Matplotlib Cheat Sheet

This cheat sheet covers the essential and advanced features of Matplotlib, a comprehensive library for creating static, animated, and interactive visualizations in Python.

1. Introduction to Matplotlib

1.1 What is Matplotlib?

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications.

1.2 Installing Matplotlib

You can install Matplotlib using pip:

```
pip install matplotlib
```

Or, if you are using Anaconda:

```
conda install matplotlib
```

1.3 Importing Matplotlib

To use Matplotlib, you need to import it in your script:

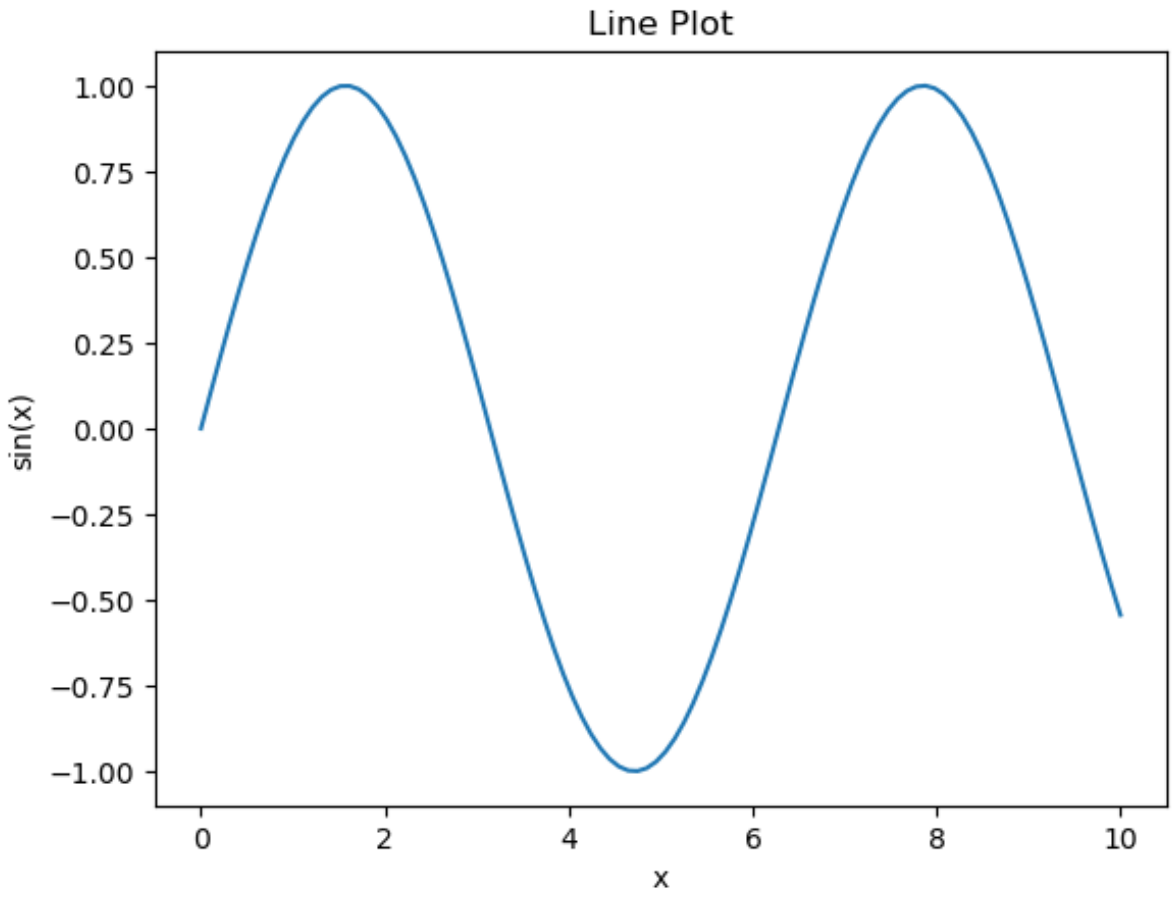
In [1]:
```
import matplotlib.pyplot as plt
# Now you can use Matplotlib functions with the plt prefix
```

2. Basic Plotting

2.1 Line Plot

Line plots are used to represent data points connected by straight lines.

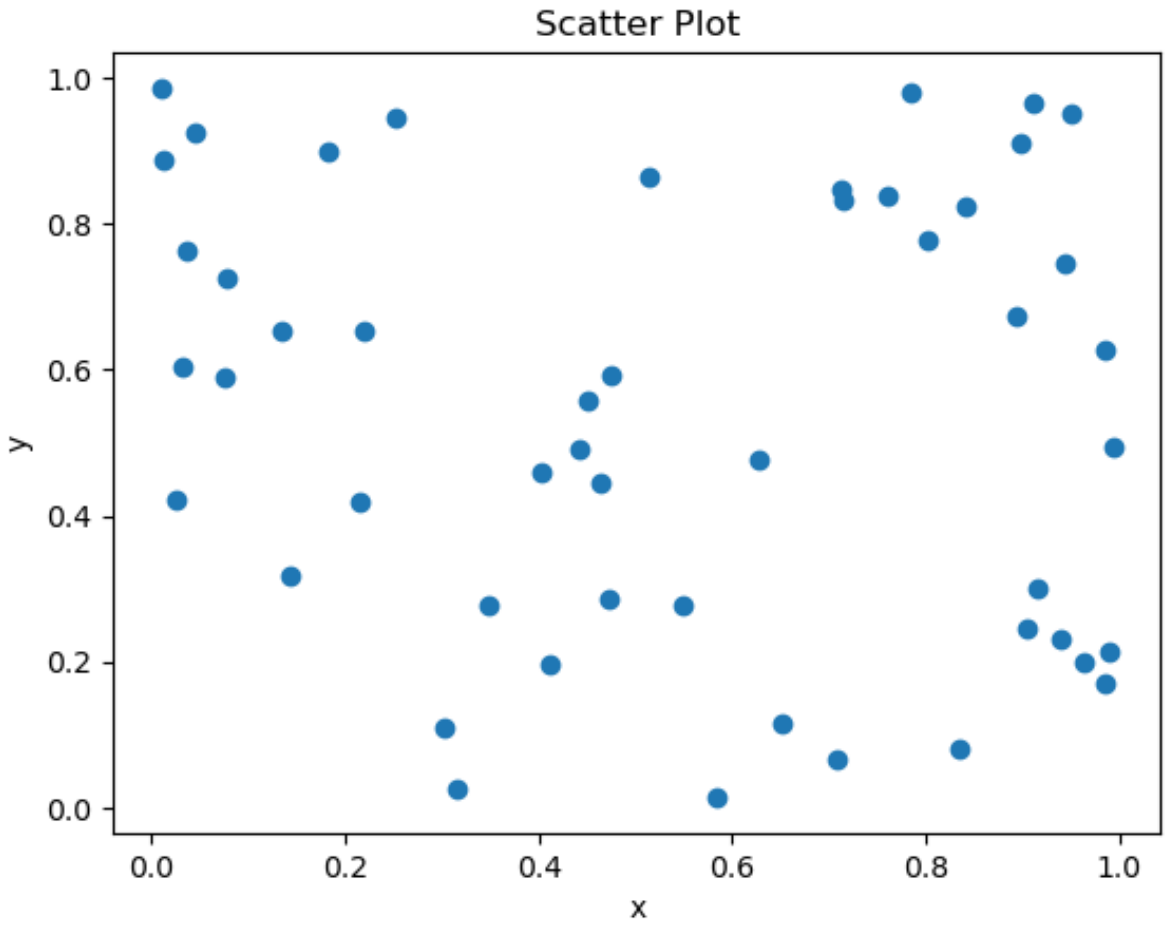
In [2]:
```
import numpy as np
x = np.linspace(0, 10, 100)
y = np.sin(x)
plt.plot(x, y)
plt.title('Line Plot')
plt.xlabel('x')
plt.ylabel('sin(x)')
plt.show()
```



2.2 Scatter Plot

Scatter plots are used to represent individual data points.

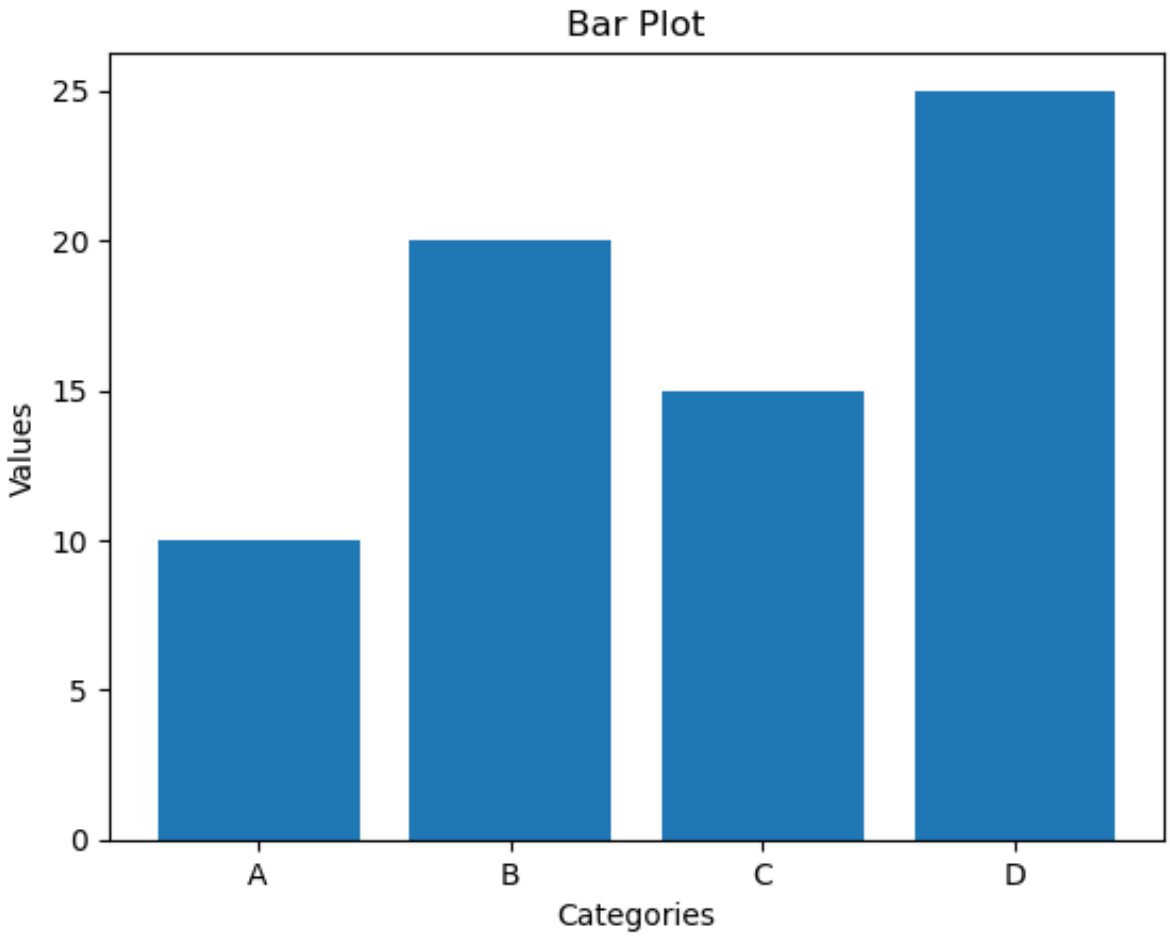
In [3]:
```
x = np.random.rand(50)
y = np.random.rand(50)
plt.scatter(x, y)
plt.title('Scatter Plot')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



2.3 Bar Plot

Bar plots are used to represent data with rectangular bars.

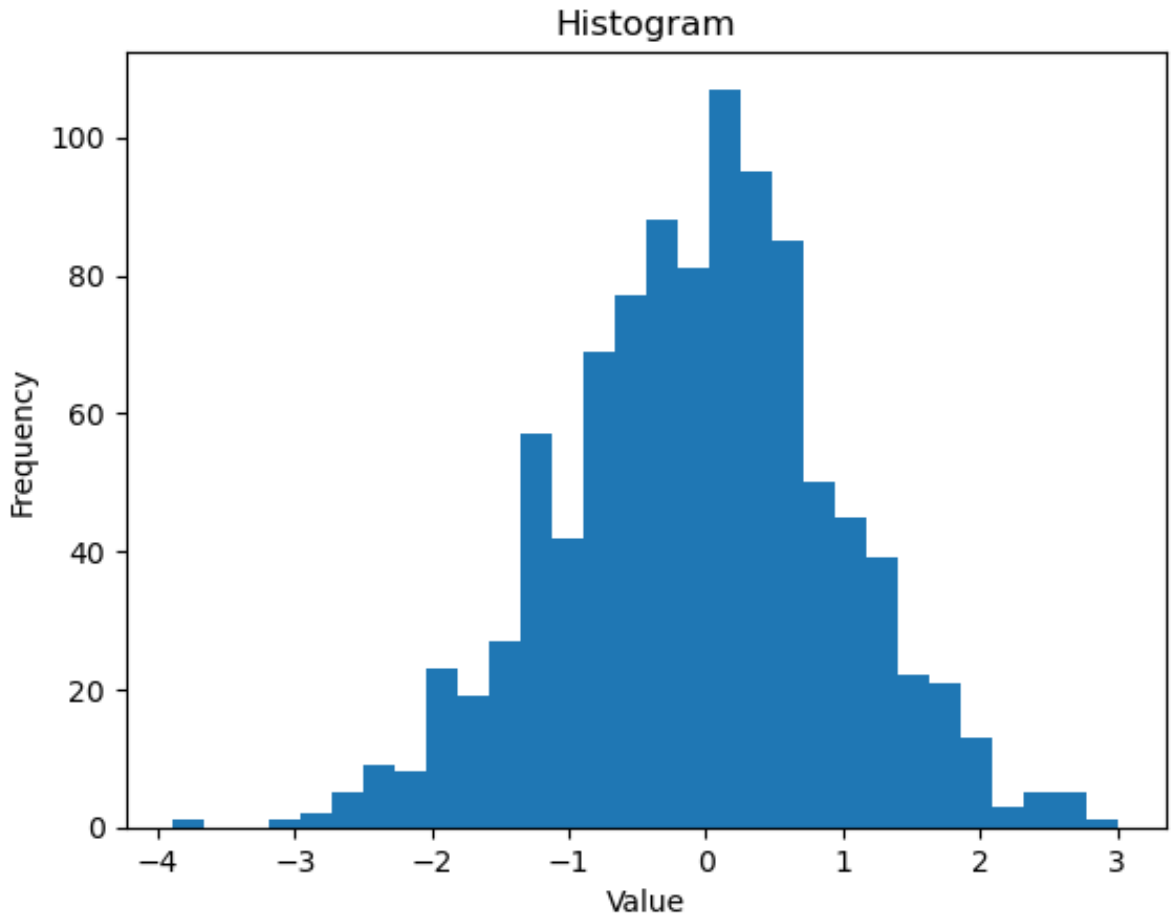
In [4]:
```
categories = ['A', 'B', 'C', 'D']
values = [10, 20, 15, 25]
plt.bar(categories, values)
plt.title('Bar Plot')
plt.xlabel('Categories')
plt.ylabel('Values')
plt.show()
```



2.4 Histogram

Histograms are used to represent the distribution of a dataset.

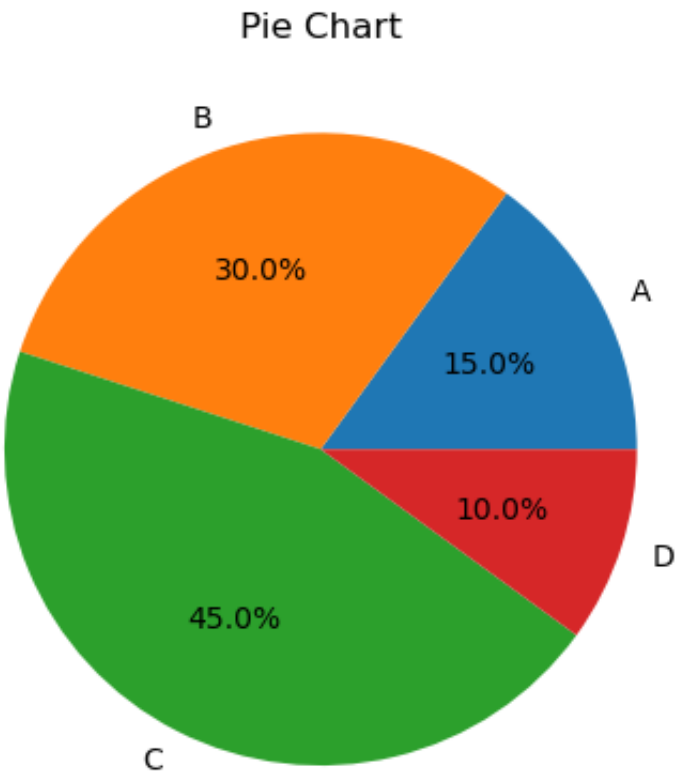
In [5]:
```
data = np.random.randn(1000)
plt.hist(data, bins=30)
plt.title('Histogram')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.show()
```



2.5 Pie Chart

Pie charts are used to represent data as slices of a pie.

In [6]:
```
sizes = [15, 30, 45, 10]
labels = ['A', 'B', 'C', 'D']
plt.pie(sizes, labels=labels, autopct='%1.1f%%')
plt.title('Pie Chart')
plt.show()
```

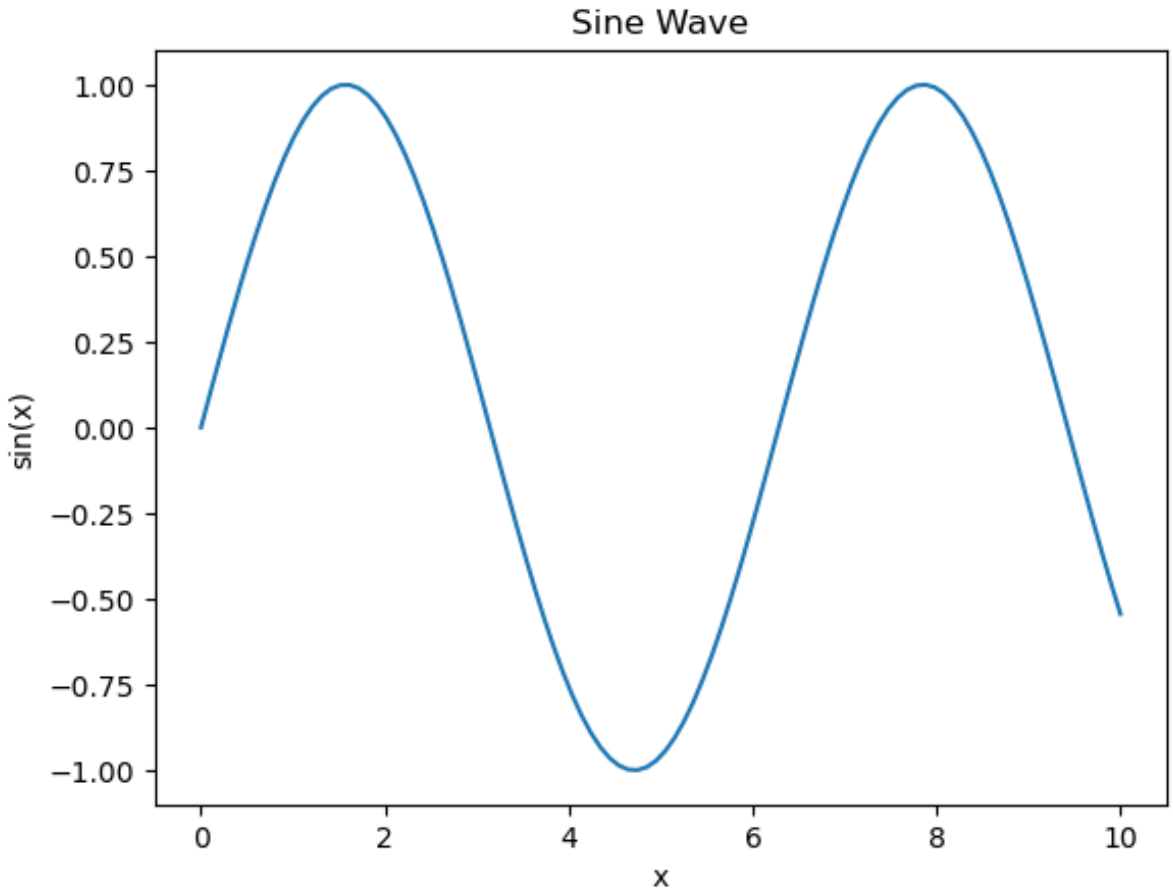


3. Customizing Plots

3.1 Titles and Labels

You can add titles and labels to your plots to make them more informative.

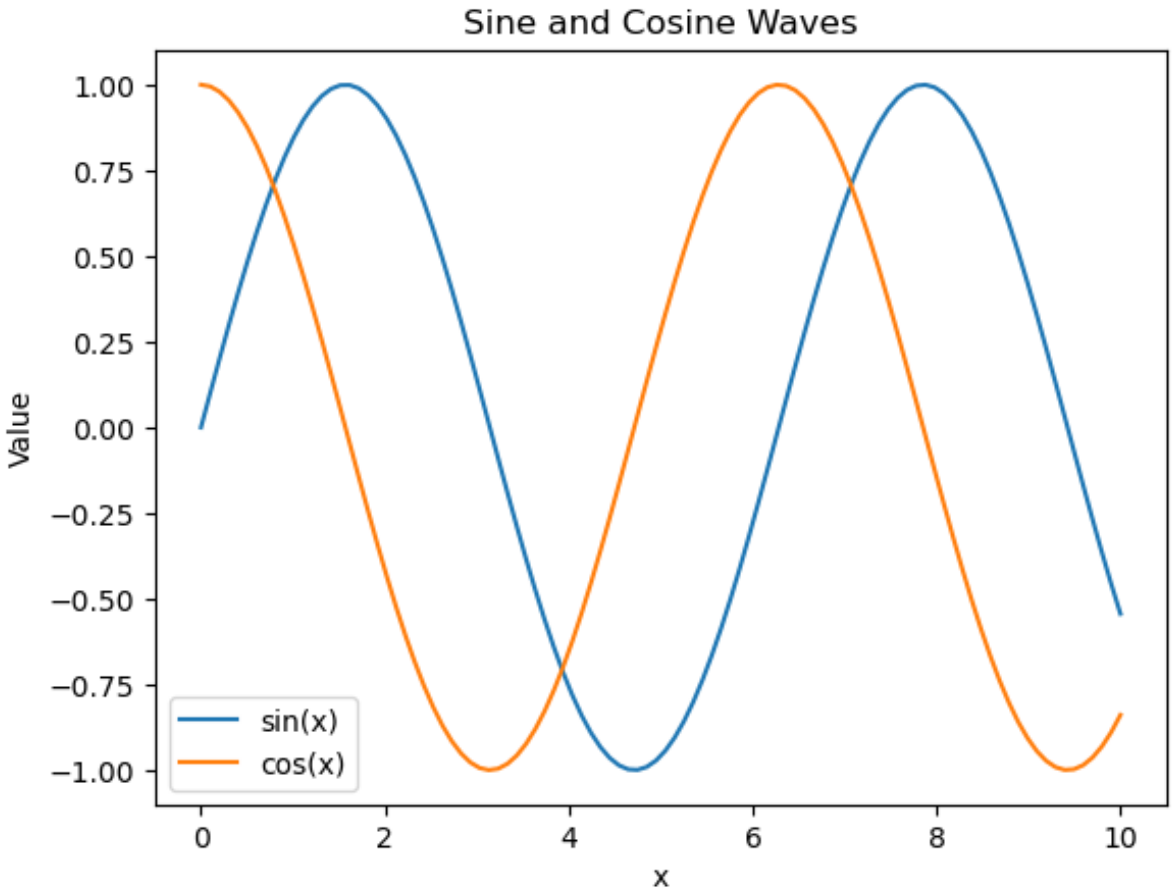
In [7]:
```
x = np.linspace(0, 10, 100)
y = np.sin(x)
plt.plot(x, y)
plt.title('Sine Wave')
plt.xlabel('x')
plt.ylabel('sin(x)')
plt.show()
```



3.2 Legends

Legends are used to label different elements of a plot.

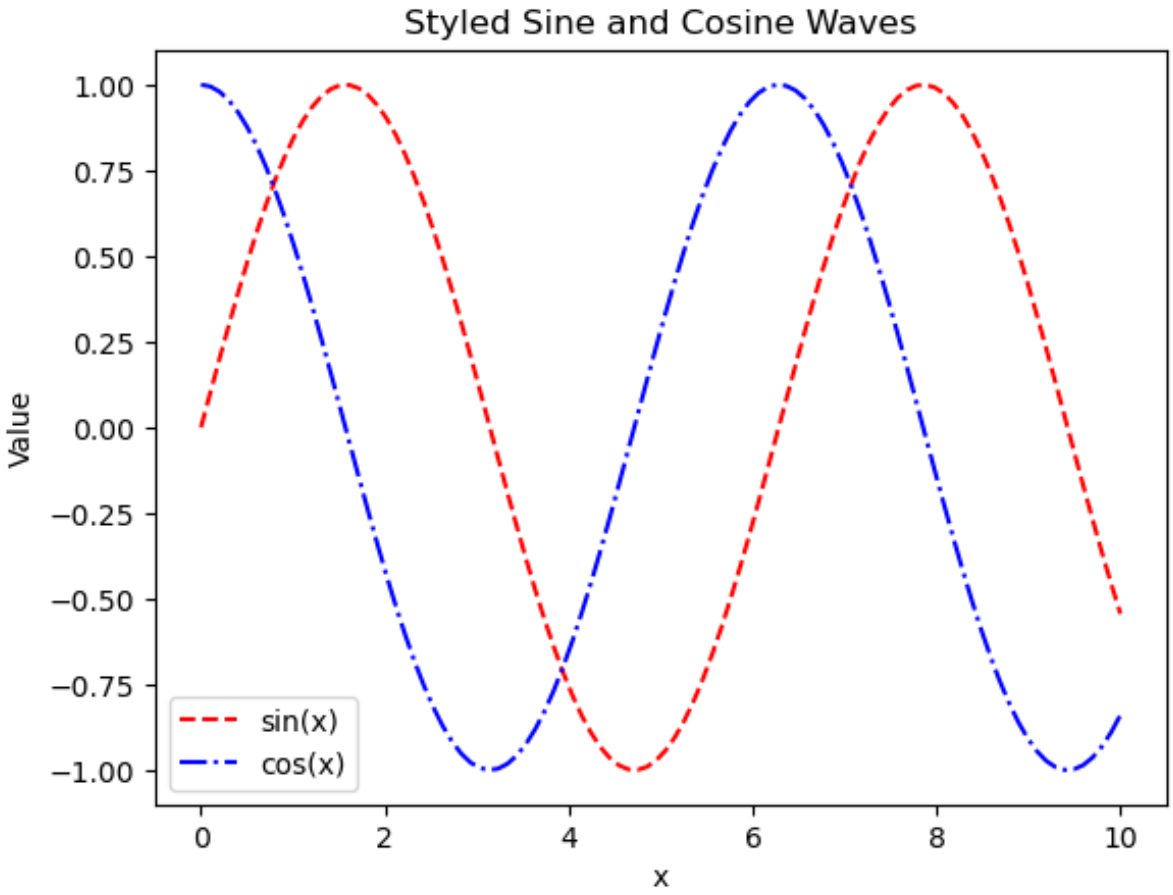
In [8]:
```
plt.plot(x, y, label='sin(x)')
plt.plot(x, np.cos(x), label='cos(x)')
plt.title('Sine and Cosine Waves')
plt.xlabel('x')
plt.ylabel('Value')
plt.legend()
plt.show()
```



3.3 Colors and Styles

You can customize the colors and styles of your plots.

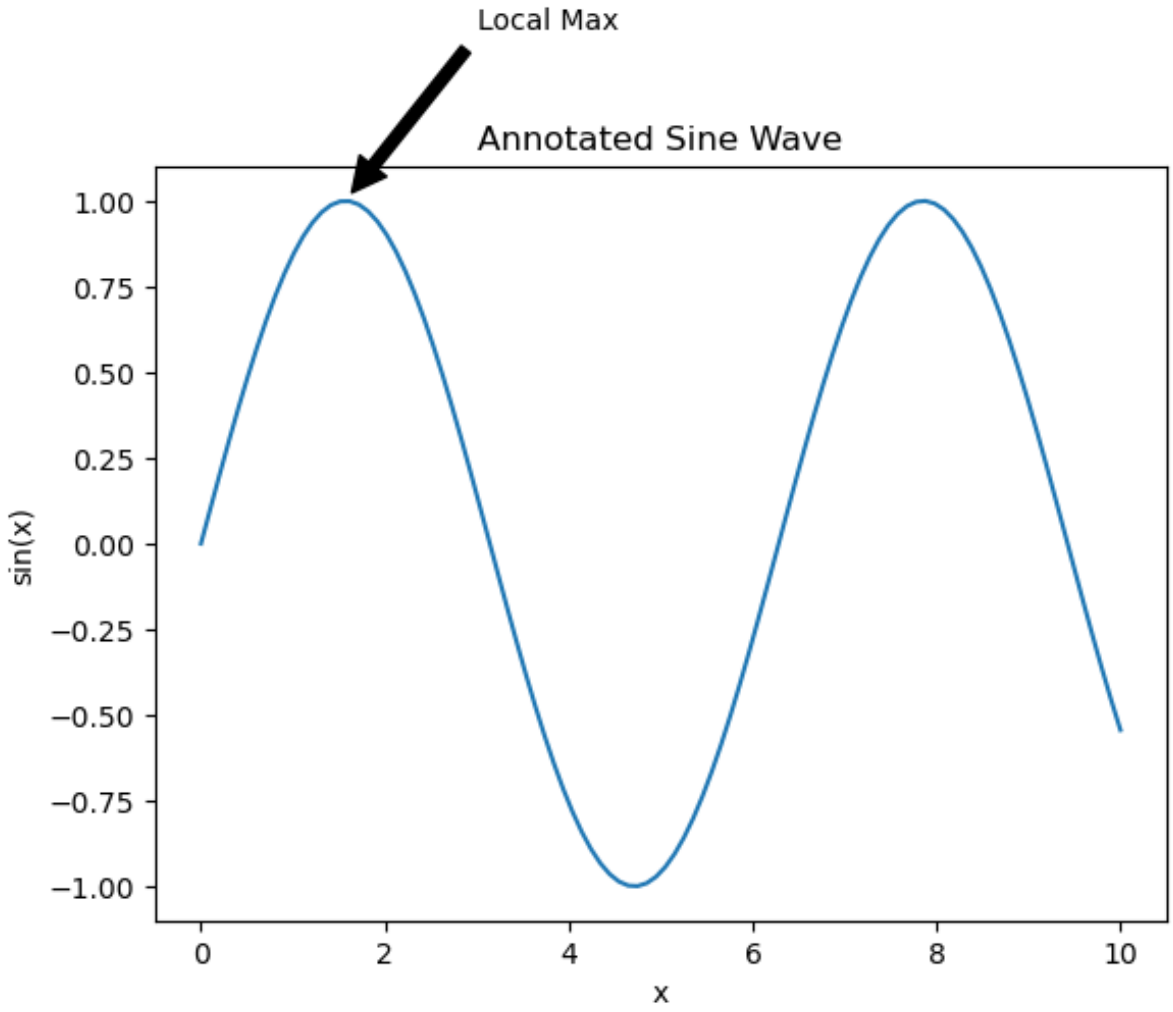
In [9]:
```
plt.plot(x, y, 'r--', label='sin(x)')
plt.plot(x, np.cos(x), 'b-.', label='cos(x)')
plt.title('Styled Sine and Cosine Waves')
plt.xlabel('x')
plt.ylabel('Value')
plt.legend()
plt.show()
```



3.4 Annotations

Annotations are used to highlight specific points on a plot.

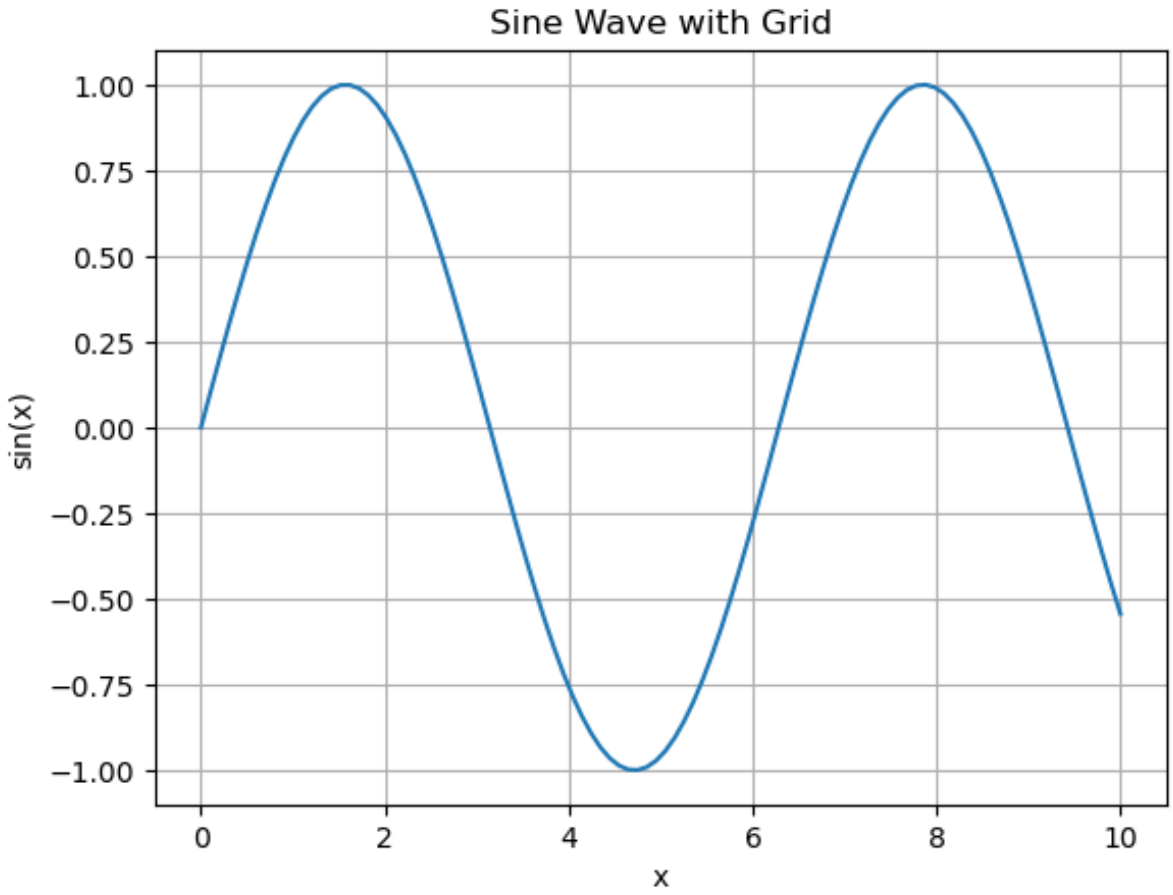
In [10]:
```
plt.plot(x, y)
plt.title('Annotated Sine Wave')
plt.xlabel('x')
plt.ylabel('sin(x)')
plt.annotate('Local Max', xy=(1.57, 1), xytext=(3, 1.5),
             arrowprops=dict(facecolor='black', shrink=0.05))
plt.show()
```



3.5 Grid and Axes

You can customize the grid and axes of your plots.

In [11]:
```
plt.plot(x, y)
plt.title('Sine Wave with Grid')
plt.xlabel('x')
plt.ylabel('sin(x)')
plt.grid(True)
plt.show()
```

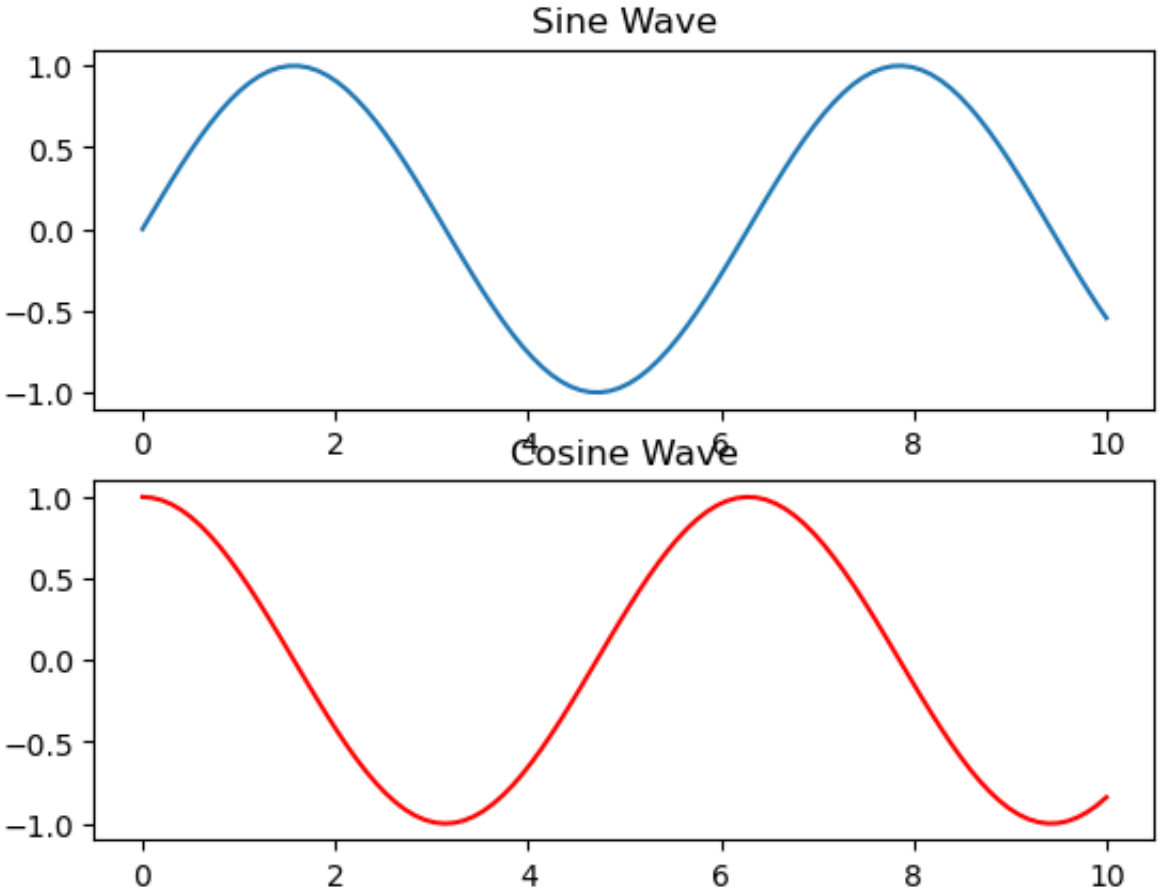


4. Advanced Plotting Techniques

4.1 Subplots

You can create multiple plots in a single figure using subplots.

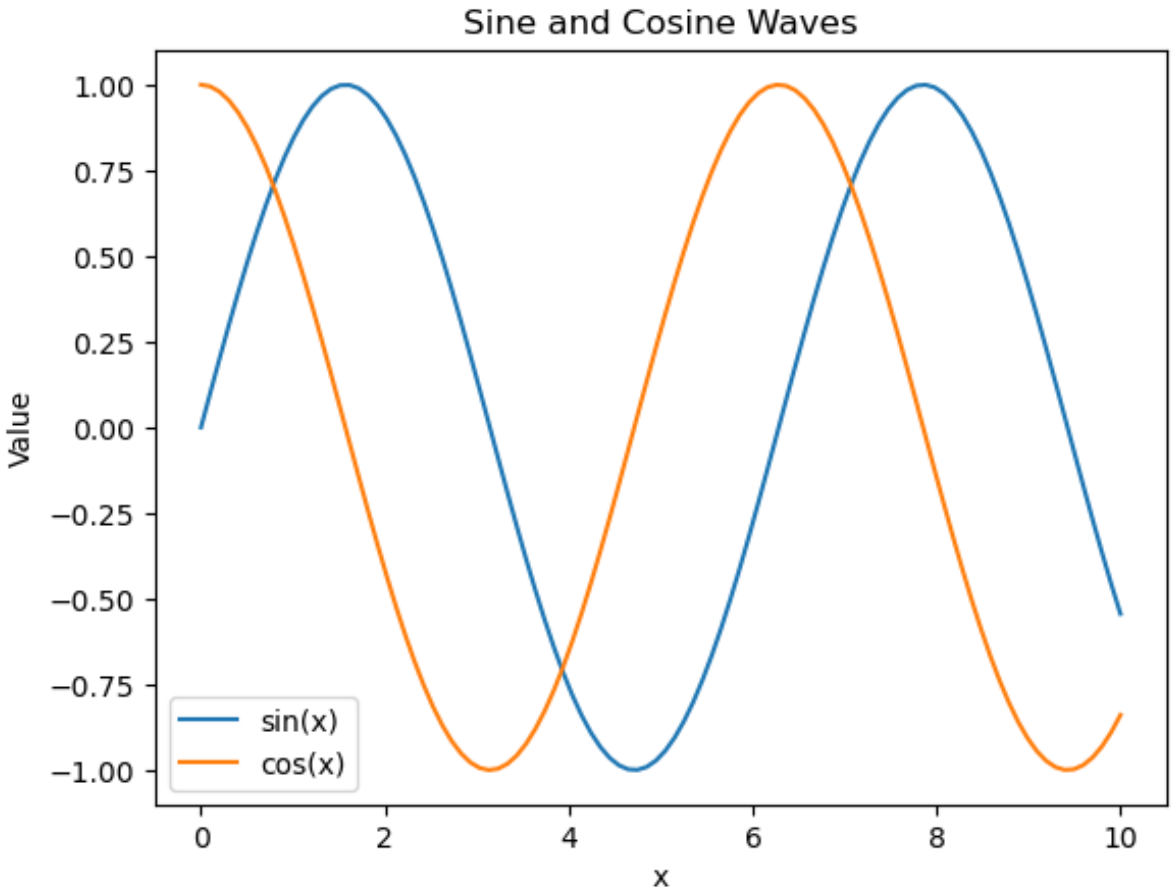
In [12]:
```
fig, axs = plt.subplots(2)
axs[0].plot(x, y)
axs[0].set_title('Sine Wave')
axs[1].plot(x, np.cos(x), 'r')
axs[1].set_title('Cosine Wave')
plt.show()
```



4.2 Multiple Plots

You can overlay multiple plots on the same axes.

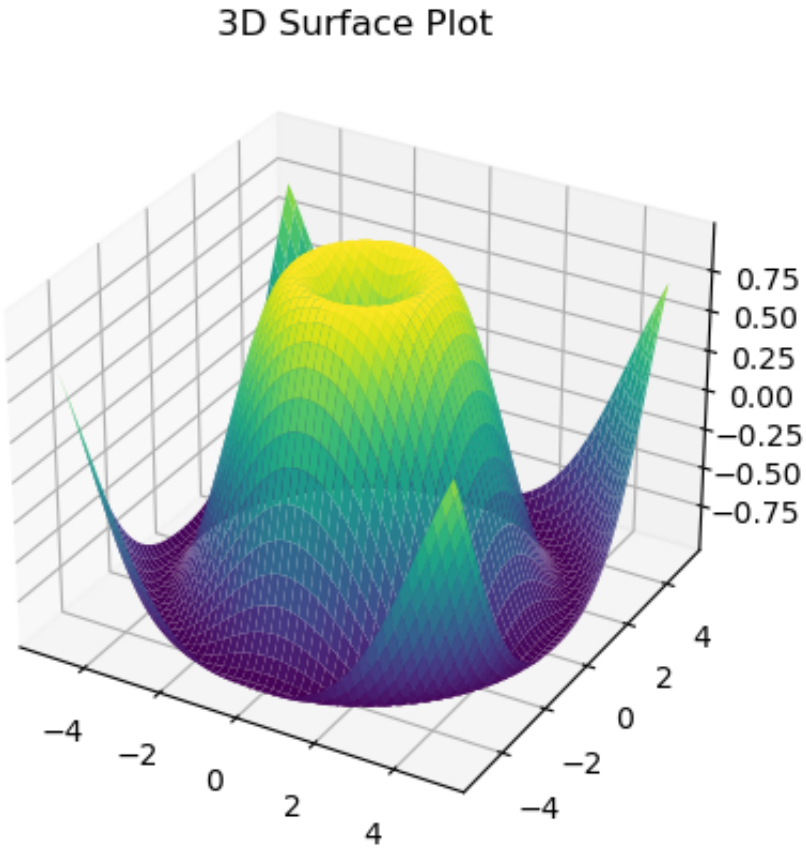
In [13]:
```
plt.plot(x, y, label='sin(x)')
plt.plot(x, np.cos(x), label='cos(x)')
plt.title('Sine and Cosine Waves')
plt.xlabel('x')
plt.ylabel('Value')
plt.legend()
plt.show()
```



4.3 3D Plotting

You can create 3D plots using Matplotlib's `mplot3d` toolkit.

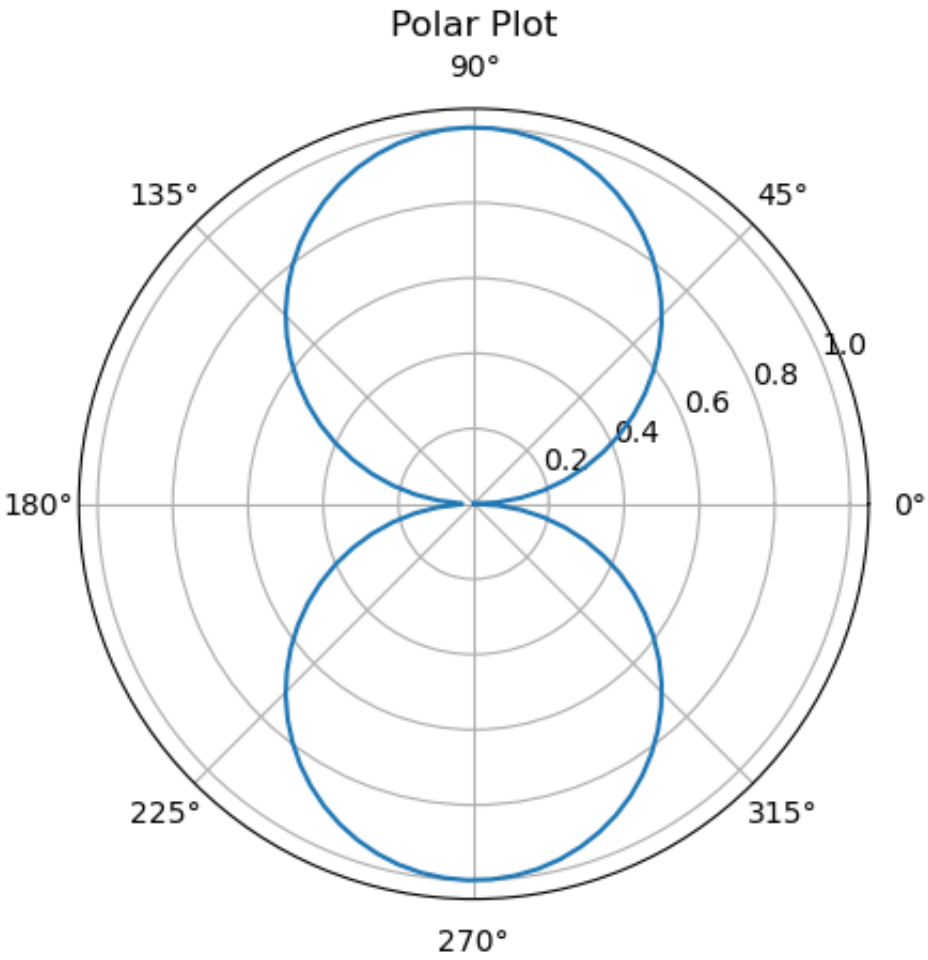
In [14]:
```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)
X, Y = np.meshgrid(x, y)
Z = np.sin(np.sqrt(X**2 + Y**2))
ax.plot_surface(X, Y, Z, cmap='viridis')
plt.title('3D Surface Plot')
plt.show()
```



4.4 Polar Plots

Polar plots are used to represent data in a circular format.

In [15]:
```
theta = np.linspace(0, 2*np.pi, 100)
r = np.abs(np.sin(theta))
plt.polar(theta, r)
plt.title('Polar Plot')
plt.show()
```



5. Working with Images

5.1 Displaying Images

You can display images using Matplotlib's `imshow` function.

In [16]:
```
import matplotlib.pyplot as plt
from PIL import Image
import urllib.request
import io  # Import the io module

# Define the image URL
url = "https://raw.githubusercontent.com/AshishJangra27/Cheat-Sheet

# Open the URL and read the data
with urllib.request.urlopen(url) as response:
  # Read the data as bytes
  data = response.read()

# Use Pillow to open the image from the byte data
img = Image.open(io.BytesIO(data))

# Convert the image to a format compatible with matplotlib (RGBA)
img = img.convert('RGBA')  # Assuming the image has transparency

# Now you can use matplotlib to display the image
plt.imshow(img)
plt.title('Displaying an Image')
plt.axis('off')
plt.show()
```

5.2 Customizing Image Display

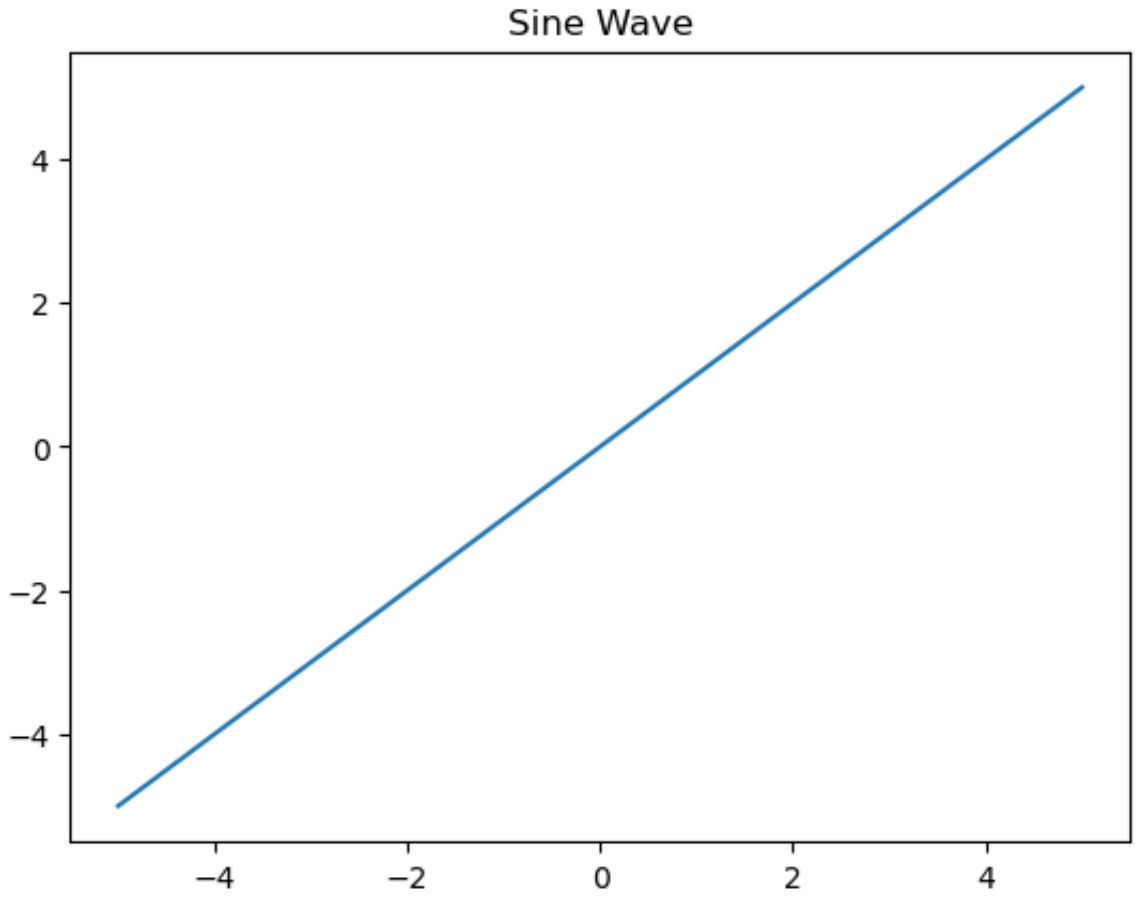
You can customize how images are displayed, including changing the color map and interpolation.

In [17]:
```
plt.imshow(img, cmap='gray', interpolation='none')
plt.title('Customized Image Display')
plt.axis('off')
plt.show()
```

5.3 Saving Plots and Images

You can save plots and images to files using the `savefig` function.

In [18]:
```
plt.plot(x, y)
plt.title('Sine Wave')
plt.savefig('sine_wave.png')
plt.show()
```



6. Interactive Plots

6.1 Using Widgets

Matplotlib supports interactive widgets using `ipywidgets`.

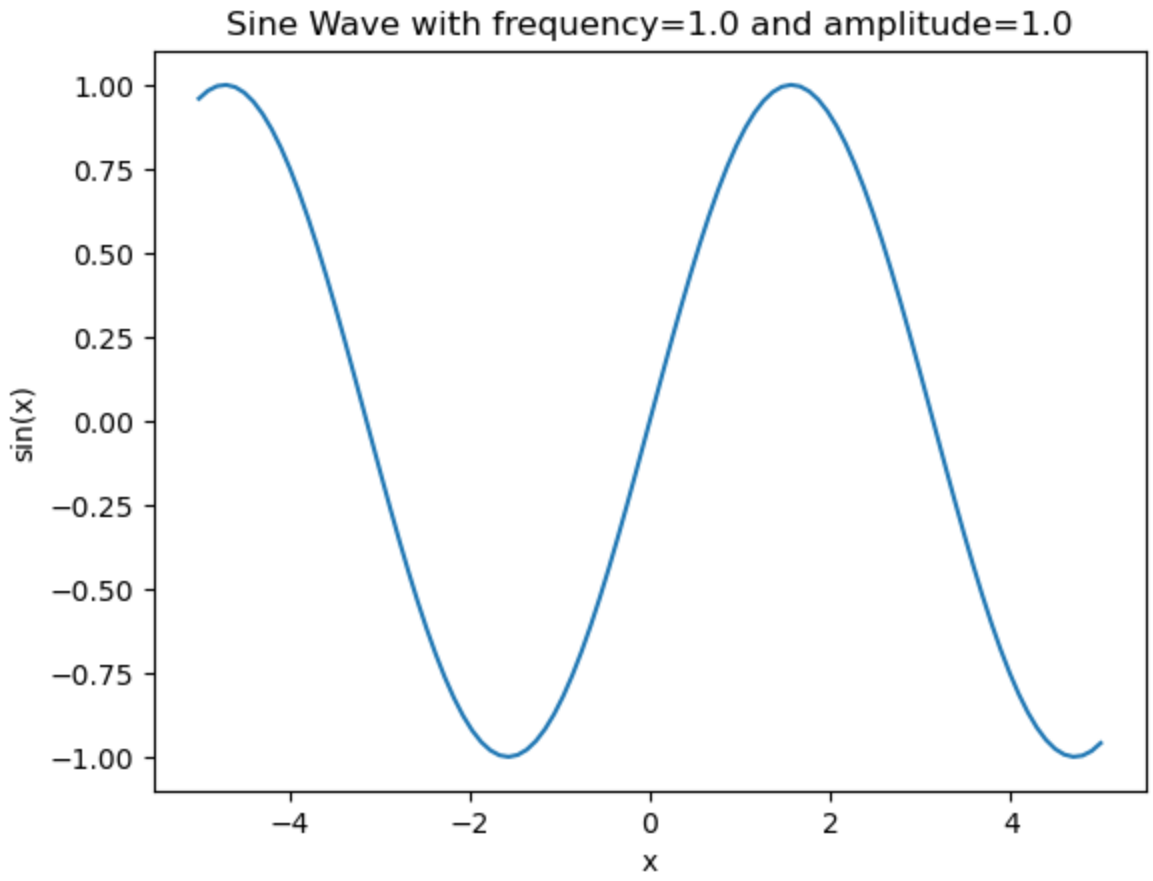
In [19]:
```
from ipywidgets import interact
import ipywidgets as widgets

def plot_func(frequency=1.0, amplitude=1.0):
    y = amplitude * np.sin(frequency * x)
    plt.plot(x, y)
    plt.title(f'Sine Wave with frequency={frequency} and amplitude=
    plt.xlabel('x')
    plt.ylabel('sin(x)')
    plt.show()

interact(plot_func, frequency=(0.1, 5.0), amplitude=(0.1, 2.0))
```

frequency 1.00

amplitude 1.00



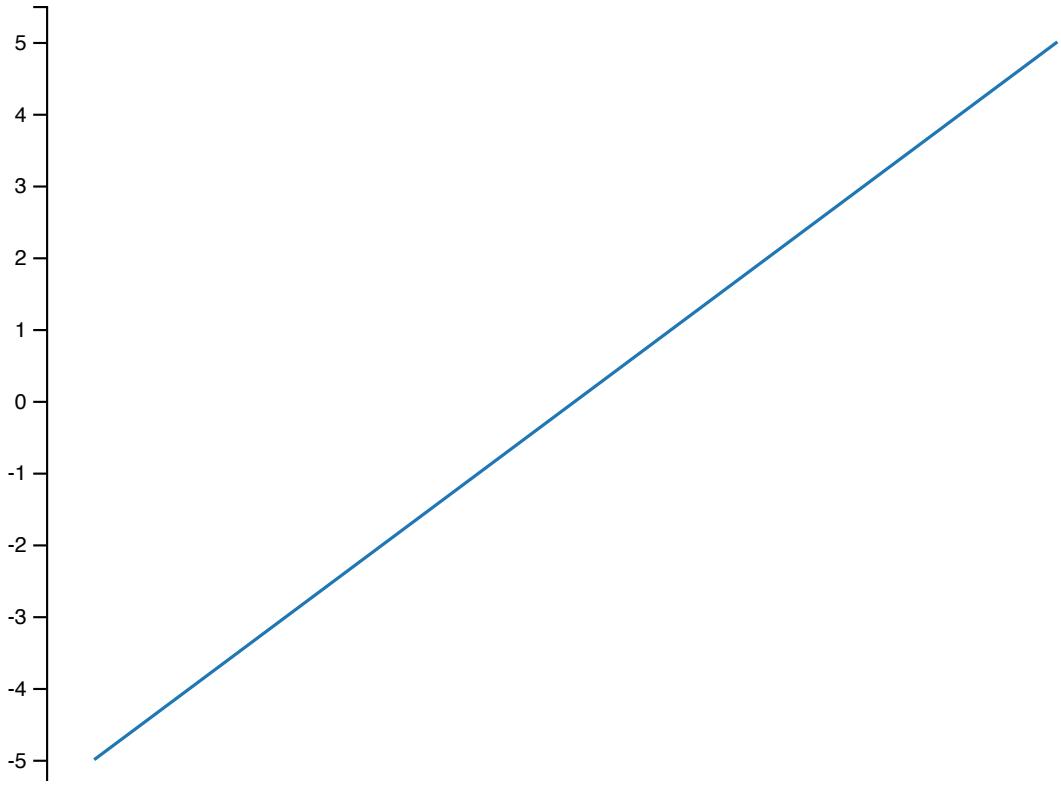
Out[19]: <function __main__.plot_func(frequency=1.0, amplitude=1.0)>

6.2 Interactive Plotting with mpld3

The `mpld3` library allows you to create interactive plots that can be viewed in a web browser.

In [20]:
```
import mpld3
fig, ax = plt.subplots()
ax.plot(x, y)
mpld3.display()
```

Out[20]:

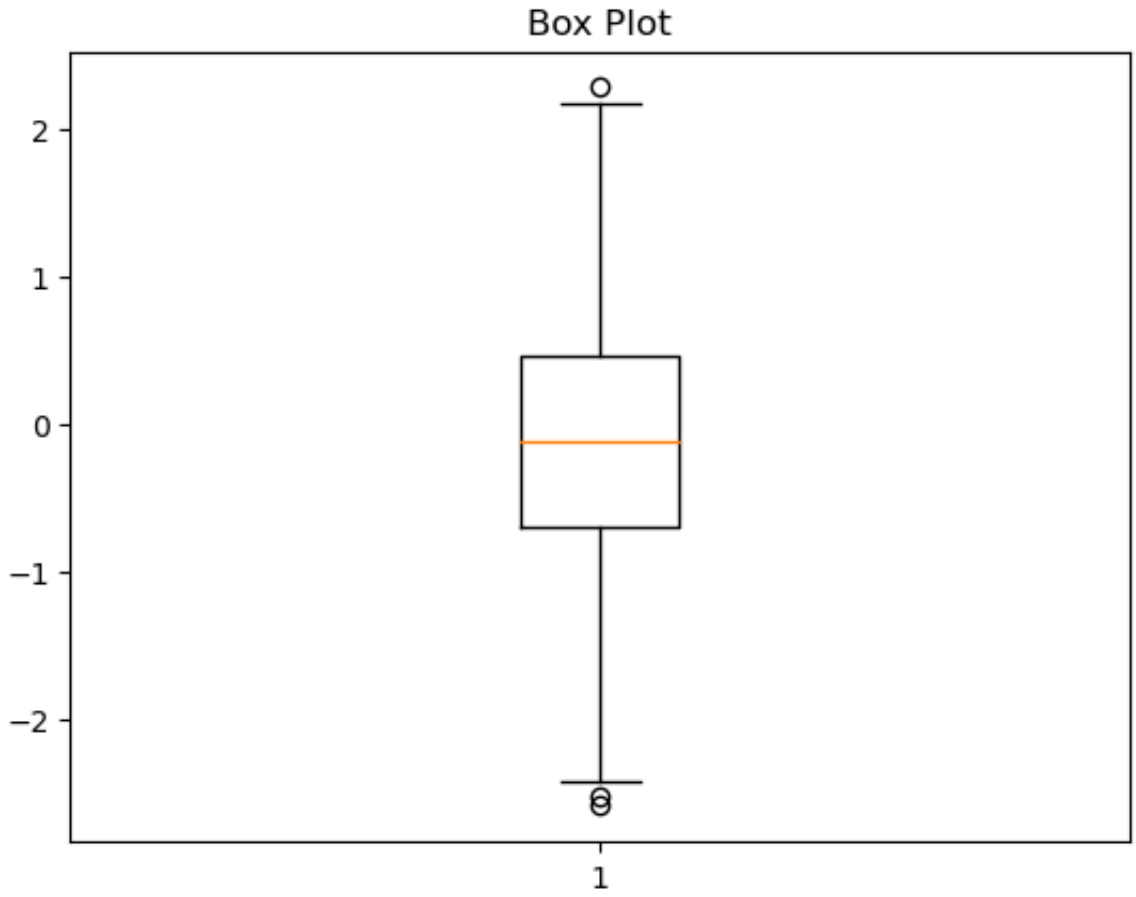


7. Statistical Plots

7.1 Box Plot

Box plots are used to visualize the distribution of data based on a five-number summary.

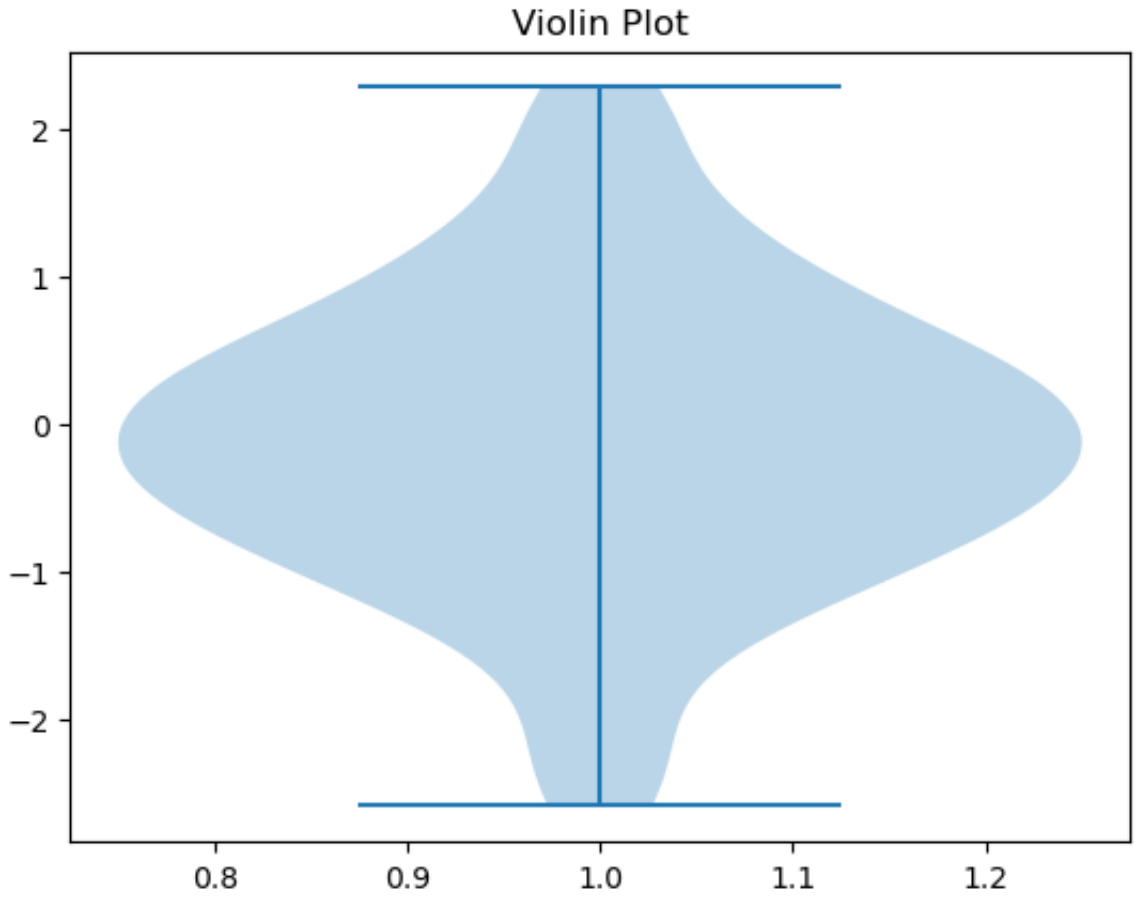
In [21]:
```
data = np.random.randn(100)
plt.boxplot(data)
plt.title('Box Plot')
plt.show()
```



7.2 Violin Plot

Violin plots are similar to box plots but also show the density of the data at different values.

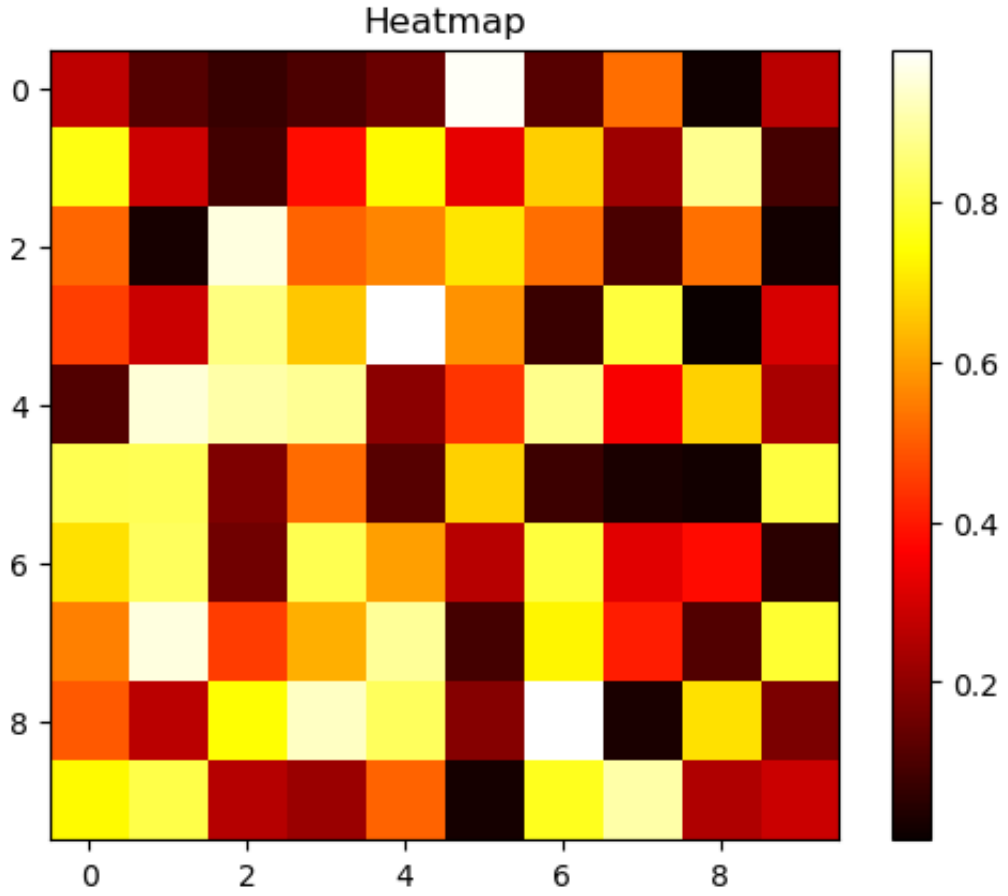
In [22]:
```
plt.violinplot(data)
plt.title('Violin Plot')
plt.show()
```



7.3 Heatmaps

Heatmaps are used to represent data values as colors in a matrix.

In [23]:
```
matrix = np.random.rand(10, 10)
plt.imshow(matrix, cmap='hot', interpolation='nearest')
plt.title('Heatmap')
plt.colorbar()
plt.show()
```



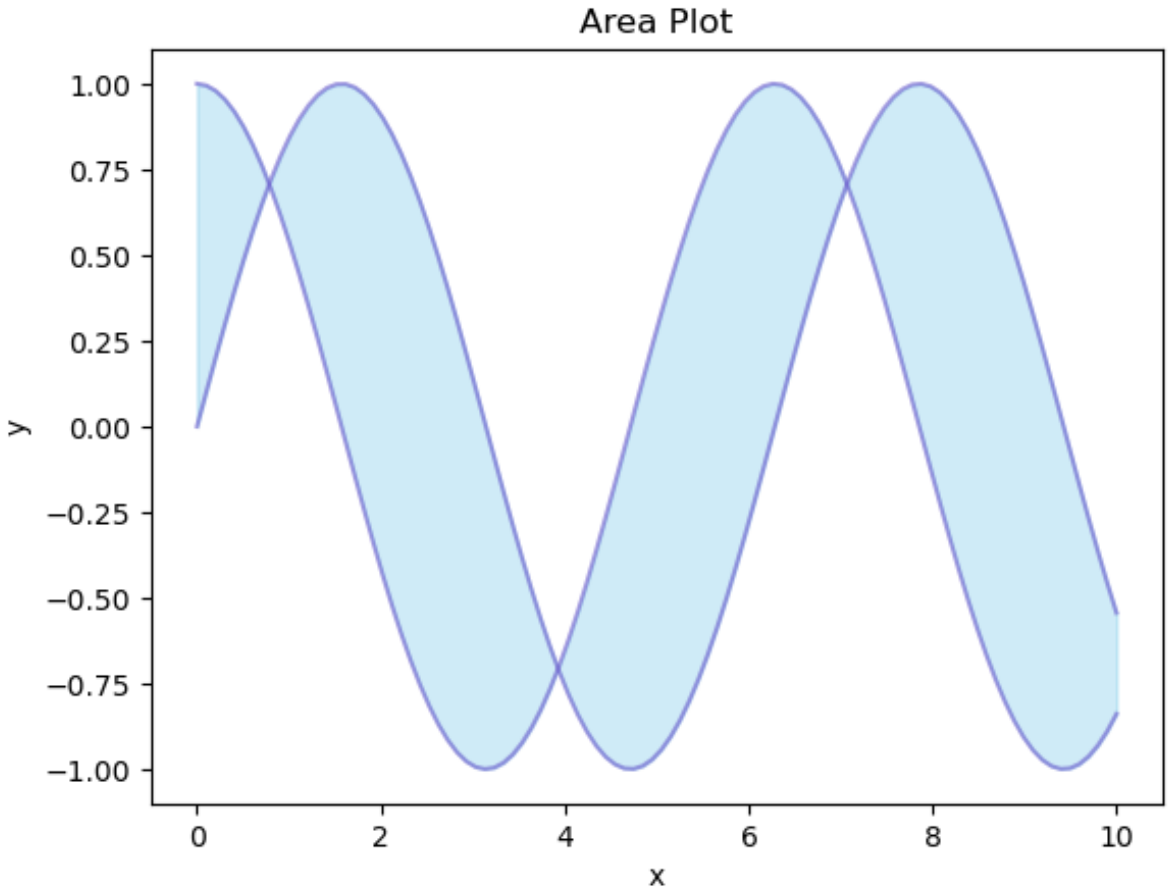
In [ ]:
```

```

7.4 Area Plot

Area plots are used to represent quantitative data visually.

In [24]:
```
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)
plt.fill_between(x, y1, y2, color='skyblue', alpha=0.4)
plt.plot(x, y1, x, y2, color='Slateblue', alpha=0.6)
plt.title('Area Plot')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



7.5 Stacked Bar Plot

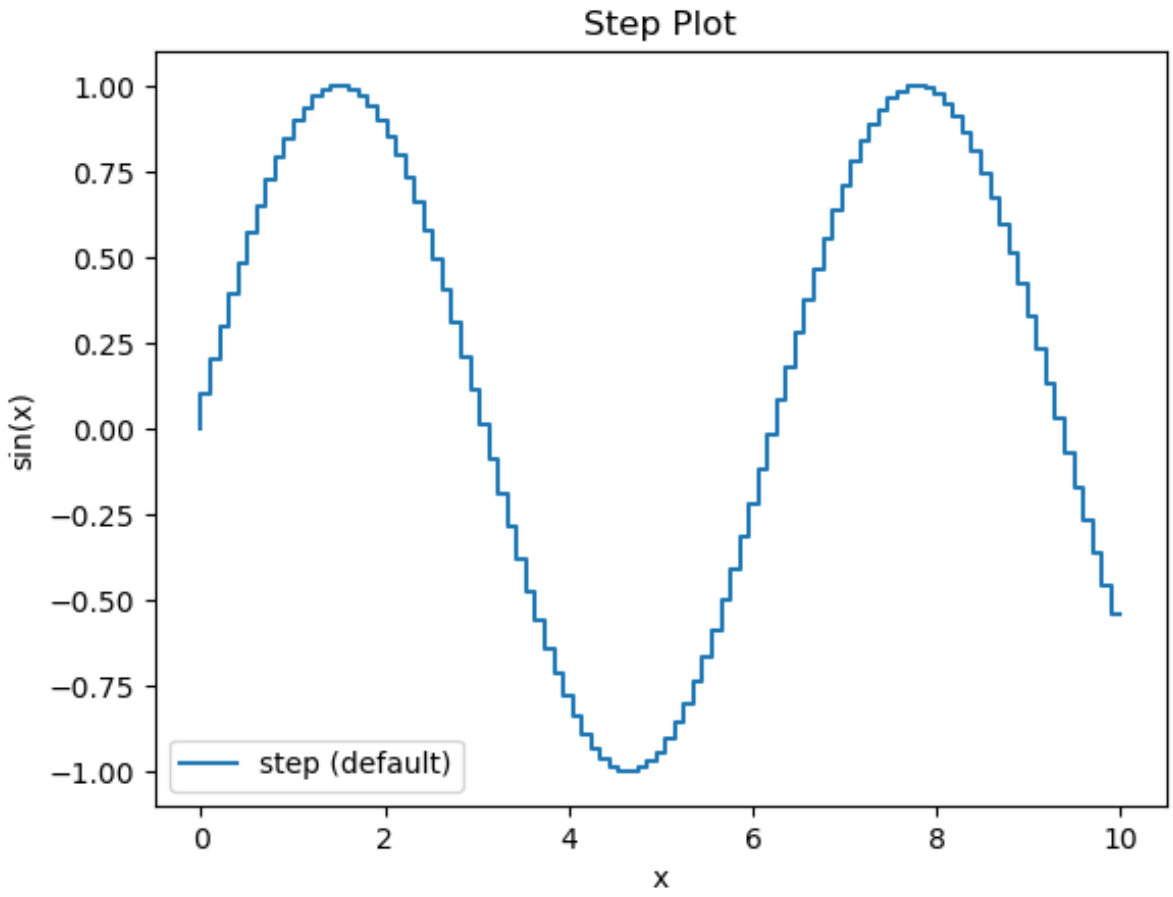
Stacked bar plots are used to represent multiple variables on a single plot.

In [25]:
```
categories = ['A', 'B', 'C', 'D']
values1 = [10, 20, 15, 25]
values2 = [5, 15, 10, 20]
plt.bar(categories, values1, color='b', label='Group 1')
plt.bar(categories, values2, bottom=values1, color='r', label='Grou
plt.title('Stacked Bar Plot')
plt.xlabel('Categories')
plt.ylabel('Values')
plt.legend()
plt.show()
```

7.6 Step Plot

Step plots are used to visualize data that changes at specific points.

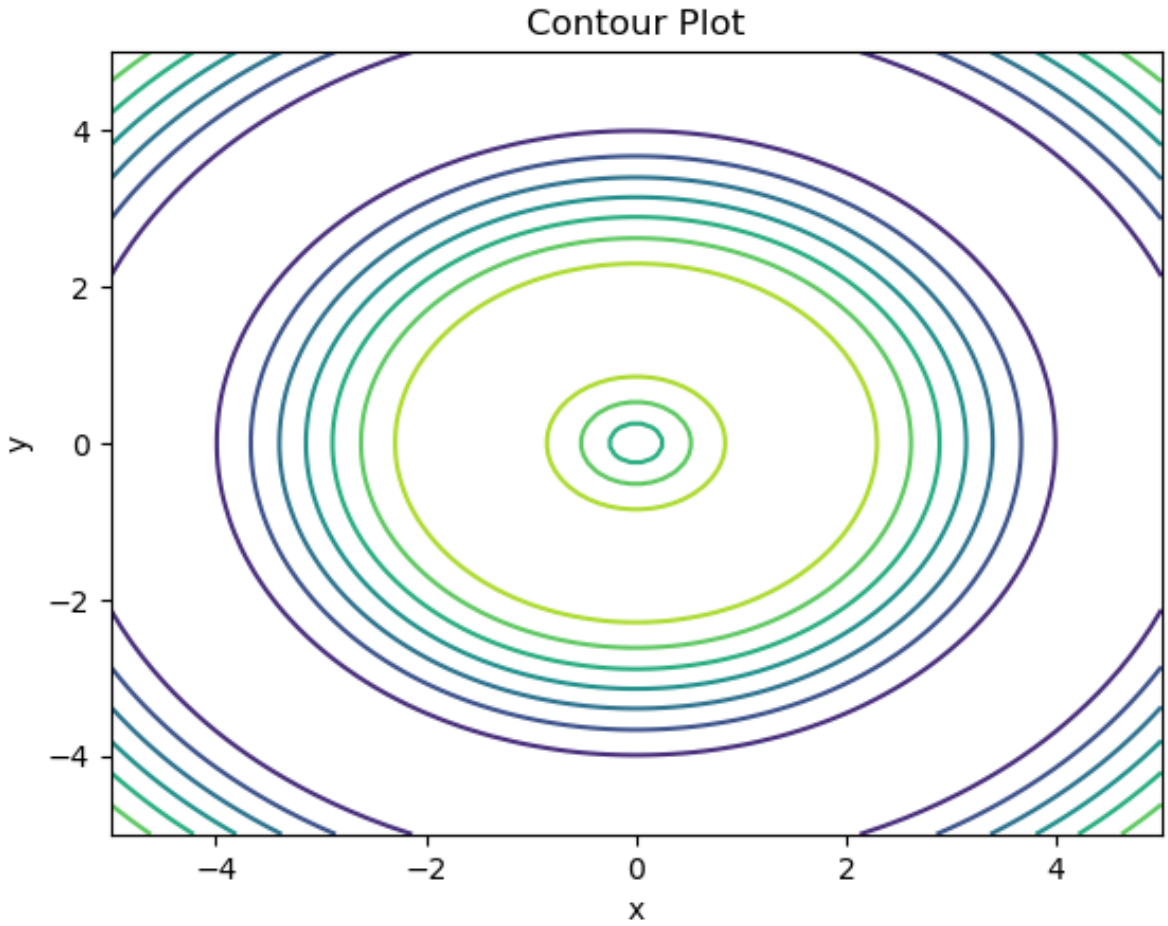
In [26]:
```
x = np.linspace(0, 10, 100)
y = np.sin(x)
plt.step(x, y, label='step (default)')
plt.title('Step Plot')
plt.xlabel('x')
plt.ylabel('sin(x)')
plt.legend()
plt.show()
```



7.7 Contour Plot

Contour plots are used to represent three-dimensional data in two dimensions using contours.

In [27]:
```
x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)
X, Y = np.meshgrid(x, y)
Z = np.sin(np.sqrt(X**2 + Y**2))
plt.contour(X, Y, Z, cmap='viridis')
plt.title('Contour Plot')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



7.8 Hexbin Plot

Hexbin plots are used to visualize the density of data points in a plot.

In [28]:
```
x = np.random.randn(10000)
y = np.random.randn(10000)
plt.hexbin(x, y, gridsize=50, cmap='Blues')
plt.colorbar(label='count')
plt.title('Hexbin Plot')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```