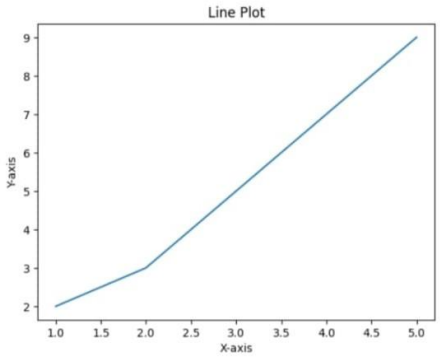
Visualization Library Documentation

Objective

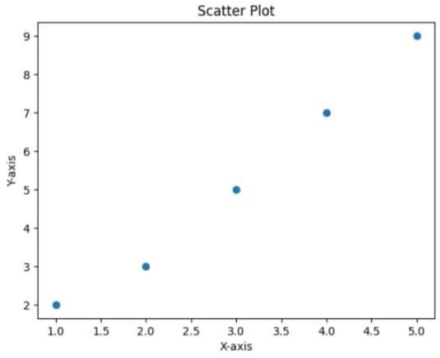
This guide provides a comprehensive documentation for two Python visualization libraries, focusing on the variety of graphs each library can generate. Each section includes practical code examples to help users understand how to implement the graphs.

Graph Types

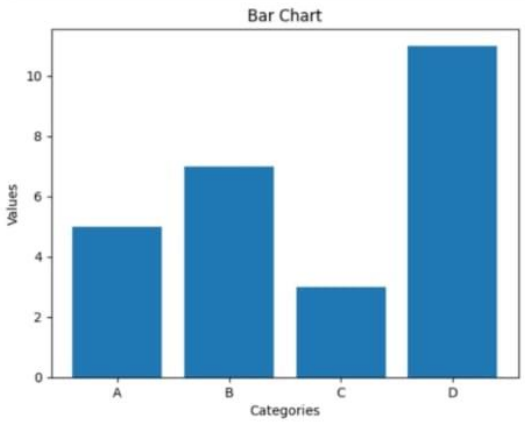
In [1]:
```python
import matplotlib.pyplot as plt

# Sample data
x = [1, 2, 3, 4, 5]
y = [2, 3, 5, 7, 9]

# Create a line plot
plt.plot(x, y)
plt.title('Line Plot')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.show()
```



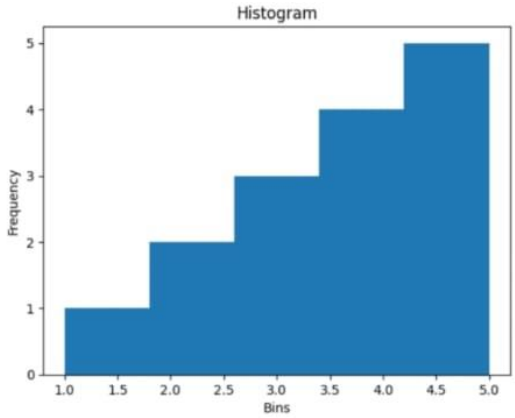
In [2]:
```python
import matplotlib.pyplot as plt

# Sample data
x = [1, 2, 3, 4, 5]
y = [2, 3, 5, 7, 9]

# Create a scatter plot
plt.scatter(x, y)
plt.title('Scatter Plot')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.show()
```



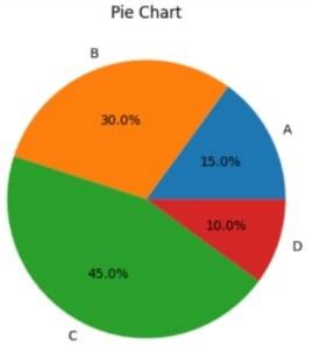
In [3]:
```python
import matplotlib.pyplot as plt

# Sample data
categories = ['A', 'B', 'C', 'D']
values = [5, 7, 3, 11]

# Create a bar chart
plt.bar(categories, values)
plt.title('Bar Chart')
plt.xlabel('Categories')
plt.ylabel('Values')
plt.show()
```



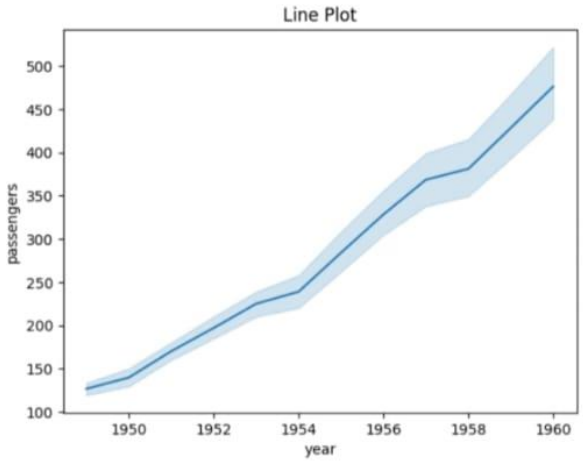
In [4]:
```
import matplotlib.pyplot as plt

# Sample data
data = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 5]

# Create a histogram
plt.hist(data, bins=5)
plt.title('Histogram')
plt.xlabel('Bins')
plt.ylabel('Frequency')
plt.show()
```



In [5]:
```
import matplotlib.pyplot as plt

# Sample data
labels = ['A', 'B', 'C', 'D']
sizes = [15, 30, 45, 10]

# Create a pie chart
plt.pie(sizes, labels=labels, autopct='%1.1f%%')
plt.title('Pie Chart')
plt.show()
```

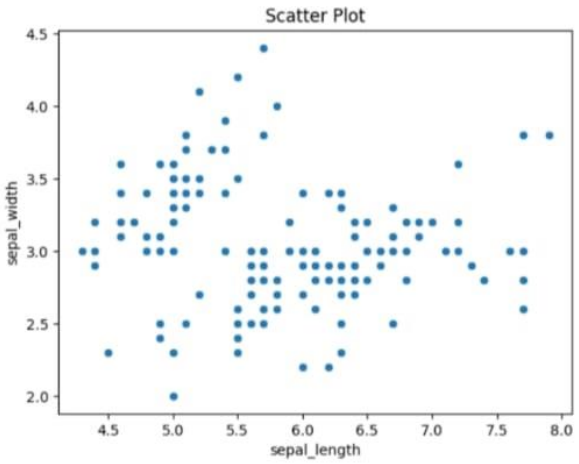


In [6]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Sample data
data = sns.load_dataset('flights')

# Create a line plot
sns.lineplot(x='year', y='passengers', data=data)
plt.title('Line Plot')
plt.show()
```



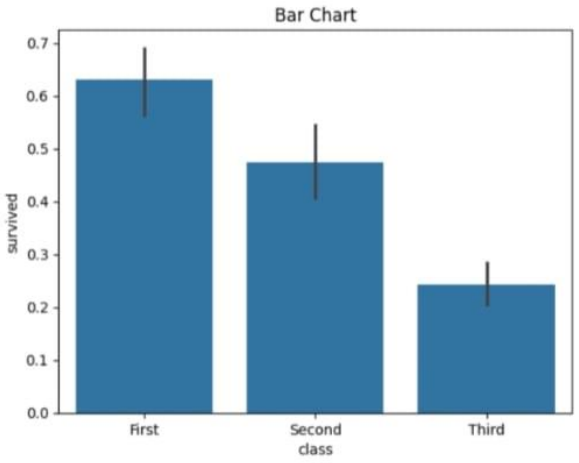
In [7]:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Sample data
data = sns.load_dataset('iris')

# Create a scatter plot
sns.scatterplot(x='sepal_length', y='sepal_width', data=data)
plt.title('Scatter Plot')
plt.show()
```



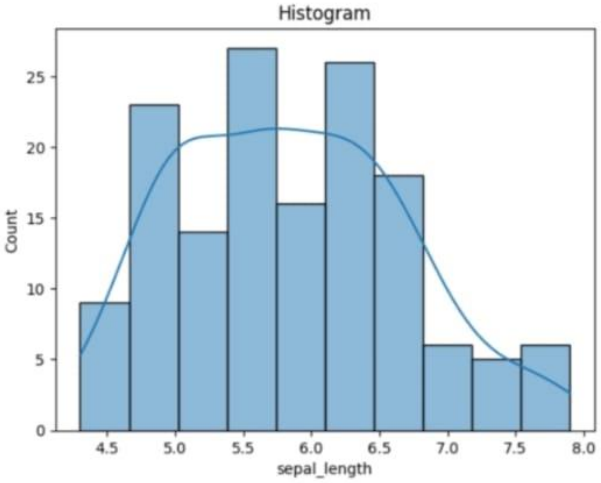
In [8]:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Sample data
data = sns.load_dataset('titanic')

# Create a bar chart
sns.barplot(x='class', y='survived', data=data)
plt.title('Bar Chart')
plt.show()
```



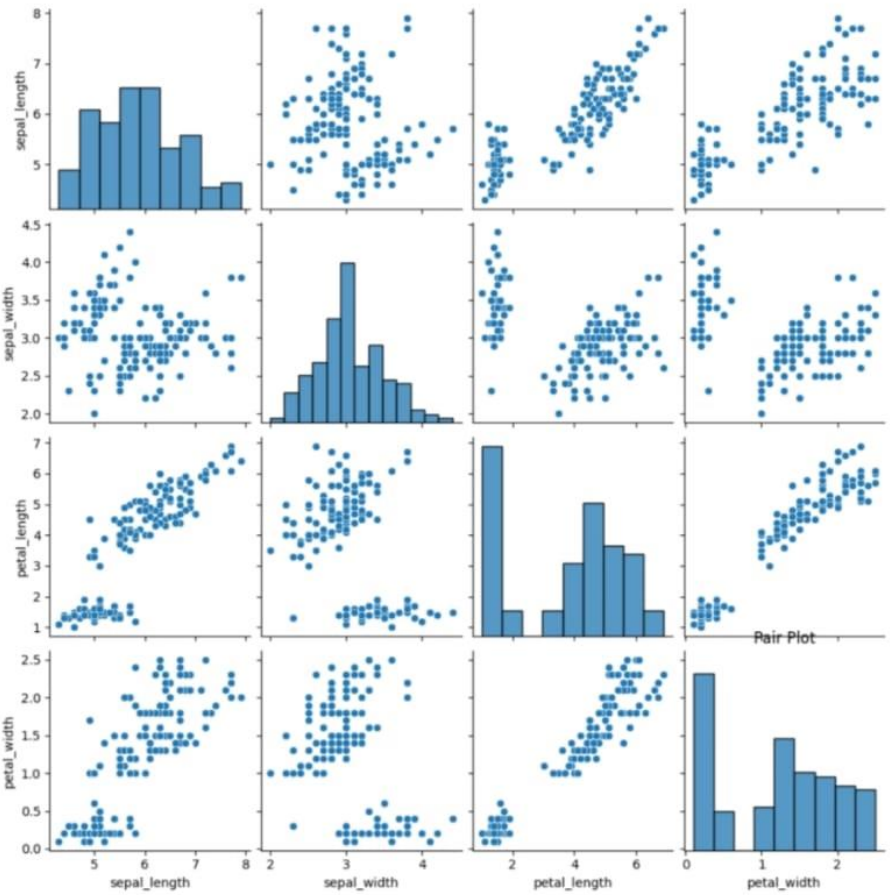
In [9]:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Sample data
data = sns.load_dataset('iris')

# Create a histogram
sns.histplot(data['sepal_length'], bins=10, kde=True)
plt.title('Histogram')
plt.show()
```



In [10]:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Sample data
data = sns.load_dataset('iris')

# Create a pair plot
sns.pairplot(data)
plt.title('Pair Plot')
plt.show()
```



Comparison of Libraries

Feature	Matplotlib	Seaborn
Ease of Use	Medium - detailed but extensive options	High - intuitive for statistical plots
Customization	Very high	Moderate
Interactivity	Limited (static)	Limited (static)
Performance	Efficient for most tasks	Efficient for statistical plots
Best Use Cases	Publication-quality figures	Statistical data visualization

---

Conclusion

In this documentation, we explored two prominent Python libraries for data visualization, each with its unique strengths and applications. We began by providing an overview of Matplotlib and Seaborn, highlighting how Matplotlib serves as the foundational tool for producing detailed, static visualizations, while Seaborn simplifies complex statistical plots and enhances Matplotlib's capabilities with a user-friendly syntax.The guide walked through various common graph types—line plots, scatter plots, bar charts, histograms, and pie charts—offering practical code examples to demonstrate how each library handles different types of visualizations. By including clear, concise examples, this guide aims to make it easier for users to replicate and adapt these visualizations to their own data and analytical needs.

In the comparison section, we analyzed the strengths and limitations of both libraries across several criteria:

Ease of Use: Seaborn's simplified syntax and default styling make it ideal for users who want fast, aesthetically pleasing plots, while Matplotlib's extensive customization options are well-suited for users with specific, publication-ready visual requirements.

Customization: Matplotlib excels with high customization, giving users complete control over plot elements, making it suitable for more detailed and tailored visualizations.

Interactivity: Although both libraries primarily produce static images, they integrate well with Jupyter Notebooks, providing some interactivity through live coding sessions.

Performance with Large Datasets: Matplotlib handles large datasets efficiently and is generally quicker for basic plotting tasks, while Seaborn adds additional processing layers for styling and advanced statistics, making it optimal for medium-sized datasets.

This guide offers a foundation for users to select the appropriate library based on their project needs—whether they require the in-depth customization of Matplotlib or the ease and statistical power of Seaborn. By following the code examples and exploring additional resources, new users are empowered to start visualizing data more effectively and creatively.

Ultimately, both Matplotlib and Seaborn complement each other in the Python ecosystem, and with practice, users can leverage their combined strengths to produce impactful, insightful visualizations across a wide range of data contexts.