Data Visualization

Advanced Programming

Overview

- **Data**
  - What is data?
  - Sources of data?
- **Overview of data visualization**
- **Importance of data visualization in data science**
  - Why visualization matters in data science
- **Common Types of Data Plots**
  - Bar chart, line plot, scatter plot, histogram, pie chart.
- **Data Visualization Tools**
  - Overview of Python libraries for data visualization
  - Matplotlib, Pandas, Seaborn, Plotly, Bokeh, Pygal, Folium.
- **Introduction to Matplotlib and Seaborn**
  - Basics of Matplotlib
  - Basics of Seaborn
- **Coding:** Python environment and installing libraries

Overview

- **Advanced Plotting Techniques**
  - Subplots, Heatmaps, 3D Plots, Network Graphs, Choropleth Maps, Contour Plots
- Creating Subplots in Matplotlib
- Creating Heatmaps in Seaborn
- **Customizing and Enhancing Visualizations**
  - Adding labels, titles, and legends
  - Customizing font styles and sizes
  - Customizing legends
- Combining Matplotlib and Seaborn for Complex Visualizations.

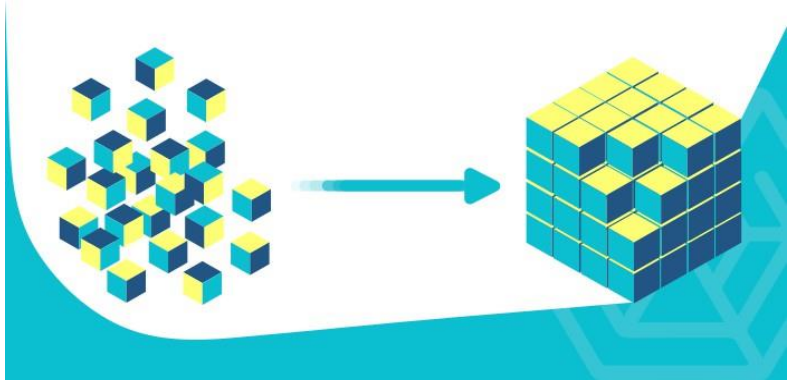
- **Introduction to Time Series Data**
- **Basic Time Series Data Visualization Techniques**
  - Line Plot, Bar Chart, Area Chart
- **Case Study:** Forecasting Stock Prices with LSTM

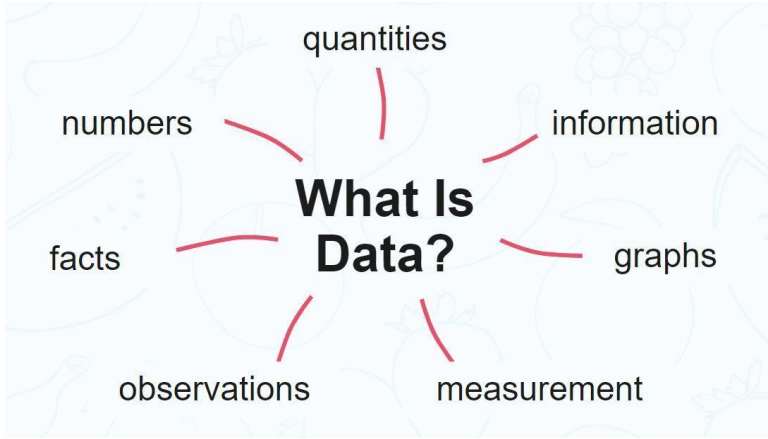
Overview

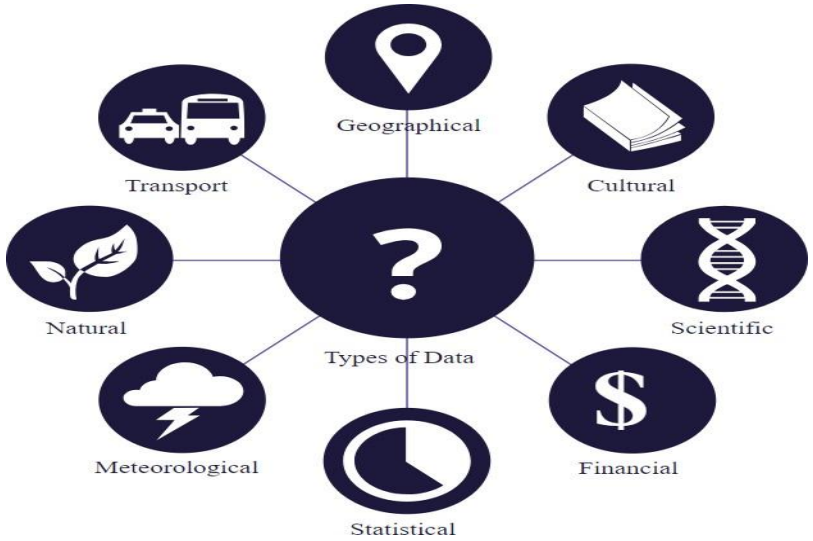
- **Introduction to Interactive Data Visualization**
  - What is Interactive Data Visualization?
  - Why use Interactive Data Visualization?
  - Benefits and Use Cases
- **Getting Started with Plotly**
  - Installation and Setup
  - Basic Plot Types (Line, Scatter, Bar)
- **Advanced Features of Plotly**
  - Animations and Transitions
  - Interactive Widgets
  - Multi-axis and subplots
- **Customizing Plotly Visualizations**
  - Layout and Styling (titles, labels, legends, themes and templates)
- Comparing Plotly with Other Visualization Libraries
- Saving and exporting visualizations
- **Coding:**
  - Practical coding examples for the topics

What is Data?

- Data is information in raw or unorganized form
- A collection of facts, statistics, or information that can be analyzed, processed, and interpreted to derive meaningful insights.
- It's the building block of insights and decisions
- **Data comes in various types:**
  - Numbers
  - Text
  - Images
  - Speech
  - etc.
- **Examples:**
  - Yes, Yes, No, Yes, No, Yes, No, Yes
  - 42, 63, 96, 74, 56, 86
  - None of the above data sets have any meaning until they are given a **CONTEXT** and **PROCESSED** into a useable form
- Information is data that has been processed, organized, or structured in a way that makes it meaningful, valuable and useful.







Sources of Data

1. Primary sources:

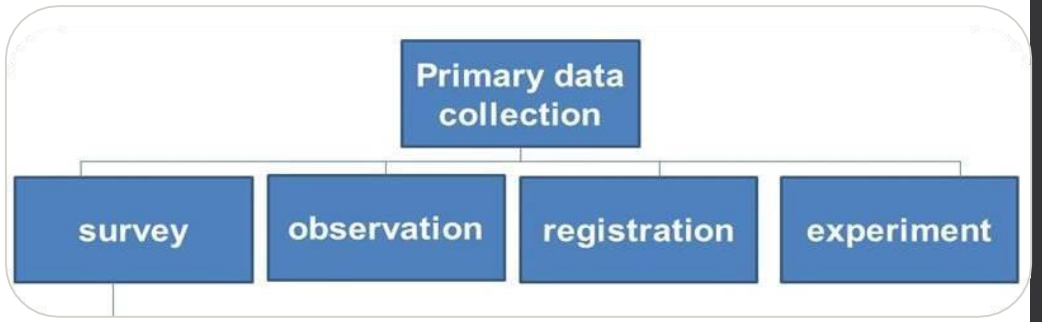
- Surveys and questionnaires
- Experiments and observations

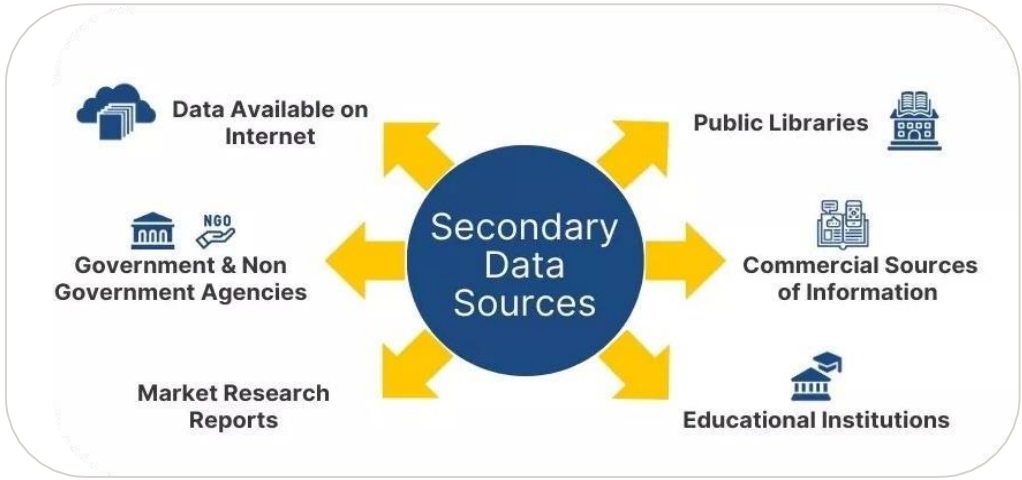
2. Secondary sources:

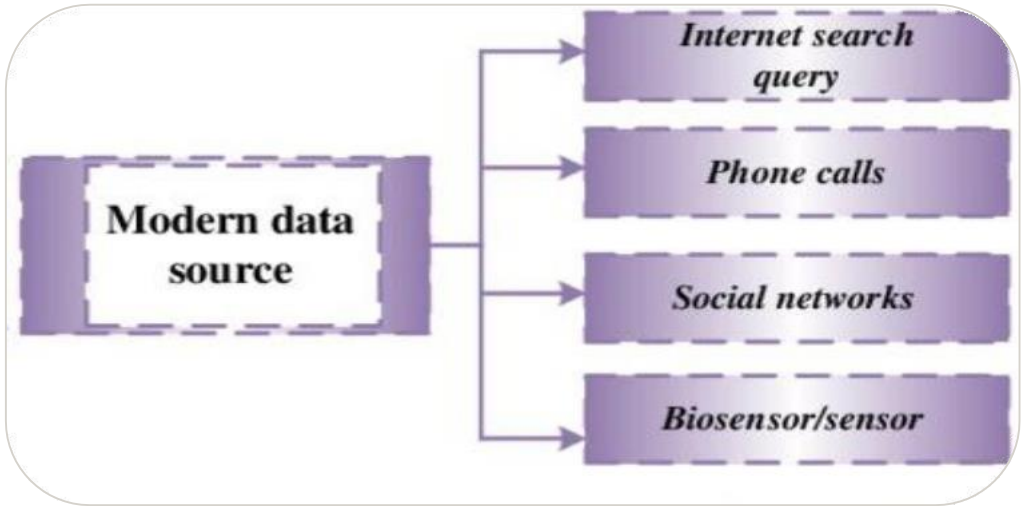
- Databases: Public or private data collections.
- Government databases (e.g., census data)
- Academic research publications
- Web scraping

3. Modern Data sources:

- **Big Data:** Vast datasets from varied sources.
  - User-generated content (social media, online reviews)
  - Business transactions (e-commerce, financial records)
  - Scientific experiments (large-scale research, simulations)
- Sensors and IoT devices (smart cities etc.)
- Surveillance systems (security cameras)
- Medical Data (electronic health records, medical imaging, etc.)



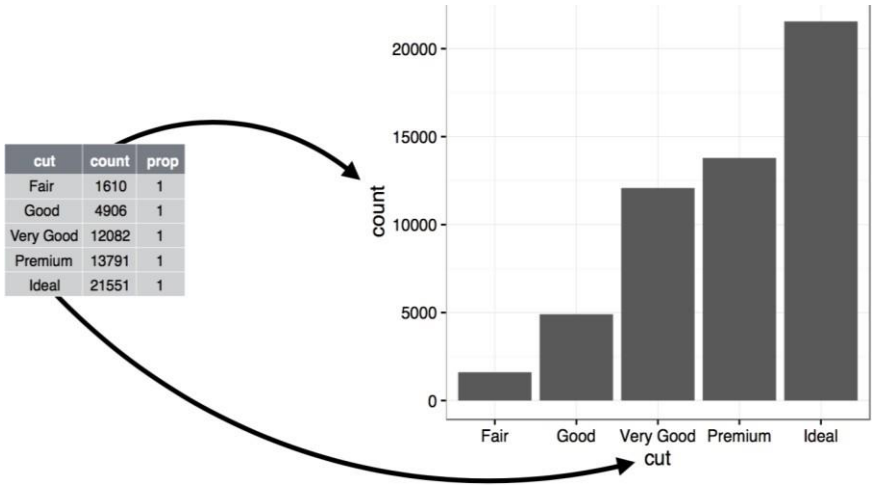


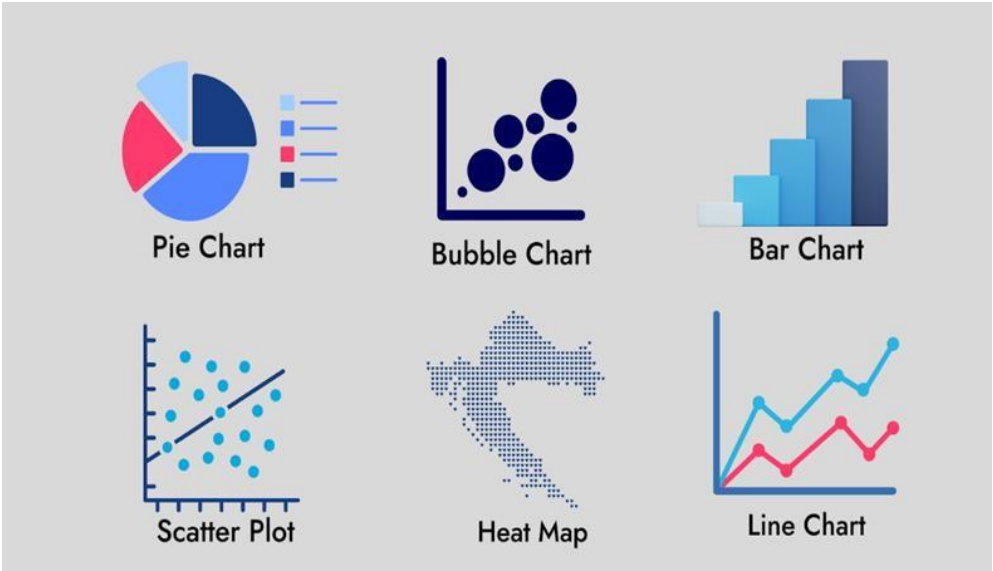


Overview of Data Visualization

What is Data Visualization?

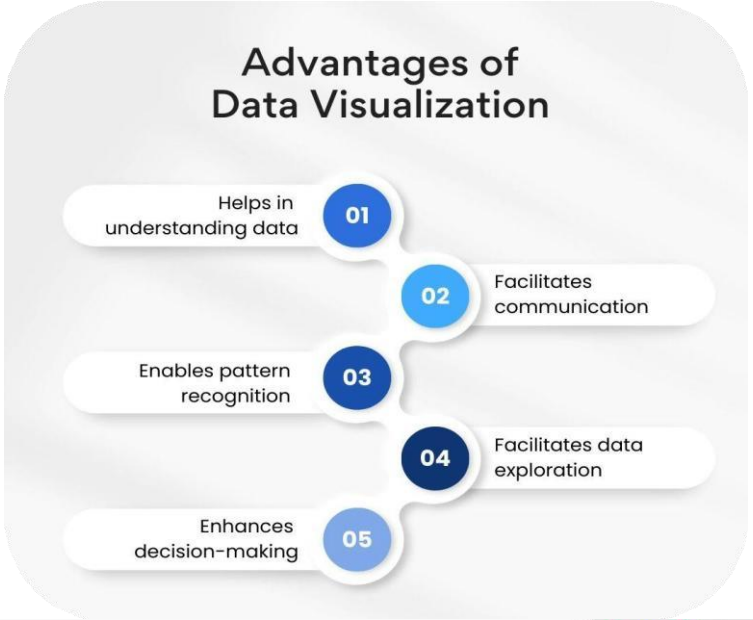
- The graphical representation of information and data using visual elements such as charts, graphs, and maps.
- It is a critical skill in data science, helping transform raw data into understandable and actionable insights.
- **Purpose**:
  - Helps in understanding complex data through visual context.
  - Visualizing data allows people to see relationships, patterns, and trends in the information you're trying to communicate.
- **Examples**:
  - **Charts**: Bar charts, pie charts, etc.
  - **Maps**: Geospatial data visualizations.
  - **Dashboards**: Interactive data summaries.



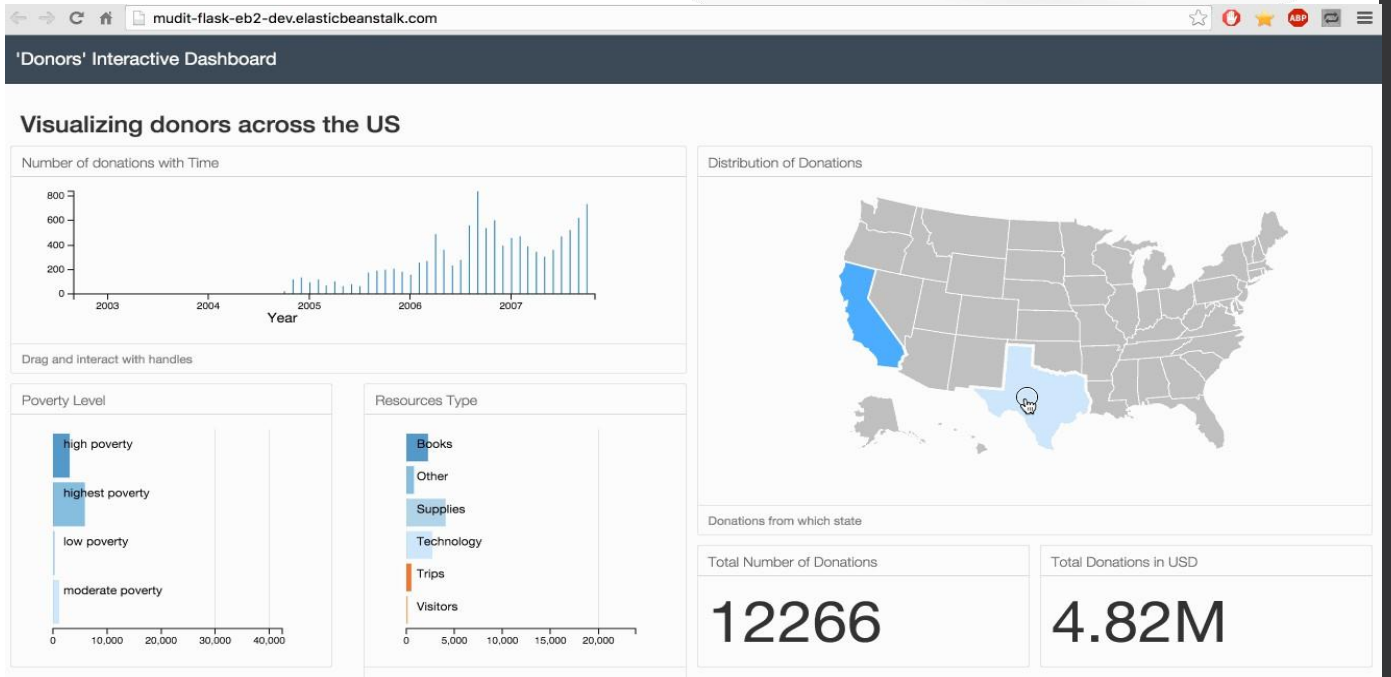


Importance of Data Visualization in Data Science

1. **Trend Analysis:** Identify patterns over time.
2. **Outlier Detection:** Easily spot anomalies.
3. **Insight Communication:** Simplifies complex data for stakeholders.
4. **Decision support:** Enabling quick and informed choices based on visual insights
5. **Pattern recognition:** Identifying trends and anomalies that may be missed in raw data.



Example of an interactive dashboard:

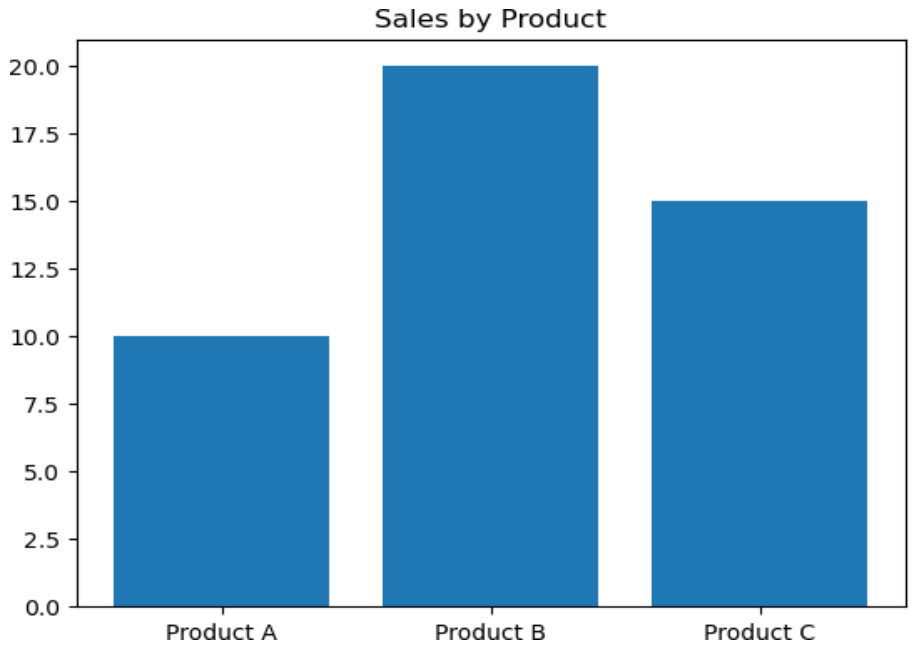


Common Types of Data Plots

Bar Chart

- **Usage**: Comparing quantities across categories.
- **Example**: Sales of different products.
- **Features**:
  - Easy to compare different categories side by side.
  - Useful for categorical data.

```
plt.bar(['Product A', 'Product B', 'Product C'], [10, 20, 15])
plt.title('Sales by Product')
plt.show()
```



Line Plot

- **Usage**: Show trends over time
- **Example**: Stock prices over a year
- **Features:**
  - Illustrates continuous data and trends.
  - Ideal for time series data.
- **Code:**

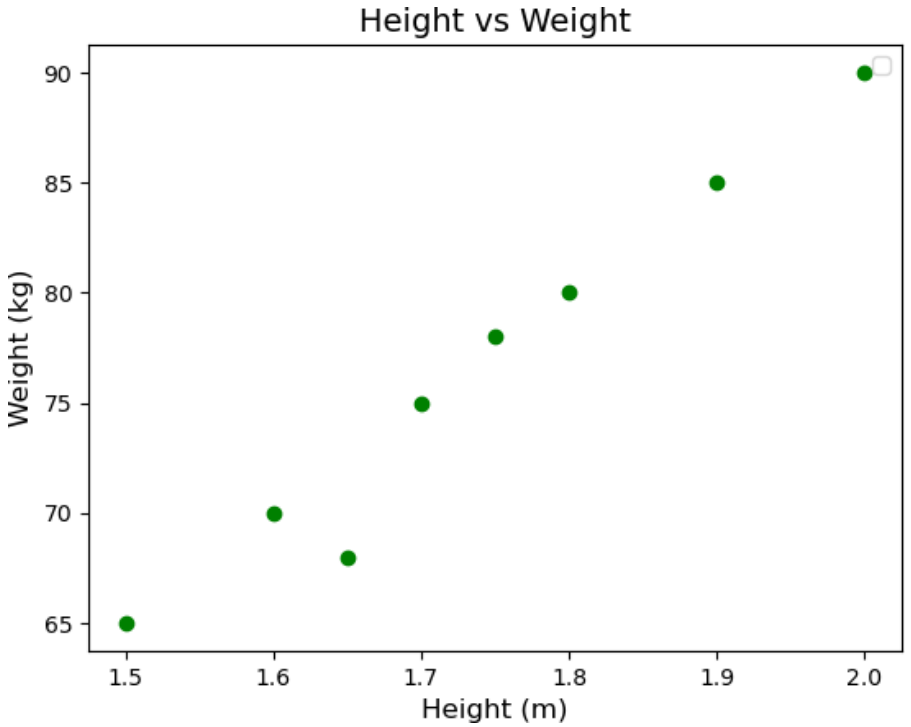
```
x = [1, 2, 3, 4]
y = [10, 20, 25, 30]
plt.plot(x, y, marker='o', linestyle='-', color='b', label='Stock A')
plt.title('Stock Prices Over Time', fontsize=14)
plt.xlabel('Time (Days)', fontsize=12)
plt.ylabel('Stock Price (USD)', fontsize=12)
plt.grid(True)
plt.legend()
plt.show()
```

Common Types of Data Plots

Scatter Plot

- **Usage:** Show the relationship between two variables.
- **Example**: Height vs. weight.
- **Code:**

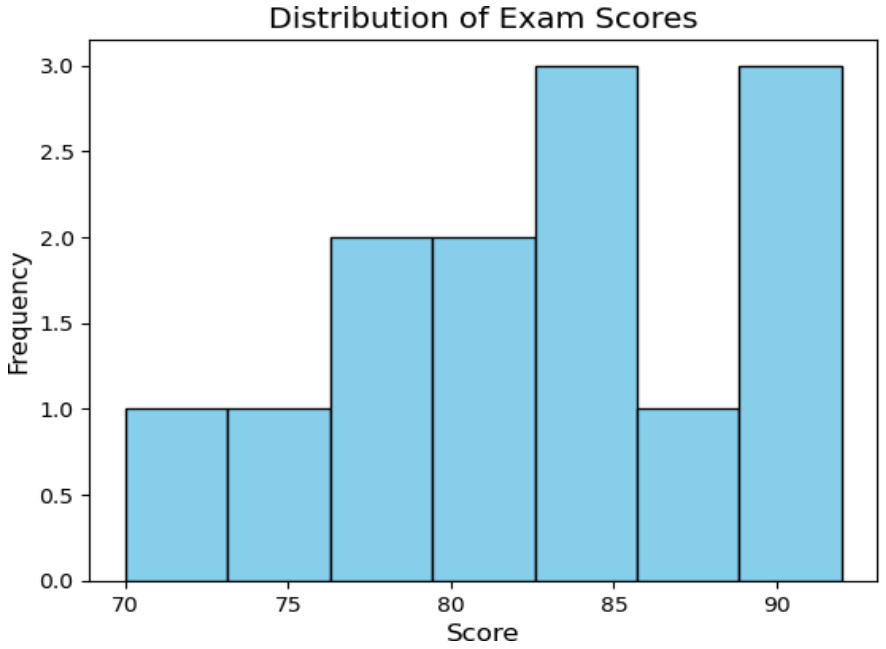
```
heights = [1.5, 1.8, 1.6, 2.0, 1.7, 1.9, 1.75, 1.65]
weights = [65, 80, 70, 90, 75, 85, 78, 68]
plt.scatter(heights, weights, color="g", marker='o')
plt.title('Height vs Weight', fontsize=14)
plt.xlabel('Height (m)', fontsize=12)
plt.ylabel('Weight (kg)', fontsize=12)
plt.show()
```



Histogram

- **Usage:** Display the frequency distribution of a variable.
- **Example:** Distribution of exam scores.
- **Code:**

```
scores = [70, 85, 90, 75, 80, 90, 85, 80, 88, 92, 77, 84, 79]
plt.hist(scores, bins = 7, color='skyblue', edgecolor='black')
plt.title('Distribution of Exam Scores', fontsize=14)
plt.xlabel('Score', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.show()
```

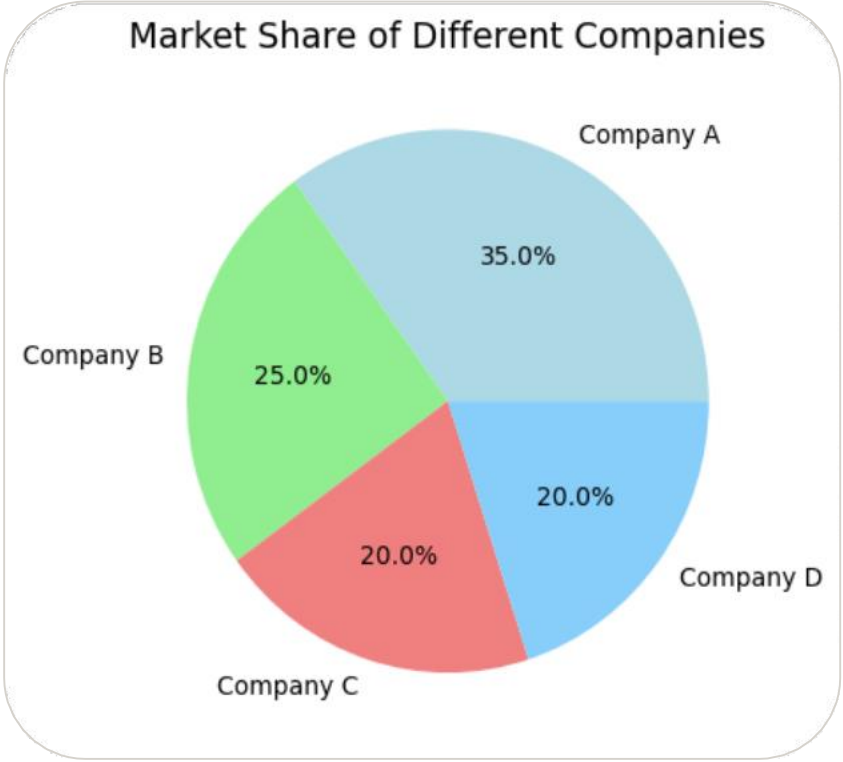


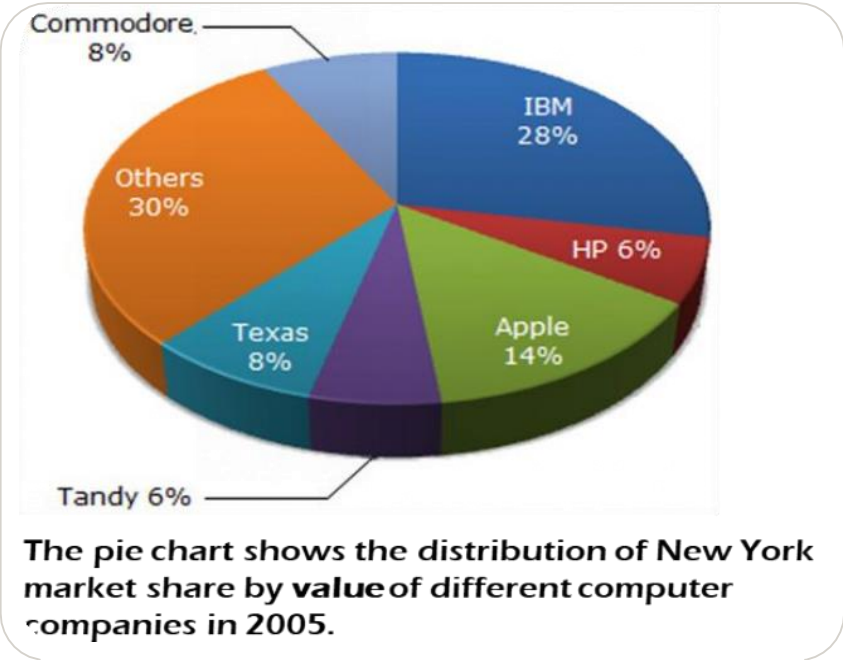
Common Types of Data Plots

Pie Charts

- **Usage**: Show proportions within a whole.
- **Example**: Market share of different companies.
- **Note**: Use sparingly, as they can be hard to interpret accurately
- **Code**:

```
companies = ['Company A', 'Company B', 'Company C', 'Company D']
market_share = [35, 25, 20, 20]

plt.pie(market_share, labels=companies, autopct='%1.1f%%',
colors=['lightblue', 'lightgreen', 'lightcoral', 'lightskyblue'])

plt.title('Market Share of Different Companies', fontsize=14)
plt.show()
```





The pie chart shows the distribution of New York market share by **value** of different computer companies in 2005.

Data Visualization Tools

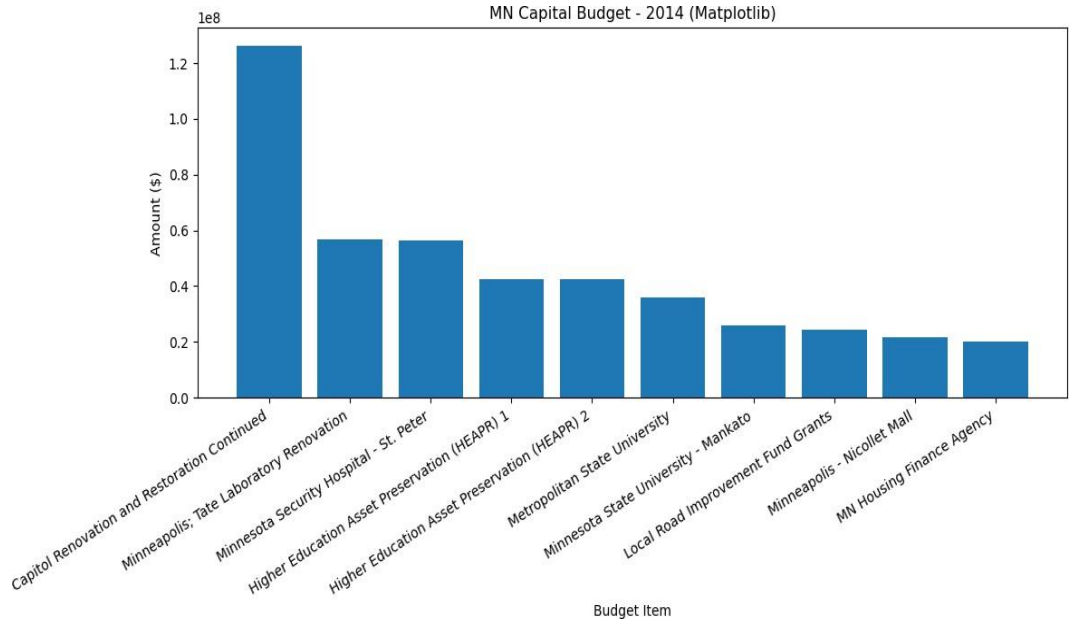
Data Visualization Tools

Overview of Python Libraries for Data Visualization

- Python has become a dominant language in data science, offering a rich ecosystem of visualization libraries.

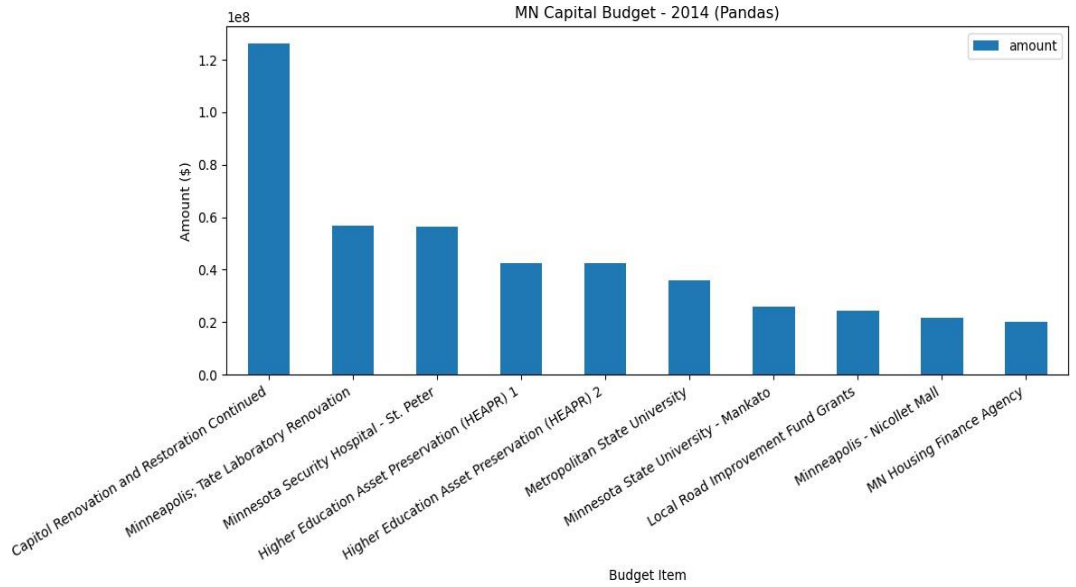
1. Matplotlib

- Matplotlib is the grandfather of python visualization packages.
- Foundation for many other libraries.
- It is extremely powerful but with that power comes complexity.
- Best for creating publication-quality static visualizations.
- **Example**: Line plots, bar charts.



2. Pandas

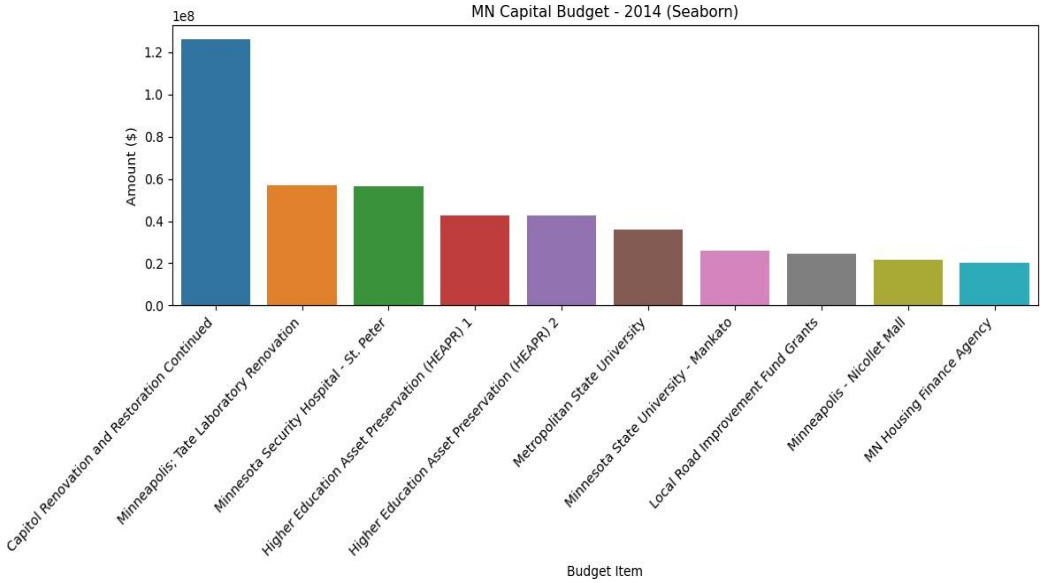
- Built-in plotting functions based on Matplotlib.
- **Features:**
  - Convenient for quick visualizations directly from DataFrames.
  - Ideal for simple, exploratory plots.
- **Use Case**: Rapid, straightforward plotting.
- **Example:** Quick plots from Dataframes.



Overview of Python Libraries for Data Visualization

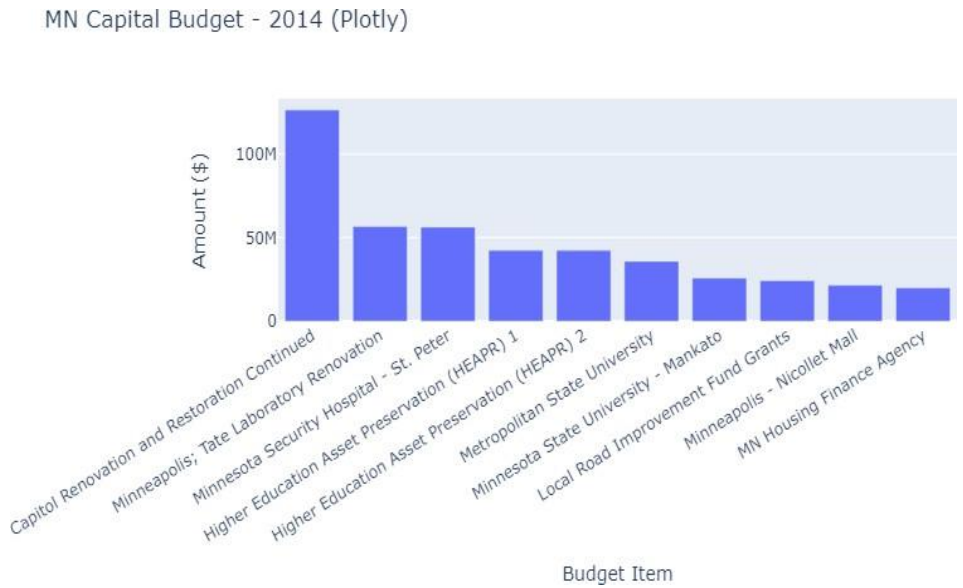
3. Seaborn

- Visualization library based on Matplotlib.
- **Features:**
  - Leverages matplotlib for beautiful, minimal-code charts.
  - It's default styles and color palettes are more modern and visually appealing.
  - It also has the goal of making more complicated plots simpler to create.
  - Specialized in statistical visualizations.
- **Example**: Heatmaps, pair plots.



4. Plotly

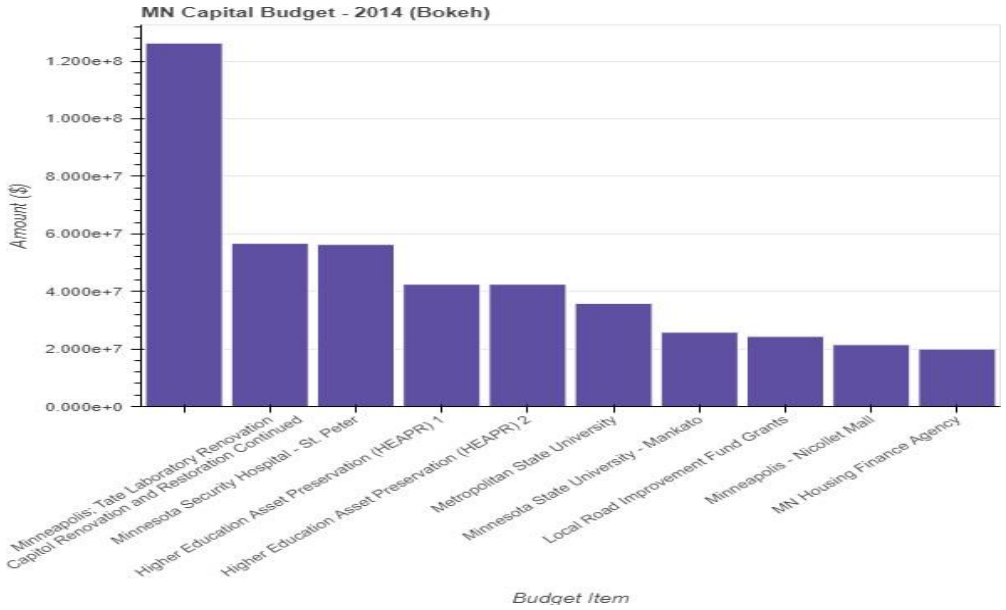
- **Overview:** Creates interactive, web-based visualizations.
- **Features:**
  - Supports diverse chart types and maps.
  - Ideal for dashboards and web applications.
  - Handles real-time data and complex interactivity.
- **Example:** 3D plots, interactive dashboards.



Overview of Python Libraries for Data Visualization

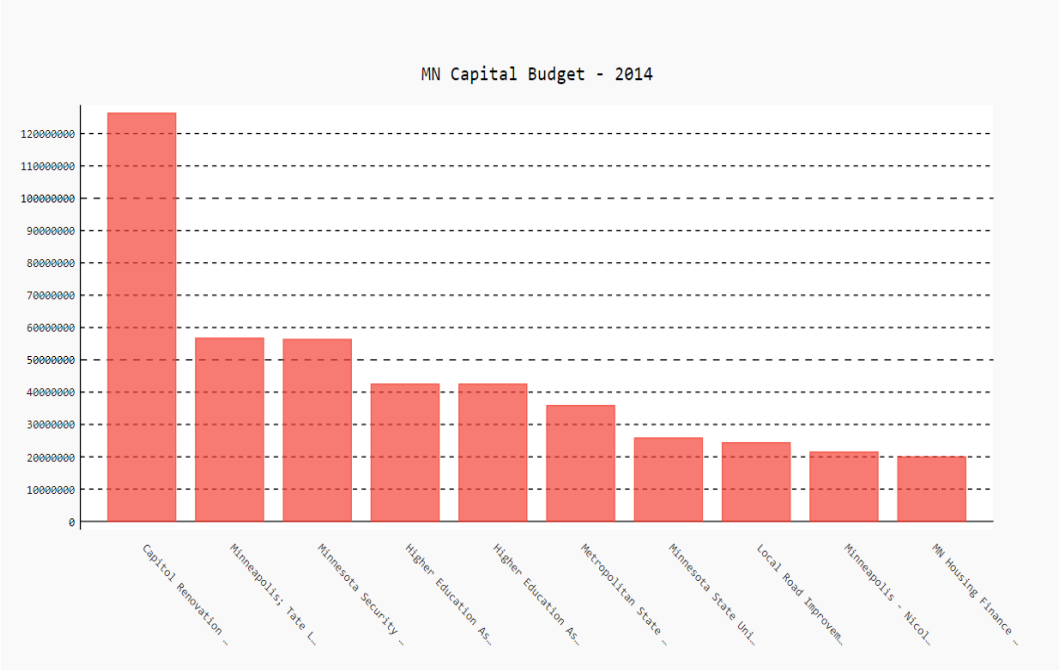
5. Bokeh

- **Overview**: Python library for interactive plots and applications.
- **Use Case:** Web-based visualizations and large datasets.
- **Features:** Supports streaming and real-time data.
- **Example:** Interactive web-based visualizations.



6. Pygal:

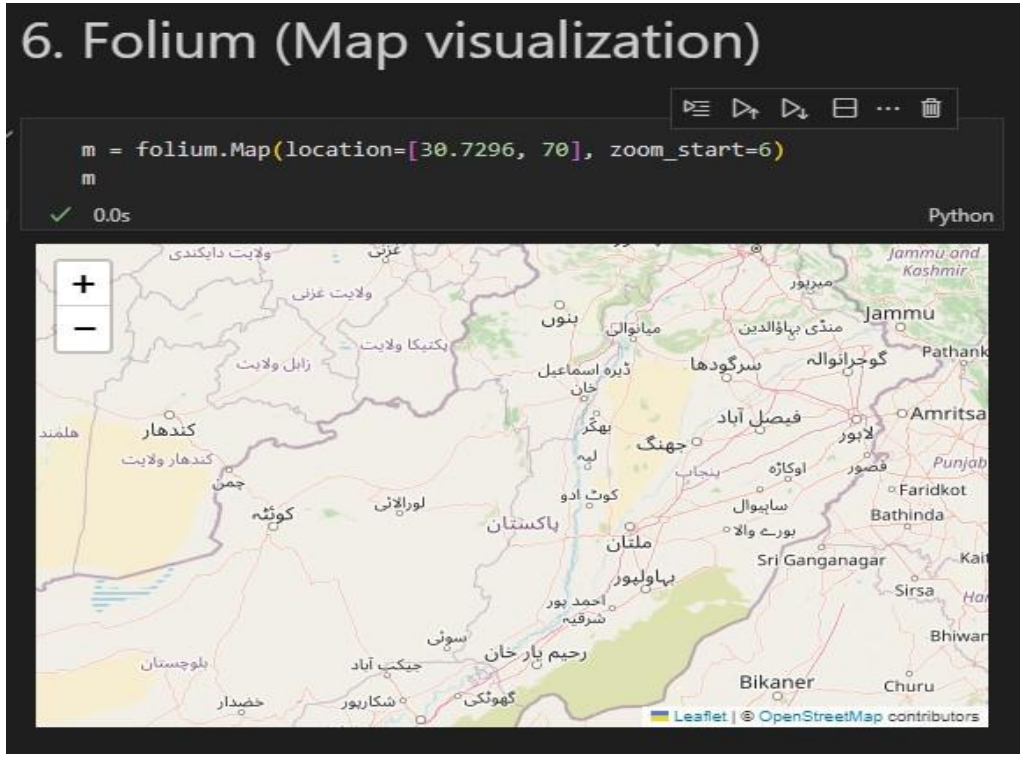
- **Overview:** Python library for creating SVG charts.
- **Features:** Generates interactive, high-quality vector graphics.
- **Use Case**: Ideal for producing visually appealing and scalable charts.
- **Example:** Interactive and customizable charts for web applications.



Overview of Python Libraries for Data Visualization

7. Folium

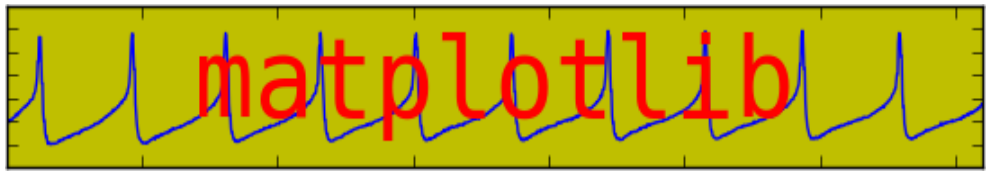
- **Overview**: Python library for interactive maps.
- **Backend**: Utilizes Leaflet.js (a popular JavaScript library for interactive maps).
- **Use Case**: Geospatial data visualization.
- **Features**: Specializes in creating interactive, mobile-friendly maps.
- **Example**: Visualizing geographic data interactively.



Introduction to Matplotlib and Seaborn

Matplotlib:

- **Initial release:** 2003
- First Python data visualization library.
  - Most popular and widely used data visualization library.
- Matplotlib is the grandfather of python visualization packages.
  - Foundation for many other libraries.
  - Popular libraries like Seaborn and Plotly are built on Matplotlib, using it as the core for rendering plots.
- **Highly Customizable:**
  - Low-level, highly customizable plotting library.
  - It offering fine-grained control over plot elements (axes, labels, ticks, colors), enabling a wide range of visualizations from basic bar charts to complex interactive 2D graphs.
- **Key Strengths:**
  - **Powerful but Complex:**
    - While flexible, it can be challenging for beginners compared to higher-level libraries like Seaborn.
  - **Multiple Visualization Modes:**
    - Supports static, animated, and interactive plots for diverse cases.



Matplotlib's original logo (2003 -- 2008).

Matplotlib's logo (2008 -- 2015).

Basics of Matplotlib

Core Components:

- **Figure:** The overall window or page that everything is drawn on.
- **Axes:** A single plot within the figure.

Common Plot Types:

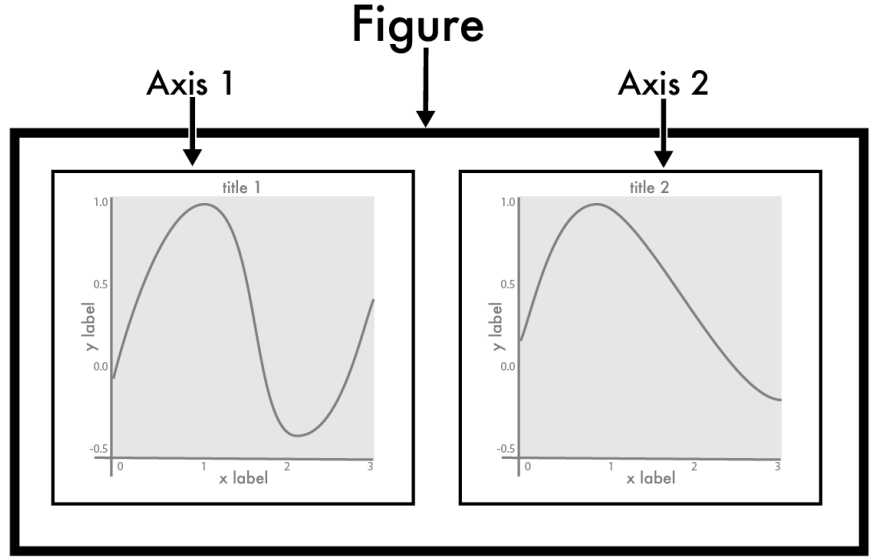
- Line plot, bar chart, scatter plot, histogram.
- **Basic Plotting Functions:**
  - **plot()**: Creates line plots.
  - **scatter():** For scatter plots.
  - **bar():** For bar charts.

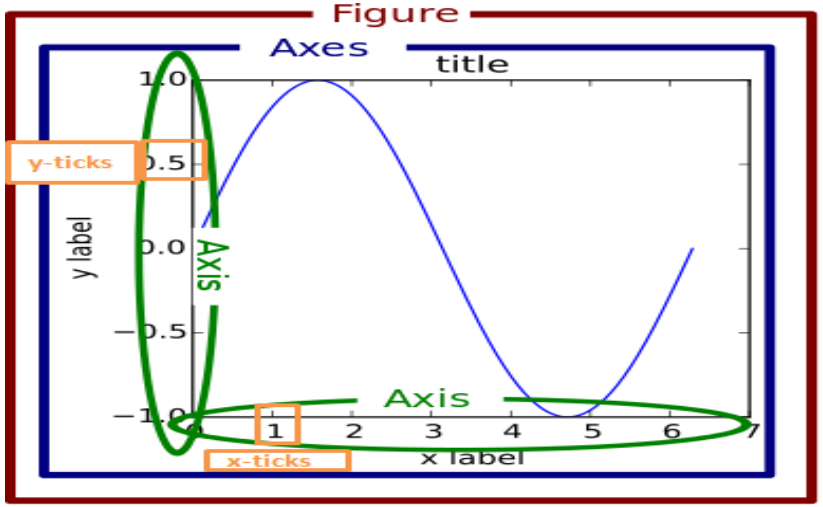
Customization:

- Change colors, labels, titles, legends and more with ease.

Example Code:

```
import matplotlib.pyplot as plt
x, y = [1, 2, 3, 4], [1, 4, 9, 16]
plt.plot(x, y)
plt.ylabel('Squared Numbers')
plt.show()
```





Seaborn:

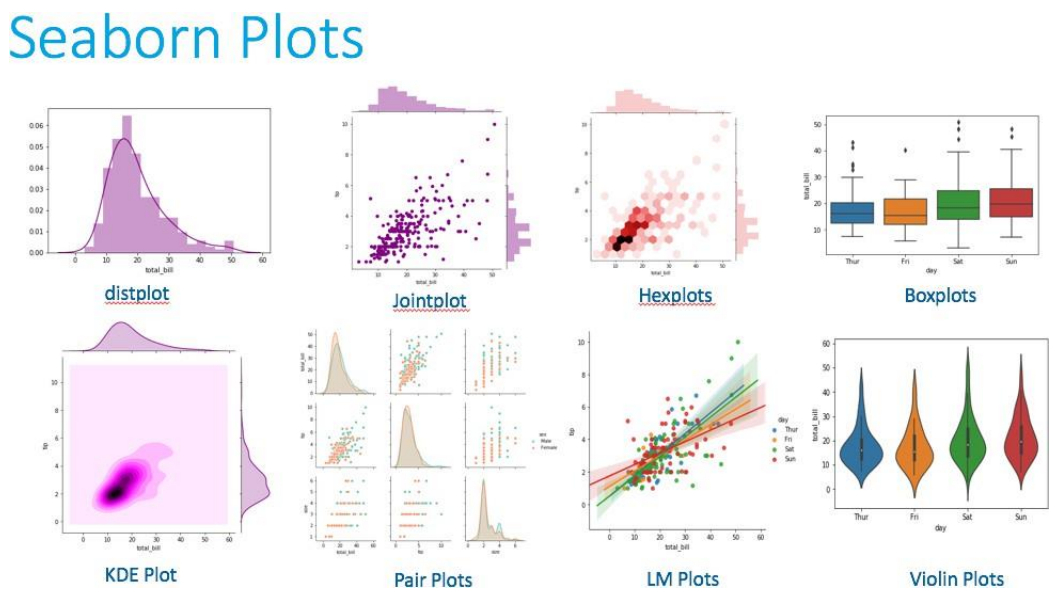
- **Initial Release:** 2014
- It is an advanced data visualization library built on top of Matplotlib, designed to make complex statistical plots easier to create.

Key Features

- **High-Level Interface**
  - Intuitive API for statistical graphics
  - Minimal effort required
- **Advanced Visualizations**
  - Built-in functions for complex plots
  - **Examples:** heatmaps, pair plots, violin plots, regression plots
- **Data Integration**
  - Seamless integration with Pandas DataFrames
  - Efficient handling of large datasets

Key Strengths

- **User-Friendly and Less Complex**
  - Simplifies statistical visualizations
  - Maintains flexibility for customization
- **Default Aesthetic Settings**
  - Default style is suitable for publication
  - More refined than basic Matplotlib plots



Basics of Seaborn

Core Features:

- Built on top of Matplotlib
- Built-in themes and color palettes for aesthetic visualizations.
- Supports complex statistical plots (e.g., pair plots, heatmaps).

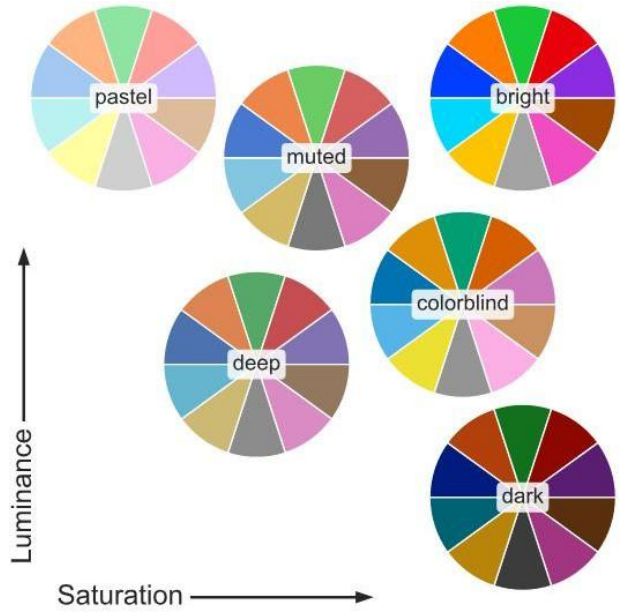
Integration with Pandas:

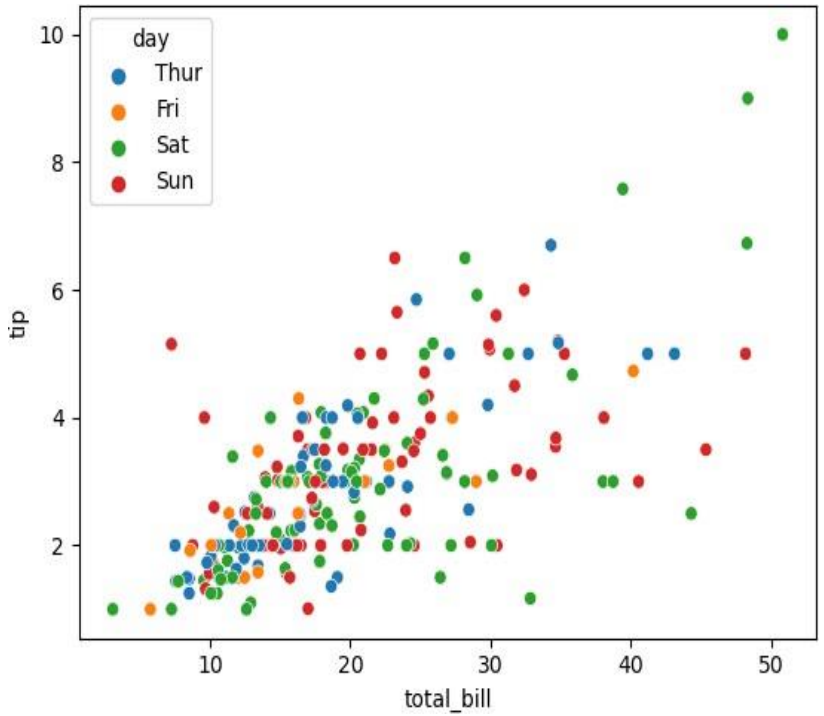
- Seamless integration with Pandas DataFrames

Example Code:

```
import seaborn as sns

tips = sns.load_dataset('tips')

sns.scatterplot(data=tips, x='total_bill', y='tip',
hue='day')
```

- Output: A scatter plot visualizing tips based on the total bill, differentiated by day.





Coding: Python Environment Setup

To get started with data visualization in Python, follow these steps:

1. **Install Python:** Download and install the latest version of Python from python.org.

2. Set up a virtual environment (optional but recommended):

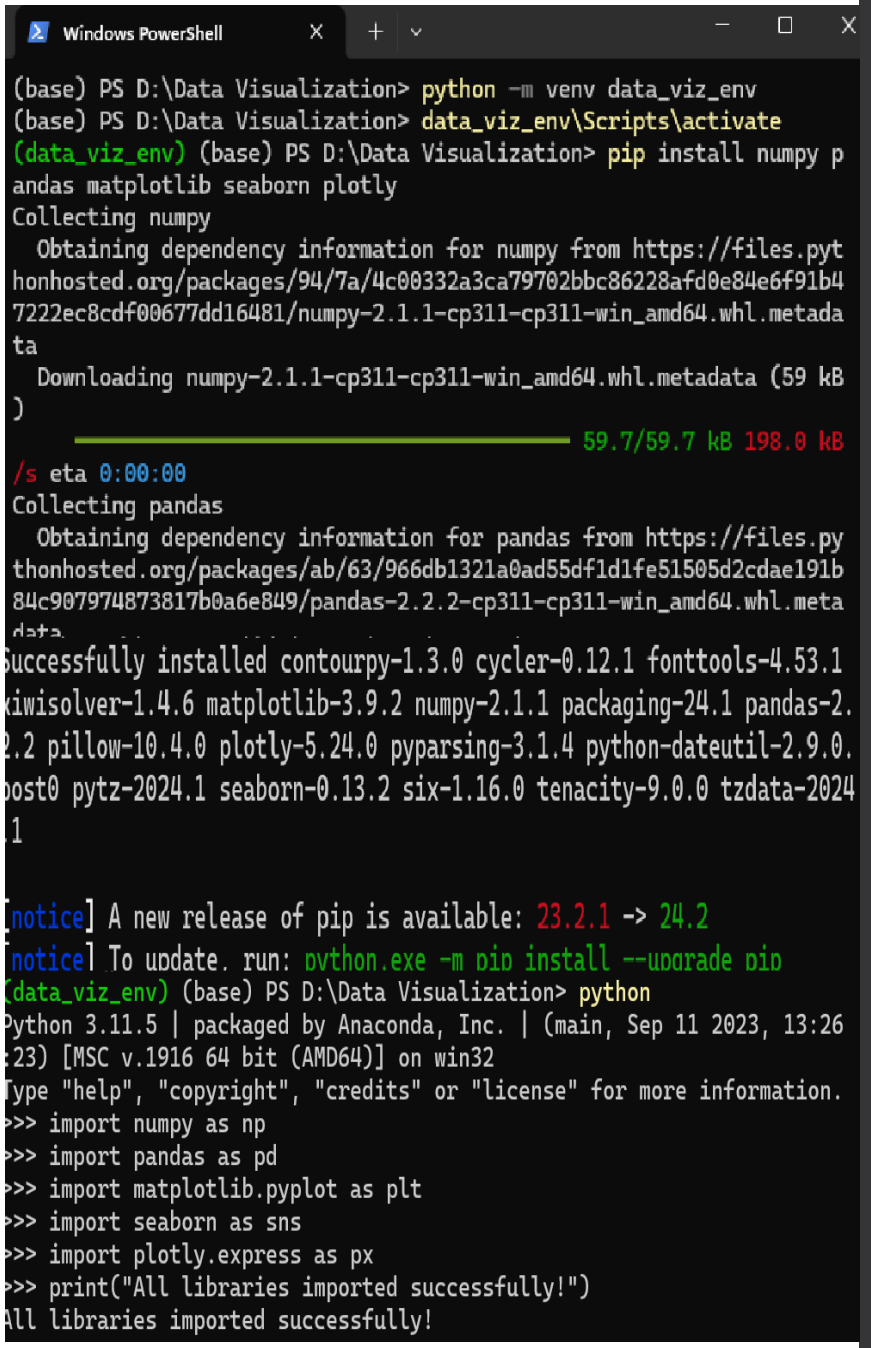
```
python -m venv data_viz_env
data_viz_env\Scripts\activate
```

3. **Set up a coding environment:**
   - **Jupyter Notebook:** Ideal for interactive coding.
   - **VS Code /PyCharm:** Full featured IDEs.

```
pip install numpy pandas matplotlib seaborn plotly
```

5.Verify installations:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
print("All libraries imported successfully!")
```



Advanced Plotting with Matplotlib & Seaborn

Advanced Plotting Techniques

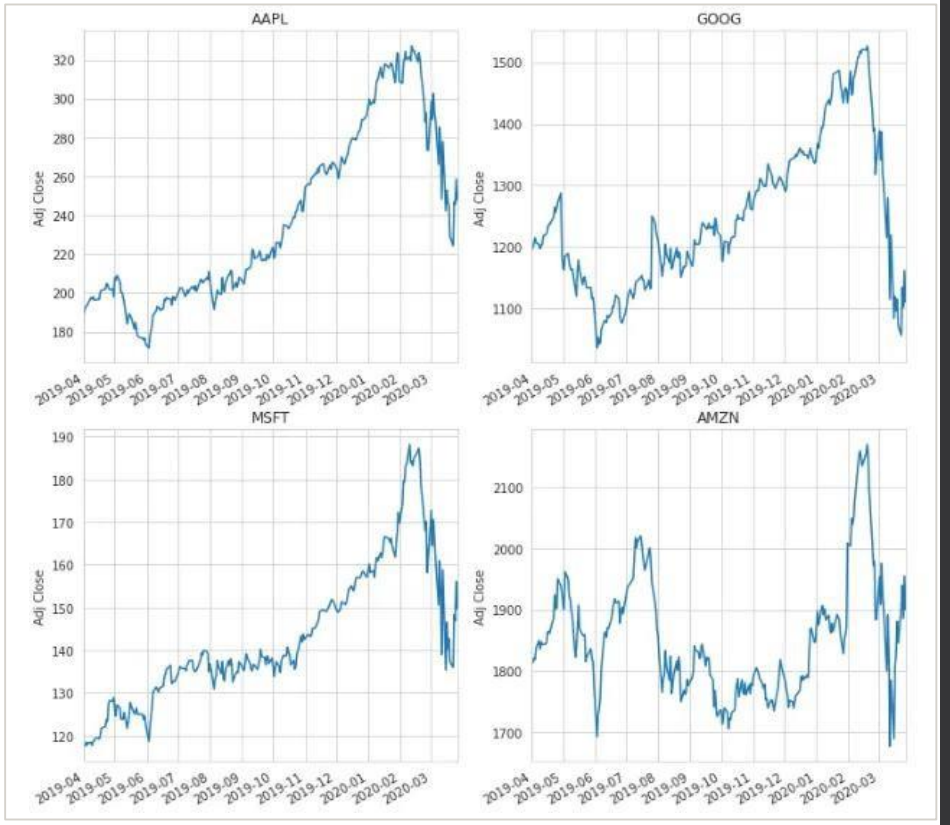
- Advanced plotting techniques offer sophisticated visualizations for exploring complex, multidimensional datasets.
- These methods reveal patterns, correlations, and insights that simpler charts may overlook.

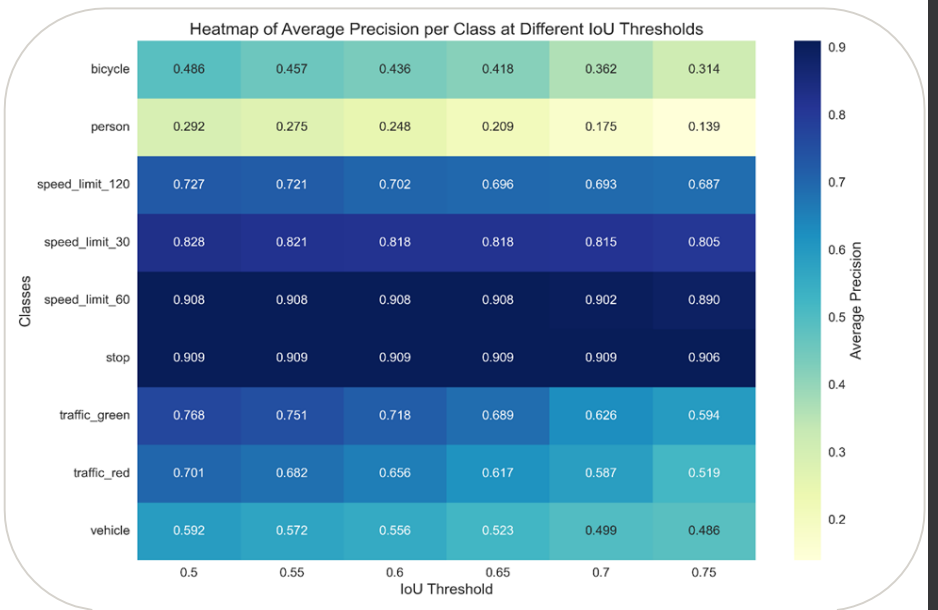
Subplots (multiple plots in one figure):

- **Use case:** Comparing different visualizations side by side.
- **Example:**
  - Line plots comparing stock prices of multiple companies over time.
- **Features:**
  - Enables comparison of multiple visualizations in a single figure.
  - Enhances clarity reduces clutter.
  - Allows for easy identification of trends and patterns across different datasets.

Heatmaps:

- **Use case:** Represent data points in a matrix format, where colors indicate magnitude or intensity
- **Example:** Visualizing correlation between variables in a dataset.
- **Features:**
  - Colors help easily identify high or low correlation points.
  - Ideal for visualizing correlations, confusion matrices, or any grid- based data.
  - Customizable color palettes for better clarity.

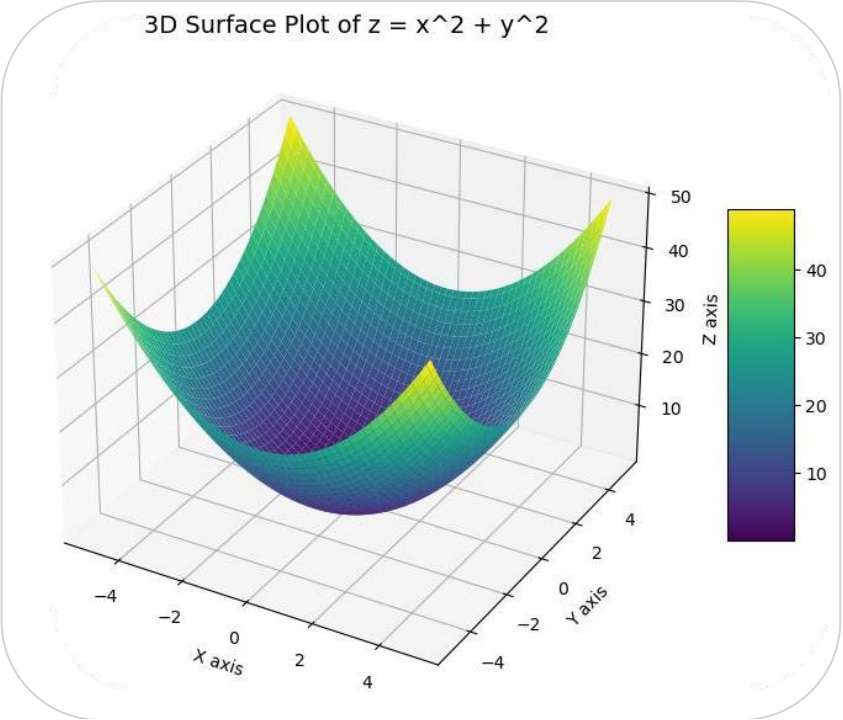




Advanced Plotting Techniques

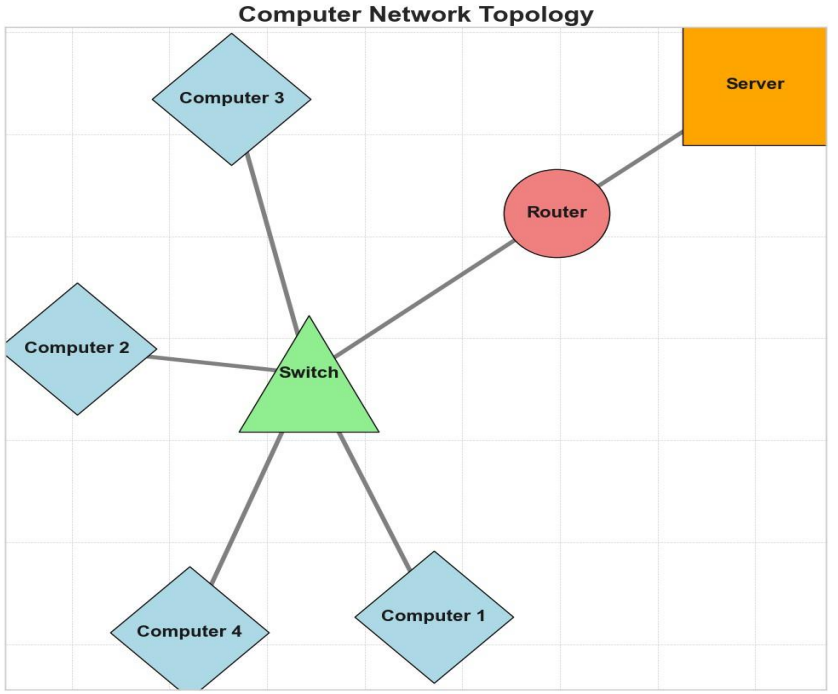
3D Plots:

- **Use case:**
  - Visualizing data in three dimensions to offer a deeper understanding of complex relationships.
- **Example**:
  - Visualize mathematical function. (e.g., $z^2 = x^2 + y^2$) using a 3D surface plot.
- **Features:**
  - Adds an extra dimension (z-axis) for visualizing spatial relationships.
  - Commonly used for surface plots, 3D scatter plots and contour plots.
  - Helps identify relationships between variables that are hard to see in 2D.
  - Allows rotation and interaction for better data exploration.



Network Graphs:

- **Use case:**
  - Visualize relationships and connections between entities in complex systems.
- **Example:**
  - Computer network topology visualization
- **Features:**
  - Ideal for analyzing and visualizing complex relationships and interconnections.
  - Can highlight clusters, hubs, and central nodes in a network.
  - Useful for understanding relationships in social, biological, and communication networks.

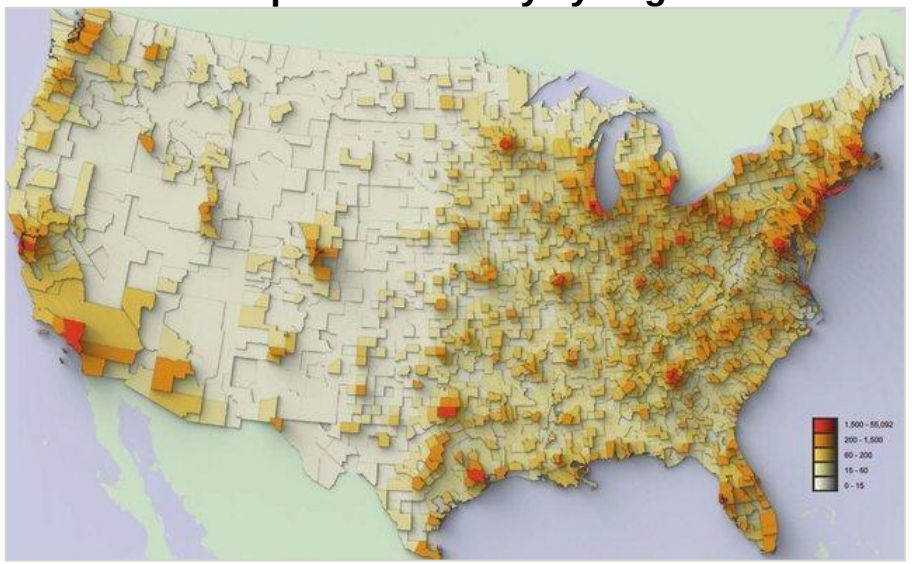


Advanced Plotting Techniques

Choropleth Maps:

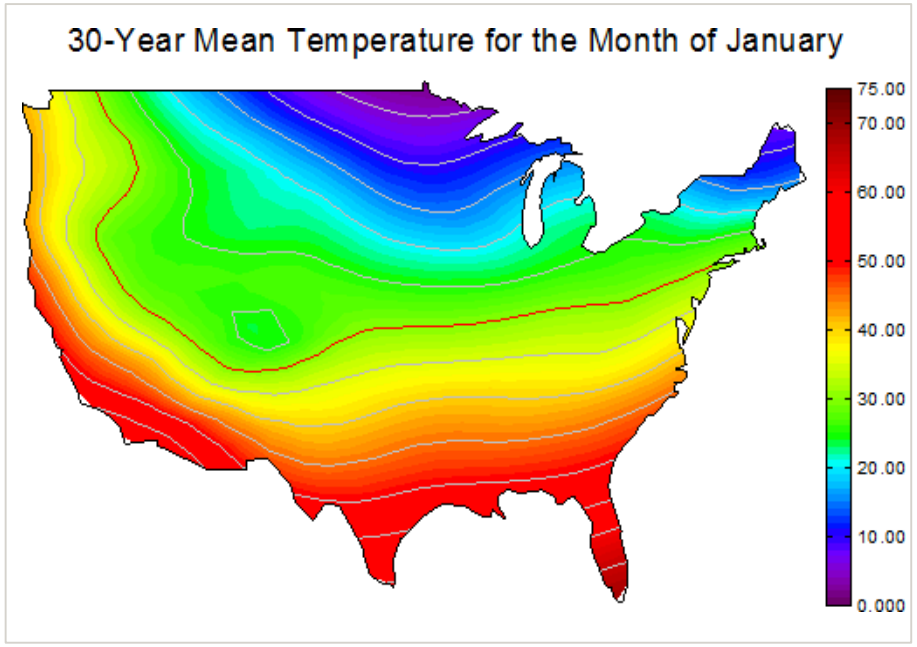
- **Use case:**
  - Displaying data distribution across geographic regions.
- **Example:**
  - Visualizing population density across different countries or states.
- **Features:**
  - Uses color gradients to represent data values over geographic areas.
  - Ideal for showing data trends across spatial regions.
  - Customizable color palettes to highlight key data points, allowing for clearer differentiation between regions.

Choropleth Map of U.S.
Population Density by Region



Contour Plots:

- **Use Case:**
  - Often used to represent 3D data in two dimensions using contour lines to show different levels.
- **Example:**
  - Visualizing weather patterns like pressure or temperature levels.
- **Features:**
  - Contour lines indicate areas of equal value, helping to identify trends and patterns in the data.
  - Useful for visualizing gradients and changes in continuous data.
  - Can display complex relationships between three variables on a 2D plane
  - Helps in understanding topographical information in fields like



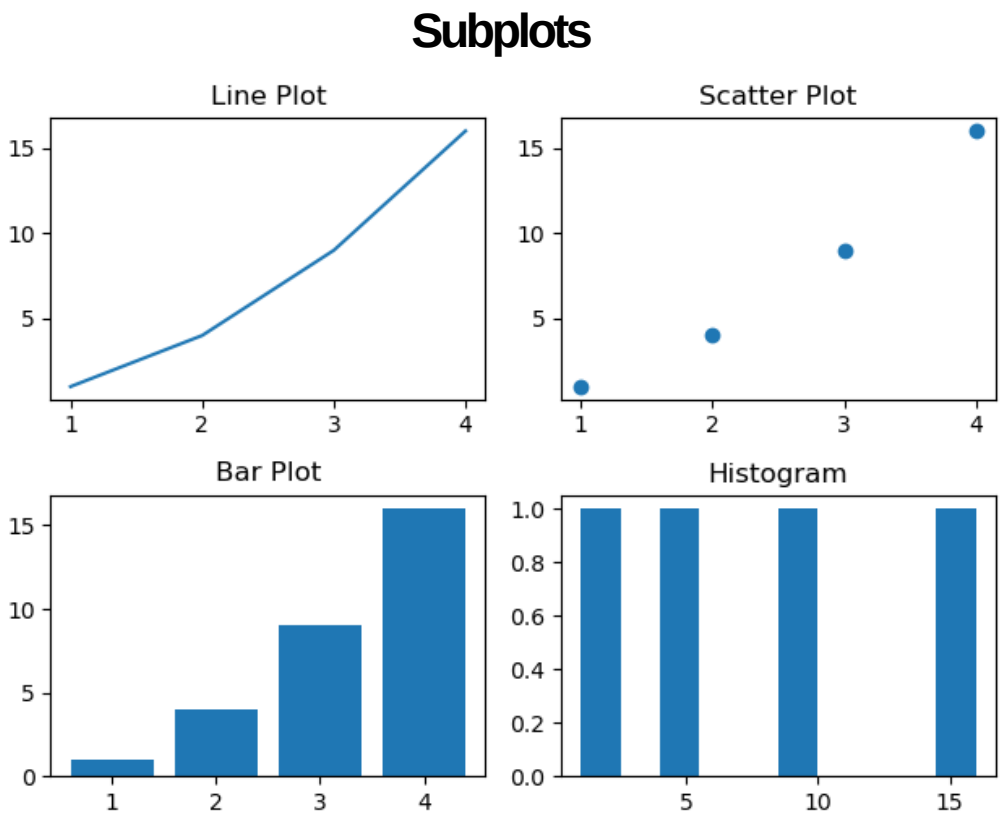
Creating Subplots in Matplotlib

Subplots (multiple plots in one figure):

- **Use Case:** Comparing different visualizations side by side.
- **Code:**

```
x = [1,2, 3,4]
y = [1, 4, 9, 16]

fig, axs = plt.subplots(2, 2)
axs[0, 0].plot(x, y), axs[0, 0].set_title('Line Plot')
axs[0, 1].scatter(x, y), axs[0, 1].set_title('Scatter Plot')
axs[1, 0].bar(x, y), axs[1, 0].set_title('Bar Plot')
axs[1, 1].hist(y), axs[1, 1].set_title('Histogram')
plt.tight_layout()

plt.show()
```

- **Best practices for readability:**
  - Use appropriate spacing (`plt.tight_layout()`) to avoid overlapping elements.
  - Add descriptive titles and labels to each subplot

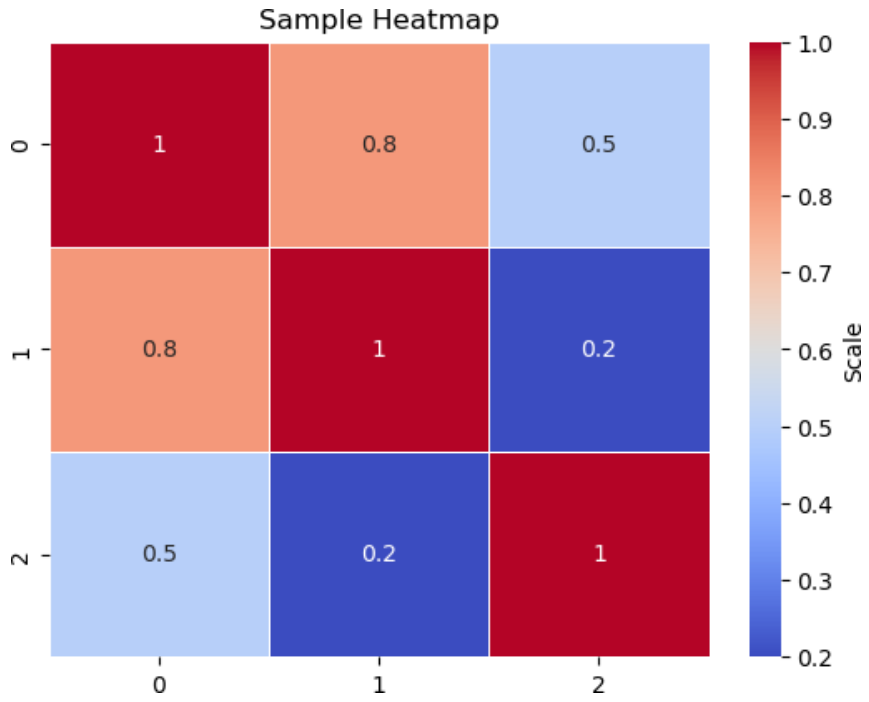


Creating Heatmaps in Seaborn

- **Use case:**
  - Represent data points in a matrix format.
  - Colors indicate magnitude or intensity of values.
  - Ideal for visualizing correlations, confusion matrices, or any grid-based data.
- **Code:**

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

data = [[1, 0.8, 0.5], [0.8, 1, 0.2], [0.5, 0.2, 1]]

sns.heatmap(data, annot=True, cmap='coolwarm',
linewidths=0.5, cbar_kws={'label': 'Scale'})
plt.title('Sample Heatmap')
plt.show()
```

- **Best Practices for Readability:**
  - **Annotate Values:** Use `annot=True` to display data values directly on the heatmap.
  - **Color Maps**: Select colormaps that improve contrast and interpretation (coolwarm, viridis, etc.).
  - **Axis Labels:** Add clear and descriptive labels for both axes.
  - **Figure Size:** Adjust the size to prevent overcrowding (`plt.figure(figsize=(8,6))`)



Customizing and Enhancing Visualizations

Why Customization is important:

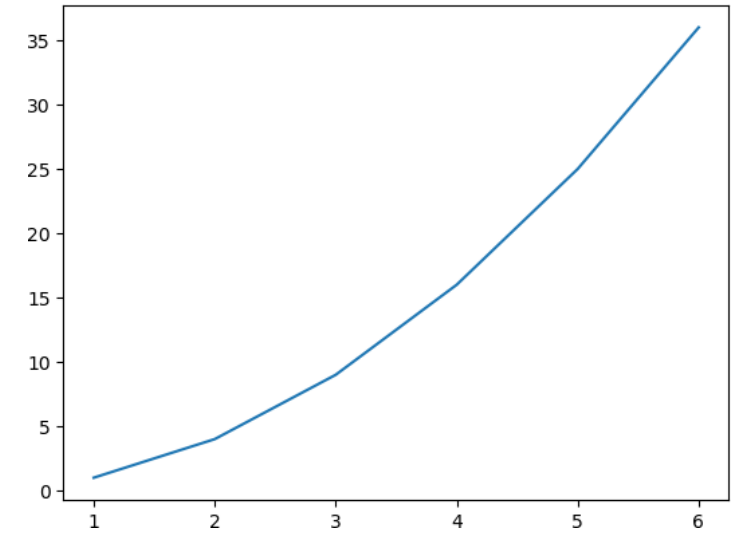
- **Enhances clarity**
  - Helps highlight key data points or trends
- **Improves readability**
  - Makes data easier to interpret at a glance
- **Tailors plots to your audience**
  - Adapts visualization style to viewer preferences and expertise

Adding Labels, Titles, and Legends

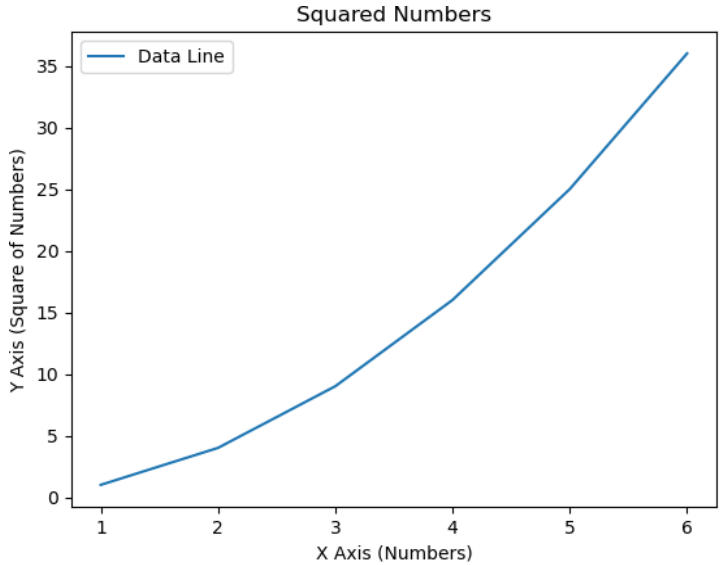
A graphs can be self-explanatory, if it have a title to the graph, labels on the axis, and a legend that explains what each line is can be necessary.

- **Best Practices:**
  - Provides clear, concise, and informative labels
  - Provide meaningful titles
- **Functions:**
  - `plt.xlabel()`, `plt.ylabel()`, and `plt.title()`
  - Use `fontdict` to adjust fonts (size, style, weight)
- **Legend Placement**:
  - Add a legend using `plt.legend()`
  - Position effectively (`loc` parameter)

Simple Line Plot without any customization



Simple Line Plot with labels, title and legend



Customizing and Enhancing Visualizations

Customizing Font Styles and Sizes

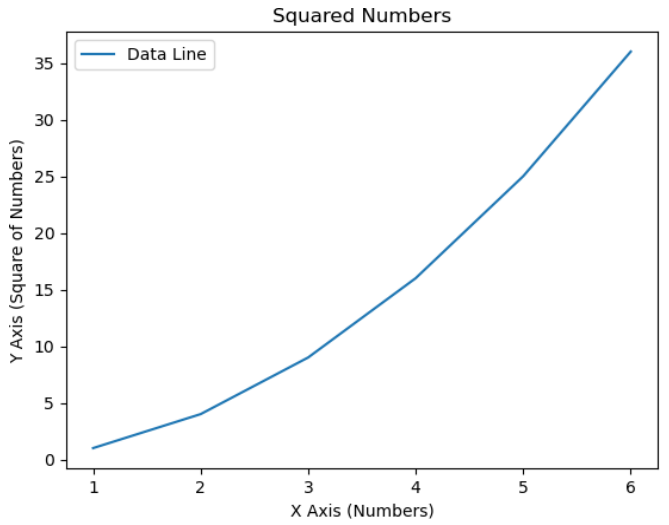
- Customize fonts to improve readability
  - Use `fontdict` to adjust fonts (size, style, weight)
- **Example:**
  - Use `fontstyle='italic'` for emphasis
  - Use `fontsize=14` for larger titles

Customizing Legends:

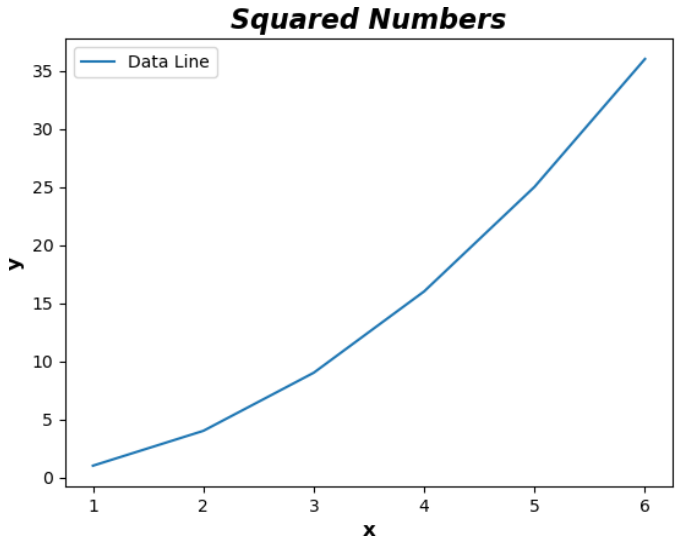
A legend is an area describing the elements of the graph.

- **Effective Legend Tips:**
  - Avoid blocking important data
  - Make sure the legend is clear and easy to read
- **Customizing Legends with `plt.legend()`:**
  - **Parameters:**
    - `loc`: Position the legend (e.g., 'upper right', 'lower left').
    - `fontsize`: Control font size of the legend text.
    - `frameon=False`: Remove the box frame around the legend.

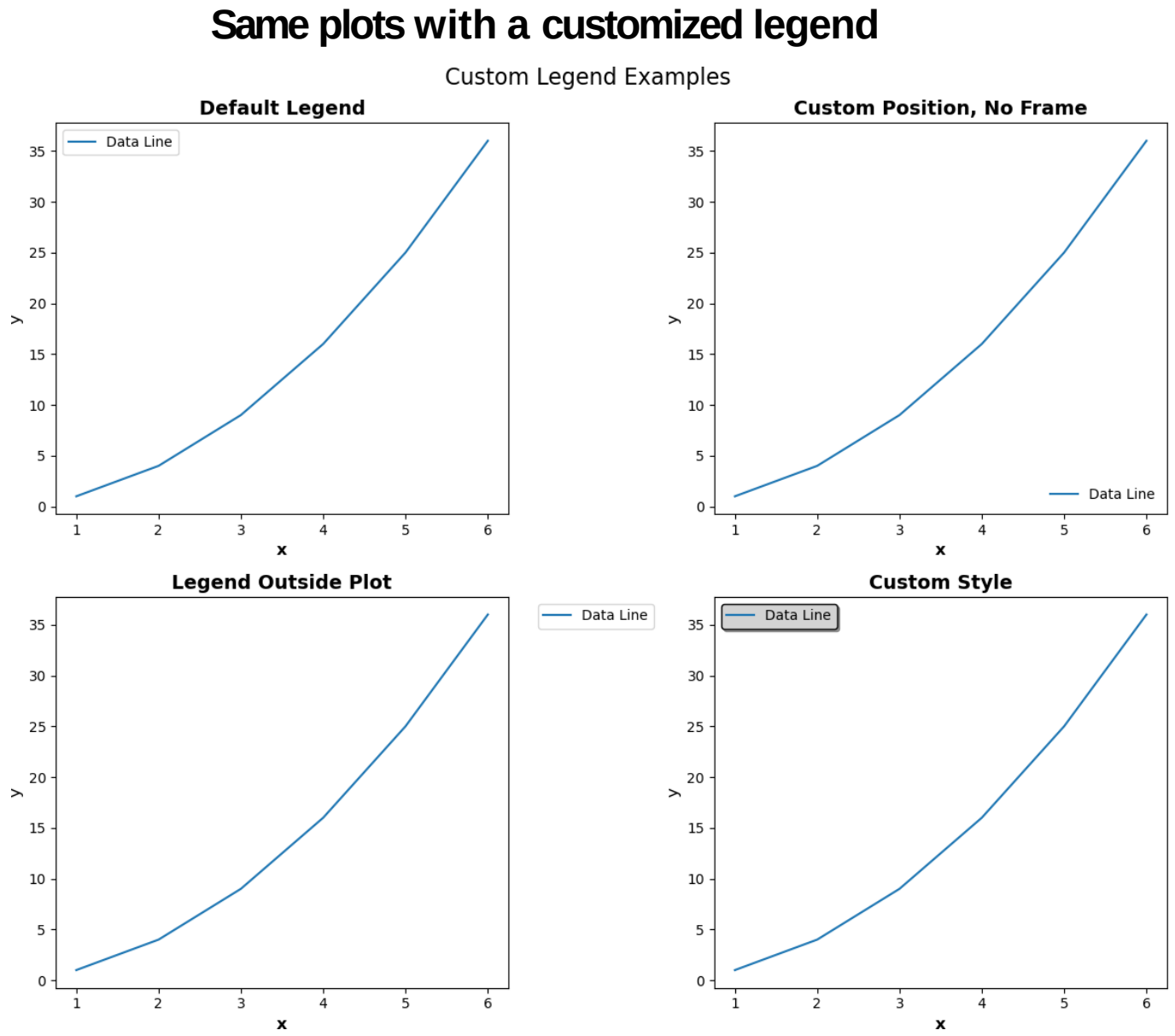
Plot with labels and title with default font style



Plot with labels and title in custom font style



Customizing Legends



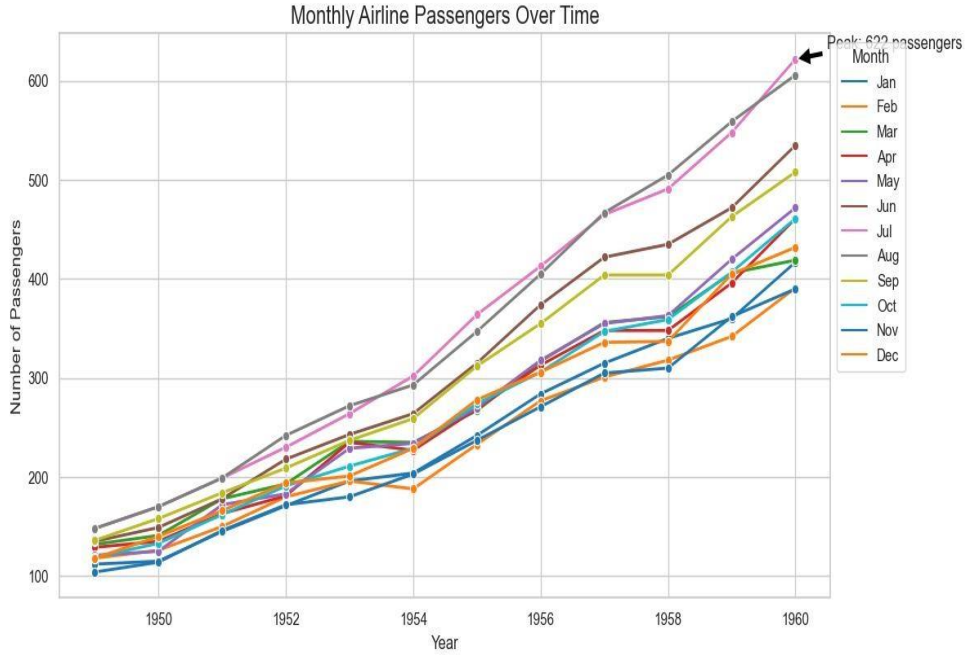
Combining Matplotlib and Seaborn for Complex Visualizations

Why Combine?

- **Matplotlib**:
  - Low-level control and customization
  - Fine-grained control over every aspect of the plot
- **Seaborn**:
  - High-level interface and statistical visualizations
  - Built-in themes for attractive plots

Key Benefits:

1. Leveraging the strengths of both libraries.
2. Enhanced aesthetics with Seaborn's styles
3. Statistical plots from Seaborn with Matplotlib's flexibility
4. Create complex, multi-layered visualizations



Case study

Analyzing Flight Data

- In this case study, we analyze the **'flights'** dataset, which records monthly airline number of passenger from 1949 to 1960.
- This dataset has three variables (year, month, and number of passengers)
- We visualize the monthly airline passenger trends over time, combining Seaborn's statistical capabilities with Matplotlib's fine- tuned control to create a multi-layered visualization.

Visualizing and Analyzing Time Series Data

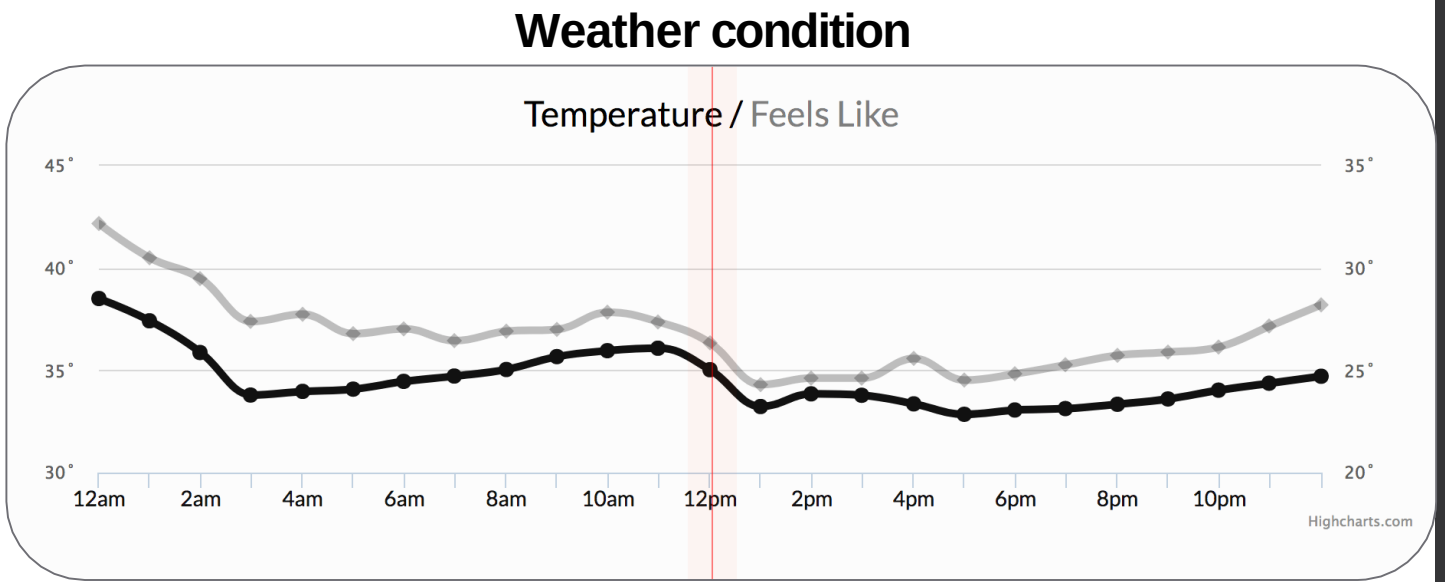
Introduction to Time Series Data

Time Series Data:

- Time series data is information that is collected at regular, consistent intervals over time.
- **Examples:**
  - Temperature measured every two hours
  - Daily stock prices
  - Weekly sales data
- This data type helps us observe trends, patterns, and correlations across time, providing deeper insights into various processes.

Why Time Series Data Matters:

- By analyzing and visualizing time series data, we can identify trends, spot anomalies, and uncover patterns that guide better decision-making and predictions.
- Time series data is widely used in fields like economics, weather forecasting, finance, and more.



Basic Time Series Data Visualization Techniques

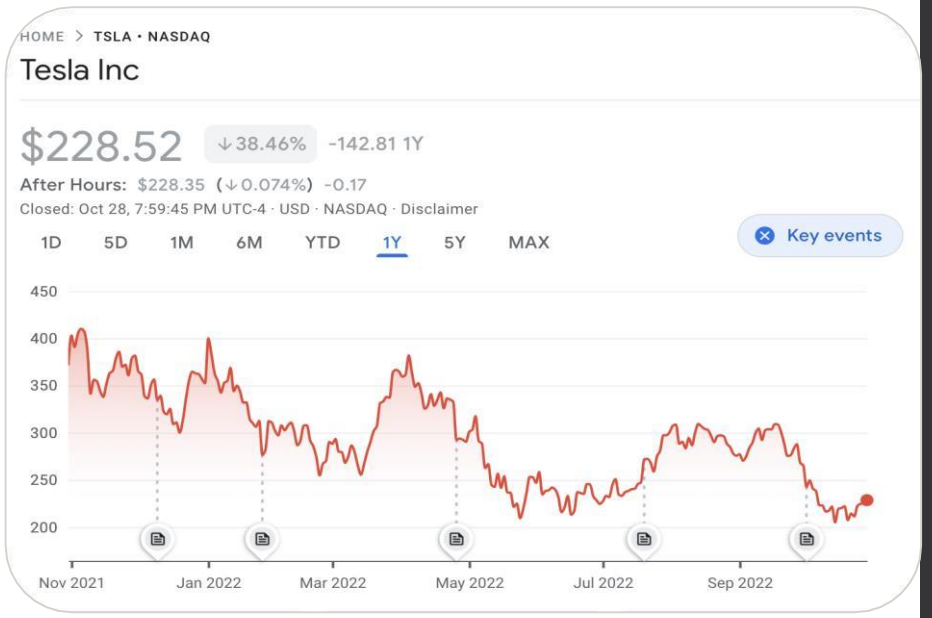
- Time series data can be effectively visualized using various techniques to reveal trends, patterns, and changes over time.
- Temporal visualizations help in analyzing data across a time axis, uncovering key insights such as seasonality or trends.

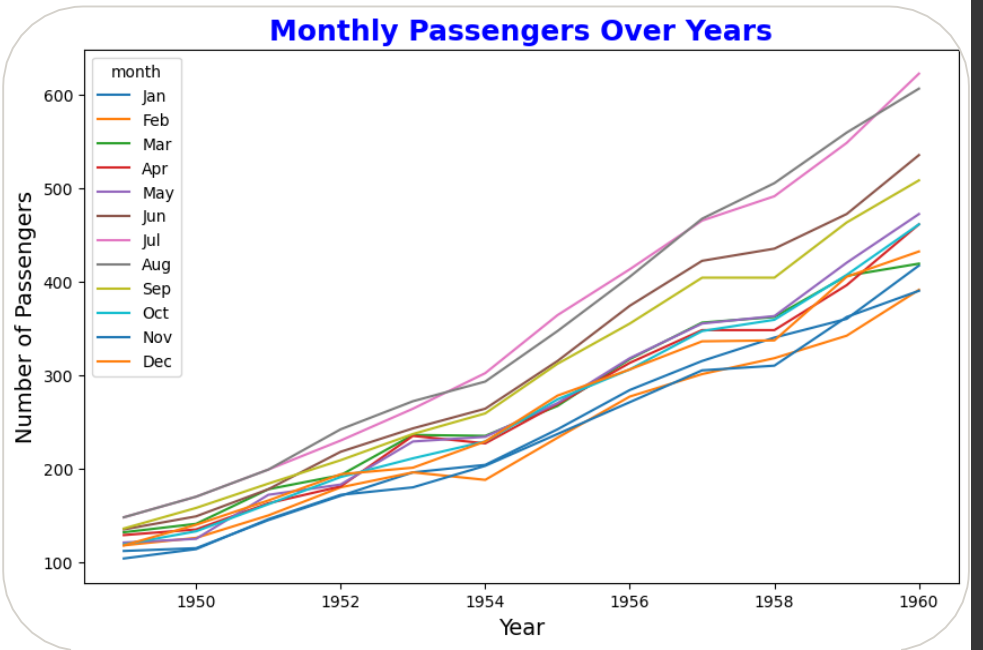
Key Visualization Techniques:

1. Line Plot:

- A fundamental and most common visualization for time series data.
- Show trends over time.
- **Axes:**
  - **X-axis**: Represents time.
  - **Y-axis:** Represents the variable of interest.
- **Use Case**: Perfect for showing how a variable changes over time.
- **Key Features:**
  - Easy to understand
  - Useful for identifying **trends** and **patterns**.
  - Can display multiple lines for comparing variables.
- **Example**: Monthly number of passengers over the years.

Graph of stock prices of Tesla

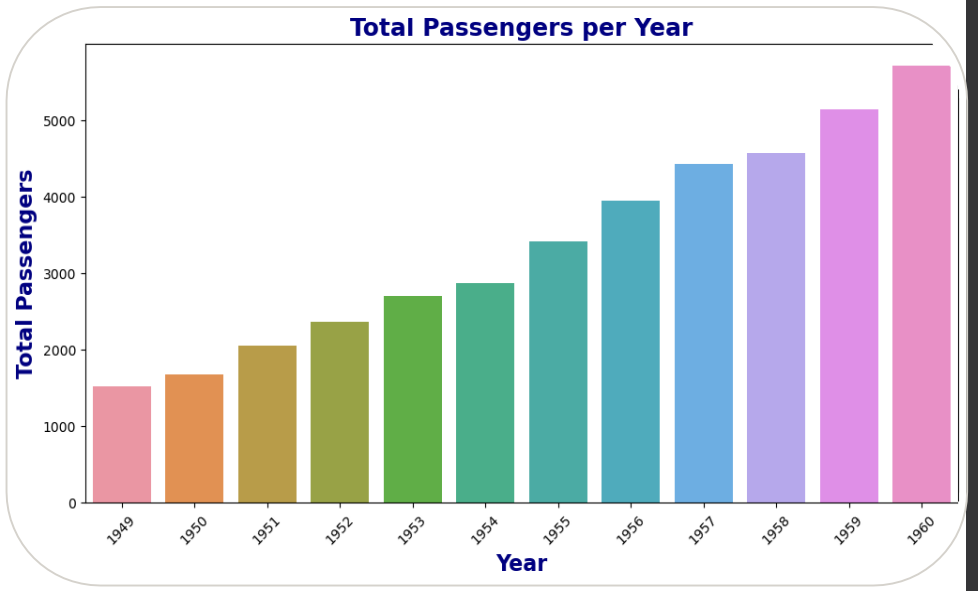


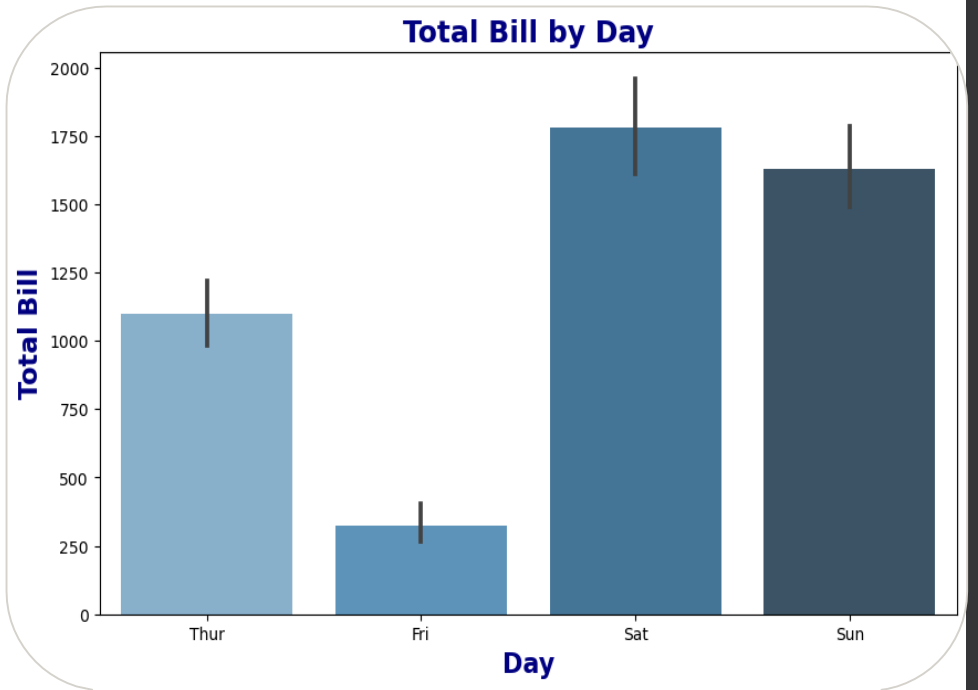


Basic Time Series Data Visualization Techniques

2. Bar Chart:

- Represents data as **horizontal or vertical bars.**
- **Axes**
  - **X-axis:** Time periods
  - **Y-axis:** Variable values
- **Key Features:**
  - Bar height/length proportional to variable value.
  - Supports grouped or stacked bars for multiple variables.
  - Best for comparing discrete time intervals.
- **Advantages:**
  - Easy to compare values across time periods
  - Effective for displaying multiple variables
  - Clear representation of discrete data.
- **Use Case:**
  - Ideal for comparing **discrete time intervals** (e.g., yearly revenue comparisons).
- **Example**:
  - Showing total bill amounts across different days of

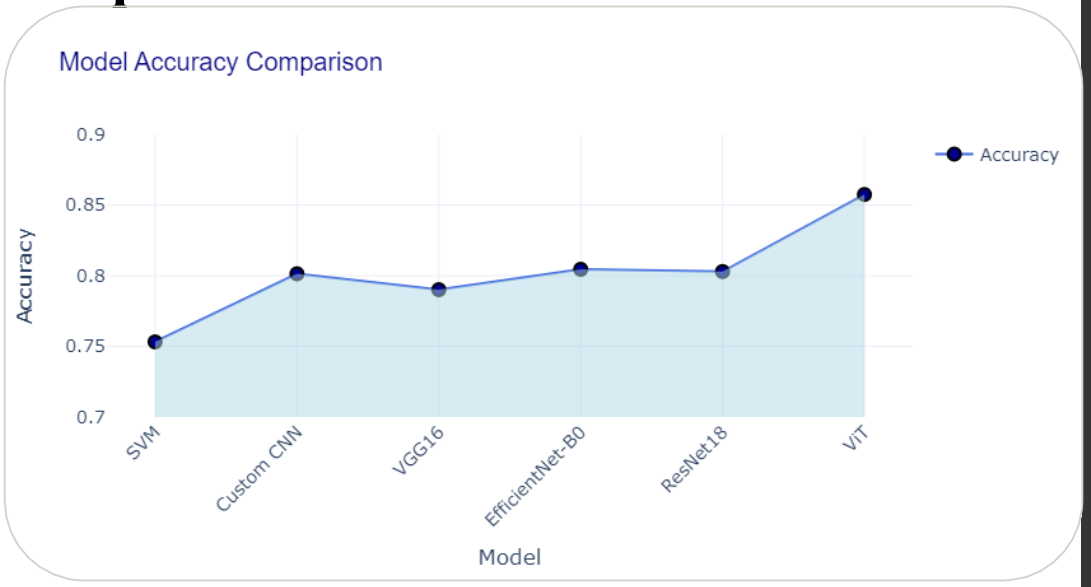


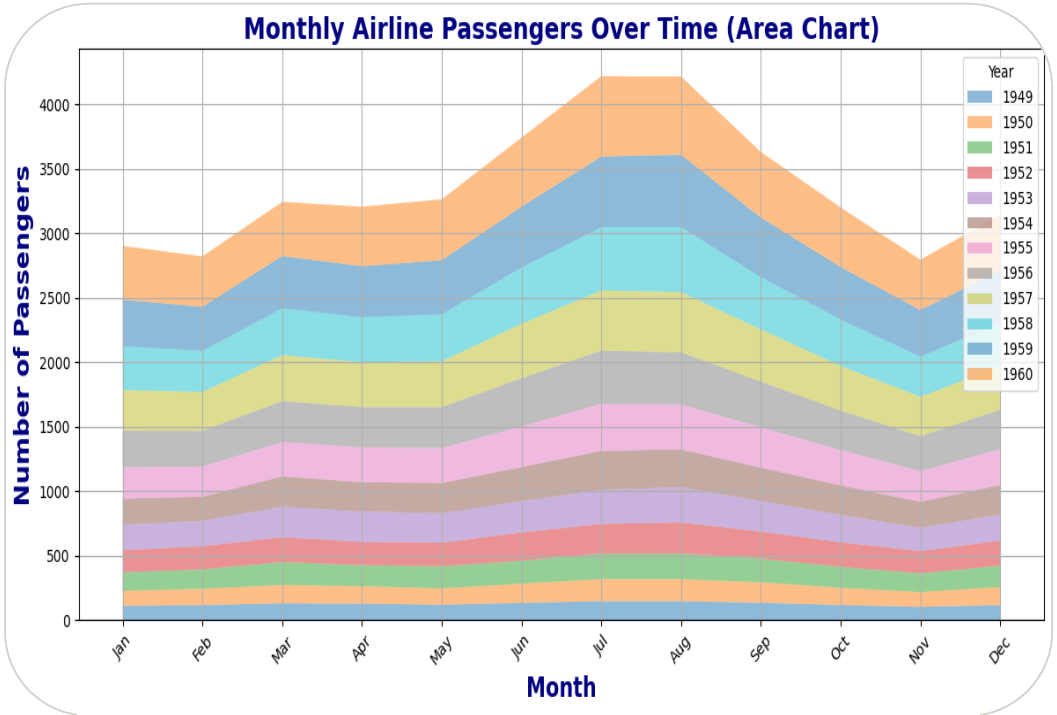


Basic Time Series Data Visualization Techniques

3. Area Chart

- Area chart extend line plots by filling the area beneath the line to show trends over time.
- **Axes:**
  - **X-axis:** Time
  - **Y-axis:** Variable values
- **Key Features:**
  - Can represent one or more variables.
  - The area under the line(s) is filled, making it easier to see the magnitude of values over time.
  - Best for visualizing distributions or magnitude changes over time.
- **Use Case:**
  - Shows cumulative totals or proportions over time.
- **Example:** Yearly sales growth of different products.





Case Study: Forecasting Stock Prices with LSTM

Objective

- To visualize, analyze, and forecast **Apple Inc. (AAPL)** stock prices using Long Short-Term Memory (LSTM) models.

Dataset

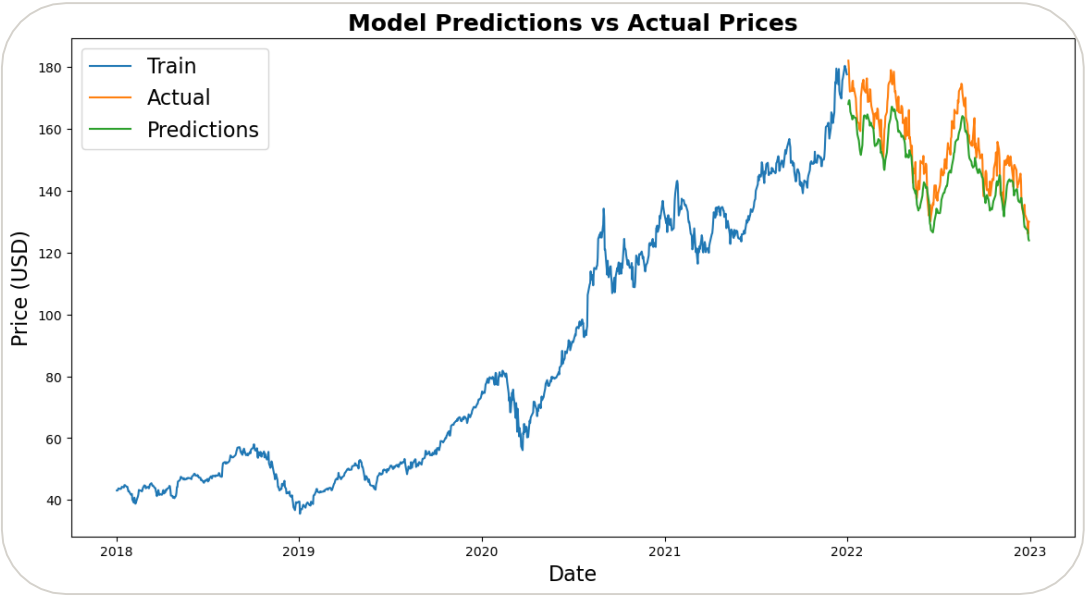
- **Dataset Source:** Yahoo Finance
- **Dataset:** Daily stock prices for Apple Inc. (AAPL) over the last 5 years.

Steps:

1. **Data Preprocessing:** Convert dates, handle missing values, and visualize stock prices with moving averages.
2. **Feature Engineering:** Prepare the data by scaling and creating sequences for LSTM input.
3. **Model Construction:** Build an LSTM model with 2 layers and dropout for regularization.
4. **Training the Model**: Train the model on 80% of the data over 10 epochs.
5. **Testing and Prediction:** Predict future stock prices using the trained LSTM model and compare them to actual prices.

Conclusion:

- This case study showcases how LSTM models can effectively forecast stock prices, offering practical experience in time series forecasting, feature engineering, and model evaluation.



Interactive Data Visualization with Plotly

Introduction to Interactive Data Visualization

What is Interactive Data Visualization?

- A type of data visualization that allows users to interact with data in order to explore and understand it.
- It uses tools to create visual data representations that users can explore and analyze interactively.
- Helps users gain deeper insights by interacting with data directly.

- Interactive visualizations enhance data exploration by enabling users to dynamically engage with and manipulate data through actions like zooming, filtering, and hovering.

Why Use Interactive Data Visualization?

- Allows dynamic exploration and manipulation of data, rather than static views.
- Supports actions like zooming, filtering, and comparing to discover patterns, trends, and relationships.
- Enhances the ability to uncover insights and outliers.

HEAVY.AI's Interactive Data Visualization of Global Confirmed COVID-19 Cases and Spread.

Introduction to Interactive Data Visualization

Benefits:

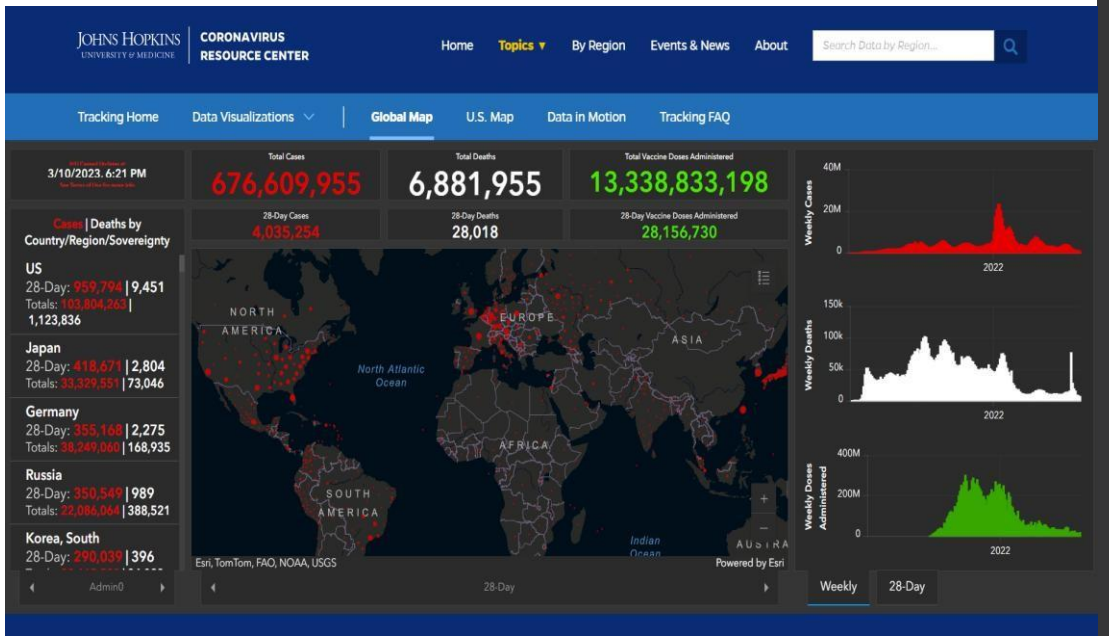
- **Enhanced Understanding:** Allows users to explore data patterns and trends interactively.
- **Improved Decision Making:** Provide quick insights through real-time interaction.
- **Engagement:** Keeps the audience actively involved and focused.
- **Customization:** Users can filter, sort, and adjust views to highlight specific insights.

Use Cases (Real-World Applications):

- **Business Analytics**: Sales dashboards, performance tracking.
- **Finance**: Stock market trends, risk assessment.
- **Education**: Interactive learning tools, research data presentations.
- **Healthcare**: Patient data visualization, epidemiological tracking.

Real-World Example:

- **COVID-19 Dashboard by Johns Hopkins University:** Interactive maps and charts tracking the pandemic's progression.



Getting Started with Plotly

What is Plotly?

- A powerful open-source graphing library for Python, R, and JavaScript.
- Creates interactive, publication-quality visualizations.
- Supports a wide range of chart types and interactive features.
- Enables Python users to create stunning web-based visualizations.
- Integrates seamlessly with web applications and Jupyter notebooks.
- Visualizations can be saved as standalone HTML files or displayed directly in Jupyter notebooks.

Installation and Setup

- Install Plotly using pip.

```
pip install plotly
```

- For notebook environments, install additional package *notebook* if required:

```
pip install “notebook>=5.3” “ipwidgets>=7.5”
```

- Basic usage in Python:

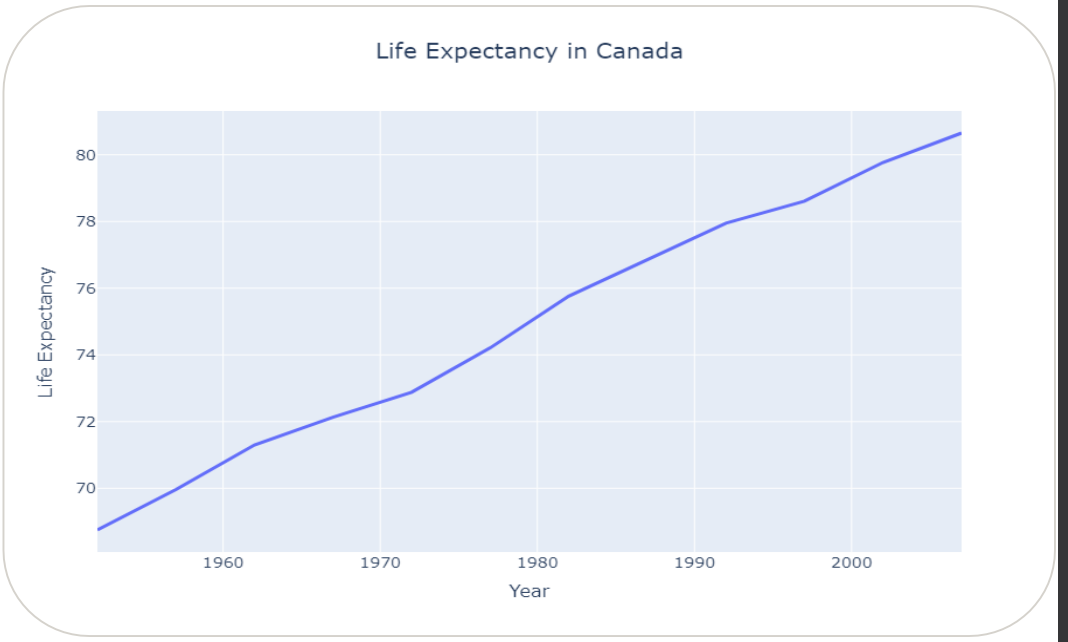
```
import plotly.graph_objects as go
import plotly.express as px
```

```
(DIP_7th) C:\Users\PC>pip install plotly
Collecting plotly
  Downloading plotly-5.24.1-py3-none-any.whl.metadata (7.
3 kB)
Collecting tenacity>=6.2.0 (from plotly)
  Downloading tenacity-9.0.0-py3-none-any.whl.metadata (1
.2 kB)
Requirement already satisfied: packaging in c:\users\pc\a
naconda3\envs\dip_7th\lib\site-packages (from plotly) (24
.1)
Downloading plotly-5.24.1-py3-none-any.whl (19.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 19.1/19.1 MB
241.9 kB/s eta 0:00:00
Downloading tenacity-9.0.0-py3-none-any.whl (28 kB)
Installing collected packages: tenacity, plotly
Successfully installed plotly-5.24.1 tenacity-9.0.0
```

Basic Plot Types

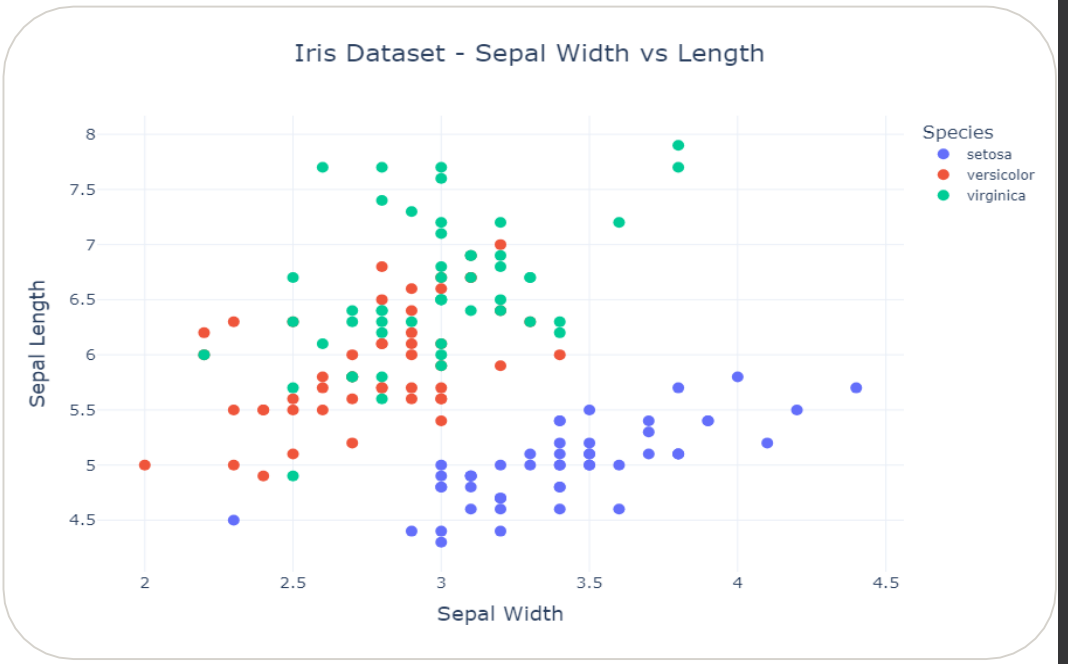
1. **Line Plots:**
   - Show trends over time
   - **Example**:
     - Showing life expectancy in a country
     - Stock prices over time.

```
import plotly.express as px

df = px.data.gapminder().query("country=='Canada'")
fig = px.line(df, x="year", y="lifeExp", title='Life
expectancy in Canada')
fig.show()
```



2. **Scatter Plots:**
   - Show the relationship between two variables.
   - **Example:**
     - Height vs Weight of individual

```
import plotly.express as px

df = px.data.iris()
fig = px.scatter(df, x="sepal_width", y="sepal_length",
color="species",
                 title="Iris Dataset - Sepal Width vs
Length")
fig.show()
```

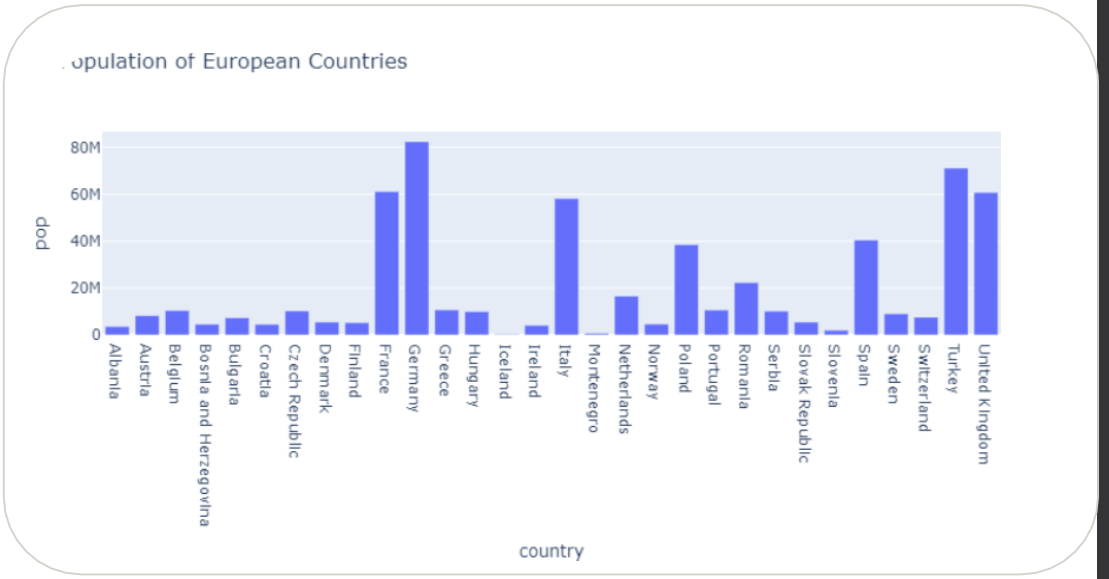


Basic Plot Types

3. Bar Charts:

- Comparing quantities across different categories.
- **Example**:
  - Sales per region.

```
import plotly.express as px

df = px.data.gapminder().query("year == 2007").query("continent
== 'Europe'")
fig = px.bar(df, x="country", y="pop", title="Population of
European Countries")
fig.show()
```



Real-World Examples

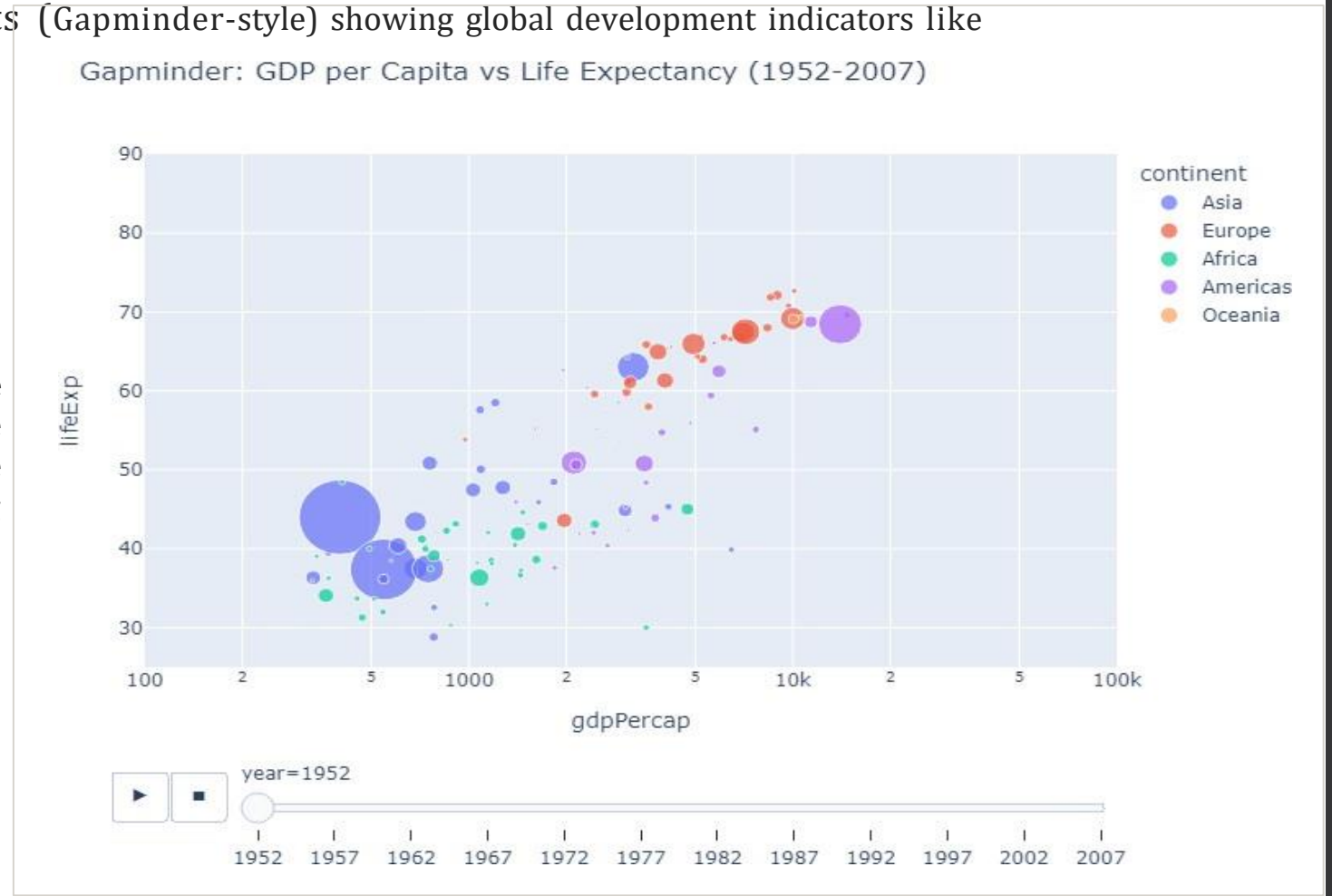
- **Line Chart:** Tracking website traffic over a month.
- **Scatter Plot:** Analyzing the correlation between advertising spend and sales.
- **Bar Chart:** Comparing product performance across different markets.

Advanced Features of Plotly

➢ Animations and Transitions

- **Dynamic Visualizations**: Bring data to life by showcasing changes over time or across different states.
- **Applications**: Highlight trends, growth patterns, or evolving relationships (e.g., population growth, financial performance, or climate change).
- **Example**: Animated bubble charts (Gapminder-style) showing global development indicators like GDP and life expectancy.

This animated scatter plot shows the relationship between GDP per capita and life expectancy (1952-2007). Bubble size represents population, with colors indicating continents for global comparisons.



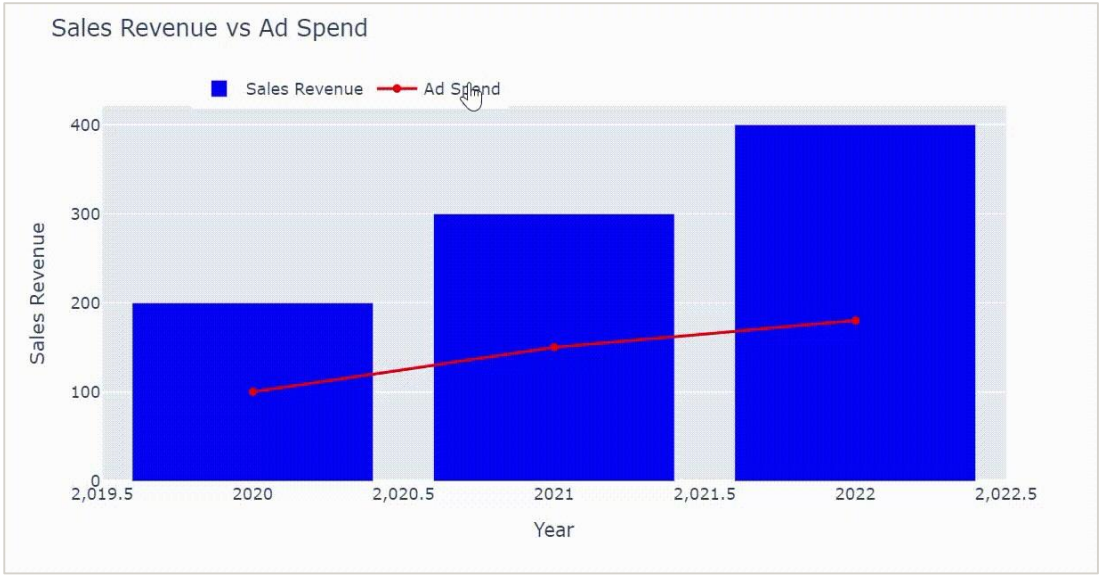
Advanced Features of Plotly

➢ Interactive Widgets

- **Custom Interactivity**: Enhance user experience with sliders, dropdowns, and buttons for tailored data exploration.
- **Applications**: Enable filtering, zooming, and dynamic customization of plots without page reloads.
- **Example**: A dashboard where users can explore sales performance by region, time, or product category.

➢ Multi-Axis and Subplots

- **Complex Layouts**: Create advanced visualizations by combining multiple plots or adding secondary axes for deeper analysis.
- **Applications**: Compare diverse metrics or overlay datasets with different scales in a single view.
- **Example:** A dual-axis chart showing sales revenue (bar plot) and ad spend (line plot) to analyze their relationship over time.



Customizing Plotly Visualizations

1. Layout and Styling

- Plotly offers extensive customization options for layouts and styles.
- Add margins for spacing, include descriptive titles,
  and customize axis properties.
- **Example**: Customize a scatter plot by adding margins, titles, and axis labels to improve readability.

2. Titles, Labels, and Legends

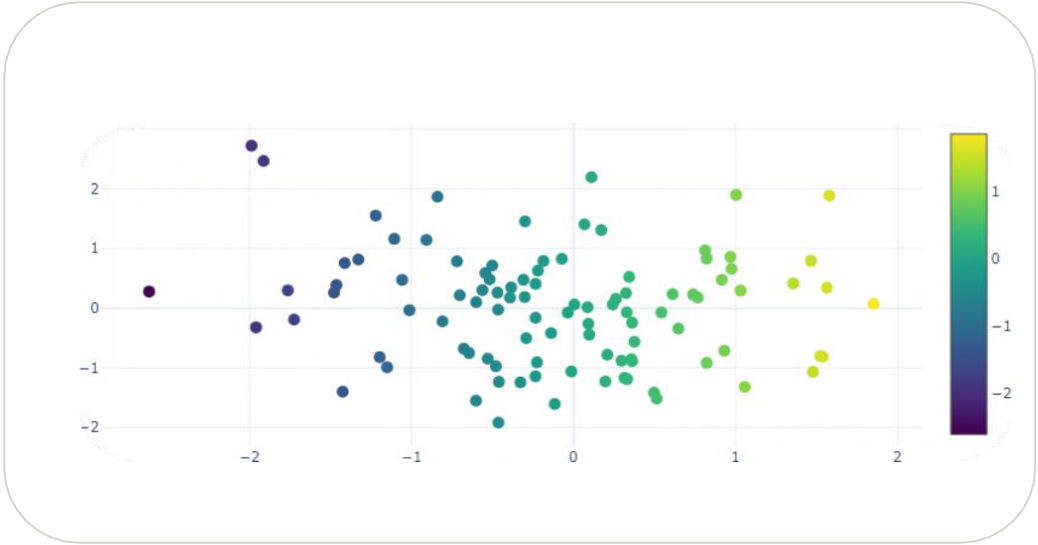
➢ **Improve Readability**:

- Use clear, descriptive titles and axis labels.
- Position legends strategically to avoid visual clutter.

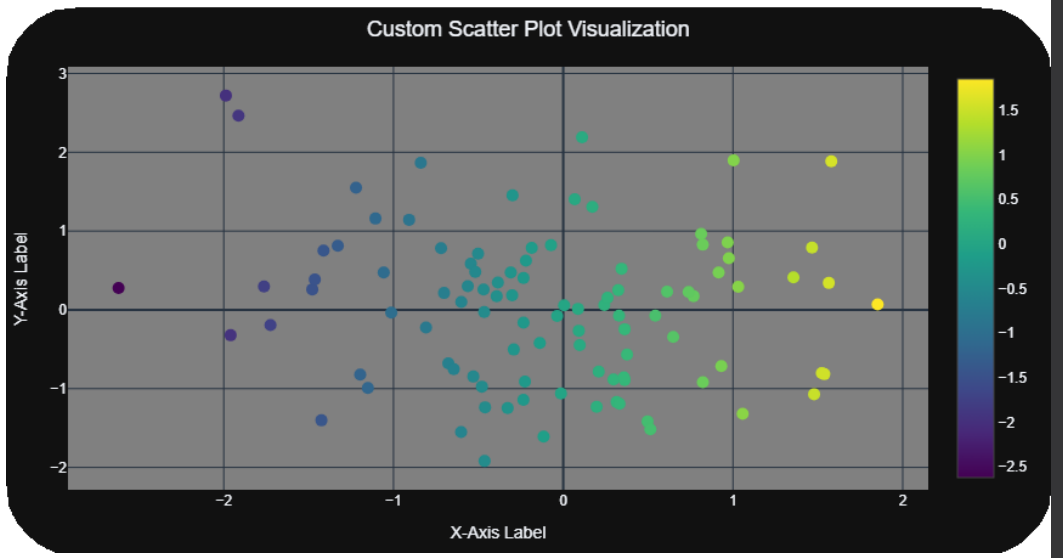
3. Themes and Templates

- Utilize Plotly's pre-defined themes for cohesive visuals.
- **Example:**
  - *plotly_dark* is ideal for dark-mode visualizations.

Without Customization



With Customization



Comparing Plotly with Other Visualization Libraries

Library	Strengths	Limitations
Matplotlib	• Highly customizable, ideal for static plots.	• Lacks interactivity, steep learning curve.
Seaborn	• Simplifies statistical visualizations.	• Limited interactivity, fewer chart types.
Plotly	• Best for interactive, web-ready visualizations.	• Slower for large datasets, reliant on JavaScript
Bokeh	• Good for interactive web-based plots.	• Smaller community, less documentation than Plotly

Saving and Exporting Visualizations

Why Export?

- Preserve work for later use
- Share with others (reports, presentations, publications)
- Use in different applications or platforms

Common File Formats:

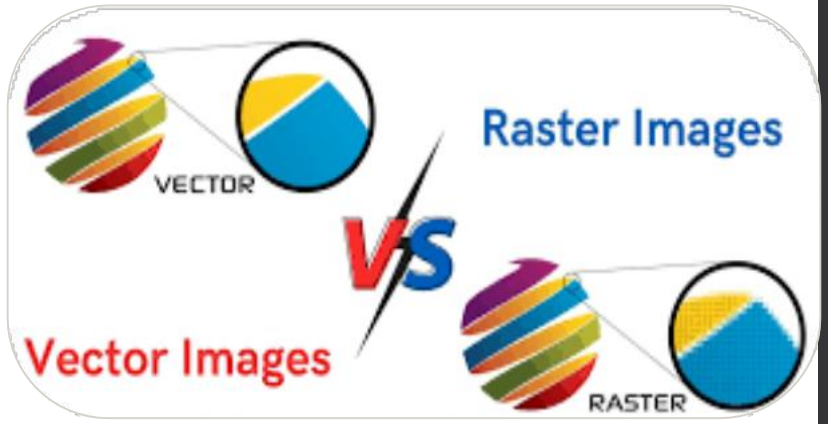
1. **Raster Graphics**: PNG, JPEG, TIFF
   - It made up of tiny pixels, are ideal for and colorful images like detailed photographs.
   - Fixed resolution (quality may decrease when enlarged)
   - **PNG**: Lossless compression (no loss of image quality when saved), best for web graphics and images with text or sharp edges.
   - **JPEG**: Lossy compression (slight reduction in image quality to achieve smaller file sizes), best for photographs, complex images with many colors.
2. **Vector Graphics**: SVG, PDF, EPS
   - Use mathematical equations to define shapes, lines, and curves
   - Scalable without quality loss (can be resized infinitely)
   - Ideal for: Logos, illustrations, charts, and graphs
   - Best for print and high-resolution displays

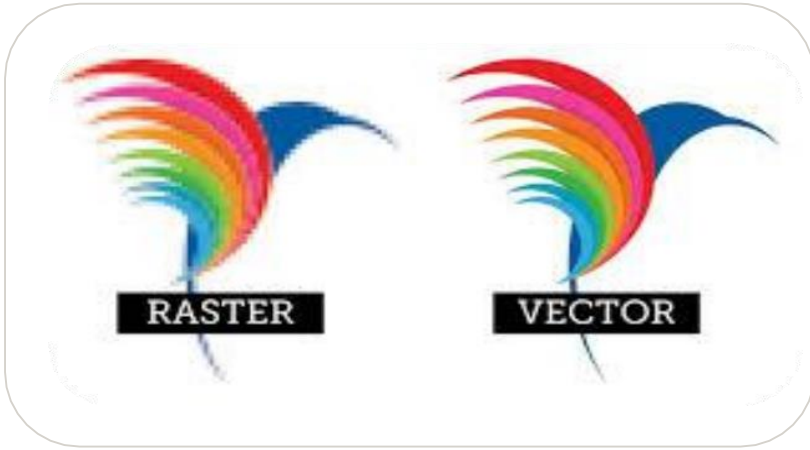
Best Practices:

- Choose appropriate format based on intended use
- Consider resolution requirements (e.g., print vs. web)
- Maintain aspect ratio when resizing
- Test exported files in target environment

Matplotlib: savefig() function

Seaborn: plt.savefig() (uses Matplotlib backend).





Example Code:

```
plt.savefig('my_plot.png', dpi=300)
```

```
plt.savefig('plot.svg', format='svg')
```

Thank You