**Python Libraries**

 **Definition**: A Python library is a bundle of Python modules that you can import and use in your code. A module is a single file containing Python code, while a library is a collection of modules.

 **Purpose**: Libraries help in:

* **Code Reusability**: Avoiding the need to write common functionalities from scratch.
* **Efficiency**: Streamlining development by providing tested and optimized code.
* **Specialization**: Offering tools and functions specific to particular domains or tasks.

Some of the python libraries are discussed below:

**Pandas(**tool for handling and manipulating structured data-data cleaning, manipulation and integration**)**

* **Purpose**: Data manipulation and analysis.
* **Key Data Structures**:
  + **Series**: 1D array-like object with labels (index). Useful for handling single-column data.
  + **DataFrame**: 2D table with rows and columns. Suitable for structured data.
* **Common Operations**:
  + **Data Loading**: pd.read\_csv(), pd.read\_excel()
  + **Data Selection**: df.loc[], df.iloc[]
  + **Data Cleaning**: df.dropna(), df.fillna()
  + **Data Aggregation**: df.groupby(), df.agg()
* **Use Cases**: Data cleaning, exploration, and manipulation.

**NumPy(**scientific computing and data analysis- includes multidimensional array, mathematical functions**)**

* **Purpose**: Numerical computing.
* **Core Data Structure**:
  + **ndarray**: N-dimensional array for numerical data. Supports vectorized operations.
* **Common Operations**:
  + **Array Creation**: np.array(), np.zeros(), np.ones()
  + **Array Operations**: Element-wise operations, broadcasting.
  + **Mathematical Functions**: np.mean(), np.std(), np.dot()
* **Use Cases**: Mathematical operations, scientific computing, as a base for other libraries.

**TensorFlow**

* **Purpose**: Machine learning and deep learning.
* **Core Components**:
  + **Tensors**: Multi-dimensional arrays used as the basic data structure.
  + **Graphs**: Computational graphs that represent the model structure.
  + **Sessions**: Execution environment for running the computational graph (in TensorFlow 1.x).
* **High-Level APIs**:
  + **Keras**: Integrated API for building and training models.
* **Common Operations**:
  + **Model Building**: tf.keras.Sequential(), tf.keras.Model()
  + **Training**: model.compile(), model.fit()
* **Use Cases**: Deep learning, neural network training and deployment.

**Keras(**used extensively for deep learning and neural networks modules**)**

* **Purpose**: High-level API for building and training neural networks.
* **Core Features**:
  + **Model Types**: Sequential, Functional API, Model Subclassing
  + **Layers**: Dense, Conv2D, LSTM, etc.
* **Common Operations**:
  + **Model Definition**: keras.Sequential(), keras.Model()
  + **Compilation**: model.compile()
  + **Training**: model.fit()
* **Use Cases**: Simplifies neural network creation and training, especially for prototyping.

**scikit-learn (sklearn)(**provides tools for building and evaluating machine learning models**)**

* **Purpose**: Traditional machine learning.
* **Core Components**:
  + **Estimators**: Objects that implement fit() and predict(), e.g., classifiers, regressors.
  + **Transformers**: Objects that implement fit\_transform() for data preprocessing.
* **Common Operations**:
  + **Model Training**: fit()
  + **Prediction**: predict()
  + **Evaluation**: accuracy\_score(), confusion\_matrix()
* **Use Cases**: Classification, regression, clustering, dimensionality reduction, and model evaluation.
* **Includes algorithms such as**: decision trees, random forest, support vector machine, logistic regression, k- nearest algorithm **etc.**

**PyTorch(**python based scientific computing package**)**

* **Purpose**: Machine learning and deep learning.
* **Core Components**:
  + **Tensors**: N-dimensional arrays similar to NumPy arrays but with support for GPU acceleration.
  + **Autograd**: Automatic differentiation for gradient computation.
  + **Dynamic Computation Graph**: Allows changes in the graph during runtime.
* **Common Operations**:
  + **Model Definition**: torch.nn.Module
  + **Training**: optimizer.step(), loss.backward()
  + **Data Loading**: torch.utils.data.DataLoader
* **Use Cases**: Deep learning research and applications, particularly for tasks requiring dynamic computation graphs.