4.1 Experiment No. 1

Aim:

Data Loading, Storage and File Format
Analyzing Sales Data from Multiple File Format. Sales data in multiple file formats (e.g., CSV, Excel, JSON)

Objective:

To Study:

1. Loading of multiple file formats, perform data cleaning, transformation and analysis on the dataset.
2. Loading and analyzing sales data from different file formats, including CSV, Excel, and JSON, and perform data cleaning, transformation, and analysis on the dataset

Theory:

Reading data and making it accessible (often called data loading) is a necessary first step for using most of the tools. The term parsing is also sometimes used to describe loading text data and interpreting it as tables and different data types. To focus on data input and output using pandas, there are numerous tools in other libraries to help with reading and writing data in various formats. Pandas features a number of functions for reading tabular data as a Data Frame object. pandas.read_csv is one of the most frequently used to load csv file.
Some of data loading functions in pandas summarizes as;

Functions	**Description**
read_csv	Load delimited data from a file, URL, or file-like object; use comma as default delimiter
read_fwf	Read data in fixed-width column format (i.e., no delimiters)
read_clipboard	Variation of read_csv that reads data from the clipboard; useful for converting tables from web pages
read_excel	Read tabular data from an Excel XLS or XLSX file
read_hdf	Read HDF5 files written by pandas
read_html	Read all tables found in the given HTML document
read_json	Read data from a JSON (JavaScript Object Notation) string representation, file, URL, or file-like object
read_feather	Read the Feather binary file format
read_orc	Read the Apache ORC binary file format
read_parquet	Read the Apache Parquet binary file format
read_pickle	Read an object stored by pandas using the Python pickle format
read_sas	Read a SAS dataset stored in one of the SAS system's custom storage formats
read_spss	Read a data file created by SPSS
read_sql	Read the results of a SQL query (using SQLAlchemy)
read_sql_table	Read a whole SQL table (using SQLAlchemy); equivalent to using a query that selects everything in that table using read_sql
read_stata	Read a dataset from Stata file format
read_xml	Read a table of data from an XML file

Indexing
Can treat one or more columns as the returned Data Frame, and whether to get column names from the file, arguments you provide, or not at all.

Type inference and data conversion
Includes the user-defined value conversions and custom list of missing value markers

Date and time parsing
Includes a combining capability, including combining date and time information spread over multiple columns into a single column in the result.

Iterating
Support for iterating over chunks of very large files.

Unclean data issues
Includes skipping rows or a footer, comments, or other minor things like numeric data with thousands separated by commas Because of how messy data in the real world can be, some of the data loading functions (especially pandas.read_csv) have accumulated a long list of optional arguments over time.
Some of these functions perform type inference, because the column data types are not part of the data format. That means you don't necessarily have to specify which columns are numeric, integer, Boolean, or string. Other data formats, like HDF5, ORC, and Parquet, have the data type information embedded in the format.

Input

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

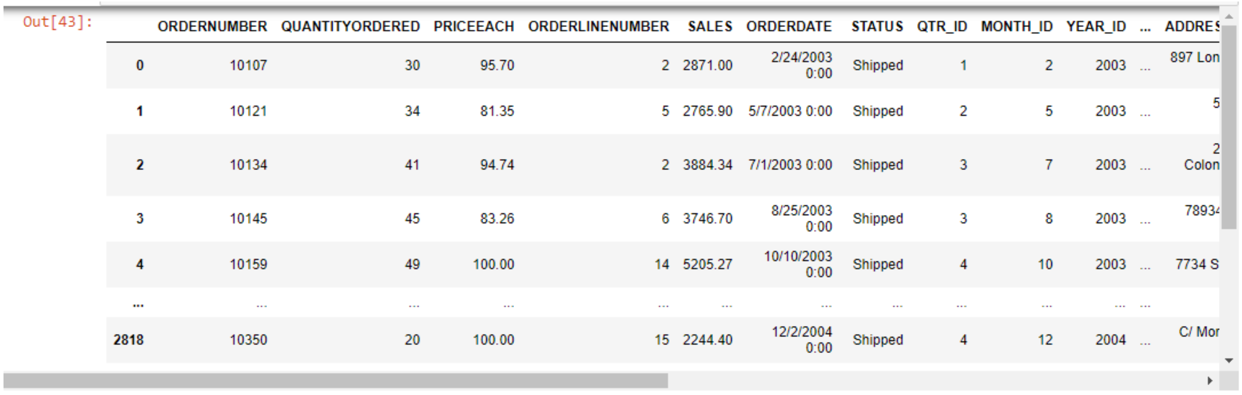
1) Load the sales data from each file format into the appropriate data structures or data frames.
2) Explore the structure and content of the loaded data, identifying any inconsistencies, missing values, or data quality issues.
3) Perform data cleaning operations, such as handling missing values, removing duplicates, or correcting inconsistencies.
4) Convert the data into a unified format, such as a common data frame 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.

Output

```
[41]: import pandas as pd
      import numpy as np
      import matplotlib.pyplot as plt
```

```
[42]: df = pd.read_csv('sales_data_sample.csv',encoding='latin-1')
```

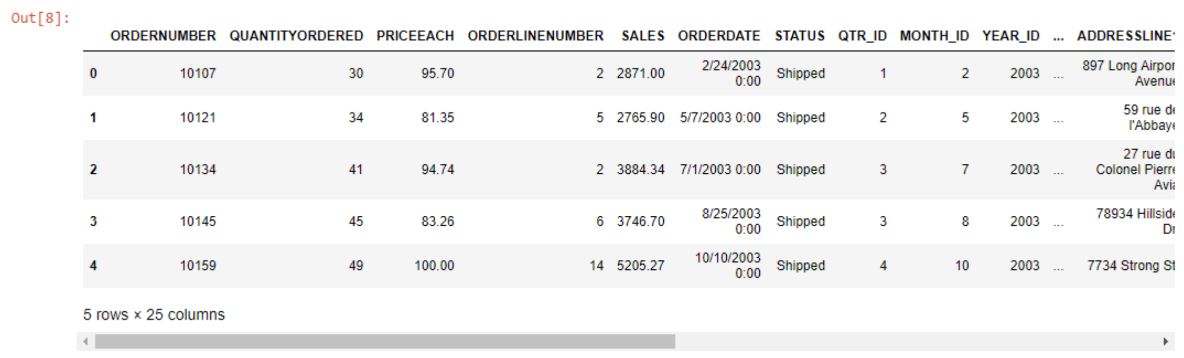
```
[43]: df
```



Out[43]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS	QTR_ID	MONTH_ID	YEAR_ID	...	ADDRES
0	10107	30	95.70	2	2871.00	2/24/2003 0:00	Shipped	1	2	2003	...	897 Lon
1	10121	34	81.35	5	2765.90	5/7/2003 0:00	Shipped	2	5	2003	...	5
2	10134	41	94.74	2	3884.34	7/1/2003 0:00	Shipped	3	7	2003	...	2 Colon
3	10145	45	83.26	6	3746.70	8/25/2003 0:00	Shipped	3	8	2003	...	7893
4	10159	49	100.00	14	5205.27	10/10/2003 0:00	Shipped	4	10	2003	...	7734 S
...	...	...	...	...	...	...	...	...	...	...	...	
2818	10350	20	100.00	15	2244.40	12/2/2004 0:00	Shipped	4	12	2004	...	C/ Mor

```
[44]: df.head()
```

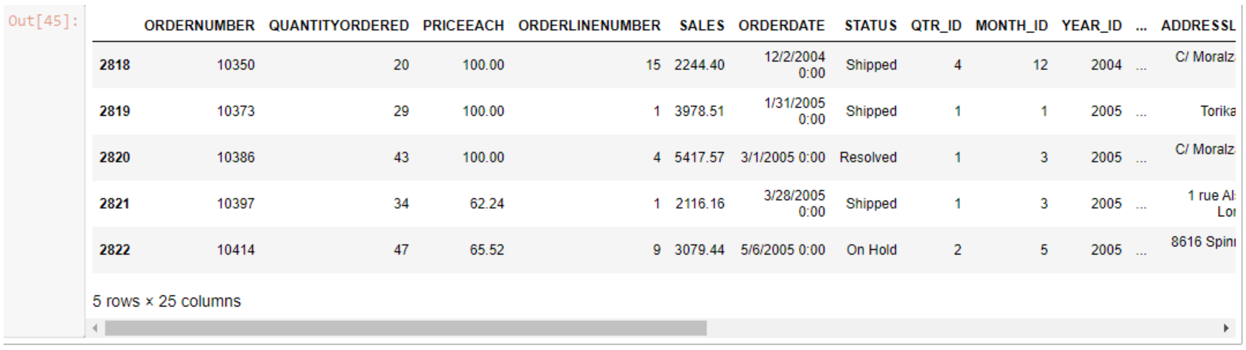


Out[8]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS	QTR_ID	MONTH_ID	YEAR_ID	...	ADDRESSLINE
0	10107	30	95.70	2	2871.00	2/24/2003 0:00	Shipped	1	2	2003	...	897 Long Airpor Avenu
1	10121	34	81.35	5	2765.90	5/7/2003 0:00	Shipped	2	5	2003	...	59 rue de l'Abbaye
2	10134	41	94.74	2	3884.34	7/1/2003 0:00	Shipped	3	7	2003	...	27 rue du Colonel Pierre Avi
3	10145	45	83.26	6	3746.70	8/25/2003 0:00	Shipped	3	8	2003	...	78934 Hillside Dr
4	10159	49	100.00	14	5205.27	10/10/2003 0:00	Shipped	4	10	2003	...	7734 Strong St

5 rows × 25 columns

```
[45]: df.tail()
```



Out[45]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS	QTR_ID	MONTH_ID	YEAR_ID	...	ADDRESSL
2818	10350	20	100.00	15	2244.40	12/2/2004 0:00	Shipped	4	12	2004	...	C/ Moralz
2819	10373	29	100.00	1	3978.51	1/31/2005 0:00	Shipped	1	1	2005	...	Torika
2820	10386	43	100.00	4	5417.57	3/1/2005 0:00	Resolved	1	3	2005	...	C/ Moralz
2821	10397	34	62.24	1	2116.16	3/28/2005 0:00	Shipped	1	3	2005	...	1 rue Al Lor
2822	10414	47	65.52	9	3079.44	5/6/2005 0:00	On Hold	2	5	2005	...	8616 Spin

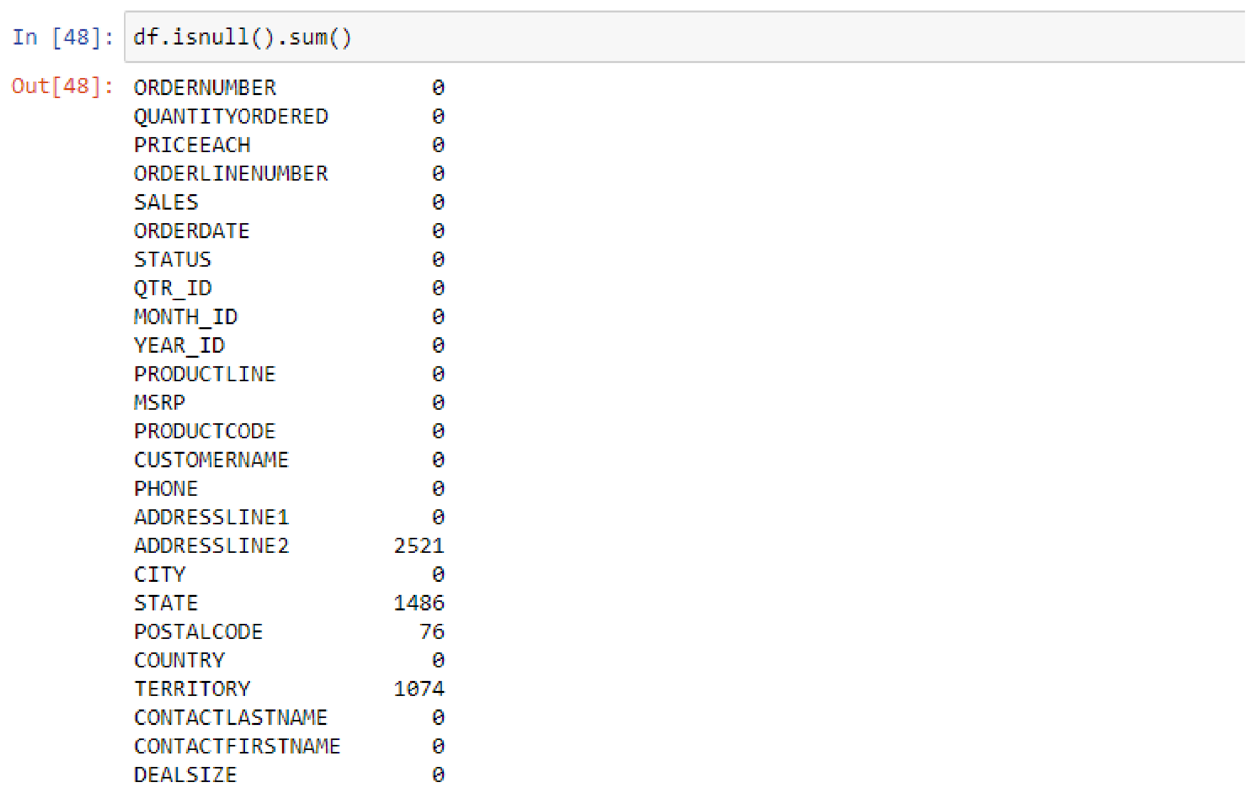
5 rows × 25 columns

```
[10]: df.shape
```

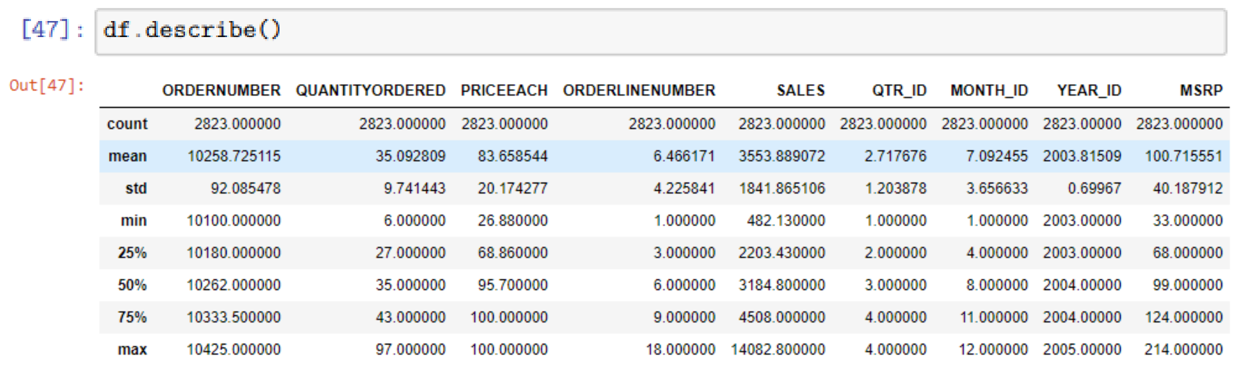
```
[10]: (2823, 25)
```

```
[46]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2823 entries, 0 to 2822
Data columns (total 25 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   ORDERNUMBER       2823 non-null   int64
 1   QUANTITYORDERED   2823 non-null   int64
 2   PRICEEACH         2823 non-null   float64
 3   ORDERLINENUMBER   2823 non-null   int64
 4   SALES             2823 non-null   float64
 5   ORDERDATE         2823 non-null   object
 6   STATUS            2823 non-null   object
 7   QTR_ID            2823 non-null   int64
 8   MONTH_ID          2823 non-null   int64
 9   YEAR_ID           2823 non-null   int64
 10  PRODUCTLINE       2823 non-null   object
 11  MSRP              2823 non-null   int64
 12  PRODUCTCODE       2823 non-null   object
 13  CUSTOMERNAME      2823 non-null   object
 14  PHONE             2823 non-null   object
 15  ADDRESSLINE1      2823 non-null   object
 16  ADDRESSLINE2      302 non-null    object
 17  CITY              2823 non-null   object
 18  STATE             1337 non-null   object
 19  POSTALCODE        2747 non-null   object
 20  COUNTRY           2823 non-null   object
 21  TERRITORY         1749 non-null   object
 22  CONTACTLASTNAME   2823 non-null   object
 23  CONTACTFIRSTNAME  2823 non-null   object
 24  DEALSIZE          2823 non-null   object
dtypes: float64(2), int64(7), object(16)
memory usage: 551.5+ KB
```



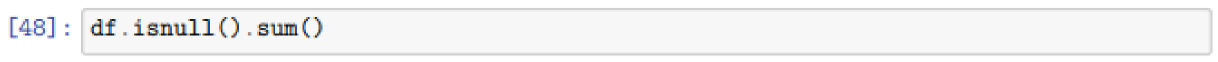
```
In [48]: df.isnull().sum()

Out[48]: ORDERNUMBER            0
         QUANTITYORDERED        0
         PRICEEACH              0
         ORDERLINENUMBER        0
         SALES                  0
         ORDERDATE              0
         STATUS                 0
         QTR_ID                 0
         MONTH_ID               0
         YEAR_ID                0
         PRODUCTLINE            0
         MSRP                   0
         PRODUCTCODE            0
         CUSTOMERNAME           0
         PHONE                  0
         ADDRESSLINE1           0
         ADDRESSLINE2        2521
         CITY                   0
         STATE               1486
         POSTALCODE            76
         COUNTRY                0
         TERRITORY           1074
         CONTACTLASTNAME        0
         CONTACTFIRSTNAME       0
         DEALSIZE               0
```



```
[47]: df.describe()
```

Out[47]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	QTR_ID	MONTH_ID	YEAR_ID	MSRP
count	2823.000000	2823.000000	2823.000000	2823.000000	2823.000000	2823.000000	2823.000000	2823.00000	2823.000000
mean	10258.725115	35.092809	83.658544	6.466171	3553.889072	2.717676	7.092455	2003.81509	100.715551
std	92.085478	9.741443	20.174277	4.225841	1841.865106	1.203878	3.656633	0.69967	40.187912
min	10100.000000	6.000000	26.880000	1.000000	482.130000	1.000000	1.000000	2003.00000	33.000000
25%	10180.000000	27.000000	68.860000	3.000000	2203.430000	2.000000	4.000000	2003.00000	68.000000
50%	10262.000000	35.000000	95.700000	6.000000	3184.800000	3.000000	8.000000	2004.00000	99.000000
75%	10333.500000	43.000000	100.000000	9.000000	4508.000000	4.000000	11.000000	2004.00000	124.000000
max	10425.000000	97.000000	100.000000	18.000000	14082.800000	4.000000	12.000000	2005.00000	214.000000



```
[48]: df.isnull().sum()
```

```
[48]: ORDERNUMBER             0
      QUANTITYORDERED         0
      PRICEEACH               0
      ORDERLINENUMBER         0
      SALES                   0
      ORDERDATE               0
      STATUS                  0
      QTR_ID                  0
      MONTH_ID                0
      YEAR_ID                 0
      PRODUCTLINE             0
      MSRP                    0
      PRODUCTCODE             0
      CUSTOMERNAME            0
      PHONE                   0
      ADDRESSLINE1            0
      ADDRESSLINE2         2521
      CITY                    0
      STATE                1486
      POSTALCODE             76
      COUNTRY                 0
      TERRITORY            1074
      CONTACTLASTNAME         0
      CONTACTFIRSTNAME        0
      DEALSIZE                0
      dtype: int64
```

```
[49]: newdf = df.fillna(0)
```

```
[50]: newdf.isnull().sum()
```

```
[50]: ORDERNUMBER         0
      QUANTITYORDERED     0
      PRICEEACH           0
      ORDERLINENUMBER     0
      SALES               0
      ORDERDATE           0
```

```
STATUS              0
QTR_ID              0
MONTH_ID            0
YEAR_ID             0
PRODUCTLINE         0
MSRP                0
PRODUCTCODE         0
CUSTOMERNAME        0
PHONE               0
ADDRESSLINE1        0
ADDRESSLINE2        0
CITY                0
STATE               0
POSTALCODE          0
COUNTRY             0
TERRITORY           0
CONTACTLASTNAME     0
CONTACTFIRSTNAME    0
DEALSIZE            0
dtype: int64
```

```
[51]: newdf
```

Out[51]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS	QTR_ID	MONTH_ID	YEAR_ID	...	ADDRESSL
0	10107	30	95.70	2	2871.00	2/24/2003 0:00	Shipped	1	2	2003	...	897 Long A Av
1	10121	34	81.35	5	2765.90	5/7/2003 0:00	Shipped	2	5	2003	...	59 r l'Ab
2	10134	41	94.74	2	3884.34	7/1/2003 0:00	Shipped	3	7	2003	...	27 r Colonel F
3	10145	45	83.26	6	3746.70	8/25/2003 0:00	Shipped	3	8	2003	...	78934 Hi
4	10159	49	100.00	14	5205.27	10/10/2003 0:00	Shipped	4	10	2003	...	7734 Stror
...	...	...	...	...	...	...	...	...	...	...	...	
2818	10350	20	100.00	15	2244.40	12/2/2004 0:00	Shipped	4	12	2004	...	C/ Moralz
2819	10373	29	100.00	1	3978.51	1/31/2005 0:00	Shipped	1	1	2005	...	Torika
2820	10386	43	100.00	4	5417.57	3/1/2005 0:00	Resolved	1	3	2005	...	C/ Moralz
2821	10397	34	62.24	1	2116.16	3/28/2005 0:00	Shipped	1	3	2005	...	1 rue Al Lor
2822	10414	47	65.52	9	3079.44	5/6/2005 0:00	On Hold	2	5	2005	...	8616 Spin

```
[52]: newdf['LOCATION'] = newdf['CITY']+','+newdf['COUNTRY']
      newdf
```

Out[52]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS	QTR_ID	MONTH_ID	YEAR_ID	...	ADDRESSL
0	10107	30	95.70	2	2871.00	2/24/2003 0:00	Shipped	1	2	2003	...	
1	10121	34	81.35	5	2765.90	5/7/2003 0:00	Shipped	2	5	2003	...	
2	10134	41	94.74	2	3884.34	7/1/2003 0:00	Shipped	3	7	2003	...	
3	10145	45	83.26	6	3746.70	8/25/2003 0:00	Shipped	3	8	2003	...	
4	10159	49	100.00	14	5205.27	10/10/2003 0:00	Shipped	4	10	2003	...	
...	...	...	...	...	...	...	...	...	...	...	...	
2818	10350	20	100.00	15	2244.40	12/2/2004 0:00	Shipped	4	12	2004	...	
2819	10373	29	100.00	1	3978.51	1/31/2005 0:00	Shipped	1	1	2005	...	
2820	10386	43	100.00	4	5417.57	3/1/2005 0:00	Resolved	1	3	2005	...	
2821	10397	34	62.24	1	2116.16	3/28/2005 0:00	Shipped	1	3	2005	...	
2822	10414	47	65.52	9	3079.44	5/6/2005 0:00	On Hold	2	5	2005	...	

2823 rows × 26 columns

[53]:
```python
total_sales = newdf['SALES'].sum()
```

[54]:
```python
total_sales
```

[54]:
```
10032628.85
```

[55]:
```python
avg_order_value = newdf['SALES'].mean()
avg_order_value
```

[55]:
```
3553.889071909316
```

[56]:
```python
product_category = newdf['PRODUCTLINE'].value_counts()
product_category
```

[56]:
```
PRODUCTLINE
Classic Cars        967
Vintage Cars        607
Motorcycles         331
Planes              306
Trucks and Buses    301
Ships               234
Trains               77
Name: count, dtype: int64
```

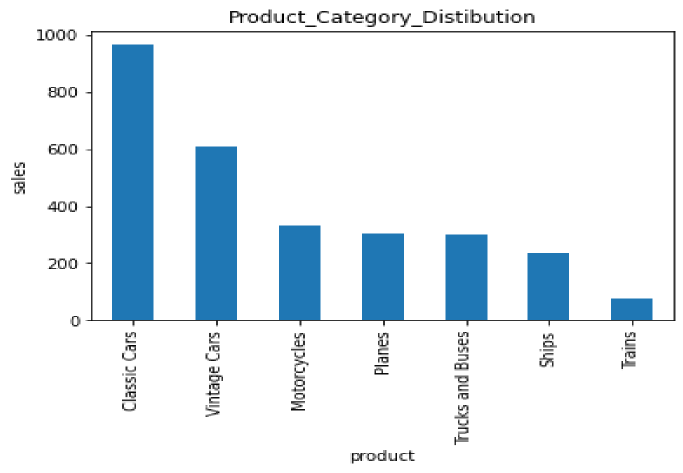
[57]:
```python
newdf
```

Out[57]:

```
      ORDERNUMBER QUANTITYORDERED PRICEEACH ORDERLINENUMBER   SALES  ORDERDATE  STATUS  QTR_ID MONTH_ID YEAR_ID ... ADDRESSL
   0        10107              30     95.70               2 2871.00  2/24/2003  Shipped      1        2    2003 ...
                                                                          0:00
   1        10121              34     81.35               5 2765.90 5/7/2003 0:00 Shipped    2        5    2003 ...
   2        10134              41     94.74               2 3884.34 7/1/2003 0:00 Shipped    3        7    2003 ...
   3        10145              45     83.26               6 3746.70  8/25/2003  Shipped      3        8    2003 ...
                                                                          0:00
   4        10159              49    100.00              14 5205.27 10/10/2003  Shipped      4       10    2003 ...
                                                                          0:00
 ...          ...             ...       ...             ...     ...        ...      ...    ...      ...     ... ...
2818        10350              20    100.00              15 2244.40  12/2/2004  Shipped      4       12    2004 ...
                                                                          0:00
2819        10373              29    100.00               1 3978.51  1/31/2005  Shipped      1        1    2005 ...
                                                                          0:00
2820        10386              43    100.00               4 5417.57 3/1/2005 0:00 Resolved   1        3    2005 ...
2821        10397              34     62.24               1 2116.16  3/28/2005  Shipped      1        3    2005 ...
                                                                          0:00
2822        10414              47     65.52               9 3079.44 5/6/2005 0:00 On Hold    2        5    2005 ...

2823 rows × 26 columns
```

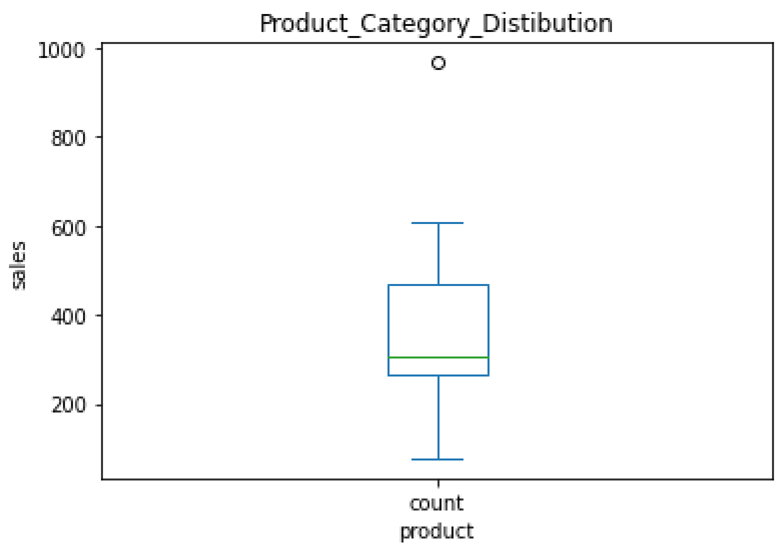
[59]:

```python
product_category.plot(kind='bar')
plt.xlabel("product")
plt.ylabel("sales")
plt.title("Product_Category_Distibution")
plt.show()
```

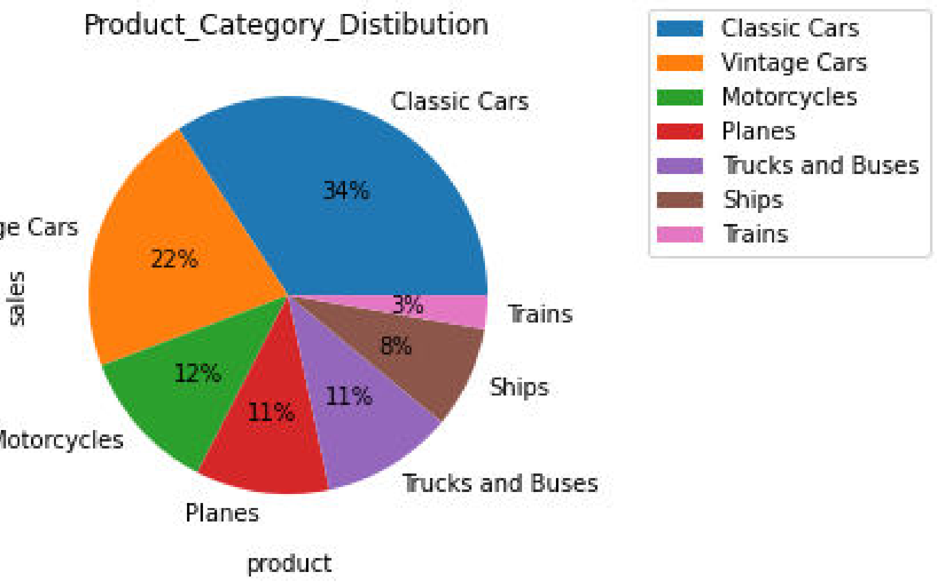


[62]:

```python
product_category.plot(kind='box')
plt.xlabel("product")
plt.ylabel("sales")
plt.title("Product_Category_Distibution")
plt.show()
```



```
[74]: product_category.plot(kind='pie',autopct='%1.0f%%')
      plt.xlabel("product")
      plt.ylabel("sales")
      plt.legend(loc = 0, bbox_to_anchor=(1.2,1.1))
      plt.title("Product_Category_Distibution")
      plt.show()
```



Applications:

1. Real-time Analytics: Streaming data processing, instant insights
2. Cloud Computing: Scalable storage on-demand computing resources

Conclusion:

In this way we have successfully load and analyze sales data from different file formats, including CSV,

Excel, and JSON, and perform data cleaning, transformation, and analysis on the dataset

Outcome:

Upon completion of this experiment, students will be able to:

Experiment level outcome (ELO1): Apply data analysis and visualization techniques in the field of exploratory data science.

Questions:

1. What are the key considerations when choosing a file format for storing large datasets, and how does the choice of file format impact data loading and processing times?
2. When dealing with heterogeneous data sources, what strategies and tools can be used to efficiently load and integrate data into a centralized storage system, ensuring data consistency and reliability?
3. Explain the differences between structured and unstructured data storage methods. How do these differences affect data retrieval and analysis, and what are some common use cases for each approach?
4. In the context of data migration, what are the best practices for transferring data from one storage system to another? How can potential data loss or corruption be minimized during the migration process?

4.2 Experiment No. 2

Aim:

Aim: Interacting with Web APIs
Problem Statement: Analyzing Weather Data from OpenWeatherMap API
Dataset: Weather data retrieved from OpenWeatherMap API

Objective:

The goal is to interact with the OpenWeatherMap API to retrieve weather datafor a specific location and perform data modeling and visualization to analyze weather patterns over time.

Theory:

The availability of weather data through APIs such as OpenWeatherMap allows users to access real-time weather conditions, forecasts, and historical data for any location globally. Analyzing this data helps in weather prediction, decision-making for industries like agriculture, transport, and outdoor events, as well as gaining insights into climate patterns. This document presents the theoretical steps involved in retrieving and analyzing weather data from OpenWeatherMap API.

1. Understanding the OpenWeatherMap API: The OpenWeatherMap API is a web service that provides various weather-related data. It supports different types of data:

- **Current weather data:** Offers real-time weather details such as temperature, humidity, wind speed, and atmospheric pressure.
- **Forecast data:** Provides weather predictions for upcoming days or hours.
- **Historical data:** Contains archived weather information for a specific time and location.

The API uses HTTP requests and returns data in JSON, XML, or HTML formats. To access the data, an API key (obtained through registering on the OpenWeatherMap website) is required.

2. Fetching Weather Data: The process begins with sending an HTTP request to OpenWeatherMap's server, which retrieves the data in a JSON format. The API has various endpoints for different types of data:

- **Weather endpoint:** Fetches the current weather for a specific city or geographic coordinates.
- **Forecast endpoint:** Provides multi-day or hourly forecast data.
- **Historical data endpoint:** Delivers archived weather data.

Each request is composed of a base URL, endpoint, parameters (such as location or time), and the API key for authentication.

3. Key Weather Parameters: OpenWeatherMap provides a wide range of meteorological data, including:

- **Temperature:** The ambient temperature in degrees Celsius or Fahrenheit.
- **Humidity:** The percentage of moisture in the air.
- **Pressure:** The atmospheric pressure at sea level, measured in hPa (hectopascals).
- **Wind Speed and Direction:** Wind data includes speed (meters per second or kilometers per hour) and direction in degrees.
- **Weather Description:** A textual summary of current weather conditions (e.g., clear sky, rain).
- **Cloudiness, Visibility, Precipitation:** Additional parameters that provide information on cloud cover, visibility distance, and rain or snow data.

4. Data Handling and Parsing: The data returned from OpenWeatherMap is usually in JSON format, a lightweight, human-readable data interchange format. Parsing this JSON allows users to extract specific parameters for analysis.

5. Data Analysis: Once weather data is retrieved and parsed, several analytical techniques can be applied to gain insights:

- **Descriptive Analysis:** Summarizing the data to understand central tendencies (e.g., mean temperature, average humidity) and variability (e.g., highest and lowest temperatures).
- **Time Series Analysis:** If data is collected over time (such as hourly or daily forecast data), time series analysis can be applied to examine trends, seasonality, and fluctuations in weather patterns.
- **Correlation Analysis:** Exploring relationships between variables, such as how wind speed correlates with pressure, or how temperature affects humidity.
- **Predictive Modeling:** If historical weather data is available, machine learning techniques such as regression or time-series forecasting can predict future weather conditions.

6. Data Visualization: Visualization plays a crucial role in making weather data understandable and interpretable. Common techniques include:

- **Line Charts:** Used to plot continuous data like temperature or humidity over time.
- **Bar Charts:** Suitable for comparing categorical weather data (e.g., comparing wind speeds at different times).
- **Scatter Plots:** Helpful in exploring the relationship between two variables, such as temperature and pressure.
- **Heat Maps:** Useful for visualizing geographic distribution of weather parameters across different regions.

For example, plotting temperature and humidity data over a 7-day period can reveal trends such as gradual temperature rise or humidity fluctuations.

7. Applications of Weather Data Analysis: Analyzing weather data has a variety of practical applications

Agriculture: Farmers rely on weather forecasts to plan irrigation, planting, and harvesting.

- **Disaster Preparedness:** Real-time analysis of weather conditions is vital for early warning systems related to storms, floods, or other natural disasters.
- **Energy Sector:** Power companies use weather data to predict electricity demand and supply since temperature and wind influence energy consumption and generation.
- **Travel and Transportation:** Airlines, shipping, and ground transportation services use weather predictions for scheduling and risk mitigation.

INPUT

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

OUTPUT

Conclusion: Analyzing weather data using the OpenWeatherMap API involves fetching real-time or historical data, parsing it to extract relevant parameters, and applying analytical techniques to gain insights. The practical applications of this data range from weather forecasting and disaster management to energy consumption prediction and agricultural planning. By automating data collection and utilizing visualizations, users can make informed decisions based on weather conditions.

Outcome:

By accessing the current weather conditions, users can monitor key weather parameters such as temperature, humidity, wind speed, and atmospheric pressure. This real-time data allows individuals, organizations, and businesses to make informed decisions based on immediate weather conditions.

Questions:

1. What are the key weather parameters provided by the OpenWeatherMap API, and how can they be used in practical applications such as disaster preparedness or agriculture?

2. Explain the process of fetching weather data using the OpenWeatherMap API. What are the essential components of an API request, and how is the response structured?

3. How can historical weather data be used to perform predictive modeling, and what kind of machine learning techniques can be applied to forecast future weather conditions?

4. What is the importance of data visualization in weather analysis, and which types of visualizations are most suitable for representing weather trends, correlations, and patterns over time?

5. Describe the significance of time series analysis in weather data. How would you analyze temperature and humidity variations over a week using OpenWeatherMap API **data?**

4.3 Experiment No. 3

Aim:

Data Cleaning and Preparation

Objective:

The goal is to perform data cleaning and preparation to gain insights into the factors that contribute to customer churn

Theory:

Data cleaning and preparation is the process of preparing data for analysis. This includes identifying and removing errors, filling in missing values, and dealing with outliers. Data preparation can be a time-consuming process, but it is essential for ensuring that your data is accurate and ready for analysis.

Steps involved in data cleaning and preparation:

Step 1: Remove duplicate or irrelevant observations.

Duplicate observations can occur when data is collected from multiple sources or when data is entered manually. Irrelevant observations are data points that are not relevant to your analysis. For example, if you are analyzing data on customer purchases, you may want to remove observations for customers who have not made a purchase in the past year.

Step 2: Fix structural errors.

Structural errors are errors in the format of the data. For example, a date column may contain dates in different formats, or a customer name column may contain names in different formats. Structural errors can be corrected by using consistent formatting rules and by converting data to a consistent format.

Step 3: Filter unwanted outliers.

Outliers are data points that are significantly different from the rest of the data. Outliers can be caused by errors in data collection or entry, or they may be legitimate data points that are simply unusual. Identifying and filtering outliers before analysing your data is important, as they can skew your results.

Step 4: Handle missing data.

Missing data is a common problem in data sets. Missing data can occur when data is not collected properly, or when data is lost during data processing. There are a number of ways to handle missing data, such as deleting observations with missing values, imputing missing values, or creating a new variable to indicate whether a value is missing.

Step 5: Validate and QA.

Once you have cleaned and prepared your data, it is important to validate it to ensure that it is accurate and ready for analysis. This involves checking for any remaining errors, inconsistencies, or outliers. You may also want to perform a quality assurance (QA) check to ensure that your data meets your specific requirements.

Input

1. Use pandas. read_csv is used to read it into a DataFrame
2. Explore the dataset to understand its structure and content
3. isnull() is the function that is used to check missing values or null values in pandas python, isna() function is also used to get the count of missing values of column and row wise count of missing values.
4. Remove duplicate records from the dataset if any
5. Check for inconsistent data, such as inconsistent formatting or spelling variations, and standardize it. This will give the unique values and most common values for each column in the dataset.
6. Handle outliers in the data. Once you have identified outliers, you can handle them in a number of ways. You can: Delete the outliers: This is the simplest approach, but it can lead to a loss of data, impute the outliers: This involves replacing the outliers with more reasonable values, transform the data: This involves transforming the data in a way that reduces the impact of outliers.
7. Perform feature engineering, creating new features that may be relevant to predicting customer churn

Output:

Applications:

1. Data analysis: Involves cleaning, transforming, and modeling data to discover meaningful information. Data cleaning is a step in the data analysis process.
2. Machine learning: Data preparation and cleaning are part of the process of curating data for machine learning model development. This process can include labeling, human review, and data cleaning.
3. Business analytics: Data preparation is a pre-processing step that involves cleaning, transforming, and consolidating data.

Conclusion:

In this way we have explored the functions of the python library for Data Cleaning and Preparation this techniques and How to Handle missing values on Telecom_Customer_Churn data set.

Outcome:

Upon completion of this experiment, students will be able to:

Experiment level outcome (ELO1): Apply data cleaning and preparation in field of data analysis

Questions:

1. What percentage of the data is missing, and is it missing at random or in specific columns?
2. Is the data current or outdated?
3. Can you cross-reference the data with other sources to verify accuracy?
4. Are there records that need to be updated, corrected, or removed?

4.4 Experiment No. 4

Aim

Data Wrangling on Real Estate Market

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

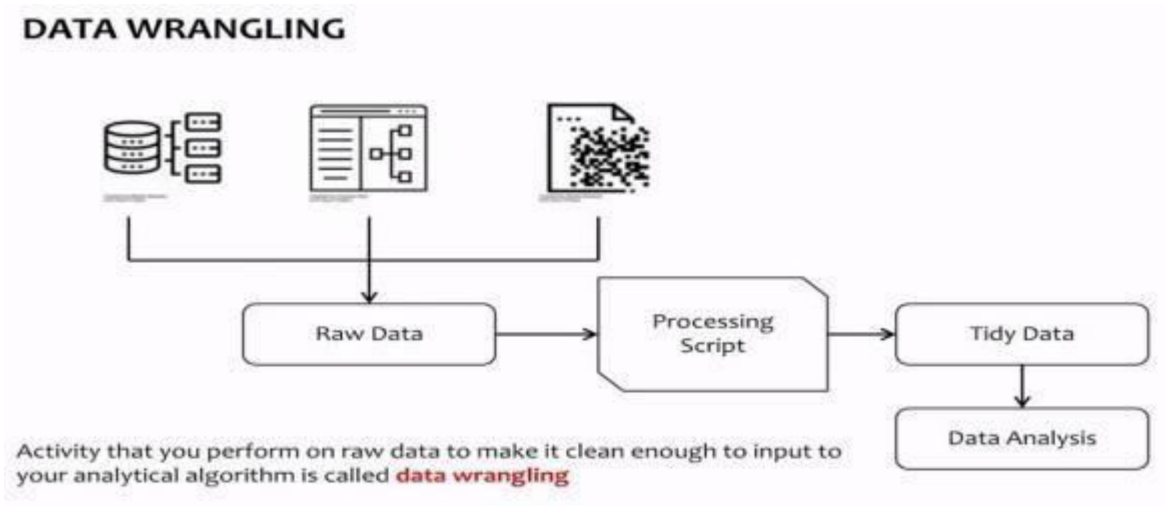
Dataset: "RealEstate_Prices.csv"

Objective

Students should be able to perform the data wrangling operation using Python on any open source dataset

Theory

Data wrangling: Data wrangling, also known as data mugging, is the process of cleaning, transforming, and organizing raw data for further analysis and integration.


DATA WRANGLING
Raw Data
Processing Script
Tidy Data
Data Analysis
Activity that you perform on raw data to make it clean enough to input to your analytical algorithm is called data wrangling

Data wrangling plays a critical role in machine learning. It refers to the process of cleaning, transforming, and preparing raw data for analysis, with the goal of ensuring that the data used in a machine learning model is accurate, consistent, and error-free.

1. Data Integration: Data integration involves combining data from multiple sources to define a unified dataset. This may involve merging data from different databases, cleaning and transforming data from different sources, and removing irrelevant data. The goal of data integration is to create a comprehensive dataset that can be used to train machine learning models.

2. Data visualization: Data visualization is the process of creating visual representations of the data. This may include scatter plots, histograms, and heat maps. The goal of data visualization is to provide insights into the data and identify patterns that can be used to improve machine learning models.

3. Data cleaning: Data cleaning is the process of identifying and correcting errors, inconsistencies, and inaccuracies in the data. This step includes removing duplicate values, filling in missing values, correcting spelling errors, and removing duplicate rows. The objective of data cleaning is to ensure that the data is accurate, complete, and consistent.

4. Data reduction: Data reduction is the process of reducing the amount of data used in a machine learning model. This may involve removing redundant data, removing irrelevant data, and sampling the data. The goal of data reduction is to reduce the computational requirements of the model and improve its accuracy.

5. Data transformation: Data transformation involves converting the data into a format that is more suitable for analysis. This may include converting categorical data into numerical data, normalizing the data, and scaling the data. The goal of data transformation is to make the data more accessible for machine learning algorithms and to improve the accuracy of the models.

Input

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.

Output

Applications

1. Weather Data: Weather data is often collected from multiple sources, making it challenging to combine and analyze. Data wrangling techniques can connect the data and convert it into a suitable format for analysis
2. Social media data is often unstructured and incomplete, making it challenging to analyze. Data wrangling techniques can clean and transform the data, allowing for more accurate analysis.

Conclusion

In this way we have explored the functions of the python library for Data Preprocessing, Data Wrangling Techniques and How to Handle missing values on RealEstate_Prices Dataset.

Outcome

Upon completion of this experiment, students will be able to:

Experiment level outcome (ELO1): Apply data wrangling in field of data analysis.

Questions

1. What is data wrangling, and why is it an essential step in the data preparation process for analytics and machine learning projects?
2. Can you explain the difference between data cleaning and data transformation in the context of data wrangling? What are some common techniques used for each?
3. When working with large and messy datasets, what are some best practices for identifying and handling missing data during the data wrangling process?
4. In the context of data wrangling, what are some typical challenges and considerations when dealing with categorical variables, and how can they be effectively converted into a format suitable for analysis or modeling?

4.5 Experiment no.5

Aim : Data Visualization using matplotlib

Problem Statement: Analyzing Air Quality Index (AQI) Trends in a City Dataset: "City_Air_Quality.csv"

Objective:
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.

Theory:

Introduction: Data visualization is a critical aspect of data analysis and interpretation. It involves representing data graphically to identify patterns, trends, and insights more efficiently than through raw numbers. Matplotlib is a popular Python library used for creating static, animated, and interactive visualizations. It offers various plotting functions that help users present their data in a visually appealing and comprehensible way.

1.Purpose of Data Visualization:
Data visualization serves multiple purposes, including:
• Summarization: Compressing large datasets into easy-to-understand visual representations.
• Pattern Recognition: Highlighting trends, correlations, and outliers that may not be obvious in raw data.
• Decision Making: Facilitating informed decisions based on visual interpretation of the data.
• Communication: Conveying complex data to stakeholders in a simple, effective way.
2. Matplotlib Overview:
Matplotlib is a flexible and customizable library that integrates well with Python. It can be used to create a wide variety of plots, including:
• Line plots
• Bar charts
• Scatter plots
• Histograms
• Pie charts
• Heatmaps
• 3D plots

It works well for both simple and complex visualizations and allows users to control every element of the plot, such as titles, labels, axes, colors, and legends.

3. Basic Structure of a Matplotlib Plot:

A Matplotlib plot is made up of the following components:

• Figure: The overall window or figure that can contain multiple subplots.

• Axes: The area where data is plotted. It consists of x and y axes, labels, ticks, and gridlines.

• Plot Elements: The actual graphical representation, such as lines, bars, markers, or patches.

• Legend: Describes different data series represented in the plot.

•Labels: Titles and axis labels help in defining the content being displayed.

In Matplotlib, plots are typically created using two main approaches: Pyplot and Object- Oriented Interface.

4. Pyplot Interface:

The pyplot module in Matplotlib is the simplest way to create plots. It provides a collection of functions that allow users to quickly generate and manipulate plots, similar to MATLAB's plotting system.

5. Object-Oriented Interface:

The Object-Oriented (OO) interface provides more control and flexibility over complex visualizations. It is useful for creating multi-plot layouts or customizing individual plot elements more granularly.

Applications of Data Visualization with Matplotlib:

Data visualization with Matplotlib is widely used across various fields:

• Scientific Research: To analyze experimental results, time series data, or biological patterns.

• Business Analytics: To present financial data, sales performance, and market trends.

• Machine Learning: To visualize model performance, decision boundaries, or data distributions.

• Public Communication: Charts and graphs are effective tools for conveying information to the general public, such as in news media.

INPUT:

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.

OUTPUT:

CONCLUSION:

Matplotlib is a powerful and flexible tool for data visualization in Python. It enables users to create a wide range of plots with minimal code while offering extensive customization for
complex visualizations. By using Matplotlib effectively, analysts can turn raw data into meaningful visual insights, facilitating better understanding and communication of complex data structures and patterns.

OUTCOME:

1. Understanding the Importance of Data Visualization
2. Knowledge of Matplotlib Basics and Structure
3. Familiarity with Pyplot and Object-Oriented Interfaces
4. Comprehensive Knowledge of Plot Types
5. Ability to Customize and Enhance Visualizations
6. Interactive Visualization Skills
7. Practical Applications Across Fields

QUESTIONS:

1.What are the key differences between the Pyplot interface and the Object-Oriented interface in Matplotlib, and when would you use one over the other?
2.How does customizing plot elements such as colors, labels, and gridlines in Matplotlib 3.improve the readability and interpretation of data visualizations?
4.What are some real-world applications of Matplotlib in different industries, and how can data visualization enhance decision-making in these fields?
5.How can interactive visualizations in Matplotlib, such as using sliders or dynamic plots, benefit data exploration and analysis, especially in a Jupyter notebook environment?

4.6 Experiment No. 6

Aim

Data Aggregation

Objective

The main goal is to analyze and visualize retail sales data by aggregating and comparing sales across regions and product categories.

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.

Theory

Data aggregation is the process of combining data from multiple sources or summarizing data from a single source to produce a more concise and meaningful representation of the data. It can be used to identify trends, patterns, and relationships in the data that would not be apparent if the data was analyzed individually

Data aggregation can be performed on a variety of data types, including numerical data, categorical data, and text data. Some common aggregation operations include:

- **Sum**: Calculates the sum of all values in a column or group of columns.
- **Mean**: Calculates the average of all values in a column or group of columns.
- **Median**: Calculates the middle value in a sorted list of values.
- **Mode**: Calculates the most frequently occurring value in a column or group of columns.
- **Count**: Counts the number of non-null values in a column or group of columns.

Examples of data aggregation:

- Identifying the top-performing sales regions: A retail company can aggregate sales data by region to identify the regions that are generating the most revenue.
- Tracking website traffic: A website owner can aggregate website traffic data by source to identify the most effective marketing channels.
- Data aggregation is a powerful tool that can be used to gain insights from data and make better decisions.

Benefits of data aggregation:

- **Improved efficiency**: Data aggregation can help businesses to improve their efficiency by automating tasks such as report generation and data analysis.
- **Increased accuracy**: Data aggregation can help businesses to increase the accuracy of their data by reducing the number of errors that occur when data is manually processed.
- **Enhanced decision-making**: Data aggregation can help businesses to make better decisions by providing them with a more complete and accurate view of their data.

Input

1. Import the "Retail_Sales_Data.csv" dataset.It is csv file, we can use pandas. The read_csv is used to read it into a DataFrame
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. Aggregating sales data involves summarizing and grouping information to derive meaningful insights.
4. Grouping sales data by region and shopping mall and calculating the total sales amount for each region serves several important purposes in business analysis and decision-making
5. Create bar plots or pie charts to visualize the sales distribution by region.
6.

Output

Applications

1. Business Intelligence and Analytics
2. Healthcare
3. Financial Services
4. E-commerce and Retail

Conclusion

In this way we have implemented data aggregation, able to analyze the sales performance of the retail company by region and identify the top-performing regions. We also identified the top- selling product categories.

Outcome

Upon completion of this experiment, students will be able to:

Experiment level outcome (ELO1): Explore the dataset to identify key variables. Sales data will be grouped by region and product category, with total sales calculated for each combination. Visualizations, including bar plots and pie charts, will display sales distribution by region and product category. Top-performing regions will be identified, and stacked or grouped bar plots will highlight sales comparisons across regions and categories.

EXPERIMENT NO.7

Aim: Analysis and Visualization of Stock Market Data Dataset:
"Stock_Prices.csv"

Objective:

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

Theory:

Analysis and visualization of stock market data involve examining historical and real-time data to understand price movements, trends, and patterns in stocks or other financial instruments. By leveraging financial data, analysts aim to gain insights into market behavior, forecast future movements, and make informed investment decisions.

1. Stock Market Data Analysis:

- **Technical Analysis:** Focuses on historical price charts and trading volumes to predict future movements using indicators like moving averages, relative strength index (RSI), and Bollinger Bands.
- **Fundamental Analysis:** Examines a company's financial health (e.g., earnings, assets, liabilities) to determine its intrinsic value and future performance.
- **Sentiment Analysis:** Analyzes market sentiment based on news, social media, or investor behavior to assess emotional responses that could affect stock prices.

2. Data Visualization Techniques:

- **Line Charts:** Display stock prices over time, making trends and fluctuations easily observable.
- **Candlestick Charts:** Show open, high, low, and close prices in a specific timeframe, useful for detailed price movement analysis.
- **Bar Charts:** Visualize volumes traded, helping to assess stock liquidity and market participation.
- **Moving Averages:** Averages stock prices over a period, smoothing out volatility to reveal trends.
- **Heatmaps:** Illustrate performance of multiple stocks in a portfolio, with color codes indicating the strength of performance (positive/negative).

3. Tools for Stock Market Analysis:

Popular tools include Python libraries such as Matplotlib, Seaborn, Plotly, and pandas for visualization, as well as APIs like Alpha Vantage, Yahoo Finance, and Quandl for retrieving stock data.

Input:

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.

Output:

Conclusion:

Analyzing and visualizing stock market data helps traders and investors track market performance, identify opportunities, and make data-driven financial decisions. These techniques are critical in financial planning and risk management.

Outcome:

1. Identification of Market Trends
2. Enhanced Decision-Making
3. Understanding Price Movements
4. Risk Management
5. Portfolio Performance Monitoring
6. Data-Driven Insights
7.

QUESTIONS:

1. How do candlestick charts help in understanding daily stock price movements?
2. What role do moving averages play in identifying market trends?
3. How can heatmaps be used to monitor the performance of multiple stocks in a portfolio?
4. What are the benefits of using data visualization for risk management in stock trading?