21CSS303T – DATA SCIENCE

UNIT - III

- Introduction to Matplotlib library
- Subplots
- Controlling axes, ticks, labels, legends
- Annotations, saving plots
- Seaborn library
- Multiple plots
- Scatter plot, line plot, histogram
- Boxplot
- Pair plot
- Playing with text
- 3D plot

Introduction to Matplotlib Library

Introduction to Matplotlib Library

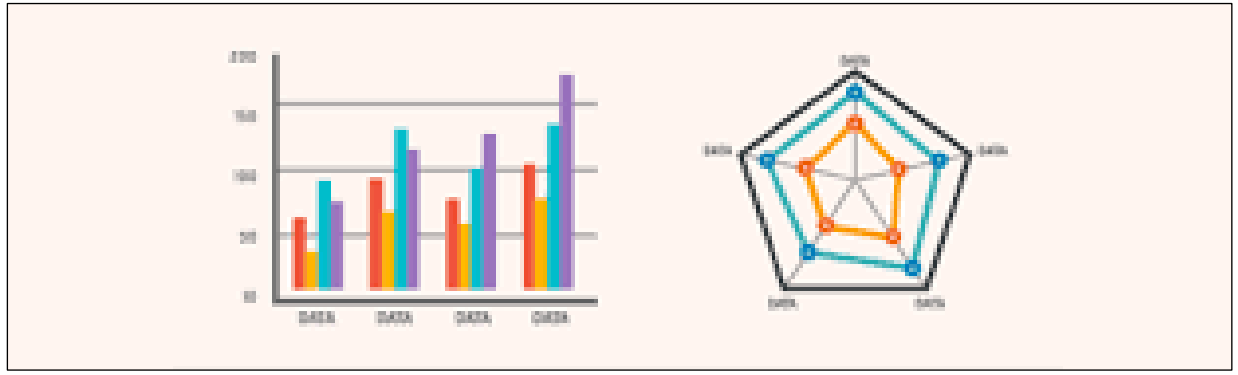
- **Matplotlib** is a powerful plotting library in Python used for creating static, animated, and interactive visualizations.
- Matplotlib's primary purpose is to provide users with the tools and functionality to represent data graphically, making it easier to analyze and understand.
- It was originally developed by John D. Hunter in 2003 and is now maintained by a large community of developers.

Features of Matplotlib

- **Versatility**: Matplotlib can generate a wide range of plots, including line plots, scatter plots, bar plots, histograms, pie charts, and more.
- **Customization**: It offers extensive customization options to control every aspect of the plot, such as line styles, colors, markers, labels, and annotations.
- **Integration with NumPy**: Matplotlib integrates with NumPy, making it easy to plot data arrays directly.
- **Publication Quality**: Matplotlib produces high-quality plots suitable for publication with fine-grained control over the plot aesthetics.

Features of Matplotlib

- **Extensible**: Matplotlib is highly extensible, with a large ecosystem of add-on toolkits and extensions like Seaborn, Pandas plotting functions, and Basemap for geographical plotting.
- **Cross-Platform**: It is platform-independent and can run on various operating systems, including Windows, macOS, and Linux.
- **Interactive Plots**: Matplotlib supports interactive plotting through the use of widgets and event handling, enabling users to explore data dynamically.



Installation of Matplotlib

Install Matplotlib with pip

- The python package manager pip is also used to install matplotlib.
- Open the command prompt window, and type the following command:

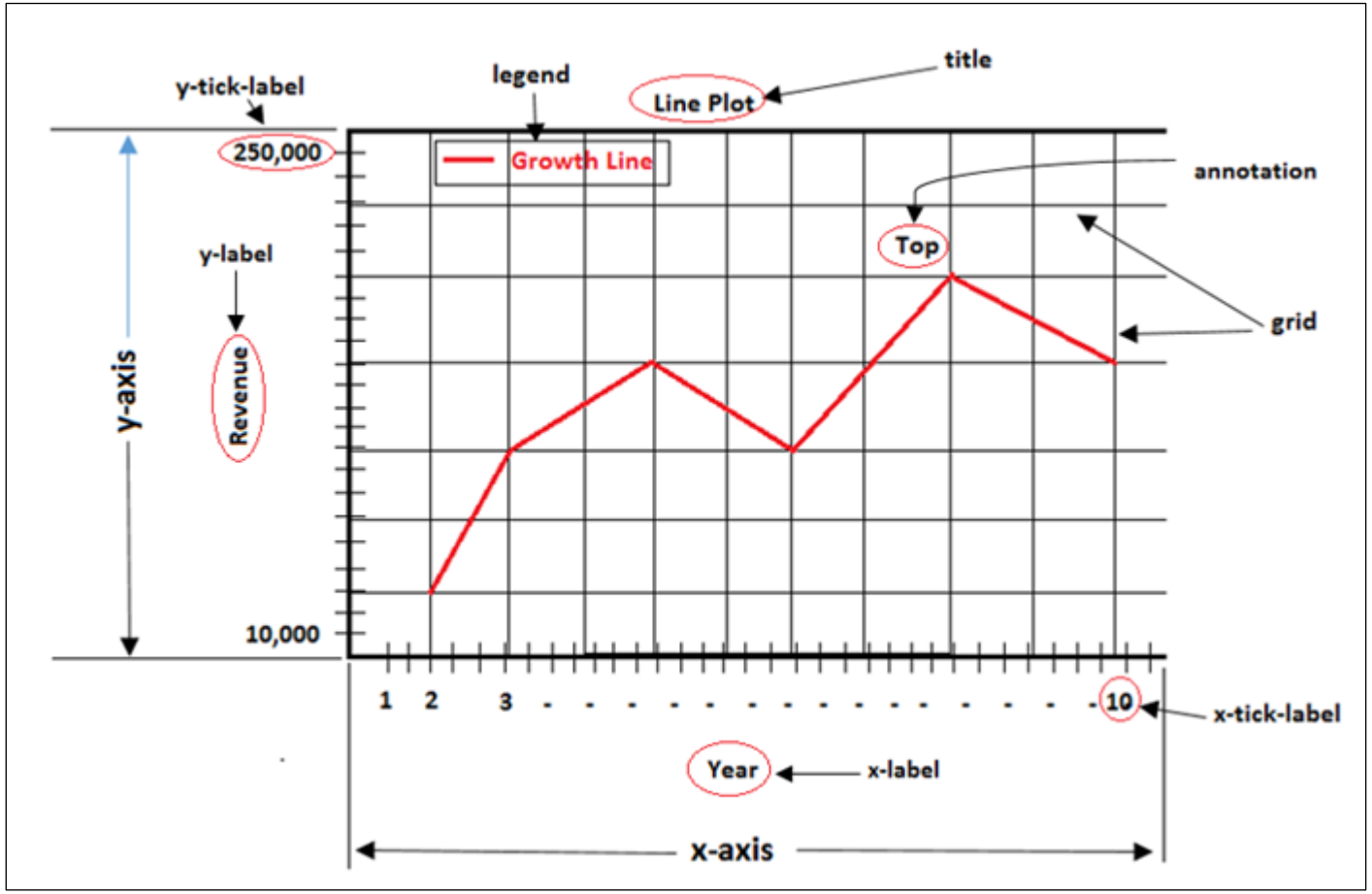
```
pip install matplotlib
```

Verify the Installation

- To verify that matplotlib is installed properly or not, type the following command includes calling .__version __ in the terminal.

```
import matplotlib
matplotlib.__version__
'3.1.1'
```

Components of Matplotlib



Different types of plots in Matplotlib

- Line Graph
- Stem Plot
- Bar chart
- Histograms
- Scatter Plot
- Stack Plot
- Box Plot
- Pie Chart
- Error Plot
- Violin Plot
- 3D Plots

Different plot styles in Matplotlib

- Python Pyplot
- Matplotlib Figure Class
- Matplotlib Axes Class
- Set Colors in Matplotlib
- Add Text, Font and Grid lines in Matplotlib
- Custom Legends with Matplotlib
- Matplotlib Ticks and Tick Labels
- Style Plots using Matplotlib
- Create Multiple Subplots in Matplotlib
- Working With Images In Matplotlib

Subplots

Pyplot

- The **matplotlib.pyplot** is the collection command style functions that make matplotlib feel like working with MATLAB.
- The pyplot functions are used to make some changes to figure such as create a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot including labels, etc.
- The pyplot module provide the **plot()** function which is frequently use to plot a graph.

Import Pyplot

- Most of the Matplotlib utilities lies under the pyplot submodule, and are usually imported under the plt alias:

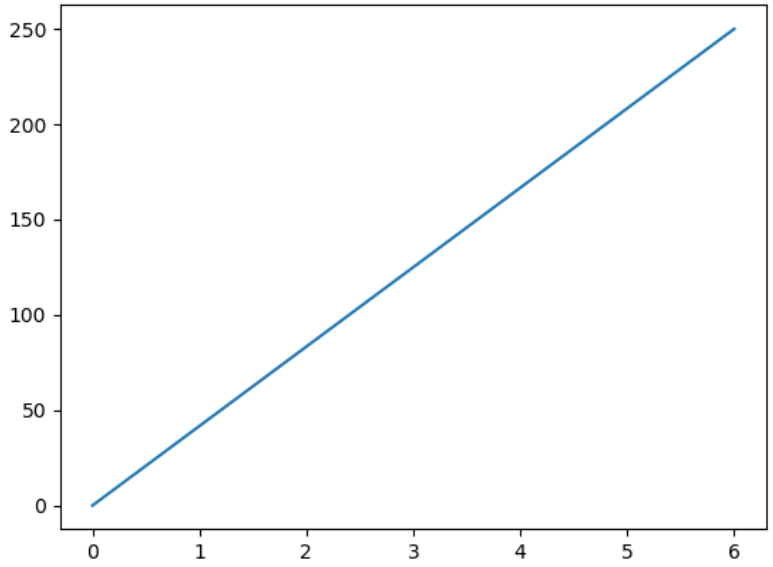
  import matplotlib.pyplot as plt

- Now the Pyplot package can be referred to as plt.

```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([0, 6])
ypoints = np.array([0, 250])

plt.plot(xpoints, ypoints)
plt.show()
```

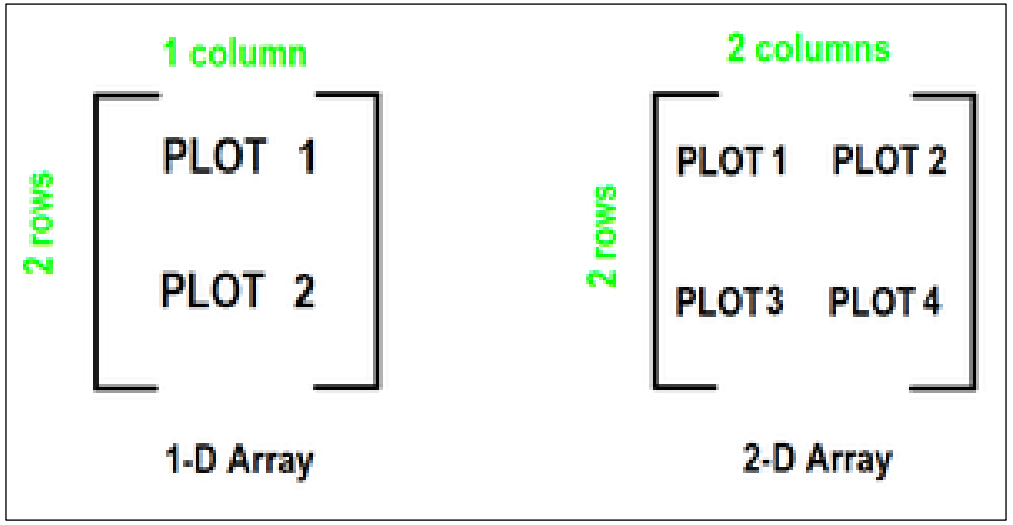


Subplots

- The Matplotlib **subplot()** function is defined as to plot two or more plots in one figure.
- We can use this method to separate two graphs which plotted in the same axis Matplotlib supports all kinds of subplots, including 2x1 vertical, 2x1 horizontal, or a 2x2 grid.
- It accepts the three arguments: they are **nrows, ncols, and index**.
- It denote the number of rows, number of columns and the index.
- Example:
- ```
  plt.subplot(1, 2, 1)
  #the figure has 1 row, 2 columns, and this plot is the first plot.
  ```
  ```

Subplots

- To create multiple plots use **matplotlib.pyplot.subplots** method which **returns the figure along with the** objects **Axes object or array of Axes object. nrows, ncols** attributes of subplots() method determine the number of rows and columns of the subplot grid.

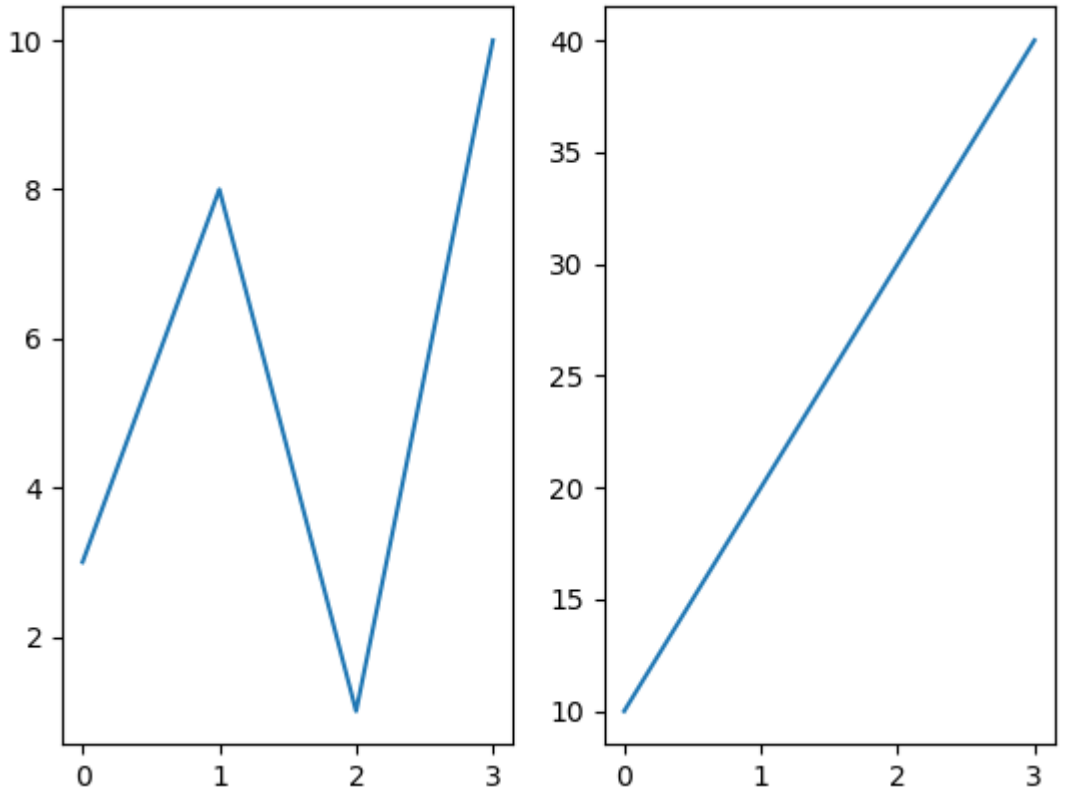


Multiple Subplot

- Create Multiple Subplots in Matplotlib in Python
- Here, we have various Ways to Matplotlib create subplots in Python.
  - 1-D Array of Subplots
  - Using plt.subplots with 2D Array of Subplots
  - Multiple Plots

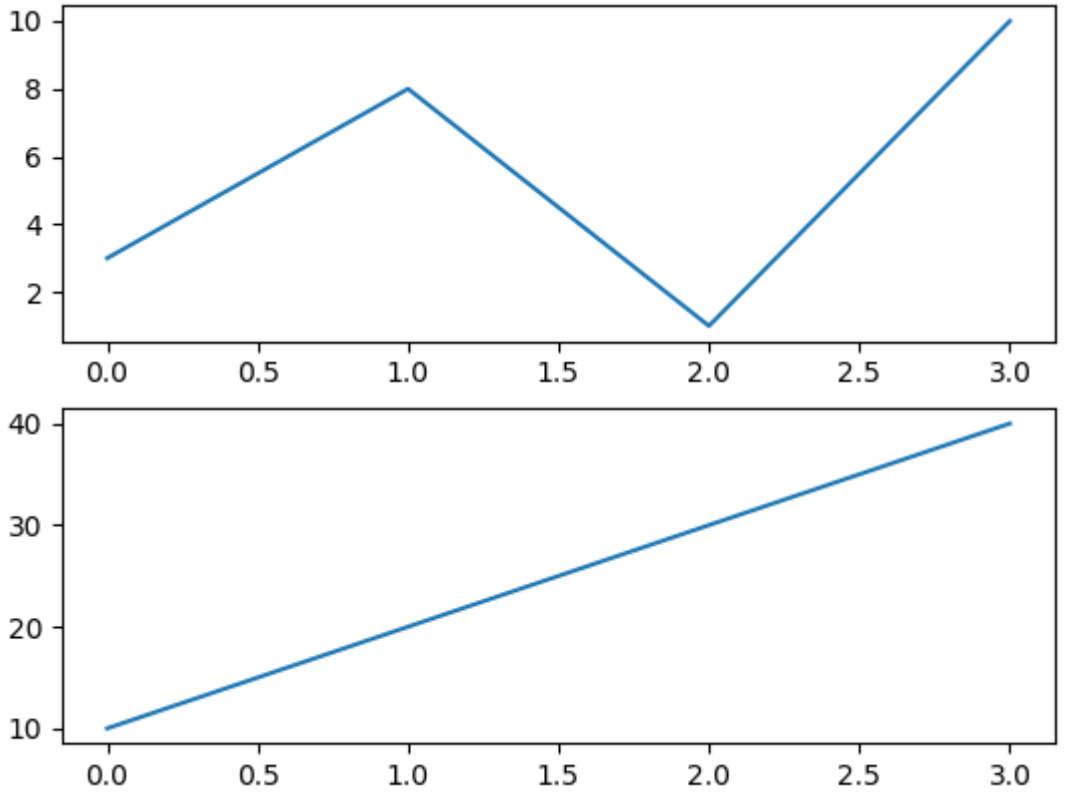
1D Array of Subplots

```
import matplotlib.pyplot as plt
import numpy as np
#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])
plt.subplot(1, 2, 1)
plt.plot(x,y)
#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])
plt.subplot(1, 2, 2)
plt.plot(x,y)
plt.show()
```



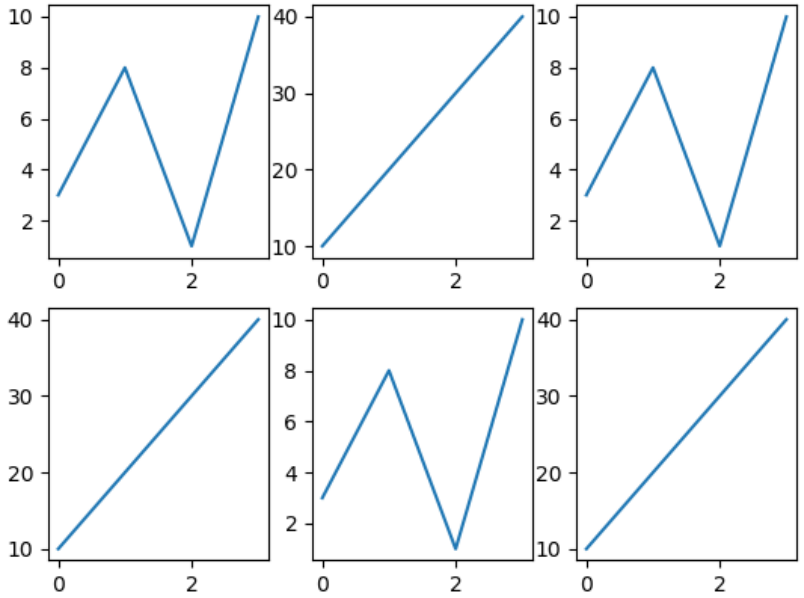
1D Array of Subplots

```
import matplotlib.pyplot as plt
import numpy as np
#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])
plt.subplot(2, 1, 1)
plt.plot(x,y)
#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])
plt.subplot(2, 1, 2)
plt.plot(x,y)
plt.show()
```



2D Array of Subplots

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])
plt.subplot(2, 3, 1)
plt.plot(x,y)
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])
plt.subplot(2, 3, 2)
plt.plot(x,y)
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])
plt.subplot(2, 3, 3)
plt.plot(x,y)
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])
plt.subplot(2, 3, 4)
plt.plot(x,y)
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])
plt.subplot(2, 3, 5)
plt.plot(x,y)
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])
plt.subplot(2, 3, 6)
plt.plot(x,y)
plt.show()
```



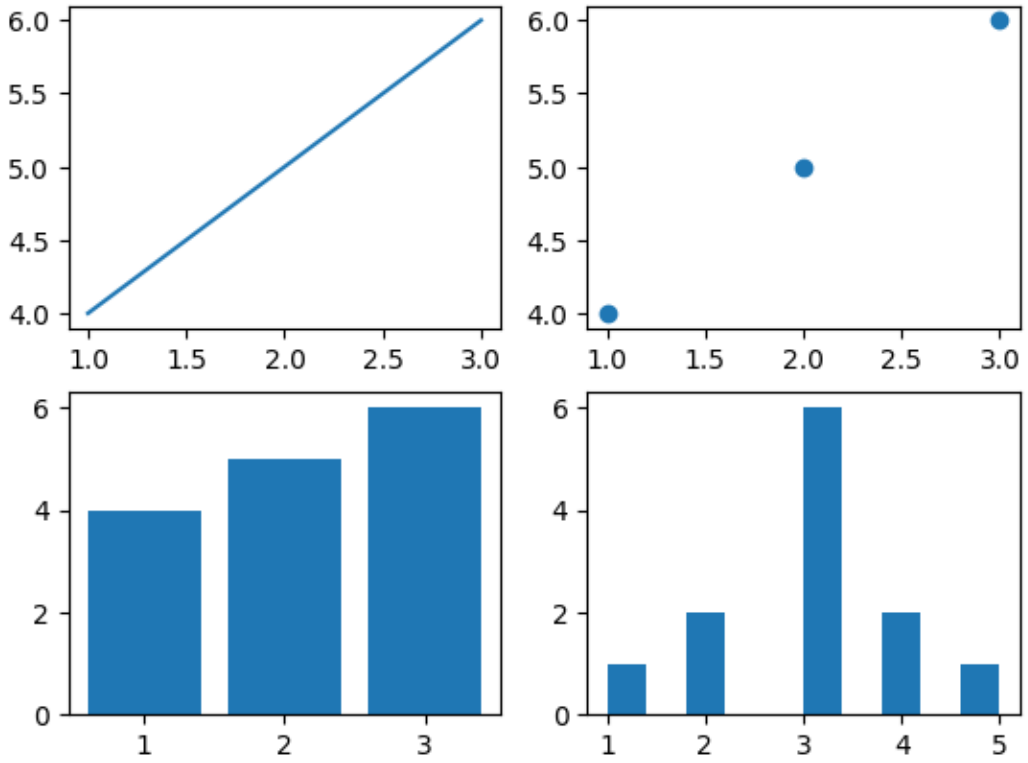
Multiple plots

```
import matplotlib.pyplot as plt

# Create a 2x2 grid of subplots
fig, axs = plt.subplots(2, 2)

axs[0, 0].plot([1, 2, 3], [4, 5, 6])
axs[0, 1].scatter([1, 2, 3], [4, 5, 6])
axs[1, 0].bar([1, 2, 3], [4, 5, 6])
axs[1, 1].hist([1, 2, 2, 3, 3, 3, 3, 3, 3, 4, 4, 5])

plt.show()
```



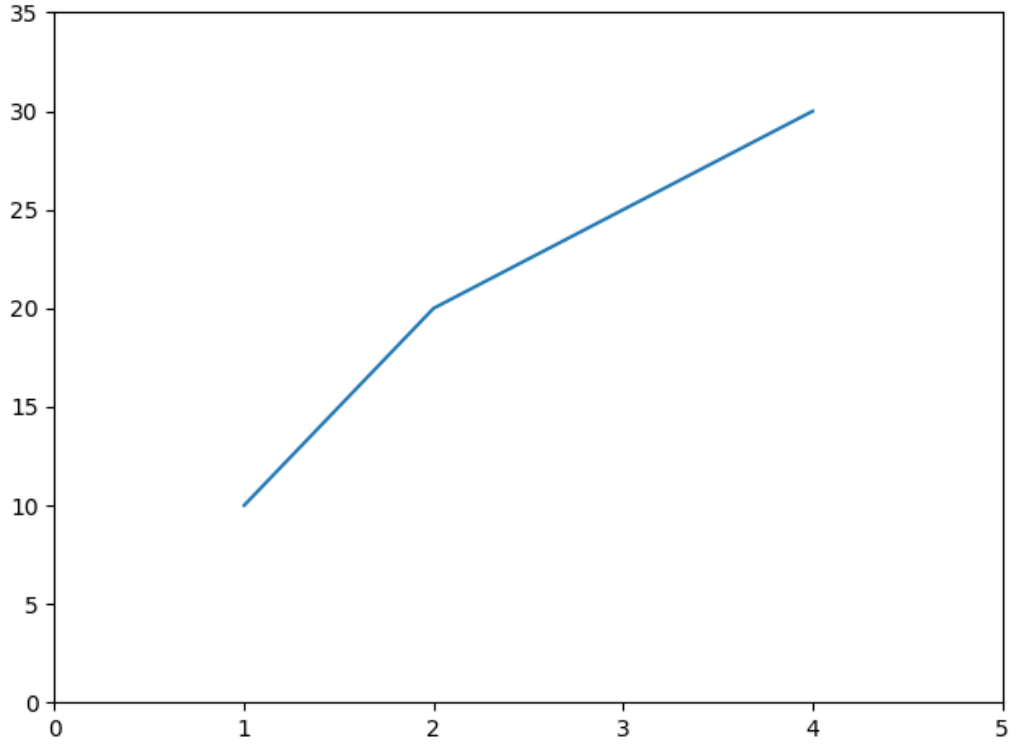
Controlling axes, ticks, labels, legends

Controlling axes

- The pyplot interface, designed for interactive use, consists of methods like xlim, xticks, and xticklabels.
- These control the plot range, tick locations, and tick labels, respectively.
- They can be used in two ways:
  - Called with no arguments returns the current parameter value. For example plt.xlim() returns the current X axis plotting range.
  - Called with parameters sets the parameter value. So plt.xlim([0, 10]), sets the X axis range to 0 to 10
- All such methods act on the active or most recently-created AxesSubplot.
- Each of them corresponds to two methods on the subplot object itself; in the case of xlim these are ax.get_xlim and ax.set_xlim.

Controlling axes

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4], [10, 20, 25, 30])
plt.xlim(0, 5)
plt.ylim(0, 35)
plt.show()
```

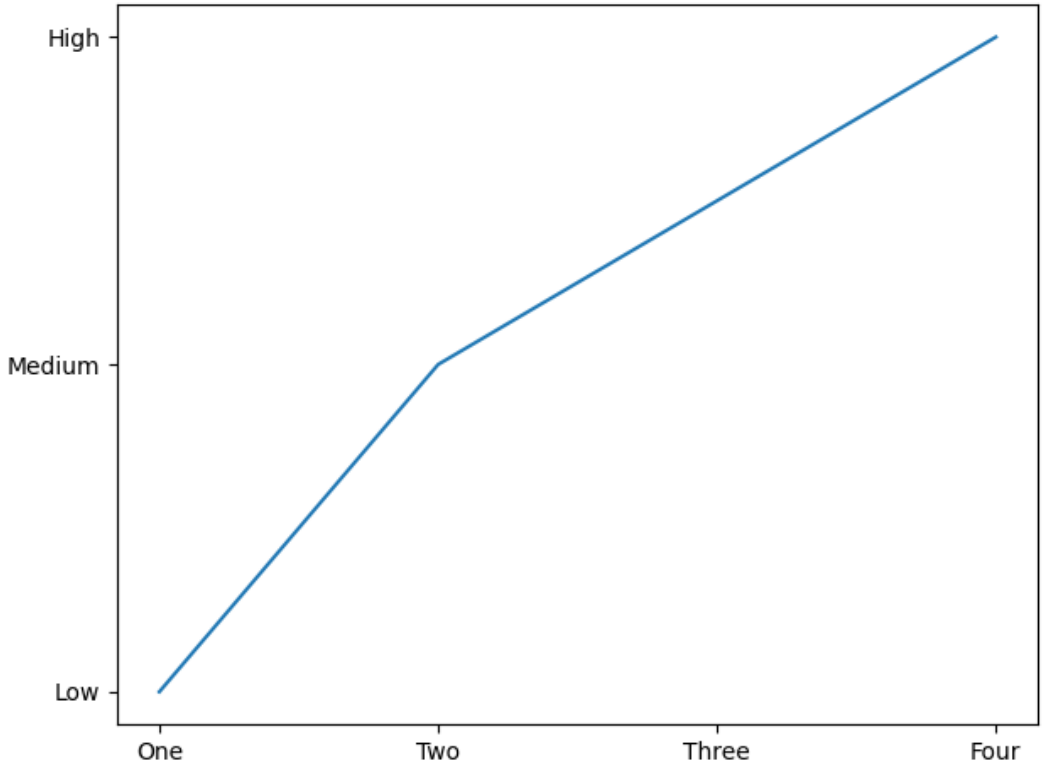


Controlling Ticks

Setting Tick Locations and Labels:

- You can manually set the locations and labels of ticks using xticks() and yticks().

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4], [10, 20, 25, 30])
plt.xticks([1, 2, 3, 4], ['One', 'Two', 'Three', 'Four’])
plt.yticks([10, 20, 30], ['Low', 'Medium', 'High’])
plt.show()
```

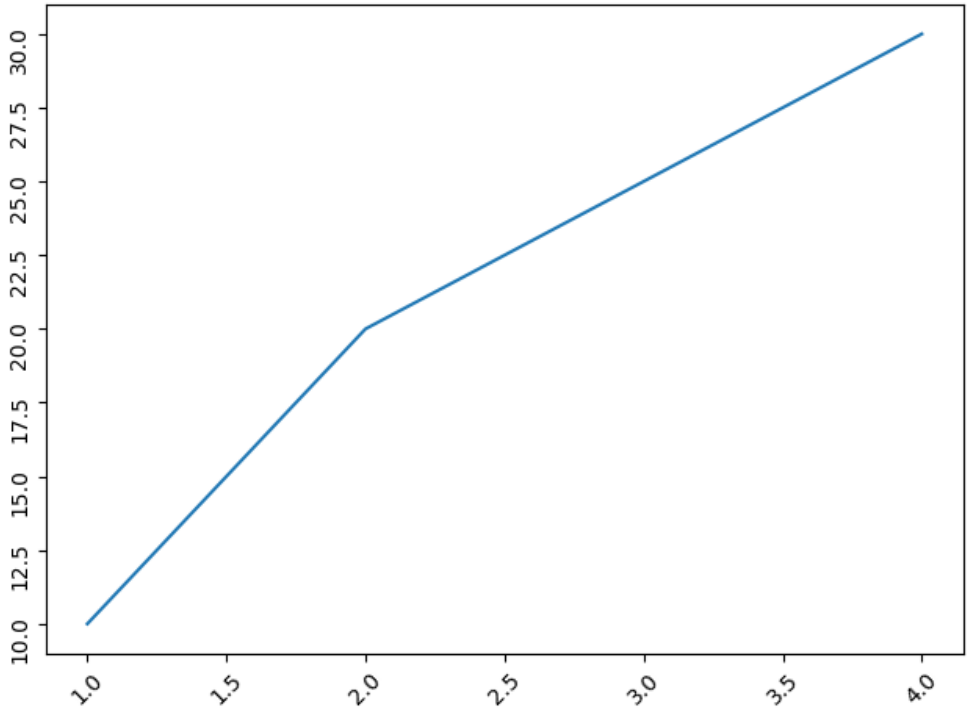


Controlling Ticks

Rotating Tick Labels:

- Tick labels can be rotated using the rotation parameter.

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4], [10, 20, 25, 30])
plt.xticks(rotation=45)
plt.yticks(rotation=90)
plt.show()
```

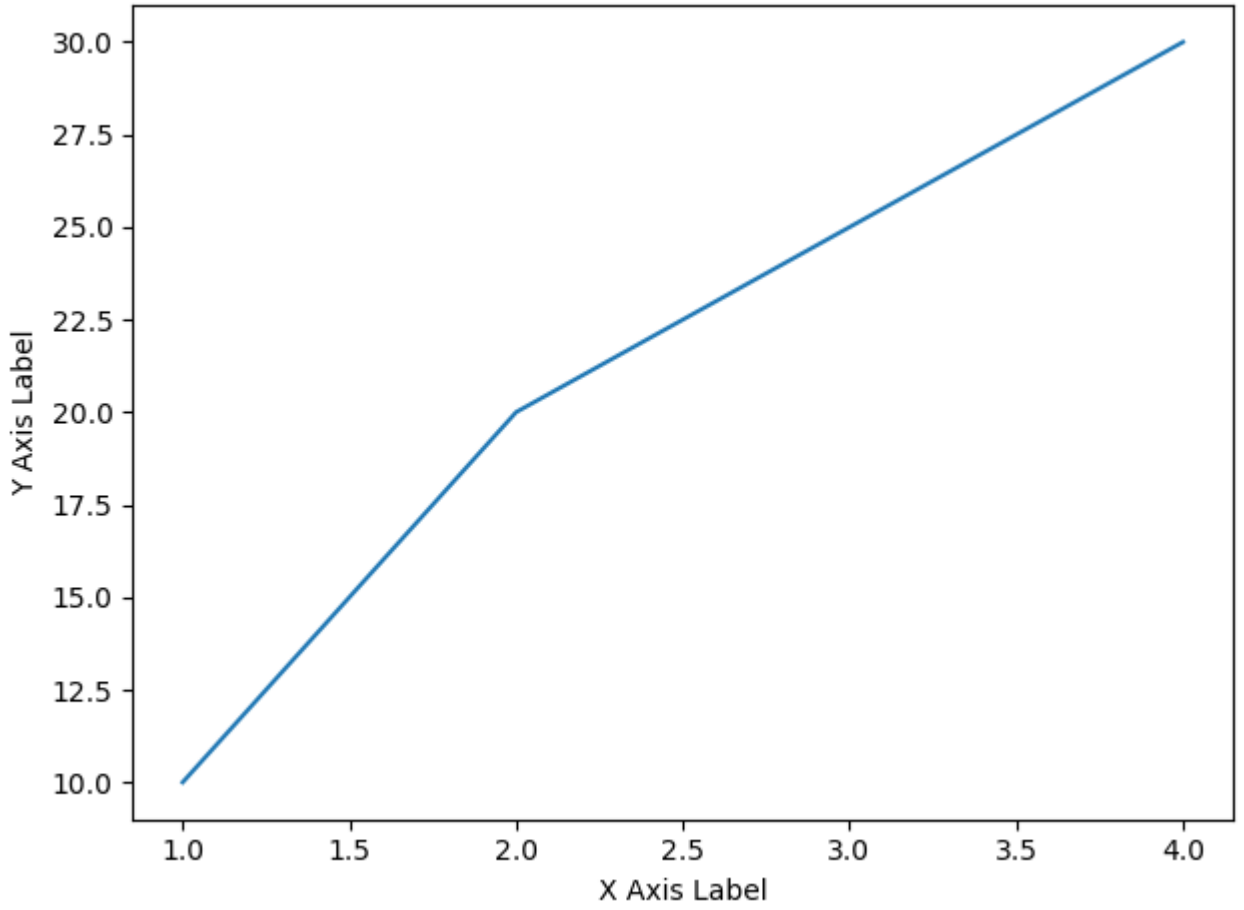


Controlling Labels

Setting Axis Labels:

- Axis labels can be set using xlabel() and ylabel().

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4], [10, 20, 25, 30])
plt.xlabel('X Axis Label’)
plt.ylabel('Y Axis Label’)
plt.show()
```

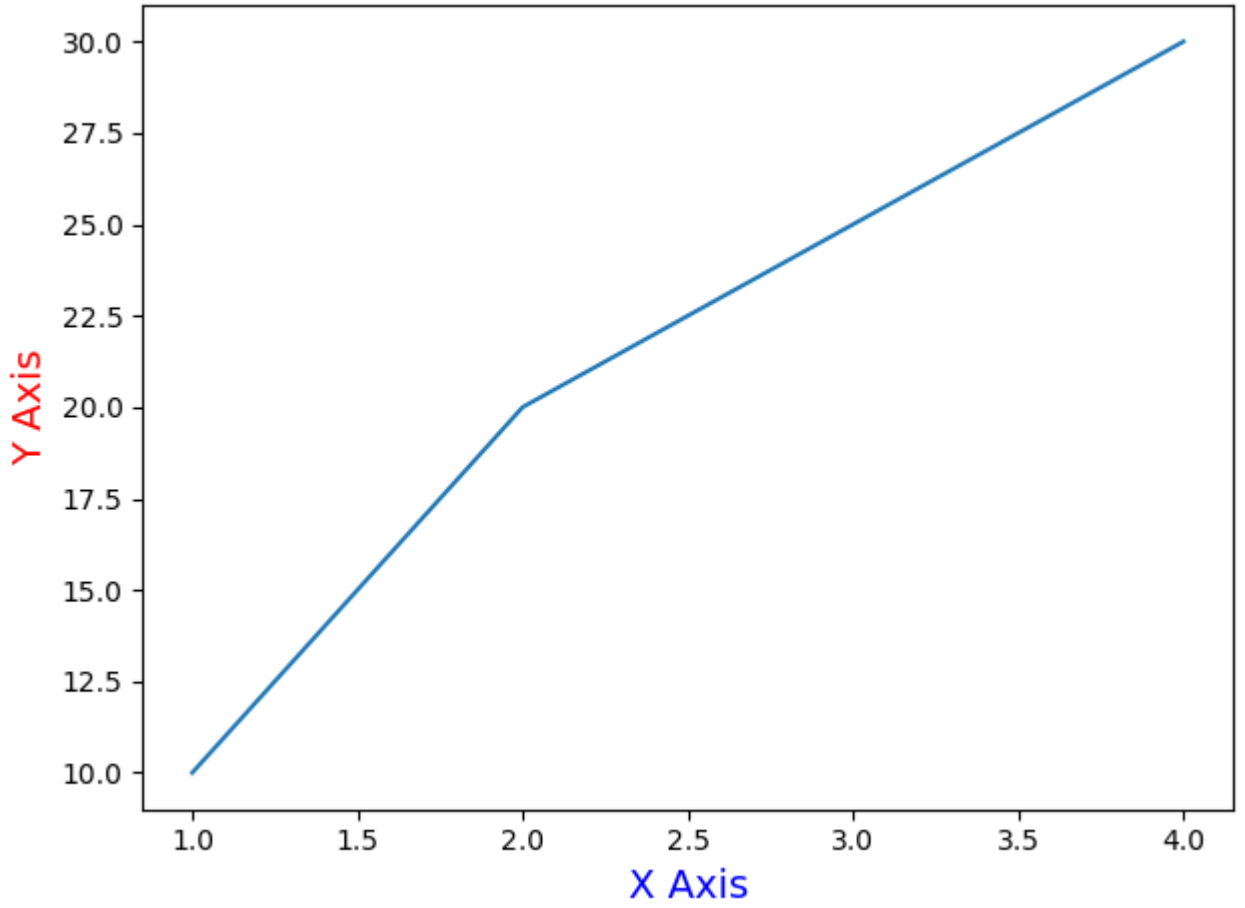


Controlling Labels

Customizing Labels:

- You can customize the font, size, color, and more.

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4], [10, 20, 25, 30])
plt.xlabel('X Axis', fontsize=14, color='blue’)
plt.ylabel('Y Axis', fontsize=14, color='red’)
plt.show()
```

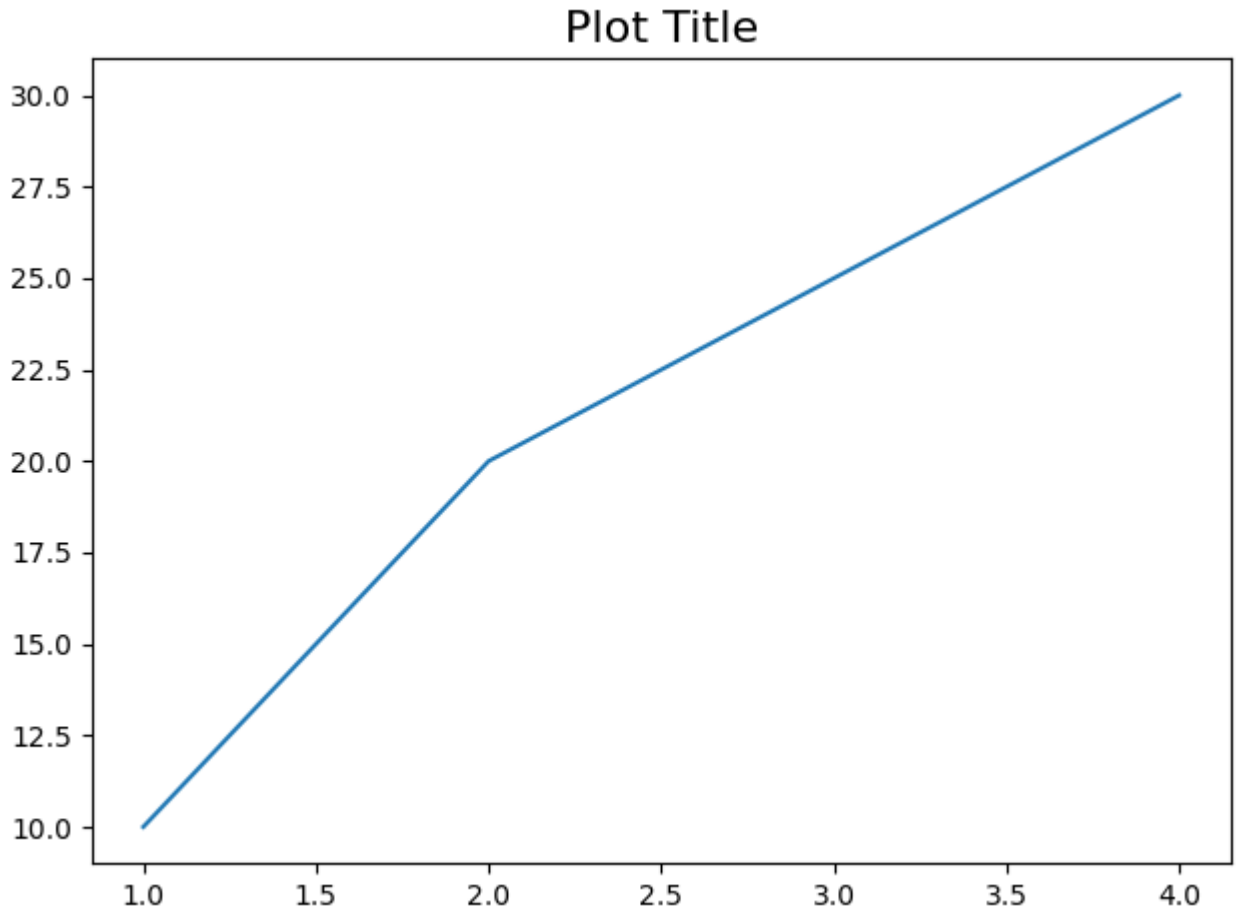


Controlling Labels

Title:

- Set the title of the plot using title().

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4], [10, 20, 25, 30])
plt.title('Plot Title', fontsize=16)
plt.show()
```

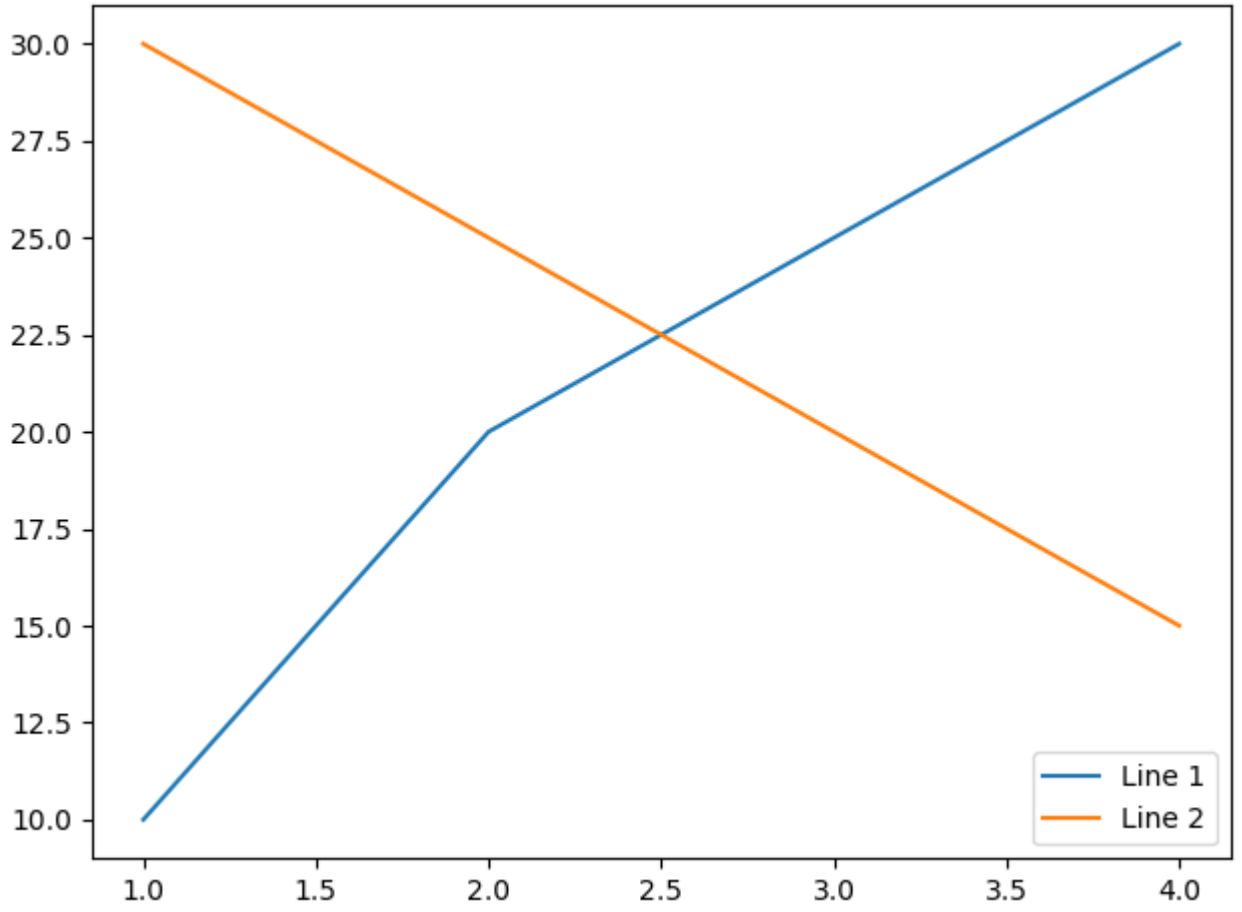


Controlling Legends

Adding a Legend:

- A legend can be added using the legend() function.

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4], [10, 20, 25, 30], label='Line 1’)
plt.plot([1, 2, 3, 4], [30, 25, 20, 15], label='Line 2’)
plt.legend()
plt.show()
```

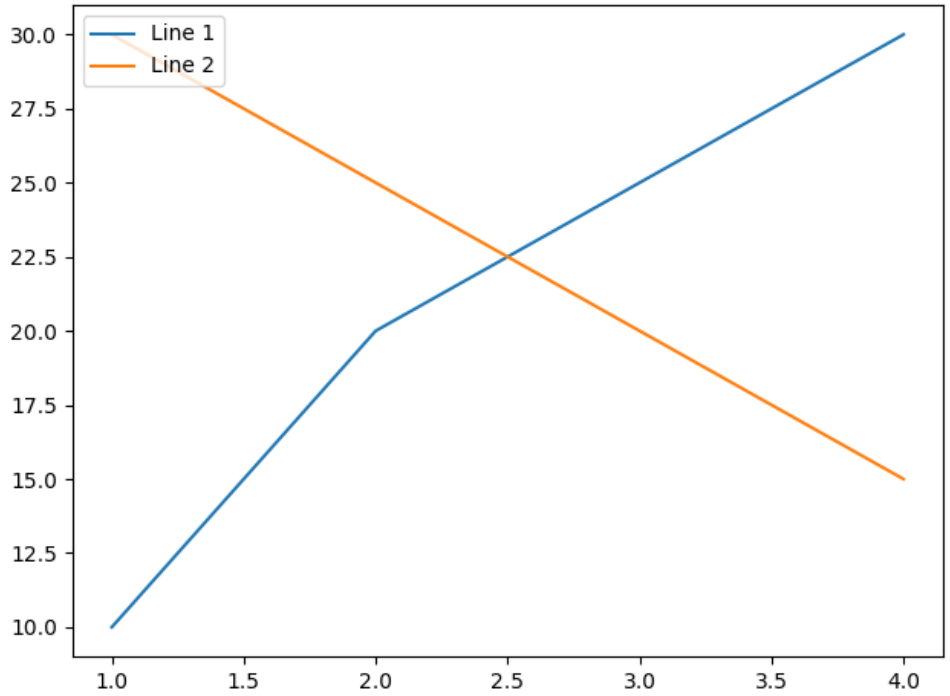


Controlling Legends

Legend Position:

- The position of the legend can be controlled using the loc parameter.

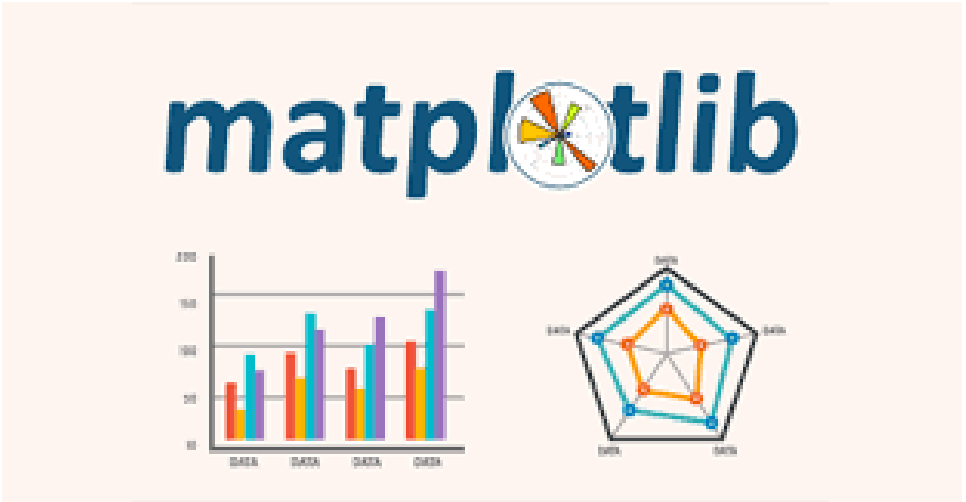
```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4], [10, 20, 25, 30], label='Line 1’)
plt.plot([1, 2, 3, 4], [30, 25, 20, 15], label='Line 2’)
plt.legend(loc='upper left’)
plt.show()
```



Annotations, saving plots

Annotations

- In Matplotlib library **annotations** refer to the capability of adding text or markers to specific locations on a plot to provide additional information or highlight particular features.
- Annotations allow users to label data points and indicate trends or add descriptions to different parts of a plot.



Annotations

- **Text annotations** are used to add explanatory or descriptive text to specific points, regions, or features within a plot.
- **Marker annotations** involve placing markers or symbols on specific points of interest within a plot to highlight or provide additional information about those points.
- **Callouts** refer to a specific type of annotation that uses visual elements like arrows, lines, or text to draw attention to a particular area or feature within a plot.

Annotations Syntax

plt.annotate(text, xy, xytext=None, xycoords='data', textcoords='data', arrowprops=None, annotation_clip=None, kwargs)

- **text (str)** − The text of the annotation.
- **xy (tuple or array)** − The point (x, y) to annotate.
- **xytext (tuple or array, optional)** − The position (x, y) to place the text. If **None** defaults to `xy`.
- **xycoords (str, Artist, or Transform, optional)** − The coordinate system that **`xy`** is given in. Default is 'data'.
- **textcoords (str, Artist, or Transform, optional)** − The coordinate system that **`xytext`** is given in. Default is 'data'.

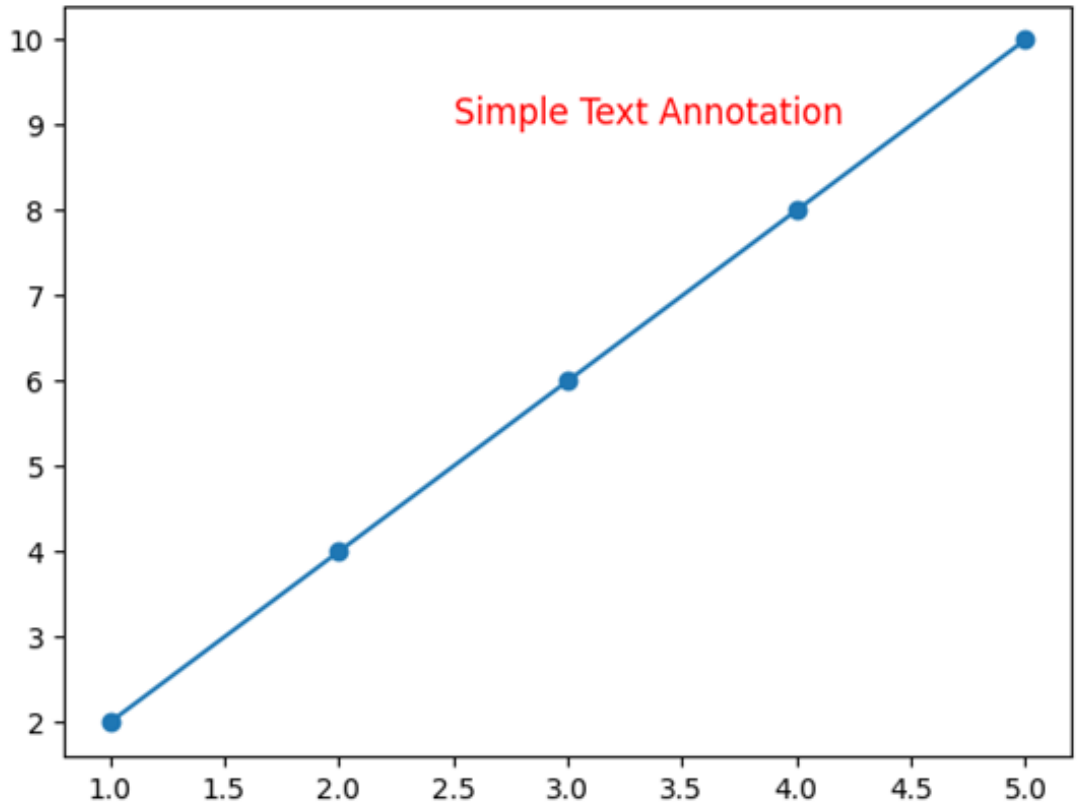
Annotations Syntax

plt.annotate(text, xy, xytext=None, xycoords='data', textcoords='data', arrowprops=None, annotation_clip=None, kwargs)

- **arrowprops (dict, optional)** − A dictionary of arrow properties. If not None an arrow is drawn from the annotation to the text.
- **annotation_clip (bool or None, optional)** − If **True** the text will only be drawn when the annotation point is within the axes. If **None** it will take the value from **rcParams["text.clip"]**.
- **kwargs (optional)** − Additional keyword arguments that are passed to **Text**.

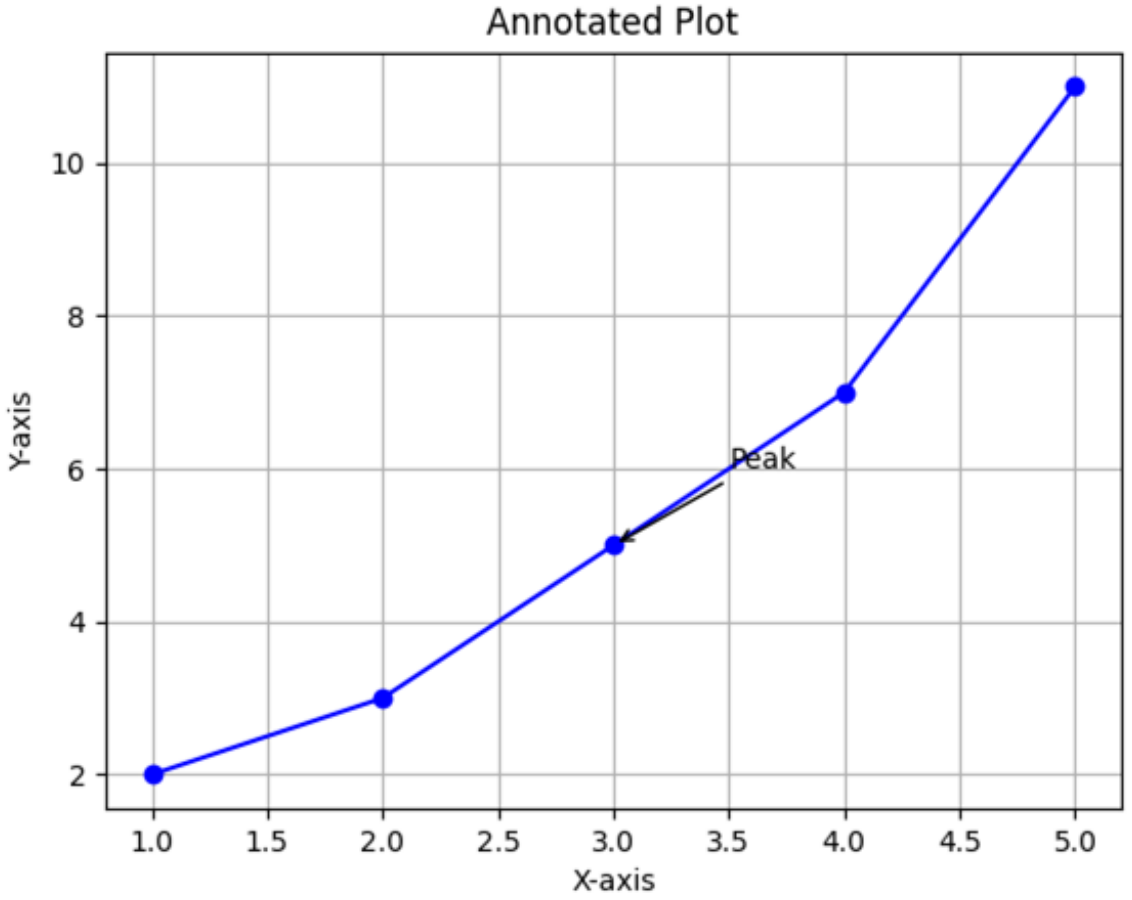
Simple Text Annotation

```
import matplotlib.pyplot as plt
# Sample data
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]
# Create plot
plt.plot(x, y, marker='o')
# Annotate a point with just text (no arrow)
plt.annotate('Simple Text Annotation',
             xy=(4, 8),
             xytext=(2.5, 9),
             fontsize=12, color='red')
plt.show()
```



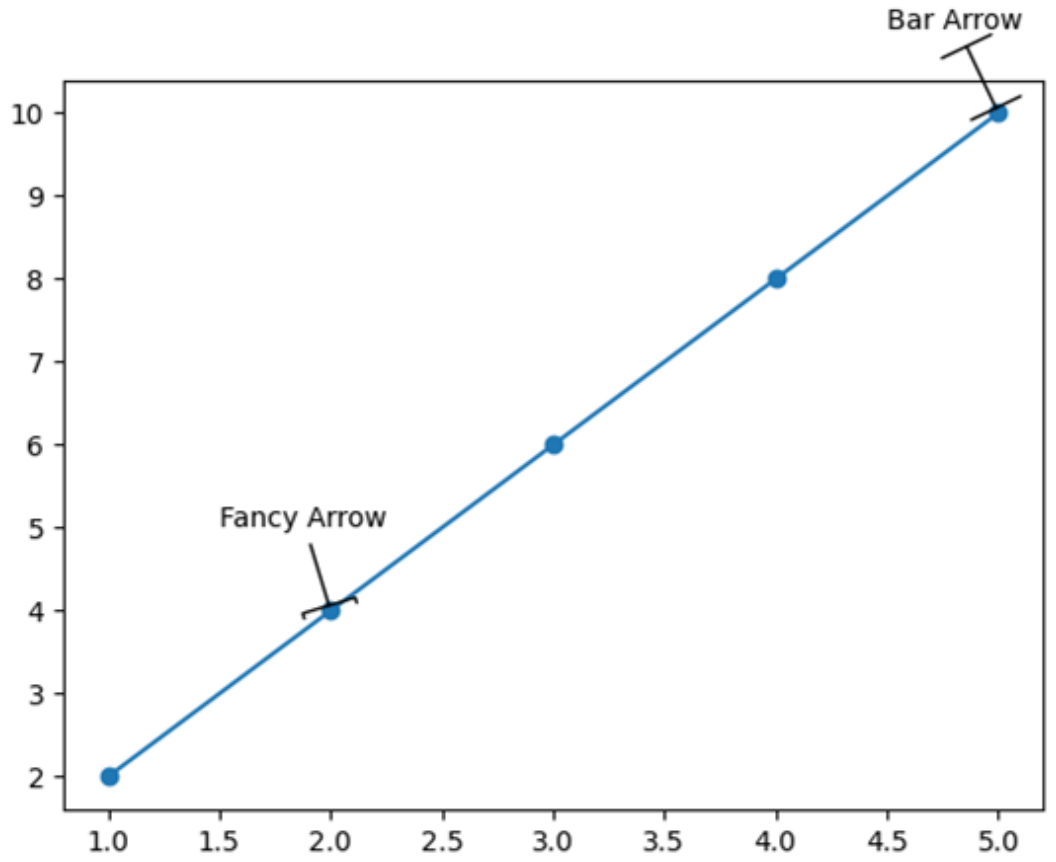
Annotation with Arrow

```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4, 5]
y = [2, 3, 5, 7, 11]
plt.plot(x, y, marker='o', linestyle='-', color='blue’)
plt.annotate('Peak', xy=(3, 5), xytext=(3.5, 6),
arrowprops=dict(facecolor='black', arrowstyle='->'),
fontsize=10)
plt.xlabel('X-axis’)
plt.ylabel('Y-axis’)
plt.title('Annotated Plot’)
plt.grid(True)
plt.show()
```



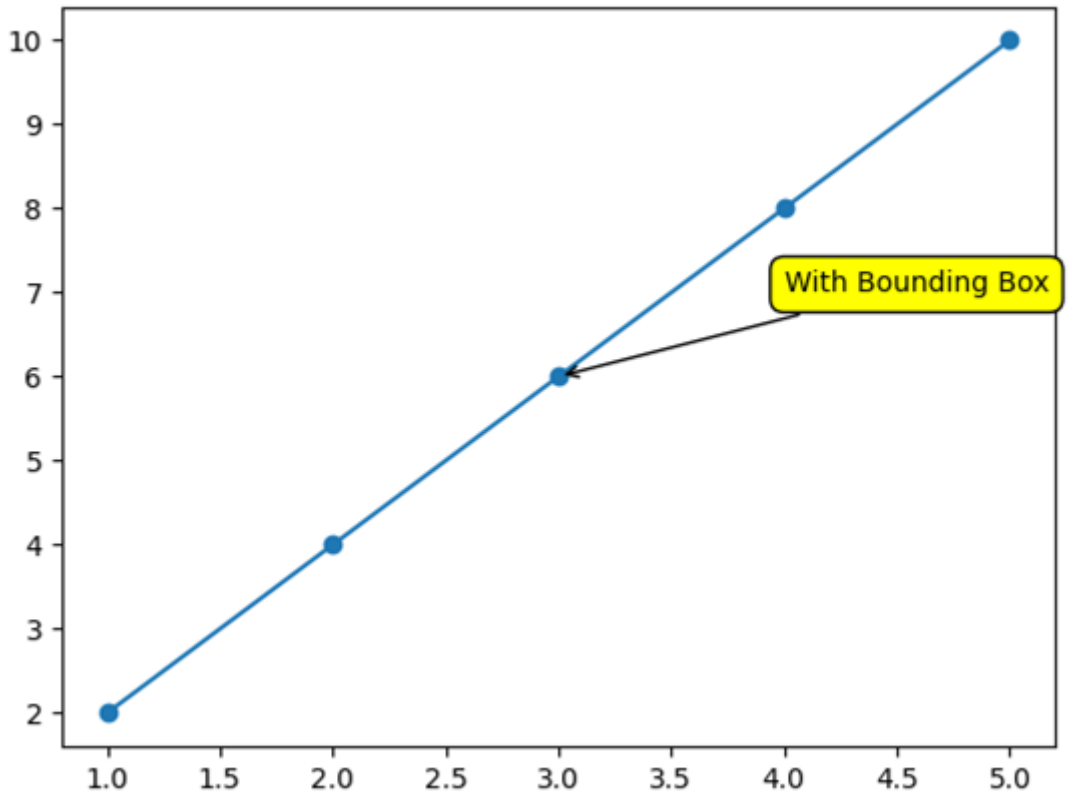
Annotation with different Arrow Styles

```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]
plt.plot(x, y, marker='o')
plt.annotate('Fancy Arrow',
        xy=(2, 4),  xytext=(1.5, 5),
        arrowprops=dict(arrowstyle='-[', facecolor='purple'))
plt.annotate('Bar Arrow',
        xy=(5, 10), xytext=(4.5, 11),
        arrowprops=dict(arrowstyle='|-|', facecolor='orange'))
plt.show()
```



Annotation with bounding box around the text

```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]
plt.plot(x, y, marker='o')
plt.annotate('With Bounding Box',  xy=(3, 6), xytext=(4, 7),
        bbox=dict(boxstyle='round,pad=0.5',  edgecolor='black',
facecolor='yellow'),        arrowprops=dict(facecolor='blue',
arrowstyle='->'))
plt.show()
```



Saving Plots

- **Saving figures** in Matplotlib library allows us to export our plots to various file formats such as PNG, PDF, SVG and so on to use those saved plots in various reports, presentations or publications.
- Matplotlib library provides the savefig() function for to save the plot that we have created.

Common File Formats for Saving

- **PNG (.png)** – Good for general-purpose images which supports transparency.
- **JPEG (.jpg)** – Suitable for images with smooth gradients but may lose some quality due to compression.
- **PDF (.pdf)** – Ideal for vector-based images scalable without loss of quality.
- **SVG (.svg)** – Scalable Vector Graphics which suitable for web-based or vector-based graphics.

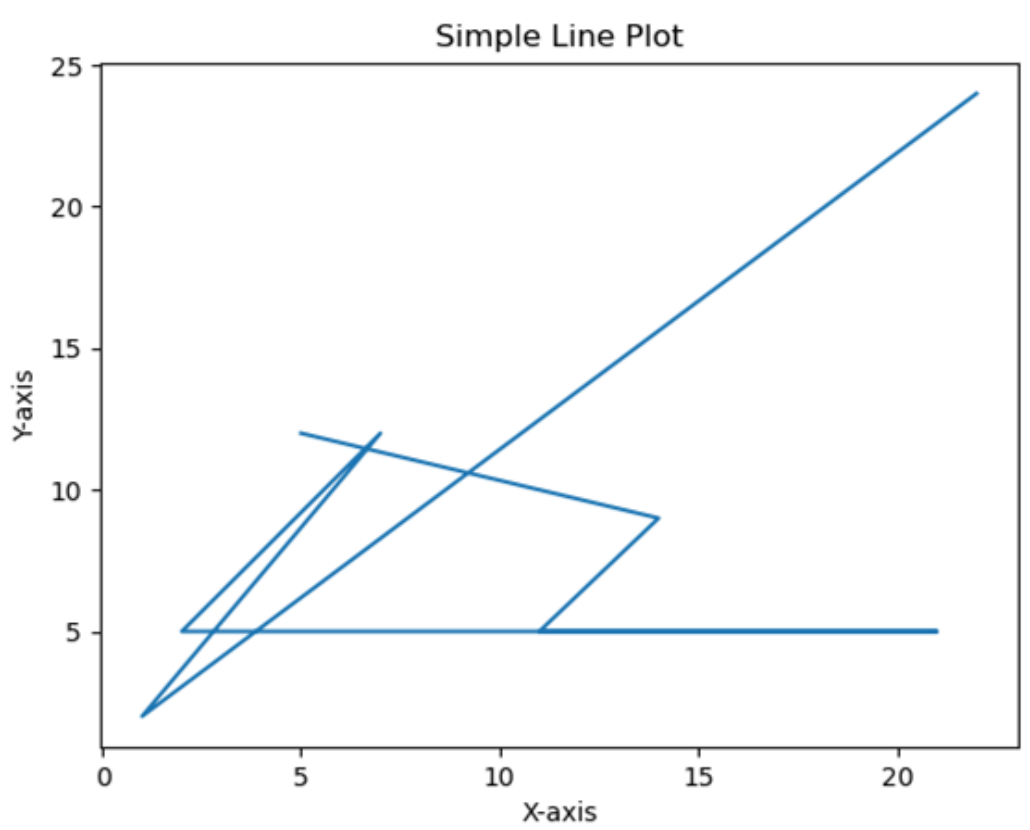
Saving Plots

Syntax: plt.savefig(fname, dpi=None, bbox_inches='tight', pad_inches=0.1, format=None, kwargs)

- **fname** − The file name or path of the file to save the figure. The file extension determines the file format such as **".png"**, **".pdf"**.
- **dpi** − Dots per inch i.e. resolution for the saved figure. Default is **"None"** which uses the Matplotlib default.
- **bbox_inches** − Specifies which part of the figure to save. Options include 'tight', 'standard' or a specified bounding box in inches.
- **pad_inches** − Padding around the figure when **bbox_inches='tight'**.
- **format** − Explicitly specify the file format. If **'None'** the format is inferred from the file extension in **fname**.
- **kwargs** − Additional keyword arguments specific to the chosen file format.

Saving Plots

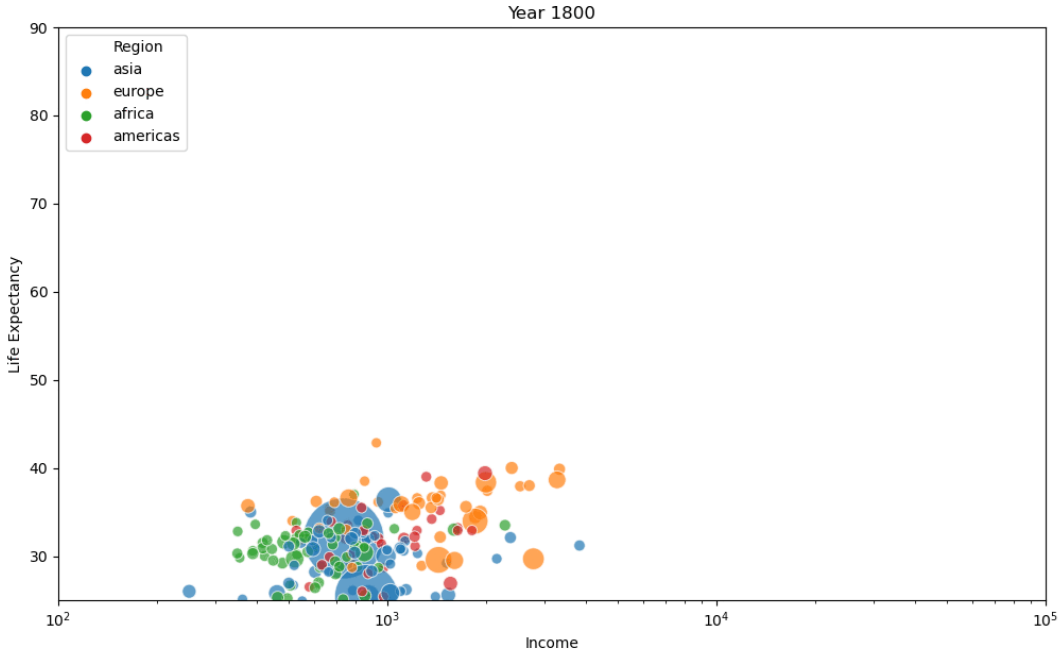
```
import matplotlib.pyplot as plt
x = [22,1,7,2,21,11,14,5]
y = [24,2,12,5,5,5,9,12]
plt.plot(x,y)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Simple Line Plot')
plt.savefig('matplotlib/Savefig/lineplot.png')
plt.show()
```



Seaborn Library

Seaborn Library

- **Seaborn** is a library mostly used for statistical plotting in Python.
- It is built on top of Matplotlib and provides beautiful default styles and color palettes to make statistical plots more attractive.



Features of Seaborn

- **Statistical Graphics:** Seaborn is specifically designed for creating statistical graphics, providing built-in functions for common visualizations like scatter plots, line plots, histograms, and more. This makes it easier to create visually appealing and informative plots for data analysis.

- **Data Visualization Themes:** Seaborn offers pre-defined styles and themes that can quickly change the overall appearance of your plots. This helps create consistent and aesthetically pleasing visualizations without requiring extensive customization.

- **Integration with Pandas and NumPy:** Seaborn seamlessly integrates with Pandas and NumPy, making it easy to work with dataframes and arrays directly. This simplifies the workflow and reduces the amount of code needed for data analysis and visualization.

Features of Seaborn

- **FacetGrid and Pair Plots:** Seaborn provides FacetGrid for grouping data and creating subplots based on categorical variables. This is useful for comparing distributions or relationships across different groups. Pair plots allow you to visualize the relationships between all pairs of numeric columns in a DataFrame, helping you identify correlations and patterns.

- **Customization and Flexibility:** While Seaborn provides a high-level interface, it's built on top of Matplotlib, giving you access to its extensive customization options. This allows you to fine-tune your plots to meet your specific needs.

- **Ease of Use:** Seaborn's API is designed to be user-friendly and intuitive, making it easier to learn and use compared to Matplotlib. Its documentation is also well-written and provides clear examples.

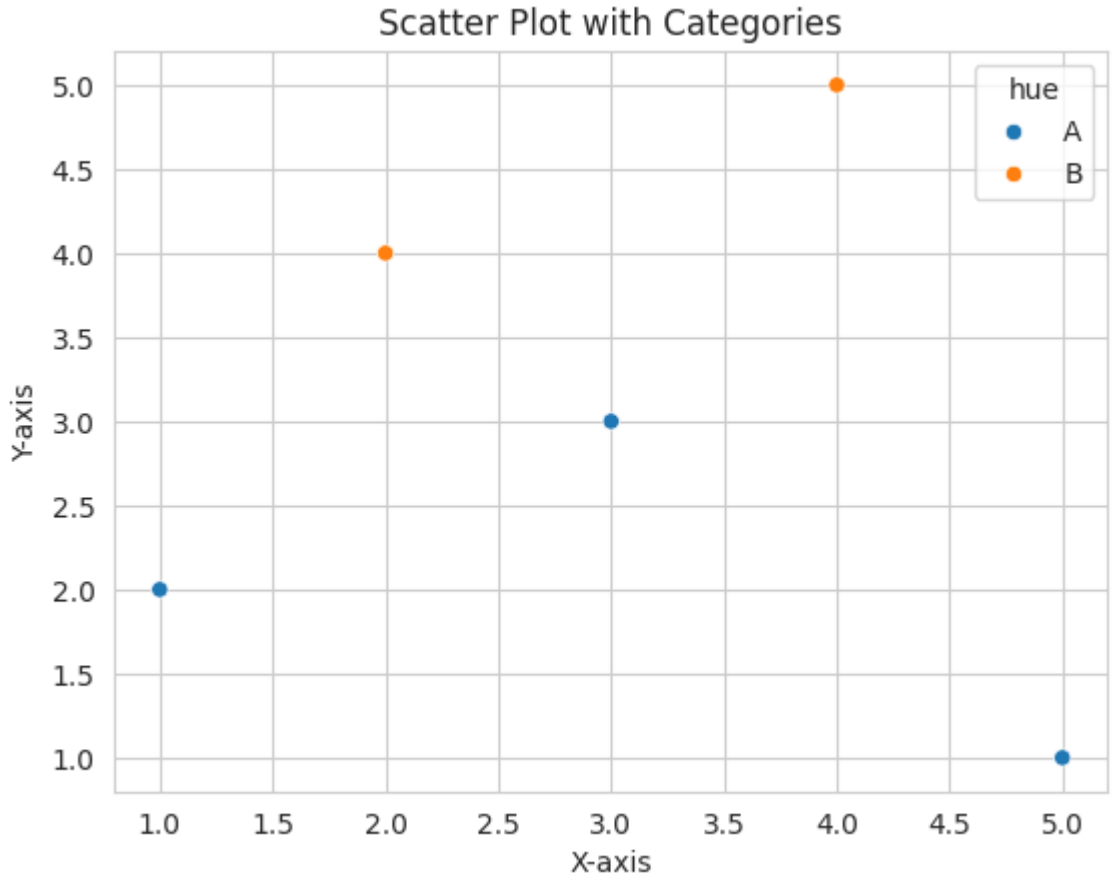
Statistical Graphics

Example

```
import seaborn as sns
import pandas as pd

# Sample data
data = {
    "x": [1, 2, 3, 4, 5],
    "y": [2, 4, 3, 5, 1],
    "hue": ["A", "B", "A", "B", "A"]
}
df = pd.DataFrame(data)
# Create a scatter plot with hue to differentiate
categories
sns.scatterplot(x="x", y="y", hue="hue", data=df)
# Add a title and labels
plt.title("Scatter Plot with Categories")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.show()
```

Output



Built-in libraries

Example

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

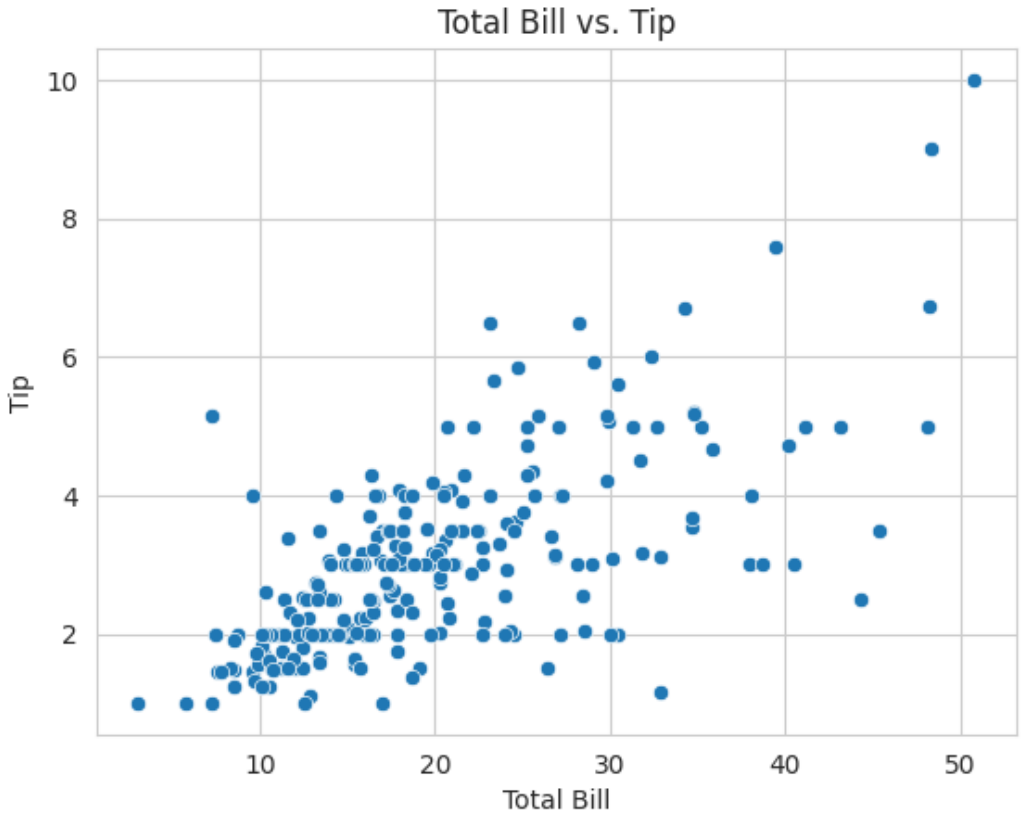
# Load the tips dataset
tips = sns.load_dataset("tips")

# Create a scatter plot
sns.scatterplot(x="total_bill", y="tip", data=tips

# Add  a title and labels
plt.title("Total Bill vs. Tip")
plt.xlabel("Total Bill")
plt.ylabel("Tip")

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

Output



Joint Plots

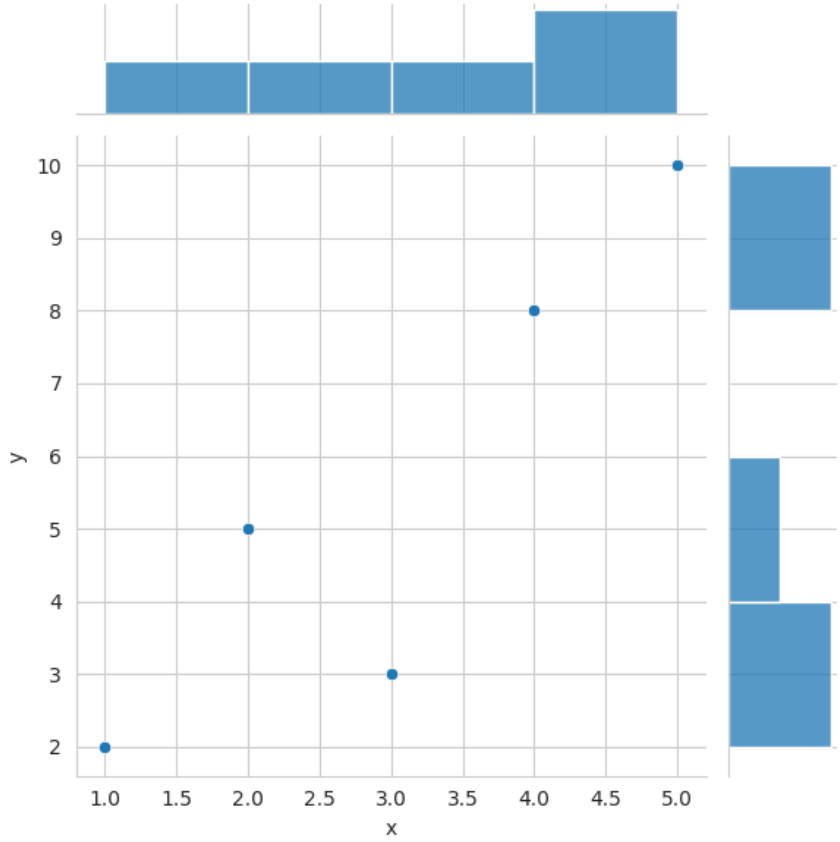
Example

```
import seaborn as sns
import pandas as pd

# Sample data
data = {
    "x": [1, 2, 3, 4, 5],
    "y": [2, 5, 3, 8, 10],
   }
sns.jointplot(x='x', y='y', kind="scatter", data
=data)

plt.show()
```

Output



Pair Plots

Example

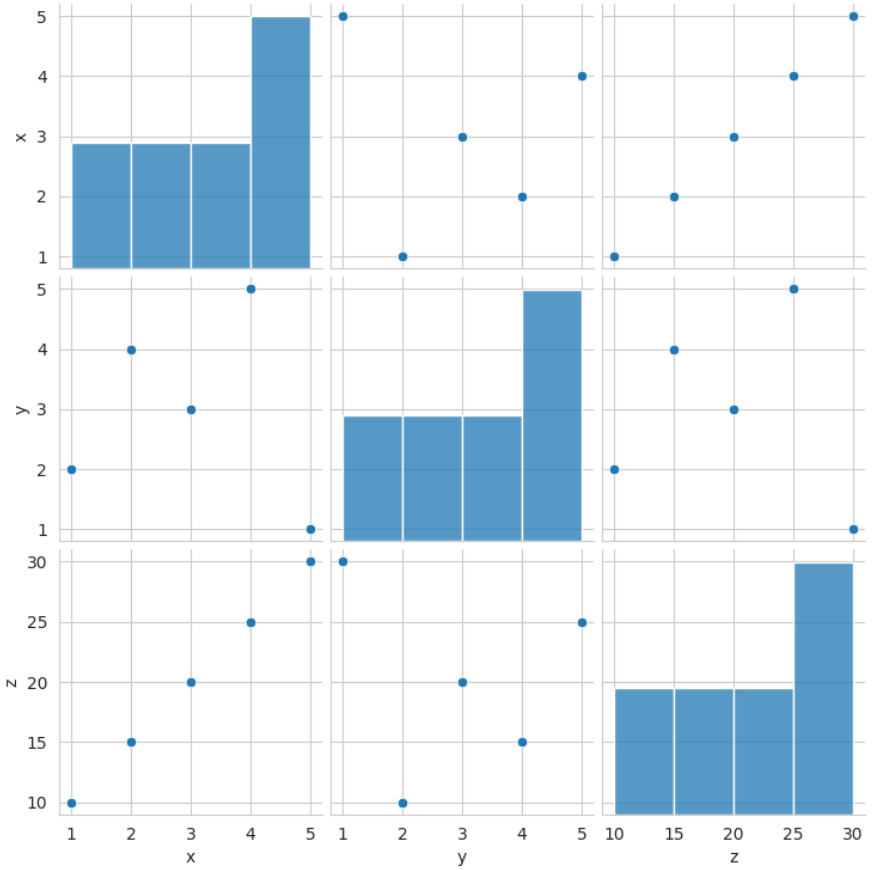
```
import seaborn as sns

# Sample data
data = {
    "x": [1, 2, 3, 4, 5],
    "y": [2, 4, 3, 5, 1],
    "z": [10, 15, 20, 25, 30]
}

df = pd.DataFrame(data)

sns.pairplot(df)
plt.show()
```

Output



Grid Plot

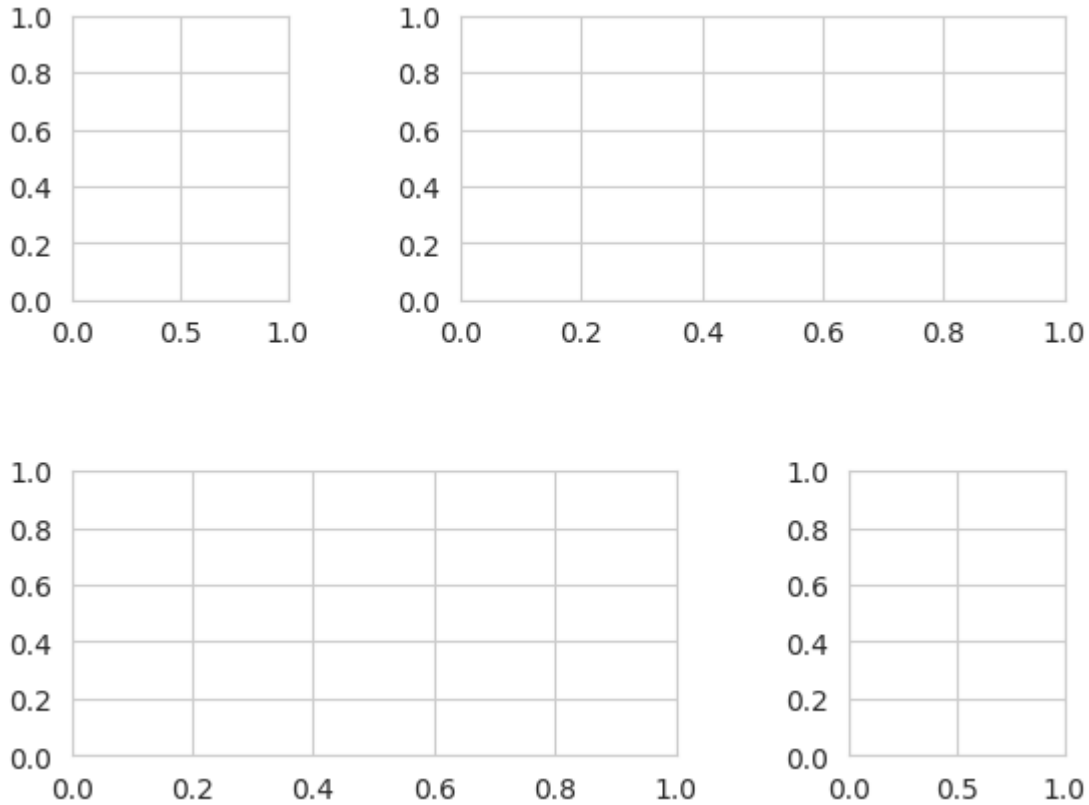
Example

```
import seaborn as sns

Grid_plot = plt.GridSpec(2, 3, wspace = 0.8,
                         hspace = 0.6)

plt.subplot(Grid_plot[0, 0])
plt.grid(True)
plt.subplot(Grid_plot[0, 1:])
plt.grid(True)
plt.subplot(Grid_plot[1, :2])
plt.grid(True)
plt.subplot(Grid_plot[1, 2])
plt.grid(True)
```

Output



Sub Plots

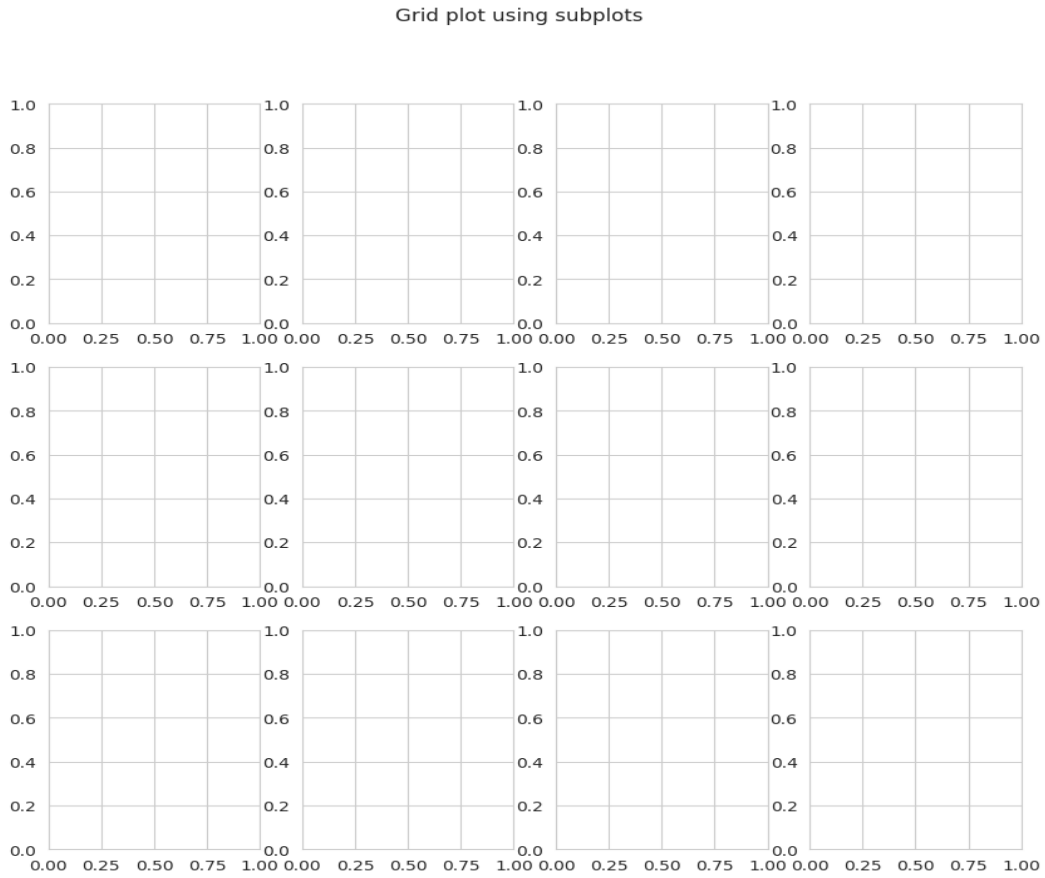
Example

```
import seaborn as sns

figure, axes = plt.subplots(3, 4,
figsize=(10, 10))

figure.suptitle('Grid plot using subplots')
```

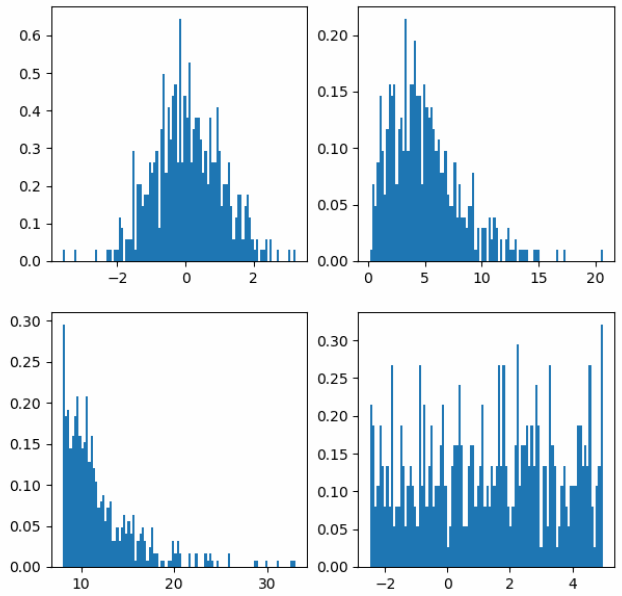
Output



Multiple Plots

Multiple Plots

- Multi-plot grid is a useful approach to draw multiple instances of the same plot on different subsets of your dataset.
- It allows a viewer to quickly extract a large amount of information about a complex dataset.



3D Plot

- **FacetGrid:** Group data by a categorical variable and plot individual subplots for each category.
- **Jointplot:** Visualize the relationship between two variables and their distributions.
- **Pairplot:** Visualize the relationships between all pairs of numeric columns in a DataFrame.
- **Gridspec:** Create custom grid layouts for subplots.
- **Subplot**: Create multiple subplots within a single figure.

Facet Grid

Example

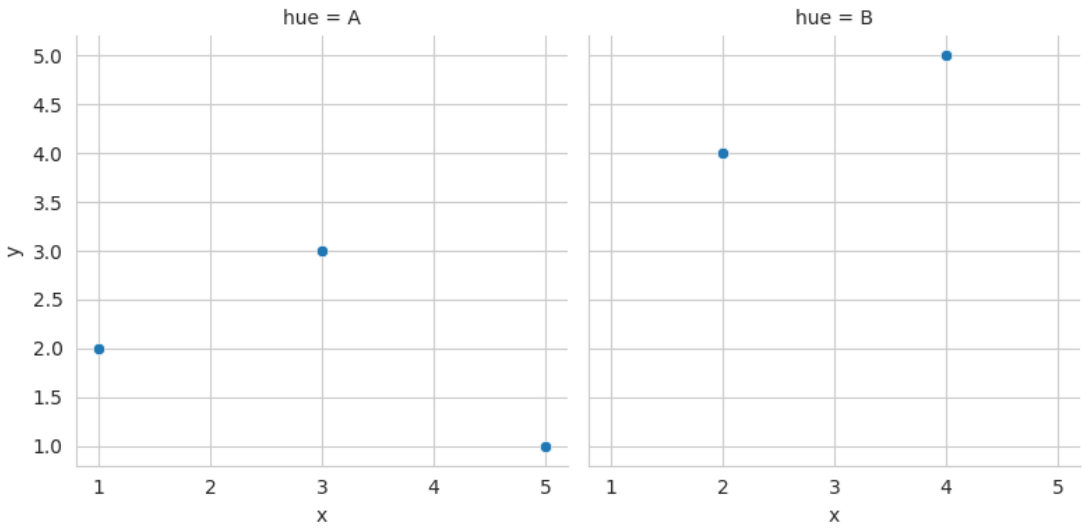
```
import seaborn as sns
import pandas as pd

# Sample data
data = {
    "x": [1, 2, 3, 4, 5],
    "y": [2, 4, 3, 5, 1],
    "hue": ["A", "B", "A", "B", "A"]
}

df = pd.DataFrame(data)
# Create a FacetGrid with one row and two columns
g = sns.FacetGrid(df, col="hue", height=4)
# Plot scatter plots for each facet
g.map(sns.scatterplot, "x", "y")

plt.show()
```

Output



Facet Grid

Example

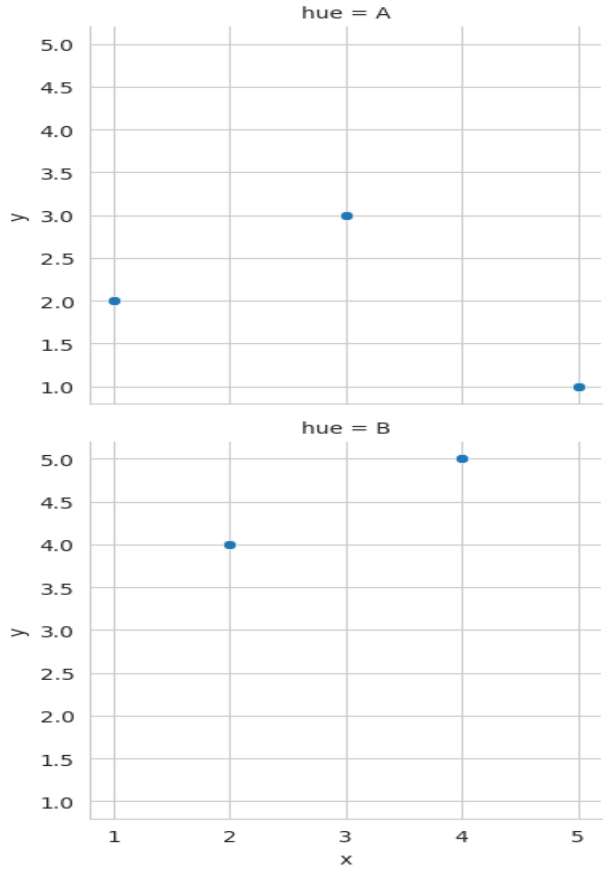
```
import seaborn as sns
import pandas as pd

# Sample data
data = {
    "x": [1, 2, 3, 4, 5],
    "y": [2, 4, 3, 5, 1],
    "hue": ["A", "B", "A", "B", "A"]
}

df = pd.DataFrame(data)
# Create a FacetGrid with one row and two columns
g = sns.FacetGrid(df, row="hue", height=4)
# Plot scatter plots for each facet
g.map(sns.scatterplot, "x", "y")

plt.show()
```

Output



Joint Plots

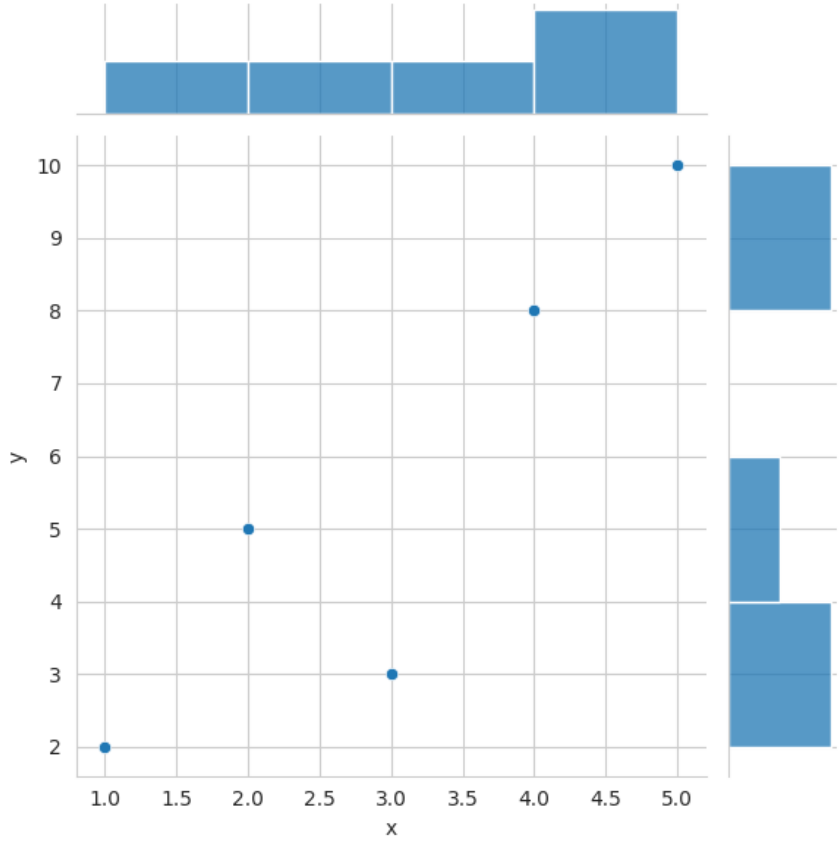
Example

```
import seaborn as sns
import pandas as pd

# Sample data
data = {
    "x": [1, 2, 3, 4, 5],
    "y": [2, 5, 3, 8, 10],
   }
sns.jointplot(x='x', y='y', kind="scatter", data
=data)

plt.show()
```

Output



Pair Plots

Example

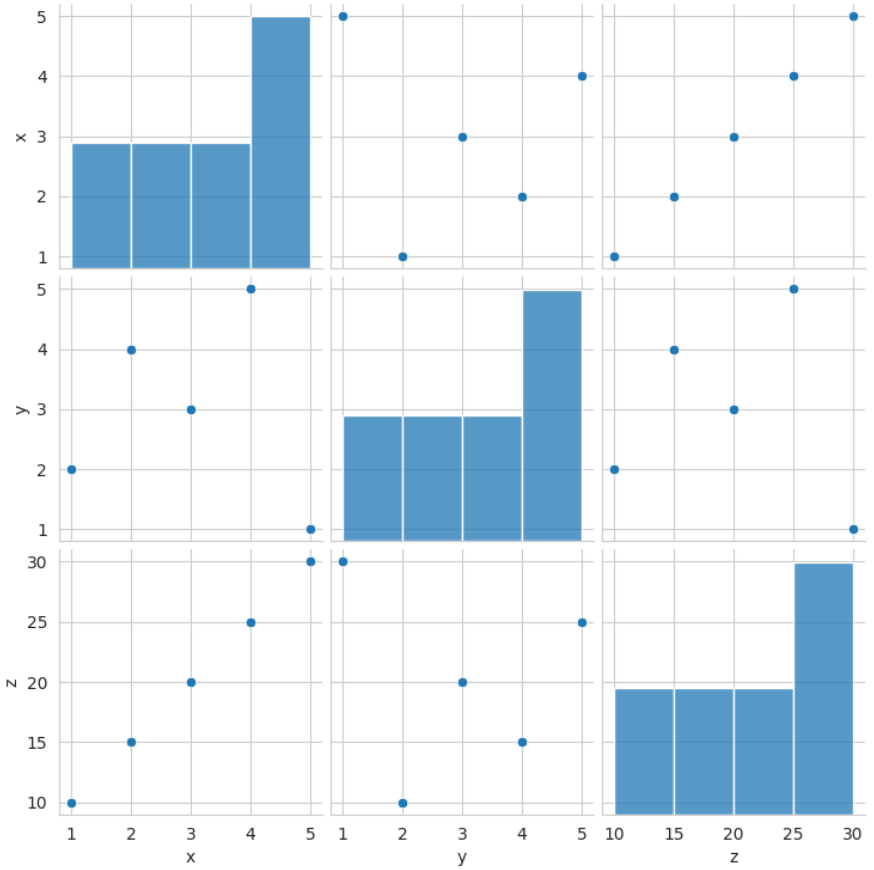
```
import seaborn as sns

# Sample data
data = {
    "x": [1, 2, 3, 4, 5],
    "y": [2, 4, 3, 5, 1],
    "z": [10, 15, 20, 25, 30]
}

df = pd.DataFrame(data)

sns.pairplot(df)
plt.show()
```

Output



Grid Plot

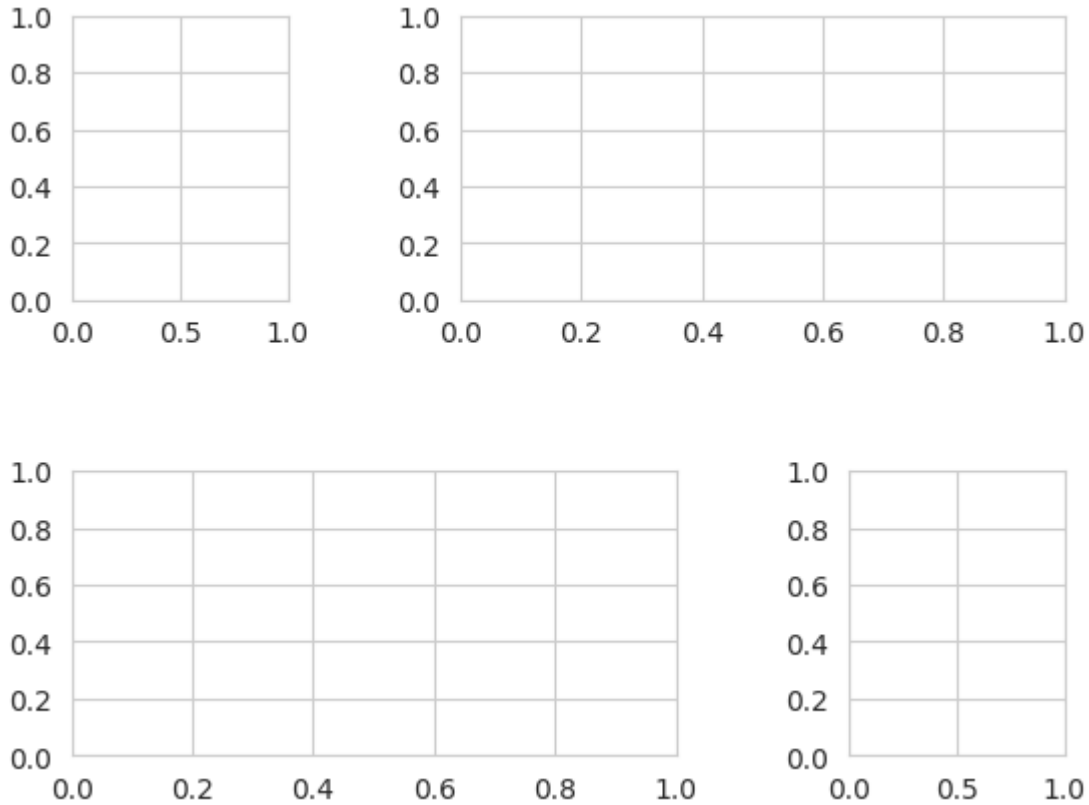
Example

```
import seaborn as sns

Grid_plot = plt.GridSpec(2, 3, wspace = 0.8,
                         hspace = 0.6)

plt.subplot(Grid_plot[0, 0])
plt.subplot(Grid_plot[0, 1:])
plt.subplot(Grid_plot[1, :2])
plt.subplot(Grid_plot[1, 2])
```

Output



Sub Plots

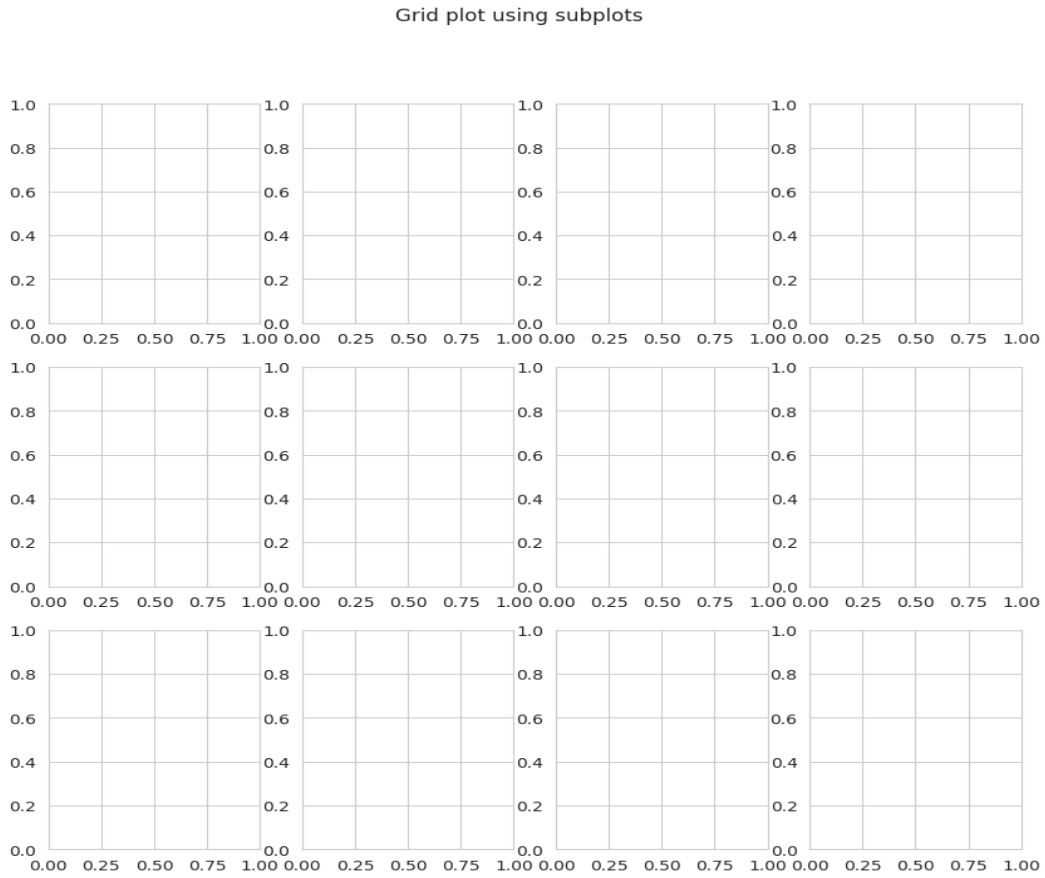
Example

```
import seaborn as sns

figure, axes = plt.subplots(3, 4,
figsize=(10, 10))

figure.suptitle('Grid plot using subplots')
```

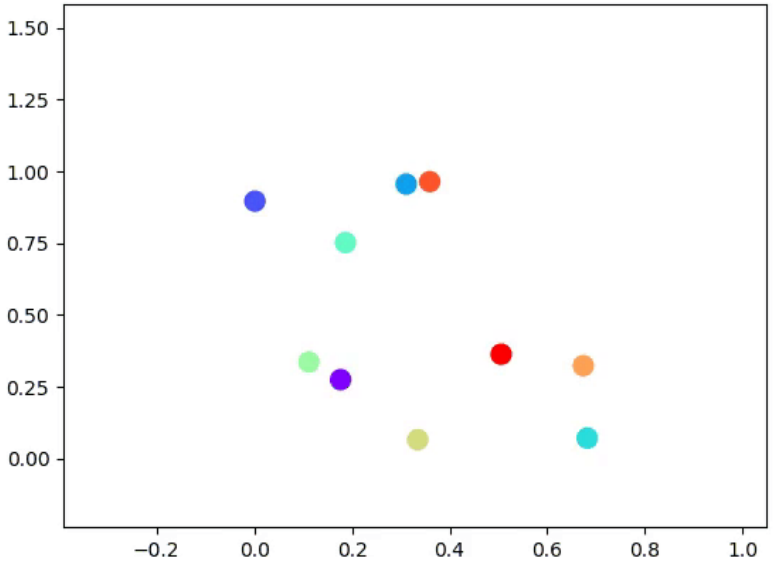
Output



Scatterplot, Lineplot and Histogram

Scatterplot

- Scatter plot is a mathematical technique that is used to represent data.
- Scatter plot also called a Scatter Graph, or Scatter Chart uses dots to describe two different numeric variables.
- The position of each dot on the horizontal and vertical axis indicates values for an individual data point.



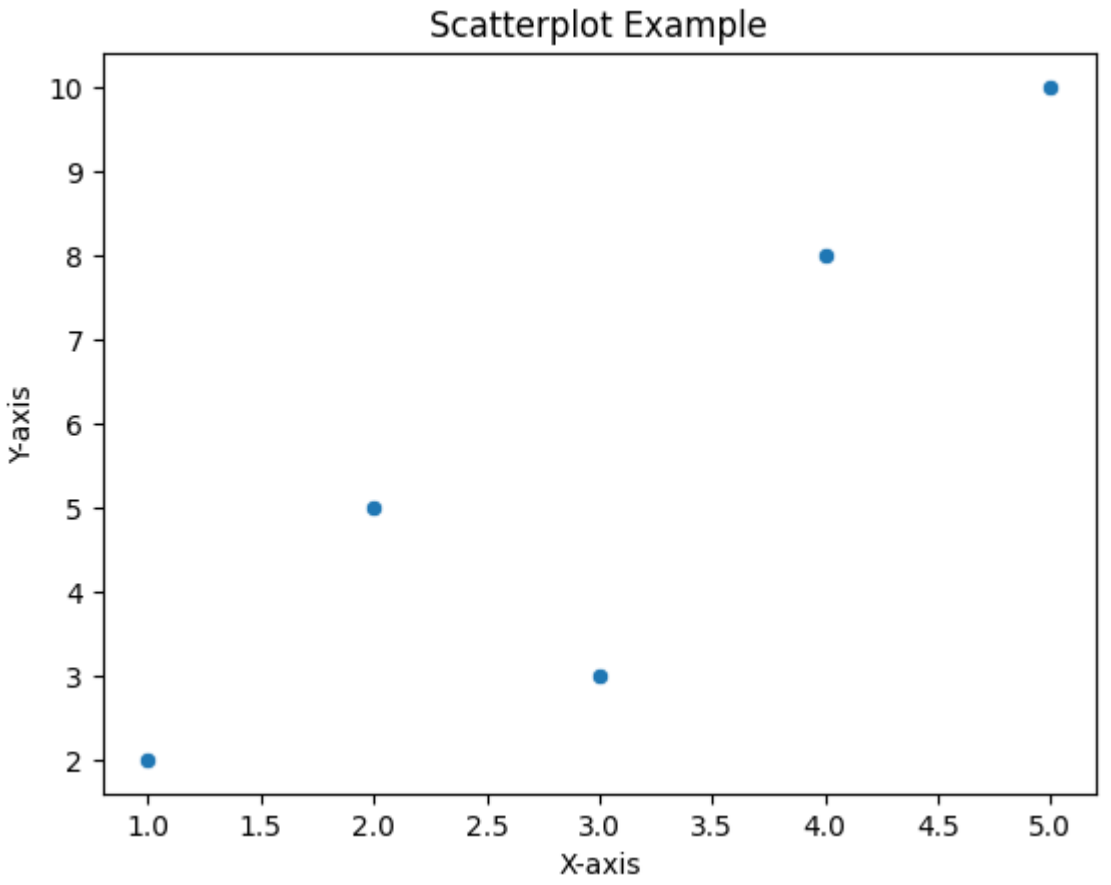
Scatterplot

Example

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

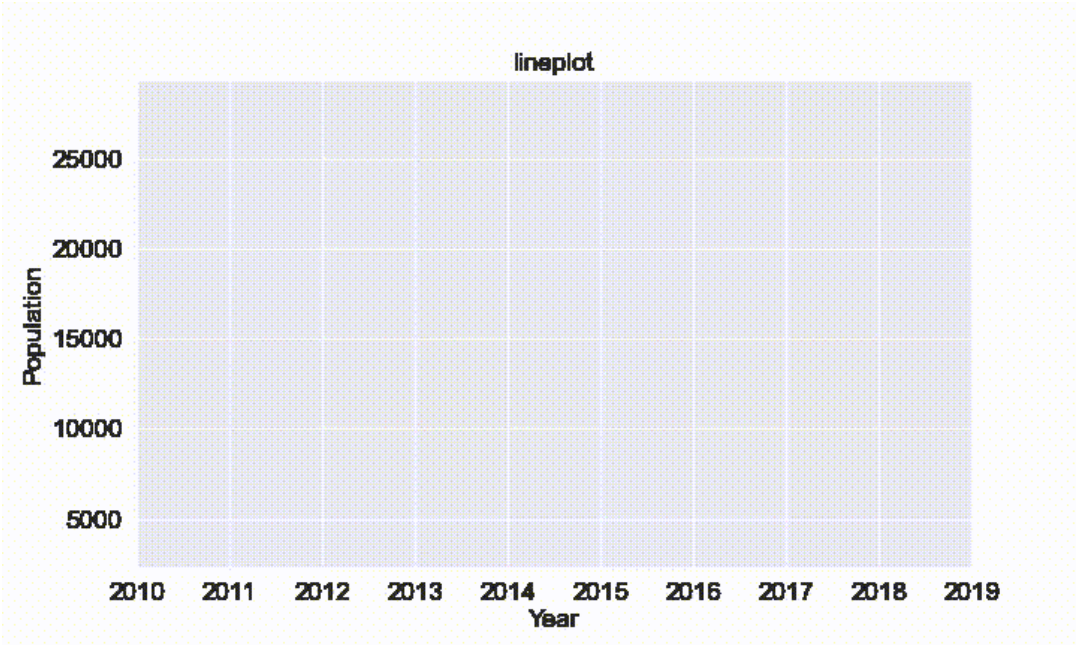
data = {
    "x_values": [1, 2, 3, 4, 5],
    "y_values": [2, 5, 3, 8, 10]
}
# Create a scatterplot
sns.scatterplot(x="x_values", y="y_values",
data=data)
# Add a title and labels
plt.title("Scatterplot Example")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
# Show the plot
plt.show()
```

Output



Lineplot

- A line plot is a graphical representation of data in which individual data points are plotted along a line to display the relationship between two variables.
- It is typically used to visualize how one variable, often referred to as the dependent variable, changes in response to changes in another variable, known as the independent variable.



Lineplot

Example

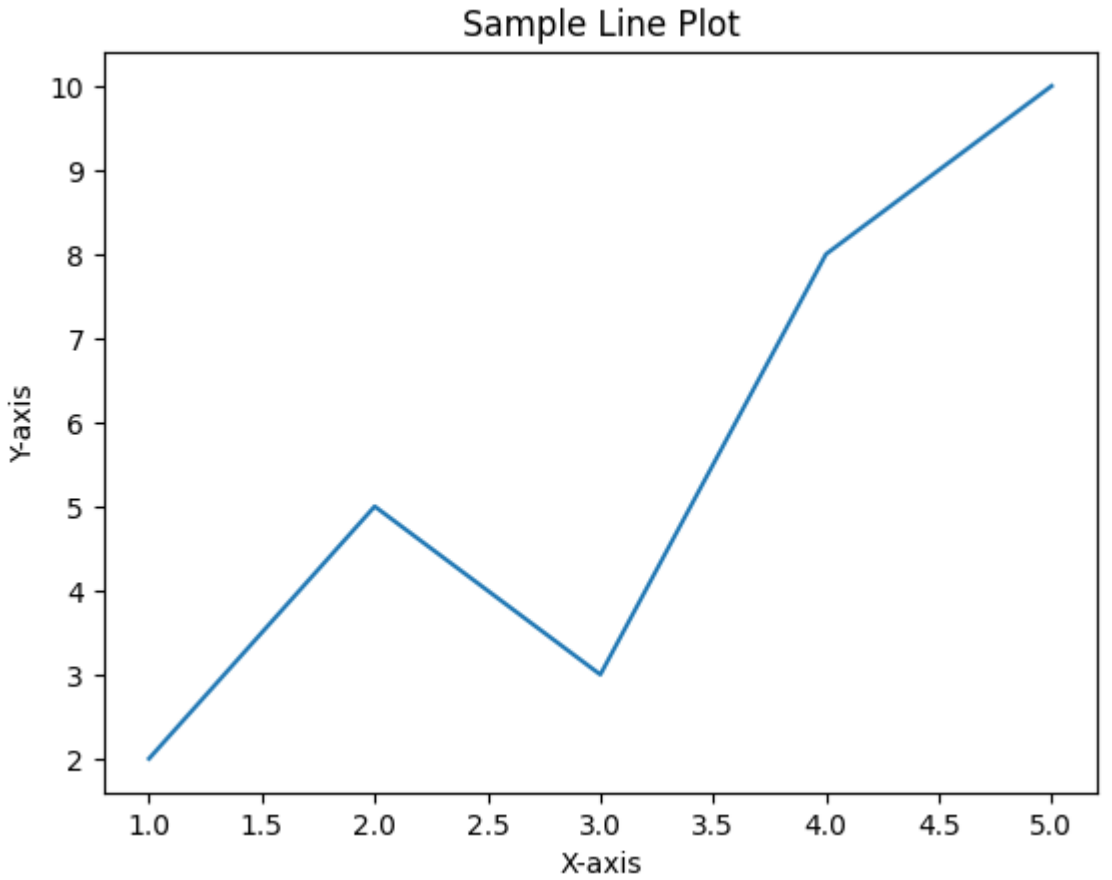
```
import seaborn as sns
import matplotlib.pyplot as plt

# Sample data
x_values = [1, 2, 3, 4, 5]
y_values = [2, 5, 3, 8, 10]

# Create the line plot
sns.lineplot(x=x_values, y=y_values)

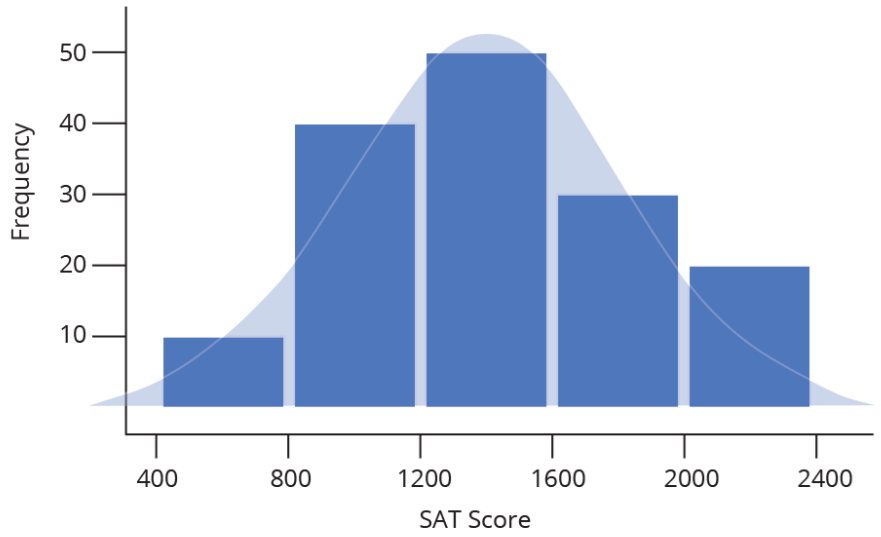
# Add a title and labels
plt.title("Sample Line Plot")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
# Show the plot
plt.show()
```

Output



Histogram

- A histogram is a widely used graph to show the distribution of quantitative (numerical) data.
- It shows the frequency of values in the data, usually in intervals of values.
- Frequency is the amount of times that value appeared in the data.
- Each interval is represented with a bar, placed next to the other intervals on a number line.
- The height of the bar represents the frequency of values in that interval.



Histogram

Example

```
import matplotlib.pyplot as plt
import numpy as np

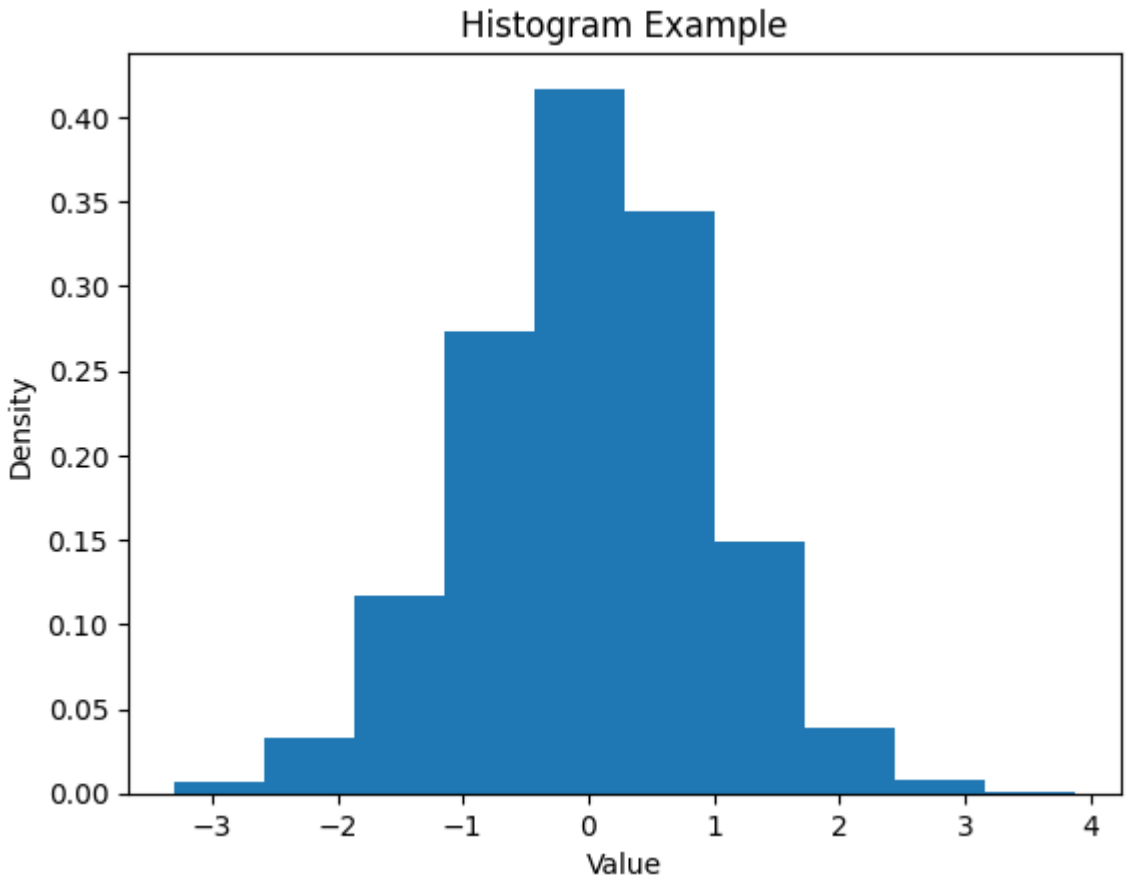
# Create sample data
data = np.random.randn(1000)

# Create a histogram
plt.hist(data, density=True)

# Add a title and labels
plt.title('Histogram Example')
plt.xlabel('Value')
plt.ylabel('Density')

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

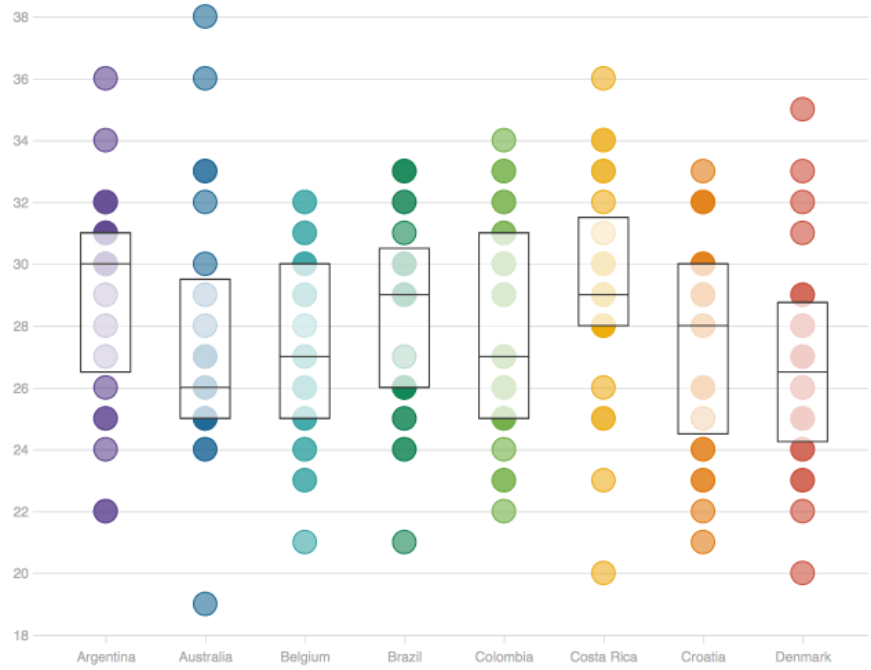
Output



Box Plot

Boxplot

- A box plot is a good way to show many important features of quantitative (numerical) data.
- It shows the median of the data. This is the middle value of the data and one type of an average value.
- It also shows the range and the quartiles of the data. This tells us something about how spread out the data is.

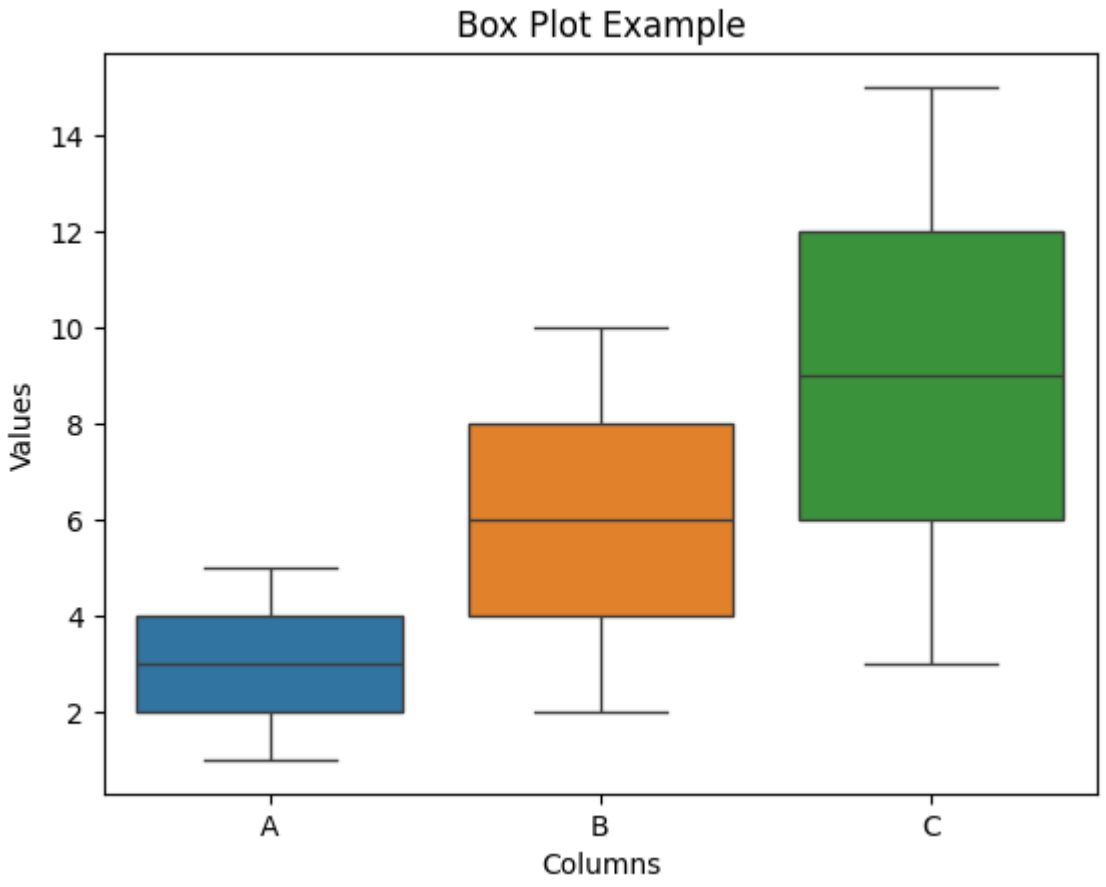


Boxplot

Example

```
import seaborn as sns
import matplotlib.pyplot as plt
# Sample data
data = {
    "A": [1, 2, 3, 4, 5],
    "B": [2, 4, 6, 8, 10],
    "C": [3, 6, 9, 12, 15]
}
# Create a box plot
sns.boxplot(data=data)
# Add a title and labels
plt.title("Box Plot Example")
plt.xlabel("Columns")
plt.ylabel("Values")
# Show the plot
plt.show()
```

Output



Boxplot

- **Box Plot** is the visual representation of the depicting groups of numerical data through their quartiles.
- Boxplot is also used for detect the outlier in data set.
- It captures the summary of the data efficiently with a simple box and whiskers and allows us to compare easily across groups.
- Boxplot summarizes a sample data using 25th, 50th and 75th percentiles. These percentiles are also known as the lower quartile, median and upper quartile.
- A box plot consist of 5 things.
    - Minimum
    - First Quartile or 25%
    - Median (Second Quartile) or 50%
    - Third Quartile or 75%
    - Maximum

Boxplot

Example

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

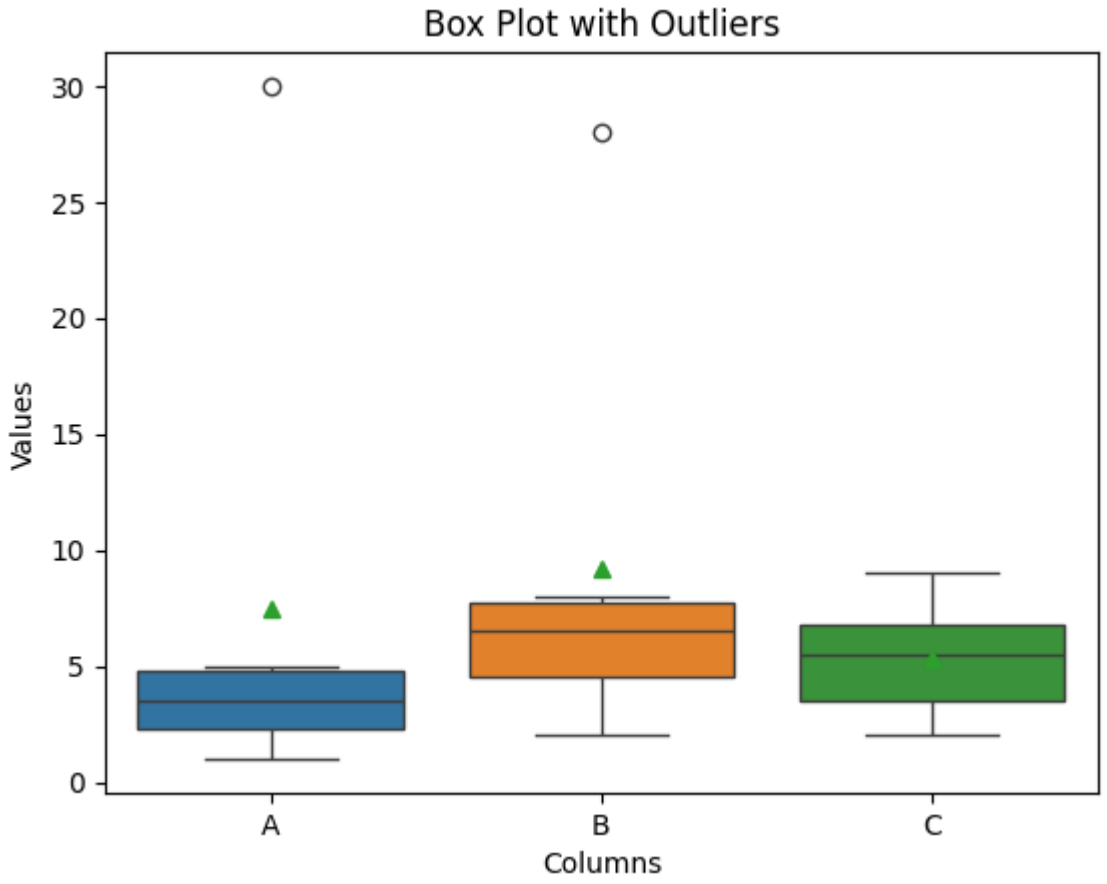
# Sample data with outliers
data = {
    "A": [1, 2, 3, 4, 5, 30],  # 30 is an outlier
    "B": [2, 4, 6, 8, 7, 28],  # 28 is an outlier
    "C": [3, 6, 9, 5, 2, 7]
}
# Convert data to DataFrame for better visualization
df = pd.DataFrame(data)

# Create a box plot with outliers explicitly shown
sns.boxplot(data=df, showmeans=True, whis=1.5)

# Add a title and labels
plt.title("Box Plot with Outliers")
plt.xlabel("Columns")
plt.ylabel("Values")

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

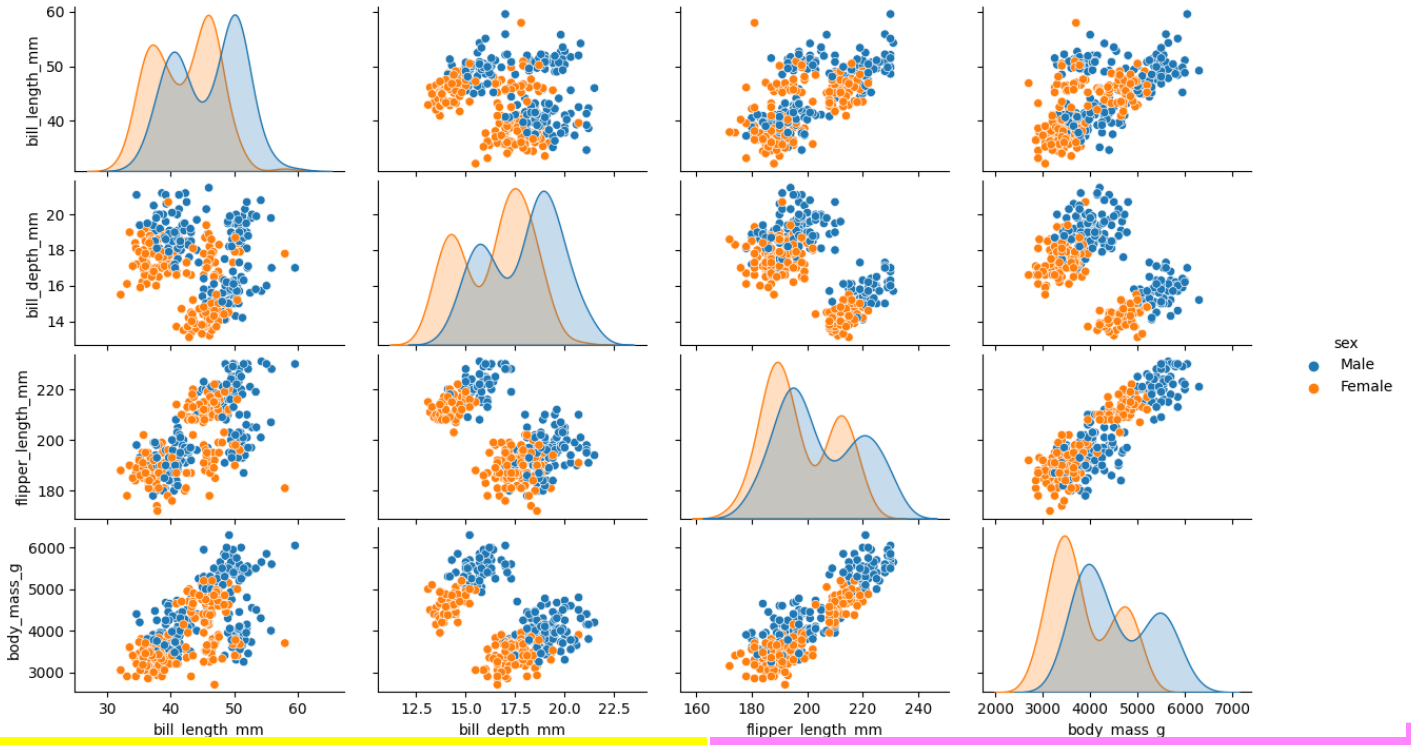
Output



Pair Plot

Pair plot

- A box plot is a good way to show many important features of quantitative (numerical) data.
- It shows the median of the data. This is the middle value of the data and one type of an average value.
- It also shows the range and the quartiles of the data. This tells us something about how spread out the data is.



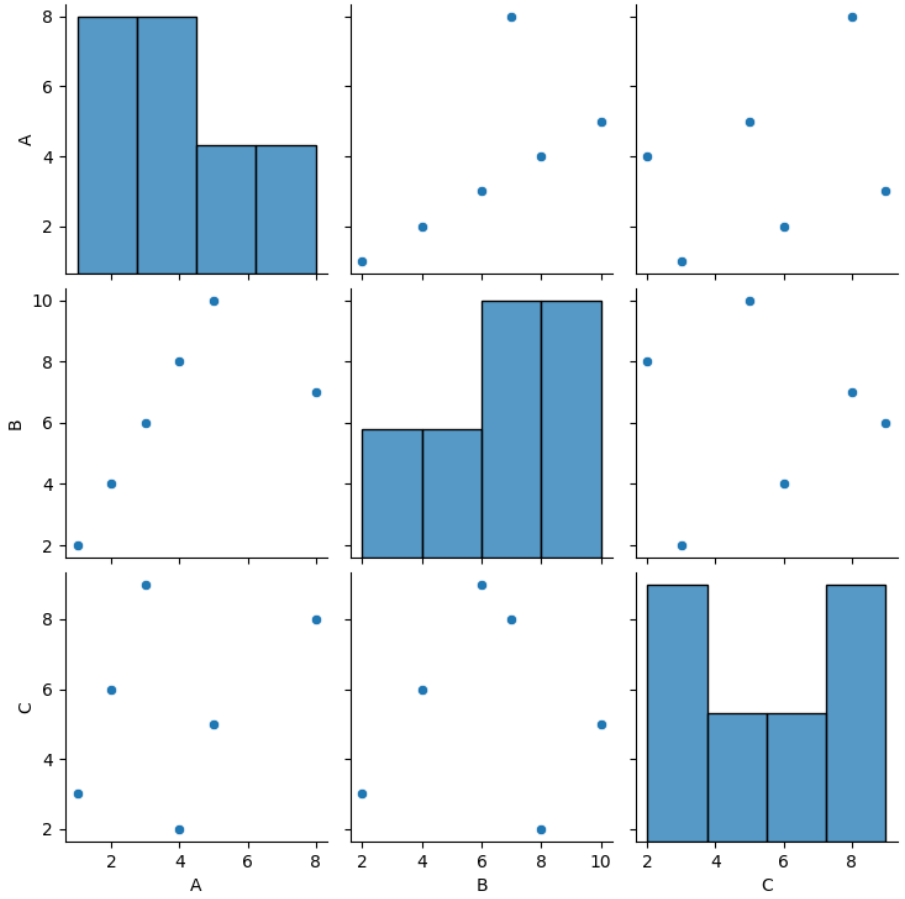
Pair plot

Example

```
import seaborn as sns
import pandas as pd

# Sample data
data = {
    "A": [1, 2, 3, 4, 5, 8],
    "B": [2, 4, 6, 8, 10, 7],
    "C": [3, 6, 9, 2, 5, 8]
}
# Convert data to DataFrame for better visualization
df = pd.DataFrame(data)
# Create a pair plot
sns.pairplot(df)
# Show the plot
plt.show()
```

Output

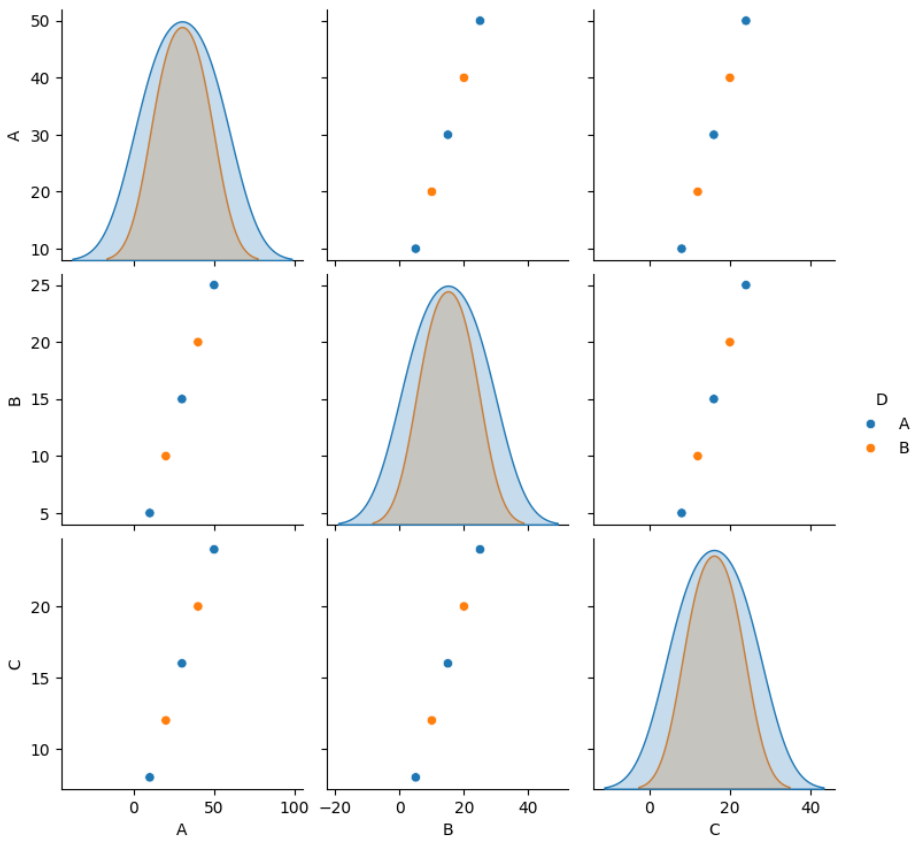


Pair plot

Example

```
import seaborn as sns
import pandas as pd
# Sample data
data = {
    "A": [10, 20, 30, 40, 50],
    "B": [5, 10, 15, 20, 25],
    "C": [8, 12, 16, 20, 24],
    "D": ["A", "B", "A", "B", "A"]
}
# Convert data to DataFrame
df = pd.DataFrame(data)
# Create a pair plot
sns.pairplot(df, hue="D")
# Use 'D' for color differentiation
plt.show()
```

Output



Playing with text

Playing with text

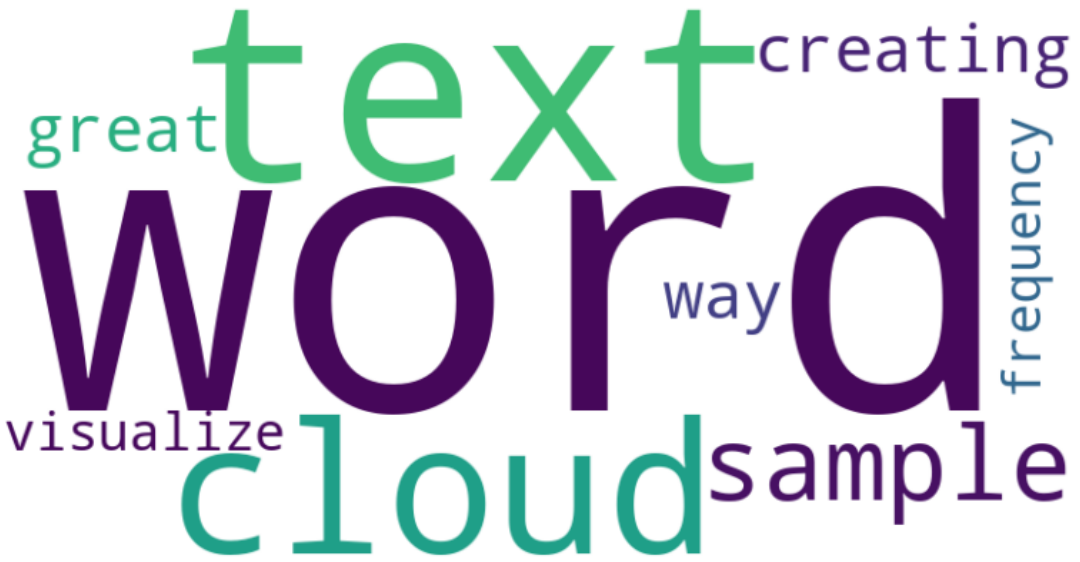
- Seaborn, a Python data visualization library built on top of Matplotlib, is primarily known for its statistical graphics and plots. While its core focus is on numerical data, it can also be used creatively to visualize text data in a variety of ways.
- By combining Seaborn's plotting capabilities with Matplotlib's text functionalities, you can create unique and informative visualizations that convey textual information visually. This can be especially useful for tasks such as:
  - Word Clouds: Visualizing the frequency of words in a text.
  - Text Annotations: Adding textual labels or descriptions to plots.
  - Text-Based Charts: Creating charts or graphs that incorporate text elements.

Word Cloud

Example

```
import seaborn as sns
import matplotlib.pyplot as plt
from wordcloud import WordCloud
# Sample text data
text = "This is a sample text for creating a
word cloud. Word clouds are a great way to
visualize the frequency of words in a text."
# Create a WordCloud object
wordcloud = WordCloud(width=800, height=400,
background_color="white")
# Generate the word cloud
wordcloud.generate(text)
# Display the word cloud
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```

Output



Text Annotation on a Plot

Example

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

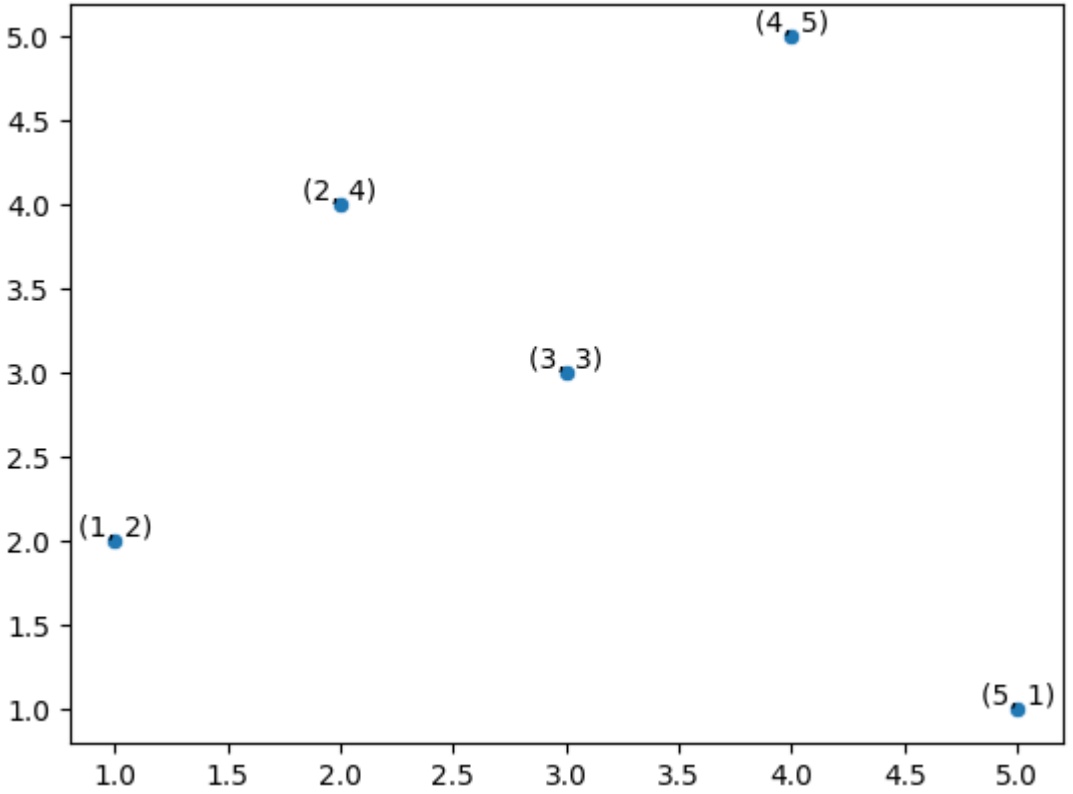
# Sample data
x = [1, 2, 3, 4, 5]
y = [2, 4, 3, 5, 1]

# Create a scatter plot
sns.scatterplot(x=x, y=y)

# Add text annotations
for i in range(len(x)):
    plt.text(x[i], y[i], f"({x[i]}, {y[i]})",
ha="center", va="bottom")

plt.show()
```

Output



Text as a Plot Element

Example

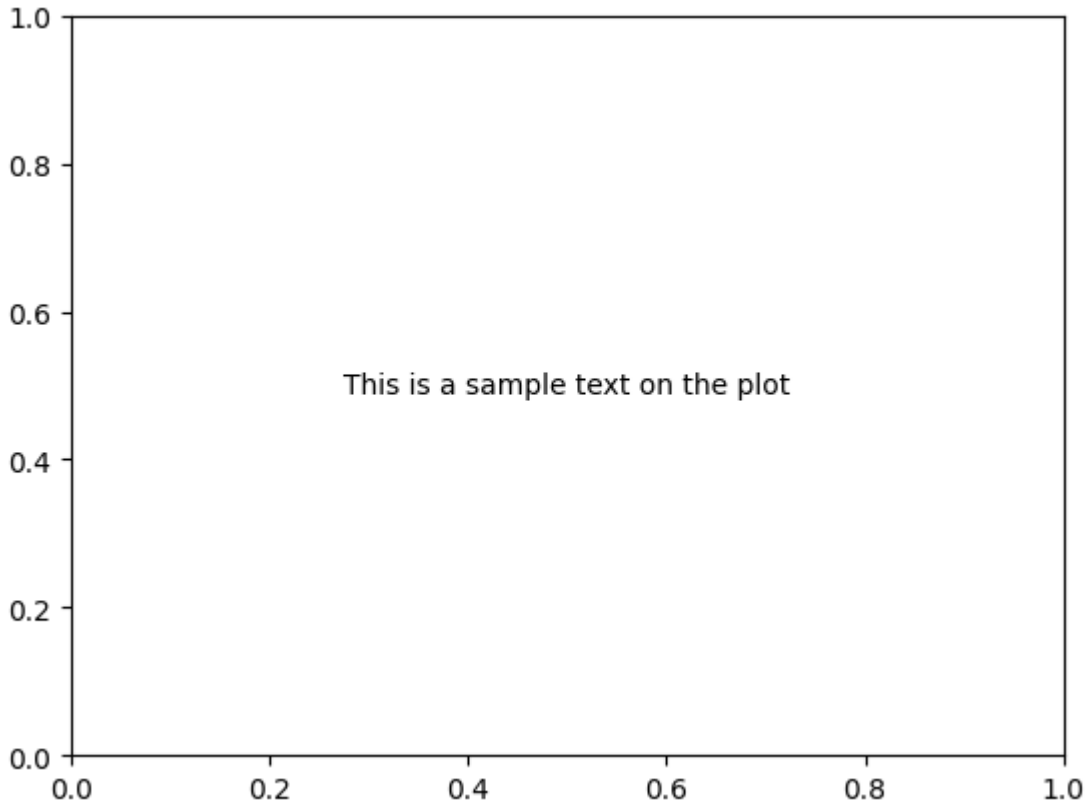
```
import seaborn as sns
import matplotlib.pyplot as plt

# Create a figure and axes
fig, ax = plt.subplots()

# Add text to the plot
ax.text(0.5, 0.5, "This is a sample text on
the plot",
        horizontalalignment='center',
        verticalalignment='center',
        transform=ax.transAxes)

plt.show()
```

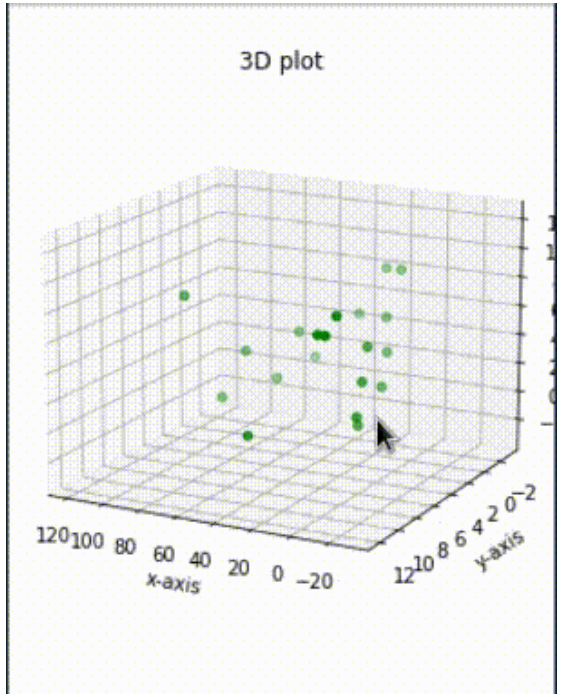
Output



3D Plot

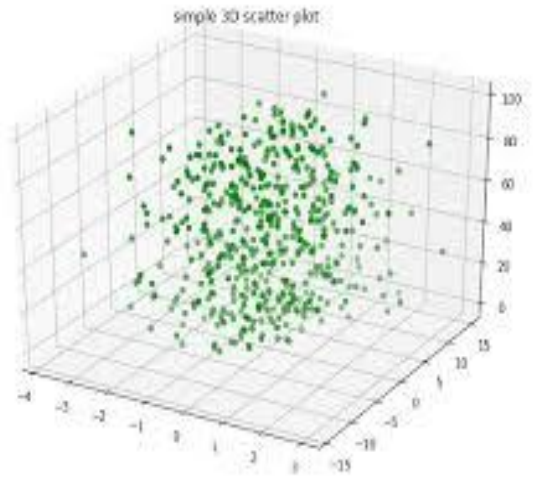
3D Plot

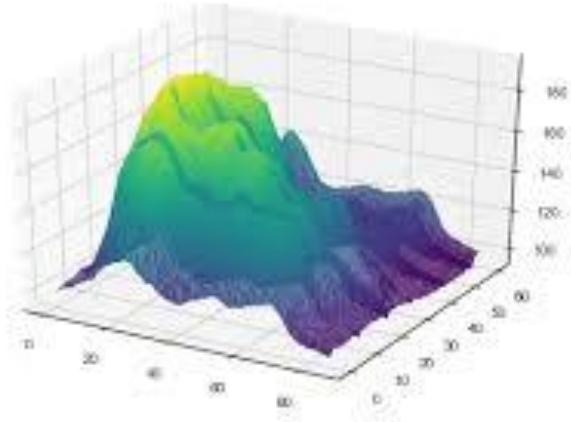
- **3D plots in Seaborn** are a powerful tool for visualizing data in three dimensions. They allow you to represent data points in a three-dimensional space, making it easier to understand relationships and patterns that might not be apparent in two dimensions.



3D Plot

- Seaborn provides a variety of 3D plot types, including:
  - **Scatter plots:** Used to visualize the relationship between three numerical variables.
  - **Line plots:** Used to plot lines in 3D space, often to represent trends or trajectories.
  - **Surface plots:** Used to visualize functions of two variables over a 3D surface.



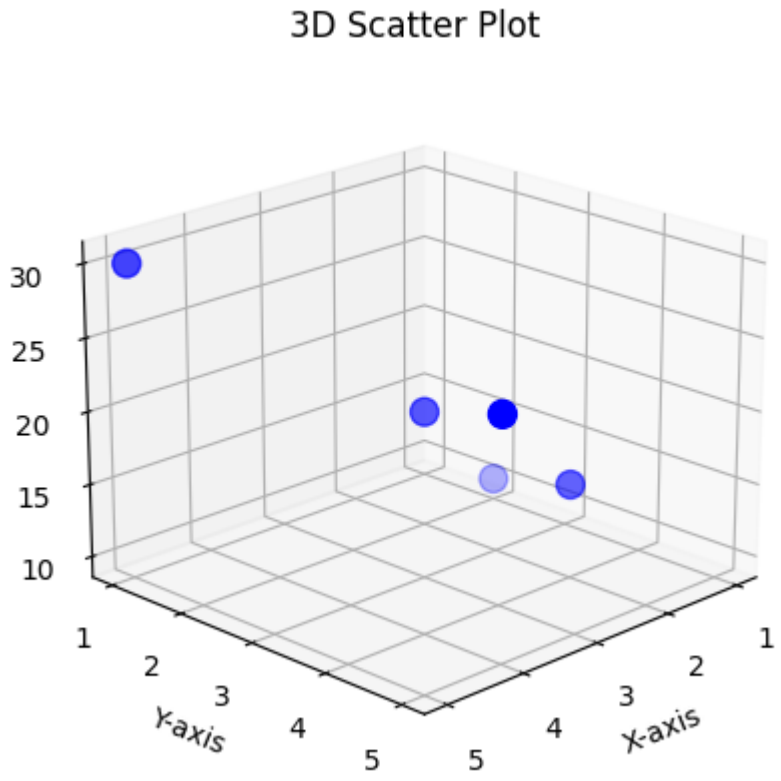


3D Scatter Plot

Example

```
import seaborn as sns
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
# Sample 3D data
x = [1, 2, 3, 4, 5]
y = [2, 4, 3, 5, 1]
z = [10, 15, 20, 25, 30]
# Create a 3D figure and axes
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(x, y, z, c='blue', s=100)
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_zlabel('Z-axis')
plt.title('3D Scatter Plot')
ax.view_init(elev=20, azim=45)
plt.grid(True)
plt.show()
```

Output



3D Line Plot

Example

```
import seaborn as sb
import matplotlib.pyplot as plot
import numpy as np

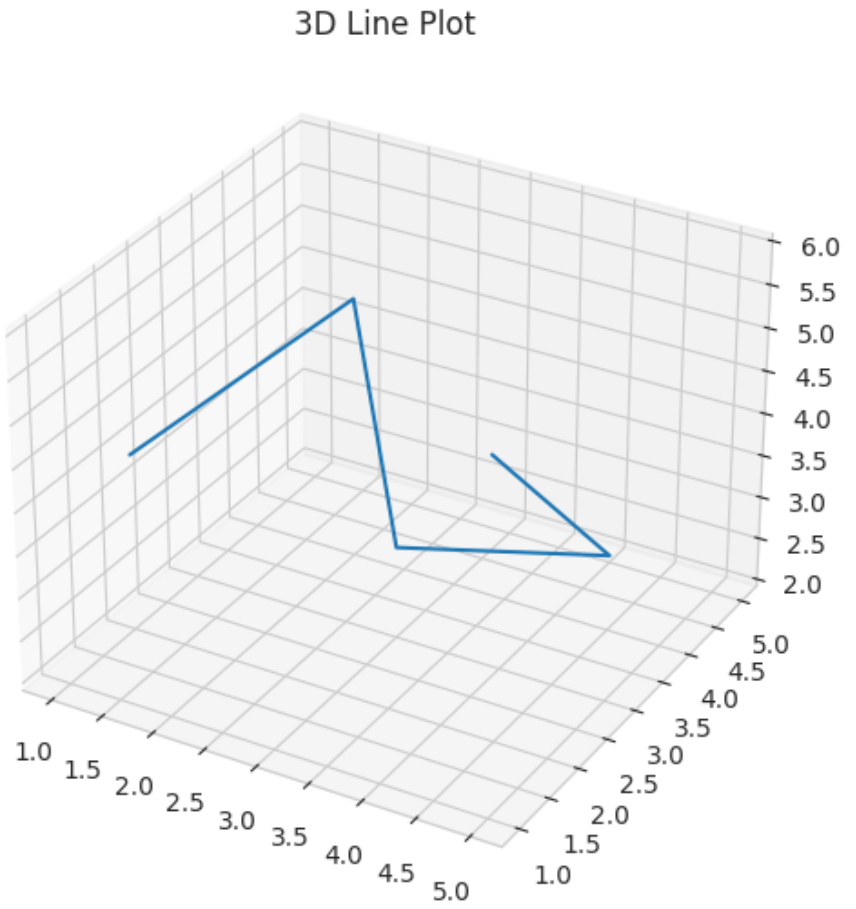
x = [1, 2, 3, 4, 5]
y = [2, 4, 3, 5, 1]
z = [4, 5, 3, 2, 6]

plot.figure(figsize=(6,5))
plot_axis = plot.axes (projection = '3d')
plot_axis.plot3D (x, y, z)
plot.tight_layout ()

plt.title('3D Line Plot')

plot.show ()
```

Output



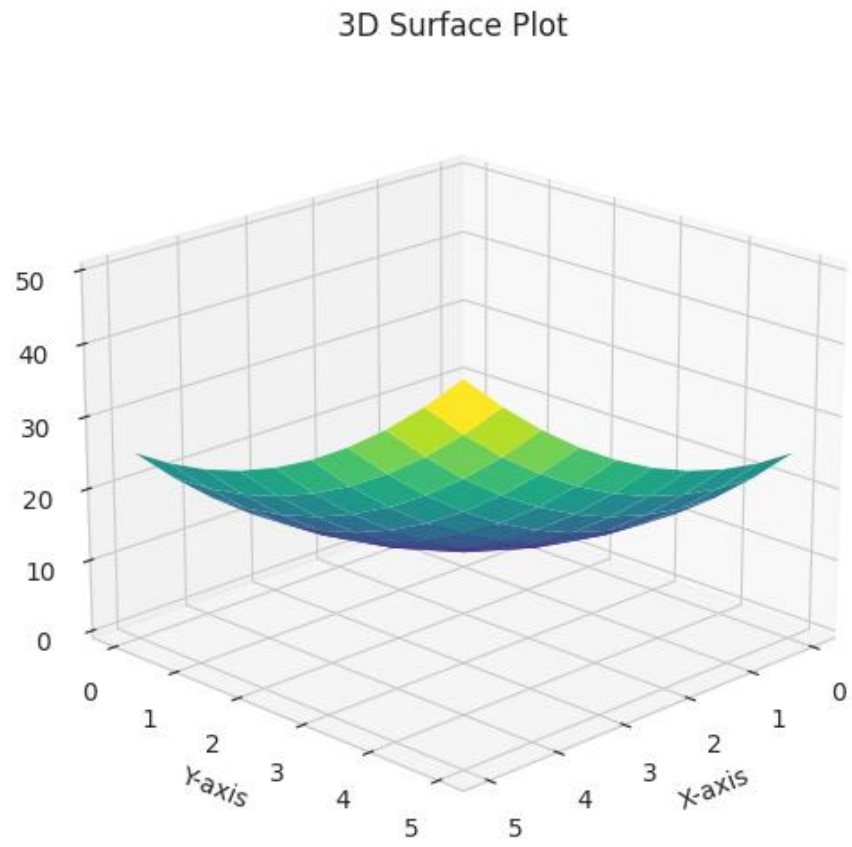
3D Surface Plot

Example

```
import seaborn as sns  # Not directly used
for surface plots
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

# Sample data (ensure x and y are 2D for
surface plot)
x = np.linspace(0, 5, 10)  # Create equally
spaced points from 0 to 5 with 10 elements
y = np.linspace(0, 5, 10)
X, Y = np.meshgrid(x, y)  # Create a 2D
grid from x and y for surface evaluation
```

Output



3D Surface Plot

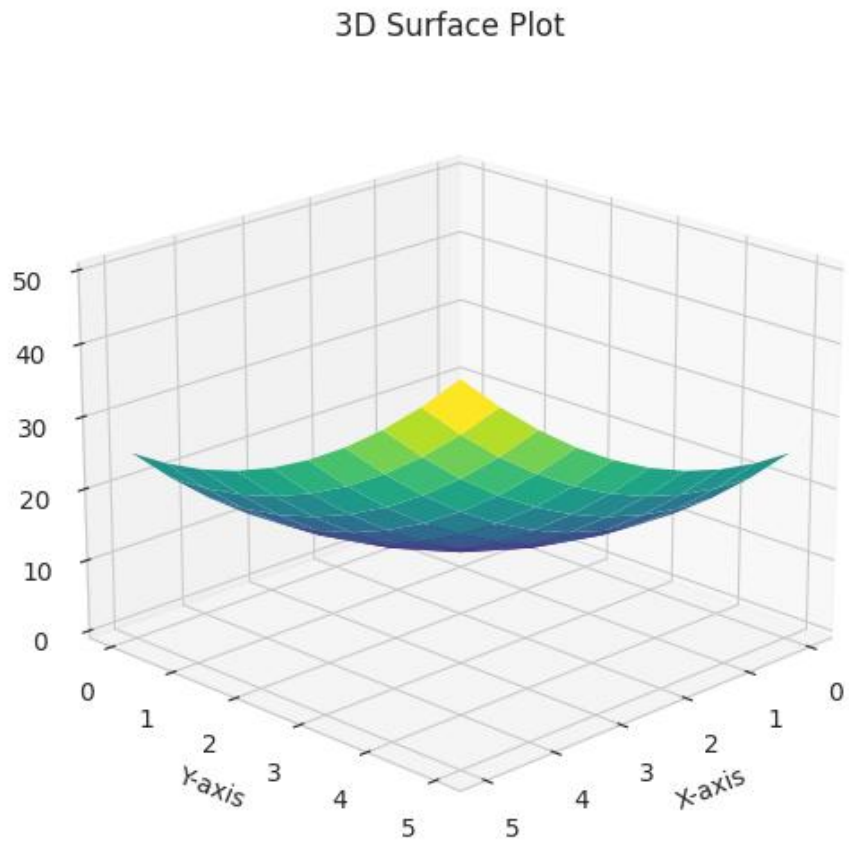
Example

```
# Define a function to evaluate z values at each grid point
def f(x, y):
    return x**2 + y**2  # Replace with your desired function

# Calculate z values based on the function
z = f(X, Y)

# Create a 3D figure and axes
fig = plt.figure(figsize=(8, 6))  # Adjust figure size as needed
ax = fig.add_subplot(111, projection='3d')
```

Output



3D Surface Plot

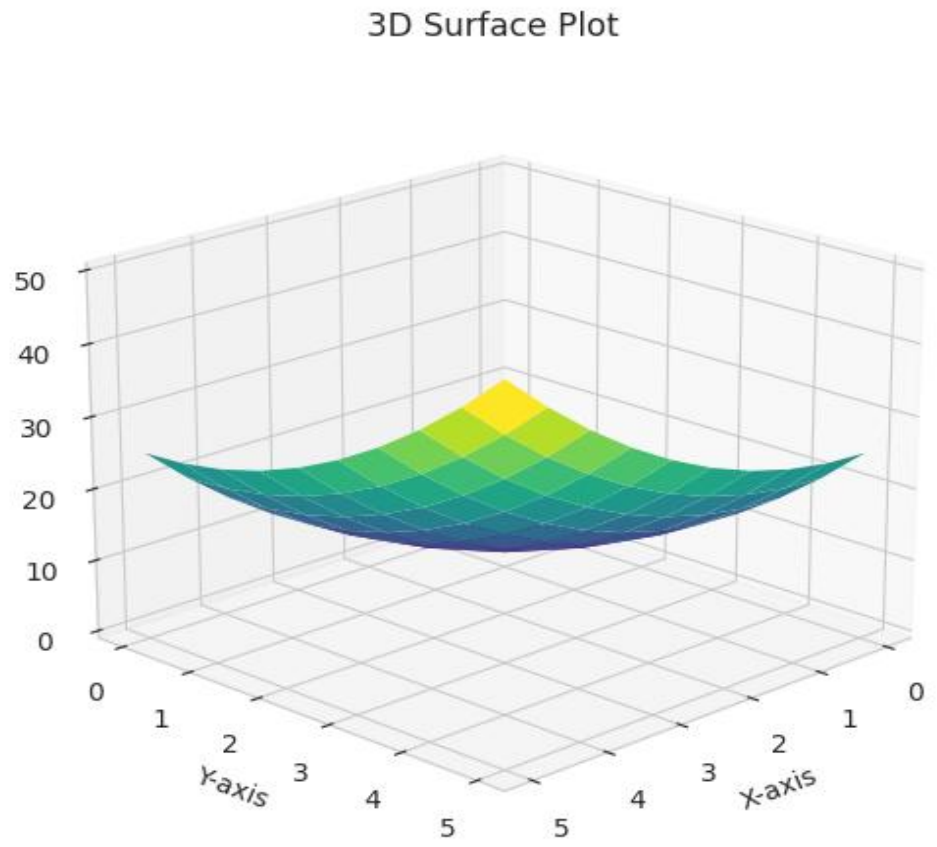
Example

```
# Create a surface plot
surf = ax.plot_surface(X, Y, z,
cmap='viridis', linewidth=0,
antialiased=True)  # Adjust colormap
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
plt.title('3D Surface Plot’)

# Customize viewing angle (optional)
ax.view_init(elev=20, azim=45)  # Adjust
elevation and azimuth angles

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

Output



Thank You