Practical Journal

MATPLOTLIB

1. Create a simple line plot.
   1. Generate x values from 0 to 10 using NumPy.
   2. Create a y variable as the square of x.
   3. Plot y against x, and add titles and labels.

CODE:-

import numpy as np

import matplotlib.pyplot as plt

# 1. Generate x values from 0 to 10

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

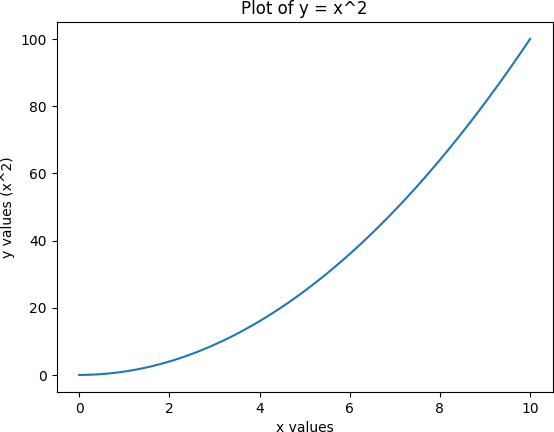
# 2. Create y as the square of x y = x\*\*2

# 3. Plot y against x plt.plot(x, y)

# Adding title and labels plt.title("Plot of y = x^2") plt.xlabel("x values") plt.ylabel("y values (x^2)")

# Show the plot plt.show()

OUTPUT:-



1. Plot multiple lines on the same graph.
   1. Generate x values from 0 to 2π2\pi2π.
   2. Compute sine and cosine for these x values.
   3. Plot both sine and cosine functions on the same graph with different colors and markers.

CODE:-

import numpy as np

import matplotlib.pyplot as plt

# 1. Generate x values from 0 to 2π

x = np.linspace(0, 2 \* np.pi, 100)

# 2. Compute sine and cosine for these x values y\_sin = np.sin(x)

y\_cos = np.cos(x)

# 3. Plot both sine and cosine functions on the same graph

plt.plot(x, y\_sin, label='Sine', color='blue', marker='o') # Sine function with blue color and circle markers

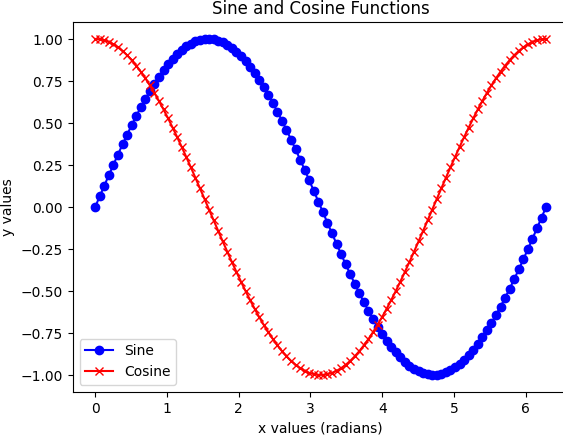
plt.plot(x, y\_cos, label='Cosine', color='red', marker='x') # Cosine function with red color and x markers

# Adding title and labels plt.title("Sine and Cosine Functions") plt.xlabel("x values (radians)") plt.ylabel("y values")

# Add a legend to differentiate the two lines plt.legend()

# Show the plot plt.show()

OUTPUT:-



1. Create a scatter plot and customize it.
   1. Generate two sets of random data (x and y).
   2. Create a scatter plot.
   3. Customize the color, size, and transparency of the points.
   4. Add grid lines and a legend.

CODE:-

import numpy as np

import matplotlib.pyplot as plt

# Generate two sets of random data (x and y)

x = np.random.rand(100) # 100 random points between 0 and 1 y = np.random.rand(100) # 100 random points between 0 and 1

# Create a scatter plot

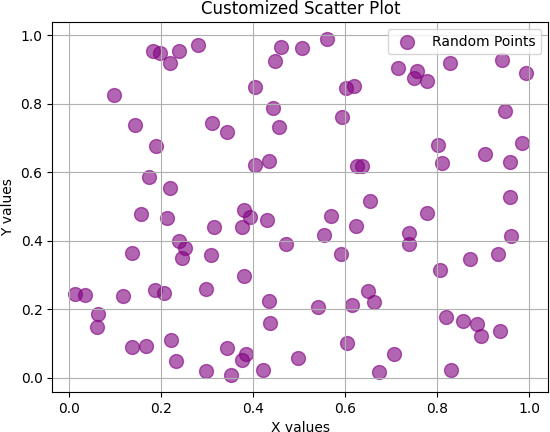
plt.scatter(x, y, label='Random Points', color='purple', s=100, alpha=0.6)

plt.grid(True) # Display grid lines plt.legend() # Show the legend

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

# Show the plot plt.show()

OUTPUT:-



1. Create and customize a histogram.
   1. Generate 1000 random numbers from a normal distribution.
   2. Create a histogram of these numbers.
   3. Adjust the number of bins and customize the color and transparency.

CODE:-

import numpy as np

import matplotlib.pyplot as plt

# Generate 1000 random numbers from a normal distribution data = np.random.normal(loc=0, scale=1, size=1000)

# Create a histogram of these numbers

plt.hist(data, bins=30, color='skyblue', edgecolor='black', alpha=0.7)

# Add title and labels

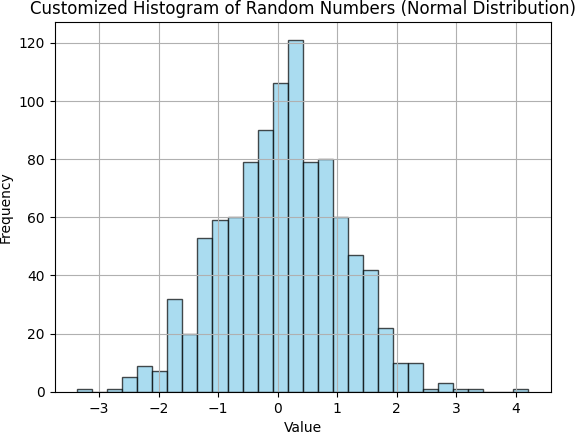
plt.title("Customized Histogram of Random Numbers (Normal Distribution)") plt.xlabel("Value")

plt.ylabel("Frequency")

# Show grid lines for better visibility plt.grid(True)

# Show the plot plt.show()

OUTPUT:-



1. Create a bar chart.
   1. Define a category list (e.g., 'A', 'B', 'C', 'D') and corresponding values.
   2. Create a bar chart to visualize these values.
   3. Add titles, labels, and customize the bar colors.

CODE:-

import matplotlib.pyplot as plt

# 1. Define the category list and corresponding values categories = ['A', 'B', 'C', 'D']

values = [23, 17, 35, 45]

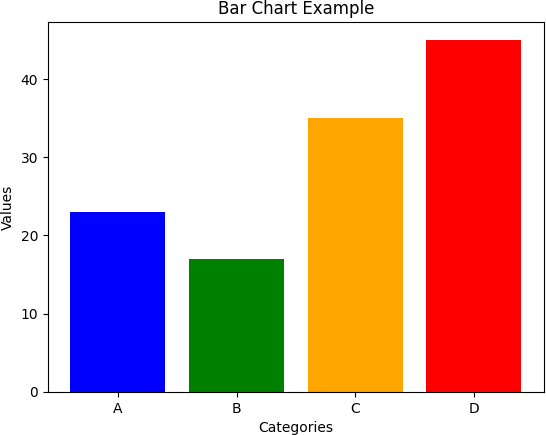
# 2. Create a bar chart to visualize these values plt.bar(categories, values, color=['blue', 'green', 'orange', 'red'])

# 3. Add titles, labels, and customize the bar colors plt.title("Bar Chart Example") plt.xlabel("Categories")

plt.ylabel("Values")

# Display the plot plt.show()

OUTPUT:-



1. Use subplots to display multiple plots in one figure.
   1. Create a 2x2 grid of subplots.
   2. Place a line plot, scatter plot, histogram, and bar chart in each subplot.
   3. Add individual titles and labels to each subplot.

CODE:-

import numpy as np

import matplotlib.pyplot as plt

# 1. Create a 2x2 grid of subplots

fig, axs = plt.subplots(2, 2, figsize=(10, 8))

# Data for each plot

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

x\_scatter = np.random.rand(50) y\_scatter = np.random.rand(50)

data\_hist = np.random.randn(1000) categories = ['A', 'B', 'C', 'D'] values = [23, 17, 35, 45]

# 2. Place a line plot, scatter plot, histogram, and bar chart in each subplot # Line plot in the top-left

axs[0, 0].plot(x, y, label='y = sin(x)', color='blue') axs[0, 0].set\_title("Line Plot")

axs[0, 0].set\_xlabel("X")

axs[0, 0].set\_ylabel("Y")

axs[0, 0].legend()

# Scatter plot in the top-right

axs[0, 1].scatter(x\_scatter, y\_scatter, color='green', label='Random Points') axs[0, 1].set\_title("Scatter Plot")

axs[0, 1].set\_xlabel("X values") axs[0, 1].set\_ylabel("Y values") axs[0, 1].legend()

# Histogram in the bottom-left

axs[1, 0].hist(data\_hist, bins=30, color='purple', edgecolor='black', alpha=0.7) axs[1, 0].set\_title("Histogram")

axs[1, 0].set\_xlabel("Value") axs[1, 0].set\_ylabel("Frequency")

# Bar chart in the bottom-right

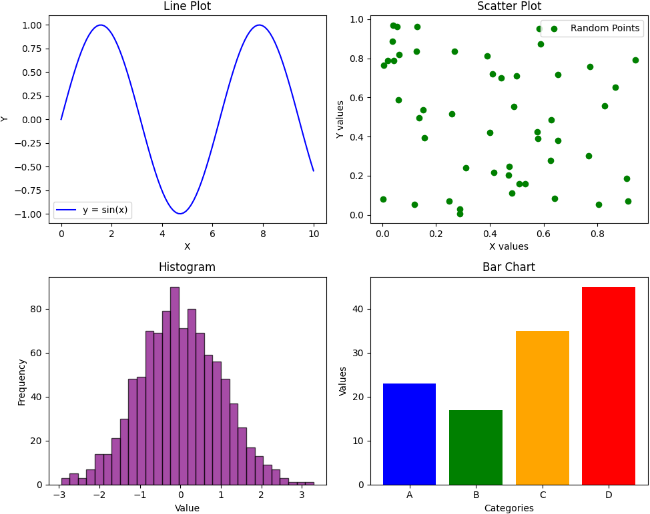
axs[1, 1].bar(categories, values, color=['blue', 'green', 'orange', 'red']) axs[1, 1].set\_title("Bar Chart")

axs[1, 1].set\_xlabel("Categories") axs[1, 1].set\_ylabel("Values")

# 3. Adjust the layout for better spacing plt.tight\_layout()

# Show the plot

plt.show() OUTPUT:-



1. Create a pie chart to represent categorical data.
2. Define a list of categories and their corresponding values.
3. Create a pie chart to visualize this data.
4. Add a title and display the percentage of each category.

CODE:-

import matplotlib.pyplot as plt

# a. Define a list of categories and their corresponding values categories = ['Category A', 'Category B', 'Category C', 'Category D'] values = [30, 15, 40, 15]

# b. Create a pie chart to visualize this data

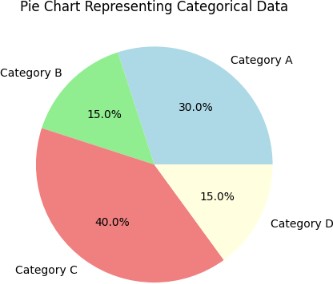
plt.pie(values, labels=categories, autopct='%1.1f%%', colors=['lightblue', 'lightgreen', 'lightcoral', 'lightyellow'])

# c. Add a title

plt.title("Pie Chart Representing Categorical Data")

# Display the pie chart plt.show()

OUTPUT:-



1. Combine different types of plots in one figure.
2. Create a line plot and a bar chart on the same axes.
3. Use twin axes to represent different y-values on the same x-axis.
4. Customize the appearance of both plots for clarity.

CODE:-

import numpy as np

import matplotlib.pyplot as plt

# 1. Create data for the line plot and the bar chart x = np.linspace(0, 10, 10)

y\_line = np.sin(x) # Line plot: sine of x

y\_bar = np.abs(np.cos(x) \* 10) # Bar chart: absolute value of cos(x) scaled by 10

# 2. Create a figure and a set of axes fig, ax1 = plt.subplots(figsize=(10, 6))

# 3. Create a line plot on the first y-axis

ax1.plot(x, y\_line, 'b-', label='Line plot (sin(x))', linewidth=2) ax1.set\_xlabel("X values")

ax1.set\_ylabel("Y values (sin(x))", color='blue') ax1.tick\_params(axis='y', labelcolor='blue')

# 4. Create twin axes for the second y-values (Bar chart)

ax2 = ax1.twinx() # Share the same x-axis, but a different y-axis

ax2.bar(x, y\_bar, alpha=0.6, color='orange', label='Bar chart (abs(cos(x)) \* 10)') ax2.set\_ylabel("Y values (abs(cos(x)) \* 10)", color='orange') ax2.tick\_params(axis='y', labelcolor='orange')

# 5. Add titles and a legend

plt.title("Combined Line Plot and Bar Chart with Twin Axes")

# Add legends ax1.legend(loc='upper left') ax2.legend(loc='upper right')

# 6. Show the plot

plt.tight\_layout() # Adjust layout for better spacing plt.show()

OUTPUT:-

