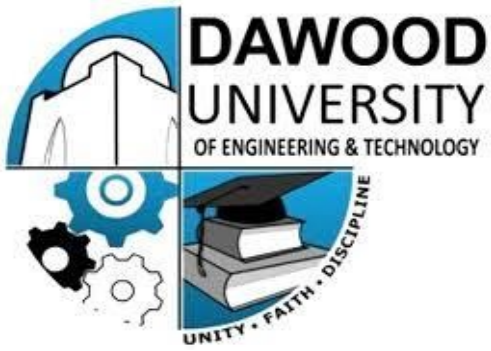
Artificial Intelligence

(Practical Manual)



4th Semester, 2nd Year
BATCH -2023

BS ARTIFICIAL INTELLIGENCE

DAWOOD UNIVERSITY OF ENGINEERING & TECHNOLOGY, KARACHI

Dawood University Of Engineering and Technology, Karachi

CERTIFICATE

This is to certify that Mr./Ms. **Mehak Faheem** with Roll # **23F-AI-47** of Batch 2023 has successfully completed all the labs prescribed for the course "Artificial Intelligence".

Engr. Hamza Farooqui
Lecturer
Department of AI

S. No.	Title of Experiment
1	Introduction to Programming in Python
2	Working with NumPy Arrays
3	Data Manipulation Using Pandas
4	Implementing Breadth First Search (BFS)
5	Open Ended Lab - 1
6	Implementing Depth First Search (DFS)
7	Implementing Best First Search (Without Heuristics)
8	A* Search Algorithm
9	Simple Linear Regression
10	Multivariate Linear Regression
11	Open Ended Lab – 2
12	Binary Classification using Logistic Regression

Lab No: 1

Objective: To introduce students to **Python programming** and develop their ability to write, understand, and execute basic Python code for data handling and problem solving.

Why Python?

- Python is a high-level, interpreted language widely used in AI, data science, and software development.
- It is known for its simple syntax, large community, and rich set of libraries.

Core Concepts: -

Concept	Description
Variables & Data Types	int, float, str, bool, list, tuple, dict
Operators	Arithmetic (+, -, *, /), Comparison (==, !=)
Control Structures	if, elif, else, for, while
Functions	Using def to define reusable code blocks
Input/Output	input(), print()
Basic Libraries	math, random, datetime, etc.

- **Simple Example Code**

```
name = input("Enter your name: ")

print("Hello,", name)

num = int(input("Enter a number: "))

print("Square is:", num ** 2)
```

Why It Matters in AI:

- Python is the primary language for AI frameworks like TensorFlow, PyTorch, and scikit-learn.
- Understanding Python is essential for implementing AI algorithms, preprocessing data, and building models.

Task:
Given two strings needle and haystack, return the index of the first occurrence of needle in haystack, or -1 if needle is not part of haystack.

Example 1:

Input: haystack = "sadbutsad", needle = "sad"

Output: 0

Explanation: "sad" occurs at index 0 and 6.

The first occurrence is at index 0, so we return 0.

Example 2:

Input: haystack = "leetcode", needle = "leeto"

Output: -1

Explanation: "leeto" did not occur in "leetcode", so we return -1.

Task: Given two strings needle and haystack, return the index of the first occurrence of needle in haystack, or -1 if needle is not part of haystack.

Code:

```
def strStr(haystack: str, needle: str) -> int:
    if needle == "":
        return 0
    return haystack.find(needle)

haystack = "sadbutsad"
needle = "sad"
print(strStr(haystack, needle))
haystack = "leetcode"
needle = "leeto"
print(strStr(haystack, needle))
haystack = "cameacross"
needle = "ro"
print(strStr(haystack, needle))
```

Output:

```
0
-1
6
```

Lab No: 2

Objective: Write Python program to demonstrate use of **Numpy**

Practical Significance: -

Though Python is simple to learn language but it also very strong with its features. As mentioned earlier Python supports various built-in packages. Apart from built-in package user can also make their own packages i.e. User Defined Packages. **Numpy** is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. This practical will allow students to write a code.

Minimum Theoretical Background: -

NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed.

Steps for Installing numpy in windows OS

1. goto Command prompt
2. run command pip install numpy
3. open IDLE Python Interpreter
4. Check numpy is working or not

```
>>> import numpy
>>> import numpy as np
>>> a=np.array([10,20,30,40,50])
>>> print(a)
[10 20 30 40 50]
```

Example: -

```
>>>  student=np.dtype([('name','S20'),('age','i1'),('marks','f4')])
>>>  a=np.array([('Hamza',43,90),('Asad',38,80)],dtype=student)
>>> print(a)
[('Hamza', 43, 90.) ('Asad', 38, 80.)]
```

Example: -

```
>>> print(a)
[10 20 30 40 50 60]
>>> a.shape=(2,3)
>>> print(a) [[10 20 30]
 [40 50 60]]
>>> a.shape=(3,2)
>>> print(a) [[10 20]
 [30 40]
 [50 60]]
```

Tasks: -

Write Python Code for the following:

1) How to get the common items between two python numpy arrays?
2) How to get the positions where elements of two arrays match?
3) How to extract all numbers between a given range from a numpy array?
4) Implement the moving average for the 1D array in NumPy.

1) How to get the common items between two python numpy arrays?

Code:

```
import numpy as np

x = np.array([1, 2, 3, 4, 5])
y = np.array([4, 5, 6, 7, 8])

common = np.intersect1d(x, y)
print("Common elements:", common)
```

Output:

```
Common elements: [4 5]
```

2) How to get the positions where elements of two arrays match?

Code:

```
a = np.array([1, 2, 3, 4, 5])
b = np.array([1, 9, 3, 8, 5])

positions = np.where(a == b)
print("Matching positions:", positions)
```

Output:

```
Matching positions: (array([0, 2, 4]),)
```

3) How to extract all numbers between a given range from a numpy array?

Code:

```
arr = np.array([101, 125, 130, 50, 40, 260, 150])
result = arr[(arr > 10) & (arr < 50)]
print("Numbers between 10 and 50:", result)
```

Output:

```
Numbers between 10 and 50: [40]
```

4) Implement the moving average for the 1D array in NumPy.

Code:

```
def moving_avg(arr, window_size):
return np.convolve(arr, np.ones(window_size)/window_size, mode='valid')

data = np.array([10, 20, 30, 40, 50, 60])
window = 3
result = moving_avg(data, window)
print("Moving Average:", result)
```

Output:

```
Moving Average: [20. 30. 40. 50.]
```

Lab No: 3

Objective: To equip students with the skills to manipulate, clean, analyze, and preprocess structured datasets using the **Pandas library** in Python, preparing data for use in AI models and algorithms.

Introduction to Pandas: -

Pandas is a powerful Python library used for data manipulation and analysis. It provides two main data structures:

1. **Series** – One-dimensional labeled array.
2. **DataFrame** – Two-dimensional labeled data structure, similar to a table in a database or an Excel sheet.

Pandas is widely used in **AI and Machine Learning** pipelines for preprocessing, analyzing, and cleaning data before feeding it into models.

Loading Data: -

You can read structured data from various file formats:

```
# Load CSV file
df = pd.read_csv('data.csv')

# Load Excel file
df_excel = pd.read_excel('data.xlsx')

# Load from dictionary
data = {'Name': ['Alice', 'Bob'], 'Age': [25, 30]}
df_dict = pd.DataFrame(data)
```

Exploring Data: -

```
df.head()          # First 5 rows
df.tail()          # Last 5 rows
df.info()          # Data types and non-null values
df.describe()      # Statistical summary of numeric columns
```

Data Selection: -

```
df['Age']              # Select a single column
df[['Name', 'Age']]    # Select multiple columns
df.iloc[0]             # Row by index position
df.loc[0]              # Row by index label
```

Filtering Data: -

```
# Filter rows where Age > 25
df[df['Age'] > 25]

# Filter rows with multiple conditions
df[(df['Age'] > 25) & (df['Gender'] == 'Male')]
```

Adding/Modifying Columns: -

```
# Add a new column
df['Is_Adult'] = df['Age'] >= 18

# Modify an existing column
df['Age'] = df['Age'] + 1
```

Handling Missing Values: -

```
df.isnull().sum()       # Count missing values
df.dropna()             # Drop rows with any missing values
df.fillna(0)            # Fill missing values with 0
df.fillna(df.mean())    # Fill with mean of the column
```

Grouping and Aggregation: -

```
# Group by Gender and calculate mean age
df.groupby('Gender')['Age'].mean()

# Count entries per category
df['Gender'].value_counts()
```

Sorting and Reordering: -

```
df.sort_values(by='Age', ascending=False)  # Sort by Age descending
df.reset_index(drop=True, inplace=True)    # Reset index after sorting
```

Dropping Columns and Rows: -

```
df.drop(columns=['Is_Adult'], inplace=True)  # Drop a column
df.drop(index=[0], inplace=True)             # Drop a row
```

Merging and Joining DataFrames: -

```
# Merge on a common column
merged_df = pd.merge(df1, df2, on='ID')

# Concatenate along rows or columns
pd.concat([df1, df2], axis=0)  # Row-wise
pd.concat([df1, df2], axis=1)  # Column-wise
```

Saving Data: -

```
df.to_csv('cleaned_data.csv', index=False)
df.to_excel('output.xlsx', index=False)
```

Why Pandas is Important in AI: -

- Prepares raw data for ML models.
- Enables feature engineering.
- Helps detect and handle missing or inconsistent values.

- Supports exploratory data analysis (EDA) and data cleaning.

Tasks: -

Kaggle IMDb Top 1000 Movies dataset

Task 1: Load and Explore the Dataset

1. Load the dataset.
2. Display the first 5 rows.
3. Check the data types of each column.
4. Find the number of rows and columns.
5. Check for missing values.

Task 2: Data Cleaning

1. Remove any duplicate rows if present.
2. Fill missing values in the dataset (e.g., replace missing ratings with the mean rating).
3. Convert the Runtime column (which is in minutes as a string, e.g., "120 min") to an integer.

Task 3: Data Filtering & Sorting

1. Find all movies with an **IMDb rating greater than 8.5**.
2. List movies that belong to the **Action or Sci-Fi genre**.
3. Find movies that were released **between 2000 and 2015**.
4. Sort the dataset based on **IMDb rating in descending order**.

Data Aggregation & Grouping

1. Find the **average IMDb rating** for each genre.
2. Determine which **year had the most movies released**.
3. Find the **top 5 directors** who have directed the most movies in the dataset.

Visualization (Optional)

Matplotlib or **Seaborn**

1. Plot a histogram of IMDb ratings.
2. Create a bar chart showing the **top 10 genres** with the most movies.
3. Visualize the **trend of IMDb ratings over the years**.

Task 1: Load and Explore the Dataset

1. Load the dataset.

```
# 1. Load the dataset.
import pandas as pd
import numpy as np
df = pd.read_csv("IMDB_top_1k.csv")
df
```

	Unnamed: 0	Title	Certificate	Duration	Genre	Rate	Metascore	Description	Cast	Info
0	0	1. The Shawshank Redemption (1994)	R	142 min	Drama	9.3	80.0	Two imprisoned men bond over a number of years...	Director: Frank Darabont | Stars: Tim Robbins,...	Votes: 2,295,987 | Gross: $28.34M
1	1	2. The Godfather (1972)	R	175 min	Crime, Drama	9.2	100.0	The aging patriarch of an organized crime dyna...	Director: Francis Ford Coppola | Stars: Marlon...	Votes: 1,584,782 | Gross: $134.97M
2	2	3. The Dark Knight (2008)	PG-13	152 min	Action, Crime, Drama	9.0	84.0	When the menace known as the Joker wreaks havo...	Director: Christopher Nolan | Stars: Christian...	Votes: 2,260,649 | Gross: $534.86M
3	3	4. The Godfather: Part II (1974)	R	202 min	Crime, Drama	9.0	90.0	The early life and career of Vito Corleone in ...	Director: Francis Ford Coppola | Stars: Al Pac...	Votes: 1,107,253 | Gross: $57.30M
4	4	5. The Lord of the Rings: The Return of the Ki...	PG-13	201 min	Action, Adventure, Drama	8.9	94.0	Gandalf and Aragorn lead the World of Men agai...	Director: Peter Jackson | Stars: Elijah Wood, ...	Votes: 1,614,369 | Gross: $377.85M

2. Display the first 5 rows.

```
df.head(5)
```

	Unnamed: 0	Title	Certificate	Duration	Genre	Rate	Metascore	Description	Cast	Info
0	0	1. The Shawshank Redemption (1994)	R	142 min	Drama	9.3	80.0	Two imprisoned men bond over a number of years...	Director: Frank Darabont | Stars: Tim Robbins,...	Votes: 2,295,987 | Gross: $28.34M
1	1	2. The Godfather (1972)	R	175 min	Crime, Drama	9.2	100.0	The aging patriarch of an organized crime dyna...	Director: Francis Ford Coppola | Stars: Marlon...	Votes: 1,584,782 | Gross: $134.97M
2	2	3. The Dark Knight (2008)	PG-13	152 min	Action, Crime, Drama	9.0	84.0	When the menace known as the Joker wreaks havo...	Director: Christopher Nolan | Stars: Christian...	Votes: 2,260,649 | Gross: $534.86M
3	3	4. The Godfather: Part II (1974)	R	202 min	Crime, Drama	9.0	90.0	The early life and career of Vito Corleone in ...	Director: Francis Ford Coppola | Stars: Al Pac...	Votes: 1,107,253 | Gross: $57.30M
4	4	5. The Lord of the Rings: The Return of the Ki...	PG-13	201 min	Action, Adventure, Drama	8.9	94.0	Gandalf and Aragorn lead the World of Men agai...	Director: Peter Jackson | Stars: Elijah Wood, ...	Votes: 1,614,369 | Gross: $377.85M

3. Check the data types of each column.

```
df.dtypes
```

```
Unnamed: 0       int64
Title           object
Certificate     object
Duration        object
Genre           object
Rate           float64
Metascore      float64
Description     object
Cast            object
Info            object
dtype: object
```

4. Find the number of rows and columns.

```
df.shape
```

```
(1000, 10)
```

5. Check for missing values.

```
df.isnull()
```

	Unnamed: 0	Title	Certificate	Duration	Genre	Rate	Metascore	Description	Cast	Info
0	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...
995	False	False	False	False	False	False	True	False	False	False
996	False	False	False	False	False	False	False	False	False	False
997	False	False	False	False	False	False	False	False	False	False
998	False	False	False	False	False	False	True	False	False	False
999	False	False	False	False	False	False	True	False	False	False

1000 rows × 10 columns

Task 2: Data Cleaning

1. Remove any duplicate rows if present.

```
df.drop_duplicates()
```

	Unnamed: 0	Title	Certificate	Duration	Genre	Rate	Metascore	Description	Cast	Info
0	0	1. The Shawshank Redemption (1994)	R	142 min	Drama	9.3	80.0	Two imprisoned men bond over a number of years...	Director: Frank Darabont | Stars: Tim Robbins,...	Votes: 2,295,987 | Gross: $28.34M
1	1	2. The Godfather (1972)	R	175 min	Crime, Drama	9.2	100.0	The aging patriarch of an organized crime dyna...	Director: Francis Ford Coppola | Stars: Marlon...	Votes: 1,584,782 | Gross: $134.97M
2	2	3. The Dark Knight (2008)	PG-13	152 min	Action, Crime, Drama	9.0	84.0	When the menace known as the Joker wreaks havo...	Director: Christopher Nolan | Stars: Christian...	Votes: 2,260,649 | Gross: $534.86M
3	3	4. The Godfather: Part II (1974)	R	202 min	Crime, Drama	9.0	90.0	The early life and career of Vito Corleone in ...	Director: Francis Ford Coppola | Stars: Al Pac...	Votes: 1,107,253 | Gross: $57.30M

2. Fill missing values in the dataset (e.g., replace missing ratings with the mean rating).

```
df['Rate'].fillna(df['Rate'].mean())

0      9.3
1      9.2
2      9.0
3      9.0
4      8.9
      ...
995    8.0
996    8.0
997    8.0
998    8.1
999    8.1
Name: Rate, Length: 1000, dtype: float64
```

3. Convert the Runtime column (which is in minutes as a string, e.g., "120 min") to an integer.

```
df['Duration'] = df['Duration'].astype(str)
df['Duration in int'] = df['Duration'].str.replace('min', '').astype('int64')
df
```

	Unnamed: 0	Title	Certificate	Duration	Genre	Rate	Metascore	Description	Cast	Info	Duration in int
0	0	1. The Shawshank Redemption (1994)	R	142 min	Drama	9.3	80.0	Two imprisoned men bond over a number of years...	Director: Frank Darabont | Stars: Tim Robbins,...	Votes: 2,295,987 | Gross: $28.34M	142
1	1	2. The Godfather (1972)	R	175 min	Crime, Drama	9.2	100.0	The aging patriarch of an organized crime dyna...	Director: Francis Ford Coppola | Stars: Marlon...	Votes: 1,584,782 | Gross: $134.97M	175
2	2	3. The Dark Knight (2008)	PG-13	152 min	Action, Crime, Drama	9.0	84.0	When the menace known as the Joker wreaks havo...	Director: Christopher Nolan | Stars: Christian...	Votes: 2,260,649 | Gross: $534.86M	152
3	3	4. The Godfather: Part II (1974)	R	202 min	Crime, Drama	9.0	90.0	The early life and career of Vito Corleone in ...	Director: Francis Ford Coppola | Stars: Al Pac...	Votes: 1,107,253 | Gross: $57.30M	202
4	4	5. The Lord of the Rings: The Return of the Ki...	PG-13	201 min	Action, Adventure, Drama	8.9	94.0	Gandalf and Aragorn lead the World of Men agai...	Director: Peter Jackson | Stars: Elijah Wood, ...	Votes: 1,614,369 | Gross: $377.85M	201

Task 3: Data Filtering & Sorting

1. Find all movies with an **IMDb rating greater than 8.5**.

```
df[df['Rate'] > 8.5]
```

	Unnamed: 0	Title	Certificate	Duration	Genre	Rate	Metascore	Description	Cast	Info	Duration in int
0	0	1. The Shawshank Redemption (1994)	R	142 min	Drama	9.3	80.0	Two imprisoned men bond over a number of years...	Director: Frank Darabont | Stars: Tim Robbins,...	Votes: 2,295,987 | Gross: $28.34M	142
1	1	2. The Godfather (1972)	R	175 min	Crime, Drama	9.2	100.0	The aging patriarch of an organized crime dyna...	Director: Francis Ford Coppola | Stars: Marlon...	Votes: 1,584,782 | Gross: $134.97M	175
2	2	3. The Dark Knight (2008)	PG-13	152 min	Action, Crime, Drama	9.0	84.0	When the menace known as the Joker wreaks havo...	Director: Christopher Nolan | Stars: Christian...	Votes: 2,260,649 | Gross: $534.86M	152
3	3	4. The Godfather: Part II (1974)	R	202 min	Crime, Drama	9.0	90.0	The early life and career of Vito Corleone in ...	Director: Francis Ford Coppola | Stars: Al Pac...	Votes: 1,107,253 | Gross: $57.30M	202
4	4	5. The Lord of the Rings: The Return of the Ki...	PG-13	201 min	Action, Adventure, Drama	8.9	94.0	Gandalf and Aragorn lead the World of Men agai...	Director: Peter Jackson | Stars: Elijah Wood, ...	Votes: 1,614,369 | Gross: $377.85M	201
5	5	6. Pulp Fiction (1994)	R	154 min	Crime, Drama	8.9	94.0	The lives of two mob hitmen, a boxer, a	Director: Quentin Tarantino | Stars:	Votes: 1,792,919 | Gross:	154

2. List movies that belong to the **Action or Sci-Fi genre**.

```
df['is_action_or_scifi'] = df['Genre'].str.contains('Action|Sci-Fi')
action_or_scifi_movies = df[df['is_action_or_scifi']]
action_or_scifi_movies
```

	Unnamed: 0	Title	Certificate	Duration	Genre	Rate	Metascore	Description	Cast	Info	Duration in int	is_action_or_scifi
2	2	3. The Dark Knight (2008)	PG-13	152 min	Action, Crime, Drama	9.0	84.0	When the menace known as the Joker wreaks havo...	Director: Christopher Nolan | Stars: Christian...	Votes: 2,260,649 | Gross: $534.86M	152	True
4	4	5. The Lord of the Rings: The Return of the Ki...	PG-13	201 min	Action, Adventure, Drama	8.9	94.0	Gandalf and Aragorn lead the World of Men agai...	Director: Peter Jackson | Stars: Elijah Wood, ...	Votes: 1,614,369 | Gross: $377.85M	201	True
8	8	9. Inception (2010)	PG-13	148 min	Action, Adventure, Sci-Fi	8.8	74.0	A thief who steals corporate secrets through t...	Director: Christopher Nolan | Stars: Leonardo ...	Votes: 2,022,655 | Gross: $292.58M	148	True
10	10	11. The Lord of the Rings: The Fellowship of t...	PG-13	178 min	Action, Adventure, Drama	8.8	92.0	A meek Hobbit from the Shire and eight compani...	Director: Peter Jackson | Stars: Elijah Wood, ...	Votes: 1,630,106 | Gross: $315.54M	178	True

3. Find movies that were released **between 2000 and 2015**.

```
df2 = pd.read_csv('data.csv')
df2
movies = df2[(df2["releaseYear"] >= 2000) & (df2["releaseYear"] <= 2015)]
print(movies)
```

```
           id                                          title  \
5    tt0468569                                The Dark Knight
6    tt0167260     The Lord of the Rings: The Return of the King
12   tt0120737  The Lord of the Rings: The Fellowship of the Ring
17   tt1375666                                      Inception
20   tt0167261            The Lord of the Rings: The Two Towers
..         ...                                            ...
991  tt1596363                                The Big Short
992  tt1535109                              Captain Phillips
993  tt0139654                                  Training Day
994  tt0325710                              The Last Samurai
996  tt0765429                             American Gangster

                       genres  averageRating  numVotes  releaseYear
5         Action, Crime, Drama            9.0   3008955         2008
6    Adventure, Drama, Fantasy            9.0   2070753         2003
12   Adventure, Drama, Fantasy            8.9   2100667         2001
17   Action, Adventure, Sci-Fi            8.8   2672591         2010
20   Adventure, Drama, Fantasy            8.8   1866871         2002
..                         ...            ...       ...          ...
991   Biography, Comedy, Drama            7.8    508752         2015
992   Action, Biography, Crime            7.8    506810         2013
993     Crime, Drama, Thriller            7.8    494417         2001
994              Action, Drama            7.8    487774         2003
996    Biography, Crime, Drama            7.8    470568         2007

[315 rows x 6 columns]
```

4. Sort the dataset based on **IMDb rating in descending order**.

```
df.sort_values('Rate', ascending = False)
```

	Unnamed: 0	Title	Certificate	Duration	Genre	Rate	Metascore	Description	Cast	Info	Duration in int	is_action_or_scifi
0	0	1. The Shawshank Redemption (1994)	R	142 min	Drama	9.3	80.0	Two imprisoned men bond over a number of years...	Director: Frank Darabont | Stars: Tim Robbins,...	Votes: 2,295,987 | Gross: $28.34M	142	False
1	1	2. The Godfather (1972)	R	175 min	Crime, Drama	9.2	100.0	The aging patriarch of an organized crime dyna...	Director: Francis Ford Coppola | Stars: Marlon...	Votes: 1,584,782 | Gross: $134.97M	175	False
3	3	4. The Godfather: Part II (1974)	R	202 min	Crime, Drama	9.0	90.0	The early life and career of Vito Corleone in ...	Director: Francis Ford Coppola | Stars: Al Pac...	Votes: 1,107,253 | Gross: $57.30M	202	False
2	2	3. The Dark Knight (2008)	PG-13	152 min	Action, Crime, Drama	9.0	84.0	When the menace known as the Joker wreaks havo...	Director: Christopher Nolan | Stars: Christian...	Votes: 2,260,649 | Gross: $534.86M	152	True
4	4	5. The Lord of the Rings: The Return of the Ki...	PG-13	201 min	Action, Adventure, Drama	8.9	94.0	Gandalf and Aragorn lead the World of Men agai...	Director: Peter Jackson | Stars: Elijah Wood, ...	Votes: 1,614,369 | Gross: $377.85M	201	True
...	...	...	...	...	...	...	...	...	...	...	...	...

Data Aggregation & Grouping

1. Find the **average IMDb rating** for each genre.

```
# Step 1: Remove missing genres (NaNs)
df = df[df['Genre'].notna()]

# Step 2: Convert to string and split
df['Genre'] = df['Genre'].astype(str).str.split(',')

# Step 3: Explode
exploded_genres = df.explode("Genre")

# Step 4: Remove any accidental whitespace
exploded_genres['Genre'] = exploded_genres['Genre'].str.strip()

# Step 5: Group and find average rating
avg_rating = exploded_genres.groupby("Genre")["Rate"].mean()
print(avg_rating)
```

```
Genre
Action       8.089372
Adventure    8.093365
Animation    8.059000
Biography    8.066667
Comedy       8.075000
Crime        8.119318
Drama        8.101374
Family       8.160000
Fantasy      8.113208
Film-Noir    8.129412
History      8.093939
Horror       8.192308
Music        8.082143
Musical      8.091667
Mystery      8.109709
Romance      8.077551
Sci-Fi       8.109524
Sport        8.175000
Thriller     8.072727
War          8.205405
Western      8.357143
Name: Rate, dtype: float64
```

2. Determine which **year had the most movies released**.

```
df2 = pd.read_csv('data.csv')
df2
most_movies = df2['releaseYear'].value_counts().idxmax()
print(most_movies)
```

```
2014
```

3. Find the **top 5 directors** who have directed the most movies in the dataset.

```
df['Cast'].value_counts().head(5)
```

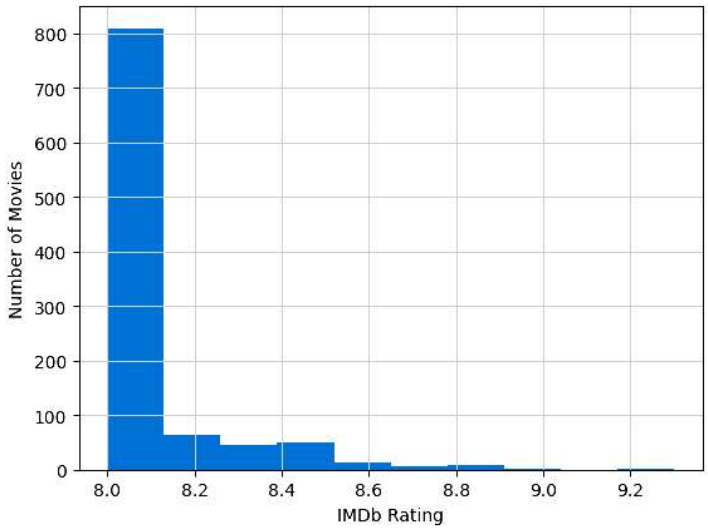
```
Cast
Director: Federico Fellini | Stars: Giulietta Masina, François Périer, Franca Marzi, Dorian Gray                    8
Director: Akira Kurosawa | Stars: Toshirô Mifune, Minoru Chiaki, Isuzu Yamada, Takashi Shimura                      8
Director: David Lynch | Stars: Naomi Watts, Laura Harring, Justin Theroux, Jeanne Bates                            7
Director: Lars von Trier | Stars: Björk, Catherine Deneuve, David Morse, Peter Stormare                            7
Directors: Pete Docter, David Silverman, Lee Unkrich | Stars: Billy Crystal, John Goodman, Mary Gibbs, Steve Buscemi    7
Name: count, dtype: int64
```

Visualization:

Matplotlib or **Seaborn**

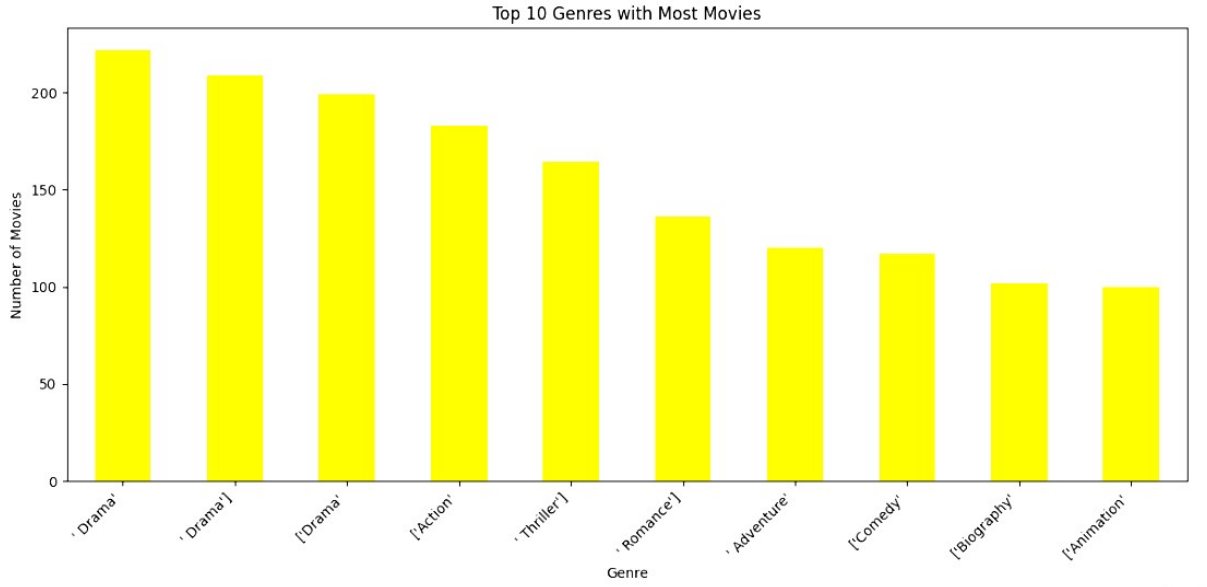
1. Plot a histogram of IMDb ratings.

```
import matplotlib.pyplot as plt
df['Rate'].hist()
plt.xlabel('IMDb Rating')
plt.ylabel('Number of Movies')
plt.show()
```



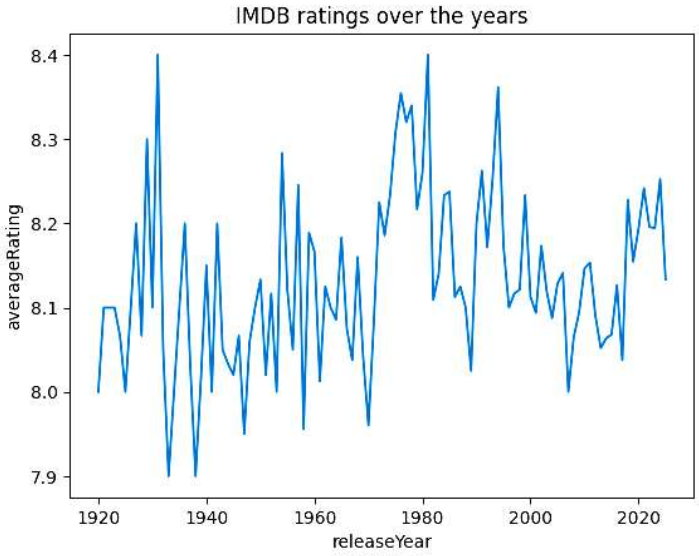
2. Create a bar chart showing the **top 10 genres** with the most movies.

```python
import matplotlib.pyplot as plt

# Clean the genre strings before splitting (strip whitespace)
df['Genre'] = df['Genre'].astype(str).str.split(',')
df['Genre'] = df['Genre'].apply(lambda x: [genre.strip() for genre in x])
exploded_genre = df.explode('Genre')

# Filter out any empty strings or NaN values
exploded_genre = exploded_genre[exploded_genre['Genre'].str.strip() != '']
exploded_genre = exploded_genre[~exploded_genre['Genre'].isna()]

# Count and get top 10 genres
genre_counts = exploded_genre['Genre'].value_counts().head(10)

# Create the bar chart
plt.figure(figsize=(12, 6))
genre_counts.plot.bar(color="yellow")
plt.title("Top 10 Genres with Most Movies")
plt.xlabel('Genre')
plt.ylabel('Number of Movies')
plt.xticks(rotation=45, ha='right')  # Rotate labels 45 degrees and align them right
plt.tight_layout()  # Adjust layout to make room for rotated labels
plt.show()
```



3. Visualize the **trend of IMDb ratings over the years**.

```
import matplotlib.pyplot as plt

df2 = pd.read_csv('data.csv')
df2
IMDBratings = df2.groupby('releaseYear')['averageRating'].mean()
IMDBratings.plot.line(title = "IMDB ratings over the years")
plt.xlabel('releaseYear')
plt.ylabel('averageRating')
plt.show()
```



Lab No: 4

Objective: To enable students to understand and implement the **Breadth-First Search algorithm** for solving graph traversal and pathfinding problems in artificial intelligence applications.

Breadth-First Search (BFS) is an **uninformed search algorithm** that explores a graph level by level. It begins at a selected node (called the root or source) and explores all neighbouring nodes at the current depth before moving on to nodes at the next depth level.

It uses a **queue** data structure (FIFO) to keep track of the nodes to be visited.

Practical Significance: -
Breadth-First Search (BFS) has practical significance in various fields and applications due to its unique characteristics. Here are some practical applications:

Network Routing and Broadcasting:
In computer networks, BFS is often used to discover neighboring nodes and determine the shortest path for routing.
It is also employed in broadcasting information across a network efficiently.

Web Crawling:
Search engines use BFS to crawl the web and index pages. Starting from a seed page, BFS explores links level by level, ensuring a systematic and comprehensive traversal.

Puzzle Solving:
BFS is used in puzzle-solving scenarios, such as the famous "Eight Puzzle" or "Fifteen Puzzle," to find the shortest sequence of moves to reach the goal state.

Maze Solving:
BFS can be applied to solve mazes by finding the shortest path from the start to the exit. It guarantees the discovery of the shortest path when the maze has uniform edge weights.

Robotics and Autonomous Vehicles:
BFS is employed in robotics and autonomous vehicle navigation to explore and map unknown environments systematically.

Optimizing Data Structures:
BFS is often used in optimizing data structures like trees and graphs, ensuring efficient access and retrieval of information.

Game Development:
BFS can be applied in game development for tasks such as pathfinding, where it helps in finding the shortest path for characters or objects.

Database Querying:
BFS is used in certain database querying scenarios to explore relationships and dependencies between different entities.

In summary, BFS is a versatile algorithm with practical applications across various domains, providing an efficient way to explore and analyze relationships in interconnected systems.

BFS Algorithm: -

Input:
Graph G represented as an adjacency list, starting vertex start, and goal vertex goal.

Initialization:
Create an empty set visited to keep track of visited vertices.
Create a deque queue and enqueue the start vertex.
Add the start vertex to the visited set.

BFS Loop:
While the queue is not empty:
Dequeue a vertex current_vertex from the front of the queue.
Print or process current_vertex.
If current_vertex is equal to the goal vertex:
Print a message indicating that the goal state is reached. Return,
indicating that the goal state is reached.
For each neighbor neighbor of current_vertex in the graph: If
neighbor is not in the visited set:
Enqueue neighbor to the back of the queue. Add
neighbor to the visited set.

Output: Print a message indicating that the goal state is not reached if the loop completes without returning.

Tasks: -

1. Write a Program to Implement Breadth First Search without goal state using Python.
2. Write a Program to Implement Breadth First Search with goal state using Python.

1.Write a Program to Implement Breadth First Search without goal state using Python.

Code:

```
import collections
def bfs(graph, root):
    visited = set()
    queue = collections.deque([root])

    while queue:
        vertex = queue.popleft()
        visited.add(vertex)

        for i in graph[vertex]:
            if i not in visited:
                queue.append(i)
    print(visited)

if __name__ == "__main__":
    graph = {0:[1,2,3] , 1:[0,2], 2:[0,1,4], 3:[0], 4:[2]}
    bfs(graph, 0)
```

Output:

```
{0, 1, 2, 3, 4}
```

2.Write a Program to Implement Breadth First Search with goal state using Python.

Code:

```
import collections
def bfs(graph, root, goalNode):
    visited = set()
    queue = collections.deque([root])

    while queue:
        vertex = queue.popleft()
        if vertex == goalNode:
            visited.add(vertex)
            print("Goal Found:", goalNode)
            print("Path:", visited)
            return

        visited.add(vertex)

        for i in graph[vertex]:
            if i not in visited:
                queue.append(i)

if __name__ == "__main__":
    graph = {0:[1,2,3] , 1:[0,2], 2:[0,1,4], 3:[0], 4:[2]}
    goalNode = 4
    bfs(graph, 0, goalNode)
```

Output:

```
Goal Found: 4
Path: {0, 1, 2, 3, 4}
```

Lab No: 5

Objective:

The objective of this lab is to enable students to:

Learn how to load and process data using the Pandas library in Python.

1) Perform basic data exploration, including counting, grouping, and computing averages.

2) Apply filtering and logic to extract meaningful information from data.

3) Practice sorting data and retrieving specific records based on conditions.

4) Write functions for custom queries on the dataset.

5) Enhance problem-solving skills through real-world data analysis tasks, such as finding the highest paying passenger or passengers under a certain age.

Part 1 – Data Processing with Pandas

1. Load the dataset and display the total number of passengers.

```
import pandas as pd
#1. Load the dataset and display the total number of passengers.
df = pd.read_csv('passenger_data.csv') #Load the dataset
df['passenger_id'].value_counts().sum() #count the passengers and then sum it
```

```
np.int64(50)
```

2. Show how many passengers are traveling to each destination.

```
df.groupby('destination')['passenger_id'].count()
```

```
destination
Islamabad     9
Karachi      10
Lahore       12
Peshawar      7
Quetta       12
Name: passenger_id, dtype: int64
```

3. Show the average ticket price paid by male and female passengers.

```
df.groupby('gender')['ticket_price'].mean()
```

```
gender
F    14855.464286
M    14581.727273
Name: ticket_price, dtype: float64
```

Part 2 – Python Filtering and Analysis

1.Write a function get_passengers_by_destination(dest) that returns a list of names of passengers going to the specified destination.

```
def get_passengers_by_destination(dest): #takes destination as an arg
 return df.loc[df['destination'] == dest, 'name'].tolist() #it will check
get_passengers_by_destination('Karachi') # calling the func
```

```
['Usman Rafiq',
 'Ayesha Noor',
 'Usman Rafiq',
 'Iqra Ahmed',
 'Taha Kamal',
 'Laiba Yousuf',
 'Hiba Noor',
 'Bilal Aslam',
 'Mariam Ali',
 'Mariam Ali']
```

2.Write a function get_highest_paying_passenger() that returns the name and destination of the passenger who paid the most.

```
def get_highest_paying_passenger():
 most_paid = df.loc[df['ticket_price'].idxmax()] #counts the highest ticekt price
 return {'name': most_paid['name'], 'destination': most_paid['destination']} #shows only the most paid passenger
get_highest_paying_passenger() #calling the func
```

```
{'name': 'Ayesha Noor', 'destination': 'Quetta'}
```

Part 3 – Sorting and Logic

1.Sort all passengers by ticket_price in descending order.

```
df.sort_values('ticket_price', ascending = False)
```

	passenger_id	name	age	gender	destination	ticket_price
14	P015	Ayesha Noor	33	M	Quetta	19950
16	P017	Ahmed Nadeem	28	M	Lahore	19780
22	P023	Ahmed Nadeem	34	M	Peshawar	19636
36	P037	Sana Javed	37	F	Quetta	19157
10	P011	Taha Kamal	38	F	Islamabad	18764
9	P010	Sarah Malik	48	F	Islamabad	18658
39	P040	Bilal Aslam	40	F	Quetta	18635
8	P009	Zainab Tariq	49	F	Quetta	18625
2	P003	Rehan Ahmed	24	M	Quetta	18227
3	P004	Ali Khan	32	M	Lahore	18130
41	P042	Fatima Hassan	24	F	Lahore	17680
1	P002	Nimra Shahid	57	M	Lahore	17048
45	P046	Bilal Aslam	59	F	Karachi	16730

2. Display the top 3 passengers with their name, age, and ticket price.

```
top_3 = df.head(3)
top_3[['name', 'age', 'ticket_price']]
```

	name	age	ticket_price
0	Usman Rafiq	23	15677
1	Nimra Shahid	57	17048
2	Rehan Ahmed	24	18227

Bonus Task – Custom Filtering

1.Ask the user to input an age threshold, and display how many passengers are younger than that age.

```
def age_threshold(age):
    age_threshold = int(input("Enter an age threshold: "))
    younger_passengers = df[df['age'] < age_threshold].shape[0] # Counts passengers younger than the threshold
    print(f"Number of passengers younger than {age_threshold}: {younger_passengers}") # Display the result
age_threshold('')
```

```
Enter an age threshold:  19
Number of passengers younger than 19: 0
```

Lab No: 6

Objective: To enable students to understand and implement the **Depth-First Search algorithm** for exploring graphs or state spaces

Practical Significance: -

Depth-First Search (DFS) is a versatile algorithm with practical significance in various domains. Here are some practical applications and use cases of DFS:

Pathfinding and Maze Solving:
DFS is commonly used to find paths and solve mazes. Its recursive nature makes it efficient in exploring paths until a solution is found.

Cycle Detection:
DFS can be applied to detect cycles in a graph. This is useful in dependency analysis, resource allocation, and preventing deadlocks in concurrent systems.

Graph Traversal:
DFS is fundamental for graph traversal and exploration. It is used in applications such as network analysis, social network mapping, and web crawling.

Puzzle Solving:
DFS is employed in solving puzzles, such as the N-Queens problem and the Tower of Hanoi. It systematically explores possible states until a solution is found.

Artificial Intelligence:
DFS is applied in AI algorithms, particularly in decision tree traversal, game playing (e.g., chess, tic-tac-toe), and state space exploration.

Anomaly Detection:
DFS can be employed in anomaly detection systems to identify unusual patterns or behaviors in data.

The practical significance of DFS lies in its ability to systematically explore and analyze complex structures, making it a valuable tool in a wide range of applications across computer science, mathematics, engineering, and artificial intelligence.

DFS Algorithm: -

Input:
Graph G represented as an adjacency list, starting vertex start, and goal vertex goal.

Initialization:
Create an empty set visited to keep track of visited vertices. Create a
deque stack and push the start vertex onto it.
Add the start vertex to the visited set.

DFS Loop:
While the stack is not empty:
Pop a vertex current_vertex from the front of the stack. Print or process current_vertex.
If current_vertex is equal to the goal vertex:
Print a message indicating that the goal state is reached. Return, indicating that the goal state is reached.
For each neighbor neighbor of current_vertex in the graph: If neighbor is not in the visited set:
Push neighbor onto the front of the stack. Add neighbor to the visited set.

Output:
Print a message indicating that the goal state is not reached if the loop completes without returning.

Tasks: -

1. Write a Program to Implement Depth First Search without goal state using Python.
2. Write a Program to Implement Depth First Search with goal state using Python.

1. Write a Program to Implement Depth First Search without goal state using Python.

Code:

```
graph = {'A': ['B', 'C', 'D'], 'B': ['E'], 'C': ['D', 'E'], 'D': [], 'E': []}
stack = []
visited = set()

def dfs(visited, stack, graph, root):
    stack.append(root)

    while stack:
        node = stack.pop()
        if node not in visited:
            visited.add(node)
            for neighbour in graph[node]:
                if neighbour not in visited:
                    stack.append(neighbour)
    print("Output", visited)

dfs(visited, stack, graph, 'A')
```

Output:

```
Output {'A', 'D', 'E', 'C', 'B'}
```

2.Write a Program to Implement Depth First Search with goal state using Python.

Code:

```
graph = {'A' : ['B', 'C', 'D'], 'B':['E'], 'C' :['D', 'E'], 'D':[], 'E':[]}
stack = []
visited = set()

def dfs(visited, stack, graph, root, goalNode):
    stack.append(root)

    while stack:
        node = stack.pop()

        if node == goalNode:
            visited.add(node)
            print("goal found:", goalNode)
            break
        elif node not in visited:
            visited.add(node)
            for neighbour in graph[node]:
                if neighbour not in visited:
                    stack.append(neighbour)
    print('Visited Nodes:', visited)

goalNode = 'E'
dfs(visited, stack, graph, 'A', goalNode)
```

Output:

```
goal found: E
Visited Nodes: {'D', 'E', 'A', 'C'}
```

Lab No: 7

Objective: To introduce students to the concept of **Best-First Search** and enable them to implement it using basic priority-based exploration

What is Best-First Search?
Best-First Search (BFS) is a search algorithm that explores a graph by selecting the most promising node based on a specific criterion. It uses a priority queue to decide the order in which nodes are explored.
When implemented without heuristics, Best-First Search can behave similarly to other uninformed search algorithms—like Breadth-First Search or Uniform Cost Search—depending on how the priority is defined.

Feature	Description
Search Type	Informed
Data Structure	Priority Queue
Goal	To reach the goal node by expanding the least costly or earliest node
Priority Basis	May use path cost (g(n)) or simple order of discovery

Example Use Case: -
A basic priority-based search where the algorithm always chooses the next node alphabetically or based on node depth (depending on the implementation) is an example of Best-First Search.

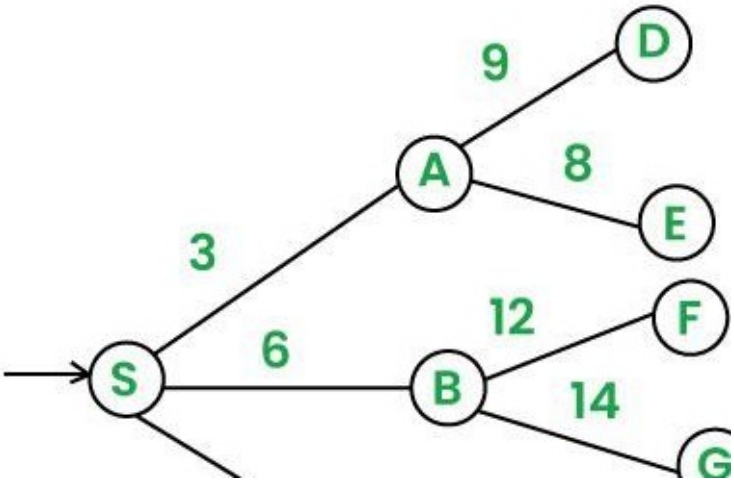
BFS Algorithm:
If we are given an edge list of a graph where every edge is represented as (u, v, w). Here u, v and w represent source, destination and weight of the edges respectively. We need to do Best First Search of the graph (Pick the minimum cost edge next).

- Initialize an empty Priority Queue named **pq**.
- Insert the starting node into **pq**.
- While **pq** is not empty:
    - Remove the node **u** with the lowest evaluation value from **pq**.
    - If **u** is the goal node, terminate the search.
    - Otherwise, for each neighbor **v** of **u**: If **v** has not been visited, Mark **v** as visited and Insert **v** into **pq**.
    - Mark **u** as examined.
- End the procedure when the goal is reached or **pq** becomes empty.

Task:
Write a Program to Implement Best First Search of the following graph from starting node "S" to goal node "I" using Python.
To help with writing the program following steps are provided for guidance:

- We start from source "S" and search for goal "I" using given costs and Best First search.
- pq initially contains S
    - We remove S from pq and process unvisited neighbors of S to pq.
    - pq now contains {A, C, B} (C is put before B because C has lesser cost)
- We remove A from pq and process unvisited neighbors of A to pq.
    - pq now contains {C, B, E, D}
- We remove C from pq and process unvisited neighbors of C to pq.
- pq now contains {B, H, E, D}
- We remove B from pq and process unvisited neighbors of B to pq.

- pq now contains {H, E, D, F, G}
- We remove H from pq.
- Since our goal "I" is a neighbor of H, we return.



Code:

```
import collections
def best_first_search(graph, start, goal):
    visited = set()
    queue = collections.deque([start])

    while queue:
        node = queue.popleft()

        if node == goal:
            visited.add(node)
            print("Goal Found:", goal)
            print("Path (explored nodes):", visited)
            return

        visited.add(node)

        neighbors = sorted(graph[node], key=lambda x: x[1])
        for neighbor, cost in neighbors:
            if neighbor not in visited and neighbor not in queue:
                queue.append(neighbor)

    print("Goal not found!")

if __name__ == "__main__":
    graph = {
        'S': [('A', 3), ('B', 6), ('C', 5)],
        'A': [('D', 9), ('E', 8)],
        'B': [('F', 12), ('G', 14)],
        'C': [('H', 7)],
```

```
    'H': [('I', 5), ('J', 6)],
    'I': [('K', 1), ('L', 10), ('M', 2)],
    'D': [],
    'E': [],
    'F': [],
    'G': [],
    'J': [],
    'K': [],
    'L': [],
    'M': []
}
best_first_search(graph, 'S', 'I')
```

Output:

```
Goal Found: I
Path (explored nodes): {'B', 'H', 'G', 'F', 'D', 'I', 'A', 'E', 'S', 'C'}
```

Lab No: 8

Objective: To implement the **A* algorithm** for finding the shortest path using both actual and heuristic costs in intelligent search problems.

What is A* Search?
A* is an informed search algorithm that finds the shortest path from a start node to a goal node by combining:

- g(n): Actual cost from the start node to the current node.
- h(n): Heuristic estimate of the cost from the current node to the goal.
- f(n) = g(n) + h(n): Total estimated cost of the cheapest solution through node n.

Key Properties of A* Search: -

Property	**Description**
Informed?	Yes – uses heuristics
Optimal?	Yes – if the heuristic is admissible (never overestimates)
Complete?	Yes
Time Complexity	Can be high depending on heuristic accuracy
Data Structure	Priority Queue based on f(n)

Use Cases in AI: -

- Pathfinding in maps or games.
- Puzzle solvers (e.g., 8-puzzle, sliding tiles).
- Planning and robotics.

A* Algorithm Steps: -

1. Initialize the open list (priority queue) with the start node.
2. Loop until the open list is empty:
    - Remove the node with the lowest f(n) from the open list.
    - If it is the goal, return the path.
    - Else, generate its neighbors.
    - For each neighbor:
        - Calculate g(n), h(n), and f(n).
        - Add to open list if not visited or if a better f(n) is found.

Task:
Write a Program to Implement A* Algorithm with goal state using Python.

Code:

```
import heapq

def a_star_search(graph, heuristics, start, goal):
    open_list = []
    heapq.heappush(open_list, (heuristics[start], 0, start, [start]))  # (f, g, node, path)
    visited = set()

    while open_list:
        f, g, current, path = heapq.heappop(open_list)
```

```
        if current == goal:
            print("Goal Found:", goal)
            print("Path:", path)
            print("Total Cost:", g)
            return

        visited.add(current)

        for neighbor, cost in graph[current]:
            if neighbor not in visited:
                g_new = g + cost
                f_new = g_new + heuristics[neighbor]
                heapq.heappush(open_list, (f_new, g_new, neighbor, path + [neighbor]))

    print("Goal not found.")

if __name__ == "__main__":
    graph = {
        'S': [('A', 1), ('B', 4)],
        'A': [('B', 2), ('C', 5)],
        'B': [('C', 1)],
        'C': []
    }
    heuristics = {
        'S': 7,
        'A': 6,
        'B': 2,
        'C': 0
    }
    a_star_search(graph, heuristics, 'S', 'C')
```

Output:

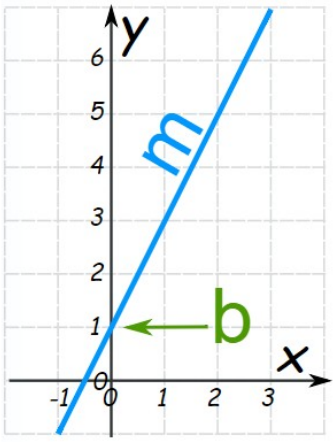
```
Goal Found: C
Path: ['S', 'B', 'C']
Total Cost: 5
```

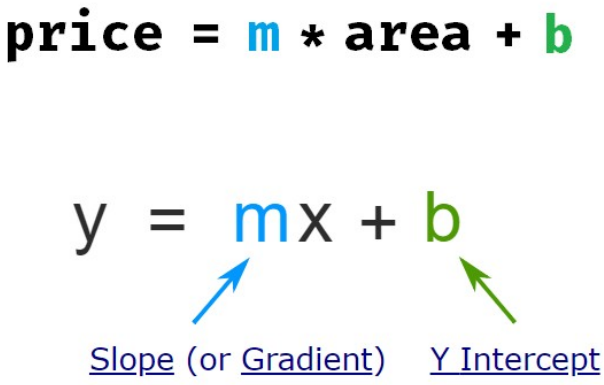
Lab No: 9

Objective: To implement **simple linear regression** and understand how it models the relationship between two variables for predictive analysis.

What is Simple Linear Regression?

Simple Linear Regression is a supervised learning algorithm that models the relationship between a dependent variable (Y) and a single independent variable (X) using a straight line.





The model has the form:

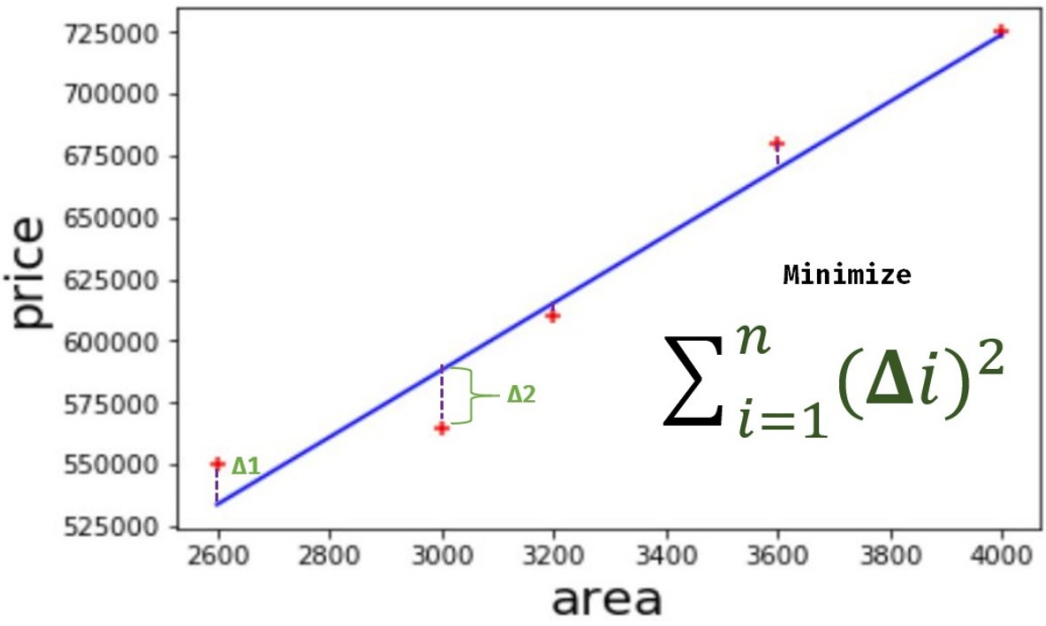
Y=mX+b

Where:

- Y = Predicted value
- X = Input feature
- m = Slope (coefficient)
- b = Intercept (bias)

Goal of the Algorithm: -

To find the best-fitting line (regression line) that minimizes the error between actual and predicted values (usually using Mean Squared Error).



Key Terms: -

- Independent variable (X) – The input or feature.
- Dependent variable (Y) – The output or label.
- Loss Function – Measures prediction error (commonly MSE).

Tasks:

Predict Canada's per capita income in year 2020. Using this build a regression model and predict the per capita income for Canadian citizens in year 2020. canada_per_capita_income_exercise.csv file has been provided for dataset.

Code:

```
import pandas as pd
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt
df = pd.read_csv('Canada_per_capita_income.csv')

df_year = df.drop('income', axis='columns')
df_income = df.income

reg = linear_model.LinearRegression()
reg.fit(df_year, df_income)

predicted_income = reg.predict([[2020]])
print(f"Predicted income for 2020: {predicted_income[0]}")
```

Output:

```
Predicted income for 2020: 41288.69409441762
```

Lab No: 10

Objective: To implement **multivariate linear regression** and understand how multiple features can be used to predict a continuous output variable.

What is Multivariate Linear Regression?

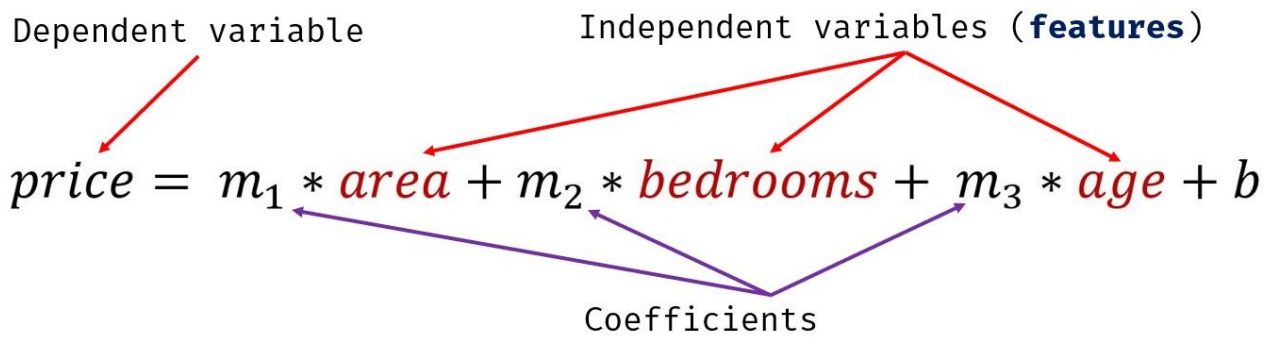
Multivariate Linear Regression extends simple linear regression by modeling the relationship between a dependent variable (Y) and multiple independent variables (X_1, X_2, ..., X_n).

The model takes the form:

$$Y=b0+b1X1+b2X2+\cdots+bnXn$$

Where:

- Y = Output (dependent variable)
- X_1 to X_n = Input features (independent variables)
- b_0 = Intercept
- b_1 to b_n = Coefficients (slopes)



$$y = m_1x_1 + m_2x_2 + m_3x_3 + b$$

Key Concepts: -

- Multiple features are used to improve prediction accuracy.
- The model learns coefficients that best fit the training data.
- Error minimization is usually done using Mean Squared Error (MSE).

Task:

There is **hiring.csv**. This file contains hiring statics for a firm such as experience of candidate, his written test score and personal interview score. Based on these 3 factors, HR will decide the salary. Given this data, you need to build a machine learning model for HR department that can help them decide salaries for future candidates. Using this predict salaries for following candidates:

- 2 yr experience, 9 test score, 6 interview score
- 12 yr experience, 10 test score, 10 interview score

Code:

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from word2number import w2n

df = pd.read_csv('hiring.csv')
print(df.head())

# Convert experience from words to numbers (handle NaN as 0)
df['experience'] = df['experience'].apply(lambda x: w2n.word_to_num(str(x)) if pd.notnull(x) else 0)

# Fill missing test scores with mean
df['test_score(out of 10)'] = df['test_score(out of 10)'].fillna(df['test_score(out of 10)'].mean())

# Features and target
X = df[['experience', 'test_score(out of 10)', 'interview_score(out of 10)']]
y = df['salary($)']

# Train the model
model = LinearRegression()
model.fit(X, y)

# Create DataFrames for predictions (with feature names to avoid warnings)
input_1 = pd.DataFrame([[2, 9, 6]], columns=['experience', 'test_score(out of 10)', 'interview_score(out of 10)'])
input_2 = pd.DataFrame([[12, 10, 10]], columns=['experience', 'test_score(out of 10)', 'interview_score(out of 10)'])

# Make predictions
pred_1 = model.predict(input_1)
pred_2 = model.predict(input_2)

# Output the predictions
print(f"Predicted salary for 2 yr experience, 9 test score, 6 interview score: ${pred_1[0]:.2f}")
print(f"Predicted salary for 12 yr experience, 10 test score, 10 interview score: ${pred_2[0]:.2f}")
```

Output:

```
  experience  test_score(out of 10)  interview_score(out of 10)  salary($)
0        NaN                    8.0                           9      50000
1        NaN                    8.0                           6      45000
2       five                    6.0                           7      60000
3        two                   10.0                          10      65000
4      seven                    9.0                           6      70000
Predicted salary for 2 yr experience, 9 test score, 6 interview score: $53290.89
Predicted salary for 12 yr experience, 10 test score, 10 interview score: $92268.07
```

Lab No: 11

Objective:
To find the shortest path from Building **A** to **G** in a university campus using **Depth First Search (DFS)** and **A* Search**.

Time: 60 minutes
Total Marks: 10
Campus Path Finder using Search Algorithms
A university campus has 7 buildings connected via paths. The connections and distances (in meters) are given in a dictionary:

```
graph = {
'A': [('B', 6), ('C', 2)],
'B': [('D', 5), ('E', 3)],
'C': [('F', 4)],
'D': [('G', 2)],
'E': [('G', 6)],
'F': [('G', 1)],
'G': []
}
```

Q) The goal is to find the shortest path from Building 'A' to 'G'.

Tasks:

1. Implement Depth First Search to find any path from A to G showing path and total distance.

Code:

```
graph = {
   'A': [('B', 6), ('C', 2)],
   'B': [('D', 5), ('E', 3)],
   'C': [('F', 4)],
   'D': [('G', 2)],
   'E': [('G', 6)],
   'F': [('G', 1)],
   'G': []
}

visited = set()

def dfs(visited, graph, root, goalNode, cost_so_far):
   if root == goalNode:
      print("Goal Node found!", goalNode)
      print("Total Cost:", cost_so_far)
      return True

   elif root not in visited:
      print(root)
      visited.add(root)
      for neighbour, cost in graph[root]:
         if dfs(visited, graph, neighbour, goalNode, cost_so_far + cost):
```

```
            return True

    return False

goalNode = 'G'
dfs(visited, graph, 'A', goalNode, 0)
```

Output:

```
A
B
D
Goal Node found! G
Total Cost: 13
```

2. Implement A* Search, assuming the following heuristic values showing path and total distance:

h = {'A': 7, 'B': 6, 'C': 4, 'D': 3, 'E': 5, 'F': 2, 'G': 0}

Code:

```
import heapq

def a_star_search(graph, heuristics, start, goal):
    pq = []
    heapq.heappush(pq, (heuristics[start], 0, start, [start]))  # (f, g, node, path)
    visited = set()

    while pq:
        f, g, node, path = heapq.heappop(pq)

        if node == goal:
            print("Goal Found:", goal)
            print("Path:", path)
            print("Total Cost:", g)
            return

        visited.add(node)

        for neighbor, cost in graph[node]:
            if neighbor not in visited:
                g_new = g + cost
                f_new = g_new + heuristics[neighbor]
                heapq.heappush(pq, (f_new, g_new, neighbor, path + [neighbor]))

    print("Goal not found.")

if __name__ == "__main__":
    graph = {
        'A': [('B', 6), ('C', 2)],
        'B': [('D', 5), ('E', 3)],
```

```
    'C': [('F', 4)],
    'D': [('G', 2)],
    'E': [('G', 6)],
    'F': [('G', 1)],
    'G': []
}

heuristics = {
    'A': 7,
    'B': 6,
    'C': 4,
    'D': 3,
    'E': 5,
    'F': 2,
    'G': 0
}

a_star_search(graph, heuristics, 'A', 'G')
```

Output:

```
Goal Found: G
Path: ['A', 'C', 'F', 'G']
Total Cost: 7
```

Lab No: 12

Objective: To implement **logistic regression** for binary classification tasks and understand how it models the probability of class membership.

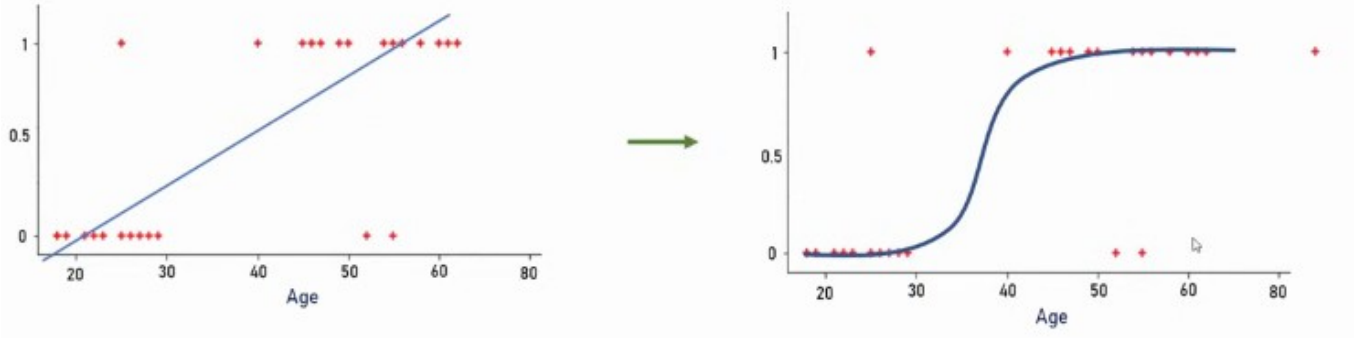
What is Logistic Regression?

Logistic Regression is a supervised learning algorithm used for binary classification. It predicts the probability that a given input belongs to a particular class (typically 0 or 1).

Unlike linear regression, it uses the sigmoid (logistic) function to map predicted values to a probability between 0 and 1.

$$y = m * x + b$$

$$y = \frac{1}{1 + e^{-(m*x+b)}}$$



Sigmoid Function: -

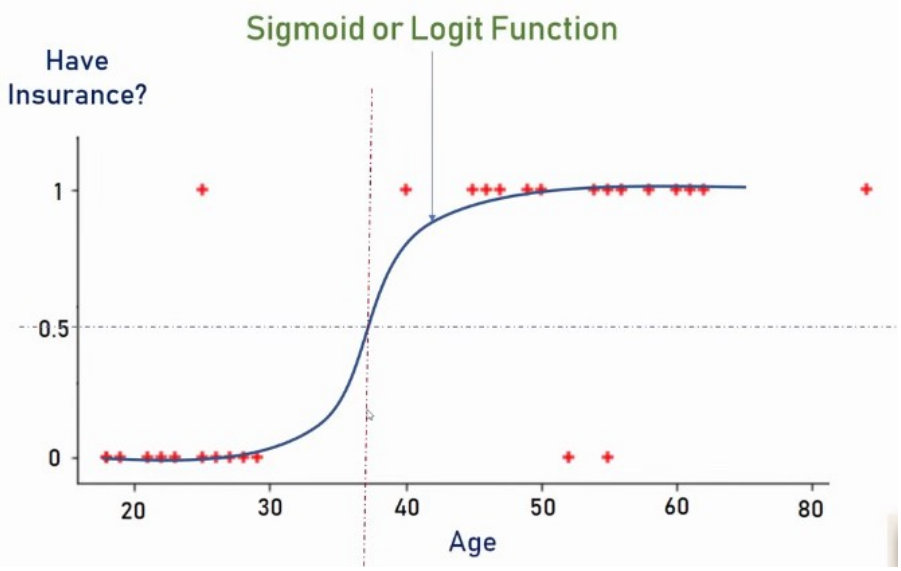
$$sigmoid(z) = \frac{1}{1 + e^{-z}}$$

e = Euler's number ~ 2.71828

Sigmoid function converts input into range 0 to 1

Where:

- $z = w_0 + w_1x_1 + w_2x_2 + ... + w_nx_n$ (linear combination of features)
- Output: probability (e.g., if > 0.5 → class 1, else class 0)



Key Concepts

- Output is a probability score.
- Decision boundary separates the two classes (e.g., at 0.5).
- Loss function used is log loss or binary cross-entropy.

Tasks:

Download employee retention dataset from here: https://www.kaggle.com/giripujar/hr-analytics.

1. Now do some exploratory data analysis to figure out which variables have direct and clear impact on employee retention (i.e. whether they leave the company or continue to work)
2. Plot bar charts showing impact of employee salaries on retention
3. Plot bar charts showing correlation between department and employee retention
4. Now build logistic regression model using variables that were narrowed down in step 1
5. Measure the accuracy of the model

1.Now do some exploratory data analysis to figure out which variables have direct and clear impact on employee retention (i.e. whether they leave the company or continue to work)

Code:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

df = pd.read_csv('HR_comma_sep.csv')
print(df.head())
print("\nCorrelation of numeric features with 'left':\n")
print(df.select_dtypes(include=['float64', 'int64']).corr()['left'].sort_values(ascending=False)
```

Output:

```
Correlation of numeric features with 'left':

left                    1.000000
time_spend_company      0.144822
average_montly_hours    0.071287
number_project          0.023787
last_evaluation         0.006567
promotion_last_5years  -0.061788
Work_accident          -0.154622
satisfaction_level     -0.388375
Name: left, dtype: float64
```

2.Plot bar charts showing impact of employee salaries on retention

Code:

```
pd.crosstab(df.salary, df.left).plot(kind='bar')
plt.xlabel('Salary Level')
plt.ylabel('Number of Employees')
plt.title('Salary vs Retention')
plt.show()
```

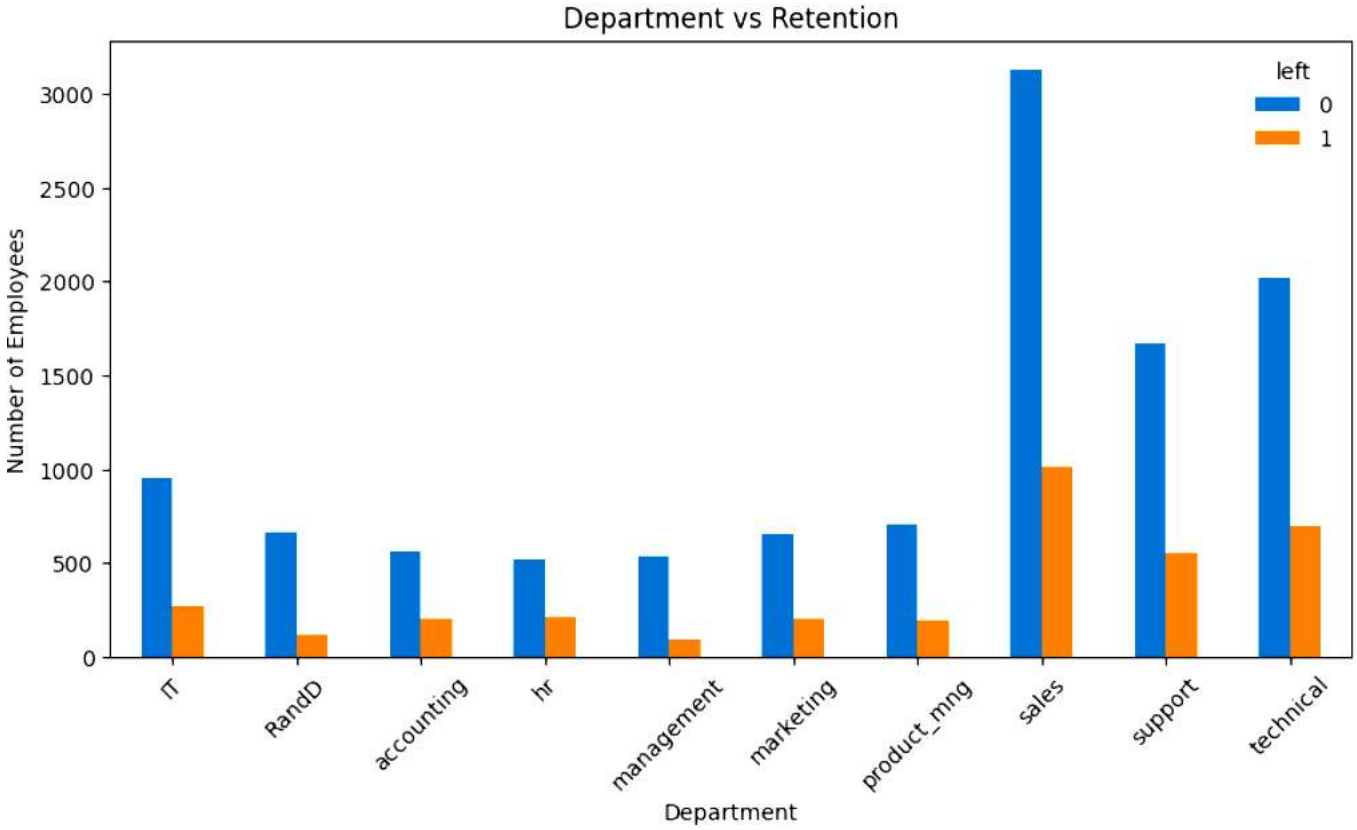
Output:

3.Plot bar charts showing correlation between department and employee retention

Code:

```
pd.crosstab(df.Department, df.left).plot(kind='bar', figsize=(10, 5))
plt.xlabel('Department')
plt.ylabel('Number of Employees')
plt.title('Department vs Retention')
plt.xticks(rotation=45)
plt.show()
```

Output:



4.Now build logistic regression model using variables that were narrowed down in step 1

Code:

```
X = pd.get_dummies(df.drop('left', axis=1), drop_first=True)
y = df.left

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
```

Output:

```
LogisticRegression
LogisticRegression(max_iter=1000)
```

5.Measure the accuracy of the model

Code:

```
y_pred = model.predict(X_test)
y_proba = model.predict_proba(X_test)
print("\nFirst 5 probability predictions:\n", y_proba[:5])
accuracy = accuracy_score(y_test, y_pred)
print(f"\nModel accuracy: {accuracy:.2f}")
```

Output:

```
First 5 probability predictions:
 [[0.9718063  0.0281937 ]
 [0.92091956 0.07908044]
 [0.76880205 0.23119795]
 [0.52233512 0.47766488]
 [0.97592189 0.02407811]]

Model accuracy: 0.78
```