1. **NumPy:**
   * Purpose: Numerical computing in Python.
   * Key features: Multidimensional arrays, mathematical functions, and linear algebra operations.
   * Example use: Array manipulation, mathematical operations on large datasets.
2. **Pandas:**
   * Purpose: Data manipulation and analysis.
   * Key features: DataFrame for structured data, data cleaning, and exploration tools.
   * Example use: Loading and cleaning datasets, performing exploratory data analysis.
3. **Matplotlib:**
   * Purpose: 2D plotting library for creating visualizations.
   * Key features: Line plots, scatter plots, bar plots, and more.
   * Example use: Data visualization, exploring patterns and trends in data.
4. **Seaborn:**
   * Purpose: Statistical data visualization based on Matplotlib.
   * Key features: High-level interface for drawing attractive and informative statistical graphics.
   * Example use: Enhanced visualization of statistical relationships in data.
5. **Scikit-learn:**
   * Purpose: Machine learning library for classical algorithms.
   * Key features: Classification, regression, clustering, and model selection tools.
   * Example use: Building and evaluating machine learning models.
6. **SciPy:**
   * Purpose: Library for scientific computing built on NumPy.
   * Key features: Integration, optimization, signal and image processing.
   * Example use: Advanced mathematical operations and scientific computing.
7. **Statsmodels:**
   * Purpose: Statistical modelling and hypothesis testing.
   * Key features: Regression models, time-series analysis, and statistical tests.
   * Example use: Statistical analysis and hypothesis testing on data.
8. **TensorFlow:**
   * Purpose: Open-source machine learning framework.
   * Key features: Deep learning, neural network models.
   * Example use: Building and training deep learning models.
9. **Keras:**
   * Purpose: High-level neural networks API (often used with TensorFlow).
   * Key features: Simplifies the process of building and training neural networks.
   * Example use: Rapid development and prototyping of neural network models.
10. **NLTK (Natural Language Toolkit):**

* Purpose: Library for working with human language data.
* Key features: Text processing, tokenization, part-of-speech tagging.
* Example use: Natural language processing tasks, such as sentiment analysis.

1. **Import Relevant Libraries:**
   * Use **import** statements to bring in libraries like Pandas for data manipulation, NumPy for numerical operations, or any other library specific to your data format.
2. **Understand Your Data Format:**
   * Know the format of your data (CSV, Excel, SQL, etc.).
   * Libraries like Pandas support a variety of data formats, and the method of loading data may vary based on the format.
3. **Loading CSV Data:**
   * Use **pd.read\_csv()** in Pandas to load data from a CSV file.

# Example for loading a CSV file df = pd.read\_csv('your\_data.csv')

1. **Loading Excel Data:**
   * Use **pd.read\_excel()** in Pandas to load data from an Excel file.

# Example for loading an Excel file df = pd.read\_excel('your\_data.xlsx')

1. **Loading SQL Data:**
   * Use **pd.read\_sql()** in Pandas to load data from a SQL database.

# Example for loading data from a SQL database import sqlite3 conn = sqlite3.connect('your\_database.db') query = "SELECT \* FROM your\_table;" df = pd.read\_sql(query, conn)

1. **Viewing Data:**
   * Use **head()** or **tail()** to quickly view the first or last few rows of the DataFrame.

# View the first 5 rows of the DataFrame df.head()

1. **Data Inspection:**
   * Use methods like **info()**, **describe()**, and **shape** to get an overview of the data.

# Display information about the DataFrame df.info() # Get descriptive statistics of the DataFrame df.describe() # Display the dimensions of the DataFrame df.shape

1. **Dealing with Missing Data:**
   * Use methods like **dropna()** or **fillna()** to handle missing values.

# Drop rows with missing values df.dropna() # Fill missing values with a specified value df.fillna(value)

1. **Saving Data:**
   * Use **to\_csv()**, **to\_excel()**, or other methods to save the manipulated data back to a file.

# Save DataFrame to a CSV file df.to\_csv('new\_data.csv', index=False)

1. **Visualizing Data:**
   * Utilize libraries like Matplotlib and Seaborn to create visualizations and explore patterns in the data.

import matplotlib.pyplot as plt import seaborn as sns # Example: Plotting a histogram sns.histplot(df['column\_name'], bins=20) plt.show()