Lecture 5

Learning outcomes:

- *Data Visualization for Engineering:*
  - *Using Matplotlib and Seaborn for plotting*

Data Visualization

- Data visualization is an easier way of presenting the data, however complex it is, to analyze trends and relationships amongst variables with the help of pictorial representation.

- The following are the advantages of Data Visualization:

- Easy Interpretation and Understanding
- Highlights good and bad performing areas
- Explores the relationship between data points
- Identifies data patterns even for larger data points

- While building visualization, it is always a good practice to keep some below mentioned points in mind.

- Ensure appropriate usage of shapes, colors, and size while building visualization

- Plots/graphs using a co-ordinate system are more pronounced

- Knowledge of suitable plot with respect to the data types brings more clarity to the information

- Usage of labels, titles, legends and pointers passes seamless information the wider audience

Nature of Visualization

- Depending on the number of variables used for plotting the visualization and the type of variables, there could be different types of charts which we could use to understand the relationship. Based on the count of variables, we could have:

- Univariate plot(involves only one variable)
- Bivariate plot(more than one variable in required)

- ✓A Univariate plot could be for a continuous variable to understand the spread and distribution of the variable while for a discrete variable it could tell us the count.
- ✓Similarly, a Bivariate plot for continuous variable could display essential statistic like correlation, for a continuous versus discrete variable could lead us to very important conclusions like understanding data distribution across different levels of a categorical variable. A bivariate plot between two discrete variables could also be developed.

Python Libraries

- There are a lot of python libraries which could be used to build visualization like matplotlib, vispy, bokeh, seaborn, pygal, folium, plotly, cufflinks, and networkx.

- Of the many, matplotlib and seaborn seems to be very widely used for basic to intermediate level of visualizations.

Using Matplotlib for plotting

- It is an amazing visualization library in Python for 2D plots of arrays, It is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack.
- Some of the benefits and features of matplotlib.
  - It’s fast, efficient as it is based on numpy and also easier to build.
  - Has undergone a lot of improvements from the open source community since inception and hence a better library having advanced features as well.
  - Well maintained visualization output with high quality graphics draws a lot of users to it.
  - Basic as well as advanced charts could be very easily built.
  - From the users/developers point of view, since it has a large community support, resolving issues and debugging becomes much easier.

When to Use Matplotlib:

- When you need detailed control over plot elements.
- For creating complex or non-standard plots.
- When working on projects requiring extensive customization.

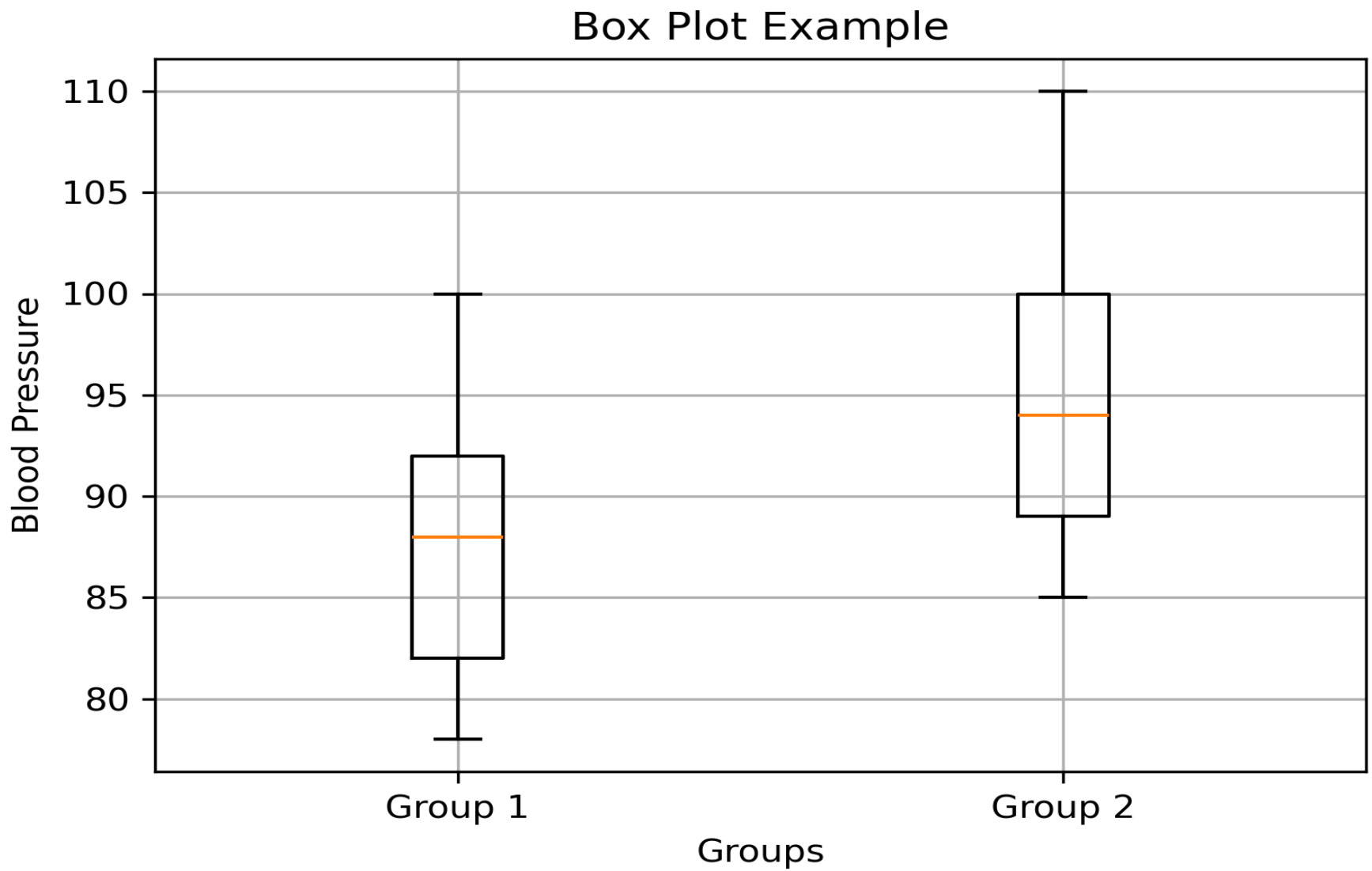
Basic plotting in Matplotlib

Box plot

- A boxplot, also known as a box and whisker plot, the box and the whisker are clearly displayed in the below image.
- It is a very good visual representation when it comes to measuring the data distribution.
- Clearly plots the median values, outliers and the quartiles. Understanding data distribution is another important factor which leads to better model building. If data has outliers, box plot is a recommended way to identify them and take necessary actions.

```
import matplotlib.pyplot as plt

# Sample data (Manually provided)
group1 = [78, 80, 82, 85, 88, 90, 92, 95, 100]  # Blood Pressure for Group 1
group2 = [85, 87, 89, 92, 94, 97, 100, 105, 110]  # Blood Pressure for Group 2
# Combine the groups in a list
data = [group1, group2]
# Create the box plot
plt.figure(figsize=(6, 4))
plt.boxplot(data, labels=["Group 1", "Group 2"])
# Labels and title
plt.xlabel("Groups")
plt.ylabel("Blood Pressure")
plt.title("Box Plot Example")
# Show the plot
plt.grid(True)
plt.show()
```


Box Plot Example
110
105
100
95
90
85
80
Blood Pressure
Group 1
Group 2
Groups

Scatter Plot

- Scatter plots or scatter graphs is a bivariate plot having greater resemblance to line graphs in the way they are built. A line graph uses a line on an X-Y axis to plot a continuous function, while a scatter plot relies on dots to represent individual pieces of data. These plots are very useful to see if two variables are correlated. Scatter plot could be 2 dimensional or 3 dimensional.

- Advantages of a scatter plot:

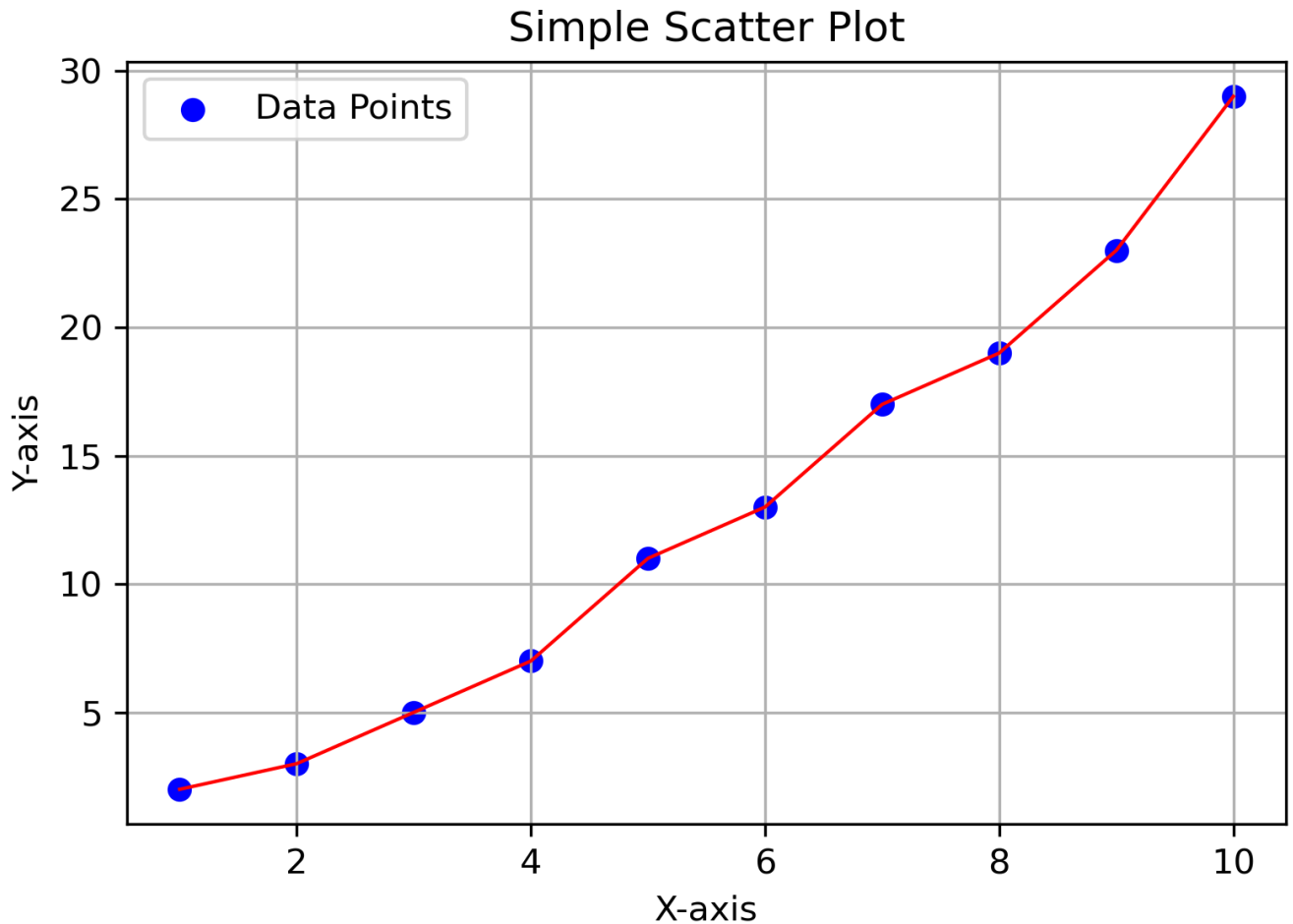
➢ Displays correlation between variables

➢ Suitable for large data sets

➢ Easier to find data clusters

➢ Better representation of each data point

```
import matplotlib.pyplot as plt
# Sample data (Manually provided)
x_values = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  # X-axis values
y_values = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]  # Y-axis values
# Create scatter plot
plt.figure(figsize=(6, 4))
plt.scatter(x_values, y_values, color='blue', marker='o', label="Data Points")
# Add a line connecting the points
plt.plot(x_values, y_values, color='red', linestyle='-', linewidth=1)
# Labels and title
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Simple Scatter Plot")
# Show legend
plt.legend()
# Show grid
plt.grid(True)
# Display the plot
plt.show()
```


Simple Scatter Plot
Data Points
30
25
20
15
10
5
Y-axis
2
4
6
8
10
X-axis

Histogram

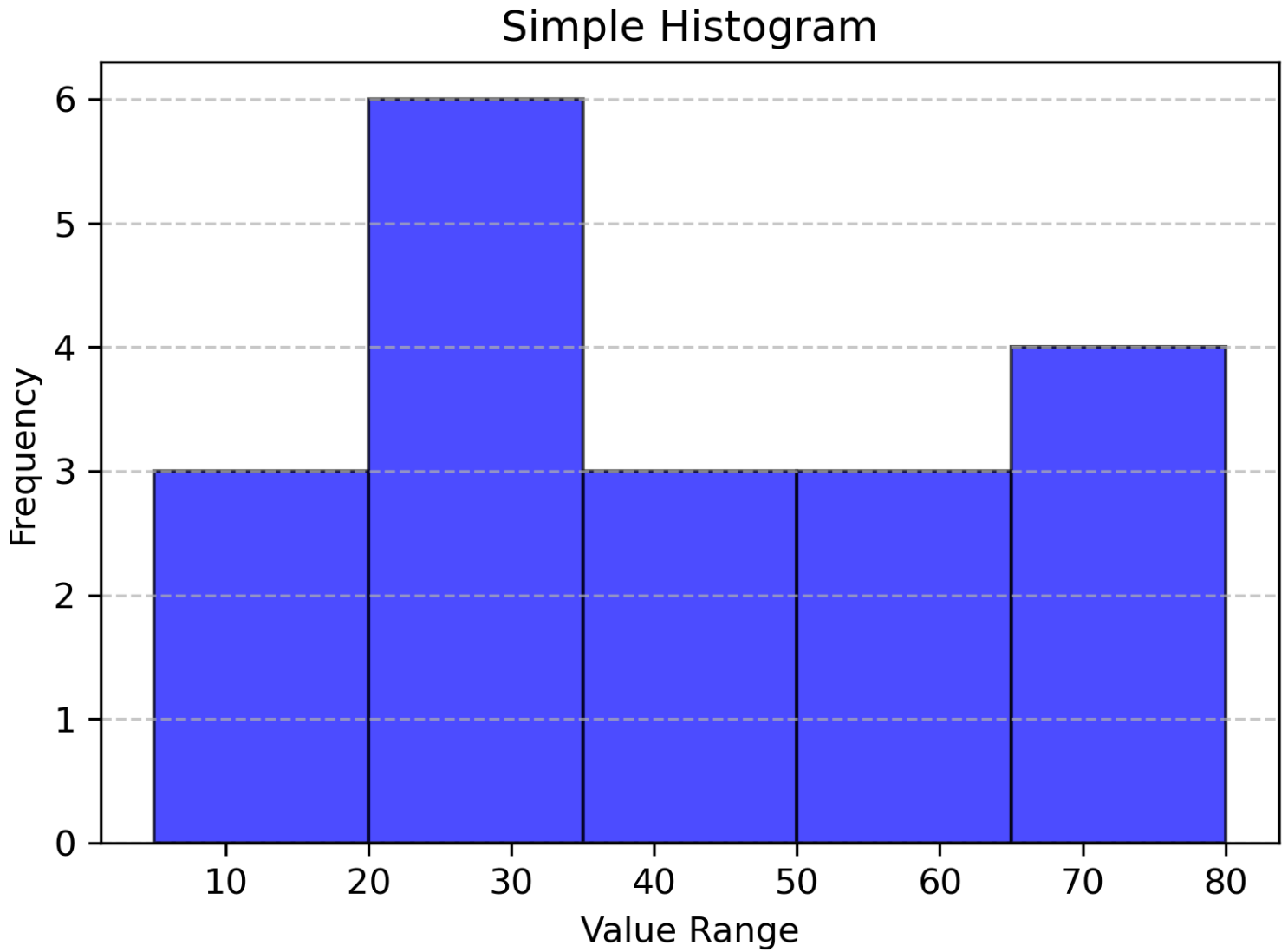
- Histograms display counts of data and are hence similar to a bar chart. A histogram plot can also tell us how close a data distribution is to a normal curve. While working out statistical method, it is very important that we have a data which is normally or close to a normal distribution. However, histograms are univariate in nature and bar charts bivariate.

- A bar graph charts actual counts against categories e.g. height of the bar indicates the number of items in that category whereas a histogram displays the same categorical variables in bins.

- Bins are integral part while building a histogram they control the data points which are within a range. As a widely accepted choice we usually limit bin to a size of 5-20, however this is totally governed by the data points which is present.

```
import matplotlib.pyplot as plt

# Sample data (Manually provided)
data = [5, 10, 15, 20, 20, 25, 30, 30, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80]
# Create histogram
plt.figure(figsize=(6, 4))
plt.hist(data, bins=5, color='blue', edgecolor='black', alpha=0.7)
# Labels and title
plt.xlabel("Value Range")
plt.ylabel("Frequency")
plt.title("Simple Histogram")
# Show grid
plt.grid(axis='y', linestyle='--', alpha=0.7)
# Display the plot
plt.show()
```


Simple Histogram
Frequency
0
1
2
3
4
5
6
10
20
30
40
50
60
70
80
Value Range

Bar plot

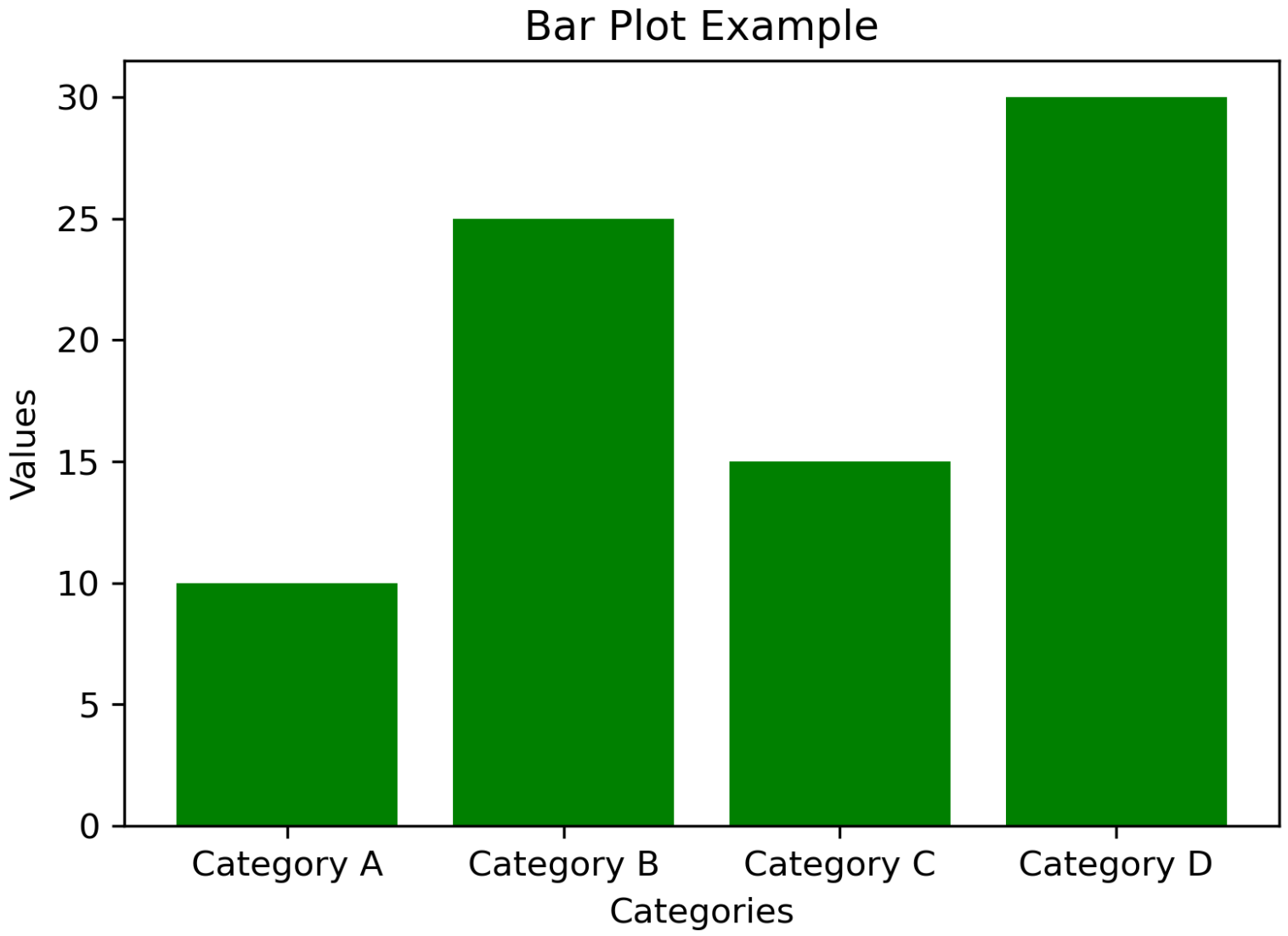
- A bar plot (or bar chart) is a type of data visualization that represents categorical or numerical data using rectangular bars. The length or height of each bar corresponds to the value it represents.
- Types of Bar Plots:

1. Vertical Bar Plot – Bars are aligned vertically (most common).
2. Horizontal Bar Plot – Bars are aligned horizontally.
3. Grouped Bar Plot – Multiple bars for different categories side by side.
4. Stacked Bar Plot – Bars stacked on top of each other to show proportions.

- When to Use a Bar Plot:

✓To compare different categories.
✓To display numerical data grouped by a category.
✓To show trends over time (if categories represent time periods).

```
import matplotlib.pyplot as plt
import numpy as np
# Sample data
categories = ['Category A', 'Category B', 'Category C', 'Category D']
values = [10, 25, 15, 30]  # Corresponding values for each category
# Create bar plot
plt.figure(figsize=(6, 4))
plt.bar(categories, values, color='green')
# Labels and title
plt.xlabel("Categories")
plt.ylabel("Values")
plt.title("Bar Plot Example")
# Display the plot
plt.show()
```


Bar Plot Example
Values
30
25
20
15
10
5
0
Category A
Category B
Category C
Category D
Categories

Pie Chart

- Pie chart is a univariate analysis and is typically used to show percentage or proportional data. The percentage distribution of each class in a variable is provided next to the corresponding slice of the pie. The python libraries which could be used to build a pie chart are matplotlib and seaborn.

- Below are the advantages of a pie chart

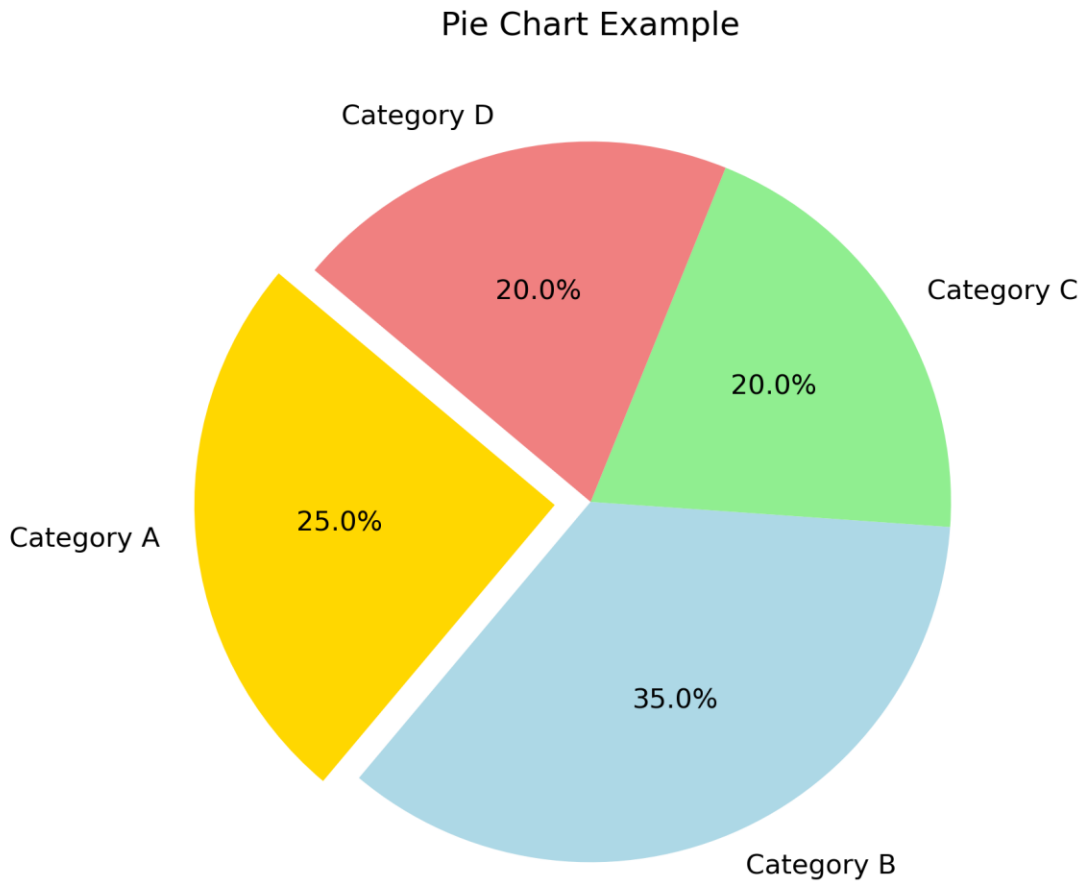
➢ Easier visual summarization of large data points

➢ Effect and size of different classes can be easily understood

➢ Percentage points are used to represent the classes in the data points

```
import matplotlib.pyplot as plt

# Manually define categories and values
labels = ['Category A', 'Category B', 'Category C', 'Category D']
sizes = [25, 35, 20, 20]  # Corresponding values (percentages)
colors = ['gold', 'lightblue', 'lightgreen', 'lightcoral']  # Colors for each slice
explode = (0.1, 0, 0, 0)  # "Explode" the first slice (Category A)
# Create the pie chart
plt.figure(figsize=(6, 6))
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=140,
explode=explode)
# Title
plt.title("Pie Chart Example")
# Display the plot
plt.show()
```


Pie Chart Example
Category D
20.0%
Category C
20.0%
Category A
25.0%
35.0%
Category B

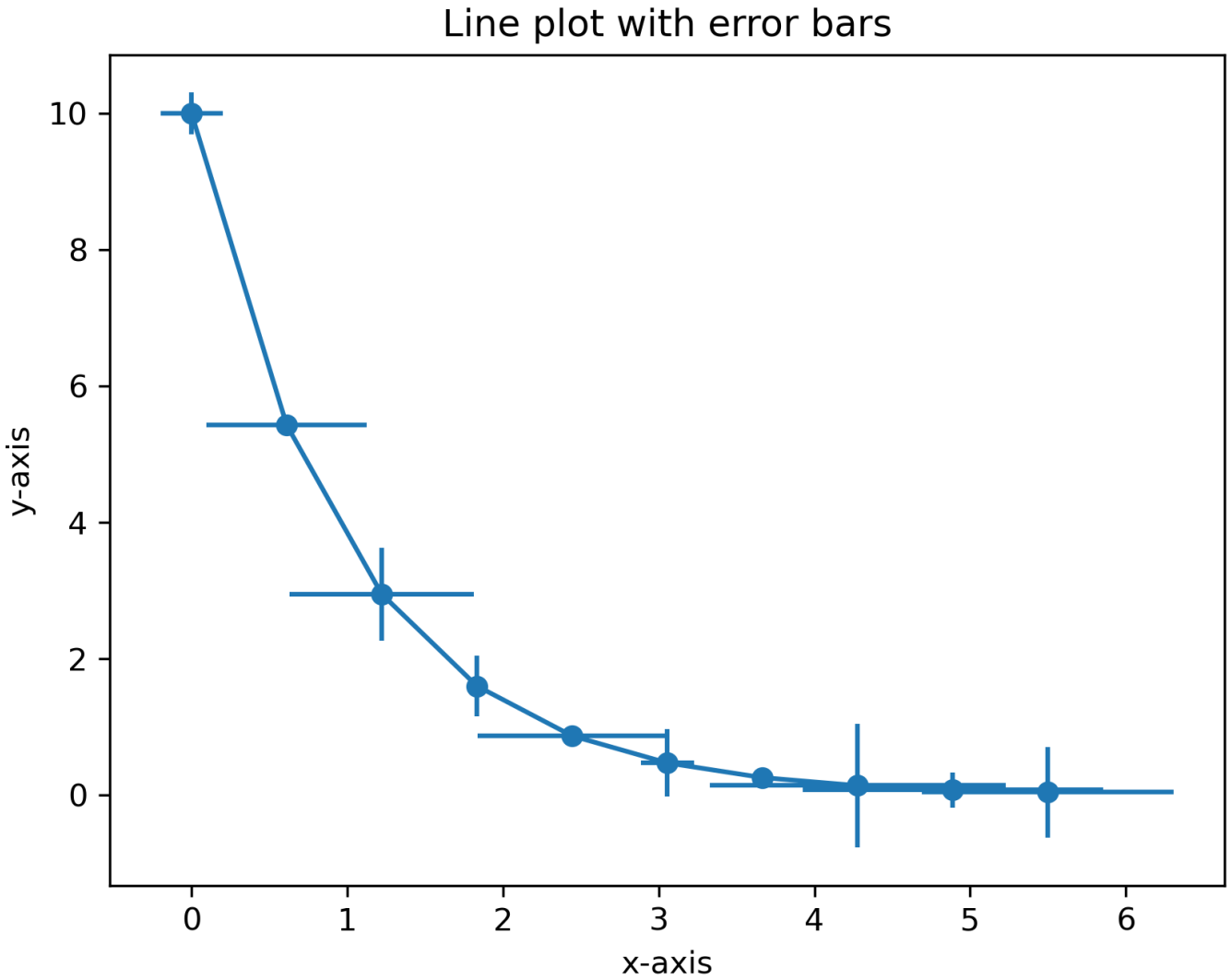
Error Bars

- Error bars could be defined as a line through a point on a graph, parallel to one of the axes, which represents the uncertainty or error of the corresponding coordinate of the point. These types of plots are very handy to understand and analyze the deviations from the target. Once errors are identified, it could easily lead to deeper analysis of the factors causing them.

- Deviation of data points from the threshold could be easily captured
- Easily captures deviations from a larger set of data points
- It defines the underlying data

```
# Import required module
import matplotlib.pyplot as plt
import numpy as np
# Assign axes
x = np.linspace(0,5.5,10)
y = 10*np.exp(-x)
# Assign errors regarding each axis
xerr = np.random.random_sample(10)
yerr = np.random.random_sample(10)

# Adjust plot
fig, ax = plt.subplots()
ax.errorbar(x, y, xerr=xerr, yerr=yerr, fmt='-o')
# Assign labels
ax.set_xlabel('x-axis'), ax.set_ylabel('y-axis')
ax.set_title('Line plot with error bars')
# Illustrate error bars
plt.show()
```


Line plot with error bars
y-axis
x-axis
0
2
4
6
8
10
0
1
2
3
4
5
6

Using Seaborn for plotting

- Seaborn is a statistical data visualization library built on top of Matplotlib. It is designed to make it easier to create aesthetically pleasing and informative statistical graphics.

- Key Features of Seaborn:

  - Ease of Use: Simplifies the creation of complex plots with fewer lines of code compared to Matplotlib.
  - Built-in Themes: Provides attractive themes and color palettes that enhance the appearance of plots.
  - Statistical Plots: Includes functions for visualizing statistical relationships and distributions, making it ideal for exploratory data analysis.

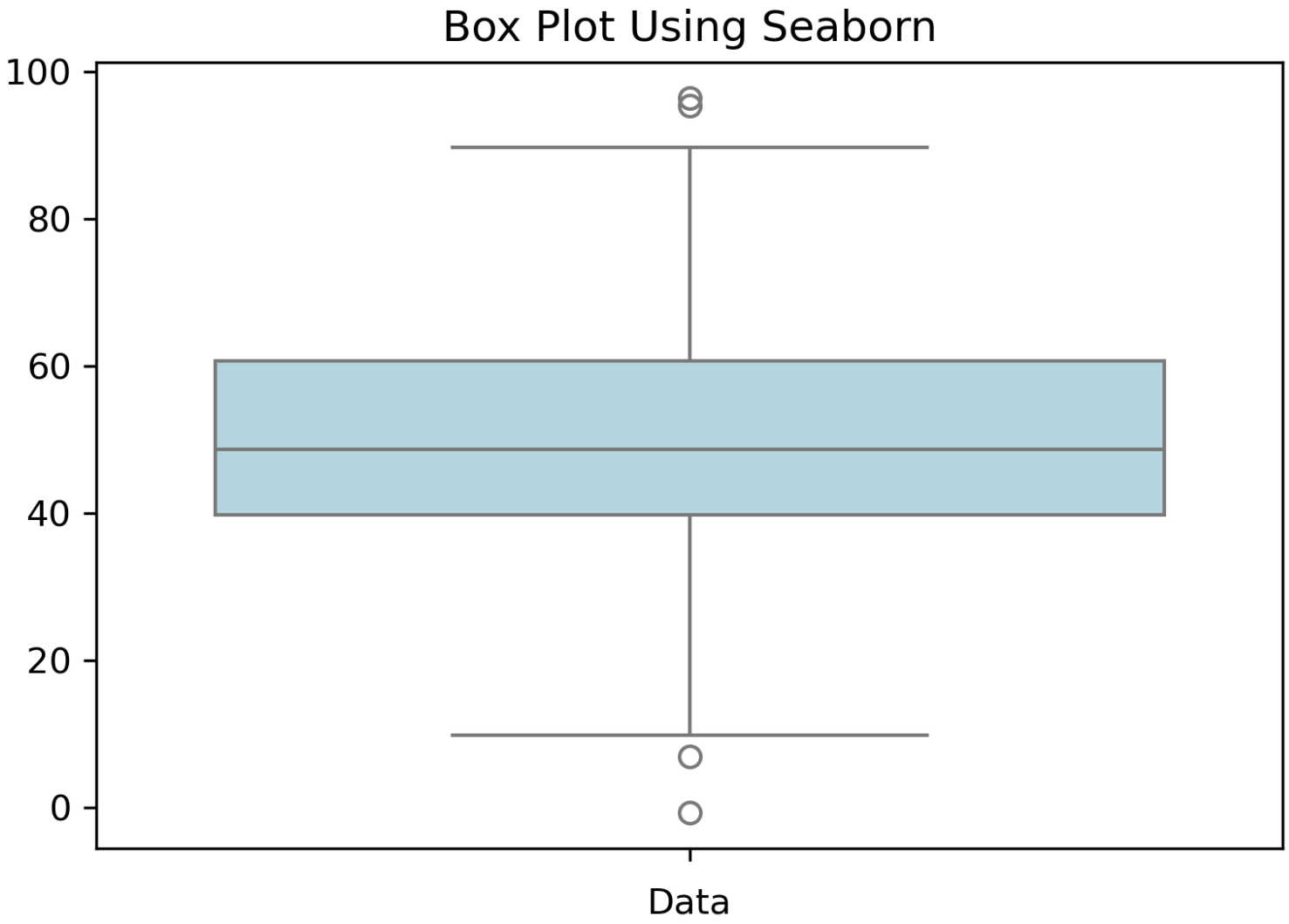
When to Use Seaborn:

- For creating statistical plots quickly and easily.
- To produce attractive and informative visualizations with minimal effort.
- When you need built-in support for visualizing distributions and relationships.

Basic plotting in seaborn

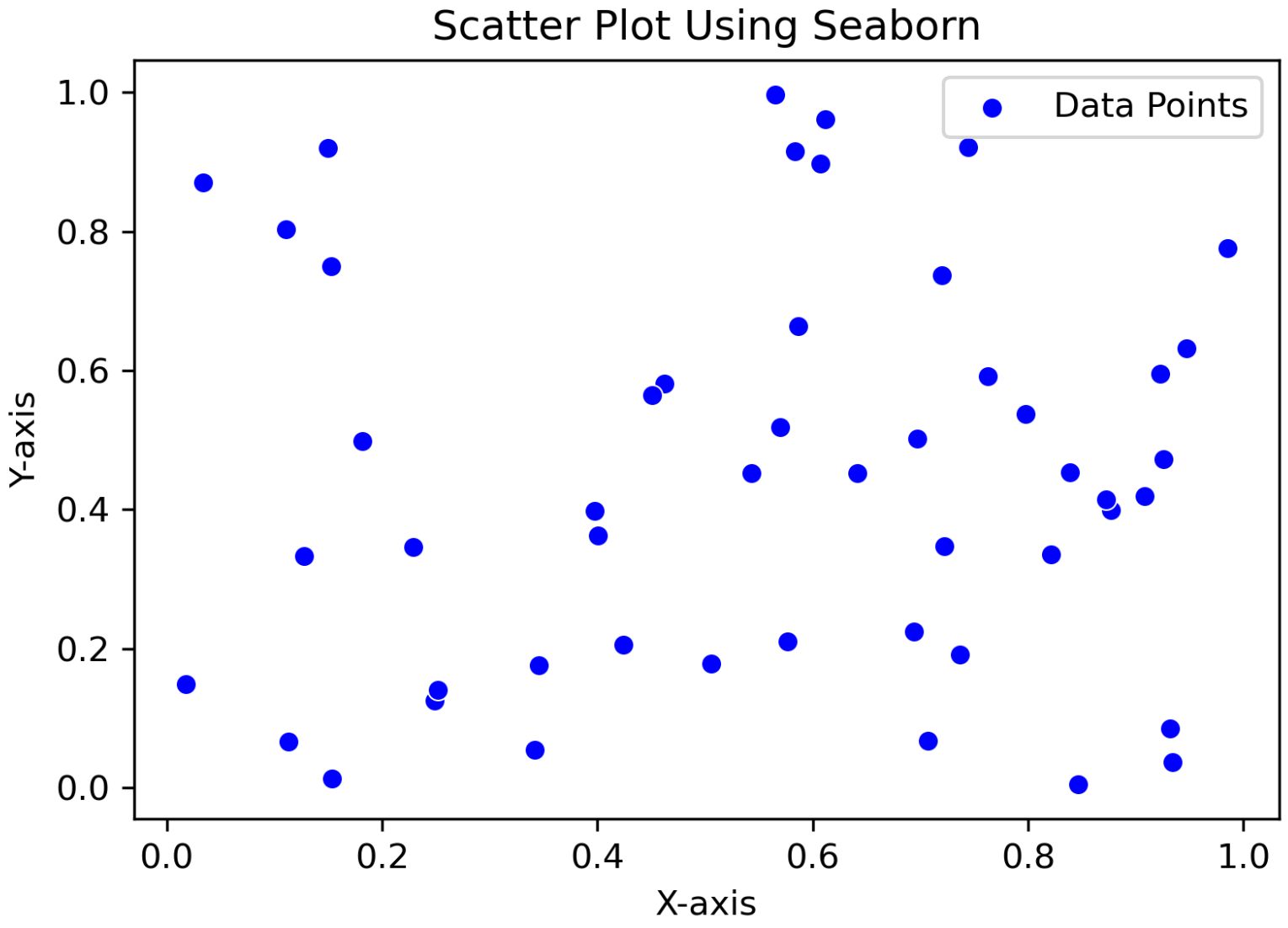
Box Plot

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
# Sample data (Manually provided)
data = np.random.normal(loc=50, scale=20, size=100)  # 100 data points with normal distribution
# Create box plot
plt.figure(figsize=(6, 4))
sns.boxplot(data=data, color='lightblue')
# Labels and title
plt.xlabel("Data")
plt.title("Box Plot Using Seaborn")
# Display the plot
plt.show()
```


Box Plot Using Seaborn
100
80
60
40
20
0
Data

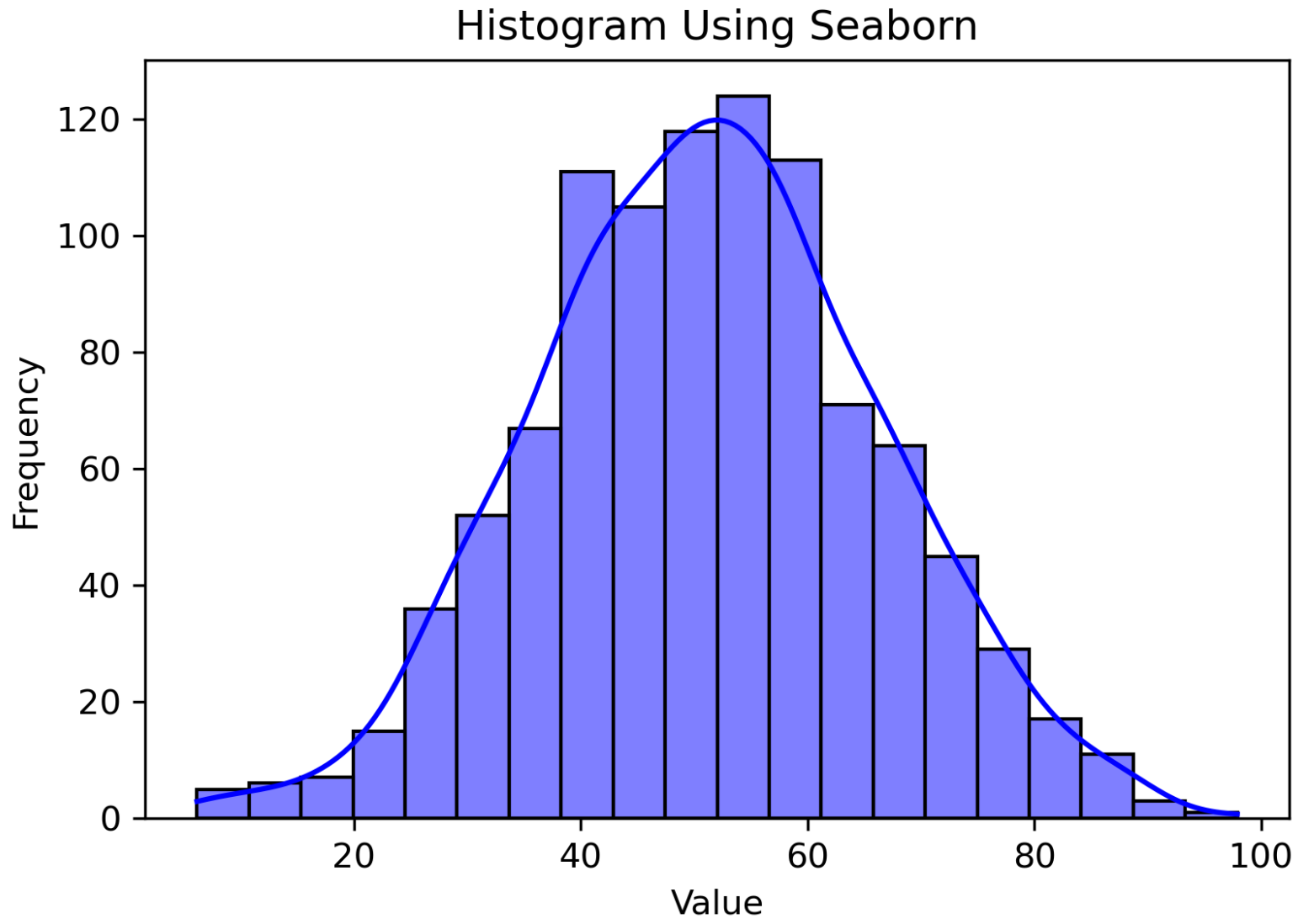
Scatter plot

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
# Sample data (Manually provided)
x_values = np.random.rand(50)  # 50 random values for X-axis
y_values = np.random.rand(50)  # 50 random values for Y-axis
# Create scatter plot
plt.figure(figsize=(6, 4))
sns.scatterplot(x=x_values, y=y_values, color='blue', marker='o', label="Data Points")
# Labels and title
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Scatter Plot Using Seaborn")
# Show legend
plt.legend()
# Display the plot
plt.show()
```


Scatter Plot Using Seaborn
Data Points
Y-axis
X-axis
0.0
0.2
0.4
0.6
0.8
1.0

Histogram

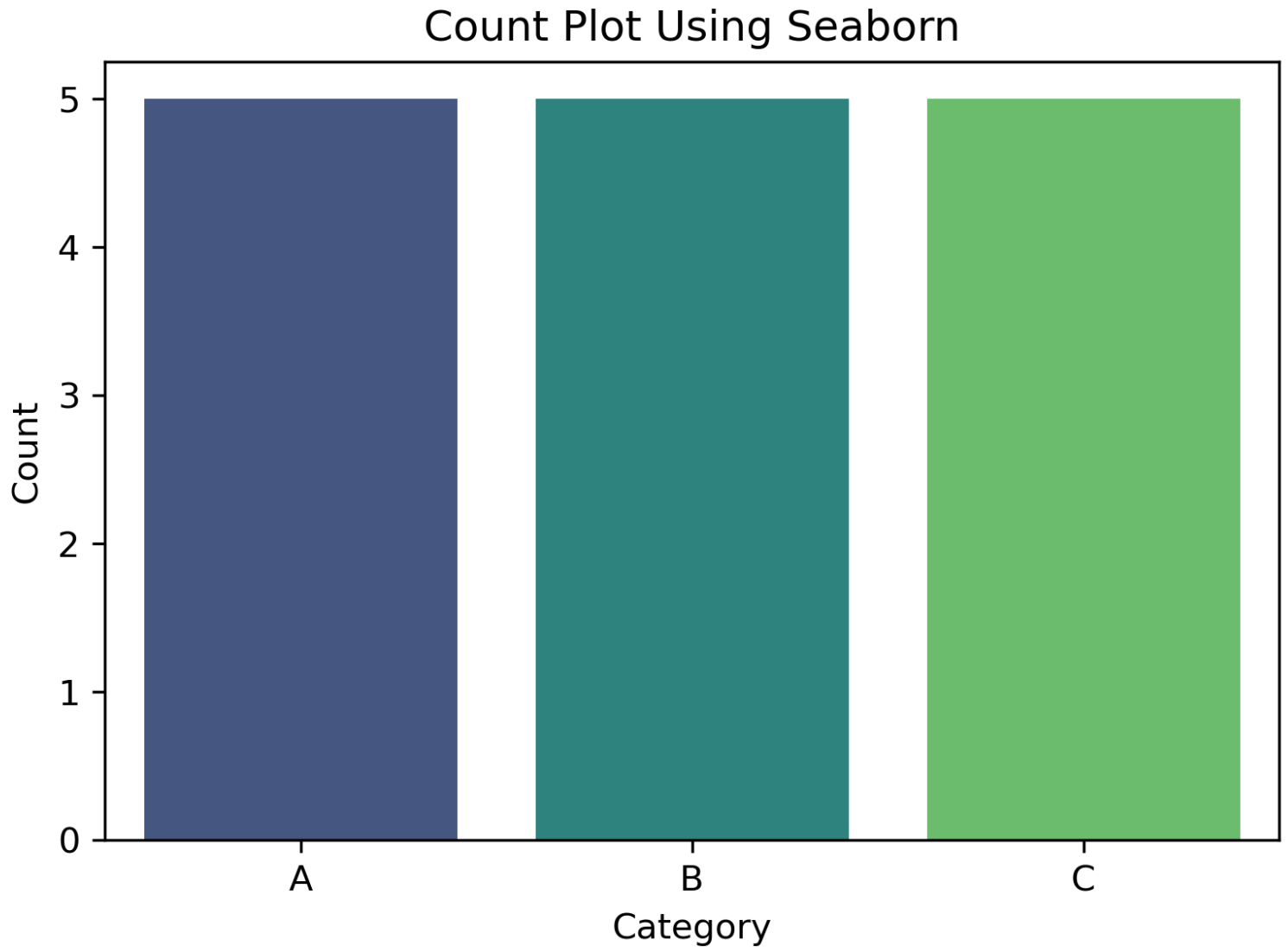
```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
# Sample data (Manually provided)
data = np.random.normal(loc=50, scale=15, size=1000)  # 1000 data points with a normal distribution
# Create histogram using Seaborn
plt.figure(figsize=(6, 4))
sns.histplot(data, bins=20, kde=True, color='blue', edgecolor='black')
# Labels and title
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.title("Histogram Using Seaborn")
# Display the plot
plt.show()
```


Histogram Using Seaborn
Frequency
Value
0
20
40
60
80
100
120
20
40
60
80
100

Countplot

- A countplot is a plot between a categorical and a continuous variable. The continuous variable in this case being the number of times the categorical is present or simply the frequency.
- In a sense, count plot can be said to be closely linked to a histogram or a bar graph.

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
# Sample categorical data (Manually provided)
data = pd.DataFrame({
    "Category": ["A", "B", "A", "C", "B", "A", "C", "C", "B", "A", "B", "C", "A", "B", "C"]
})
# Create count plot
plt.figure(figsize=(6, 4))
sns.countplot(x="Category", data=data, palette="viridis")
# Labels and title
plt.xlabel("Category")
plt.ylabel("Count")
plt.title("Count Plot Using Seaborn")
# Display the plot
plt.show()
```

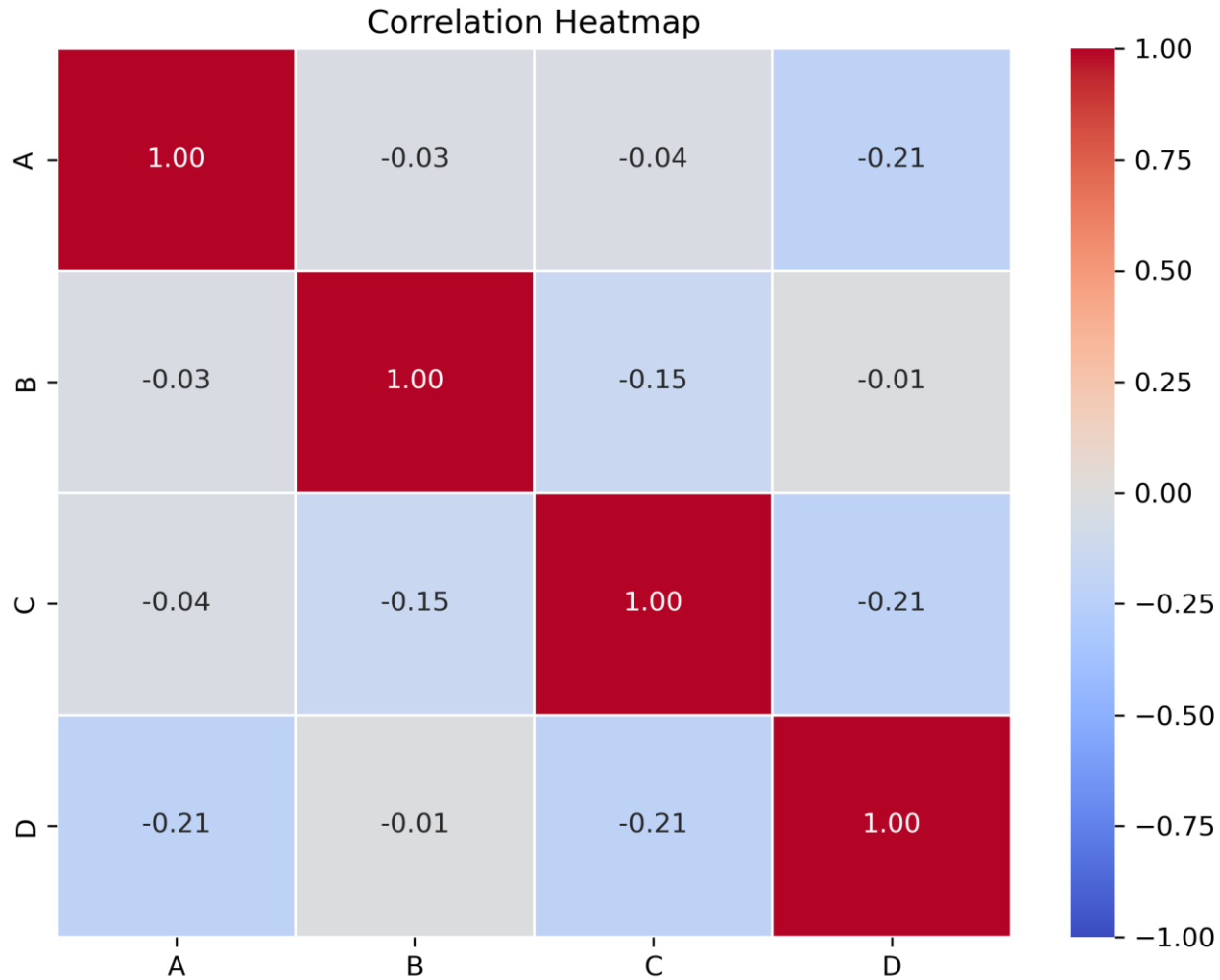

Count Plot Using Seaborn
Count
0
1
2
3
4
5
A
B
C
Category

Correlation plot

- Correlation plot is a multi-variate analysis which comes very handy to have a look at relationship with data points. Scatter plots helps to understand the affect of one variable over the other. Correlation could be defined as the affect which one variable has over the other.

- Correlation could be calculated between two variables or it could be one versus many correlations as well which we could see the below plot. Correlation could be positive, negative or neutral and the mathematical range of correlations is from -1 to 1. Understanding the correlation could have a very significant effect on the model building stage and also understanding the model outputs.

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
# Manually generate random numerical data
np.random.seed(42)
data = pd.DataFrame({
    'A': np.random.rand(100),
    'B': np.random.rand(100),
    'C': np.random.rand(100),
    'D': np.random.rand(100)
})
# Compute the correlation matrix
corr_matrix = data.corr()
# Create the correlation heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5, vmin=-1, vmax=1)
# Title
plt.title("Correlation Heatmap")
# Display the plot
plt.show()
```

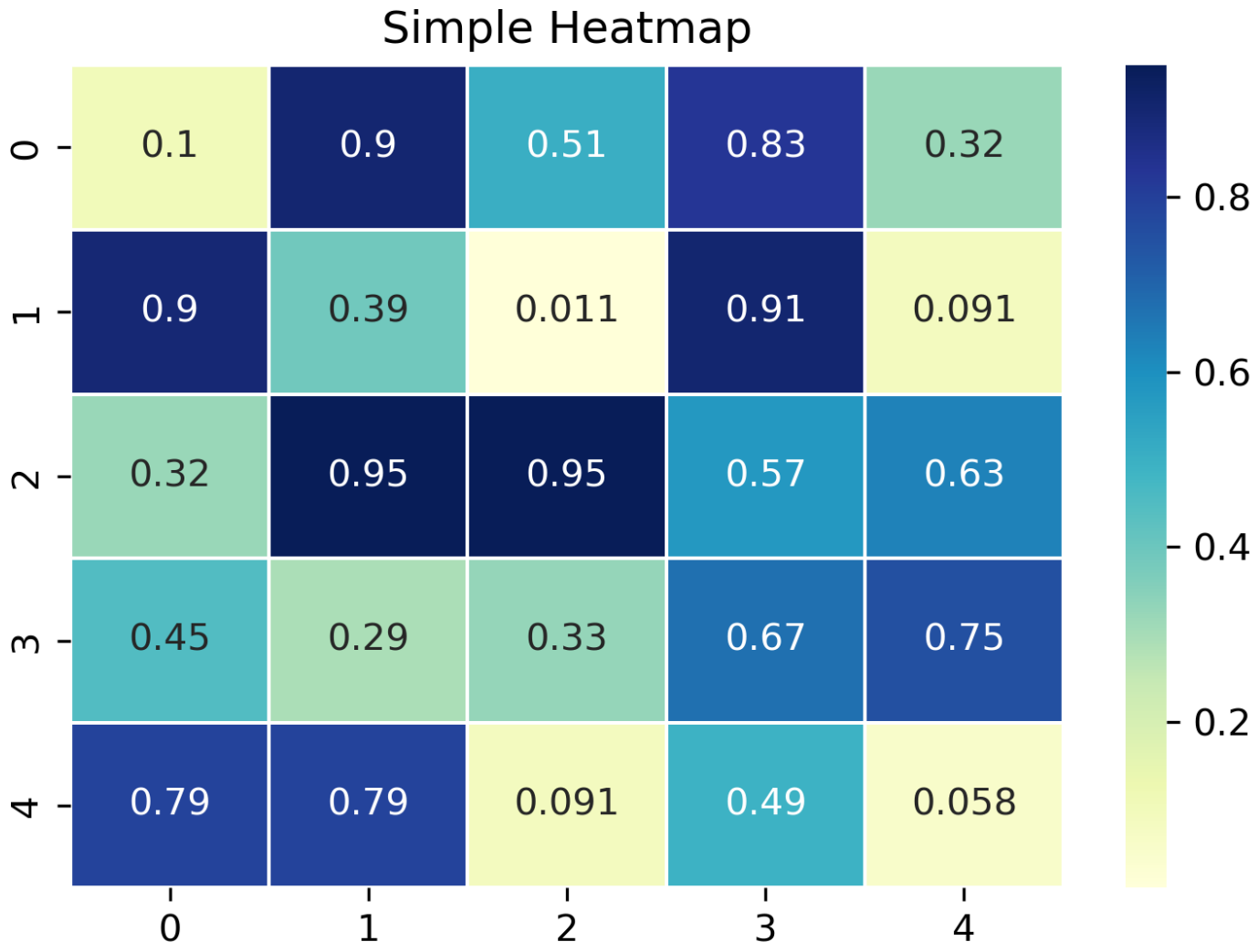

Correlation Heatmap
A
B
C
D
1.00
-0.03
-0.04
-0.21
-0.03
1.00
-0.15
-0.01
-0.04
-0.15
1.00
-0.21
-0.21
-0.01
-0.21
1.00
A
B
C
D
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

Heat Maps

- Heat map is a multi-variate data representation. The color intensity in a heat map displays becomes an important factor to understand the affect of data points.
- Heat maps are easier to understand and easier to explain as well. When it comes to data analysis using visualization, its very important that the desired message gets conveyed with the help of plots.

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
# Manually generate numerical data (random matrix)
data = np.random.rand(5, 5)  # 5x5 matrix of random values

# Create the heatmap
plt.figure(figsize=(6, 4))
sns.heatmap(data, annot=True, cmap="YlGnBu", linewidths=0.5)
# Title
plt.title("Simple Heatmap")
# Display the plot
plt.show()
```


Simple Heatmap
0.1
0.9
0.51
0.83
0.32
0.9
0.39
0.011
0.91
0.091
0.32
0.95
0.95
0.57
0.63
0.45
0.29
0.33
0.67
0.75
0.79
0.79
0.091
0.49
0.058
0
1
2
3
4
0.8
0.6
0.4
0.2

Summary of Visualization Techniques

- Line Plot: Useful for showing trends over time.
- Bar Plot: Effective for comparing quantities across categories.
- Histogram: Helps in understanding the distribution of numerical data.
- Scatter Plot: Explores relationships between two numerical variables.
- Box Plot: Visualizes the distribution of data across different categories.
- Heatmap: Provides a color-coded matrix to visualize data patterns.

Seaborn vs Matplotlib

- Matplotlib is a low-level plotting library that provides a high degree of control over individual elements. Even for basic functionalities, it requires more code.
- Whereas seaborn is a high level library for visualization and requires less coding compared to Matplotlib.
- Matplotlib lets users customize the appearances of plots, including color and styles.
- Seaborn has in-built themes and color palettes making it easier for users to create visually appealing plots.
- Matplotlib can work with pandas but users may need to manipulate data for certain type of plots.
- Seaborn is very much flexible with pandas and it doesn’t require as much manipulation as Matplotlib.