# 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. **Muhammad Rafay Anwar** with Roll # **23f- Ai-17** 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.

```python
def find_needle_in_haystack(haystack, needle):
    """
    This function returns the index of the first occurrence of `needle` in `haystack`,
    or -1 if `needle` is not found.
    """
    return haystack.find(needle)


haystack1 = "sadbutsad"
needle1 = "sad"
result1 = find_needle_in_haystack(haystack1, needle1)
print(f"Example 1 - Index of '{needle1}' in '{haystack1}' is: {result1}")


haystack2 = "leetcode"
needle2 = "leeto"
result2 = find_needle_in_haystack(haystack2, needle2)
print(f"Example 2 - Index of '{needle2}' in '{haystack2}' is: {result2}")
```

```
Example 1 - Index of 'sad' in 'sadbutsad' is: 0
Example 2 - Index of 'leeto' in 'leetcode' is: -1
```

# 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.

```python
import numpy as np

arr1 = np.array([10, 20, 30, 40, 50])
arr2 = np.array([30, 40, 60, 70])
common_items = np.intersect1d(arr1, arr2)
print("Task 1 - Common items:", common_items)

arr3 = np.array([1, 2, 3, 4, 5])
arr4 = np.array([1, 2, 0, 4, 0])
matching_positions = np.where(arr3 == arr4)
print("Task 2 - Matching positions:", matching_positions[0])

arr5 = np.array([5, 10, 15, 20, 25, 30])
range_filtered = arr5[(arr5 >= 10) & (arr5 <= 25)]
print("Task 3 - Numbers between 10 and 25:", range_filtered)

arr6 = np.array([10, 20, 30, 40, 50, 60])
window_size = 3
moving_avg = np.convolve(arr6, np.ones(window_size)/window_size, mode='valid')
print("Task 4 - Moving average:", moving_avg)
```

```
Task 1 - Common items: [30 40]
Task 2 - Matching positions: [0 1 3]
Task 3 - Numbers between 10 and 25: [10 15 20 25]
Task 4 - 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**.

```
[23] import kagglehub
     import pandas as pd
     import os

     path = kagglehub.dataset_download("inductiveanks/top-1000-imdb-movies-dataset")
     print("Path to dataset files:", path)

     print("Files in dataset folder:", os.listdir(path))
     df = pd.read_csv(os.path.join(path, "Top_1000_IMDb_movies_New_version.csv"))
```

```
Path to dataset files: /kaggle/input/top-1000-imdb-movies-dataset
Files in dataset folder: ['Top_1000_IMDb_movies_New_version.csv']
```

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

print(df.head())
print(df.dtypes)
print("Rows and columns:", df.shape)
print("Missing values:\n", df.isnull().sum())
```

```
   Unnamed: 0                Movie Name Year of Release  Watch Time  \
0           0  The Shawshank Redemption            1994         142
1           1             The Godfather            1972         175
2           2           The Dark Knight            2008         152
3           3          Schindler's List            1993         195
4           4              12 Angry Men            1957          96

   Movie Rating  Metascore of movie   Gross      Votes  \
0           9.3                82.0   28.34  27,77,378
1           9.2               100.0  134.97  19,33,588
2           9.0                84.0  534.86  27,54,087
3           9.0                95.0    96.9  13,97,886
4           9.0                97.0    4.36   8,24,211

                                         Description
0  Over the course of several years, two convicts...
1  Don Vito Corleone, head of a mafia family, dec...
2  When the menace known as the Joker wreaks havo...
3  In German-occupied Poland during World War II,...
4  The jury in a New York City murder trial is fr...
Unnamed: 0              int64
Movie Name             object
Year of Release        object
Watch Time              int64
Movie Rating          float64
Metascore of movie    float64
Gross                  object
Votes                  object
Description            object
dtype: object
Rows and columns: (1000, 9)
Missing values:
 Unnamed: 0              0
Movie Name             0
Year of Release        0
Watch Time             0
Movie Rating           0
Metascore of movie   155
Gross                162
Votes                  0
Description            0
dtype: int64
```

```python
# 1. Remove duplicate rows
df.drop_duplicates(inplace=True)

# 2. Fill missing values in Metascore
df['Metascore of movie'] = df['Metascore of movie'].fillna(df['Metascore of movie'].mean())

# 3. Fix Gross column
df['Gross'] = df['Gross'].astype(str).str.replace(',', '', regex=False)
df['Gross'] = pd.to_numeric(df['Gross'], errors='coerce')
df['Gross'] = df['Gross'].fillna(df['Gross'].mean())
# 4. Convert Year of Release to integer
df['Year of Release'] = pd.to_numeric(df['Year of Release'], errors='coerce')
print(df.head())
print(df.dtypes)
```

```
   Unnamed: 0                Movie Name  Year of Release  Watch Time  \
0           0  The Shawshank Redemption           1994.0         142
1           1             The Godfather           1972.0         175
2           2           The Dark Knight           2008.0         152
3           3          Schindler's List           1993.0         195
4           4              12 Angry Men           1957.0          96

   Movie Rating  Metascore of movie   Gross      Votes  \
0           9.3                82.0   28.34  27,77,378
1           9.2               100.0  134.97  19,33,588
2           9.0                84.0  534.86  27,54,087
3           9.0                95.0   96.90  13,97,886
4           9.0                97.0    4.36   8,24,211

                                         Description
0  Over the course of several years, two convicts...
1  Don Vito Corleone, head of a mafia family, dec...
2  When the menace known as the Joker wreaks havo...
3  In German-occupied Poland during World War II,...
4  The jury in a New York City murder trial is fr...
Unnamed: 0              int64
Movie Name             object
Year of Release       float64
Watch Time              int64
Movie Rating          float64
Metascore of movie    float64
Gross                 float64
Votes                  object
Description            object
dtype: object
```

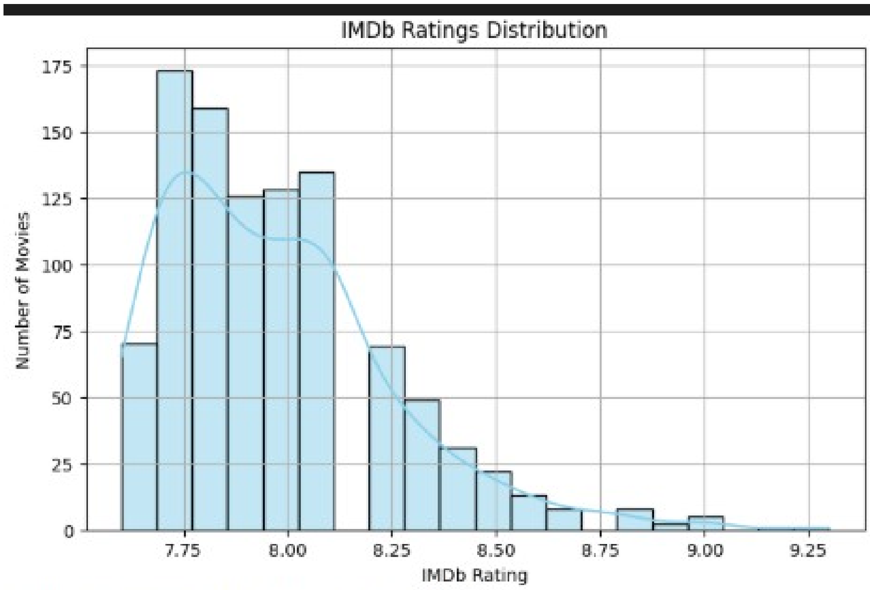
```
[33] # 1. IMDb rating > 8.5
     print(df[df['Movie Rating'] > 8.5][['Movie Name', 'Movie Rating']])

     # 2. Action or Sci-Fi genre (assume there's a Genre column)
     if 'Genre' in df.columns:
         genre_filter = df[df['Genre'].str.contains('Action|Sci-Fi', case=False, na=False)]
         print(genre_filter[['Movie Name', 'Genre']])
     else:
         print("Genre column not found.")

     # 3. Movies released between 2000 and 2015
     filtered_years = df[(df['Year of Release'] >= 2000) & (df['Year of Release'] <= 2015)]
     print(filtered_years[['Movie Name', 'Year of Release']])

     # 4. Sort by IMDb rating (descending)
     sorted_by_rating = df.sort_values(by='Movie Rating', ascending=False)
     print(sorted_by_rating[['Movie Name', 'Movie Rating']].head())
```

```
                                            Movie Name  Movie Rating
0                             The Shawshank Redemption           9.3
1                                        The Godfather           9.2
2                                      The Dark Knight           9.0
3                                     Schindler's List           9.0
4                                         12 Angry Men           9.0
5        The Lord of the Rings: The Return of the King           9.0
6                                The Godfather Part II           9.0
7                      Spider-Man: Across the Spider-Verse       8.9
8                                         Pulp Fiction           8.9
9                                            Inception           8.8
10                                          Fight Club           8.8
11   The Lord of the Rings: The Fellowship of the Ring           8.8
12                                        Forrest Gump           8.8
13                       Il buono, il brutto, il cattivo         8.8
14               The Lord of the Rings: The Two Towers           8.8
15                                            Jai Bhim           8.8
16                                         777 Charlie           8.8
17                                         Oppenheimer           8.7
18                                        Interstellar           8.7
19                                          GoodFellas           8.7
20                     One Flew Over the Cuckoo's Nest           8.7
21                                          The Matrix           8.7
22      Star Wars: Episode V - The Empire Strikes Back           8.7
23                          Rocketry: The Nambi Effect           8.7
24                                     Soorarai Pottru           8.7
25                                               Se7en           8.6
26                                 Saving Private Ryan           8.6
27                            The Silence of the Lambs           8.6
28                          Terminator 2: Judgment Day           8.6
29                                      The Green Mile           8.6
30                                           Star Wars           8.6
31                                      Cidade de Deus           8.6
32                     Sen to Chihiro no kamikakushi             8.6
33                                      La vita è bella          8.6
34                                  Shichinin no samurai         8.6
35                                It's a Wonderful Life          8.6
36                                             Seppuku           8.6
37                                          Sita Ramam           8.6
Genre column not found.
                                            Movie Name  Year of Release
2                                      The Dark Knight           2008.0
5        The Lord of the Rings: The Return of the King           2003.0
9                                            Inception           2010.0
11   The Lord of the Rings: The Fellowship of the Ring           2001.0
14               The Lord of the Rings: The Two Towers           2002.0
..                                                 ...              ...
992                                          21 Grams            2003.0
994                                           Control            2007.0
995                                         Philomena            2013.0
996                    Un long dimanche de fiançailles           2004.0
999                                         Celda 211            2009.0

[340 rows x 2 columns]
                 Movie Name  Movie Rating
0  The Shawshank Redemption           9.3
1             The Godfather           9.2
```

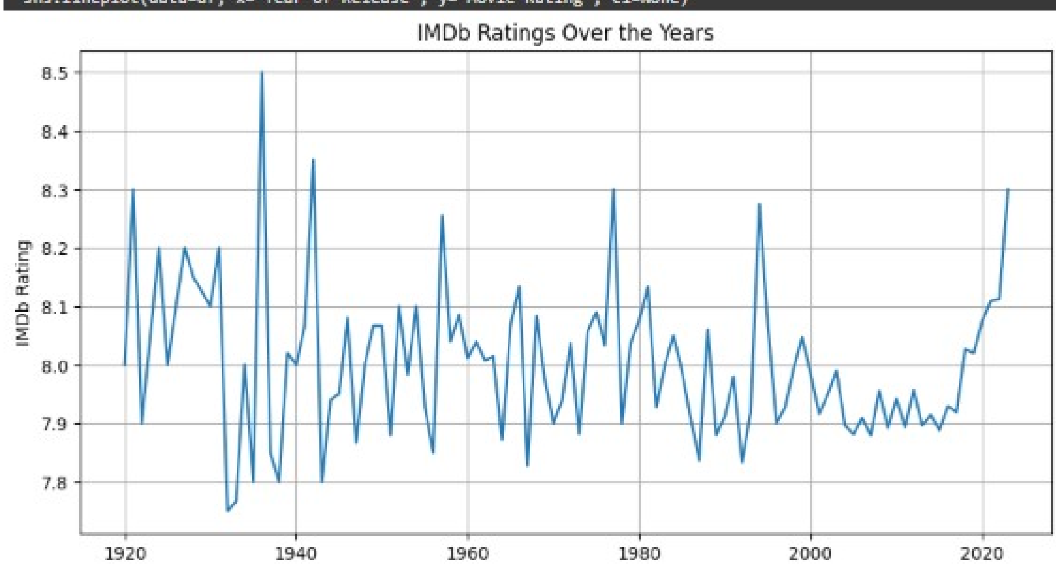
```
# 1. Average IMDb rating per genre
if 'Genre' in df.columns:
    print(df.groupby('Genre')['Movie Rating'].mean())

# 2. Year with most movie releases
print("Most movies released in:", df['Year of Release'].value_counts().idxmax())

# 3. Top 5 directors with most movies
if 'Director' in df.columns:
    print(df['Director'].value_counts().head(5))
else:
    print("Director column not found.")
```

```
Most movies released in: 2014.0
Director column not found.
```

[37]
```
# 1. Histogram of IMDb Ratings
plt.figure(figsize=(8, 5))
sns.histplot(df['Movie Rating'], bins=20, kde=True, color='skyblue')
plt.title("IMDb Ratings Distribution")
plt.xlabel("IMDb Rating")
plt.ylabel("Number of Movies")
plt.grid(True)
plt.show()

# 2. Bar chart of top 10 genres
if 'Genre' in df.columns:
    top_genres = df['Genre'].value_counts().head(10)
    plt.figure(figsize=(10, 6))
    sns.barplot(x=top_genres.values, y=top_genres.index, palette='Set2')
    plt.title("Top 10 Genres with Most Movies")
    plt.xlabel("Number of Movies")
    plt.ylabel("Genre")
    plt.show()

# 3. Trend of IMDb Ratings over the years
plt.figure(figsize=(10, 5))
sns.lineplot(data=df, x='Year of Release', y='Movie Rating', ci=None)
plt.title("IMDb Ratings Over the Years")
plt.xlabel("Year")
plt.ylabel("IMDb Rating")
plt.grid(True)
plt.show()
```




```
/tmp/ipython-input-37-811179754.py:22: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.lineplot(data=df, x='Year of Release', y='Movie Rating', ci=None)
```



# 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.

**<u>Tasks: -</u>**

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.

```python
from collections import deque

def bfs_without_goal(graph, start):
    visited = set()
    queue = deque([start])
    visited.add(start)

    print("BFS Traversal Order:")
    while queue:
        vertex = queue.popleft()
        print(vertex, end=" → ")

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

# Sample graph as adjacency list
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}

# Run the BFS traversal
bfs_without_goal(graph, 'A')
```

```
BFS Traversal Order:
A → B → C → D → E → F →
```

```python
from collections import deque

def bfs_with_goal(graph, start, goal):
    visited = set()
    queue = deque([start])
    visited.add(start)

    print(f"Searching for goal node '{goal}'...")

    while queue:
        vertex = queue.popleft()
        print("Visiting:", vertex)

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

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

    print(f"Goal node '{goal}' not found in the graph.")

# Sample graph
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}

# Run BFS with goal
bfs_with_goal(graph, start='A', goal='F')
```

```
Searching for goal node 'F'...
Visiting: A
Visiting: B
Visiting: C
Visiting: D
Visiting: E
Visiting: F
Goal node 'F' found!
```

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

```
def dfs_without_goal(graph, start):
    visited = set()
    stack = [start]

    print("DFS Traversal Order:")
    while stack:
        vertex = stack.pop()
        if vertex not in visited:
            print(vertex, end=" → ")
            visited.add(vertex)
            # Add neighbors in reverse order to maintain left-to-right traversal
            stack.extend(reversed(graph[vertex]))

# Sample graph as adjacency list
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}

dfs_without_goal(graph, 'A')
```

```
DFS Traversal Order:
A → B → D → E → F → C →
```

```python
def dfs_with_goal(graph, start, goal):
    visited = set()
    stack = [start]

    print(f"Searching for goal node '{goal}' using DFS...")

    while stack:
        vertex = stack.pop()
        if vertex not in visited:
            print("Visiting:", vertex)
            if vertex == goal:
                print(f"Goal node '{goal}' found!")
                return
            visited.add(vertex)
            stack.extend(reversed(graph[vertex]))

    print(f"Goal node '{goal}' not found in the graph.")

# Sample graph
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}

dfs_with_goal(graph, start='A', goal='F')
```

```
Searching for goal node 'F' using DFS...
Visiting: A
Visiting: B
Visiting: D
Visiting: E
Visiting: F
Goal node 'F' found!
```

# Lab No: 6

**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.

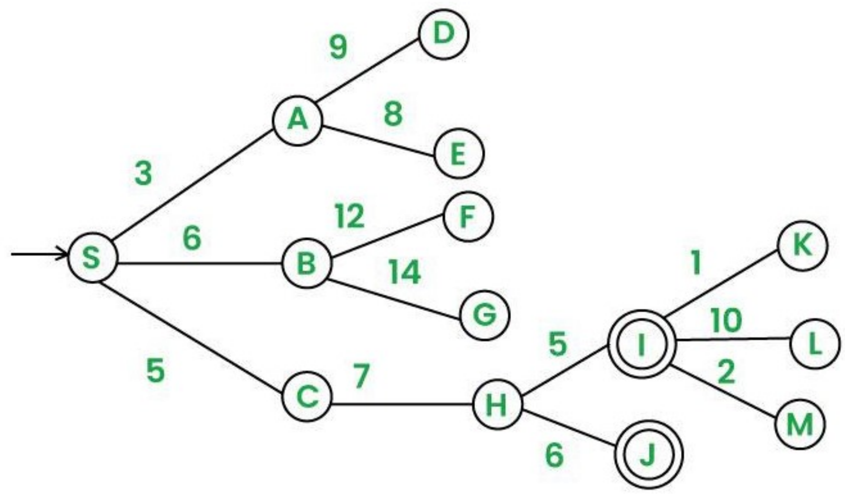
**BFS Algorithm:**
If we are given an edge list of a graph where every edge is represented as (u, v, w). Here u, v and w represent source, destination and weight of the edges respectively. We need to do Best First Search of the graph (Pick the minimum cost edge next).

- 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

graph = {
    'S': [('A', 3), ('B', 6), ('C', 5)],
    'A': [('D', 9), ('E', 8)],
    'B': [('E', 12), ('F', 14), ('G', 4)],
    'C': [('H', 7)],
    'H': [('I', 5), ('J', 6)],
    'I': [('K', 1), ('L', 10), ('M', 2)],
    'D': [], 'E': [], 'F': [], 'G': [], 'J': [], 'K': [], 'L': [], 'M': []
}

def best_first_search(start, goal):
    visited = set()
    pq = []
    heapq.heappush(pq, (0, start))

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

        if current == goal:
            print("Goal reached!")
            return

        if current in visited:
            continue
        visited.add(current)

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

    print("Goal not reachable.")

best_first_search("S", "I")
```

```
Visiting: S
Visiting: A
Visiting: C
Visiting: B
Visiting: G
Visiting: H
Visiting: I
Goal reached!
```

# 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

# Graph: node -> [(neighbor, cost)]
graph = {
    'S': [('A', 3), ('B', 6), ('C', 5)],
    'A': [('D', 9), ('E', 8)],
    'B': [('E', 12), ('F', 14), ('G', 4)],
    'C': [('H', 7)],
    'H': [('I', 5), ('J', 6)],
    'I': [('K', 1), ('L', 10), ('M', 2)],
    'D': [], 'E': [], 'F': [], 'G': [],
    'J': [], 'K': [], 'L': [], 'M': []
}

# Heuristic values (h) - Estimated cost from node to goal 'I'
heuristic = {
    'S': 10, 'A': 9, 'B': 7, 'C': 8,
    'D': 9, 'E': 4, 'F': 6, 'G': 5,
    'H': 3, 'I': 0, 'J': 1, 'K': 2,
    'L': 3, 'M': 4
}

def a_star(start, goal):
    open_list = []
    heapq.heappush(open_list, (0 + heuristic[start], 0, start, [start]))  # (f(n), g(n), node, path)
    visited = {}

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

        if current == goal:
            print("Goal reached!")
            print("Path:", " -> ".join(path))
            print("Total cost:", g)
            return

        if current in visited and visited[current] <= g:
            continue
        visited[current] = g

        for neighbor, cost in graph.get(current, []):
            g_new = g + cost
            f_new = g_new + heuristic.get(neighbor, 0)
            heapq.heappush(open_list, (f_new, g_new, neighbor, path + [neighbor]))

    print("Goal not reachable.")

# Run A* Search from 'S' to 'I'
a_star('S', 'I')
```

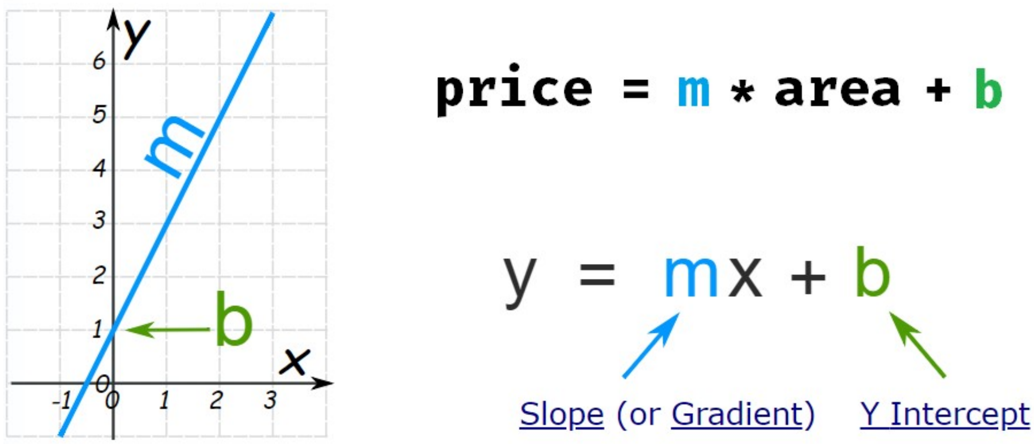
```
Goal reached!
Path: S -> C -> H -> I
Total cost: 17
```

# 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:**

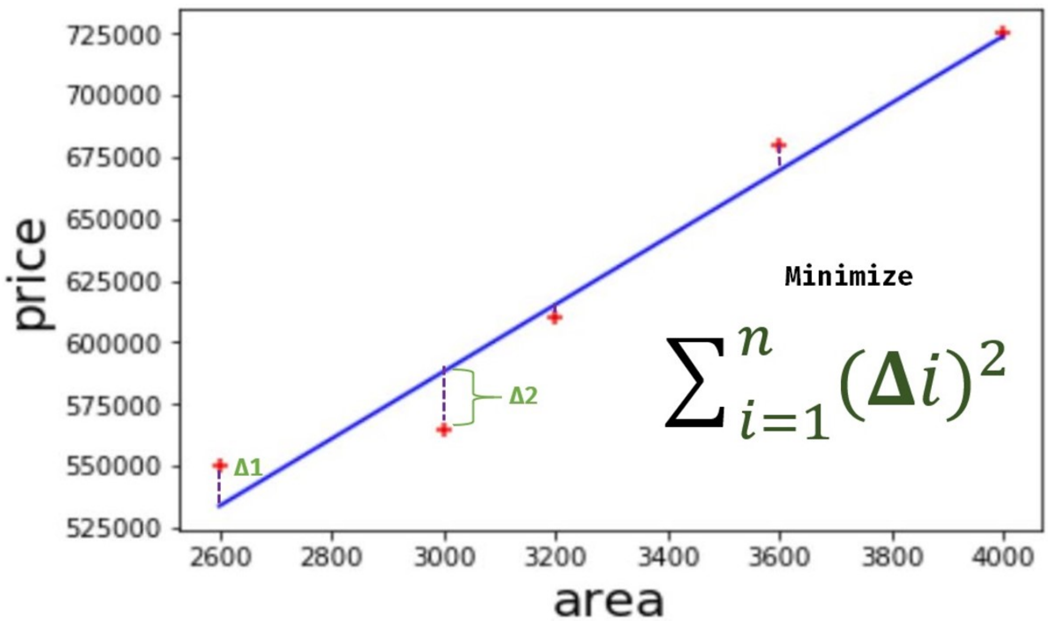
Y=mX+b

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: -**

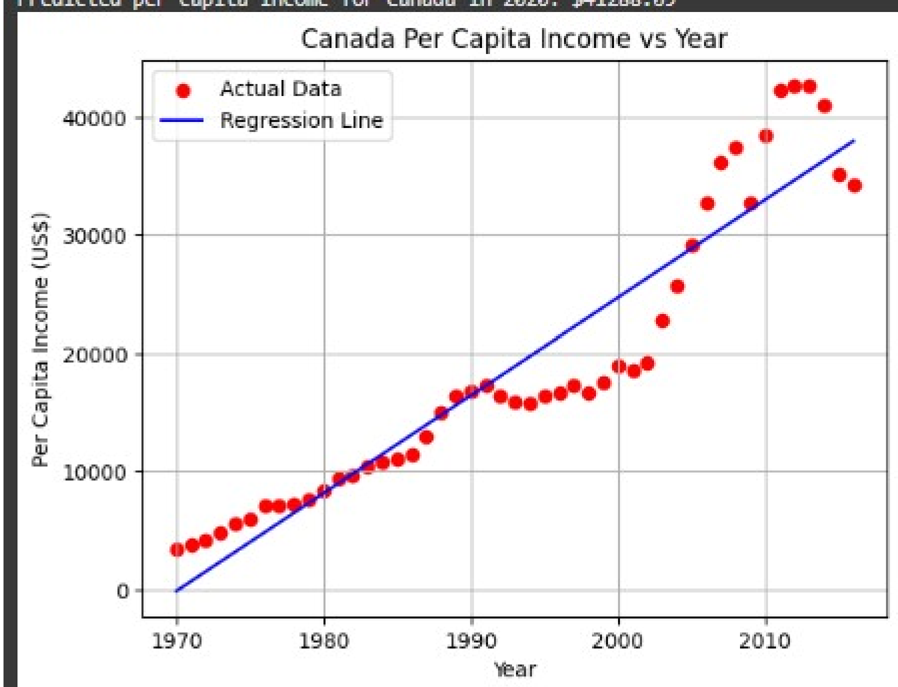
- Independent variable (X) – The input or feature.
- Dependent variable (Y) – The output or label.
- Loss Function – Measures prediction error (commonly MSE).

**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
from sklearn.linear_model import LinearRegression

# Step 1: Load dataset
df = pd.read_csv("/content/canada_per_capita_income_exercise.csv")

# Step 2: Prepare data
X = df[['year']]  # Feature
y = df['per capita income (US$)']  # Label

# Step 3: Train the model
model = LinearRegression()
model.fit(X, y)

# Step 4: Predict income for 2020
prediction = model.predict([[2020]])
print(f"Predicted per capita income for Canada in 2020: ${prediction[0]:.2f}")

# Step 5: Visualize the regression line
plt.scatter(X, y, color='red', label='Actual Data')
plt.plot(X, model.predict(X), color='blue', label='Regression Line')
plt.xlabel("Year")
plt.ylabel("Per Capita Income (US$)")
plt.title("Canada Per Capita Income vs Year")
plt.legend()
plt.grid(True)
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserW
  warnings.warn(
Predicted per capita income for Canada in 2020: $41288.69
```



# 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?**

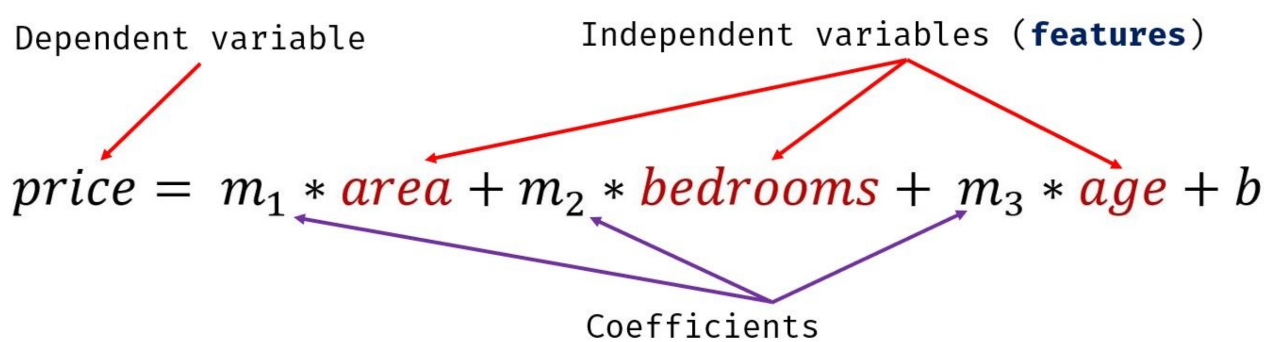
Multivariate Linear Regression extends simple linear regression by modeling the relationship between a dependent variable (Y) and multiple independent variables ($X_1, X_2, ..., X_n$).

**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

```python
df = pd.read_csv("/content/hiring.csv")
df.columns = df.columns.str.strip().str.lower().str.replace(' ', '_').str.replace(r'\(.*\)', '', regex=True)

def convert_experience(x):
    try:
        return float(x)
    except:
        word_to_num = {
            '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
        }
        return word_to_num.get(str(x).lower(), np.nan)  # <-- return NaN if unknown

df['experience'] = df['experience'].apply(convert_experience)


df['experience'] = df['experience'].fillna(df['experience'].mean())
df['test_score'] = df['test_score'].fillna(df['test_score'].mean())
df['interview_score'] = df['interview_score'].fillna(df['interview_score'].mean())

print("Missing values:\n", df.isnull().sum())

X = df[['experience', 'test_score', 'interview_score']]
y = df['salary']
model = LinearRegression()
model.fit(X, y)

candidates = np.array([[2, 9, 6], [12, 10, 10]])
predicted_salaries = model.predict(candidates)

for i, salary in enumerate(predicted_salaries, 1):
    print(f"Predicted salary for candidate {i}: ${salary:.2f}")
```

```
Missing values:
 experience        0
test_score         0
interview_score    0
salary             0
dtype: int64
Predicted salary for candidate 1: $47738.89
Predicted salary for candidate 2: $86424.67
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names
  warnings.warn(
```

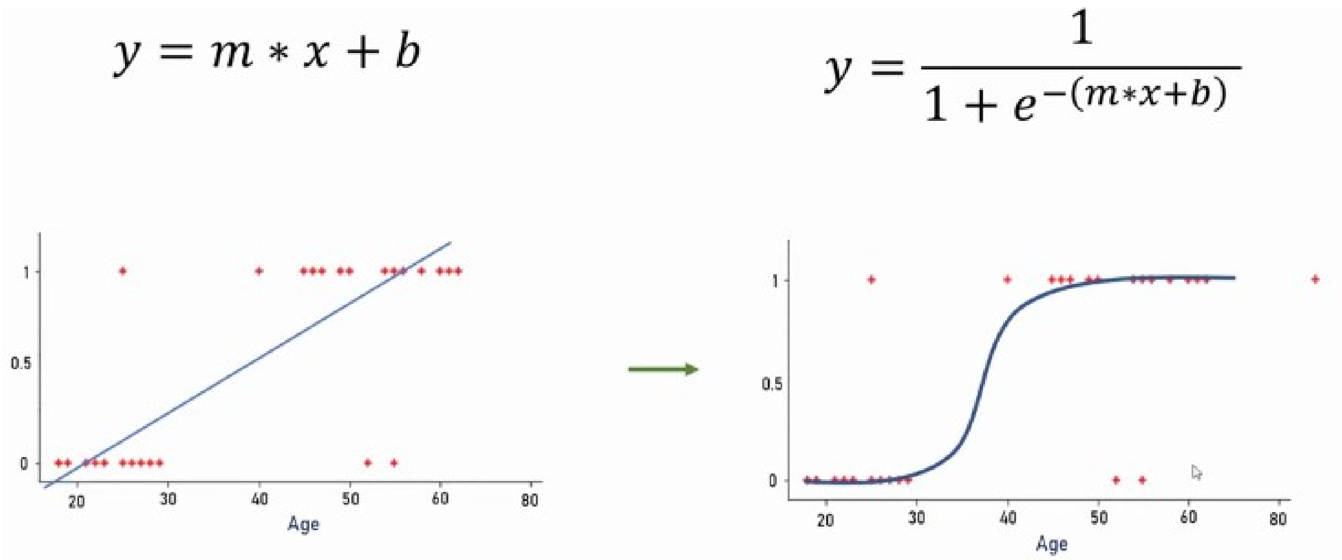
# Lab No: 10

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

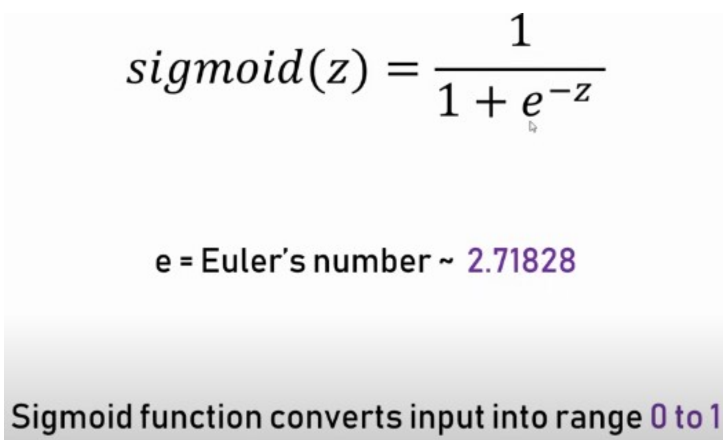
**What is Logistic Regression?**

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.



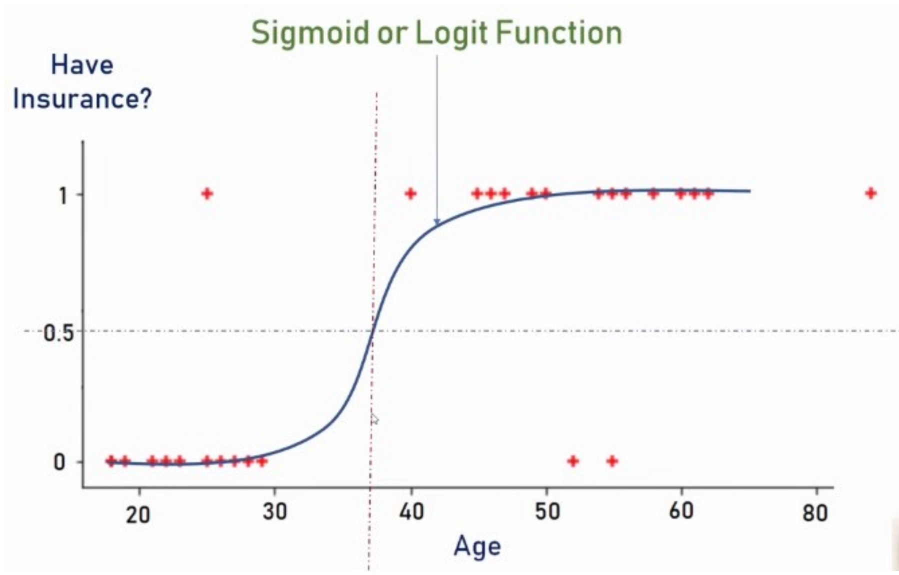

**Sigmoid Function: -**




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.

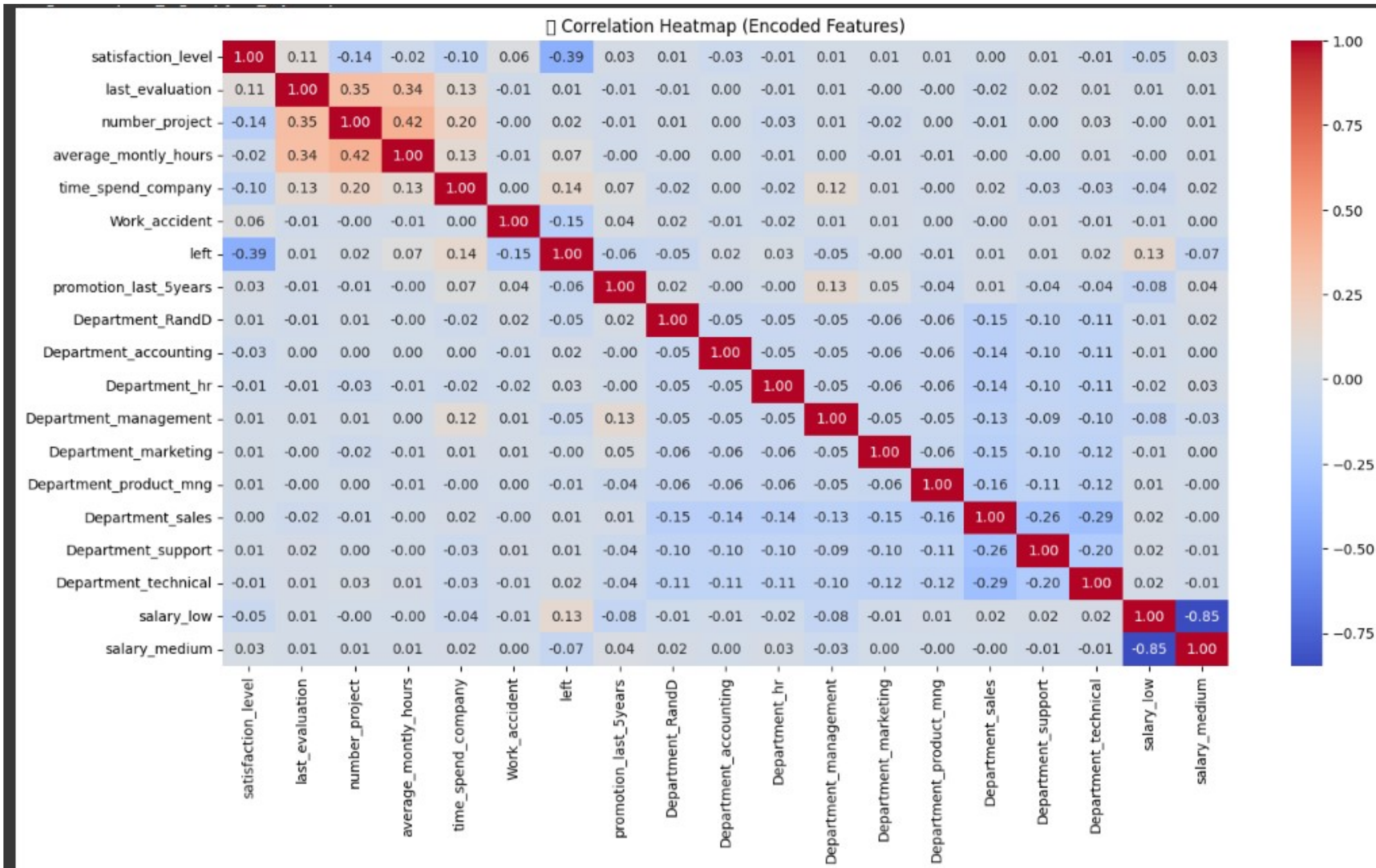
**Tasks:**

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)
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

```python
df = pd.read_csv("/content/HR_comma_sep_exercise.csv")


print("\n🔎 First 5 rows of dataset:\n", df.head())
print("\nℹ️ Dataset Info:\n")
print(df.info())
print("\n📊 Target value counts:\n", df['left'].value_counts())


df_encoded = pd.get_dummies(df, columns=['Department', 'salary'], drop_first=True)

plt.figure(figsize=(16, 8))
sns.heatmap(df_encoded.corr(), annot=True, fmt=".2f", cmap='coolwarm')
plt.title("🔗 Correlation Heatmap (Encoded Features)")
plt.show()

sns.countplot(data=df, x='salary', hue='left')
plt.title("💰 Salary vs Employee Retention")
plt.xlabel("Salary Level")
plt.ylabel("Number of Employees")
plt.legend(["Stayed", "Left"])
plt.show()

plt.figure(figsize=(10, 6))
sns.countplot(data=df, x='Department', hue='left')
plt.title("🏢 Department vs Employee Retention")
plt.xlabel("Department")
plt.ylabel("Number of Employees")
plt.xticks(rotation=45)
plt.legend(["Stayed", "Left"])
plt.show()

X = df_encoded.drop('left', axis=1)
y = df_encoded['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)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"\n✅ Model Accuracy: {accuracy * 100:.2f}%\n")

print("📋 Classification Report:\n", classification_report(y_test, y_pred))
conf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=["Stayed", "Left"], yticklabels=["Stayed", "Left"])
plt.title("🔍 Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
🔎 First 5 rows of dataset:
   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  left  promotion_last_5years Department  \
0                   3              0     1                      0      sales
1                   6              0     1                      0      sales
2                   4              0     1                      0      sales
3                   5              0     1                      0      sales
4                   3              0     1                      0      sales

   salary
0     low
1  medium
2  medium
3     low
4     low

ℹ️ Dataset Info:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14999 entries, 0 to 14998
Data columns (total 10 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   satisfaction_level     14999 non-null  float64
 1   last_evaluation        14999 non-null  float64
 2   number_project         14999 non-null  int64
 3   average_montly_hours   14999 non-null  int64
 4   time_spend_company     14999 non-null  int64
 5   Work_accident          14999 non-null  int64
 6   left                   14999 non-null  int64
 7   promotion_last_5years  14999 non-null  int64
 8   Department             14999 non-null  object
 9   salary                 14999 non-null  object
dtypes: float64(2), int64(6), object(2)
memory usage: 1.1+ MB
None

📊 Target value counts:
 left
0    11428
1     3571
Name: count, dtype: int64
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128279 (\N{LINK SYMBOL}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
```

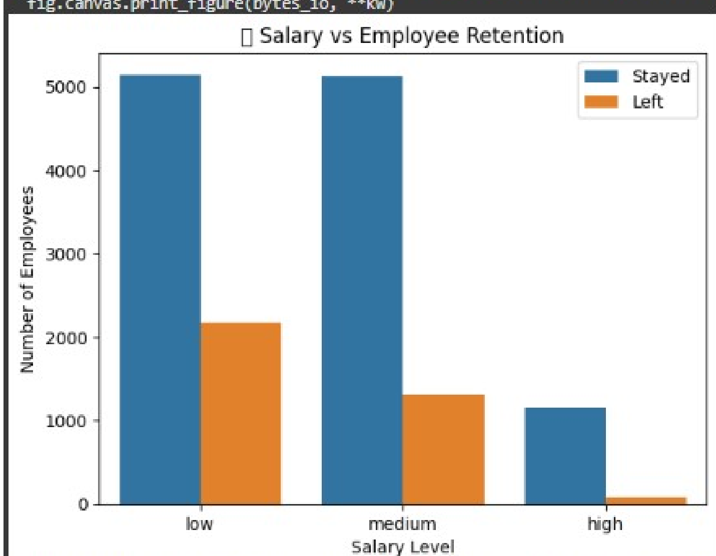



```
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128176 (\N{MONEY BAG}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
```

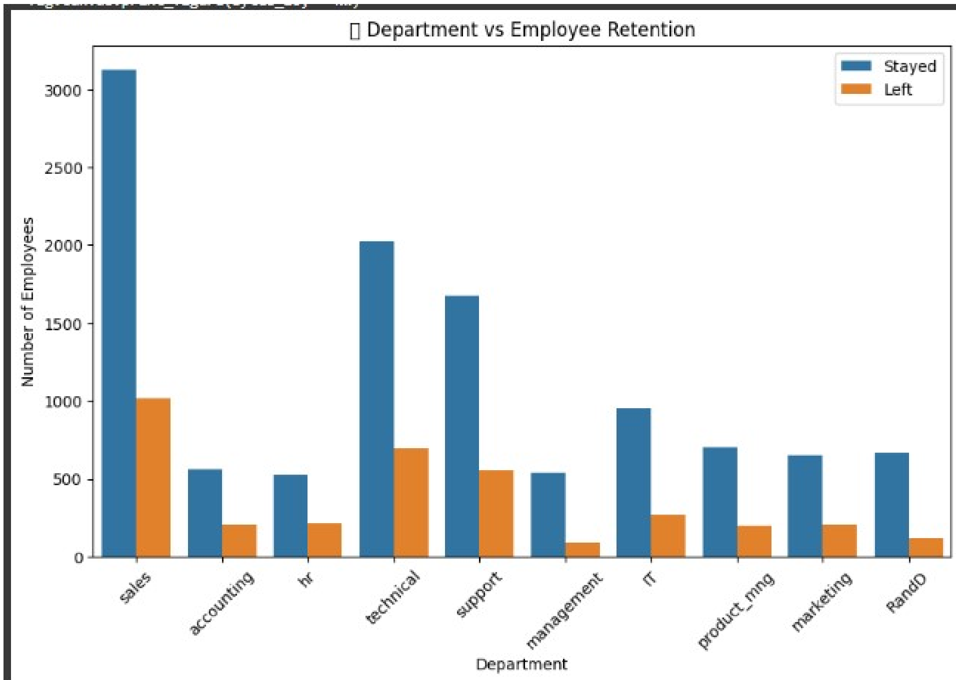


```
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128176 (\N{MONEY BAG}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
```






```
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 127970 (\N{OFFICE BUILDING}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
```




```
✅ Model Accuracy: 78.23%

📋 Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.92      0.87      2294
           1       0.57      0.33      0.41       706

    accuracy                           0.78      3000
   macro avg       0.69      0.62      0.64      3000
weighted avg       0.76      0.78      0.76      3000

/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128269 (\N{LEFT-POINTING MAGNIFYING GLASS}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
```

```
✅ Model Accuracy: 78.23%

📋 Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.92      0.87      2294
           1       0.57      0.33      0.41       706

    accuracy                           0.78      3000
   macro avg       0.69      0.62      0.64      3000
weighted avg       0.76      0.78      0.76      3000

/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128269 (\N{LEFT-POINTING MAGNIFYING GLASS}) missing from font(s) DejaVu Sans
  fig.canvas.print_figure(bytes_io, **kw)
```