MATPLOT LIBRARY

Matplotlib is a python library used for creating static, animated and interactive visualization in a variety of formats. It is a versatile tool that enables users to generate publication-quality plots, interactive figures and customized visual styles.

There are three main parts to a Matplotlib figure:

- **Figure**: This is the whole region of space that's created when you create any figure. The Figure object is the overall object that contains everything else.
- **Axes**: An Axes object is the object that contains the *x*-axis and *y*-axis for a 2D plot. Each Axes object corresponds to a plot or a graph. You can have more than one Axes object in a Figure, as you'll see later on in this Chapter.
- **Axis**: An Axis object contains one of the axes, the *x*-axis or the *y*-axis for a 2D plot.

Key Features of Matplotlib

- **Versatile Plotting**: Create a wide variety of visualizations, including line plots, scatter plots, bar charts, and histograms.
- **Extensive Customization**: Control every aspect of your plots, from colors and markers to labels and annotations.
- **Seamless Integration with NumPy**: Effortlessly plot data arrays directly, enhancing data manipulation capabilities.
- **High-Quality Graphics**: Generate publication-ready plots with precise control over aesthetics.
- **Cross-Platform Compatibility**: Use Matplotlib on Windows, macOS, and Linux without issues.

Key Uses of Matplotlib:

- **Basic Plots:** Line plots, bar charts, histograms, scatter plots, etc.
- **Statistical Visualization:** Box plots, error bars, and density plots.
- **Customization:** Control over colors, labels, gridlines, and styles.
- **Subplots & Layouts:** Create multiple plots in a single figure.
- **3D Plotting:** Surface plots and 3D scatter plots using mpl_toolkits.mplot3d.

Advantages

- **Versatility:** It can generate various plot types, including line plots, scatter plots, bar charts, histograms, and more.
- **Customization:** It allows fine-grained control over plot elements like colors, markers, labels, and annotations.
- **Integration with NumPy:** It works seamlessly with NumPy arrays, which facilitates data manipulation and plotting.
- **High-quality graphics:** It produces publication-ready plots with precise control over aesthetics.
- **Cross-platform compatibility:** It functions smoothly on Windows, macOS, and Linux.

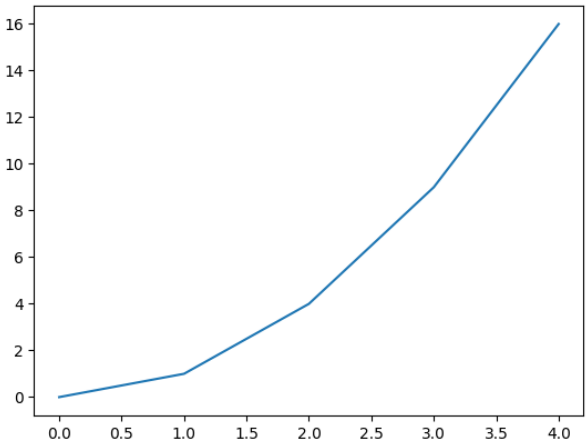
Disadvantages

- **Complexity:** The extensive customization options can be overwhelming for beginners, leading to a steeper learning curve.
- **Verbose code:** Creating complex visualizations may require multiple lines of code, making it less concise than some other libraries.

Example of a Plot in Matplotlib:
Let's create a simple line plot using Matplotlib, showcasing the ease with which you can visualize data.

```
import matplotlib.pyplot as plt

x = [0, 1, 2, 3, 4]

y = [0, 1, 4, 9, 16]

plt.plot(x, y)

plt.show()
```

Output:



Simplest plot in Matplotlib

Graph Types and Examples

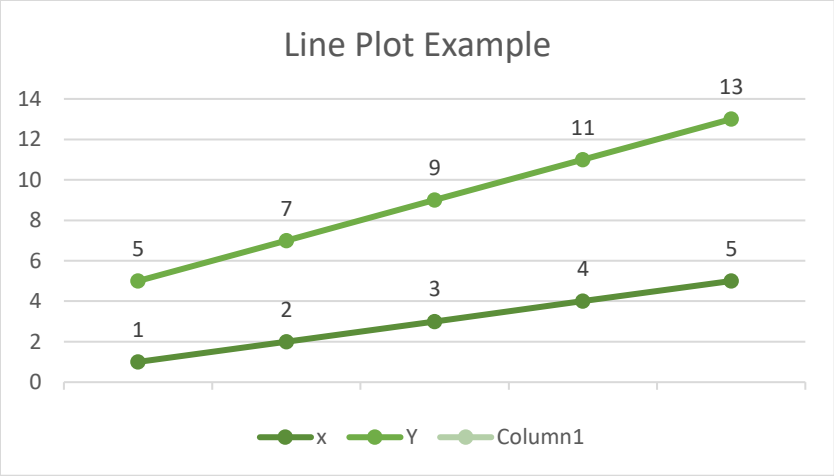
Matplotlib is a low-level, highly customizable library for creating static, animated, and interactive plots. It serves as the foundation for many other visualization libraries in Python, including Seaborn. Its pyplot interface provides a MATLAB-like API for creating a wide variety of plots.

1. Line Plot

- **Description**: A line plot is used to display data points connected by straight lines. This type of graph is ideal for showing trends or changes in values over a continuous interval or time span.
- **Use Case**: Monitoring stock prices over time, temperature variations across days, or monthly sales trends.

Code Example:

```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4, 5]
y = [5, 7, 9, 11, 13]
plt.plot(x, y, color='green', marker='o')
plt.title("Line Plot Example")
plt.xlabel("Time (days)")
plt.ylabel("Value")
plt.grid(True)
plt.show()
```



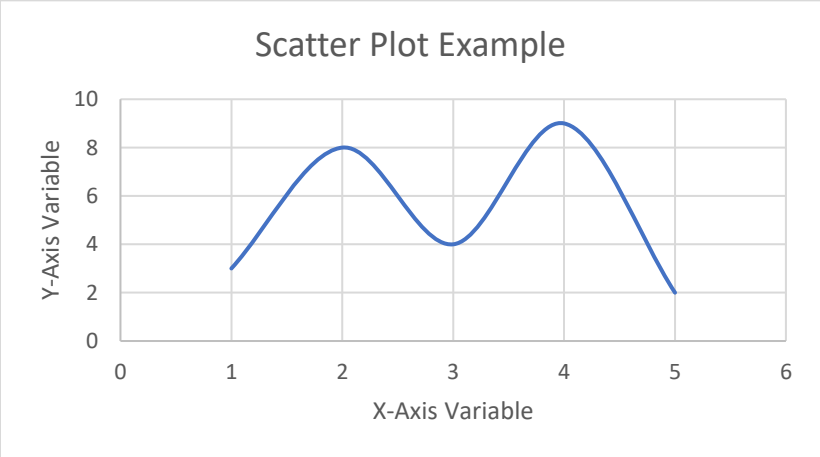
2. Scatter Plot

- **Description**: A scatter plot displays individual data points using Cartesian coordinates. It is typically used to observe relationships or correlations between two numerical variables.

- **Use Case**: Examining the relationship between advertising spend and sales revenue, or comparing height and weight of individuals.

Code Example:

```
x = [1, 2, 3, 4, 5]
y = [3, 8, 4, 9, 2]

plt.scatter(x, y, color='blue')
plt.title("Scatter Plot Example")
plt.xlabel("X-axis Variable")
plt.ylabel("Y-axis Variable")
plt.show()
```



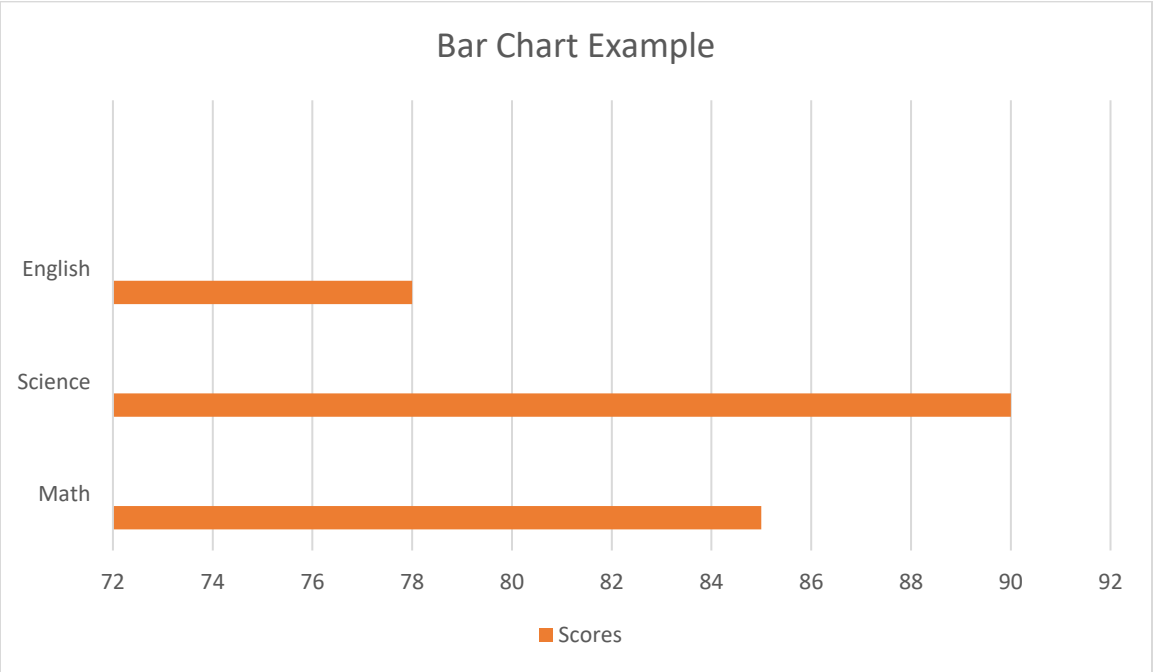
3. Bar Chart

- **Description**: A bar chart represents categorical data using rectangular bars where the length of each bar is proportional to the value it represents.
- **Use Case**: Comparing population sizes by country, revenue by product category, or average test scores by class.

Code Example:

```
categories = ['Math', 'Science', 'English']
scores = [85, 90, 78]
```

```
plt.bar(categories, scores, color='orange')
plt.title("Bar Chart Example")
plt.ylabel("Scores")
plt.show()
```



---

4. Histogram

- **Description**: A histogram is a graphical representation of the distribution of numerical data. It groups data into bins and shows how many values fall into each range.
- **Use Case**: Analyzing the distribution of ages in a population, test score frequencies, or income brackets.

Code Example:

```
import numpy as np
data = np.random.normal(0, 1, 1000)
plt.hist(data, bins=30, color='skyblue', edgecolor='black')
plt.title("Histogram Example")
```

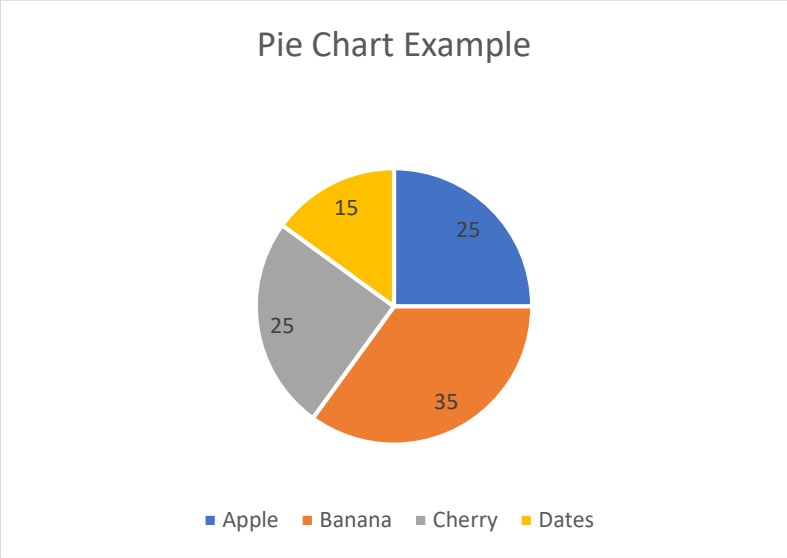
```
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.show()
```

5. Pie Chart

- **Description**: A pie chart is a circular chart divided into sectors, where each sector represents a proportion of the whole.
- **Use Case**: Visualizing percentage distribution of market share, budget allocation, or survey responses.

Code Example:

```
labels = ['Apples', 'Bananas', 'Cherries', 'Dates']
sizes = [25, 35, 25, 15]

plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90)
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.title("Pie Chart Example")
plt.show()
```



PANDAS LIBRARY

Pandas is a powerful Python library used for data manipulation and analysis. It provides data structures and functions needed to work with structured data seamlessly, making it a fundamental tool in data science and analytics.

Core Components of Pandas:

- **Series:** A one-dimensional labeled array capable of holding any data type (integers, strings, floats, etc.). Think of it like a single column in a spreadsheet or database.
- **Data Frame:** A two-dimensional labeled data structure with columns that can be of different types. It's similar to a table in a database or an Excel spreadsheet.
- **Index:** Labels for rows and columns in Series and DataFrames, enabling fast access and alignment of data.

Key Features of Pandas:

- **Data Handling:** Easily load, clean, manipulate, and analyze large datasets from various file formats (CSV, Excel, SQL, JSON, etc.).
- **Data Alignment & Missing Data Handling:** Automatic alignment of data and built-in tools to handle missing or null data gracefully.
- **Data Wrangling:** Powerful group-by operations, reshaping, pivoting, and merging/joining datasets.
- **Time Series Support:** Specialized tools to handle time-stamped data, resampling, and frequency conversion.
- **Efficient Performance:** Optimized for performance using highly efficient C and Cython code under the hood.
- **Integration:** Works well with other libraries like NumPy, Matplotlib, and Scikit-learn for analysis and visualization.

Key Uses of Pandas:

- **Data Cleaning:** Handle missing values, filter data, and transform data formats.
- **Exploratory Data Analysis (EDA):** Summarize data with descriptive statistics and visualize distributions.
- **Data Manipulation:** Sorting, filtering, grouping, aggregating, and pivoting datasets.
- **Time Series Analysis:** Analyze and manipulate data indexed by time.
- **Data Input/Output:** Read/write data from/to various formats such as CSV, Excel, JSON, and SQL databases.

Advantages:

- **User-Friendly Data Structures:** Intuitive Series and DataFrame objects simplify working with tabular data.
- **Rich Functionality:** Comprehensive set of functions for cleaning, transforming, and analyzing data.
- **Speed:** Highly optimized for fast data processing.
- **Integration:** Easily works alongside other popular scientific libraries in Python.
- **Community Support:** Large and active community providing tutorials, extensions, and ongoing development.

Disadvantages:

- **Memory Usage:** Can consume large amounts of memory with very big datasets.
- **Steep Learning Curve for Complex Operations:** Advanced operations like multi-indexing and pivot tables can be challenging for beginners.
- **Performance Bottlenecks:** For extremely large datasets, Pandas may not be as efficient as some specialized tools or databases.

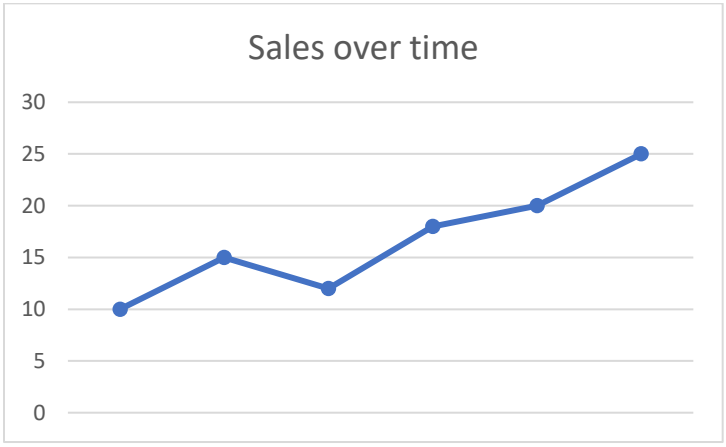
Graph Types and Examples

1. Line Plot

- **Description:**
  A line plot connects data points with a continuous line. It's ideal for showing trends or changes over intervals like time.
- **Use Case:**
  Tracking stock prices over days, monitoring temperature changes over time, or displaying sales trends month-by-month.

Example Code:

```
import pandas as pd
import matplotlib.pyplot as plt
data = pd.Series([10, 15, 12, 18, 20, 22])
data.plot(kind='line', title='Sales Over Time')
plt.show()
```

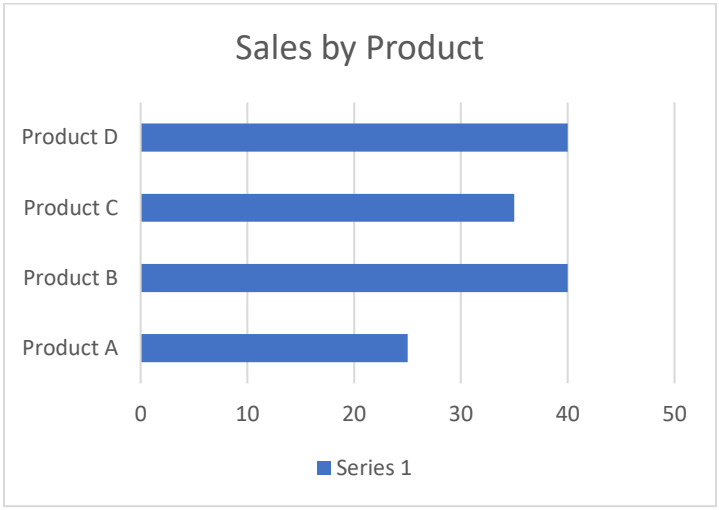


2. Bar Chart

- **Description:**
  Bar charts use rectangular bars to represent quantities for different categories, making it easy to compare values.

- **Use Case:**
  Comparing total sales by product category, population by country, or survey results by group.

Example Code:
data = pd.Series([25, 40, 35, 50], index=['Product A', 'Product B', 'Product C', 'Product D'])
data.plot(kind='bar', title='Sales by Product')
plt.show()



3. Histogram

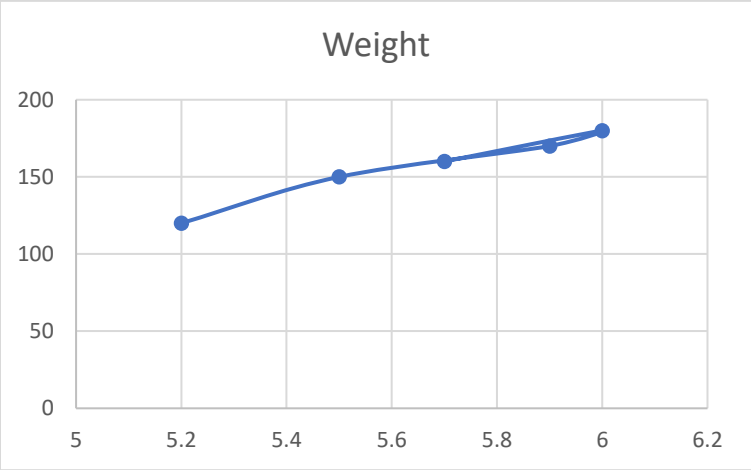
- **Description:**
  A histogram groups numerical data into bins and displays how many data points fall into each bin. It's useful for understanding data distribution.
- **Use Case:**
  Analyzing the age distribution in a population, exam scores distribution, or frequency of customer purchase amounts.

Example Code:
df = pd.DataFrame({'ages': [22, 25, 29, 31, 40, 35, 37, 29, 23, 25, 30]})
df['ages'].plot(kind='hist', bins=5, title='Age Distribution')
plt.show()

4. Scatter Plot

- **Description:**
  Scatter plots show the relationship between two numeric variables with points plotted on x and y axes.
- **Use Case:**
  Examining correlation between height and weight, advertising budget vs sales, or hours studied vs exam score.

Example Code:
df = pd.DataFrame({'height': [5.2, 5.5, 5.9, 6.0, 5.7], 'weight': [120, 150, 170, 180, 160]})
df.plot(kind='scatter', x='height', y='weight', title='Height vs Weight')
plt.show()



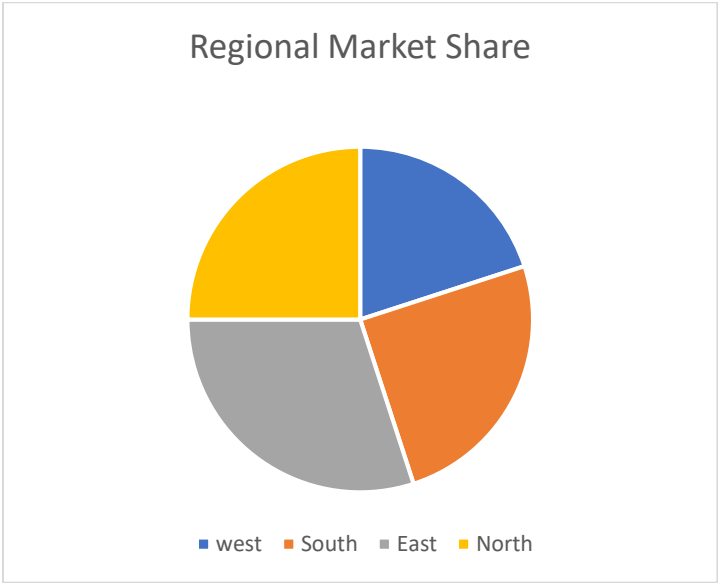
5. Pie Chart

Description:
Pie charts display data as slices of a circle, showing proportional parts of a whole.

Use Case:
Showing market share by company, budget allocation, or population distribution by region.

Code Example:

```
data = pd.Series([30, 20, 25, 25], index=['East', 'West', 'North', 'South'])
data.plot(kind='pie', autopct='%1.1f%%', title='Regional Market Share')
plt.show()
```



Feature	Matplotlib	Seaborn
Ease of Use	Low (More manual setup)	High (Simpler syntax)
Customization	Very High (Full control over plots)	Moderate to High (uses Matplotlib underneath)
Default Aesthetics	Basic and plain	Beautiful and professional-looking by default
Interactivity	Limited (static by default)	Limited (static by default, but integrates with matplotlib)
Statistical Support	Minimal	Excellent (e.g., regression lines, KDE, confidence intervals)
Performance	Efficient for most cases	Efficient, but slightly slower due to built-in calculations
Best For	Publication-quality visuals, full control	Quick statistical analysis and attractive plots