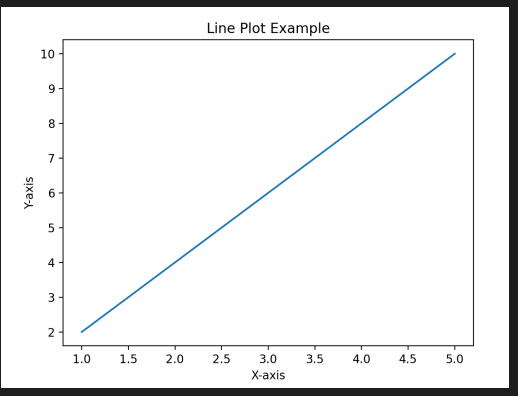
Matplotlib in Python

- Python library used for creating 2D visualizations and plots.

- It is like a canvas where you can draw different kinds of graphs, such as line plots, bar charts, histograms, scatter plots, etc.

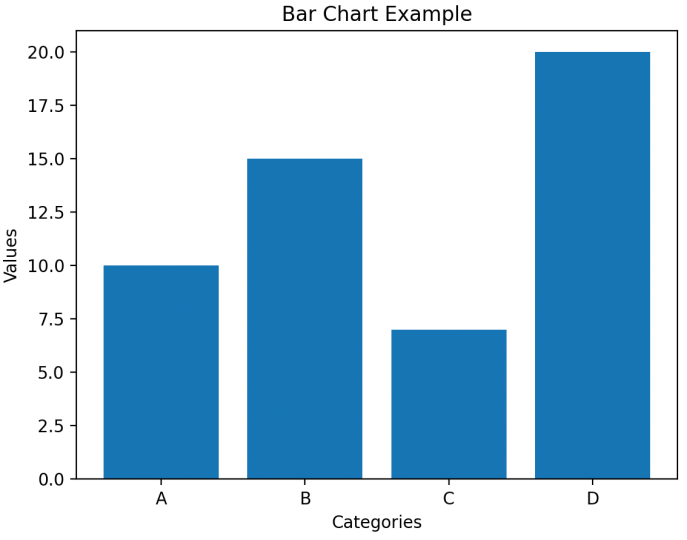
1. Line Plot

```python
import matplotlib.pyplot as plt

#Data
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

# Create a line plot
plt.plot( *args: x, y)

# Add labels and title
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Line Plot Example")

# Show the graph
plt.show()
```

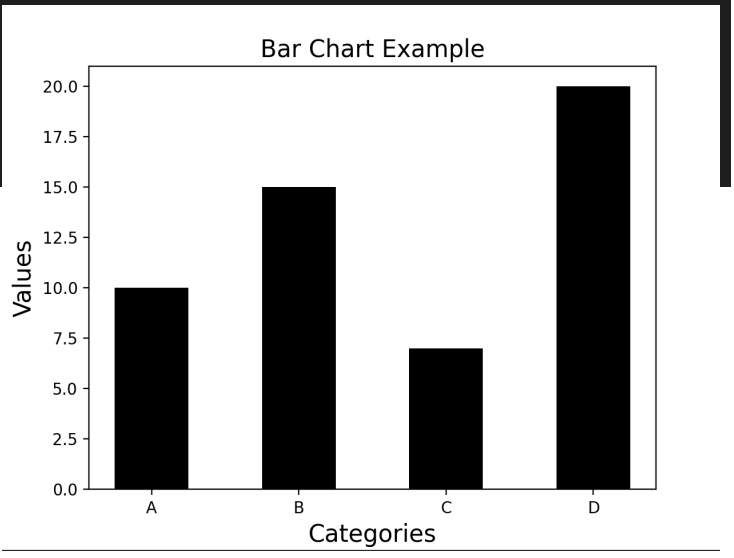


2. Bar Chart

```python
import matplotlib.pyplot as plt

# Data
categories = ['A', 'B', 'C', 'D']
values = [10, 15, 7, 20]

# Create a bar chart
plt.bar(categories, values)

# Add labels and title
plt.xlabel("Categories")
plt.ylabel("Values")
plt.title("Bar Chart Example")

# Show the graph
plt.show()
```

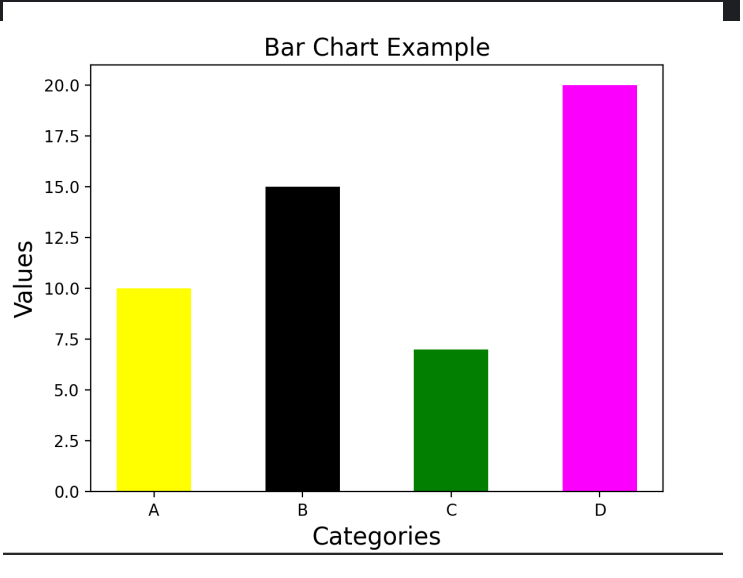


```
# Data
categories = ['A', 'B', 'C', 'D']
values = [10, 15, 7, 20]

# Create a bar chart
plt.bar(categories, values, width=0.5, color="black")

# Add labels and title
plt.xlabel( xlabel: "Categories", fontsize = 15)
plt.ylabel( ylabel: "Values", fontsize = 15)
plt.title( label: "Bar Chart Example", fontsize = 15)

# Show the graph
plt.show()
```



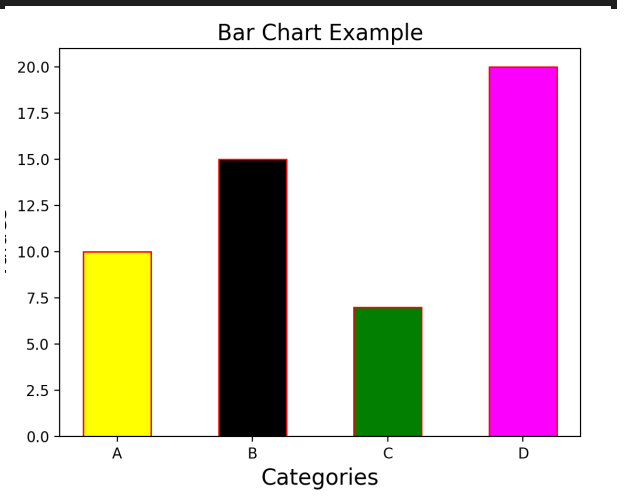
Color each bar with different color

```
c = ["Yellow", "black", "green", "magenta"]
plt.bar(categories, values, width=0.5, color=c)
```



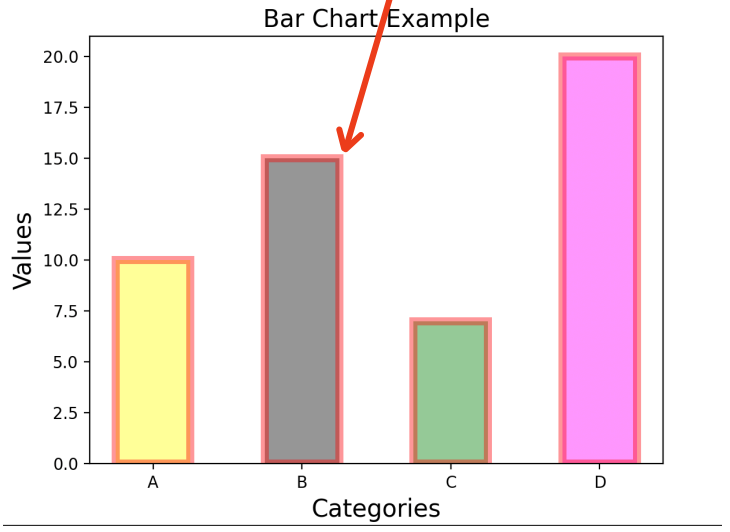
Change edge color of bars

```
c = ["Yellow", "black", "green", "magenta"]
plt.bar(categories, values, width=0.5, color=c, edgecolor = "red")
```



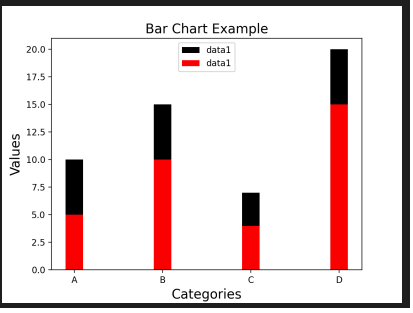

```
c = ["Yellow", "black", "green", "magenta"]
plt.bar(categories, values, width=0.5, color=c, edgecolor = "red", linewidth=5, alpha=0.4)
```

to reduce the visiblity

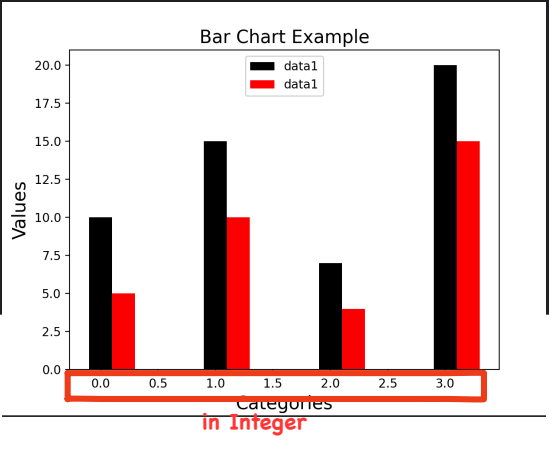


creating multiple bar graphs in the single chart

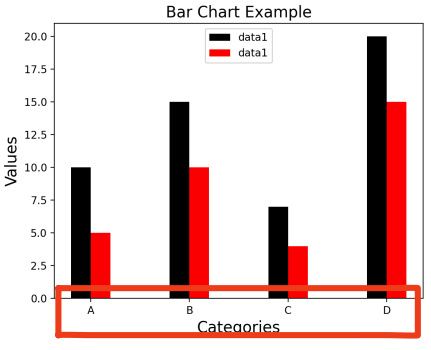
```python
import matplotlib.pyplot as plt

# Data
categories = ['A', 'B', 'C', 'D']
values1 = [10, 15, 7, 20]
values2 = [5, 10, 4, 15]


plt.bar(categories, values1, width=0.2, color="black",label = "data1")
plt.bar(categories, values2, width=0.2, color="red",label = "data1")
plt.legend(loc = "upper center")    # to make label visible we need to provide legend()

plt.xlabel( xlabel: "Categories", fontsize = 15)
plt.ylabel( ylabel: "Values", fontsize = 15)
plt.title( label: "Bar Chart Example", fontsize = 15)

# Show the graph
plt.show()
```




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

categories = ['A', 'B', 'C', 'D']
values1 = [10, 15, 7, 20]
values2 = [5, 10, 4, 15]

width = 0.2
p1 = np.arange(len(categories))    # [0 1 2 3]
p2 = [i+width for i in p1]

plt.bar(p1, values1, width, color="black",label = "data1")
plt.bar(p2, values2, width, color="red",label = "data1")
plt.legend(loc = "upper center")    # to make label visible we need to provide legend()

plt.xlabel( xlabel: "Categories", fontsize = 15)
plt.ylabel( ylabel: "Values", fontsize = 15)
plt.title( label: "Bar Chart Example", fontsize = 15)

plt.show()
```

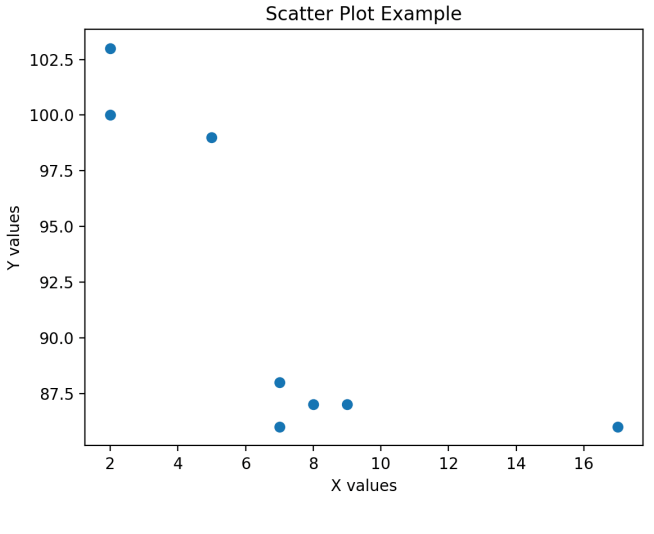


```python
import matplotlib.pyplot as plt
import numpy as np

categories = ['A', 'B', 'C', 'D']
values1 = [10, 15, 7, 20]
values2 = [5, 10, 4, 15]

width = 0.2      # Bar width (smaller values create narrower bars)
p1 = np.arange(len(categories))    # [0 1 2 3]  positions for the first set of bars
p2 = [i+width for i in p1]  # Generate the positions for the second group of bars
                            # Offset the positions by the bar width for the second dataset
print(p2)
plt.bar(p1, values1, width, color="black",label = "data1")
plt.bar(p2, values2, width, color="red",label = "data1")
plt.legend(loc = "upper center")    # to make label visible we need to provide legend()

plt.xlabel( xlabel: "Categories", fontsize = 15)
plt.ylabel( ylabel: "Values", fontsize = 15)
plt.title( label: "Bar Chart Example", fontsize = 15)

plt.xticks(p1+width/2, categories)   # Set custom x-axis tick positions and labels
                divide by 2 to keep categories at center

plt.show()
```

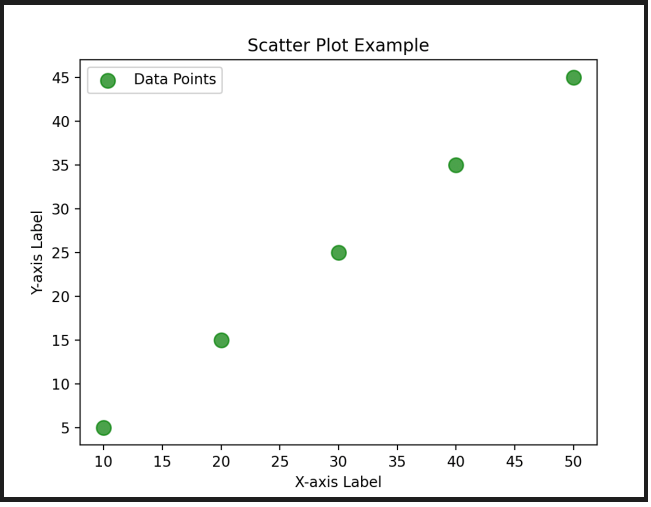


3. Scatter Plot

```
import matplotlib.pyplot as plt

# Data
x = [5, 7, 8, 7, 2, 17, 2, 9]
y = [99, 86, 87, 88, 100, 86, 103, 87]

# Create a scatter plot
plt.scatter(x, y)

# Add labels and title
plt.xlabel("X values")
plt.ylabel("Y values")
plt.title("Scatter Plot Example")

# Show the graph
plt.show()
```



```python
import matplotlib.pyplot as plt

# Data
x = [10, 20, 30, 40, 50]
y = [5, 15, 25, 35, 45]

# Create scatter plot
plt.scatter(x, y, color='green', label='Data Points', s=100, alpha=0.7)

# color: Set the color of the points.
# label: Add a label for the legend.
# s: Adjust the size of the points.
# alpha: Set the transparency of the points (0 is transparent, 1 is opaque)

# Add labels, title, and legend
plt.title("Scatter Plot Example")
plt.xlabel("X-axis Label")
plt.ylabel("Y-axis Label")
plt.legend()

# Show the plot
plt.show()
```

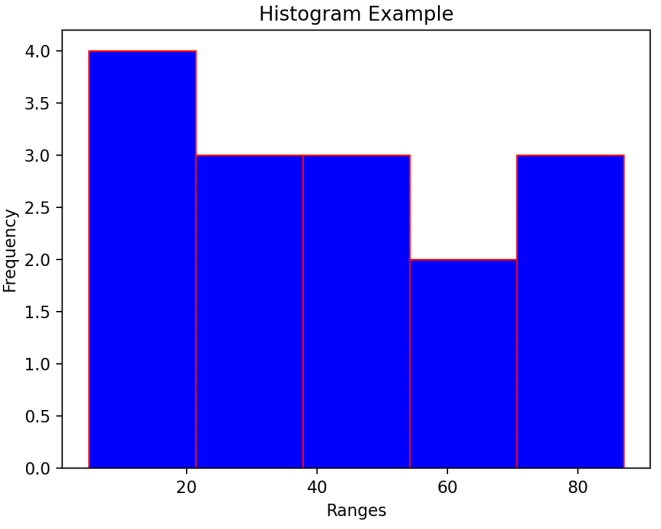


Scatter plots are best used when you want to:

1. **Show Relationships:** To see if two things are related (e.g., studying hours vs. exam scores).
2. **Spot Patterns:** Identify trends, like increasing or decreasing relationships

4. Histogram

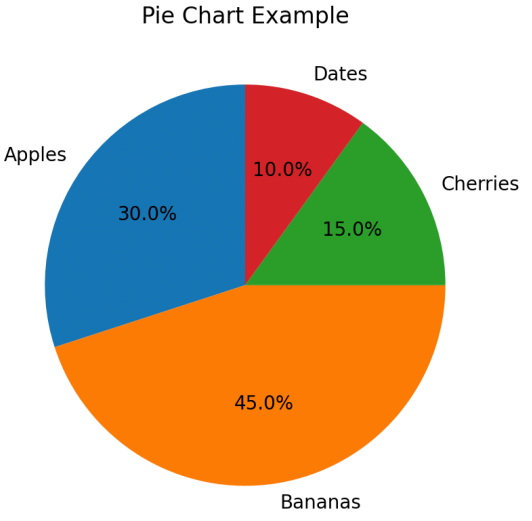
```python
import matplotlib.pyplot as plt

# Data
data = [22, 87, 5, 43, 56, 73, 55, 54, 11, 20, 51, 5, 79, 31, 27]

# Create a histogram
plt.hist(data, bins=5, color='blue', edgecolor='red')

# Add labels and title
plt.xlabel("Ranges")
plt.ylabel("Frequency")
plt.title("Histogram Example")

# Show the graph
plt.show()
```



5. Pie Chart

Show proportions or percentages:

```python
import matplotlib.pyplot as plt

# Data
labels = ['Apples', 'Bananas', 'Cherries', 'Dates']
sizes = [30, 45, 15, 10]

# Create a pie chart
plt.pie(sizes, labels=labels, autopct='%.1f%%', startangle=90)

# Add title
plt.title("Pie Chart Example")

# Show the graph
plt.show()
```

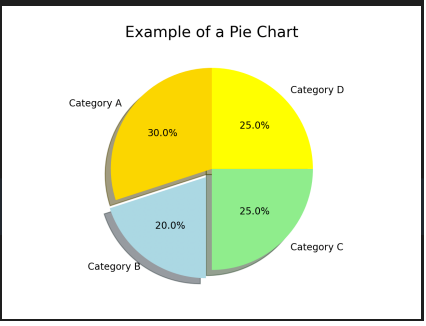


autopct='%1.1f%%'

%1.1f: Display one digit after decimal.

%%: Escape the % symbol to actually show it as part of the text.

```python
import matplotlib.pyplot as plt

# The sizes (proportions) for each section of the pie
sizes = [30, 20, 25, 25]  # Percentages or proportions

labels = ['Category A', 'Category B', 'Category C', 'Category D']
# Optional colors for each slice
colors = ['gold', 'lightblue', 'lightgreen', 'yellow']

plt.pie(
    sizes,                  # Data for the pie chart
    labels=labels,          # Labels for each slice
    colors=colors,          # Colors for the slices
    autopct='%1.1f%%',      # Display percentages with 1 decimal place
    startangle=90,          # Start the first slice at the top (12 o'clock)
    explode=[0, 0.1, 0, 0],  # Slightly "explode" the 2nd slice for emphasis
    shadow=True             # Add a shadow to make the chart look 3D
)

plt.title( label: "Example of a Pie Chart", fontsize=16)

plt.show()
```




```python
plt.title( label: "Example of a Pie Chart", fontsize=16)
plt.savefig( *args: "pie_chart.png", dpi=300)
# plt.show()
```

- The plt.savefig() function in Matplotlib is used to save your plot as an image file.

- Supported file formats include PNG, JPEG, SVG, PDF, and others.
  Example:
  "pie_chart.jpeg" for JPEG format.
  "pie_chart.pdf" for PDF format.

dpi=300: DPI stands for "dots per inch."
It determines the resolution of the saved image. A higher DPI produces a higher-quality image.
Default DPI in Matplotlib is 100.

Common DPI settings:
72 DPI: Low resolution, suitable for quick previews.
300 DPI: High resolution, commonly used for publishing.

Why Use plt.savefig Instead of plt.show?
plt.show(): Displays the plot interactively in a GUI or notebook but doesn't save it.
plt.savefig(): Saves the plot directly to a file and does not display it.
Save Before plt.show():

Always call plt.savefig() before plt.show() in your script. Once plt.show() is called, the plot may be cleared, and saving won't work.

Subplot in Python

A subplot is a way to display multiple plots (charts/graphs) within a single figure in Python using Matplotlib. It allows us to compare different datasets or relationships side by side.

Where to Use a Subplot?

1. Comparing Multiple Datasets
   Example: Showing sales trends of different products in one figure.

2. Analyzing Relationships Between Variables
   Example: Plotting temperature vs. humidity and wind speed vs. temperature in the same figure.

3. Displaying Different Types of Plots Together
   Example: Line chart for revenue, bar chart for expenses, and pie chart for profit distribution.

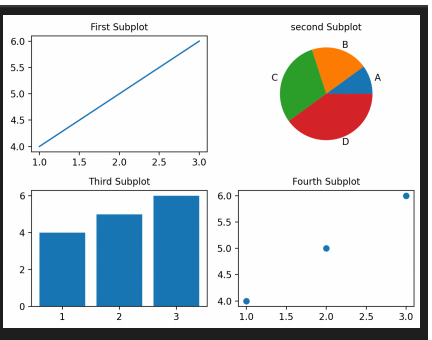
4. Saving Space and Making Reports More Readable
   Instead of multiple separate plots, subplots arrange everything neatly in one figure.

Using plt.subplot()

Syntax:

plt.subplot(total_rows, total_columns, plot_number)

- plot_number: The position of the plot (starting from 1, left to right, top to bottom)




```
import matplotlib.pyplot as plt

x = [1,2,3]
y = [4,5,6]
plt.subplot( *args: 2,2,1)
plt.plot( *args: x,y)
plt.title( label: "First Subplot", fontsize=10)

x1 = [10,20,30,40]
y1 = ["A", "B", "C", "D"]
plt.subplot( *args: 2,2,2)
plt.pie(x1,labels=y1)
plt.title( label: "second Subplot", fontsize=10)

plt.subplot( *args: 2,2,3)
plt.bar(x,y)
plt.title( label: "Third Subplot", fontsize=10)

plt.subplot( *args: 2,2,4)
plt.scatter(x,y)
plt.title( label: "Fourth Subplot", fontsize=10)

plt.tight_layout()   # Adjust layout to prevent overlap
plt.show()
```

Create Two Subplots (One Below the Other)

```python
import matplotlib.pyplot as plt

# Create the first subplot (top)
plt.subplot( *args: 2, 1, 1)  # 2 rows, 1 column, 1st plot
plt.plot( *args: [1, 2, 3], [4, 5, 6], color='red')  # Line plot
plt.title("First Subplot")

# Create the second subplot (bottom)
plt.subplot( *args: 2, 1, 2)  # 2 rows, 1 column, 2nd plot
plt.bar( x: ["A", "B", "C"],  height: [3, 7, 1], color='blue')  # Bar plot
plt.title("Second Subplot")

# Show the figure
plt.tight_layout()  # Adjust layout to prevent overlap
plt.show()
```