Artificial Intelligence

(Practical Manual)

4^{th} Semester, 2^{nd} 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. ___________Kainat Moin_______with Roll # 23-AI-48 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.

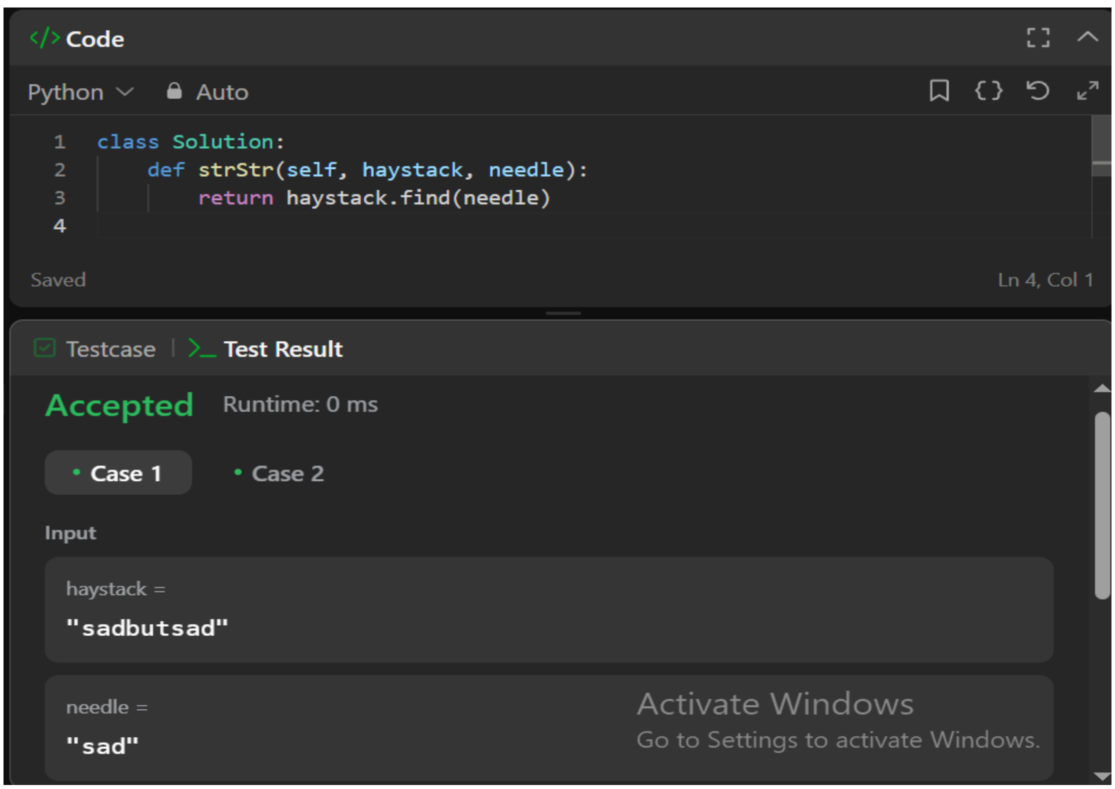
Example 2:

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

Output: -1

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

<u>Output:</u>



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?

```
import numpy as np

# Example arrays
a = np.array([1, 2, 3, 4, 5])
b = np.array([4, 5, 6, 7])

# Common items
common_items = np.intersect1d(a, b)
print("Common items:", common_items)
```

```
Common items: [4 5]
```

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

```
# Arrays
a = np.array([1, 2, 3, 4, 5])
b = np.array([1, 0, 3, 0, 5])

# Matching positions
matching_positions = np.where(a == b)[0]
print("Matching positions:", matching_positions)
```

```
Matching positions: [0 2 4]
```

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

```
# Example array and range
arr = np.array([10, 25, 30, 5, 50, 60])
low, high = 20, 50

# Numbers within the range [20, 50]
in_range = arr[(arr >= low) & (arr <= high)]
print(f"Numbers between {low} and {high}:", in_range)
```

```
Numbers between 20 and 50: [25 30 50]
```

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

```python
# Example 1D array and window size
data = np.array([1, 2, 3, 4, 5, 6, 7])
window_size = 3

# Moving average
moving_avg = np.convolve(data, np.ones(window_size)/window_size, mode='valid')
print("Moving average:", moving_avg)
```

```
Moving average: [2. 3. 4. 5. 6.]
```

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]  import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns


     %matplotlib inline
```

```
[3]  df=pd.read_csv('imdb_top_1000.csv')
```

2. Display the first 5 rows.

```
df.head()
```

	Poster_Link	Series_Title	Released_Year	Certificate	Runtime	Genre	IMDB_Rating	Overview	Meta_score	Director
0	https://m.media-amazon.com/images/M/MV5BMDFkYT...	The Shawshank Redemption	1994	A	142 min	Drama	9.3	Two imprisoned men bond over a number of years...	80.0	Frank Darabont
1	https://m.media-amazon.com/images/M/MV5BM2MyNj...	The Godfather	1972	A	175 min	Crime, Drama	9.2	An organized crime dynasty's aging patriarch t...	100.0	Francis Ford Coppola
2	https://m.media-amazon.com/images/M/MV5BMTMxNT...	The Dark Knight	2008	UA	152 min	Action, Crime, Drama	9.0	When the menace known as the Joker wreaks havo...	84.0	Christopher Nolan

3. Check the data types of each column.

```
df.dtypes
```

	0
Poster_Link	object
Series_Title	object
Released_Year	object
Certificate	object
Runtime	object
Genre	object
IMDB_Rating	float64
Overview	object
Meta_score	float64
Director	object
Star1	object
Star2	object
Star3	object
Star4	object

4. Find the number of rows and columns.

```
[ ]  df.shape

(1000, 16)
```

5. Check for missing values.

```
df.isnull().sum()
```

	0
Poster_Link	0
Series_Title	0
Released_Year	0
Certificate	101
Runtime	0
Genre	0
IMDB_Rating	0
Overview	0
Meta_score	157
Director	0
Star1	0
Star2	0
Star3	0
Star4	0

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.

```
[ ]  df.drop_duplicates(inplace=True)

[10]  df['IMDB_Rating'].fillna(df['IMDB_Rating'].mean(),inplace=True)

ython-input-10-3077144680.py:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained as
avior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting val

mple, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].m

MDB_Rating'].fillna(df['IMDB_Rating'].mean(),inplace=True)

df['Runtime'] = pd.to_numeric(df['Runtime'].astype(str).str.replace('min', '', regex=False), errors='coerce')
df['Runtime']
```

	Runtime
0	142.0
1	175.0
2	152.0
3	202.0
4	96.0

Task 3: Data Filtering & Sorting

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

```python
high_rating=df[df['IMDB_Rating']>8.5]
```

```python
high_rating.head()
```

	Poster_Link	Series_Title	Released_Year	Certificate	Runtime	Genre	IMDB_Rating	Overview	Meta_score	Director	St
0	https://m.media-amazon.com/images/M/MV5BMDFkYT...	The Shawshank Redemption	1994	A	142	Drama	9.3	Two imprisoned men bond over a number of years...	80.0	Frank Darabont	Robb
1	https://m.media-amazon.com/images/M/MV5BM2MyNj...	The Godfather	1972	A	175	Crime, Drama	9.2	An organized crime dynasty's aging patriarch t...	100.0	Francis Ford Coppola	Mar Brar
2	https://m.media-amazon.com/images/M/MV5BMTMxNT...	The Dark Knight	2008	UA	152	Action, Crime, Drama	9.0	When the menace known as the Joker wreaks havo...	84.0	Christopher Nolan	Christ E

Variables Terminal 17:23 Python 3

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

```python
action_scifi=df[df["Genre"].str.contains('Action|Sci-Fi',na=False)]
```

```python
action_scifi
```

5	https://m.media-amazon.com/images/M/MV5BNzA5ZD...	the Rings: The Return of the King	2003	U	201	Adventure, Drama	8.9	Aragorn lead the World of Men agai...	94.0	Peter Jackson
8	https://m.media-amazon.com/images/M/MV5BMjAxMz...	Inception	2010	UA	148	Action, Adventure, Sci-Fi	8.8	A thief who steals corporate secrets through t...	74.0	Christopher Nolan
10	https://m.media-amazon.com/images/M/MV5BN2EyZj...	The Lord of the Rings: The Fellowship of the Ring	2001	U	178	Action, Adventure, Drama	8.8	A meek Hobbit from the Shire and eight compani...	92.0	Peter Jackson
13	https://m.media-amazon.com/images/M/MV5BZGMxZT...	The Lord of the Rings: The Two Towers	2002	UA	179	Action, Adventure, Drama	8.7	While Frodo and Sam edge closer to Mordor with...	87.0	Peter Jackson

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

```python
df['Year'] = pd.to_numeric(df['Released_Year'], errors='coerce')  # Converts invalid entries to NaN
movies_2000_2015 = df[(df['Year'] >= 2000) & (df['Year'] <= 2015)]

df['Year']
```

	Year
0	1994.0
1	1972.0
2	2008.0
3	1974.0
4	1957.0
...	...
995	1961.0
996	1956.0
997	1953.0
998	1944.0
999	1935.0

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

```python
[22] sorted_df=df.sort_values(by='IMDB_Rating',ascending=False)

sorted_df
```

	Poster_Link	Series_Title	Released_Year	Certificate	Runtime	Genre	IMDB_Rating	Overview	Meta_score	Director
0	https://m.media-amazon.com/images/M/MV5BMDFkYT...	The Shawshank Redemption	1994	A	142	Drama	9.3	Two imprisoned men bond over a number of years...	80.0	Frank Darabont
1	https://m.media-amazon.com/images/M/MV5BM2MyNj...	The Godfather	1972	A	175	Crime, Drama	9.2	An organized crime dynasty's aging patriarch t...	100.0	Francis Ford Coppola
4	https://m.media-amazon.com/images/M/MV5BMWU4N2...	12 Angry Men	1957	U	96	Crime, Drama	9.0	A jury holdout attempts to prevent miscarria...	96.0	Sidney Lumet

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.

```python
# 1. Average IMDb rating for each genre
avg_rating_by_genre = df.groupby('Genre')['IMDB_Rating'].mean()

# 2. Year with the most movies released
df['Year'] = pd.to_numeric(df['Released_Year'], errors='coerce')  # Handle invalid years
most_movies_year = df['Year'].value_counts().idxmax()

# 3. Top 5 directors by number of movies
top_5_directors = df['Director'].value_counts().head(5)

# Display all results
print("1. Average IMDb rating by Genre:\n", avg_rating_by_genre)
print("\n2. Year with most movies released:", int(most_movies_year))
print("\n3. Top 5 Directors by number of movies:\n", top_5_directors)
```

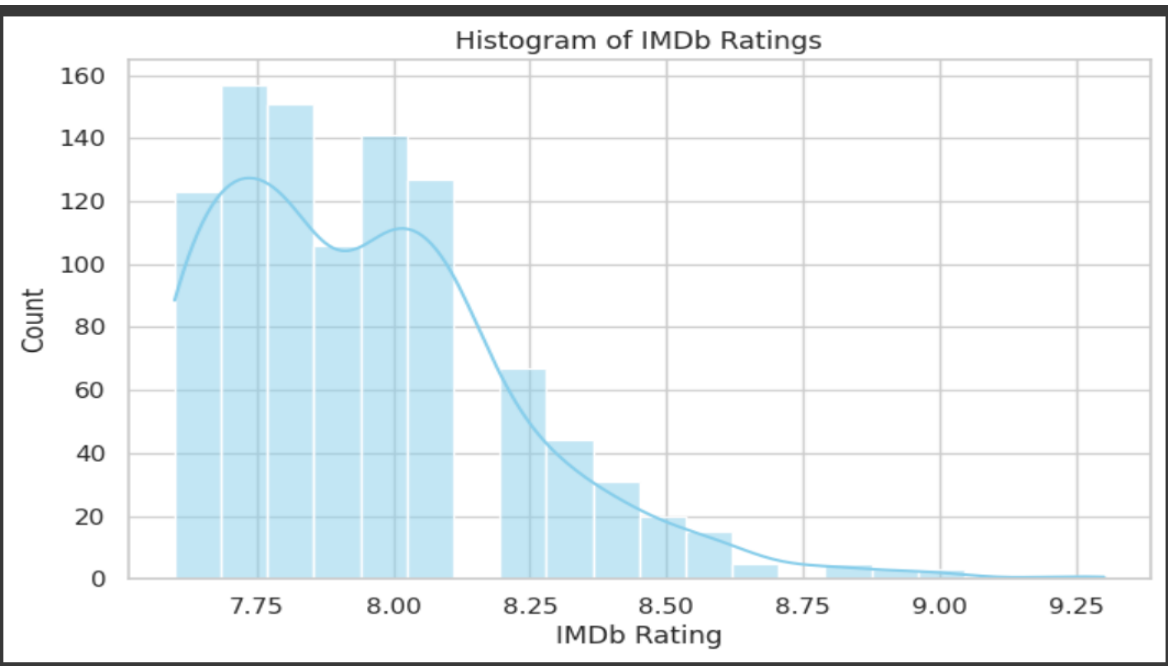
```
1. Average IMDb rating by Genre:
 Genre
Action, Adventure              8.180000
Action, Adventure, Biography   7.900000
Action, Adventure, Comedy      7.910000
Action, Adventure, Crime       7.600000
Action, Adventure, Drama       8.150000
                                 ...
Mystery, Romance, Thriller     8.300000
Mystery, Sci-Fi, Thriller      7.800000
Mystery, Thriller              7.977778
Thriller                       7.800000
Western                        8.350000
Name: IMDB_Rating, Length: 202, dtype: float64

2. Year with most movies released: 2014

3. Top 5 Directors by number of movies:
 Director
Alfred Hitchcock     14
Steven Spielberg     13
Hayao Miyazaki       11
Akira Kurosawa       10
Martin Scorsese      10
Name: count, dtype: int64
```

Visualization (Optional)

Matplotlib or **Seaborn**

1. Plot a histogram of IMDb ratings.

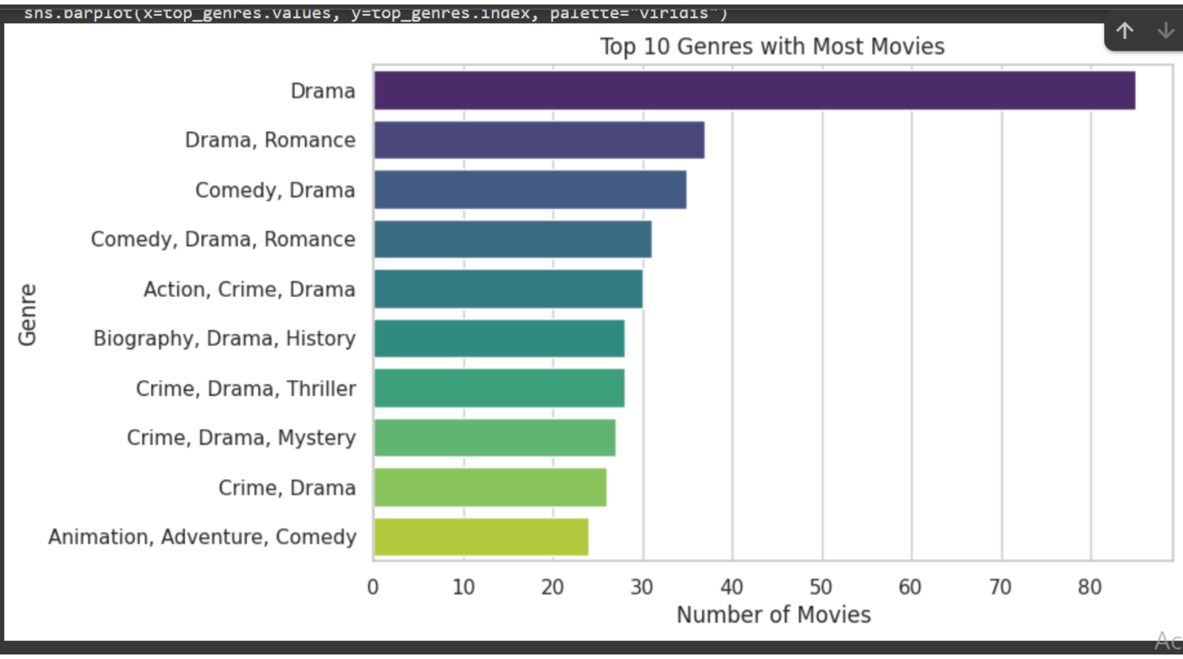
```python
import matplotlib.pyplot as plt
import seaborn as sns

# Set style
sns.set(style="whitegrid")

# Plot histogram
plt.figure(figsize=(8, 5))
sns.histplot(df['IMDB_Rating'].dropna(), bins=20, kde=True, color='skyblue')
plt.title('Histogram of IMDb Ratings')
plt.xlabel('IMDb Rating')
plt.ylabel('Count')
plt.show()
```



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

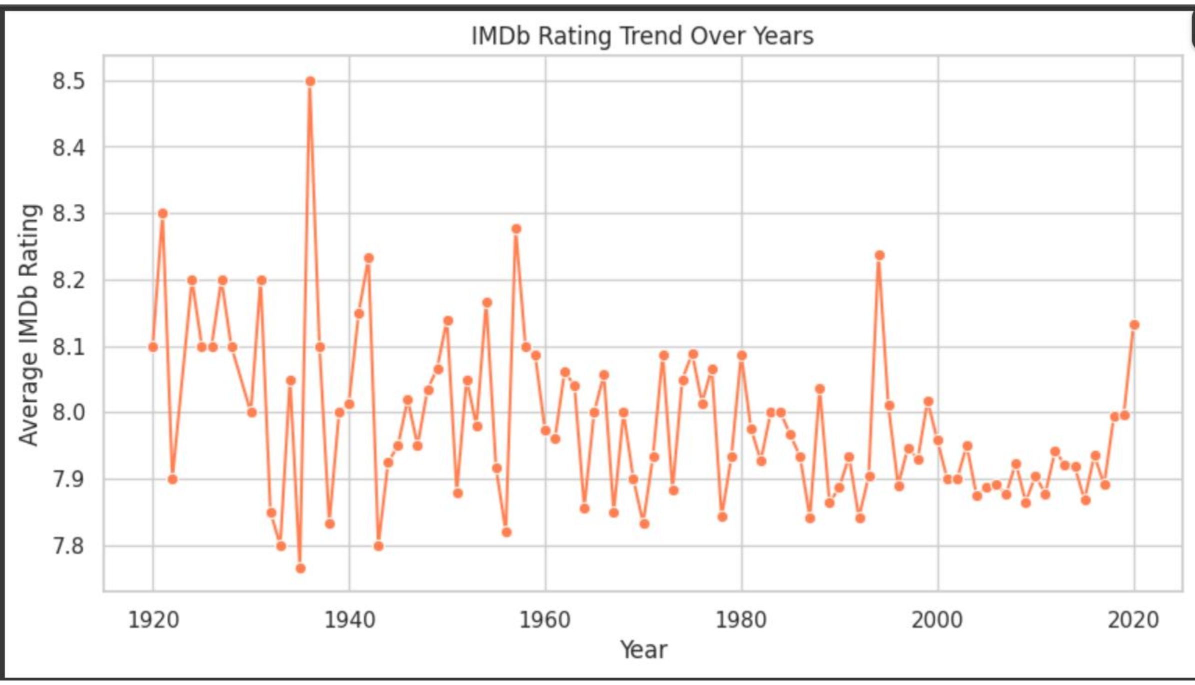
```python
# Count top 10 genres
top_genres = df['Genre'].value_counts().head(10)

# Plot bar chart
plt.figure(figsize=(8, 5))
sns.barplot(x=top_genres.values, y=top_genres.index, palette="viridis")
plt.title('Top 10 Genres with Most Movies')
plt.xlabel('Number of Movies')
plt.ylabel('Genre')
plt.show()
```

```
sns.barplot(x=top_genres.values, y=top_genres.index, palette="viridis")
```



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

```
# Ensure year is numeric
df['Year'] = pd.to_numeric(df['Released_Year'], errors='coerce')

# Group and calculate average rating per year
ratings_by_year = df.groupby('Year')['IMDB_Rating'].mean().dropna()

# Plot trend line
plt.figure(figsize=(10, 5))
sns.lineplot(x=ratings_by_year.index, y=ratings_by_year.values, marker='o', color='coral')
plt.title('IMDb Rating Trend Over Years')
plt.xlabel('Year')
plt.ylabel('Average IMDb Rating')
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.

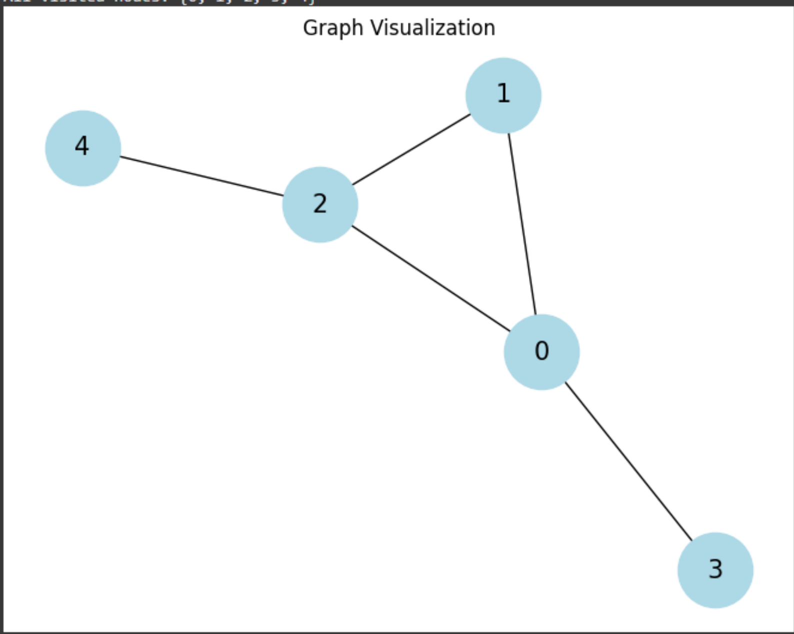
Tasks: -

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

```python
import collections
import networkx as nx
import matplotlib.pyplot as plt

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

    while queue:
        vertex = queue.popleft()
        if vertex not in visited:
            print("Visiting:", vertex)
            visited.add(vertex)
            for neighbor in graph[vertex]:
                if neighbor not in visited and neighbor not in queue:
                    queue.append(neighbor)

    print("All visited nodes:", visited)

if __name__ == "__main__":
    # Define graph as adjacency list
    graph_dict = {
        0: [1, 2, 3],
        1: [0, 2],
        2: [0, 1, 4],
        3: [0],
        4: [2]
    }

    # Create graph with networkx
    G = nx.Graph()
    for node, neighbors in graph_dict.items():
        for neighbor in neighbors:
            G.add_edge(node, neighbor)
```

```python
# Run BFS starting from node 0
bfs(graph_dict, 0)

# Visualize the graph
nx.draw(G, with_labels=True, node_color='lightblue', edge_color='black', node_size=2000, font_size=15)
plt.title("Graph Visualization")
plt.show()
```

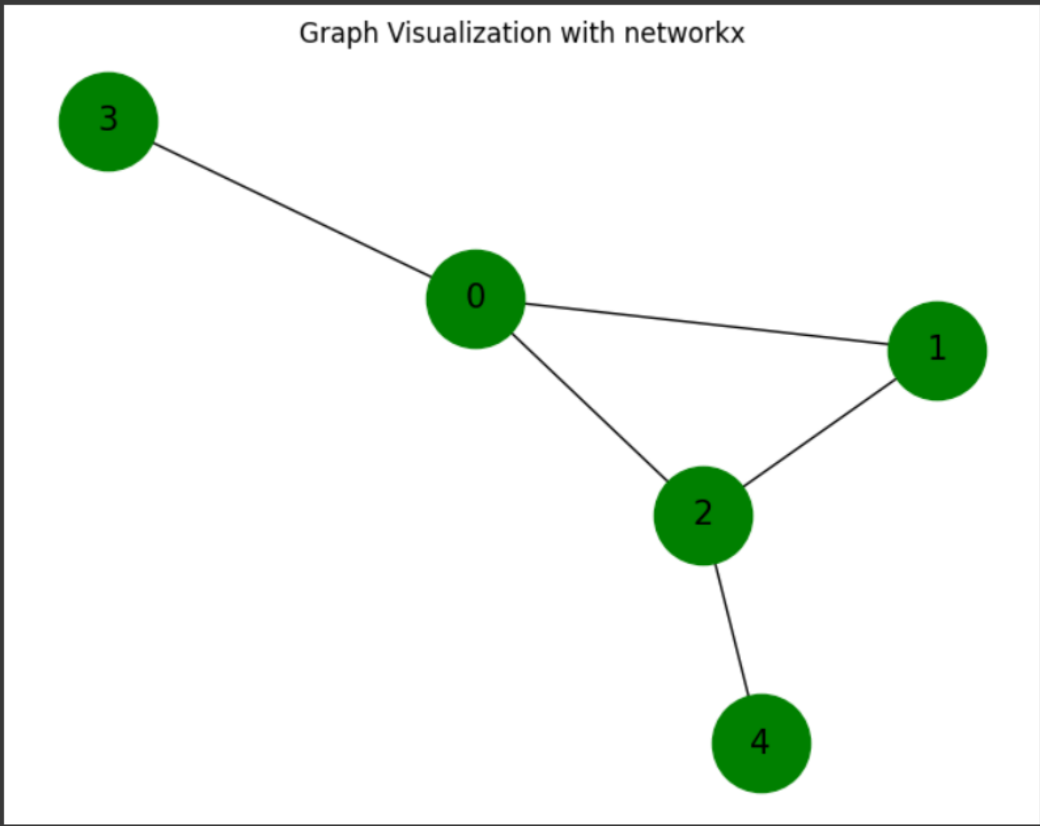
```
Visiting: 0
Visiting: 1
Visiting: 2
Visiting: 3
Visiting: 4
All visited nodes: {0, 1, 2, 3, 4}
```



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

```python
import collections
import networkx as nx
import matplotlib.pyplot as plt

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

    while queue:
        vertex = queue.popleft()
        if vertex == goal:
            print("Goal state found:", vertex)
            return
        print("Visiting:", vertex)
        visited.add(vertex)
        for neighbor in graph[vertex]:
            if neighbor not in visited and neighbor not in queue:
                queue.append(neighbor)

    print("Visited nodes:", visited)

if __name__ == "__main__":
    # Define graph as adjacency list
    graph_dict = {
        0: [1, 2, 3],
        1: [0, 2],
        2: [0, 1, 4],
        3: [0],
        4: [2]
    }

    # Create graph with networkx
    G = nx.Graph()
    for node, neighbors in graph_dict.items():
        for neighbor in neighbors:
            G.add_edge(node, neighbor)

    # Run BFS
    bfs(graph_dict, 0, 3)

    # Visualize the graph
    nx.draw(G, with_labels=True, node_color='green', edge_color='black', node_size=2000, font_size=15)
    plt.title("Graph Visualization with networkx")
    plt.show()
```

```
Visiting: 0
Visiting: 1
Visiting: 2
Goal state found: 3
```



Lab No: 5

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.

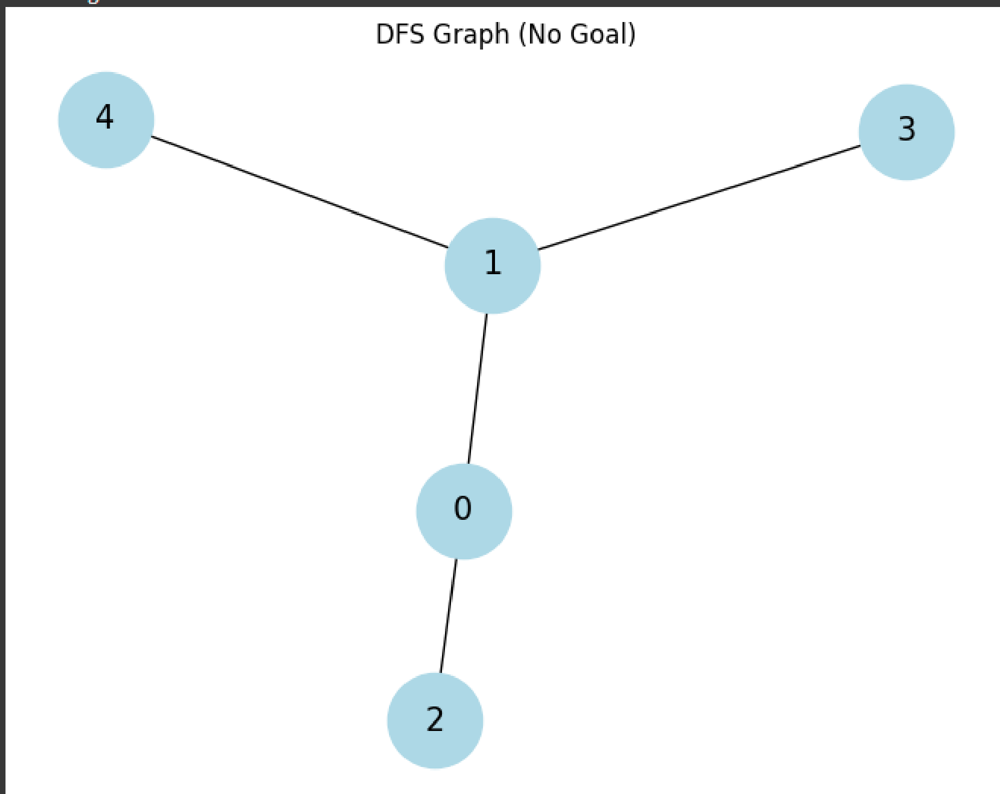
Output:
Print a message indicating that the goal state is not reached if the loop completes without returning.

Tasks: -

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

```
import networkx as nx
import matplotlib.pyplot as plt

def dfs(graph, start, visited=None):
    if visited is None:
        visited = set()
    print("Visiting:", start)
    visited.add(start)
    for neighbor in graph[start]:
        if neighbor not in visited:
            dfs(graph, neighbor, visited)
    return visited

if __name__ == "__main__":
    # Define the graph as an adjacency list
    graph = {
        0: [1, 2],
        1: [0, 3, 4],
        2: [0],
        3: [1],
        4: [1]
    }

    # Run DFS
    dfs(graph, 0)

    # Visualize the graph
    G = nx.Graph()
    for node, neighbors in graph.items():
        for neighbor in neighbors:
            G.add_edge(node, neighbor)

    nx.draw(G, with_labels=True, node_color='lightblue', edge_color='black', node_size=2000, font_size=15)
    plt.title("DFS Graph")
    plt.show()
```

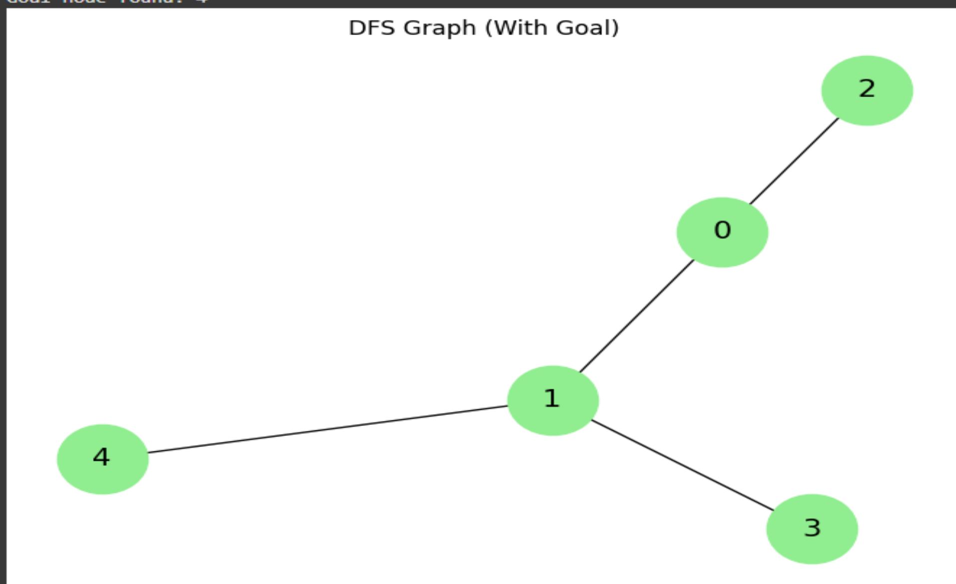
```
Visiting: 0
Visiting: 1
Visiting: 3
Visiting: 4
Visiting: 2
```



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

```python
import networkx as nx
import matplotlib.pyplot as plt

def dfs_with_goal(graph, start, goal, visited=None):
    if visited is None:
        visited = set()
    print("Visiting:", start)
    if start == goal:
        print("Goal node found:", goal)
        return True
    visited.add(start)
    for neighbor in graph[start]:
        if neighbor not in visited:
            if dfs_with_goal(graph, neighbor, goal, visited):
                return True
    return False

if __name__ == "__main__":
    # Define the graph as an adjacency list
    graph = {
        0: [1, 2],
        1: [0, 3, 4],
        2: [0],
        3: [1],
        4: [1]
    }

    # Run DFS with goal node
    dfs_with_goal(graph, 0, 4)

    # Visualize the graph
    G = nx.Graph()
    for node, neighbors in graph.items():
        for neighbor in neighbors:
            G.add_edge(node, neighbor)

    nx.draw(G, with_labels=True, node_color='lightgreen', edge_color='black', node_size=2000, font_size=15)
    plt.title("DFS Graph (With Goal)")
    plt.show()
```

```
Visiting: 0
Visiting: 1
Visiting: 3
Visiting: 4
Goal node found: 4
```



Lab No : 06

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.

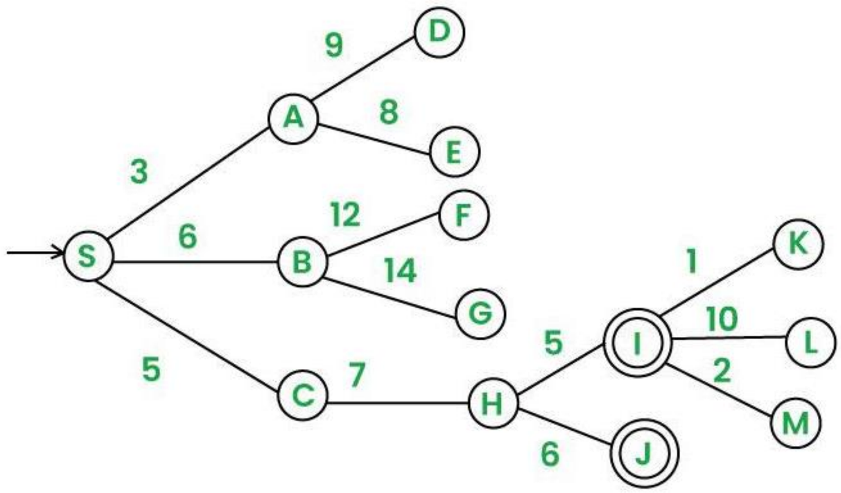
BFS Algorithm:
If we are given an edge list of a graph where every edge is represented as (u, v, w). Here u, v and w represent source, destination and weight of the edges respectively. We need to do Best First Search of the graph (Pick the minimum cost edge next).

- 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.




```python
import heapq

# Define the graph (adjacency list)
graph = {
    'S': ['A', 'B', 'C'],
    'A': ['D', 'E'],
    'B': ['F', 'G'],
    'C': ['H'],
    'H': ['I', 'J'],
    'I': ['K', 'L'],
    'L': ['M'],
    'D': [], 'E': [], 'F': [], 'G': [], 'J': [], 'K': [], 'M': []
}

# Heuristic values for Best First Search (assumed for demonstration)
heuristic = {
    'S': 10,
    'A': 9,
    'B': 7,
    'C': 8,
    'D': 999,   # high cost as not useful in path
    'E': 999,
    'F': 999,
    'G': 999,
    'H': 6,
    'I': 0,     # Goal node
    'J': 999,
    'K': 999,
    'L': 999,
    'M': 999
}

def best_first_search(graph, start, goal, heuristic):
    visited = set()
    pq = []
    heapq.heappush(pq, (heuristic[start], start))

    while pq:
        h, current = heapq.heappop(pq)
        print(f"Visiting: {current}")
        visited.add(current)

        if current == goal:
            print(f" Goal node '{goal}' found!")
            return

        for neighbor in graph.get(current, []):
            if neighbor not in visited:
                heapq.heappush(pq, (heuristic[neighbor], neighbor))

    print(" Goal node not found.")

# Run Best First Search from S to I
best_first_search(graph, 'S', 'I', heuristic)
```

```
Visiting: S
Visiting: B
Visiting: C
Visiting: H
Visiting: I
 Goal node 'I' found!
```

Lab No: 7

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.

```python
import heapq

# Small graph
graph = {
    'A': [('B', 1), ('C', 4)],
    'B': [('D', 5)],
    'C': [('D', 1)],
    'D': []
}

# Heuristic values
h = {
    'A': 7,
    'B': 6,
    'C': 2,
    'D': 0
}

def a_star(start, goal):
    open_list = []
    heapq.heappush(open_list, (h[start], 0, start, [start]))  # (f, g, node, path)

    while open_list:
        f, g, node, path = heapq.heappop(open_list)
        print("Visiting:", node)

        if node == goal:
            print(" Goal found!")
            print("Path:", " → ".join(path))
            return

        for neighbor, cost in graph[node]:
            g_new = g + cost
            f_new = g_new + h[neighbor]
            heapq.heappush(open_list, (f_new, g_new, neighbor, path + [neighbor]))

    print("Goal not found.")

# Run it
a_star('A', 'D')
```

Output:

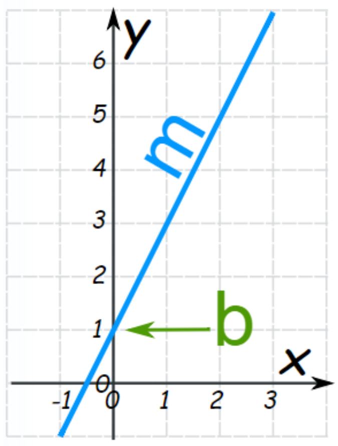
```
Visiting: A
Visiting: C
Visiting: D
 Goal found!
Path: A → C → D
```

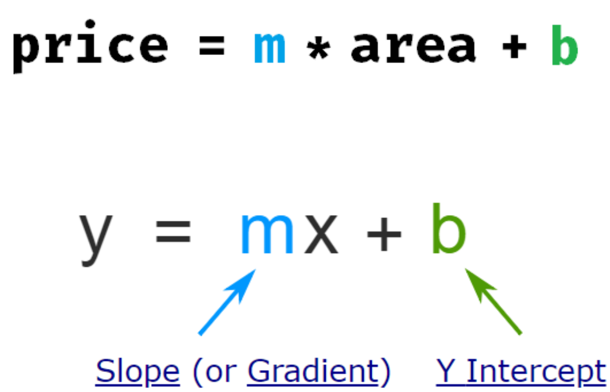
Lab No: 8

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:

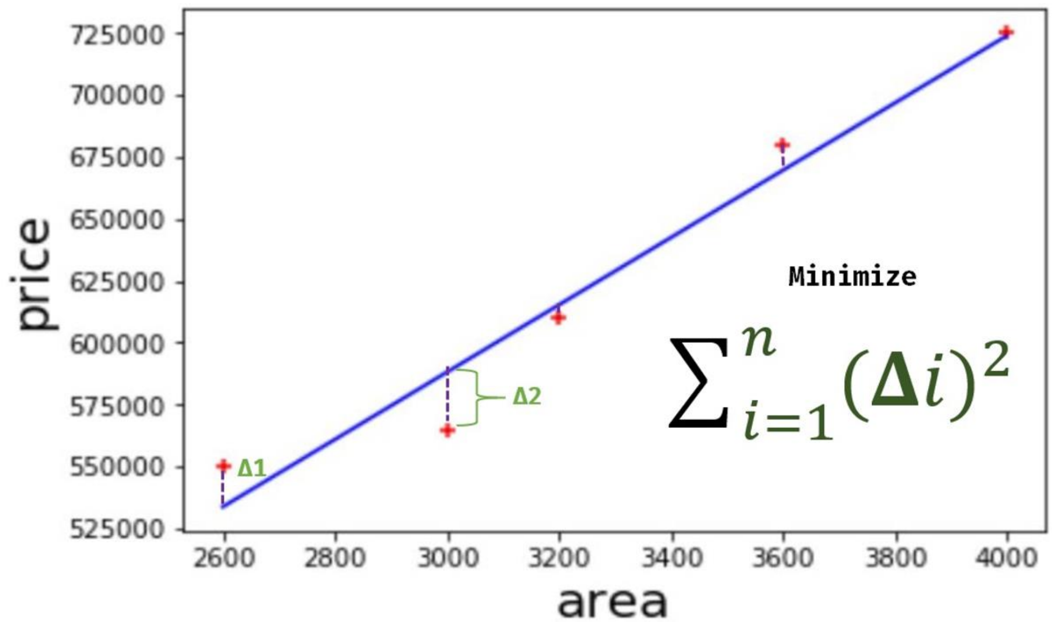
Y=mX+b

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: -

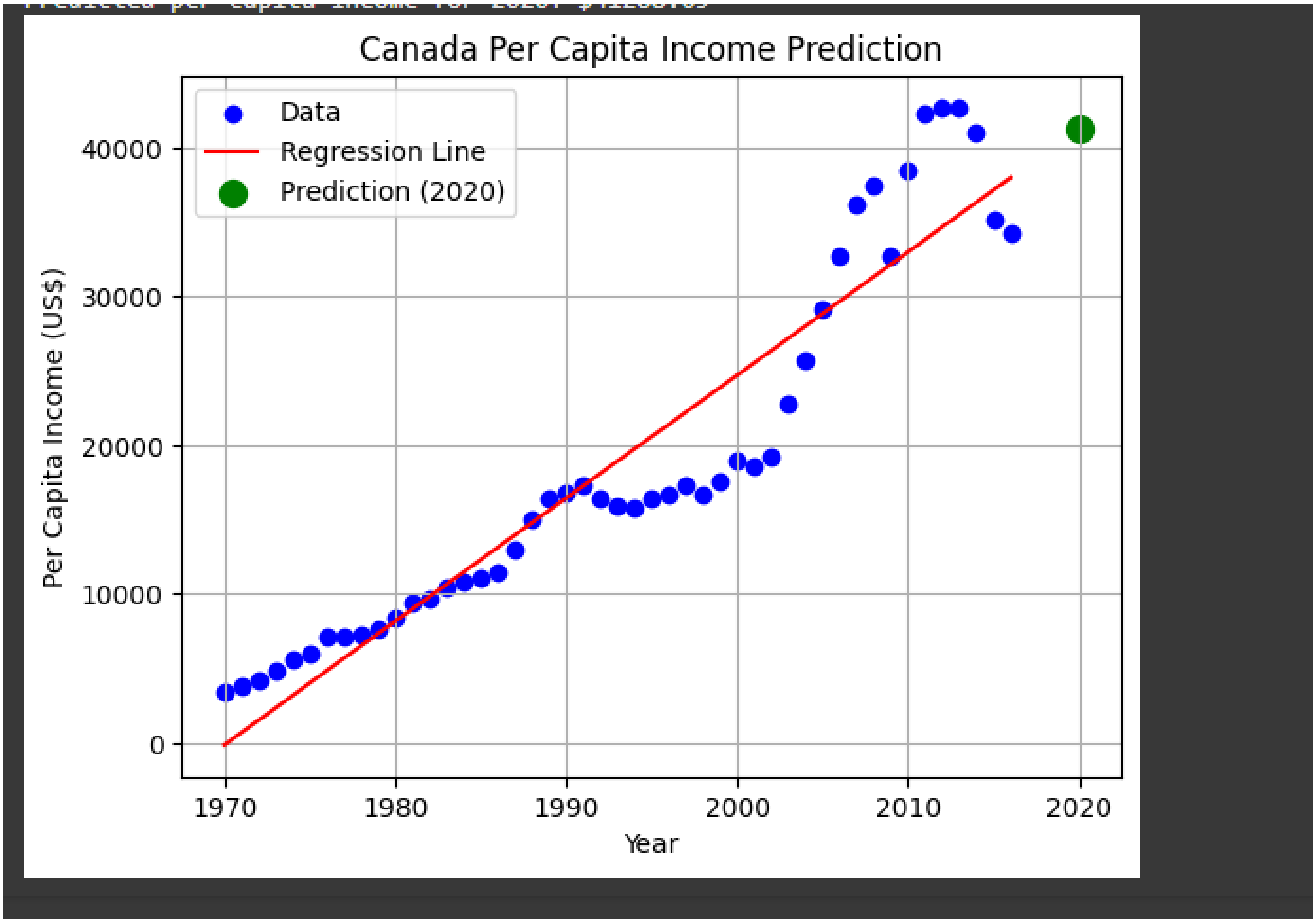
- Independent variable (X) – The input or feature.
- Dependent variable (Y) – The output or label.
- Loss Function – Measures prediction error (commonly MSE).

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.

```python
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Load the dataset
df = pd.read_csv("canada_per_capita_income_exercise.csv")

# Prepare the data
X = df[['year']]      # Input
y = df['per capita income (US$)']  # Output

# Create and train the model
model = LinearRegression()
model.fit(X, y)

# Predict income for 2020
income_2020 = model.predict([[2020]])
print(f"Predicted per capita income for 2020: ${income_2020[0]:.2f}")

# Plotting the regression line
plt.scatter(df['year'], y, color='blue', label='Data')
plt.plot(df['year'], model.predict(X), color='red', label='Regression Line')
plt.scatter(2020, income_2020, color='green', s=100, label='Prediction (2020)')
plt.xlabel('Year')
plt.ylabel('Per Capita Income (US$)')
plt.title('Canada Per Capita Income Prediction')
plt.legend()
plt.grid(True)
plt.show()
```

Output:


Canada Per Capita Income Prediction
Data
Regression Line
Prediction (2020)
Per Capita Income (US$)
0
10000
20000
30000
40000
1970
1980
1990
2000
2010
2020
Year

Lab No: 09

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?

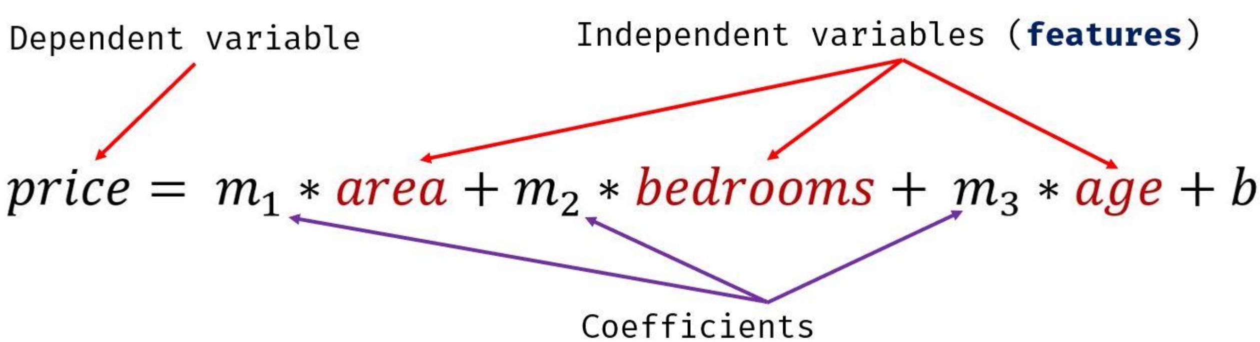
Multivariate Linear Regression extends simple linear regression by modeling the relationship between a dependent variable (Y) and multiple independent variables (X_1, X_2, ..., X_n).

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

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.impute import SimpleImputer

# Step 2: Load the dataset
df = pd.read_csv("hiring.csv")

# Step 3: Preprocessing - fill missing values and convert text
# Convert 'two' to 2, 'five' to 5
df['experience'] = df['experience'].fillna('zero')
df['experience'] = df['experience'].replace({
    'zero': 0, 'one': 1, 'two': 2, 'three': 3, 'four': 4,
    'five': 5, 'six': 6, 'seven': 7, 'eight': 8, 'nine': 9,
    'ten': 10, 'eleven': 11, 'twelve': 12
})

# Fill missing test or interview scores with mean
imputer = SimpleImputer()
df[['test_score(out of 10)', 'interview_score(out of 10)']] = imputer.fit_transform(
    df[['test_score(out of 10)', 'interview_score(out of 10)']]
)

# Step 4: Train the model
X = df[['experience', 'test_score(out of 10)', 'interview_score(out of 10)']]
y = df['salary($)']

model = LinearRegression()
model.fit(X, y)

# Step 5: Make predictions
pred_1 = model.predict([[2, 9, 6]])
pred_2 = model.predict([[12, 10, 10]])

print(f"Predicted salary for candidate 1: ${pred_1[0]:.2f}")
print(f"Predicted salary for candidate 2: ${pred_2[0]:.2f}")
```

__Output:__

```
Predicted salary for candidate 1: $53290.89
Predicted salary for candidate 2: $92268.07
```

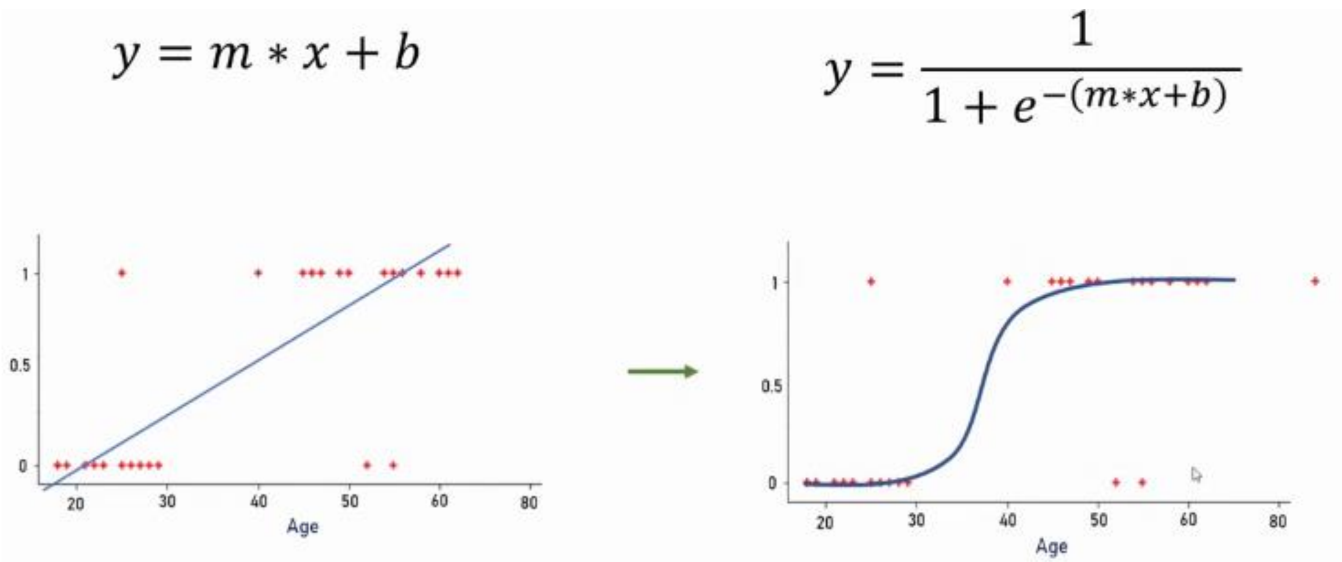
Lab No: 10

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

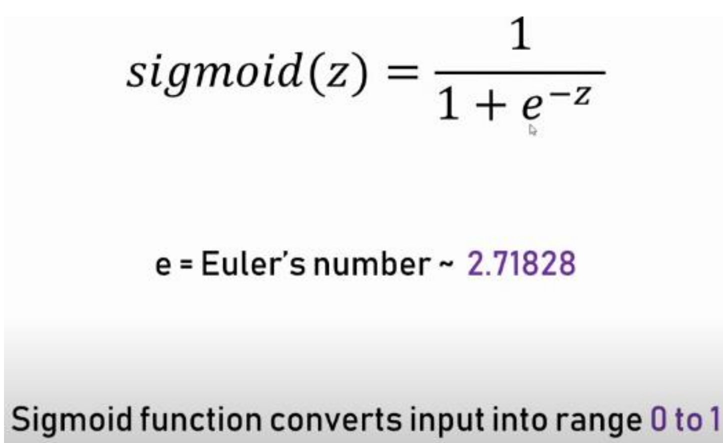
What is Logistic Regression?

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.

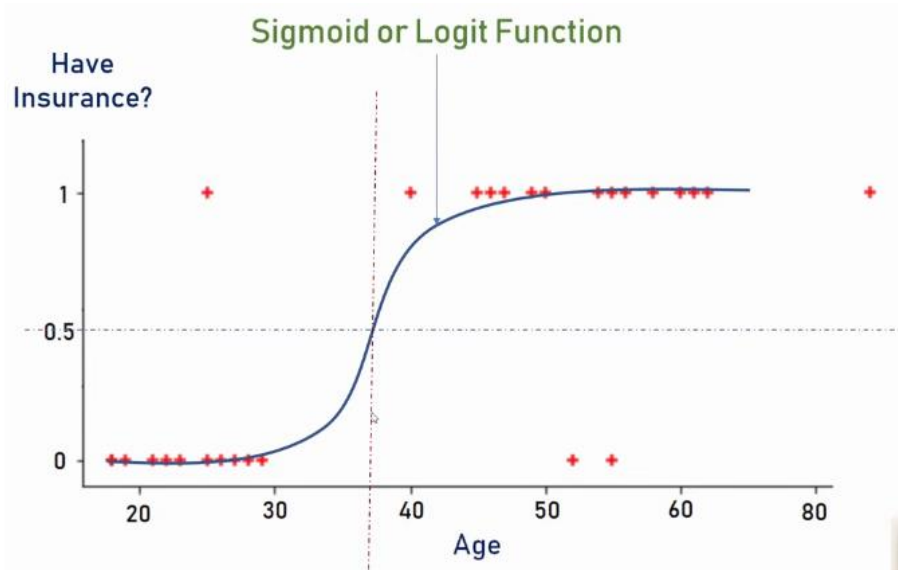


Sigmoid Function: -



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)

```python
import pandas as pd

# Load the dataset
df = pd.read_csv("HR_comma_sep.csv")

# Rename 'left' column to 'retained' and flip values (1 = stayed, 0 = left)
df = df.rename(columns={'left': 'retained'})
df['retained'] = 1 - df['retained']

# Display first few rows
print(df.head())
```

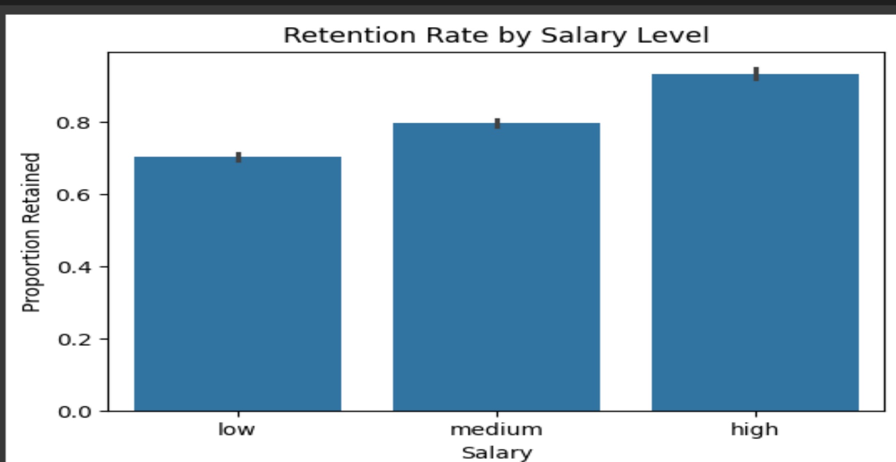
```
   satisfaction_level  last_evaluation  number_project  average_montly_hours  \
0                0.38             0.53               2                   157
1                0.80             0.86               5                   262
2                0.11             0.88               7                   272
3                0.72             0.87               5                   223
4                0.37             0.52               2                   159

   time_spend_company  Work_accident  retained  promotion_last_5years  \
0                   3              0         0                      0
1                   6              0         0                      0
2                   4              0         0                      0
3                   5              0         0                      0
4                   3              0         0                      0

  Department  salary
0      sales     low
1      sales  medium
2      sales  medium
3      sales     low
4      sales     low
```

```python
# Drop categorical columns before correlation
numeric_df = df.drop(['Department', 'salary'], axis=1)

# Show correlation with 'retained'
print("Correlation with 'retained':")
print(numeric_df.corr()['retained'].sort_values(ascending=False))
```

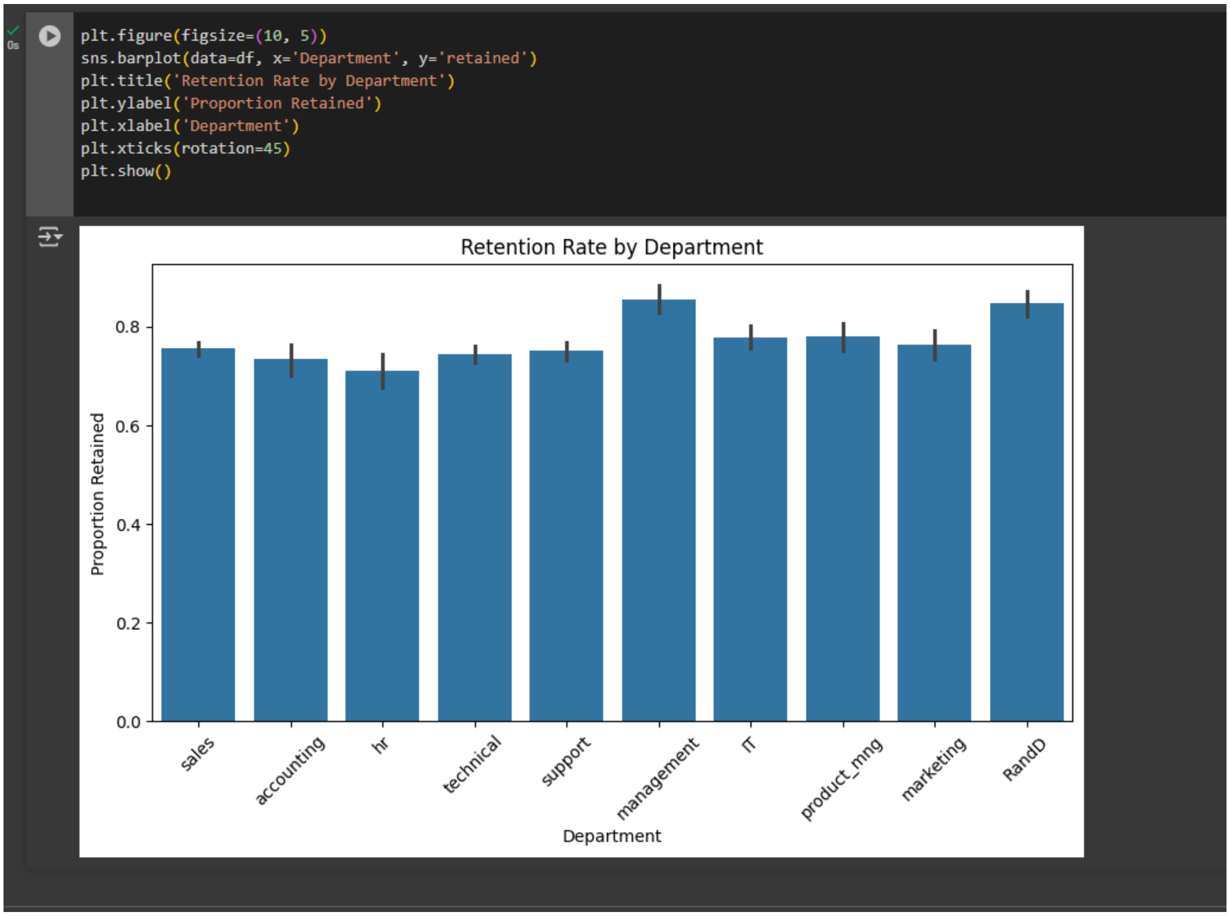
```
Correlation with 'retained':
retained                 1.000000
satisfaction_level       0.388375
Work_accident            0.154622
promotion_last_5years    0.061788
last_evaluation         -0.006567
number_project          -0.023787
average_montly_hours    -0.071287
time_spend_company      -0.144822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

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

```python
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(6, 4))
sns.barplot(data=df, x='salary', y='retained')
plt.title('Retention Rate by Salary Level')
plt.ylabel('Proportion Retained')
plt.xlabel('Salary')
plt.show()
```



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

```
[12] from sklearn.preprocessing import LabelEncoder

     le_salary = LabelEncoder()
     df['salary'] = le_salary.fit_transform(df['salary'])  # low=1, medium=2, high=0

     le_dept = LabelEncoder()
     df['Department'] = le_dept.fit_transform(df['Department'])
```

```
[13] # Select features
     features = ['satisfaction_level', 'last_evaluation', 'number_project',
                 'average_montly_hours', 'time_spend_company',
                 'Work_accident', 'promotion_last_5years',
                 'salary', 'Department']

     X = df[features]
     y = df['retained']
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

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

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

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

5.Measure the accuracy of the model

```
[15] from sklearn.metrics import accuracy_score

     # Predict and check accuracy
     y_pred = model.predict(X_test)
     accuracy = accuracy_score(y_test, y_pred)
     print(f" Logistic Regression Accuracy: {accuracy:.2%}")
```

```
 Logistic Regression Accuracy: 75.83%
```

```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```