Batch: C4-1 Roll No.: 16010124216

Experiment / assignment / tutorial No

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

**TITLE: To demonstrate Matplot library in python by studying its key features and its functions**

**AIM: Write a program to demonstrate matplotlib visualisation functions using datasets**

**OUTCOME: Student will be able to:**

**CO1: Formulate problem statement and develop the logic (algorithm/flowchart) for its solution.**

**CO5: Illustrate the use of Python packages.**

**AIM:** Write a program to explore the Matplotlib library

# Resource Needed: Python IDE



**Theory:**

# What is Matplotlib?

1. Matplotlib

Matplotlib is a data visualization library and 2-D plotting library of Python It was initially released in 2003 and it is the most popular and widely-used plotting library in the Python community. It comes with an interactive environment across multiple platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, etc. It can be

used to embed plots into applications using various GUI toolkits like Tkinter, GTK+, wxPython, Qt, etc. So you can use Matplotlib to create plots, bar charts, pie charts, histograms, scatterplots, error charts, power spectra, stemplots, and whatever other visualization charts you want! The Pyplot module also provides a MATLAB-like interface that is just as versatile and useful as MATLAB while being free and open source.

1. Plotly

Plotly is a free open-source graphing library that can be used to form data visualizations. Plotly (plotly.py) is built on top of the Plotly JavaScript library (plotly.js) and can be used to create web-based data visualizations that can be displayed in Jupyter notebooks or web applications using Dash or saved as individual HTML files. Plotly provides more than 40 unique chart types like scatter plots, histograms, line charts, bar charts, pie charts, error bars, box plots, multiple axes, sparklines, dendrograms, 3-D charts, etc. Plotly also provides contour plots, which are not that common in other data visualization libraries. In addition to all this, Plotly can be used offline with no internet connection

# Plotting x and y points

The plot() function is used to draw points (markers) in a diagram. By default, the plot() function draws a line from point to point.

**Syntax:**

matplotlib.pyplot.plot(\\*args, scalex=True, scaley=True, data=None, \\*\\*kwargs)

* + x, y: These parameters are the horizontal and vertical coordinates of the data points. x values are optional.
  + fmt: This parameter is optional and it contains the string value.
  + data: This parameter is optional and it is an object with labeled data.

This returns the following:

**lines:** This returns the list of Line2D objects representing the plotted data. Example:-

**Outpu**

**t**

**Draw a line in a diagram from position (1, 3) to position (8,**

**10):**

|  |  |
| --- | --- |
| import matplotlib.pyplot as plt import numpy as np  xpoints = np.array([1, 8]) ypoints = np.array([3, 10])  plt.plot(xpoints, ypoints) plt.show() |  |

# Multiple Points

You can plot as many points as you like, just make sure you have the same number of points in both axis.

|  |  |
| --- | --- |
| import matplotlib.pyplot as plt import numpy as np  xpoints = np.array([1, 2, 6, 8])  ypoints = np.array([3, 8, 1, 10])  plt.plot(xpoints, ypoints) plt.show() |  |

# Matplotlib Line

**Linestyle:--- You can use the keyword argument linestyle, or shorter ls, to change the style of the plotted line:**

Following are the linestyles available in ***matplotlib*: Using *linestyle* Argument:**

* Solid
* Dashed
* Dotted
* Dashdot
* None

|  |  |  |
| --- | --- | --- |
| **Syntax:** plt.plot(xdata, ydata, linestyle='dotted') | |  |
| **Program** | **Output:** | |
| **Use a dotted line:**  import matplotlib.pyplot as plt import numpy as np  ypoints = np.array([3, 8, 1, 10]) plt.plot(ypoints, linestyle = 'dotted') plt.show() |  | |

# Matplotlib Labels and Title a.Create Labels for a Plot

With Pyplot, you can use the xlabel() and ylabel() functions to set a label for the x- and y-axis.

The label () function in the pyplot module of the matplotlib library sets the label for the x-axis.

**Syntax:** matplotlib.pyplot.xlabel(xlabel, fontdict=None, labelpad=None, \*\*kwargs)

# b. Create a Title for a Plot

With Pyplot, you can use the title() function to set a title for the plot.

|  |  |
| --- | --- |
| Program:  import matplotlib.pyplot as plt  import numpy as np  x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])  y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])  plt.plot(x, y)  plt.title("Sports Watch Data")  plt.xlabel("Average Pulse")  plt.ylabel("Calorie Burnage")  plt.show() | Output: |

# Matplotlib Scatter Creating Scatter Plots

With Pyplot, you can use the scatter() function to draw a scatter plot.

The scatter() function plots one dot for each observation. It needs two arrays of the same length, one for the values of the x-axis, and one for values on the y-axis:

|  |  |
| --- | --- |
| Program:  import matplotlib.pyplot as plt  import numpy as np  x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])  y=np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])  plt.scatter(x, y)  plt.show() | Output: |

# Add Grid Lines to a Plot

With Pyplot, you can use the grid() function to add grid lines to the plot.

|  |  |
| --- | --- |
| Program:  import matplotlib.pyplot as plt  import numpy as np  x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])  y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])  plt.title("Sports Watch Data")  plt.xlabel("Average Pulse")  plt.ylabel("Calorie Burnage")  plt.plot(x, y)  plt.grid()  plt.show() | Output: |

# Display Multiple Plots

With the subplot() function you can draw multiple plots in one figure.

subplot(nrows, ncols, index, \*\*kwargs)

**The layout is organized in rows and columns, which are represented by the *first* and *second* argument.**

**The third argument represents the index of the current plot.**

|  |  |
| --- | --- |
| **Program:-**  import matplotlib.pyplot as plt import numpy as np  #plot 1:  x = np.array([0, 1, 2, 3])  y = np.array([3, 8, 1, 10])  plt.subplot(1, 2, 1) plt.plot(x,y)  #plot 2:  x = np.array([0, 1, 2, 3])  y = np.array([10, 20, 30, 40])  plt.subplot(1, 2, 2) plt.plot(x,y)  plt.show() | **Output:** |
| import matplotlib.pyplot as plt import numpy as np  #plot 1:  x = np.array([0, 1, 2, 3])  y = np.array([3, 8, 1, 10])  plt.subplot(2, 1, 1) plt.plot(x,y)  #plot 2:  x = np.array([0, 1, 2, 3])  y = np.array([10, 20, 30, 40])  plt.subplot(2, 1, 2) plt.plot(x,y)  plt.show() | **Output:--** |

# Creating Bars

With Pyplot, you can use the bar() function to draw bar graphs.

|  |  |
| --- | --- |
| import matplotlib.pyplot as plt import numpy as np  x = np.array(["A", "B", "C", "D"])  y = np.array([3, 8, 1, 10])  plt.bar(x,y) plt.show() | **Output:--** |
| import matplotlib.pyplot as plt import numpy as np  x = np.array(["A", "B", "C", "D"])  y = np.array([3, 8, 1, 10])  plt.bar(x, y, color = "#4CAF50") plt.show() |  |

# Creating Pie Chart with Labels:

|  |  |
| --- | --- |
| import matplotlib.pyplot as plt import numpy as np  y = np.array([35, 25, 25, 15])  mylabels =  ["Apples", "Bananas", "Cherries", "Dates"  ]  plt.pie(y, labels = mylabels) plt.show() | **Output:** |

**Problem Definition**:

# Note:-- All plots should be labeled on the X-axis and Y-axis with a Grid for each program.

1. Write a Python program to draw a line using given axis values with suitable labels in the x-axis, y-axis, and title.
2. a)Write Python programming to display a bar chart of the popularity of programming Languages. Also, draw a Pie chart for **popularity** Data values.

# Sample data:

**Programming languages: Java, Python, PHP, JavaScript, C#, C++**

# Popularity: 22.2, 17.6, 8.8, 8, 7.7, 6.7

b) Write a Python program to display a horizontal bar chart showing the popularity of programming languages. **Hint: use the barh() function.**

1. Prepare a dataset using list as **Weight** and **height** parameters for your batch students and draw a scatter plot with appropriate label and title.

# Post Lab Questions:--

* 1. Considering datasets of your choice, create and explain the utility of the following charts:

Regression plot

Count plot

Bar plot

Violin plot

Heat map

Swarn chart

Pair chart

Pair grid

Facet Grid

Scatter plot

1. Swarn Chart - A **Swarm Chart** is a categorical scatter plot that avoids overlapping points. It is useful for visualizing the distribution of small datasets while showing individual data points. Each point is placed on the chart so that no two points overlap, offering insights into data density within categories.  
   Eg -   
   import matplotlib.pyplot as plt

import numpy as np

categories = ['A', 'B', 'C']

values = [np.random.rand(30) \* 10 for \_ in categories]

for i, (cat, val) in enumerate(zip(categories, values)):

    jitter = np.random.uniform(-0.2, 0.2, *size*=len(val))

    plt.scatter(np.full\_like(val, i) + jitter, val, *alpha*=0.7, *label*=*f*'Category {cat}')

plt.xticks(range(len(categories)), categories)

plt.title('Swarm Chart Approximation')

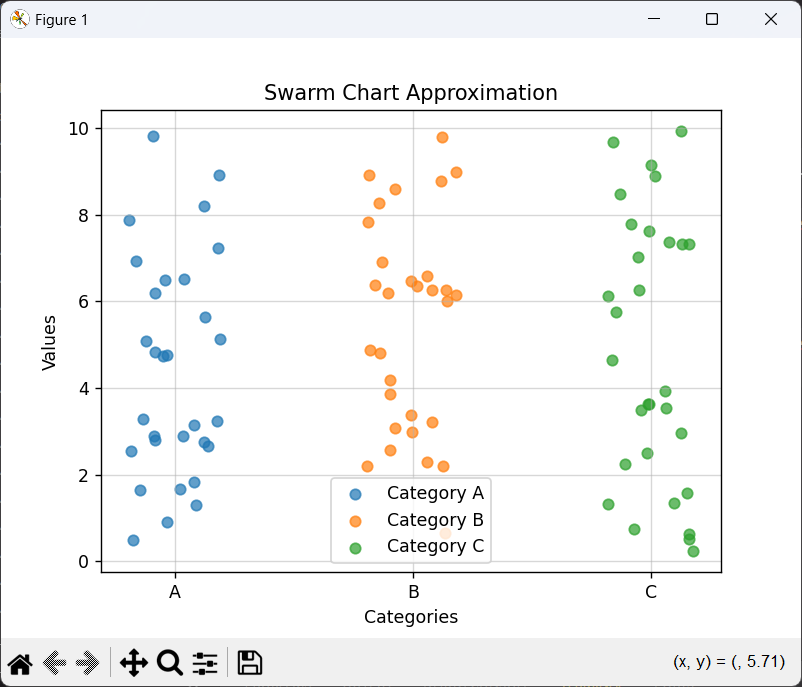
plt.xlabel('Categories')

plt.ylabel('Values')

plt.legend()

plt.grid(*alpha*=0.5)

plt.show()



1. Pair Chart - A **Pair Chart** displays scatter plots for every pair of variables in a dataset, along with histograms or density plots for individual variables on the diagonal. It’s commonly used to explore relationships and correlations between multiple features.
2. Pair Grid - A **Pair Grid** is an advanced version of the pair chart. It provides more flexibility by letting you customize each plot in the grid, such as adding different types of visualizations (e.g., scatter, regression, or density plots). It offers a structured way to analyze pairwise variable relationships.
3. Facet Grid - A **Facet Grid** organizes data into a grid of subplots based on one or more categorical variables. This allows for the comparison of plots across different subsets of the data. It’s especially useful for visualizing trends within groups or categories.
4. Scatter Plot - A **Scatter Plot** visualizes the relationship between two continuous variables by plotting them as individual points on an x-y axis. It’s an effective tool for identifying trends, patterns, clusters, and outliers in data.

Eg –

import matplotlib.pyplot as plt

import numpy as np

x = np.random.rand(100)

y = np.random.rand(100)

plt.scatter(x, y, *color*='blue', *alpha*=0.7)

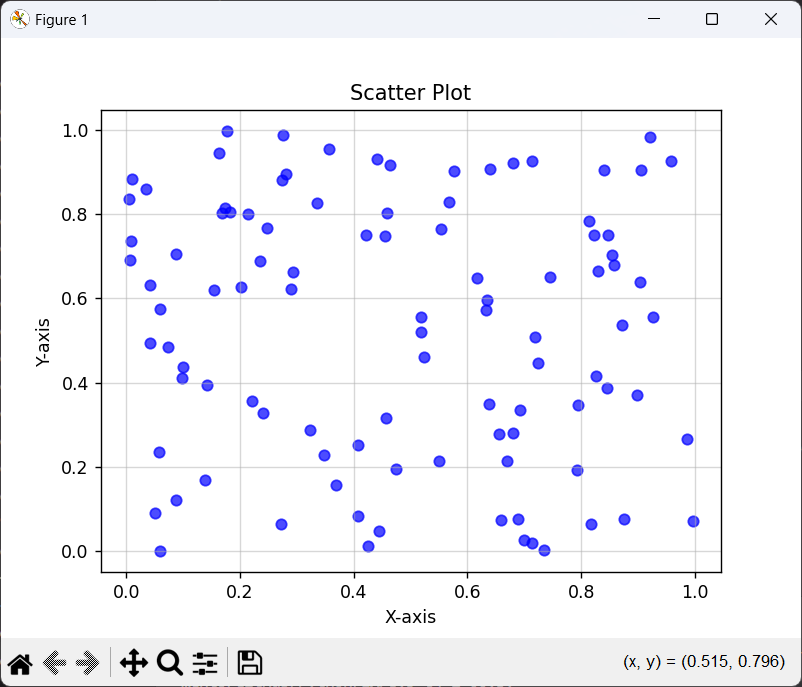
plt.title('Scatter Plot')

plt.xlabel('X-axis')

plt.ylabel('Y-axis')

plt.grid(*alpha*=0.5)

plt.show()



1. Regression Plot - A **Regression Plot** combines a scatter plot with a regression line, showcasing the relationship between two variables. It helps to understand whether and how two variables are linearly correlated, providing both visual and statistical insights.

Eg –

import matplotlib.pyplot as plt

import numpy as np

x = np.linspace(0, 10, 100)

y = 2 \* x + np.random.randn(100) \* 2

plt.scatter(x, y, *color*='blue', *alpha*=0.5, *label*='Data points')

plt.plot(x, 2 \* x, *color*='red', *label*='Regression Line')

plt.title('Regression Plot')

plt.xlabel('X-axis')

plt.ylabel('Y-axis')

plt.legend()

plt.grid(*alpha*=0.5)

plt.show()

A screen shot of a graph

Description automatically generated

1. Count Plot - A **Count Plot** displays the count of observations in each category of a categorical variable using bars. It is a straightforward way to visualize the distribution of categorical data.

Eg-   
import matplotlib.pyplot as plt

import numpy as np

categories = ['A', 'B', 'C', 'D', 'E']

counts = [10, 15, 7, 12, 9]

plt.bar(categories, counts, *color*='skyblue', *edgecolor*='black')

plt.title('Count Plot')

plt.xlabel('Category')

plt.ylabel('Count')

plt.grid(*axis*='y', *alpha*=0.5)

plt.show()

A screenshot of a graph

Description automatically generated

1. Bar Plot - A **Bar Plot** represents data using rectangular bars, where the height of each bar corresponds to the value of the variable. It is typically used to compare values across different categories or groups.  
   Eg -   
   import matplotlib.pyplot as plt

import numpy as np

labels = ['Group 1', 'Group 2', 'Group 3', 'Group 4']

values = [20, 34, 30, 35]

plt.bar(labels, values, *color*='lightgreen', *edgecolor*='black')

plt.title('Bar Plot')

plt.xlabel('Groups')

plt.ylabel('Values')

plt.grid(*axis*='y', *linestyle*='--', *alpha*=0.5)

plt.show()

A screenshot of a graph

Description automatically generated

1. Violin Plot - A **Violin Plot** is a combination of a box plot and a kernel density plot. It shows the distribution of data, including its probability density, across categories. It’s ideal for visualizing multiple distributions side-by-side.  
   Eg -   
   import matplotlib.pyplot as plt

import numpy as np

data = [np.random.normal(*loc*=0, *scale*=1, *size*=100),

        np.random.normal(*loc*=1, *scale*=1.5, *size*=100),

        np.random.normal(*loc*=-1, *scale*=0.8, *size*=100)]

plt.boxplot(data, *vert*=True, *patch\_artist*=True, *boxprops*=*dict*(*facecolor*='lightblue', *color*='blue'))

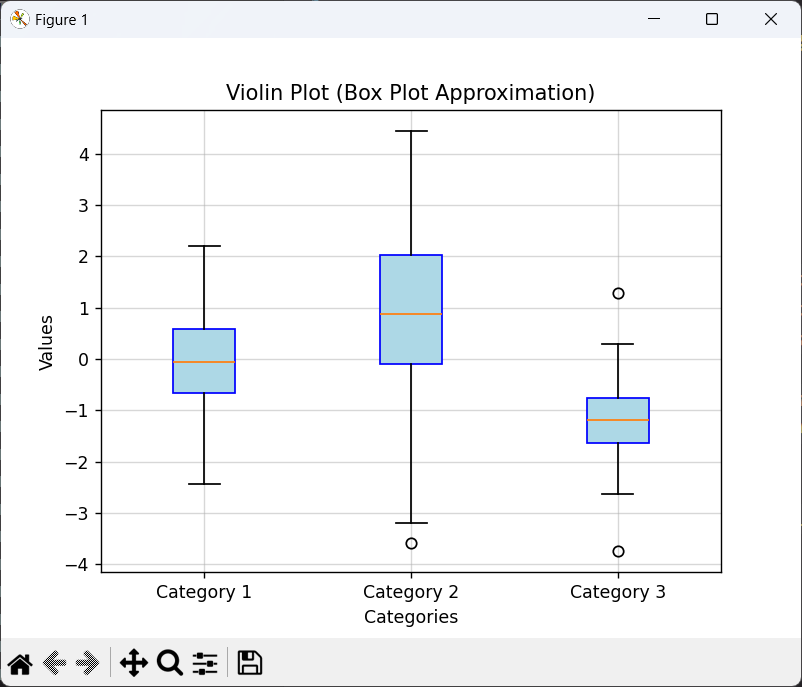
plt.title('Violin Plot (Box Plot Approximation)')

plt.xlabel('Categories')

plt.ylabel('Values')

plt.xticks([1, 2, 3], ['Category 1', 'Category 2', 'Category 3'])

plt.grid(*alpha*=0.5)

plt.show()  


1. Heat Map - A **Heat Map** represents data in a grid format where values are encoded using colors. It’s commonly used to visualize correlation matrices or other tabular data, helping identify patterns, relationships, or anomalies quickly.  
   Eg -   
   import matplotlib.pyplot as plt

import numpy as np

matrix = np.random.rand(10, 10)

plt.imshow(matrix, *cmap*='viridis', *interpolation*='nearest')

plt.colorbar(*label*='Value')

plt.title('Heatmap')

plt.xlabel('X-axis')

plt.ylabel('Y-axis')

plt.show()  
A screenshot of a computer screen

Description automatically generated

* 1. What is the Seaborn library? What are the Different categories of plot in 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 simplifies the process of visualizing complex datasets and includes built-in themes, color palettes, and tools for plotting relationships between variables.

# Books/ Journals/ Websites referred:

1. [Matplotlib Plotting (w3schools.com)](https://www.w3schools.com/python/matplotlib_plotting.asp) – Reference website.
2. Reema Thareja, Python Programming: Using Problem-Solving Approach, Oxford University Press, First Edition 2017, India
3. Sheetal Taneja and Naveen Kumar, Python Programming: A modular Approach, Pearson India, Second Edition 2018, India



Implementation details:  
1.

import matplotlib.pyplot as plt

import numpy as np

Month = np.array(['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])

Performance = np.array([23, 45, 38, 50, 42, 60, 55, 70, 65, 58, 75, 68])

plt.xlabel("Months")

plt.ylabel("Performance")

plt.title("Monthly Performance Trends")

plt.grid(True)

plt.plot(Month, Performance, marker="o", linestyle="-", color="r", label="Performance")

plt.show()

2. a)  
import matplotlib.pyplot as plt

import numpy as np

languages = np.array(["Java", "Python", "PHP", "JavaScript", "C#", "C++"])

popularity\_lang = np.array([22.2, 17.6, 8.8, 8, 7.7, 6.7])

data\_types = np.array(['Integer', 'String', 'Float', 'Boolean', 'List/Array'])

popularity\_types = np.array([30, 25, 20, 15, 10])

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

plt.subplot(1, 2, 1)

plt.bar(languages, popularity\_lang, color='skyblue')

plt.title('Popularity of Programming Languages')

plt.xlabel('Programming Language')

plt.ylabel('Popularity (%)')

plt.xticks(rotation=45)

plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.subplot(1, 2, 2)

plt.pie(popularity\_types, labels=data\_types, colors=['red', 'blue', 'green', 'orange', 'pink'])

plt.title('Popularity of Data Types')

plt.tight\_layout()

plt.show()

2. b)  
import matplotlib.pyplot as plt  
import numpy as np

languages = np.array(["Java", "Python", "PHP", "JavaScript", "C#", "C++"])

popularity\_lang = np.array([22.2, 17.6, 8.8, 8, 7.7, 6.7])

plt.bar(languages, popularity\_lang, *color*='skyblue')

plt.title('Popularity of Programming Languages')

plt.xlabel('Programming Language')

plt.ylabel('Popularity (%)')

plt.xticks(*rotation*=45)

plt.grid(*axis*='y', *linestyle*='--', *alpha*=0.7)

plt.show()

3.  
 import matplotlib.pyplot as plt

import numpy as np

height = np.array([170, 160, 175, 180, 165, 155, 185, 172, 168, 177, 162, 174, 158, 169, 173, 178, 160,176])

weight = np.array([60, 55, 68, 75, 58, 50, 80, 65, 62, 70, 54, 72, 52, 63, 67, 74, 56, 69])

plt.xlabel("Height")

plt.ylabel("Weight")

plt.title("Weight v/s Height")

plt.scatter(height, weight)

plt.show()

**Output:**

1.

A graph with red lines

Description automatically generated  
  
  
  
  
2. a)  
A screenshot of a graph

Description automatically generated  
  
  
2. b)  
**A screen shot of a graph

Description automatically generated**  
  
  
  
  
  
3.  
A screen shot of a graph

Description automatically generated

**Conclusion:**

Matplotlib is a versatile and powerful visualization library in Python, designed to cater to a wide range of plotting needs. From simple line charts to complex multi-plot layouts, it provides extensive functionality and customization options. As developers, we value Matplotlib for its ability to create publication-quality graphs, while also being highly adaptable for exploratory data analysis. Its compatibility with NumPy and other Python libraries ensures seamless integration into scientific and analytical workflows. Matplotlib gives us the tools to effectively communicate data-driven insights, making it an indispensable library in the Python ecosystem.

# Date: Signature of faculty in-charge