Task 1

Visualization Library Documentation: Matplotlib and Seaborn

Library Overview

1. Matplotlib

Matplotlib is one of the most powerful and widely used data visualization libraries in Python. Originally developed by John D. Hunter, it enables users to create a wide range of static, animated, and interactive plots. Matplotlib serves as the foundation for many other plotting libraries such as Seaborn, pandas visualization, and Plotnine, making it a core tool in the Python data science ecosystem. The library is highly customizable and supports full control over plot elements including figure size, colors, fonts, axes, ticks, and legends. It integrates seamlessly with NumPy and pandas, and supports multiple backends for exporting to formats such as PNG, PDF, SVG, and JPG, which is particularly useful for publishing scientific and academic reports.

Matplotlib can generate a wide variety of plots including line charts, bar charts, histograms, scatter plots, pie charts, error bars, stack plots, and even 3D visualizations via mpl_toolkits.mplot3d. It is well-suited for exploratory data analysis (EDA), real-time visual monitoring, and creating dashboards when used with frameworks like Tkinter or Jupyter. Despite its learning curve—especially when compared to high-level libraries like Seaborn—Matplotlib remains the go-to choice for precision plotting. For example, a line plot using plt.plot() can illustrate stock prices over time, a plt.bar() chart can compare sales across regions, and plt.hist() can visualize distribution patterns in datasets. Moreover, Matplotlib allows subplot arrangement (plt.subplot()) for multi-panel visualizations and supports LaTeX rendering in plot text, making it a favourite for researchers. Its vast documentation and active community support further enhance its reliability and usability in both academic and industrial projects.

2. Seaborn

Seaborn is a high-level Python data visualization library built on top of Matplotlib. It is specifically designed to make statistical graphics more attractive and easier to create.

Developed by Michael Waskom, Seaborn provides a clean and modern interface for drawing informative and aesthetically pleasing plots with minimal lines of code. It integrates seamlessly with pandas DataFrames, allowing users to pass entire datasets directly into plotting functions. Seaborn is especially well-suited for exploring relationships between multiple variables and includes powerful built-in support for grouping, aggregation, and statistical estimation.

The library supports a variety of visualization types including line plots, bar plots, scatter plots, box plots, violin plots, histograms, KDE plots, heatmaps, and pair plots. These charts often come with added statistical context—for instance, bar plots in Seaborn can display confidence intervals by default, and scatter plots can include regression lines for trend analysis using sns.regplot() or sns.lmplot(). Seaborn also offers a variety of built-in themes and color palettes that enhance visual appeal and readability, which makes it a favorite for dashboards, academic presentations, and quick exploratory data analysis.

What sets Seaborn apart from Matplotlib is its simplicity and declarative syntax. For example, to create a boxplot showing the distribution of total bills across days in a restaurant dataset, one only needs to use sns.boxplot(x='day', y='total_bill', data=tips). Furthermore, Seaborn enables multi-variable comparisons through functions like sns.pairplot() and sns.heatmap(), which can uncover hidden patterns and correlations in data with minimal effort. While Seaborn abstracts many Matplotlib functions under the hood, it still allows for Matplotlib-level customization when needed. Overall, Seaborn strikes a balance between functionality, simplicity, and visual polish, making it an essential tool for any data analyst or scientist working in Python.

Graph Types with Examples

1. Matplotlib

a. Line Plot

A line plot displays information as a series of data points connected by straight line segments, making it ideal for illustrating trends over intervals or continuous data. The x and y axes represent variables, with data points joined to show progression or changes clearly. Line plots can include multiple lines for comparing different data sets on the same graph and offer extensive customization options such as colours, markers, line styles, labels, and grids. This

versatility makes line plots especially useful for visualizing time series data, comparing variable trends, and monitoring patterns over time.

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4]
y = [10, 20, 25, 30]

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

b. Bar Chart

A bar chart is a commonly used visualization in Matplotlib to represent categorical data with rectangular bars. Each bar's height (or length, in the case of horizontal bars) corresponds to the value of the variable it represents. Bar charts are useful for comparing quantities across different categories, making it easy to see which groups have higher or lower values. They can be plotted vertically using plt.bar() or horizontally using plt.barh(). Matplotlib allows for extensive customization of bar charts, including color, width, alignment, labels, and adding value annotations. Bar charts are ideal for visualizing survey data, sales comparisons, or any situation where you need to show differences between discrete groups.

```
categories = ['A', 'B', 'C']
values = [10, 24, 36]

plt.bar(categories, values)
plt.title("Bar Chart")
plt.show()
```

c. Histogram

A histogram is a type of plot in Matplotlib used to represent the distribution of numerical data. Unlike a bar chart, which displays data for distinct categories, a histogram groups continuous data into bins (intervals) and shows how many data points fall into each bin. The x-axis represents the intervals (or bins), while the y-axis shows the frequency or count of data points within each bin. Histograms are especially useful for understanding the shape, spread, and central tendency of a dataset, such as whether the data is normally distributed, skewed, or contains outliers. Using plt.hist() in Matplotlib, users can customize the number of bins, color, edge style, and whether to normalize the data. This makes histograms essential for exploratory data analysis and statistical insight.

```
import numpy as np
data = np.random.randn(1000)

plt.hist(data, bins=30)
plt.title("Histogram")
plt.show()
```

d. Pie Chart

A pie chart is a circular statistical graphic used in Matplotlib to represent data as slices of a pie, where each slice shows the proportion of a category relative to the whole. It is most effective for displaying percentage or proportional data and comparing parts of a whole. Each slice's size corresponds to the value it represents, and all slices together sum up to 100%. Matplotlib provides the plt.pie() function to create pie charts, with customization options like labels, colors, explode (to separate a slice), and autopct (to display percentage values). Although visually appealing, pie charts are best used when there are a limited number of categories, as too many slices can make interpretation difficult. They are ideal for showing market shares, budget distributions, or demographic compositions.

```
labels = ['A', 'B', 'C']
sizes = [30, 50, 20]

plt.pie(sizes, labels=labels, autopct='%1.1f%%')
plt.title("Pie Chart")
plt.show()
```

e. Scatter Plot

A scatter plot is a type of data visualization in Matplotlib used to display the relationship between two continuous variables. Each point on the plot represents an individual data observation, with its position determined by values on the x and y axes. Scatter plots are especially useful for identifying patterns, correlations, or trends in data, such as linear relationships, clusters, or outliers. The plt.scatter() function in Matplotlib allows for the creation of scatter plots and offers customization options like color, size, shape, and transparency of the points. They are commonly used in statistical and scientific analyses to explore how two variables interact, making them essential for regression analysis, clustering, and exploratory data analysis.

```
x = [1, 2, 3, 4]
y = [10, 15, 13, 17]

plt.scatter(x, y)
plt.title("Scatter Plot")
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```

2. Seaborn

a. Line Plot

A line plot is a powerful and visually appealing way to display trends in data over a continuous variable, such as time or measurement intervals. Built on top of Matplotlib, Seaborn provides the `sns.lineplot()` function, which automatically handles statistical estimation and confidence intervals. It is commonly used to show relationships between two numeric variables, where the x-axis typically represents the independent variable and the y-axis represents the dependent variable. Seaborn's line plot stands out because it can easily incorporate grouping through the `hue`, `style`, and `size` parameters, allowing for the comparison of multiple categories in a single chart. It also smooths noisy data using built-in estimators like mean or median. This makes it ideal for analysing trends, comparing groups, and visualizing time series data with clarity and elegance.

```
import seaborn as sns
import pandas as pd

df = pd.DataFrame({
  'x': [1, 2, 3, 4],
 'y': [10, 20, 25, 30]
})

sns.lineplot(data=df, x='x', y='y')
```

b. Bar Plot

A Seaborn barplot is a versatile and informative chart used to display the average (or other estimator) of a numerical variable across different categories. Unlike a simple bar chart that shows raw values, `sns.barplot()` in Seaborn aggregates data and plots the central tendency—usually the mean—for each category along with confidence intervals by default. The x-axis represents categorical variables, while the y-axis shows the aggregated numeric values. Barplots are useful for comparing group-wise statistics, such as average scores, sales,

or survey results. Seaborn also allows customization with parameters like `hue` to compare multiple groups, `ci` to adjust confidence intervals, and `estimator` to change the aggregation function (e.g., median instead of mean). This makes barplots ideal for highlighting comparisons and drawing statistical insights from grouped data.

```
df = pd.DataFrame({
  'category': ['A', 'B', 'C'],
   'value': [10, 20, 15]
})

sns.barplot(x='category', y='value', data=df)
```

c. Histogram / Distplot

A Seaborn distplot was a popular function used to visualize the distribution of a dataset by combining a histogram with a kernel density estimate (KDE) curve. It allowed users to see both the frequency of data points within intervals (via the histogram) and the estimated probability density function (via the KDE curve), making it useful for understanding the shape and spread of continuous data.

However, the sns.distplot() function is deprecated in recent versions of Seaborn (from v0.11.0 onwards). It has been replaced by two separate functions:

- sns.histplot() – for plotting histograms (with or without KDE)
- sns.kdeplot() – for plotting KDE curves

```
data = np.random.randn(1000)

sns.histplot(data, kde=True)
```

c. Boxplot

A Seaborn boxplot is a statistical chart used to display the distribution, spread, and outliers of a dataset across different categories. It provides a concise summary of a dataset's minimum**,** first quartile (Q1)**,** median (Q2)**,** third quartile (Q3)**,** and maximum values. These five-number summaries are visualized in a rectangular box with a line at the median and "whiskers" extending to the rest of the data, excluding outliers which are shown as individual points.

The `sns.boxplot()` function in Seaborn makes it easy to create boxplots and supports grouping by categorical variables using the `x`, `y`, and `hue` parameters. This allows for comparisons between different groups or subgroups. Boxplots are particularly useful for detecting skewness, variability, and the presence of outliers in the data.

Common applications include comparing exam scores across different classes, analyzing income across regions, or visualizing any grouped numerical data in a compact, statistically informative way.

```
df = sns.load_dataset("tips")

sns.boxplot(x="day", y="total_bill", data=df)
```

e. Heatmap

A Seaborn heatmap is a powerful visualization tool used to represent data in a matrix format where individual values are displayed as color-coded cells. It is particularly useful for visualizing correlation matrices, confusion matrices, or any 2D data structure, allowing patterns, trends, and relationships between variables to be quickly identified.

The sns.heatmap() function in Seaborn offers various customization options, such as adding annotations (numerical values inside the cells), color maps to adjust the color gradient, and line colors to separate cells. It supports options for formatting numbers, controlling axis ticks, and masking parts of the matrix if needed.

Heatmaps are commonly used in data analysis to:

- Understand relationships between variables in a dataset (e.g., using correlation matrices),
- Analyze performance in classification tasks (e.g., confusion matrices),
- Highlight areas of high or low intensity in spatial data.

Their ability to convert complex numerical data into an intuitive visual format makes heatmaps an essential part of exploratory data analysis.

```
corr = df.corr()

sns.heatmap(corr, annot=True, cmap='coolwarm')
```