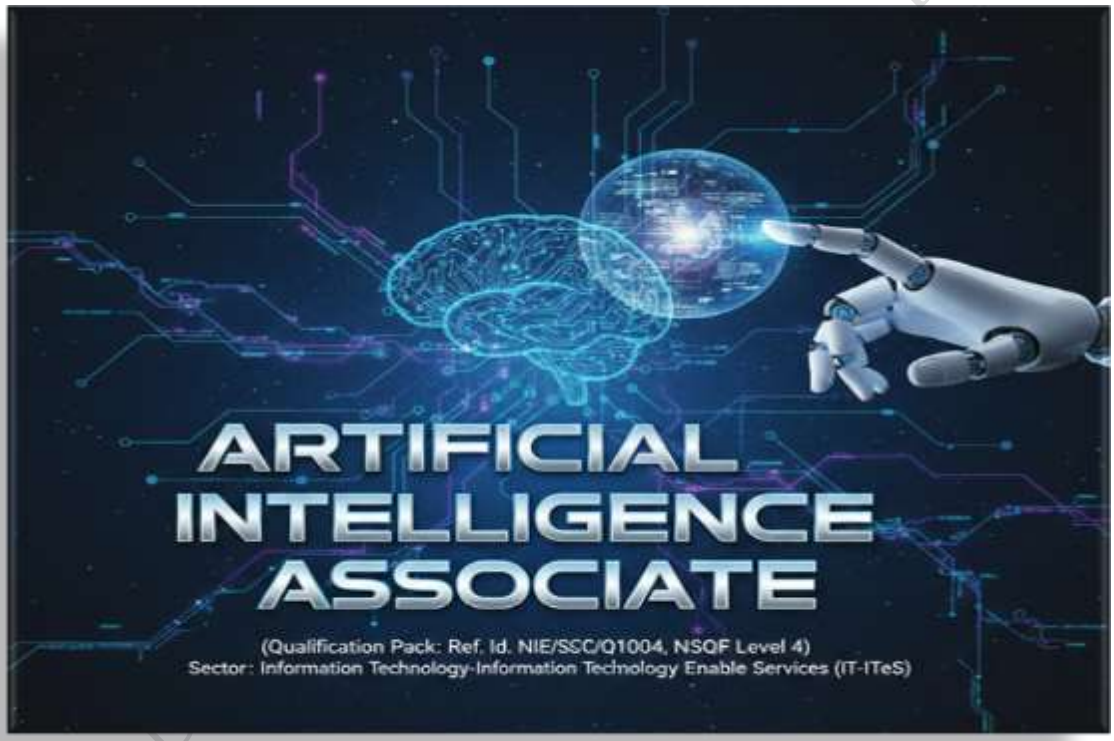
**Draft Study Material**

# ARTIFICIAL INTELLIGENCE ASSOCIATE

**(Qualification Pack: Ref. Id. NIE/SSC/Q1004, NSQF Level 4)**
**Sector: Information Technology-Information Technology Enable Services (IT-ITeS)**

## (Grade XII)








**PSS CENTRAL INSTITUTE OF VOCATIONAL EDUCATION**
**(a constituent unit of NCERT, under Ministry of Education, Government of India)**
**Shyamla Hills, Bhopal- 462 002, M.P., India**
**https://www.psscive.ac.in**

# Preface

Vocational Education is a dynamic and evolving field, and ensuring that every student has access to quality learning materials is of paramount importance. The journey of the PSS Central Institute of Vocational Education (PSSCIVE) toward producing comprehensive and inclusive study material is rigorous and time-consuming, requiring thorough research, expert consultation, and publication by the National Council of Educational Research and Training (NCERT). However, the absence of finalized study material should not impede the educational progress of our students. In response to this necessity, we present the draft study material, a provisional yet comprehensive guide, designed to bridge the gap between teaching and learning, until the official version of the study material is made available by the NCERT. The draft study material provides a structured and accessible set of materials for teachers and students to utilize in the interim period. The content is aligned with the prescribed curriculum to ensure that students remain on track with their learning objectives.

The contents of the modules are curated to provide continuity in education and maintain the momentum of teaching-learning in vocational education. It encompasses essential concepts and skills aligned with the curriculum and educational standards. We extend our gratitude to the academicians, vocational educators, subject matter experts, industry experts, academic consultants, and all other people who contributed their expertise and insights to the creation of the draft study material.

Teachers are encouraged to use the draft modules of the study material as a guide and supplement their teaching with additional resources and activities that cater to their students' unique learning styles and needs. Collaboration and feedback are vital; therefore, we welcome suggestions for improvement, especially by the teachers, in improving upon the content of the study material.





**Deepak Paliwal**
**(Joint Director)**
**PSSCIVE, Bhopal**

# STUDY MATERIAL DEVELOPMENT COMMITTEE

***Members***

*Akshya Arya, Assistant Professor, Department of Artificial Intelligence & Data Science, Arya College of Engineering, Main Campus, SP40, Kukas, Jaipur (Rajsthan)*

*Ganesh Kumar Dixit, Professor & Head, Department of Artificial Intelligence & Data Science, Arya College of Engineering, Main Campus, SP40, Kukas, Jaipur (Rajsthan)*

*Prakash Khanale, Professor and Head, Department of Computer Science, DSM College, Parbhani, Maharashtra*

***Member Coordinator***

*Deepak D. Shudhalwar, Professor (CSE), Head, Department of Engineering and Technology, PSSCIVE, NCERT, Bhopal, Madhya Pradesh*

# TABLE OF CONTENTS

# Module 1. Data Science

## Module Overview

Data science is about using data to gain knowledge and make better decisions by collecting, cleaning, analyzing, and interpreting information to uncover patterns and insights. Data science empowers the industries to make smarter, faster, and more informed decisions.

Imagine a supermarket that records the daily sales of thousands of products. If the manager wants to know which product sells the most, which day had the highest sales, or how sales are changing over time, they cannot check each bill manually. Instead, they use data science tools that can organize, analyze, and process huge amounts of numbers in seconds.

In the same way, this Module introduces students to NumPy (Numerical Python), a powerful tool for handling large amounts of data in an efficient manner. Just like neatly arranging books in a library or organizing clothes in a cupboard helps us save time, NumPy arranges numbers in arrays that can be easily accessed, reshaped, and computed.

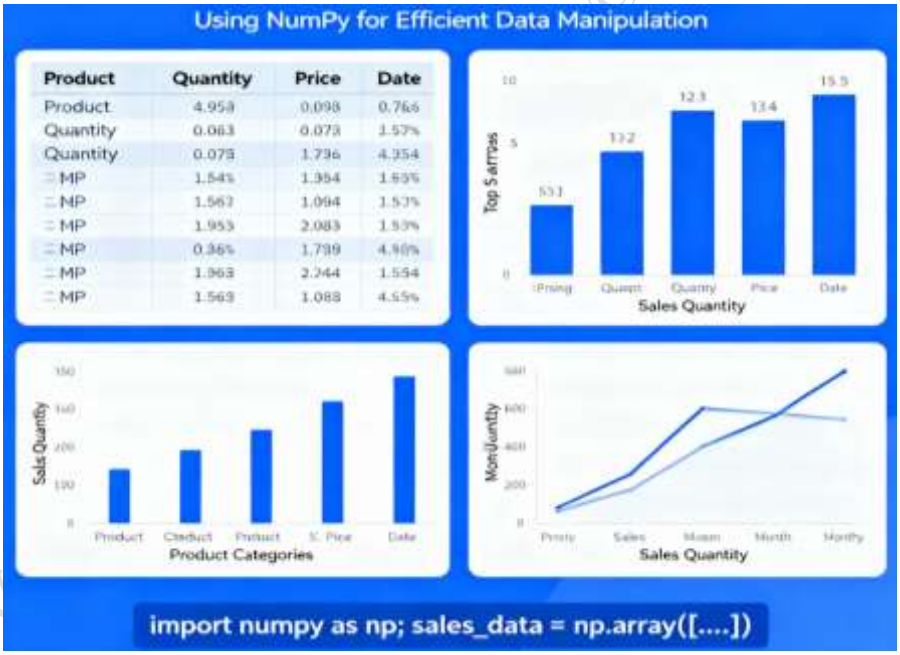



Fig. 1.0: Super market sales data analysis with NumPy

By the end of this Module, students will be able to use NumPy not only as a foundation for Data Science, but also as a stepping stone toward Machine Learning and Artificial Intelligence applications, where large amounts of data need to be processed quickly and accurately.

## Learning Outcomes

After completing this module, you will be able to:

- Create and use NumPy arrays for storing large datasets.
- Manipulate arrays by joining, splitting, reshaping, resizing, and changing dimensions.
- Perform arithmetic and mathematical computations efficiently on arrays.
- Apply aggregate functions and matrix operations for data analysis.

## Module Structure

Session 1. Introduction to NumPy

Session 2. Array Manipulation using NumPy

Session 3. Array Computation using NumPy



## Session 1. Introduction to NumPy

Think about a library where thousands of books are stored. If the books are kept randomly, finding one becomes difficult and time-consuming. But when they are arranged systematically in shelves, searching and managing becomes fast and easy. In the same way, NumPy arrays help organize data in a structured way, making it easier and faster to process compared to normal Python lists.

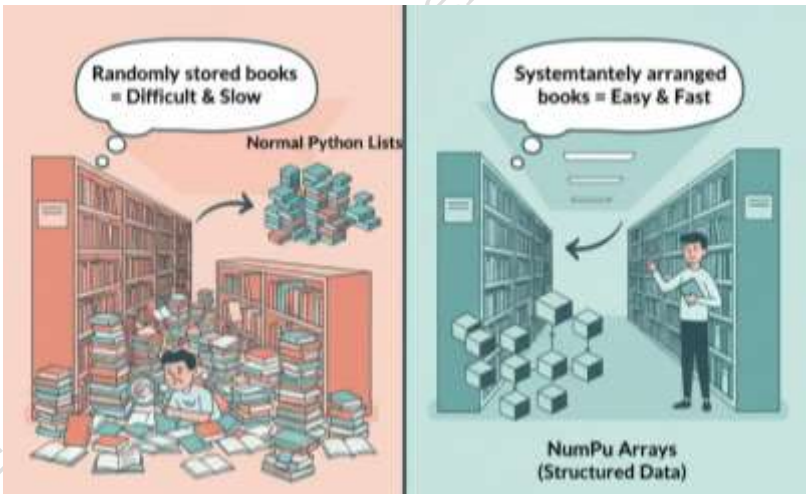



Fig. 1.1: Using NumPy arrays organize data

In this session you will learn how NumPy arrays work, why they are faster than lists, and how to create and use them. They will also learn the key properties of arrays such as shape, size, dimensions, data type, and memory usage. Students will practice creating 1-D, 2-D and multi-dimensional arrays, and explore techniques like reshaping, flattening, and broadcasting.

### 1.1 INTRODUCTION

In Python, the lists data structure serves the purpose of arrays, but they are slow to process. NumPy aims to provide an array object that is much faster than traditional Python lists. NumPy arrays are stored at one continuous place in memory unlike lists, so processes

can access and manipulate them very efficiently. The elements of a NumPy array must all be of the same type, whereas the elements of a Python list can be of completely different types.

**NumPy** stands for *Numerical Python*. It is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object. It provides functions for fast mathematical computation on arrays and matrices. The powerful n-dimensional array in NumPy speeds-up data processing. NumPy objects are primarily used to create arrays or matrices that can be applied to Deep Learning or Machine Learning models.

### 1.2 ARRAY

An array is a data type used to store multiple values using a single variable name. An array contains an ordered collection of data elements where each element is of the same type and can be referenced by its index (position).

***The important characteristics of an array are:***

- Array is used to store the different values of the same data type.
- Array is stored contiguously in memory to fast operations.
- Array element is identified using array name with the index of that element. The index is unique for each element. The array is zero based indexing, means the index starts with 0.

For example, consider an array with 5 elements:

```
a = [21, 9, 29, 17, 30]
```

Here, the 1st element of the array is 21 with index value [0] associated with it; the 2nd value in the array is 9 and has the index value [1] associated with it, and so on. The last element is the 5th element with value equal to 30 and index [4]. This is very similar to the indexing of lists in Python.

#### 1.2.1 NumPy Array

NumPy arrays are used to store lists of numerical data, vectors and matrices. The NumPy library has a large set of routines (built-in functions) for creating, manipulating, and transforming NumPy arrays. Python language also has an array data structure, but it is not as versatile, efficient and useful as the NumPy array. The NumPy array is officially called *ndarray* but commonly known as array.

#### 1.2.2 Importance of NumPy Array

There are several reasons to use NumPy for data analysis as given below.

1. In NumPy we can create arrays for large amounts of data.
2. Various basic linear algebra operations, statistical operation, Fourier transform, sorting, searching, shape manipulation and random simulation operations can be performed by using NumPy.
3. NumPy package is the ndarray object. It means we can have n dimensional arrays of homogeneous data elements.
4. NumPy arrays have a fixed size. When the size is changed, then it will delete the original and a new array is created.

5. Consider a simple example of multiplying two arrays. In python we can write a code for the multiplication as given below.

```
c = [ ]
for i in range(len(a)):
    c.append(a[i]*b[i])
```

The above code when executed will produce the correct result, but will require time when both the arrays **a** and **b** will contain millions of numbers. The loop in python is inefficient when compared with the C programming language. When we use NumPy then the above multiplication can be achieved simply by writing c=a*b

Observe that NumPy gives the advantage of efficient programming.

6. NumPy is very fast because its code is more concise and easier to read. It contains fewer lines of codes therefore there are fewer bugs.
7. NumPy can be easily integrated with other scientific libraries of Python such as Pandas, Matplotlib, SciPy, and TensorFlow.

### 1.2.3 NumPy Arrays Vs Python Lists

Although NumPy arrays also hold elements like Python List, yet Numpy arrays are different data structures from Python lists. The key differences are as follows.

| **NumPy Array** | **Python List** |
|---|---|
| It is required to install and import Numpy Library to access Numpy Arrays. | It is a built-in function of python available in the core library of python. |
| Once a NumPy array is created, it is not possible to change its size. It is required to create a new array or overwrite the existing one. | It is possible to append/insert values in List. |
| NumPy array contain elements of same type (homogeneous). For example, an array of floats may be: [1.2, 5.4, 2.7] | List contain elements of different type (heterogeneous). For example, [1,3.4, 'hello', 'a@'] |
| Arrays support element wise operations. For example, if A1 is an array, it is possible to say A1/3 to divide each element of the array by 3. | Lists do not support element wise operations, for example, addition, multiplication, etc. because elements may not be of same type. |
| NumPy array takes up less space in memory as compared to a list because arrays do not require to store datatype of each element separately. | Lists can contain objects of different datatype that Python must store the type information for every element along with its element value. Thus, lists take more space in memory and are less efficient. |
| NumPy array supports vectorized operations. | List does not support vectorized operations. |

### 1.2.4 Installation of NumPy

To use NumPy, it should be installed. NumPy can be installed by using the Anaconda or pip command. If you don't have anaconda installed then install anaconda. To install NumPy using pip command issue the pip command on command promt.

```
pip install NumPy
```

## 1.3 CREATING NUMPY ARRAY

NumPy is a main object for the creation of homogeneous multidimensional arrays. An array is like a list of elements. Homogeneous means all the elements are similar in nature. So, a Numpy array consists of a table of elements. All the elements in the NumPy are indexed by a tuple of non-negative integers. The number of dimensions of the array are called as axex in the numpy.

### 1.3.1 Creating basic arrays using NumPy

There are several ways to create arrays. To create a basic array using NumPy, follow these steps:

**Step 1. Import NumPy**: To create an array and to use its methods, first you need to import the NumPy library, typically aliased as `np`.

```
>>> import numpy as np
```

**Step 2. Create a Python List or Tuple**: Define a standard Python list or tuple containing the elements you want in your array.

```
>> list = [1, 2, 3, 4, 5]
```

**Step 3. Convert to NumPy Array**: Use the np.array() function to convert the Python list or tuple into a NumPy array.

```
>> array = np.array (list)
```

**Step 4. Print or Use the Array**: The resulting array is a NumPy ndarray object, ready for numerical operations.

```
>> print (array)
>> print (type(array))
```

### 1.3.2 Creating a 1-D Array

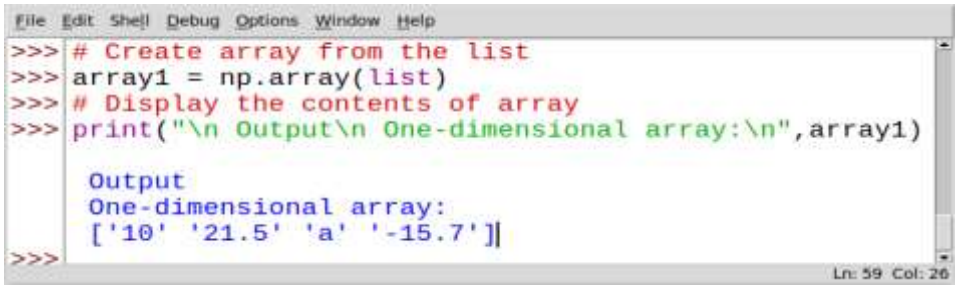
An array with only single row of elements is called 1-D array. Let us try to create a 1-D array from a list which contains numbers as well as strings.

**Example:** Create a one-dimensional array from a list

```
# Create a one-dimensional array from a list
list = [10,21.5,'a',-15.7]
# Create array from the list
array1 = np.array(list)
# Display the contents of array
print("\n Output\n One-dimensional array:\n",array1)
```

**Output**

```
One-dimensional array:
['10' '21.5' 'a' '-15.7']
```




Observe that since there is a string value in the list, all integer and float values have been promoted to string, while converting the list to array.

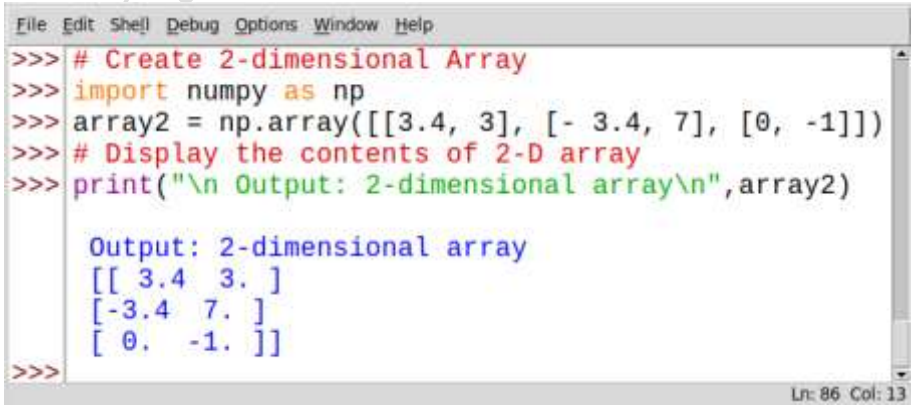
**Note:** A common mistake occurs while passing argument to array() if we forget to put square brackets. Make sure only a single argument containing list of values is passed.

```
# incorrect way
>>> a = np.array(1,2,3,4)
# correct way
>>> a = np.array([1,2,3,4])
```

### 1.3.3 Creating a 2-D Array

We can create a two dimensional (2-D) arrays by passing nested lists to the array() function.

```
# Create 2-dimensional Array
import numpy as np
array2 = np.array([[3.4, 3], [- 3.4, 7], [0, -1]])
# Display the contents of 2-D array
print("\n Output: 2-dimensional array\n",array2)

 Output: 2-dimensional array
 [[ 3.4  3. ]
 [-3.4  7. ]
 [ 0.  -1. ]]
```



## 1.3 ATTRIBUTES OF NUMPY ARRAY

Some important attributes of a NumPy ndarray object are:

### 1.3.1 Shape and Size Properties

**ndarray.shape:** Returns a tuple representing the dimensions of the array such as rows and columns. *Example 1.13* illustrates to print the shape of the array.

```
>>> a = np.array([[1, 2], [3, 4], [5, 6]])
>>> print(a.shape)
(3, 2)
```

The output (3, 2) indicates that the array consists of 3 rows and 2 columns. This is the shape of an array.

**ndarray.ndim:** It gives the number of dimensions of the array as an integer value. Arrays can be 1-D, 2-D or n-D. NumPy calls the dimensions as axes. Thus, a 2-D array has two axes. The row-axis is called axis-0 and the column-axis is called axis-1. The number of axes is also called the array's rank. ***Example 1.1*** illustrates to print the dimension of the array.

Returns the number of dimensions (axes) of the array.

```
>>> a1 = np.array([1, 2, 3])
>>> print(a1.ndim)
1
```

The output 1 indicates the array of one dimension.

```
>>> a2 = np.array([[1, 2], [3, 4], [5, 6]])
>>> print(a2.ndim)
2
```

The output 2 indicates the array of two dimension.

**ndarray.size:** Returns the total number of elements in the array. Following example illustrates the total number of elements in the array. ***Example 1.2*** illustrates to print the size of array elements in bytes.

```
>>> a = np.array([[1, 2], [3, 4]])
>>> print(a.size)
4
```

The output 4 indicates that there are a total 4 elements in the array.

**ndarray.itemsize:** Returns the size (in bytes) of each element in the array. Data type int32 and float32 means each element of the array occupies 32 bits in memory. Thus, an array of elements of type int32 has itemsize 32/8=4 bytes. Likewise, int64/float64 means each item has itemsize 64/8=8 bytes. ***Example 1.3*** illustrates to print the size of array elements in bytes.

```
>>> a = np.array([1, 2, 3], dtype=np.int32)
>>> print(a.itemsize) # memory allocated to integer type
4
```

The output 4 indicates the size of array elements is 32 bits means 4 bytes.

```
>>> a = np.array([1.2, 2.4, 3.6], dtype=np.float64)
>>> print(a.itemsize) # memory allocated to float type
8
```

The output 8 indicates the size of array elements is 64 bits means 8 bytes.

**ndarray.nbytes :** Returns the total memory consumed by the array (in bytes). ***Example 1.4*** illustrates to print the total memory consumed by the array (in bytes).

```
>>> a = np.array([1, 2, 3])
>>> print(a.nbytes)
24
```

The output 24 indicates the total 24 bytes of memory consumed by the array. (assuming 64-bit integers)

### 1.3.2. Data Type Properties

**ndarray.dtype:** Returns the data type of the elements in the array. All the elements of an array are of same data type. Common data types are int32, int64, float32, float64, U32, etc.

```
>>> a = np.array([1, 2, 3])
>>> print(arr.dtype)
int64
```

The output 64 indicates the data type of the elements in the array is 64 bits integer.

**`ndarray.astype(dtype)`:** Allows conversion of array elements to a specified type. ***Example 1.5*** illustrates to convert the data type of the array elements to the specified data type.

```
>>> a = np.array([4.3, 5.4])
>>> int_a = a.astype(int)
>>> print(int_a)
[4 5]
```

The output [4 5] shows that the float elements of array are converted into integer values.

### 1.3.3. Reshaping and Views

Reshaping means changing the shape of an array. The shape of an array is the number of elements in each dimension. Reshaping can change the shape of an array. It is possible to add or remove dimensions or change the number of elements in each dimension, provided the number of elements should be of exact count.

**`ndarray.reshape(shape)`:** Returns a new array with the same data but a different shape. ***Example 1.6*** illustrates to reshape the array of the same data with different shapes.

```
>>> a = np.array([1, 2, 3, 4])
>>> reshaped = a.reshape(2, 2)
>>> print(reshaped)
[[1 2]
```

[3 4]]

**`ndarray.ravel()`:** Flattens the array into a 1D array. ***Example 1.7*** illustrates to convert the 2D array to a 1D array with the same elements.

```
>>> a = np.array([[1, 2], [3, 4]])
>>> print(a.ravel())
[1 2 3 4]
```

The output [1 2 3 4] shows that the 2D array is converted to 1D array.

**Mathematical and Logical Properties**

NumPy arrays enable element-wise operations:

```
>>> a = np.array([1, 2, 3])
>>> print(a * 2)
[2 4 6]
```

The output shows that each element of the array is multiplied by 2.

### 1.3.4 Broadcasting

Arrays can operate with different shapes using **broadcasting** rules.

```
>>> a1 = np.array([1, 2, 3])
>>> a2 = np.array([[1], [2], [3]])
>>> a = a1 + a2
>>> print (a)
[[2 3 4]
 [3 4 5]
 [4 5 6]]
```

The output shows that each element of array a1 is added to all the elements of array a2 to form the new array.

### 1.3.5 Mathematical and Boolean Properties

NumPy arrays enable element-wise operations:

```
>>> a = np.array([1, 2, 3])
>>> print(a * 2)
[2 4 6]
```

The output shows that each element of the array is multiplied by 2.

**ndarray.all():** Returns True if all elements are non-zero or True.

```
>>> a = np.array([1, 2, 3])
>>> print(a.all())
True
```

The output shows that all the elements array are non-zero.

**ndarray.any():** Returns True if at least one element is non-zero or True.

```
>>> a = np.array([0, 0, 1])
>>> print(a.any())
```

True

The output shows that at least one element of the array is non-zero

### 1.3.6 Copy vs View

**ndarray.view() :** Creates a new array object with shared data.

**ndarray.copy() :** Creates a new array with a copy of the data.

These properties make NumPy arrays efficient and versatile for numerical and scientific computing.

## 1.4 ACCESSING ARRAY ELEMENTS

Accessing elements within NumPy arrays is a fundamental operation, offering various techniques to extract or modify data. Indexing and Slicing are the basic methods to acces elements of NumPy arrays.

### 1.4.1 Indexing

We have learnt about indexing single-dimensional array in section 6.2. For 2-D arrays indexing for both dimensions starts from 0, and each element is referenced through two indexes i and j, where i represents the row number and j represents the column number.

**Basic indexing:** Individual elements are accessed using square brackets [] and their corresponding index. Remember that NumPy uses zero-based indexing, so the first element has an index of 0, the second element has an index of 1, and so on. Following example illustrates to access elements of one-dimensional array.

**Example:** Accessing elements of 1D array

```
# Accessing elements of 1D array
import numpy as np
# Access third element of array
arr = np.array([10, 20, 30, 40, 50])
print("Output:)
print("First element of array:", arr[0])
print("Third element of array:", arr[2])

Output:

First element of array: 10

Third element of array: 30
```

**Multi-dimensional arrays:** Elements in 2D and higher dimensional arrays are accessed by providing multiple indices, separated by commas, representing the dimension and the index within that dimension.

**Example:** Accessing elements of 2D array

```
# Accessing elements of 2D array
```

```
import numpy as np
matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print("Output:")
print("\n Element in the first row, second column:", matrix[0,1])
print("\n Element in the second row, third column:", matrix[1,2])

Output:

Element in the first row, second column: 2

Element in the second row, third column: 6
```

**Example:** Accessing elements of 3D array

```
# Accessing elements of 3D array
import numpy as np
cube = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print("Output:")
print("\nFirst 2D array, first row, first column:",cube[0,0,0])
print("\nSecond 2D array, second row, second column:",cube[1,1,1])

 Output:

 First 2D array, first row, first column: 1

 Second 2D array, second row, second column: 11
```

**Negative indexing:** Access elements from the end of the array using negative indices, starting with -1 for the last element.

**Example:** Access the elements of array using negative indexing.

```
# Accessing elements array using negative indexing.
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print("Output:")
print("First element from end:",arr[-1])
print("Third element from end:",arr[-3])

Output:
First element from end: 5
Third element from end: 3
```

### 1.4.2. Slicing

Sometimes we need to extract part of an array. This is done through slicing. Slicing extracts a subset of elements from an array using the format start:stop:step.

**Example:** Accessing elements using slicing

```
# Accessing elements using slicing
import numpy as np
arr = np.array([10, 20, 30, 40, 50])
# Elements from index 1 to 4)
print("Output:")
print("Elements from index 1 to 4:",arr[1:4])

Output:
Elements from index 1 to 4: [20 30 40]
```

**Multi-dimensional arrays:** Apply slicing to each dimension separately.

```
# Accessing elements of 2D array using slicing
import numpy as np
matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
# Elements from first two rows, second and third columns
print("Output:")
print("Original matrix:\n",matrix[0:8])
print("Elements from first two rows, second and third
columns:\n",matrix[0:2, 1:3])

Output:
Original matrix:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
Elements from first two rows, second and third columns:
 [[2 3]
 [5 6]]
```

**Accessing rows or columns:** Use slicing with a single colon (:) to select all elements along a dimension. Example illustrates the slicing of 2D array.

```
# Accessing only rows or column of 2D array using slicing
import numpy as np
matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
# Elements from first two rows, second and third columns
```

```
print("Output:")
print("Original matrix:\n",matrix[0:8])
print("Elements of second row:\n",matrix[1,:])
print("Elements of third column:\n",matrix[:,2])

Output:
Original matrix:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
Elements of second row:
 [4 5 6]
Elements of third column:
 [3 6 9]
```

### 1.5 OTHER WAYS OF CREATING NUMPY ARRAYS

Creating NumPy arrays from a Python list is a common method. There are many other ways to create arrays without relying on lists.

#### 1.5.1 Intrinsic NumPy array creation functions

NumPy offers specialized functions that generate arrays with specific initial values or properties, without requiring you to manually define the elements in a list first.

We can specify data type (integer, float, etc.) while creating array using dtype as an argument to array(). This will convert the data automatically to the mentioned type. In the following example, nested list of integers are passed to the array function. Since data type has been declared as float, the integers are converted to floating point numbers.

```
# Creating array of integers
import numpy as np
array = np.array([[1,2],[3,4]])
print("\n Output: Original array of integers\n",array)

Output: Original array of integers
 [[1 2]
 [3 4]]

# Converting array of integers converted to float
array = np.array([[1,2],[3,4]], dtype=float)
print("\n Output: Array of integers converted to float\n",array)

Output: Array of integers converted to float
 [[1. 2.]
 [3. 4.]]
```

**np.zeros(shape, dtype):** Creates an array with all elements initialised to 0 using the function zeros(). You provide the desired shape as a tuple and can optionally specify the dtype (e.g., int, float). By default, the data type of the array created by zeros() is float. The following code will create an array with 2 rows and 3 columns with each element set to 0.

```
# Creating array of zeros of float by default
import numpy as np
zeros_array = np.zeros((2, 3))
print("\nOutput: Array of zero by default \n",zeros_array)
Output: Array of zero by default
 [[0. 0. 0.]
 [0. 0. 0.]]

# Creating array of zeros of integer
zeros_array = np.zeros((2, 3),dtype=int)
print("\nOutput: Array of zero with integer \n",zeros_array)

Output: Array of zero with integer
 [[0 0 0]
 [0 0 0]]
```

**np.ones(shape, dtype):** Similar to np.zeros(), we can create an array with all elements initialised to 1 using the function ones(). By default, the data type of the array created by ones() is float. The following code will create an array with 3 rows and 2 columns.

```
# Creating array of Ones(1)
import numpy as np
ones_array = np.ones((3,2))
print("\nOutput: Array of Ones \n",ones_array)

Output: Array of Ones
 [[1. 1.]
 [1. 1.]
 [1. 1.]]
```

**np.full(shape, fill_value, dtype):** Generates an array with the specified shape and fills it with a constant fill_value. The following code will create an array with 3 rows and 2 columns with vaue of all the elements to 5.

```
# Creating array with fill value
import numpy as np
full_array = np.full((3, 2), 5)
print("\nOutput: Array with fill value 5 \n",full_array)
```

```
Output: Array with fill value 5
 [[5 5]
 [5 5]
 [5 5]]
```

**np.arange(start, stop, step, dtype):** Similar to Python's range() function, this creates an array with evenly spaced values within a given interval. The stop value is excluded. The following code will create an array from 0 to 10 with range 2. Note that the stop value 10 is excluded.

```
# Creating array with range
import numpy as np
range_array = np.arange(0, 10, 2)
print("\nOutput: Array with range 2 \n",range_array)

Output: Array with range 2
 [0 2 4 6 8]
```

**np.linspace(start, stop, num, endpoint, retstep, dtype):** Generates an array of evenly spaced numbers over a specified interval. You define the start and stop points, and the desired num (number of elements) in the array. The stop value is included by default.

```
# Creating array with linespace
import numpy as np
linear_array = np.linspace(0, 2, 5)
print("\nOutput: Array with linespace \n",linear_array)

Output: Array with linespace
 [0.  0.5 1.  1.5 2. ]
```

### 1.5.2 Generating arrays using random numbers

NumPy includes functions within the numpy.random module to create arrays filled with random values from different distributions.

**np.random.rand(d0, d1, ..., dn):** Creates an array of the specified shape and fills it with random values sampled from a uniform distribution over [0, 1).

**np.random.randn(d0, d1, ..., dn):** Creates an array of the specified shape and fills it with random values sampled from a standard normal distribution.

**np.random.randint(low, high, size, dtype):** Generates an array of the specified size filled with random integers within the range [low, high)

## 1.6 OPERATIONS ON ARRAYS

NumPy, provides a powerful and efficient way to perform various operations on arrays. These operations are faster than using traditional Python lists, especially when dealing with large datasets.

Some of the common operations you can perform on NumPy arrays are:

### 1.6.1 Arithmetic Operations

The basic arithmetic operation like addition, subtraction, multiplication, division can be performed on NumPy arrays. These operations are performed on on each corresponding pair of elements. For instance, adding two arrays will result in the first element in the first array to be added to the first element in the second array, and so on.

It is important to note that for element-wise operations, size of both arrays must be same. That is, array1.shape must be equal to array2.shape.

**Addition:** You can add two arrays or an array and a scalar using the + operator or np.add() function.

```
# Addition of two arrays
import numpy as np
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
result = arr1 + arr2
print("Output")
print("Array1 is :", arr1)
print("Array2 is :", arr2)
print("Addition of two arrays using + operator")
print(result)
add = np.add(arr1, arr2)
print("Addition of two arrays using add() function")
print(add)

Output
Array1 is : [1 2 3]
Array2 is : [4 5 6]
Addition of two arrays using + operator
[5 7 9]
Addition of two arrays using add() function
[5 7 9]
```

**Subtraction:** Subtract two arrays element-wise using the – operator or np.subtract() function.

```
# Subtraction of two arrays
import numpy as np
arr1 = np.array([11, 22, 33])
arr2 = np.array([10, 20, 30])
result = arr1 - arr2
print("Output")
print("Array1 is :", arr1)
print("Array2 is :", arr2)
```

```
print("Array1 – Array2:",result)

Output
Array1 is : [11 22 33]
Array2 is : [10 20 30]
Array1 - Array2: [1 2 3]
```

**Multiplication:** Perform element-wise multiplication using the * operator or np.multiply() function.

```
# Multiplication of two arrays
import numpy as np
arr1 = np.array([1, 2, 3])
arr2 = np.array([11, 22, 33])
result = arr1 * arr2
print("Output")
print("Array1 is :", arr1)
print("Array2 is :", arr2)
print("Array1*Array2:",result)

Output
Array1 is : [1 2 3]
Array2 is : [11 22 33]
Array1*Array2: [11 44 99]
```

**Division:** Divide two arrays element-wise using the / operator or np.divide() function.

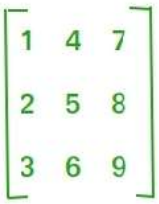
```
# Division of two arrays
import numpy as np
arr1 = np.array([11, 22, 33])
arr2 = np.array([10, 20, 30])
result = arr1/ arr2
print("Output")
print("Array1 is :", arr1)
print("Array2 is :", arr2)
print("Array1/Array2:",result)

Output
Array1 is : [11 22 33]
Array2 is : [10 20 30]
Array1/Array2: [1.1 1.1 1.1]
```

**Other arithmetic operations:** NumPy also supports other element-wise operations like exponentiation (** operator or np.power()), modulo (% operator or np.mod()), and floor division (// operator).

### 1.6.2 Transpose

Transposing an array turns its rows into columns and columns into rows just like matrices in mathematics as shown below.


Input




Output

```
# Transpose array
import numpy as np
arr = np.array([[1,2,3],[4,5,6],[7,8,9]])
print("Output”)
print("Original Array:\n",arr)
arr_t = arr.transpose()
print("Transpose of Array:\n",arr_t)

Original Array:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
Transpose of Array:
 [[1 4 7]
 [2 5 8]
 [3 6 9]]
```

### 1.6.3 Sorting

Sorting is to arrange the elements of an array in hierarchical order either ascending or descending. By default, numpy does sorting in ascending order.

**np.sort():** This function returns a sorted copy of the array without modifying the original. By default, np.sort() sorts elements in ascending order.

```
# Sorting of array
import numpy as np
arr = np.array([1,0,2,-3,6,8,4,7])
print("Output")
print("Original Array:\n",arr)
sorted_arr = np.sort(arr)
print("Sorted Array:\n",sorted_arr)

Output
Original Array:
 [ 1  0  2 -3  6  8  4  7]
```

```
Sorted Array:
 [-3  0  1  2  4  6  7  8]
```

In 2-D array, sorting can be done along either of the axes i.e., row-wise or column-wise. By default, sorting is done row-wise (i.e., on axis = 1). It means to arrange elements in each row in ascending order. When axis=0, sorting is done column-wise, which means each column is sorted in ascending order.

```
# Sorting of 2-D array
import numpy as np
ar = np.array([[10,-7,0, 20],
[-5,1,200,40],[30,1,-1,4]])
print("Output")
print("Original Array:\n",ar)
sorted_ar_r = np.sort(ar)
print("Row-wise sorted array(default):\n",sorted_ar_r)
sorted_ar_c = np.sort(ar,axis=0)
print("Column-wise sorted array:\n",sorted_ar_c)

Output
Original Array:
 [[ 10  -7   0  20]
 [ -5   1 200  40]
 [ 30   1  -1   4]]
Row-wise sorted array(default):
 [[ -7   0  10  20]
 [ -5   1  40 200]
 [ -1   1   4  30]]
Column-wise sorted array:
 [[ -5  -7  -1   4]
 [ 10   1   0  20]
 [ 30   1 200  40]]
```



## Summary

- NumPy provides fast, memory-efficient arrays compared to Python lists.
- Arrays are homogeneous (all elements of the same type).
- Students learn to create 1-D, 2-D, and multi-dimensional arrays.
- Key attributes: shape, size, dimensions (ndim), data type (dtype), memory usage (itemsize, nbytes).
- Arrays can be generated using lists, functions (ones, zeros, full, arange, linspace), and random values.
- Arrays support reshaping, flattening, broadcasting, and arithmetic operations.

## Check Your Progress

**A. Multiple choice questions**

1. Which of the following is not true for NumPy (a) NumPy process the arrays faster than the List (b) NumPy is the library package for scientific computing in Python (c) NumPy stands for Number Python (d) NumPy can process the multidimensional array (e) NumPy is used in Deep Learning or Machine Learning models
2. Which of the following is not true for NumPy code (a) It is more concise and easier to read (b) It contains fewer lines of codes therefore there are fewer bugs (c) It can be easily integrated with other scientific libraries of Python (d) It is difficult to understand and code
3. Which of the following Python library is similar to Pandas (a) NPy (b) RPy (c) NumPy (d) None of the mentioned above
4. Which of the following is a correct syntax to check the data type of an array? (a) arr.dtype (b) arr.ntype (c) arr.datatype (d) arr.type
5. Which of the following is a correct syntax to return the shape of an array (a) arr.shape() (b) shape(arr) (c) arr.shape (d) None of the above
6. Which of the following is a correct syntax to create an array of type float. (a) arr=np.array([1,2,3,4],dtype='f') (b) arr=np.array([1,2,3,4].toFloat() (c) arr=np.float([1,2,3,4]) (d) None of the above
7. Which of the following is syntax to create a NumPy array? (a) np.array([1,2,3,4,5]) (b) np.object([1,2,3,4,5]) (c) np.createArray([1,2,3,4,5]) (d) None of the above
8. Which of the following arrays is a two dimensional (2-D) array? (a) [1,2,3,4,5] (b) 42 (c) [[1,2,3],[4,5,6]] (d) None of the above
9. Which of the following is a correct syntax to check the number of dimensions in an array? (a) arr.dim (b) arr.ndim() (c) arr.ndim (d) arr.dim
10. What is a correct syntax to print the first item of an array? (a) print)arr[0]) (b) print(arr,1) (c) print(arr[1]) (d) (d) None of the above

**Suppose the array is given as: arr = np.array([1,2,3,4,5,6,7])**

11. What is a correct syntax to print the numbers [3, 4, 5] (a) print(arr[2:6]) (b) print(arr[2:4]) (c) print(arr[3:6]) (d) print(arr[2:5])
12. What is a correct syntax to print the last 4 numbers (a) print(arr[4]) (b) print(arr:[4]) (c) print(arr[4:]) (d) print(arr[3:])
13. What is a correct syntax to print every other item from the array (a) print(arr[1:3:5:7]) (b) print(arr(0: step = 2)) (c) print(arr[::2])
14. What is a correct syntax to return the index of all items that has the value 4 from the array : arr = np.array([1,4,3,4,5,4,4]) (a) arr.search(4) (b) arr.where() (c) np.where(arr == 4) (d) None of the above
15. What is a correct syntax to mathematically add the numbers of arr1 to the numbers of arr2? (a) sum(arr1, arr2) (b) np.add(arr1,arr2) (c) np.append (arr1, arr2) (d) None of the above

**B. Fill in the blanks**

1. The array object in NumPy is called _____ .
2. NumPy ndarray objects can be created by using _____
3. NumPy one dimensional array is known as _____ and multidimensional arrays is known as _____.
4. The _____ attribute of NumPy is used to check number of dimensions the array.
5. The _____ is the number of elements in each dimension.
6. ndarray.all() returns _____ if all elements are non-zero
7. ndarray.any() returns True if at least one element is _____.
8. The index of the array when referred from the end by using the minus operator, then it is _____
9. ndarray.shape, returns a tuple _____ representing the dimensions of the array.
10. ndarray.ravel() flattens the array into a _____.

**C. State whether true or false**

1. NumPy for data analysis.
2. NumPy can have n dimensional arrays of homogeneous data elements.
3. NumPy can be installed only by using the Anaconda toolbox.
4. NumPy is a main object for the creation of homogeneous multidimensional arrays.
5. Pip is a standard package management system used to install and manage the software packages written in Python
6. The view is not affected by the changes made to the original array.
7. The copy is not affected by the changes made to the original array.
8. NumPy arrays support advanced slicing and indexing.
9. ndarray.ndim, returns the number of axes of the array.
10. ndarray.nbytes returns the total number of bits consumed by the array

**D. Answer the following questions in short**

1. Lists the various operations that can be performed by using NumPy.
2. Compare NumPy array with Python List.
3. List the data types supported by NumPy.
4. List the properties of NumPy array.

## Session 2. Array Manipulation using NumPy

Imagine organizing clothes in your cupboard. Sometimes you put them together in one pile (joining), sometimes you separate them into different shelves (splitting), sometimes you fold them in a different style (reshaping), and sometimes you make space by resizing. Similarly, NumPy allows us to join, split, reshape, or resize arrays to manage data better.


Fig. 2.1: using NumPy array manipulation

In this session, you will learn different ways to manipulate arrays — including joining (concatenate, vstack, hstack, dstack), splitting (split, array_split, vsplit, hsplit, dsplit), reshaping arrays into new forms, flattening multi-dimensional arrays into 1-D, resizing arrays (expanding or truncating), and changing dimensions (adding or removing axes).

### 2.1 INTRODUCTION

NumPy is a foundation library for scientific computations in Python. It contains sophisticated functions and tools for integrating with other programming languages as well. During data analysis, it is widely used to handle arrays as it offers a powerful n-dimensional array object, faster than a traditional list in Python. In this session, we will discuss different array manipulation techniques.

### 2.2 JOINING AND SPLITTING ARRAYS

Joining and splitting arrays are fundamental operations in NumPy. Joining merges multiple arrays into one. NumPy provides several functions to join arrays along specified axes.

#### 2.2.1 Joining of Arrays

Concatenation means joining two or more arrays. Concatenating 1-D arrays means appending the sequences one after another.

**np.concatenate() :** This function is used to join two or more arrays along an existing axis. All the dimensions of the arrays to be concatenated must match exactly except for the dimension or axis along which they need to be joined. Any mismatch in the dimensions results in an error. By default, the concatenation of the arrays happens along axis=0. *Example 2.1* illustrates the joining of two arrays.

***Example 2.1:*** Concatenation of two Arrays along rows

```
# Concatenation of Arrays along rows
import numpy as np
a1=np.array([[1,2],[3,4]])
a2=np.array([[5,6],[7,8]])
# Join arrays along rows (axis=0)
a3 = np.concatenate((a1,a2), axis=0)
print ("Output")
print ("Array 1:\n",a1)
print ("Array 2:\n",a2)
print ("Arrays after joining along row\n",a3)

Output
Array 1:
 [[1 2]
 [3 4]]
Array 2:
 [[5 6]
 [7 8]]
Result after joining Array1 & Array2
 [[1 2]
 [3 4]
 [5 6]
 [7 8]]
```

$$a1 = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$a2 = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

$$result = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{bmatrix}$$

In the output, observe that array a1 and a2 are joined together to produce a resultant array consisting of all elements of array a1 and a2. Observe that when axis=0 then joining will be a long row as illustrated in the *Example 2.2*.

***Example 2.2:*** Concatenation of two Arrays along columns

```
# Join arrays along columns (axis=1)
a4 = np.concatenate((a1,a2), axis=1)
print ("Arrays after joining along column\n",a4)
```

```
Arrays after Joining along column
 [[1 2 5 6]
 [3 4 7 8]]
```

Observe that when axis=1 then arrays are joined along columns.

$$a1 = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$a2 = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

$$result = \begin{bmatrix} 1 & 2 & 5 & 6 \\ 3 & 4 & 7 & 8 \end{bmatrix}$$

```
File Edit Shell Debug Options Window Help
>>> # Join along columns (axis=1)
>>> result = np.concatenate((a1, a2), axis=1)
>>> print (result)
[[1 2 5 6]
 [3 4 7 8]]
>>>
                                        Ln: 50 Col: 0
```

**2. Vertical Stack**

**np.vstack() :** This function is used to stack arrays vertically, that is, row-wise. This is equivalent to concatenation along the first axis after 1-D arrays of shape *(N,)* have been reshaped to *(1,N)*.

***Example 2.3:***

```
# Vertical Stack
>>> a1 = np.array([1, 2])
>>> a2 = np.array([3, 4])
>>> result = np.vstack((a1, a2))
>>> print (result)
[[1 2]
 [3 4]]
```

In *Example 2.3,* observe that the array a1 and a2 are stacked vertically.

$$a1 = [1 \quad 2]$$

$$a2 = [3 \quad 4]$$

$$result = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
File Edit Shell Debug Options Window Help
>>> # Vertical Stack
>>> a1 = np.array([1, 2])
>>> a2 = np.array([3, 4])
>>> result = np.vstack((a1, a2))
>>> print (result)
[[1 2]
 [3 4]]
>>>
Ln: 37 Col: 0
```

**3. Horizontal Stack**

**np.hstack() :** This function is used to stack arrays horizontally, means, column-wise. This is equivalent to concatenation along the second axis, except for 1-D arrays where it concatenates along the first axis. *Example 2.4* illustrates how to stack the array horizontally.

***Example 2.4:***

```
# Horizontal Stack
>>> a1 = np.array([1, 2])
>>> a2 = np.array([3, 4])
>>> result = np.hstack((a1, a2))
>>> print (result)
[1 2 3 4]
```

In this output, you can observe that array a1 and a2 are stacked horizontally.

$a1 = [1 \;\; 2]$

$a2 = [3 \;\; 4]$

$result = [1 \;\; 2 \;\; 3 \;\; 4]$

```
File Edit Shell Debug Options Window Help
>>> # Horizontal Stack
>>> a1 = np.array([1, 2])
>>> a2 = np.array([3, 4])
>>> result = np.hstack((a1, a2))
>>> print (result)
[1 2 3 4]
>>>
Ln: 58 Col: 0
```

**4. Depth Stack**

**np.dstack() :** This function is used to stack arrays along the third dimension, that is, depth-wise. This is equivalent to concatenation along the third axis after 2-D arrays of shape *(M, N)* have been reshaped to *(M, N,1)* and 1-D arrays of shape *(N,)* have been reshaped to *(1, N,1)*. ***Example 2.5*** illustrates how to stack the array depth-wise.

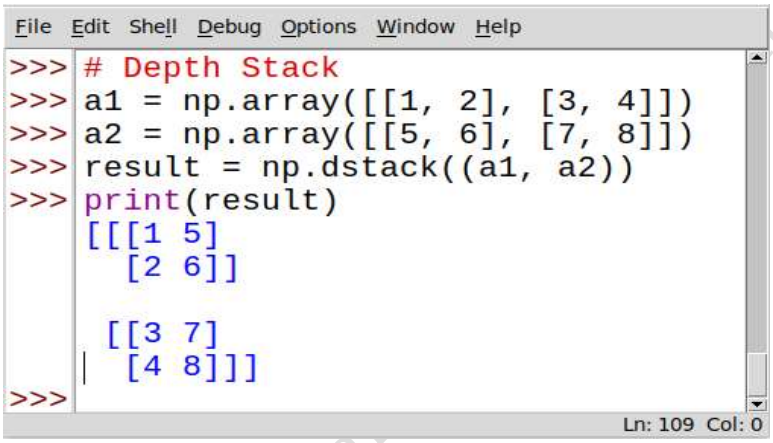
***Example 2.5:***

```
# Depth Stack
>>> a1 = np.array([[1, 2], [3, 4]])
>>> a2 = np.array([[5, 6], [7, 8]])
>>> result = np.dstack((a1, a2))
>>> print(result)
[[[1 5]
  [2 6]]
 [[3 7]
  [4 8]]]
```

$$a1 = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$a2 = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

$$result = \begin{bmatrix} 1 & 5 \\ 2 & 6 \\ 3 & 7 \\ 4 & 8 \end{bmatrix}$$




**5. Column Stack**

**np.column_stack() :** This function stacks 1D arrays as columns into a 2D array. Take a sequence of 1-D arrays and stack them as columns to make a single 2-D array. 2-D arrays are stacked as-is, just like with hstack. 1-D arrays are turned into 2-D columns first. ***Example 2.6*** illustrates how to stack the array in a column.

***Example 2.6:***

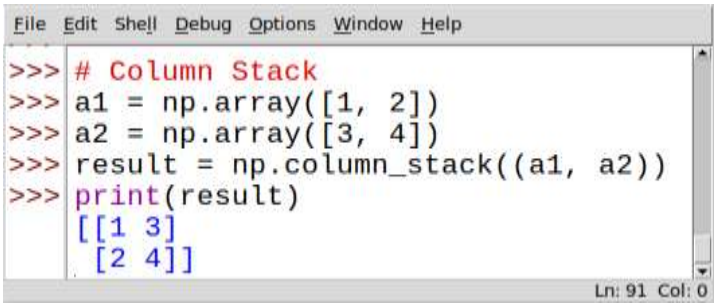
```
# Column Stack
>>> a1 = np.array([1, 2])
>>> a2 = np.array([3, 4])
>>> result = np.column_stack((a1, a2))
>>> print(result)
[[1 3]
 [2 4]]
```

Observe that in *Example 2.6,* row of array a1 and a2 is inserted as column in result. Result is a two-dimensional array.

$$a1 = [1 \quad 2]$$

$$a2 = [3 \quad 4]$$

$$\text{result} = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$




**6. Row Stack**

**np.row_stack() :** This function stacks 1D arrays as rows into a 2D array. It is similar to vstack. It stack arrays in sequence vertically, that is, row wise. This is equivalent to concatenation along the first axis after 1-D arrays of shape *(N)* have been reshaped to *(1,N)*. ***Example 2.7*** illustrates to stack the array in a column.

***Example 2.7***

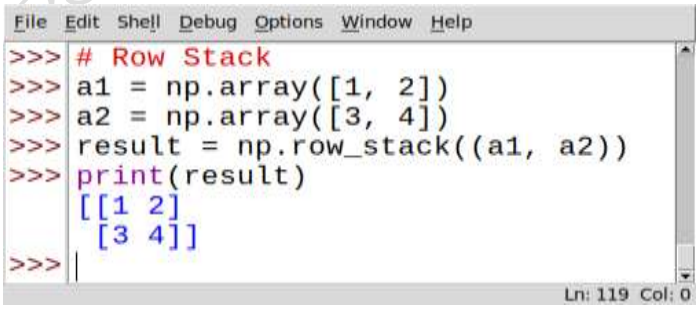
```
# Row Stack
>>> a1 = np.array([1, 2])
>>> a2 = np.array([3, 4])
>>> result = np.row_stack((a1, a2))
>>> print(result)
[[1 2]
 [3 4]]
```

Observe that one dimensional arrays a1 and a2 are stacked as rows in result. Result is a two-dimensional array.

$$a1 = [1 \quad 2]$$

$$a2 = [3 \quad 4]$$

$$\text{result} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$



### 2.2.2 Splitting Arrays

Splitting is the reverse operation of Joining. Joining merges multiple arrays into one and Splitting breaks one array into multiple. NumPy provides functions to split arrays into multiple sub-arrays.

**1. Split an Array**

**np.split() :** This function is used to Split an array into multiple sub-arrays along a specified axis. Its general format is: numpy.split(ary, indices_or_sections, axis=0).

If indices_or_sections are an integer, N, the array will be divided into N equal arrays along the axis. If such a split is not possible, an error is raised.

If indices_or_sections are a 1-D array of sorted integers, the entries indicate where along the axis the array is split. ***Example 2.8*** illustrates the splitting of array.

***Example 2.8***

```
# Split an Array
>>> a = np.array([1, 2, 3, 4, 5, 6])
>>> # Split into 3 equal parts
>>> result = np.split(a, 3)
>>> print(result)
[array([1, 2]), array([3, 4]), array([5, 6])]
```

Observe that in the above example are is split into three equal parts. Each part contains two elements. If the array cannot be split equally, it raises an error.

$$a = [1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6]$$

$$\text{result} = [[1 \quad 2] \; [3 \quad 4] \; [5 \quad 6]]$$

**2. Unequal Splitting**

**np.array_split() :** This function splits an array into multiple sub-arrays but allows unequal splitting. It's general format is: numpy.array_split(ary, indices_or_sections, axis=0).

This function allows indices or sections to be an integer that does not equally divide the axis. For an array of length l that should be split into n sections, it returns l % n sub-arrays of size l//n + 1 and the rest of size l//n.

***Example 2.9*** illustrates the splitting of array.

```
# Splitting of array
>>> a = np.array([1, 2, 3, 4, 5])
>>> # Split into 3 parts (unequal)
>>> result = np.array_split(a, 3)
>>> print(result)
[array ([1, 2]), array ([3, 4]), array ([5])]
```

Observe that the given array contains five elements. It is splitted into three arrays. The first two arrays contain two elements each and the third array contain only one element.

$$a = [1 \quad 2 \quad 3 \quad 4 \quad 5]$$

$$\text{result} = [[1 \quad 2] \; [3 \quad 4] \; [5]]$$

**3. Vertical Split**

**np.vsplit() :** This function splits an array vertically, meaning, row-wise. It works on arrays with at least 2 dimensions. This is equivalent to split with axis=0 by default. The array is always split along the first axis regardless of the array dimension.

***Example 2.10*** illustrates the vertical splitting of array.

```
# Vertical splitting of array.
>>> a = np.array([[1, 2], [3, 4], [5, 6]])
>>> result = np.vsplit(a, 3)
>>> print(result)
[array([[1, 2]]), array([[3, 4]]), array([[5, 6]])]
```

Observe that the array a is vertically splitted into three arrays each containing two elements.

$$a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

$$\text{result} = [[1 \quad 2] \; [3 \quad 4] \; [5 \; 6]]$$

**4. Horizontal Split**

**np.hsplit() :** This function splits an array horizontally, meaning column-wise. It works on arrays with at least 2 dimensions. It is equivalent to split with axis=1, the array is always split along the second axis except for 1-D arrays, where it is split at axis=0.

***Example 2.11*** illustrates the horizontal splitting of array.

```
# Horizontal splitting of array
>>> a = np.array([[1, 2, 3], [4, 5, 6]])
>>> result = np.hsplit(a, 3)
>>> print(result)
[array([[1],
       [4]]), array([[2],
       [5]]), array([[3],
       [6]])]
```

Observe that array a is split into three arrays horizontally.

$$a = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

$$\text{result} = \begin{bmatrix} 1 & \\ 4 & 2 \\ 5 & 3 \\ 6 & \end{bmatrix}$$

**5. Depth Split**

**np.dsplit() :** This function splits an array along the third dimension, that is, depth-wise. It works on arrays with at least 3 dimensions. It is equivalent to split with axis=2, the array is always split along the third axis provided the array dimension is greater than or equal to 3. ***Example 2.12*** illustrates the splitting of array in depth.

```
# Splitting of array in depth
>>> a = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
>>> result = np.dsplit(a, 2)
>>> print(result)
[array([[[1],
         [3]],
        [[5],
         [7]]]), array([[[2],
         [4]],
        [[6],
         [8]]])]
```

Observe that the array a is a three-dimensional array. It is split into two arrays.

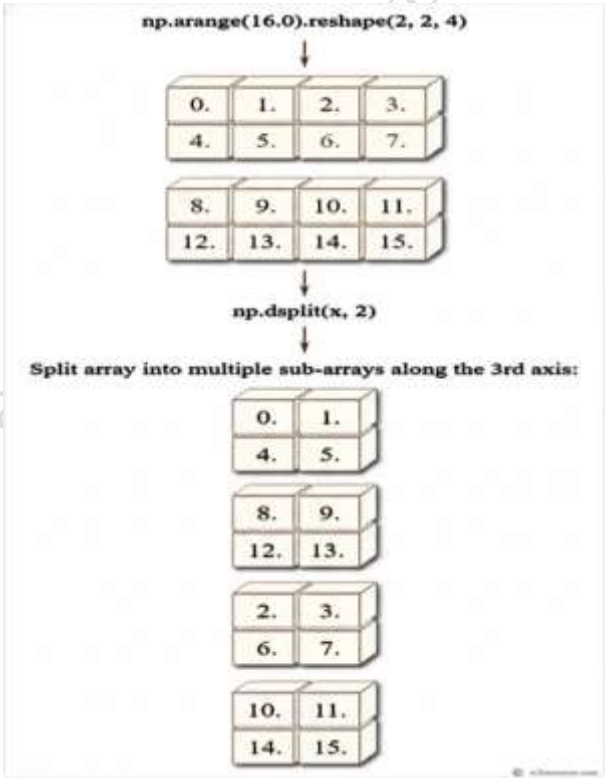



Fig. 2.2: three-dimensional array

## 2.3 RESHAPING ARRAYS

Reshaping means changing the shape of an array. The shape of an array is the number of elements in each dimension. By reshaping we can add or remove dimensions or change the number of elements in each dimension.

**Reshape Function**

**np.reshape() :** This function creates a new array with the same data but a different shape. The total number of elements must remain the same. You can specify one dimension as -1, and NumPy will infer its value.

**The general format is:**

numpy.reshape (a, /, shape=None, order='C', *, newshape=None, copy=None).

**Syntax**

```
np.reshape(array, new_shape)
```

***Example 2.13*** illustrates the reshapping of array.

```
# Reshapping of array
>>> import numpy as np
>>> a = np.array([1, 2, 3, 4, 5, 6])
>>> # Reshape into a 2x3 array
>>> reshaped = np.reshape(a, (2, 3))
>>> print(reshaped)
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

```
>>> # Infer one dimension using -1
>>> reshaped = np.reshape(a, (-1, 2))
>>> print(reshaped)
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

Observe that in the above example, the given array arr is reshaped into two arrays with the same number of elements. Each array contains three elements. When you infer one dimension using -1 then each array contains 2 elements.

**2. ndarray.reshape() :** It works like np.reshape() but is a method of the array object.

```
a = np.array([1, 2, 3, 4])
reshaped = a.reshape(2, 2)
print(reshaped)
# Output:
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Observe that in the above example, the given array is reshaped into two arrays with the same elements.

**3. Flattening Arrays**

Flattening means to convert an array into a one-dimensional array. Flattening an array is the process of taking the nested elements within an array and putting them into a single array, in other words converting a multi-dimensional array into a single one-dimensional array. There are two functions to flatten the array.

**ndarray.ravel() :** Returns a flattened view if possible.

**ndarray.flatten() :** Returns a copy of the flattened array.

***Example 2.14*** illustrates the Flattening an array.

```
>>> Flattening an array
>>> a = np.array([[1, 2], [3, 4]])
>>> print(a.ravel())
[1 2 3 4]
>>> print(a.flatten())
[1 2 3 4]
```

Observe that in the above example a two-dimensional array is converted into one dimensional array.

$$a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$\text{result} = [1 \;\; 2 \;\; 3 \;\; 4]$$

## 2.4 RESIZING ARRAYS

To resize an array, you can create a new array with a larger capacity, copy the elements from the old array to the new one, and then replace the old array with the new one.

**np.resize() :** It creates a new array with a specified shape. If the new shape has more elements, the array is repeated to fill it. If the new shape has fewer elements, the array is truncated.

**Syntax:**

np.resize(array, new_shape)

***Example 2.15*** illustrates the to resize the array.

```
>>> # Resizing Array
>>> a = np.array([1, 2, 3, 4])
>>> # Resize to a 2x3 array (data repeats)
>>> resized = np.resize(a, (2, 3))
>>> print(resized)
```

$$a = [1 \;\; 2 \;\; 3 \;\; 4]$$

$$\text{result} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 1 & 2 \end{bmatrix}$$

```
>>>
>>> # Resize to a smaller shape (truncated)
```

```
>>> resized = np.resize(arr, (2, 2))
>>> print(resized)
```

$$a = [1 \quad 2 \quad 3 \quad 4]$$

$$\text{result} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

>>> In the above example the given array arr is resized into a 2x3 array with data repeated. Further it is resized to a smaller shape.

**2. In-Place Resizing**

**ndarray.resize() :** This function modifies the array itself to match the new shape. If the new shape is larger, the array is filled with default values usually 0.

**Syntax:**

```
ndarray.resize(new_shape)
```

***Example 2.16*** illustrates how to resize the array in-place.

```
# Resize the array in place
>>> a = np.array([1, 2, 3, 4])
>>> # Resize the array in-place
>>> a.resize((2, 3))
>>> print(a)
```

$$a = [1 \quad 2 \quad 3 \quad 4]$$

$$\text{result} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 0 & 0 \end{bmatrix}$$

>>>

Observe that in the above example the given array arr is resized into a two-dimensional array.

**Key Differences Between Reshape and Resize**

| Feature | Reshape | Resize |
|---|---|---|
| Returns/Modifies | Returns a new array (view or copy). | Modifies the array in-place. |
| Size Match | Total elements must match before reshaping. | May add or truncate elements. |
| Behaviour on Change | Keeps the data intact, just rearranges it. | May repeat or truncate elements. |

**2.5 CHANGING DIMENSIONS**

**1. Adding Dimensions**

We can add or remove dimensions to an array by using Numpy functions.

**np.newaxis :** Adds a new dimension to an array.

**np.expand_dims() :** Explicitly adds a new axis at a specific position.

***Example 2.17*** illustrates how to add a new dimension to array.

```
>>> # adding dimension to array.
>>> a = np.array([1, 2, 3])
>>> # Add a new dimension
>>> print(a[np.newaxis, :])
[[1 2 3]]
>>>
>>> print(np.expand_dims(a, axis=1))
```

$$a = [1 \quad 2 \quad 3]$$

$$\text{result} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

In the above example the given one dimensional array is converted into a two-dimensional array.

**2. Removing Dimensions**

**np.squeeze() :** Removes dimensions of size 1.

***Example 2.18*** illustrates how to remove new dimension of array.

```
>>> # Removing dimension of array.
>>> a = np.array([[[11, 22, 33]]])
>>> # Shape: (1, 1, 3)
>>> # Remove size-1 dimensions
>>> squeezed = np.squeeze(a)
>>> print(squeezed)
```

$$a = [[11 \; 22 \; 33]]$$

$$\text{result} = [11 \quad 22 \quad 33]$$

In the above example observe that the dimension of array a is reduced.



## Summary

- Focus on managing arrays through joining, splitting, reshaping, flattening, resizing, and changing dimensions.
- Joining: concatenate(), vstack(), hstack(), dstack().

- Splitting: split(), array_split(), vsplit(), hsplit(), dsplit().
- Reshaping: reshape() (new view/copy), resize() (in-place, may truncate/expand data).
- Flattening: ravel() (view), flatten() (copy).
- Changing dimensions: add (newaxis, expand_dims) or remove (squeeze) axes.
- Provides flexibility to organize and reformat data for analysis.

## Check Your Progress

**A. Multiple choice questions**

1. Which of the following function is used to stacks 1D arrays as columns into a 2D array. (a) np.hstack() (b) np.dstack() (c) np.column_stack() np.row_stack()
2. Which of the following is used to reshape a NumPy array? (a) reshape() (b) resize() (c) Both A and B (d) None of the above
3. Which of the following function stacks 1D arrays as columns into a 2D array? (a) row_stack (b) column_stack (c) com_stack (d) All of the above
4. Which of the following function is used to stack arrays along the third dimension (a) np.dstack() (b) np.column_stack() (c) np.row_stack() (d) None of the above
5. If a dimension is given as _____ in a reshaping operation, the other dimensions are automatically calculated. (a) 0 (b) 1 (c) -1 (d) Infinite
6. Which of the following function allows unequal splitting of array. (a) np.array_split() (b) np.vsplit() (c) np.hsplit() (d) np.dsplit()
7. Which of the following function works on arrays with at least 3 dimensions. (a) np.vsplit() (b) np.hsplit() (c) np.dsplit() (d) All of the above

**B. Fill in the blanks**

1. The function to stack arrays horizontally is _____
2. The function _____ stacks 1D arrays as columns into a 2D array
3. np.row_stack() function is similar to _____
4. np.split() function is used to split an array into multiple sub-arrays along a _____.
5. The function _____ adds a new dimension to an array.
6. The function _____ removes dimensions of size 1.
7. The function _____ explicitly adds a new axis at a specific position.
8. In _____, if the new shape is larger, the array is filled with default values usually 0.
9. The function _____ returns a flattened view if possible.

**C. State whether True or False**

1. The function np.concatenate() is used to join two or more arrays
2. The function np.vstack() is used to stack arrays column-wise.

3. The function np.row_stack() stacks 1D arrays as rows into a 2D array
4. The function np.vsplit() works on arrays with at least 2 dimensions.
5. The function np.array_split() equal splits an array into multiple sub-arrays.
6. The function np.reshape() creates a new array with the same data but different shape.
7. The function ndarray.reshape() modifies the array itself to match the new shape.
8. The function ndarray.ravel() returns a copy of the flattened array.

**D. Answer the following questions in short**

1. What is joining arrays?
2. What is splitting arrays?
3. What are reshaping arrays?
4. What is Flattening arrays?
5. What do you mean by changing dimensions.np.hsplit



## Session 3. Array Computation using NumPy

Think about handling your monthly expenses. You add income, subtract expenses, multiply recurring bills, and divide savings across needs. These are all computations. Similarly, NumPy allows such arithmetic and mathematical operations on arrays, but at a much faster speed and with greater accuracy.


Fig. 3.1: Monthly expenses operation charts

NumPy, provides a powerful and efficient way to perform various operations on arrays. These operations are faster than using traditional Python lists, especially when dealing with large datasets. Some of the common operations that can be performed on NumPy arrays are covered in this session. The students will be able to perform arithmetic operations (addition, subtraction, multiplication, division) and mathematical operations (modulus, exponentiation, floor division) on arrays. They will also learn to apply aggregate

functions like sum, mean, product, and standard deviation, and explore matrix operations such as dot product and transpose.

### 3.1 ARITHMETIC OPERATIONS

The basic arithmetic operation like addition, subtraction, multiplication, division can be performed on NumPy arrays. These opertations are performed on on each corresponding pair of elements. For instance, adding two arrays will result in the first element in the first array to be added to the first element in the second array, and so on.

It is important to note that for element-wise operations, size of both arrays must be same. That is, array1.shape must be equal to array2.shape.

The primary arithmetic operations supported by NumPy are:

#### 3.2.1 Addition

Addition corresponding elements of two arrays or an array and a scalar. Operator used for addition is +. Function for addition is np.add().

***Example 3.1*** illustrates the addition of two arrays.

```
# Addition of two arrays
>>> import numpy as np
>>> a1 = np.array([11, 22, 33])
>>> a2 = np.array([10, 20, 30])
>>> # Element-wise addition
>>> result = a1 + a2
>>> print(result)
[21 42 63]
```

In the above example, observe that column wise addition of two arrays a1 and a2 takes place.

$$a1 = [11 \quad 22 \quad 33]$$

$$a2 = [10 \quad 20 \quad 30]$$

$$result = [21 \quad 42 \quad 63]$$

It is also possible to add a scalar in the array. By adding a scalar in the array, the scalar number will get added to each element of the array in the resultant array.

***Example 3.2*** illustrates to add a scalar to array.

```
>>> # Adding a scalar to array
>>> result = a1 + 10
>>> print(result)
[21 32 43]
```

In the above Example 3.2, a scalar 10 is added into each element of array a1.

$$a1 = [11 \quad 22 \quad 33]$$

$$result = a1 + 10$$

$$[21 \quad 32 \quad 43]$$

NumPy's **numpy.add()** is a function that performs element-wise addition on NumPy arrays. This means it adds the corresponding elements between two arrays, element by element, instead of treating them as single values.

**numpy.add()** function is used when we want to compute the addition of two arrays. It adds arguments element-wise. If the shape of two arrays is not the same, that is a1. shape! = a2. shape, they must be broadcastable to a common shape.

***Example 3.3*** illustrates to use numpy add function.

```
# Using numpy add function.
>>> import numpy as np
>>> # Numpy array and a scalar
>>> a1 = np.array([11, 22, 33])
>>> # Using numpy.add() with an array and a scalar
>>> result = np.add(a1, scalar)
>>> print("Result of adding array and scalar:", result)
Result of adding array and scalar: [15 26 37]
```

$$a1 = [11 \quad 22 \quad 33]$$

$$result = [15 \quad 26 \quad 37]$$

### 3.2.2. Subtraction

Subtraction corresponds to the difference of elements of two arrays or a scalar from an array. Operator used for subtraction is – and the function for subtraction is np.subtract().

***Example 3.4 illustrates*** the subtraction on arrays.

```
# Subtraction on arrays.
>>> import numpy as np
>>> a1 = np.array([10, 20, 30])
>>> a2 = np.array([40, 50, 60])
>>> result = a2 - a1
>>> print(result)
[30 30 30]
```

In the above output, observe the element wise difference of two arrays a1 and a2.

$$a1 = [10 \quad 20 \quad 30]$$

$$a2 = [40 \quad 50 \quad 60]$$

$$result = a2 - a1$$

$$[30 \quad 30 \quad 30]$$

It is also possible to subtract a number from an array. It will subtract the specified number from each element of the array.

***Example 3.5*** illustrates to subtract a specified number from arrays.

```
>>> Subtracting a specified number from arrays
>>> result = a1 - 11
>>> print(result)
[-1  9 19]
```

In the above example, the number 11 is subtracted from each element of the array and gives the resultant array.

$$a1 = [10 \quad 20 \quad 30]$$

$$result = a1 - 11$$

$$[-1 \quad 9 \quad 19]$$

**numpy.subtract()** function is used when we want to compute the difference of two arrays. It returns the difference of arr1 and arr2, element-wise.

***Example 3.6*** illustrates to use numpy.subtract() function.

```
>>> # use numpy.subtract() function
>>> import numpy as np
>>> a1_in = np.array([[21, -40, 45], [-16, 12, 10]])
>>> a2_in = np.array([[10, -15, 25], [20, -22, 19]])
>>> print ("1st Input array :\n", a1_in)
1st Input array :
 [[ 21 -40  45]
 [-16  12  10]]
>>> print ("1st Input array :\n", a2_in)
1st Input array :
 [[ 10 -15  25]
 [ 20 -22  19]]
>>> a_out = np.subtract(a1_in, a2_in)
>>> print ("Output array:\n", a_out)
```

```
Output array:
 [[ 11 -25  20]
 [-36  34  -9]]
>>>
```

$$a1 = \begin{bmatrix} 21 & -40 & 45 \\ -16 & 12 & 10 \end{bmatrix}$$

$$a2 = \begin{bmatrix} 10 & -15 & 25 \\ 20 & -22 & 19 \end{bmatrix}$$

$$result = a1 - a2$$

$$\begin{bmatrix} 11 & -25 & 20 \\ -36 & 34 & -9 \end{bmatrix}$$

### 3.2.3. Multiplication

This operation multiplies corresponding elements of two arrays or scales an array by a scalar. Operator used for multiplication is *. Function used for multiplication is np.multiply().

***Example 3.7*** illustrates the multiplication of arrays.

```
>>> # Multiplication of arrays
>>> import numpy as np
>>> a1 = np.array([11, 22, 33])
>>> a2 = np.array([3, 2, 1])
>>> result = a1 * a2
>>> print(result)
[33 44 33]
```

Observe that element wise multiplication of two arrays a1 and a2 will take place.

$$a1 = [11 \quad 22 \quad 33]$$

$$a2 = [3 \quad 2 \quad 1]$$

$$result = a1 \times a2$$

$$[33 \quad 44 \quad 33]$$

Similarly, it is possible to multiply the array with scalar. The scalar will get multiplied by each element of the array.

```
>>> result = a1 * 2
>>> print(result)
[22 44 66]
```

Observe the output the scalar 2 is multiplied with all elements of array a1.

$$a1 = [11 \quad 22 \quad 33]$$

$$result = a1 \times 2$$

$$[22 \quad 44 \quad 66]$$

**numpy.multiply()** function is used when we want to compute the multiplication of two arrays. It returns the product of arr1 and arr2, element-wise.

***Example 3.8*** illustrates to use of the numpy.multiply() function.

```
>>> # Python program explaining numpy.multiply() function
>>> import numpy as np
>>> a1_in = np.array([[21, -17, 15], [-6, 12, 10]])
>>> a2_in = np.array([[10, -7, 8], [15, -2, 19]])
>>> print ("1st Input array : \n", a1_in)
1st Input array :
 [[ 21 -17  15]
 [ -6  12  10]]
>>> print ("2nd Input array : \n", a2_in)
2nd Input array :
 [[10 -7  8]
 [15 -2 19]]
>>> out_arr = np.multiply(a1_in, a2_in)
>>> print ("Resultant output array: \n", a_out)
Resultant output array:
 [[ 11 -25  20]
 [-36  34  -9]]
>>>
```

$$a1 = \begin{bmatrix} 21 & -17 & 15 \\ -6 & 12 & 10 \end{bmatrix}$$

$$a2 = \begin{bmatrix} 10 & -7 & 8 \\ 15 & -2 & 19 \end{bmatrix}$$

$$\begin{bmatrix} 11 & -25 & 20 \\ -36 & 34 & -9 \end{bmatrix}$$

### 3.2.4. Division

This operation divides corresponding elements of two arrays or divides an array by a scalar. Operator used for division is / and the function used for division is np.divide().

***Example 3.9*** illustrates the the division of arrays.

```
>>>  # Python program for division of array
>>> import numpy as np
```

```
>>> a1 = np.array([11, 22, 33])
>>> a2 = np.array([44, 55, 66])
>>> result = a2 / a1
>>> print(result)
[4.  2.5 2. ]
```

Observe that elements of array a2 are divided by elements of array a1.

$$a1 = [11 \quad 22 \quad 33]$$

$$\text{a2} = [44 \quad 55 \quad 66]$$

$$\text{result} = \text{a2} \div \text{a1}$$

$$[4 \quad 2.5 \quad 2]$$

**Division by Scalar**

It is possible to multiply the array with scalar. The scalar will get multiplied by each element of the array.

***Example 3.9.1*** illustrates the division by saclar

```
>>> result = a1 / 2
>>> print(result)
[ 5.5 11.  16.5]
```

In the above example all the elements of array a1 are divided by 2.

$$a1 = [11 \quad 22 \quad 33]$$

$$\text{result} = \text{a1} \div 2$$

$$[5.5 \quad 11 \quad 16.5]$$

**numPy.divide() :** Array element from the first array is divided by elements from the second element (all happens element-wise). Both arr1 and arr2 must have the same shape and element in arr2 must not be zero; otherwise it will raise an error.

***Example 3.10*** illustrates to use of the numpy.divide() function.

```
>>> # Python program explaining divide () function
>>> import numpy as np
>>> # input_array
>>> a1 = [22, 27, 12, 21, 23]
>>> a2 = [2, 3, 4, 5, 6]
>>> out = np.divide(a1, a2)
>>> print ("a1: \n", a1)
a1: [22, 27, 12, 21, 23]
>>> print ("a2: \n", a2)
a2: [2, 3, 4, 5, 6]
>>> print(out)
```

```
[11.          9.          3.          4.2         3.83333333]
>>>
```

$a1 = [22 \quad 27 \quad 12 \quad 21 \quad 23]$

$a2 = [2 \quad 3 \quad 4 \quad 5 \quad 6]$

$result = [11 \quad 9 \quad 3 \quad 4.2 \quad 3.834]$

**Divide by zero error**

If any element of the second array is 0 then it is not possible to divide the array and will raise the division by zero error as illustrated below.

***Example 3.11*** illustrates the division by zero error .

```
>>> # Python program explaining divide() function
>>> import numpy as np
>>> # input_array
>>> a1 = [22, 27, 12, 21, 23]
>>> a2 = [2, 3, 0, 5, 6]
>>> print ("a1: \n", a1)
a1: [22, 27, 12, 21, 23]
>>> print ("a2: \n", a2)
a2: [2, 3, 0, 5, 6]
>>> out = np.divide(a1, a2)
```

```
<stdin>:1: RuntimeWarning: divide by zero encountered in divide
```

In this output, observe that error is generated on division by 0.

$a1 = [22 \quad 27 \quad 12 \quad 21 \quad 23]$

$a2 = [2 \quad 3 \quad 4 \quad 5 \quad 6]$

$result = [0]$

### 3.2.5 Floor Division

This operation performs integer division, that is, it truncates the decimal. Operator used for floor division is // and the function used is np.floor_divide().

***Example 3.12*** illustrates the floor division of arrays.

```
>>> # Python code illustrates the floor division of arrays.
>>> import numpy as np
>>> a1 = np.array([10, 20, 30])
>>> a2 = np.array([44, 59, 62])
>>> result = a2 // a1
>>> print(result)
[4 2 2]
```

Observe that in the above output the Fractional part of division does not appear in the result.

$$a1 = [10 \quad 20 \quad 30]$$

$$\text{a2} = [44 \quad 59 \quad 62]$$

$$\text{result} = \text{a2} \div \text{a1}$$

$$[4 \quad 2 \quad 2]$$

**np.floor_divide() Function:** Array element from the first array is divided by the elements from the second array element-wise. Both array 1 and array 2 must have the same shape. It is equivalent to the Python // operator.

***Example 3.13*** illustrate to use np.floor_divide() function.

```
# Python program illustrating np.floor_divide() function
import numpy as np
>>> a1 = [2, 2, 2, 3, 3]
>>> a2 = [2, 3, 4, 3, 6]
>>> out = np.floor_divide(a1, a2)
>>> print ("array 1: \n ", a1)
array 1: [2, 2, 2, 3, 3]
>>> print ("array2: \n ", a2)
array2: [2, 3, 4, 3, 6]
>>> print ("\nOutput array :\n", out)
Output array :
 [1 0 0 1 0]
>>>
```

$$a1 = [2 \quad 2 \quad 2 \quad 3 \quad 3]$$

$$\text{a2} = [2 \quad 3 \quad 4 \quad 3 \quad 6]$$

$$\text{result} = [1 \quad 0 \quad 0 \quad 1 \quad 0]$$

### 3.2.6 Modulus (Remainder)

This operation calculates the remainder of division for each element. Operator used is % and the function used is np.mod().

***Example 3.14*** illustrate the use of Modulus.

```
>>> # Python program to illustrate Modulus (Reminder)
>>> import numpy as np
>>> a1 = np.array([10, 20, 30])
>>> a2 = np.array([42, 55, 66])
>>> result = a2 % a1
>>> print(result)
[ 2 15  6]
>>>
```

$$a1 = [10 \quad 20 \quad 30]$$

$$a2 = [42 \quad 55 \quad 66]$$

$$result = a2 \% a1$$

$$[2 \quad 15 \quad 6]$$

**np.mod() Function:** This function returns element-wise remainder of division between two arrays a1 and a2 i.e. a1 % a2. It returns 0 when a2 is 0 and both a1 and a2 are (arrays of) integers.

***Example 3.15*** illustrate the use of Np.mod() function.

```
>>> # Python program to illustrate numpy.mod() function
>>> import numpy as np
>>> a1_in = np.array([2, -4, 7])
>>> a2_in = np.array([2, 3, 4])
>>> a_out = np.mod(a1_in, a2_in)
>>> print ("Dividend array : ", a1_in)
Dividend array :  [ 2 -4  7]
>>> print ("Divisor array : ", a2_in)
Divisor array :  [2 3 4]
>>> a_out = np.mod(a1_in, a2_in)
>>> print ("Output remainder array:", a_out)
Output remainder array: [0 2 3]
>>>
```

$$a1 = [2 \quad -4 \quad 7]$$

$$a2 = [2 \quad 3 \quad 4]$$

$$result = [0 \quad 2 \quad 3]$$

### 3.2.7 Exponentiation

This operation raises each element of an array to the power of corresponding elements in another array or a scalar. Operator used for operation is ** and the function used is np.power().

***Example 3.16*** illustrate the Exponentiation operation.

```
>>> # Python program to illustrate exponentiation operation
>>> import numpy as np
>>> a1 = np.array([0, 1, 2])
>>> a2 = np.array([4, 5, 6])
>>> a1exp = a1 ** 2
>>> a2exp = a2 ** 3
>>> print(a1exp)
[0 1 4]
```

$a1 = [0 \quad 1 \quad 2]$

$a2 = [4 \quad 5 \quad 6]$

$\text{result} = (a1)^2$

$[0 \quad 1 \quad 4]$

```
>>> print(a2exp)
[ 64 125 216]
```

$a1 = [0 \quad 1 \quad 2]$

$a2 = [4 \quad 5 \quad 6]$

$\text{result} = (a2)^3$

$[64 \quad 125 \quad 216]$

```
>>> Observe that all elements of array a1 are raised to 2 and a2 raised
by 3.
>>> result = np.power(a1, a2)
>>> print(result)
[ 0  1 64]
>>>
```

$a1 = [0 \quad 1 \quad 2]$

$a2 = [4 \quad 5 \quad 6]$

$\text{result} = (a1)^{a2}$

$[0 \quad 1 \quad 64]$

Here all elements of arr1 are raised by elements of arr2. 1**4=1, 2**5=32, 3**6=729.

**np.power() Function:** Array element from the first array is raised to the power of the element from the second element, element-wise. Both arr1 and arr2 must have the same shape and each element in arr1 must be raised to corresponding +ve value from arr2; otherwise it will raise a ValueError.

***Example 3.17*** illustrate the use of np.power function

```
>>> # Python program illustrating power() function
>>> import numpy as np
>>> # input_array
>>> a1 = [2, 2, 2, 2, 2]
>>> a2 = [2, 3, 4, 5, 6]
```

```
>>> out = np.power(a1, a2)
>>> print ("array 1 : ", a1)
array 1 :  [2, 2, 2, 2, 2]
>>> print ("array 2 : ", a2)
array 2 :  [2, 3, 4, 5, 6]
>>> # output_array
>>> print ("\nOutput array : ", out)
Output array :  [ 4  8 16 32 64]
>>>
```

$$a1 = [2 \;\; 2 \;\; 2 \;\; 2 \;\; 2]$$

$$\text{a2} = [2 \;\; 3 \;\; 4 \;\; 5 \;\; 6]$$

$$\text{result} = (\text{a1})^{\text{a2}}$$

$$[4 \;\; 8 \;\; 16 \;\; 32 \;\; 64]$$

### 3.2.8 Broadcasting in Arithmetic

NumPy uses broadcasting to perform operations on arrays of different shapes, provided they are compatible.

***Example 3.18*** illustrate the Broadcasing in Arithmetic

```
>>> # Python program illustrating broadcasting in arithmetic
>>> a = np.array([[1, 2], [3, 4]])
>>> # Broadcasting a scalar
>>> result = a + 12
>>> print(result)
[[13 14]
 [15 16]]
```

$$a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$\text{result} = \text{a} + 12$$

$$\begin{bmatrix} 13 & 14 \\ 15 & 16 \end{bmatrix}$$

```
>>> # Broadcasting a 1D array
>>> result = a + np.array([1, 2])
>>> print(result)
[[2 4]
 [4 6]]
>>>
```

$$a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$\text{result} = \text{a} + [1 \quad 2]$$

$$\begin{bmatrix} 2 & 4 \\ 4 & 6 \end{bmatrix}$$

### 3.3 AGGREGATE ARITHMETIC OPERATIONS

These functions compute aggregate results across elements of the array.

#### 3.3.1 Sum

**np.sum()** function returns the sum of array elements over the specified axis.

***Example 3.19*** illustrate the use of np.sum() function.

```
>>> # Python code to illustrate np.sum() function
>>> import numpy as np
>>> a = np.array([3, 5, 6, 7])
>>> print(np.sum(a))
21
```

$$a = [3 \quad 5 \quad 6 \quad 7]$$

$$\text{result} = [3 + 5 + 6 + 7]$$

$$21$$

Observe that all elements of array are added together to get result 10.

#### 3.3.2 Product

np.prod() function is used to return the product of array elements over a given axis.

***Example 3.20*** illustrate the use of np.prod() function.

```
>>> # Python code to illustrate np.prod() function
>>> import numpy as np
>>> a = np.array([1, 2, 3, 4])
>>> print(np.prod(a))
24
```

All elements of the array are multiplied together to return value 24.

$$a = [1 \quad 2 \quad 3 \quad 4]$$

$$\text{result} = [1 \times 2 \times 3 \times 4]$$

$$24$$

#### 3.3.3 Mean

The function np.mean() returns the arithmetic mean along the specified axis.

***Example 3.21*** illustrate the use of mean() function.

```
>>> # Python code to illustrate np.mean() function
>>> import numpy as np
>>> a = np.array([5, 6, 3, 4])
>>> print(np.mean(a))
4.5
```

Observe that all 4 elements of the array are added together and then it is divided by 4 to get the result.

$a = [1 \quad 2 \quad 3 \quad 4]$

$$\text{result} = \mu = \frac{\sum \text{Xi}}{\text{N}}$$

$$\left[\frac{3+4+5+6}{4}\right]$$

$4.5$

### 3.3.4 Standard Deviation

The function np.std(a) returns the standard deviation of the given array elements along the specified axis. Standard Deviation (SD) is measured as the spread of data distribution in the given data set.

***Example 3.22*** illustrate the use of np.std() function.

```
>>> # Python code to illustrate np.std() function
>>> import numpy as np
>>> a = np.array([1, 2, 3, 4])
>>> # Output
>>> print(np.std(a))
1.118033988749895
>>>
```

$a = [1 \quad 2 \quad 3 \quad 4]$

$$\text{result} = \sigma 2 = \frac{\sum (\text{Xi} - \mu)2}{\text{N}}$$

$$\sigma 2 = \frac{2.25 + 0.25 + 0.25 + 2.25}{4}$$

$$\sigma 2 = \frac{5}{4} = 1.25$$

$$\sigma = \sqrt{1.25} \approx 1.118$$

### 3.3.5 Cumulative Sum

The function np.cumsum(a) returns the cumulative sum of the elements along a given axis.

***Example 3.23*** illustrate the use of numpy cumulative sum function.

```
>>> # Python code to illustrate np.cumsum() function
>>> import numpy as np
>>> a = np.array([1, 2, 3, 4])
>>> # Output
>>> print(np.cumsum(a))
[ 1  3  6 10]
>>>
```

$$a = [1 \;\; 2 \;\; 3 \;\; 4]$$

$$\text{result} = [1 \;\; 3 \;\; 6 \;\; 10]$$

### 3.3.6 Cumulative Product

The function np.cumprod(a) is used to return cumulative products of elements along a given axis.

***Example 3.24*** illustrate the use of numpy cumulative product function.

```
>>> # Python code to illustrate np.cumprod() function
>>> import numpy as np
>>> a = np.array([1, 2, 3, 4])
>>> # Output
>>> print(np.cumprod(a))
[ 1  2  6 24]
>>>
```

$$a = [1 \;\; 2 \;\; 3 \;\; 4]$$

$$\text{result} = [1 \;\; 2 \;\; 6 \;\; 24]$$

### 3.3.7 Matrix Operations

For matrix-specific operations, NumPy provides a specialized function np.dot() for dot Product. It can handle 2D arrays but considers them as matrices and will perform matrix multiplication. For N dimensions it is a sum-product over the last axis of a and the second-to-last of b.

***Example 3.25*** illustrate the matrix operations.

```
>>> # Python code to illustrate matrix operations
>>> a = np.array([[1, 2], [3, 4]])
>>> b = np.array([[5, 6], [7, 8]])
>>> result = np.dot(a, b)
>>> print(result)
[[19 22]
```

```
 [43 50]]
```

These arithmetic operations make NumPy highly versatile for mathematical and scientific computations.

$$a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$\mathrm{b} = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

$$\text{result} = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

### 3.4 TRIGONOMETRIC FUNCTIONS

NumPy provides a comprehensive set of trigonometric functions to perform operations involving angles and trigonometric calculations. Here's a detailed explanation of these functions:

#### 3.4.1 Basic Trigonometric Functions

These functions compute the trigonometric values of each element in an array. Angles must be in radians unless specified otherwise.

**1. Sine**

Sine function computes the sine of each element in the array. The function used is np.sin().

***Example 3.26*** illustrate to compute Sine using np.sin() function.

```
>>> # Python code to illustrate the Sine function
>>> import numpy as np
>>> angles = np.array([0, np.pi/2, np.pi])
>>> result = np.sin(angles)
>>> print('Output \n',result)
Output
 [0.0000000e+00 1.0000000e+00 1.2246468e-16]
```

Observe that in the result the first value is 0, second value is 1 and third value is close to 0.

**2. Cosine**

Cosine function computes the cosine of each element. The function used is np.cos().

***Example 3.27*** illustrate to compute Cosine using np.cos() function.

```
>>> # Python code to illustrate the Cosine function
>>> import numpy as np
>>> angles = np.array([0, np.pi/2, np.pi])
>>> result = np.cos(angles)
>>> print('Output \n',result)
Output
```

```
 [ 1.000000e+00  6.123234e-17 -1.000000e+00]
```

Observe that in the result the first value is 1, second value is close to 0 and third value is -1.

**3. Tangent**

Tangent function computes the tangent of each element. The function used is np.tan().

***Example 3.28*** illustrate to compute Tangent using np.tan() function.

```
>>> # Python code to illustrate the Tangent function
>>> angles = np.array([0, np.pi/2, np.pi])
>>> result = np.tan(angles)
>>> print('Output \n',result)
Output
 [ 0.00000000e+00  1.63312394e+16 -1.22464680e-16]
```

```
>>> Observe that in the result the first value is 0, second value is 1 and
third value is close to 0.
```

**3.4.2 Inverse Trigonometric Functions**

These functions compute the inverse trigonometric values, returning angles in **radians**.

**1. Arcsine**

This function computes the arcsine of each element. The function used is np.arcsin().

***Example 3.29*** illustrate to compute Arcsine using np.arcsin() function.

```
>>> # Python code to illustrate the Arcsine function
>>> values = np.array([0,1,-1])
>>> result = np.arcsin(values)
>>> print('Output \n',result)
Output
 [ 0.          1.57079633 -1.57079633]
>>>
```

**2. Arccosine**

This function computes the arccosine of each element. The function used is np.arccos().

***Example 3.30*** illustrate to compute Arccosine using np.arccos() function.

```
>>> # Python code to illustrate the Arcsine function
>>> values = np.array([0,1,-1])
>>> result = np.arccos(values)
>>> print('Output \n',result)
Output
 [1.57079633 0.         3.14159265]
>>>
```

**3. Arctangent**

This function computes the arctangent of each element. The function used is np.arctan().

***Example 3.31*** illustrate to compute Arctangent using np.arctan() function.

```
>>> # Python code to illustrate the Arctangent function
>>> values = np.array([0,1,-1])
>>> result = np.arctan(values)
>>> print('Output \n',result)
Output
 [ 0.          0.78539816 -0.78539816]
>>>
```

**4. Arctangent2 (Two-argument Arctangent)**

This function computes the arctangent of y/xy/xy/x considering the quadrant of the point (x,y)(x, y)(x,y). The function used is np.arctan2().

***Example 3.32*** illustrate to compute Arctangent with two arguments using np.arctan2() function.

```
>>> # Python code to illustrate the Arctangent function with two
arguments
>>> y = np.array([1, -1])
>>> x = np.array([1, 1])
>>> result = np.arctan2(y, x)
>>> print('Output \n',result)
Output
 [ 0.78539816 -0.78539816]
>>>
```

### 3.4.3 Hyperbolic Trigonometric Functions

These functions compute hyperbolic trigonometric values.

**1. Hyperbolic Sine**

This function is used to compute hyperbolic sine. The function used is np.sinh().

***Example 3.33*** illustrate to compute Hyperbolic sine using using np.sinh() function.

```
>>> # Python code to illustrate Hyperbolic Sine function
>>> values = np.array([0, 1, -1])
>>> result = np.sinh(values)
>>> print('Output \n',result)
Output
 [ 0.          1.17520119 -1.17520119]
>>>
```

**2. Hyperbolic Cosine**

This function is used to compute hyperbolic cosine. The function used is np.cosh().

***Example 3.34*** illustrate to compute Hyperbolic Cosine using using np.cos() function.

```
>>> # Python code to illustrate Hyperbolic Cosine function
>>> values = np.array([0, 1, -1])
>>> result = np.cosh(values)
>>> print('Output \n',result)
Output
 [1.         1.54308063 1.54308063]
>>>
```

**3. Hyperbolic Tangent**

This function is used to compute hyperbolic tangents. The function used is np.tanh().

***Example 3.35*** illustrate to compute Hyperbolic tangent using using np.tanh() function.

```
>>> # Python code to illustrate Hyperbolic Tangent function
>>> values = np.array([0, 1, -1])
>>> result = np.tanh(values)
>>> print('Output \n',result)
Output
 [ 0.          0.76159416 -0.76159416]
>>>
```

### 3.4.4 Inverse Hyperbolic Functions

The function arcsinh is an inverse sine hyperbolic function. The function used is np.arcsinh(). The function arccosh is an inverse cosine hyperbolic function. The function used is np.arccosh(). The function arctanh is an inverse tangent hyperbolic function. The function used is np.arctanh().

***Example 3.36*** illustrate to compute Inverse Hyperbolic sine using using np.arcsinh() function.

```
>>> # Python code to illustrate Inverse Hyperbolic Tangent function
>>> values = np.array([0, 1, -1])
>>> result = np.arcsinh(values)
>>> print('Output \n',result)
Output
 [ 0.          0.88137359 -0.88137359]
>>>
```

### 3.4.5 Angle Conversion

NumPy includes functions to convert between radians and degrees.

**1. Convert Degrees to Radians**

The Function np.radians() is used to convert degrees to radians.

**Example: 3.37**

```
>>> # Python code to convert Degrees to Radians
>>> degrees = np.array([0, 90, 180])
```

```
>>> result = np.radians(degrees)
>>> print('Output \n',result)
Output
 [0.         1.57079633 3.14159265]
>>>
```

**2. Convert Radians to Degrees**

The function np.degrees() is used to convert radians to degrees.

***Example 3.38*** illustrate to convert radians to degrees using npdegrees() function.

```
>>> # Python code to convert Radians Degrees
>>> radians = np.array([0, np.pi/2, np.pi])
>>> result = np.degrees(radians)
>>> print('Output \n',result)
Output
 [  0.  90. 180.]
>>>
```

### 3.4.6 Other Trigonometric Utilities

**1. Compute Hypotenuse**

This function computes the hypotenuse of a right triangle given the lengths of the two perpendicular sides. The function used is np.hypot().

***Example 3.39*** illustrate to compute Hypotenuse using nphypot() function.

```
>>> # Python code to compute Hypotenuse
>>> x = np.array([3, 5])
>>> y = np.array([4, 12])
>>> result = np.hypot(x, y)
>>> print('Output \n',result)
Output
 [ 5. 13.]
```

These trigonometric functions make NumPy a powerful tool for mathematical computations involving angles and periodic functions.

## Summary

- Arithmetic operations: addition, subtraction, multiplication, division, modulus, exponentiation, floor division.
- Operations can be element-wise (same shape) or via broadcasting (different shapes, but compatible).
- Aggregate functions: sum(), prod(), mean(), std(), cumsum(), cumprod().
- Matrix operations: dot() for matrix multiplication, transpose for rearranging rows/columns.

- Trigonometric functions: sin(), cos(), tan(), inverse functions, hypot() for hypotenuse.
- Enables scientific and mathematical computations efficiently on large datasets.

## Check Your Progress

**A. Multiple choice questions**

1. Which of the following NumPy functions performs element-wise addition on two arrays (a) np.add() (b) np.sum() (c) np.multiply() (d) np.prod()
2. Which NumPy function calculates the sum of all elements in an array? (a) np.add() (b) np.sum() (c) np.multiply() (d) np.prod()
3. Which NumPy function performs element-wise multiplication on two arrays? (a) np.add() (b) np.sum() (c) np.multiply() (d) np.prod()
4. Which NumPy function calculates the product of all elements in an array? (a) np.add() (b) np.sum() (c) np.multiply() (d) np.prod()
5. Which of the following functions can be used to calculate the average of elements in a NumPy array? (a) np.mean() (b) np.average() (c) np.mean() or np.average() (d) None of the abov
6. Which of the following is NOT a trigonometric function available in NumPy? (a) sin (b) cos (c) tan (d) log
7. What is the output of numpy.sin(numpy.pi) ? (a) 0 (b) 1 (c) -1 (d) None
8. Which function would you use to calculate the cosine of an array in NumPy? (a) numpy.cos (b) numpy.sin (c) numpy.tan (d) numpy.trig
9. What is the purpose of trigonometric functions in NumPy? (a) To perform matrix operations (b) To handle statistical analysis (c) To perform element-wise calculations on arrays (d) To manipulate strings

**B. Fill in the blanks**

1. The function _____ is used to compute the addition of two arrays.
2. The function _____ is used to compute the difference of two arrays.
3. The function _____ is used for floor division.
4. The function _____ is used to convert radians to degrees.
5. The inverse trigonometric functions returns angles in _____.

**C. State whether True or False**

1. NumPy supports efficient element-wise arithmetic operations on arrays.
2. NumPy's inverse trigonometric functions return angles in radians.
3. The function, np.arccos(0) returns np.pi/2.
4. The function, np.arctan(1) returns np.pi/4.
5. The function, np.arcsin(0.5) returns approximately 0.5236 radians.

6. The function, np.arctan2(0, 1) returns 0.
7. The function, np.arctan2(1, 0) returns 0.
8. The function, np.arccos(-1) returns np.pi.
9. The function, np.arctan2(1, 1) returns np.pi/4.
10. The function, np.arcsin(0) returns 0.

**D. Answer the following questions in short**

1. What is broadcasting in arithmetic?
2. How can inverse trigonometric functions be computed using NumPy?
3. What are the domains and ranges of the inverse trigonometric functions implemented in NumPy?
4. How to convert the output of NumPy's inverse trigonometric functions from radians to degrees?
5. What is the result of applying inverse trigonometric functions to values outside their domain?



## APPENDIX I: Data Types in NumPy

There are a large number of data types that are supported by NumPy. For example, in NumPy we can have integer type, floating type, complex type, Boolean type, string type, object type, date and time type.

**Integer Type**

This type is used for storing whole numbers with varying bit sizes and signs.

| Data Type | Description | Range | Example |
|---|---|---|---|
| int8 | 8-bit signed integer | -128 to 127 | >>> np.array([127, -128], dtype =np.int8)<br>array([ 127, -128], dtype=int8) |
| uint8 | 8-bit unsigned integer | 0 to 255 | >>> np.array([0,255], dtype=np.uint8)<br>array([ 0, 255], dtype=uint8) |
| int16 | 16-bit signed integer | -32,768 to 32,767 | >>> np.array([-32768, 32767], dtype=np.int16)<br>array([-32768, 32767], dtype=int16) |
| uint16 | 16-bit unsigned integer | 0 to 65,535 | >>> np.array([0, 65535], dtype=np.uint16)<br>array([0, 65535], dtype=uint16) |
| int32 | 32-bit signed integer | -2,147,483,648 to 2,147,483,647 | >>> np.array([1, -2147483648], dtype=np.int32)<br>array([1, -2147483648], dtype=int32) |
| uint32 | 32-bit unsigned integer | 0 to 4,294,967,295 | >>> np.array([0, 4294967285], dtype=np.uint32) |

| Data Type | Description | Range | Example |
|---|---|---|---|
| | | | array([0, 4294967285], dtype=uint32) |
| int64 | 64-bit signed integer | Very large range | >>> np.array([1, -9223372036854775808], dtype=np.uint64)<br>array([1, 9223372036854775808], dtype=uint64) |
| uint64 | 64-bit unsigned integer | Very large range | >>> np.array([1, -9223372036854775808], dtype=np.int64)<br>array([1, -9223372036854775808]) |

**2. Floating-Point Types**

This type is used for numbers with decimal points.

| Data Type | Description | Range | Example |
|---|---|---|---|
| float16 | Half precision | Approx. ±65,504 | >>> np.array([1.5, -2.5],dtype=np.float16)<br>array([ 1.5, -2.5], dtype=float16) |
| float32 | Single precision | Approx. ±3.4 × 10^38 | >>> np.array([1.234, -5.678], dtype=np.float32)<br>array([ 1.234, -5.678], dtype=float32) |
| float64 | Double precision | Very large range | >>> np.array([1.123456789, -9.87654321], dtype=np.float64)<br>array([ 1.12345679, -9.87654321]) |

**3. Complex Types**

This type is for complex numbers with real and imaginary parts.

| Data Type | Description | Example |
|---|---|---|
| complex64 | Complex number with two 32-bit floats | >>> np.array([1+2j, 3-4j], dtype=np.complex64)<br>array([1.+2.j, 3.-4.j], dtype=complex64) |
| complex128 | Complex number with two 64-bit floats | >>> np.array([1.5+2.5j, -3.5+4.5j], dtype=np.complex128)<br>array([ 1.5+2.5j, -3.5+4.5j]) |

**4. Boolean Type**

This type is used for True or False Boolean Type. It is used for logical operations.

| Data Type | Description | Example |
|---|---|---|
| bool_ | Boolean type (True/False) | >>> np.array([True, False], dtype=np.bool_)<br>array([ True, False]) |

**5. String Types**

This type is used for fixed-size strings.

| Data Type | Description | Example |
|---|---|---|
| string_ | Fixed-size ASCII string | >>> np.array(['a', 'bc'], dtype=np.string_)<br>array([b'a', b'bc'], dtype='\|S2') |

| Data Type | Description | Example |
|---|---|---|
| unicode_ | Fixed-size Unicode string | >>> np.array(['hello', 'world'], dtype=np.unicode_)<br>array(['hello', 'world'], dtype='<U5') |

**6. Object Type**

Used for arbitrary Python objects.

| Data Type | Description | Example |
|---|---|---|
| object_ | Python object type | >>> np.array([1, 'string', 3.14], dtype=np.object_)<br>array([1, 'string', 3.14], dtype=object) |

**7. Datetime and Timedelta Types**

This is used for handling dates, times, and durations.

| Data Type | Description | Example |
|---|---|---|
| datetime64 | Date and time representation | >>> np.array(['2024-12-17'], dtype=np.datetime64)<br>array(['2024-12-17'], dtype='datetime64[D]') |
| timedelta64 | Duration or time difference | >>> np.array([10, 20], dtype='timedelta64[D]')<br>array([10, 20], dtype='timedelta64[D]') |

# Module 2. Data Analysis

## Module Overview

Imagine a shopkeeper who records daily sales in a notebook. If the notebook only stores numbers, it becomes difficult to quickly understand which product is selling the most, how sales are changing over time, or whether there is a seasonal trend. By organizing this data into tables, analyzing it with formulas, and displaying it in graphs, the shopkeeper can easily identify patterns and make better business decisions. This is exactly what data analysis does—it helps us organize, process, and visualize raw data to extract meaningful insights.


Fig. 2.0: Data analysis by the shopkeeper

In this Module, you will learn how to use Python libraries like NumPy, Pandas, and Matplotlib to handle, analyze, and visualize data. From creating and manipulating Series and DataFrames, to importing and exporting data, applying statistical operations, and finally presenting results in the form of charts and graphs, this Module will equip you with the essential skills needed for real-world data analysis.

## Learning Outcome

After completing this module, you will be able to:

- Setup the Pandas environment and understand its role in data analysis.
- Load external datasets (CSV, Excel) into Python.
- Create and manipulate 1D data arrays with custom indexing.
- Perform efficient vectorized math operations on data sequences.
- Perform CRUD operations (Create, Read, Update, Delete) on tabular data.
- Filter and clean datasets using conditional logic and missing value handling.
- Generate line, bar, and scatter plots using Matplotlib.
- Format charts with labels, legends, and colors for professional reporting.

## Module Structure

Session 1. Introduction to Pandas

Session 2. Pandas Series

Session 3. Pandas DataFrame

Session 4. Data Visualisation using Matplotlib



## Session 1. Introduction to Pandas

Imagine you are handling marks of all students in your class. If the data is stored only in a notebook, it is hard to search, sort, or calculate quickly. But if it is arranged neatly in a table on a computer, you can instantly find the highest marks, calculate averages, or compare results. Pandas helps us do exactly this with data.

Fig. 1.1 handling marks of all students

In this session, the students will learn the basics of Pandas, its advantages over lists and dictionaries, and how to create and use Series and DataFrames for organizing data. Students will learn to create data structures using lists, arrays, and dictionaries.

### 1.1 INTRODUCTION

PANDAS (PANel DAta) is a open-source library specially built for Python Programming. It is a high-level data manipulation tool used for analysing data. It is used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data. It is built on top of NumPy and provides easy-to-use data structures and data analysis tools. It is very easy to import and export data using Pandas library which has a very rich set of functions. It is built on packages like NumPy and Matplotlib and gives us a single, convenient place to do most of our data analysis and visualisation work. Pandas has three important data structures, namely – *Series, DataFrame* and *Panel* to make the process of analysing data organised, effective and efficient.

The name "Pandas" has a reference to both "*Panel Data*", and *"Python Data Analysis"* and was created by Wes McKinney in 2008.

The term Pandas is derived from "Panel Data System", which is an echometric term for multidimensional, structured dataset. Pandas is widely used in data science, machine learning, and analytics due to its efficiency and flexibility. Relevant data is very important in data science. Pandas can clean messy data sets, and make them readable and relevant.

### 1.1.1 Need of Pandas

It is obvious to think about the need for Pandas when NumPy can be used for data analysis. Following are some of the points that gives the importance of Pandas over Numpy:

1. A Numpy array requires homogeneous data, while a Pandas DataFrame can have different data types (float, int, string, datetime, etc.).
2. Pandas have a simpler interface for operations like file loading, plotting, selection, joining, GROUP BY, which come very handy in data-processing applications.
3. Pandas DataFrames (with column names) make it very easy to keep track of data.
4. Pandas is used when data is in Tabular Format, whereas Numpy is used for numeric array-based data manipulation.

### 1.1.2 Pandas for Data Analysis

Pandas is the ideal tool used for data analysis for following reasons.

1. Easy-to-use syntax for data manipulation and transformation.
2. Efficient handling of large datasets.
3. Seamless integration with other Python libraries such as Matplotlib and NumPy.
4. Broad support for varied data formats such as CSV, SQL and Excel.

Pandas optimizes memory usage and computation speed. Supports filtering, aggregating, and transforming data with minimal code. Pandas' intuitive syntax makes complex tasks straightforward. Data loading and storage is easy in Pandas. Data exploration, data cleaning, data transformation, data merging, data aggregation and grouping can be done in Pandas with ease. The time series analysis can also be done in Pandas.

### 1.1.3 Difference between Numpy and Pandas

NumPy and Pandas are two of the most widely used libraries in Python for data manipulation and analysis. While they have overlapping functionalities, they are designed for different purposes and have distinct features. Below is a detailed comparison:

**Key Differences Between NumPy and Pandas**

| Feature | NumPy | Pandas |
|---|---|---|
| **Primary Purpose** | Numerical computations with n-dimensional arrays. | Data manipulation and analysis with labeled data structures. |
| **Data Structure** | ndarray (multi-dimensional array). | Series (1D) and DataFrame (2D). |
| **Flexibility** | Efficient for numeric and homogeneous data. | Handles mixed data types (e.g., numeric, strings, and dates). |
| **Data Indexing** | Indexed by integer positions. | Indexed with labels (row/ column names) and integers. |

| Feature | NumPy | Pandas |
|---|---|---|
| **Ease of Use** | Requires more manual work for data manipulation. | Provides high-level methods for cleaning, transforming, and exploring data. |
| **File I/O** | Limited support for reading/writing files. | Extensive support for formats like CSV, Excel, SQL, JSON. |
| **Performance** | Faster for numerical operations and larger datasets. | Slightly slower due to higher-level abstractions. |
| **Data Handling** | Cannot handle missing data directly (e.g., NaN requires extra handling). | Built-in support for missing data (e.g., NaN, None). |

## 1.2 INSTALLATION OF PANDAS

To install Python libraries, it is essential that Python is installed on the computer system.

So, first check if your system is pre-installed with Python or not by issuing the following command on the command prompt.

python –version

If Python is already installed, it will generate a message with the Python version available, else first install Python,

Pandas can be installed in multiple ways on Windows, Linux and MacOS as listed below:

Pandas can be installed using Python's package manager, pip, or via conda if you're using the Anaconda distribution.

### 1.2.1 Install Pandas using pip

Pandas can be installed using PIP by using the following command.

**pip install pandas**

PIP is a package management system used to install and manage software packages/libraries written in Python. These files are stored in a large “online repository” termed as Python Package Index (PyPI).

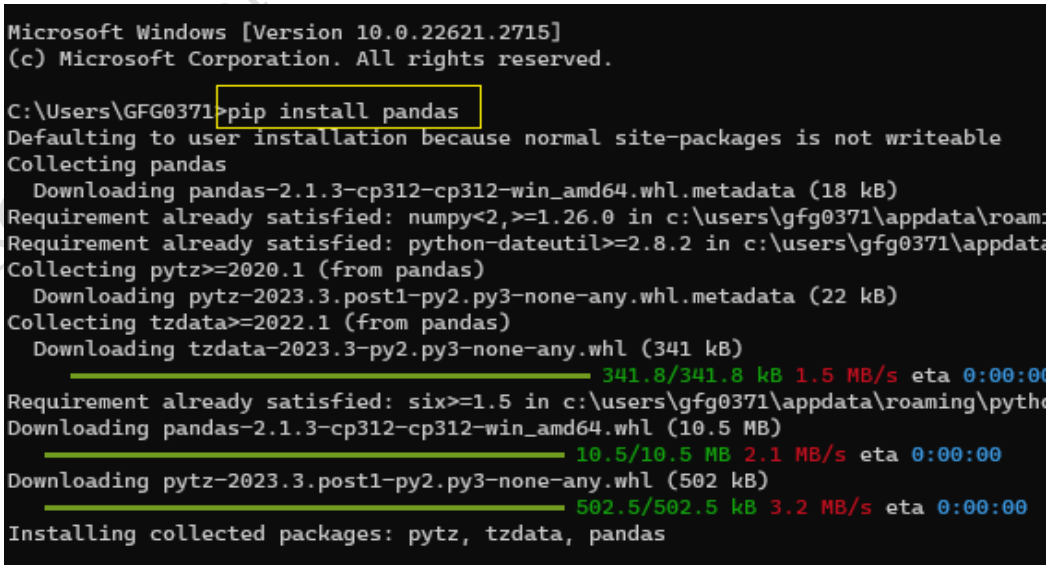



Fig. 1.2: Installed Pandas

**Import Pandas**

Once Pandas is installed, import the pandas package and start working with it. import by adding the import keyword as follows:

import pandas

Now Pandas are imported and ready to use.

**Pandas as pd**

In Python alias are an alternate name for referring to the same thing. Loading pandas as pd is assumed standard practice for all of the pandas documentation. Pandas is usually imported under the pd alias. To create an alias with the as keyword while importing as.

import pandas as pd

Now the Pandas package can be referred to as pd instead of pandas. Here pd is an object of Pandas library to which you can use in your program. Following example illustrate to use pandas package.

**Example:** Using pandas package

## 1.3 DATA STRUCTURE IN PANDAS

A data structure is a collection of data values and operations that can be applied to that data. It enables efficient storage, retrieval and modification to the data. Two commonly used data structures in Pandas are: *Series* and *DataFrame.*

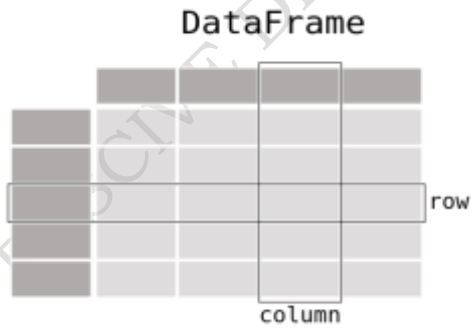
***Series:*** a one-dimensional labeled array holding data of any type such as integers, strings, Python objects etc.

***Dataframe:*** a two-dimensional data structure that holds data like a two-dimension array or a table with rows and columns.

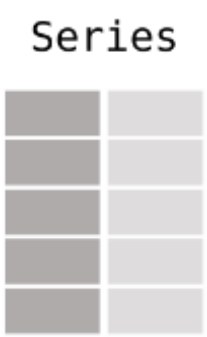
Pandas data structures is as flexible containers for lower dimensional data. For example, DataFrame is a container for Series, and Series is a container for scalars.

**Pandas data table representation**

A DataFrame is a 2-dimensional data structure that can store data of different types (including characters, integers, floating point values, categorical data and more) in columns. It is similar to a spreadsheet, a SQL table.



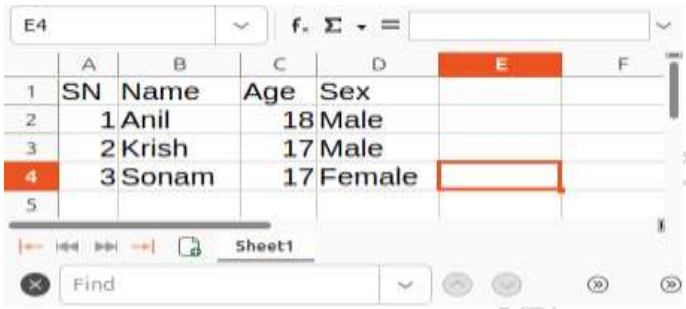

Each column in a DataFrame is a Series




Consider a table of first three students. The table has 3 columns, each of them with a column label. The column labels are respectively Name, Age and Sex. The column Name consists of textual data with each value a string, the column Age are numbers and the column Sex is textual data.

**In spreadsheet, the table representation of this data would look as shown below:**

Similar to spreadsheet software, pandas represent data as a table with columns and rows. Apart from representation, data manipulations and calculations which you are doing in spreadsheet are also supported by pandas.




**Comparison of Series and DataFrame**

| **Feature** | **Series** | **DataFrame** |
|---|---|---|
| Dimensions | One-dimensional | Two-dimensional |
| Indexing | Single index | Row and column indexing |
| Data Type | Homogeneous (same type for all data) | Heterogeneous (different types for columns) |
| Structure | Similar to a single column or array | Similar to a spreadsheet or SQL table |

Apart from series and dataframe we can also have *index* and *panel* as data structures in Pandas.

## Summary

- Pandas is an open-source Python library for data analysis, built on NumPy.
- It offers easy-to-use data structures: Series (1D), DataFrame (2D), and Panel (3D).
- Useful for analyzing, cleaning, exploring, and manipulating datasets.

- Provides advantages over lists and dictionaries: handles mixed data types, easy indexing, powerful data-processing functions.
- Commonly used for tabular data (rows & columns) with better readability than raw arrays.

## Check your progress

**A. Multiple choice questions**

1. What is Pandas in Python primarily used for? a) Building websites b) Numerical calculations with arrays c) Working with tabular data and analysis d) Creating video games
2. Pandas introduces which two main data structures? a) Lists and Dictionaries b) Arrays and Matrices c) Series and DataFrame d) Sets and Tuples
3. Which of these is a one-dimensional labeled array in Pandas? a) DataFrame b) Series c) Panel d) Table
4. A Pandas DataFrame is like a: a) Single column of data b) Two-dimensional table with rows and columns c) Simple list d) Mathematical equation
5. How do you usually import Pandas in your Python code? a) import panda b) import pandas as pd c) include pandas d) use pandas
6. Which other Python library is Pandas built on top of? a) Matplotlib b) Scikit-learn c) NumPy d) SciPy
7. To see the first few rows of a DataFrame named my_data, what function would you use? a) my_data.first() b) my_data.top() c) my_data.head() d) my_data.start()
8. What does NaN represent in a Pandas DataFrame or Series? a) A number equal to zero b) A string value c) Missing or undefined data d) A calculation error
9. If you create a Series from a list like [10, 20, 30] without specifying an index, what will the default index be? a) ['A', 'B', 'C'] b) [1, 2, 3] c) [0, 1, 2] d) ['item1', 'item2', 'item3']
10. Pandas is especially useful for handling: a) Images b) Tabular data, like spreadsheets c) Audio files d) Binary data

**B. Fill in the blanks**

1. Pandas is a popular Python library used mainly for data _____ and analysis.
2. The two main data structures in Pandas are Series and _____.
3. A Pandas Series is a _____-dimensional labeled array.
4. A Pandas DataFrame is like a _____-dimensional table or a spreadsheet.
5. To start using Pandas, you typically import it using import pandas as _____.
6. Pandas is built on top of the _____ library, which provides efficient array operations.
7. The method df._____() is used to view the first few rows of a DataFrame df.

8. Missing data points in Pandas are commonly represented as NaN, which stands for _____ a Number.
9. When creating a Series from a list without specifying an index, the default index will be integer-based, starting from _____.
10. Pandas is particularly good at working with structured data, often referred to as _____ data

**C. State whether True or False**

1. Pandas is a Python library used for creating websites.
2. The two main data structures in Pandas are Series and DataFrame.
3. A Pandas Series is a two-dimensional labeled array.
4. A Pandas DataFrame is like a table with rows and columns.
5. You usually import Pandas using import pandas as pd.
6. Pandas is built on top of the Matplotlib library.
7. The df.head() method shows the last 5 rows of a DataFrame.
8. NaN in Pandas means "Not a Number" and represents missing data.
9. When you create a Series from a list, the default index starts from 1.
10. Pandas is very good at handling data that looks like a spreadsheet.

**D. Answer the following questions in short**

1. What is pandas in Python?
2. How to install Pandas?
3. What is the difference between a Series and a DataFrame?
4. Mention the different types of Data Structures in Pandas
5. Define Series in Pandas?
6. Define DataFrame in Pandas?
7. What is an index in pandas?



## Session 2. Pandas Series

Imagine you are managing the marks of students in a class. You may store the marks in a notebook (like a simple list), but then searching for a particular student's marks or calculating averages becomes time-consuming. Now think of arranging the marks in a neat table where each student's name (label) is linked to their marks (value). This makes it much easier to find, compare, and analyze.


Fig. 2.1: Introducing the Pandas series

A Pandas Series works in a similar way. It is like a single column in a spreadsheet where each piece of data (marks) has a label (student name or index). This helps us retrieve, analyze, and manipulate data more efficiently than using plain lists.

In this session, you will be able to create Pandas Series in different ways, access and slice data using labels and positions, explore useful attributes and methods, perform mathematical operations, and carry out basic statistical calculations on data for easier analysis.

## 2.1 INTRODUCTION

A Pandas Series is a powerful one-dimensional labeled array in Python, analogous to a single column in a spreadsheet. It can hold various data types, like integers, floats, strings, and other Python objects. The data label associated with a particular value is called its index. The axis labels are collectively referred to as the index. Each value in a Series is associated with an **index**, which makes data retrieval and manipulation easy.

## 2.2 CREATION OF SERIES

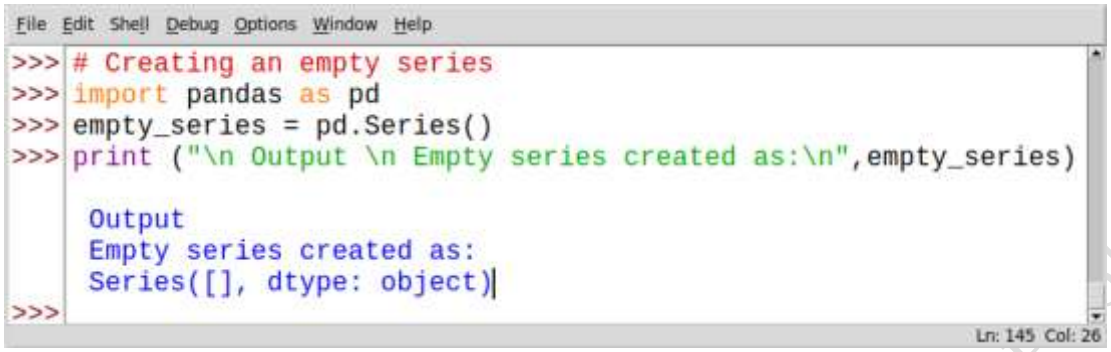
There are different ways to create a series in Pandas. To create or use series, we first need to import the Pandas library.

### 1. Creating an empty series

An empty Series can be created using the pd.Series() constructor without data or index.

```
# Creating an empty series
import pandas as pd
empty_series = pd.Series()
print ("\n Output \n Empty series created as:\n",empty_series)
```

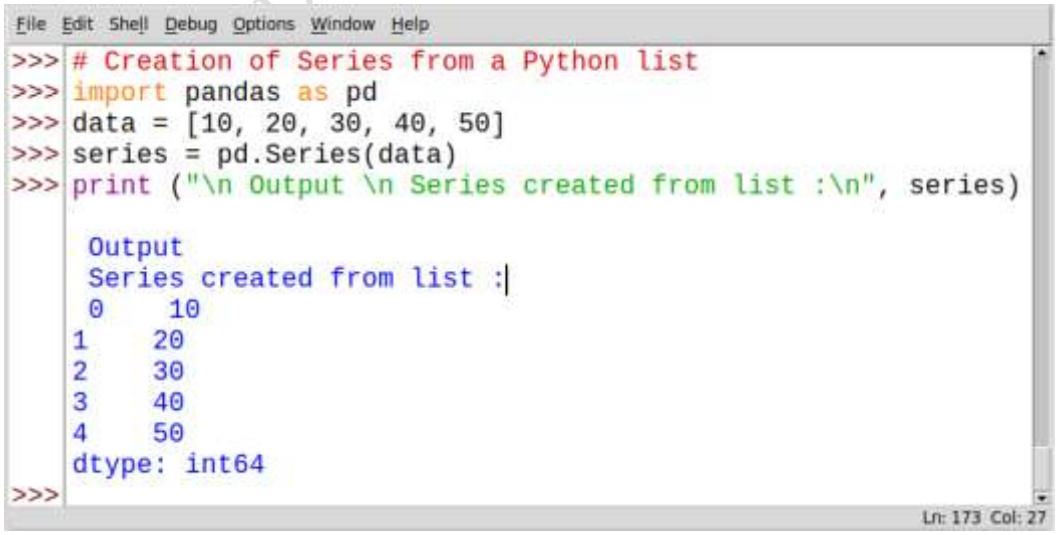
```
Output
 Empty series created as:
 Series([], dtype: object)
```




**2. Creation of Series from a Python list**

This is one of the simplest methods. You can pass a Python list directly to the pd.Series() constructor. Pandas will automatically assign a default numeric index starting from 0.

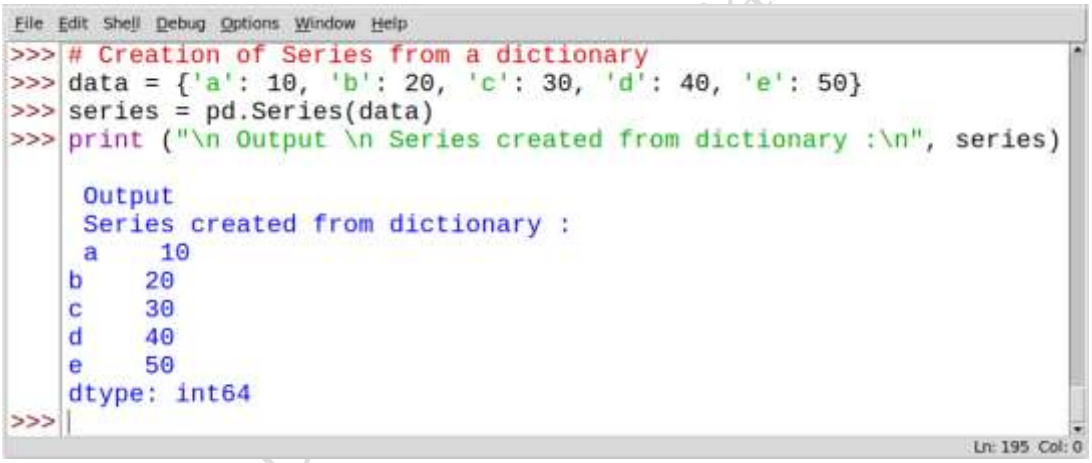
```
# Creation of Series from a Python list
import pandas as pd
data = [10, 20, 30, 40, 50]
series = pd.Series(data)
print ("\n Output \n Series created from list :\n", series)

 Output
 Series created from list :
 0    10
1    20
2    30
3    40
4    50
dtype: int64
```



### 3. Creation of Series from a dictionary

When creating a Series from a dictionary, the keys of the dictionary become the index labels, and the dictionary values become the data of the Series. This method is useful for labeled data preserving structure and enabling quick access.

```
# Creation of Series from a dictionary
data = {'a': 10, 'b': 20, 'c': 30, 'd': 40, 'e': 50}
series = pd.Series(data)
print ("\n Output \n Series created from dictionary :\n", series)

 Output
 Series created from dictionary :
 a    10
b    20
c    30
d    40
e    50
dtype: int64
```

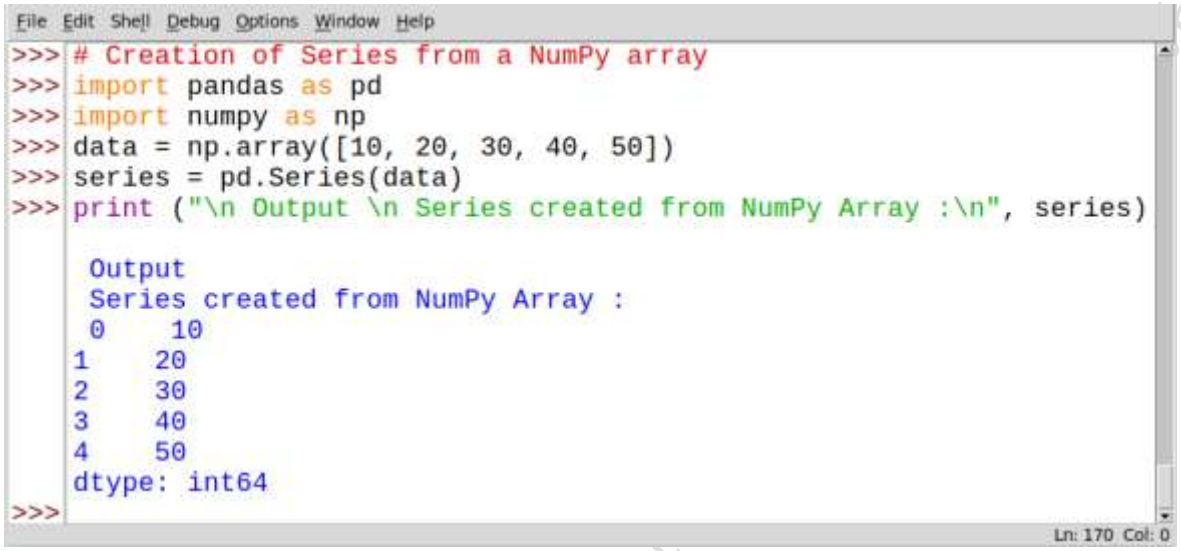



### 4. Creation of Series from a NumPy array

A NumPy array can be converted into a Pandas Series. Similar to lists, a default numeric index is assigned if not specified. This is helpful when working with numerical data.

```
# Creation of Series from a NumPy array
import pandas as pd
import numpy as np
data = np.array([10, 20, 30, 40, 50])
series = pd.Series(data)
print ("\n Output \n Series created from NumPy Array :\n", series)

 Output
```

```
 Series created from NumPy Array :
 0    10
1    20
2    30
3    40
4    50
dtype: int64
```




**5. Creation of Series from a scalar value**

A Series can be created from a single scalar value by providing an index, and the scalar value will be repeated for each index label.

```
# Creation of Series from a scalar value
import pandas as pd
scalar = 10
index = ['a', 'b', 'c', 'd', 'e']
series = pd.Series(scalar, index=index)
print ("\n Output \n Series created from Scalar Value :\n", series)

 Output
 Series created from Scalar Value :
 a    10
b    10
c    10
d    10
e    10
dtype: int64
```

```
File Edit Shell Debug Options Window Help
>>> # Creation of Series from a scalar value
>>> import pandas as pd
>>> scalar = 10
>>> index = ['a', 'b', 'c', 'd', 'e']
>>> series = pd.Series(scalar, index=index)
>>> print ("\n Output \n Series created from Scalar Value :\n", series)

 Output
 Series created from Scalar Value :
 a    10
b    10
c    10
d    10
e    10
dtype: int64
>>>
                                                              Ln: 242 Col: 7
```

**6. Creation of Series using the range() function**

The range() function can be used to generate sequences of numbers for a Pandas Series.

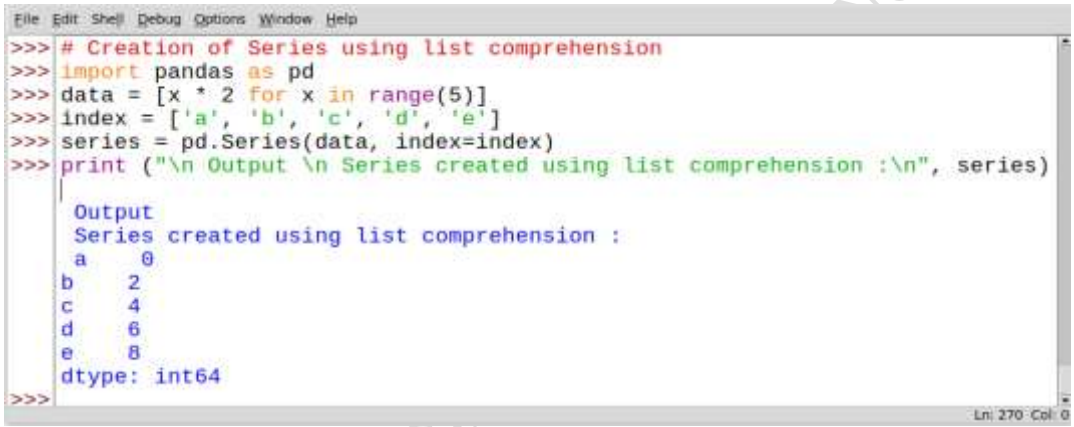
```
import pandas as pd

series = pd.Series(range(5))
print(series)

Output:
0    0
1    1
2    3
3    3
4    4
dtype: int64
```

```
File Edit Shell Debug Options Window Help
>>> # Creation of Series using the range() function
>>> import pandas as pd
>>> series = pd.Series(range(5))
>>> print ("\n Output \n Series created from range () function :\n", series)

 Output
 Series created from range () function :
 0    0
1    1
2    2
3    3
4    4
dtype: int64
>>>
                                                              Ln: 261 Col: 12
```

**7. Creation of Series using list comprehension**

List comprehensions can create sequences and apply transformations before converting to a Series, potentially with a custom index.

```
import pandas as pd

data = [x * 2 for x in range(5)]
index = ['a', 'b', 'c', 'd', 'e']
series = pd.Series(data, index=index)
print(series)

Output:
a    0
b    2
c    4
d    6
e    8
dtype: int64
```




## 2.3 ACCESSING ELEMENTS OF A PANDAS SERIES

Like a Python list or NumPy array, a Pandas Series, allows to access individual elements or subsets of elements. However, unlike lists or NumPy arrays, Pandas Series elements can be accessed not only by their integer position but also by their unique label (index).

There are two ways for accessing the elements of a series: *Indexing* and *Slicing.*

### 2.3.1 Indexing

Indexing in Series is similar to that for NumPy arrays, and is used to access elements in a series. Indexes are of two types: *positional index* and *labelled index*.

Positional index takes an integer value that corresponds to its position in the series starting from 0, whereas labelled index takes any user-defined label as index.

**1. Accessing by positional index (integer-based indexing)**

Pandas Series elements can be accessed by their numerical position, starting from 0, similar to how you would access elements in a list.

```
import pandas as pd
series = pd.Series([10, 20, 30, 40, 50])
# Access the first element
print(series[0])

Output
10

# Access the element at index 2 (third element)
print(series[2])

Output
30
```

**2. Accessing by label (label-based indexing)**

If your Pandas Series has custom labels (indices), you can use them to access the corresponding elements.

```
# Accessing by label (label-based indexing)
import pandas as pd

series = pd.Series([10, 20, 30, 40, 50], index=['a', 'b', 'c', 'd',
'e'])

# Access the element with label 'c'
print(series['c'])

# Access multiple elements by providing a list of labels
print(series[['a', 'd']])
Output
30
a    10
d    40
dtype: int64
```



### 2.3.2 Slicing

Sometimes, we may need to extract a part of a series. Slicing allows you to select a range of elements from a Series. This is similar to slicing used with NumPy arrays. We can define which part of the series is to be sliced by specifying the start and end parameters [start :end] with the series name. When we use positional indices for slicing, the value at

the endindex position is excluded, i.e., only (end - start) number of data values of the series are extracted.

```
import pandas as pd

series = pd.Series([10, 20, 30, 40, 50])

# Slice the first three elements
print(series[:3])

# Slice elements from index 1 to 3 (inclusive of the start, exclusive of the end in Python-like slicing)
print(series[1:4])

# Slice the last three elements
print(series[-3:])
Output:
0    10
1    20
2    30
dtype: int64
1    20
2    30
3    40
dtype: int64
2    30
3    40
4    50
dtype: int64
```

**1. Using .loc for label-based selection**

The .loc accessor provides label-based indexing for both single elements and slices. It is generally recommended for explicit label-based operations.

```
import pandas as pd

series = pd.Series([10, 20, 30, 40, 50], index=['a', 'b', 'c', 'd', 'e'])

# Access a single element using .loc
print(series.loc['b'])
```

```
# Slice using .loc with labels (inclusive of both start and end labels)
print(series.loc['a':'d'])
```

**Output:**

```
20
a    10
b    20
c    30
d    40
dtype: int64
```

**2. Using .iloc for position-based selection**

The .iloc accessor provides integer-location based indexing, similar to Python's list slicing, according to Built In.

```
import pandas as pd

series = pd.Series([10, 20, 30, 40, 50], index=['a', 'b', 'c', 'd',
'e'])

# Access a single element using .iloc
print(series.iloc[1])

# Slice using .iloc with integer positions (exclusive of the end
position)
print(series.iloc[0:4])
```

**Output:**

```
20
a    10
b    20
c    30
d    40
dtype: int64
```

**Boolean indexing**

You can filter Series elements based on a condition, which creates a Boolean Series (True/False values) used to select the elements that satisfy the condition.

```
import pandas as pd

series = pd.Series([10, 20, 30, 40, 50])

# Select elements greater than 30
print(series[series > 30])
Output:
3    40
4    50
dtype: int64
```

These are the primary methods for accessing elements in a Pandas Series. These are useful for efficient data manipulation Choosing the right method depends on whether you need to access elements by their integer position or by their label.

### 2.4 ATTRIBUTES OF PANDAS SERIES

A Pandas Series is a powerful data structure with several built-in attributes that provide essential information about its contents and structure. These attributes allow you to understand and manipulate your data effectively,

Here are some of the most commonly used attributes:

**1. .index**

The .index attribute returns the labels of the Series, allowing you to access elements using custom names or labels.

```
# Labels of the Series using .index attribute to
import pandas as pd

series = pd.Series([10, 20, 30, 40, 50], index=['a', 'b', 'c', 'd',
'e'])
print(series.index)

Output:
Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
```

**2. .values**

The .values attribute returns the data stored in the Series as a NumPy array, providing access to the raw data without the index labels

```
# Data stored in the Series as NumPy array using .values attribute

import pandas as pd

series = pd.Series([10, 20, 30, 40, 50], index=['a', 'b', 'c', 'd',
```

```
'e'])
print(series.values)

Output:
[10 20 30 40 50]
```

**3. .dtype**

The .dtype attribute returns the data type of the elements in the Series. This is crucial for ensuring compatibility with various operations.

```
# Data type of the elements in the Series using .dtype attribute

import pandas as pd

series = pd.Series([10, 20, 30, 40, 50])
print(series.dtype)

Output:
int64
```

**4. .size**

The .size attribute returns the total number of elements in the Series, including any missing (NaN) values.

```
# Total number of elements in the Series using .size attribute
import pandas as pd

series = pd.Series([10, 20, 30, 40, 50])
print(series.size)

Output:
5
```

**5. .shape**

The .shape attribute returns the shape of the Series as a tuple. For a Series, this will always be a single dimension.

```
# Shape of the Series as a tuple using .shape attribute
import pandas as pd

series = pd.Series([10, 20, 30, 40, 50])
print(series.shape)

Output:
(5,)
```

**6. .name**

The .name attribute allows you to assign or retrieve the name of the Series. Naming a Series can be useful when working with DataFrames or merging data.

```
# Name of the Series using .name attribute
import pandas as pd

series = pd.Series([10, 20, 30, 40, 50], name='My Series')
print(series.name)

Output:
My Series
```

**7. .ndim**

The .ndim attribute returns the number of dimensions of the underlying data. For a Series, this value is always 1.

```
# Checking the number of dimensions of series using .ndim attribute
import pandas as pd

series = pd.Series([10, 20, 30, 40, 50])
print(series.ndim)

Output:
1
```

**8. .empty**

The .empty attribute checks if a Series is empty (contains no elements), returning True if it's empty and False otherwise.

```
# Checking if a Series is empty using .empty attribute
import pandas as pd

series1 = pd.Series([])
series2 = pd.Series([1, 2, 3])
print(series1.empty)
print(series2.empty)

Output:
True
False
```

**9. .hasnans**

The .hasnans attribute checks for the presence of missing values (NaN) within the Series. It returns True if any NaN values are present, False otherwise.

```
9. .hasnans
The .hasnans attribute checks for the presence of missing values (NaN)
within the Series. It returns True if any NaN values are present, False
otherwise.

Output:
True
False
```

These attributes provide valuable information about your Pandas Series, enabling you to inspect and manipulate your data effectively.

### 2.5 PANDAS SERIES METHODS

Pandas Series offers a wide array of methods. These are functions associated with the Series object used for data manipulation and analysis. Methods perform operations on the Series data, and return new Series objects.

Following are some methods that are used for viewing and inspecting data. You can use the methods .head(), .tail(), and .count() to inspect and analyze the data.

**head(n):** Retrieves the first 'n' rows (default to 5), useful for quickly reviewing the beginning of a Series.

**tail(n):** Retrieves the last 'n' rows (default to 5), useful for verifying data loading and structure.

**unique():** Returns the unique values present in the Series.

**nunique():** Returns the count of unique elements in the Series.

**count() :** Returns the number of non-NaN values in the Series.

**value_counts():** Returns a new Series containing the frequency of each unique value.

```
import pandas as pd

# Create a Series
data = pd.Series([10, 20, 30, 40, 50, 60, None, 80])

# Display first 5 elements
print("Head:\n", data.head())

# Display last 5 elements
print("\nTail:\n", data.tail())

# Count non-NaN (non-missing) values
print("\nCount of non-NaN values:", data.count())
```

```
Output

Head:
0    10.0
1    20.0
2    30.0
3    40.0
4    50.0
dtype: float64

Tail:
3    40.0
4    50.0
5    60.0
6     NaN
7    80.0
dtype: float64

Count of non-NaN values: 7
```



## 2.6 MATHEMATICAL OPERATIONS ON SERIES

You can perform mathematical operations on Pandas Series just like with NumPy arrays or Python lists. While performing mathematical operations on series, index matching is implemented and all missing values are filled in with NaN by default.

### 1. Operations with scalars

You can perform basic arithmetic operations (addition, subtraction, multiplication, division, modulo, exponentiation, and floor division) directly on a Series using Python's standard operators ( +, -, *, /, %, **, // ). This applies the operation to each element of the Series, resulting in a new Series

| Operation | Operator | Description |
|---|---|---|
| Addition | + | Adds a scalar to each element |
| Subtraction | - | Subtracts a scalar from each element |
| Multiplication | * | Multiplies each element by a scalar |
| Division | / | Divides each element by a scalar |
| Exponentiation | ** | Raises each element to the power of a scalar |
| Modulus | % | Returns the remainder after dividing each element by a scalar |
| Floor Division | // | Divides each element by a scalar and rounds down to the nearest integer |

**2. Operations between two Series**

When performing operations between two Series, Pandas automatically aligns the data based on their index labels. This is a powerful feature as it ensures that operations are performed on corresponding elements even if the Series have different orders or some indices are missing. If an index is not present in both Series, the result for that index will be NaN (Not a Number).

**1. Addition of Two Series**

Adding two Pandas Series in Python can be achieved using the standard arithmetic operator + or the add() method. Both methods perform element-wise addition, aligning elements based on their index.

**a. Using the + operator:**

Two series can be added simply by using the + operator. Pandas automatically handles index alignment. If an index label exists in one Series but not the other, the corresponding element in the result will be NaN (Not a Number). Following example illustrate to add two series using + operator.

```
# Python code to add two series using + operator
import pandas as pd

series1 = pd.Series([10, 20, 30], index=['a', 'b', 'c'])
series2 = pd.Series([1, 3, 5], index=['a', 'c', 'd'])

series = series1 + series2
print(series)

Output
a     11.0
b     NaN
c     33.0
d     NaN
dtype: float64
```

Here, the 'b' index in series1 and 'd' index in series2 do not have a corresponding element in the other Series, resulting in NaN values in the output. You can also use methods like `.add()`, `.sub()`, `.mul()`, `.div()`, instead of operators, which allows for specifying a fill_value to replace missing values.

| index | Value from seriesA | Value from seriesB | SeriesA + seriesB |
|---|---|---|---|
| a | 10 | 1 | 11 |
| b | 20 | | NaN |
| c | 30 | 3 | 33 |
| d | | 5 | Nan |

***2. Using the add() method:***

The add() method offers more control, particularly for handling missing values. The fill_value parameter allows you to specify a value to substitute for missing data in either Series before performing the addition.

```
# Python code to add two series using add() method
import pandas as pd

series1 = pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])
series2 = pd.Series([5, 6, 7], index=['b', 'c', 'e'])

# Add with fill_value=0 to treat missing values as 0
series_filled = series1.add(series2, fill_value=0)
print(series_filled)

Output
a     1.0
b     7.0
c    10.0
d     4.0
e     7.0
dtype: float64
```

**2. Subttraction of Two Series**

Subtraction of two pandas Series can be performed in two primary ways:

***a. Using the subtraction operator (-):***

This is the most straightforward method and performs element-wise subtraction based on the Series indices. If indices do not align, the result will contain NaN (Not a Number) for the non-matching positions.

```
# Python code to subtract two series using – operator
import pandas as pd
s1 = pd.Series([10, 20, 30, 40], index=['a', 'b', 'c', 'd'])
s2 = pd.Series([5, 15, 25, 35], index=['a', 'b', 'c', 'e'])
result = s1 – s2
print(result)

Output
a     5.0
b     5.0
c     5.0
```

```
d     NaN
e     NaN
dtype: float64
```

**b. Using the sub() method:**

This method provides more control, especially for handling missing values during alignment. It is equivalent to the subtraction operator but offers the fill_value parameter.

```
# Python code to add two series using add() method
import pandas as pd
import numpy as np

s1 = pd.Series([10, 20, 30, np.nan], index=['a', 'b', 'c', 'd'])
s2 = pd.Series([5, np.nan, 25, 35], index=['a', 'b', 'e', 'f'])

# Subtracting with fill_value=0 for missing values
result_filled = s1.sub(s2, fill_value=0)
print(result_filled)

Output

a     5.0
b    20.0
c    30.0
d     0.0
e   -25.0
f   -35.0
dtype: float64
```

In this example, `fill_value=0` ensures that `NaN` values in either Series are treated as 0 for the purpose of subtraction, preventing `NaN` in the result where only one Series has a missing value. If both corresponding locations are `NaN`, the result will still be `NaN`.

**3. Multiplication of two Series**

Multiplication of two Series in pandas can be done using the `*` operator or the `mul()` method. Both perform element-wise multiplication, aligning the Series based on their indices.

***a. Using the * operator:***

This is the most straightforward and commonly used method for element-wise multiplication of two Series.

```
# Python code to subtract two series using * operator
```

```
import pandas as pd

# Create two Series
series1 = pd.Series([1, 2, 3, 4, 5])
series2 = pd.Series([6, 7, 8, 9, 10])

# Multiply the two Series
result_series = series1 * series2

print(result_series)
```

***b. Using the mul() method:***

The mul() method provides more control, especially when dealing with missing values or misaligned indices, through its fill_value parameter.

```
# Python code to add two series using mul() method
import pandas as pd

# Create two Series, one with a missing value and misaligned index
series_a = pd.Series([10, 20, 30], index=['a', 'b', 'c'])
series_b = pd.Series([2, 4, 6, 8], index=['a', 'c', 'd', 'e'])

# Multiply the two Series, filling missing values with 1 before
multiplication
result_mul = series_a.mul(series_b, fill_value=1)

print(result_mul)
```

**4. Division of two Series**

In Pandas, the division of two Series can be performed using either the standard division operator (/) or the div() method. Both approaches perform element-wise division, aligning the Series based on their indices.

***a. Using the Division Operator (/)***

This method is commonly used for basic arithmetic operations.

```
# Python code to subtract two series using * operator
import pandas as pd
# Create two Series
series1 = pd.Series([10, 20, 30, 40])
series2 = pd.Series([2, 5, 10, 8])
# Divide series1 by series2
```

```
result_series = series1 / series2

print(result_series)
```

***b. Using the div() Method***

The div() method provides more control, especially when dealing with missing values or when aligning Series with different indices.

```
# Python code to add two series using div() method
import pandas as pd
# Create two Series
series_a = pd.Series([10, 20, 30, 40], index=['a', 'b', 'c', 'd'])
series_b = pd.Series([2, 5, 10], index=['a', 'b', 'e'])

# Divide series_a by series_b using div()
# Elements where indices don't align will result in NaN
result_div = series_a.div(series_b)

print(result_div)

# Using fill_value to handle missing data during division
result_fill_value = series_a.div(series_b, fill_value=1) # Treat missing
values in series_b as 1

print(result_fill_value)
```



## 2.7 STATISTICAL CALCULATION ON PANDAS SERIES VALUES

You can perform a wide range of **statistical calculations** on a **Pandas Series** using built-in methods. sum(), mean(), median(), min(), max(), count(), std(), var(), perform various statistical calculations on the Series data. These functions are fast, efficient, and automatically ignore NaN values.

```
# Common Statistical Methods on Series
import pandas as pd

data = pd.Series([10, 20, 30, 40, 50, None])
print("Count:", data.count())       # Non-NaN count
print("Sum:", data.sum())           # Total sum
print("Mean:", data.mean())         # Average
print("Median:", data.median())     # Middle value
print("Min:", data.min())           # Minimum
print("Max:", data.max())           # Maximum

Output
```

```
Count: 5
Sum: 150.0
Mean: 30.0
Median: 30.0
Min: 10.0
Max: 50.0
Standard Deviation: 15.81
Variance: 250.0
```

## Summary

- A Series is a one-dimensional labeled array (like a single column in Excel).
- Each value is linked to an index (label), making data retrieval easier.
- Can be created from lists, arrays, or dictionaries.
- Attributes: .dtype, .index, .empty, .hasnans.
- Methods: .head(), .tail(), .count(), .unique(), .value_counts().
- Supports arithmetic operations and automatically aligns by index, filling missing values with NaN.

## Check your progress

**A. Multiple choice questions**

1. Which function is used to create a Pandas Series? (a) pd.series() (b) pd.Series() (c) pandas.create_series() (d) Series()
2. To create an empty Pandas Series, which of the following is the correct syntax? (a) s = pd.Series(empty) (b) s = pd.Series() (c) s = pd.Series(None) (d) Both b and c
3. When creating a Pandas Series from a Python dictionary, what do the dictionary keys become by default? (a) Values of the Series. (b) Data type of the Series. (c) Index labels of the Series. (d) Name of the Series.
4. Which of the following will create a Pandas Series where all elements have the value 5, with index labels 'a', 'b', and 'c'? (a) pd.Series(5, index=['a', 'b', 'c']) (b) pd.Series([5, 5, 5], index=['a', 'b', 'c']) (c) pd.Series({'a': 5, 'b': 5, 'c': 5}) (d) All of the above
5. If you create a Pandas Series from a NumPy array and do NOT specify an index, what will be the default index? (a) Alphabetic labels ('A', 'B', 'C'…) (b) Integer labels starting from 1 (c) Integer labels starting from 0 (d) Randomly generated unique labels
6. Which of the following data types or structures can be used as data to create a Pandas Series? (a) A Python list (b) A Python dictionary (c) A NumPy ndarray (d) All of the above
7. To create a Series where each element has a scalar value that repeats based on the length of the provided index, you must: (a) Provide both a scalar value and an index.

(b) Provide only a scalar value. (c) Provide only an index. (d) It's not possible to create a Series this way.

8. Which attribute of a Pandas Series returns the array of data values in the Series? (a) .index (b) .values (c) .dtype (d) .name
9. What does the .index attribute of a Pandas Series return? (a) A list of the data values. (b) The name of the Series. (c) A Pandas Index object containing the labels for each element. (d) The number of elements in the Series.
10. Which attribute is used to get the number of dimensions of the Series object? (a) .dimensions (b) .ndim (c) .shape (d) .axis
11. The .hasnans attribute of a Series returns True if: (a) The Series contains only integer values. (b) The Series contains at least one NaN (missing) value. (c) The Series has no name. (d) The Series has a custom index.
12. Which attribute provides the memory usage of the Series in bytes? (a) .size (b) .nbytes (c) .memory (d) .itemsize

**B. Fill in the blanks**

1. A Pandas Series is a _____-dimensional labeled array capable of holding any data type.
2. The two primary components of a Pandas Series are its data values and its _____ labels.
3. By default, if no index is specified when creating a Series from a list or NumPy array, Pandas will create a _____ index starting from zero.
4. When creating a Series from a Python dictionary, the dictionary's _____ become the Series' index labels by default.
5. To create a Series where all elements have the same scalar value, you need to provide both the scalar value and a list for the _____ argument.
6. The attribute used to retrieve the underlying NumPy array of data values from a Series is_____.
7. The attribute_____ returns the data type of the elements in the Series.
8. When performing mathematical operations between two Series with unaligned indexes, Pandas will fill the missing positions with _____ values by default.
9. The_____ attribute returns the total number of elements in the Series.
10. If a scalar value is added to a Series, the operation is applied _____wise to each element.
11. The_____ attribute returns True if the Series contains at least one NaN value.

**C. State whether True or False**

1. All elements within a Pandas Series must be of the same data type.
2. The index of Pandas Series must always be a sequence of integers starting from 0.
3. You can create a Pandas Series from a Python dictionary.

4. When performing arithmetic operations between two Series with different indexes, the operation will only be performed on the common index labels.
5. The .values attribute of a Series returns the index labels.
6. The pd.Series() constructor can be used to create an empty Series.
7. Adding a scalar value to a Series will add the scalar to each element in the Series.
8. The .dtype attribute returns the number of elements in the Series.
9. The .name attribute allows you to assign a descriptive name to the Series.
10. If you try to create a Series from a list of 5 elements and provide an index list of 4 elements, it will result in an error.
11. The NaN value in a Series indicates a missing or undefined data point.

**D. Answer the following questions in short**

1. What is a Pandas Series, in your own words?
2. Name the two main components that make up a Pandas Series.
3. How do you typically import the Pandas library to work with Series?
4. Explain how the index is determined when creating a Series from a Python dictionary.
5. What happens when you perform a mathematical operation (e.g., addition, subtraction) between a Pandas Series and a single scalar value?
6. What is the purpose of the .dtype attribute of a Series?
7. Describe the default index that Pandas assigns when a Series is created from a list or NumPy array without explicitly providing an index.
8. What value does Pandas use to represent missing or undefined data points within a Series, and what is its typical data type implication?
9. If you have two Series with different index labels, and you add them together, how does Pandas handle the index alignment, especially for labels that are not common to both?
10. Briefly explain the difference between the *.values* attribute and the *.index* attribute of a Series.



## Session 3. Pandas DataFrame

Imagine you are maintaining a school record of students where each row represents a student and each column represents details such as name, age, marks, and city. In a notebook, this information may look like a table, but searching, updating, or analyzing becomes tedious. A Pandas DataFrame works like a digital version of this table — organized into rows and columns with labels, where each column can hold different types of data. Just like a school record can be expanded with new subjects (columns) or new students (rows), DataFrames allow adding, removing, and modifying data easily, making them a powerful tool for managing and analyzing real-world datasets.

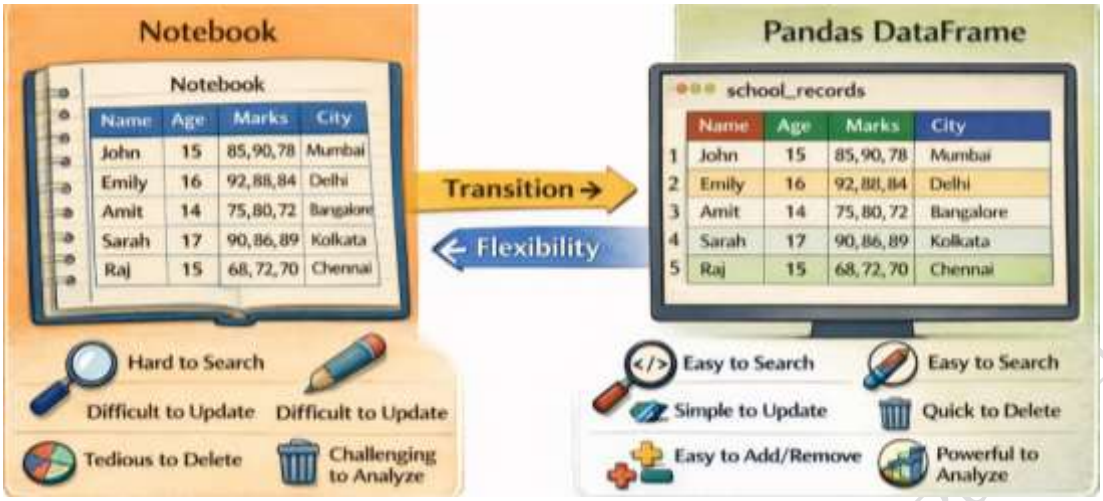



Fig. 3.1: School records and Pandas DataFrames

In this session, you will be able to create Pandas DataFrames from different sources, access and manipulate rows and columns, use indexing methods to retrieve data, explore key attributes, and perform common operations like adding, deleting, renaming, and analyzing data in a tabular format.

### 3.1 INTRODUCTION

Pandas DataFrame is a way to represent and work with tabular data. It can be seen as a table that organizes data into rows and columns, making it a two-dimensional data structure. It is a fundamental data structure for data manipulation and analysis in Python. DataFrames can hold data of various types such as numeric, string, boolean, in different columns. Like Series, DataFrame accepts many different kinds of input.

**Key characteristics of a Pandas DataFrame:**

**Tabular structure:** Data is organized in rows and columns, similar to a table.

**Labeled axes:** Both rows and columns have labels, known as index and column names, respectively.

**Heterogeneous data types:** Columns can contain data of different types.

**Size mutability:** Rows and columns can be added or removed after the DataFrame is created.

**Operations:** Pandas provides a rich set of functions and methods for data manipulation, including filtering, sorting, grouping, merging, and reshaping.

### 3.2 CREATING PANDAS DATAFRAME

A DataFrame can be created from scratch, or from various data sources, such as: Dictionaries of lists or NumPy arrays, 2-D NumPy arrays, CSV files, and SQL databases. The syntax to create DataFrame is as below.

```
pandas.DataFrame(data, index, columns)
```

Along with the data, you can optionally pass index (row labels) and columns (column labels) arguments. If you pass an index and / or columns, you are guaranteeing the index and / or columns of the resulting DataFrame. Thus, a dict of Series plus a specific index will discard all data not matching up to the passed index.

**1. Creation of an Empty DataFrame**

An empty DataFrame in pandas is a table with no data but can have defined column names and indexes. It is useful for setting up a structure before adding data dynamically. An empty DataFrame() can be created just by calling a dataframe constructor. Following example illustrate to create an empty dataframe. It will result in an empty DataFrame with zero rows and zero columns.

```
# Cteating an Empty DataFrame
import pandas as pd
# Create an empty DataFrame
df = pd.DataFrame()
print(df)

Output

Empty DataFrame
Columns: []
Index: []
```

Alternatively, an empty DataFrame can be created with predefined column names or index labels by passing the columns or index parameters to the DataFrame() constructor.

```
# Creating an empty DataFrame with specified columns

import pandas as pd

# Create an empty DataFrame with specified columns
df_cols = pd.DataFrame(columns=['Name', 'Age', 'City'])

print(df_cols)
```

```
# Creating an empty DataFrame with specified index
import pandas as pd
# Create an empty DataFrame with specified index
df_index = pd.DataFrame(index=['Row1', 'Row2'])

print(df_index)
```

```
File Edit Shell Debug Options Window Help
>>> # Cteating an Empty DataFrame
>>> import pandas as pd
>>> df = pd.DataFrame()
>>> print("Empty DataFrame is created as:",df)
Empty DataFrame is created as: Empty DataFrame
Columns: []
Index: []
>>>
>>> # Creating Empty DataFrame with specified columns
>>> df_cols = pd.DataFrame(columns=['Name','Age','City'])
>>> print("Empty DataFrame with specified columns :",df_cols)
Empty DataFrame with specified columns : Empty DataFrame
Columns: [Name, Age, City]
Index: []
>>>
>>> # Creating Empty DataFrame with specified index
>>> import pandas as pd
>>> df_index = pd.DataFrame(index=['Row1', 'Row2'])
>>> print("Empty DataFrame with specified columns :",df_index)
Empty DataFrame with specified columns : Empty DataFrame
Columns: []
Index: [Row1, Row2]
>>>
                                                    Ln: 32 Col: 21
```

**2. Creation of DataFrame from NumPy ndarrays**

A Pandas DataFrame can be created from a NumPy ndarray using the pd.DataFrame() constructor. Each key in the dictionary represents a column name and the corresponding NumPy array provides the values for that column.

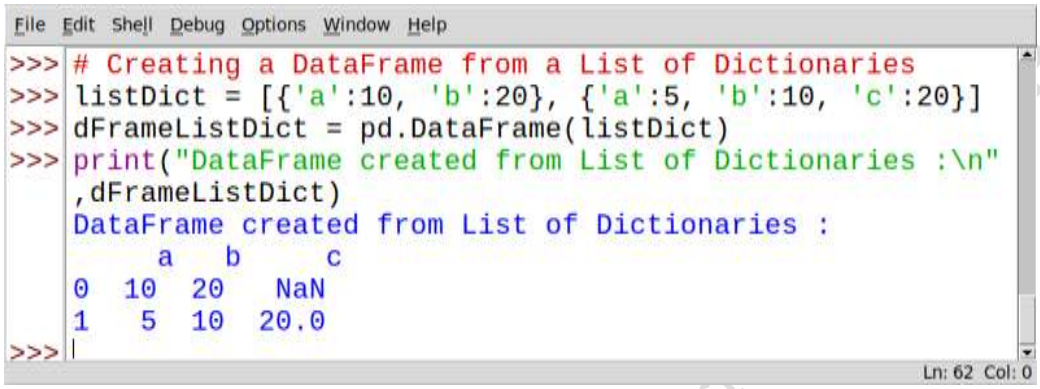
```
# Creation of DataFrame from NumPy ndarrays
import numpy as np
import pandas as pd
data = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
df = pd.DataFrame(data, columns=['A', 'B', 'C'])
print("Output:\n",df)
```

```
File Edit Shell Debug Options Window Help
>>> import numpy as np
>>> import pandas as pd
>>> data = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
>>> df = pd.DataFrame(data, columns=['A', 'B', 'C'])
>>> print("Output:\n",df)
Output:
   A  B  C
0  1  2  3
1  4  5  6
2  7  8  9
>>>
                                                    Ln: 224 Col: 21
```

**3. Creating a DataFrame from a List of Dictionaries**

It is also possible to create dataframe using List of Dictionaries. Each dictionary in the list will typically represent a row in the DataFrame, with the dictionary keys becoming the column names.

```
# Creating a DataFrame from a List of Dictionaries
>>> listDict = [{'a':10, 'b':20}, {'a':5, 'b':10, 'c':20}]
>>> dFrameListDict = pd.DataFrame(listDict)
>>> dFrameListDict

Output
      a     b     c
0     10    20    NaN
1     5     10    20.0
```




Here, the dictionary keys are taken as column labels, and the values corresponding to each key are taken as rows. There will be as many rows as the number of dictionaries present in the list. In the above example there are two dictionaries in the list. So, the DataFrame consists of two rows. Number of columns in a DataFrame is equal to the maximum number of keys in any dictionary of the list. Hence, there are three columns as the second dictionary has three elements. Also, note that NaN (Not a Number) is inserted if a corresponding value for a column is missing.

**4. Creation of DataFrame from Series**

A Pandas DataFrame can be created from one or more Pandas Series.

**1. Using pd.DataFrame() constructor**

A Pandas DataFrame can be created using pd.DataFrame() constructor from one or more Pandas Series.

***a. From a single Series:***

You can pass a Series directly to the pd.DataFrame() constructor. The Series' values will form the new DataFrame's column, and its index will become the DataFrame's index.

```
# Creation of DataFrame from a single series
import pandas as pd

# Create a Series
s = pd.Series([10, 20, 30, 40, 50], index = ['a', 'b', 'c', 'd', 'e'])

# Create a DataFrame from the Series
```

```
df = pd.DataFrame(s)

print(df)
```

```
File Edit Shell Debug Options Window Help
>>> # Creation of DataFrame from a single series
>>> import pandas as pd
>>> # Create a Series
>>> s = pd.Series([10,20,30,40,50], index = ['a','b','c','d','e'])
>>> # Create a DataFrame from the Series
>>> df = pd.DataFrame(s)
>>> print('DataFrame created from Series :\n',df)
DataFrame created from Series :
    0
a  10
b  20
c  30
d  40
e  50
>>>
Ln: 112 Col: 10
```

**Naming the DataFrame Column**

Note that the column name assigned to 0 by default. You can also specify a custom column name to the Series. Also, if the index is not specified it starts with 0. The above code can be modified as below to illustrate this.

```
File Edit Shell Debug Options Window Help
>>> # Creation of DataFrame from a single series
>>> import pandas as pd
>>> # Create a Series
>>> s = pd.Series([10,20,30,40,50], name="Numbers")
>>> df = pd.DataFrame(s)
>>> print('DataFrame created from Series with column name :\n',df)
DataFrame created from Series with column name :
   Numbers
0       10
1       20
2       30
3       40
4       50
>>>
Ln: 177 Col: 0
```

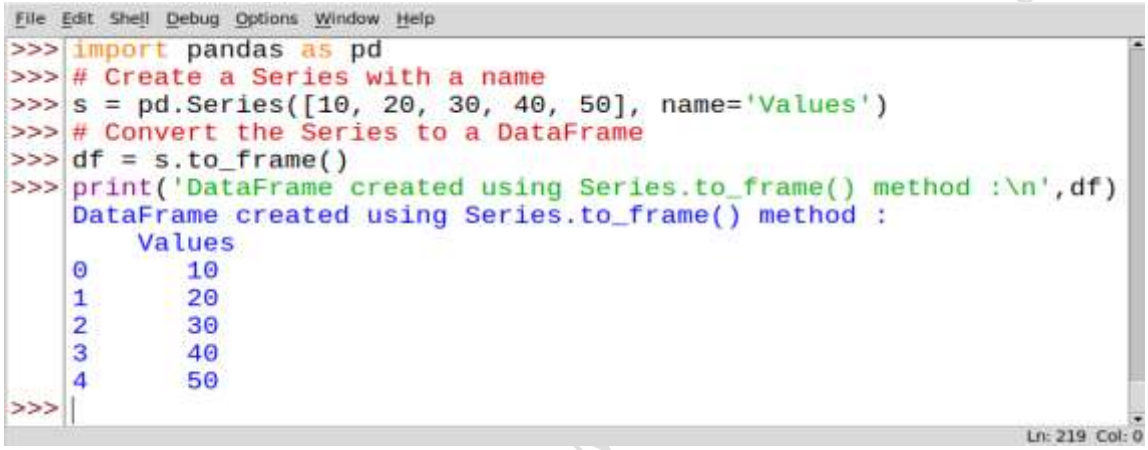
In the above example, the DataFrame dFrame has as many numbers of rows as the numbers of elements in the series, but has only one column. To create a DataFrame using more than one series, we need to pass multiple series in the list as shown below:

**2. Using Series.to_frame() method**

This method provides a convenient way to convert a Series into a DataFrame with the Series as a column. The name of the Series (if set) becomes the column name in the DataFrame.

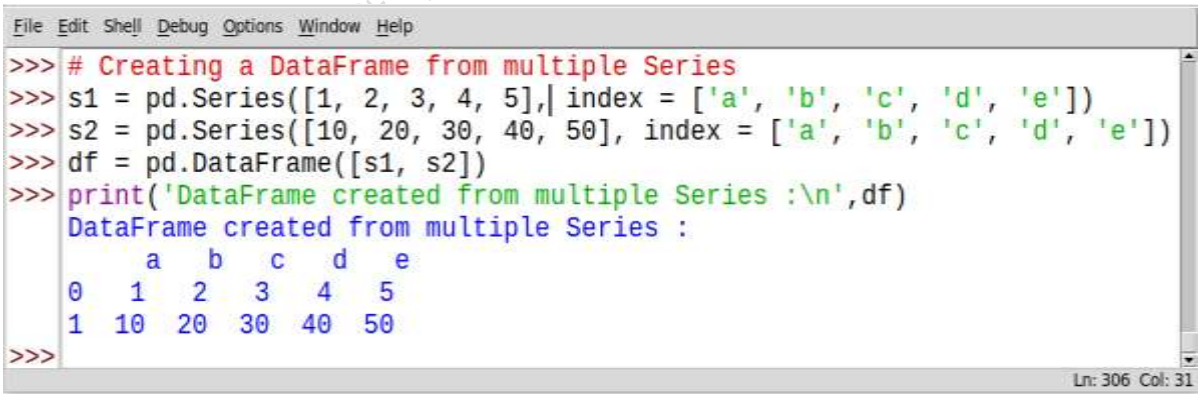
```
# Creation of Series using Series.to_frame() method
import pandas as pd
```

```
# Create a Series with a name
s = pd.Series([10, 20, 30, 40, 50], name='Values')
# Convert the Series to a DataFrame
df = s.to_frame()
print('DataFrame created using Series.to_frame() method :\n',df)
DataFrame created using Series.to_frame() method :
    Values
0       10
1       20
2       30
3       40
4       50
```




**b. From multiple Series (as columns):**

You can combine multiple Series into a single DataFrame, treating each Series as a column. To create a DataFrame using more than one series, we need to pass multiple series in the list as shown below.



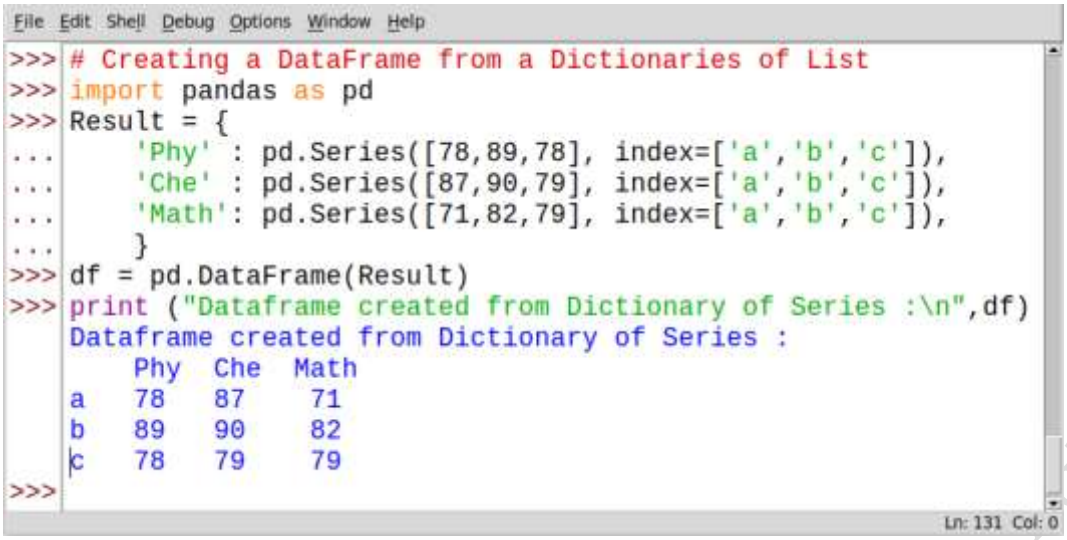
```
File  Edit  Shell  Debug  Options  Window  Help
>>> # Creating a DataFrame from multiple Series using concat function
>>> import pandas as pd
>>> s1 = pd.Series(["apple","orange","grape","bannana"],name="Fruits")
>>> s2 = pd.Series([200, 120, 300, 40], name="Rates")
>>> print('DataFrame created using concat function :\n',df)
DataFrame created using concat function :
         Fruits  Rates
0       apple    200
1      orange    120
2       grape    300
3  watermelon     40
4         NaN     50
>>>
                                                        Ln: 391  Col: 0
```

To combine multiple Series into a DataFrame where each Series represents a column, use pd.concat(). Pass a list of Series to pd.concat() and set axis=1 to concatenate them as columns.

**5. Creation of DataFrame from Dictionary of Series**

Creating a Pandas DataFrame from a dictionary of Series is a highly effective way to construct a tabular data structure where each Series represents a column. Following example illustrate to create the DataFrame of 3 students in the marks obtained in three subjects. The names of the students are the keys to the dictionary, and the index values of the series are the subject names as shown below.

```
# Creating a DataFrame from a Dictionaries of List
import pandas as pd
Result = {
    'Phy' : pd.Series([78,89,78], index=['a','b','c']),
    'Che' : pd.Series([87,90,79], index=['a','b','c']),
    'Math': pd.Series([71,82,79], index=['a','b','c']),
    }
df = pd.DataFrame(Result)
print ("Dataframe created from Dictionary of Series :\n",df)
Dataframe created from Dictionary of Series :
    Phy  Che  Math
a   78   87    71
b   89   90    82
c   78   79    79
```




### 3.3 OPERATIONS ON ROWS AND COLUMNS IN DATAFRAMES

Pandas DataFrames are 2-dimensional, labeled data structures resembling tables with rows and columns, like a spreadsheet or SQL table. Pandas DataFrames, are widely used for data manipulation and analysis. They offer powerful functionalities for interacting with data arranged in rows and columns, similar to a spreadsheet or database table.

You can perform various operations on rows and columns for data manipulation and analysis.

**(A) Adding a New Column to a DataFrame**

It is possible to add a new column to a DataFrame. Let us consider the DataFrame Result defined above. In order to add a new column for another subject "AI"', we can write the following statement:

```
# Adding a New Column to a DataFrame
df['AI']=[89,78,76]
print ("Dataframe after adding a new column :\n",df)

Dataframe after adding a new column :
    Phy  Che  Math  AI
a   78   87    71  89
b   89   90    82  78
c   78   79    79  76
```

Assigning values to a new column label that does not exist will create a new column at the end. If the column already exists in the DataFrame then the assignment statement will update the values of the already existing column, for example:

```
# Updating the values of Column in DataFrame
df['AI']=[60,70,80]
print ('DataFrame after changing value of column :\n',df)
```

```
DataFrame after changing value of column :
    Phy  Che  Math  AI
a   89   92    89  60
b   74   87    82  70
c   58   85    81  80
```

It is also possible to change data of an entire column to a particular value in a DataFrame. For example, the statement sets marks=80 to AI for all students a, b, c.

```
# Changing data of an entire column to a particular value
df['AI']=80
print ('DataFrame after changing value of AI to 80 for all :\n',df)

DataFrame after changing value of AI to 80 for all :
    Phy  Che  Math  AI
a   89   92    89  80
b   74   87    82  80
c   58   85    81  80
```

**(B) Adding a New Row to a DataFrame**

It is possible to add a new row to a DataFrame using the DataFrame.loc[ ] method. Following code illustate to add the marks for d.

```
# Adding a New Row to a DataFrame
df.loc['d']=[60,70,80,90]
print ('DataFrame after adding a new row :\n',df)

DataFrame after adding a new row :
    Phy  Che  Math  AI
a   78   87    71  80
b   89   90    82  80
c   78   79    79  80
d   60   70    80  90
```

**(C) Deleting Rows or Columns from a DataFrame**

The DataFrame.drop() method is used to delete rows and columns from a DataFrame. For that it is required to specify the names of the labels to be dropped and the axis from which they need to be dropped. To delete a row, the parameter axis is assigned the value 0 and for deleting a column,the parameter axis is assigned the value 1. Consider the following DataFrame:

The following example shows to delete the row with label 'd':

```
# Deleting a Row from a DataFrame
df = df.drop('d',axis=0)
print ('DataFrame after deleting a row :\n',df)
DataFrame after deleting a row :
    Phy  Che  Math  AI
a   78   87    71  80
b   89   90    82  80
c   78   79    79  80
```

**(D) Renaming Row and Column Labels of a DataFrame**

To change the column names, we can use the rename() method, as shown below. The parameter axis='columns' change the specifed column labels.

The following example shows that row lables 'a', 'b', 'c', are changed to 1,2,3 and column lable "AI' changed to 'Voc'.

```
# Renaming Row Labels of a DataFrame
df = df.rename({'AI':'Voc'}, axis='columns')
print ('DataFrame after renaming Column lables :\n',df)

DataFrame after renaming a Column lables :
    Phy  Che  Math  Voc
1   78   87    71   80
2   89   90    82   80
3   78   79    79   80
```

### 3.4 ACCESSING DATAFRAMES ELEMENT THROUGH INDEXING

Data elements in a DataFrame can be accessed using indexing. Pandas DataFrames provide various methods for selecting and accessing data. Some of them are discussed below.

**1. Primary indexing methods**

These are the most commonly used methods for selecting data subsets. Consider the example below to illustrate the indexing.

```
# DataFrame indexing methods
import pandas as pd

data = {'Name': ['Asha', 'Boby', 'Diya', 'Deepak'],
        'Age': [20, 22, 23, 40],
        'City': ['Nagpur', 'Bhopal', 'Pune', 'Mumbai']}
df = pd.DataFrame(data, index=['A', 'B', 'C', 'D'])
```

```
# Original DataFrame
print ("Original DataFrame\n",df)

Original DataFrame
      Name  Age    City
A    Asha   20  Nagpur
B    Boby   22  Bhopal
C    Diya   23    Pune
D  Deepak   40  Mumbai
```

**.loc[] (label-based indexing):** It allows to access rows and columns based on the labels (names). It is used to when you know the exact names of the rows and columns to select. The .loc[] accessor is used for label-based indexing.

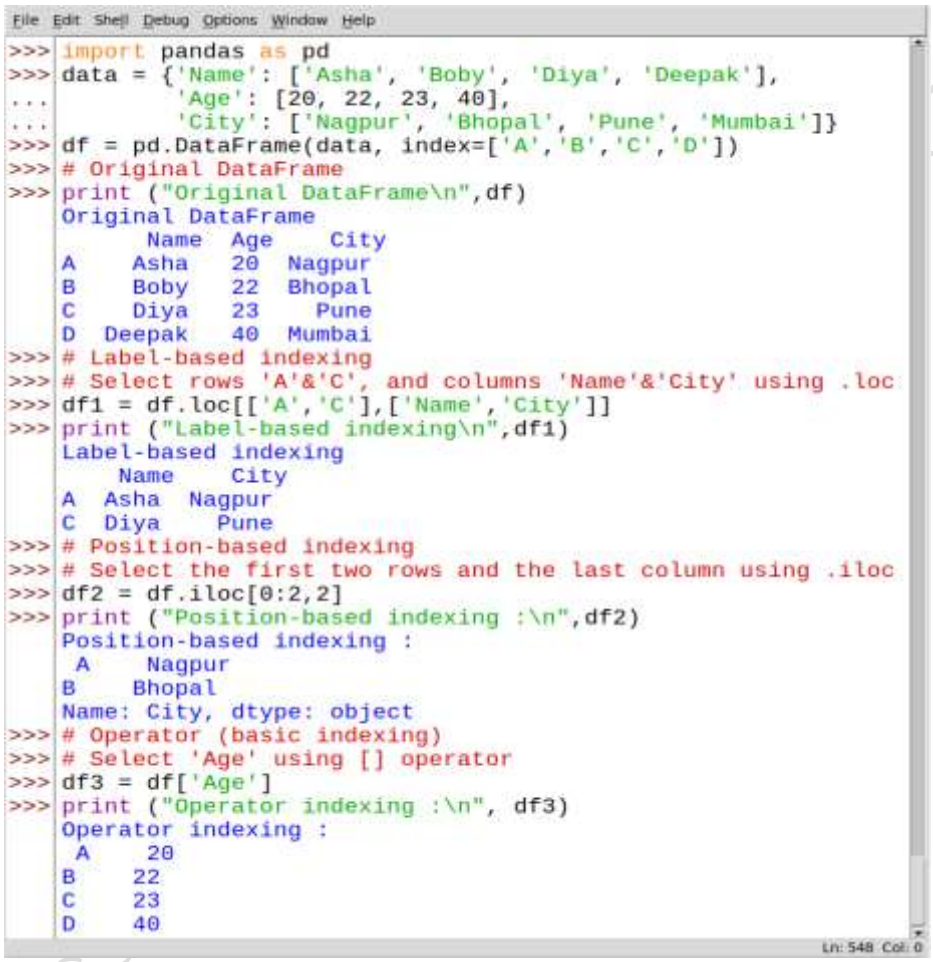
```
# Label-based indexing
# Select rows 'A'&'C', and columns 'Name'&'City' using .loc
df1 = df.loc[['A','C'],['Name','City']]
print ("Label-based indexing\n",df1)

Label-based indexing
    Name    City
A  Asha  Nagpur
C  Diya    Pune
```

**.iloc[] (position-based indexing):** Selects data based on the integer position (starting from 0) of rows and columns. This is useful when you need to select data based on its numerical order or when the labels are not meaningful. The .iloc[] accessor is used for position-based indexing.

```
# Position-based indexing
# Select the first two rows and the last column using .iloc
df2 = df.iloc[0:2,2]
print ("Position-based indexing :\n",df2)

Position-based indexing
A    Nagpur
B    Bhopal
Name: City, dtype: object
```

**[] operator (basic indexing):** Allows for selecting single or multiple columns by name or by filtering rows using a boolean condition. It is a basic method with limitations for more complex selections.

```
# Operator (basic indexing)
# Select 'Age' using [] operator
df3 = df['Age']
print (“Operator indexing :\n”, df3)

Operator indexing :
A    20
B    22
C    23
D    40
```




**2. Boolean indexing**

This technique uses boolean arrays (True/False values) to filter data based on specified conditions. It is supported by both .loc[] and .iloc[], as well as the basic [] operator.

Boolean indexing allows to select rows from a DataFrame based on whether values in a specific column meet a certain condition. In Boolean indexing, we can select the subsets of data based on the actual values in the DataFrame rather than their row/column labels. Thus, we can use conditions on column names to filter data values. It is illustrated below.

```
# Boolean indexing
# Filter rows where 'Age' is greater than 30
df4 = df['Age'] > 30
print ("Boolean indexing :\n", df4)

Boolean indexing :
A    False
B    False
C    False
D    True
Name: Age, dtype: bool
```

### 3.5 ATTRIBUTES OF PANDAS DATAFRAMES

Pandas DataFrames, is a two-dimensional, labeled data structures, come with various attributes that provide important information about the data they hold. These attributes helps to understand the structure, dimensions, and contents of DataFrame.

Some of the commonly used attributes and their functions are illustrated by taking the sample code as below:

```
>> import pandas as pd
>> data = {'Name': ['Diya', 'Deepak'], 'Age': [25, 30]}
>> df = pd.DataFrame(data)
```

| Attribute Name | Purpose | Example |
|---|---|---|
| DataFrame.shape | Returns a tuple representing the number of rows and number of columns in the dataframe. It is useful for understanding the size of the dataset. | print (df.shape)<br># Output: (2, 2) |
| DataFrame.size | Returns the total number of elements in the DataFrame, which is the product of the number of rows and columns | print(df.size)<br># Output: 4 |
| DataFrame.ndim | Returns the number of dimensions of the DataFrame, which is always 2. | print(df.ndim)<br># Output: 2 |
| DataFrame.dtypes | Returns a Series indicating the data type (dtype) of each column. It is useful for identifying the type of data stored in each column | print(df.dtypes)<br># Output:<br># Name object<br># Age int64<br># dtype: object |

| | | |
|---|---|---|
| DataFrame.columns | Returns the column labels as an Index object. | print(df.columns)<br># Output: Index(['Name','Age'], dtype='object') |
| DataFrame.index | Provides the row labels (index) of the DataFrame as an Index object. | print(df.index)<br># Output: Index(['Row1', 'Row2'], dtype='object') |
| DataFrame.values | Returns the underlying data as a 2D NumPy array. It is useful for converting a DataFrame to a NumPy array | print(df.values)<br># Output:<br># [['Diya' 20]<br># ['Deepak' 50]] |
| DataFrame.T (Transpose) | Returns the transpose of the DataFrame, swapping rows and columns. | print(df.T)<br># Output:<br># 0 1<br># Name Alice Bob<br># Age 25 30 |

```
File Edit Shell Debug Options Window Help
>>> # Attributes of DataFrames
>>> import pandas as pd
>>> data = {'Name': ['Diya', 'Deepak'], 'Age': [20, 50]}
>>> df = pd.DataFrame(data)
>>> print("Dataframe Shape :", df.shape)
Dataframe Shape : (2, 2)
>>> print("Dataframe Size :", df.size)
Dataframe Size : 4
>>> print("Dataframe Dimension :", df.ndim)
Dataframe Dimension : 2
>>> print("Dataframe Data type :\n", df.dtypes)
Dataframe Data type :
 Name    object
Age      int64
dtype: object
>>> print("Dataframe column labels :\n", df.columns)
Dataframe column labels :
 Index(['Name', 'Age'], dtype='object')
>>> print("Dataframe Row labels :\n", df.index)
Dataframe Row labels :
 RangeIndex(start=0, stop=2, step=1)
>>> print("Dataframe Values :\n", df.values)
Dataframe Values :
 [['Diya' 20]
 ['Deepak' 50]]
>>> print("Transpose of Dataframe :\n", df.T)
Transpose of Dataframe :
            0       1
Name     Diya  Deepak
Age        20      50
>>>
                                                    Ln: 643 Col: 21
```



## Summary

- Columns can hold heterogeneous data (numeric, string, boolean, etc.).
- Features: labeled axes, flexible size, supports complex operations (filtering, grouping, merging, reshaping).
- Operations: add/delete rows & columns, rename labels, handle missing values.

- DataFrames are powerful for organizing, analyzing, and visualizing structured datasets.

## Check Your Progress

**A. Multiple choice questions**

1. Which of the following is NOT a valid way to create a Pandas DataFrame? (a) From a dictionary of Series. (b) From a NumPy ndarray (c) From a scalar value without an index and columns specified (d) From a list of dictionaries.
2. To select a single column named 'Age' from a DataFrame df, which of the following is the correct syntax? (a) df.Age (b) df['Age'] (c) df.loc[:, 'Age'] (d) All of the above
3. How do you access a group of rows or columns by labels in a DataFrame? (a) iloc (b) loc (c) at (d) iat
4. Which attribute returns a tuple representing the dimensionality (rows, columns) of a DataFrame? (a) .size (b) .ndim (c) .shape (d) .axes
5. To return the first n rows of a DataFrame df, which method would you use? (a) df.first(n) (b) df.top(n) (c) df.head(n) (d) df.start(n)
6. What does the .columns attribute of a DataFrame return? (a) A list of the row index labels (b) A list of the column labels (column names) (c) The data types of the columns (d) The number of columns.
7. If you create a DataFrame from a dictionary where the keys are column names and the values are lists of unequal length, what will happen? (a) An error will be raised (b) The shorter lists will be padded with 0 (c) The shorter lists will be padded with NaN (d) The DataFrame will be created only with the length of the shortest list
8. Which method is used to drop rows with missing values (NaN) from a DataFrame? (a) df.drop_missing() (b) df.remove_nan() (c) df.dropna() (d) df.fill_na()
9. To get the transpose of a DataFrame df, you can write: (a) df.Transpose() (b) df.T (c) df.swap_axes() (d) df.transpose_dataframe()
10. Which attribute tells you the number of rows and columns in a DataFrame? (a) .size (b) .count (c) .shape (d) .ndim

**B. Fill in the blanks**

1. DataFrames consist of rows and _____.
2. The common alias used for the Pandas library when importing is pd, as in import pandas as _____.
3. To create a DataFrame from a dictionary of lists, the dictionary keys usually become the DataFrame's _____ names.
4. The ._____ attribute returns a tuple indicating the number of rows and columns in a DataFrame.
5. To display the first 5 rows of a DataFrame df, you would use the method df._____.

6. To select a column named 'Price' from a DataFrame my_df, you can use my_df[_____].
7. The ._____ attribute provides a list of the column labels (names) of a DataFrame.
8. When you perform mathematical operations between two DataFrames, Pandas aligns the data based on both the row _____ and the column_____
9. The ._____ attribute provides a list of the row labels (index) of a DataFrame
10. The ._____ attribute returns the data type of each column in a DataFrame, usually as a Series.

**C. State whether True or False**

1. All columns in a Pandas DataFrame must have the same data type.
2. You can create a DataFrame from a dictionary of Series.
3. The .index attribute of a DataFrame returns the column labels.
4. The df.head() method displays the first 5 rows of the DataFrame by default.
5. The iloc accessor is used to select data based on integer-location based indexing.
6. The df.shape attribute returns the total number of elements in the DataFrame.
7. Missing values in a DataFrame are typically represented by the string "NA".
8. You can add a new column to an existing DataFrame.
9. The df.T attribute returns the transpose of the DataFrame (swaps rows and columns).
10. When creating a DataFrame from a dictionary where values are lists of unequal length, an error will always be raised.

**D. Answer the following questions in short**

1. What is the fundamental difference between a Pandas Series and a Pandas DataFrame?
2. Name two common ways to create a Pandas DataFrame.
3. Explain the purpose of the .shape attribute of a DataFrame.
4. What is the primary difference between the loc and iloc accessors when selecting data from a DataFrame?
5. When is the head() method useful, and what does it display by default?
6. What does the df.columns attribute provide?
7. How are missing values represented in a Pandas DataFrame, and what method can be used to remove rows or columns containing them?
8. What does the df.describe() method output, and for which data types is it most relevant?
9. Briefly describe the functionality of the fillna() method.
10. How does Pandas handle arithmetic operations (e.g., addition) between two DataFrames with misaligned indexes and/or columns?

## Session 4. Data Visualisation using Matplotlib

Imagine reading a cricket scorecard full of numbers—it is difficult to instantly know who played better or how the game progressed. But when the same data is shown as a line graph of runs over overs or a bar chart comparing players' scores, the story becomes clear at a glance. This is the power of data visualization—it turns complex numerical results into easy-to-understand visuals. In this session, you will learn how to use the Matplotlib library in Python to create different types of plots such as line charts, bar graphs, scatter plots, histograms, pie charts, and box plots, and also customize them with titles, labels, legends, and grids for better clarity.

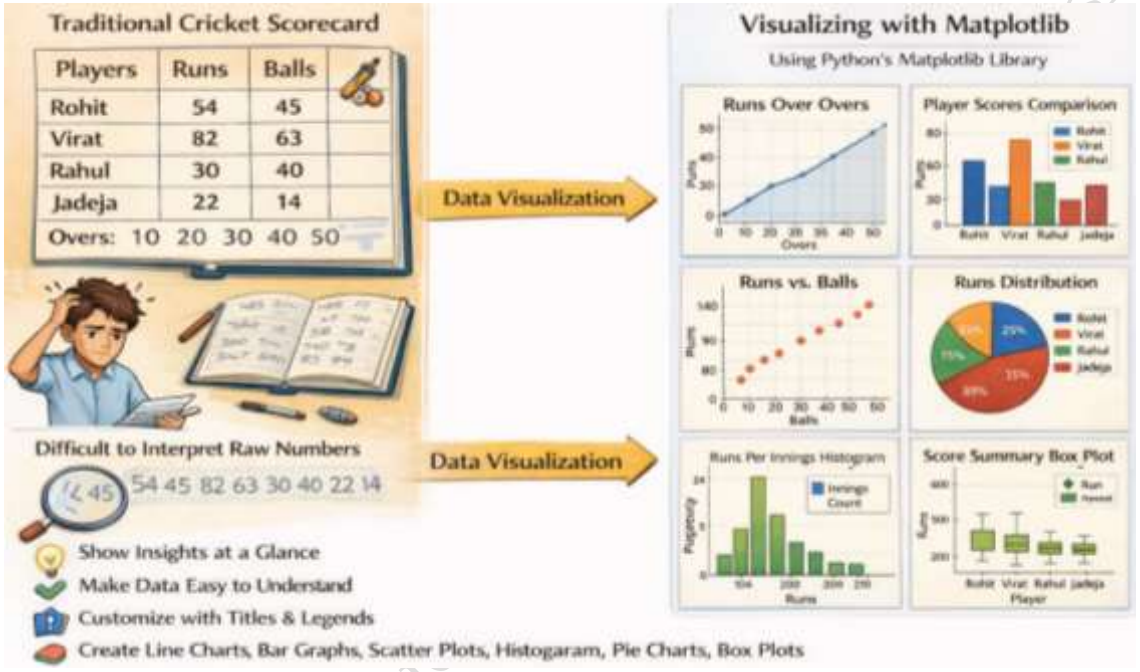



Fig. 4.1: Visualizing Cricket scores with Matplotlib

In this session, you will understand the importance of data visualization, use the Matplotlib library to create different types of plots such as line, bar, scatter, histogram, pie, and box plots, and customize them with titles, labels, legends, and grids to make data easier to understand and interpret.

### 4.1 INTRODUCTION

We have learned how to organise and analyse data and perform various statistical operations on Pandas DataFrames. Also, we have learned how to analyse numerical data using NumPy. The results obtained after analysis is used to make inferences or draw conclusions about data as well as to make decisions. Sometimes, it is not easy to infer by merely looking at the results. In such cases, visualisation helps in better understanding of results of the analysis.

Data visualization is the graphical representation of data using graph, chart, etc. The purpose of plotting data is to visualise variation or show relationships between variables.

Data visualization in the form of charts, graphs, animation, and maps are very easy and simple to understand the trends, outliers, and patterns in data. Data visualization techniques for such big data are very important for the purpose of analysis of data.

Visualisation of data is effectively used in fields like health, finance, science, mathematics, engineering, etc.

In this session, we will learn how to visualise data using Matplotlib library of Python by plotting charts such as line, bar, scatter with respect to the various types of data.

**Data Visualization Tools**

Some popular data visualization tools are as given below:

1. Matplotlib (Python) – For static plots.
2. Tableau & Power BI – Interactive dashboards.
3. Google Data Studio – Web-based visualization.
4. D3.js – JavaScript library for dynamic visualizations.

## 4.2 MATPLOTLIB

The Matplotlib is a python library that provides many interfaces and functionality for 2D-graphics similar to MATLAB. Python scripts can be used to create 2D graphs and plots using the Matplotlib module. With features to control line styles, font attributes, formatting axes, and other features, it offers a module named pyplot that makes things simple for plotting. It offers a huge range of graphs and plots, including error charts, bar charts, power spectra, and histograms. It is combined with NumPy to provide a powerful open source MatLab substitute environment.

### 4.2.1 Installing Matplotlib

To install Matplotlib, issue the following command on the command prompt.

```
pip install matplotlib
```

It will give the message for successful installation of Matplotlib library.

### 4.2.2 Importing Pyplot

It is common practice to import the pyplot module using the alias plt, which simplifies code and follows standard conventions.

```
import matplotlib.pyplot as plt
```

Here, plt is an alias or an alternative name for matplotlib.pyplot. We can use any other alias also. You can use all the functions in the pyplot library using this object as per your need.

### 4.2.3 The pyplot module of matplotlib

The pyplot module of matplotlib contains a collection of functions that can be used to work on a plot. The plot() function of the pyplot module is used to create a figure. A figure is the overall window where the outputs of pyplot functions are plotted. A figure contains a plotting area, legend, axis labels, ticks, title, etc. Each function makes some change to a figure: example, creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.

It is always expected that the data presented through charts easily understood. Hence, while presenting data we should always give a chart title, label the axis of the chart and provide legend in case we have more than one plotted data.

**Example 4.1:** Let us plot a simple chart of daily temperature fluctuations over a week. The sample code to plot a line chart with its output is given below.

```
# Creating a line chart for temperature over a week
import matplotlib.pyplot as plt
days = ['Mon','Tue','Wed','Thu','Fri', 'Sat','Sun']
temp = [35,37,38,36,39,37,38] # temperatures in Celsius
plt.plot(days, temp) # Plotting the line chart
plt.show()
```

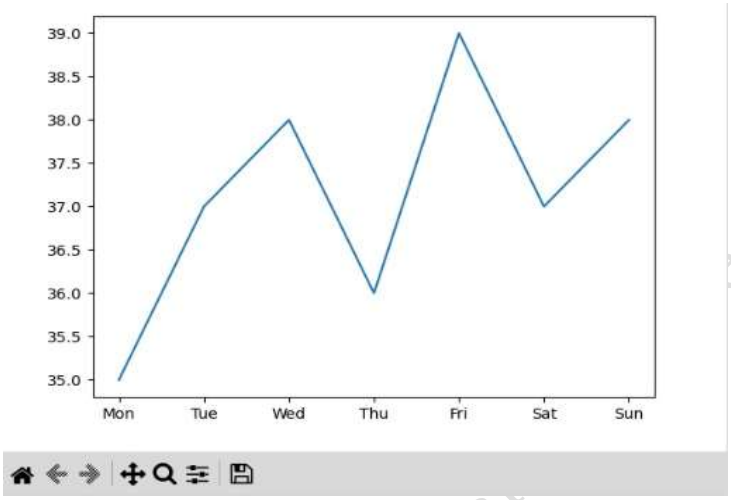



*Fig. 4.2: Line chart as output of Example 4.1*

```
File  Edit  Shell  Debug  Options  Window  Help
>>> # Creating a line chart for temperature over a week
>>> import matplotlib.pyplot as plt
>>> days = ['Mon','Tue','Wed','Thu','Fri', 'Sat','Sun']
>>> temp = [35,37,38,36,39,37,38] # temperatures in Celsius
>>> plt.plot(days, temp) # Plotting the line chart
    [<matplotlib.lines.Line2D object at 0x7dde311a1130>]
>>> plt.show()
                                                    Ln: 191  Col: 0
```

In Example 4.1, plot() is provided with two parameters, which indicates values for x-axis and y-axis, respectively. The x and y ticks are displayed accordingly. As shown in Figure 4.1, the plot() function by default plots a line chart. Click on the save button on the output window and save the plot as an image. A figure can also be saved by using *savefig()* function. The name of the figure is passed to the function as parameter.

For example: plt.savefig('x.png').

In this example, plot() function is used to plot a line graph. There are different types of data available for analysis. The type of chart to be plotted is determined by the type of data.

### 4.2.4 Basic plotting functions

The common pyplot functions used for plotting different charts are:

**`plt.plot(x, y):`** Creates a line plot, showing trends and relationships between two variables.

**`plt.scatter(x, y):`** Generates a scatter plot, displaying individual data points to identify correlations or clusters.

**`plt.bar(x, height):`** Creates a bar chart for comparing quantities across different categories.

**`plt.hist(data, bins):`** Plots a histogram to visualize the distribution of numerical data.

**`plt.pie(values, labels):`** Generates a pie chart to illustrate proportions and parts of a whole.

**`plt.boxplot(data):`** Creates a box plot (or whisker plot) to display the distribution of data, including the median, quartiles, and outliers.

### 4.2.4 Customisation of Plots

The common plot customizations include adjusting colors, line styles, marker types, titles, axis labels, and legends. Pyplot library gives various functions used to customise charts such as adding titles or legends. The common customisaton functions for Matplotlib are given below.

### 4.2.5 Functions for customizing plots

Matplotlib offers a variety of functions for customizing plots to enhance their clarity, aesthetics, and overall effectiveness. Some of the most commonly used functions are:

**`plt.plot():`** Plots y versus x as lines and/or markers.

**`plt.show():`** Displays the figure.

**`plt.title():`** Sets the title of the current plot.

**`plt.xlabel():`** Sets the label for the x-axis.

**`plt.ylabel():`** Sets the label for the y-axis.

**`plt.title():`** Sets the title of the current plot.

**`plt.legend():`** Displays a legend to identify different data series in the plot.

**`plt.figure():`** Creates a new figure or activates an existing one.

**`plt.savefig():`** Saves the figure to a file (e.g., PNG, PDF).

**`plt.grid():`** Adds a grid to the plot. You can control its color, line style, and opacity.

**`plt.xticks() and plt.yticks():`** Customize the location, format, and rotation of tick labels on the x and y axes.

**Example 4.2.** Let us customise the line chart plotted in Example 3.1 by adding a Title and Grids to the chart.

```
# Creating a line chart with customisation
import matplotlib.pyplot as plt
days = ['Mon','Tue','Wed','Thu','Fri', 'Sat','Sun']
temp = [35,37,38,36,39,37,38] # temperatures in Celsius
plt.plot(days, temp)          # Plotting the line chart
plt.xlabel("Day")             #add the Label on x-axis
Text(0.5, 0, 'Day')
```

```
plt.ylabel("Temperature")     #add the Label on y-axis
Text(0, 0.5, 'Temperature')
plt.title("Weekly Temperature")
Text(0.5, 1.0, 'Weekly Temperature')  #add the title to the chart
plt.grid(True)                # add gridlines
plt.yticks(temp)
plt.show()  # Display the plot
```

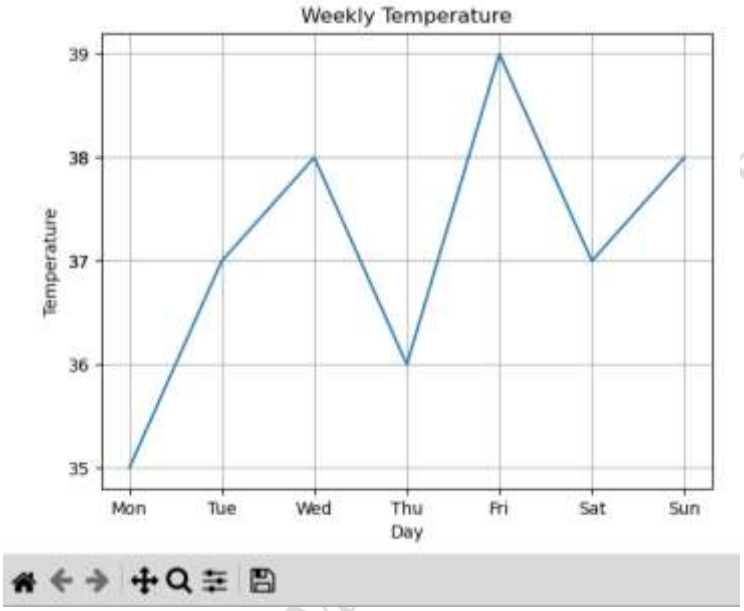



Fig. 4.3: Line chart with customization example 4.2

### 4.2.6 Customization Functions for various plots

**Line plot**

- **plt.plot():** This fundamental function used for line plots also accepts arguments to customize line styles, colors, and markers.
- **color:** Sets the color of the line.
- **linestyle:** Specifies the line style, such as solid ('-'), dashed ('--'), dotted (':'), or dash-dot ('-.').
- **linewidth:** Adjusts the thickness of the line.
- **marker:** Sets the marker style for points on the line, such as a circle ('o') or a star ('*'). There are various codes for marker parameters.
- **markersize:** Adjusts the size of the markers.

**Bar Chart**

- **plt.bar():** Used to create bar charts, offers customization of bar colors, width, and edge properties.
- **color:** Sets the fill color of the bars.
- **edgecolor:** Sets the color of the bar edges.
- **width:** Controls the width of each bar.

**Scatter Plot**

- **plt.scatter():** Used for scatter plots, it offers similar customization options as plt.plot() for markers and colors, along with additional arguments.
- **s:** Sets the size of the markers, either a single value or an array.
- **c:** Sets the color of the markers, which can be a single color or an array of colors (useful for color-coding points based on data).
- **edgecolors:** Sets the color of the marker borders.
- **cmap:** Allows you to apply a colormap to the markers, especially helpful for visualizing a third variable through color.

**Histogram**

- **plt.hist():** Used to create histograms, with customization options for bins, colors, and edge properties.
- **bins:** Defines the number or width of bins.

### 4.3 TYPES OF CHARTS FOR DATA VISUALIZATIONS

Matplotlib is a powerful Python library for creating a wide variety of static, interactive, and animated plots and visualizations. Some of the most common types of charts you can create using Matplotlib:

#### 4.3.1. Line charts

Line charts are used to visualize the relationship between two variables. It shows how a variable change across continuous data points. Line charts are mainly used for tracking trends, time series data, and showing changes in a variable over time.

**Function:** plt.plot(x, y) funtion is used to plot the line chart.

**Example 4.3:** Create a line plot with customization

```
# Example of line plot with customisation
import matplotlib.pyplot as plt
import numpy as np
# Sample data
x_values = np.array([1, 2, 3, 4, 5])
y_values = np.array([2, 5, 3, 7, 4])

# Create the plot with customization
plt.plot(x_values, y_values,
         color='green',          # Set line color to green
         linestyle='--',         # Use a dashed line style
         linewidth=2,            # Set line thickness to 2
         marker='o',             # Add markers at data points
         markersize=8,           # Set marker size to 8
         label='Data Trend')     # Label the line for the legend
```

```
# Add title and axis labels
plt.title('Simple Line Chart with Customization')
plt.xlabel('X-axis Values')
plt.ylabel('Y-axis Values')

# Add a legend
plt.legend()

# Add a grid
plt.grid(True)

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

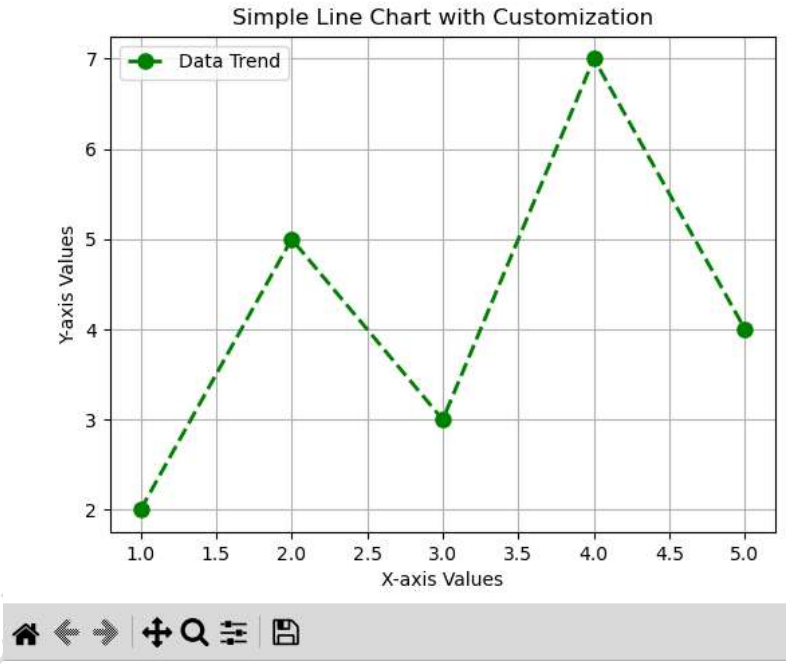



Fig. 4.4: Simple Line chart with customization example 4.3

**Multiline Chart**

**Example 4.4:** Create aMultiline Chart for Comparison of Scores ofIndia vs England

```
# Multiline Chart for Comparison of Scores ofIndia vs England
import matplotlib.pyplot as plt
import numpy as np

# Runs scored by each team at different points in the comparison
overs = np.array([10, 20, 30, 40, 50])
```

```
india_scores = np.array([74,105,155,220,280])
england_scores = np.array([54,115,158,201,249])

# Create the line chart
plt.figure(figsize=(10, 6)) # figure size for better readability

# Plot India's scores
plt.plot(overs, india_scores, marker='o', linestyle='-', color='blue',
label='India')

# Plot England's scores
plt.plot(overs, england_scores, marker='x', linestyle='--', color='red',
label='England')

# Add labels, title, and legend
plt.xlabel('Over Interval')
plt.ylabel('Score')
plt.title('Multiline Chart for Comparison of Scores: India vs England')
plt.legend() # Displays the labels for each line (India, England)
plt.grid(True) # Adds a grid for easier reading of values

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

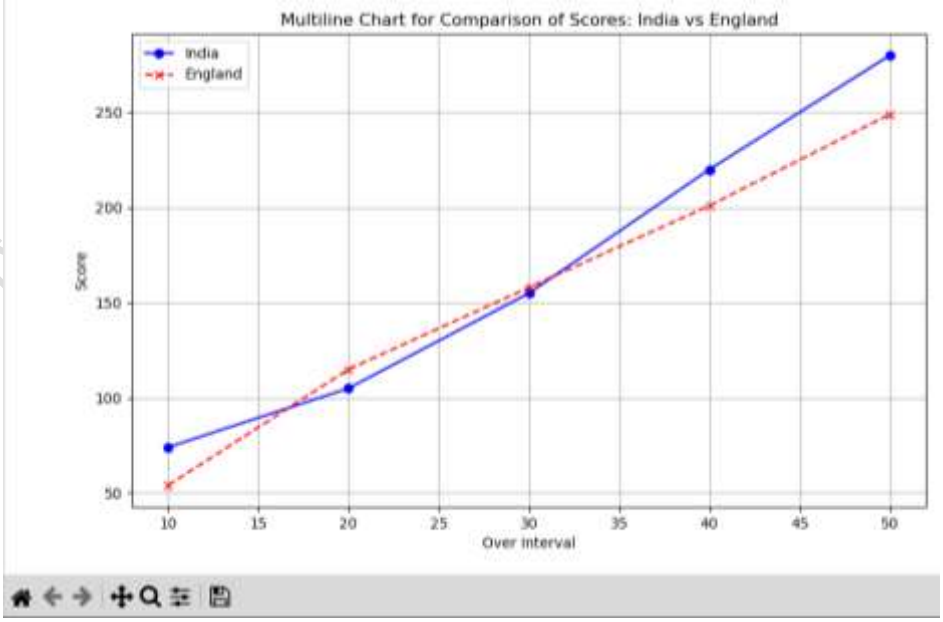



Fig. 4.5: Multiline chart for comparison of score example 4.4

### 4.3.2. Bar charts

Bar charts display categorical data with rectangular bars, where the length or height of the bar is proportional to the value it represents. Bar charts are normally used for comparing quantities across different categories or groups, and comparing sales figures by product category. There are two types of bar charts – vertical and horizontal bar charts.

**Function:** The function, plt.bar(categories, values) is used for vertical bars and plt.barh(categories, values) is used for horizontal bars.

**Example 4.5:** Create a bar chart showing sales of various fruits.

```
# Bar Chart showing the Sales of Various Fruits
import matplotlib.pyplot as plt
# Data
categories = ['Apples', 'Bananas', 'Oranges', 'Grapes']
sales = [30, 45, 20, 55]
# Create the bar chart
plt.bar(categories, sales) # Plot a bar for each category
# Add labels and a title for clarity
plt.xlabel('Fruit Type')
plt.ylabel('Modules Sold')
plt.title('Sales of Various Fruits')
# Display the chart
plt.show()
```

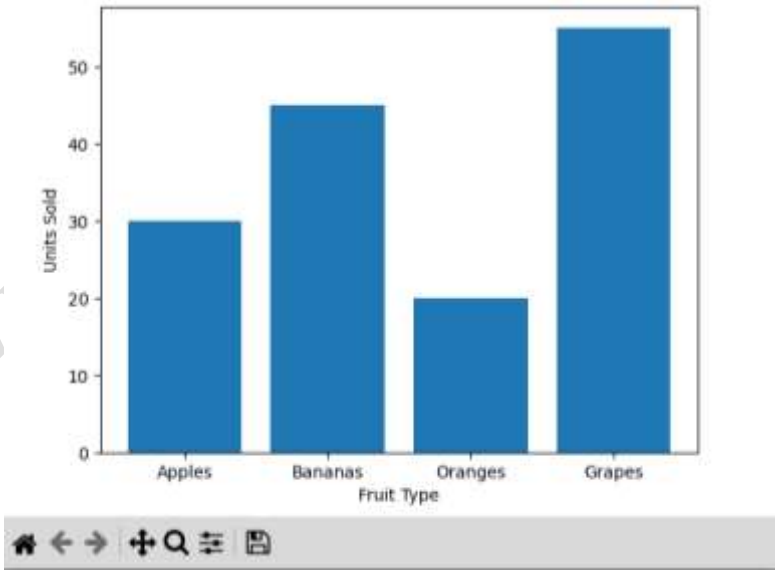



Fig. 4.6: Sales of various fruits example 4.5

### 4.3.3. Scatter plots

Scatter plots visualize the relationship between two numerical variables using individual data points, helping to identify correlations or clusters. Scatter plots are used for analyzing

the relationship between two continuous variables, identifying patterns, correlations, and potential outliers.

**Function:** The function, plt.scatter(x, y) is used to plot scatter plots.

**Example 4.6:** Create a Scatter Plot Showing the Fall of Wickets.

```
# Scatter Plot Showing the Fall of Wickets
import matplotlib.pyplot as plt
import numpy as np

# Sample data
wickets = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
runs = np.array([25, 30, 45, 48, 60, 65, 70, 80, 85, 92])

# Create the scatter plot
plt.scatter(wickets, runs) # Plots points at the intersection of x and y values

# Add labels and a title
plt.xlabel('Fall of Wickets')
plt.ylabel('Runs Scored')
plt.title('Scatter Plot Showing the Fall of Wickets')

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

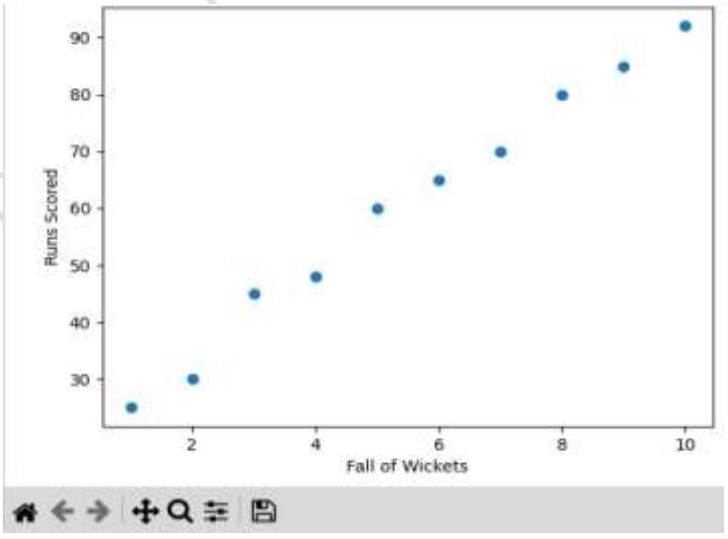



Fig. 4.7: Scatter plot showing the fall of wickets example 4.6

### 4.3.4. Histograms

Histograms represent the distribution of numerical data by showing the frequency or count of data points within specified ranges or bins. It is usef for analyzing the distribution of a

single numerical variable, identifying the frequency of data points within certain ranges, and inspecting the skewness of data.

**Function:** The function, plt.hist(data, bins) is used to plot the histogram.

**Example 4.7:** Histogram for distribution of Marks for 20 Students

```
# Histogram for distribution of Marks for 20 Students'
import matplotlib.pyplot as plt
import numpy as np

# Marks of 20 students
marks = np.array([35,28,55,62,78,41,15,85,92,58,
                  65,71,30,48,70,25,95,50,60,75])

# Create the histogram plot with 5 bins
plt.hist(marks, bins=5, color= '#FF9999', edgecolor='black')

# Add labels and a title for clarity
plt.xlabel('Test Score Ranges')     # X-axis title
plt.ylabel('Number of Students')    # Y-axis title

# Histogram title
plt.title('Histogram for distribution of Marks for 20 Students')

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

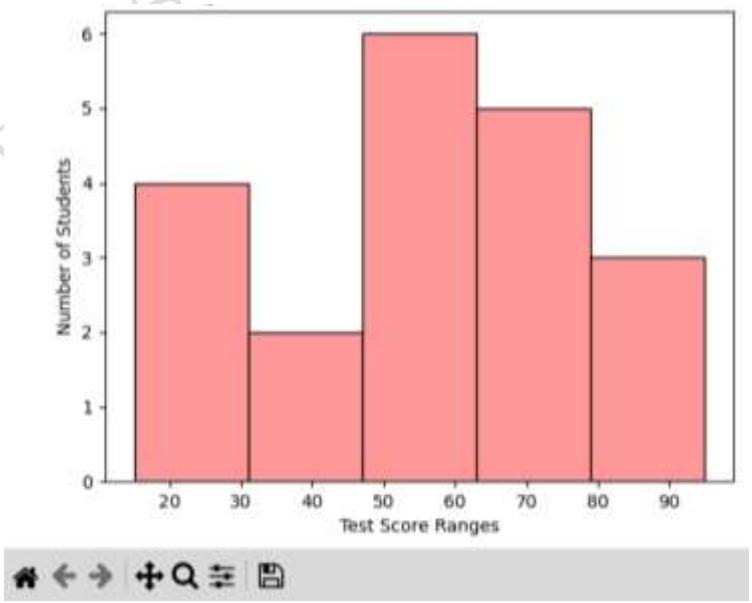



Fig. 4.8: Histogram for distribution of marks for 20 students' example 4.7

### 4.3.5. Pie charts

Pie charts illustrate the composition of a whole, showing the proportion of each value to the total sum. Pie charts are generally used for showing parts of a whole, or showing the breakdown into different categories. It is suggested to limit the number of categories (ideally less than five) to ensure clarity.

**Function:** The function, plt.pie(values, labels) is used to plot the pie chart.

**Example 4.8:** Create a Pie Chart showing Students Passing in Grades.

```
# Pi Chart showing Students Passing in Grades
import matplotlib.pyplot as plt

students = [40, 35, 15, 5]  # Number of students
grades = ['A','B','C','D']  # Corresponding Grades

# Create the pie chart
plt.pie(students,autopct="1.2f%",labels=grades)

# Add a title
plt.title('Pi Chart showing of Grades of Students')

# Ensure the circle is proportional
plt.axis('equal')
# Display the chart
plt.show()
```

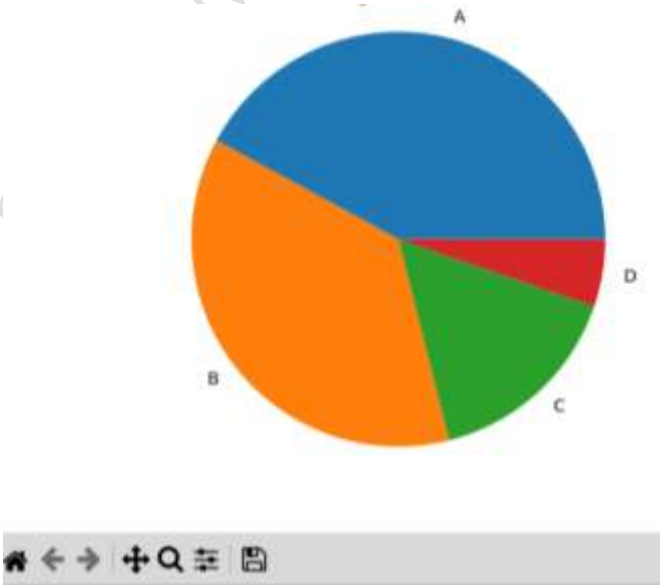



Fig. 4.9: Pi Chart showing of grades of students' example 4.8

### 4.3.6. Box plots

Box plots provide a standardized way of displaying the distribution of data based on a five-number summary: minimum, first quartile (Q1), median (Q2), third quartile (Q3), and

maximum. Box plots are normally used for displaying the distribution of a dataset, comparing distributions between multiple groups, identifying outliers.

**Function:** The function, plt.boxplot(data) is used to plot the Box plots.

**Example 4.9:** Create a Box plot for distribution of marks of students iin 5 subjects.

```
# Box Plot for Marks Distribution of 5 Subjects
import matplotlib.pyplot as plt
import numpy as np

# Sample data
# Data of 10 students for 5 subjects
English = np.array([78,36,89,87,79,32,57,68,65,82])
Physics = np.array([78,36,80,81,79,30,59,60,65,82])
Chemistry = np.array([78,59,89,87,79,62,57,68,60,82])
Maths = np.array([78,66,39,87,79,32,57,68,65,80])
AI = np.array([77,56,59,77,79,62,71,64,69,81])

# Combine the data into a list of arrays for plotting multiple box plots
data_to_plot = [English, Physics, Chemistry, Maths, AI]

# Create the box plot
plt.boxplot(data_to_plot)

# Add labels and a title for clarity
plt.xlabel('Five Subjects')
plt.ylabel('Student Marks')
plt.title('Box Plot for Marks Distribution of 5 Subjects')

# Customize x-axis ticks to label the cities
plt.xticks([1,2,3,4,5], ['English','Physics','Chemistry','Maths','AI'])

# Add a grid for better readability
plt.grid(axis='y', linestyle='--', alpha=0.7)

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

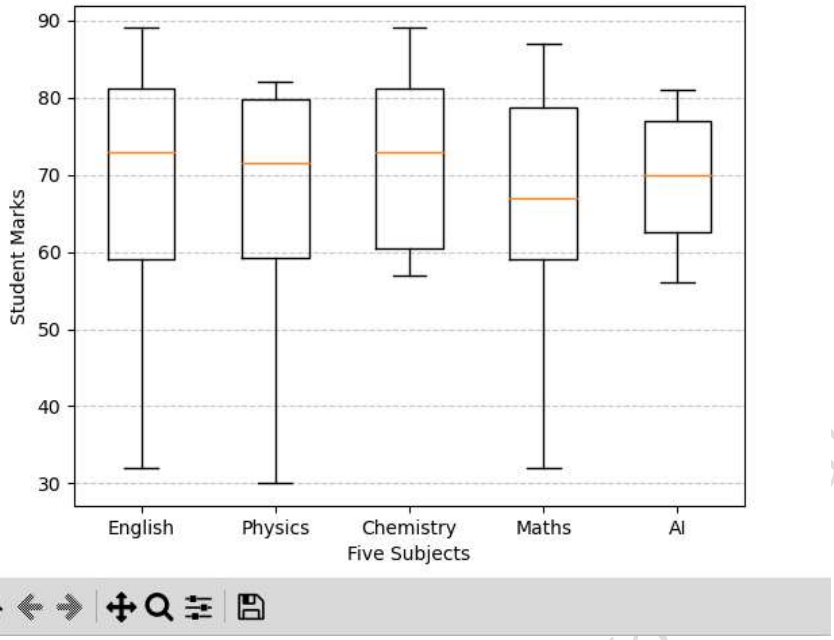



Fig. 4.10: Boc Plot for mark Distribution of 5 subjects' example 4.9

## Summary

- Data visualization turns complex results into easy-to-understand graphs.
- Matplotlib is a popular Python library for 2D plotting.
- Basic plots: line, bar, scatter, histogram, pie, box plots.
- Customizations: titles, labels, legends, grids, colors, markers, line styles.
- Real-world use: trends in sales, sports performance, financial analysis.
- Popular visualization tools include Matplotlib, Tableau, Power BI, Google Data Studio, and D3.js.

## Check Your Progress

**A. Multiple choice questions**

1. What is the primary purpose of Matplotlib? (a) Data manipulation and analysis (b) Building machine learning models (c) Creating static, animated, and interactive visualizations in Python (d) Web development
2. Which module within Matplotlib is commonly imported for creating plots, usually with the alias plt? (a) matplotlib.graph (b) matplotlib.figure (c) matplotlib.pyplot (d) matplotlib.visualize
3. Which function is used to create a simple line plot in Matplotlib? (a) plt.line() (b) plt.plot() (c) plt.draw() (d) plt.graph()
4. To display the plot on the screen, which function must be called after creating the plot? (a) plt.show() (b) plt.display() (c) plt.render() (d) plt.view()

5. How can you set the title of a plot in Matplotlib? (a) plt.set_title("My Plot") (b) plt.plot_title("My Plot") (c) plt.title("My Plot") (d) plt.add_title("My Plot")
6. Which function is used to add a label to the x-axis of a Matplotlib plot? (a) plt.ylabel() (b) plt.x_label() (c) plt.set_xlabel() (d) plt.xlabel()
7. To save a Matplotlib plot as an image file (e.g., PNG, JPEG), which method is used? (a) plt.save() (b) plt.export() (c) plt.save_image() (d) plt.savefig()
8. What is the purpose of the plt.legend() function? (a) To add a title to the plot (b) To display grid lines on the plot (c) To label different elements (lines, bars, etc.) within the plot (d) To change the background color of the plot.
9. Which of the following functions is used to create a scatter plot? (a) plt.line() (b) plt.scatter() (c) plt.plot(kind='scatter') (d) Both b and c.
10. How can you add grid lines to a graph in Matplotlib? (a) plt.gridlines() (b) plt.add_grid() (c) plt.grid() (d) plt.show_grid()

**B. Fill in the blanks**

1. Matplotlib is primarily used for creating static, animated, and _____ visualizations in Python.
2. The module within Matplotlib commonly imported for plotting functions is matplotlib. _____.
3. To create a simple line plot, the function plt._____ is used.
4. After creating a plot, you must call plt._____ to display the plot window.
5. To add a title to a plot, the function plt._____() is used.
6. The plt._____() function is used to add a label to the y-axis.
7. To save the current plot as an image file (e.g., PNG, PDF), the plt._____() function is utilized.
8. The plt._____() function displays labels for different lines or elements on a plot.
9. To create a scatter plot, the function plt._____() is commonly used.
10. You can add grid lines to a plot using the plt._____() function.

**C. State whether True or False**

1. Matplotlib is primarily used for numerical computation with arrays.
2. The matplotlib.pyplot module is a sub-library of Matplotlib.
3. The plt.show() function is optional and doesn't need to be called to display a plot.
4. You can set the title of a plot using plt.title().
5. Matplotlib is only capable of creating 2D plots.
6. The plt.xlabel() and plt.ylabel() functions are used to add labels to the x and y axes, respectively.
7. The plt.legend() function is used to add grid lines to a plot.
8. You can save a Matplotlib plot as a PNG image file using plt.savefig().

9. Matplotlib plots are always static and cannot be made interactive.
10. The plt.grid() function can be used to add a grid to a plot

**D. Answer the following questions in short**

1. What is the main purpose of the Matplotlib library in Python?
2. Which module of Matplotlib is most commonly used for creating plots in a MATLAB-like interface?
3. What are the basic steps involved in creating and displaying a simple plot using Matplotlib?
4. How can you add a title and labels to the x and y axes of a Matplotlib plot?
5. Explain the difference between a scatter plot and a line plot in Matplotlib.
6. What function would you use to save a Matplotlib plot to a file, and what types of files can you save it as?
7. What is the purpose of the plt.legend() function?
8. How do you create multiple plots within a single figure using Matplotlib?
9. Briefly describe the purpose of the plt.grid() function.
10. What is meant by "customizing a plot" in Matplotlib, and why is it important?

# Module 3. Machine Learning Models

## Module Overview

Suppose you're watching a music video on YouTube. Once the video ends, YouTube doesn't just stop. Instead, it automatically suggests other videos for you to watch next — based on your preferences, trends, and patterns. So, how does YouTube decide what to suggest?

The algorithm of YouTube looks at the behavior of similar users to make recommendations. The technique analyzes the content of the video like tags, title, description and compares it with what you've already watched. YouTube uses deep learning models to analyze and understand more complex data — including the actual video content. Fig 3.0 shows YouTube uses machine learning models. There are various machine learning models. In this Module we will understand different machine learning models.


Fig 3.0: YouTube Uses Machine Learning Models

## Learning Outcome

After completing this module, you will be able to:

- Classify problems into Supervised, Unsupervised, and Reinforcement Learning.
- Map the standard ML pipeline from raw data to model prediction.
- Build models to predict continuous targets (e.g., housing prices).
- Understand the relationship between independent variables and a dependent output.
- Group unlabeled data into meaningful segments using K-Means.
- Reduce high-dimensional data using PCA to improve model efficiency.
- Validate models using Confusion Matrices, Accuracy, and MSE.
- Distinguish between Overfitting and Underfitting.
- Identify and mitigate algorithmic bias in training data.
- Apply ethical frameworks to ensure fairness, transparency, and privacy.

## Module structure

Session 1. Introduction to Machine Learning (ML)

Session 2: Regression Analysis

Session 3. Clustering and Dimensionality Reduction

Session 4. Model Evaluation and Metrics

Session 5. Ethics, Bias, and Myths in AI



## Session 1. Introduction to Machine Learning (ML)

Think about how YouTube recommends the next video after you finish watching one. It doesn't guess randomly but learns from your past viewing habits and from patterns of millions of other users. This is possible because of machine learning models**,** which allow systems to learn from data and make smart decisions.


Fig. 3.1.1: Smart Video recommendation with Machine Learning

In this session, you will understand the difference between AI, ML, and Deep Learning, explore types of ML (supervised, unsupervised, reinforcement), learn the steps of the ML workflow, and discover how ML is applied in industries like healthcare, finance, retail, and entertainment.

### 1.1 Artificial Intelligence (AI)

AI is the broader concept of creating intelligent machines that can simulate human thinking and behavior. The goal is to enable machines to solve problems, reason, learn, and act autonomously. Examples of AI are Chess-playing bots (rule-based AI) and Smart home devices.

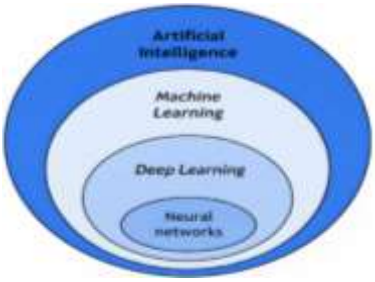



**Fig 3.1.2: AI, ML and DL**

**Machine Learning (ML)**

ML is a subset of AI. It enables machines to learn from data and make predictions or decisions without being explicitly programmed. Key Idea of ML is to "Learn from experience, that is, data." Fig 3.1.2 shows AI, ML and DL.

There are various categories of ML as given below:

1. Supervised Learning: It learns from labeled data, for example, spam detection in mails.
2. Unsupervised Learning: It finds patterns in unlabeled data, for example, customer segmentation in online shopping.
3. Reinforcement Learning: It learns by interacting with an environment, for example, game-playing bots.

**Deep Learning (DL)**

A subset of ML that uses artificial neural networks with many layers, hence "deep". It is excellent at processing unstructured data like images, audio, and natural language. For example, Self-driving cars, Facial recognition and Language translation makes use of DL.

**ML Workflow (Lifecycle)**

Machine Learning is not just about model building. It's a complete process:

**Step 1: Problem Definition**

(i) What do you want to predict or classify?

(ii) Example: Predicting house prices, detecting fraud, recommending products.

**Step 2: Data Collection**

(i) Gather data from various sources: databases, sensors, web scraping, APIs.

(ii) Quality and quantity of data heavily impact model performance.

**Step 3: Data Preprocessing**

(i) Cleaning: Handle missing, noisy, or duplicate data.

(ii) Transformation: Convert data into suitable formats (e.g., scaling numbers, encoding categories).

(iii) Splitting: Divide data into training, validation, and test sets.

**Step 4: Model Selection**

(i) Choose the right algorithm based on the task:

(a) Classification: Decision Tree, Random Forest, SVM, etc.

(b) Regression: Linear Regression, SVR, etc.

(c) Clustering: K-Means, DBSCAN.

**Step 5: Model Training**

(i) Feed training data to the algorithm so it can learn patterns.

**Step 6: Model Evaluation**

(i) Use test data to assess model performance using metrics like:

(a) Accuracy, Precision, Recall, F1-score (for classification),

(b) RMSE, MAE (for regression).

**Step 7: Deployment**

(i) Integrate the model into a real-world application (e.g., a web app, mobile app, or production server).

**Step 8: Monitoring & Maintenance**

(i) Monitor performance over time.

(ii) Update the model with new data as necessary to avoid "model drift."

**Importance of Machine Learning**

1. Data-Driven Decisions: Enables organizations to derive insights from vast datasets.
2. Automation: Automates repetitive tasks such as, email filtering, image classification.
3. Adaptability: Models can improve with new data without reprogramming.
4. Smart Applications: Forms the backbone of intelligent systems like chatbots and recommendation engines.
5. Competitive Advantage: Businesses that adopt ML can gain insights faster and improve decision-making.

**Applications of ML in Various Industries**

| Industry | Use Cases |
|---|---|
| Healthcare | Disease diagnosis (e.g., cancer detection), drug discovery, personalized medicine |
| Finance | Credit scoring, fraud detection, algorithmic trading, risk assessment |
| Retail & E-Commerce | Personalized recommendations, dynamic pricing, inventory forecasting |
| Manufacturing | Predictive maintenance, quality assurance using image recognition |
| Transportation | Self-driving cars, route optimization, traffic prediction |
| Agriculture | Crop yield prediction, pest detection, soil analysis |
| Education | Adaptive learning platforms, student performance prediction |
| Entertainment | Content recommendations (e.g., Netflix), automated dubbing or subtitles |

**Real-life Examples of ML**

**Voice Assistants:** It uses speech recognition (ASR), Natural Language Processing (NLP), and ML to:

(i) Understand spoken commands,

(ii) Interpret user intent,

(iii) Respond appropriately.

Examples: Amazon Alexa, Google Assistant, Apple Siri.

**Spam Detection:** ML models classify emails as spam or not spam based on patterns such as, certain keywords or sender behavior.

**Fraud Detection:** It monitors transactions in real-time. Flags anomalies based on learned behavior such as, unusual locations or amounts.

**Facial Recognition:** It is used in phone unlocking, social media tagging, and security systems. Uses deep learning to map facial features.

**Recommendation Systems:** Netflix, YouTube, and Spotify use ML to recommend content based on:

(i) Your past activity,

(ii) Similar users' preferences.

**Autonomous Vehicles:** It uses a combination of ML, computer vision, and sensor data. It makes real-time decisions such as, lane detection, obstacle avoidance and speed control.

**Google Maps / Uber:** It predicts traffic, suggests optimal routes. It uses historical and live data with ML models to estimate arrival times and detect road closures.

## Summary

Machine Learning (ML) is a branch of Artificial Intelligence focused on building systems that learn from data to improve performance without explicit programming. It encompasses supervised, unsupervised, and reinforcement learning to identify patterns, make predictions, and drive automated decision-making across diverse industries.

## Check Your Progress

**A. Multiple Choice Questions (MCQs)**

1. Machine Learning is a branch of: (a) Physics (b) Artificial Intelligence (c) Chemistry (d) Robotics
2. Which of these is an example of Unsupervised Learning? (a) Predicting house prices (b) Customer segmentation (c) Email spam detection (d) Stock price prediction
3. Which learning method uses trial-and-error with rewards and penalties? (a) Supervised Learning (b) Unsupervised Learning (c) Reinforcement Learning (d) Deep Learning
4. Which is NOT an application of ML? (a) Self-driving cars (b) Weather forecasting (c) Manual filing of records (d) Speech recognition
5. Deep Learning uses which key structure? (a) Decision Trees (b) Neural Networks (c) Histograms (d) Databases

**B. Fill in the Blanks**

1. _____ is a subfield of AI where machines learn from data.
2. In _____ learning, data comes with input-output labels.
3. Clustering is an example of _____ learning.
4. Reward and punishment signals are used in _____ learning.
5. The subset of ML inspired by the human brain is _____.

**C. True or False**

1. Machine Learning is unrelated to Artificial Intelligence.
2. Supervised learning requires labeled datasets.
3. Reinforcement learning works on feedback.
4. Deep Learning is less data-intensive than classical ML.
5. Email spam filtering is an application of ML.

**D. Short Answer Questions**

1. Define Machine Learning in simple words.
2. Explain the difference between Supervised and Unsupervised Learning.
3. Give one real-life example of Reinforcement Learning.
4. Why is data called the "fuel" of Machine Learning?
5. How is ML applied in YouTube or Netflix recommendations?



## Session 2 Regression Analysis

Imagine running an ice cream shop and wanting to know how temperature affects sales. On hotter days, sales increase; on cooler days, they drop. This can be done by using regression analysis. Fig 2.1 shows Ice cream sales Vs Temperature graph. Regression analysis helps us find and predict such relationships between variables.

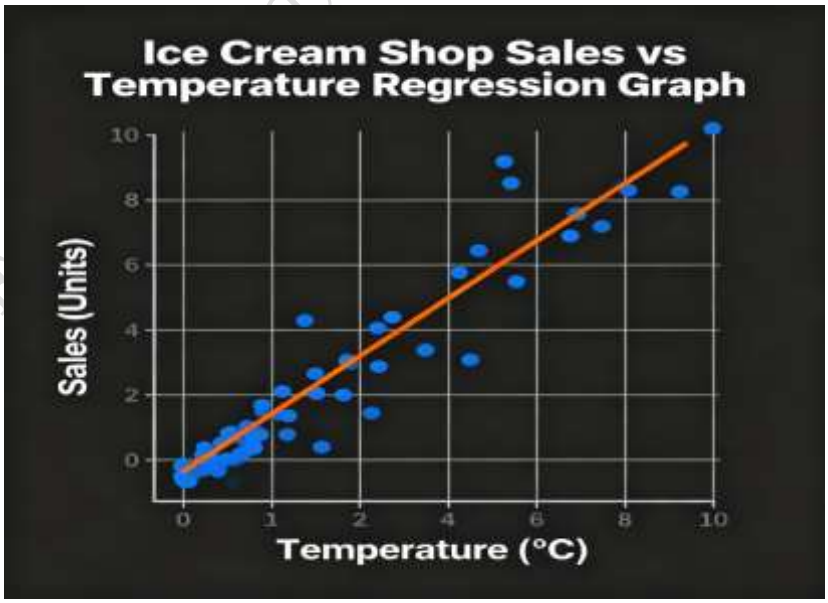



Fig 2.1: Ice cream sales Vs Temperature

Regression is a fundamental concept in machine learning and statistics. It is used for predicting outcomes based on input features. Two widely used regression techniques are

Linear Regression and Logistic Regression. While both fall under the category of supervised learning, they are applied to different types of prediction problems.

In this session, you will learn about Linear Regression for predicting continuous values (like house prices) and Logistic Regression for classification problems (like predicting pass/fail in exams). You will also explore real-world applications such as weather forecasting, stock price prediction, and spam detection.

**Linear Regression**

Linear Regression is used for predicting a continuous numerical outcome. It assumes a linear relationship between the independent variable(s) (input features) and the dependent variable (output).

**Mathematical Representation**

For one independent variable:

$$Y=\beta_0+\beta_1X+\epsilon$$

Where:

(i) Y = Dependent variable. It is target to be predicted.

(ii) X = Independent variable. It is input.

(iii) β = Intercept.

(iv) β1 = Coefficient or slope.

(v) ϵ = Error term

For multiple independent variables, we have, Multiple Linear Regression:

$$Y=\beta_0+\beta_1X_1+\beta_2X_2+...+\beta nXn+\epsilon$$

**Example: Predicting House Prices**

In this example, input features (X) could be area of the house, number of rooms, and location.

Output (Y) is the price of the house.

If the model learns the coefficients properly, it can estimate a house price when new input features are given.

**Logistic Regression**

Logistic Regression is used for predicting a categorical outcome (classification problem), usually binary (Yes/No, 0/1, True/False). Instead of predicting the value directly, it predicts the probability that the dependent variable belongs to a particular class.

**Mathematical Representation**

The logistic regression model uses the sigmoid function to map predictions to probabilities:

Where:

(i) P(Y=1|X) = Probability that output belongs to class 1

(ii) e = Base of natural log

(iii) The value is always between 0 and 1

The decision boundary is usually set at 0.5:

(i) If probability ≥ 0.5 → Class 1

(ii) If probability < 0.5 → Class 0

**Example**

Predicting if a Student Passes an Exam:
Input feature (X) could be number of hours studied.
Output (Y) is whether the student passes (1) or fails (0).

The model gives probability values such as, 0.8 → 80% chance of passing.

**Key Differences between Linear and Logistic Regression**

| Aspect | Linear Regression | Logistic Regression |
|---|---|---|
| Output Type | Continuous value (e.g., price, salary) | Probability of class (0 to 1), classification |
| Use Case | Prediction of numbers | Classification (binary or multiclass) |
| Equation | Straight line equation | Sigmoid/logit function |
| Examples | Predicting house prices, sales forecasting | Predicting spam emails, disease diagnosis |

**Applications**

1. Linear Regression:
   (i) Predicting stock prices
   (ii) Estimating demand for products
   (iii) Forecasting weather conditions
2. Logistic Regression:
   (i) Predicting whether an email is spam
   (ii) Credit card fraud detection
   (iii) Medical diagnosis (disease: yes/no)

Note that Linear Regression is Best for continuous prediction and Logistic Regression is Best for classification (probability-based) predictions.

**Example 1: Linear Regression**

Problem: Predict the house price based on its size.

Dataset (simplified):

| House Size (sq. ft) | Price (₹ in lakhs) |
|---|---|
| 1000 | 50 |
| 1500 | 65 |
| 2000 | 80 |
| 2500 | 95 |
| 3000 | 110 |

Model Equation:

Price = $\beta 0+\beta 1\times$SizePrice

After training, suppose the model learns:

Price = 20+0.03 × SizePrice

Prediction:

For a 2000 sq.ft house:

Price = 20 + 0.03 × 2000 = 80 lakhs

It matches real data.

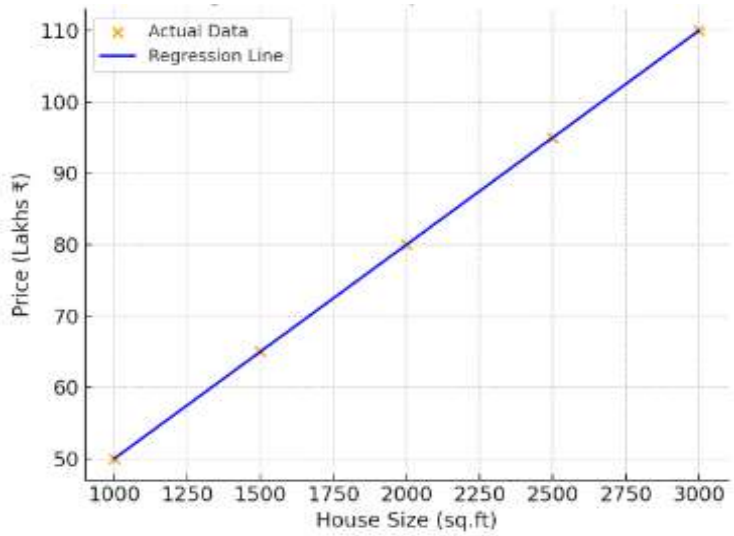



**Fig 2.2: Linear Regression**

**Example 2: Logistic Regression**

Problem: Predict whether a student will pass (1) or fail (0) based on hours studied.

Dataset (simplified):

| Hours Studied | Result (Pass=1, Fail=0) |
|---|---|
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 1 |
| 5 | 1 |
| 6 | 1 |

Model Equation (using sigmoid function):

$$P(Pass|Hours) = \frac{1}{1+e^{-(-4+1.2\times Hours)}}$$

Prediction:

- If a student studies **3 hours**:

$$P(Pass) = \frac{1}{1+e^{-(-4+1.2\times 3)}} \approx 0.31$$

→ 31% chance of passing → Model predicts **Fail**

**If a student studies 5 hours:**

P(Pass)≈0.88P(Pass)→ 88% chance of passing → Model predicts Pass

Logistic regression doesn't give a direct 0/1 but a probability, then we apply a threshold (usually 0.5).

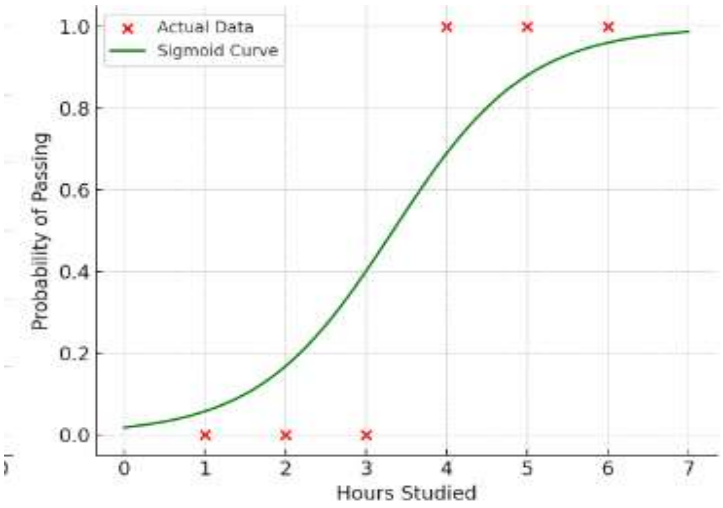



**Fig. 2.3: Logistic Regression**

**Python Program for Regression**

```
# Linear and Logistic Regression Examples
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression, LogisticRegression
# ------------------------------
# Example 1: Linear Regression
# ------------------------------
# Dataset: House Size vs Price
house_size = np.array([1000, 1500, 2000, 2500, 3000]).reshape(-1, 1)
house_price = np.array([50, 65, 80, 95, 110])  # in lakhs
# Train Linear Regression model
lin_reg = LinearRegression()
lin_reg.fit(house_size, house_price)
# Predict values
size_test = np.linspace(900, 3100, 100).reshape(-1, 1)
price_pred = lin_reg.predict(size_test)
# Plot Linear Regression
plt.subplot(1, 2, 1)
plt.scatter(house_size, house_price, color='orange', label='Actual
Data')
plt.plot(size_test, price_pred, color='blue', label='Regression Line')
plt.title("Linear Regression: House Price vs Size")
plt.xlabel("House Size (sq.ft)")
plt.ylabel("Price (Lakhs ₹)")
```

```
plt.legend()

# ------------------------------
# Example 2: Logistic Regression
# ------------------------------

# Dataset: Hours Studied vs Exam Result (Pass=1, Fail=0)
hours = np.array([1, 2, 3, 4, 5, 6]).reshape(-1, 1)
result = np.array([0, 0, 0, 1, 1, 1])  # binary outcome

# Train Logistic Regression model
log_reg = LogisticRegression()
log_reg.fit(hours, result)

# Predict probabilities for continuous values
hours_test = np.linspace(0, 7, 100).reshape(-1, 1)
prob_pass = log_reg.predict_proba(hours_test)[:, 1]

# Plot Logistic Regression
plt.subplot(1, 2, 2)
plt.scatter(hours, result, color='red', label='Actual Data')
plt.plot(hours_test, prob_pass, color='green', label='Sigmoid Curve')
plt.title("Logistic Regression: Exam Pass vs Hours Studied")
plt.xlabel("Hours Studied")
plt.ylabel("Probability of Passing")
plt.legend()
plt.tight_layout()
plt.show()
```

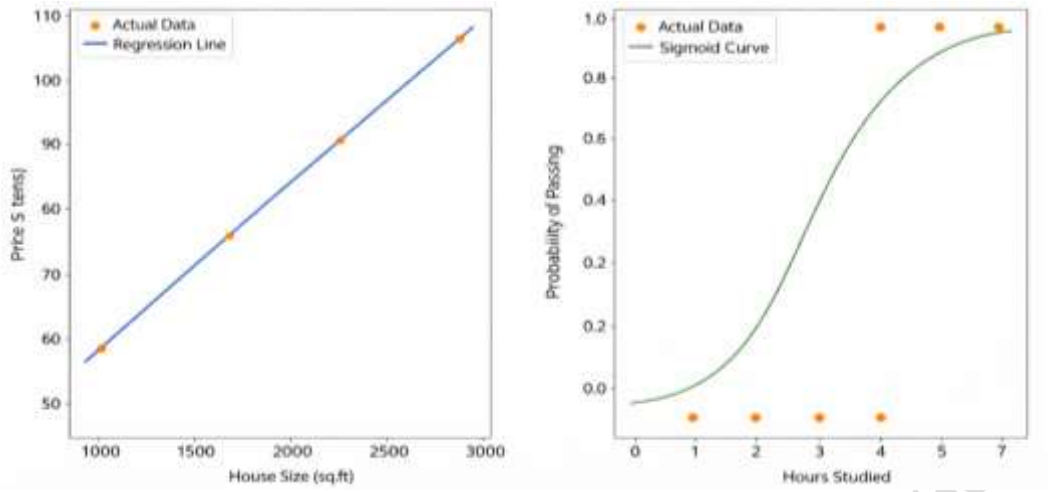
**This program does:**

1. Linear Regression
   (i) Trains a model to predict house prices from house size.
   (ii) Plots the actual data (orange points) and regression line (blue).
2. Logistic Regression
   (i) Trains a model to predict pass/fail based on hours studied.
   (ii) Plots actual data (red points) and sigmoid probability curve (green).

**Sample Output of Program**

```
Linear Regression (House Price vs Size)
Coefficient (slope): 0.03
Intercept: 20.0
Equation: Price = 20 + 0.03 * Size
Logistic Regression (Exam Pass vs Hours Studied)
```

```
Coefficient: 1.12
Intercept: -3.92
```




```
Equation: P(Pass) = 1 / (1 + e^(-(-3.92 + 1.12 * Hours)))
```

## Summary

Regression Analysis is a fundamental supervised learning technique used to model the relationship between variables. It focuses on predicting a continuous numerical output (such as price, temperature, or sales) based on one or more input features.

## Check Your Progress

### A. Multiple Choice Questions (MCQs)

1. Regression is mainly used for: (a) Predicting continuous values (b) Grouping data into clusters (c) Sorting text (d) Drawing charts
2. Which of the following is an example of Linear Regression? (a) Predicting pass/fail in exams (b) Predicting the price of a house based on area (c) Grouping customers into segments (d) Detecting spam emails
3. Logistic Regression is used for: (a) Predicting continuous values (b) Binary classification problems (c) Clustering tasks (d) Data visualization
4. The output of Logistic Regression is always in terms of: (a) Graphs (b) Probabilities (c) Raw numbers (d) Tables
5. Which of the following is NOT a regression type? (a) Linear Regression (b) Logistic Regression (c) Polynomial Regression (d) Cluster Regression

### B. Fill in the Blanks

1. _____ regression predicts continuous values like salary or house prices.
2. Logistic Regression is commonly used for _____ classification.
3. The output of Logistic Regression is a _____ value between 0 and 1.
4. Linear Regression draws a best-fit _____ through the data points.
5. Logistic Regression is based on the _____ function.

**C. True or False**

1. Linear Regression is used for predicting categories.
2. Logistic Regression is a classification technique.
3. Regression analysis is widely used in finance and economics.
4. Logistic Regression can only solve multi-class problems.
5. Linear Regression assumes a linear relationship between variables.

**D. Short Answer Questions**

1. Define Regression in Machine Learning.
2. Differentiate between Linear and Logistic Regression.
3. Give one real-life example where Linear Regression can be applied.
4. Why is Logistic Regression used for classification?
5. Explain the term "best-fit line" in Linear Regression.



## Session 3. Clustering and Dimensionality Reduction

Imagine you are the manager of a supermarket. You have data about customers, such as:

(i) Age

(ii) Annual Income

(iii) Spending Score (1–100, based on purchase history)

You want to divide your customers into groups for targeted marketing. You can design discount coupons differently for each group. You may discover sub-groups inside clusters such as, high-income spenders split into "tech lovers" vs. "grocery bulk buyers". Suppose your supermarket collects 50 features about each customer such as, age, salary, purchase frequency, product categories, etc. You want plot customers on a 2D graph, where clusters are visible and easy to understand. All these can be achieved by using clustering and dimensionality reduction techniques. Fig 3.3.1 shows clustering of supermarket data. In this session we will discuss about these techniques.

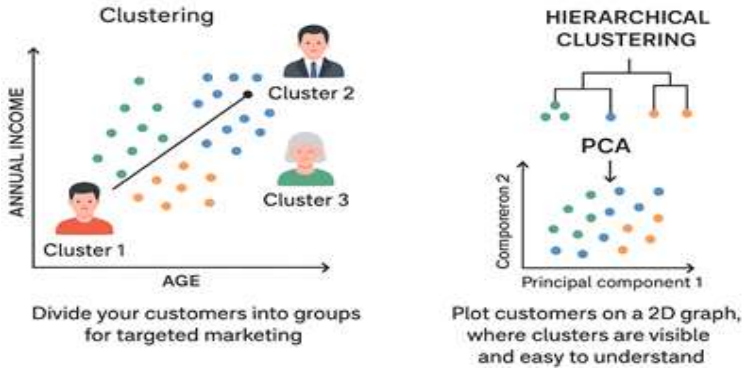



Fig 3.1: Clustering of Supermarket Data

In this session, you will understand clustering techniques like K-Means and Hierarchical Clustering**,** learn about Principal Component Analysis (PCA) for reducing dimensions, and explore applications in customer segmentation, image recognition, and data visualization.

**Clustering**

Clustering is an unsupervised machine learning technique used to group similar data points into clusters, such that:

(i) Data points in the same cluster are more similar to each other.

(ii) Data points in different clusters are more dissimilar.

It is widely used in data mining, pattern recognition, and customer segmentation.

**(a) K-Means Clustering**

K-Means is the most popular clustering algorithm.

**Steps of the Algorithm**:

1. Choose the number of clusters (K).
2. Randomly initialize K centroids.
3. Assign each data point to the nearest centroid (form clusters).
4. Recalculate the centroid of each cluster.
5. Repeat steps 3–4 until centroids do not change significantly.

Mathematical Objective of this technique is to minimize the sum of squared distances between data points and their assigned cluster centroids.

$$J = \sum_{i=1}^{K} \sum_{x \in C_i} \|x - \mu_i\|^2$$

Where:

- K = number of clusters
- $C_i$ = cluster i
- $\mu_i$ = centroid of cluster i

**Example: K-Means Clustering**

Problem: Suppose we have data about customers in a shopping mall:

(i) Annual Income (in ₹1000s)

(ii) Spending Score (1–100, based on behavior)

We want to group customers into 3 clusters (K=3) for marketing.

**Step 1. Sample Data**

| Customer | Annual Income | Spending Score |
|---|---|---|
| C1 | 15 | 39 |
| C2 | 16 | 81 |
| C3 | 17 | 6 |
| C4 | 18 | 77 |

| Customer | Annual Income | Spending Score |
|---|---|---|
| C5 | 19 | 40 |
| C6 | 20 | 76 |
| C7 | 21 | 6 |
| C8 | 22 | 94 |

**Step 2. Choose K = 3**

We want 3 clusters such as, Low spenders, medium spenders and High spenders.

Step 3: Run K-Means Algorithm

1. Randomly select 3 initial centroids (say C1, C4, C7).
2. Assign each customer to the nearest centroid (using Euclidean distance).
3. Calculate the new centroid of each cluster.
4. Repeat until centroids don't change.

**Step 4. Final Clusters**

1. Cluster 1 (Low spenders) → (C3, C7) → Low income, low spending.
2. Cluster 2 (Moderate spenders) → (C1, C5) → Moderate spending.
3. Cluster 3 (High spenders) → (C2, C4, C6, C8) → High income, high spending.

**Step 5. Interpretation**

1. Cluster 1 → Customers who don't shop much → Give them small offers.
2. Cluster 2 → Customers with medium spending → Attract them with loyalty points.
3. Cluster 3 → Big spenders → Provide premium offers and VIP services.

This shows how K-Means helps in customer segmentation.

**Python Program**

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.cluster import KMeans
# -----------------------------
# Step 1. Customer Data
# -----------------------------
data = {
   "Customer": ["C1", "C2", "C3", "C4", "C5", "C6", "C7", "C8"],
    "Annual Income": [15, 16, 17, 18, 19, 20, 21, 22],
    "Spending Score": [39, 81, 6, 77, 40, 76, 6, 94]
}
df = pd.DataFrame(data)
# Convert to numpy array for clustering
X = df[["Annual Income", "Spending Score"]].values
# -----------------------------
```

```
# Step 2: Apply K-Means
# ----------------------------
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
df["Cluster"] = kmeans.fit_predict(X)
centroids = kmeans.cluster_centers_
# ----------------------------
# Step 3: Print Results
# ----------------------------
print("Cluster Assignments:")
print(df)
# ----------------------------
# Step 4: Plot Clusters
# ----------------------------
plt.figure(figsize=(8,6))
# Plot each cluster
for i in range(3):
    cluster_points = df[df["Cluster"] == i]
    plt.scatter(cluster_points["Annual Income"], cluster_points["Spend-
ing Score"], label=f"Cluster {i+1}")

# Plot centroids
plt.scatter(centroids[:,0], centroids[:,1], s=200, c='black',
marker='X', label='Centroids')
plt.title("K-Means Clustering: Customer Segmentation")
plt.xlabel("Annual Income (₹1000s)")
plt.ylabel("Spending Score (1-100)")
plt.legend()
plt.grid(True)
plt.show()
```

**Here's the K-Means clustering result**:

1. Each color represents a cluster of customers**.**
2. Black X marks are the centroids (cluster centers).
3. Customers with similar Annual Income and Spending Score are grouped together**.**

This is how businesses identify low, medium, and high spenders for targeted marketing. Figure 3.3.2 shows result.

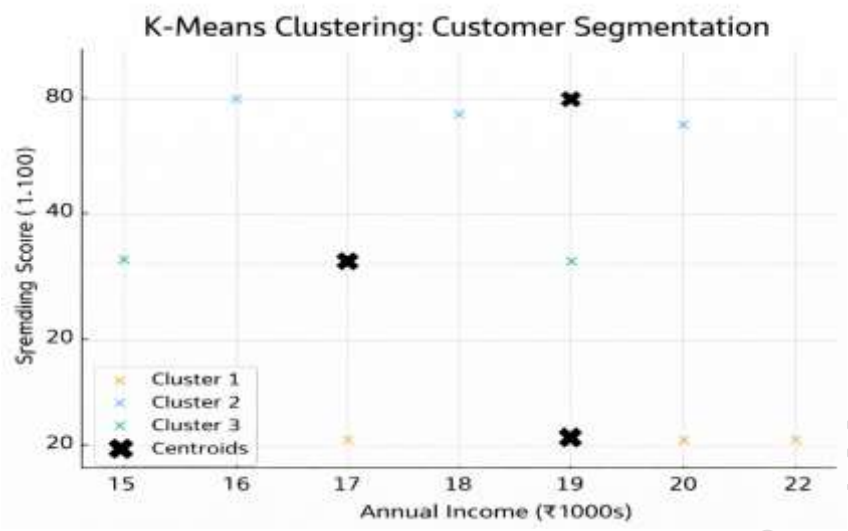



**Fig 3.3.2: Result**

**(b) Hierarchical Clustering**

Hierarchical clustering builds a tree-like structure (dendrogram) to represent nested clusters.

There are two main approaches:

1. Agglomerative (Bottom-Up):
   (i) Start with each data point as a single cluster.
   (ii) Merge the closest clusters step by step until only one cluster remains.
2. Divisive (Top-Down):
   (i) Start with one big cluster.
   (ii) Split clusters recursively into smaller ones.

Distance Measures techniques are Euclidean distance, Manhattan distance and Cosine similarity.

Linkage Criteria used are Single linkage (nearest), Complete linkage (farthest) and Average linkage.

**Example: Hierarchical Clustering**

Problem: We have the same shopping mall customers data (Annual Income vs Spending Score).

We want to use Hierarchical Clustering to group them.

Sample Data

| Customer | Annual Income | Spending Score |
|---|---|---|
| C1 | 15 | 39 |
| C2 | 16 | 81 |
| C3 | 17 | 6 |
| C4 | 18 | 77 |
| C5 | 19 | 40 |

| Customer | Annual Income | Spending Score |
|---|---|---|
| C6 | 20 | 76 |
| C7 | 21 | 6 |
| C8 | 22 | 94 |

**Step 1. Distance Calculation**

We calculate distances (Euclidean) between all customers.

For example:

$$d(C1, C2) = \sqrt{(15-16)^2 + (39-81)^2} \approx 42.01$$

This is repeated for all pairs.

**Step 2. Agglomerative Clustering**

(i) Start with each customer as its own cluster.

(ii) Iteratively merge the two closest clusters based on chosen linkage criteria (single, complete, average).

**Step 3. Dendrogram**

We build a tree diagram (dendrogram) showing how clusters merge step by step.

Example grouping:

(i) (C3, C7) → both low income & low spending.

(ii) (C2, C4, C6, C8) → high income, high spenders.

(iii) (C1, C5) → moderate group.

Finally, 3 main clusters appear.

**Step 4. Interpretation**

(i) Cluster 1: Low income, low spending → occasional shoppers.

(ii) Cluster 2: Medium income, medium spending → loyal customers.

(iii) Cluster 3: High income, high spending → premium customers.

This approach is different from K-Means because we don't need to predefine K, we can see nested clusters and choose cut-off later.

**Python Program**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import linkage, dendrogram, fcluster
#-----------------------------
# Step 1: Customer Data
# -----------------------------
data = {
    "Customer": ["C1", "C2", "C3", "C4", "C5", "C6", "C7", "C8"],
    "Annual Income": [15, 16, 17, 18, 19, 20, 21, 22],
```

```
    "Spending Score": [39, 81, 6, 77, 40, 76, 6, 94]
}
df = pd.DataFrame(data)
X = df[["Annual Income", "Spending Score"]].values
# ----------------------------
# Step 2: Hierarchical Clustering (Ward method)
# ----------------------------
Z = linkage(X, method='ward')
# Dendrogram
plt.figure(figsize=(8,6))
dendrogram(Z, labels=df["Customer"].values, leaf_rotation=90,
leaf_font_size=12)
plt.title("Hierarchical Clustering Dendrogram")
plt.xlabel("Customers")
plt.ylabel("Euclidean Distance")
plt.show()
# ----------------------------
# Step 3: Form Clusters
# ----------------------------
clusters = fcluster(Z, 3, criterion='maxclust')
df["Cluster"] = clusters

print("Cluster Assignments:")
print(df)

# ----------------------------
# Step 4: Scatter Plot of Clusters
# ----------------------------
plt.figure(figsize=(8,6))

for i in range(1, 4):  # clusters are labeled 1,2,3
    cluster_points = df[df["Cluster"] == i]
    plt.scatter(cluster_points["Annual Income"],
                cluster_points["Spending Score"],
                label=f"Cluster {i}")
plt.title("Hierarchical Clustering: Customer Segmentation")
plt.xlabel("Annual Income (₹1000s)")
plt.ylabel("Spending Score (1-100)")
plt.legend()
plt.grid(True)
plt.show()
```



**Output:** Fig 3.3.3 shows result.

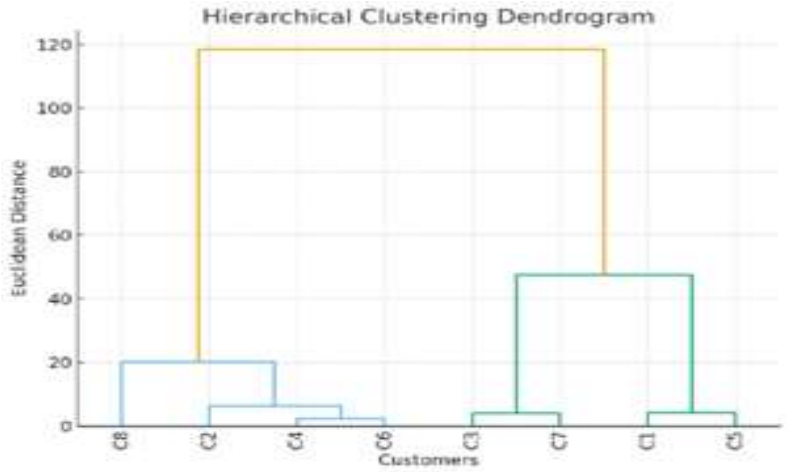



Fig 3.3: Result

**Interpretation of result:**

1. Cluster 1 (C2, C4, C6, C8): High-income, high spenders → **Premium customers**.
2. Cluster 2 (C3, C7): Low-income, low spenders → **Occasional buyers**.
3. Cluster 3 (C1, C5): Medium spenders → **Moderate customers**.

**Dimensionality Reduction**

Dimensionality Reduction is the process of reducing the number of input variables (features) while preserving as much important information as possible.

This helps:

- Remove noise/redundancy.
- Improve visualization.
- Reduce computation.

**(a) Principal Component Analysis (PCA)**

PCA is a statistical technique that transforms original correlated variables into a new set of uncorrelated variables called principal components (PCs).

Steps in PCA:

1. Standardize the data.
2. Compute the covariance matrix.
3. Find eigenvalues and eigenvectors.
4. Select top-k eigenvectors (components).
5. Project original data onto new axes (principal components).

Key Idea:

- 1st Principal Component → direction of maximum variance.
- 2nd Principal Component → orthogonal to the 1st, captures next highest variance.

**Example**

Problem Statement: As a supermarket manager, you have customer data with several variables:

1. Age (numeric, years)

2. Annual Income (in $000s)
3. Spending Score (1–100, assigned by the store based on purchase behavior)

Now you want to:

1. Understand patterns among customers.
2. Reduce the complexity of data by combining correlated features.
3. Visualize customers in a 2D plot (instead of 3D or higher dimensions).

This is where Principal Component Analysis (PCA) is applied.

**Steps in the PCA Problem**

1. Collect Data

   Customer features are given as a table (Age, Income, Spending).

2. Standardize Features

   Since Age (~20–70), Income (~10–150), and Spending (~1–100) have different scales, PCA requires standardization to treat them equally.

3. Apply PCA
   (i) Compute linear combinations of original features.
   (ii) Generate new uncorrelated variables (Principal Components, PCs).
   (iii) PCs are ordered: PC1 explains maximum variance, then PC2, and so on.
4. Analyze Results
   (i) Explained Variance Ratio (EVR): tells how much information each PC captures.
   (ii) Loadings (component matrix): show how strongly each original feature contributes to a PC.
   (iii) Scores: transformed values of each customer in PCA space.
5. Use Case
   (i) Plot customers in PC1–PC2 space → clusters or natural groupings may appear.
   (ii) Reduce dimensionality → easier visualization and faster downstream algorithms (e.g., clustering, classification).

**Interpretation from Our Example**

- PC1 (≈58%): Captures the combination of Annual Income + Spending Score.
- PC2 (≈34%): Captures mainly the Age factor.
- Together, PC1 + PC2 explain ~92% of the total variation → data can be represented well in 2D.

**In short, the problem PCA solves here is:**

"How do we reduce 3 customer variables into 2 dimensions, while retaining >90% of the information, so that we can visualize and analyze customer segments more effectively?"

Python Code

```
# PCA Example: Supermarket Customer Data
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
# ----------------------------
# Step 1: Create synthetic dataset
# ----------------------------
rng = np.random.default_rng(42)
n = 200  # number of customers
age = rng.normal(35, 10, n).clip(18, 70)              # Age in years
income = rng.normal(65, 20, n).clip(10, 150)          # Annual Income in
$000s
spending = (100 - 0.5*(age-35) + 0.8*(income-65)
            + rng.normal(0, 10, n)).clip(1, 100)      # Spending Score
(1–100)
# Create DataFrame
df = pd.DataFrame({
    "Age": age.round(1),
    "AnnualIncome": income.round(1),
    "SpendingScore": spending.round(1)
})
print("First 5 rows of data:")
print(df.head())
# ----------------------------
# Step 2: Standardize the features
# ----------------------------
X = df[["Age", "AnnualIncome", "SpendingScore"]].values
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# ----------------------------
# Step 3: Apply PCA
# ----------------------------
pca = PCA(n_components=3)
X_pca = pca.fit_transform(X_scaled)
# Explained variance ratio
print("\nExplained Variance Ratio:", pca.explained_variance_ratio_)
# Loadings (contribution of features to PCs)
loadings = pd.DataFrame(
    pca.components_.T,
    columns=["PC1", "PC2", "PC3"],
    index=["Age", "AnnualIncome", "SpendingScore"]
)
```

```
print("\nPCA Loadings:")
print(loadings.round(3))
# ----------------------------
# Step 4: Visualization
# ----------------------------
# Scree Plot - explained variance
plt.figure(figsize=(6,4))
plt.bar(range(1, 4), pca.explained_variance_ratio_,
tick_label=["PC1","PC2","PC3"])
plt.ylabel("Explained Variance Ratio")
plt.title("PCA - Explained Variance by Components")
plt.show()
# Scatter Plot - first 2 PCs
plt.figure(figsize=(6,5))
plt.scatter(X_pca[:,0], X_pca[:,1], alpha=0.7)
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.title("Customers in PCA Space (PC1 vs PC2)")
plt.show()
```

**Output**

Fig. 3.4 shows the result.

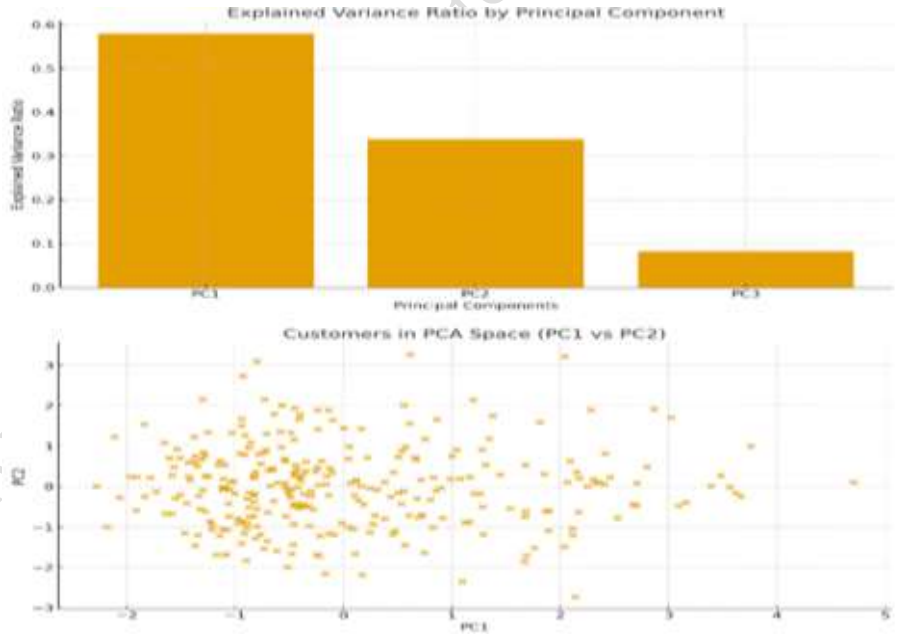



Fig. 3.4: Result

**Applications**

(a) Customer Segmentation

(i) Businesses use K-Means clustering to group customers based on shopping behavior (age, spending habits, product preference).

(ii) Helps in targeted marketing and personalized recommendations.

(b) Grouping Data

(i) Scientific research: Grouping genes with similar expression patterns.

(ii) Document clustering: Grouping news articles by topics.

(iii) Image segmentation: Dividing an image into meaningful parts (e.g., separating background and objects).

(c) Dimensionality Reduction (PCA)

(i) Reducing image dimensions in face recognition.

(ii) Preprocessing step before applying machine learning models.

(iii) Visualization of high-dimensional data (e.g., plotting 100 features into 2D or 3D space).

## Summary

Clustering is unsupervised learning used to group similar data points. K-Means is centroid-based; Hierarchical builds a tree structure. PCA reduces dimensions while keeping most variance. Applications such as customer segmentation, data grouping, visualization, noise reduction makes use of clustering and PCA.

## Check Your Progress

**A. Multiple Choice Questions (MCQs)**

1. Clustering belongs to which type of learning? (a) Supervised (b) Unsupervised (c) Reinforcement (d) Deep
2. Which algorithm is most commonly used for clustering? (a) Linear Regression (b) K-Means (c) Logistic Regression (d) Decision Trees
3. Dimensionality reduction is used to: (a) Increase the number of variables (b) Reduce the number of features while keeping important information (c) Remove all variables (d) Make data less accurate
4. PCA stands for: (a) Principal Component Analysis (b) Primary Cluster Algorithm (c) Parallel Component Analysis (d) Partitioned Cluster Approach
5. Which of the following is an application of clustering? (a) Predicting house prices (b) Customer segmentation (c) Forecasting stock market values (d) Calculating loan interest

**B. Fill in the Blanks**

1. _____ learning is used when data is not labeled.
2. The most popular clustering algorithm is _____.
3. Dimensionality reduction helps in removing _____ features.
4. _____ is a technique used to reduce dimensions in data.
5. Clustering groups data points based on their _____.

**C. True or False**

1. Clustering is always supervised.
2. K-Means algorithm requires specifying the number of clusters beforehand.
3. PCA is a method of dimensionality reduction.
4. Dimensionality reduction helps in visualization of complex data.
5. Clustering can be used in customer segmentation.

**D. Short Answer Questions**

1. Define clustering with an example.
2. What is dimensionality reduction? Why is it important?
3. Explain the K-Means algorithm in simple terms.
4. State one real-life use of clustering in marketing.
5. What problem does PCA solve in Machine Learning?



## Session 4. Model Evaluation and Metrics

Imagine your college wants to predict whether students will pass or fail an exam, based on:

1. Hours studied per week
2. Attendance percentage

This is possible by training a machine learning model with past student data. Figure 4.1 shows predicted pass matrix. In this session we will discuss about model evaluation and metrics.

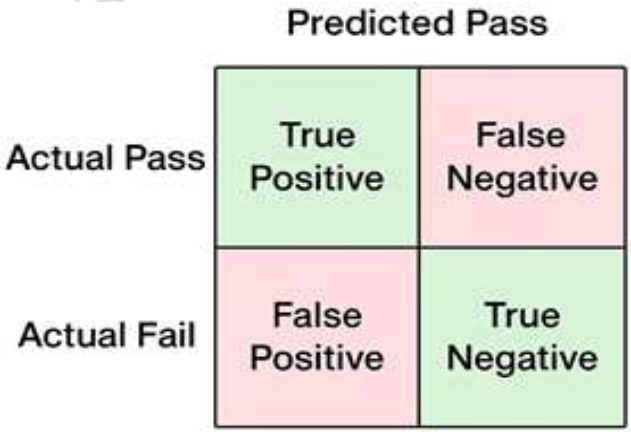



Fig. 4.1: Predicted Pass Matrix

In this session, you will learn about key evaluation measures like Accuracy, Precision, Recall, F1-score, Specificity, and Confusion Matrix**.** You will also explore issues like over-fitting and under-fitting**,** understand cross-validation**,** and practice techniques like hyper parameter tuning to make models reliable.

**Accuracy, Precision, Recall**

When evaluating classification models, we don't just look at overall accuracy, but other metrics help understand model behavior, especially when data is imbalanced.

**Accuracy:**

It measures how often the classifier is correct. It can be misleading if classes are imbalanced, for example, detecting rare fraud cases.

$$Accuracy = \frac{\text{Correct Predictions}}{\text{Total Predictions}}$$

**Precision:**

Precision is used to measure out of all predicted positives, how many were truly positive. High precision means low false alarms.

$$Precision = \frac{TP}{TP + FP}$$

For example, in spam detection, precision means most emails predicted "spam" are actually spam.

**Recall:**

Recall is used to measure out of all actual positives, how many did we correctly identify. High recall means fewer misses.

$$Recall = \frac{TP}{TP + FN}$$

For example, in cancer detection, recall is important, you don't miss patients who actually have cancer.

Precision and Recall often trade off, that is, higher one may reduce the other.

**F1-Score:**

The F1-score is the harmonic mean of Precision and Recall.

F1-Score is needed because, sometimes Precision and Recall tell different stories. For example, Precision measures how many of the students we predicted as Pass actually passed. Recall is a measure of how many of the students who really passed were correctly predicted. If Precision is high but Recall is low (or vice versa), F1-score balances both.

$$F1 = \frac{2 \times (Precision \times Recall)}{Precision + Recall}$$

It is used to interpret that our model is good at both predicting "Pass" correctly and not missing too many actual "Pass" students.

**Specificity (True Negative Rate):**

Specificity measures how well the model identifies the negative class (students who actually Fail).

$$Specificity = \frac{TN}{TN + FP}$$

For example, if specificity = 75%, then, that means out of all students who really Fail, 75% are correctly predicted as Fail.

**Quick Comparison**

1. Recall (Sensitivity): Focuses on correctly catching the Pass students.
2. Specificity: Focuses on correctly catching the Fail students.
3. F1-Score: Balances Precision & Recall, a good measure when classes are imbalanced (e.g., many more passes than fails).

**Example:**

Imagine a medical test.

1. Recall = how many sick people are detected (don't miss positives).
2. Specificity = how many healthy people are correctly told they are healthy (don't raise false alarms).
3. F1-score = balances both when deciding how reliable the test is overall.

**Confusion Matrix**

A confusion matrix is a table summarizing predictions vs actual outcomes:

| | **Predicted Positive** | **Predicted Negative** |
|---|---|---|
| **Actual Positive** | True Positive (TP) | False Negative (FN) |
| **Actual Negative** | False Positive (FP) | True Negative (TN) |

Here,

1. TP means correctly predicted positives.
2. FP means incorrectly predicted positives ("false alarm").
3. FN means missed positives.
4. TN means correctly predicted negatives.

From this table, we compute Accuracy, Precision, Recall, F1-score and specificity.

**Example:**

For example, in student pass prediction, suppose we get confusion matrix as:

| | **Predicted Pass** | **Predicted Fail** |
|---|---|---|
| **Actual Pass** | 70 (TP) | 10 (FN) |
| **Actual Fail** | 5 (FP) | 15 (TN) |

From this matrix we can compute accuracy, precision, recall, F1-score, and specificity.

Accuracy:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$= \frac{70 + 15}{70 + 15 + 5 + 10} = \frac{85}{100} = 0.85 = 85\%$$

Precision:

$$Precision = \frac{TP}{TP + FP}$$

$$= \frac{70}{70 + 5} = \frac{70}{75} \approx 0.9333 = 93.3\%$$

Recall:

$$Recall = \frac{TP}{TP + FN}$$

$$= \frac{70}{70 + 10} = \frac{70}{80} = 0.875 = 87.5\%$$

F1-Score:

$$F1 = \frac{2 \times (Precision \times Recall)}{Precision + Recall}$$

$$= \frac{2 \times (0.9333 \times 0.875)}{0.9333 + 0.875}$$

$$= \frac{2 \times 0.8166}{1.8083} = \frac{1.6332}{1.8083} \approx 0.903 = 90.3\%$$

Specificity:

$$Specificity = \frac{TN}{TN + FP}$$

$$= \frac{15}{15 + 5} = \frac{15}{20} = 0.75 = 75\%$$

**Overfitting vs. Underfitting**

The machine learning model can be overfitting, underfitting or balanced as discussed below.

**Underfitting:**

Here model is too simple to capture the underlying pattern. It performs poorly on both training and test data. It cannot capture the real pattern in the data.

The symptom is low accuracy everywhere. For example, using a straight line to fit highly curved data. Model has high bias (too few rules, too rigid).

Good models balance bias and variance to avoid both extremes.

**Example:**

Imagine predicting exam results (Pass/Fail) using:

(i) Hours studied

(ii) Attendance

If your model uses a very simple rule like:

"If attendance > 50% → Pass, else Fail"

This ignores study hours, motivation, and other factors. Many good students who studied hard but had 49% attendance may be wrongly classified.

**Result:** Model underfits and performs poorly everywhere.

**Overfitting:**

Here the model is too complex. It learns not only the patterns but also the noise in the training data. Here model learns the training data too well, including noise. It performs very well on training but poorly on new/unseen data. Model has high variance (too many rules, memorizes training set).

The symptom is high training accuracy and low-test accuracy. For example, a decision tree that keeps splitting until every training point is perfectly classified.

**Example:**

Suppose we use a very deep decision tree for the same exam dataset. The tree learns super-specific rules like:

"If hours studied = 12.5 and attendance = 74%, then Pass"

It perfectly predicts all training students (100% accuracy), but fails badly when predicting for new students, because it memorized details instead of generalizing. *Result:* Model overfits.

Both models are bad for generalization. Good models balance complexity → not too simple, not too complex.

**Balanced Model (Just Right)**

It uses enough features to capture the real pattern.

Example rule:

"If hours studied > 5 and attendance > 60% → Pass"

This generalizes well to new students.

Note:

1. *Underfitting:* Like a student who barely studies → fails in both practice tests and final exams.
2. *Overfitting:* Like a student who memorizes previous year's question papers word-for-word → scores perfect in practice but fails when new questions appear.
3. *Balanced Learning:* A student who studies concepts and practices → performs well in both practice and final exams.

**Cross-Validation (K-Fold)**

Instead of splitting data once into train/test, K-Fold Cross-Validation divides the dataset into *k* equal parts.

**Steps:**

1. Train the model on *k-1* folds.
2. Test it on the remaining fold.
3. Repeat *k* times (each fold serves as test once).
4. Average performance across folds.

Advantage: More reliable evaluation, since all data is used for both training and testing. Common choice is k = 5 or 10.

**Hyperparameter Tuning**

Models have hyperparameters. These settings are chosen before training, such as, learning rate, number of neighbors in KNN and max depth in Decision Trees. We need to find the best combination.

1. Grid Search
    (i) Try all possible combinations of hyperparameters from a predefined set.
    (ii) Exhaustive but can be slow.
2. Random Search
    (i) Randomly samples hyperparameters from a distribution.
    (ii) Faster, often finds near-optimal results with fewer trials.

It is often combined with cross-validation to evaluate performance robustly.

**Bias–Variance Tradeoff**

Bias is an error from overly simple models (underfitting). Variance is an error from overly complex models (overfitting).

**Tradeoff:**

1. High Bias → model ignores patterns.
2. High Variance → model memorizes patterns.
3. Goal = find balance, that is, just complex enough to generalize well.

**Train/Test Split using train_test_split()**

**In scikit-learn:**

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

1. test_size=0.2 → 20% data for testing, 80% for training.
2. random_state ensures reproducibility.
3. Can also use stratify=y to maintain class balance.

Train/Test split is the simplest way to evaluate, but can give misleading results on small datasets → that's why cross-validation is often preferred.

**Example: Predicting if a Fruit is an Apple or Orange**

Imagine you're building a machine learning model for a fruit shop. You want to classify fruits based on two features:

1. Weight (grams)
2. Color score (closer to 0 = green, closer to 1 = orange/red)

You want the model to predict:

- Apple (0) or
- Orange (1)

**Step 1. Train/Test Split**

You first split your fruit dataset:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

1. 80% fruits for training.
2. 20% fruits for testing.

**Step 2. Train a Classifier (say Logistic Regression)**

The model learns from training fruits.

**Step 3. Confusion Matrix**

On the test set, suppose predictions are:

| | **Predicted Orange** | **Predicted Apple** |
|---|---|---|
| **Actual Orange** | 40 (TP) | 10 (FN) |
| **Actual Apple** | 5 (FP) | 45 (TN) |

From this:

1. Accuracy = (TP+TN)/(Total) = (40+45)/100 = 85%
2. Precision (Orange) = 40/ (40+5) = 88.9%
3. Recall (Orange) = 40/ (40+10) = 80%

Interpretation: Our model is fairly good, but it sometimes misses oranges (false negatives).

**Step 4. Overfitting vs. Underfitting**

1. If we use a very deep decision tree, it might memorize training fruits (100% accuracy) but misclassify new fruits → Overfitting.
2. If we use a very simple linear rule like "If weight > 150g → Orange, else Apple", it may miss many cases → Underfitting.

**Step 5. Cross-Validation (K-Fold)**

Instead of one train/test split, we do 5-Fold CV.

(i) Data split into 5 parts.

(ii) Each part acts as test once.

(iii) Average accuracy is more reliable.

**Step 6. Hyperparameter Tuning**

(i) With Grid Search, we test all possible settings of the decision tree such as, max depth = 2, 3, 4 ….

(ii) With Random Search, we sample a few random depths.

Best setting might be max_depth=3, balancing accuracy and generalization.

**Step 7. Bias–Variance Tradeoff**

(i) High Bias (underfit): Predicts all fruits as "apple" → always wrong for oranges.

(ii) High Variance (overfit): Model makes overly complex rules like "If weight=152.5g and color=0.74 → Orange", which fails on new fruits.

(iii) Balanced Model: Captures the true general rule: "Oranges are generally heavier and more orange-colored, apples lighter and greener."

This example naturally demonstrates accuracy, precision, recall, confusion matrix, over/underfitting, cross-validation, hyperparameter tuning, and bias–variance tradeoff.

**Python Code**

```
# Apple vs Orange Classification using Logistic Regression
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score,
recall_score, f1_score, ConfusionMatrixDisplay
# ----------------------------
# Step 1: Create synthetic dataset
# ----------------------------
# Features: [Weight (grams), Color Score (0=Green, 1=Orange/Red)]
# Labels: 0 = Apple, 1 = Orange
np.random.seed(42)
# Apples: lighter, greener
apple_weight = np.random.normal(150, 15, 50)     # mean=150g
apple_color  = np.random.normal(0.3, 0.05, 50)   # closer to green
apples = np.column_stack((apple_weight, apple_color))
apple_labels = np.zeros(50)
# Oranges: heavier, more orange
orange_weight = np.random.normal(200, 20, 50)    # mean=200g
orange_color  = np.random.normal(0.8, 0.05, 50)  # closer to orange
oranges = np.column_stack((orange_weight, orange_color))
orange_labels = np.ones(50)
# Combine dataset
X = np.vstack((apples, oranges))
y = np.hstack((apple_labels, orange_labels))
# ----------------------------
# Step 2: Train/Test Split
# ----------------------------
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
# ----------------------------
# Step 3: Train Logistic Regression Model
# ----------------------------
model = LogisticRegression()
model.fit(X_train, y_train)
```

```
# ----------------------------
# Step 4: Predictions and Evaluation
# ----------------------------
y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1-Score:", f1_score(y_test, y_pred))
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Apple",
"Orange"])
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix - Apple vs Orange")
plt.show()
# ----------------------------
# Step 5: Decision Boundary Visualization
# ----------------------------
# Create meshgrid for plotting boundary
x_min, x_max = X[:, 0].min() - 10, X[:, 0].max() + 10
y_min, y_max = X[:, 1].min() - 0.1, X[:, 1].max() + 0.1
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 200),
                     np.linspace(y_min, y_max, 200))

Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, alpha=0.3, cmap=plt.cm.coolwarm)
plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors="k", cmap=plt.cm.coolwarm)
plt.xlabel("Weight (grams)")
plt.ylabel("Color Score")
plt.title("Apple vs Orange Classification (Decision Boundary)")
plt.show()
```

**Output:** Fig 3.4.2 shows results**.**

Sample output for the classification problem:

Accuracy: 0.95

Precision: 0.889

Recall: 1.0

F1-Score: 0.941

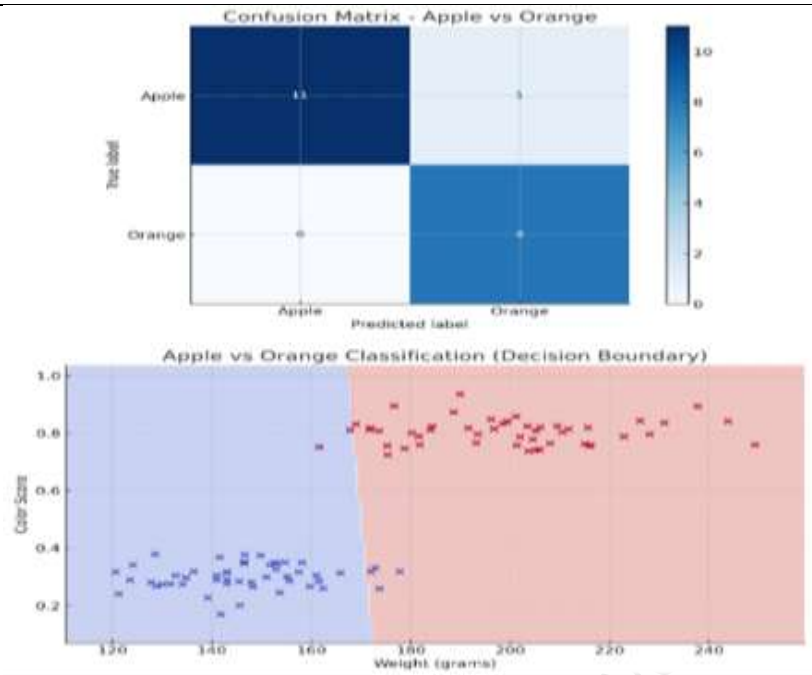



**Fig 4.2: Results**

The plots show a confusion matrix heatmap (blue shades), a decision boundary visualization where Apples and Oranges are separated based on weight and color score.

## Summary

- Accuracy, Precision, Recall, Confusion Matrix are core evaluation metrics.
- Overfitting vs. Underfitting is used for choosing the right model complexity.
- Cross-Validation is used for reliable performance estimation.
- Hyperparameter Tuning can be Grid Search (exhaustive) or Random Search (faster).
- Bias–Variance Tradeoff is the balancing act of ML.
- Train/Test Split is used for basic evaluation and it is foundation for cross-validation.

## Check Your Progress

**A. Multiple Choice Questions (MCQs)**

1. Which of the following is NOT a model evaluation metric? (a) Accuracy (b) Precision (c) Recall (d) Regression Line
2. A confusion matrix is used to: (a) Train the model (b) Test the speed of algorithms (c) Show actual vs predicted results (d) Store raw data
3. Which problem occurs when a model performs well on training data but poorly on test data? (a) Underfitting (b) Overfitting (c) Clustering (d) Normalization
4. Recall is also known as: (a) Accuracy (b) Specificity (c) Sensitivity (d) Probability
5. Cross-validation is mainly used to: (a) Reduce overfitting (b) Increase dataset size (c) Replace missing values (d) Simplify algorithms

**B. Fill in the Blanks**

1. _____ is the proportion of correct predictions out of all predictions.
2. A _____ matrix is used to evaluate classification results.
3. Precision measures how many of the predicted positives are actually _____.
4. Overfitting happens when a model learns the _____ in training data.
5. _____ validation helps to test model performance on multiple subsets of data.

**C. True or False**

1. Accuracy is always the best metric for imbalanced datasets.
2. Recall measures how many actual positives were correctly identified.
3. Overfitting occurs when the model is too simple.
4. Underfitting means the model fails to capture patterns in data.
5. Cross-validation helps in building more reliable models.

**D. Short Answer Questions**

1. Define accuracy and explain its limitation.
2. Differentiate between Precision and Recall with an example.
3. What is a confusion matrix and why is it important?
4. Explain overfitting with a real-life analogy.
5. How does cross-validation improve model evaluation?



## Session 5 Ethics, Bias and Myths in AI

Imagine a big company develops an AI-powered recruitment system that automatically shortlists candidates for job interviews.

The AI looks at resumes, social media profiles, and even video interviews. Many candidates were rejected without giving reasons for it. Students started complaining about AI system. Later on, it was found that the system is not following ethics and is taking bias decisions. Fig 5.1 shows AI powered recruitment system.

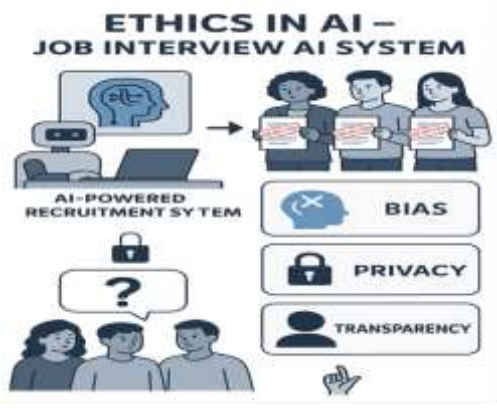



Fig 5.1: AI Powered Recruitment System

In this session, you will understand the importance of ethics, fairness, and transparency in AI**,** learn about causes and impacts of bias in data**,** explore concepts of privacy and data security**,** and clear common myths about AI**.** You will also realize why clean data, explainability, and fairness are essential for building trustworthy AI systems.

**Ethics in AI**

As AI becomes more powerful, ethical issues become more important. Ethics in AI refers to the set of principles that guide the development and use of AI systems.

**Key Ethical Concerns:**

1. ***Fairness:*** AI should not discriminate based on race, gender, or other personal attributes.
2. ***Privacy:*** AI systems must protect personal data and avoid unauthorized surveillance
3. ***Accountability:*** It should be clear who is responsible when AI makes a mistake.
4. ***Transparency:*** Users should understand how AI makes decisions.
5. ***Human Control:*** AI should not operate without human oversight in sensitive areas like healthcare or defense.

Ethical AI is necessary to ensure that technology serves all people fairly and safely. Figure 5.2 shows ethical AI framework.


Fig 5.2: Ethical AI Framework

**Bias in AI**

Bias in AI happens when an AI system makes unfair or incorrect decisions because of biased data or programming. This can lead to discrimination and inequality in AI-based decisions

**Causes of AI Bias:**

1. Training Data Bias: If the data used to train the AI contains biases, the AI will learn those biases.
2. Human Bias: If developers unknowingly introduce their own biases into the system.
3. Sampling Bias: Using unbalanced datasets that don't represent all groups equally. Figure 5.3 shows prevention of AI bias.

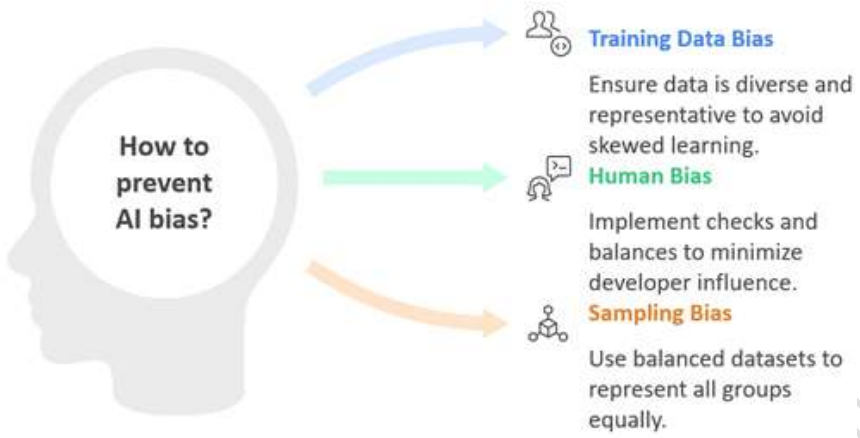



**Fig 3.5.3: Prevention of AI bias**

Machine Learning models learn from the data they are trained on. If the data is biased, that is, skewed towards a particular group, opinion, or situation, the AI system will also give biased results.

**Example**:

If a hiring AI is trained mostly on resumes of men, it may learn that "being male" is associated with selection, leading to unfair rejection of women.

**Impact:**

1. Discrimination against groups such as, gender, caste, region, race.
2. Wrong predictions in sensitive areas like healthcare, finance, or recruitment.

**Solution:**

Use diverse, balanced datasets and test AI for fairness before deploying.

**Real-World Examples:**

(i) Facial Recognition: Often less accurate for darker-skinned individuals.

(ii) Hiring Tools: Some AI tools favored male resumes over female ones due to biased data.

(iii) Loan Approval: AI systems denying loans based on zip codes associated with certain races.

To reduce bias, developers must use diverse datasets and regularly test AI for fairness.

**Practical Activity on Bias**

**Bias in Data – Resume Sorting Activity**

1. Setup:
   (i) Prepare 10–15 fake resumes of candidates.
   (ii) Add subtle bias (e.g., more male candidates in engineering roles, fewer female candidates).
2. Activity:
   (i) Ask one group of students to train a simple "selection rule" (choose based on past data).
   (ii) Another group must select based only on skills.

3. Learning Outcome:
   (i) Students see how biased data can lead to unfair outcomes, even if the rule seems logical.

**Myths in AI**

**Myth 1: AI will take over all jobs.**

Reality: AI will automate some repetitive tasks but also create new job opportModuleies in AI development, data science, ethics, and more.

Humans will still be needed for creativity, emotional intelligence, and complex decision-making.

**Myth 2: AI can think like a human.**

Reality: AI can simulate human-like behavior, but it doesn't "think" or "feel" like humans.

It does not have emotions, self-awareness, or consciousness. It follows patterns based on data. Figure 5.4 shows myths in AI.

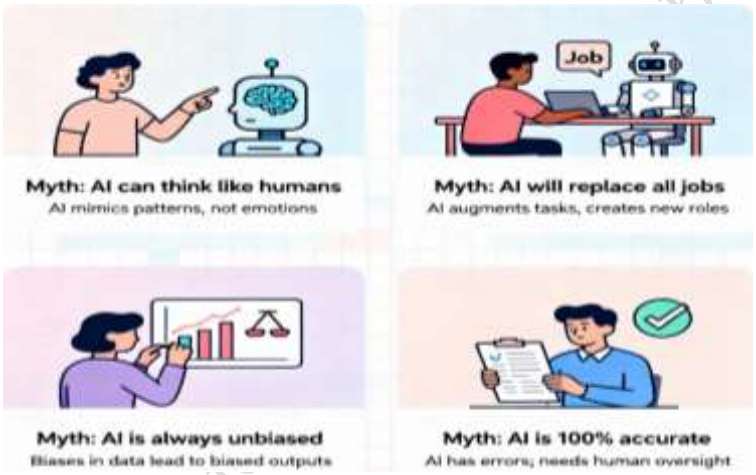
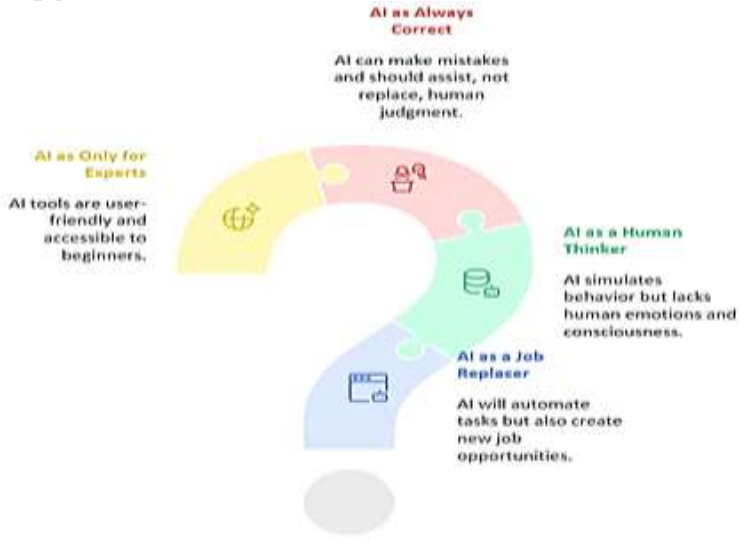



Fig. 5.4: Myths in AI

**Myth 3: AI is always correct.**

Reality: AI systems can make mistakes, especially if the input data is poor or biased. AI should be used as a tool to assist humans, not replace human judgment completely.



**Myth 4: AI is only for tech experts.**

Reality: Many AI tools today are user-friendly and don't require programming knowledge. Platforms like Teachable Machine and Scratch AI extensions allow beginners and students to experiment with AI. Understanding, what AI is and what it is not, is important to avoid fear and use it responsibly.

**Importance of Clean Data**

Garbage in is Garbage out. If the data contains errors, duplicates, missing values, or irrelevant features, the model's accuracy and reliability will suffer.

**Example:**

A student marks dataset with spelling mistakes in names such as, "Prakash", "Prakesh", "Prkash" will confuse the model, treating them as different people.

**Impact:**

(i) It reduces accuracy.

(ii) It misleads predictions.

**Solution:**

Perform data cleaning, remove duplicates, handle missing values, and standardize data before training. Figure 5.5 shows importance of clean data.

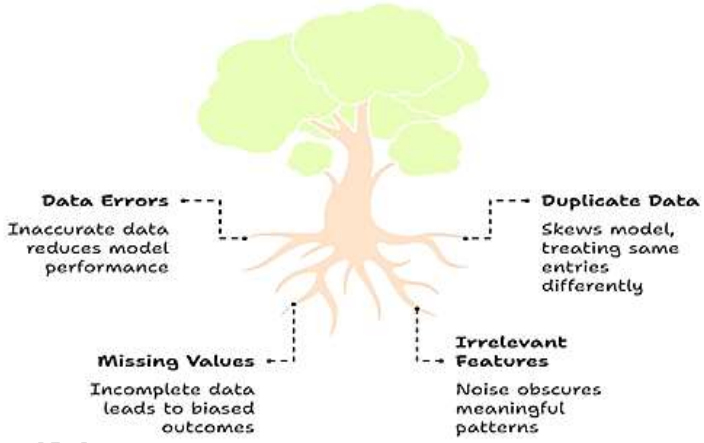



Fig. 5.5: Importance of clean data

**Practical Activity on Clean Data**

Clean Data – Messy Student Marks Dataset

1. Setup:
   (i) Provide a dataset of student marks with:
      (a) Missing values
      (b) Duplicate entries
      (c) Wrong spellings in names
2. Activity:
   (i) Students must clean the dataset. Remove duplicates, fill missing marks with average and correct names.
   (ii) Compare results before and after cleaning.
3. Learning Outcome:

(i) Understand how dirty data reduces accuracy and why clean data is necessary.

**Transparency and Explainability**

AI systems should not act like a black box. Users should understand why and how a decision was made.

**Example:**

A student is rejected by AI in a scholarship application but is not given any reason. Without explainability, trust in AI decreases.

**Impact:**

(i) Users lose confidence in AI.

(ii) Hard to detect and correct mistakes.

**Solution:**


Fig 3.5.6: Transparency in AI

Use explainable AI (XAI) methods to show reasons behind decisions such as, “Rejected due to low GPA, not personal details”. Fig 3.5.6 shows transparency in AI.

**Practical Activity**

**Transparency and Explainability – Black Box Game**

1. Setup:

(i) Teacher secretly creates a “rule” to select students such as, choose students wearing glasses, or sitting in even-numbered chairs.

2. Activity:

(i) Students guess why certain classmates are “selected” or “rejected.”

(ii) Then teacher reveals the hidden rule.

3. Learning Outcome:

(i) Students realize the confusion and unfairness when AI systems don’t explain their decisions.

**Privacy and Data Security**

AI systems often use sensitive personal data such as, age, location, health records, financial data. It is critical to protect this from misuse, leaks, or unauthorized access.

**Example:**

A health AI collects patient data. If not secured, hackers may steal it and misuse it.

**Impact:**

(i) Identity theft.

(ii) Loss of trust in AI systems.

**Solution:**

(i) Apply data encryption, strict access control, and anonymization.

(ii) Collect only data that is necessary. Use data Minimization principle.

Figure 5.7 shows AI privacy and security risks.

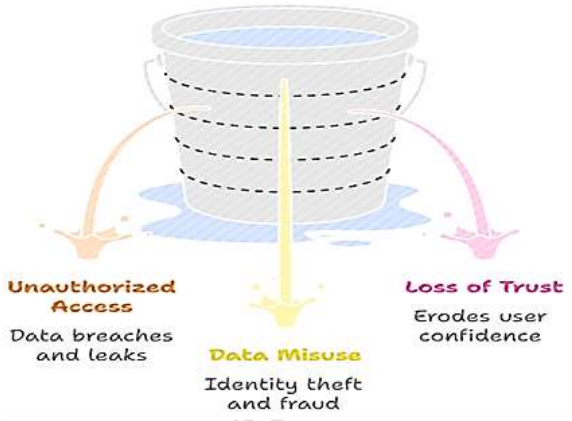



Fig. 5.7: AI Privacy and Security risks

**Data Minimization Principle**

The Data Minimization Principle is one of the core principles of data protection and privacy especially emphasized in laws like the GDPR – General Data Protection Regulation.

The principle of data minimization states that:

An organization should collect and process only the personal data that is adequate, relevant, and limited to what is necessary for the specific purpose for which it is collected.

**Key Points**

1. Adequate – The data collected should be sufficient to achieve the intended purpose. Example: Asking for an email address for account login is adequate; asking for home address in this case may not be.
2. Relevant – The data should be directly related to the purpose. Example: An online bookstore may need your delivery address, but it doesn't need your medical history.
3. Limited (to necessity) – No excessive or unnecessary data should be collected. Example: A job application may require your qualifications and experience, but not your religious beliefs.

**Importance of Data Minimization Principle**

1. Protects privacy – Reduces the amount of personal information exposed.

2. Reduces risks – Less data means less chance of misuse, breaches, or leaks.
3. Builds trust – Users feel safer when they know only essential data is collected.
4. Legal compliance – Required by GDPR, Indian DPDP Act (2023), and other data protection laws.

**Examples in Practice**

1. A fitness app should only collect data about exercise and health metrics, not unrelated personal details.
2. An e-commerce site may need your payment details and delivery address, but not your family background.
3. A school form may need student's name, age, and guardian contact details, but not parents' income unless relevant to scholarships.

**Practical Activity**

**Privacy and Data Security – Social Media Example**

1. Setup:
   (i) Give students a short description of a person such as, Name, Age, Hobbies.
   (ii) Add some extra personal details such as, phone number, address.
2. Activity:
   (i) Discuss: "Should AI systems collect this information without consent?"
   (ii) Ask students: "Which data is okay to use, and which is private?"
3. Learning Outcome:
   (i) Understand importance of data privacy and the dangers of oversharing.

**Fairness and Transparency**

AI should treat all individuals fairly, without favoring or discriminating against specific groups. Transparency ensures people understand the rules used by AI.

**Example:**

An AI-based loan approval system must ensure fairness: people with similar financial profiles should be treated equally, regardless of gender or region.

**Impact:**

(i) Prevents social discrimination.
(ii) Builds trust among users.

**Solution:**

(i) Perform fairness checks.
(ii) Ensure equal opportModuleies in outcomes.
(iii) Make AI decisions auditable. Figure 5.8 shows fairness and transparency in AI.


Fig. 5.8: Fairness and Transparency in AI

**Practical Activity**

**Fairness and Transparency – Loan Approval Simulation**

1. Setup:
   (i) Give a dataset of 6–8 people applying for loans with attributes such as, Income, Job Type, Gender, Credit Score.
   (ii) Create one version with a hidden bias rule such as, rejecting applicants from a specific job type.
2. Activity:
   (i) Students must evaluate whether the loan approvals are fair.
   (ii) Discuss what makes a system "transparent" and "fair."
3. Learning Outcome:
   (i) Students experience how unfair hidden rules affect people and why fairness checks matter.

## Summary

- Bias of Data may result into wrong/unfair outcomes.
- Clean Data will give accurate and reliable results.
- Transparency and Explainability will builds trust.
- Privacy and Security will protect users' rights.
- Fairness and Transparency ensures equality in AI decisions.

## Check Your Progress

**A. Multiple Choice Questions (MCQs)**

1. Bias in AI occurs due to: (a) Balanced training data (b) Incorrect or skewed data (c) Transparent algorithms (d) High accuracy

2. Which of the following is an ethical concern in AI? (a) Privacy and data protection (b) Accurate predictions (c) Faster computation (d) Use of neural networks
3. Which of these is a myth about AI? (a) AI can automate repetitive tasks (b) AI will completely replace all human jobs (c) AI helps in healthcare and finance (d) AI systems require training data
4. Why is transparency important in AI systems? (a) To reduce bias and build trust (b) To slow down computations (c) To increase costs (d) To hide decision-making
5. Which of the following best represents responsible use of AI? (a) Ignoring fairness (b) Protecting privacy (c) Using biased data (d) Misusing AI for unethical purposes

**B. Fill in the Blanks**

1. _____ in AI means unfair treatment of groups due to biased training data.
2. Protecting personal information is related to _____.
3. A common myth is that AI will _____ all human jobs.
4. _____ and transparency are key principles of ethical AI.
5. An AI system trained on biased data will produce _____ results.

**C. True or False**

1. Bias in AI only comes from algorithms, not data.
2. Ethics in AI ensures fairness and trust.
3. AI should be designed with transparency.
4. Myths about AI can lead to fear and misunderstanding.
5. AI decisions should always be free from human oversight.

**D. Short Answer Questions**

1. What is bias in AI? Give an example.
2. Why is ethics important in AI development?
3. Name one common myth about AI and explain why it's incorrect.
4. How can AI systems ensure fairness and transparency?
5. What role does privacy play in AI applications?

# Module 4. Neural Network

## Module Overview

Think about how your brain works in daily life. When you see your friend from far away, you instantly recognize them — not by memorizing every detail, but by noticing patterns like their face, walk, or voice. Similarly, when you ask your smartphone assistant to "set an alarm for 6 AM," it understands your speech, processes it, and sets the reminder. Behind these smart tasks lies the power of neural networks**,** which are computer systems inspired by the way our brain works.

Every day, we unknowingly interact with systems powered by neural networks**.** When your phone unlocks using face recognition**,** when Gmail automatically sends junk mails to the spam folder**,** when Google Maps suggests a faster route, or when you ask Alexa or Siri to play a song — all these are possible because machines have learned to recognize patterns, much like our own brains.

By the end of this Module, you will not only understand the theory behind neural networks but also gain hands-on experience in building and training models using Python. This journey will help you connect classroom learning with real-world AI applications — from healthcare to navigation, from social media to smart assistants.

## Learning Outcome

After completing this module, you will be able to:

- Understand the biological inspiration behind Artificial Neural Networks (ANN).
- Define the architecture components: Input, Hidden, and Output layers.
- Explain the mathematical model of a single neuron (Weight, Bias, and Summation).
- Compare common Activation Functions like Sigmoid, ReLU, and Tanh to introduce non-linearity.
- Identify real-world use cases such as Image Recognition, Natural Language Processing (NLP), and Predictive Analytics.
- Distinguish when to use ANNs over traditional Machine Learning models.
- Gain hands-on familiarity with TensorFlow, Keras, and PyTorch.
- Learn to set up a deep learning environment and manage tensors.
- Understand Backpropagation and the role of the Chain Rule in updating weights.
- Implement Optimizers (e.g., Adam, SGD) and Loss Functions to improve model accuracy.
- Develop a complete end-to-end classification model using Python.
- Evaluate performance using Accuracy and Loss curves to detect overfitting.

## Module Structure

Session 1. Introduction to Neural Networks

Session 2. Neurons and Activation Functions

Session 3. Applications of Neural Networks

Session 4. Python Libraries for Neural Networks

Session 5. Training Neural Networks

Session 6. Building a Basic Neural Network for Classification

## Session 1. Introduction to Neural Networks

We can think of a neural network as similar to how our brain recognizes patterns. Just like when you see a friend from far away and identify them by their face, height, or way of walking, computers use neural networks to identify patterns in data. For example, a smartphone uses neural networks to recognize your face and unlock itself.

In this session, you will understand what neural networks are, how they are inspired by biological neurons, how they process data through layers, and the concepts of feedforward propagation, backpropagation, weights, and bias.

### 1.1 Neural Network

A Neural Network is a computational model inspired by how the human brain works. Just as our brain processes information through a network of biological neurons, an artificial neural network (ANN) processes data through a system of interconnected artificial neurons or nodes.

In essence, Neural Networks are designed to recognize patterns. Whether it is identifying faces in a photograph, classifying email as spam or not, or predicting weather conditions, Neural Networks are used to make sense of complex data.

The core idea is *"learning from data"*. A Neural Network doesn't follow a fixed set of instructions. Instead, it learns from examples by identifying patterns and relationships in data.

### 1.2 Applications of Neural Networks

Neural Networks are now part of everyday technology and industries. Some of the real life applicatons of Neural Networks are depicted in Figure 1.1.

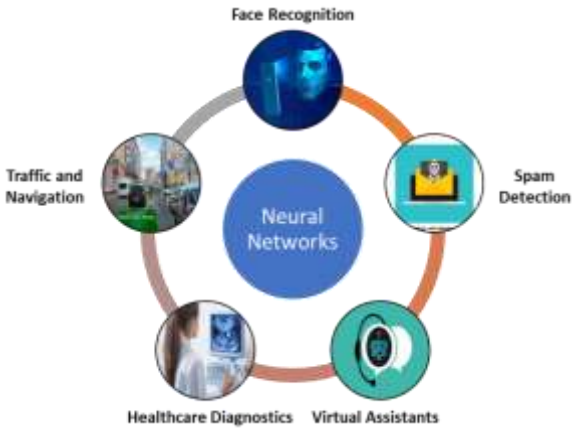



Fig. 1.1: Real-Life Applications of Neural Networks

### 1.2.1 Face Recognition

Smartphones unlock using your face. Behind the scenes, a Neural Network scans your facial features and compares them to stored data. These models are trained to differentiate faces under varying lighting, angles, or expressions.

### 1.2.2 Spam Detection

Email services like Gmail use Neural Networks to scan messages. They learn from thousands of spam and non-spam emails to automatically filter junk mail out of your inbox.

### 1.2.3 Virtual Assistants

Tools like Siri, Alexa, and Google Assistant understand voice commands using deep neural networks. They convert your speech into text, process the meaning, and respond accordingly.

### 1.2.4 Healthcare Diagnostics

Doctors use AI-based tools trained with Neural Networks to read X-rays or MRIs. The network identifies patterns of diseases like tumors or fractures with high accuracy.

### 1.2.5 Traffic and Navigation

Apps like Google Maps use trained networks to analyze traffic, recommend faster routes, and estimate travel time using real-time data.

These applications demonstrate how Neural Networks support decision-making and automation across fields.

## 1.3 Biological vs Artificial Neurons

Let's understand where the idea of Neural Networks came from.

### 1.3.1 Biological Neurons

A biological neuron is a nerve cell that transmits information in the brain. It has three main parts as shown in Figure 1.2.

***Dendrites****:* Receive signals from other neurons.

***Cell Body (Soma)****:* Processes the signal.

***Axon****:* Sends the processed signal to the next neuron.

Each neuron connects to thousands of others, forming a large network. This is how the brain performs complex actions like remembering, walking, or speaking.

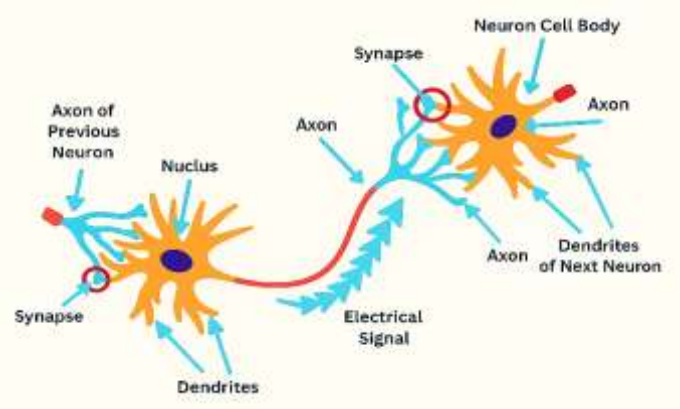



Fig. 1.2: Biological Neuron

### 1.3.2 Artificial Neurons

An Artificial Neuron (also called a node) mimics this structure as shown in Figure 1.3.

- Receives inputs (numbers).
- Multiplies each input by a weight.
- Sums them up and applies an activation function to decide the output.
- Sends the output to the next layer.

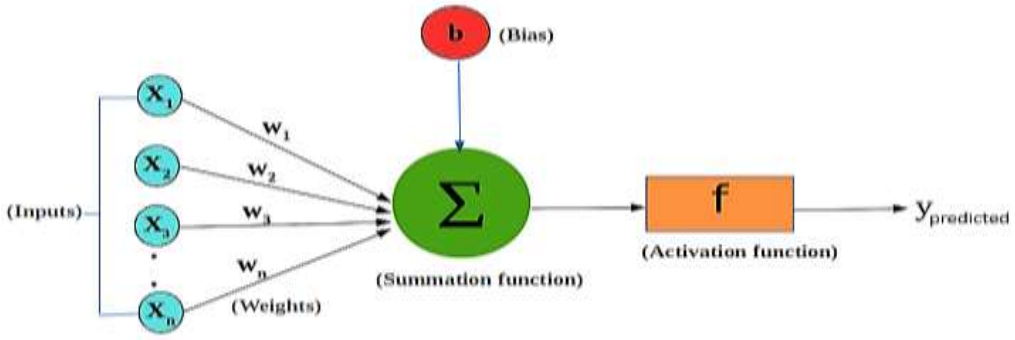



Fig. 1.3: Artificial Neuron

The comparison of Bilological Neural and Artificial Neuron is given in Table 1.1.

Table 1.1: Bilological Neural and Artificial Neuron

| **Biological Neuron** | **Artificial Neuron** |
|---|---|
| Dendrites → input signals | Inputs (numerical values) |
| Soma → processing | Weighted sum + bias |
| Axon → output signals | Output after activation |

## 1.4 Working of a Neural Network

The functioning of a Neural Network can be understood through several key components and steps.

### 1.4.1 Structure of a Neural Network

A basic Neural Network has three types of layers:

- ***Input Layer****:* Receives data (e.g., pixels from an image).
- ***Hidden Layer(s)****:* Processes data using weights and activation functions.
- ***Output Layer****:* Produces the result (e.g., yes or no, class labels).

The hidden layers are where learning happens. They are called "hidden" because we don't directly interact with them. Figure 1.4 shows that each layer has neurons (also called nodes) connected by **weights**, which determine the importance of the input.

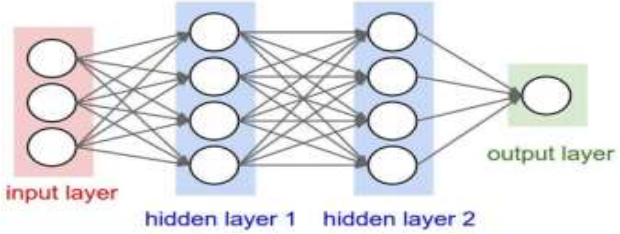



Fig. 1.4: Structure of a Neural Network

### 1.4.2 Feedforward Propagation

Data moves in one direction: from input to output.

- Each input value is multiplied by its corresponding weight.
- The weighted values are summed.
- A bias is added to shift the result.
- An activation function (e.g., ReLU or Sigmoid) decides the output.

This process is repeated for every neuron until the final output is generated.

### 1.4.3 Backpropagation

After generating the output, the Neural Network compares it to the correct answer using a loss function. If there's an error, it goes back and adjusts the weights.

This method is called Backpropagation. Figure 1.5 shows, how the backpropagation works:

- It finds the error.
- It changes the weights slightly to reduce the error.
- This is repeated many times during training until the error is minimal.

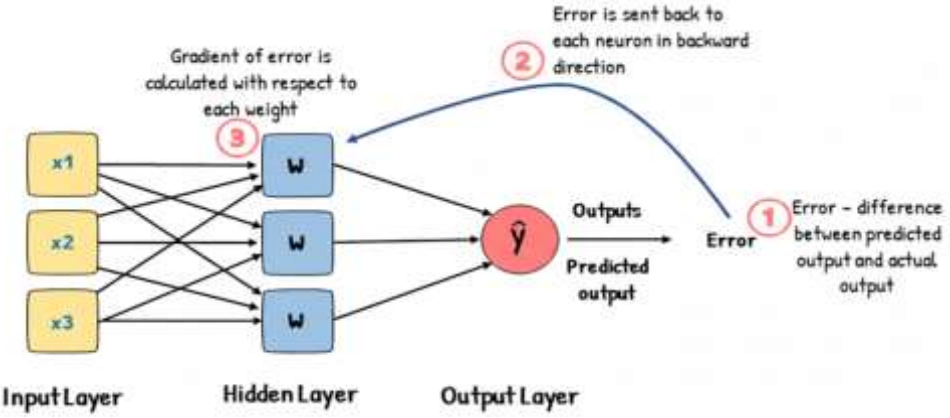



Fig. 1.5: Backpropagation

### 1.4.4 Importance of Weights and Bias

- **Weights**: Show how important an input is.
- **Bias**: Helps shift the result to fit the data better.

Small changes in weights can lead to big changes in predictions. Training a network is essentially finding the right weight values to make accurate predictions.

## 1.5 Practical Activities

### 1.5.1. Activity: Draw a Diagram Comparing Brain and Artificial Neurons

- Use a simple side-by-side diagram on paper.
- Label key parts: dendrites, soma, axon for biological neuron; inputs, weights, activation for artificial neuron.

### 1.5.2. Activity: Draw a Neural Network with One Hidden Layer

- On paper or drawing tool, draw:
  - 4 input neurons

- 1 hidden layer with 3 neurons
- 1 output neuron

- Connect all layers with arrows and label sample weights (e.g., w1, w2).

### 1.5.3. Activity: Weight Change and Prediction (Python or Visual Tool)

If you have a computer, try this basic Python example:

```
# Install required library first (if needed)
# pip install matplotlib

import matplotlib.pyplot as plt

# Simple simulation: Input × Weight = Output
inputs = [1, 2, 3, 4]
weights = [0.2, 0.5, 0.8, 1.0]
outputs = [i * w for i, w in zip(inputs, weights)]

plt.plot(inputs, outputs, marker='o')
plt.title("Effect of Weight on Output")
plt.xlabel("Input")
plt.ylabel("Output")
plt.grid(True)
plt.show()
```

**What You Learn:** As weight increases, the output changes. The line becomes steeper.

**Alternate Tool:** Use browser-based tools like Teachable Machine by Google to visually train a basic classifier (no code required).

## Summary

- Neural Networks are inspired by the human brain and help computers learn from data.
- They are made of neurons arranged in layers—input, hidden, and output.
- Each neuron processes data using weights, bias, and activation functions.
- They are used in many areas: healthcare, finance, education, security, and entertainment.
- Training involves adjusting weights through a process called backpropagation.
- Simple activities like diagram drawing or Python visualization help understand the role of weights and predictions.

## Check Your Progress

### A. Multiple Choice Questions

1. What is the main function of a Neural Network? (a) Performing arithmetic (b)

Recognizing patterns in data (c) Storing images (d) Sending messages

2. In a biological neuron, which part receives signals? (a) Axon (b) Soma (c) Dendrites (d) Output layer
3. Which of the following is an activation function? (a) Matrix (b) Sigmoid (c) Gradient (d) Backpropagation
4. The output of a neuron is calculated using: (a) Only inputs (b) Inputs + bias (c) Weighted sum of inputs + bias (d) None of the above
5. Which of these is a real-life application of Neural Networks? (a) Typing in WordPad (b) Counting files in a folder (c) Face recognition in phones (d) Installing software

**B. Fill in the Blanks**

1. Artificial Neural Networks are inspired by the _____.
2. The first layer in a Neural Network is called the _____ layer.
3. Each connection between neurons has a value called _____.
4. The process of updating weights to reduce error is known as _____.
5. An example of an activation function is _____.

**C. State Whether True or False**

1. Neural Networks cannot learn from data.
2. The hidden layer is directly visible to users.
3. Spam detection is a use case of Neural Networks.
4. All neurons in a network are connected to the output layer.
5. Changing weights affects predictions made by the model.

**D. Short Answer Questions**

1. What are the main parts of a biological neuron?
2. Define an Artificial Neuron and its components.
3. How does the activation function help in a Neural Network?
4. Name three real-life applications of Neural Networks.
5. What is the difference between feedforward and backpropagation?

## Session 2. Neurons and Activation Functions

Imagine a factory assembly line: each worker adds something to the product, and finally, a finished item comes out. Similarly, in a neural network, each neuron processes inputs and passes results forward. But just like machines may decide to turn ON or OFF depending on the situation, neurons use activation functions to decide whether to send a signal or not.

In this session, you will understand the structure of a neuron, the role of weights and bias, different types of activation functions (Sigmoid, ReLU, Softmax), and types of neural networks like Feedforward, CNN, and RNN.

### 2.1 Structure of a Neural Network

A Neural Network is a system inspired by how the human brain processes information. Just as the brain consists of billions of interconnected neurons, a neural network is made of artificial neurons arranged in layers.

Each neuron in a neural network performs simple mathematical operations. These neurons are connected by links called weights. A neural network typically consists of:

- **Input Layer** – where the data enters the network.
- **Hidden Layers** – where data is processed and patterns are learned.
- **Output Layer** – which gives the final prediction or result.

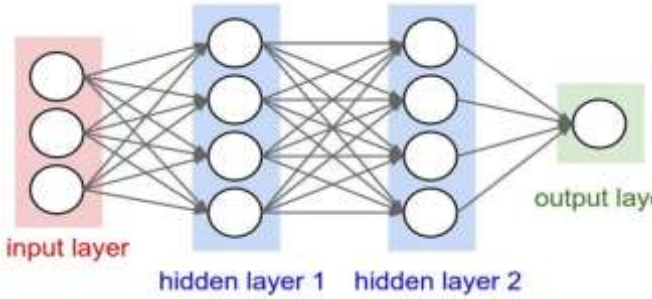



Fig. 2.1: Structure of a Neural Network

### 2.1.1 Real-Life Analogy

Imagine a neural network like a factory assembly line. Each machine (neuron) receives parts (inputs), processes them, and passes them to the next machine. At the end, the finished product (output) comes out. Similarly, in a neural network, data is processed layer by layer to reach the final prediction.

## 2.2 Layers of a Neural Network

### 2.2.1 Input Layer

This is the first layer. It receives the initial data. For example, if we are building a model to recognize handwritten digits, each pixel of the image is an input.

### 2.2.2 Hidden Layer(s)

These are intermediate layers between input and output. They do the real computation by transforming the input into something the output layer can use. A network can have one or many hidden layers.

### 2.2.3 Output Layer

This layer provides the final result. For classification problems, it often contains one node for each possible class.

## 2.3 Neurons, Weights, and Bias

### 2.3.1 Neuron (Node)

An artificial neuron takes several inputs, applies weights, adds a bias, and uses an activation function to produce an output.

**Mathematical Expression:**

$$Output = ActivationFunction(w_1x_1 + w_2x_2 + \ldots + w_nx_n + b)$$

Where:

- $x_1, x_2, \ldots, x_n$ = inputs

- $w_1, w_2, \ldots, w_n$ = weights
- $b$ = bias
- ActivationFunction = ReLU, Sigmoid, etc.

### 2.3.2 Weights

Each connection between neurons has a weight that determines how important that input is. A higher weight means more influence.

### 2.3.3 Bias

Bias is a constant value added to the weighted sum. It helps shift the activation function and makes the model more flexible.

## 2.4 Activation Functions

Activation functions help the neural network learn complex patterns. They decide whether a neuron should be "activated" or not. Without them, the neural network would behave like a linear function and couldn't learn non-linear relationships.

### 2.4.1 Sigmoid Function

The sigmoid function maps values between 0 and 1. It is used in binary classification problems.

$$f(x) = \frac{1}{1 + e^{-x}}$$

**Example:**

Let $x = 0$

$$f(0) = \frac{1}{1 + e^{0}} = \frac{1}{2} = 0.5$$

Let $x = 2$

$$f(2) = \frac{1}{1 + e^{-2}} \approx \frac{1}{1 + 0.135} \approx 0.88$$

Here, $e^2 \approx 7.389$, and it helps map the output between 0 and 1.

> **Note:** letter "e"
>
> In activation functions like Sigmoid and Softmax, the letter "e" represents the mathematical constant known as Euler's number.
>
> e ≈ 2.71828
>
> It is an irrational number used widely in mathematics, especially in exponential growth, logarithms, and probability.
>
> Activation functions like Sigmoid and Softmax use exponential functions to control the output range and make learning smooth and differentiable.

### 2.4.2 ReLU (Rectified Linear Unit)

It is the most used activation function in deep learning.

$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$

**Example:**

- $f(-3) = 0$

- $f(4) = 4$

It introduces non-linearity and helps models learn complex patterns faster.

### 2.4.3 Softmax Function

Used in multi-class classification problems.

$$Softmax(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

**Example:**

Let input vector be: [2.0, 1.0, 0.1]

$$e^x = [7.39, 2.71, 1.10] Softmax = [0.65, 0.24, 0.10]$$

Now the network predicts the most probable class (highest softmax value).

## 2.5 Types of Neural Networks

### 2.5.1 Feedforward Neural Network (FNN)

The simplest type. Data flows in one direction — from input to output. No loops or backward connections.

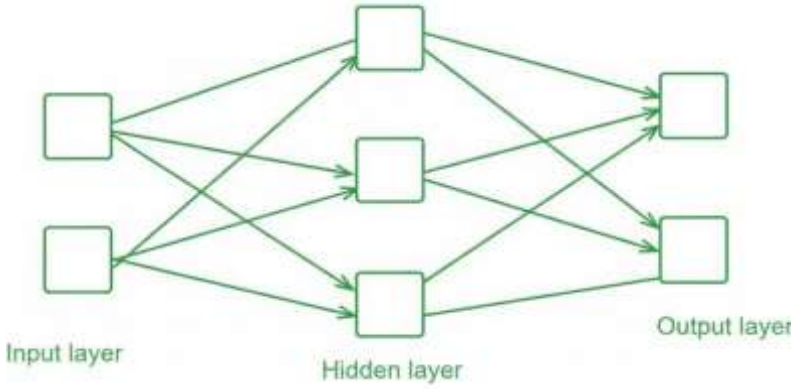



Fig. 2.2: Feedforward Neural Network

### 2.5.2 Convolutional Neural Network (CNN)

Used for image recognition. It detects edges, textures, and shapes from images.

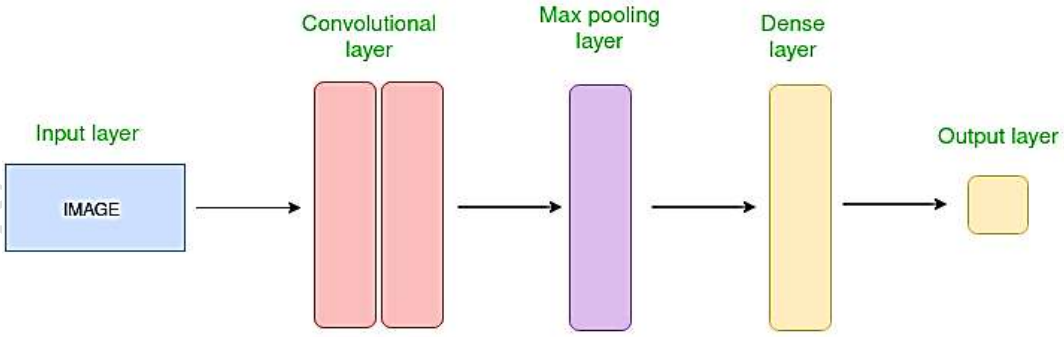



Fig. 2.3: Convolutional Neural Network

### 2.5.3 Recurrent Neural Network (RNN)

Used for sequential data like text, speech, and time series. It has loops that allow memory of past inputs.

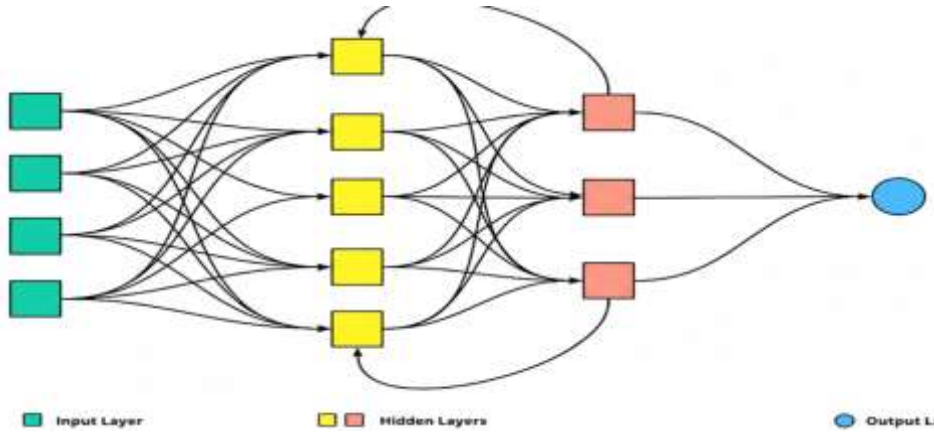



Fig. 2.4: Recurrent Neural Network

### 2.6 Practical Activity 1. Implementing a Simple Neuron in Python Using NumPy

**Objective:**

Understand how a neuron processes inputs using weights, bias, and an activation function.

```
import numpy as np
# Inputs
x = np.array([2, 3])        # Example inputs
w = np.array([0.4, 0.6])    # Weights
b = 0.5                     # Bias
# Compute weighted sum
z = np.dot(x, w) + b        # z = (2*0.4 + 3*0.6) + 0.5 = 2.9
# Apply ReLU
output = max(0, z)
print("Neuron output:", output)

output
Neuron output: 2.9
```

### 2.7 Practical Activity 2. Plotting Sigmoid and ReLU Functions Using Matplotlib

**Objective:**

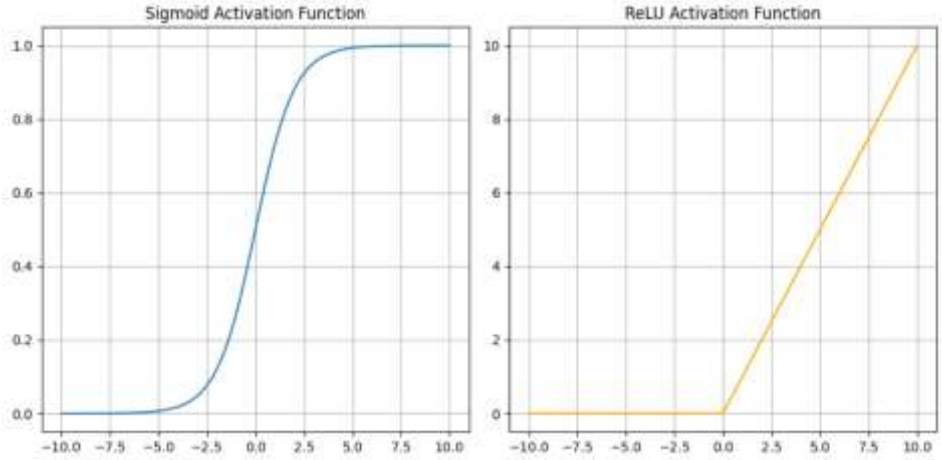
Visualize how activation functions behave.

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(-10, 10, 100)
# Sigmoid Function
sigmoid = 1 / (1 + np.exp(-x))
# ReLU Function
relu = np.maximum(0, x)
plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
```

```
plt.plot(x, sigmoid, label='Sigmoid')
plt.title("Sigmoid Activation Function")
plt.grid(True)
plt.subplot(1,2,2)
plt.plot(x, relu, label='ReLU', color='orange')
plt.title("ReLU Activation Function")
plt.grid(True)
plt.tight_layout()
plt.show()
```

This activity runs smoothly on low-end PCs using simple data.

**Output**






## Check Your Progress

### A. Multiple Choice Questions

1. Which layer receives the initial data in a neural network? (a) Hidden Layer (b) Input Layer (c) Output Layer (d) Convolutional Layer
2. Which activation function is most commonly used in deep learning? (a) Tanh (b) ReLU (c) Sigmoid (d) Linear
3. In the softmax function, the output values represent: (a) Weighted sums (b) Probabilities of each class (c) Random numbers (d) Errors
4. Which type of neural network is best suited for text and sequential data? (a) CNN (b) FNN (c) RNN (d) GAN
5. What role do weights play in a neuron? (a) They adjust the learning rate (b) They determine the importance of inputs (c) They act as output functions (d) They replace activation functions

### B. Fill in the Blanks

1. The _____ function is used in binary classification to map values between 0 and 1.
2. In a neural network, _____ are the connections that determine input

importance.

3. A _____ Neural Network is the simplest type with data flowing only forward.
4. The constant added after weighted sum in a neuron is called _____.
5. The function ReLU introduces _____ to the model, allowing it to learn complex patterns.

**C. State Whether True or False**

1. Without activation functions, neural networks behave like linear models.
2. A feedforward neural network allows loops between its layers.
3. Softmax is mainly used for binary classification.
4. ReLU outputs the same value for positive inputs and 0 for negative inputs.
5. Bias improves the flexibility of a neuron by shifting the activation function.

**D. Short Answer Questions**

1. What is the role of activation functions in a neural network?
2. Explain why softmax is suitable for multi-class classification.
3. Give an example where RNNs are more useful than CNNs.
4. What happens if we remove bias from neurons in a neural network?
5. Write the mathematical formula of a sigmoid function and explain its output range.



## Session 3. Applications of Neural Networks

Think of all the ways AI touches your daily life — when Google Maps suggests a faster route, when Gmail filters out spam, or when Siri converts your speech into text. These are direct applications of neural networks.

In this session, you will understand the real-world applications of neural networks such as object detection, image segmentation, classification, speech-to-text conversion, and machine translation, along with simple Scratch activities that simulate these applications.

### 3.1 Introduction to Applications of Neural Networks

Artificial Neural Networks (ANNs) are designed to mimic the way the human brain processes information. With advancements in computational power and algorithms, ANNs are now widely used in various real-world applications. These applications often include visual perception, language understanding, voice recognition, and intelligent decision-making.

The main advantage of ANNs lies in their ability to learn patterns from data. Once trained, they can make decisions or predictions even with new or unseen data.

### 3.2 Object Detection using ANN

Object detection is a computer vision technique that identifies and locates objects in an image or video. It answers two main questions:

- What is the object?
- Where is the object located?

Artificial neural networks, especially Convolutional Neural Networks (CNNs), are highly effective for object detection. The ANN learns patterns like shapes, colors, and textures to identify objects such as cars, trees, people, or animals.

***Working:***

1. The input image is passed to the neural network.
2. The network scans the image in small parts (like looking at small windows).
3. It calculates features and predicts which object is present in each region.
4. The result shows the object with a bounding box around it.

***Example:***

- Detecting vehicles in traffic cameras.
- Identifying fruits on a conveyor belt in agriculture.

**3.3 Image Segmentation using ANN**

Image segmentation is the process of dividing an image into multiple parts or regions to make it easier to analyze. Each part represents an object or a section of interest.

ANNs, particularly U-Net and FCN (Fully Convolutional Networks) models, are used to segment images based on learned features.

***Working:***

1. The input image is broken into small pixels.
2. ANN assigns a label to each pixel (e.g., sky, road, person).
3. The output is a segmented image where each object is color-coded.

**Example:**

- Separating roads from cars in self-driving car systems.
- Identifying organs in medical imaging (e.g., CT scans).

**3.4 Classification using ANN**

Classification means identifying which category or class an item belongs to. For example, deciding whether an email is spam or not spam.

A simple feed-forward ANN can classify data based on training. It works especially well for:

- Image classification (dog vs. cat)
- Text classification (spam vs. not spam)
- Medical diagnosis (disease or no disease)

***Working:***

1. Input data is fed to the ANN.
2. The network applies weights and activation functions.
3. The output is a class label (e.g., class 1: apple, class 2: banana).

***Example:***

- Classifying handwritten digits in postal services.
- Sorting products in an online store by category.

**3.5 Speech to Text using ANN**

Speech-to-text means converting spoken words into written text. This is useful in virtual assistants like Google Assistant or Siri.

Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks are used for sequential data like speech.

***Working:***

1. Microphone input is converted into sound waveforms.
2. The ANN processes these waveforms and identifies phonemes (sound units).
3. These are converted into words using language models.

***Example:***

- Voice typing in smartphones.
- Assisting differently-abled users in typing using voice.

### 3.6 Machine Translation using ANN

Machine translation is the automatic conversion of text from one language to another, like English to Hindi.

Sequence-to-sequence models using RNNs, LSTMs, and Transformer models are used for this task.

***Working:***

1. The source language is input into the network.
2. The model understands sentence meaning (encoding).
3. It generates an equivalent sentence in the target language (decoding).

***Example:***

- Translating user reviews on e-commerce websites.
- Assisting in reading foreign language texts.

### 3.7 Practical Activities

To make the learning interactive, let's perform hands-on activities using Scratch — a visual block-based programming language ideal for school students and low-end PCs.

#### 3.7.1 Object Detection using Scratch Script

***Goal****:* Detect whether an object is a "Ball" or "Box" using color detection.

**Step 1.** Open Scratch.

**Step 2.** Use two sprites (e.g., one red, one blue).

**Step 3.** Use if-then blocks to check:

**Step 4.** Run and test by changing colors.

```
if color touching color then
   say "Object Detected: Ball"
else
   say "Object Detected: Box"
```

***Explanation****:* Though Scratch doesn't use actual neural networks, this exercise simulates decision-making logic as seen in ANN.

### 3.7.2 Speech to Text using Scratch

***Goal**:* Convert voice command into displayed text.

**Step 1.** Use a speech-to-text extension in Scratch or use Google Chrome with voice input.

**Step 2.** Use blocks like:

```
when green flag clicked
say (speech recognition result)
```

**Step 3.** Speak into the microphone and see the text appear.

***Explanation**:* This simulates speech-to-text conversion where audio input is translated into text.

### 3.7.3 Image Segmentation using Scratch

***Goal**:* Separate an image into two color regions.

**Step 1.** Load an image of a fruit with two colors (e.g., half red, half green).

**Step 2.** Use the pen or color sensing block:

```
if touching color red then
   say "Region 1"
if touching color green then
   say "Region 2"
```

**Step 3.** Move the sprite across the image.

***Explanation**:* The program simulates basic segmentation by detecting colors in different regions.

## Summary

- Neural Networks are used in various real-life applications.
- Applications include object detection, segmentation, classification, speech processing, and language translation.
- Scratch can be used to simulate ANN applications using simple logic blocks.
- These examples provide a base for understanding advanced AI concepts.

## Check Your Progress

**A. Multiple Choice Questions (MCQs)**

1. Which of the following is used in image classification? (a) U-Net (b) Feedforward ANN (c) Transformer (d) LSTM
2. What is the function of object detection? (a) Translate language (b) Segment image (c) Identify and locate objects (d) Convert text to speech
3. Which network is best for sequential data like speech? (a) CNN (b) RNN (c) ANN (d) GAN
4. Image segmentation helps in: (a) Creating audio (b) Dividing image into regions (c)

Playing videos (d) Reading sensors

5. Machine translation means: (a) Translating signals into circuits (b) Changing language of text (c) Detecting animals (d) Classifying diseases

**B. Fill in the Blanks**

1. _____ is the technique of separating images into regions.
2. Feedforward ANN is commonly used in _____.
3. _____ models are used in speech-to-text systems.
4. In object detection, the ANN predicts the _____ of the object.
5. _____ is used to convert English text to Hindi.

**C. State Whether True or False**

1. Neural Networks can be used for object detection.
2. Scratch cannot be used to simulate AI logic.
3. CNN is used for image tasks like segmentation.
4. ANN cannot process audio data.
5. Machine translation is not possible using ANN.

**D. Short Answer Questions**

1. What is the difference between classification and object detection?
2. How does speech-to-text work use ANN?
3. Describe a simple Scratch activity that mimics object detection.
4. Why is image segmentation useful?
5. What are the types of neural networks used in machine translation?



## Session 4. Python Libraries for Neural Networks

Suppose you want to cook a new recipe. You could either make everything from scratch (grind spices, churn butter) or use ready-made ingredients to save time. In the same way, Python provides libraries like NumPy, TensorFlow, and Keras to build neural networks easily. Instead of doing all the math manually, these tools give us ready-made building blocks to focus on ideas.

In this session, you will understand the importance of Python libraries, how to set up an environment, the difference between low-level (NumPy) and high-level (Keras) approaches, and how to build and train a simple neural network using the MNIST dataset.

### 4.1. Introduction

Python is the dominant language in education and industry for machine learning because it is readable, well-documented and supported by a large ecosystem of libraries, see the Figure 4.1 for better understanding. Libraries such as NumPy provide efficient numerical operations, while TensorFlow and its high-level interface Keras provide ready-made

building blocks (layers, optimizers, loss functions) that let us focus on ideas rather than implementation details.

- **NumPy**: efficient numerical operations.
- **TensorFlow**: deep learning framework.
- **Keras**: high-level API for TensorFlow (simple, student-friendly).
- **Matplotlib**: plotting graphs and visualizations.
- **scikit-learn**: machine learning tools.

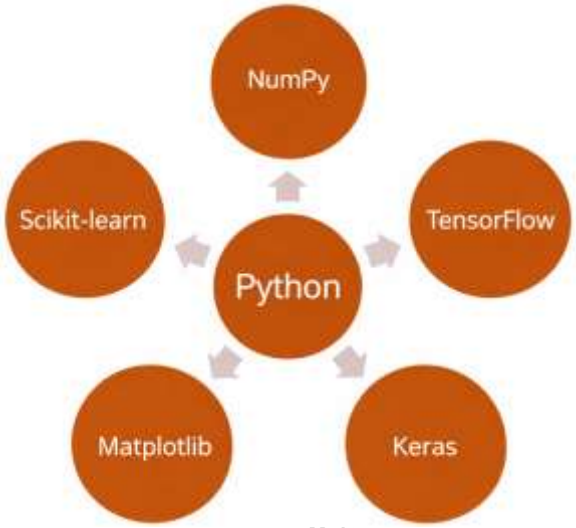



Fig. 4.1: Python ML ecosystem

### 4.2. Setting up the Python environment

You can set up the Python environment either by using Anaconda or Python + pip.

**Anaconda** distribution (easy, includes many packages)

**Installation:**

```
conda create -n nn_env python=3.9
conda activate nn_env
conda install -c conda-forge tensorflow matplotlib numpy pandas
```

Python + pip (lightweight) – useful if disk space or installation time is limited.

*Installation:*

```
python -m pip install --upgrade pip
pip install tensorflow matplotlib numpy pandas
```

**Minimum suggestions for a low-end PC**

- Python 3.8–3.10 (choose one compatible with TensorFlow builds),
- 8 GB RAM recommended but 4 GB can work with small datasets,
- Disk space: 5–10 GB for basic environments (more if datasets or models are stored).

**Verification** (run in Python or a terminal):

```
import tensorflow as tf
print("TensorFlow version:", tf.__version__)
```

**Expected output:**

TensorFlow version: 2.11.0

***Note**: (Exact version may differ)*

### 4.3 Low-level vs High-level Programming Approaches

When working in Artificial Intelligence (AI) and Machine Learning (ML), there are generally two ways of programming models: Low-Level and High-Level approach.

Understanding both methods is useful. Low-level programming gives insight into the mathematics behind AI, while high-level APIs allow us to prototype quickly.

#### 4.3.1 Low-level: NumPy-based Neuron

In the Low-level approach (using libraries like NumPy), every mathematical step is written manually. A single artificial neuron performs three steps:

1. Takes inputs and multiplies them with weights.
2. Adds a bias term.
3. Applies an activation function (e.g., ReLU).

**Example: Single neuron with ReLU**

```
import numpy as np
x = np.array([2.0, -1.0])       # inputs
w = np.array([0.5, -0.5])       # weights
b = 0.1                          # bias
z = np.dot(x, w) + b             # weighted sum
a = max(0.0, z)                  # ReLU activation
print("z =", z, "activation =", a)
```

**Step-by-step working:**

- `Weighted sum: z = 2.0*0.5 + (-1.0)*(-0.5) + 0.1 = 1.6`
- `Activation: ReLU(1.6) = 1.6`

**Expected output:**

```
z = 1.6 activation = 1.6
```

This illustrates how a neuron works mathematically; see Figure 4.4.3 for visual plots of ReLU and Sigmoid.

#### 4.3.2 High-level: Keras API (TensorFlow backend)

In the High-level approach (using frameworks like *Keras*), predefined layers and functions simplify the process. Keras provides ready-to-use components for defining layers, activations, loss functions, and optimizers. Instead of writing mathematical details, we can describe a full neural network in just a few lines.

**Advantages of Keras:**

- Helps to prototype models quickly.
- Works on both CPU and GPU without code changes.
- Runs on low-end PCs for small datasets and models.

**Example code (conceptual model):**

```
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential([
    layers.Dense(64, activation='relu', input_shape=(784,)),
    layers.Dense(10, activation='softmax')
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

The above model corresponds to the network diagram in Figure 4.4.2, where a 28×28 input image is flattened and passed through Dense layers.

### 4.3.3 Comparison of Both Approaches

*Table 4.1: Comparison between NumPy (low-level) and Keras (high-level).*

| Feature | NumPy (Low-Level) | Keras (High-Level) |
|---|---|---|
| **Code Size** | Many lines (manual math) | Few lines (ready modules) |
| **Learning Benefit** | Shows internal working | Focus on applications |
| **Flexibility** | Very high (custom design) | Limited to given functions |
| **Speed** | Slower (manual ops) | Faster (TensorFlow backend) |
| **Best For** | Learning fundamentals | Real-world prototyping |

## 4.4. Key components for neural networks

A layer in a neural network is a collection of neurons (or nodes) that work together to process inputs and pass their outputs to the next stage. Each layer transforms the data in some way, allowing the network to learn patterns step by step.

There are different types of layers, each serving a special purpose:

**Dense (Fully Connected Layer):**

- Every neuron in this layer is connected to every neuron in the previous layer.
- Dense layers are commonly used in simple feed-forward models.
- Example: A Dense (64) layer has 64 neurons, all receiving inputs from the previous layer.

**Flatten (Reshape Layer):**

- Converts a 2D matrix (like an image) into a 1D vector.
- Example: A 28×28 image (784 pixels) becomes a 784-element vector before entering a Dense layer.

See Figure 4.2, which shows how an input image passes through Flatten → Dense (64) → Dense (10).

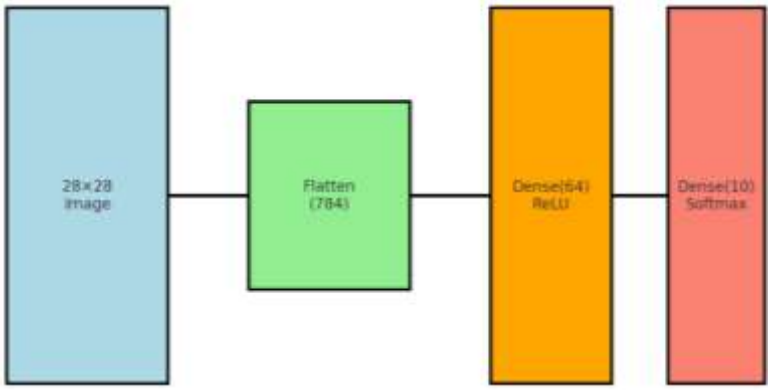



Fig. 4.2: Pipeline: Flatten → Dense (64) → Dense (10)

### 4.4.2. Activation functions

Activation functions introduce non-linearity into neural networks, allowing them to learn complex patterns.

**ReLU**:

- Formula: $f(x) = max(0, x)$
- If the input is positive, it returns the same value; if negative, it returns 0.
- Commonly used in hidden layers because it makes training faster.

**Softmax**:

Formula: $\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$

- Converts raw scores (logits) into probabilities for multi-class classification.
- Example: For handwritten digit recognition (0–9), Softmax ensures the output layer gives probabilities that add up to 1.

**Sigmoid**:

Formula: $\sigma(x) = \frac{1}{1+e^{-x}}$

- Outputs values between 0 and 1.
- Commonly used in binary classification problems (e.g., yes/no, true/false).

See the Figure 4.3 for plots of ReLU, Sigmoid, and Softmax. Notice that while ReLU and Sigmoid are scalar functions, Softmax operates on vectors and outputs a probability distribution.

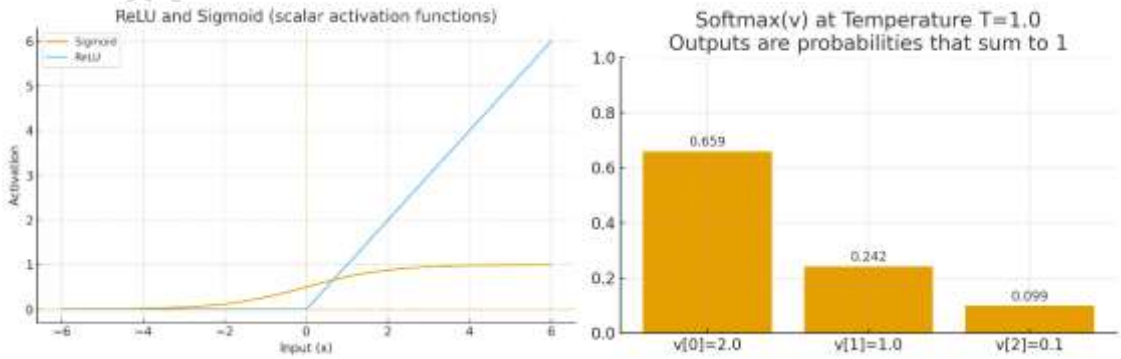



Fig. 4.3: Plot of ReLU, sigmoid and softmax concept

### 4.4.3. Loss functions and optimizers

**Loss Function**

- A loss function tells us *how far the model's predictions are from the correct answers (labels).*
- The goal of training is to reduce the loss value step by step.

*Example:* In digit classification (0–9), the commonly used loss is sparse_categorical_crossentropy, which compares the predicted probability distribution (from Softmax, see Figure 4.3) with the true label.

**Optimizer**

- An optimizer decides *how the network updates its weights* to reduce the loss.
- It adjusts weights gradually after each batch of training data.
- For beginners, Adam is widely recommended because it automatically adapts the learning rate, making training stable and efficient.

Together, loss functions and optimizers act like the "teacher" and the "learning strategy" of the neural network:

- The loss tells the network what mistakes it made.
- The optimizer helps the network improve by correcting those mistakes.

### 4.5. Practical Task: Building a small MNIST digit classifier

This is an end-to-end activity that students can run on low-end PCs by using a reduced subset of MNIST.

MNIST contains 70,000 small greyscale images of handwritten digits (0–9) at 28×28 pixels. For classroom practice we use only a subset (e.g., 10,000 training images and 2,000 test images) to keep resource use low. See table 4.2 that shows full and tiny datasets.

| Dataset Type | Training Samples | Testing Samples | Purpose |
|---|---|---|---|
| **Full MNIST** | 60,000 | 10,000 | Standard dataset (large, complete) |
| **Tiny Subset MNIST** | 10,000 | 2,000 | Used in this chapter (low-end PC friendly) |

**Step-by-step code:** Save as mnist_small.py

```
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np

# Load MNIST and take a small subset
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
x_train, y_train = x_train[:10000], y_train[:10000]
x_test, y_test = x_test[:2000], y_test[:2000]

# Preprocess: flatten and scale
x_train = x_train.reshape(-1, 28*28).astype('float32') / 255.0
```

```
x_test = x_test.reshape(-1, 28*28).astype('float32') / 255.0

# Build model
model = keras.Sequential([
    layers.Dense(64, activation='relu', input_shape=(784,)),
    layers.Dense(10, activation='softmax'),
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train (small number of epochs for low-end PC)
history = model.fit(x_train, y_train, epochs=5, batch_size=128,
verbose=2)

# Evaluate
loss, acc = model.evaluate(x_test, y_test, verbose=2)
print(f'Test accuracy: {acc:.2f}')
```

**Training output:**

(Actual numbers vary by machine and random initialization.)

```
Epoch 1/5
79/79 - 3s - loss: 0.4306 - accuracy: 0.8686
Epoch 2/5
79/79 - 2s - loss: 0.2017 - accuracy: 0.9423
Epoch 3/5
79/79 - 2s - loss: 0.1420 - accuracy: 0.9578
Epoch 4/5
79/79 - 2s - loss: 0.1122 - accuracy: 0.9645
Epoch 5/5
79/79 - 2s - loss: 0.0929 - accuracy: 0.9711
63/63 - 0s - loss: 0.1088 - accuracy: 0.9660
Test accuracy: 0.97
```

**Notes about the outputs**

- The accuracy shown is for this reduced example; using the full MNIST dataset and more sophisticated models (e.g. convolutional networks) will give different results.
- Training time on a low-end PC will be longer than on a GPU. Use small epochs and batch sizes for classroom exercises.

See Figure 4.4, which plots the training loss and accuracy from history

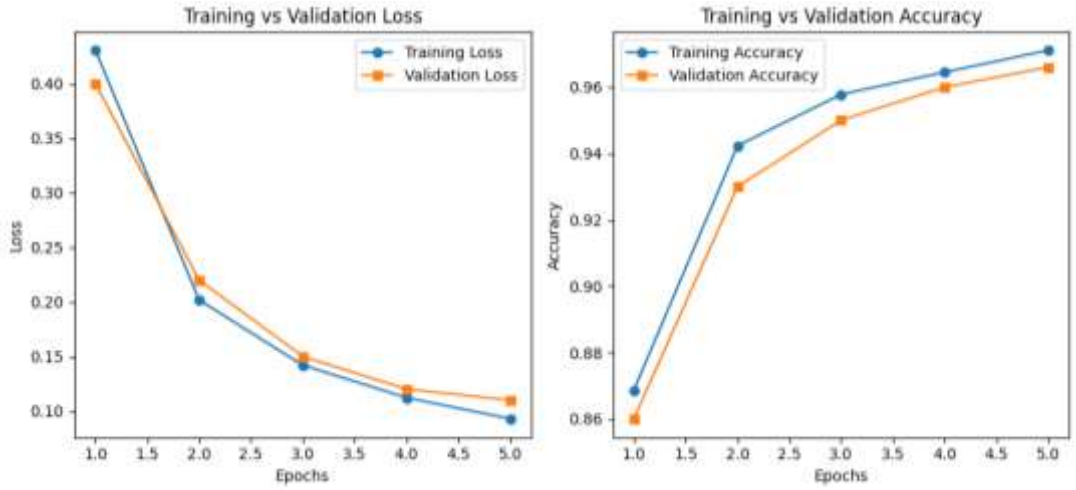



Fig. 4.4: Training Loss and Accuracy over Epochs

### 4.5.1 Interpreting model components

- **Input shape (784)**: the flattened 28×28 image.
- **Dense(64, ReLU)**: hidden layer with 64 units and ReLU activation.
- **Dense(10, Softmax)**: output layer producing a probability distribution across 10 classes.
- **Loss = sparse_categorical_crossentropy**: suitable when labels are integers rather than one-hot vectors.
- **Optimizer = adam**: reasonable default that adapts learning rates.

Table 4.3, Explain layer shapes and parameter counts for the small model.

***Table 4.3: shapes and parameter count***

| Layer (Keras) | Input Shape | Output Shape | Parameters Count | Explanation |
|---|---|---|---|---|
| Dense (64, ReLU) | (784,) | (64,) | 50,240 | Each of the 784 inputs connects to 64 neurons: (784 × 64) + 64 (bias terms) |
| Dense (10, Softmax) | (64,) | (10,) | 650 | Each of the 64 neurons connects to 10 outputs: (64 × 10) + 10 (bias terms) |
| **Total Parameters** | – | – | **50,890** | Sum of parameters across all layers |

### 4.5.2 Predicting for a single image (example)

After training, use model.predict() on new images. For a single test image:

```
import numpy as np
img = x_test[0].reshape(1, 784)   # already normalized in previous code
probs = model.predict(img)
pred = np.argmax(probs, axis=1)[0]
print("Predicted digit:", pred, "with probability:", probs[0][pred])
```

**Possible output**:

Predicted digit: 7 with probability: 0.99

**4.5.3 Saving and loading a model**

**Save the trained model to disk:**

model.save('mnist_small_model.h5')

**Load it later:**

from tensorflow.keras.models import load_model

model2 = load_model('mnist_small_model.h5')

**4.5.4 Best practices for low-end PCs**

- **Use small subsets of datasets** (10–25%): reduces memory and compute needs. See *Table 4.4.2* for recommended sizes.
- **Limit epochs** (5–20) for experiments; increase only when needed.
- **Use small batch sizes** (32–128) — they balance memory and training stability.
- **Keep the model architecture simple**: one or two hidden layers are enough for classroom tasks.
- **Use model checkpoints** (save best weights) and early stopping (stop when validation loss stops improving).
- **Run experiments overnight** if they take longer.

**4.5.5 Troubleshooting common problems**

- **Import errors after installation**: check Python version and TensorFlow compatibility. Use the official compatibility matrix online (teacher should provide link).
- **Slow training**: reduce dataset size, use fewer epochs or a smaller model.
- **Memory errors**: reduce batch_size or use smaller data subsets.
- **Different TF versions on pip vs conda**: prefer one installation method and create isolated environments for experiments.

**4.6 Practical activities and classroom tasks**

1. **Install and verify**: Students install Python/TensorFlow and run print(tf.__version__). Record and discuss differences.
2. **NumPy neuron lab**: change weights and bias in the simple neuron program and write observations.
3. **Mini MNIST project**: follow the mnist_small.py script, train for 5 epochs, produce the training plot and final accuracy, and comment on where misclassifications occur.
4. **Model comparison**: train with and without a hidden layer (only output layer) When downloading or using datasets, check licenses and ensure data privacy. If students create datasets using personal data (e.g., their own handwriting), get consent and avoid sharing personally identifiable information.

## Summary

This chapter introduced Python libraries and practical steps to build a simple neural network using Keras. You learned how to set up an environment, compared low-level and

high-level approaches, built a compact MNIST classifier suitable for low-end PCs, and reviewed practices to keep experiments efficient and safe. These foundations prepare you to explore more advanced models such as convolutional neural networks (CNNs) in later chapters.

## Check Your Progress

### A. Multiple Choice Questions

1. Which library provides a high-level API specifically designed to help build neural networks quickly? (a) NumPy (b) pandas (c) Keras (d) Matplotlib
2. Which command will install TensorFlow using pip? (a) pip install numpy (b) pip install tensorflow (c) conda install tf (d) install tensorflow
3. Softmax activation is most suitable for: (a) Binary classification (b) Multi-class classification c) Regression d) Clustering
4. In the MNIST example, which metric is used to monitor model performance during training? (a) precision (b) accuracy (c) entropy (d) MAE (mean absolute error)
5. Which layer type is used to flatten a 28×28 image into a vector before passing to Dense layers? (a) Conv2D (b) Dense (c) Flatten (d) Reshape via numpy

### B. Fill in the blanks

1. TensorFlow is developed by _____.
2. The high-level API built on TensorFlow is called _____.
3. NumPy allows performing matrix operations such as _____.
4. We used the _____ optimizer in the MNIST example.
5. The softmax activation transforms outputs into probabilities that sum to _____.

### C. State whether True or False

1. Keras requires fewer lines of code than a NumPy implementation for the same network.
2. Installing TensorFlow via pip and Anaconda always gives identical results.
3. Softmax is the correct activation for multi-class output layers.
4. Dense layers are fully connected between inputs and outputs.
5. Using a very large batch size always speeds up training on low-end PCs.

### D. Short answer questions

1. List the main steps to install TensorFlow using pip and verify the installation.
2. Outline the key differences between a NumPy implementation of a neuron and a Keras Dense layer.
3. Describe the structure of the simple MNIST classifier used in the practical activity.
4. Why should we limit epochs and dataset size when using low-end PCs? Give two reasons.
5. What does the model.compile() step do in Keras? Mention at least two actions it performs.

# Session 5. Training Neural Networks

Preparing data for training is like preparing for an exam. A student doesn't just read raw notes — they organize them, highlight key points, and practice with tests. Similarly, data must be normalized, reshaped, and encoded before training. The training process itself is like repeated practice tests (epochs), where mistakes (loss) reduce and performance (accuracy) improves.

In this session, you will understand the steps of data preprocessing (normalization, reshaping, one-hot encoding), the difference between training and testing data, evaluation measures like accuracy and confusion matrix, and the role of activation functions (ReLU, Sigmoid, Tanh) during training.

## 5.1. Data Preparation/Preprocessing

Before training a neural network, the data must be prepared in a way that the computer can easily understand and process. Raw data often comes in different formats, scales, and categories, which can confuse the model. To make learning smooth and effective, we perform some important preprocessing steps.

Three of the most common steps are:

1. **Normalization** – adjusting numbers so they fall within a small, consistent range (like 0 to 1). This helps the model learn faster and avoid errors caused by very large or uneven values.
2. **Reshaping** – arranging data into the required shape or structure. Neural networks expect data in specific formats (like rows, columns, or flat arrays), so reshaping ensures the input fits properly without changing the actual values.
3. **One-Hot Encoding** – converting categories (like fruits, colors, or pass/fail labels) into numerical form using 0s and 1s. Since computers understand numbers, this step allows categorical data to be used in classification tasks.

Together, these steps make sure the input data is clean, organized, and ready for the neural network to process effectively.

### 5.1.1. Normalization

Normalization is the process of scaling data values into a smaller, standard range, usually between 0 and 1. Neural networks learn faster when the input data is small and consistent. If the data has very large or uneven values, the model struggles to learn properly.

**Mathematical Expression**

The most common method is **Min-Max Normalization**:

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

Where:

$x$ = original value

$x_{\min}$ = smallest value in dataset

$x_{\max}$ = largest value in dataset

$x'$ = normalized value (between 0 and 1)

**Real-Life Analogy**

Imagine different students in a class scoring marks out of different totals:

- Riya scores 80/100 in Science.
- Aryan scores 45/50 in Math.

To compare fairly, the teacher converts marks into percentages.

- Riya = 80%
- Aryan = 90%

This is normalization: converting values into the same scale for fair comparison.

**Practical Example**

```
import numpy as np
# Original marks of students
marks = np.array([45, 80, 90, 60, 30])
# Normalization (0 to 1 scale)
norm_marks = (marks - marks.min()) / (marks.max() - marks.min())
print("Normalized Marks:", norm_marks)

Output:
Normalized Marks: [0.25 0.83 1.0 0.5 0.0]
```

### 5.1.2. Reshaping

Reshaping means changing the structure or dimensions of data without changing its values. Neural networks often need inputs in a specific shape. For example, images may need to be reshaped into a single row of numbers before being processed.

**Real-Life Analogy**

Think of arranging books in a shelf:

- Sometimes you keep them vertically.
- Sometimes you lay them horizontally.

The books remain the same, but their arrangement (shape) changes. That's what reshaping does to data.

**Practical Example**

```
import numpy as np

# A 2x3 matrix (2 rows, 3 columns)
data = np.array([[1, 2, 3],
                 [4, 5, 6]])

print("Original shape:", data.shape)

# Reshape into 3x2 matrix
```

```
reshaped = data.reshape(3, 2)
print("Reshaped to 3x2:\n", reshaped)

# Reshape into 1D array (flatten)
flattened = data.reshape(-1)
print("Flattened array:", flattened)

Output:
Original shape: (2, 3)
Reshaped to 3x2:
 [[1 2]
  [3 4]
  [5 6]]
Flattened array: [1 2 3 4 5 6]
```

### 5.1.3. One-Hot Encoding for Classification

One-hot encoding is a method of converting categorical data (labels) into a numerical form that a machine can understand. Each category is represented as a vector of 0s and 1s.

Only one position is 1 (hot), and the rest are 0.

**Example**

Suppose we have 3 fruit categories:

- Apple → [1, 0, 0]
- Banana → [0, 1, 0]
- Orange → [0, 0, 1]

If the dataset is: [Apple, Banana, Orange, Apple]

After one-hot encoding:

[1, 0, 0]

[0, 1, 0]

[0, 0, 1]

[1, 0, 0]

**Real-Life Analogy**

Think of traffic signals:

- Red light ON → Stop
- Yellow light ON → Wait
- Green light ON → Go

At any moment, only one light is ON (1) and the others are OFF (0). That's just like one-hot encoding.

**Practical Example**

```
from sklearn.preprocessing import OneHotEncoder
```

```
import numpy as np

# Fruits dataset
fruits = np.array([["Apple"], ["Banana"], ["Orange"], ["Apple"]])

# One-hot encoding
encoder = OneHotEncoder(sparse=False)
encoded = encoder.fit_transform(fruits)

print("One-Hot Encoded:\n", encoded)
print("Categories:", encoder.categories_)

Output:
One-Hot Encoded:
 [[1. 0. 0.]
  [0. 1. 0.]
  [0. 0. 1.]
  [1. 0. 0.]]
Categories: [array(['Apple', 'Banana', 'Orange'], dtype=object)]
```

### 5.2 Training Data vs Testing Data

In machine learning, we split our data into two parts:

- **Training data** – used to teach the neural network (typically 80% of data)
- **Testing data** – used to check if the network learned well (typically 20%)

This concept ensures the model performs well not only on data it has seen (training) but also on new data (testing). If a model works only on training data, it may be *overfitting*.

### 5.3 Epochs, Loss, Accuracy and Confusion Matrix

#### 5.3.1 Epoch

One epoch means the model has processed the entire training dataset once. Models usually train for many epochs (e.g., 20–50) to improve.

#### 5.3.2 Loss

The loss function calculates how far the model's predictions are from the correct answers. In binary classification, we commonly use binary crossentropy. Lower loss is better.

#### 5.3.3 Accuracy

Accuracy is the share of correct predictions. For example, 8 correct predictions out of 10 gives 80% accuracy.

#### 5.3.4 Confusion Matrix

A confusion matrix is a table used to check how well a classification model performs. It compares the actual answers with the model's predictions. See Table 4.5.1, which shows the structure of a 2-Class Confusion Matrix.

*Table 4.5.1: Structure of a 2-Class Confusion Matrix*

| | Predicted Positive | Predicted Negative |
|---|---|---|
| Actual Positive | True Positive (TP) | False Negative (FN) |
| Actual Negative | False Positive (FP) | True Negative (TN) |

**Meaning of Each Term**

- True Positive (TP): The model correctly predicted Positive.
- True Negative (TN): The model correctly predicted Negative.
- False Positive (FP): The model predicted Positive, but it was actually Negative.
- False Negative (FN): The model predicted Negative, but it was actually Positive.

**Imagine a medical test for detecting a disease:**

- TP: Patient has the disease, and the test says "Yes." (Correct detection)
- TN: Patient does not have the disease, and the test says "No." (Correct rejection)
- FP: Patient is healthy, but the test says "Yes." (Unnecessary worry)
- FN: Patient is sick, but the test says "No." (Dangerous miss)

Accuracy alone may be misleading if the data is imbalanced (e.g., many more healthy people than sick ones).

**The confusion matrix gives detailed insight:**

- How many positives were caught?
- How many negatives were missed?

It helps improve the model by showing exactly where mistakes are happening.

**Example: Real-Life Analogy of Practice Tests**

Riya is preparing for a science quiz with a set of 50 questions. On Monday, she attempts all 50 questions for the first time and makes 20 mistakes, scoring 60%. This is like the first epoch of training a neural network.

She studies the topics again and tries the same set of questions on Tuesday. This time, she makes only 10 mistakes and scores 80%. This represents the second epoch.

As she keeps practicing the same set multiple times (more epochs), her performance improves:

- She makes fewer mistakes (loss goes down),
- Her score improves (accuracy increases).

**In Riya's Words:**

- **Epoch**: One complete round of learning by going through all the questions once.
- **Loss**: The number of mistakes made while learning (20 on Day 1, 10 on Day 2).
- **Accuracy**: The percentage of correct answers.
- **Confusion Matrix:** shows the details: how many pass/fail predictions were correct or mistaken.

As Riya keeps practicing (more epochs), she makes fewer mistakes (loss goes down) and scores better (accuracy goes up).

Just like Riya needs multiple practice rounds to improve her score, a neural network also needs to go through the data multiple times (epochs) to reduce mistakes (loss) and increase prediction accuracy.

But sometimes, just looking at the total score (accuracy) is not enough. We also want to know what kinds of mistakes Riya made.

For example:

- Did she mark an answer as correct when it was actually wrong? (False Positive)
- Did she miss a question that was actually right? (False Negative)
- Or did she correctly answer both right and wrong ones? (True Positive, True Negative)

This detailed breakdown of correct and incorrect answers is shown by a confusion matrix.

So, while accuracy shows Riya's overall performance, and loss shows how far her answers are from being perfect, the confusion matrix explains the exact pattern of her mistakes — helping her (and the model) know where improvement is needed.

### 5.4 Activation Functions

Activation functions help neurons decide whether to "fire" or not based on the input. They introduce non-linearity, which helps neural networks learn complex patterns, not just straight-line relationships.

Let's understand the three most common activation functions using real-life relatable stories.

#### 5.4.1 ReLU (Rectified Linear Unit)

A ReLU (Rectified Linear Unit) is an activation function in neural networks that outputs the input directly if it is positive, and outputs 0 if the input is negative. ReLU allows only positive signals to pass and blocks negative ones.

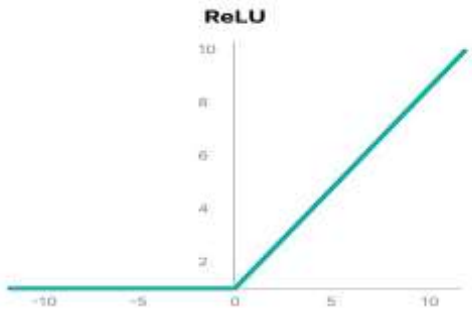



Fig. 5.1: ReLU Result

**Mathematical Expression:**

$f(x) = max(0, x)$

**Real-Life Analogy: Automatic Night Lamp**

Riya has an automatic lamp in her room. It only turns on when the room is dark.

- If the brightness is –2 (bright day), the lamp stays OFF → Output: 0
- If the brightness is 1 (a bit dim), the lamp turns ON slightly → Output: 1
- If the brightness is 5 (very dark), the lamp turns ON brightly → Output: 5

**Explanation:**

- ReLU acts just like the lamp. It blocks all negative or weak signals and allows strong signals to pass.
- In neural networks, this helps the model focus only on important information and ignore noise.

### 5.4.2 Sigmoid

The Sigmoid activation function is a mathematical function that converts any input value into a number between 0 and 1. Sigmoid "squeezes" values into the range of 0 to 1, making it useful for probabilities.

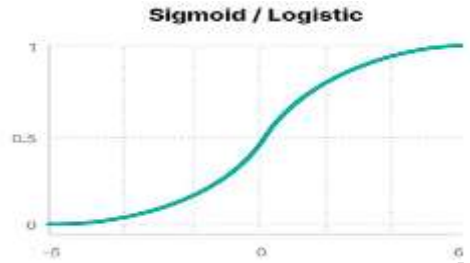



Fig. 5.2: Sigmoid Result

**Mathematical Expression:**

$$f(x) = \frac{1}{1 + e^{-x}}$$

**Real-Life Analogy:** Sweater Decision Based on Temperature

Riya decides whether to wear a sweater based on the outside temperature:

- At 5°C, she definitely wears a sweater → Output close to 1
- At 30°C, she doesn't wear it → Output close to 0
- At 18°C, she's unsure → Output = 0.5

**Explanation:**

- Sigmoid converts input into a probability between 0 and 1.
- It is useful when the model needs to predict probabilities, like yes or no decisions — for example, "Will Riya wear a sweater today?"

### 5.4.3 Tanh (Hyperbolic Tangent)

The Tanh (Hyperbolic Tangent) activation function is similar to the Sigmoid function but it outputs values between –1 and +1 instead of 0 and 1. Tanh squashes numbers into the range –1 to +1, keeping negative inputs negative and positive inputs positive.

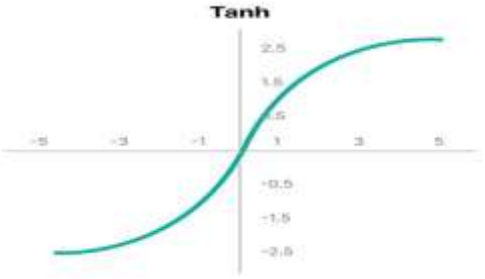



Fig. 5.3: Sigmoid Result

**Mathematical Expression:**

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

**Real-Life Analogy: Feedback on Riya's Science Project**

Riya shows her science project to her friends.

- One says, "Amazing!" → Output: +1
- One says, "Awful!" → Output: –1
- Most say, "It's okay" → Output: 0

**Explanation:**

Tanh gives outputs between –1 and 1, unlike Sigmoid. It captures both positive and negative feedback, and is often used when the direction of feedback matters.

### 5.5 Model Training Process

Training a neural network means helping it learn patterns from data. Just like a student prepares for an exam by practicing and reviewing, a neural network also improves through repeated learning cycles. Let's understand this process step by step using a story-based analogy of Riya's exam preparation.

**Step 1. Prepare the Data**

Suppose we want to create a model that predicts whether Riya will pass or fail a weekly quiz, based on the number of correct practice questions she solves the day before. See Table 5.1, that have a small dataset based on her past performance:

Table 5.1: Dataset for model training

| Correct Questions in Weekly Quiz (Input) | Pass (1) / Fail (0) |
|---|---|
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 0 |
| 5 | 1 |
| 6 | 1 |
| 7 | 1 |
| 8 | 1 |
| 9 | 1 |
| 10 | 1 |

This is called a labeled dataset, where we know both the input (questions solved) and the output (pass/fail).

```
# Input: Correct practice questions | Output: 1 = Pass, 0 = Fail
import numpy as np
x = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])   # Questions Riya solved
```

```
y = np.array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1])     # 1=Pass, 0=Fail
```

**Step 2. Split the Dataset**

We divide the data into:

- **Training set (80%)**: To teach the model.
- **Testing set (20%)**: To check how well the model learned.

In this case:

```
x_train = x[:8]
y_train = y[:8]
x_test = x[8:]
y_test = y[8:]
```

See Table 5.2, where we divided the dataset.

Table 5.2: Split dataset

| Train Set (8 rows) | Test Set (2 rows) |
|---|---|
| Inputs: 1,2,3,4,5,6,7,8 | Inputs: 9,10 |
| Outputs: 0,0,0,0,1,1,1,1 | Outputs: 1,1 |

**Step 3. Build the Model**

We now build a simple neural network model with:

- **Input layer**: Takes number of correct questions.
- **Hidden layer**: 4 neurons using **ReLU** activation.
- **Output layer**: 1 neuron using **Sigmoid** to predict pass/fail.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential([
    Dense(4, activation='relu', input_dim=1),
    Dense(1, activation='sigmoid')
])
```

**Step 4. Compile the Model**

We tell the model how to learn:

```
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
```

- **Optimizer**: 'adam' helps improve learning.
- **Loss function**: Measures mistakes (we use binary cross-entropy for pass/fail).
- **Metric**: Accuracy (how often it's right).

**Step 5. Train the Model**

Now we let the model learn from Riya's data:

```
model.fit(x_train, y_train, epochs=100)
```

- Each epoch is one full pass through training data.
- With more epochs, the model becomes more accurate.

**Step 6. Evaluate the Model**

Finally, we check how well the model predicts on test data:

```
loss, accuracy = model.evaluate(x_test, y_test)
print(f"Loss: {loss:.2f}, Accuracy: {accuracy:.2f}")

Output:
Loss: 0.13, Accuracy: 1.00
```

This means the model correctly predicted that Riya would pass when she solved 9 and 10 questions.

**Riya's Learning Analogy with Stats**

Think of Riya preparing for a science quiz:

- **Day 1**: She solves only 2 questions → She fails the quiz.
- **Day 2**: She solves 5 questions → She barely passes.
- **Day 3**: She solves 10 questions → She scores full marks.

Table 5.3: Riya's performance

| Day | Questions Solved | Result | Comment |
|---|---|---|---|
| 1 | 2 | Fail | Not enough effort |
| 2 | 5 | Pass | Just crossed cutoff |
| 3 | 10 | Pass | Excellent work |

Now let's compare this to model training:

Table 5.4: Human vs Neural Network learning

| Human Learning (Riya) | Neural Network Model |
|---|---|
| Solving questions | Input data (questions solved) |
| Weekly test | Output (pass/fail) |
| Practice rounds | Epochs |
| Mistakes she made | Loss |
| Quiz score | Accuracy |

See Table 5.4, Just like Riya improves her result (Table 5.3) by practicing more, the model becomes better by running more epochs.

**5.6 Common Pitfalls & Fixes in Model Training**

Even after creating and training a neural network model, sometimes it doesn't work as expected. The model might give wrong predictions or perform well on known data but fail

on new data. These problems are called training pitfalls. Let's understand the common ones and how to fix them — using the example of Riya's quiz preparation model.

### 5.6.1 Low Accuracy

Low accuracy means the model makes many mistakes, even on the training data. For example, if the model predicts that Riya will pass when she actually failed (or vice versa), it's not learning correctly.

In Our Example, imagine we trained the model for only 5 epochs. The training might stop too early before the model has had a chance to learn the patterns.

```
# Too few training    rounds
codemodel.fit(x_train, y_train, epochs=5)
Output:
codeAccuracy: 0.50
```

This means the model is correct only half of the time.

**How to Fix it:**

- **Increase epochs:** Give the model more chances to learn.
- **Use more data:** If available, add more examples of Riya's quiz performances.

```
# More epochs improve learning
model.fit(x_train, y_train, epochs=100)
```

### 5.6.2 Overfitting

Overfitting happens when the model memorizes the training data but fails on test data. It's like Riya only preparing exact questions from her notes. If the teacher asks anything new, she can't answer.

In Our Example, Suppose we trained the model for 1000 epochs with the same small dataset.

```
model.fit(x_train, y_train, epochs=1000)
```

- It learns the training data perfectly (Accuracy: 1.0).
- But when tested on unseen inputs (e.g., solving 9 or 10 questions), it may still fail.

```
model.evaluate(x_test, y_test)
# Output might be: Accuracy: 0.50 or less
```

**How to Fix it:**

- **Reduce the number of epochs** (example: from 1000 to 100).
- **Simplify the model**: Use fewer neurons or layers to avoid over-learning.

```
# Fewer epochs prevent memorizing
model.fit(x_train, y_train, epochs=100)
```

### 5.6.3 "Dead" Neurons (ReLU Problem)

A "dead" neuron is one that always gives output = 0. It stops responding to input completely. This can happen with ReLU activation when the input becomes too small or negative.

In Our Example, If many inputs are small (e.g., questions solved = 1, 2, 3), then ReLU in the hidden layer will output 0, as:

$$f(x) = max(0, x)$$

If x < 0 → output = 0

If x = 2 → ReLU might still output 0 after weights & bias are applied internally.

So the neuron becomes inactive — like a student who stops participating because they're always ignored.

**How to Fix it:**

- Replace ReLU with Leaky ReLU**,** which allows small negative outputs.

```
from tensorflow.keras.layers import LeakyReLU

model = Sequential([
    Dense(4),
    LeakyReLU(alpha=0.1),
    Dense(1, activation='sigmoid')
])
```

- Reduce the learning rate so neurons adjust slowly without "dying out".

```
from tensorflow.keras.optimizers import Adam
model.compile(optimizer=Adam(learning_rate=0.001),
loss='binary_crossentropy', metrics=['accuracy'])
```

**5.7 Practical Activity**

**Activity 1: Train and Evaluate a Neural Network**

**Objective:** Use an 80/20 data split to train and test a neural network that predicts a student passing based on study hours.

**Real-Life Tiny Dataset**

```
# Tiny dataset: hours studied vs pass (0=fail, 1=pass)
# 10 students for classroom demonstration
X = [[1], [2], [3], [4], [5], [6], [7], [8], [9], [10]]
y = [0,   0,   0,   1,   1,   1,   1,   1,   1,    1]
```

**Example**

```
import numpy as np
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

X = np.array([[i] for i in range(1, 11)])
y = np.array([0,0,0,1,1,1,1,1,1,1])

X_train, X_test, y_train, y_test = train_test_split(
```

```
    X, y, test_size=0.2, random_state=42)

model = Sequential([
    Dense(4, activation='relu', input_dim=1),
    Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.fit(X_train, y_train, epochs=20, verbose=0)

loss, acc = model.evaluate(X_test, y_test, verbose=0)
print(f"Test loss: {loss:.3f}, Test accuracy: {acc:.2f}")
```

**Expected Output**

Test loss: 0.098, Test accuracy: 0.50

(New results may vary slightly.)

**Activity 2: Plot Activation Functions**

**Objective**: Plot ReLU, Sigmoid, and Tanh over a range to understand their shapes.

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

x = np.linspace(-5,5,200)
relu = np.maximum(0, x)
sigmoid = 1 / (1 + np.exp(-x))
tanh = np.tanh(x)

plt.plot(x, relu, label='ReLU')
plt.plot(x, sigmoid, label='Sigmoid')
plt.plot(x, tanh, label='Tanh')
plt.title('Activation Functions')
plt.xlabel('Input x')
plt.ylabel('Function output')
plt.legend()
plt.grid(True)
plt.show()
```

**Expected Plot Description**

- **ReLU**: flat at 0 for $x < 0$, then linear for $x > 0$
- **Sigmoid**: smooth S-curve between 0 and 1
- **Tanh**: S-curve between –1 and 1, centered around zero

## Summary

- How to divide data into training and testing sets
- The importance of epochs, loss, and accuracy
- Mathematical forms of ReLU, Sigmoid, and Tanh
- Implementing a small, real-life dataset model suitable for classroom PCs
- Visualizing activation functions to reinforce understanding

## Check Your Progress

### A. Multiple Choice Questions

1. What does an epoch represent? (a) A single training example (b) One pass through the complete training data (c) A prediction step (d) A unit of loss
2. Which activation function outputs zero for negative inputs and its input value for positive inputs? (a) Sigmoid (b) ReLU (c) Tanh (d) Softmax
3. Which metric measures how many correct predictions the model made? (a) Loss (b) Epoch (c) Accuracy (d) Weight
4. The Sigmoid function formula is: (a) $1/(1+e^{-x})$ (b) $max(0,x)$ (c) $(e^{x}-e^{-x})/(e^{x}+e^{-x})$ (d) $x^{2}$
5. What split ratio is used for training and testing in this chapter? (a) 50/50 (b) 60/40 (c) 80/20 (d) 100/0

### B. Fill in the Blanks

1. One _____ means the model has seen all the training data once.
2. _____ is a measure of prediction error used to update model weights.
3. ReLU returns zero when x is _____.
4. Accuracy = _____ correct predictions ÷ total predictions.
5. The Tanh function outputs values between _____ and _____.

### C. State Whether True or False

1. Using the same data for training and testing always gives a fair performance measure.
2. Sigmoid output ranges from –1 to 1.
3. Increasing epochs indefinitely can lead to overfitting.
4. ReLU introduces non-linearity in neural networks.
5. Loss always decreases as accuracy increases.

### D. Short Answer Questions

1. Why is it important to separate data into training and testing sets?
2. How does the Tanh activation function differ from Sigmoid?
3. What does a low loss value indicate during training?
4. In what situation would you consider reducing the number of epochs?
5. Describe one reason why ReLU is preferred in hidden layers.

# Session 6. Building a Basic Neural Network for Classification

Imagine Riya noticing her focus changes depending on how many hours have passed since her meal: too soon → sleepy, right time → focused, too late → hungry. This is a classification problem — "focused" or "distracted." Similarly, neural networks classify data into categories.

In this session, you will understand how to build a basic neural network for classification using Keras, the training and evaluation workflow, loss functions and optimizers, evaluation metrics (accuracy, precision, recall, confusion matrix), and techniques like dropout to prevent overfitting.

### 6.1 Layers: Dense (Fully Connected)

Think of a Dense layer like a classroom where every student (neuron) talks to every other student in the next class.

- Each input connects to all neurons in the next layer.
- Each connection has a weight, which is adjusted during training.

See Figure 6.1, we have built a small network with:

- **1 Input**: time after eating (in minutes).
- **1 Hidden Layer**: 4 neurons, using ReLU activation.
- **1 Output Neuron**: with Sigmoid activation (for binary 0/1 classification).

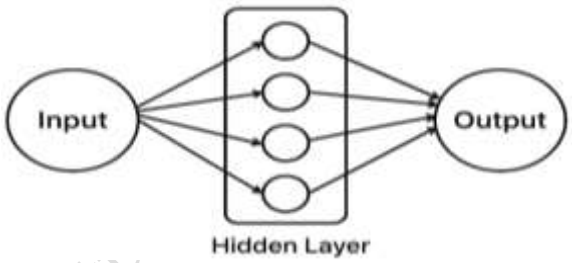



Fig. 6.1: Dense small network

### 6.2 Loss Functions and Optimizers

A neural network learns by minimizing loss.

- **Loss Function:** We use Binary Crossentropy, because we are solving a binary classification problem (two classes: distracted or focused).
- **Optimizer:**
  - We use Adam, which automatically adjusts learning rates and converges faster.
  - You may also use SGD (Stochastic Gradient Descent), but Adam is easier for beginners.

### 6.3 Training and Evaluation Workflow

To train a neural network, follow these steps:

1. **Prepare Data** – Split into training (80%) and test (20%).
2. **Build Model** – Define layers using Keras Sequential API.
3. **Compile Model** – Select optimizer, loss, and metrics.

4. **Train Model** – Run for multiple epochs.
5. **Evaluate** – Test on unseen data.
6. **Metrics** – Calculate confusion matrix, precision, recall.
7. **Visualize** – Plot accuracy/loss to understand learning.

See Figure 6.2, The flowchart of workflow from data → training → evaluation → metrics → visualization.

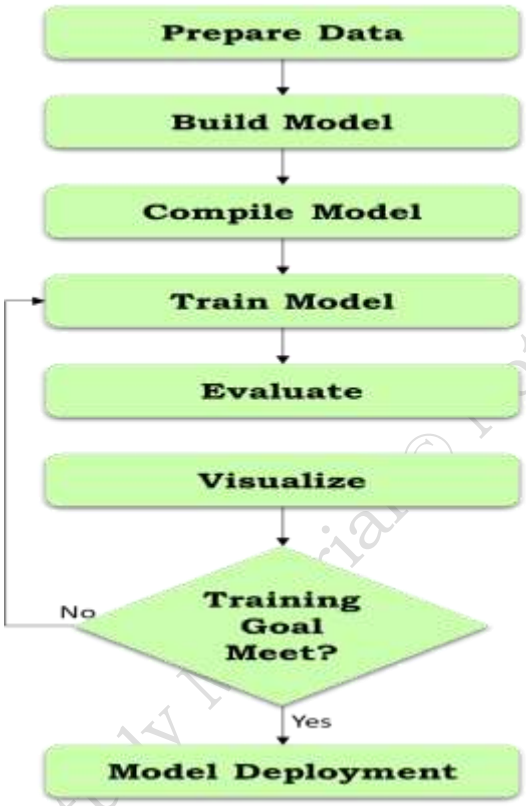



Fig. 6.2: Training and Evaluation Workflow

**6.4 Real-Life Example: Meal Before Study Classifier**

**Story: Riya's Focus and Meals**

Riya noticed that her focus depends on when she studies after a meal:

- **0–2 hours after eating** → feels sleepy/distracted (**0**)
- **3–5 hours after eating** → feels focused (**1**)
- **More than 5 hours** → feels hungry again, distracted (**0**)

This is a **binary classification problem**: Focused (1) vs. Distracted (0).

**Dataset**

We collect a **tiny dataset** (9 samples):

```
import numpy as np
x = np.array([1, 60, 120, 180, 240, 300, 360, 420, 480])  # Minutes
after meal
y = np.array([0, 0, 0, 1, 1, 1, 0, 0, 0])                          # 0
= Distracted, 1 = Focused
```

**Observations:**

- 1–120 min → distracted (too full).
- 180–300 min → focused (best digestion time).
- 360+ min → distracted (too hungry).

### 6.5 Model Construction, Training, and Output

**Step 1. Split the Data**

We split into training (80%) and testing (20%).

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=42)
```

**Step 2. Build the Model**

We use a Sequential model with 1 hidden layer.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential([
    Dense(4, activation='relu', input_dim=1),  # hidden layer
    Dense(1, activation='sigmoid')             # output layer
])
```

**Step 3. Compile the Model**

```
model.compile(optimizer='adam',loss='binary_crossentropy',
metrics=['accuracy'])
```

**Step 4. Train the Model**

```
history = model.fit(x_train, y_train, epochs=100, verbose=0)
```

**Step 5. Evaluate the Model**

```
loss, acc = model.evaluate(x_test, y_test, verbose=0)
print(f"Test Loss: {loss:.3f}, Test Accuracy: {acc:.3f}")
```

**Step 6. Predictions and Metrics**

```
from sklearn.metrics import confusion_matrix, precision_score,
recall_score

y_pred = (model.predict(x_test) > 0.5).astype(int).flatten()
cm = confusion_matrix(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)

print("Confusion Matrix:\n", cm)
print(f"Precision: {prec:.2f}, Recall: {rec:.2f}")
```

**Possible Output:**

Test Loss: 0.032, Test Accuracy: 1.000

**Confusion Matrix:**

[[1 0]
 [0 1]]

Precision: 1.00, Recall: 1.00

**Step 7: Visualize Training Accuracy**

```
import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'])
plt.title("Training Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.show()
```

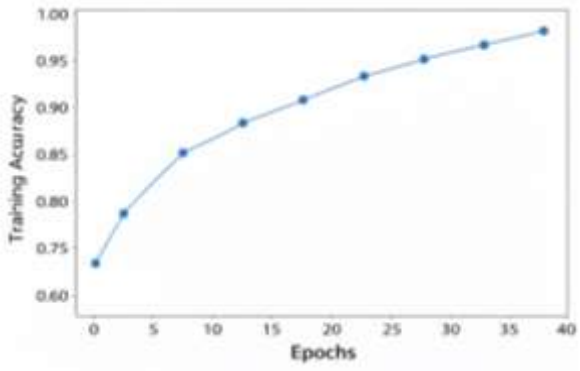



Fig. 6.3 — placeholder: Training accuracy vs. epochs plot

### 6.6 Model Performance: Metrics

- **Accuracy** → Overall correctness.
- **Precision** → Of all predicted positives, how many were correct?
- **Recall** → Of all actual positives, how many did we find?
- **Confusion Matrix** → Table showing True Positives, True Negatives, False Positives, False Negatives.

In our simple dataset, the model performs perfectly.

### 6.7 Overfitting vs. Underfitting & Dropout

**Underfitting**

- Happens when the model is too simple.
- Both training and test accuracy are low.

**Overfitting**

- Happens when the model **memorizes training data** but fails on new data.
- Training accuracy = high, Test accuracy = low.

**Solution: Dropout**

Dropout randomly turns off neurons during training, preventing over-reliance.

```
from tensorflow.keras.layers import Dropout

model_do = Sequential([
    Dense(4, activation='relu', input_dim=1),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])

model_do.compile(optimizer='adam',           loss='binary_crossentropy',
metrics=['accuracy'])
history_do = model_do.fit(x_train, y_train, epochs=100, verbose=0)
loss_do, acc_do = model_do.evaluate(x_test, y_test, verbose=0)
print(f"With Dropout — Loss: {loss_do:.3f}, Accuracy: {acc_do:.3f}")
```

**Possible Output:**

With Dropout — Loss: 0.045, Accuracy: 1.000

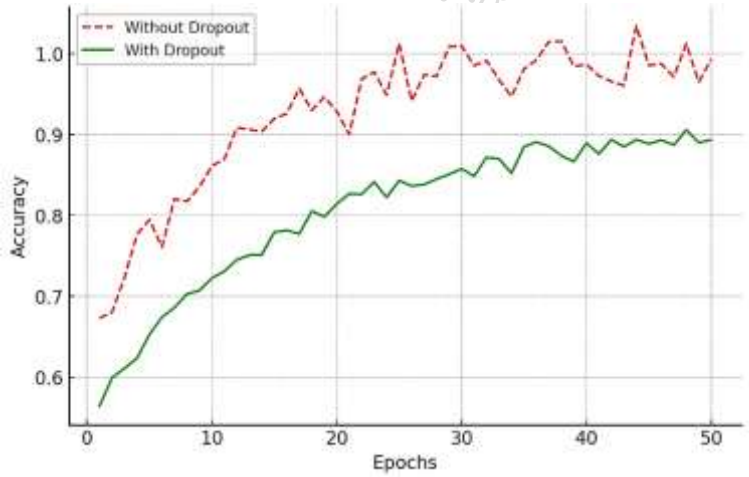



Fig. 6.4: Training accuracy with vs without dropout

Here's Figure 6.4, showing how dropout helps stabilize training accuracy compared to the fluctuating pattern without dropout.

### 6.8 Practical Activity

1. Load and split the dataset.
2. Train models with and without dropout.
3. Compile and fit on training data.
4. Evaluate on test data.
5. Plot accuracy comparisons.
6. Display confusion matrix, precision, recall.

## Summary

- Dense layers connect all neurons between layers.
- Binary Crossentropy is suitable for binary classification.
- Adam optimizer helps with faster convergence.
- Confusion matrix, precision, recall give deeper insights than accuracy alone.
- Overfitting vs. Underfitting can be managed with Dropout.

### Check Your Progress

**A. Multiple Choice Questions**

1. Which activation function is used in the output layer for binary classification? (a) ReLU (b) Sigmoid (c) Tanh (d) Softmax
2. The dropout layer helps to: (a) Increase model size (b) Prevent overfitting (c) Improve activation function (d) Change optimizer
3. Precision measures: (a) Correct negative predictions (b) True positives among predicted positives (c) All accurate predictions (d) Error rate of predictions
4. Underfitting happens when: (a) Both training and test accuracy are high (b) Training accuracy is high but test accuracy is low (c) Both accuracies are low (d) Test accuracy is higher than training accuracy
5. Which loss function is correct for our example? (a) Mean Squared Error (b) Binary Crossentropy (c) Categorical Crossentropy (d) Hinge Loss

**B. Fill in the Blanks**

1. Dense layers are also called _____ layers.
2. The process of updating weights is managed by the _____.
3. In binary classification, the output activation is typically _____.
4. _____ shows actual vs. predicted counts in classification.
5. Adding dropout reduces _____.

**C. True or False**

1. A confusion matrix includes TP, FP, TN, FN.
2. Accuracy alone is always sufficient for evaluating a model.
3. Dropout randomly disables neurons during training.
4. Overfitting means the model is too simple.
5. Adam optimizer adapts the learning rate during training.

**D. Short Answer Questions**

1. Describe the difference between training and test datasets.
2. Explain precision and recall in your own words.
3. Why is binary_crossentropy suitable here instead of mean squared error?
4. How does dropout help prevent overfitting?
5. Suggest another real-life example where binary classification could apply.

# Module 5. AI Project

## Module Overview

In today's data-driven world, Artificial Intelligence (AI) has become a powerful to solved the complex problems and transforming industries. This Module on AI projects explores the development of intelligent systems that can perform tasks that usually require human cognitive abilities, like learning, reasoning, and decision-making.

The Module covers the complete lifecycle of an AI project. This includes defining the core problem, gathering data, choosing a model, training it, evaluating its performance, and deploying it for real-world applications. This systematic process is essential for successful AI development.

The Module will explore how AI projects differ from traditional software development. The project development process, project format and how to write the project report is explained here. This Module also includes one sample project, which will give an idea to the students how to develop the AI project.

Upon completion of this Module, you will have a comprehensive understanding of the entire AI project ecosystem. This will give you the knowledge and skills needed to begin your own AI development and contribute to the rapidly evolving field of Artificial Intelligence.



## Learning Outcome

After completing this module, you will be able to:

- Master the five key stages: Scoping, Acquisition, Exploration, Modelling, and Evaluation.
- Define project goals clearly using the 4Ws framework (Who, What, Where, Why).
- Apply data exploration techniques to identify patterns and trends.
- Select and implement the correct AI model type based on the defined problem.
- Draft a professional AI Project Report to document the development process.
- Communicate technical findings and model performance to stakeholders.

## Module Structure

Session 1. AI Project Cycle

Session 2. Project Work

Session 3. Sample Project & AI Project Report

# Session 1. AI Project Cycle

## 1.1 Introduction

The AI project cycle enables organizations and individuals to resolve problems while developing an AI project. This step-by-step process is also known as AI development life cycle, providing a structured strategy in order to accomplish goals. The AI project life cycle is a structured method for developing and deploying Artificial Intelligence solutions. AI projects largely depend on data, iterative improvements, and may require an experimental approach. This life cycle helps ensure AI solutions are robust and provide ongoing value.

The AI Project Cycle is a structured process involving problem definition, data collection, model development, evaluation, deployment, and monitoring to build and manage effective artificial intelligence- AI solutions.

## 1.2 Stages of the AI Project Cycle

There are five stages of the AI project cycle. Let's take a look at each stage one by one.

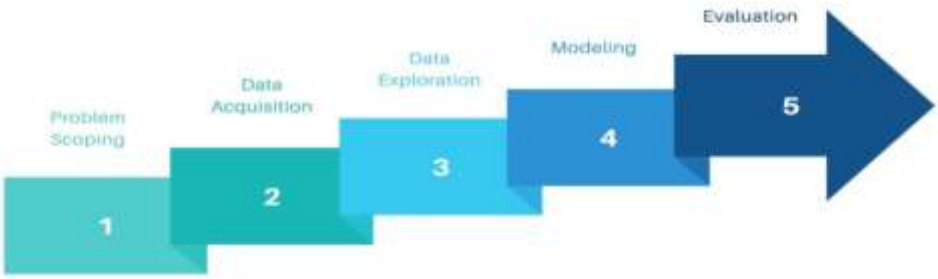



Fig. 1.1: Stages of AI Project Cycle

### 1.2.1. Problem Scoping: Understanding Problems and Their Solutions

Problems exist everywhere. They may be small or big, easy or critical. But every problem has a solution, and to find it, the problem must first be solved.

The main goal of any project is to find a solution to a problem. To solve a problem, the first step is to analyze it carefully. This means identifying the scope of the problem and having a clear vision to solve it.

Problem Scoping is the process of finding out the scope of a problem. It highlights an issue or concern that needs to be addressed with Artificial Intelligence. It also involves setting clear objectives and strategies. To do this successfully, one needs a strong understanding of the specific issue.

**The 4WS Problem Canvas**

The 4Ws Problem Canvas is a tool that helps in understanding a problem better. It focuses on four questions: Who, What, Where, and Why.


Fig. 1.2 4WS Problem Canvas

**1. Who**

This step identifies the people affected by the problem and those who will benefit from the solution. These people are called stakeholders.
Questions to ask:

(i) Who are the stakeholders?

(ii) What do you know about them?

**2. What**

This step defines the problem clearly. You need to study the problem thoroughly and gather proof that it really exists such as, news articles, reports, announcements.
Questions to ask:

(i) What is the problem?

(ii) How do you know it is a problem? What is the evidence?

**3. Where**

This step looks at the context, situation, and location where the problem occurs. Understanding the pattern helps in better analysis.
Questions to ask:

(i) In what context or situation do stakeholders face the problem?

(ii) Where is the problem located?

**4. Why**

This step explains the value of solving the problem. It focuses on how the solution will help stakeholders and society.
Questions to ask:

(i) Why will this solution be valuable to stakeholders?

(ii) How will the solution improve their situation?

### 1.2.2 Data Acquisition

The second step in the project process is all about collecting information which is required for the project. One must train the AI system with accurate data to enable it to make predictions.

Let's say, one wants to invent a system which is capable of predicting an employee's future earnings. In order to achieve this, one must also have the historical salary information of that particular employee. We call this past salary data 'training data', while the data used for making future predictions is known as 'testing data'.

The specific details you want to collect are called data features. In our example, these could include the employee's salary, the percentage increase they got, the time between raises, any bonuses, and so on. There are different ways to gather this data, like:


Fig. 1.3: Different ways to gather this data


### 1.2.3. Data Exploration

Data can be tricky and results in confusion, especially data involving numbers. The data visualization can be used to observe the complicated information. It converts the numbers into visuals which are easy to comprehend. To present the data more user-friendly manner it is presented in visuals like bar graphs, histograms, pie and line charts.

Fig. 1.4: Data visualization

This enables one to:

1. **Select the Right Tools** - Visualization assists in finding the right model or method works for analysis.
2. **Detecting Trends** - Visuals like graphs or charts are useful to display if something is going up, down, or staying the same, helping you identify trends.
3. **Share Findings** - Once you get insights, visuals make it easier to explain your findings to others.

### 1.2.4. Modeling

Modeling is an important element in the AI project process to simplify complicated data for computers. This enables the computer to understand and come up with the right predictions. Initially, data might be shown in charts or graphs for pattern spotting. For AI systems to function, we need to turn this data into a format that computers can work with, which usually means converting it into binary (0s and 1s).

Modeling is about creating a mathematical framework that shows how different data points relate to one another. It's similar to how computers are taught to recognize patterns or make choices. These models can range from simple equations to complex neural networks, depending on what you need to accomplish. The system can make predictions with new data once it learns the rules from historical data.

Generally, AI models can be classified as follows:

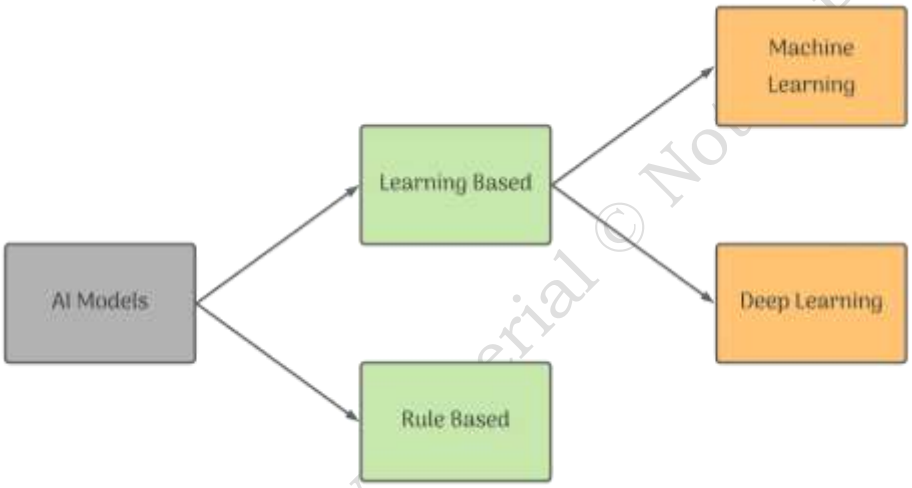



Fig. 1.5: Classification of AI Models

### 1.2.5. Evaluation

Evaluation is the final stage of the AI project process. After creating and training a model, it's essential to test it thoroughly to see how well it performs. For this, we use a separate dataset called testing data. We evaluate the model's performance based on several criteria:

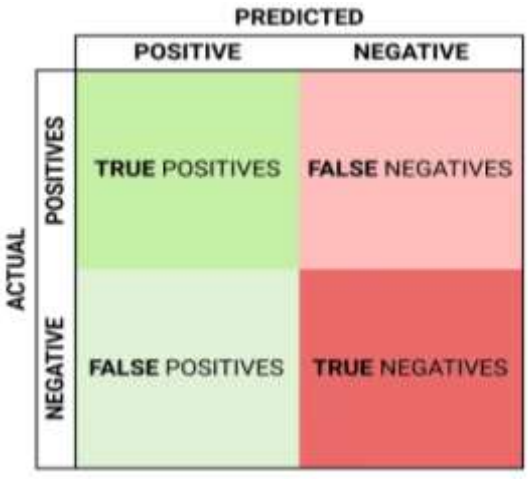



Fig. 1.6: Criteria foe evaluation of model's performance

- **Accuracy** - This calculates the percentage of correct predictions out of the total dataset and provides an entire image of the model's accuracy.
- **Precision** -It measures how many of its positive predictions were correct.
- **F1 Score** - This metric combines precision and recall, which is useful when one class is heavily represented over the other. This enables to bring a balance between false positives and negatives.
- **Recall** - Also known as sensitivity, this metric shows how well the model correctly identifies all positive instances. This indicates its capability to capture positive cases.

## Summary

The AI Project Cycle defines the structured framework for developing AI solutions. It comprises five essential stages: Problem Scoping, Data Acquisition, Data Exploration, Modelling, and Evaluation. This iterative process ensures that AI projects are goal-oriented, data-driven, and systematically tested to solve real-world problems effectively and ethically.

## Check your progress

**A. Multiple choice questions**

1. Which of the following is typically the first stage in the AI Project Cycle? (a) Data Acquisition (b) Model Evaluation (c) Problem Scoping (d) Model Deployment
2. During the Problem Scoping phase, what tool is often used to break down the problem by considering its key elements? (a) Data Visualization Chart (b) System Map (c) 4Ws Problem Canvas (d) Neural Network
3. Which phase focuses on collecting and preparing a high-quality dataset for training AI models? (a) Data Exploration (b) Data Acquisition and Preparation (c) Model Training (d) Problem Scoping
4. What is the purpose of Data Exploration in the AI Project Cycle? (a) To define the problem statement (b) To train the AI model (c) To understand data characteristics, identify patterns, and detect anomalies (d) To deploy the AI model
5. Which of the following is NOT a typical type of learning-based approach in AI modeling? (a) Supervised Learning (b) Unsupervised Learning (c) Reinforcement Learning (d) Rule-based Learning
6. The phase where the trained model's performance is assessed using unseen data is known as: (a) Data Acquisition (b) Model Training (c) Model Evaluation (d) Problem Scoping
7. Why is Continuous Monitoring and Maintenance essential in AI projects? (a) It ensures the model is deployed quickly. (b) It helps detect model drift and maintain performance over time. (c) It is only required for image recognition tasks. (d) It's primarily for initial model training.

8. What does "model drift" refer to in the context of AI project monitoring? (a) The model is becoming too complex. (b) The model is being updated with new features. (c) The model's performance degrades over time due to changes in real-world data. (d) The model is moving to a different server.
9. Which of these is a common metric used to evaluate AI models, particularly for classification tasks? (a) Mean Squared Error (MSE) (b) Accuracy (c) Computational Cost (d) Latency
10. The AI project lifecycle is described as an iterative process. What does this mean? (a) Each stage is completed only once. (b) The process is linear and doesn't allow revisiting steps. (c) Steps are often revisited throughout the process to refine the model and improve outcomes. (d) Only the deployment phase is iterative.

**B. Fill in the blanks**

1. The first stage of the AI project cycle is called _____ .
2. During _____ , the business problem, project objectives, and success metrics are defined.
3. The _____ is a tool often used in the initial stage to break down the problem by considering who, what, where, and why.
4. In the _____ phase, data is collected from various sources to form the base of the project.
5. _____ involves handling missing values, inconsistencies, and errors in the collected data.
6. The phase of representing data graphically to identify trends and patterns is called _____ .
7. _____ involves selecting the appropriate AI model and fitting it to the prepared data.
8. _____ is the process of testing the trained model's performance on unseen data.
9. _____ is the stage where the AI model is integrated into a production environment.
10. Continuous _____ helps detect model drift and maintain performance over time.
11. The AI project cycle is inherently an _____ process, allowing for revisiting steps to refine the mode

**C. State whether True or False**

1. The AI project cycle is a linear process where each stage is completed only once.
2. Problem Scoping is the first stage of the AI Project Cycle.
3. Data Acquisition primarily focuses on training the AI model with algorithms.
4. Model Deployment is the final and absolute endpoint of an AI project, with no further steps.

5. The 4Ws Problem Canvas is a tool used during the Data Exploration phase.
6. Training data and testing data can be the same for evaluating an AI model.
7. Model evaluation helps in identifying potential biases in the AI model.
8. Model drift refers to the model becoming more efficient over time due to real-world data exposure.
9. Deep learning is a type of rule-based AI model.
10. Data quality and quantity have no significant impact on the performance of an AI model.

## Session 2. Project Work

### 2.1 Introduction

Project work is crucial for individual development. It provides a hands-on opportunity to apply learned concepts in a real-world setting. Projects cultivate a wide range of essential skills, including technical expertise in programming, machine learning algorithms, and data analysis techniques. A student passing out any course should complete at least one project. This will help them to know about the subject well and give the practical exposure.

### 2.2 About the Project Work

The project carries certain Credit Points and is graded in the examination. The project is guided by school teacher as Internal Guide as well expert from industry or business world as External Guide. The student has to report the school teacher to show the progress of the project. At the end of the project, students must prepare a document of their work in the form of a Project Report.

**Objectives of Project Work**

The ultimate objectives for the project work can be stated as:

- To train students to independently formulate and solve the problems by using AI and present the results in both written and oral form.
- To expose students to real-life problems in the World of Work.
- To provide students with opportunities to interact with people and understand human relations.

### 2.3 Project Development Process

Students are supposed to complete their project work within a period of ---months. Development process contain following points.

Developing an Artificial Intelligence (AI) project requires a structured approach. The AI project development process outlines the stages involved from conceptualization to deployment as follows.

#### 2.3.1. Problem definition and goal setting

- Clearly identify the problem or opportunity that the AI project aims to address.
- Define specific, measurable, achievable, relevant, and time-bound (SMART) objectives for the AI system.

- Align the AI project goals with broader objectives and stakeholder expectations.
- Use tools like the "4Ws problem canvas" to gain a thorough understanding of the problem by analyzing who is affected, what the problem is, where and when it occurs, and why it's important to solve.
- Establish success criteria and key performance indicators (KPIs).

**2.3.2. Data collection and exploration**

- Identify the necessary data to train and evaluate the AI model.
- Gather data from various sources, such as databases, surveys, web scraping, sensors, cameras, and APIs.
- Ensure data quality by handling missing values, inconsistencies, and noise.
- Explore and understand the data's characteristics, patterns, and potential issues using techniques like data visualization and statistical analysis.
- Split data into training, validation, and test sets.
- Address challenges related to data quality, quantity, comply with ethical and legal standards, including data privacy regulations

**2.3.3. Data preparation and preprocessing**

- Clean the data by removing errors, inconsistencies, duplicates, and irrelevant information.
- Preprocess the data for model training through tasks like data normalization, scaling, and handling missing values.
- Perform feature engineering, which involves selecting, creating, or transforming data features to enhance model performance.
- Label or annotate the data as needed, particularly for supervised learning projects.

**2.3.4. Model design and development**

- Select the appropriate AI algorithms and architectures based on the problem domain, data characteristics, and desired outcomes.
- Explore and choose from various model types, such as machine learning (ML), deep learning (DL), or natural language processing (NLP) models, among others.
- Design and build the AI model, potentially exploring pre-existing models or training one from scratch.
- Consider factors like computational resources, model explainability, and scalability during model design.

**2.3.5. Model training and testing**

- Train the model using the prepared training data.
- Adjust model parameters and fine-tune hyperparameters to optimize performance.
- Rigorously test the model's performance and accuracy on separate validation and test datasets.

- Evaluate model performance using suitable metrics (accuracy, precision, recall, F1-score, etc.).
- Perform cross-validation to prevent overfitting.
- Compare model results with baseline methods.

**2.3.6. Deployment and integration**

- Select an appropriate deployment platform like cloud hosting, edge deployments, or hybrid approaches.
- Deploy the trained model into a production environment where it can solve the identified problem.
- Integrate the model with existing applications, platforms. Use APIs for integration with existing applications.
- Focus on scalability, security, and compliance during deployment

**2.3.7. Performance Monitoring and Maintenance**

- Continuously monitor model accuracy and update as needed.
- Set up automated alerts for model drift detection.
- Regularly retrain the model with new data.
- Maintain logs and documentation for reproducibility.

**2.3.8. Ethical Considerations and Compliance**

- Ensure fairness and eliminate bias in the AI model.
- Adhere to AI governance policies and ethical guidelines.
- Maintain transparency by documenting decision-making processes.
- Ensure AI-driven decisions align with societal and business ethics.

**2.3.9. Project Documentation and Reporting**

- Maintain thorough documentation, including methodology, code, and results.
- Provide periodic reports to stakeholders on project progress.
- Create user guides and technical documentation for end-users.
- Store source code and datasets securely for future reference.

**2.3.10. Collaboration and Team Coordination**

- Assign clear roles and responsibilities to team members.
- Use project management tools like Jira, Trello, or Asana.
- Maintain version control with Git and conduct regular code reviews.
- Foster cross-functional collaboration between data scientists, engineers, and business teams.

**2.3.11. Scalability and Future Improvements**

- Design models with scalability in mind for future enhancements.
- Plan for system upgrades and performance optimization.
- Explore automation possibilities for reducing manual efforts.
- Keep track of emerging AI trends to incorporate innovations into the project.

By following these guidelines, AI projects can be executed efficiently, ensuring robustness, fairness, and long-term sustainability.

### 2.4 Project Formats

Project formats refer to the various ways project information and documentation can be structured, presented, and stored. The format for project documentation depends on the project's nature, the target group, and the purpose of the document.

Following sample synopsis is given for AI project. You can use it for preparation of synopsis at the beginning of project work.

1. **Title of the Project:** <Provide a concise and clear title for the project.>
2. **Introduction:** <A brief introduction to the project, its purpose, and its significance.>
3. **Objectives/ Existing System and Need for System**
   - <List the main objectives of the project, specifying what it aims to achieve.>
4. **Scope of the Project**
   - <Define the boundaries of the project, including functionalities and limitations.>
5. **Technologies Used**
   - <Outline the programming languages, frameworks, databases, and tools used.>
6. **System Architecture**
   - <A high-level description of the system design, including a block diagram if applicable.>
7. **Modules and Functionalities**
   - <Divide the project into key modules and briefly describe their roles and functionalities.>
   - Module 1: Description
   - Module 2: Description
   - Module 3: Description
8. **Methodology**
   - <Describe the software development model used, e.g., Agile, Waterfall, etc.>
9. **Expected Outcome**
   - <Explain the expected results and benefits of the project.>
10. **Conclusion**
    - <A summary of the project's importance and impact.>
11. **References (if any)**
    - <List of books, websites, or papers referred to in the project.>



### 2.5 Project Review

A project review is a formal assessment of a project's progress, performance, and outcomes at a specific point in its lifecycle. It is a process that helps to determine if the project is on track to achieve its goals and objectives and identify areas that need improvement.

Following Project Review Chart can be used for Evaluation of Project.

| Project Stage | Evaluation Criteria | Rating (1-5) | Comments |
|---|---|---|---|
| Project Definition | Clarity of problem statement and objectives | | |
| | Feasibility and relevance of the project | | |
| Research and Background | Adequacy of literature review | | |
| | Comparison with existing solutions | | |
| Data Collection | Data sources identified and validated | | |
| | Data preprocessing and cleaning | | |
| Methodology | Appropriateness of ML/AI models used | | |
| | Justification for chosen algorithms | | |
| Implementation | Accuracy of model execution and training | | |
| | Efficiency and optimization of the solution | | |
| Evaluation & Testing | Performance metrics assessment | | |
| | Model validation and cross-validation | | |
| Deployment | Deployment strategy (cloud, edge, local) | | |
| | Scalability and usability of the solution | | |
| Results & Analysis | Presentation and visualization of results | | |
| | Interpretation and discussion of findings | | |
| Ethical Considerations | Bias mitigation strategies | | |
| | Compliance with regulations | | |
| Documentation | Clarity and completeness of project documentation | | |
| | References and citations included | | |
| Overall Evaluation | Project innovation and uniqueness | | |

**Final Score:** _____/-----

This review chart provides a structured approach to assessing each phase of an AI project. Ratings (1-5) help gauge the project's quality and effectiveness.

## Summary

Project Work involves the practical application of the AI cycle to build functional models. It focuses on data exploration to uncover hidden patterns and model selection to ensure the algorithm aligns with the project goals. This hands-on phase bridges theoretical concepts with real-world technical execution and team collaboration.

## Check your progress

**A. Multiple choice questions**

1. Which phase in the project development process involves identifying the project's goals, scope, and feasibility? (a) Planning (b) Execution (c) Initiation (d) Closure
2. What is the primary objective of the Monitoring and Controlling phase? (a) To close the project and archive documents (b) To define the project's scope and objectives (c) To track project progress, manage deviations, and ensure quality (d) To develop the project team and facilitate communication
3. Which of the following is considered a key best practice in project management? (a) Limiting communication to formal reports only (b) Avoiding stakeholder involvement until project completion (c) Documenting everything related to the project (d) Ignoring the possibility of project setbacks
4. Which of the following is NOT a typical method for collecting data in AI projects? (a) Surveys and Questionnaires (b) Observational Studies (c) Model Training (d) Web Scraping
5. What is the main objective of data exploration in the AI project cycle? (a) To define the problem statement (b) To train the AI model (c) To understand data characteristics, identify patterns, and detect anomalies (d) To deploy the AI model
6. Which of these is a key challenge in data collection for AI? (a) Automating the process (b) Ensuring data quality and avoiding bias (c) Finding enough data (d) Integrating with other AI tools
7. What does "feature engineering" aim to achieve? (a) To automate data collection (b) To create new or modified features to enhance model performance (c) To visualize data patterns using charts (d) To evaluate the model's accuracy
8. hich tool is commonly used for data visualization in AI projects? (a) TensorFlow (b) Tableau (c) Apache Spark (d) Scikit-learn
9. Which of the following is the primary goal of data preprocessing? (a) To visualize data for better understanding (b) To select the right machine learning algorithm (c) To prepare raw data into a suitable format for machine learning algorithms (d) To deploy the trained model
10. What is the primary purpose of a project review? (a) To allocate resources to the project (b) To evaluate the success of the project (c) To determine the project manager's performance (d) To generate new project ideas

**B. Fill in the blanks**

1. _____ is a technique for automatically extracting data from websites.
2. _____ is the graphical representation of information, using charts, graphs, and maps.
3. The _____ framework is a popular approach for creating well-defined and achievable goals.
4. The process of identifying and correcting errors, inconsistencies, and missing values in the data is called _____
5. The process of tuning a model's settings to optimize its performance is called _____
6. A project review is a formal assessment of project's _____, _____ , and _____at a specific point in its lifecycle.
7. Feature engineering enhance _____ e
8. Fine-tuning the _____ to optimize performance.
9. Deploying the trained model on _____ solve the identified problem.
10. AI model is trained using the _____.

**C. State whether the following statement is True or False**

1. Data collection is only relevant for large-scale AI projects.
2. Data visualization is not an important part of data exploration.
3. APIs are only used for collecting real-time data.
4. Poor data quality, even after preprocessing, can negatively impact the performance of AI models.
5. Data preprocessing is an optional step in the AI project cycle.
6. Model parameters are manually set by the data scientist before training, while hyperparameters are learned during training.
7. Project review is a process to check whether people are working or not?
8. Effective project reviews contribute to continuous improvement by helping organizations refine their project management strategies and practices.
9. The format for project documentation depends on the project's nature,
10. Final score of evaluation of project is based on overall impression of project.



## Session 3. Sample of AI Project & Report Summary

A sample project report of AI Project is presented below.

**1. Title Page**

**Project Title:** Sentiment Analysis on Customer Reviews

**Author(s) / Team Members:** X, Y

**Institution / Organization:** XYZ University

**Date of Submission:** March 7, 2025

**2. Abstract**

This project aims to develop a sentiment analysis model to classify customer reviews as positive, negative, or neutral. Using Natural Language Processing (NLP) techniques and machine learning, we preprocess and analyze text data to extract meaningful insights. The project employs a logistic regression model and deep learning techniques for sentiment classification, achieving an accuracy of 85%. The results can help businesses understand customer opinions and improve their services.

**3. Introduction**

**Background Information**

Customer reviews are a valuable source of feedback for businesses. Analyzing these reviews manually is time-consuming and inefficient. Sentiment analysis, an NLP technique, automates this process by categorizing opinions into sentiment classes.

**Problem Statement**

Existing sentiment analysis systems often struggle with contextual understanding and sarcasm, leading to misclassification. Our project aims to improve sentiment classification accuracy using machine learning.

**Objectives**

- Develop a machine learning model for sentiment classification.
- Improve accuracy through preprocessing and model optimization.
- Provide a user-friendly interface for sentiment analysis.

**4. Literature Review**

**Existing Research**

Several approaches exist for sentiment analysis, including lexicon-based methods and machine learning models like Naive Bayes, Support Vector Machines, and deep learning models.

**Comparison of Approaches**

- **Lexicon-based methods**: Depend on predefined word lists but lack context understanding.
- **Machine learning models**: Require labeled data but generalize well.
- **Deep learning models**: Capture complex patterns but need large datasets.

**Justification**

We chose logistic regression for baseline performance and a deep learning model for improved accuracy due to its ability to understand complex linguistic patterns.

**5. Methodology**

**Data Collection**

- Dataset: IMDB and Amazon customer reviews.
- Preprocessing: Tokenization, stopword removal, stemming, and vectorization using TF-IDF.

**Machine Learning Models**

- Logistic Regression for baseline.
- LSTM (Long Short-Term Memory) neural network for deep learning.

**Tools and Frameworks**

- Python, Scikit-learn, TensorFlow, NLTK, Pandas, Matplotlib.

**Training and Evaluation**

- Split dataset into training (80%) and testing (20%).
- Evaluation metrics: Accuracy, Precision, Recall, F1-score.

**6. ImplementationSteps**

1. ***Data preprocessing:*** Cleaning and transforming text data.
2. ***Feature extraction:*** Converting text into numerical representation.
3. ***Model training:*** Training both logistic regression and LSTM models.
4. ***Model evaluation:*** Measuring performance using metrics.

**System Architecture**

- ***Input:*** Customer reviews.
- ***Processing:*** NLP preprocessing and model prediction.
- ***Output:*** Sentiment classification (Positive/Negative/Neutral).

**Code Implementation**

```
import pandas as pd
import numpy as np
import re
import nltk
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense,
SpatialDropout1D
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Load dataset
df = pd.read_csv('customer_reviews.csv')

# Preprocessing
def clean_text(text):
```

```
    text = re.sub(r'[^a-zA-Z]', ' ', text)
    text = text.lower()
    text = text.split()
    text = [word for word in text if word not in
stopwords.words('english')]
    return ' '.join(text)
df['cleaned_reviews'] = df['review'].apply(clean_text)

# TF-IDF Vectorization
vectorizer = TfidfVectorizer(max_features=5000)
X = vectorizer.fit_transform(df['cleaned_reviews']).toarray()
y = df['sentiment']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Logistic Regression Model
lr_model = LogisticRegression()
lr_model.fit(X_train, y_train)

# Evaluate Logistic Regression
y_pred = lr_model.predict(X_test)
print("Logistic Regression Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

# Deep Learning Model with LSTM
max_words = 5000
max_len = 200
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(df['cleaned_reviews'])
X_seq = tokenizer.texts_to_sequences(df['cleaned_reviews'])
X_pad = pad_sequences(X_seq, maxlen=max_len)

X_train_dl, X_test_dl, y_train_dl, y_test_dl = train_test_split(X_pad,
y, test_size=0.2, random_state=42)

model = Sequential()
model.add(Embedding(max_words, 128, input_length=max_len))
```

```
model.add(SpatialDropout1D(0.2))
model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(3, activation='softmax'))
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

model.fit(X_train_dl, y_train_dl, epochs=5, batch_size=64,
validation_data=(X_test_dl, y_test_dl))
```

**7. Results and Analysis**

**Performance Metrics**

| Model | Accuracy | Precision | Recall | F1-score |
|---|---|---|---|---|
| Logistic Regression | 78% | 76% | 74% | 75% |
| LSTM | 85% | 83% | 82% | 83% |

**Visualizations**

- Confusion matrices for both models.
- Accuracy and loss curves for LSTM model.

**8. Discussion**

**Interpretation of Results**

- The LSTM model performed better than logistic regression.
- Challenges included handling sarcasm and ambiguous sentiments.

**Potential Improvements**

- Incorporating transformer-based models like BERT for better contextual understanding.
- Expanding the dataset to include multiple domains.

**9. Conclusion**

This project successfully implemented sentiment analysis using machine learning and deep learning. The LSTM model achieved an 85% accuracy, demonstrating its potential for real-world applications in customer feedback analysis.

**10. References**


- Jurafsky, D., & Martin, J. H. (2021). Speech and Language Processing.
- Pang, B., & Lee, L. (2008). Opinion Mining and Sentiment Analysis.
- Research papers on sentiment analysis techniques.


**11. Appendices**

- Sample code for data preprocessing and model training.
- Additional graphs and visualizations.
- Hyperparameter tuning details for deep learning models.

# Sample AI Project Report

**1. Title of the Project: AI-Based Smart Waste Management System**

**2. Introduction:**

Waste management is a major challenge in urban and rural areas of India. Overflowing garbage bins, inefficient collection, and lack of segregation lead to unhygienic conditions and environmental hazards. This project proposes an AI-based smart waste management system that can monitor waste levels in bins, classify waste types, and optimize collection routes. The purpose of this project is to make waste management efficient, cost-effective, and sustainable, while promoting cleanliness and public health.

**3. Objectives / Existing System and Need for System**

1) Existing System: At present, most waste collection follows a fixed schedule, regardless of whether bins are full or not. Manual segregation is time-consuming and inefficient, leading to improper disposal.
2) Need for System: There is a need for a smart, AI-driven approach to monitor bins, separate waste, and plan optimized collection routes.

- Objectives:
    1. To monitor waste levels in bins using sensors or cameras.
    2. To classify waste (biodegradable, recyclable, plastic, etc.) using image recognition.
    3. To optimize garbage truck routes using AI algorithms.
    4. To reduce operational costs, save time, and promote recycling.

**4. Scope of the Project:**

- Functionalities:
    - Real-time monitoring of garbage bins.
    - Image-based waste classification.
    - Automatic alerts to municipal authorities when bins are full.
    - Route optimization for waste collection vehicles.
- Limitations:
    - Initial cost of installing smart bins may be high.
    - Requires internet/GPS connectivity.
    - Accuracy depends on dataset quality for waste classification.

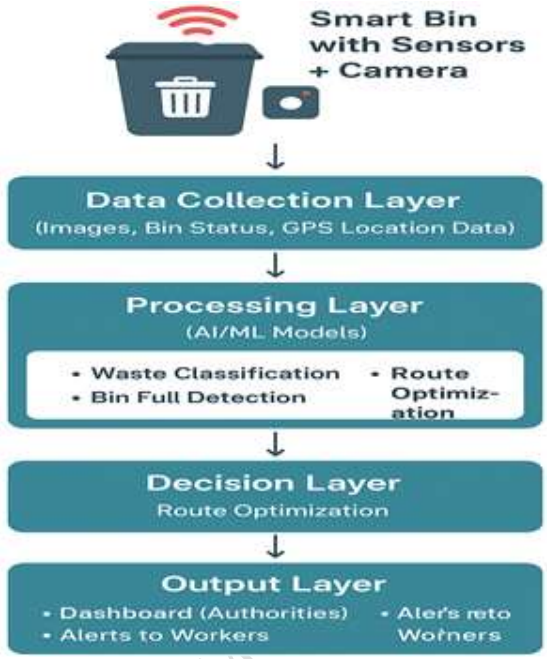
**5. Technologies Used:**

- Programming Languages: Python, Java
- Frameworks: TensorFlow / PyTorch (for AI models), Flask/Django (for web dashboard)
- Databases: MySQL / MongoDB
- Tools: OpenCV (for image processing), GPS/GIS tools, Arduino/IoT sensors
- Others: Cloud platform (AWS / Google Cloud) for data storage and AI model deployment

**6. System Architecture:**

High-level design:

1. Input Layer – Sensors & cameras on bins collect data.
2. Processing Layer – AI models classify waste and analyze bin status.
3. Decision Layer – Route optimization algorithm schedules garbage truck routes.
4. Output Layer – Dashboard + Alerts sent to municipal authorities.




Block Diagram of System Architecture

**7. Modules and Functionalities:**

- Module 1: Waste Monitoring Module
  - Uses sensors and cameras to check bin status (empty, half-full, full).
  - Sends real-time data to the server.
- Module 2: Waste Classification Module
  - AI model processes images of waste.
  - Categorizes waste into biodegradable, recyclable, and non-recyclable.
- Module 3: Route Optimization Module
  - Uses AI algorithms to generate shortest and most efficient routes for collection trucks.
  - Reduces fuel consumption and collection time.
- Module 4: Dashboard & Alerts Module
  - Provides a web dashboard for authorities.
  - Sends notifications when bins are full.

**8. Methodology:**

The project follows the Agile Methodology –

- Iterative development in small sprints.

- Regular testing of AI models with updated datasets.
- Continuous feedback and improvements.

**9. Expected Outcome:**

- Smart bins that alert authorities when full.
- Automated waste classification using AI.
- Optimized waste collection routes, reducing cost and time.
- Cleaner cities, healthier environment, and better recycling practices.

**10. Conclusion:**

This project demonstrates how Artificial Intelligence can be applied to solve real-world problems like waste management. By combining IoT, AI, and route optimization, this system can transform urban cleanliness, reduce costs, and improve sustainability. It has the potential to be adopted by municipal corporations and NGOs across India.

**11. References:**

1. Government of India – Swachh Bharat

**Tensor Flow Installation**

Using TensorFlow in Anaconda is straightforward once you set up the right environment. Here's a step-by-step guide:

**Step 1. Open Anaconda Prompt**

- Search for Anaconda Prompt in your start menu (Windows) or terminal (Mac/Linux).
- We'll create a new environment to keep things clean.

**Step 2. Create a New Environment**

conda create -n tf_env python=3.9

TensorFlow works best with Python 3.9 or 3.10 (avoid the latest versions unless officially supported).

**Step 3. Activate the Environment**

conda activate tf_env

**Step 4. Install TensorFlow**

You have two choices:

Option A – Install CPU version (lighter, no GPU needed)

pip install tensorflow

Option B – Install GPU version (if you have NVIDIA GPU + CUDA)

pip install tensorflow-gpu

Make sure you have compatible CUDA and cuDNN installed if using GPU.

**Step 5. Verify Installation**

Run Python inside your environment:

```
python
```

Then type:

```
import tensorflow as tf
print("TensorFlow version:", tf.__version__)
```

If you see a version number (e.g., 2.20.0), TensorFlow is installed correctly.

**Python Code: Waste Classification using CNN (TensorFlow/Keras)**

```
# AI Project: Waste Classification (Smart Waste Management System)

import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense, Dropout
import matplotlib.pyplot as plt

# ----------------------------
# Step 1: Data Preparation
# ----------------------------
# Assume dataset is structured like:
# dataset/
#    train/
#       biodegradable/
#       recyclable/
#       non_recyclable/
#    test/
#       biodegradable/
#       recyclable/
#       non_recyclable/

train_datagen = ImageDataGenerator(rescale=1./255, shear_range=0.2,
zoom_range=0.2, horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1./255)

train_data = train_datagen.flow_from_directory(
    'dataset/train',
    target_size=(64, 64),
    batch_size=32,
    class_mode='categorical'
)

test_data = test_datagen.flow_from_directory(
    'dataset/test',
    target_size=(64, 64),
    batch_size=32,
    class_mode='categorical'
)
```

```
# ----------------------------
# Step 2: Build CNN Model
# ----------------------------
model = Sequential()

model.add(Conv2D(32, (3,3), activation='relu', input_shape=(64, 64, 3)))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(64, (3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())
model.add(Dense(units=128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(units=3, activation='softmax'))  # 3 categories: biode-
gradable, recyclable, non_recyclable

# ----------------------------
# Step 3: Compile Model
# ----------------------------
model.compile(optimizer='adam', loss='categorical_crossentropy', met-
rics=['accuracy'])

# ----------------------------
# Step 4: Train Model
# ----------------------------
history = model.fit(
    train_data,
    steps_per_epoch=len(train_data),
    epochs=10,
    validation_data=test_data,
    validation_steps=len(test_data)
)

# ----------------------------
# Step 5: Evaluate Model
# ----------------------------
acc = model.evaluate(test_data)
print(f"Test Accuracy: {acc[1]*100:.2f}%")

# ----------------------------
# Step 6: Save Model
# ----------------------------
```

```
model.save("waste_classifier_model.h5")

# ----------------------------
# Step 7: Plot Accuracy & Loss
# ----------------------------
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend()
plt.show()
```

**Sample Output of Project**

**1. AI Waste Classification Model Results**

- Training Accuracy: 92.5%
- Validation Accuracy: 88.3%
- Test Accuracy: 89.7%

**Sample Predictions:**

| Input Image (Waste Item) | Predicted Category | Confidence Score |
|---|---|---|
| Banana Peel | Biodegradable | 96% |
| Newspaper | Recyclable | 91% |
| Chocolate Wrapper | Non-Recyclable | 87% |
| Apple Core | Biodegradable | 94% |
| Plastic Bottle | Recyclable | 89% |

**2. Smart Bin Sensor Output**

- Bin 1 (Location: Shivaji Nagar, Pune): 80% Full – Alert Generated
- Bin 2 (Location: Nashik Road): 45% Full – Normal
- Bin 3 (Location: Mumbai CST): 95% Full – Immediate Action Required

**3. Dashboard (For Authorities)**

Features shown in demo output:

- Live map with bin locations (green = empty, yellow = half, red = full).
- Graph of daily waste collection (segregated into biodegradable, recyclable, non-recyclable).
- Notifications panel:
  - "Bin #3 at Mumbai CST is overflowing. Route updated for nearest garbage truck."
  - "Bin #1 at Shivaji Nagar reached 80% capacity. Collection scheduled."

**4. Route Optimization Output**

- Truck an Assigned Route: Bin 3 → Bin 1 → Bin 5
- Estimated Time Saved: 25 minutes

- Fuel Saving: 12%

**5. Expected Social Impact (Sample Report Output)**

- Segregated waste collection improved by 35%.
- Reduction in overflowing bins complaints by 40%.
- Data insights available for better urban planning.

**Mock Screenshots**


## Summary

This session focuses on documenting the AI journey through a formal Project Report. It guides students in presenting their methodology, from problem scoping to evaluation metrics, ensuring technical clarity and accountability. By analyzing a sample project, learners understand how to communicate findings and suggest future model improvements.

## Check Your Progress

**A. Multiple Choice Questions (MCQs)**

1. Which technique is mainly used in the Sentiment Analysis project to process text data? (a) Computer Vision (b) Natural Language Processing (NLP) (c) Robotics (d) Reinforcement Learning
2. Which machine learning model was used as a baseline model in the sentiment analysis project? (a) Decision Tree (b) Naive Bayes (c) Logistic Regression (d) Random Forest
3. In the Smart Waste Management System, which technology is used for waste classification? (a) Speech Recognition (b) Image Recognition (c) Text Mining (d) Web

Scraping

4. Which framework is used to build the CNN model for waste classification? (a) Scikit-learn (b) OpenCV (c) TensorFlow / Keras (d) NumPy
5. What is the main purpose of route optimization in the waste management system? (a) Increase garbage production (b) Improve image quality (c) Increase number of trucks (d) Reduce fuel and time consumption

**B. Fill in the Blanks**

1. Sentiment analysis classifies customer reviews into _____, _____, or _____ categories.
2. TF-IDF is used for _____ extraction in text data.
3. LSTM is a type of _____ neural network.
4. The smart waste management system uses _____ and _____ to monitor garbage bins.
5. The loss function used in the CNN waste classification model is _____.

**C. State Whether True or False**

1. Sentiment analysis is done manually without the help of AI.
2. Deep learning models require large datasets to perform well.
3. Logistic Regression performed better than LSTM in sentiment analysis.
4. Route optimization increases fuel consumption of garbage trucks.
5. Smart waste management helps in reducing operational costs.

**D. Answer the Following Questions (Short Answers)**

1. What is sentiment analysis?
2. Why is preprocessing important in NLP?
3. Name any two tools used in the sentiment analysis project.
4. What is the role of CNN in the smart waste management system?
5. Mention one social benefit of the AI-based smart waste management system.

# Glossary

**Aggregate Functions** – Functions that calculate results over a set of array elements (e.g., sum, mean, product, standard deviation).

**Array** – A collection of elements of the same type stored in continuous memory, accessible by an index.

**Axis** – A direction or dimension of an array. For example, rows are axis 0 and columns are axis 1 in a 2D array.

**Broadcasting** – A feature in NumPy that allows arrays of different shapes to be combined and used together in operations without making extra copies.

**Column Stack (**np.column_stack()**)** – Function to combine 1D arrays as columns into a 2D array.

**Concatenation** – Joining two or more arrays into one along a specified axis.

**Copy** – A separate duplicate of an array with its own data; changes in the copy do not affect the original array.

**Cumulative Functions** – Functions that provide a running total or product, such as cumulative sum or cumulative product.

**Data Type (dtype)** – The type of data elements stored in a NumPy array (e.g., int32, float64).

**Dimension** – The number of directions/axes in an array (1D = line, 2D = table, 3D = cube).

**Element-wise Operation** – An operation applied individually to each element in an array (e.g., multiplying every element by 2).

**Flattening** – Converting a multi-dimensional array into a single one-dimensional array.

**Indexing** – Accessing array elements by their position (index), starting from 0 in NumPy.

**Matrix** – A 2D array of numbers arranged in rows and columns, used for mathematical operations.

**ndarray** – The core data structure in NumPy; represents an n-dimensional array.

**Random Array** – An array created using random numbers (e.g., np.random.rand() or np.random.randint()).

**Reshape** – Changing the shape (rows, columns, dimensions) of an array without altering its data.

**Resize** – Changing the size of an array, which may add or remove elements.

**Slicing** – Extracting a portion of an array using index ranges (start:stop:step).

**Splitting** – Dividing one array into multiple sub-arrays (e.g., np.split(), np.hsplit()).

**Stacking** – Combining arrays in different ways: row-wise (vertical), column-wise (horizontal), or depth-wise (3D).

**Transpose** – Flipping a 2D array by turning rows into columns and columns into rows.

**Vectorization** – Performing operations on whole arrays at once instead of using loops, making computations faster.

**View** – A new array object that shares data with the original array; changes in one affect the other.

**Aggregation**: Combining multiple values into one, like finding the average marks of a class.

**Attribute**: A property or characteristic of an object, e.g., .shape tells the size of a DataFrame.

**Box Plot**: A chart that shows the spread of data using minimum, quartiles, median, and maximum values.

**CSV (Comma Separated Values)**: A simple file format where data is stored as text separated by commas, like an Excel sheet without formatting.

**Data Cleaning**: Process of correcting or removing wrong, incomplete, or duplicated data.

**DataFrame**: A 2D table in Pandas with rows and columns, like a spreadsheet.

**Filtering**: Selecting only the rows or columns of data that meet certain conditions.

**GroupBy**: A Pandas function that groups rows of data based on a column, e.g., sales grouped by region.

**Histogram**: A graph that shows the frequency of data within certain ranges (bins).

**Index**: Labels or positions used to identify rows or columns in Series or DataFrame.

**JSON (JavaScript Object Notation)**: A file format used to store and exchange data in key-value pairs, often used in web applications.

**Legend**: A box in a chart explaining the meaning of different colors, symbols, or lines.

**Matplotlib**: A Python library used for creating graphs and visualizations.

**Missing Values (NaN)**: Data that is not available or empty in a dataset.

**NumPy**: A Python library for fast numerical operations using arrays.

**Outlier**: A data point that is very different from other values, like a student scoring 100 when most scored 40–60.

**Panel Data**: Multidimensional structured data often used in economics and statistics.

**Pandas**: A Python library for handling and analyzing data in tables (Series and DataFrame).

**Pie Chart**: A circular chart divided into slices to show proportions.

**Pivot Table**: A tool to quickly summarize and reorganize data, e.g., total sales per product category.

**Plot**: A graph or chart used to represent data visually.

**Pyplot (plt)**: A submodule of Matplotlib used for plotting graphs.

**Quartile**: Divisions of data into four equal parts; Q1 (25%), Q2/Median (50%), Q3 (75%).

**Scatter Plot**: A graph showing data points on an X and Y axis to study relationships.

**Series**: A one-dimensional labeled array in Pandas, like a column of marks.

**SQL (Structured Query Language)**: A language used to manage and query data stored in databases.

**Statistical Functions**: Functions like mean, median, sum, and standard deviation used to analyze data.

**Tabular Data**: Data arranged in rows and columns, like in Excel.

**Visualization**: Displaying data in a graphical form (charts/graphs) for easy understanding.

**Accuracy**: The percentage of correctly predicted results out of the total predictions.

**Algorithm**: A step-by-step method or procedure used by a computer to solve a problem.

**Bias (in AI)**: An unfair tendency of a model to favor certain outcomes because of imbalanced or incorrect training data.

**Classification**: A machine learning task that predicts categories (e.g., email as spam or not spam).

**Clustering**: Grouping data into clusters or sets where members of the same group are more similar to each other than to others.

**Confusion Matrix**: A table used to measure the performance of a classification model by showing correct and incorrect predictions.

**Cross-validation**: A method of testing a model's performance by splitting the data into parts and testing it multiple times to avoid overfitting.

**Data Preprocessing**: Preparing raw data by cleaning, normalizing, or transforming it before feeding it into a model.

**Decision Boundary**: A line or surface that separates different classes in a classification problem.

**Deep Learning**: A branch of machine learning that uses neural networks with many layers to process large and complex data.

**Dimensionality Reduction**: Reducing the number of features (columns) in data while keeping the important information.

**Evaluation Metrics**: Standards like accuracy, precision, recall, and F1-score used to measure model performance.

**F1-Score**: A metric that balances precision and recall, used when both false positives and false negatives are important.

**Feature**: An input variable (like height, weight, or age) used by a machine learning model.

**Hyperparameters**: Model settings that are chosen before training (e.g., number of clusters in K-Means).

**Logistic Regression**: A classification method used to predict binary outcomes (yes/no, true/false).

**Machine Learning (ML)**: A type of AI where systems learn from data and improve performance without explicit programming.

**Model Overfitting**: When a model performs very well on training data but fails on new, unseen data because it has learned noise instead of patterns.

**Model Underfitting**: When a model is too simple and fails to capture patterns in data, leading to poor performance.

**Neural Network**: A system of algorithms inspired by the human brain, designed to recognize patterns.

**Outlier**: A data point that is very different from other values, like an exam score of 100 when most scores are between 30–60.

**Precision**: The proportion of correct positive predictions out of all predicted positives.

**Principal Component Analysis (PCA)**: A technique for dimensionality reduction that transforms data into fewer, more important features.

**Recall (Sensitivity)**: The proportion of actual positives that were correctly identified by the model.

**Regression**: A machine learning technique used to predict continuous values (like predicting house prices).

**Supervised Learning**: Machine learning where the model is trained on labeled data (data with correct answers).

**Unsupervised Learning**: Machine learning where the model finds patterns in unlabeled data (without predefined answers).

# Answer Key

## Module 1. Data Science

### Session 1. Introduction to NumPy

**A. Multiple Choice Questions**

1. (c), 2. (d), 3. (c), 4. (a), 5. (c), 6. (a), 7. (a), 8. (c), 9. (c), 10. (d), 11. (d), 12. (c), 13. (c), 14. (c), 15. (b)

**B. Fill in the blanks**

1. ndarray, 2. np.array(), 3. Vector, Matrix, 4. ndim, 5. shape, 6. True, 7. non-zero, 8. negative indexing, 9. integers, 10. 1-D array

**C. True/False**

1. True, 2. True, 3. False, 4. True, 5. True, 6. False, 7. True, 8. True, 9. True, 10. False

### Session 2. Array Manipulation using NumPy

**A. Multiple Choice Questions**

1.(c), 2. (c), 3. (b), 4. (a), 5. (c), 6. (a), 7. (c)

**B. Fill in the blanks**

**1.** np.hstack(), 2. np.column_stack(), 3. np.vstack(), 4. specified axis, 5. np.newaxis, 6. np.squeeze(), 7. expand_dims(), 8. resize(), 9. ravel()

**C. True/False**

**1.** True, 2. False, 3. True, 4. True, 5. False, 6. True, 7. False, 8. False

### Session 3. Array Computation using NumPy

**A. Multiple Choice Questions**

1.(a), 2. (b), 3. (c), 4. (d), 5. (c), 6. (d), 7. (a), 8. (a), 9. (c)

**B. Fill in the blanks**

**1.** np.add(), 2. np.subtract(), 3. np.floor_divide(), 4. np.degrees(), 5. radians

**C. True/False**

**1.** True, 2. True, 3. False, 4. True, 5. True, 6. True, 7. False, 8. True, 9. True, 10. True

## Module 2. Data Analysis

### Session 1. Introduction to Pandas

**A. Multiple choice questions**

1. (c) 2. (c) 3. (b) 4. (b) 5. (b) 6. (c) 7. (c) 8. (c) 9. (c) 10. (b)

**B. Fill in the blanks**

1. manipulation 2. DataFrame 3. one 4. two 5. pd 6. NumPy 7. head 8. Not 9. 0 (zero) 10. tabular

**C. State whether True or False**

1. False 2. True. 3. False 4. True. 5. True 6. False 7. False 8. True 9. False 10. True.

## Session 2. Pandas Series

**A. Multiple choice questions**

1. (b) 2. (d) 3. (c) 4. (d) 5. (c) 6. (d) 7. (a) 8. (b) 9. (c) 10. (b) 11. (b) 12. (b)

**B. Fill in the blanks**

1. one 2. index 3. RangeIndex (or integer) 4. keys 5. index 6. values 7. dtype 8. NaN 9. size 10. element 11. hasnans

**C. State whether True or False**

1. False 2. False 3. True 4. False 5. False. 6. True. 7. True 8. False 9. True 10. True 11. True

## Session 3. Pandas DataFrame

**A. Multiple choice questions**

1. (c) 2. (d) 3. (b) 4. (c) 5. (c) 6. (b) 7. (c) 8. (c) 9. (b) 10. (c)

**B. Fill in the blanks**

1. columns 2. pd 3. column 4. shape 5. head() 6. 'Price' 7. columns 8. index, columns 9. index 10. dtypes

**C. State whether True or False**

1. False 2. True 3. False 4. True 5. True 6. False 7. False 8. True 9. True 10. False

## Session 4. Data Visualisation using Matplotlib

**A. Multiple choice questions**

1. (c) 2. (c) 3. (b) 4. (a) 5. (c) 6. (d) 7. (d) 8. (c) 9. (b) 10. (c)

**B. Fill in the blanks**

1. interactive 2. pyplot 3. plot() 4. show() 5. title 6. ylabel 7. savefig 8. legend 9. scatter 10. grid

**C. State whether True or False**

1. False 2. True 3. False 4. True 5. False 6. True 7. False 8. True 9. False 10. True

# Module 3. Machine Learning Models

## Session 1. Introduction to Machine Learning (ML)

**A. Multiple Choice Questions (MCQs)**

1(b), 2(b), 3(c), 4(c), 5(b)

**B. Fill in the Blanks**

1. Machine Learning, 2. Supervised, 3. Unsupervised, 4. Reinforcement, 5. Deep Learning

**C. True or False**

1. False, 2. True, 3. True, 4. False, 5. True

## Session 2: Regression Analysis

**A. Multiple Choice Questions (MCQs)**

1(a), 2(b), 3(b), 4(b), 5(d)

**B. Fill in the Blanks**

1. Linear, 2. Binary, 3. Probability, 4. Line, 5. Sigmoid

**C. True or False**

1. False, 2. True, 3. True, 4. False, 5. True

## Session 3. Clustering and Dimensionality Reduction

**A. Multiple Choice Questions (MCQs)**

1(b), 2(b), 3(b), 4(a), 5(b)

**B. Fill in the Blanks**

1. Unsupervised, 2. K-Means, 3. Irrelevant/redundant, 4. PCA, 5. Similarity

**C. True or False**

1. False, 2. True, 3. True, 4. True, 5. True

## Session 4. Model Evaluation and Metrics

**A. Multiple Choice Questions (MCQs)**

1(d), 2(c), 3(b), 4(c), 5(a)

**B. Fill in the Blanks**

1. Accuracy, 2. Confusion, 3. True, 4. Noise, 5. Cross

**C. True or False**

1. False, 2. True, 3. False, 4. True, 5. True

## Session 5. Ethics, Bias, and Myths in AI

**A. Multiple Choice Questions (MCQs)**

1(b), 2(a), 3(b), 4(a), 5(b)

**B. Fill in the Blanks**

1. Bias, 2. Privacy, 3. Replace, 4. Fairness, 5. Unfair

**C. True or False**

1. False, 2. True, 3. True, 4. True, 5. False

## Module 4. Neural Networks

## Session 1. Introduction to Neural Networks

**A. Multiple Choice Questions**

1. (b), 2. (c), 3. (b), 4. (c), 5. (c)

**B. Fill in the Blanks**

**1.** human brain, 2. input, 3. weight, 4. backpropagation, 5. sigmoid (or ReLU/tanh etc.)

**C. True/False**

**1.** False, 2. False, 3. True, 4. False, 5. True

## Session 2. Neurons and Activation Functions

**A. Multiple Choice Questions**

1. (b), 2. (b), 3. (b), 4. (c), 5. (b)

**B. Fill in the Blanks**

**1.** Sigmoid, 2. Weights, 3. Feedforward, 4. Bias, 5. Non-linearity

**C. True/False**

**1.** True, 2. False, 3. False, 4. True, 5. True

## Session 3. Applications of Neural Networks

**A. Multiple Choice Questions**

1. (b), 2. (c), 3. (b), 4. (b), 5. (b)

**B. Fill in the Blanks**

1. Image segmentation, 2. Classification, 3. RNN, 4. Location/Position, 5. Machine translation

**C. True/False**

1. True, 2. True, 3. True, 4. False, 5. False

## Session 4. Python Libraries for Neural Networks

**A. Multiple Choice Questions**

**1.** c 2. b 3. b 4. b 5. c

**B. Fill in the Blanks**

**1.** Google 2. Keras 3. matrix operations 4. Adam 5. 1

**C. True/False**

**1.** T 2. F 3. T 4. T 5. F

## Session 5. Training Neural Networks

**A. Multiple Choice Questions**

**1.** b 2. b 3. c 4. a 5. c

**B. Fill in the Blanks**

**1.** epoch 2. loss 3. negative 4. correct predictions ÷ total predictions 5. –1, 1

**C. True/False**

**1.** F 2. F 3. T 4. T 5. F

## Session 6. Building a Basic Neural Network for Classification

**A. Multiple Choice Questions**

**1.** b 2. b 3. b 4. c 5. b

**B. Fill in the Blanks**

**1.** fully connected 2. optimizer 3. sigmoid 4. confusion matrix 5. overfitting

**C. True/False**

**1.** T 2. F 3. T 4. F 5. T

## Module 5. AI Project

## Session 1. AI Project Cycle

**A. Multiple choice questions**

1. (c) 2. (c) 3. (b) 4. (c) 5. (d) 6. (c) 7. (b) 8. (c) 9. (b) 10. (c)

**B. Fill in the blanks**

1. Problem Scoping 2. Problem Scoping 3. 4Ws Problem Canvas 4. Data Acquisition 5. Data Cleaning 6. Data Exploration 7. Model Training 8. Model Evaluation 9. Deployment 10. Monitoring 11. iterative

**C. State whether True or False**

1. False. 2. True. 3. False. 4. False. 5. False. 6. False. 7. True. 8. False. 9. False. 10. False.

## Session 2. Project Work

**A. Multiple choice questions**

1. (c) 2. (c) 3. (c) 4. (c) 5. (c) 6. (b) 7. (b) 8. (b) 9. (c) 10. (b)

**B. Fill in the blanks**

1. Web scraping 2. Data visualization 3. SMART 4. Data Cleaning. 5. Hyperparameter Tuning. 6. progress, performance, outcomes 7. model performanc 8. hyperparameters 9. production environment 10. training data

**C. State whether the following statement is True or False**

1. False. 2. False. 3. False 4. True 5. False 6. False 7. False 8. True 9. False 10. False

## Session 3. Sample of AI Project & Report Summary

**A. Multiple Choice Questions (MCQs)**

1. (b) 2. (c) 3. (b) 4. (c) 5. (d)

**B. Fill in the Blanks**

1. Positive, Negative, Neutral 2. Feature 3. Feature 4. Sensors and Cameras 5. Categorical Crossentropy

**C. State Whether True or False**

1. (F) 2. (T) 3. (F) 4. (F) 5. (T) False