Advanced Plot Types in Matplotlib

1. **Scatter Plots**: A scatter plot is a two-dimensional data visualization that uses individual data points to represent values for two different variables. Each point on the plot corresponds to a single observation in the dataset, with the x and y coordinates indicating the values of the two variables.

   a. Key Features:

   - Markers: Points on the plot representing individual data observations.
   - Axes: The x and y axes represent the variables being compared.
   - Patterns: Patterns or trends in the scatter plot may reveal relationships or correlations between the variables.

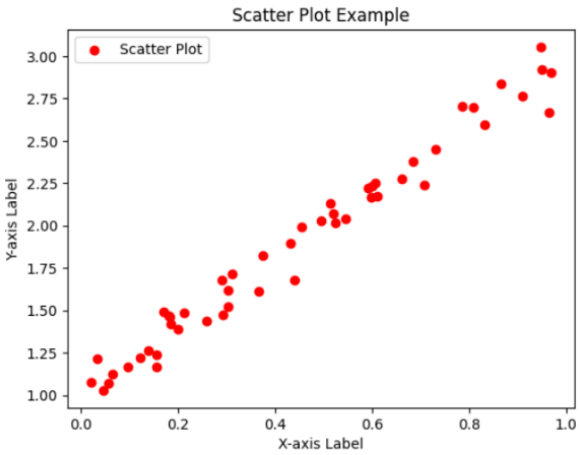
   b. Creating and Customizing Scatter Plots: Scatter plots are effective for visualizing the relationship between two continuous variables.

   Example Code:

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

# Generate random data
np.random.seed(42)
x = np.random.rand(50)
y = 2 * x + 1 + 0.1 * np.random.randn(50)

# Create a scatter plot
plt.scatter(x, y, color='red', marker='o',
label='Scatter Plot')

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

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

Output

This code generates a scatter plot using randomly generated data points (x and y) and demonstrates how to customize the appearance of the plot using Matplotlib functions. The resulting plot will have red circular markers, a title, and labeled axes. The legend will indicate that the plot represents a scatter plot.



Import Libraries:

- matplotlib.pyplot: This imports the **pyplot** module from the Matplotlib library in Python.
- NumPy: It is used for numerical operations and generating random data.

Generate Random Data:

- np.random.seed(42): Sets the random seed to ensure reproducibility of the random data.
- x = np.random.rand(50): Generates an array of 50 random values between 0 and 1.
- y = 2 * x + 1 + 0.1 * np.random.randn(50): Creates a corresponding array 'y' using a linear relationship with some random noise.

Create a Scatter Plot:

```
plt.scatter(x, y, color='red', marker='o',
label='Scatter Plot')
```

- plt.scatter(): Creates a scatter plot with x as the x-axis values, y as the y-axis values.
- color='red': Sets the color of the markers to red.
- marker='o': Specifies that circular markers ('o') should be used.
- label='Scatter Plot': Adds a label to the scatter plot for later use in the legend.

Add Title and Labels:

- plt.title(): Sets the title of the plot.
- plt.xlabel(): Sets the label for the x-axis.
- plt.ylabel(): Sets the label for the y-axis.

Display the plots:

- **plt.legend()**: Displays the legend using the label provided in the scatter plot.
- **plt.show()**: Displays the entire plot.

Let us now understand what histograms are and how to make them.

2. **Histograms**: A histogram is a graphical representation of the distribution of a dataset. It is a way to visualize the underlying frequency distribution of continuous data.

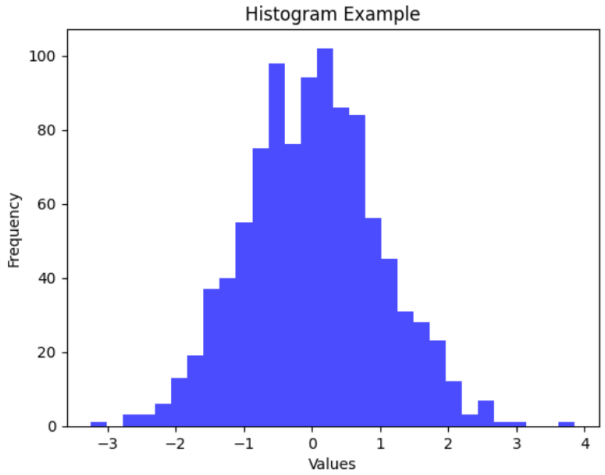
   a. **Key Features:**
   - Bins: The range of values is divided into intervals, or "bins," and the number of data points falling into each bin is represented by the height of a bar.
   - Frequency: The height of each bar indicates the frequency of data points within a specific bin.

   b. **Creating Histograms**

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

# Generate random data
np.random.seed(42)
data = np.random.randn(1000)

# Create a histogram
plt.hist(data, bins=30, color='blue', alpha=0.7)
plt.title('Histogram Example')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.show()
```

Output

This code generates a histogram to visualize the distribution of random data. The histogram represents the frequency (or count) of values in different bins, providing insight into the shape of the data distribution. The plot has a title, and the x-axis and y-axis are labelled for clarity.



Import Libraries:

- matplotlib.pyplot: This imports the **pyplot** module from the Matplotlib library in Python.
- NumPy: It is used for numerical operations and generating random data.

Generate Random Data:

- np.random.seed(42): This sets the random seed for reproducibility. It ensures that if you run the code again, you will get the same random numbers.
- np.random.randn(1000): This generates an array of 1000 random numbers from a standard normal distribution (mean=0, standard deviation=1).

Create a Histogram:

```
plt.hist(data, bins=30, color='blue', alpha=0.7)
```

- **plt.hist**: This function is used to create a histogram.
- **data**: The array of random data to be plotted.
- **bins=30**: This specifies the number of bins (intervals) in the histogram. In this case, there are 30 bins.
- **color='blue'**: This sets the color of the bars in the histogram to blue.
- **alpha=0.7**: This controls the transparency of the bars, where 0 is fully transparent and 1 is fully opaque. Here, it's set to 0.7.

Add Title and Labels:

```
plt.title('Histogram Example')
plt.xlabel('Values')
plt.ylabel('Frequency')
```

- **plt.title**: Adds a title to the plot.
- **plt.xlabel**: Adds a label to the x-axis.
- **plt.ylabel**: Adds a label to the y-axis.

Display the Plot:

plt.show() function is used to display the histogram.

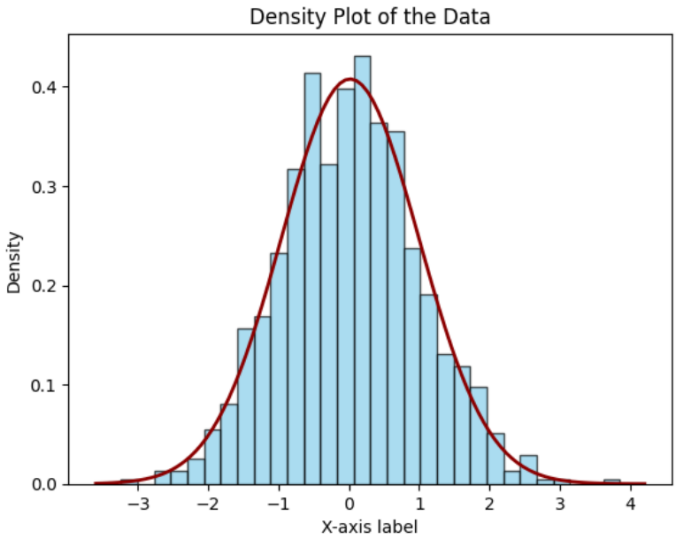
Now let's learn to create density plots.

3. **Density Plots:** A density plot is a smoothed, continuous representation of a dataset's distribution. It is often used to estimate the probability density function of the underlying population.
   a. **Key Features:**
      Kernel Density Estimation (KDE): The density plot is created by smoothing the histogram using a kernel function, providing a continuous estimate of the distribution.
   b. **Creating density plots to show the probability density of a dataset:**

```
plt.hist(data, bins=30, color='blue', alpha=0.7)
```

Output

This code creates a density plot using Matplotlib with a histogram and a kernel density estimate (KDE) overlaid on top of it.



The Python code can be explained as follows:

Generate Example Data:

The code generates a random dataset of 1000 points drawn from a standard normal distribution (mean = 0, standard deviation = 1).

Histogram:

```
plt.hist(data, bins=30, density=True, alpha=0.7,
color='skyblue', edgecolor='black')
```

This line generates a histogram using **plt.hist()**. The **density=True** argument normalizes the histogram to represent a probability density. The **bins** parameter controls the number of bins in the histogram. The **alpha**, **color**, and **edgecolor** parameters are used for visual **customization**.

Kernel Density Estimate (KDE):

```
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)
kde = (1 / (data.std() * np.sqrt(2 * np.pi))) *
np.exp(-0.5 * ((x - data.mean()) / data.std())**2)
plt.plot(x, kde, linewidth=2, color='darkred')
```

This part manually adds a kernel density estimate (KDE) to the plot. It calculates a Gaussian KDE using a set of points (**x**) and then plots the KDE on top of the histogram.

Labels and Title:

```
plt.title('Density Plot of the Data')
plt.xlabel('X-axis label')
plt.ylabel('Density')
```

These lines add a title and axis labels to the plot for better understanding.

The combination of the histogram and the KDE provides a visualization of the probability density of the dataset. The histogram represents the distribution of the data, while the KDE smooths out the distribution to give a more continuous estimate of the underlying probability density function.

4. **Box Plots:** A box plot (also known as a whisker plot) is a graphical representation of the distribution of a dataset. It provides a summary of the central tendency, spread, and shape of the distribution. Box plots are particularly useful for comparing distributions across different categories or groups.

   a. **Key Features**:

   - Box: It depicts the **interquartile range (IQR)**, the span from the first quartile (Q1) to the third quartile (Q3), showcasing the middle 50% of the data.
   - Whiskers: They stretch from the box to the minimum and maximum values within a specified range, typically 1.5 times the interquartile range. Data beyond the whiskers are outliers, often shown individually.
   - Median Line: Inside the box, a line denotes the dataset's median.
   - Outliers: Data points beyond the whiskers are outliers and are usually plotted separately.

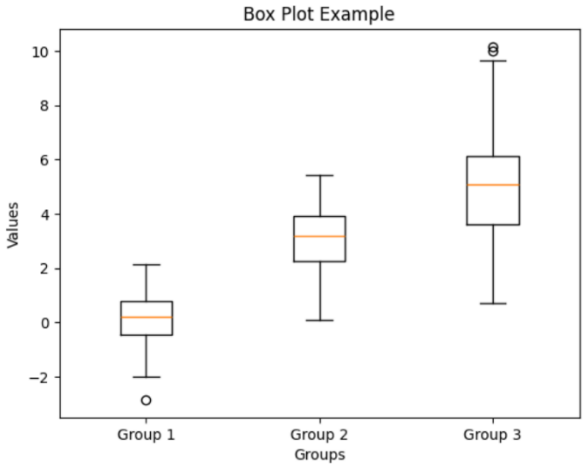
   b. **Constructing Box Plots:** Box plots are excellent for visualizing the distribution of a dataset and identifying outliers. Here's an example:

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

# Generate some example data
np.random.seed(42)
data = np.random.randn(1000)

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

# Add a kernel density estimate (KDE) using Gaussian
kernel
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)
kde = (1 / (data.std() * np.sqrt(2 * np.pi))) * np.exp(-
0.5 * ((x - data.mean()) / data.std())**2)
plt.plot(x, kde, linewidth=2, color='darkred')

# Add labels and title
```

```
plt.title('Density Plot of the Data')
plt.xlabel('X-axis label')
plt.ylabel('Density')

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

Output:

This code generates a box plot with three categories of random data, each with a different standard deviation. The resulting plot provides a visual representation of the distribution of each dataset, including key statistics such as the median, quartiles, and potential outliers. The **patch_artist=True** option adds colours to the boxes for better visualization.



Generating Random Data:

```
np.random.seed(42)
data = [np.random.normal(0, std, 100) for std in
range(1, 4)]
```

These lines are setting random seed for reproducibility and generating three sets of random data with increasing standard deviations (1, 2, and 3).

Creating box plot

```
plt.boxplot(data, vert=True, patch_artist=True)
```

boxplot function is used to create a box plot. The **data** variable contains the random datasets, **vert=True** specifies a vertical orientation, and **patch_artist=True** fills the boxes with colors.