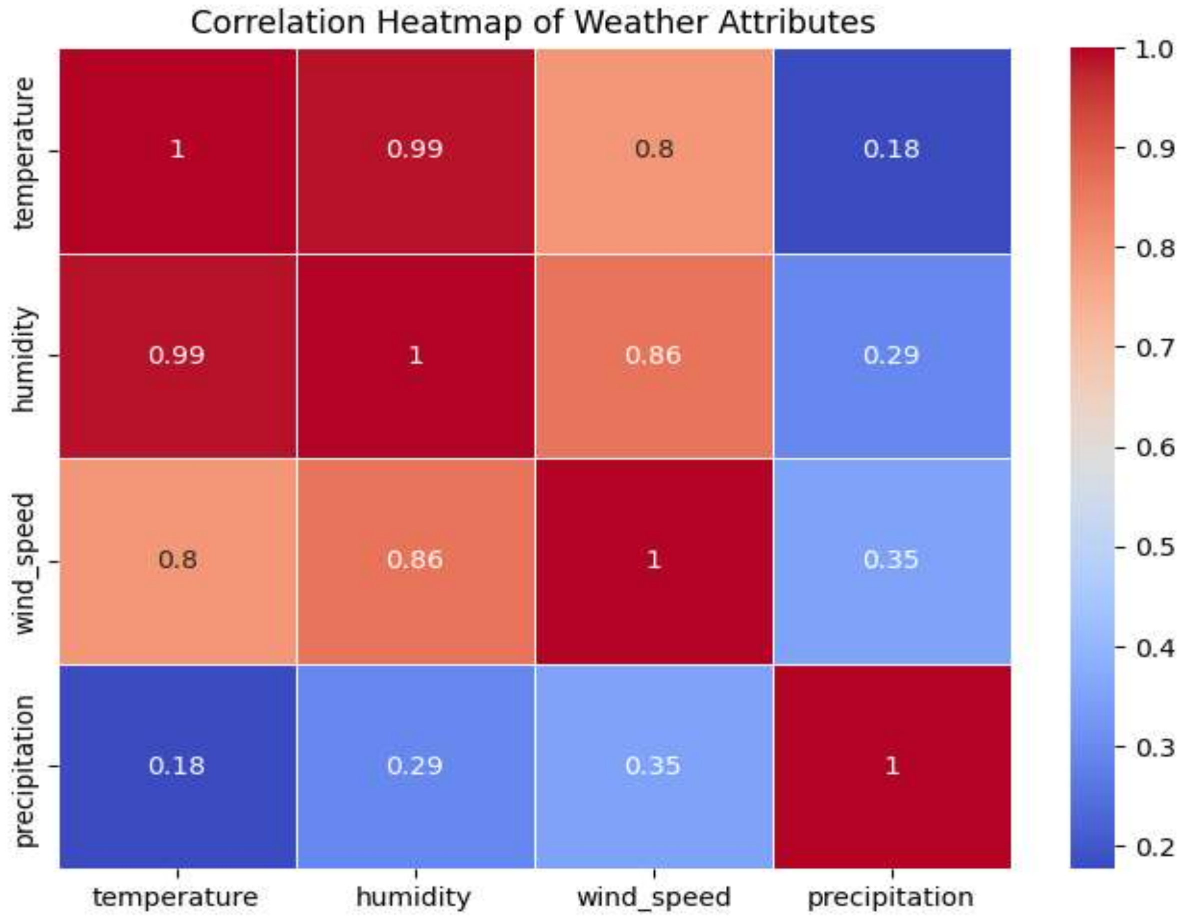
```
# Create a map centered at a specific location (e.g., New York)
map_center = [40.7128, -74.0060]
mymap = folium.Map(location=map_center, zoom_start=3)

# Add markers for each location with weather information
for data in weather_data:
    popup_text = f"<b>{data['name']}</b><br>Temperature:
{data['temperature']}°C<br>Humidity: {data['humidity']}%<br>Wind Speed:
{data['wind_speed']} m/s"
    folium.Marker(location=[data['latitude'], data['longitude']],
popup=popup_text).add_to(mymap)

# Save the map as an HTML file
mymap.save('weather_map.html')
mymap

# 3. Viualize Weather Patterns on a Static Map using Matplotlib.
# Plot each location with a scatter plot based on temperature
plt.figure(figsize=(10, 6))
for data in weather_data:
    plt.scatter(data['longitude'],data['latitude'],s=data['temperature']*10,alpha=0.5,
label=data['name'])
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Weather Patterns Across Different Locations')
plt.legend()
plt.grid(True)
plt.show()
```



Step No. 09 - Explore and Visualize Realationships.

```
# Calculate correlation matrix
correlation_matrix = df[['temperature', 'humidity', 'wind_speed', 'precipitation']].corr()
print(correlation_matrix)
# Plot heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Heatmap of Weather Attributes')
plt.show()
```

```
              temperature  humidity  wind_speed  precipitation
temperature      1.000000  0.988483    0.800000       0.176777
humidity         0.988483  1.000000    0.864923       0.294875
wind_speed       0.800000  0.864923    1.000000       0.353553
precipitation    0.176777  0.294875    0.353553       1.000000
```



Conclusion – Thus we can successfully interacting and analyzing weather data from open weather map API by using this following steps, and also the open weather map API provides valuable real time weather data, this data we can easily used for various analytical purposes.

Lab Assignment No.	09
Title	**Data Cleaning and Preparation** **Problem Statement:** Analyzing Customer Churn in a Telecommunications Company **Dataset:** "Telecom_Customer_Churn.csv" **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 modeling.
Roll No.	
Class	BE AI & DS
Date Of Completion	
Subject	Computer Laboratory I[417525]
Assessment Marks	
Assessor's Sign	

Experiment No. 09

Aim : Data Cleaning and Preparation.

Problem Statement : Analyzing Customer Churn in a Telecommunications Company, and perform following tasks – 1. Import the dataset. 2. Explore the dataset. 3. Handle missing values. 4. Remove any duplicates. 5. Check for inconsistent data. 6. Convert columns to the correct data types. 7. Identify and handle outliers. 8. Perform feature engineering. 9. Normalize or scale the data. 10. Split the dataset into training and testing sets. and 11. Export the cleaned dataset.

Dataset: "Telecom_Customer_Churn.csv"

Software Requirements : Python and Jupyter notebook.

Hardware Requirements : 8 GB RAM, Intel I5 Processor, and Storage.

Objectives : i) Identify and handle missing values. ii) Convert categorical variables to numerical formats and scale numerical features. iii) Create a new feature that might be useful for predicting customer churn.

Theroy : Data cleaning and prepration are essential steps in any data analysis project. They ensure that the dataset is accurate, consistent, and ready for further analysis and modelling.

1. Data Cleaning

Data cleaning is an essential step in data analysis that involves preparing and improving raw data to make it suitable for analysis. Here's a broad overview of the process:

1. **Remove Duplicates**: Ensure there are no redundant records in your dataset. Duplicates can skew results and analyses.
2. **Handle Missing Values**: Address gaps in data by either filling them in with a statistical measure (mean, median) or by using algorithms that can handle missing values, or by removing the incomplete records if appropriate.
3. **Correct Errors**: Identify and fix errors or inconsistencies in the data. This includes correcting typos, standardizing formats (e.g., date formats), and ensuring data consistency.
4. **Normalize Data**: Transform data into a common format or scale. For example, converting all text to lowercase, or standardizing units of measurement.

5. **Filter Outliers**: Detect and handle outliers—data points that significantly deviate from the norm. Depending on your analysis, you might choose to remove or adjust them.
6. **Validate Data**: Ensure that the data conforms to the expected format and constraints. This could include checking data types, ranges, or valid values.
7. **Standardize Data**: Ensure consistency in the dataset. For example, standardizing names, address formats, or categorical variables.
8. **Integrate Data**: Combine data from multiple sources and ensure they are compatible and consistent.
9. **Create Data Documentation**: Document the cleaning process, including any transformations or modifications made to the data. This ensures transparency and helps in future data management tasks.
10. **Automate Where Possible**: Use scripts or data cleaning tools to automate repetitive tasks, which can save time and reduce errors.

2. Data Transformation

Data transformation is a crucial process in data preparation that involves converting data from its original format or structure into a format that is more suitable for analysis or further processing. Here’s a detailed look at common data transformation techniques:

1. **Normalization**: Adjusting the scale of numerical data to ensure uniformity. For example, scaling values to a range between 0 and 1 or converting them to z-scores.
2. **Standardization**: Transforming data to have a mean of 0 and a standard deviation of This is particularly useful when comparing data that originally had different units or scales.
3. **Aggregation**: Combining multiple data points into a summary metric. For instance, calculating the average, sum, or count of values over a period or across categories.
4. **Encoding**: Converting categorical data into numerical format. Common methods include one-hot encoding (creating binary columns for each category) or label encoding (assigning a unique number to each category).

3. Exploratory Data Analysis [EDA]

Exploratory Data Analysis (EDA) is a crucial phase in data analysis that involves examining and understanding the data before applying formal statistical or machine learning models. The goal of EDA is to uncover patterns, spot anomalies, test hypotheses, and check assumptions through various techniques.

4. **Feature Engineering**

   Feature engineering is a crucial step in the data preprocessing phase of machine learning and data analysis. It involves creating, modifying, or selecting features (variables) from raw data to improve the performance and effectiveness of predictive models.

Implementation

Step No.1 - Import the Dataset.

```
import pandas as pd
import numpy as np
df = pd.read_csv(r"C:\Users\saira\Downloads\Telco-Customer-Churn.csv")
df.head()
```

```
customerID	gender	SeniorCitizen	Partner	Dependents	tenure
	PhoneService	MultipleLines	InternetService	OnlineSecurity	...
	DeviceProtection	TechSupport	StreamingTV	StreamingMovies
	Contract	PaperlessBilling	PaymentMethod	MonthlyCharges	TotalCharges
	Churn
0	7590-VHVEG	Female	0	Yes	No	1	No	No
phone service	DSL	No	...	No	No	No	No	Month-
to-month	Yes	Electronic check	29.85	29.85	No
1	5575-GNVDE	Male	0	No	No	34	Yes	No
	DSL	Yes	...	Yes	No	No	No	One year	No
	Mailed check	56.95	1889.5	No
2	3668-QPYBK	Male	0	No	No	2	Yes	No
	DSL	Yes	...	No	No	No	No	Month-to-month
	Yes	Mailed check	53.85	108.15	Yes
3	7795-CFOCW	Male	0	No	No	45	No	No
phone service	DSL	Yes	...	Yes	Yes	No	No	One
year	No	Bank transfer (automatic)	42.30	1840.75	No
4	9237-HQITU	Female	0	No	No	2	Yes	No
	Fiber optic	No	...	No	No	No	No	Month-
to-month	Yes	Electronic check	70.70	151.65	Yes
5 rows × 21 columns
```

```
df.tail()
```

```
customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService
	MultipleLines	InternetService	OnlineSecurity	...	DeviceProtection
```

TechSupport StreamingTV StreamingMovies Contract PaperlessBilling PaymentMethod MonthlyCharges TotalCharges Churn

7038 6840-RESVB Male 0 Yes Yes 24 Yes Yes DSL Yes ... Yes Yes Yes Yes One year Yes Mailed check 84.80 1990.5 No

7039 2234-XADUH Female 0 Yes Yes 72 Yes Yes Fiber optic No ... Yes No Yes Yes One year Yes Credit card (automatic) 103.20 7362.9 No

7040 4801-JZAZL Female 0 Yes Yes 11 No No phone service DSL Yes ... No No No No Month-to-month Yes Electronic check 29.60 346.45 No

7041 8361-LTMKD Male 1 Yes No 4 Yes Yes Fiber optic No ... No No No No Month-to-month Yes Mailed check 74.40 306.6 Yes

7042 3186-AJIEK Male 0 No No 66 Yes No Fiber optic Yes ... Yes Yes Yes Yes Two year

Step No.2 - Explore the Dataset.

```
df.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 7043 entries, 0 to 7042

Data columns (total 21 columns):

 #   Column            Non-Null Count  Dtype

---  ------            --------------  -----

 0   customerID        7043 non-null   object

 1   gender            7043 non-null   object

 2   SeniorCitizen     7043 non-null   int64

 3   Partner           7043 non-null   object

 4   Dependents        7043 non-null   object
```

```
 5   tenure           7043 non-null   int64
 6   PhoneService     7043 non-null   object
 7   MultipleLines    7043 non-null   object
 8   InternetService  7043 non-null   object
 9   OnlineSecurity   7043 non-null   object
 10  OnlineBackup     7043 non-null   object
 11  DeviceProtection 7043 non-null   object
 12  TechSupport      7043 non-null   object
 13  StreamingTV      7043 non-null   object
 14  StreamingMovies  7043 non-null   object
 15  Contract         7043 non-null   object
 16  PaperlessBilling 7043 non-null   object
 17  PaymentMethod    7043 non-null   object
 18  MonthlyCharges   7043 non-null   float64
 19  TotalCharges     7043 non-null   object
 20  Churn            7043 non-null   object
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
df.columns
Index(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents',
       'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',
       'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling',
       'PaymentMethod', 'MonthlyCharges', 'TotalCharges', 'Churn'],
```

dtype='object')

df.describe()

	SeniorCitizen	tenure	MonthlyCharges
count	7043.000000	7043.000000	7043.000000
mean	0.162147	32.371149	64.761692
std	0.368612	24.559481	30.090047
min	0.000000	0.000000	18.250000
25%	0.000000	9.000000	35.500000
50%	0.000000	29.000000	70.350000
75%	0.000000	55.000000	89.850000
max	1.000000	72.000000	118.750000

Step No.3 - Handle Missing Values.

df.isnull().sum()

customerID 0

gender 0

SeniorCitizen 0

Partner 0

Dependents 0

tenure 0

PhoneService 0

MultipleLines 0

InternetService 0

OnlineSecurity 0

OnlineBackup 0

```
DeviceProtection    0
TechSupport         0
StreamingTV         0
StreamingMovies     0
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges        0
Churn               0
dtype: int64
```

Step No.4 - Remove Duplicate Records.

```
df.duplicated()
0       False
1       False
2       False
3       False
4       False
        ...
7038    False
7039    False
7040    False
7041    False
7042    False
```

Length: 7043, dtype: bool

df = df.drop_duplicates()

df.duplicated().sum()

0

Step No.5 - Check for Inconsistent Data.

```
def standardize_text(df, text_columns):
    for col in text_columns:
        df[col] = df[col].str.strip().str.lower()
    return df
```

text_columns = df.select_dtypes(include='object').columns

text_columns

```
Index(['customerID', 'gender', 'Partner', 'Dependents', 'PhoneService',
       'MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup',
       'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies',
       'Contract', 'PaperlessBilling', 'PaymentMethod', 'TotalCharges',
       'Churn'],
      dtype='object')
```

df = standardize_text(df, text_columns)

df.head()

```
customerID  gender  SeniorCitizen  Partner  Dependents  tenure  PhoneService
    MultipleLines  InternetService  OnlineSecurity  ...  DeviceProtection
    TechSupport  StreamingTV  StreamingMovies  Contract  PaperlessBilling
    PaymentMethod  MonthlyCharges  TotalCharges  Churn

0   7590-vhveg  female  0  yes  no  1  no  no phone service  dsl
    no  ...  no  no  no  no  month-to-month  yes  electronic
check  29.85  29.85  no
```

1 5575-gnvde male 0 no no 34 yes no dsl yes ... yes no no no one year no mailed check 56.95 1889.5 no

2 3668-qpybk male 0 no no 2 yes no dsl yes ... no no no no month-to-month yes mailed check 53.85 108.15 yes

3 7795-cfocw male 0 no no 45 no no phone service dsl yes ... yes yes no no one year no bank transfer (automatic) 42.30 1840.75 no

4 9237-hqitu female 0 no no 2 yes no fiber optic no ... no no no no month-to-month yes electronic check 70.70 151.65 yes

5 rows × 21 columns

Step No.6 - Convert Columns to Correct Data Types.

df.dtypes

customerID object

gender object

SeniorCitizen int64

Partner object

Dependents object

tenure int64

PhoneService object

MultipleLines object

InternetService object

OnlineSecurity object

OnlineBackup object

DeviceProtection object

```
TechSupport         object
StreamingTV         object
StreamingMovies     object
Contract            object
PaperlessBilling    object
PaymentMethod       object
MonthlyCharges      float64
TotalCharges        object
Churn               object
dtype: object
```

```
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors='coerce')
df['SeniorCitizen'] = df['SeniorCitizen'].astype(bool)
df.dtypes
```

```
customerID          object
gender              object
SeniorCitizen       bool
Partner             object
Dependents          object
tenure              int64
PhoneService        object
MultipleLines       object
InternetService     object
OnlineSecurity      object
OnlineBackup        object
```

```
DeviceProtection    object
TechSupport         object
StreamingTV         object
StreamingMovies     object
Contract            object
PaperlessBilling    object
PaymentMethod       object
MonthlyCharges      float64
TotalCharges        float64
Churn               object
dtype: object
```

Step No.7 - Identify and Handle Outliers.

```
# Function to identify outliers using the IQR method
def identify_outliers_iqr(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return df[(df[column] < lower_bound) | (df[column] > upper_bound)]
# Identify outliers for each numerical column
numerical_columns = df.select_dtypes(include=[np.number]).columns
outliers = {col: identify_outliers_iqr(df, col) for col in numerical_columns}
```

```
# Display the number of outliers for each numerical column
outlier_counts = {col: len(outliers[col]) for col in outliers}
outlier_counts
{'tenure': 0, 'MonthlyCharges': 0, 'TotalCharges': 0}
# Function to remove outliers using the IQR method
def remove_outliers_iqr(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]
# Remove outliers for each numerical column
for col in numerical_columns:
    df = remove_outliers_iqr(df, col)
# Display the dataframe shape after outlier removal
df.shape
(7032, 21)
```

Step No.8 - Perform Feature Engineering.

```
# Create a new feature for total services count
service_columns = [
    'PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity',
    'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies'
]
```

```
df['TotalServices'] = df[service_columns].apply(lambda x: x.eq('yes').sum(), axis=1)

# Create a new feature for the ratio of MonthlyCharges to TotalCharges

df['ChargesRatio'] = df['MonthlyCharges'] / (df['TotalCharges'] + 1)  # Add 1 to avoid division
by zero

# Create tenure groups

def tenure_group(tenure):
    if tenure <= 12:
        return '0-1 year'
    elif tenure <= 24:
        return '1-2 years'
    elif tenure <= 48:
        return '2-4 years'
    elif tenure <= 60:
        return '4-5 years'
    else:
        return '5+ years'

df['TenureGroup'] = df['tenure'].apply(tenure_group)

# Display the first few rows to verify the new features

df[['TotalServices', 'ChargesRatio', 'TenureGroup']].head()
```

	TotalServices	ChargesRatio	TenureGroup
0	1	0.967585	0-1 year
1	3	0.030124	2-4 years
2	3	0.493358	0-1 year
3	3	0.022967	2-4 years
4	1	0.463151	0-1 year

```
# Step No.9 - Normalize or Scale the Data.
from sklearn.preprocessing import MinMaxScaler
# List of numerical columns to scale
numerical_columns = ['tenure', 'MonthlyCharges', 'TotalCharges']
# Initialize the scaler
scaler = MinMaxScaler()
# Apply the scaler to the numerical columns
df[numerical_columns] = scaler.fit_transform(df[numerical_columns])
# Display the first few rows to verify the changes
df[numerical_columns].head()
```

	tenure	MonthlyCharges	TotalCharges
0	0.000000	0.115423	0.001275
1	0.464789	0.385075	0.215867
2	0.014085	0.354229	0.010310
3	0.619718	0.239303	0.210241
4	0.014085	0.521891	0.015330

```
from sklearn.preprocessing import StandardScaler
# List of numerical columns to scale
numerical_columns = ['tenure', 'MonthlyCharges', 'TotalCharges']
# Initialize the scaler
scaler = StandardScaler()
# Apply the scaler to the numerical columns
df[numerical_columns] = scaler.fit_transform(df[numerical_columns])
# Display the first few rows to verify the changes
```

df[numerical_columns].head()

	tenure	MonthlyCharges	TotalCharges
0	-1.280248	-1.161694	-0.994194
1	0.064303	-0.260878	-0.173740
2	-1.239504	-0.363923	-0.959649
3	0.512486	-0.747850	-0.195248
4	-1.239504	0.196178	-0.940457

Step No.10 - Split the Dataset into Training and Testing Sets.

```
from sklearn.model_selection import train_test_split

# Separate features (X) and target variable (y)

X = df.drop('Churn', axis=1)  # Features

y = df['Churn']  # Target variable

# Split the dataset into training and testing sets

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

# Display the shapes of the resulting datasets

(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

((5625, 23), (1407, 23), (5625,), (1407,))
```

Step no.11 - Export the Cleaned Dataset.

```
df.to_csv("Cleaned_Telecom_Customer_Churn.csv", index=False)
```

Conclusion : We can successfully Analysing customer churn in a Telecommunications company and also we can easily data cleaning, prepration, and visualize the data on a “Telecom_customer_churn.csv”.

Lab Assignment No.	10
Title	**Data Wrangling** **Problem Statement:** Data Wrangling on Real Estate Market **Dataset:** "RealEstate_Prices.csv" **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., neighborhood 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 neighborhood or property type. 7. Identify and handle outliers or extreme values in the data that may affect the analysis or modeling process.
Roll No.	
Class	BE AI & DS
Date Of Completion	
Subject	Computer Laboratory I[417525]
Assessment Marks	
Assessor's Sign	

Experiment No. 10

Aim : Data Wrangling.

Problem Statement: Data Wrangling on Real Estate Market, and perform following tasks – 1. Import the dataset and explore. 2. Handle missing values. 3. Perform data merging. 4. Filter and subset the data. 5. Handle categorical variables. 6. Aggregate the data, and 7. Identify and handle outliers.

Dataset: "RealEstate_Prices.csv"

Software Requirements : Python and Jupyter Notebook.

Hardware Requirements : 8GB RAM, Storge and Processor.

Objectives : Clean, Integrate, and format data to remove errors, handle missing values, standardize format, and prepare for analysis, ensuring accuracy and reliability.

Theory : Data Wrangling

Data wrangling, also known as data munging or data preprocessing, is the process of cleaning, transforming, and organizing raw data into a structured format that is suitable for analysis or modeling. It's a critical step in the data analysis pipeline, as it ensures that the data is accurate, consistent, and usable.

key aspects of data wragling -

1.Data Collection and Acquisition

- **Source Identification**: Determine the sources of data, which may include databases, spreadsheets, APIs, or web scraping.
- **Data Extraction**: Extract data from various sources while ensuring data quality and completeness.

2. Data Cleaning

- **Handling Missing Values**: Address missing data through imputation (e.g., filling in missing values with the mean or median) or by removing incomplete records if necessary.
- **Removing Duplicates**: Identify and eliminate duplicate entries to avoid redundancy and ensure the accuracy of the dataset.

- **Error Correction**: Fix errors such as typos, incorrect data types, or out-of-range values to maintain data integrity.

3. Data Transformation

- **Normalization and Standardization**: Scale numerical features to a consistent range or standardize them to have zero mean and unit variance.
- **Encoding Categorical Variables**: Convert categorical data into numerical formats using techniques like one-hot encoding or label encoding.
- **Feature Engineering**: Create new features from existing ones to capture additional insights, such as interaction terms or polynomial features.

4. Data Integration

- **Merging Datasets**: Combine data from different sources or tables using join operations, ensuring that the merging process maintains data integrity and consistency.
- **Consolidation**: Aggregate data to a higher level of granularity or summarize it for easier analysis (e.g., aggregating sales data by month).

5. Data Reshaping and Aggregation

- **Pivoting**: Transform data from long to wide format or vice versa using pivot tables to facilitate analysis.
- **Aggregation**: Compute summary statistics such as totals, averages, or counts to consolidate data into meaningful metrics.

6. Outlier Detection and Handling

- **Detection Methods**: Identify outliers using statistical methods (e.g., z-scores, IQR) or visual techniques (e.g., box plots).
- **Handling Outliers**: Decide on the appropriate action for outliers, which may involve removing, adjusting, or retaining them based on their impact on the analysis.

7. Data Validation

- **Consistency Checks**: Verify that data is consistent across different records and within expected ranges (e.g., ensuring dates are in the correct format).

- **Accuracy Verification**: Cross-check data against external sources or validation rules to ensure correctness.

8. Data Enrichment

- **Incorporating External Data**: Enhance your dataset by adding information from external sources, such as demographic or geographical data.
- **Creating New Insights**: Derive new features or insights from enriched data to improve the quality of analysis.

9. Data Formatting

- **Standardizing Formats**: Ensure uniform data formats across the dataset, such as consistent date formats or numerical representations.
- **Cleaning Text Data**: Process text data by removing unnecessary characters, normalizing text (e.g., lowercasing), and handling text encoding issues.

10. Documentation and Metadata

- **Documenting Changes**: Keep detailed records of the data wrangling process, including transformations, cleaning steps, and feature engineering tasks.
- **Metadata Management**: Track metadata such as data source, definitions of features, and modifications to facilitate transparency and reproducibility.

Implementation

Step No.1 - Import the Dataset and Clean the Columns name.

```
import pandas as pd

import numpy as np

df = pd.read_csv(r"C:\Users\saira\Downloads\Mumbai_Property.csv")

df.columns

Index(['Property_Name', 'Location', 'Region', 'Property_Age', 'Availability',
```

```
       'Area_Tpye', 'Area_SqFt', 'Rate_SqFt', 'Floor_No', 'Bedroom',
       'Bathroom', 'Price_Lakh'],
      dtype='object')
df.columns = df.columns.str.replace(' ', '_').str.replace('(', '').str.replace(')', '').str.lower()
df.columns
Index(['property_name', 'location', 'region', 'property_age', 'availability',
       'area_tpye', 'area_sqft', 'rate_sqft', 'floor_no', 'bedroom',
       'bathroom', 'price_lakh'],
      dtype='object')
df.head()
property_name location region property_age availability area_tpye area_sqft rate_sqft floor_no bedroom bathroom price_lakh
0 Omkar Alta Monte W E Highway Malad East Mumbai Malad Mumbai 0 to 1 Year Ready To Move Super Built Up Area 2900.0 17241 14 3 4 500.0
1 T Bhimjyani Neelkanth Woods Manpada Thane Mumbai Manpada Thane 1 to 5 Year Ready To Move Super Built Up Area 1900.0 12631 8 3 3 240.0
2 Legend 1 Pramila Nagar Dahisar West Mumbai Dahisar Mumbai 10+ Year Ready To Move Super Built Up Area 595.0 15966 3 1 2 95.0
3 Unnamed Property Vidyavihar West Vidyavihar West Central Mumbai... Central Mumbai 5 to 10 Year Ready To Move Built Up Area 1450.0 25862 1 3 3 375.0
```

4 Unnamed Property 176 Cst Road Kalina Mumbai 400098 Santacruz Ea... Santacruz Mumbai 5 to 10 Year Ready To Move Carpet Area 876.0 39954 5 2 2 350.0

df.tail()

property_name location region property_age availability area_tpye area_sqft rate_sqft floor_no bedroom bathroom price_lakh

2575 Shagun White Woods Sector 23 Ulwe Navi Mumbai Mumbai Ulwe Navi-Mumbai 1 to 5 Year Ready To Move Built Up Area 1180.0 10338 2 2 2 122.0

2576 Guru Anant Sector 2 Ulwe Navi Mumbai Mumbai Ulwe Navi-Mumbai 0 to 1 Year Ready To Move Built Up Area 1090.0 8073 11 2 2 88.0

2577 Balaji Mayuresh Delta Ulwe Navi Mumbai Mumbai Ulwe Navi-Mumbai 1 to 5 Year Ready To Move Built Up Area 1295.0 10579 6 2 2 137.0

2578 Balaji Mayuresh Delta Ulwe Navi Mumbai Mumbai Ulwe Navi-Mumbai 1 to 5 Year Ready To Move Built Up Area 1850.0 9243 6 3 3 171.0

2579 Gurukrupa Tulsi Heights Ulwe Navi Mumbai Mumbai Ulwe Navi-Mumbai 0 to 1 Year Ready To Move Built Up Area 1100.0 8636 4 2 2 95.0

Step No.2 - Handle Missing Values.

df.isnull().sum()

property_name 0

location 0

region 0

property_age 0

availability 0

area_tpye 0

area_sqft 0

rate_sqft 0

floor_no 0

bedroom 0

bathroom 0

price_lakh 0

dtype: int64

Step No.3 - Perform Data Merging.

additional_df = pd.read_csv(r"C:\Users\saira\Downloads\Cleaned_Mumbai_RealEstate_Data.csv")

additional_df.columns

Index(['Property_Name', 'Location', 'Region', 'Availability', 'Area_Type',

'Area_SqFt', 'Rate_SqFt', 'Floor_No', 'Bedroom', 'Bathroom',

'Price_Lakh', 'Property_Age_0 to 1 Year', 'Property_Age_1 to 5 Year',

'Property_Age_10+ Year', 'Property_Age_5 to 10 Year',

'Property_Age_Under Construction'],

dtype='object')

additional_df.columns = additional_df.columns.str.replace(' ', '_').str.replace('(', '').str.replace(')', '').str.lower()

additional_df.columns

```
Index(['property_name', 'location', 'region', 'availability', 'area_type',

       'area_sqft', 'rate_sqft', 'floor_no', 'bedroom', 'bathroom',

       'price_lakh', 'property_age_0_to_1_year', 'property_age_1_to_5_year',

       'property_age_10+_year', 'property_age_5_to_10_year',

       'property_age_under_construction'],

      dtype='object')

df_merged = pd.merge(df, additional_df, on='property_name', how='left')

df_merged
```

property_name location_x region_x property_age availability_x area_tpye area_sqft_x rate_sqft_x floor_no_x bedroom_x ... rate_sqft_y floor_no_y bedroom_y bathroom_y price_lakh_y property_age_0_to_1_year property_age_1_to_5_year property_age_10+_year property_age_5_to_10_year property_age_under_construction

0 Omkar Alta Monte W E Highway Malad East Mumbai Malad Mumbai 0 to 1 Year Ready To Move Super Built Up Area 2900.0 17241 14 3 ... 17241.0 14.0 3.0 4.0 500.0 True False False False False

1 Omkar Alta Monte W E Highway Malad East Mumbai Malad Mumbai 0 to 1 Year Ready To Move Super Built Up Area 2900.0 17241 14 3 ... 20238.0 7.0 5.0 5.0 850.0 True False False False False

2 Omkar Alta Monte W E Highway Malad East Mumbai Malad Mumbai 0 to 1 Year Ready To Move Super Built Up Area 2900.0 17241 14 3 ... 16220.0 4.0 2.0 2.0 212.0 False True False False False

3 Omkar Alta Monte W E Highway Malad East Mumbai Malad Mumbai 0 to 1 Year Ready To Move Super Built Up Area 2900.0 17241 14 3 ... 16466.0 6.0 3.0 3.0 316.0 False True False False False

4 Omkar Alta Monte W E Highway Malad East Mumbai Malad Mumbai 0 to 1 Year Ready To Move Super Built Up Area 2900.0 17241 14 3 ... 19404.0 25.0 3.0 3.0 326.0 False True False False False

... ...

763428 Balaji Mayuresh Delta Ulwe Navi Mumbai Mumbai Ulwe Navi-Mumbai 1 to 5 Year Ready To Move Built Up Area 1850.0 9243 6 3 ... 10240.0 10.0 2.0 2.0 127.0 False True False False False

763429 Balaji Mayuresh Delta Ulwe Navi Mumbai Mumbai Ulwe Navi-Mumbai 1 to 5 Year Ready To Move Built Up Area 1850.0 9243 6 3 ... 9569.0 8.0 3.0 3.0 178.0 False True False False False

763430 Balaji Mayuresh Delta Ulwe Navi Mumbai Mumbai Ulwe Navi-Mumbai 1 to 5 Year Ready To Move Built Up Area 1850.0 9243 6 3 ... 10579.0 6.0 2.0 2.0 137.0 False True False False False

763431 Balaji Mayuresh Delta Ulwe Navi Mumbai Mumbai Ulwe Navi-Mumbai 1 to 5 Year Ready To Move Built Up Area 1850.0 9243 6 3 ... 9243.0 6.0 3.0 3.0 171.0 False True False False False

763432 Gurukrupa Tulsi Heights Ulwe Navi Mumbai Mumbai Ulwe Navi-Mumbai 0 to 1 Year Ready To Move Built Up Area 1100.0 8636 4 2 ... 8636.0 4.0 2.0 2.0 95.0 True False False False False

763433 rows × 27 columns

Step No.4 - Filter and Subset the Data.

Filter the data based on location

df_filtered_location = df[df['location'] == 'Sector 23 Ulwe Navi Mumbai Mumbai'] df_filtered_location.head()

property_name location region property_age availability area_tpye
area_sqft rate_sqft floor_no bedroom bathroom price_lakh

1154 Titanium One Sector 23 Ulwe Navi Mumbai Mumbai Ulwe Navi-Mumbai 1 to 5 Year Ready To Move Built Up Area 640.0 7500 4 1 2 48.0

1184 Unnamed Property Sector 23 Ulwe Navi Mumbai Mumbai Ulwe Navi-Mumbai 0 to 1 Year Ready To Move Carpet Area 680.0 9558 2 2 2 65.0

1539 Unnamed Property Sector 23 Ulwe Navi Mumbai Mumbai Ulwe Navi-Mumbai 1 to 5 Year Ready To Move Carpet Area 680.0 8382 1 2 2 57.0

1584 Platinum Palacio Sector 23 Ulwe Navi Mumbai Mumbai Ulwe Navi-Mumbai 1 to 5 Year Ready To Move Super Built Up Area 665.0 8307 5 1 1 54.0

2529 Shagun White Woods Sector 23 Ulwe Navi Mumbai Mumbai Ulwe Navi-Mumbai 1 to 5 Year Ready To Move Super Built Up Area 1160.0 10775 2 2 2 125.0

df_filtered_location.tail()

property_name location region property_age availability area_tpye
area_sqft rate_sqft floor_no bedroom bathroom price_lakh

1184 Unnamed Property Sector 23 Ulwe Navi Mumbai Mumbai Ulwe Navi-Mumbai 0 to 1 Year Ready To Move Carpet Area 680.0 9558 2 2 2 65.0

1539 Unnamed Property Sector 23 Ulwe Navi Mumbai Mumbai Ulwe Navi-Mumbai 1 to 5 Year Ready To Move Carpet Area 680.0 8382 1 2 2 57.0

1584 Platinum Palacio Sector 23 Ulwe Navi Mumbai Mumbai Ulwe Navi-Mumbai 1 to 5 Year Ready To Move Super Built Up Area 665.0 8307 5 1 1 54.0

2529 Shagun White Woods Sector 23 Ulwe Navi Mumbai Mumbai Ulwe Navi-Mumbai 1 to 5 Year Ready To Move Super Built Up Area 1160.0 10775 2 2 2 125.0

2575 Shagun White Woods Sector 23 Ulwe Navi Mumbai Mumbai Ulwe Navi-Mumbai 1 to 5 Year Ready To Move Built Up Area 1180.0 10338 2 2 2 122.0

df_filtered_property_type = df[(df['area_tpye'] == 'Plot Area') & (df['bedroom'] >2)]

df_filtered_property_type.head()

property_name location region property_age availability area_tpye area_sqft rate_sqft floor_no bedroom bathroom price_lakh

97 Unnamed Property Ramdev Park Ramdev Park Mira Road And Beyond M... Mira Road 10+ Year Ready To Move Plot Area 3000.0 18666 4 6 7 560.0

104 Unnamed Property New Panvel Navi Mumbai Mumbai Panvel Navi-Mumbai 0 to 1 Year Ready To Move Plot Area 210.0 247619 4 5 4 520.0

183 Unnamed Property Wada Mumbai Beyond Thane Mumbai Wada Mumbai5 to 10 Year Ready To Move Plot Area 2400.0 2291 1 3 2 55.0

237 Unnamed Property O 13 Sector 9 Belapur Navi Mumbai Mumbai Belapur Navi-Mumbai 10+ Year Ready To Move Plot Area 2000.0 8750 2 3 2 175.0

281 Unnamed Property Sector 2 Airoli Navi Mumbai Mumbai Airoli Navi-Mumbai 10+ Year Ready To Move Plot Area 1200.0 13750 3 3 2 165.0

df_filtered_property_type.tail()

property_name location region property_age availability area_tpye area_sqft rate_sqft floor_no bedroom bathroom price_lakh

345 Unnamed Property Sector 9 Belapur Navi Mumbai Mumbai Belapur Navi-Mumbai 10+ Year Ready To Move Plot Area 922.0 27223 2 4 3 251.0

521 Unnamed Property Sector 1 Koparkhairane Sector 1 Koparkhairane ... Koparkhairane Navi-Mumbai 5 to 10 Year Ready To Move Plot Area 200.0 34500 3 3 3 69.0

567 Unnamed Property 101 Manpada Thane Mumbai Manpada Thane 5 to 10 Year Ready To Move Plot Area 2700.0 27777 2 4 4 750.0

1201 Ravi Gaurav Greens Mira Road East Mira Road And Beyond Mumbai Mira Road 5 to 10 Year Ready To Move Plot Area 100000.0 290 2 4 4 290.0

1984 Unnamed Property Khardi Mumbai Beyond Thane Mumbai Mumbai Thane 10+ Year Ready To Move Plot Area 8500.0 1752 2 3 3 149.0

Step No.5 - Handle Categorical Variables.

df.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 2580 entries, 0 to 2579

Data columns (total 12 columns):

```
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   property_name  2580 non-null   object
 1   location       2580 non-null   object
 2   region         2580 non-null   object
 3   property_age   2580 non-null   object
 4   availability   2580 non-null   object
 5   area_tpye      2580 non-null   object
 6   area_sqft      2580 non-null   float64
 7   rate_sqft      2580 non-null   int64
 8   floor_no       2580 non-null   int64
 9   bedroom        2580 non-null   int64
 10  bathroom       2580 non-null   int64
 11  price_lakh     2580 non-null   float64
dtypes: float64(2), int64(4), object(6)
memory usage: 242.0+ KB
```

```
df['location'].unique()
```

```
array(['W E Highway Malad East Mumbai', 'Manpada Thane Mumbai',
       'Dahisar West Mumbai', ..., '501 Sector 5 Ulwe Navi Mumbai Mumbai',
       '1503 Mira Road East Mira Road And Beyond Mumbai',
```

'Sector 22 Kamothe Navi Mumbai Mumbai'], dtype=object)

df['area_tpye'].unique()

array(['Super Built Up Area', 'Built Up Area', 'Carpet Area', 'Plot Area'],

dtype=object)

Apply One-Hot Encoding to categorical columns

df_encoded = pd.get_dummies(df, columns=['location', 'area_tpye'])

df_encoded.head()

property_name region property_age availability area_sqft rate_sqft floor_no bedroom bathroom price_lakh ... location_Yagna Nagar Mumbai location_kavesar Thane Mumbai location_kurla west Central Mumbai suburbs Mumbai location_secter 7 koparkhairne Navi Mumbai Mumbai location_thakurli Mumbai Beyond Thane Mumbai location_y K nagar Nx virar west Mira Road And Beyond Mumbai area_tpye_Built Up Area area_tpye_Carpet Area area_tpye_Plot Area area_tpye_Super Built Up Area

0 Omkar Alta Monte Malad Mumbai 0 to 1 Year Ready To Move 2900.0 17241 14 3 4 500.0 ... False False False False False False False False False True

1 T Bhimjyani Neelkanth Woods Manpada Thane 1 to 5 Year Ready To Move 1900.0 12631 8 3 3 240.0 ... False False False False False False False False False True

2 Legend 1 Pramila Nagar Dahisar Mumbai 10+ Year Ready To Move 595.0 15966 3 1 2 95.0 ... False False False False False False False False False True

3 Unnamed Property Central Mumbai 5 to 10 Year Ready To Move 1450.0 25862 1 3 3 375.0 ... False False False False False False True False False False

4 Unnamed Property Santacruz Mumbai 5 to 10 Year Ready To Move 876.0 39954 5 2 2 350.0 ... False False False False False False False True False False

5 rows × 1322 columns

```
from sklearn.preprocessing import LabelEncoder

# Initialize LabelEncoder

label_encoder = LabelEncoder()

# Apply Label Encoding to categorical columns

df['Location_encoded'] = label_encoder.fit_transform(df['location'])

df['Area_Tpye_encoded'] = label_encoder.fit_transform(df['area_tpye'])

df[['location', 'Location_encoded', 'area_tpye', 'Area_Tpye_encoded']].head()
```

	location	Location_encoded	area_tpye	Area_Tpye_encoded
0	W E Highway Malad East Mumbai	1276	Super Built Up Area	3
1	Manpada Thane Mumbai	886	Super Built Up Area	3
2	Dahisar West Mumbai	683	Super Built Up Area	3
3	Vidyavihar West Vidyavihar West Central Mumbai...	1263	Built Up Area	0
4	176 Cst Road Kalina Mumbai 400098 Santacruz Ea...	246	Carpet Area	1

Step No.6 - Aggregate the Data.

```
# Group by 'Location' and calculate the average sale price

average_price_by_neighborhood = df.groupby('location')['price_lakh'].mean().reset_index()

average_price_by_neighborhood
```

	location	price_lakh
0	000 4 Bunglows Mumbai	700.0000
1	000 Anand Nagar Thane Mumbai	69.0000
2	000 Andheri West Mumbai	450.0000
3	000 Balkum Thane Mumbai	210.0000
4	000 Borivali West Mumbai	102.0000
...	...	...
1303	kavesar Thane Mumbai	120.3750
1304	kurla west Central Mumbai suburbs Mumbai	156.6000
1305	secter 7 koparkhairne Navi Mumbai Mumbai	110.0000
1306	thakurli Mumbai Beyond Thane Mumbai	61.5125
1307	y K nagar Nx virar west Mira Road And Beyond M...	38.0000

1308 rows × 2 columns

```
# Group by 'area_Tpye' and calculate the average sale price
average_price_by_area_type = df.groupby('area_tpye')['price_lakh'].mean().reset_index()
average_price_by_area_type
```

	area_tpye	price_lakh
0	Built Up Area	138.621000
1	Carpet Area	184.005299
2	Plot Area	215.687500

3 Super Built Up Area 177.751004

Step No.7 - Identify and Handle Outliers.

```
from scipy import stats

# Calculate Z-scores

z_scores = stats.zscore(df['price_lakh'])

df['z_score'] = z_scores

z_scores

0       0.881426
1       0.177607
2      -0.214908
3       0.543052
4       0.475377
          ...
2575   -0.141819
2576   -0.233857
2577   -0.101214
2578   -0.009176
2579   -0.214908
Name: price_lakh, Length: 2580, dtype: float64

# Identify outliers (e.g., Z-score > 3 or < -3)
```

outliers_z_score = df[(df['z_score'] > 3) | (df['z_score'] < -3)]

Display outliers

outliers_z_score

property_name location region property_age availability area_tpye area_sqft rate_sqft floor_no bedroom bathroom price_lakh Location_encoded Area_Tpye_encoded z_score

39 Swan Lake Apartment1 101 Khar West Mumbai South West Mumbai South Mumbai 10+ Year Ready To Move Carpet Area 2715.0 66298 0 4 4 1800.0 69 1 4.400525

203 Sagar Mahal Opposite Gopi Birla School And Sheetal Baug Wa... Walkeshwar Mumbai 10+ Year Ready To Move Built Up Area 2450.0 67350 9 4 5 1650.0 991 0 3.994475

329 Jolly Maker Apartment Cuffe Parade South Mumbai Mumbai South Mumbai 10+ Year Ready To Move Built Up Area 2135.0 74941 20 5 4 1600.0 673 0 3.859125

605 Hiranandani Gardens Richmond Tower Hiranandani Gardens Powai Hiranandani Gardens ... Central Mumbai 1 to 5 Year Ready To Move Super Built Up Area 5000.0 33000 6 5 6 1650.0 766 3 3.994475

634 Kalpataru Solitaire Juhu Mumbai South West Mumbai Juhu Mumbai 1 to 5 Year Ready To Move Super Built Up Area 3000.0 46000 6 3 5 1380.0 781 3 3.263586

635 Kalpataru Solitaire Juhu Mumbai South West Mumbai Juhu Mumbai 1 to 5 Year Ready To Move Super Built Up Area 2800.0 46428 5 3 3 1300.0 781 3 3.047026

1064 Unnamed Property Juhu Mumbai South West Mumbai Juhu Mumbai 10+ Year Ready To Move Built Up Area 4363.0 36672 2 4 4 1600.0 781 0 3.859125

1067 Unnamed Property Juhu Mumbai South West Mumbai Juhu Mumbai 10+ Year Ready To Move Super Built Up Area 4200.0 45238 1 4 4 1900.0 781 3 4.671225

1416 Unnamed Property Juhu Mumbai South West Mumbai Juhu Mumbai 10+ Year Ready To Move Super Built Up Area 5700.0 42105 1 4 4 2400.0 781 3 6.024724

1675 Piramal Aranya Byculla East Byculla East Mumbai Harbour Mumbai Mumbai Harbour 0 to 1 Year Ready To Move Carpet Area 2800.0 49107 21 4 4 1375.0 637 1 3.250051

2065 White City 005 Kandivali East Mumbai Kandivali Mumbai 0 to 1 Year Ready To Move Super Built Up Area 1000.0 1650000 21 2 2 16500.0 60 3 44.193409

```
# Remove outliers based on Z-scores

df_cleaned = df[(df['z_score'] <= 3) & (df['z_score'] >= -3)]

# Drop the Z-score column if no longer needed

df_cleaned = df_cleaned.drop(columns=['z_score'])

df_cleaned.head()
```

property_name location region property_age availability area_tpye area_sqft rate_sqft floor_no bedroom bathroom price_lakh Location_encoded Area_Tpye_encoded

0 Omkar Alta Monte W E Highway Malad East Mumbai Malad Mumbai 0 to 1 Year Ready To Move Super Built Up Area 2900.0 17241 14 3 4 500.0 1276 3

1 T Bhimjyani Neelkanth Woods Manpada Thane Mumbai Manpada Thane 1 to 5 Year Ready To Move Super Built Up Area 1900.0 12631 8 3 3 240.0 886 3

2 Legend 1 Pramila Nagar Dahisar West Mumbai Dahisar Mumbai 10+ Year Ready To Move Super Built Up Area 595.0 15966 3 1 2 95.0 683 3

3 Unnamed Property Vidyavihar West Vidyavihar West Central Mumbai... Central Mumbai 5 to 10 Year Ready To Move Built Up Area 1450.0 25862 1 3 3 375.0 1263 0

4 Unnamed Property 176 Cst Road Kalina Mumbai 400098 Santacruz Ea... Santacruz Mumbai 5 to 10 Year Ready To Move Carpet Area 876.0 39954 5 2 2 350.0 246 1

Conclusion : We can successfully done effective data wrangling ensures clean, integrated on a “RealEstate_Price.csv” dataset, and also enhancing insights and decisions making and also addressing errors and standardizing formats it ensures accuracy and reliability.

Lab Assignment No.	11
Title	**Data Visualization using matplotlib** **Problem Statement:** Analyzing Air Quality Index (AQI) Trends in a City. **Dataset:** "City_Air_Quality.csv" **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 color schemes.
Roll No.	
Class	BE AI & DS
Date Of Completion	
Subject	Computer Laboratory I[417525]
Assessment Marks	
Assessor's Sign	

Experiment No. 11

Aim : Data Visualization using matplotlib.

Problem Statement: Analyzing Air Quality Index (AQI) Trends in a City, and perform following tasks – 1. Import dataset. 2. Explore dataset. 3. Identify the relevant variables for visualizing AQI trends. 4. Create line plots or time series plots. 5. Plot individual pollutant levels. 6. Use bar plots or stacked bar plots to compare the AQI values. 7. Create box plots or violin plots to analyze the distribution of AQI values. 8. Use scatter plots or bubble charts to explore the relationship. 9. Customize the visualizations.

Dataset: "City_Air_Quality.csv"

Software Requirements : Python, and Jupyter Notebook.

Hardware Requirements : 8GB RAM, Storage and Processor.

Objectives : i) Import and explore the "City_Air_Quality.csv" dataset to understand it's structure. ii) Identify and extract relevant variables for visualizing AQI trends. iii) Customize visualizations for better reliability and interpretation.

Theory : The Air Quality Index (AQI) is a numerical scale used to communicate how polluted the air currently is or how polluted it is forecasted to become. It is designed to provide the public with a clear and easily understandable measure of air quality, helping individuals make informed decisions about their health and activities.

Data Visualization

Data visualization is the graphical representation of information and data. It uses visual elements like charts, graphs, and maps to make complex data more accessible, understandable, and actionable. Effective data visualization helps to uncover insights, reveal patterns, and communicate findings in a clear and compelling way.

Matplotlib

Matplotlib is a powerful and widely used plotting library for the Python programming language. It provides a flexible and comprehensive way to create static, animated, and interactive visualizations in Python.

Customization in Matplotlib

Customizing visualizations in Matplotlib allows you to tailor plots to specific needs, enhance readability, and effectively communicate insights. Matplotlib provides a rich set of customization options for various elements of your plots.

Types of plots for AQI Analysis

1 . Line Plot

Purpose: To display trends over time and track changes in AQI values.

Application: Time Series Analysis: Use line plots to show how AQI values fluctuate over time, such as daily, monthly, or yearly trends. This helps in identifying patterns, seasonal variations, and long-term changes.

2. Bar Plot

Purpose: To compare AQI values across different categories or locations.

Application: Comparative Analysis: Use bar plots to compare average AQI values for different locations, cities, or time periods. This can highlight areas with higher or lower air quality.

3.Box Plot

Purpose: To summarize the distribution of AQI values and identify outliers.

Application: Distribution Analysis: Use box plots to visualize the spread of AQI values and to detect any anomalies or outliers in the data. It provides a concise summary of the data's minimum, first quartile, median, third quartile, and maximum.

4.Scatter Plot

Purpose: To examine the relationship between AQI and another variable (e.g., temperature, humidity).

Application: Correlation Analysis: Use scatter plots to visualize how AQI correlates with other environmental factors. This can help identify patterns or trends in how AQI changes with different conditions.

5.Violin Plot

Purpose: To visualize the distribution of AQI values across different categories, showing both the distribution shape and density.

Application: Distribution Comparison: Use violin plots to compare the distribution of AQI values across different locations or time periods. It provides a deeper understanding of the data distribution beyond just summary statistics.

Implementation

Step No.1 - Import The Dataset.

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

df = pd.read_csv(r"C:\Users\saira\Downloads\city_day.csv\city_day.csv")

df
```

```
City    Date    PM2.5   PM10    NO      NO2     NOx     NH3     CO      SO2     O3      Benzene
        Toluene         Xylene  AQI     AQI_Bucket

0       Ahmedabad       2015-01-01      NaN     NaN     0.92    18.22   17.15   NaN     0.92
        27.64   133.36  0.00    0.02    0.00    NaN     NaN

1       Ahmedabad       2015-01-02      NaN     NaN     0.97    15.69   16.46   NaN     0.97
        24.55   34.06   3.68    5.50    3.77    NaN     NaN

2       Ahmedabad       2015-01-03      NaN     NaN     17.40   19.30   29.70   NaN     17.40
        29.07   30.70   6.80    16.40   2.25    NaN     NaN

3       Ahmedabad       2015-01-04      NaN     NaN     1.70    18.48   17.97   NaN     1.70
        18.59   36.08   4.43    10.14   1.00    NaN     NaN

4       Ahmedabad       2015-01-05      NaN     NaN     22.10   21.42   37.76   NaN     22.10
        39.33   39.31   7.01    18.89   2.78    NaN     NaN
```

```
...     ...     ...     ...     ...     ...     ...     ...     ...     ...     ...     ...     ...
        ...     ...     ...     ...

29526   Visakhapatnam           2020-06-27      15.02   50.94   7.68    25.06   19.54   12.47
        0.47    8.55    23.30   2.24    12.07   0.73    41.0    Good

29527   Visakhapatnam           2020-06-28      24.38   74.09   3.42    26.06   16.53   11.99
        0.52    12.72   30.14   0.74    2.21    0.38    70.0    Satisfactory

29528   Visakhapatnam           2020-06-29      22.91   65.73   3.45    29.53   18.33   10.71
        0.48    8.42    30.96   0.01    0.01    0.00    68.0    Satisfactory

29529   Visakhapatnam           2020-06-30      16.64   49.97   4.05    29.26   18.80   10.03
        0.52    9.84    28.30   0.00    0.00    0.00    54.0    Satisfactory

29530   Visakhapatnam           2020-07-01      15.00   66.00   0.40    26.85   14.05   5.20
        0.59    2.10    17.05   NaN     NaN     NaN     50.0    Good
```

df.head()

```
City    Date    PM2.5   PM10    NO      NO2     NOx     NH3     CO      SO2     O3      Benzene
        Toluene         Xylene  AQI     AQI_Bucket

0       Ahmedabad       2015-01-01      NaN     NaN     0.92    18.22   17.15   NaN     0.92
        27.64   133.36  0.00    0.02    0.00    NaN     NaN

1       Ahmedabad       2015-01-02      NaN     NaN     0.97    15.69   16.46   NaN     0.97
        24.55   34.06   3.68    5.50    3.77    NaN     NaN

2       Ahmedabad       2015-01-03      NaN     NaN     17.40   19.30   29.70   NaN     17.40
        29.07   30.70   6.80    16.40   2.25    NaN     NaN

3       Ahmedabad       2015-01-04      NaN     NaN     1.70    18.48   17.97   NaN     1.70
        18.59   36.08   4.43    10.14   1.00    NaN     NaN

4       Ahmedabad       2015-01-05      NaN     NaN     22.10   21.42   37.76   NaN     22.10
        39.33   39.31   7.01    18.89   2.78    NaN     NaN
```

df.tail()

```
City    Date    PM2.5   PM10    NO      NO2     NOx     NH3     CO      SO2     O3      Benzene
        Toluene         Xylene  AQI     AQI_Bucket
```

```
29526  Visakhapatnam  2020-06-27  15.02  50.94  7.68  25.06  19.54  12.47
       0.47  8.55  23.30  2.24  12.07  0.73  41.0  Good

29527  Visakhapatnam  2020-06-28  24.38  74.09  3.42  26.06  16.53  11.99
       0.52  12.72  30.14  0.74  2.21  0.38  70.0  Satisfactory

29528  Visakhapatnam  2020-06-29  22.91  65.73  3.45  29.53  18.33  10.71
       0.48  8.42  30.96  0.01  0.01  0.00  68.0  Satisfactory

29529  Visakhapatnam  2020-06-30  16.64  49.97  4.05  29.26  18.80  10.03
       0.52  9.84  28.30  0.00  0.00  0.00  54.0  Satisfactory

29530  Visakhapatnam  2020-07-01  15.00  66.00  0.40  26.85  14.05  5.20
       0.59  2.10  17.05  NaN  NaN  NaN  50.0  Good
```

Step No.2 - Explore the Structure and Content Of Dataset.

```
df.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 29531 entries, 0 to 29530

Data columns (total 16 columns):

 #   Column      Non-Null Count  Dtype

---  ------      --------------  -----

 0   City       29531 non-null  object

 1   Date        29531 non-null  object

 2   PM2.5       24933 non-null  float64

 3   PM10        18391 non-null  float64

 4   NO         25949 non-null  float64

 5   NO2         25946 non-null  float64

 6   NOx         25346 non-null  float64

 7   NH3         19203 non-null  float64

 8   CO         27472 non-null  float64
```

```
 9   SO2         25677 non-null  float64
 10  O3          25509 non-null  float64
 11  Benzene     23908 non-null  float64
 12  Toluene     21490 non-null  float64
 13  Xylene      11422 non-null  float64
 14  AQI         24850 non-null  float64
 15  AQI_Bucket  24850 non-null  object
dtypes: float64(13), object(3)
memory usage: 3.6+ MB
```

df.describe()

	PM2.5	PM10	NO	NO2	NOx	NH3	CO	SO2	O3	Benzene	Toluene	Xylene	AQI
count	24933.000000	18391.000000	25949.000000	25946.000000	25346.000000	19203.000000	27472.000000	25677.000000	25509.000000	23908.000000	21490.000000	11422.000000	24850.000000
mean	67.450578	118.127103	17.574730	28.560659	32.309123	23.483476	2.248598	14.531977	34.491430	3.280840	8.700972	3.070128	166.463581
std	64.661449	90.605110	22.785846	24.474746	31.646011	25.684275	6.962884	18.133775	21.694928	15.811136	19.969164	6.323247	140.696585
min	0.040000	0.010000	0.020000	0.010000	0.000000	0.010000	0.000000	0.010000	0.010000	0.000000	0.000000	0.000000	13.000000
25%	28.820000	56.255000	5.630000	11.750000	12.820000	8.580000	0.510000	5.670000	18.860000	0.120000	0.600000	0.140000	81.000000

```
50%   48.570000   95.680000   9.890000    21.690000   23.520000   15.850000
      0.890000    9.160000    30.840000   1.070000    2.970000    0.980000
      118.000000

75%   80.590000   149.745000  19.950000   37.620000   40.127500   30.020000
      1.450000    15.220000   45.570000   3.080000    9.150000    3.350000
      208.000000

max   949.990000  1000.000000 390.680000  362.210000  467.630000
      352.890000  175.810000  193.860000  257.730000  455.030000
      454.850000  170.370000  2049.000000
```

df.isnull().sum()

```
City          0
Date          0
PM2.5      4598
PM10      11140
NO         3582
NO2        3585
NOx        4185
NH3       10328
CO         2059
SO2        3854
O3         4022
Benzene    5623
Toluene    8041
Xylene    18109
AQI        4681
AQI_Bucket 4681
```

```
dtype: int64

df.dropna(inplace=True)

df.isnull().sum()

City          0
Date          0
PM2.5         0
PM10          0
NO            0
NO2           0
NOx           0
NH3           0
CO            0
SO2           0
O3            0
Benzene       0
Toluene       0
Xylene        0
AQI           0
AQI_Bucket    0
dtype: int64

df.columns

Index(['City', 'Date', 'PM2.5', 'PM10', 'NO', 'NO2', 'NOx', 'NH3', 'CO', 'SO2',
       'O3', 'Benzene', 'Toluene', 'Xylene', 'AQI', 'AQI_Bucket'],
      dtype='object')
```

Step No.3 - Identify the relevant variables for visualizing AQI trends.

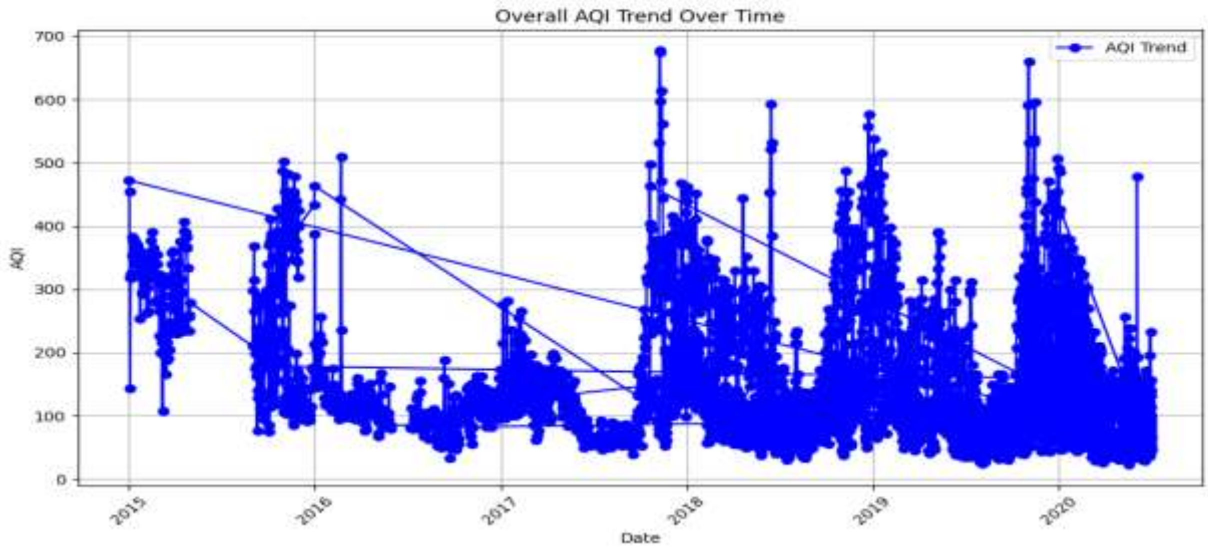
```
# Based on typical air quality datasets, relevant columns are identified as:
# - Date: To track AQI and pollutant levels over time
# - AQI: The Air Quality Index values
# - Pollutant levels: Including PM2.5, PM10, CO, NO2, SO2, O3
relevant_columns = ['Date', 'AQI', 'PM2.5', 'PM10', 'CO', 'NO2', 'SO2', 'O3']
relevant_columns
['Date', 'AQI', 'PM2.5', 'PM10', 'CO', 'NO2', 'SO2', 'O3']
# Identify the available relevant columns in the dataset
available_relevant_columns = [col for col in relevant_columns if col in df.columns]
available_relevant_columns
['Date', 'AQI', 'PM2.5', 'PM10', 'CO', 'NO2', 'SO2', 'O3']
if 'Date' in available_relevant_columns:
    df['Date'] = pd.to_datetime(df['Date'])
df[available_relevant_columns].head()
```

	Date	AQI	PM2.5	PM10	CO	NO2	SO2	O3
2123	2017-11-25	184.0	81.40	124.50	0.12	20.50	15.24	127.09
2124	2017-11-26	197.0	78.32	129.06	0.14	26.00	26.96	117.44
2125	2017-11-27	198.0	88.76	135.32	0.11	30.85	33.59	111.81
2126	2017-11-28	188.0	64.18	104.09	0.09	28.07	19.00	138.18
2127	2017-11-29	173.0	72.47	114.84	0.16	23.20	10.55	109.74

Step No.4 - Create line plots or time series plots to visualize the overall AQI trend over time.

```
plt.figure(figsize=(10, 6))
plt.plot(df['Date'], df['AQI'], marker='o', linestyle='-', color='b', label='AQI Trend')
```

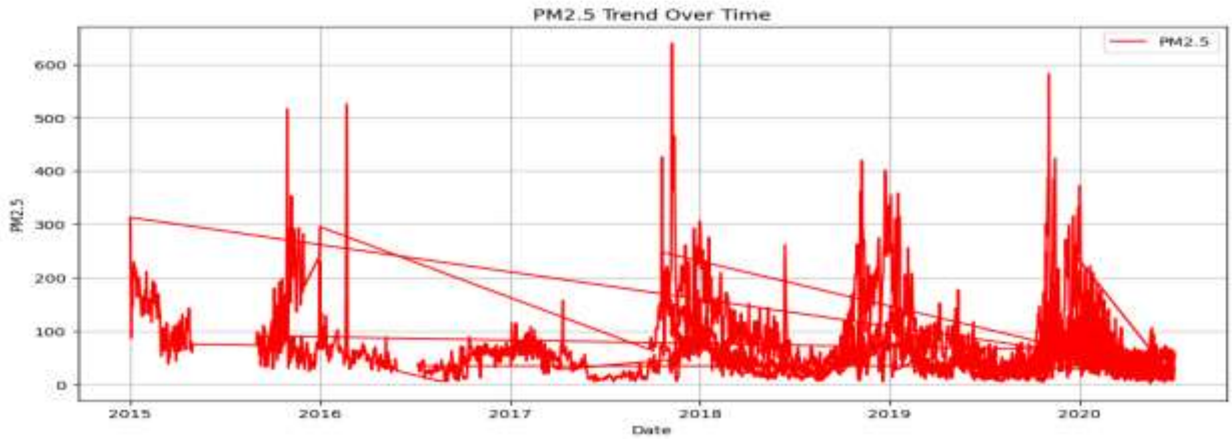
```
plt.xlabel('Date')
plt.ylabel('AQI')
plt.title('Overall AQI Trend Over Time')
plt.xticks(rotation=45)
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

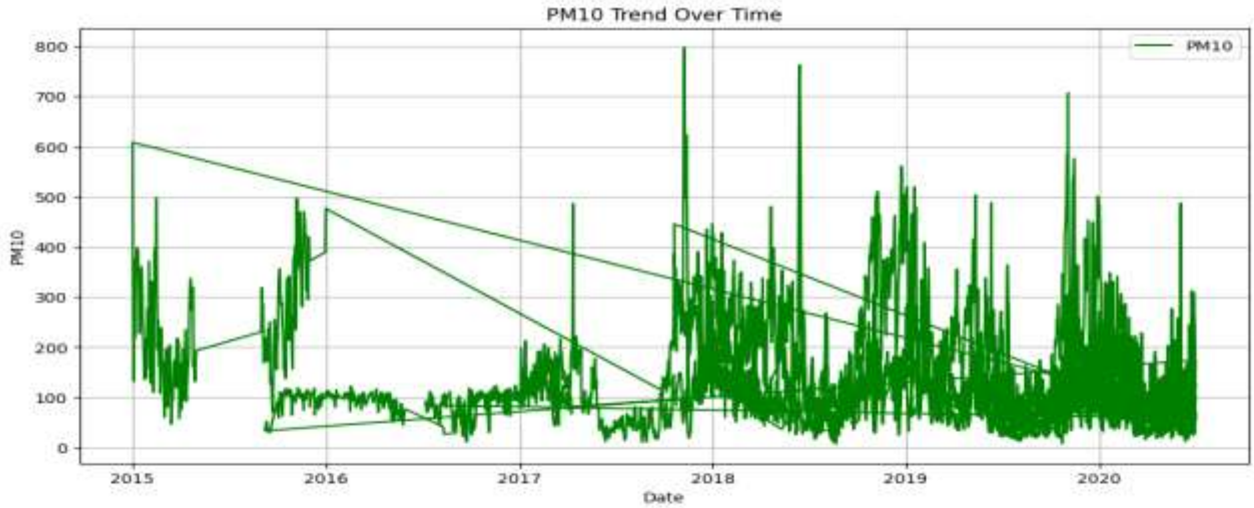


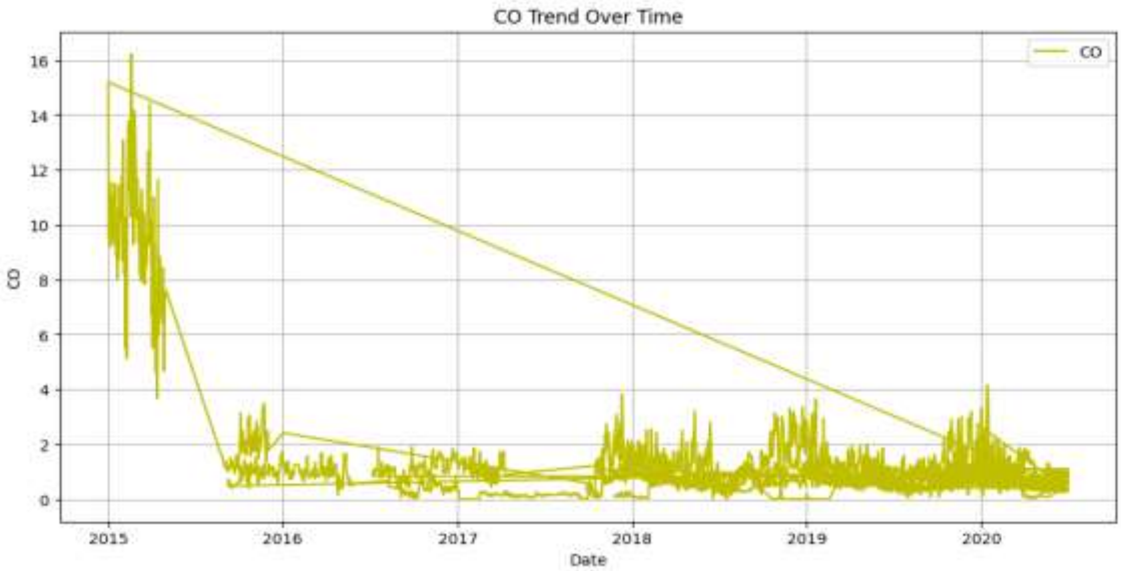
Step No.5 - Plot individual pollutant levels over time.

```
pollutants = ['PM2.5', 'PM10', 'CO']
for pollutant in pollutants:
    plt.figure(figsize=(12, 6))
    plt.plot(df['Date'], df[pollutant], label=pollutant, color='r' if pollutant == 'PM2.5' else 'g' if pollutant == 'PM10' else 'y')
    plt.xlabel('Date')
    plt.ylabel(pollutant)
    plt.title(f'{pollutant} Trend Over Time')
```

```
plt.legend()

plt.grid(True)

plt.show()
```

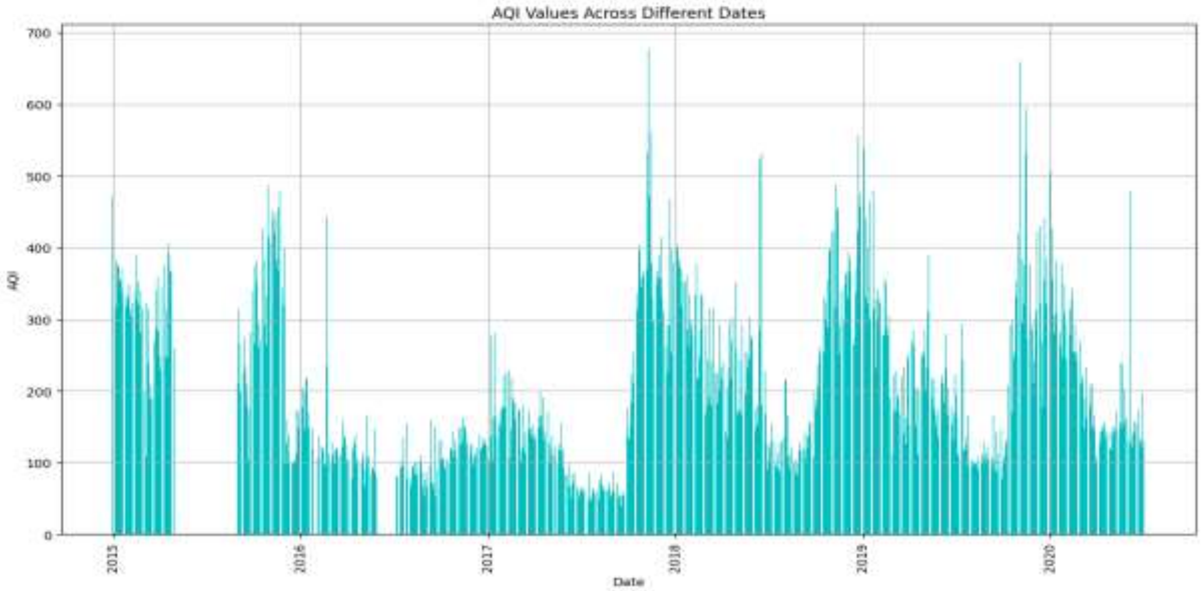






Step No. 6 - Use bar plots or stacked bar plots to compare the AQI values across different dates or time periods.

```
# Plot bar plot for AQI values across different dates

plt.figure(figsize=(15, 8))

plt.bar(df['Date'], df['AQI'], color='c')

plt.xlabel('Date')

plt.ylabel('AQI')

plt.title('AQI Values Across Different Dates')

plt.xticks(rotation=90)

plt.grid(True)

plt.show()
```




```
# Plot stacked bar plot for AQI values with different pollutants

plt.figure(figsize=(15, 8))

bar_width = 0.5

plt.bar(df['Date'], df['PM2.5'], label='PM2.5', color='b', width=bar_width)

plt.bar(df['Date'], df['PM10'], bottom=df['PM2.5'], label='PM10', color='r', width=bar_width)
```

plt.bar(df['Date'], df['CO'], bottom=df['PM2.5'] + df['PM10'], label='CO', color='g', width=bar_width)

plt.xlabel('Date')

plt.ylabel('Pollutant Levels')

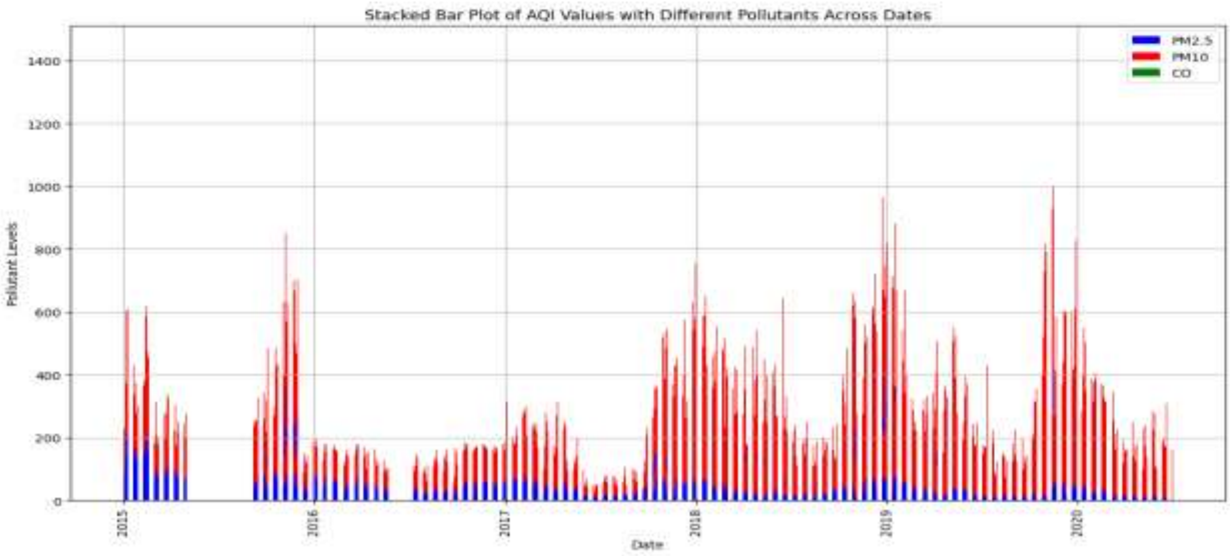
plt.title('Stacked Bar Plot of AQI Values with Different Pollutants Across Dates')

plt.xticks(rotation=90)

plt.legend()

plt.grid(True)

plt.show()



Step No. 7 - Create box plots or violin plots to analyze the distribution of AQI values for different pollutant categories.

Create box plot for AQI values by pollutant categories

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

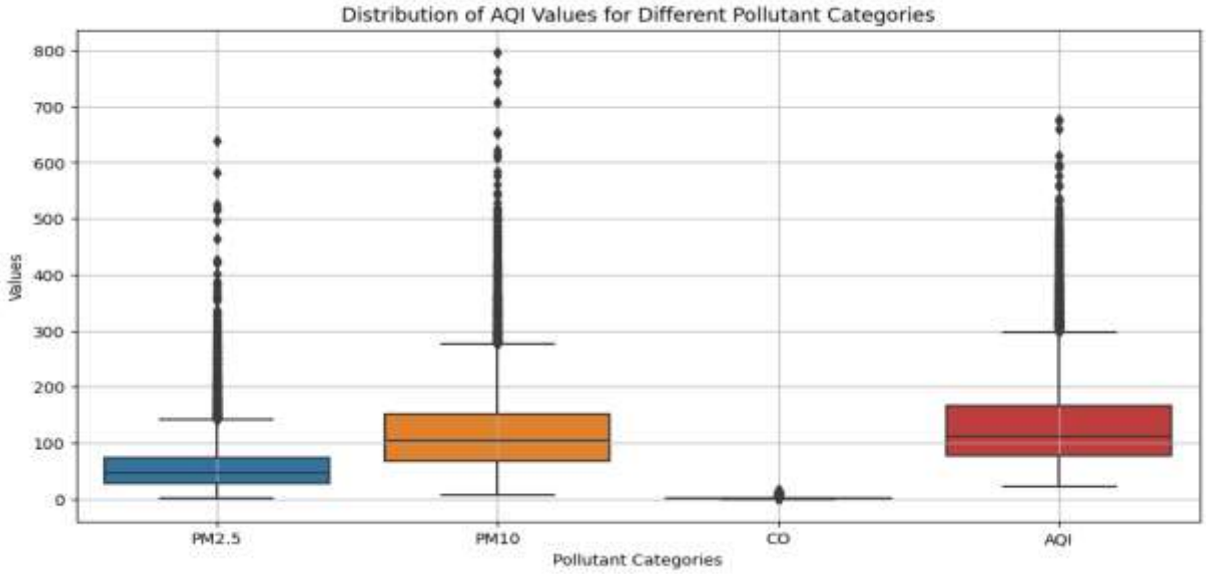
sns.boxplot(data=df[['PM2.5', 'PM10', 'CO', 'AQI']])

plt.xlabel('Pollutant Categories')

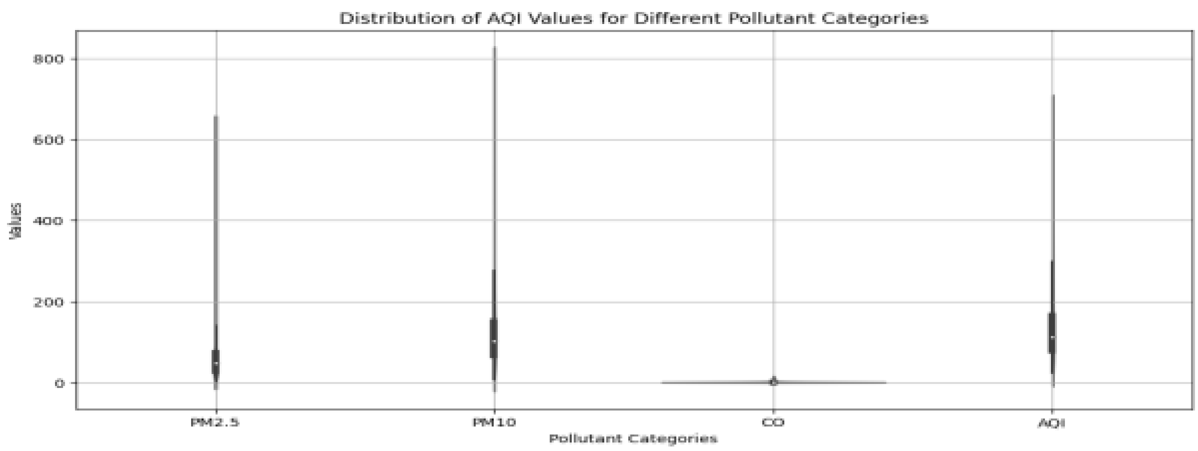
plt.ylabel('Values')

plt.title('Distribution of AQI Values for Different Pollutant Categories')

```
plt.grid(True)

plt.show()
```

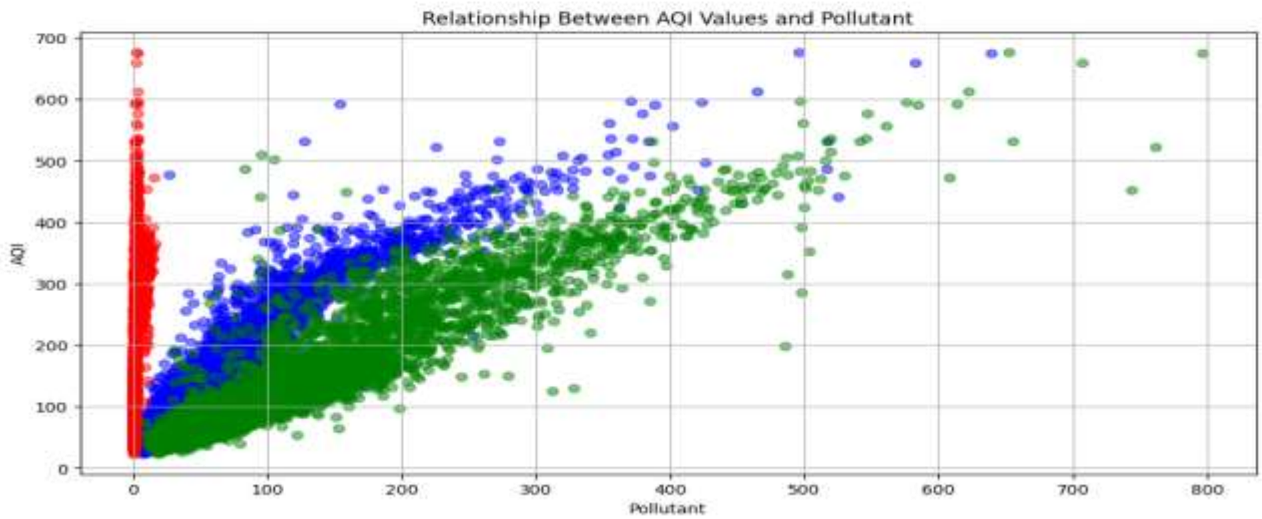



```
# Create violin plot for AQI values by pollutant categories

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

sns.violinplot(data=df[['PM2.5', 'PM10', 'CO', 'AQI']])

plt.xlabel('Pollutant Categories')

plt.ylabel('Values')

plt.title('Distribution of AQI Values for Different Pollutant Categories')

plt.grid(True)

plt.show()
```



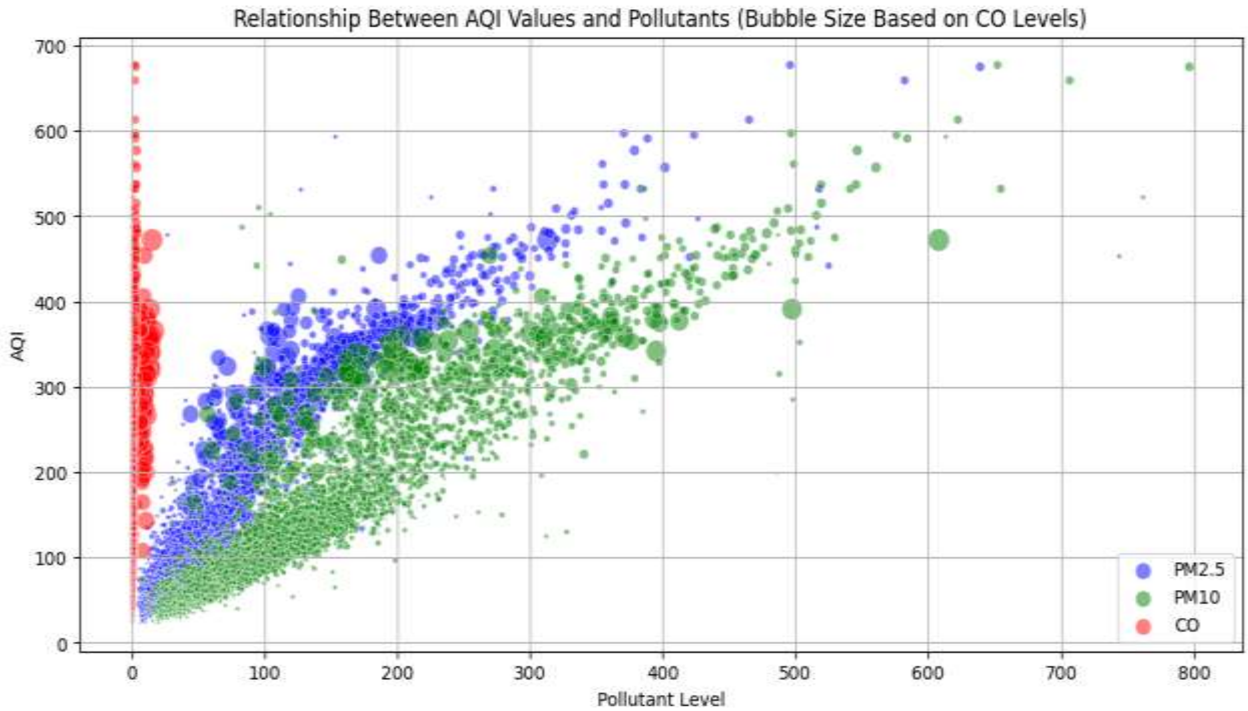
Step No. 8 - Use scatter plots or bubble charts to explore the relationship between AQI values and pollutant levels.

```
# Scatter plot for AQI values vs. Pollutants

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

plt.scatter(df['PM2.5'], df['AQI'], alpha=0.5, color='b')

plt.scatter(df['PM10'], df['AQI'], alpha=0.5, color='g')

plt.scatter(df['CO'], df['AQI'], alpha=0.5, color='r')

plt.xlabel('Pollutant')

plt.ylabel('AQI')

plt.title('Relationship Between AQI Values and Pollutant')

plt.grid(True)

plt.show()
```




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

plt.scatter(df['PM2.5'], df['AQI'], s=df['CO']*10, alpha=0.5, color='b', edgecolors='w', linewidth=0.5, label='PM2.5')

plt.scatter(df['PM10'], df['AQI'], s=df['CO']*10, alpha=0.5, color='g', edgecolors='w', linewidth=0.5, label='PM10')
```

```
plt.scatter(df['CO'], df['AQI'], s=df['CO']*10, alpha=0.5, color='r', edgecolors='w', linewidth=0.5, label='CO')

plt.xlabel('Pollutant Level')

plt.ylabel('AQI')

plt.title('Relationship Between AQI Values and Pollutants (Bubble Size Based on CO Levels)')

plt.legend(loc='best')

plt.grid(True)

plt.show()
```



Conclusion : We can successfully analyzing Air Quality Index [AQI] trends in a city on a “Air_Quality.csv” dataset, also we can visualizing various AQI trends easily by using matplotlib.

Lab Assignment No.	12
Title	**Data Aggregation** **Problem Statement:** Analyzing Sales Performance by Region in a Retail Company **Dataset:** "Retail_Sales_Data.csv" **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.
Roll No.	
Class	BE AI & DS
Date Of Completion	
Subject	Computer Laboratory I[417525]
Assessment Marks	
Assessor's Sign	

Experiment No. 12

Aim : Data Aggregation.

Problem Statement: Analyzing Sales Performance by Region in a Retail Company and perform the following tasks – 1. Import the dataset. 2. Explore the dataset. 3. Identify the relevant variables for aggregating sales data. 4. Group the sales data by region. 5. Create bar plots or pie charts. 6. Identify the top-performing regions. 7. Group the sales data, and 8. Create stacked bar plots or grouped bar plots.

Dataset: "Retail_Sales_Data.csv"

Software Requirements : Python and Jupyter Notebook.

Hardware Requirements : 8GB RAM, Storage and Processor.

Objective : i) Analyze sales performance by region. ii) Visualize the sales distribution. iii) Identify key product categories. iv) Compare regional sales trends.

Theory : Data Aggregation

Data aggregation refers to the process of collecting and summarizing data from various sources to provide a comprehensive view or to perform analysis. This can be done in several ways, depending on the type of data and the goals of the analysis.

Aggregation Techniques

Aggregation techniques are used to combine data to make it more manageable and insightful. These techniques can be categorized into single-level and multilevel aggregation. Here's an overview of both:

Single-Level Aggregation

Single-level aggregation involves summarizing data at one specific level or granularity. This technique is straightforward and typically used when you want to analyze data from a single perspective or dimension.

Multilevel Aggregation

Multilevel aggregation involves summarizing data across multiple levels of granularity or dimensions. This technique is useful for analyzing data in a more nuanced way by examining different hierarchies or layers.

Key concepts of Data Aggregation

1.Grouping

Grouping is the process of organizing data into categories or groups based on shared attributes or dimensions. To simplify data analysis by aggregating records that share common characteristics.

2.Summarization

Summarization involves calculating aggregate metrics that provide a concise view of the data. To reduce the complexity of data by providing summary statistics that highlight key information.

3.Hierarchical Aggregation

Hierarchical aggregation refers to summarizing data at various levels of a hierarchy or dimensions, often used in multidimensional data models. To provide insights at different levels of granularity, allowing for detailed analysis from a high-level overview to granular details.

4.Visualization

Visualization involves representing aggregated data in graphical formats to make it easier to interpret and analyze. To provide a visual representation of data that highlights patterns, trends, and insights that might be less obvious in raw data or summary tables.

Implantation

Step No.1 - Import the Dataset.

```
import pandas as pd

import numpy as np

df = pd.read_csv(r"C:\Users\saira\Downloads\Retail_Sales_Data (1).csv")

df.head()
```

Transaction Date Region Product Category Quantity Sold Sales Amount Customer Name Transaction IDPayment Method

0 2019-01-16 West Home Decor 9 909.84 Melinda Pham 7b094307-bcd3-4f16-84a7-2bca783fff4f Credit Card

1 2021-09-17 North Clothing 8 900.29 Shelly Perez fb437a2e-4ebf-4807-b84e-f2dfae83541a Credit Card

2 2020-03-27 East Electronics 3 506.07 Scott White b6ead965-ed1c-4bdc-95ac-864685467abd Online Banking

3 2019-02-11 South Clothing 9 744.70 Gloria Williams 400773f4-a820-47b6-b3c4-2cc2a5467e73 Cash

4 2020-01-15 East Books 4 245.55 Michael Sims 10b62e7a-38f8-4f27-a989-b99b55d76223 Cash

df.tail()

Transaction Date Region Product Category Quantity Sold Sales Amount Customer Name Transaction IDPayment Method

95 2020-06-26 East Electronics 3 914.06 Erica Franklin DVM 56855833-4f68-4312-b5e0-88c7dc7ce72b Online Banking

96 2021-07-04 South Electronics 3 652.93 Ricky Walsh 7e03f607-9b90-4075-b05f-e704c81fb165 PayPal

97 2020-04-08 North Books 9 640.88 Luis Wong 4f1f1533-6fb0-468d-80ac-a8c1db865b1a Online Banking

98 2021-12-24 East Electronics 1 727.21 Christopher Reese 0c3d09c3-469c-4b2f-860e-89365bab7f88 Online Banking

99 2020-10-04 South Clothing 4 554.22 Barry Johnson 95921b52-e166-4d8f-86c6-2149cf1398d8 Credit Card

Step No2. - Explore the Dataset.

df.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 100 entries, 0 to 99

Data columns (total 8 columns):

\# Column Non-Null Count Dtype

```
---  ------            -------------  -----
 0   Transaction Date  100 non-null    object
 1   Region            100 non-null    object
 2   Product Category  100 non-null    object
 3   Quantity Sold     100 non-null    int64
 4   Sales Amount      100 non-null    float64
 5   Customer Name     100 non-null    object
 6   Transaction ID    100 non-null    object
 7   Payment Method    100 non-null    object
dtypes: float64(1), int64(1), object(6)
memory usage: 6.4+ KB
df.describe()
       Quantity Sold  Sales Amount
count  100.000000     100.000000
mean   5.700000       544.873300
std    2.904194       276.530738
min    1.000000       23.140000
25%    3.000000       336.295000
50%    5.500000       554.715000
75%    8.000000       781.757500
max    10.000000      984.850000
df.shape
(100, 8)
df.columns
```

```
Index(['Transaction Date', 'Region', 'Product Category', 'Quantity Sold',
       'Sales Amount', 'Customer Name', 'Transaction ID', 'Payment Method'],
      dtype='object')
```

```
df.dtypes
```

```
Transaction Date     object
Region               object
Product Category     object
Quantity Sold         int64
Sales Amount        float64
Customer Name        object
Transaction ID       object
Payment Method       object
dtype: object
```

Step No.3 - Identify relevant variables.

```
df['Region'].unique()
```

```
array(['West', 'North', 'East', 'South'], dtype=object)
```

```
df['Sales Amount'].describe()
```

```
count    100.000000
mean     544.873300
std      276.530738
min       23.140000
25%      336.295000
50%      554.715000
75%      781.757500
```

max 984.850000

Name: Sales Amount, dtype: float64

df['Product Category'].unique()

array(['Home Decor', 'Clothing', 'Electronics', 'Books'], dtype=object)

Step No.4 - Group sales data by region and calculate total sales amount.

sales_by_region = df.groupby('Region')['Sales Amount'].sum().reset_index()

sales_by_region

	Region	Sales Amount
0	East	14382.28
1	North	13031.74
2	South	11300.33
3	West	15772.98

Step No.5 - Create bar plots or pie charts to visualize the sales distribution by region.

import matplotlib.pyplot as plt

Bar plot

plt.figure(figsize=(10, 6))

plt.bar(sales_by_region['Region'], sales_by_region['Sales Amount'])

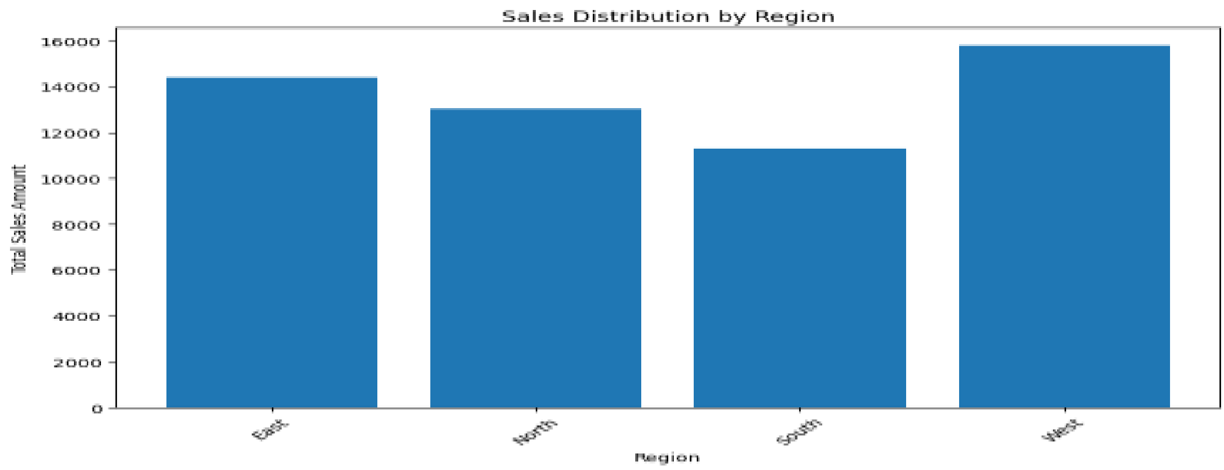
plt.xlabel('Region')

plt.ylabel('Total Sales Amount')

plt.title('Sales Distribution by Region')

plt.xticks(rotation=45)

plt.show()



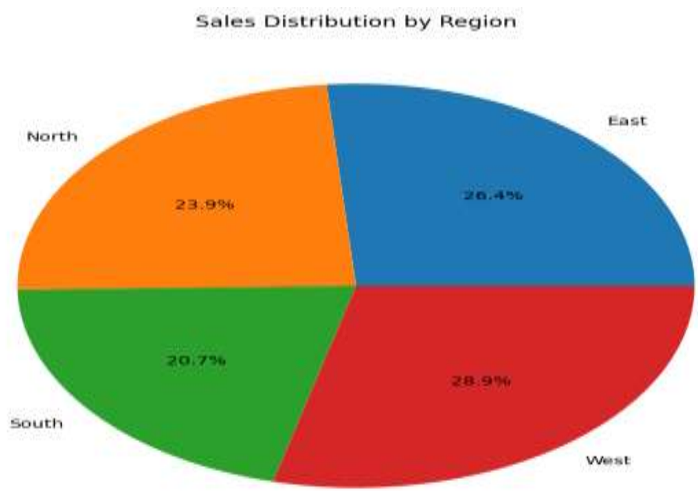
Pie chart

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

plt.pie(sales_by_region['SalesAmount'],labels=sales_by_region['Region'], autopct='%1.1f%%')

plt.title('Sales Distribution by Region')

plt.show()



Step No.6 - Identify top-performing regions.

Sort the regions by sales amount in descending order

top_regions = sales_by_region.sort_values(by='Sales Amount', ascending=False)

top_regions

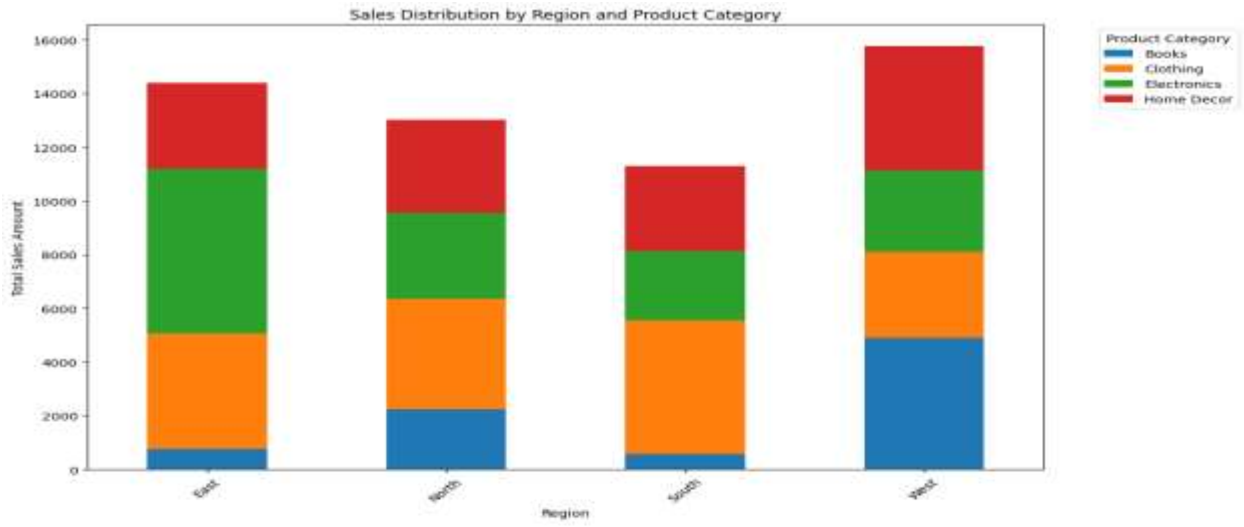
	Region	Sales Amount
3	West	15772.98
0	East	14382.28
1	North	13031.74
2	South	11300.33

Step No.7 - Group sales data by region and product category to calculate total sales amount for each combination.

```
sales_by_region_category = df.groupby(['Region', 'Product Category'])['Sales Amount'].sum().unstack().fillna(0)

sales_by_region_category
```

Product Category	Books	Clothing	Electronics	Home Decor
Region				
East	759.89	4293.54	6153.44	3175.41
North	2235.48	4121.55	3208.94	3465.77
South	573.38	4977.85	2581.59	3167.51
West	4907.12	3185.51	3043.47	4636.88

```
# Step No.8 - Create stacked bar plots or grouped bar plots.

# Stacked bar plot

sales_by_region_category.plot(kind='bar', stacked=True, figsize=(12, 8))

plt.xlabel('Region')

plt.ylabel('Total Sales Amount')

plt.title('Sales Distribution by Region and Product Category')

plt.legend(title='Product Category', bbox_to_anchor=(1.05, 1), loc='upper left')

plt.xticks(rotation=45)

plt.show()
```



Grouped bar plot

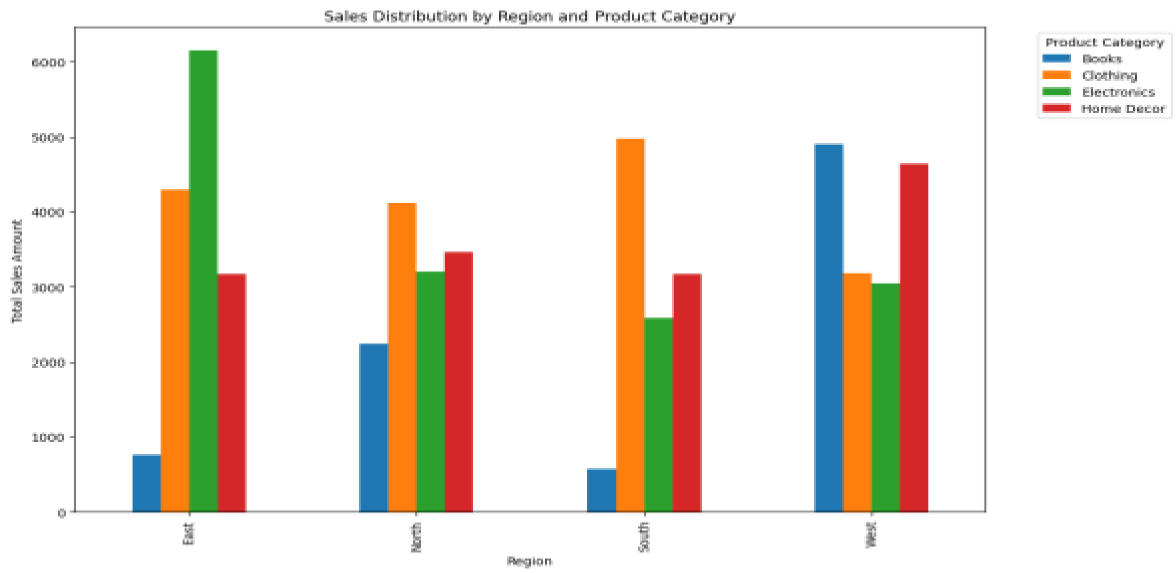
sales_by_region_category.plot(kind='bar', figsize=(12, 8))

plt.xlabel('Region')

plt.ylabel('Total Sales Amount')

plt.title('Sales Distribution by Region and Product Category')

plt.legend(title='Product Category', bbox_to_anchor=(1.05, 1), loc='upper left')



Conlusion : We can successfully analysing sales performance by region in retail company on a “Reatail_sales_data.csv”. This analysis helps in identifying in top performing regions and understanding sales distribution and also strategic decision making.

Lab Assignment No.	13
Title	**Time Series Data Analysis** **Problem statement:** Analysis and Visualization of Stock Market Data **Dataset:** "Stock_Prices.csv" **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.
Roll No.	
Class	BE AI & DS
Date Of Completion	
Subject	Computer Laboratory I[417525]
Assessment Marks	
Assessor's Sign	

Experiment No. 13

Aim : Time Series Data Analysis.

Problem statement: Analysis and Visualization of Stock Market Data and perform following tasks – 1. Import the dataset. 2. Explore the dataset. 3. Ensure that the date column is in the appropriate format. 4. Plot line charts or time series plots. 5. Calculate and plot moving averages or rolling averages. 6. Perform seasonality analysis to identify periodic patterns. 7. Analyze and plot the correlation. and 8. Use autoregressive integrated moving average (ARIMA) models or exponential smoothing models to forecast future.

Dataset: "Stock_Prices.csv"

Software Requirements : Python and Jupyter notebook.

Hardware Requirements : 8GB RAM, Storage and Processor.

Objective : i) Visualize the historical trends. ii) Calculate and plot moving averages. iii) Perform seasonality Analysis. iv) Build and evaluate forcasting models.

Theory : Time Series Analysis

Time series analysis is a statistical technique used to analyze time-ordered data points to identify patterns, trends, and insights over time. It is widely used in various fields, such as finance, economics, environmental science, and engineering, to make forecasts, understand historical patterns, and guide decision-making.

Components of Time Series:

1.Trend: The long-term movement or direction in the data. For example, an upward trend in sales over several years.

2.Seasonality: Regular and predictable patterns that repeat at specific intervals, such as seasonal variations in retail sales.

3.Cyclic Patterns: Fluctuations in data that occur at irregular intervals, often related to economic cycles or business cycles.

4.Noise: Random variability or irregular fluctuations in the data that cannot be attributed to the trend, seasonality, or cycles.

Methods of Time series analysis

1.Exploratory Data Analysis (EDA) for Time Series

To gain initial insights into the time series data, identify patterns, and detect anomalies. Time Series Plot: A line plot showing the data points over time to visualize trends, seasonality, and any irregular patterns. Seasonal Plots: Plots that show data points for each season (e.g., month, quarter) to assess seasonal patterns.

2. Moving Averages

To smooth out short-term fluctuations and highlight longer-term trends in time series data. Simple Moving Average (SMA):The average of data points within a fixed window size. Each data point in the time series is replaced by the average of its neighboring values within the window. Exponential Moving Average (EMA: A type of weighted moving average that gives exponentially more weight to recent observations.

3.ARIMA Models (AutoRegressive Integrated Moving Average)

To model and forecast stationary time series data, taking into account autocorrelation. AR (AutoRegressive) Component: Models the current value as a linear combination of previous values. I (Integrated) Component: Involves differencing the series to make it stationary. MA (Moving Average) Component: Models the current value as a function of past forecast errors. SARIMA (Seasonal ARIMA): Extends ARIMA to handle seasonality by including seasonal autoregressive and moving average terms.

Implementation

Step 1: Importing the Dataset.

```
import pandas as pd

import numpy as np

df = pd.read_csv(r"C:\Users\saira\Downloads\Google_Stock_Price_Test.csv")

df.head()
```

Date Open High Low Close Volume

0 1/3/2017 778.81 789.63 775.80 786.14 1,657,300

1 1/4/2017 788.36 791.34 783.16 786.90 1,073,000

2 1/5/2017 786.08 794.48 785.02 794.02 1,335,200

3 1/6/2017 795.26 807.90 792.20 806.15 1,640,200

4 1/9/2017 806.40 809.97 802.83 806.65 1,272,400

df.tail()

Date Open High Low Close Volume

15 1/25/2017 829.62 835.77 825.06 835.67 1,494,500

16 1/26/2017 837.81 838.00 827.01 832.15 2,973,900

17 1/27/2017 834.71 841.95 820.44 823.31 2,965,800

18 1/30/2017 814.66 815.84 799.80 802.32 3,246,600

19 1/31/2017 796.86 801.25 790.52 796.79 2,160,600

Step 2: Exploring the Dataset.

df.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 20 entries, 0 to 19

Data columns (total 6 columns):

```
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    20 non-null     object
 1   Open    20 non-null     float64
 2   High    20 non-null     float64
 3   Low     20 non-null     float64
 4   Close   20 non-null     float64
 5   Volume  20 non-null     object
```

dtypes: float64(4), object(2)

memory usage: 1.1+ KB

df.describe()

	Open	High	Low	Close
count	20.000000	20.000000	20.000000	20.000000
mean	807.526000	811.926500	801.949500	807.904500
std	15.125428	14.381198	13.278607	13.210088
min	778.810000	789.630000	775.800000	786.140000
25%	802.965000	806.735000	797.427500	802.282500
50%	806.995000	808.640000	801.530000	806.110000
75%	809.560000	817.097500	804.477500	810.760000
max	837.810000	841.950000	827.010000	835.670000

df.isnull().sum()

Date 0

Open 0

High 0

Low 0

Close 0

Volume 0

dtype: int64

Step 3: Ensuring the Date Column is in the Appropriate Format.

df['Date'] = pd.to_datetime(df['Date'])

df.set_index('Date', inplace=True)

df.info()

<class 'pandas.core.frame.DataFrame'>

```
DatetimeIndex: 20 entries, 2017-01-03 to 2017-01-31
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Open    20 non-null     float64
 1   High    20 non-null     float64
 2   Low     20 non-null     float64
 3   Close   20 non-null     float64
 4   Volume  20 non-null     object
dtypes: float64(4), object(1)
memory usage: 960.0+ bytes
```

Step 4: Plotting Line Charts to Visualize Historical Stock Price Trends.

```
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 6))
plt.plot(df['Close'], label='Close Price')
plt.title('Historical Stock Price Trends')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```

Step 5: Calculating and Plotting Moving Averages.

```
# Calculate the moving averages

df['MA_50'] = df['Close'].rolling(window=50).mean()

df['MA_200'] = df['Close'].rolling(window=200).mean()

df.MA_50
```

Date

2017-01-03 NaN

2017-01-04 NaN

2017-01-05 NaN

2017-01-06 NaN

2017-01-09 NaN

2017-01-10 NaN

2017-01-11 NaN

2017-01-12 NaN

2017-01-13 NaN

2017-01-17 NaN

2017-01-18 NaN

2017-01-19 NaN

2017-01-20 NaN

2017-01-23 NaN

2017-01-24 NaN

2017-01-25 NaN

2017-01-26 NaN

2017-01-27 NaN

2017-01-30 NaN

2017-01-31 NaN

Name: MA_50, dtype: float64

df.MA_200

Date

2017-01-03 NaN

2017-01-04 NaN

2017-01-05 NaN

2017-01-06 NaN

2017-01-09 NaN

2017-01-10 NaN

2017-01-11 NaN

2017-01-12 NaN

2017-01-13 NaN

2017-01-17 NaN

2017-01-18 NaN

2017-01-19 NaN

2017-01-20 NaN

2017-01-23 NaN

2017-01-24 NaN

2017-01-25 NaN

2017-01-26 NaN

2017-01-27 NaN

2017-01-30 NaN

2017-01-31 NaN

Name: MA_200, dtype: float64

```
# Plot the closing price along with moving averages

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

plt.plot(df['Close'], label='Close Price')

plt.plot(df['MA_50'], label='50-Day Moving Average')

plt.plot(df['MA_200'], label='200-Day Moving Average')

plt.title('Stock Price with Moving Averages')

plt.xlabel('Date')

plt.ylabel('Price')

plt.legend()

plt.show()
```

Step 6: Performing Seasonality Analysis.

```
# Create a month column for seasonality analysis
df['Month'] = df.index.month
df['Weekday'] = df.index.weekday
df['Year'] = df.index.year
df.Month
Date
2017-01-03    1
2017-01-04    1
2017-01-05    1
2017-01-06    1
2017-01-09    1
2017-01-10    1
2017-01-11    1
2017-01-12    1
2017-01-13    1
2017-01-17    1
2017-01-18    1
2017-01-19    1
2017-01-20    1
2017-01-23    1
2017-01-24    1
2017-01-25    1
2017-01-26    1
```

2017-01-27 1

2017-01-30 1

2017-01-31 1

Name: Month, dtype: int32

df.Weekday

Date

2017-01-03 1

2017-01-04 2

2017-01-05 3

2017-01-06 4

2017-01-09 0

2017-01-10 1

2017-01-11 2

2017-01-12 3

2017-01-13 4

2017-01-17 1

2017-01-18 2

2017-01-19 3

2017-01-20 4

2017-01-23 0

2017-01-24 1

2017-01-25 2

2017-01-26 3

2017-01-27 4

2017-01-30 0

2017-01-31 1

Name: Weekday, dtype: int32

df.Year

Date

2017-01-03 2017

2017-01-04 2017

2017-01-05 2017

2017-01-06 2017

2017-01-09 2017

2017-01-10 2017

2017-01-11 2017

2017-01-12 2017

2017-01-13 2017

2017-01-17 2017

2017-01-18 2017

2017-01-19 2017

2017-01-20 2017

2017-01-23 2017

2017-01-24 2017

2017-01-25 2017

2017-01-26 2017

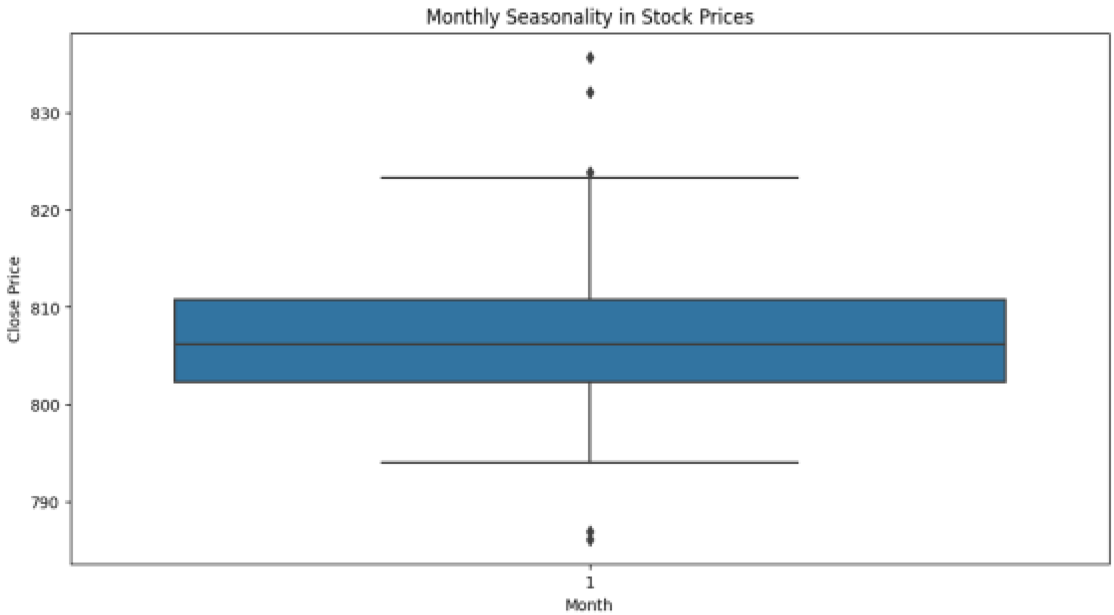
2017-01-27 2017

2017-01-30 2017

2017-01-31 2017

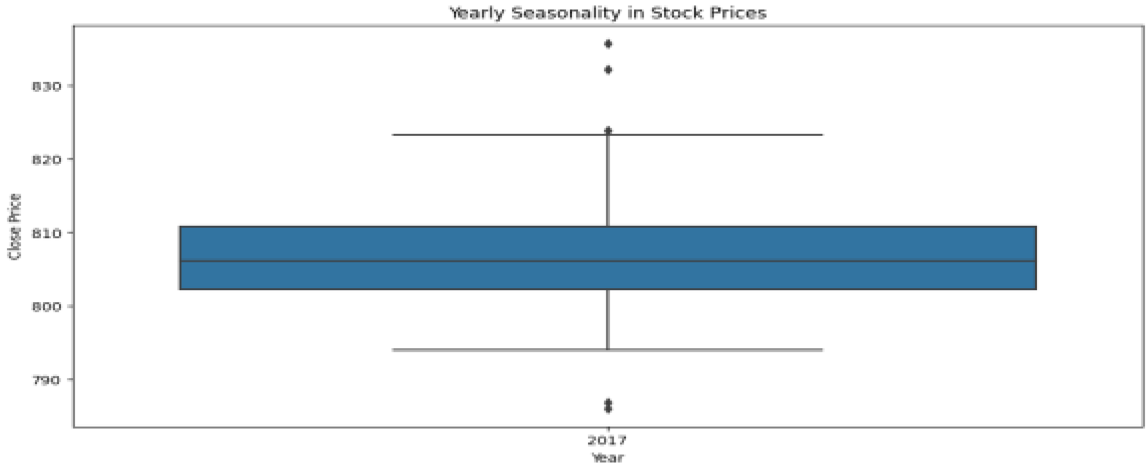
Name: Year, dtype: int32

```
import seaborn as sns

# Boxplot to show monthly seasonality

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

sns.boxplot(x='Month', y='Close', data=df)

plt.title('Monthly Seasonality in Stock Prices')

plt.xlabel('Month')

plt.ylabel('Close Price')

plt.show()
```




```
# Boxplot to show yearly seasonality

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

sns.boxplot(x='Year', y='Close', data=df)

plt.title('Yearly Seasonality in Stock Prices')

plt.xlabel('Year')

plt.ylabel('Close Price')
```

plt.show()



Boxplot to show weekday seasonality

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

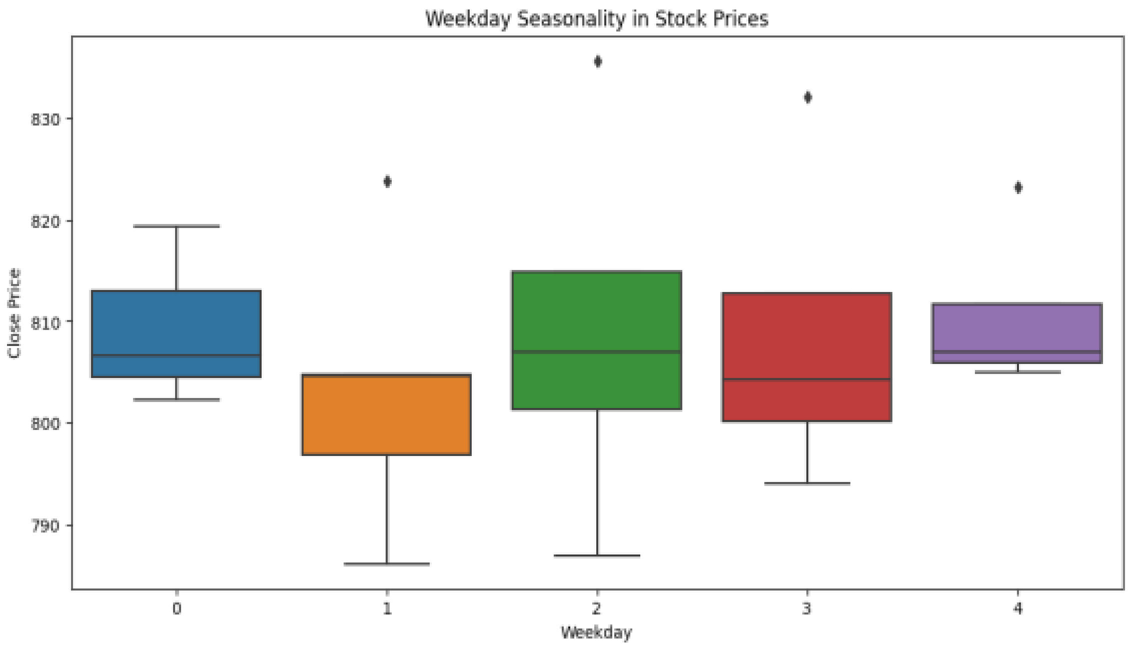
sns.boxplot(x='Weekday', y='Close', data=df)

plt.title('Weekday Seasonality in Stock Prices')

plt.xlabel('Weekday')

plt.ylabel('Close Price')

plt.show()



Step 7: Analyzing and Plotting Correlation Between Stock Prices and Other Variables.

```
df['Volume'] = df['Volume'].str.replace(',', '').astype(float)

df.Volume

Date

2017-01-03    1657300.0
2017-01-04    1073000.0
2017-01-05    1335200.0
2017-01-06    1640200.0
2017-01-09    1272400.0
2017-01-10    1176800.0
2017-01-11    1065900.0
2017-01-12    1353100.0
2017-01-13    1099200.0
2017-01-17    1362100.0
2017-01-18    1294400.0
2017-01-19     919300.0
2017-01-20    1670000.0
2017-01-23    1963600.0
2017-01-24    1474000.0
2017-01-25    1494500.0
2017-01-26    2973900.0
2017-01-27    2965800.0
2017-01-30    3246600.0
2017-01-31    2160600.0
```

Name: Volume, dtype: float64

correlation_matrix = df.corr()

correlation_matrix

Open High Low Close Volume MA_50 MA_200 Month Weekday
Year

Open 1.000000 0.960636 0.972508 0.907690 0.502175 NaN
NaN NaN 0.119306 NaN

High 0.960636 1.000000 0.946877 0.947077 0.539165 NaN
NaN NaN 0.123275 NaN

Low 0.972508 0.946877 1.000000 0.951060 0.332733 NaN
NaN NaN 0.148737 NaN

Close 0.907690 0.947077 0.951060 1.000000 0.345362 NaN
NaN NaN 0.110724 NaN

Volume 0.502175 0.539165 0.332733 0.345362 1.000000
NaN NaN NaN -0.062616 NaN

MA_50 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN

MA_200 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN

Month NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN

Weekday 0.119306 0.123275 0.148737 0.110724 -0.062616
NaN NaN NaN 1.000000 NaN

Year NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN

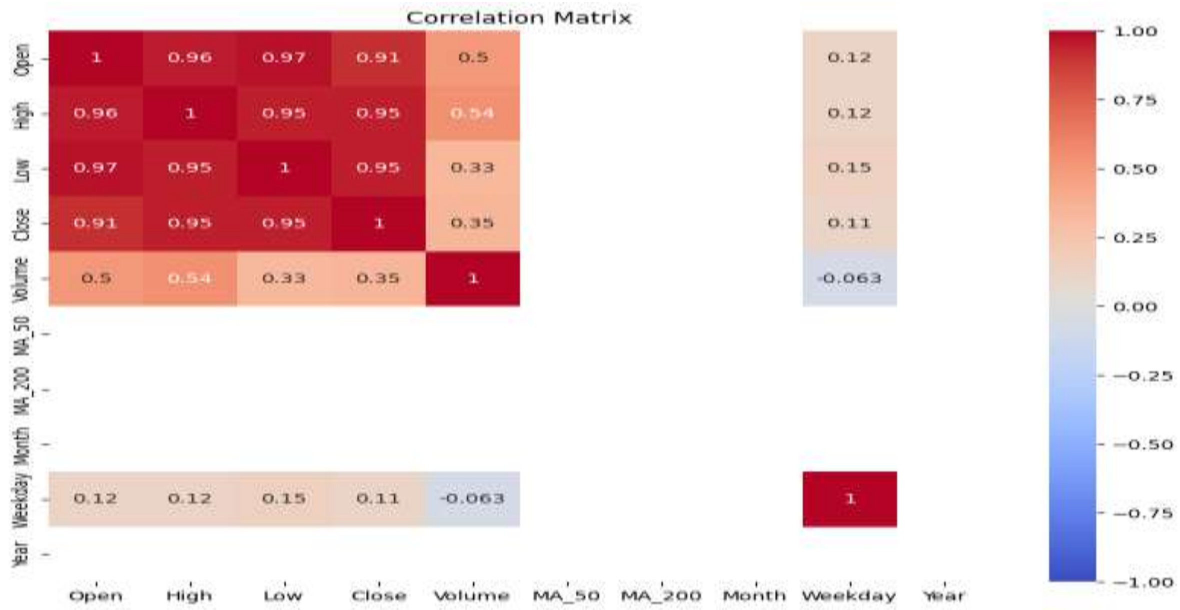
Plot the heatmap

plt.figure(figsize=(10, 8))

sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', vmin=-1, vmax=1)

plt.title('Correlation Matrix')

plt.show()



Step 8: Forecasting Future Stock Prices Using ARIMA.

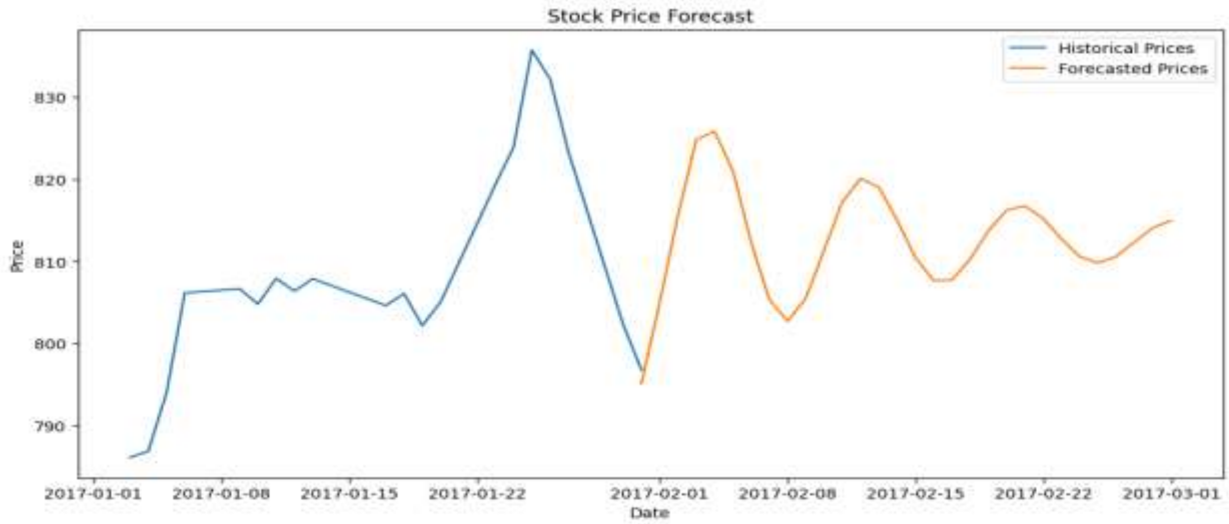
```
!pip install statsmodels

from statsmodels.tsa.arima.model import ARIMA

import warnings

warnings.filterwarnings('ignore')

# Fit the ARIMA model

model = ARIMA(df['Close'], order=(5, 1, 0))

model_fit = model.fit()

# Forecast the future prices

forecast = model_fit.forecast(steps=30)

# Plot the forecast

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

plt.plot(df['Close'], label='Historical Prices')

plt.plot(pd.date_range(start=df.index[-1], periods=30, freq='D'), forecast, label='Forecasted Prices')

plt.title('Stock Price Forecast')
```

plt.xlabel('Date')

plt.ylabel('Price')

plt.legend()

plt.show()



from statsmodels.tsa.holtwinters import ExponentialSmoothing

Fit the Exponential Smoothing model

model_es = ExponentialSmoothing(df['Close'], trend='add', seasonal=None, seasonal_periods=None)

model_fit_es = model_es.fit()

Forecast the future prices

forecast_steps = 30 # Number of days to forecast

forecast_es = model_fit_es.forecast(steps=forecast_steps)

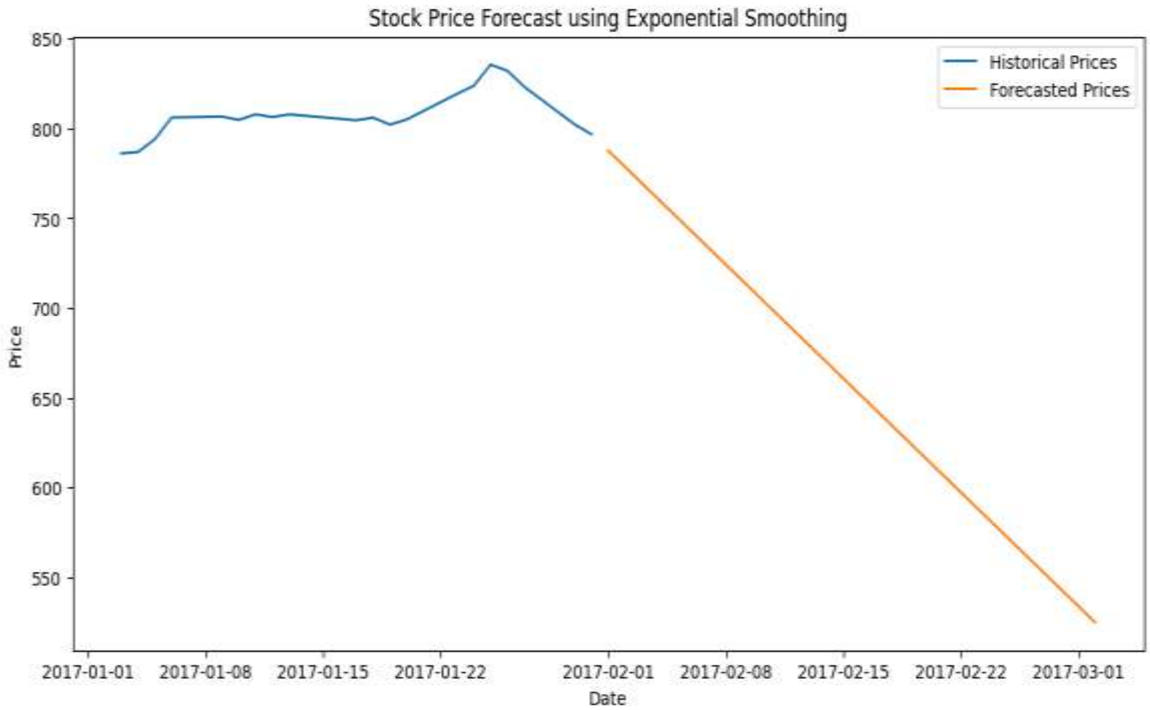
Plot the forecast

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

plt.plot(df['Close'], label='Historical Prices')

plt.plot(pd.date_range(start=df.index[-1], periods=forecast_steps+1, freq='D')[1:], forecast_es, label='Forecasted Prices')

```
plt.title('Stock Price Forecast using Exponential Smoothing')

plt.xlabel('Date')

plt.ylabel('Price')

plt.legend()

plt.show()
```



Conclusion : We can successfully analysis and visualization of stock market data on a "Stock_prices.csv" dataset and also we can easily identify trends and patterns, and build predictive model to forcast future stock prices.