Data Visualization:

Python provides powerful libraries for data analysis and visualization, notably **Matplotlib** and **Seaborn**. These tools help explore data, identify patterns, and communicate insights effectively.

Bivariate Plotting (Two Variables)

Bivariate plots show relationships between **two** variables. These are useful for understanding correlations, distributions, and trends.

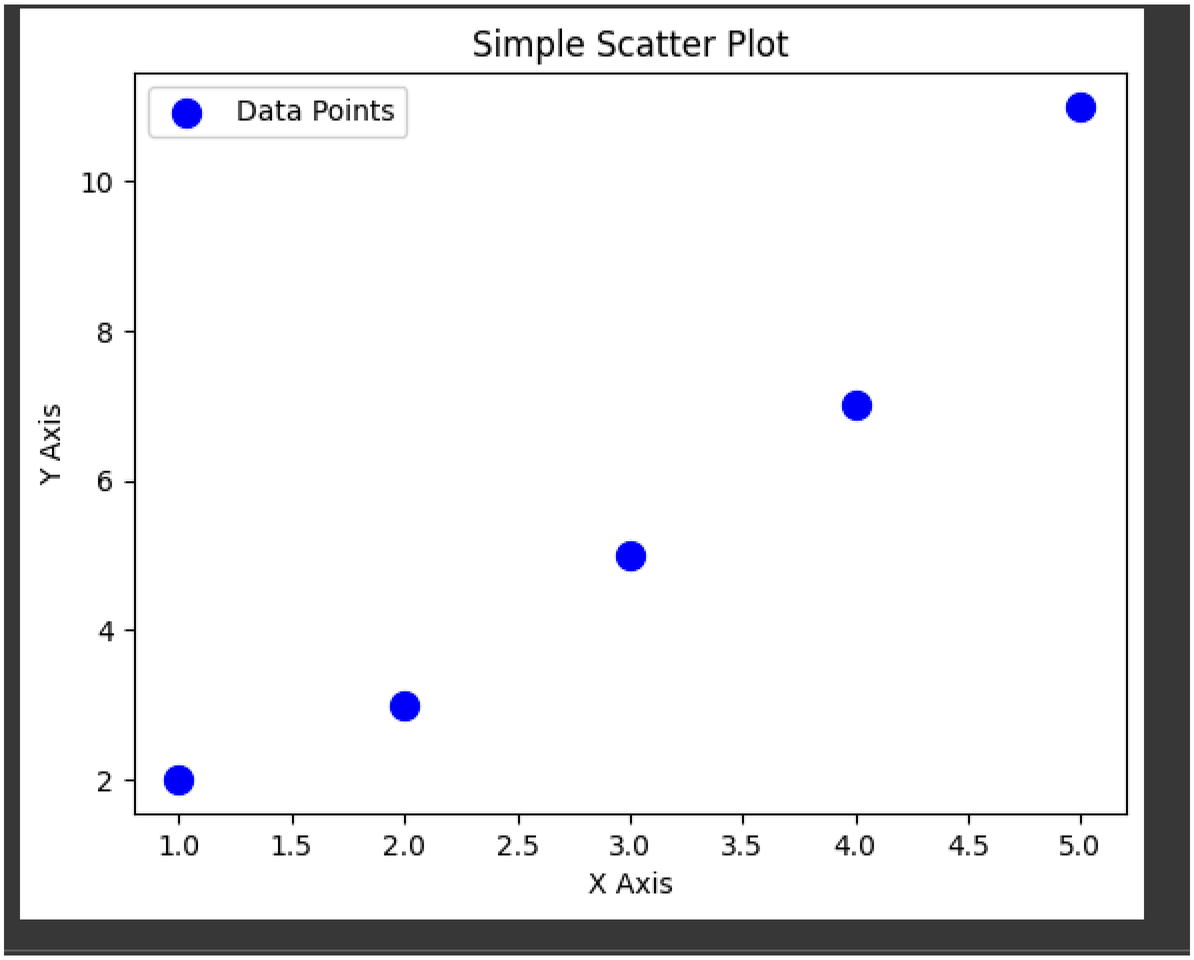
Examples:

- **Scatter Plot** – Shows relationships between two continuous variables.
- **Box Plot** – Helps detect **outliers** and visualize the distribution of a single variable across different categories.
- **Line Plot** – Represents trends between two numerical variables over time.

Note**:** Read the official documentation for **Matplotlib** and **Seaborn** to understand their built-in functionalities. Pay close attention to the **parameters** you can pass to different functions.

Following is the example to create a simple **scatter plot using matplot lib**

```
import matplotlib.pyplot as plt

# Sample data for x and y axes

x = [1, 2, 3, 4, 5]

y = [2, 3, 5, 7, 11]

# Create a scatter plot

plt.scatter(x, y,

            s=100,            # 's' sets the marker size (in points^2)

            c='blue',         # 'c' sets the color of the markers

            marker='o',       # 'marker' sets the shape of the marker ('o' 
for circle)

            label='Data Points')  # 'label' is used for the legend

# Adding labels and title

plt.xlabel('X Axis')     # Label for the x-axis
plt.ylabel('Y Axis')     # Label for the y-axis

plt.title('Simple Scatter Plot')   # Title of the plot

# Display legend to show the label

plt.legend()

# Display the plot

plt.show()
```



You can do the same thing using **Seaborn library**:

```python
import seaborn as sns

import matplotlib.pyplot as plt


# Sample data for x and y axes

x = [1, 2, 3, 4, 5]

y = [2, 3, 5, 7, 11]


# Create a scatter plot using Seaborn

sns.scatterplot(x=x, y=y,

                s=100,       # 's' sets the marker size

                color='blue', # 'color' sets the marker color

                marker='o',   # 'marker' sets the shape of the marker

                label='Data Points')  # 'label' adds a legend entry


# Adding labels and title

plt.xlabel('X Axis')

plt.ylabel('Y Axis')

plt.title('Simple Scatter Plot with Seaborn')


# Display legend

plt.legend()


# Show the plot

plt.show()
```

<u>If you observe code carefully Matplotlib’s Pyplot is being used with seaborn, why?</u>

- Seaborn works on top of Matplotlib, but it does not manage figures and axes directly. Using `plt` allows you to set figure sizes, adjust spacing, and modify axis labels.
- Seaborn can plot the data, but adding a **title (`plt.title()`), axis labels (`plt.xlabel()`, `plt.ylabel()`), and legends (`plt.legend()`)** still requires Matplotlib functions.
- For tasks like **customizing ticks, setting grid lines, or saving the plot as an image (`plt.savefig()`)**, Matplotlib is necessary.

Multivariate Plotting (Three or More Variables)

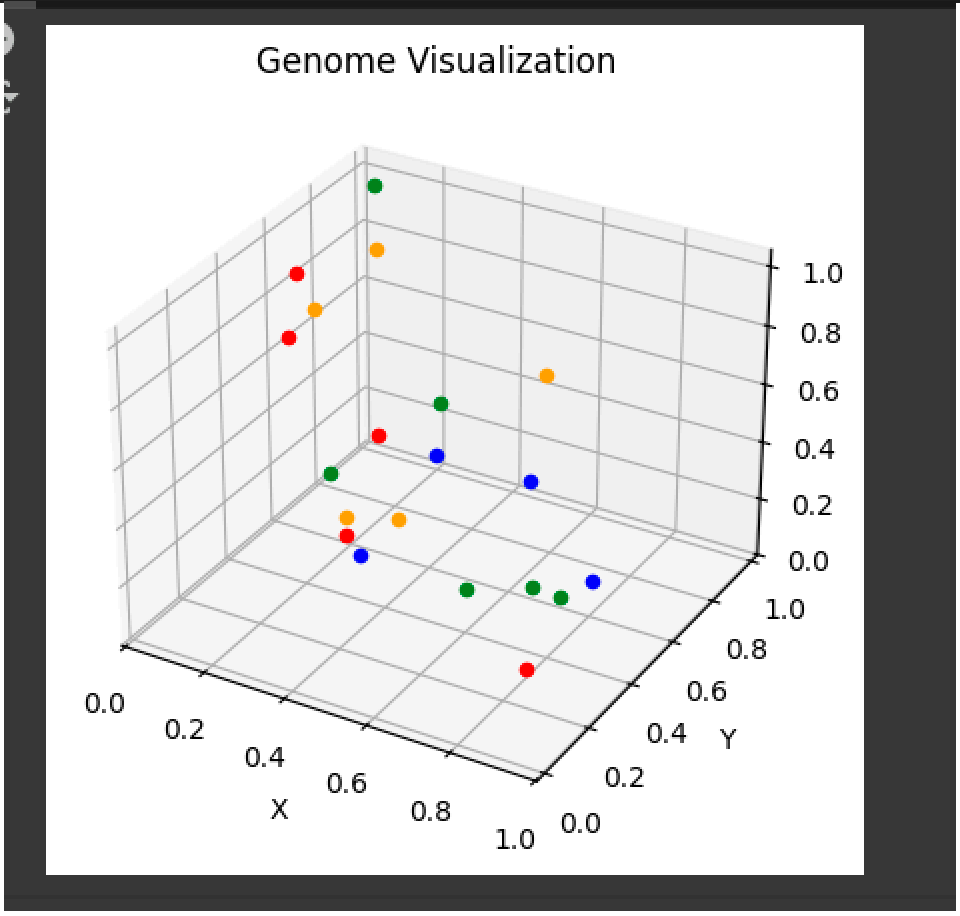
Multivariate plots analyze relationships among **three or more** variables, allowing for deeper insights.

Heatmap – Uses colors to show correlations or frequency distributions across multiple variables.
3D Scatter Plot – Extends the scatter plot into three dimensions using `matplotlib.pyplot.scatter` with the `projection='3d'` parameter.

For instance, in the field of bioinformatics you can use these tools to map and visualise human-genomes, DNA,rNA and protein sequences. Following is just a simple demonstration of that using random sequences in 3d scatter plot.

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

# Generate random genome sequence
nucleotides = ["A", "T", "C", "G"]
genome_length = 20
genome = ""
for _ in range(genome_length):
    genome += random.choice(nucleotides)

# Create 3D plot
fig = plt.figure()
ax = fig.add_subplot(projection='3d')

# Generate random coordinates for each nucleotide
coordinates = np.random.rand(genome_length, 3)

# Color mapping
colors = {'A': 'red', 'T': 'blue', 'C': 'green', 'G': 'orange'}
```

```python
# Plot each nucleotide with its corresponding color and coordinates
for i, nucleotide in enumerate(genome):
    ax.scatter(coordinates[i, 0], coordinates[i, 1], coordinates[i, 2],
               color=colors[nucleotide], marker='o')

ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("Z")
ax.set_title("Genome Visualization")

plt.show()
```



Numpy:

NumPy is a fundamental library in Python that is widely used in data science and machine learning for storing and preprocessing tabular ,image, video, audio data that can readily feed into AI models for input.

Numpy Functions : https://www.programiz.com/python-programming/numpy/array-functions

Following is a code read an image into an np.array

```
from PIL import Image
from numpy import asarray


# load the image and convert into
# numpy array
img = Image.open('Sample.png')
numpydata = asarray(img)

# data
print(numpydata)
```

```
[[[111  60   0]
  [116  65   0]
  [122  69   0]
  ...
  [ 97  47   0]
  [ 99  47   0]
  [100  49   0]]
 [[111  61   0]
  [118  65   0]
  [122  69   0]
  ...
  [ 97  47   0]
  [ 99  48   0]
```

As you can see, the image was first converted and opened as bytes using PIL(pillow library) then converted into a numpy array that represents each PIXEL RGB(red,green,blue) channel into a multidimensional array.

You can also plot that numpy array back into an image using matplot lib and apply different filters.

Panda:

Panda are most commonly used libraries in data science, which we will use for EDA (exploratory data analysis) which is used by data scientists to analyze and investigate data sets and summarize their main characteristics, often employing data visualization methods.

What is a DataFrame?

A Pandas DataFrame is a 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns.

Basic EDA pipe-line:

> Read the csv, excel file into panda dataframe

Reading dataframes: https://www.w3schools.com/python/pandas/pandas_csv.asp

- Rather than using print function use df.head() or df.tail to view top and below value respectively [assuming you stored your dataframe into df using pd.read_csv or pd.read_xlsx]
- Use apply specific function to specific rows of dataframe just by referencing df[‘column_name’] **lambda functions** are mostly used in this regard which is a one line function to lower_case each row in city_name column.

df['city_names'].apply(lambda x: x.lower())

You can use this combination of apply and lambdas to apply your defined function on each row of column.

> remove null and duplicate values.

> apply statistical analysis i.e mean, medium , mode values , what values are categorical and what are numerical values. Mostly done by df.describe(), max(), mean(), value_count functions.
State Space Search: