```
pip install matplotlib
```

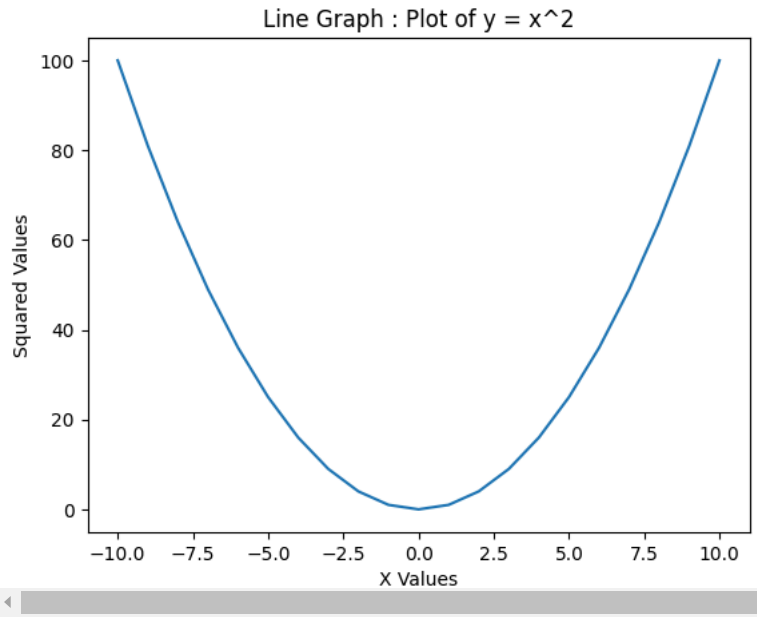
```
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (3.8.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (4.55.3)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.4.7)
Requirement already satisfied: numpy<2,>=1.21 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.26.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (11.0.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (3.2.0)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib) (1.17.0)
```

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

1. Basic Line Plot

- Plot a simple line graph of $y = x2$ for x values ranging from -10 to 10.
- Label the x-axis as "X Values" and the y-axis as "Squared Values."
- Add a title to the graph.

```
# Generate x values from -10 to 10
x = np.arange(-10, 11, 1)
# Compute y values
y = x ** 2
# Create the plot
plt.plot(x,y)
# Label the axes
plt.xlabel('X Values')
plt.ylabel('Squared Values')
# Add a title
plt.title('Line Graph : Plot of y = x^2')
# Display the plot
plt.show()
```



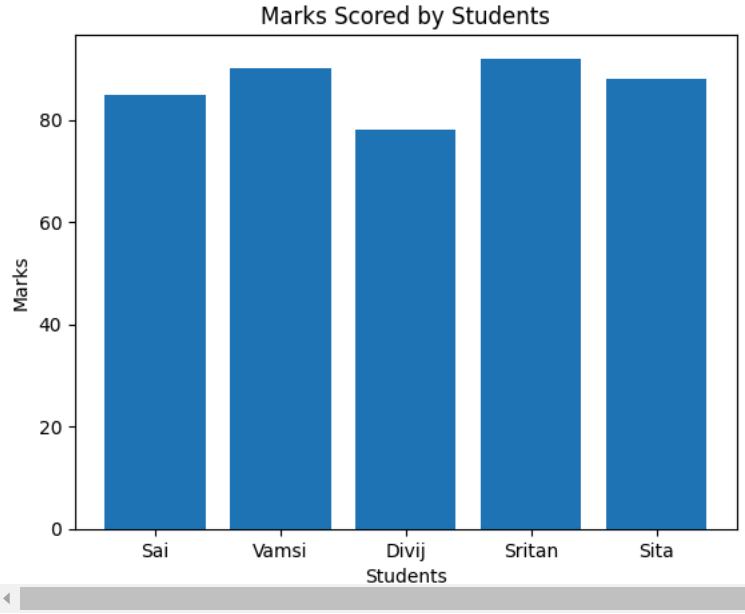
2. Bar Plot

- Create a bar chart showing the marks scored by 5 students:

  [Sai, 'Vamsi', 'Divij', 'Sritan', 'Sita'] with marks [85, 90, 78, 92, 88].
- Add labels to the x-axis and y-axis and give the chart a title.

```
# Given Data
students = ['Sai', 'Vamsi', 'Divij', 'Sritan', 'Sita']
marks = [85, 90, 78, 92, 88]
# Create the bar chart
plt.bar(students, marks)
# Label the axes
```

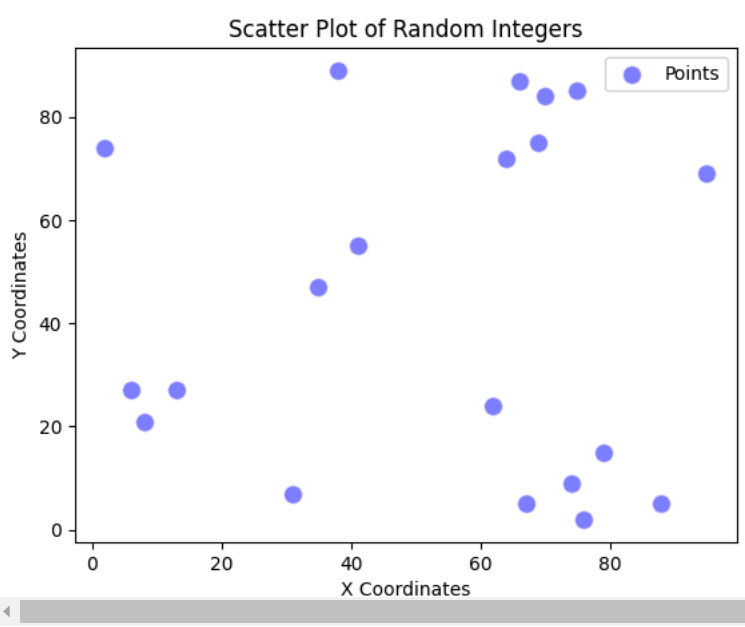
```
plt.xlabel('Students')
plt.ylabel('Marks')
# Add a title
plt.title('Marks Scored by Students')
# Display the chart
plt.show()
```



3. Scatter Plot

- Generate two lists of 20 random integers each (for x and y coordinates).
- Create a scatter plot with these values, and customize it by adding color, size, and labels to the points.

```
# Generate two lists of 20 random integers each
x = np.random.randint(1, 100, 20)
y = np.random.randint(1, 100, 20)
# Create the scatter plot
plt.scatter(x, y, c='blue', s=100, alpha=0.5, edgecolors='w', label='Points')
# Customize the plot
plt.xlabel('X Coordinates')
plt.ylabel('Y Coordinates')
plt.title('Scatter Plot of Random Integers')
plt.legend()
# Display the plot
plt.show()
```

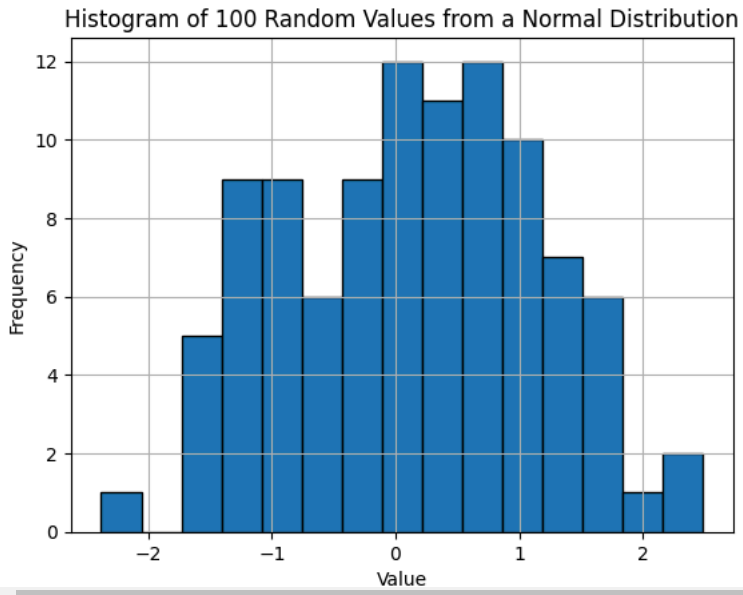


4. Histogram

- Simulate 100 random values from a normal distribution using np.random.randn() and plot a histogram.

- Customize the number of bins and add a grid to the plot.

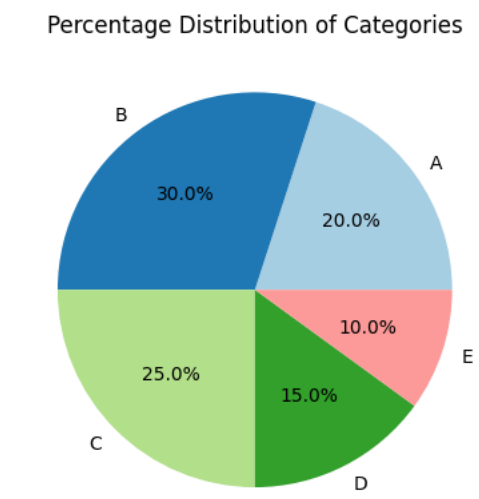
```
# Simulate 100 random values from a normal distribution
data = np.random.randn(100)
# Create the histogram
plt.hist(data, bins=15, edgecolor='black')
# Customize the plot
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Histogram of 100 Random Values from a Normal Distribution')
plt.grid(True)
# Display the plot
plt.show()
```



5. Pie Chart

- Visualize the percentage distribution of five categories ['A', 'B', 'C', 'D', 'E'] with values [20, 30, 25, 15, 10].
- Display the percentage values on the chart and use a color palette for differentiation

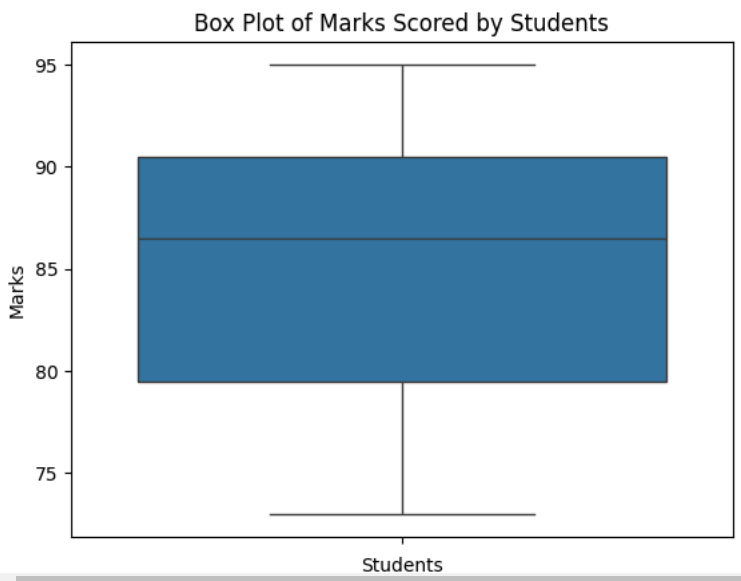
```
# given Data
categories = ['A', 'B', 'C', 'D', 'E']
values = [20, 30, 25, 15, 10]
# Creating a pie chart
plt.pie(values, labels=categories, autopct='%1.1f%%', colors=plt.cm.Paired.colors)
# Adding  title
plt.title('Percentage Distribution of Categories')
# Display the chart
plt.show()
```



6. Box Plot (Seaborn)

- Create a box plot to visualize the distribution of the marks scored by students in a class: [78, 85, 92, 88, 76, 95, 89, 84, 91, 73].
- Highlight the median and any outliers.

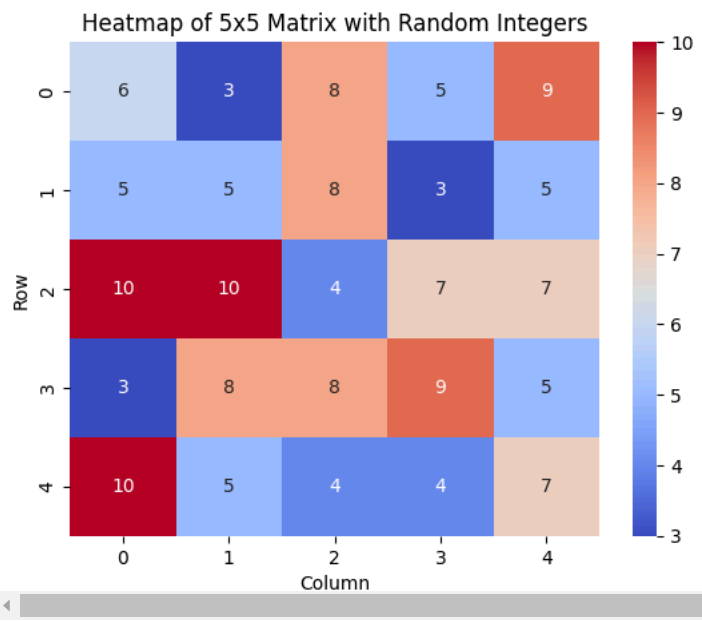
```
# Data
marks = [78, 85, 92, 88, 76, 95, 89, 84, 91, 73]
# Create the box plot
sns.boxplot(data=marks)
# Customize the plot
plt.xlabel('Students')
plt.ylabel('Marks')
plt.title('Box Plot of Marks Scored by Students')
# Display the plot
plt.show()
```



7. Heatmap (Seaborn)

- Create a 5x5 matrix with random integers between 1 and 10 using np.random.randint() and visualize it as a heatmap using Seaborn.
- Add a color bar and annotate the heatmap with the values.

```
# Create a 5x5 matrix with random integers between 1 and 10
matrix = np.random.randint(1, 11, size=(5, 5))
# Create the heatmap with a different color map
sns.heatmap(matrix, annot=True, cmap='coolwarm', cbar=True)
# Customize the plot
plt.title('Heatmap of 5x5 Matrix with Random Integers')
plt.xlabel('Column')
plt.ylabel('Row')
# Display the heatmap
plt.show()
```

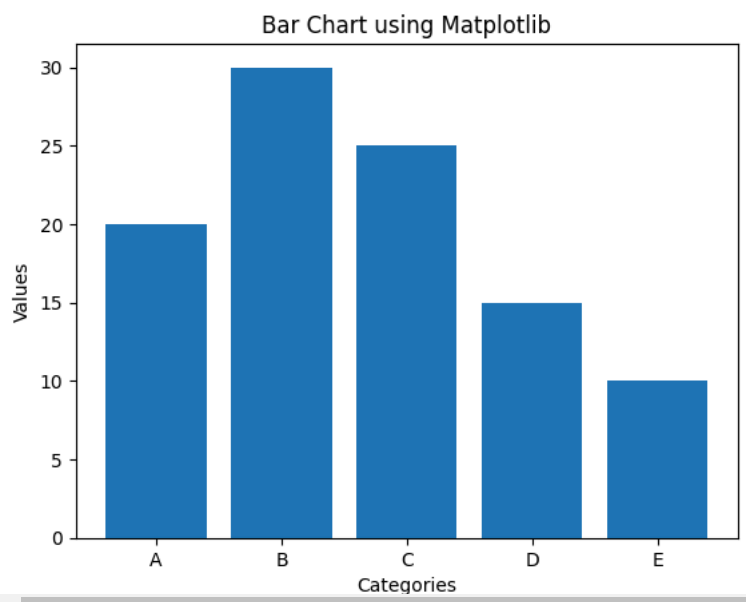


8. Comparison of Matplotlib and Seaborn

- Generate a pair of plots (bar charts or line plots) using the same data in both Matplotlib and Seaborn.
- Highlight the differences in the code, syntax, and aesthetics of the two libraries.

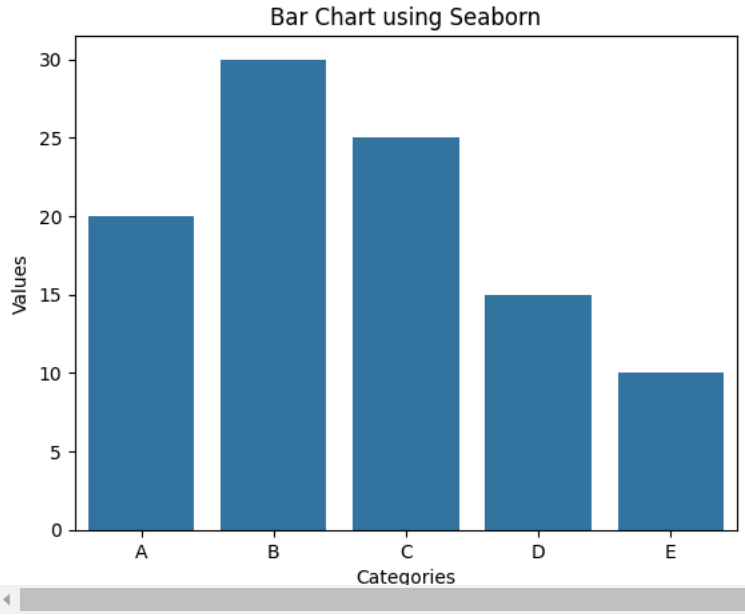
Matplotlib

```
import matplotlib.pyplot as plt
# Data
categories = ['A', 'B', 'C', 'D', 'E']
values = [20, 30, 25, 15, 10]
# Create the bar chart
plt.bar(categories, values)
# Customize the plot
plt.xlabel('Categories')
plt.ylabel('Values')
plt.title('Bar Chart using Matplotlib')
# Display the plot
plt.show()
```



Seaborn

```
import seaborn as sns
import pandas as pd
# Data
categories = ['A', 'B', 'C', 'D', 'E']
values = [20, 30, 25, 15, 10]
# Create a DataFrame for Seaborn
data = pd.DataFrame({'Categories': categories, 'Values': values})
# Create the bar chart using Seaborn
sns.barplot(x='Categories', y='Values', data=data)
# Customize the plot
plt.xlabel('Categories')
plt.ylabel('Values')
plt.title('Bar Chart using Seaborn')
# Display the plot
plt.show()
```



Differences in Code, Syntax, and Aesthetics

Code and Syntax:

Matplotlib: The code is more straightforward and requires fewer imports. You directly use the plt.bar() function to create the bar chart.

Seaborn: You need to create a DataFrame to use Seaborn effectively. The sns.barplot() function is used, and you pass the DataFrame along with the column names for the x and y axes.

Aesthetics:

Matplotlib: The default aesthetics are simpler and more basic. Customization requires more manual adjustments.

Seaborn: The default aesthetics are more polished and visually appealing. Seaborn automatically applies a more sophisticated style to the plots.

Customization:

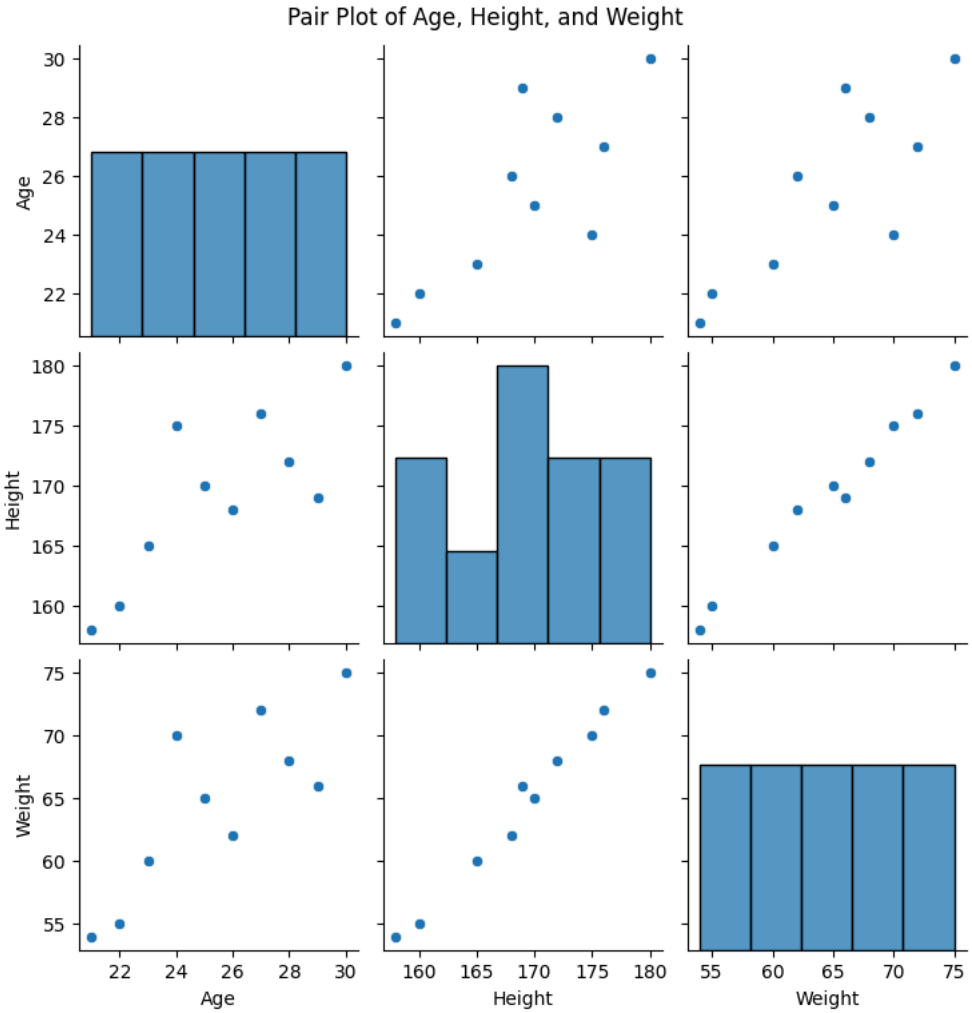
Matplotlib: Offers extensive customization options, but you need to manually set many of the properties.

Seaborn: Provides high-level interfaces for drawing attractive statistical graphics, making it easier to create complex visualizations with less code.

9. Pair Plot (Seaborn)

- Manually create a small dataset with columns such as 'Age', 'Height', 'Weight' for 10 individuals and visualize the relationships using a pair plot.

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
# Create a small dataset
data = {
    'Age': [23, 25, 22, 24, 26, 28, 30, 21, 27, 29],
    'Height': [165, 170, 160, 175, 168, 172, 180, 158, 176, 169],
    'Weight': [60, 65, 55, 70, 62, 68, 75, 54, 72, 66]
}
# Create DataFrame
df = pd.DataFrame(data)
# Create the pair plot
sns.pairplot(df)
# Customize the plot
plt.suptitle('Pair Plot of Age, Height, and Weight', y=1.02)
# Display the plot
plt.show()
```

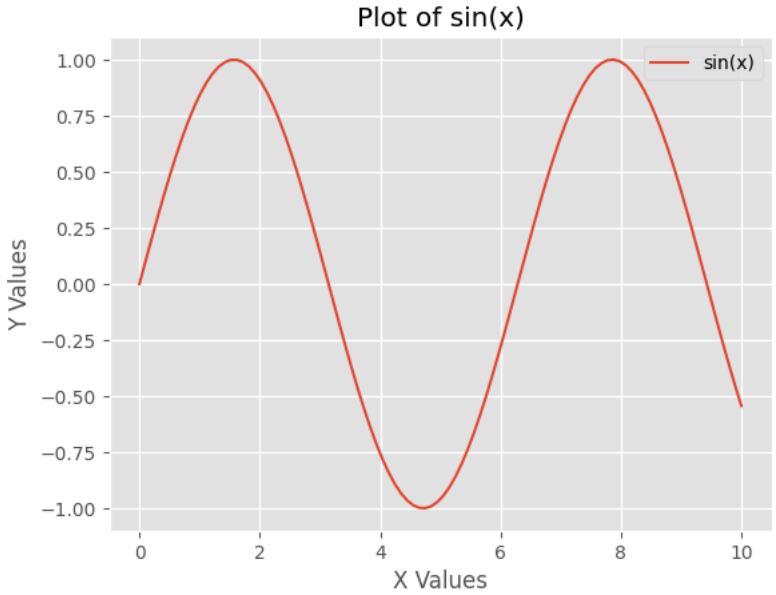


10. Save and Style Plots

• Plot any graph of your choice and save it in both .png and .pdf formats.

• Apply at least one built-in style (e.g., 'ggplot' or 'seaborn-darkgrid') and explain how the style impacts the visualization.

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

# Apply the 'ggplot' style
plt.style.use('ggplot')

# Generate x values
x = np.linspace(0, 10, 100)

# Compute y values
y = np.sin(x)

# Create the plot
plt.plot(x, y, label='sin(x)')

# Customize the plot
plt.xlabel('X Values')
plt.ylabel('Y Values')
plt.title('Plot of sin(x)')
plt.legend()

# Save the plot in .png and .pdf formats
plt.savefig('plot.png')
plt.savefig('plot.pdf')

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



Explanation of the 'ggplot' Style: