MATPLOTLIB

- Low level graph plotting python library that serves as a visualization utility.
- We use **pyplot** submodule from **matplotlib** for plotting the graphs.
- **plot()** is used to develop the diagram using the inputs passed through the function
- **show()** is used to display the developed graph.
- By default, the **plot()** draws a line through the markers(point to point)
- It takes parameters for marking the points:
  - 1st parameter takes points on the **x-axis. [x1,x2]**
  - 2nd parameter takes points on the **y-axis. [y1,y2]**
- To plot only the markers, you can use the shortcut *string notation* parameter **'o'**, which means 'rings'.

- **Figure Size :**
  - **plt.figure(figsize=(<**length**>,<**height**>))**
  - To change the size of the plot.
  - Max length is : **12**

- **Markers :**
  - They are the points on the graph against the value provided.
  - You can use the keyword argument **marker** to emphasize each point with a specified marker:
    - **'o' :** circle
    - **' * ' :** star
    - **' , ' :** pixel
    - **' . ' :** point
    - **' + ' :** plus
    - And so on…….
  - **Format String : (** fmt **)**
    - We can also use the **shortcut string notation parameter** to specify the marker.
    - Syntax : `marker|line|color`

Line Syntax	Description
'-'	Solid line
':'	Dotted line
'--'	Dashed line
'-.'	Dashed/dotted line

Color Syntax	Description
'r'	Red
'g'	Green
'b'	Blue
'c'	Cyan
'm'	Magenta
'y'	Yellow
'k'	Black
'w'	White

- **Marker size :**
  - Keywords : **ms** or **markersize**

- **Marker color :**
  - To set the color to the edges of the marker
    - Keywords : **mec** or **markeredgecolor**
  - To set the color to the face of the marker
    - Keywords : **mfc** or **markerfacecolor**

- **Lines :**
  - Line Style :
    - Keyword : **ls** or **linestyle**

Style	Or
'solid' (default)	'-'
'dotted'	':'
'dashed'	'--'
'dashdot'	'-.'
'None'	'' or ' '

  - Line Color :
    - Keyword : **color** or **c**
    - You can also provide the color in **Hexadecimal** format.
  - Line Width :
    - Keyword : **linewidth** or **lw**

  - For plotting multiple lines, just use **plt.plot()** as many as the number of lines are.
  - You can also plot many lines by adding the points for the x- and y-axis for each line in the same **plt.plot()** function.

- **Labels and Title :**
  - Create **labels** for a plot :
    - **xlabel() :** for labeling the X-axis
    - **ylabel() :** for labeling the Y-axis
  - Create **title** for a plot :
    - **title() :** used to create title for the plot
  - Set font properties :
    - **fontdict :** parameter is used to set the font properties of Labels and Title.

- This parameter accepts the dictionary as input

```
font1 = {'family':'serif','color':'blue','size':20}
font2 = {'family':'serif','color':'darkred','size':15}

plt.title("Sports Watch Data", fontdict = font1)
plt.xlabel("Average Pulse", fontdict = font2)
plt.ylabel("Calorie Burnage", fontdict = font2)
```

- Position the title :
  - **loc** parameter inside the **title()** to set the position of the title.
  - Legal values : '**left**', '**right**', '**center**' {Default}

- **Adding Grid lines** :
  - **grid()** : used to plot the grid lines on the graph.
    - **which = 'both' / 'major' / 'minor' :** Grid line to apply changes on.
    - **axis = 'x' / 'y' / 'both' :** Which axis to apply grid lines.
    - Setting the grid line properties :
    - **color**, **linestyle**, **linewidth** : values are as specified for the graph line

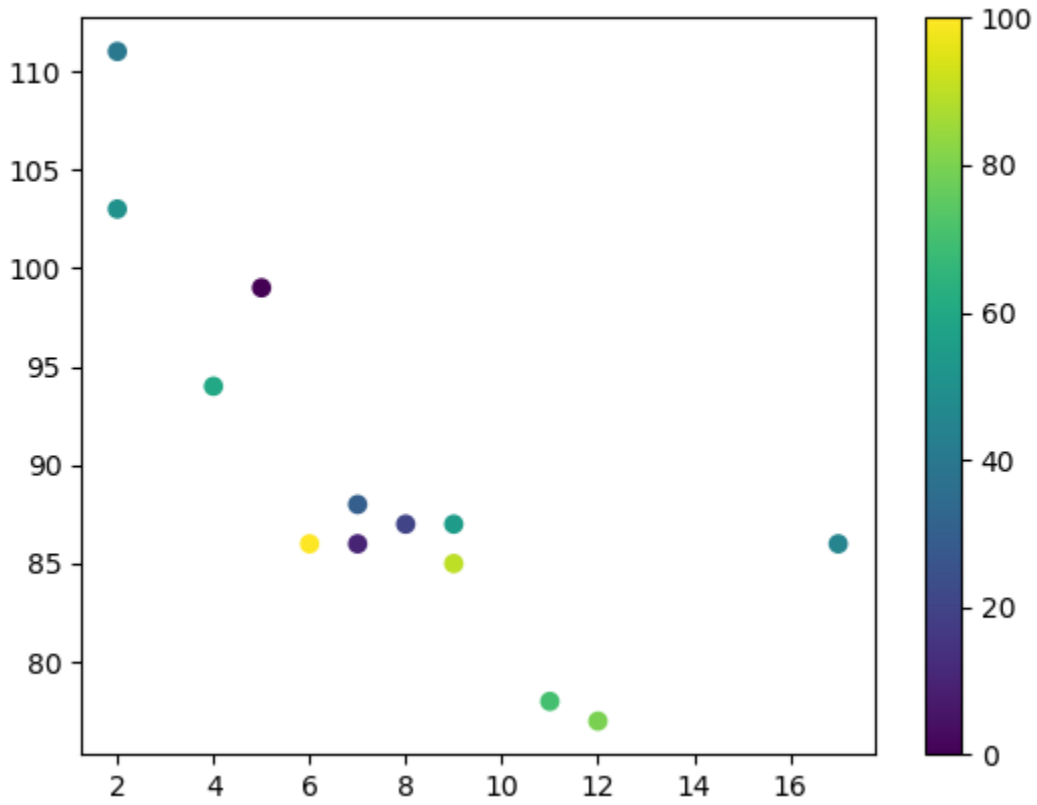
- **Subplot :**
  - Displaying multiple plots.
  - **subplot()** : using this function we can display multiple graphs in a single diagram.
  - This function takes three arguments which defines the layout of the figure.
  - The layout is organized by *rows* and *columns,* represented by *first* and *second* argument respectively.
  - And the third argument represents the index of the current plot.
  - Examples : plt.subplot(1,2,2) #This has 2 plots, where the current plot is the second one.

    plt.subplot(2,3,1) # This has 6 plots, where the current plot is the first one.
  - **suptitle()** : used to set the title for the entire diagram.
  - **title() :** used to set the title for an individual plot.

- **Scatter :**
  - Pass **kind='scatter'** inside the **plot().** OR
  - **scatter() :** function is used to plot the scatter graphs.
  - It leaves a **dot for each observation,** it requires two arrays of same length as x-axis and y-axis
  - You can compare two different sets on the same plot , by just calling it twice with different data sets.

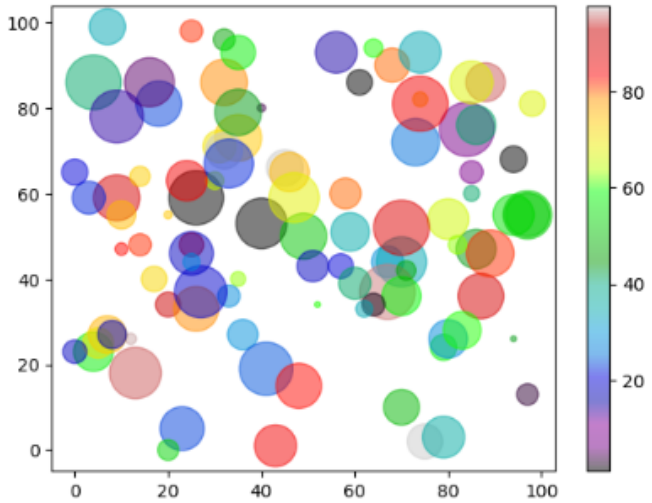
- To change the color, we use **color** or **c** inside the **scatter()** or **plot()** function.
- The default color will be **blue** and **orange.**
- You can also pass the colors as an array to **color each dot,** or also by passing the color maps using **cmp.**
- There are so many in-built color maps.
- You can display the **colorbar** using **plt.colorbar()**

```
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])
colors = np.array([0, 10, 20, 30, 40, 45, 50, 55, 60, 70, 80, 90, 100])

plt.scatter(x, y, c=colors, cmap='viridis')

plt.colorbar()

plt.show()
```



- You can also resize the dots by passing **s=<**value**> or** by passing array of values to the **s**
- **alpha** : parameter is used to set the transparency of the dots.

```
x = np.random.randint(100, size=(100))
y = np.random.randint(100, size=(100))

# Defining  different color values for the colorbar
colors = np.random.randint(100, size=(100))

#Defining the different sizes for the dots
sizes = 10 * np.random.randint(100, size=(100))

# Plotting the SCATTER graph with color map : nipy_spectral, transparency : 0.5
plt.scatter(x, y, c=colors, s=sizes, alpha=0.5, cmap='nipy_spectral')

# To display the colorbar with the plot
plt.colorbar()

plt.show()
```



- **Bars :**
  - **bar()** is used to plot the bar graphs.
  - Takes two arguments, which represents the layout of the bars.
  - Categories and their values are represented by 1st and 2nd arguments respectively.
  - **barh() :** to get the **horizontal** bars
  - **color** parameter inside the function takes the color name for the bars.
  - **width** parameter for vertical bars inside the function takes the width value of the bars. [**bar()**]
  - **height** parameter for horizontal bars inside the function takes the height value of the bars. [**barh()**]
  - Default is **0.8** for width/height.

- **Histograms :**
  - Pass **kind='hist'** inside the **plot().** OR
  - **hist()** is used to plot the histogram.
  - This form of representation gives the **frequency distribution**, i.e., number of observations in each interval.
  - Takes only one argument .
  - Pass **orientation='horizontal'** to plot the histogram horizontally

- To draw a curve over the histogram, we use the **pdf()** method which is the probability density function inside the **plot().**

- **Pie Charts :**
  - **pie()** is used to get the pie charts.
  - Starts from the x-axis [**0 degree**] and moves counter-clockwise by default.
  - Labels :
    - Pass the array of labels of same length as the value array inside the **pie()** function using **label**
  - Start Angle :
    - To change the start angle from 0, use **startangle** parameter inside the function.
  - Explode :
    - When you want one of your wedges to stand out use this method.
    - **explode** argument is used inside the function
    - Should pass an array of length same as the values array.
    - Supply the value of how much it needs to get separate from the center inside the array corresponding to the value.
  - Shadow :
    - By setting **shadow** parameter **True** inside the function enables the shadow to the pie chart.
  - To show the value of the wedge :
    - **autopct** parameter is passed inside **pie().**
    - For example :

      pie(arrayValues, labels=arrayValues.index**,** autopct=%1.2f%%**)**

      This displays the wedge's floating value with 2 digits after decimal point. (**.2f**)
  - Colors :
    - Use the **colors** parameter inside the function and pass an array of colors for each wedge.
  - Legend :
    - To add a list of explanation to the pie chart
    - Use **plt.legend()** function for adding legend.
    - To have a title for the legend, pass **title** parameter inside the **legend()**
    - **loc** to specify the location of the legend

```
The strings ``'upper left'``, ``'upper right'``, ``'lower left'``,
``'lower right'`` place the legend at the corresponding corner of the
axes.

The strings ``'upper center'``, ``'lower center'``, ``'center left'``,
``'center right'`` place the legend at the center of the corresponding edge
of the axes.

The string ``'center'`` places the legend at the center of the axes.
```

    - **“best”** can be passed to **loc** to make the tool to decide the best area to place the **legend.**

- To save the charts, use **plt.savefig().**
-

- We use histogram plot through **seaborn**
    - **seaborn.histplot()**
        - **data :** pass the value to plot
        - **x, y :** give names to x and y axis
        - **kde :** set to **True** to get a smooth line over the bar, which represents the distribution.
    - **seaborn.kdeplot()** : Plot univariate or bivariate distributions using kernel density estimation.