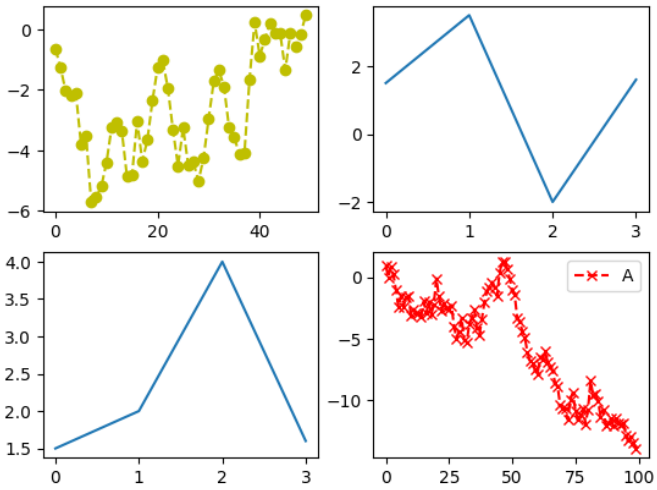
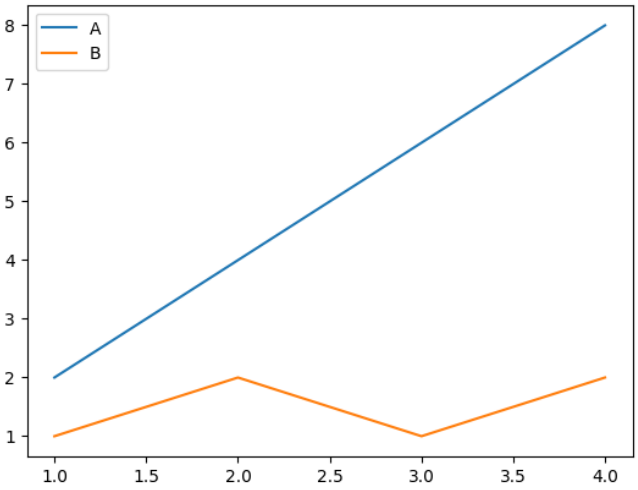
In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

In [2]:
```
fig = plt.figure()
ax1 = fig.add_subplot(2, 2, 1)
ax2 = fig.add_subplot(2, 2, 2)
ax3 = fig.add_subplot(2, 2, 3)
ax4 = fig.add_subplot(2, 2, 4)
ax2.plot([1.5, 3.5, -2, 1.6])
ax3.plot([1.5, 2, 4, 1.6])
ax1.plot(np.random.randn(50).cumsum(),linestyle='--',marker='o',color='y')
plt.plot(np.random.randn(100).cumsum(),linestyle='--',marker='x',color='r',label='A' )
plt.legend()
plt.show()

# supported values for linestyle are '-', '--', '-.', ':', 'None', ' ', '', 'solid', 'dashed', 'dashdot', 'dotted'
```



Add legend

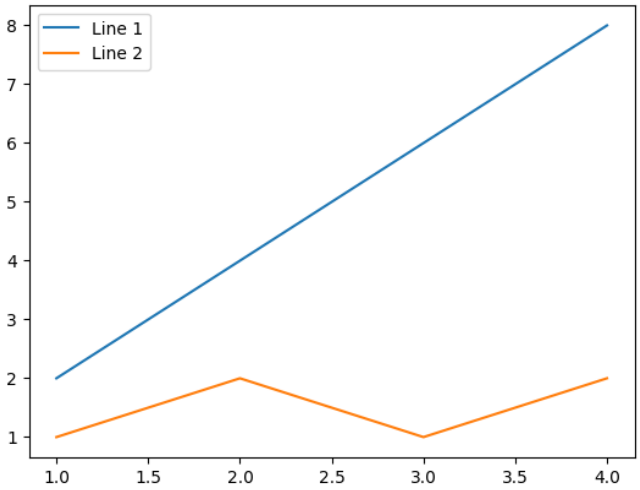
In [3]:
```
# Plotting two lines
x = [1, 2, 3, 4]
y1 = [2, 4, 6, 8]
y2 = [1, 2, 1, 2]

plt.plot(x, y1,label='A')
plt.plot(x, y2, label='B')

# Display the Legend
plt.legend()

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



Customizing Legend Location

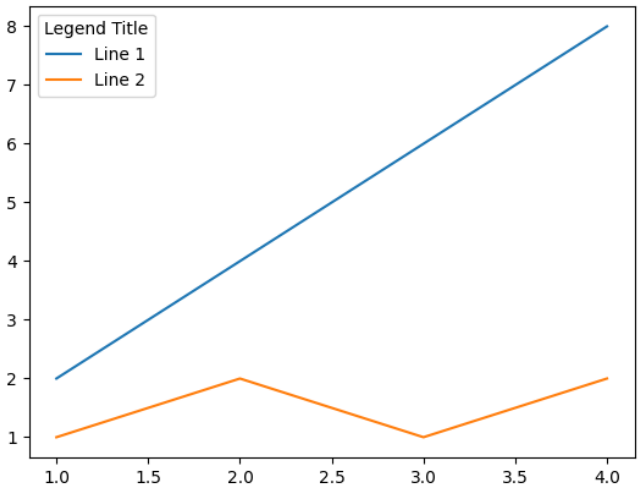
the **loc** parameter is used to specify the location of the legend. You can use strings like 'upper left', 'upper right', 'lower left', 'lower right', and many others.

In [4]:
```
# Plotting two lines
x = [1, 2, 3, 4]
y1 = [2, 4, 6, 8]
y2 = [1, 2, 1, 2]

plt.plot(x, y1, label='Line 1')
plt.plot(x, y2, label='Line 2')

# Display the Legend at the upper left corner
plt.legend(loc='upper left')

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



Adding a Title to the Legend

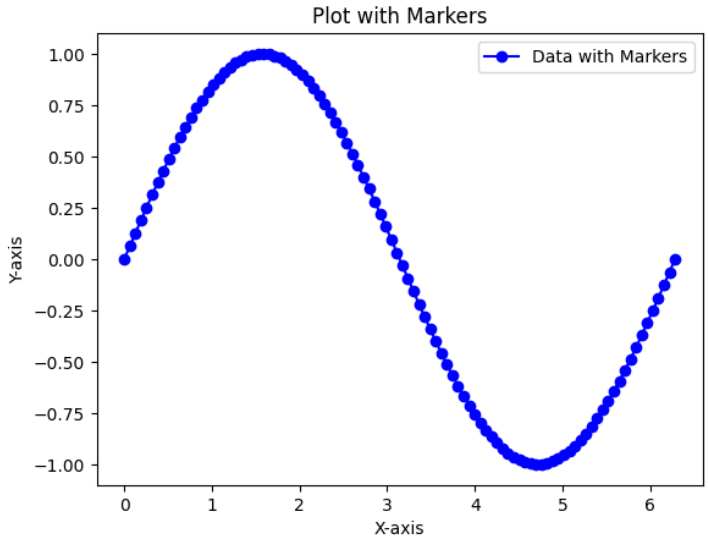
In [5]:
```
# Plotting two lines
x = [1, 2, 3, 4]
y1 = [2, 4, 6, 8]
y2 = [1, 2, 1, 2]

plt.plot(x, y1, label='Line 1')
plt.plot(x, y2, label='Line 2')

# Display the legend with a title
plt.legend(title='Legend Title')

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



Markers

Markers are symbols that can be placed at data points. You can use the marker parameter in the plt.plot() function to specify a marker style.

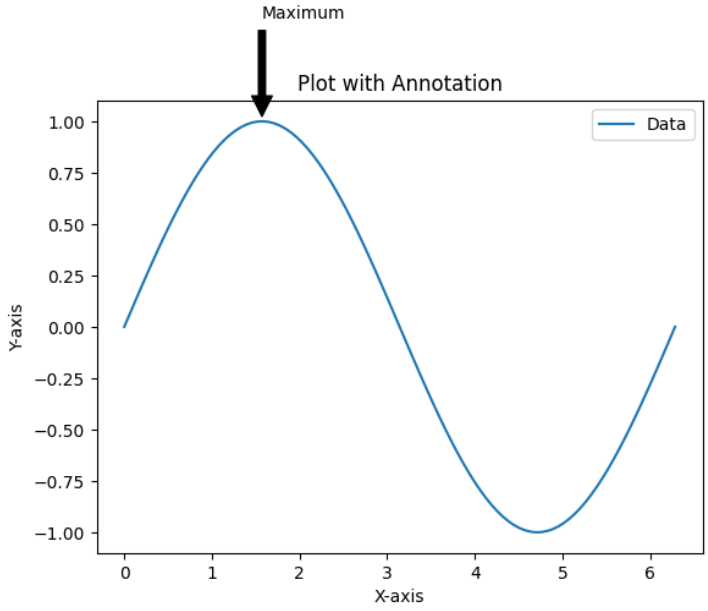
In [6]:
```
# Sample data
x = np.linspace(0, 2 * np.pi, 100)
y = np.sin(x)

# Plot with markers
plt.plot(x, y, marker='o', linestyle='-', color='b', label='Data with Markers')

# Add labels and legend
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Plot with Markers')
plt.legend()

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



Annotations

Annotations can be used to add text or arrows to the plot. You can use the **plt.annotate()** function for this purpose

In [7]:
```
# Sample data
x = np.linspace(0, 2 * np.pi, 100)
y = np.sin(x)

# Plot the data
plt.plot(x, y, label='Data')

# Annotate a specific point
plt.annotate('Maximum', xy=(np.pi/2, 1), xytext=(np.pi/2, 1.5),
             arrowprops=dict(facecolor='black', shrink=0.05))

# Add labels and legend
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Plot with Annotation')
plt.legend()

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

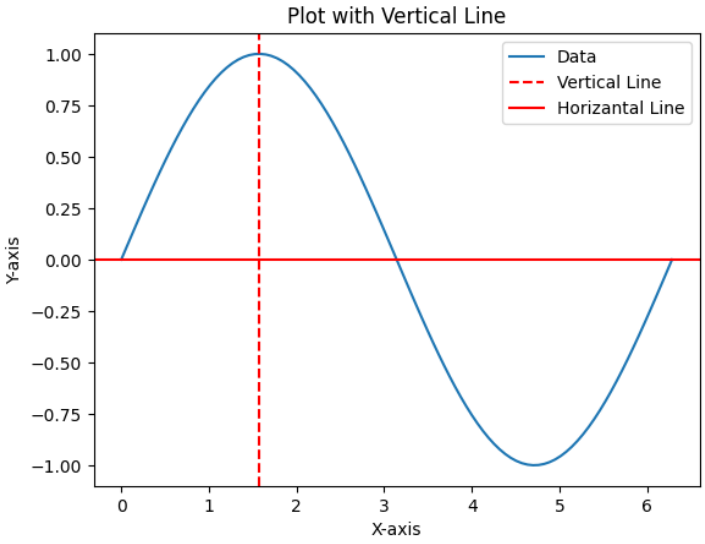


Vertical and Horizontal Lines

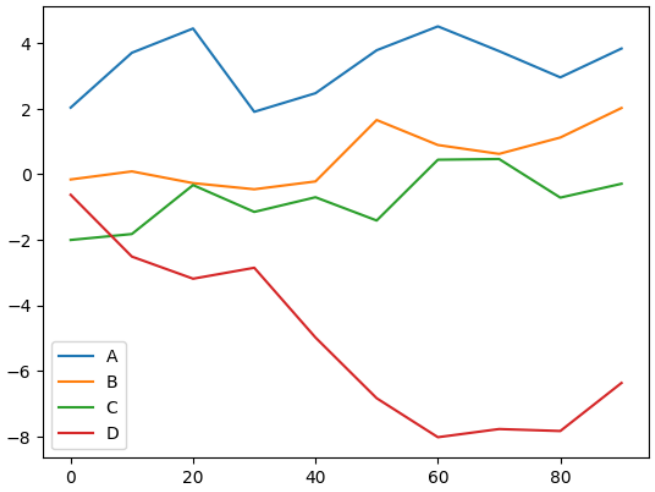
You can use **plt.axvline() and plt.axhline()** to add vertical and horizontal lines to the plot, respectively.

In [8]:
```
x = np.linspace(0, 2 * np.pi, 100)
y = np.sin(x)

# Plot the data
plt.plot(x, y, label='Data')

# Add a vertical line
plt.axvline(x=np.pi/2, color='r', linestyle='--', label='Vertical Line')
plt.axhline(y=0, color='r', linestyle='-', label='Horizontal Line')

# Add labels and legend
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Plot with Vertical Line')
plt.legend()
```

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



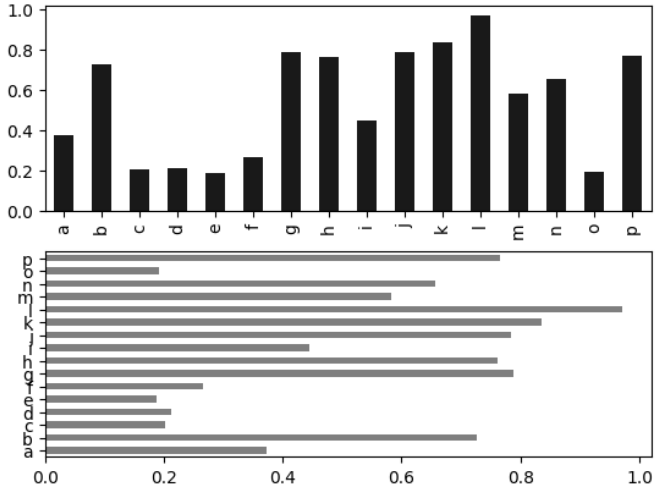
In [9]:
```
df = pd.DataFrame(np.random.randn(10, 4).cumsum(0),columns=['A', 'B', 'C', 'D'],index=np.arange(0, 100, 10))
df.plot()
plt.show()
```



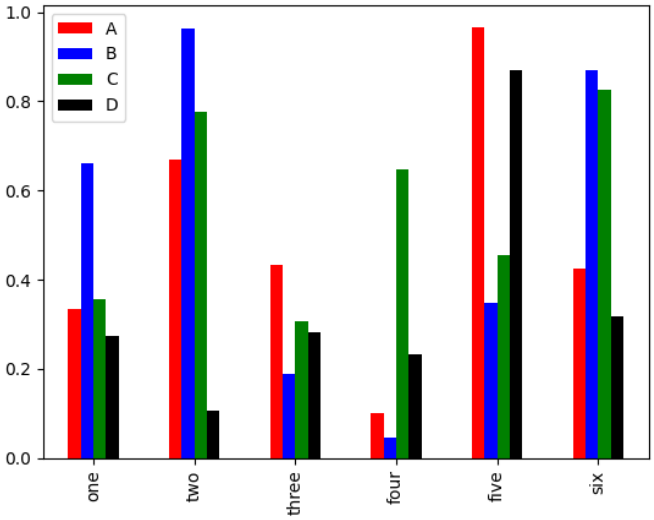
Bar Plots

plot.bar() and plot.barh()

In [10]:
```
fig, axes = plt.subplots(2, 1)
data = pd.Series(np.random.rand(16), index=list('abcdefghijklmnop'))
data.plot.bar(ax=axes[0], color='k', alpha=0.9)
data.plot.barh(ax=axes[1], color='k', alpha=0.5)
plt.show()
```

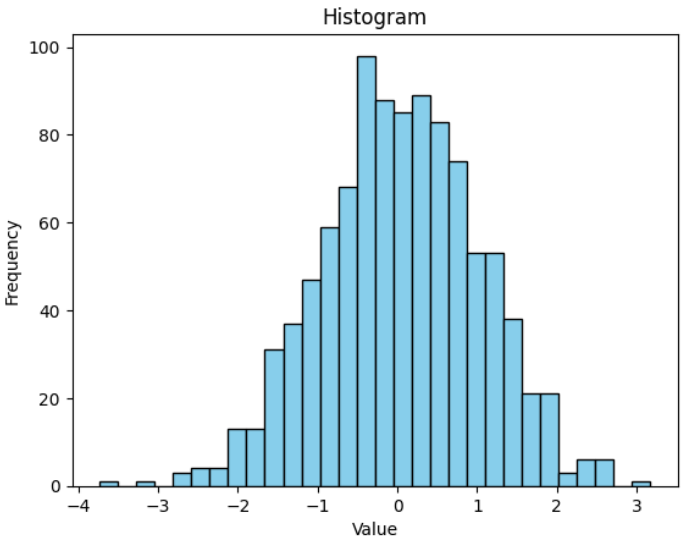


In [11]:
```
df = pd.DataFrame(np.random.rand(6, 4),index=['one', 'two', 'three', 'four', 'five', 'six'],columns=pd.Index(['A', 'B', 'C', 'D'],name='Genus'))
df.plot.bar(color=['r','b','g','k'])
plt.legend(loc='best')
plt.show()
```



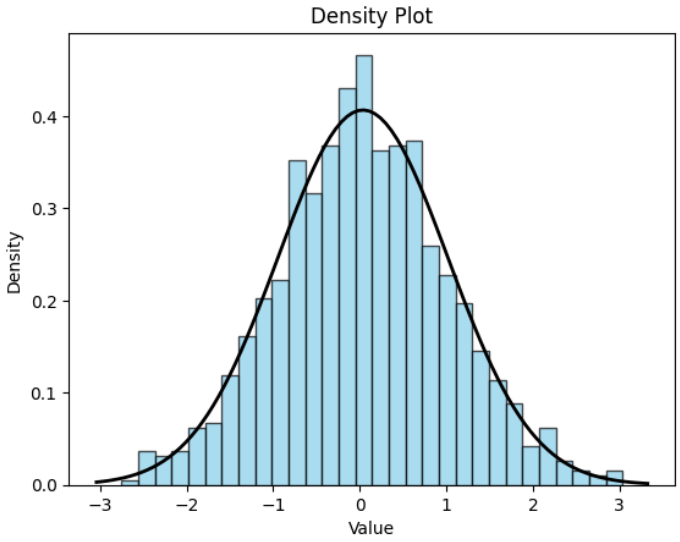
Histograms

In [12]:
```
# Generate random data
data = np.random.randn(1000)

# Create a histogram
plt.hist(data, bins=30, edgecolor='black', color='skyblue')

# Add labels and title
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Histogram')

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

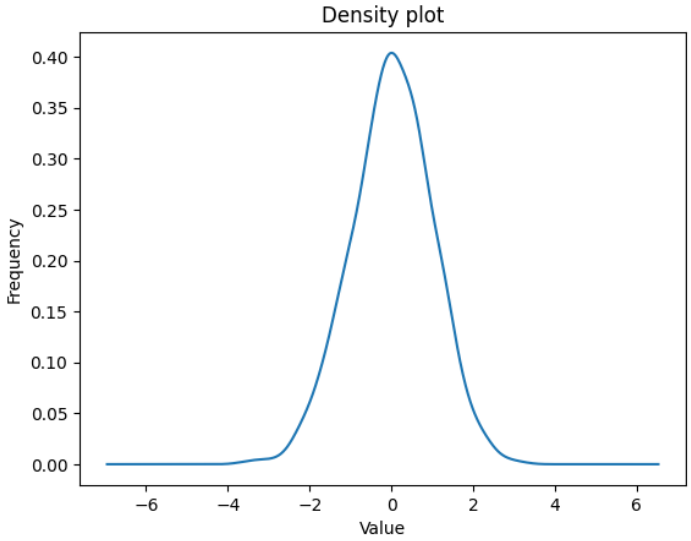


Density Plot with Matplotlib

In [13]:
```
from scipy.stats import norm

# Generate random data
data = np.random.randn(1000)

# Create a density plot
plt.hist(data, bins=30, density=True, edgecolor='black', color='skyblue', alpha=0.7)

# Plot the fitted normal distribution
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)
p = norm.pdf(x, np.mean(data), np.std(data))
# pdf=probability density function
plt.plot(x, p, 'k', linewidth=2)

# Add labels and title
plt.xlabel('Value')
plt.ylabel('Density')
plt.title('Density Plot')

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

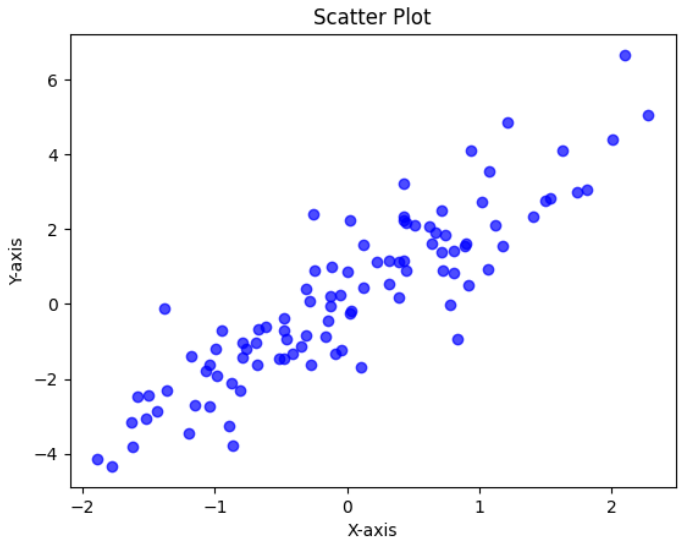


In [14]:
```
# Generate random data
data = np.random.randn(1000)
df= pd.DataFrame(data)
#  Create a histogram
df[0].plot.density()

# Add labels and title
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Density plot')

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



Basic Scatter Plot with Matplotlib

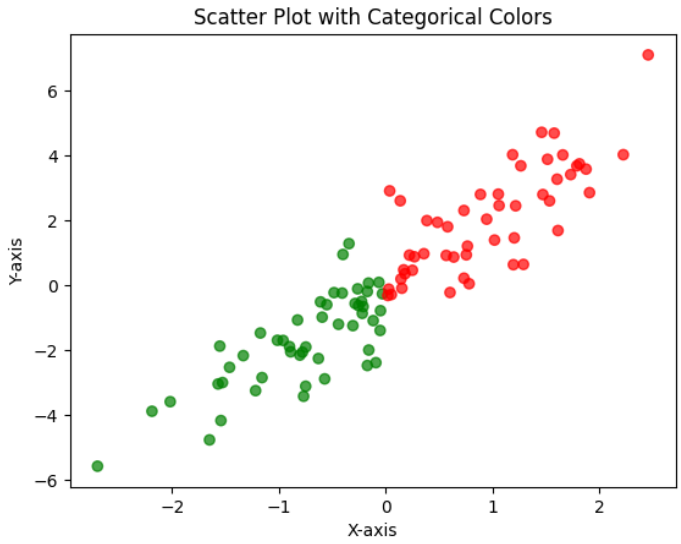
In [15]:
```
# Generate random data
x = np.random.randn(100)
y = 2 * x + np.random.randn(100)

# Create a scatter plot
plt.scatter(x, y, color='blue', alpha=0.7)

# Add labels and title
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Scatter Plot')

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



Scatter Plot with Categorical Colors:

In [16]:
```
# Generate random data
x = np.random.randn(100)
y = 2 * x + np.random.randn(100)
categories = np.where(x > 0, 'r', 'g')

# Create a scatter plot with categorical colors
plt.scatter(x, y, c=categories, cmap='coolwarm', alpha=0.7)

# Add labels and title
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Scatter Plot with Categorical Colors')

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

```
C:\Users\hp\AppData\Local\Temp\ipykernel_5924\320744206.py:7: UserWarning: No data for colormapping provided via 'c'. Parameters 'cmap' will be ignored
  plt.scatter(x, y, c=categories, cmap='coolwarm', alpha=0.7)
```

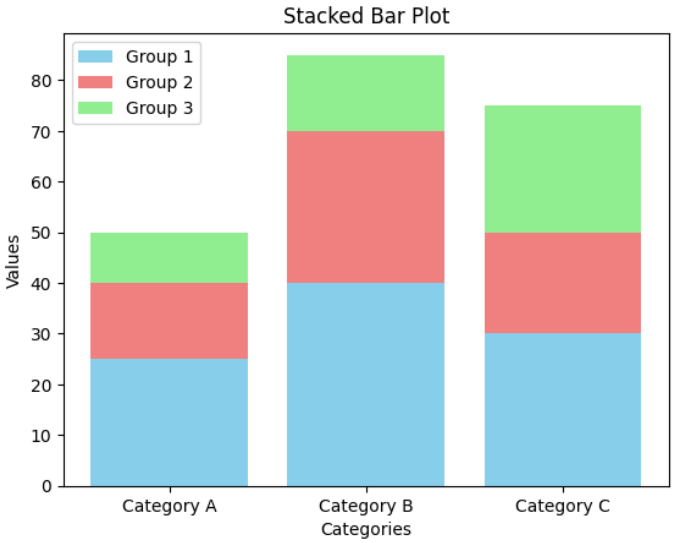


Stacked bar

Stacked bar plots are useful for visualizing the contribution of different subgroups to a total. You can create stacked bar plots using Matplotlib.

In [17]:
```
# Sample data
categories = ['Category A', 'Category B', 'Category C']
values1 = [25, 40, 30]
values2 = [15, 30, 20]
values3 = [10, 15, 25]

# Create a stacked bar plot
bar_width = 0.8
bar_positions = np.arange(len(categories))

plt.bar(bar_positions, values1, color='skyblue', label='Group 1', width=bar_width)
plt.bar(bar_positions, values2, bottom=values1, color='lightcoral', label='Group 2', width=bar_width)
plt.bar(bar_positions, values3, bottom=np.array(values1) + np.array(values2), color='lightgreen', label='Group 3', width=bar_width)


# Add labels, title, and legend
plt.xlabel('Categories')
plt.ylabel('Values')
plt.title('Stacked Bar Plot')
plt.xticks(bar_positions, categories)
plt.legend()
```

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

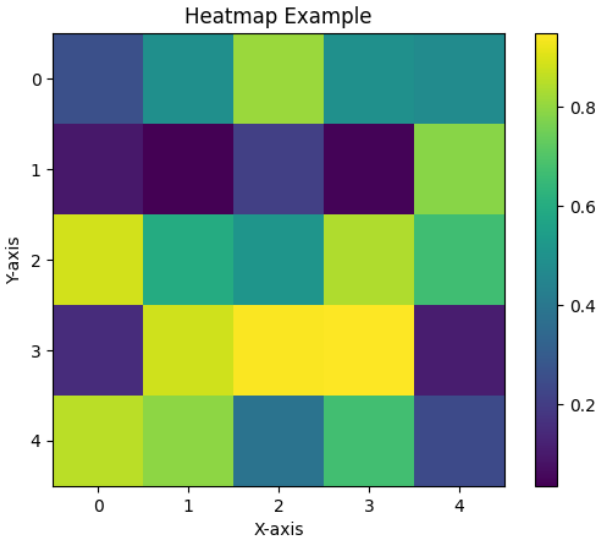


Heatmap

Creating a heatmap using Matplotlib involves using imshow or pcolormesh functions to represent data as a colored grid

a.

In [18]:
```
# Create a random matrix as sample data
data = np.random.rand(5, 5)

# Create a heatmap using Matplotlib's imshow
plt.imshow(data, cmap='viridis', interpolation='nearest')

# Add a color bar
plt.colorbar()

# Add labels and title
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Heatmap Example')

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



In [ ]:
```

```