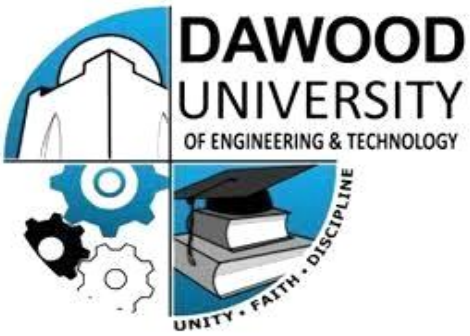
Artificial Intelligence

(Practical Manual)



4th Semester, 2nd Year BATCH -2023

BS ARTIFICIAL INTELLIGENCE

DAWOOD UNIVERSITY OF ENGINEERING & TECHNOLOGY, KARACHI

Dawood University Of Engineering and Technology, Karachi

CERTIFICATE

This is to certify that Mr./Ms. Muhammad Bilal with Roll # 23F-AI-50of 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.

ANS:

Source Code:

```
class Solution:
    def strStr(self, haystack: str, needle: str) -> int:
        return haystack.find(needle)
```

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:

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

ANS:

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

Source Code:

```
import numpy as np

a = np.array([2, 4, 7, 1, 4])
b = np.array([7, 2, 9, 8, 5])

print("Arrays A =", a)
print("Arrays B =", b)

c = np.intersect1d(a, b)
print("Common values = ", c)
```

Output:

```
Arrays A = [2 4 7 1 4]
Arrays B = [7 2 9 8 5]
Common values =  [2 7]
```

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

Source Code:

```
import numpy as np

a = np.array([2, 4, 7, 1, 4])
b = np.array([2, 4, 5, 7, 2])

print("Arrays A =", a)
print("Arrays B =", b)

c = np.where(a==b)
print("Position of match array = ", c )
```

Output:

```
Arrays A = [2 4 7 1 4]
Arrays B = [2 4 5 7 2]
Position of match array =  (array([0, 1]),)
```

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

Source Code:

```
import numpy as np

arr = np.arange(50)
print("Enter Range between 0 to 50")

start = int(input("Enter start value: "))
end = int(input("Enter end value: "))

print(arr[start:end])
```

Output:

```
Enter Range between 0 to 50
Enter start value: 5
Enter end value: 36
[ 5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
 29 30 31 32 33 34 35]
```

- Implement the moving average for the 1D array in NumPy.

Source Code:

```
import numpy as np

arr = np.arange(50)
moving_number = int(input("Enter number: "))

my_moving_list = (arr[:-1] + arr[1:]) / moving_number
my_moving_list = np.insert(my_moving_list, 0, 0)

print(my_moving_list)
```

Output:

```
Enter number: 2
[ 0.   0.5  1.5  2.5  3.5  4.5  5.5  6.5  7.5  8.5  9.5 10.5 11.5 12.5
 13.5 14.5 15.5 16.5 17.5 18.5 19.5 20.5 21.5 22.5 23.5 24.5 25.5 26.5
 27.5 28.5 29.5 30.5 31.5 32.5 33.5 34.5 35.5 36.5 37.5 38.5 39.5 40.5
 41.5 42.5 43.5 44.5 45.5 46.5 47.5 48.5]
```

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:

- **Series** – One-dimensional labeled array.
- **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

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

Task 2: Data Cleaning

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

Task 3: Data Filtering & Sorting

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

Data Aggregation & Grouping

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

Visualization (Optional)

Matplotlib or **Seaborn**

- Plot a histogram of IMDb ratings.
- Create a bar chart showing the **top 10 genres** with the most movies.

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

ANS:

Task 1: Load and Explore the Dataset

- Load the sataset.

Source Code:

```
data = pd.read_csv("imdb_top_1000.csv")
print(data)
```

Output:

```
                                    Poster_Link             Series_Title  ... No_of_Votes       Gross
0    https://m.media-amazon.com/images/M/MV5BMDFkYT...  The Shawshank Redemption  ...     2343110  28,341,469
1    https://m.media-amazon.com/images/M/MV5BM2MyNj...           The Godfather  ...     1620367  134,966,411
2    https://m.media-amazon.com/images/M/MV5BMTMxNT...         The Dark Knight  ...     2303232  534,858,444
3    https://m.media-amazon.com/images/M/MV5BMWMwMG...  The Godfather: Part II  ...     1129952  57,300,000
4    https://m.media-amazon.com/images/M/MV5BMWU4N2...            12 Angry Men  ...      689845   4,360,000
..                                                 ...                     ...  ...         ...         ...
995  https://m.media-amazon.com/images/M/MV5BNGEwMT...  Breakfast at Tiffany's  ...      166544         NaN
996  https://m.media-amazon.com/images/M/MV5BODk3Yj...                   Giant  ...       34075         NaN
997  https://m.media-amazon.com/images/M/MV5BM2U3Yz...  From Here to Eternity  ...       43374  30,500,000
998  https://m.media-amazon.com/images/M/MV5BZTBmMj...               Lifeboat  ...       26471         NaN
999  https://m.media-amazon.com/images/M/MV5BMTY5OD...           The 39 Steps  ...       51853         NaN

[1000 rows x 16 columns]
```

- Display the first 5 rows.

Source Code:

```
data = pd.read_csv("imdb_top_1000.csv")
print(data.head())
```

Output:

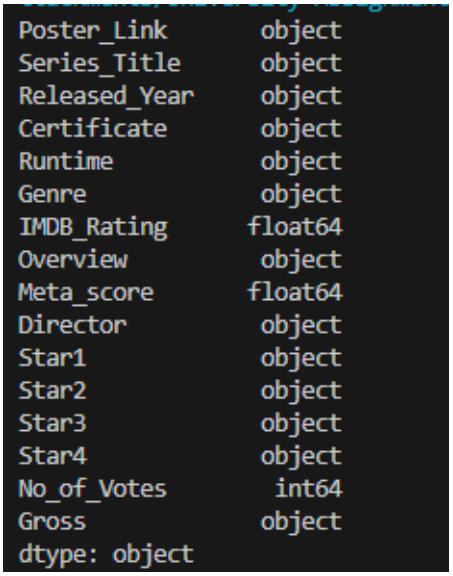
```
                                    Poster_Link             Series_Title  ... No_of_Votes       Gross
0  https://m.media-amazon.com/images/M/MV5BMDFkYT...  The Shawshank Redemption  ...     2343110  28,341,469
1  https://m.media-amazon.com/images/M/MV5BM2MyNj...           The Godfather  ...     1620367  134,966,411
2  https://m.media-amazon.com/images/M/MV5BMTMxNT...         The Dark Knight  ...     2303232  534,858,444
3  https://m.media-amazon.com/images/M/MV5BMWMwMG...  The Godfather: Part II  ...     1129952  57,300,000
4  https://m.media-amazon.com/images/M/MV5BMWU4N2...            12 Angry Men  ...      689845   4,360,000

[5 rows x 16 columns]
```

- Check the data types of each column.

Source Code:

```
data = pd.read_csv("imdb_top_1000.csv")
print(data.dtypes)
```

Output:



For Specific Column.

```
data = pd.read_csv("imdb_top_1000.csv")
print(data['Genre'].dtypes)
# or
print(data.Genre.dtypes)
```

Output:

- Find the number of rows and columns.

Source Code:

```
data = pd.read_csv("imdb_top_1000.csv")
print("The number of rows is: ", len(data))
print("The number of columns is: ", len(data.columns))
```

Output:

```
The number of rows is:  1000
The number of columns is:  16
```

- Check for missing values.

Source Code:

```
data = pd.read_csv("imdb_top_1000.csv")
print(data.isnull())
```

Output:

	Poster_Link	Series_Title	Released_Year	Certificate	Runtime	...	Star2	Star3	Star4	No_of_Votes	Gross
0	False	False	False	False	False	...	False	False	False	False	False
1	False	False	False	False	False	...	False	False	False	False	False
2	False	False	False	False	False	...	False	False	False	False	False
3	False	False	False	False	False	...	False	False	False	False	False
4	False	False	False	False	False	...	False	False	False	False	False
..	...	...	...	...	...	...	...	...	...	...	...
995	False	False	False	False	False	...	False	False	False	False	True
996	False	False	False	False	False	...	False	False	False	False	True
997	False	False	False	False	False	...	False	False	False	False	False
998	False	False	False	True	False	...	False	False	False	False	True
999	False	False	False	True	False	...	False	False	False	False	True

[1000 rows x 16 columns]

Task 2: Data Cleaning

- Remove any duplicate rows if present.

Source Code:

```
data = pd.read_csv("imdb_top_1000.csv")
print(data.drop_duplicates(inplace=True))
```

Output:
It returns None.

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

Source Code:

```
data = pd.read_csv("imdb_top_1000.csv")
print(data.fillna(data.mean(numeric_only=True), inplace=True)
```

Output:
It returns None.

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

Source Code:

```
data = pd.read_csv("imdb_top_1000.csv")

data['Runtime'] = data['Runtime'].str.replace(' min', '')
data['Runtime'] = data['Runtime'].astype(int)
print(data)
```

Task 3: Data Filtering & Sorting

- Find all movies with an IMDB rating greater than 8.5.

Source Code:

```
data = pd.read_csv("imdb_top_1000.csv")

data.drop_duplicates(inplace=True)
data.fillna(data.mean(numeric_only=True), inplace=True)
data['Runtime'] = data['Runtime'].str.replace(' min', '')
```

```
data['Runtime'] = data['Runtime'].astype(int)

greater = data[data['IMDB_Rating'] > 8.5]['Series_Title']
print(greater)
```

Output:

```
0                                The Shawshank Redemption
1                                           The Godfather
2                                         The Dark Knight
3                                  The Godfather: Part II
4                                            12 Angry Men
5                  The Lord of the Rings: The Return of the King
6                                            Pulp Fiction
7                                        Schindler's List
8                                               Inception
9                                              Fight Club
10    The Lord of the Rings: The Fellowship of the Ring
11                                           Forrest Gump
12                        Il buono, il brutto, il cattivo
13                    The Lord of the Rings: The Two Towers
14                                              The Matrix
15                                               Goodfellas
16             Star Wars: Episode V - The Empire Strikes Back
17                        One Flew Over the Cuckoo's Nest
18                                               Hamilton
19                                             Gisaengchung
20                                          Soorarai Pottru
21                                            Interstellar
22                                         Cidade de Deus
19                                             Gisaengchung
20                                          Soorarai Pottru
19                                             Gisaengchung
19                                             Gisaengchung
19                                             Gisaengchung
20                                          Soorarai Pottru
21                                            Interstellar
22                                         Cidade de Deus
23                          Sen to Chihiro no kamikakushi
24                                      Saving Private Ryan
25                                        The Green Mile
26                                     La vita è bella
27                                                 Se7en
28                              The Silence of the Lambs
29                                           Star Wars
30                                             Seppuku
31                                 Shichinin no samurai
32                                It's a Wonderful Life
Name: Series_Title, dtype: object
```

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

Source Code:

```
data = pd.read_csv("imdb_top_1000.csv")

data.drop_duplicates(inplace=True)
data.fillna(data.mean(numeric_only=True), inplace=True)
data['Runtime'] = data['Runtime'].str.replace(' min', '')
data['Runtime'] = data['Runtime'].astype(int)

df = data['Genre'].str.contains('Action', 'Sci-fi')
print(data[df])
```

Output:

```
                                          Poster_Link  ...        Gross
2    https://m.media-amazon.com/images/M/MV5BMTMxNT...  ...  534,858,444
5    https://m.media-amazon.com/images/M/MV5BNzA5ZD...  ...  377,845,905
8    https://m.media-amazon.com/images/M/MV5BMjAxMz...  ...  292,576,195
10   https://m.media-amazon.com/images/M/MV5BN2EyZj...  ...  315,544,750
13   https://m.media-amazon.com/images/M/MV5BZGMxZT...  ...  342,551,365
..                                                 ...  ...          ...
968  https://m.media-amazon.com/images/M/MV5BYjcxMz...  ...   40,903,593
979  https://m.media-amazon.com/images/M/MV5BZTllNW...  ...   65,207,127
982  https://m.media-amazon.com/images/M/MV5BN2VlNj...  ...   12,465,371
983  https://m.media-amazon.com/images/M/MV5BYTU2MW...  ...   22,490,039
985  https://m.media-amazon.com/images/M/MV5BNDQ3Mz...  ...   43,000,000

[189 rows x 16 columns]
```

- Find movies that were released between 2000 and 2015.

Source Code:

```
data = pd.read_csv("imdb_top_1000.csv")

data.drop_duplicates(inplace=True)
data.fillna(data.mean(numeric_only=True), inplace=True)
data['Runtime'] = data['Runtime'].str.replace(' min', '')
data['Runtime'] = data['Runtime'].astype(int)

year_range = data[(data['Released_Year'] > '2000') & (data['Released_Year'] > '2015')]
print(year_range)
```

Output:

```
                                Poster_Link       Series_Title  ... No_of_Votes       Gross
18   https://m.media-amazon.com/images/M/MV5BNjViNW...           Hamilton  ...       55291         NaN
19   https://m.media-amazon.com/images/M/MV5BYWZjMj...       Gisaengchung  ...      552778  53,367,844
20   https://m.media-amazon.com/images/M/MV5BOTc2ZT...     Soorarai Pottru  ...       54995         NaN
33   https://m.media-amazon.com/images/M/MV5BNGVjNW...              Joker  ...      939252  335,451,311
53   https://m.media-amazon.com/images/M/MV5BMmExNz...          Capharnaüm  ...       62635   1,661,096
..                                                ...                 ...  ...         ...         ...
891  https://m.media-amazon.com/images/M/MV5BMTEzNz...       Incredibles 2  ...      250057  608,581,744
892  https://m.media-amazon.com/images/M/MV5BMjI4Mz...              Moana  ...      272784  248,757,044
896  https://m.media-amazon.com/images/M/MV5BMTg4ND...  Hell or High Water  ...      204175   26,862,450
903  https://m.media-amazon.com/images/M/MV5BNmE5Zm...      A Star Is Born  ...      334312  215,288,866
966  https://m.media-amazon.com/images/M/MV5BNjEzYj...           Apollo 13  ...      269197  173,837,933

[99 rows x 16 columns]
```

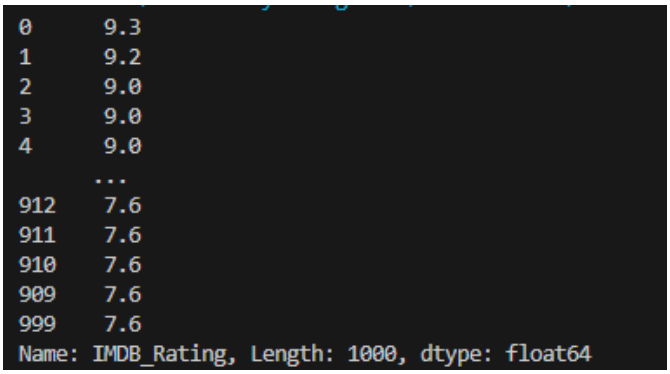
- Sort the dataset based on IMDB rating in descending order.

Source Code:

```
data = pd.read_csv("imdb_top_1000.csv")

data.drop_duplicates(inplace=True)
```

```
data.fillna(data.mean(numeric_only=True), inplace=True)
data['Runtime'] = data['Runtime'].str.replace(' min', '')
data['Runtime'] = data['Runtime'].astype(int)

data.sort_values(by='IMDB_Rating', ascending=False, inplace=True)
print(data['IMDB_Rating'])
```

Output:



Data Aggregation & Grouping.

- Find the average IMDB rating for each genre.

Source Code:

```
data = pd.read_csv("imdb_top_1000.csv")

data.drop_duplicates(inplace=True)
data.fillna(data.mean(numeric_only=True), inplace=True)
data['Runtime'] = data['Runtime'].str.replace(' min', '')
data['Runtime'] = data['Runtime'].astype(int)

avg = data.groupby('Genre')['IMDB_Rating'].mean()
print(avg)
```

Output:

```
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
```

- Determine which year had the most movies released.

Source Code:

```
data = pd.read_csv("imdb_top_1000.csv")

data.drop_duplicates(inplace=True)
data.fillna(data.mean(numeric_only=True), inplace=True)
data['Runtime'] = data['Runtime'].str.replace(' min', '')
data['Runtime'] = data['Runtime'].astype(int)

year_released = data.groupby('Released_Year')['Released_Year'].count().sort_values(ascending=False).head(1)
print("The most movies released year is: ", year_released)
```

Output:

```
The most movies released year is:  Released_Year
2014    32
Name: Released_Year, dtype: int64
```

- Find the top 5 Directors who have directed the most movies in the dataset.

Source Code:

```
data = pd.read_csv("imdb_top_1000.csv")

data.drop_duplicates(inplace=True)
data.fillna(data.mean(numeric_only=True), inplace=True)
data['Runtime'] = data['Runtime'].str.replace(' min', '')
data['Runtime'] = data['Runtime'].astype(int)

top = data.groupby('Director')['Series_Title'].count().sort_values(ascending=False).head()
print(top)
```

Output:

```
Director
Alfred Hitchcock    14
Steven Spielberg    13
Hayao Miyazaki      11
Martin Scorsese     10
Akira Kurosawa      10
Name: Series_Title, dtype: int64
```

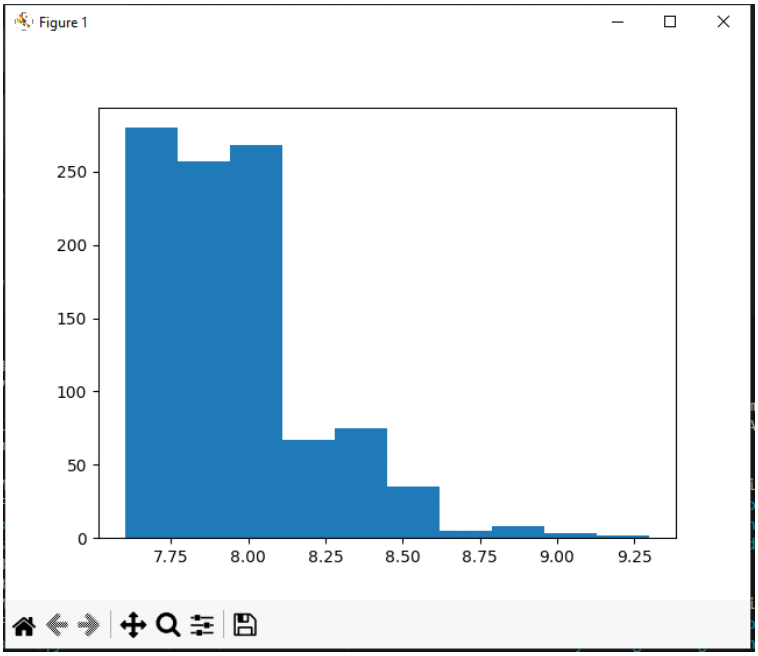
Visualization
Matplotlib or **Seaborn**

- Plot a histogram of IMDB Rating.

Source Code:

```
import pandas as pd
import matplotlib.pyplot as plt

data = pd.read_csv("imdb_top_1000.csv")
plt.hist(data["IMDB_Rating"])
plt.show()
```

Output:

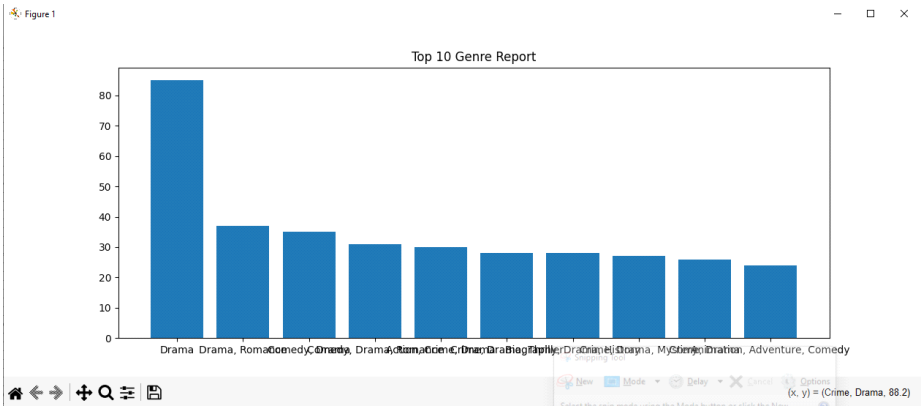


- Create a bar chart showing the top 10 genres with the most movies.

Source Code:

```
import pandas as pd
import matplotlib.pyplot as plt

data = pd.read_csv("imdb_top_1000.csv")
genre = data.groupby('Genre')['Series_Title'].count().sort_values(ascending=False).head(10)
print(genre.to_numpy())

plt.bar(genre.index, genre.values)
plt.title("Top 10 Genre Report")

plt.show()
```

Output:

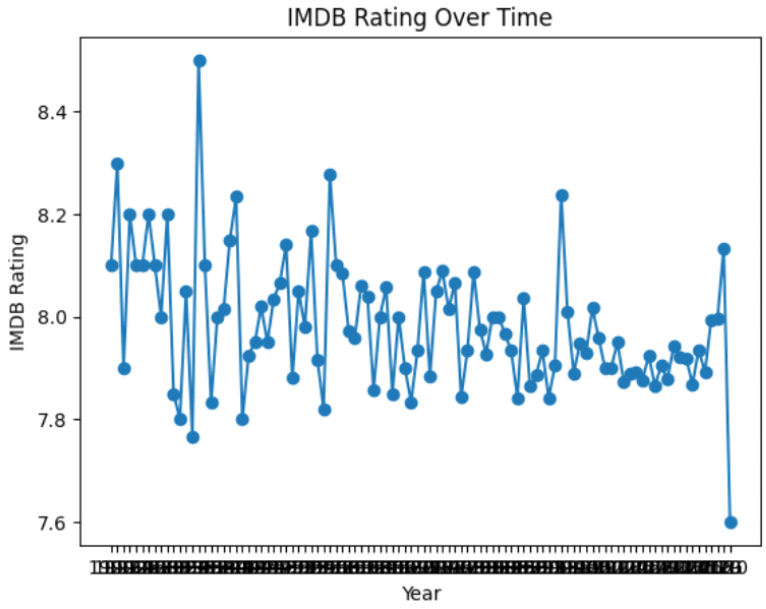


- Visualize the trend of IMDB ratings over the years.

Source Code:

```
import pandas as pd
import matplotlib.pyplot as plt

data = pd.read_csv("imdb_top_1000.csv")

rating = data.groupby('Released_Year')['IMDB_Rating'].mean()
plt.plot(rating.index, rating.values, 'o-')
plt.xlabel('Year')
plt.ylabel('IMDB Rating')
plt.title('IMDB Rating Over Time')
plt.show()
```

Output:



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: -

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

ANS:

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

Source Code:

```
import collections

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

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

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

    return visited

graph = {
    0: [1,2,3,4],
    1: [0, 3],
    2: [0, 3],
    3: [0, 1, 2],
    4: [0]
}

root = 0

# Function Call
```

```
print("The traversing using Breadth-First-Search is: ", bfs(graph, root))
```

Output:

```
The traversing using Breadth-First-Search is:  {0, 1, 2, 3, 4}
```

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

Source Code:

```
import collections

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

    while queue:
        value = queue.popleft()
        visited.add(value)
        print(value)

        if goal == value:
            print("Goal state found")
            return

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

    return visited

graph = {
    0: [1,2,3,4],
    1: [0, 3],
    2: [0, 3],
    3: [0, 1, 2],
    4: [0]
}

root = 0
goal = 2

# Function Call
```

```
bfs(graph, root, goal)
```

Output:

```
0
1
2
Goal state 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: -

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

ANS:

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

Source Code:

```
graph = {
    0: [2, 4, 5],
    1: [2, 3, 4],
    2: [0, 1],
    3: [1],
    4: [0, 1],
    5: [0]
}

visited = set()

root = 0
```

```
def dfs(visited, graph, root):
    if root not in visited:
        queue = []
        queue.append(root)
        visited.add(root)
        value = queue.pop()
        print(value)

        for neighbor in graph[root]:
            dfs(visited, graph, neighbor)

# Function Call
dfs(visited, graph, root)
```

Output:

```
0
2
1
3
4
5
```

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

Source Code:

```
def dfs(graph, root, goal):
    visited = set()
    stack = [root]

    while stack:
        value = stack.pop()

        if value not in visited:
            visited.add(value)
            print(value)

        if goal == value:
            print("Goal state found")
            return

        for i in graph[value]:
            if i not in visited:
                stack.append(i)

graph = {
```

```
    0: [2, 4, 5],
    1: [2, 3, 4],
    2: [0, 1],
    3: [1],
    4: [0, 1],
    5: [0]
}

root = 0
goal = 1

# Function Call
dfs(graph, root, goal)
```

Output:

```
0
5
4
1
Goal state 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: -

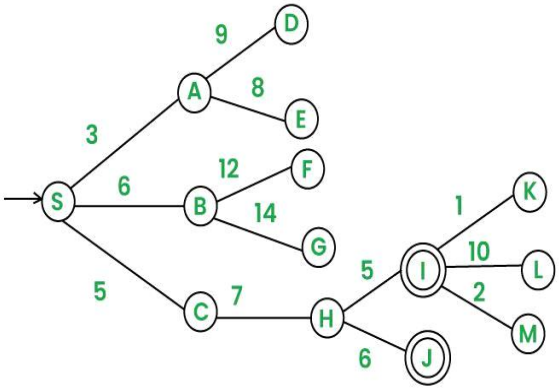
A basic priority-based search where the algorithm always chooses the next node alphabetically or based on node depth (depending on the implementation) is an example of Best-First Search.

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.



Best First Search (Informed Search)

ANS:

Source Code:

```
import heapq

def bestfs(visited, graph, root, goal):
    queue = [[0, root]]

    while queue:
        heapq.heapify(queue)
        value = heapq.heappop(queue)

        if value[1] not in visited:
            visited.append(value[1])

        if goal == value[1]:
            return visited

        for i in graph[value[1]]:
            if i[1] not in visited:
                queue.append(i)

    return visited

graph = {
    'S': [[3, 'A'], [6, 'B'], [5, 'C']],
    'A': [[9, 'D'], [8, 'E']],
    'B': [[12, 'F'], [14, 'G']],
```

```
    'C': [[7, 'H']],
    'H': [[5, 'I'], [6, 'J']],
    'I': [[1, 'K'], [10, 'L'], [2, 'M']],
    'D': [],
    'E': [],
    'F': [],
    'G': [],
    'J': [],
    'K': [],
    'L': [],
    'M': []
}

root = 'S'
visited = [root]
goal = 'I'

bestfs(visited, graph, root, goal)

length = len(visited) - 1

print("The Path from S to I is: ")
for i in range(length):
    print(visited[i], end=' -> ')

print(visited[-1])
```

Output:

```
The Path from S to I is:
S -> A -> C -> B -> H -> I
```

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: -

- Initialize the open list (priority queue) with the start node.
- 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.

ANS:

Source Code:

```python
import heapq

def a_star(graph, start, goal, heuristic):
    open_list = [(heuristic[start], 0, start, [start])]
    g_scores = {start: 0}

    while open_list:
        a, g, node, path = heapq.heappop(open_list)

        if node == goal:
            return path, g

        for cost, neighbor in graph[node]:
            new_g = g + cost
            if neighbor not in g_scores or new_g < g_scores[neighbor]:
                g_scores[neighbor] = new_g
                f = new_g + heuristic[neighbor]
                heapq.heappush(open_list, (f, new_g, neighbor, path +
[neighbor]))

    return None, float('inf')

# Example
graph = {
    'A': [[5, 'B'], [3, 'C']],
    'B': [[2, 'D'], [1, 'E']],
    'C': [[4, 'D'], [8, 'F']],
    'D': [[3, 'E'], [5, 'G']],
    'E': [[2, 'G']],
    'F': [[1, 'G']],
    'G': []
}

heuristic = {
    'A': 10, 'B': 7, 'C': 8, 'D': 5, 'E': 3, 'F': 2, 'G': 0
}

path, cost = a_star(graph, 'A', 'G', heuristic)

if path:
    print("Path:", " -> ".join(path))
    print("Cost:", cost)
else:
```

```
    print("No path found.")
```

Output:

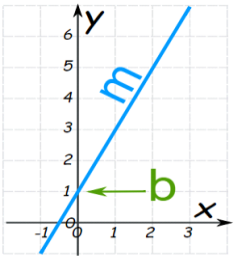
```
Path: A -> B -> E -> G
Cost: 8
```

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.



$$price = m * area + b$$

$$y = mx + b$$

Slope (or Gradient) Y Intercept

Reference: https://www.mathsisfun.com/algebra/linear-equations.html

The model has the form:

Y=mX+b

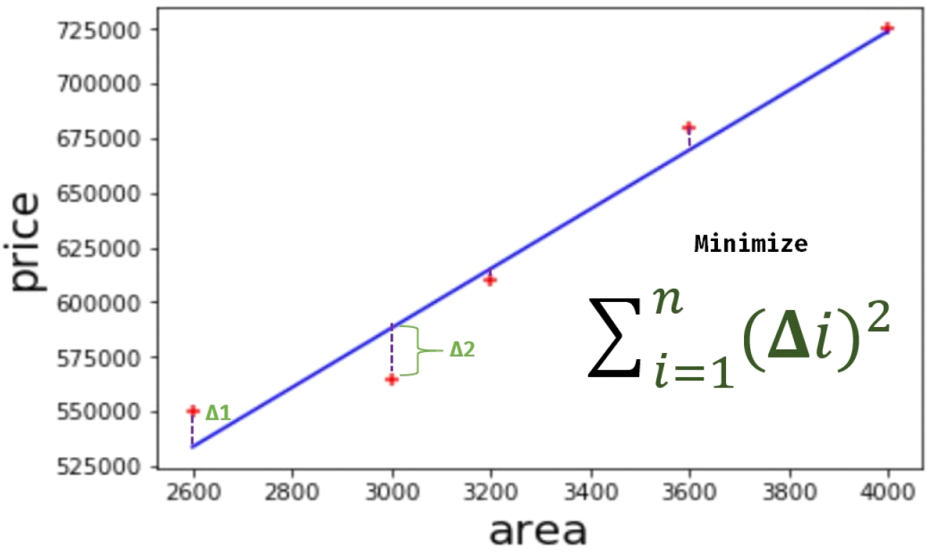
Where:

- 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.

ANS:

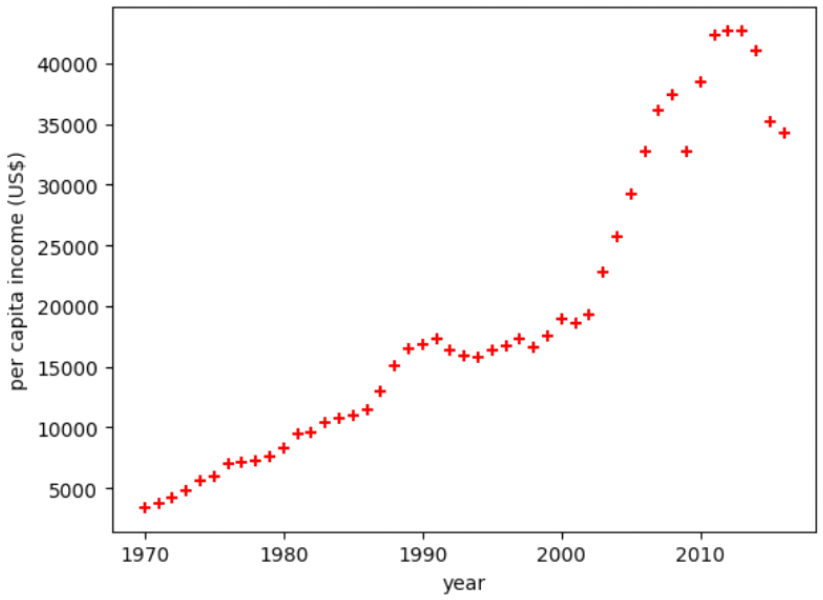
Source Code:

```
[1] import pandas as pd
    import matplotlib.pyplot as plt
    from sklearn import linear_model

    %matplotlib inline
```

```
[37] data = pd.read_csv('canada_per_capita_income_exercise.csv')
```

```
plt.xlabel('year')
plt.ylabel('per capita income (US$)')
plt.scatter(data["year"], data["per capita income (US$)"], c = "r", marker = "+")
```

```
<matplotlib.collections.PathCollection at 0x78dab6f055d0>
```




```
[38] year = data.drop("per capita income (US$)", axis = "columns")
```

```
[39] income = data["per capita income (US$)"]
```

```
reg = linear_model.LinearRegression()
reg.fit(year, income)
```

```
LinearRegression
LinearRegression()
```

Predict for Year 2020 (using .predict() method and y = m*x + b method):

```
[41] reg.predict([[2020]])

/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names,
  warnings.warn(
array([41288.69409442])
```

```
reg.coef_

array([828.46507522])
```

```
[43] reg.intercept_

np.float64(-1632210.7578554575)
```

```
[44] val = reg.coef_ * 2020 + reg.intercept_
     val

array([41288.69409442])
```

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?

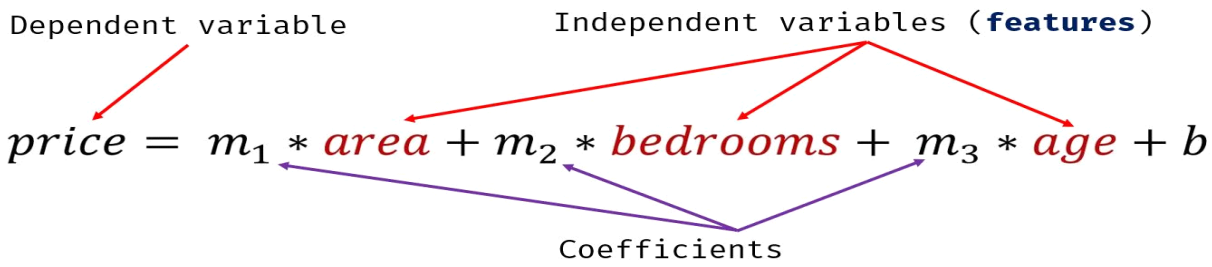
Multivariate Linear Regression extends simple linear regression by modeling the relationship between a dependent variable (Y) and multiple independent variables (X_1, X_2, ..., X_n).

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)



$$y = m_1 x_1 + m_2 x_2 + m_3 x_3 + b$$

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

ANS:

Source Code:

```
[1]  import pandas as pd
     import matplotlib.pyplot as plt
     from sklearn import linear_model

     %matplotlib inline
```

```
[46] f = pd.read_csv('hiring.csv')
```

```
f.describe(include=object)
```

	experience
count	6
unique	6
top	five
freq	1

```
[49] f.experience = f["experience"].fillna("five")
```

```
[50] f["test_score(out of 10)"] = f["test_score(out of 10)"].fillna(f["test_score(out of 10)"].median())
```

```
f
```

	experience	test_score(out of 10)	interview_score(out of 10)	salary($)
0	five	8.0	9	50000
1	five	8.0	6	45000
2	five	6.0	7	60000
3	two	10.0	10	65000
4	seven	9.0	6	70000
5	three	7.0	10	62000
6	ten	8.0	7	72000
7	eleven	7.0	8	80000

```
num = [5, 5, 5, 2, 7, 3, 10, 11]
for idx, i in enumerate(f["experience"]):
  f["experience"][idx] = num[idx]
```

```
f
```

	experience	test_score(out of 10)	interview_score(out of 10)	salary($)
0	5	8.0	9	50000
1	5	8.0	6	45000
2	5	6.0	7	60000
3	2	10.0	10	65000
4	7	9.0	6	70000
5	3	7.0	10	62000
6	10	8.0	7	72000
7	11	7.0	8	80000

```
[17] reg_mul = linear_model.LinearRegression()
     reg_mul.fit(f.drop("salary($)", axis="columns"), f["salary($)"])
```

```
LinearRegression()
```

Predict for 2 yr experience, 9 test score, 6 interview score (using .predict() method and y = m1 * x1 + m2 * x2 + m3 * x3 + b):

```
[18] reg_mul.predict([[2, 9, 6]])
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names,
  warnings.warn(
array([45979.08171436])
```

```
[19] reg_mul.coef_
```

```
array([3221.39134934, 1617.86554643, 3176.24086827])
```

```
[20] reg_mul.intercept_
```

```
np.float64(5918.063888238692)
```

```
[21] val = 3221.39134934 * 2 + 1617.86554643 * 9 + 3176.24086827 * 6 + 5918.063888238692
     val
```

```
45979.08171440869
```

Predict for 12 yr experience, 10 test score, 10 interview score (using .predict() method):

```
reg_mul.predict([[12, 10, 10]])
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names,
  warnings.warn(
array([92515.82422727])
```

Predict for 12 yr experience, 10 test score, 10 interview score (Using y = m1 * x1 + m2 * x2 + m3 * x3 + b):

```
[72] y = reg_mul.coef_[0] * 12 + reg_mul.coef_[1] * 10 + reg_mul.coef_[2] * 10 + reg_mul.intercept_
     y
```

```
np.float64(92515.8242272677)
```

Lab No: 10

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

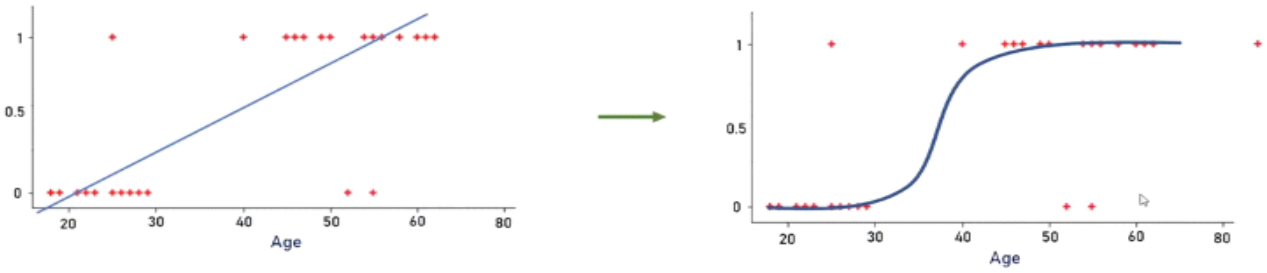
What is Logistic Regression?

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.

$$y = m * x + b$$

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



Sigmoid Function: -

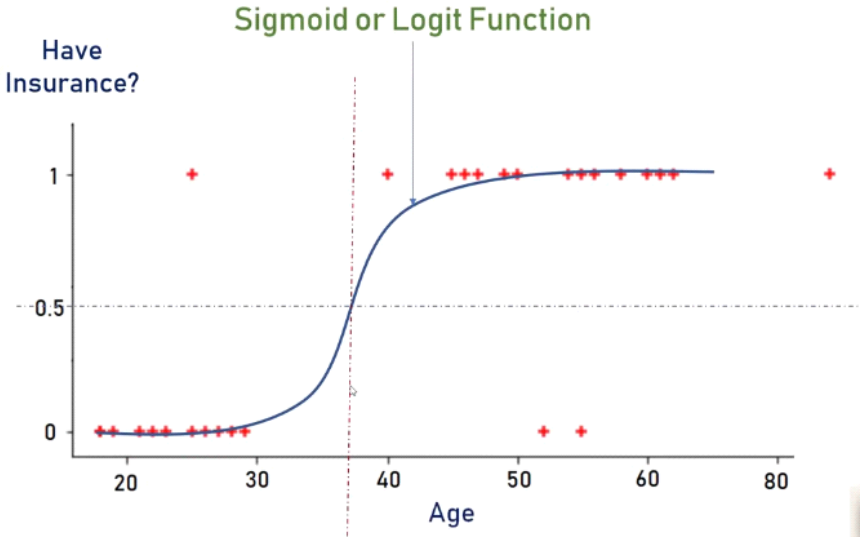
$$sigmoid(z) = \frac{1}{1 + e^{-z}}$$

e = Euler's number ~ 2.71828

Sigmoid function converts input into range 0 to 1

Where:

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



Key Concepts

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

Tasks:

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

- 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)
- Plot bar charts showing impact of employee salaries on retention
- Plot bar charts showing correlation between department and employee retention
- Now build logistic regression model using variables that were narrowed down in step 1
- Measure the accuracy of the model

Source Code:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import OneHotEncoder

%matplotlib inline
```

```
data = pd.read_csv("HR_comma_sep_exercise.csv")
data
```

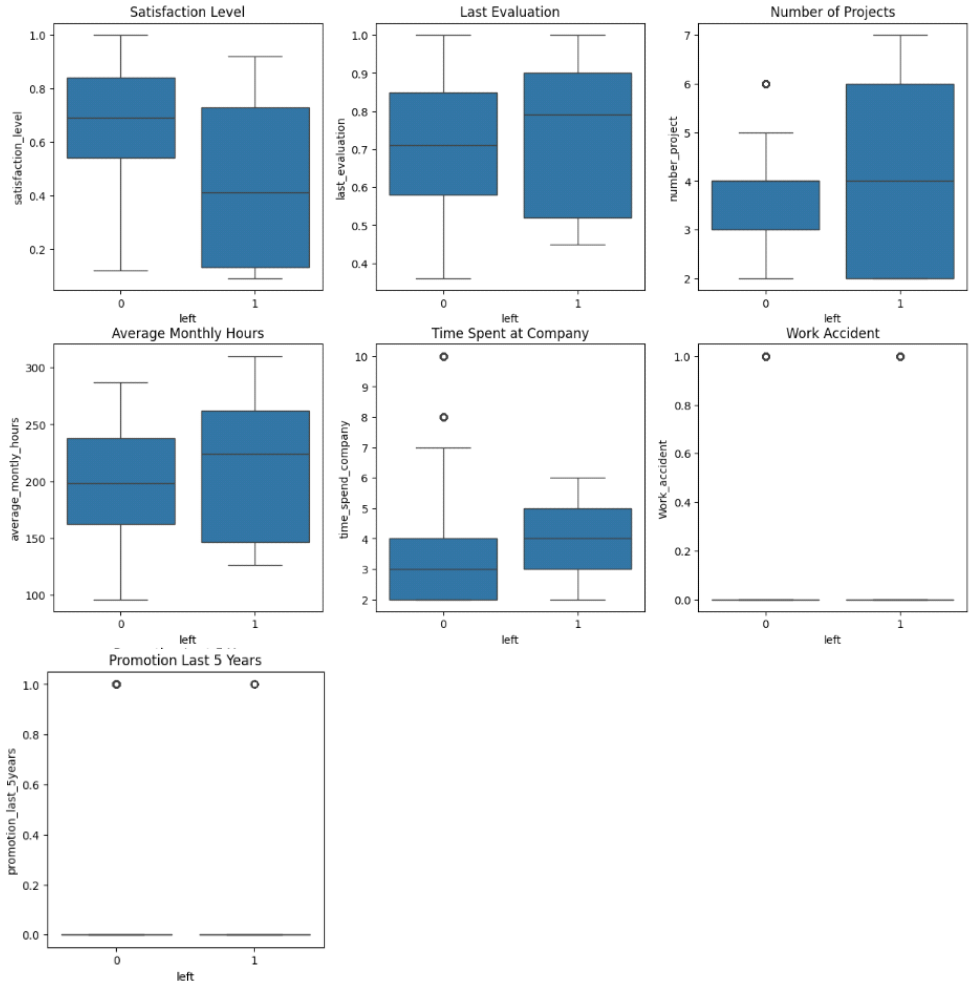
	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	left	promotion_last_5years	Department	salary
0	0.38	0.53	2	157	3	0	1	0	sales	low
1	0.80	0.86	5	262	6	0	1	0	sales	medium
2	0.11	0.88	7	272	4	0	1	0	sales	medium
3	0.72	0.87	5	223	5	0	1	0	sales	low
4	0.37	0.52	2	159	3	0	1	0	sales	low
...	...	...	...	...	...	...	...	...	...	...
14994	0.40	0.57	2	151	3	0	1	0	support	low
14995	0.37	0.48	2	160	3	0	1	0	support	low
14996	0.37	0.53	2	143	3	0	1	0	support	low
14997	0.11	0.96	6	280	4	0	1	0	support	low
14998	0.37	0.52	2	158	3	0	1	0	support	low

14999 rows × 10 columns

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

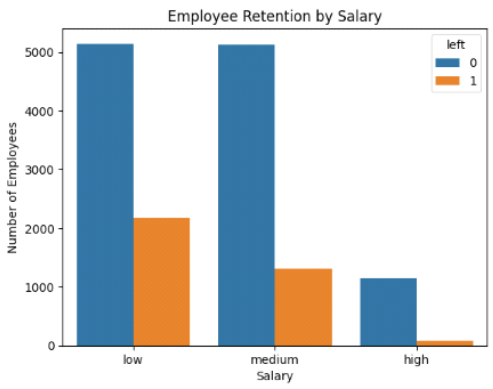
```
plt.figure(figsize=(15, 15))

plt.subplot(3, 3, 1)
sns.boxplot(data, x=data['left'], y=data['satisfaction_level'])
plt.title('Satisfaction Level')

plt.subplot(3, 3, 2)
sns.boxplot(data, x=data['left'], y=data['last_evaluation'])
plt.title('Last Evaluation')

plt.subplot(3, 3, 3)
sns.boxplot(data, x=data['left'], y=data['number_project'])
plt.title('Number of Projects')

plt.subplot(3, 3, 4)
sns.boxplot(data, x=data['left'], y=data['average_montly_hours'])
plt.title('Average Monthly Hours')

plt.subplot(3, 3, 5)
sns.boxplot(data, x=data['left'], y=data['time_spend_company'])
plt.title('Time Spent at Company')

plt.subplot(3, 3, 6)
sns.boxplot(data, x=data['left'], y=data['Work_accident'])
plt.title('Work Accident')

plt.subplot(3, 3, 7)
sns.boxplot(data, x=data['left'], y=data['promotion_last_5years'])
plt.title('Promotion Last 5 Years')
```

```
Text(0.5, 1.0, 'Promotion Last 5 Years')
```



Plot bar charts showing impact of employee salaries on retention

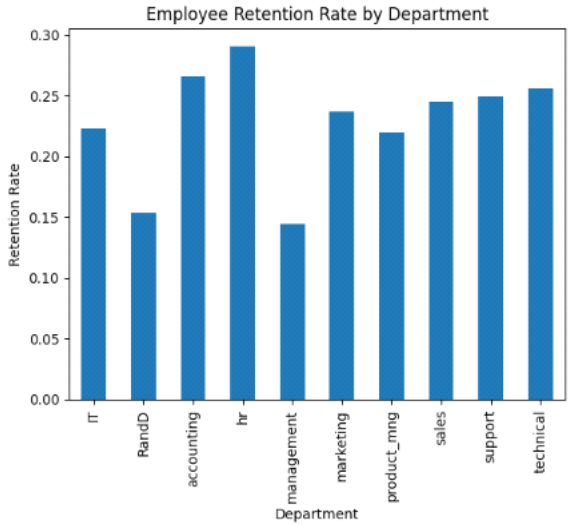
```
[ ] sns.countplot(data=data, x='salary', hue='left')
    plt.title('Employee Retention by Salary')
    plt.xlabel('Salary')
    plt.ylabel('Number of Employees')
    plt.show()
```



Plot bar charts showing corelation between department and employee retention

```
rate_by_department = data.groupby('Department')['left'].mean()

rate_by_department.plot(kind='bar')
plt.title('Employee Retention Rate by Department')
plt.xlabel('Department')
plt.ylabel('Retention Rate')
```

```
Text(0, 0.5, 'Retention Rate')
```



Now build Logistic Regression model using variables that were narrowed down in step 1

```
[37] categorical_columns = data.select_dtypes(include=["object"]).columns.tolist()
     categorical_columns
```

```
['Department', 'salary']
```

```
encoder = OneHotEncoder(sparse_output=False)
one_hot_encoder = encoder.fit_transform(data[categorical_columns])
one_hot_encoder
```

```
array([[0., 0., 0., ..., 0., 1., 0.],
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 1.],
       ...,
       [0., 0., 0., ..., 0., 1., 0.],
       [0., 0., 0., ..., 0., 1., 0.],
       [0., 0., 0., ..., 0., 1., 0.]])
```

```
one_hot_df = pd.DataFrame(one_hot_encoder, columns=encoder.get_feature_names_out(categorical_columns))
one_hot_df
```

	Department_IT	Department_RandD	Department_accounting	Department_hr	Department_management	Department_marketing	Depar
0	0.0	0.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	
14994	0.0	0.0	0.0	0.0	0.0	0.0	
14995	0.0	0.0	0.0	0.0	0.0	0.0	
14996	0.0	0.0	0.0	0.0	0.0	0.0	
14997	0.0	0.0	0.0	0.0	0.0	0.0	
14998	0.0	0.0	0.0	0.0	0.0	0.0	

14999 rows × 13 columns

Department_product_mng	Department_sales	Department_support	Department_technical	salary_high	salary_low	salary_medium
0.0	1.0	0.0	0.0	0.0	1.0	0.0
0.0	1.0	0.0	0.0	0.0	0.0	1.0
0.0	1.0	0.0	0.0	0.0	0.0	1.0
0.0	1.0	0.0	0.0	0.0	1.0	0.0
0.0	1.0	0.0	0.0	0.0	1.0	0.0
...	...	...	...	...	...	...
0.0	0.0	1.0	0.0	0.0	1.0	0.0
0.0	0.0	1.0	0.0	0.0	1.0	0.0
0.0	0.0	1.0	0.0	0.0	1.0	0.0
0.0	0.0	1.0	0.0	0.0	1.0	0.0
0.0	0.0	1.0	0.0	0.0	1.0	0.0

```
[50] join_columns = pd.concat([data, one_hot_df])
```

```
[51] join_columns = join_columns.drop(categorical_columns, axis=1)
```

```
[52] for i in join_columns.columns:
       join_columns[i].fillna(0.0, inplace=True)
```

```
/tmp/ipython-input-52-2566369569.py:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are

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


  join_columns[i].fillna(0.0, inplace=True)
```

```
[53] from sklearn.model_selection import train_test_split
     X_train, X_test, y_train, y_test = train_test_split(join_columns.drop("left", axis = 1), join_columns.left,train_size=0.9)
```

```
X_train
```

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	promotion_
560	0.00	0.00	0.0	0.0	0.0	0.0	
9444	0.00	0.00	0.0	0.0	0.0	0.0	
12433	0.00	0.00	0.0	0.0	0.0	0.0	
7934	0.00	0.00	0.0	0.0	0.0	0.0	
8583	0.78	0.61	3.0	227.0	3.0	0.0	
...	...	...	...	...	...	...	
6057	0.63	0.49	3.0	252.0	3.0	0.0	
2190	0.87	0.52	3.0	237.0	3.0	0.0	
12654	0.10	0.89	7.0	308.0	4.0	0.0	
4470	0.00	0.00	0.0	0.0	0.0	0.0	
992	0.45	0.57	2.0	151.0	3.0	0.0	

26998 rows × 20 columns

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
```

```
[56] model.fit(X_train, y_train)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

LogisticRegression

LogisticRegression()

```
[57] y_predicted = model.predict(X_test)
y_predicted
```

```
array([0., 0., 0., ..., 0., 0., 0.])
```

Measure the accuracy of the model

```
model.score(X_test,y_test)
```

```
0.8786666666666667
```

```
[59] from sklearn.metrics import accuracy_score
acc = accuracy_score(y_test, y_predicted)
acc
```

```
0.8786666666666667
```