

Batch: C1_2 Roll No.: 16010123038

Experiment / assignment / Tutorial

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

Signature of the Staff In-charge with date

TITLE: Matplotlib library in Python

AIM: Write a program to explore the Matplotlib library

Expected OUTCOME of Experiment: To demonstrate Matplot library in python

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.

2. 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)`

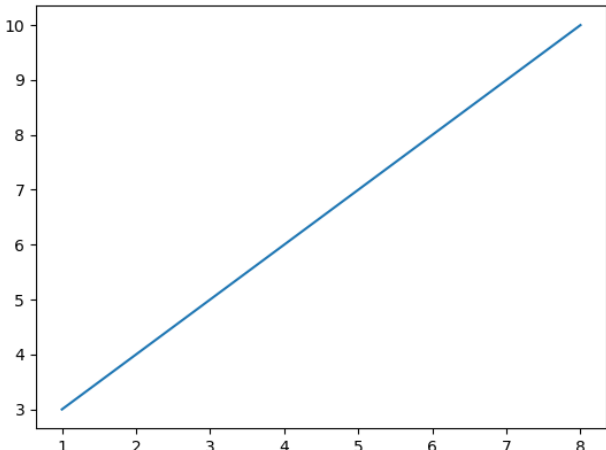
- `x, y`: These parameter are the horizontal and vertical coordinates of the data points. `x` values are optional.
- `fmt`: This parameter is an optional parameter and it contains the string value.
- `data`: This parameter is an optional parameter and it is an object with labelled data.

Returns:

This returns the following:

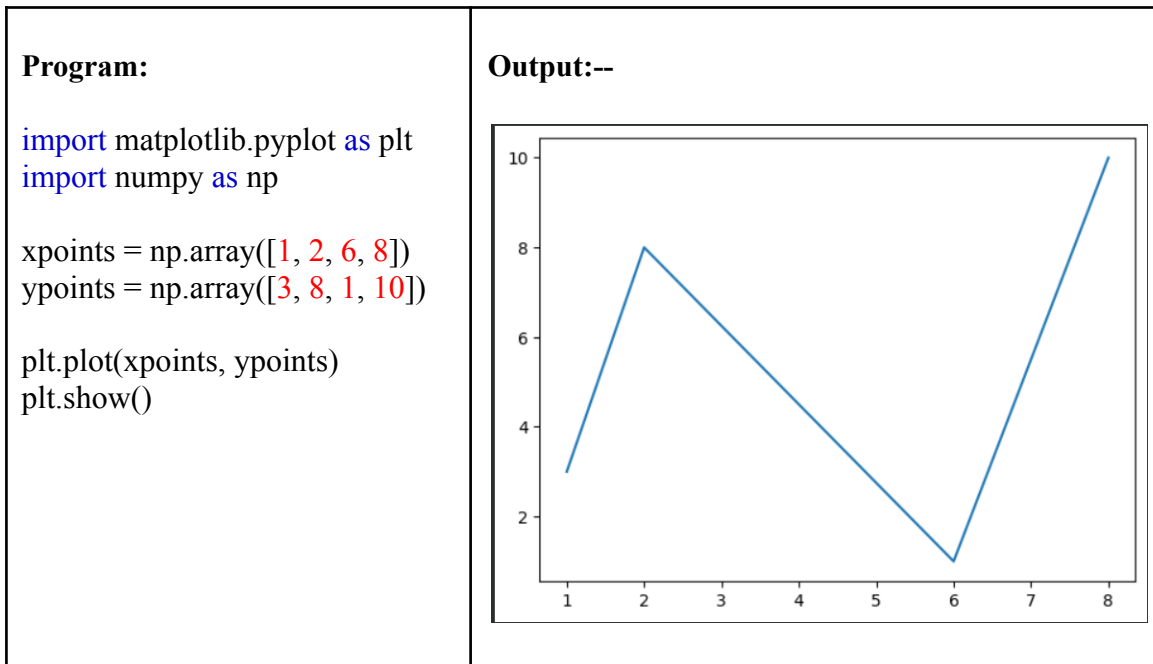
lines : This returns the list of Line2D objects representing the plotted data.

Example:-

Draw a line in a diagram from position (1, 3) to position (8, 10):	Output
<pre>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()</pre>	

1) Multiple Points

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



2) 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:	

<pre>import matplotlib.pyplot as plt import numpy as np ypoints = np.array([3, 8, 1, 10]) plt.plot(ypoints, linestyle = 'dotted') plt.show()</pre>	
--	--

3)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 `xlabel()` function in pyplot module of matplotlib library is used to set the label for the x-axis.

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

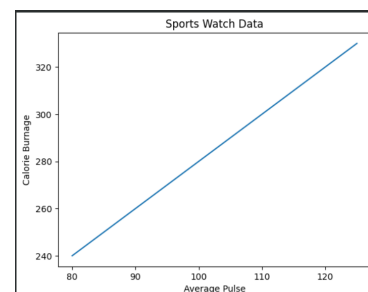
b. Create a Title for a Plot

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

Program:--

```
import numpy as np
import matplotlib.pyplot as plt
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:--



4) 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:

Syntax:-- `matplotlib.pyplot.scatter(x_axis_data, y_axis_data, s=None, c=None, marker=None, cmap=None, vmin=None, vmax=None, alpha=None, linewidths=None, edgecolors=None)`

- **x_axis_data**- An array containing x-axis data
- **y_axis_data**- An array containing y-axis data
- **s**- marker size (can be scalar or array of size equal to size of x or y)
- **c**- color of sequence of colors for markers
- **marker**- marker style
- **cmap**- cmap name
- **linewidths**- width of marker border
- **edgecolor**- marker border color
- **alpha**- blending value, between 0 (transparent) and 1 (opaque)

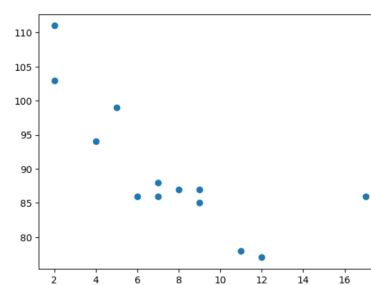
Except `x_axis_data` and `y_axis_data` all other parameters are optional and their default value is None. Below are the scatter plot examples with various parameters.

```
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.



```
import numpy as np
import matplotlib.pyplot as plt

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])

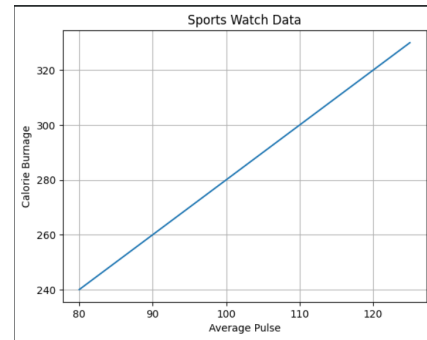
plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.plot(x, y)

plt.grid()

plt.show()
```

Output:



5) 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)
```

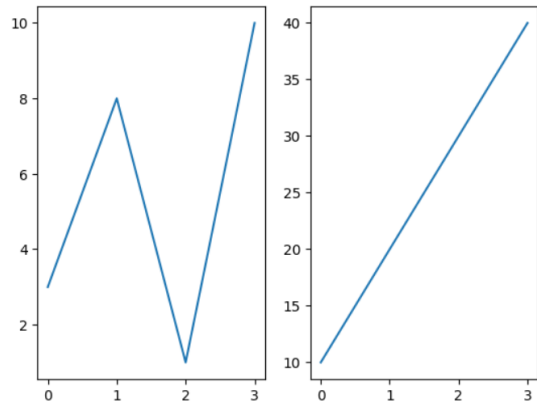
#plot 2:

```
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)
```

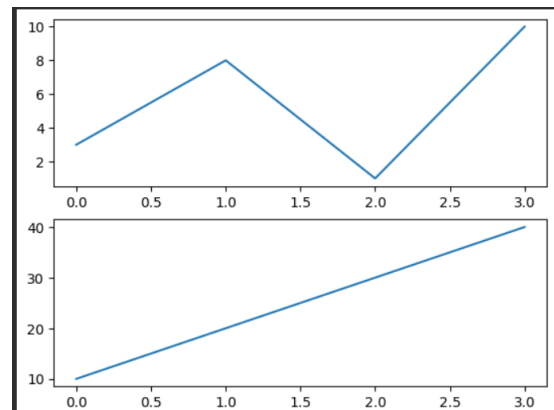
#plot 2:

```
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:--



6) Creating Bars

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

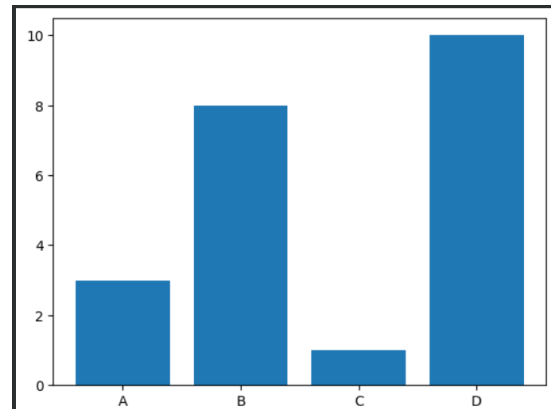


```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x,y)
plt.show()
```

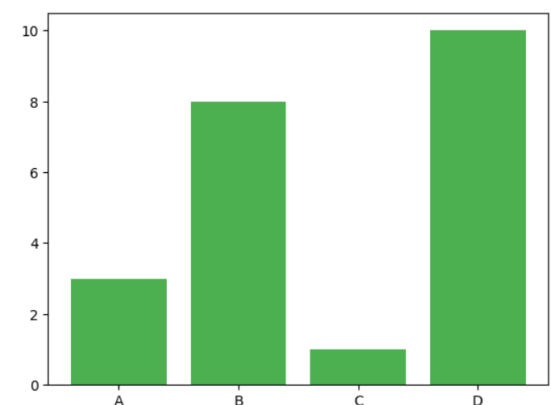
Output:--



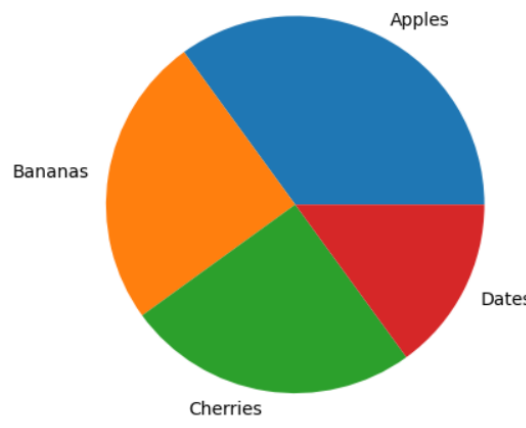
```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x, y, color = "#4CAF50")
plt.show()
```



7) Creating Pie Chart with Labels:

<pre>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()</pre>	Output: 
--	---

Problem Definition:

Note:-- All plot should be labelled on X-axis and Y-axis with Grid for each program.

1. Write a Python program to draw a line using given axis values with suitable label in the x axis, y axis and a title.

2. a) Write a Python programming to display a bar chart of the popularity of programming Languages. Also draw 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 of the popularity of programming Languages. **Hint: use the `barh()` function**

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



Implementation details:

1)

```
import matplotlib.pyplot as plt
import numpy as np

x1 = int(input("Enter x axis coordinate for startpoint:"))
x2 = int(input("Enter y axis coordinate for startpoint:"))
y1 = int(input("Enter x axis coordinate for endpoint:"))
y2 = int(input("Enter y axis coordinate for endpoint:"))

labx = input("Enter x axis label name:")
laby = input("Enter y axis label name:")

xn = np.array([x1, x2])
yn = np.array([y1, y2])

plt.xlabel(labx)
plt.ylabel(laby)

plt.plot(xn, yn)
plt.show()
```

2)

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["Java", "Python", "PHP", "JavaScript", "C#", "C++"])
y = np.array([22.2, 17.6, 8.8, 8, 7.7, 6.7 ])

plt.bar(x, y, color = "#4cafa7")
plt.show()

plt.pie(y, labels = x)
plt.show()

plt.barh(x, y, color = "#814caf")
plt.show()
```

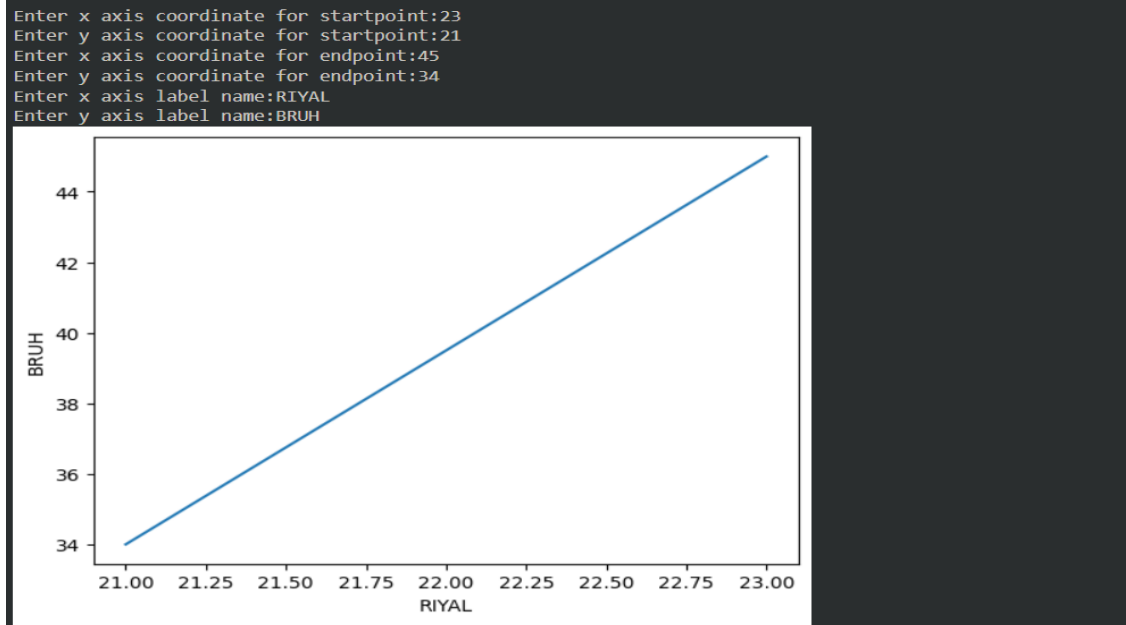
3)

```
import matplotlib.pyplot as plt
import numpy as np

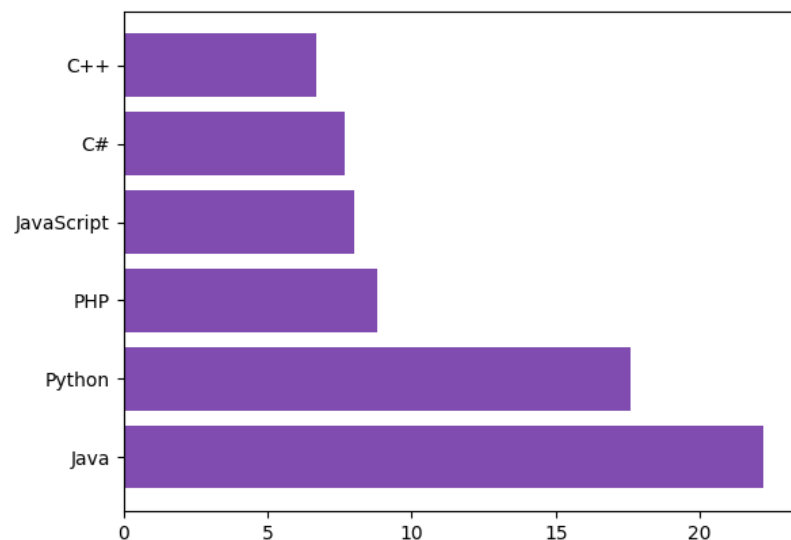
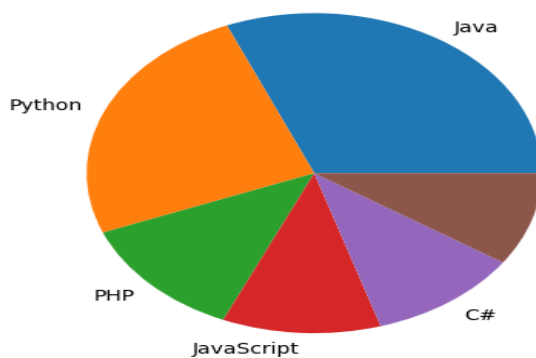
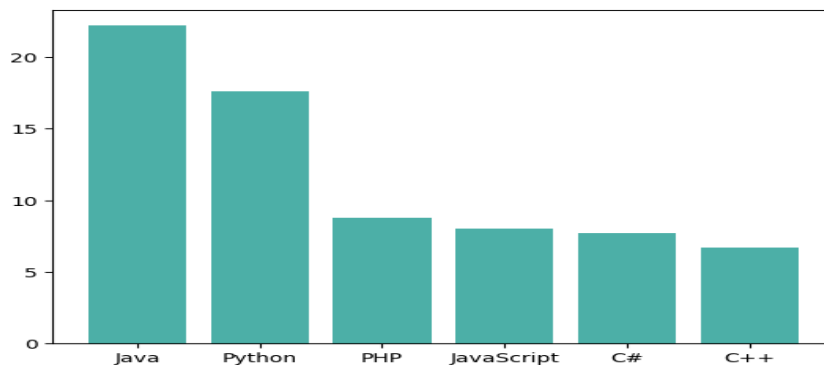
weight=[66,65,74,56,78,66,53,64,88,68,78,71,89]
height=[122,155,173,156,178,188,123,155,133,123,164,123,156]

plt.scatter(weight,height)
plt.xlabel("weight")
plt.ylabel("height")
plt.show()
```

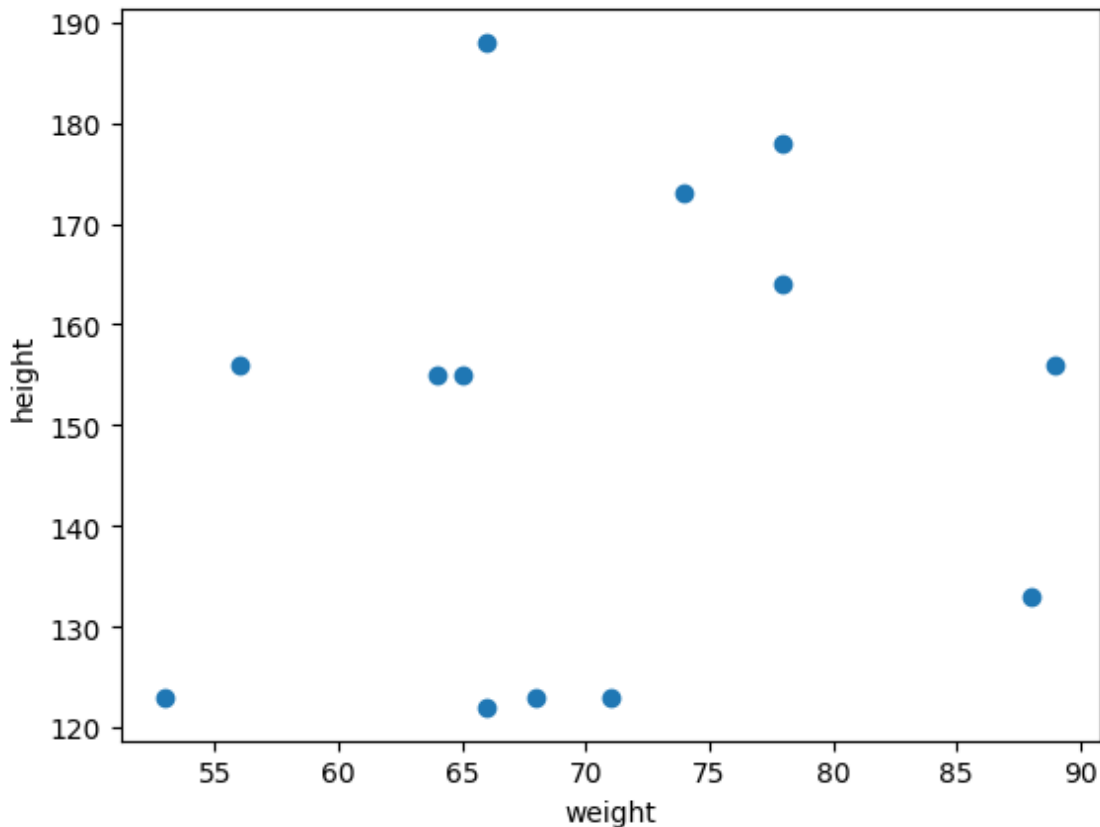
1)



2)



3)



Post Lab Questions:--

- 1) Considering datasets of your choice, create and explain the utility of following charts:

1) Swarm chart	6) Regression plot
2) Pair chart	7) Count plot
3) Pair grid	8) Bar plot
4) Facet Grid	9) Violin plot
5) Scatter plot	10) Heat map

ans: Swarm Chart:

Utility: Perfect for displaying how a continuous variable is distributed across various categories.

Example Dataset: Height measurements categorized by gender.

Explanation: It shows individual data points along a categorical axis, allowing easy comparison of height distributions between genders.

Pair Chart:

Utility: to display the pairwise connections between various dataset variables.

Example Dataset: Multiple features like height, weight, age, etc., for individuals.

Explanation: It creates a grid of scatterplots or other visual representations for each pair of variables, displaying correlations and patterns between them.

Pair Grid:

Utility: Similar to Pair Chart, but more customizable and flexible.

Example Dataset: Multiple features or dimensions of a dataset for thorough analysis.

Explanation: Allows the creation of customized visualizations (scatterplots, histograms, etc.) for every pairwise combination of variables, facilitating in-depth exploration and analysis.

Facet Grid:

Utility: To create a grid of subplots based on categorical variables.

Example Dataset: Sales data categorized by region, time, and product type.

Explanation: Useful for visualizing how different categorical factors influence a particular variable (e.g., sales) by creating separate plots for each category combination.

Scatter Plot:

Utility: Shows the relationship between two continuous variables.

Example Dataset: Relationship between temperature and ice cream sales.

Explanation: Displays individual data points, making it easy to identify patterns, trends, or correlations between two variables, such as how sales vary with temperature.

Utility: Visualizes the relationship between two continuous variables and depicts a regression line that best fits the data.

Example Dataset: Examining the relationship between hours studied and exam scores.

Explanation: Helps to identify the direction and strength of the relationship between variables and showcases the regression line for predictive analysis.

Count Plot:

Utility: Displays the count of occurrences of categorical variables.

Example Dataset: Frequency of different car brands in a survey.

Explanation: Presents a bar chart where each bar represents the count of occurrences of a categorical variable, useful for understanding distribution and frequency.

Bar Plot:

Utility: Compares categorical data using rectangular bars with heights proportional to the values they represent.

Example Dataset: Sales figures for different products in a store.

Explanation: Efficiently compares quantities of categorical data, making it easy to see which category has the highest or lowest values.

Violin Plot:

Utility: Visualizes the distribution of a continuous variable across several levels of a categorical variable.

Example Dataset: Distribution of test scores across different school subjects.

Explanation: Shows the distribution of data similar to a box plot but also includes a kernel density estimation, providing more insight into data distribution and density.

Heat Map:

Utility: Represents data values in a matrix using color intensity, useful for visualizing correlations or relationships in large datasets.

Example Dataset: Correlation matrix of stock market data.

Explanation: Enables quick identification of patterns or relationships between variables by representing values using colors, especially helpful for large datasets.

2) What is Seaborn library? What are Different categories of plot in Seaborn.

ans: Based on Matplotlib, Seaborn is a robust Python data visualization library. Its goal is to produce visually appealing and educational statistical graphics. By offering aesthetically pleasing default themes and high-level abstractions over Matplotlib, Seaborn makes the process of creating complex visualizations simpler.

Seaborn provides various categories of plots to explore and represent different types of data:

Relational Plots, Categorical Plots, Distribution Plots, Matrix Plots, Multi-plot Grids

Books/ Journals/ Websites referred:

1. [Matplotlib Plotting \(w3schools.com\)](https://www.w3schools.com/matplotlib/matplotlib_intro.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

Conclusion:

This module covered the usage of matplotlib and its many functions, including pyplot, bar(), barh(), pie(), scatter(), and so forth.

We learned to plot graphs using both pre-provided data and user input, and we also learned how to use the .show() method to output a visual representation of the graphs.

Date: 20 -10-2023

Signature of faculty in-charge