UNIT 8 – DATA VISUALIZATION USING PYTHON

8.1 Introduction to Data Visualization

Data visualization is the process of representing data in a graphical or visual format such as charts, graphs, or maps.
It helps to communicate insights from data clearly and efficiently by using visuals instead of raw numbers.

In Python, data visualization is performed using powerful libraries like Matplotlib, Seaborn, and Plotly which make it easy to create professional charts and plots.

Purpose of Data Visualization

1. To make complex data easier to understand.
2. To identify trends, patterns, and outliers in datasets.
3. To summarize large amounts of data quickly.
4. To communicate results clearly and effectively.
5. To assist in decision-making through visual insights.

Advantages of Data Visualization

- Converts raw data into meaningful visuals.
- Makes data comparison simple and quick.
- Highlights patterns and correlations.
- Helps in storytelling with data.
- Enhances the quality of analysis and reporting.

Applications of Data Visualization

1. **Business Intelligence:**
   Visual dashboards help analyze sales, profits, and performance.
2. **Scientific Research:**
   Used to visualize experimental results and trends.
3. **Machine Learning:**
   Helps in understanding the data before model training.
4. **Finance:**
   Used to display stock movements and trends.
5. **Healthcare:**
   To monitor patient records and disease spread patterns.

Python Libraries Used for Visualization

Library	Description

Matplotlib	Most widely used library for creating static, animated, and interactive plots.
Seaborn	Built on Matplotlib, used for creating attractive and informative statistical plots.
Plotly	Used for creating interactive, web-based visualizations.
Pandas	Has built-in plotting methods for quick data visualization.

Example

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [10, 12, 8, 15, 10]

plt.plot(x, y)
plt.title("Simple Line Graph")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.show()
```

Explanation:

- Import the Matplotlib library.
- Define data lists for the x-axis and y-axis.
- Use the plot function to create a line chart.
- Add title and labels for clarity.
- Display the chart using show().

Output Description

This program displays a simple line graph showing how Y values change with respect to X.
It helps visualize the relationship between two sets of data.

8.2 Data Visualization using Matplotlib

Matplotlib is one of the most commonly used libraries in Python for data visualization.
It provides various functions to create different types of charts such as line, bar, histogram, scatter, and pie charts.
It is mainly used for creating static, high-quality 2D plots.

8.2.1 Introduction to Matplotlib

Matplotlib is a Python library used for creating plots and charts that help visualize data.
It was developed by John D. Hunter and is built on top of NumPy.

It works well with other libraries like Pandas and SciPy.
Most plotting is done using the module called pyplot from Matplotlib.

Importing Matplotlib

```
import matplotlib.pyplot as plt
```

The pyplot module provides simple functions like plot(), bar(), hist(), scatter(), and show() for easy plotting.

Basic Syntax

```
plt.plot(x, y)
plt.title("Graph Title")
plt.xlabel("X-axis Label")
plt.ylabel("Y-axis Label")
plt.show()
```

The show() function is used to display the plot window.

8.2.2 Creating Different Types of Charts Using Matplotlib

1. Line Chart

A line chart is used to represent data over a continuous interval or time period.
It shows the trend of data clearly.

Example:

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [10, 12, 8, 15, 10]

plt.plot(x, y, color='blue', marker='o')
plt.title("Sales Trend")
plt.xlabel("Month")
plt.ylabel("Sales")
plt.show()
```

This program creates a simple line graph showing sales across different months.

2. Bar Chart

A bar chart is used to compare different categories of data using rectangular bars.
Bars can be vertical or horizontal.

Example:

```
import matplotlib.pyplot as plt

products = ['A', 'B', 'C', 'D']
sales = [30, 50, 20, 40]

plt.bar(products, sales, color='green')
plt.title("Product Sales")
plt.xlabel("Products")
plt.ylabel("Sales")
plt.show()
```

This program displays bars representing product-wise sales.

3. Histogram

A histogram displays the frequency distribution of continuous data.
It helps to understand the shape of the data distribution.

Example:

```
import matplotlib.pyplot as plt

data = [22, 25, 29, 30, 35, 36, 40, 42, 45, 50, 55, 58, 60, 65, 70]

plt.hist(data, bins=5, color='orange', edgecolor='black')
plt.title("Age Distribution")
plt.xlabel("Age")
plt.ylabel("Frequency")
plt.show()
```

This creates a histogram showing how values are distributed across intervals.

4. Scatter Plot

A scatter plot shows the relationship between two numeric variables using dots.
It is used to identify correlation or clustering in data.

Example:

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [5, 7, 6, 8, 7]

plt.scatter(x, y, color='red')
plt.title("Marks vs Study Hours")
plt.xlabel("Study Hours")
plt.ylabel("Marks")
plt.show()
```

This plot shows how marks change based on study hours.

5. Pie Chart

A pie chart represents data as slices of a circle, showing percentage contribution of each category.

Example:

```
import matplotlib.pyplot as plt

activities = ['Sleep', 'Work', 'Exercise', 'Leisure']
time = [8, 8, 2, 6]

plt.pie(time, labels=activities, autopct='%1.1f%%', startangle=90)
```

```
plt.title("Daily Routine Distribution")
plt.show()
```

This pie chart shows the percentage of time spent on daily activities.

6. Multiple Lines in One Chart

Matplotlib can display more than one line in a single chart for comparison.

Example:

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y1 = [10, 15, 20, 25, 30]
y2 = [8, 12, 18, 22, 28]

plt.plot(x, y1, label='Product A', color='blue')
plt.plot(x, y2, label='Product B', color='green')
plt.xlabel("Months")
plt.ylabel("Sales")
plt.title("Sales Comparison")
plt.legend()
plt.show()
```

This example compares sales of two products in the same chart using different colored lines.

8.2.3 Customizing Charts

Matplotlib allows customizing charts by changing colors, line styles, markers, and labels.

Common Parameters:

- color – changes the color of the line or bar.
- linestyle – changes the style of line ('dotted', 'dashed', etc.).
- marker – marks each data point with a symbol ('o', 's', '*').
- linewidth – sets the thickness of the line.
- label – defines a label for the data series.

Example of Customization

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [5, 10, 8, 12, 7]

plt.plot(x, y, color='purple', linestyle='--', marker='o', linewidth=2,
label='Data Line')
plt.xlabel("X Values")
plt.ylabel("Y Values")
plt.title("Customized Line Chart")
plt.legend()
plt.show()
```

This example shows how to modify appearance and add labels for better readability.

8.2.4 Adding Titles, Labels, and Legends

- title() adds a title to the chart.
- xlabel() and ylabel() set axis labels.
- legend() adds a legend for identifying data series.

Example:

```
plt.title("Population Growth")
plt.xlabel("Year")
plt.ylabel("Population")
plt.legend(["Growth Rate"])
```

8.2.5 Saving the Chart

Charts can be saved as image files using the savefig() function.

Example:

```
plt.savefig("chart.png")
```

This saves the chart as a PNG file in the working directory.

8.3 Data Visualization using Seaborn

Seaborn is a Python data visualization library built on top of Matplotlib.
It provides a high-level interface for creating attractive and informative statistical graphics.
Seaborn makes complex visualizations easy with fewer lines of code and better design by default.

8.3.1 Introduction to Seaborn

Seaborn is mainly used for exploring and understanding data through visuals.
It integrates well with Pandas DataFrames, which makes it convenient to plot data directly from datasets.

It comes with several built-in themes and color palettes that make charts look professional and modern without additional formatting.

Importing Seaborn

```
import seaborn as sns
import matplotlib.pyplot as plt
```

Seaborn works along with Matplotlib, so both are usually imported together.

Loading Sample Datasets

Seaborn has built-in sample datasets such as tips, iris, and flights which can be easily loaded for testing.

```
data = sns.load_dataset("tips")
print(data.head())
```

This loads the tips dataset containing restaurant bill information.

8.3.2 Types of Plots in Seaborn

Seaborn provides different plot types for statistical data visualization.
Below are some commonly used plots.

1. Line Plot

Used to display the relationship between two continuous variables.

```
import seaborn as sns
import matplotlib.pyplot as plt

data = sns.load_dataset("flights")
sns.lineplot(x="year", y="passengers", data=data)
plt.title("Yearly Airline Passengers")
plt.show()
```

This creates a line graph showing the increase in the number of airline passengers over years.

2. Bar Plot

Used to display average values of categories.
It automatically calculates and shows mean values by default.

```
import seaborn as sns
import matplotlib.pyplot as plt

data = sns.load_dataset("tips")
sns.barplot(x="day", y="total_bill", data=data, palette="Blues")
plt.title("Average Total Bill per Day")
plt.show()
```

This chart shows the average restaurant bill amount for each day.

3. Count Plot

Used to show the frequency of categorical data.
It counts the occurrences of each category automatically.

```
sns.countplot(x="day", data=data, palette="Set2")
plt.title("Number of Customers by Day")
plt.show()
```

This displays how many customers visited on each day.

4. Histogram (Distplot / Histplot)

Used to display the distribution of numerical data.
Histograms help identify how data is spread or concentrated.

```
sns.histplot(data["total_bill"], kde=True, color="orange")
plt.title("Distribution of Total Bill")
plt.show()
```

Here, kde=True adds a smooth curve representing the probability density.

5. Box Plot

Used to show the spread of data and detect outliers.
It displays the median, quartiles, and extreme values.

```
sns.boxplot(x="day", y="total_bill", data=data, palette="coolwarm")
plt.title("Box Plot of Total Bill by Day")
plt.show()
```

This shows how the total bill varies on different days and highlights extreme values.

6. Violin Plot

Combines box plot and kernel density plot to show data distribution in more detail.

```
sns.violinplot(x="day", y="total_bill", data=data, palette="muted")
plt.title("Violin Plot of Total Bill by Day")
plt.show()
```

It gives both the range and the probability distribution of data.

7. Scatter Plot

Used to examine the relationship between two numeric variables.

```
sns.scatterplot(x="total_bill", y="tip", data=data, color="green")
plt.title("Relationship between Total Bill and Tip")
plt.show()
```

This shows how the tip amount changes with respect to the total bill.

8. Pair Plot

Displays multiple relationships in a dataset using multiple scatter plots.
It is very useful for multivariate analysis.

```
sns.pairplot(data)
plt.show()
```

This generates pairwise plots for all numeric columns in the dataset.

9. Heatmap

Used to show correlations between numeric variables in a matrix format.
Each cell represents the strength of the relationship between two variables.

```
corr = data.corr()
sns.heatmap(corr, annot=True, cmap="coolwarm")
plt.title("Correlation Heatmap")
plt.show()
```

This helps identify strong or weak relationships between dataset variables.

8.3.3 Customization in Seaborn

Seaborn provides several customization options to improve the readability of plots.

- Changing color palette using palette parameter.
- Adding titles using plt.title().
- Adjusting figure size using plt.figure(figsize=(width, height)).
- Adding labels using plt.xlabel() and plt.ylabel().

Example:

```
plt.figure(figsize=(7,4))
sns.barplot(x="sex", y="tip", data=data, palette="pastel")
plt.title("Average Tip by Gender")
plt.xlabel("Gender")
plt.ylabel("Tip Amount")
plt.show()
```

8.3.4 Advantages of Seaborn

1. Provides high-level functions for easy visualization.
2. Beautiful default themes and color palettes.
3. Integrates smoothly with Pandas DataFrames.
4. Supports advanced statistical visualizations.
5. Saves time with less code and better design.

8.4 Data Visualization using Pandas

Pandas is one of the most popular Python libraries for data analysis and manipulation.
In addition to data handling, Pandas also provides built-in methods for basic data visualization.
It uses the Matplotlib library in the background to generate charts directly from a DataFrame or Series.

8.4.1 Introduction

Pandas allows creating simple and quick visualizations without explicitly importing Matplotlib.
This feature helps data analysts visualize data trends directly while working with datasets.

The plot() function in Pandas is used to create different types of charts.

Syntax

```
DataFrame.plot(kind='chart_type')
```

The parameter kind specifies the type of chart to draw.
Some commonly used types include:

- line
- bar
- barh
- histogram
- pie
- scatter

Example

```
import pandas as pd
import matplotlib.pyplot as plt

data = {'Year': [2018, 2019, 2020, 2021, 2022],
        'Sales': [200, 250, 300, 280, 350]}

df = pd.DataFrame(data)
df.plot(x='Year', y='Sales', kind='line', color='blue', marker='o')
plt.title("Year-wise Sales Trend")
plt.xlabel("Year")
plt.ylabel("Sales")
plt.show()
```

This code generates a line chart showing sales growth over years using the Pandas plot() function.

8.4.2 Types of Charts in Pandas

1. Line Chart

A line chart is the default type of chart in Pandas.
It is used to show trends or changes over time.

```
df.plot(x='Year', y='Sales', kind='line')
plt.title("Line Chart Example")
plt.show()
```

2. Bar Chart

A bar chart displays values for different categories using vertical bars.

```
df.plot(x='Year', y='Sales', kind='bar', color='green')
plt.title("Bar Chart Example")
plt.show()
```

This chart helps compare data values among categories.

3. Horizontal Bar Chart

Used when category names are long or there are many items to display.

```
df.plot(x='Year', y='Sales', kind='barh', color='orange')
plt.title("Horizontal Bar Chart Example")
plt.show()
```

4. Histogram

A histogram shows the frequency distribution of continuous data.

```
import numpy as np

data = {'Marks': [45, 56, 67, 78, 88, 92, 55, 67, 79, 85]}
df = pd.DataFrame(data)
df['Marks'].plot(kind='hist', bins=5, color='purple', edgecolor='black')
plt.title("Marks Distribution")
plt.xlabel("Marks")
plt.ylabel("Frequency")
plt.show()
```

This histogram displays how students' marks are distributed across intervals.

5. Pie Chart

A pie chart shows the percentage contribution of each category to the total.

```
data = {'Activity': ['Work', 'Sleep', 'Exercise', 'Leisure'],
        'Hours': [8, 8, 2, 6]}
df = pd.DataFrame(data)
df.plot(kind='pie', y='Hours', labels=df['Activity'], autopct='%1.1f%%')
plt.title("Daily Time Allocation")
plt.ylabel("")
plt.show()
```

This pie chart shows how daily hours are distributed among activities.

6. Scatter Plot

Used to show relationships between two numerical variables.

```
data = {'Hours_Studied': [2, 4, 6, 8, 10],
        'Marks': [30, 45, 60, 75, 85]}
df = pd.DataFrame(data)
df.plot(kind='scatter', x='Hours_Studied', y='Marks', color='red')
plt.title("Study Hours vs Marks")
plt.show()
```

This chart shows how marks increase with study hours.

8.4.3 Customizing Charts

Pandas charts can be customized using additional parameters in the plot() function and Matplotlib functions.

Common Parameters

- color – sets the color of the chart
- title – adds a chart title
- grid – adds gridlines
- legend – displays the legend
- figsize – changes chart size

Example

```
df.plot(x='Year', y='Sales', kind='line', color='brown', marker='s',
grid=True, figsize=(6,4))
plt.title("Customized Sales Chart")
plt.show()
```

This creates a well-formatted chart with customized color, gridlines, and figure size.

8.4.4 Using Multiple Columns in a Chart

Multiple columns can be plotted in the same chart to compare data values easily.

```
data = {'Year': [2019, 2020, 2021, 2022],
        'Product_A': [200, 220, 250, 270],
        'Product_B': [150, 180, 230, 260]}
```

```
df = pd.DataFrame(data)
df.plot(x='Year', kind='line', marker='o')
plt.title("Sales Comparison of Two Products")
plt.show()
```

This line chart displays both product sales trends in the same plot.

8.4.5 Advantages of Pandas Visualization

1. Quick and simple way to visualize data directly from a DataFrame.
2. No need to import Matplotlib separately for basic plots.
3. Supports all common chart types.
4. Works well with large datasets.
5. Allows combining Pandas operations with visual analysis.

8.5 Data Visualization using Plotly

Plotly is a powerful Python library used to create interactive and web-based data visualizations.
Unlike Matplotlib and Seaborn, which produce static charts, Plotly generates interactive charts that allow zooming, hovering, and real-time data exploration.
It is widely used for dashboards, data analysis, and business intelligence applications.

8.5.1 Introduction to Plotly

Plotly supports a wide variety of charts such as line, bar, pie, scatter, histogram, and even 3D plots.
It works both in Jupyter Notebooks and normal Python environments.
The charts can be exported as HTML files, making them easy to share or embed in web pages.

Plotly has two main interfaces:

- plotly.graph_objects
- plotly.express

Plotly Express is simpler and preferred for quick visualization.

Installing Plotly

Before using Plotly, it needs to be installed using the command:

```
pip install plotly
```

Importing Plotly Express

```
import plotly.express as px
```

8.5.2 Line Chart

A line chart is used to show data changes over time or continuous data relationships.

```
import plotly.express as px

data = {'Year': [2018, 2019, 2020, 2021, 2022],
        'Sales': [100, 150, 200, 180, 230]}

fig = px.line(data, x='Year', y='Sales', title='Yearly Sales Growth')
fig.show()
```

This creates an interactive line chart where you can hover over points to view exact values.

8.5.3 Bar Chart

A bar chart displays categorical data using rectangular bars for easy comparison.

```
import plotly.express as px

data = {'Product': ['A', 'B', 'C', 'D'],
        'Sales': [250, 300, 180, 220]}

fig = px.bar(data, x='Product', y='Sales', color='Product', title='Product-
wise Sales')
fig.show()
```

Each bar represents the sales of a product with distinct colors for better visibility.

8.5.4 Scatter Plot

A scatter plot is used to show relationships between two numerical variables.

```
import plotly.express as px

data = {'Hours_Studied': [1, 2, 3, 4, 5, 6],
        'Marks': [40, 45, 50, 55, 65, 75]}

fig = px.scatter(data, x='Hours_Studied', y='Marks', title='Study Hours vs
Marks', color='Marks')
fig.show()
```

The hover feature shows details of each data point interactively.

8.5.5 Pie Chart

A pie chart displays proportional data as slices of a circle.

```
import plotly.express as px

data = {'Activity': ['Sleep', 'Work', 'Exercise', 'Leisure'],
        'Hours': [8, 9, 2, 5]}
```

```
fig = px.pie(data, values='Hours', names='Activity', title='Daily Time
Distribution', hole=0)
fig.show()
```

This pie chart shows how time is divided across different daily activities.

8.5.6 Histogram

A histogram represents the frequency distribution of continuous data.

```
import plotly.express as px

data = {'Age': [20, 22, 25, 26, 30, 32, 35, 38, 40, 42, 45, 50]}
fig = px.histogram(data, x='Age', nbins=5, title='Age Distribution')
fig.show()
```

Each bar represents the number of observations within a specific age range.

8.5.7 Box Plot

A box plot is used to display the spread and outliers of a dataset.

```
import plotly.express as px

data = {'Category': ['A', 'A', 'B', 'B', 'C', 'C'],
        'Value': [10, 12, 15, 18, 20, 22]}

fig = px.box(data, x='Category', y='Value', title='Box Plot Example')
fig.show()
```

This chart helps detect variation and outliers within each category.

8.5.8 Bubble Chart

A bubble chart is similar to a scatter plot but adds a third variable represented by the size of bubbles.

```
import plotly.express as px

data = {'City': ['Delhi', 'Mumbai', 'Pune', 'Kolkata', 'Chennai'],
        'Population': [18, 12, 5, 8, 6],
        'GDP': [400, 350, 200, 250, 300]}

fig = px.scatter(data, x='GDP', y='Population', size='GDP', color='City',
title='City Population vs GDP')
fig.show()
```

Each bubble's size indicates GDP, helping visualize multiple parameters together.

8.5.9 3D Scatter Plot

Plotly supports 3D plots, which provide deeper visual insights into data relationships.

```
import plotly.express as px

data = {'X': [1, 2, 3, 4, 5],
        'Y': [10, 15, 20, 25, 30],
        'Z': [5, 10, 15, 20, 25]}

fig = px.scatter_3d(data, x='X', y='Y', z='Z', title='3D Scatter Plot')
fig.show()
```

This creates an interactive 3D plot where users can rotate and explore the data.

8.5.10 Advantages of Plotly

1. Produces interactive and web-based visualizations.
2. Easy to integrate with Jupyter Notebook and web applications.
3. Supports a wide variety of charts, including 3D plots.
4. Allows exporting charts to HTML and images.
5. Provides hover information and zooming features for better insights.