A

Journal on
Data Science
Soft Computing

DESIGNED BY

MISS. SHRAVANI DILIP SUTAR

SEAT NO.________________

Submitted To

K.M.S.P MANDAL'S

SANT RAWOOL MAHARAJ MAHAVIDYALAYA, KUDAL

(NAAC Accreditation "B+" Grade)

In the partial fulfilment of

M.Sc. Information Technology

(Part 1)

Under Guidance of

Asst. Prof. K. A. Kubal

Asst. Prof. P. S. Keravadekar

Through

DEPARTMENT OF INFORMATION TECHNOLOGY

A

Journal on

Data Science

Soft Computing

DESIGNED BY

MISS. SHRAVANI DILIP SUTAR

SEAT NO.______________

Submitted To

K.M.S.P MANDAL'S

SANT RAWOOL MAHARAJ MAHAVIDYALAYA, KUDAL

(NAAC Accreditation "B+" Grade)

In the partial fulfilment of

M.Sc. Information Technology

(Part 1)

Under Guidance of

Asst. Prof. K. A. Kubal

Asst. Prof. P. S. Keravadekar

Through

DEPARTMENT OF INFORMATION TECHNOLOGY

DATA SCIENCE

UNIVERSITY OF MUMBAI

K.M.S.P. Mandal's

Sant Rawool Maharaj Mahavidyalaya

Kudal, Dist-Sindhudurg

DEPARTMEMT OF INFORMATION TECHNOLOGY

CERTIFICATE

This to certify that,

Mr / Miss. Shravani Dilip Sutar,

Exam Seat No. __________________ student of the M.Sc. (Information Technology, Part-I). He / She has been successfully completed practical as prescribed by University of Mumbai in the Sant Rawool Maharaj Mahavidyalaya during the semester I of the academic year 2025-26

In the following topics

____

____

Teacher In -Charge: **External Examiner:**

Date:

INDEX

Sr no.	Title	Sign
1.	Creating and using database in Cassandra	
2.	Conversion from different formats to HOURS format. a. Text b. XML c. JSON d. Picture (JPEG) e. Video, Audio f. MySQL Database	
3.	Write the program for following: a. Fixer Utilities b. Averaging c. Outlier Detection d. Logging e. Data Binning / Bucketing	
4.	Implement the Following a. Data processing using R b. Program retrieve different attributes of data c. Data Pattern	
5.	Implementation of Error Management, Network Routing, Generating GML Files, Warehouses Location, Simple Forex Trading Planner, Generating Payroll, Balance Sheet	
6.	Build the time hub, links and satellites.	
7.	Transforming data.	
8.	Organizing Data	
9.	Generating Data	
10.	Data visualisation using Power BI	

K. M. S. P Mandal's **Sant Rawool Maharaj Mahavidyalaya, Kudal** **Department of Information Technology**	**Date: 02/08/2025** **Roll No: 20** **Expt. No: 01**	**Signature**
Title: Creating and using database in Cassandra		

Cassandra:
Apache Cassandra is a distributed NoSQL database designed to handle very large amounts of data across many servers without any single point of failure. Cassandra stores data on multiple machines at the same time, so even if one machine fails, the data is still available. This makes it highly reliable and scalable.

Installing Cassandra by using Docker Desktop:
Using Docker to run Cassandra makes installation very easy and fast, without worrying about system setup or errors. Cassandra runs in an isolated container, so it does not affect other software on your system. Docker allows you to easily start, stop, or delete Cassandra whenever needed.

Using basic commands:

a) Pulling the Cassandra image into the Docker:
Command used: docker pull cassandra:latest
This command downloads the latest available Cassandra image so you can run Cassandra with the newest features and updates using Docker.

b) Running Cassandra in Docker:
Command used: docker run --name cassandra-db -d Cassandra
This command starts Cassandra inside a Docker container and keeps it running in the background. ('-d' is used to run the Cassandra in background in detached mode)

c) Accessing the Cassandra database shell (CQLSH):
Command used: docker exec -it cassandra-db cqlsh
This command connects you to the Cassandra shell (cqlsh), where you can create keyspaces, tables, and run Cassandra queries.

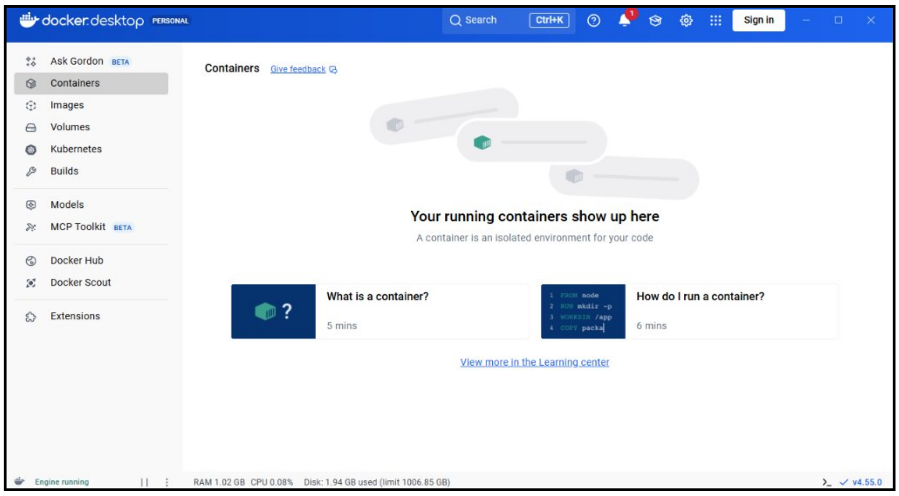
d) Creating keyspace(database):
Command used: CREATE KEYSPACE keyspace_name WITH replication = {'class': 'replication_strategy', 'replication_factor': number};

e) Creating tables:
Command used: CREATE TABLE table_name(column_name1 data_type, column_name2 data_type, ...PRIMARY KEY (column_name));

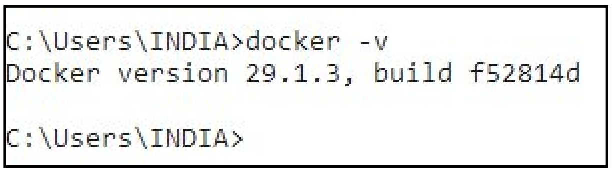
f) Inserting values into the tables:
Command used: INSERT INTO table_name (column1, column2, column3) VALUES (value1, value2, value3);

A) Installing Cassandra by using Docker Desktop

Docker Desktop:

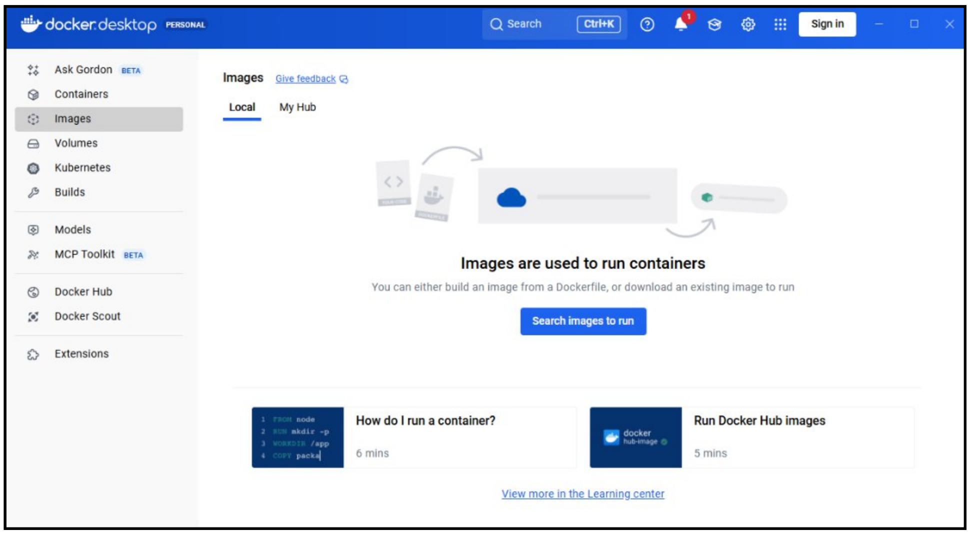


Checking installed Docker Version:

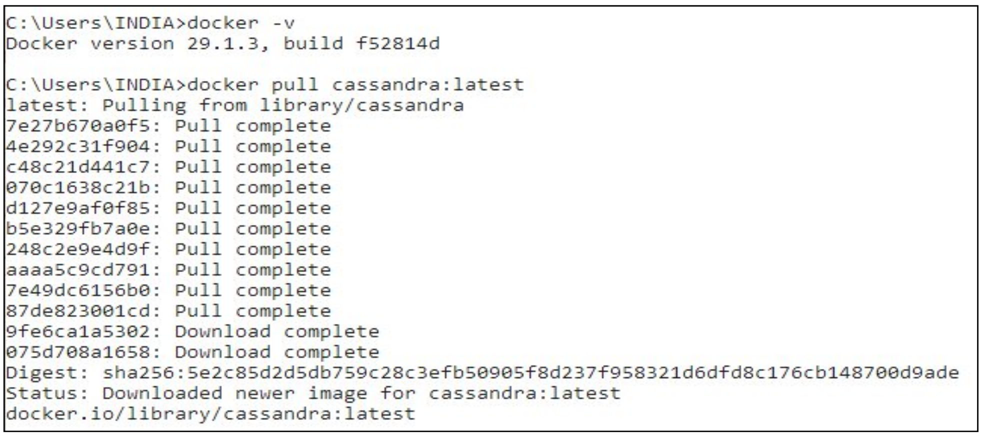


```
C:\Users\INDIA>docker -v
Docker version 29.1.3, build f52814d

C:\Users\INDIA>
```

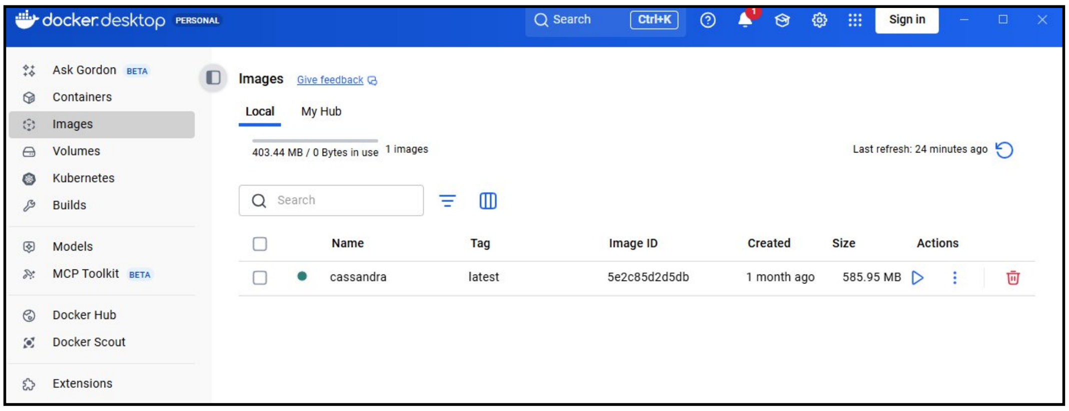
Docker Image



Pull the Cassandra Docker Image:



```
C:\Users\INDIA>docker -v
Docker version 29.1.3, build f52814d

C:\Users\INDIA>docker pull cassandra:latest
latest: Pulling from library/cassandra
7e27b670a0f5: Pull complete
4e292c31f904: Pull complete
c48c21d441c7: Pull complete
070c1638c21b: Pull complete
d127e9af0f85: Pull complete
b5e329fb7a0e: Pull complete
248c2e9e4d9f: Pull complete
aaaa5c9cd791: Pull complete
7e49dc6156b0: Pull complete
87de823001cd: Pull complete
9fe6ca1a5302: Download complete
075d708a1658: Download complete
Digest: sha256:5e2c85d2d5db759c28c3efb50905f8d237f958321d6dfd8c176cb148700d9ade
Status: Downloaded newer image for cassandra:latest
docker.io/library/cassandra:latest
```



Images Give feedback

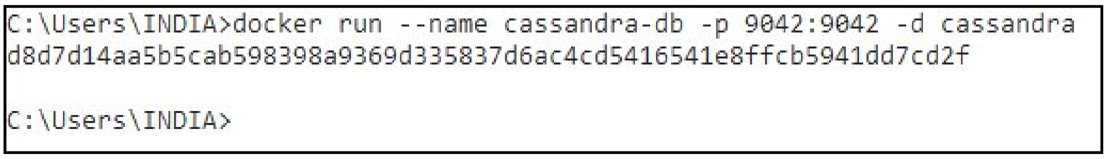
Local My Hub

403.44 MB / 0 Bytes in use 1 images

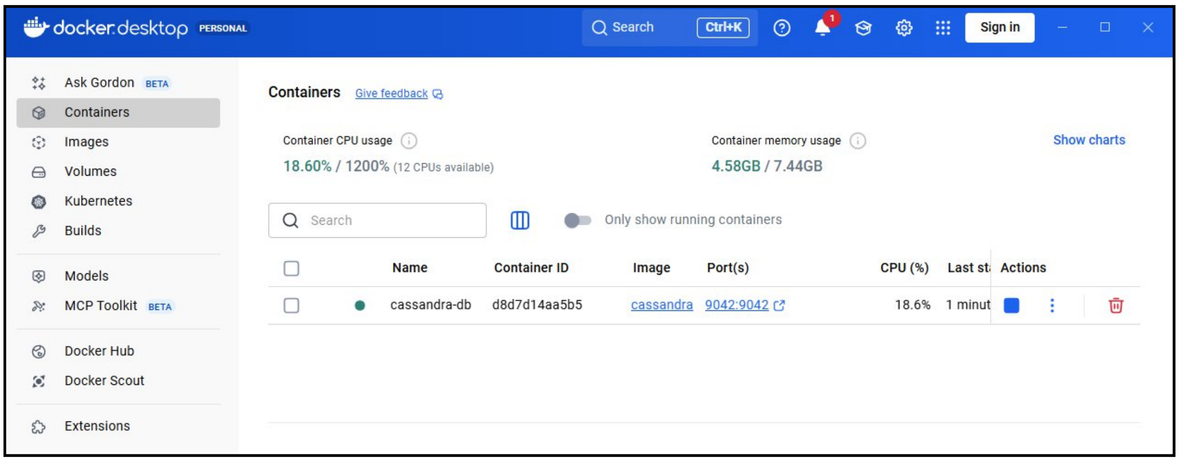
Last refresh: 24 minutes ago

	Name	Tag	Image ID	Created	Size	Actions
●	cassandra	latest	5e2c85d2d5db	1 month ago	585.95 MB	

Running Cassandra in a Docker:



```
C:\Users\INDIA>docker run --name cassandra-db -p 9042:9042 -d cassandra
d8d7d14aa5b5cab598398a9369d335837d6ac4cd5416541e8ffcb5941dd7cd2f

C:\Users\INDIA>
```



Containers Give feedback

Container CPU usage
18.60% / 1200% (12 CPUs available)

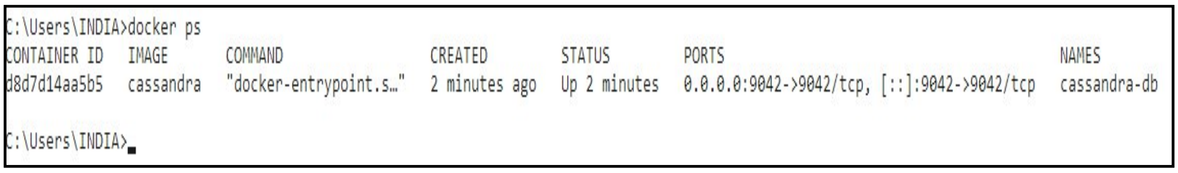
Container memory usage
4.58GB / 7.44GB

Show charts

Only show running containers

	Name	Container ID	Image	Port(s)	CPU (%)	Last st	Actions
●	cassandra-db	d8d7d14aa5b5	cassandra	9042:9042	18.6%	1 minut	

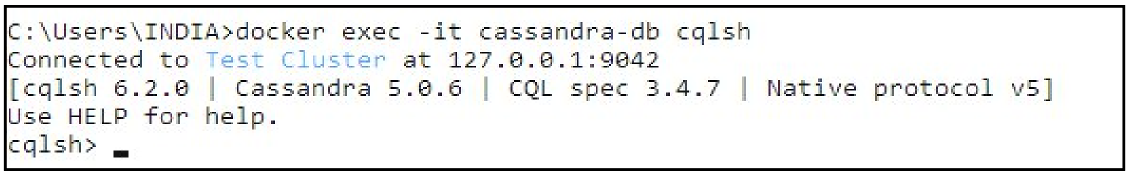
Check if Cassandra in Running:



```
C:\Users\INDIA>docker ps
CONTAINER ID   IMAGE       COMMAND                  CREATED         STATUS         PORTS                                        NAMES
d8d7d14aa5b5   cassandra   "docker-entrypoint.s…"   2 minutes ago   Up 2 minutes   0.0.0.0:9042->9042/tcp, [::]:9042->9042/tcp   cassandra-db

C:\Users\INDIA>
```

Accessing the Cassandra database shell (CQLSH) running inside a Docker container:

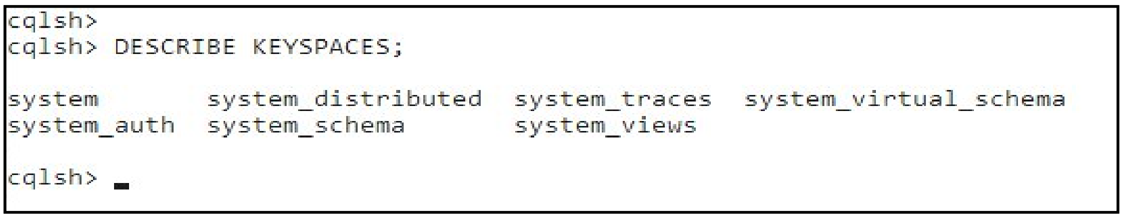
Command used: docker exec -it cassandra-db cqlsh



```
C:\Users\INDIA>docker exec -it cassandra-db cqlsh
Connected to Test Cluster at 127.0.0.1:9042
[cqlsh 6.2.0 | Cassandra 5.0.6 | CQL spec 3.4.7 | Native protocol v5]
Use HELP for help.
cqlsh>
```

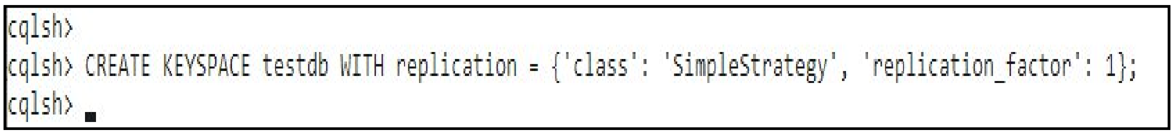
B) Using basic Cassandra Commands:

Show Keyspace:



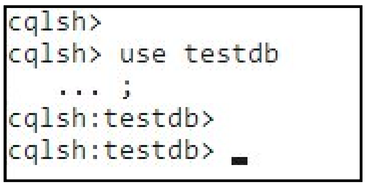
```
cqlsh>
cqlsh> DESCRIBE KEYSPACES;

system       system_distributed  system_traces  system_virtual_schema
system_auth  system_schema       system_views

cqlsh>
```

Creating a Keyspace:



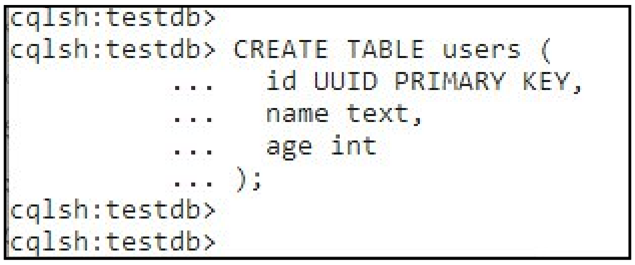
```
cqlsh>
cqlsh> CREATE KEYSPACE testdb WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 1};
cqlsh>
```

Using the Keyspace:



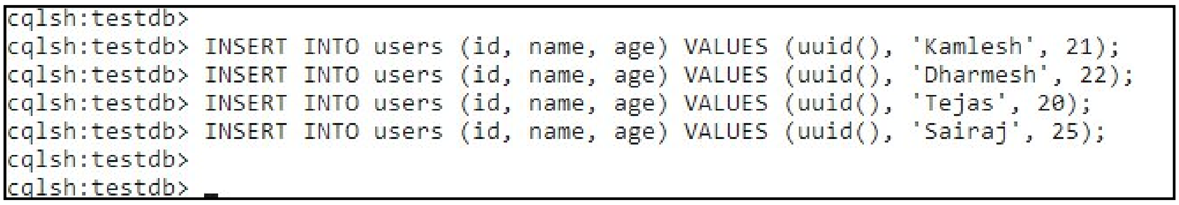
```
cqlsh>
cqlsh> use testdb
   ... ;
cqlsh:testdb>
cqlsh:testdb>
```

Creating a Table in Cassandra:



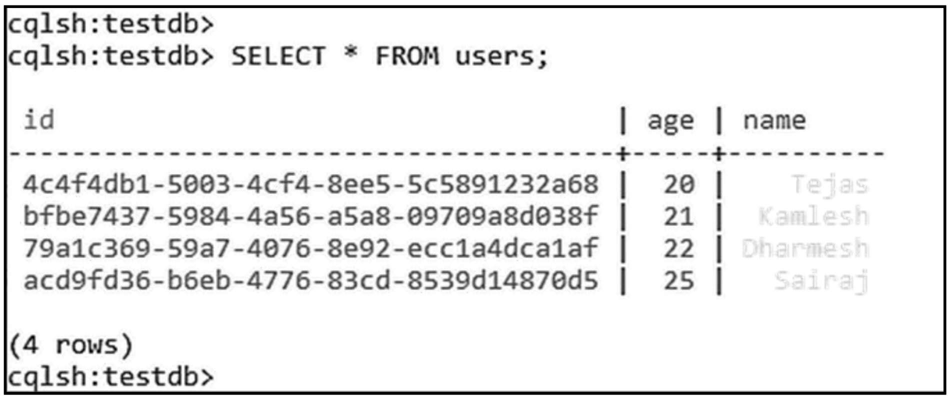
```
cqlsh:testdb>
cqlsh:testdb> CREATE TABLE users (
          ...   id UUID PRIMARY KEY,
          ...   name text,
          ...   age int
          ... );
cqlsh:testdb>
cqlsh:testdb>
```

Inserting data into the Table:



```
cqlsh:testdb>
cqlsh:testdb> INSERT INTO users (id, name, age) VALUES (uuid(), 'Kamlesh', 21);
cqlsh:testdb> INSERT INTO users (id, name, age) VALUES (uuid(), 'Dharmesh', 22);
cqlsh:testdb> INSERT INTO users (id, name, age) VALUES (uuid(), 'Tejas', 20);
cqlsh:testdb> INSERT INTO users (id, name, age) VALUES (uuid(), 'Sairaj', 25);
cqlsh:testdb>
cqlsh:testdb> _
```

Fetching the Data:



```
cqlsh:testdb>
cqlsh:testdb> SELECT * FROM users;

 id                                   | age | name
--------------------------------------+-----+----------
 4c4f4db1-5003-4cf4-8ee5-5c5891232a68 |  20 |    Tejas
 bfbe7437-5984-4a56-a5a8-09709a8d038f |  21 |  Kamlesh
 79a1c369-59a7-4076-8e92-ecc1a4dca1af |  22 | Dharmesh
 acd9fd36-b6eb-4776-83cd-8539d14870d5 |  25 |   Sairaj

(4 rows)
cqlsh:testdb>
```

K. M. S. P Mandal's **Sant Rawool Maharaj Mahavidyalaya, Kudal** **Department of Information Technology**	**Date: 08/08/2025** **Roll No: 20** **Expt. No: 02**	**Signature**
Title: Programs of HORUS Format Conversion		

HORUS Format:

HORUS format is a way to organize and store data, especially for videos, images, or sensor recordings. It structures the data into frames or units along with metadata like timestamps, size, or labels. This makes it easier for programs to read, process, and analyse the information efficiently. In simple words, it's like packing your data neatly with extra details so it can be used quickly by software.

Advantages of using HORUS format:

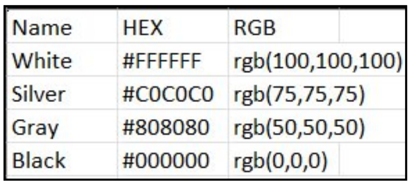
1. **Structured Storage** – Keeps data organized with frames and metadata, making it easy to read and manage.
2. **Efficient Processing** – Programs can quickly access and analyse the data without extra conversions.
3. **Supports Multimedia** – Can handle videos, images, and sensor data in a single format.
4. **Time-stamped Data** – Metadata like timestamps helps in tracking when each frame or data point was recorded.
5. **Portable and Standardized** – Makes sharing and using data across different systems simpler.

a) Text Delimited CSV to HORUS format

Code:

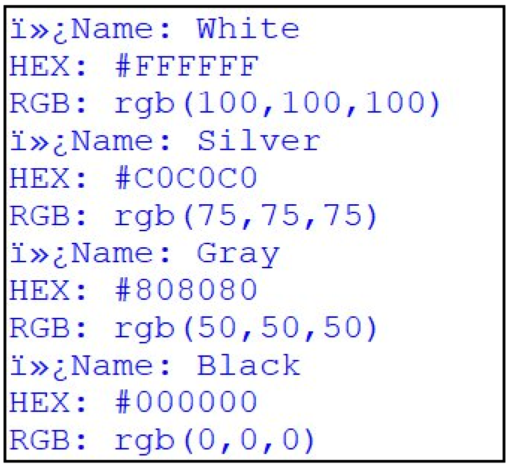
```
import csv
def csv_to_horus(filename, delimiter=','):
    try:
        with open(filename, mode='r') as file:
            reader = csv.DictReader(file, delimiter=delimiter)
            for row in reader:
                for field, value in row.items():
                    print(f"{field}: {value}")
    except FileNotFoundError:
        print("Error: File not found.")
    except Exception as e:
        print("An error occurred:", str(e))
csv_to_horus('color_srgb.csv')
```

CSV File:



Name	HEX	RGB
White	#FFFFFF	rgb(100,100,100)
Silver	#C0C0C0	rgb(75,75,75)
Gray	#808080	rgb(50,50,50)
Black	#000000	rgb(0,0,0)

Output:



```
ï»¿Name: White
HEX: #FFFFFF
RGB: rgb(100,100,100)
ï»¿Name: Silver
HEX: #C0C0C0
RGB: rgb(75,75,75)
ï»¿Name: Gray
HEX: #808080
RGB: rgb(50,50,50)
ï»¿Name: Black
HEX: #000000
RGB: rgb(0,0,0)
```

b) XML to HORUS format

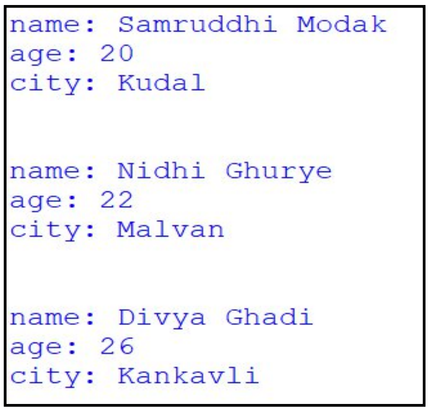
Code:

```
import xml.etree.ElementTree as ET
def xml_to_horus(filename):
    try:
        tree = ET.parse(filename)
        root = tree.getroot()
        for record in root:
            for element in record:
                print(f"{element.tag}: {element.text}")
            print("\n")
    except FileNotFoundError:
        print("Error: File not found.")
    except ET.ParseError:
        print("Error: Failed to parse XML. Make sure it's well-formed.")
    except Exception as e:
        print("An error occurred:", str(e))
xml_to_horus('basic.xml')
```

XML Code/File:

```
<?xml version="1.0" encoding="UTF-8"?>
<root><person>
    <name>Samruddhi Modak</name>
    <age>20</age>
    <city>Kudal</city></person>
  <person>
    <name>Nidhi Ghurye</name>
    <age>22</age>
    <city>Malvan</city></person>
  <person>
    <name>Divya Ghadi</name>
    <age>26</age>
    <city>Kankavli</city></person></root>
```

Output:



```
name: Samruddhi Modak
age: 20
city: Kudal


name: Nidhi Ghurye
age: 22
city: Malvan


name: Divya Ghadi
age: 26
city: Kankavli
```

c) JSON to HORUS format

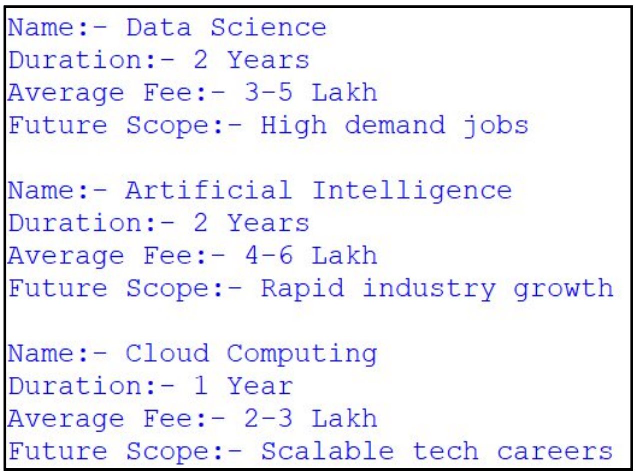
Code:

```
import json
with open("course.json", "r") as file:
    data = json.load(file)
for course in data["courses"]:
    print(f"Name:- {course['course_name']}")
    print(f"Duration:- {course['duration']}")
    print(f"Average Fee:- {course['average_fee']}")
    print(f"Future Scope:- {course['future_scope']}\n")
```

JSON File:

```
{
  "courses": [
    {
      "course_name": "Data Science",
      "duration": "2 Years",
      "average_fee": "3-5 Lakh",
      "future_scope": "High demand jobs"
    },
    {
      "course_name": "Artificial Intelligence",
      "duration": "2 Years",
      "average_fee": "4-6 Lakh",
      "future_scope": "Rapid industry growth"
    },
    {
      "course_name": "Cloud Computing",
      "duration": "1 Year",
      "average_fee": "2-3 Lakh",
      "future_scope": "Scalable tech careers"
    }]}
```

Output:



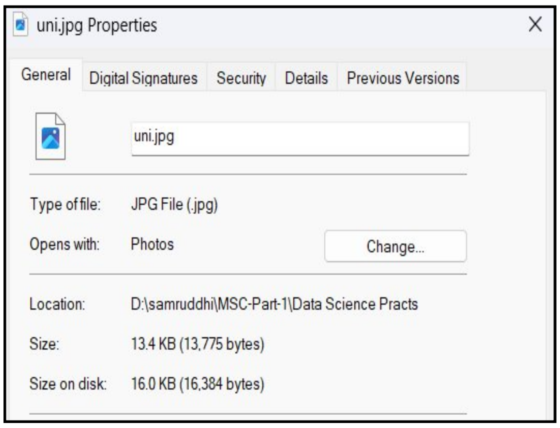
```
Name:- Data Science
Duration:- 2 Years
Average Fee:- 3-5 Lakh
Future Scope:- High demand jobs

Name:- Artificial Intelligence
Duration:- 2 Years
Average Fee:- 4-6 Lakh
Future Scope:- Rapid industry growth

Name:- Cloud Computing
Duration:- 1 Year
Average Fee:- 2-3 Lakh
Future Scope:- Scalable tech careers
```

d) Picture(JPEG) to HORUS format

Code:

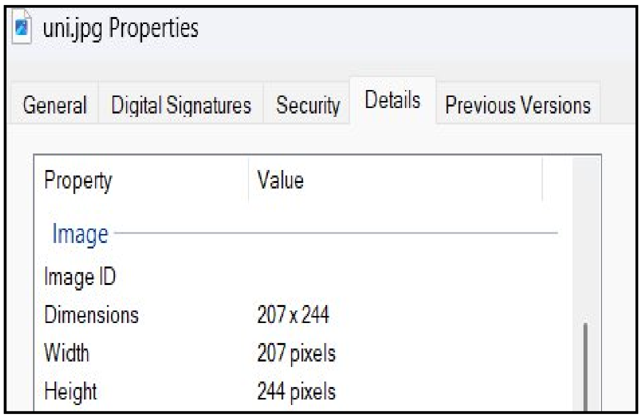
```
from PIL import Image
def image_info_to_horus(filename):
    try:
        with Image.open(filename) as img:
            print(f"Filename: {filename}")
            print(f"Format: {img.format}")
            print(f"Mode: {img.mode}")
            print(f"Size: {img.size[0]} x {img.size[1]} pixels")
    except FileNotFoundError:
        print("Error: File not found.")
    except Exception as e:
        print("An error occurred:", str(e))
image_info_to_horus("uni.jpg")
```

Image Properties:



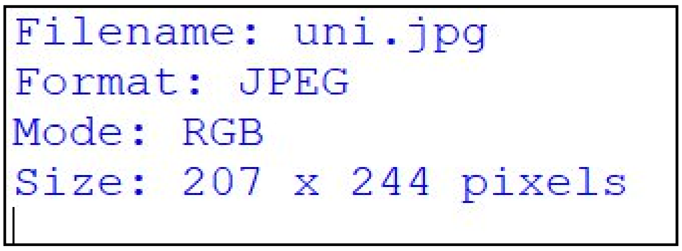
uni.jpg

Type of file:	JPG File (.jpg)
Opens with:	Photos
Location:	D:\samruddhi\MSC-Part-1\Data Science Practs
Size:	13.4 KB (13,775 bytes)
Size on disk:	16.0 KB (16,384 bytes)



Property	Value
Image	
Image ID	
Dimensions	207 x 244
Width	207 pixels
Height	244 pixels

Output:



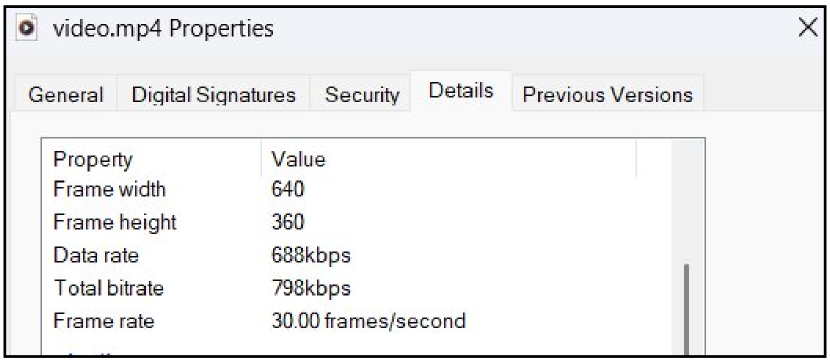
```
Filename: uni.jpg
Format: JPEG
Mode: RGB
Size: 207 x 244 pixels
```

e) Video to HORUS format

Code:

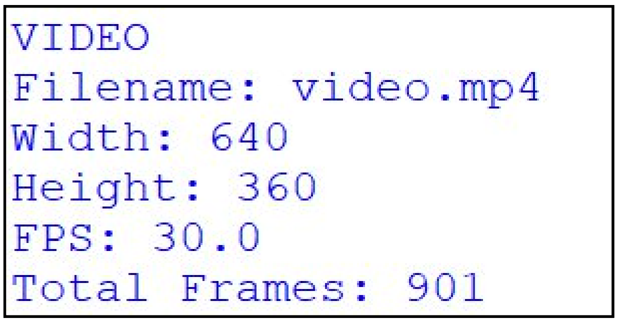
```
import cv2
file = "video.mp4"
cap = cv2.VideoCapture(file)
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = round(cap.get(cv2.CAP_PROP_FPS), 2)
frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
duration = round(frames / fps, 2)
cap.release()
print("VIDEO")
print(f"Filename: {file}")
print(f"Width: {width}")
print(f"Height: {height}")
print(f"FPS: {fps}")
print(f"Total Frames: {frames}")
print(f"Duration(sec): {duration}")
```

Video Properties:



Property	Value
Frame width	640
Frame height	360
Data rate	688kbps
Total bitrate	798kbps
Frame rate	30.00 frames/second

Output:



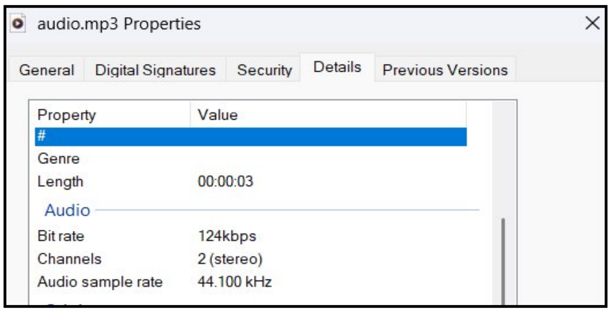
```
VIDEO
Filename: video.mp4
Width: 640
Height: 360
FPS: 30.0
Total Frames: 901
```

f) Audio to HORUS format

Code:

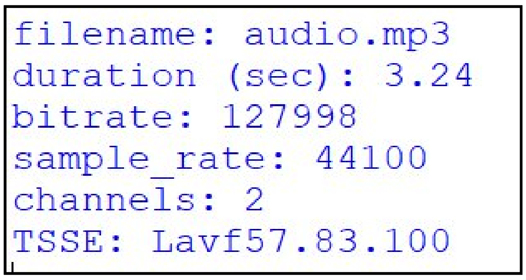
```
from mutagen import File
import os
def audio_to_horus(filename):
    try:
        audio = File(filename)
        if not audio:
            print("Unsupported or invalid audio format.")
            return
        print(f"filename: {os.path.basename(filename)}")
        if audio.info:
            if hasattr(audio.info, 'length'):
                print(f"duration (sec): {round(audio.info.length, 2)}")
            if hasattr(audio.info, 'bitrate'):
                print(f"bitrate: {audio.info.bitrate}")
            if hasattr(audio.info, 'sample_rate'):
                print(f"sample_rate: {audio.info.sample_rate}")
            if hasattr(audio.info, 'channels'):
                print(f"channels: {audio.info.channels}")
        if audio.tags:
            for tag, value in audio.tags.items():
                print(f"{tag}: {value}")
    except Exception as e:
        print("An error occurred:", str(e))
audio_to_horus("audio.mp3")
```

Audio Properties:



Property	Value
#	
Genre	
Length	00:00:03
Audio	
Bit rate	124kbps
Channels	2 (stereo)
Audio sample rate	44.100 kHz

Output:



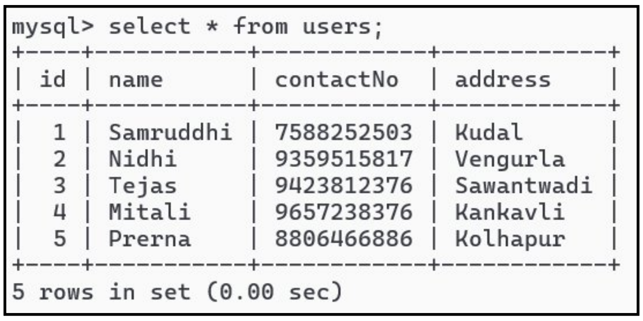
```
filename: audio.mp3
duration (sec): 3.24
bitrate: 127998
sample_rate: 44100
channels: 2
TSSE: Lavf57.83.100
```

g) MySql database to HORUS format

Code:

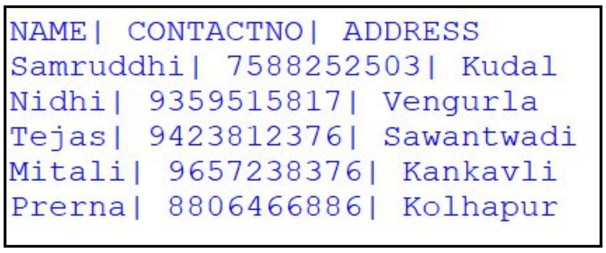
```
import mysql.connector
db_config = {
    "host": "localhost",
    "user": "root",
    "password": "123456",
    "database": "mscit"}
conn = mysql.connector.connect(**db_config)
cursor = conn.cursor()
query = "SELECT name, contactNo, address FROM users"
cursor.execute(query)
rows = cursor.fetchall()
print("NAME| CONTACTNO| ADDRESS")
for row in rows:
    name, contact, address = row
    print(f"{name}| {contact}| {address}")
cursor.close()
conn.close()
```

MySql Database/Table Records:



```
mysql> select * from users;
+----+-----------+------------+------------+
| id | name      | contactNo  | address    |
+----+-----------+------------+------------+
|  1 | Samruddhi | 7588252503 | Kudal      |
|  2 | Nidhi     | 9359515817 | Vengurla   |
|  3 | Tejas     | 9423812376 | Sawantwadi |
|  4 | Mitali    | 9657238376 | Kankavli   |
|  5 | Prerna    | 8806466886 | Kolhapur   |
+----+-----------+------------+------------+
5 rows in set (0.00 sec)
```

Output:



```
NAME| CONTACTNO| ADDRESS
Samruddhi| 7588252503| Kudal
Nidhi| 9359515817| Vengurla
Tejas| 9423812376| Sawantwadi
Mitali| 9657238376| Kankavli
Prerna| 8806466886| Kolhapur
```

K. M. S. P Mandal's	**Date: 12/09/2025**	
Sant Rawool Maharaj Mahavidyalaya, Kudal	**Roll No: 20**	
Department of Information Technology	**Expt. No: 03**	**Signature**

Title: Program to find Fixer Utilities, Averaging, Logging, Data Binning / Bucketing, Outlier Detection

Fixer Utilities:

In Data Science, fixer utilities are tools or functions used to clean, correct, or standardize data before analysis. They help handle messy or inconsistent datasets so that models and analysis give accurate results.

Averaging:

Averaging is a simple fixer utility used to replace missing or incorrect numeric values with a representative number, usually the mean (average) of that column. This helps maintain the dataset's consistency without losing too much information.

Logging:

Logging is a utility used to record events, operations, or errors while processing data. It helps you track what happens in your program, debug issues, and understand data processing steps.

Data Binning / Bucketing:

Data binning (or bucketing) is a technique to group continuous numeric data into discrete intervals or "bins." It's useful for simplifying data, reducing noise, and making patterns easier to analyse.

Outlier Detection:

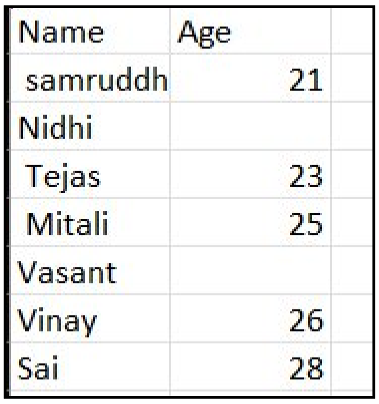
Outlier Detection is the process of identifying data points that are significantly different from most of the data. Outliers can affect analysis and model performance, so detecting (and sometimes handling) them is important.

a) Fixers Utilities

Code:

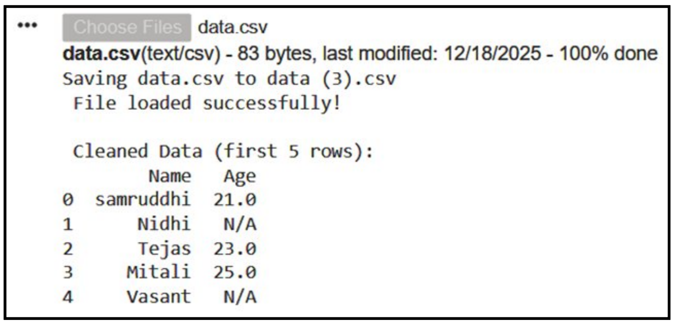
```
import pandas as pd
from google.colab import files

uploaded = files.upload()
file_name = list(uploaded.keys())[0]
df = pd.read_csv(file_name)
print(" File loaded successfully!")
df = df.apply(lambda x: x.str.strip() if x.dtype == "object" else x)
df = df.fillna('N/A')

print("\n Cleaned Data (first 5 rows):")
print(df.head())
```

CSV File(data.csv):



Name	Age
samruddh	21
Nidhi	
Tejas	23
Mitali	25
Vasant	
Vinay	26
Sai	28

Output:



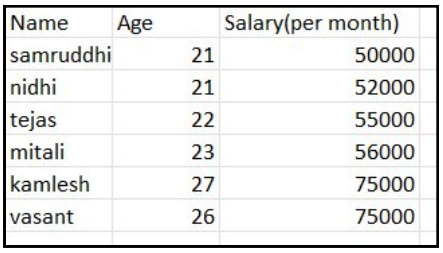
```
data.csv
data.csv(text/csv) - 83 bytes, last modified: 12/18/2025 - 100% done
Saving data.csv to data (3).csv
 File loaded successfully!

 Cleaned Data (first 5 rows):
        Name   Age
0  samruddhi  21.0
1      Nidhi   N/A
2      Tejas  23.0
3     Mitali  25.0
4     Vasant   N/A
```

b) Averaging

Code:

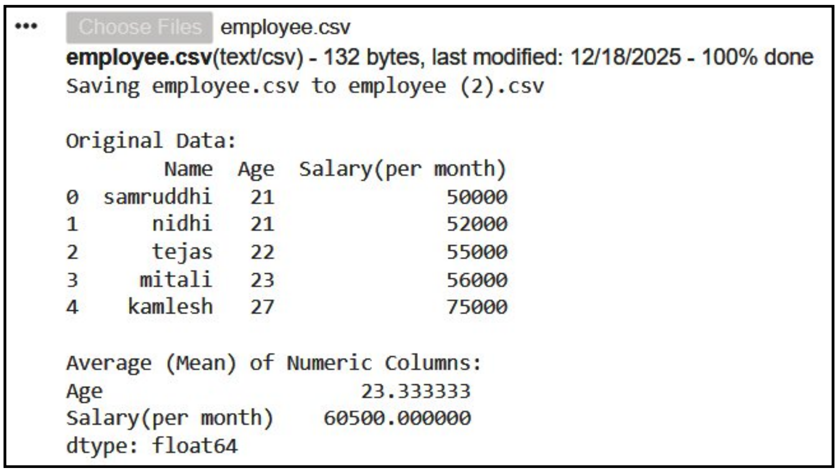
```
import pandas as pd
from google.colab import files
uploaded = files.upload()   # File chooser will open
file_name = list(uploaded.keys())[0]
df = pd.read_csv(file_name)
print("\nOriginal Data:")
print(df.head())
numeric_cols = df.select_dtypes(include='number').columns

if len(numeric_cols) > 0:
    print("\nAverage (Mean) of Numeric Columns:")
    averages = df[numeric_cols].mean()
    print(averages)
else:
    print("\nNo numeric columns found for averaging.")
```

CSV File(employee.csv):



Name	Age	Salary(per month)
samruddhi	21	50000
nidhi	21	52000
tejas	22	55000
mitali	23	56000
kamlesh	27	75000
vasant	26	75000

Output:



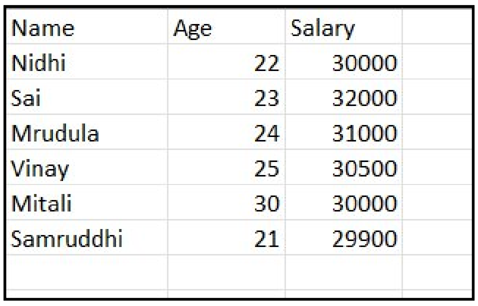
```
employee.csv
employee.csv(text/csv) - 132 bytes, last modified: 12/18/2025 - 100% done
Saving employee.csv to employee (2).csv

Original Data:
        Name  Age  Salary(per month)
0  samruddhi   21              50000
1      nidhi   21              52000
2      tejas   22              55000
3     mitali   23              56000
4    kamlesh   27              75000

Average (Mean) of Numeric Columns:
Age                     23.333333
Salary(per month)    60500.000000
dtype: float64
```

c) Outlier Detection

Code:

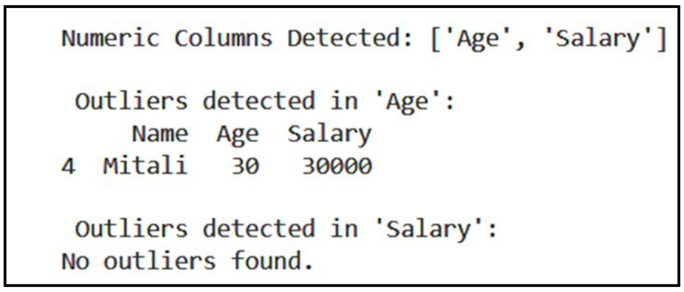
```
import pandas as pd
from google.colab import files

print("Please upload your CSV file (example: data.csv)")
uploaded = files.upload()
file_name = next(iter(uploaded))
df = pd.read_csv(file_name)
numeric_cols = df.select_dtypes(include='number').columns
print("\nNumeric Columns Detected:", list(numeric_cols))
for col in numeric_cols:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR
    outliers = df[(df[col] < lower) | (df[col] > upper)]
    print(f"\n Outliers detected in '{col}':")
    if outliers.empty:
        print("No outliers found.")
    else:
        print(outliers)
```

CSV File:



Name	Age	Salary
Nidhi	22	30000
Sai	23	32000
Mrudula	24	31000
Vinay	25	30500
Mitali	30	30000
Samruddhi	21	29900

Output:



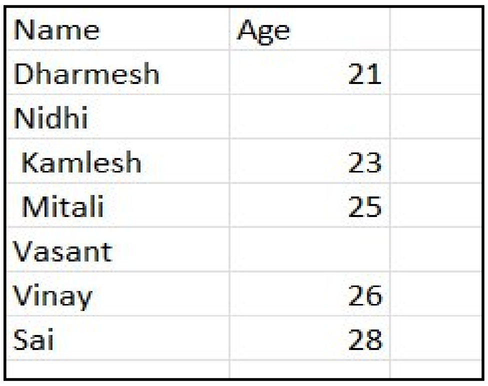
```
Numeric Columns Detected: ['Age', 'Salary']

 Outliers detected in 'Age':
     Name  Age  Salary
4  Mitali   30   30000

 Outliers detected in 'Salary':
No outliers found.
```

d)Logging

Code:

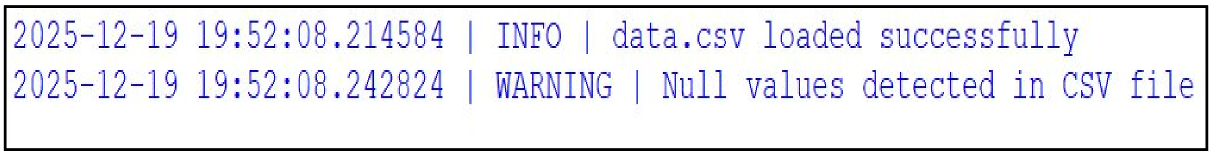
```
import pandas as pd
from datetime import datetime

csv_file = "data.csv"   # your input CSV file
def log(level, message):
    print(f"{datetime.now()} | {level} | {message}")

try:
    df = pd.read_csv(csv_file)
    log("INFO", f"{csv_file} loaded successfully")
    if df.isnull().values.any():
        log("WARNING", "Null values detected in CSV file")
    else:
        log("INFO", "No null values found")
except FileNotFoundError:
    log("ERROR", f"{csv_file} not found")
except Exception as e:
    log("ERROR", str(e))
```

"data.csv" file (With NULL values):



Name	Age
Dharmesh	21
Nidhi	
Kamlesh	23
Mitali	25
Vasant	
Vinay	26
Sai	28

Output:



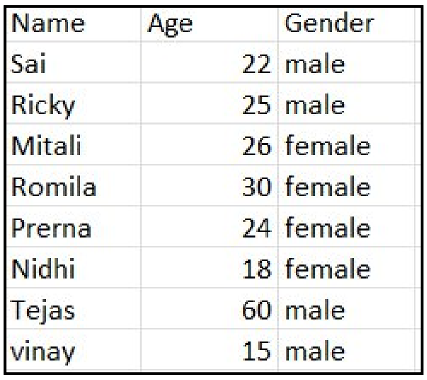
```
2025-12-19 19:52:08.214584 | INFO | data.csv loaded successfully
2025-12-19 19:52:08.242824 | WARNING | Null values detected in CSV file
```

e) Data Binning or Bucketing

Code:

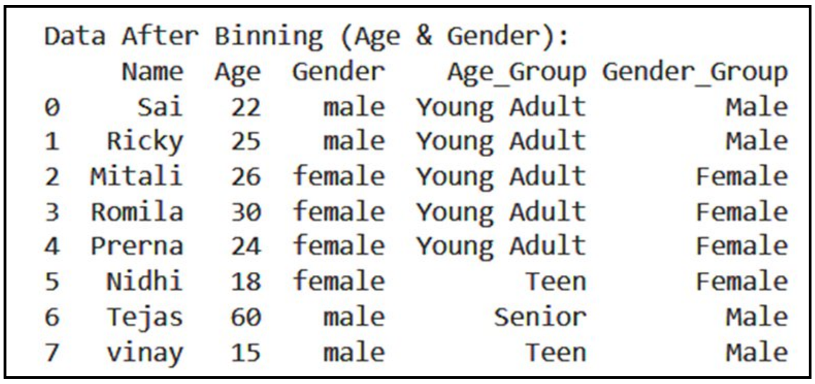
```
import pandas as pd
from google.colab import files
uploaded = files.upload()   # File chooser opens
file_name = list(uploaded.keys())[0]
df = pd.read_csv(file_name)
print("\nOriginal Data:")
print(df)
if 'Age' in df.columns:
    age_bins = [0, 20, 30, 50, 100]
    age_labels = ['Teen', 'Young Adult', 'Adult', 'Senior']
    df['Age_Group'] = pd.cut(df['Age'], bins=age_bins, labels=age_labels)
else:
    print("Error: 'Age' column not found")
if 'Gender' in df.columns:
    df['Gender_Group'] = df['Gender'].str.lower().map({
        'male': 'Male',
        'female': 'Female'}).fillna('Other')
else:
    print("Error: 'Gender' column not found")
print("\nData After Binning (Age & Gender):")
print(df)
```

CSV File:



Name	Age	Gender
Sai	22	male
Ricky	25	male
Mitali	26	female
Romila	30	female
Prerna	24	female
Nidhi	18	female
Tejas	60	male
vinay	15	male

Output:



```
Data After Binning (Age & Gender):
     Name  Age  Gender    Age_Group Gender_Group
0     Sai   22    male  Young Adult         Male
1   Ricky   25    male  Young Adult         Male
2  Mitali   26  female  Young Adult       Female
3  Romila   30  female  Young Adult       Female
4  Prerna   24  female  Young Adult       Female
5   Nidhi   18  female         Teen       Female
6   Tejas   60    male       Senior         Male
7   vinay   15    male         Teen         Male
```

K. M. S. P Mandal's **Sant Rawool Maharaj Mahavidyalaya, Kudal** **Department of Information Technology**	**Date: 19/09/2025** **Roll No: 20** **Expt. No: 04**	**Signature**

Title: Performing data processing using R, retrieving different attributes of data, Data Pattern

Data Processing:
Data processing in data science means preparing raw data so it can be used for analysis. It includes collecting data, cleaning errors, handling missing values, and removing duplicates. The data is then organized, transformed, or normalized into a usable format. Proper data processing helps improve accuracy and reliability of results. Without good data processing, even advanced models can give wrong outcomes.

Data Cleaning:

Data cleaning means fixing or removing incorrect, incomplete, or duplicate data. It includes handling missing values, correcting errors, and removing noise. Clean data improves accuracy and reliability of analysis.

Data Transformation:

Data transformation means converting data into a suitable format for analysis. It includes normalization, scaling, encoding categorical data, and aggregation. Transformed data helps models understand patterns more effectively.

Data Pattern:

Data pattern refers to meaningful trends, relationships, or structures found in data. It shows how values change, repeat, or relate to each other over time or categories. Examples include trends, clusters, correlations, and anomalies. Identifying data patterns helps in prediction, decision-making, and understanding behavior in data science.

Using R programming:

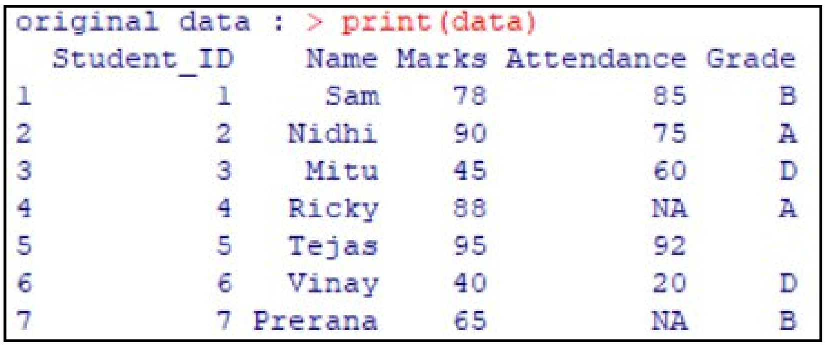
R provides powerful tools for data cleaning, transformation, and analysis in a simple way. It has many built-in functions and packages. R makes it easy to visualize data patterns using graphs and plots. It handles large datasets efficiently and supports statistical analysis. R is open-source and widely used in research and academics.

Data Processing:

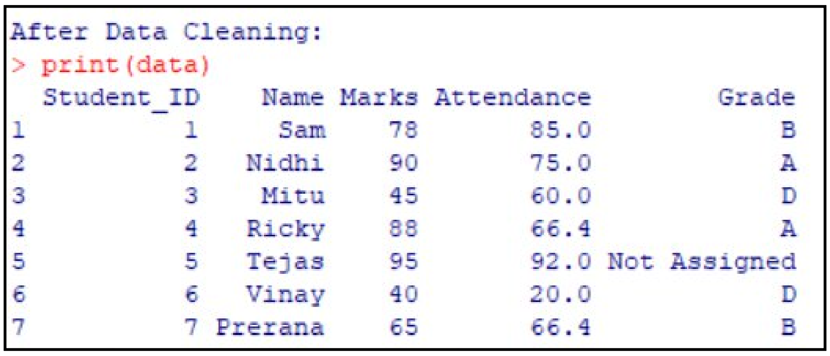
Code:

```
data <- read.csv(
  "D:\\samruddhi\\MSC-Part-1\\Data Science Practs\\nidhi.csv",
  stringsAsFactors = FALSE)
cat("original data : ")
print(data)

# Replace missing Marks with mean
data$Marks[is.na(data$Marks)] <- mean(data$Marks, na.rm = TRUE)
# Replace missing Attendance with mean
data$Attendance[is.na(data$Attendance)] <- mean(data$Attendance, na.rm = TRUE)
# Replace empty Name values
data$Name[data$Name == ""] <- "Unknown"
# Replace empty Grade values
data$Grade[data$Grade == ""] <- "Not Assigned"
cat("\nAfter Data Cleaning:\n")
print(data)

data$Result <- ifelse(data$Marks >= 50, "Pass", "Fail")
# Normalize Marks
data$Normalized_Marks <- round(data$Marks / max(data$Marks), 2)
cat("\nAfter Data Transformation:\n")
print(data)
# Sort data by Marks (Descending)
data <- data[order(-data$Marks), ]

# Select required columns
final_data <- data[, c("Student_ID", "Name", "Marks", "Attendance", "Result")]
cat("\nFinal Prepared Dataset:\n")
print(final_data)
```

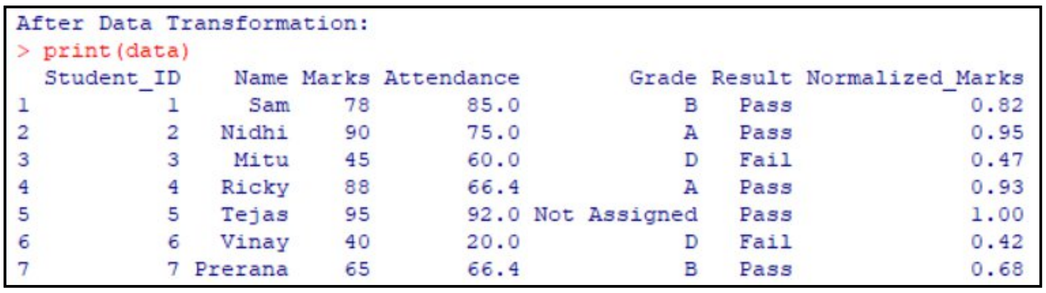
Output:



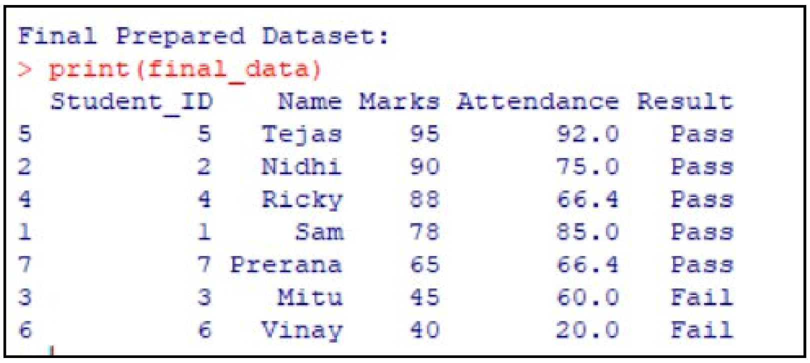
```
original data : > print(data)
  Student_ID    Name Marks Attendance Grade
1          1     Sam    78         85     B
2          2   Nidhi    90         75     A
3          3    Mitu    45         60     D
4          4   Ricky    88         NA     A
5          5   Tejas    95         92      
6          6   Vinay    40         20     D
7          7 Prerana    65         NA     B
```



```
After Data Cleaning:
> print(data)
  Student_ID    Name Marks Attendance        Grade
1          1     Sam    78       85.0            B
2          2   Nidhi    90       75.0            A
3          3    Mitu    45       60.0            D
4          4   Ricky    88       66.4            A
5          5   Tejas    95       92.0 Not Assigned
6          6   Vinay    40       20.0            D
7          7 Prerana    65       66.4            B
```



```
After Data Transformation:
> print(data)
  Student_ID    Name Marks Attendance        Grade Result Normalized_Marks
1          1     Sam    78       85.0            B   Pass             0.82
2          2   Nidhi    90       75.0            A   Pass             0.95
3          3    Mitu    45       60.0            D   Fail             0.47
4          4   Ricky    88       66.4            A   Pass             0.93
5          5   Tejas    95       92.0 Not Assigned   Pass             1.00
6          6   Vinay    40       20.0            D   Fail             0.42
7          7 Prerana    65       66.4            B   Pass             0.68
```



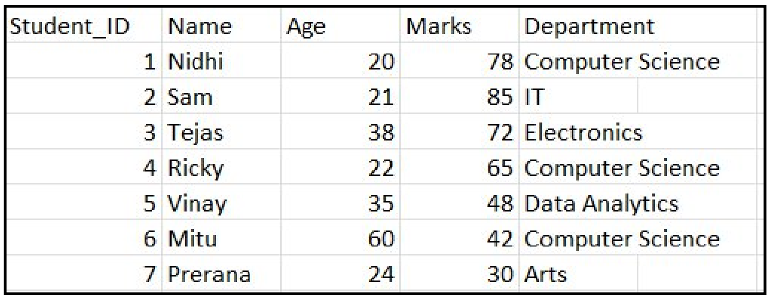
```
Final Prepared Dataset:
> print(final_data)
  Student_ID    Name Marks Attendance Result
5          5   Tejas    95       92.0   Pass
2          2   Nidhi    90       75.0   Pass
4          4   Ricky    88       66.4   Pass
1          1     Sam    78       85.0   Pass
7          7 Prerana    65       66.4   Pass
3          3    Mitu    45       60.0   Fail
6          6   Vinay    40       20.0   Fail
```

Retrieving different attributes of data:

Code:

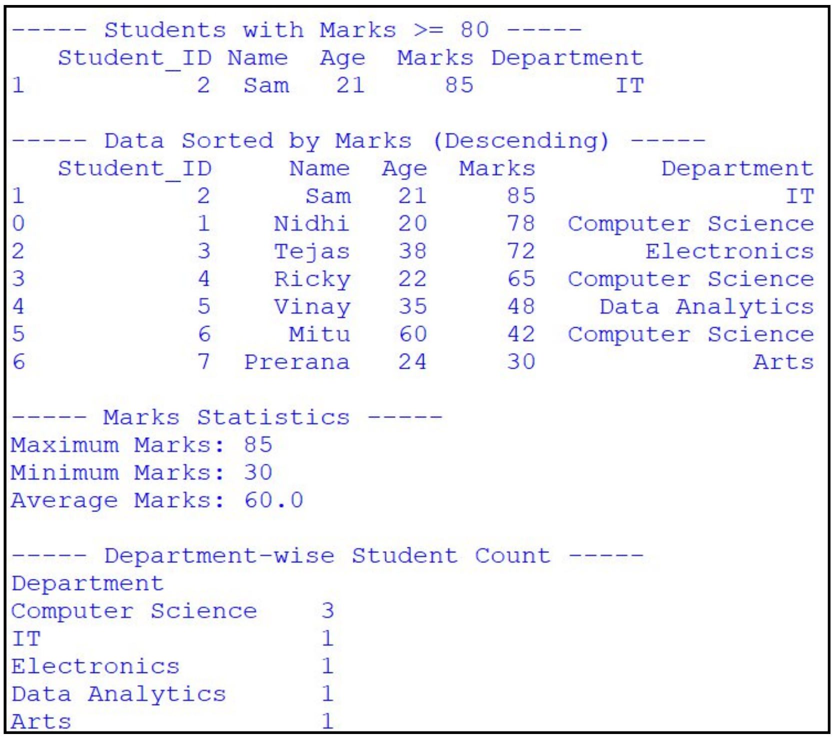
```
import pandas as pd
data = pd.read_csv("stud(4B).csv")
print("\n----- Students with Marks >= 80 -----")
print(data[data["Marks"] >= 80])
print("\n----- Data Sorted by Marks (Descending) -----")
print(data.sort_values(by="Marks", ascending=False))
print("\n----- Marks Statistics -----")
print("Maximum Marks:", data["Marks"].max())
print("Minimum Marks:", data["Marks"].min())
print("Average Marks:", data["Marks"].mean())
print("\n----- Department-wise Student Count -----")
print(data["Department"].value_counts())
```

Data in CSV File(stud(4B).csv):



Student_ID	Name	Age	Marks	Department
1	Nidhi	20	78	Computer Science
2	Sam	21	85	IT
3	Tejas	38	72	Electronics
4	Ricky	22	65	Computer Science
5	Vinay	35	48	Data Analytics
6	Mitu	60	42	Computer Science
7	Prerana	24	30	Arts

Output:



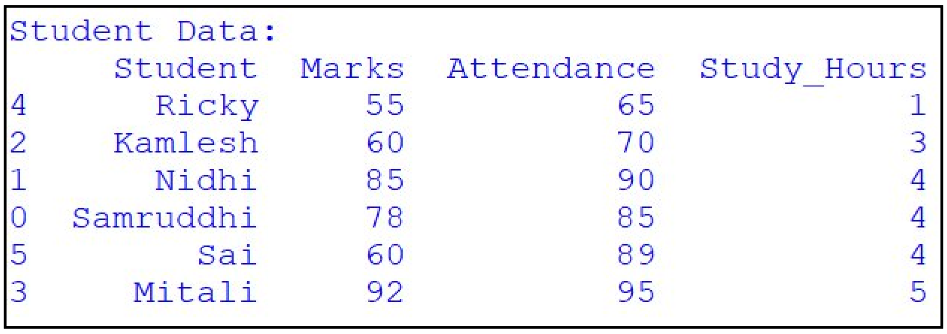
```
----- Students with Marks >= 80 -----
   Student_ID Name  Age  Marks Department
1           2  Sam   21     85         IT

----- Data Sorted by Marks (Descending) -----
   Student_ID     Name  Age  Marks        Department
1           2      Sam   21     85                IT
0           1    Nidhi   20     78  Computer Science
2           3    Tejas   38     72       Electronics
3           4    Ricky   22     65  Computer Science
4           5    Vinay   35     48    Data Analytics
5           6     Mitu   60     42  Computer Science
6           7  Prerana   24     30              Arts

----- Marks Statistics -----
Maximum Marks: 85
Minimum Marks: 30
Average Marks: 60.0

----- Department-wise Student Count -----
Department
Computer Science    3
IT                  1
Electronics         1
Data Analytics      1
Arts                1
```

Data Pattern:

Code:

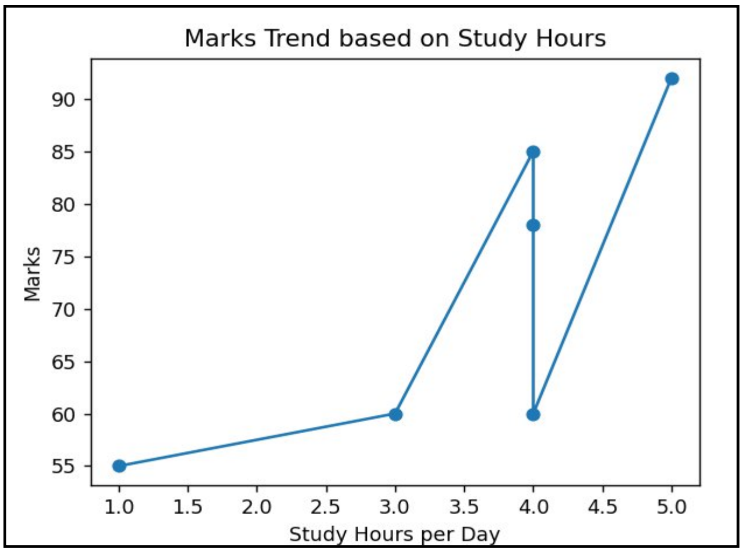
```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv("students_data-2.csv")
df = df.sort_values("Study_Hours")
print("Student Data:")
print(df)

plt.plot(df["Study_Hours"], df["Marks"], marker='o')
plt.xlabel("Study Hours per Day")
plt.ylabel("Marks")
plt.title("Marks Trend based on Study Hours")
plt.show()
df = df.sort_values("Attendance")

plt.plot(df["Attendance"], df["Marks"], marker='o')
plt.xlabel("Attendance Percentage")
plt.ylabel("Marks")
plt.title("Marks Trend based on Attendance")
plt.show()
study_corr = df["Study_Hours"].corr(df["Marks"])
attendance_corr = df["Attendance"].corr(df["Marks"])
print("Correlation between Study Hours and Marks:", study_corr)
print("Correlation between Attendance and Marks:", attendance_corr)
```

Output:

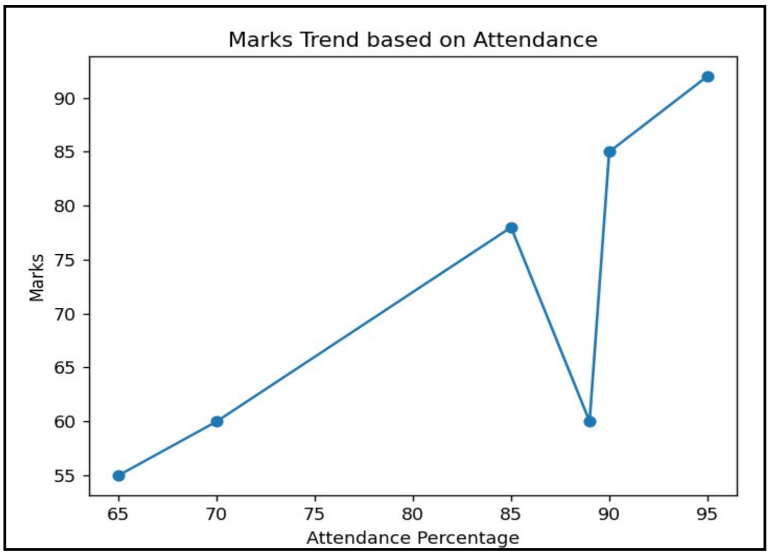


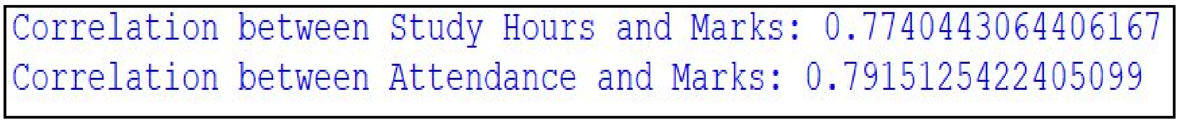
```
Student Data:
     Student  Marks  Attendance  Study_Hours
4      Ricky     55          65            1
2    Kamlesh     60          70            3
1      Nidhi     85          90            4
0  Samruddhi     78          85            4
5        Sai     60          89            4
3     Mitali     92          95            5
```

Data pattern between Marks and Study Hours:



Data pattern between Marks and Attendance:





```
Correlation between Study Hours and Marks: 0.7740443064406167
Correlation between Attendance and Marks: 0.7915125422405099
```

K. M. S. P Mandal's	**Date: 17/10/2025**	
Sant Rawool Maharaj Mahavidyalaya, Kudal	**Roll No: 20**	
Department of Information Technology	**Expt. No: 05**	**Signature**

Title: Implementation of Error Management, Network Routing, Generating GML Files, Warehouses location, Simple Forex Trading Planner, Generating Payroll, Balance Sheet

Error Management:
Error management in data science means finding and reducing mistakes in data and models. Errors can occur during data collection, cleaning, or model training. Handling missing values, removing wrong data, and choosing the right model help reduce errors. Proper evaluation and checking results improve accuracy. Good error management makes data analysis more reliable.

pandas library:
Error management in data science using Pandas means handling wrong or missing data while working with datasets.Pandas helps to detect missing values, duplicates, and incorrect data types. Errors can be reduced by cleaning data using functions like dropna(), fillna(), and astype().

networkx library:
NetworkX is a Python library used to create, analyze, and visualize networks as graphs. Nodes represent devices (like routers, PCs), and edges represent connections. It helps find shortest paths, routing paths, and analyze network structure. We can also visualize networks and apply graph algorithms easily.

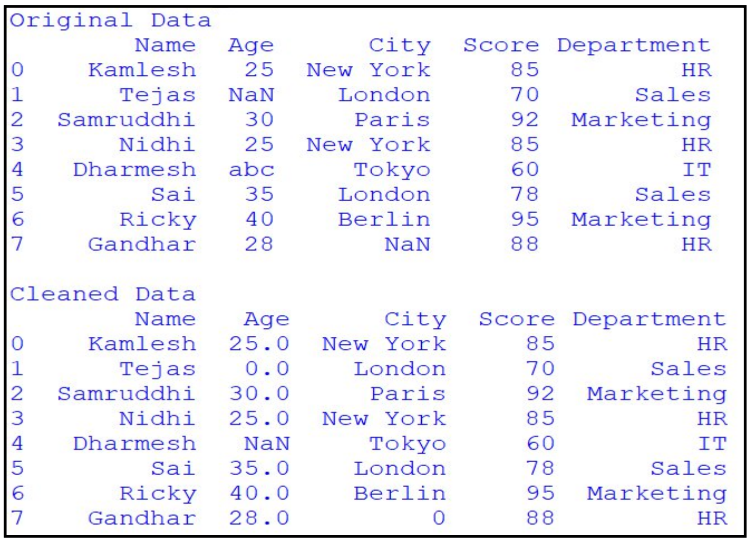
sklearn library:
Scikit-learn (sklearn) is a popular Python library used in data science and machine learning. It provides ready-made tools for clustering, classification, regression, and prediction. The library works well with pandas and NumPy data. It makes complex algorithms like K-Means easy to use with just a few lines of code.

a) Error Management:

Code:

```
import pandas as pd
df = pd.read_csv("data-error.csv")
print("Original Data")
print(df)
df.fillna(0, inplace=True)
df.drop_duplicates(inplace=True)
df["Age"] = pd.to_numeric(df["Age"], errors="coerce")
print("\nCleaned Data")
print(df)
```

Output:



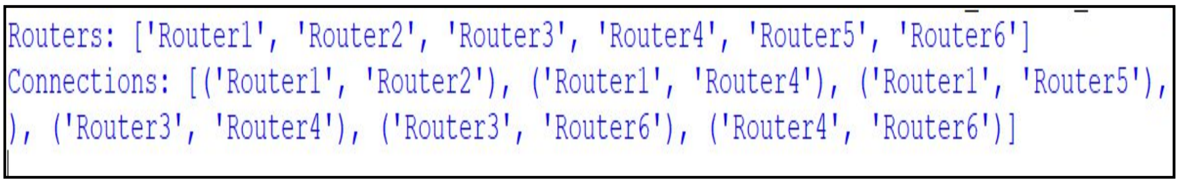
```
Original Data
        Name  Age      City  Score Department
0    Kamlesh   25  New York     85         HR
1      Tejas  NaN    London     70      Sales
2  Samruddhi   30     Paris     92  Marketing
3      Nidhi   25  New York     85         HR
4   Dharmesh  abc     Tokyo     60         IT
5        Sai   35    London     78      Sales
6      Ricky   40    Berlin     95  Marketing
7    Gandhar   28       NaN     88         HR

Cleaned Data
        Name   Age      City  Score Department
0    Kamlesh  25.0  New York     85         HR
1      Tejas   0.0    London     70      Sales
2  Samruddhi  30.0     Paris     92  Marketing
3      Nidhi  25.0  New York     85         HR
4   Dharmesh   NaN     Tokyo     60         IT
5        Sai  35.0    London     78      Sales
6      Ricky  40.0    Berlin     95  Marketing
7    Gandhar  28.0         0     88         HR
```

b) Creating Network Routing diagram:

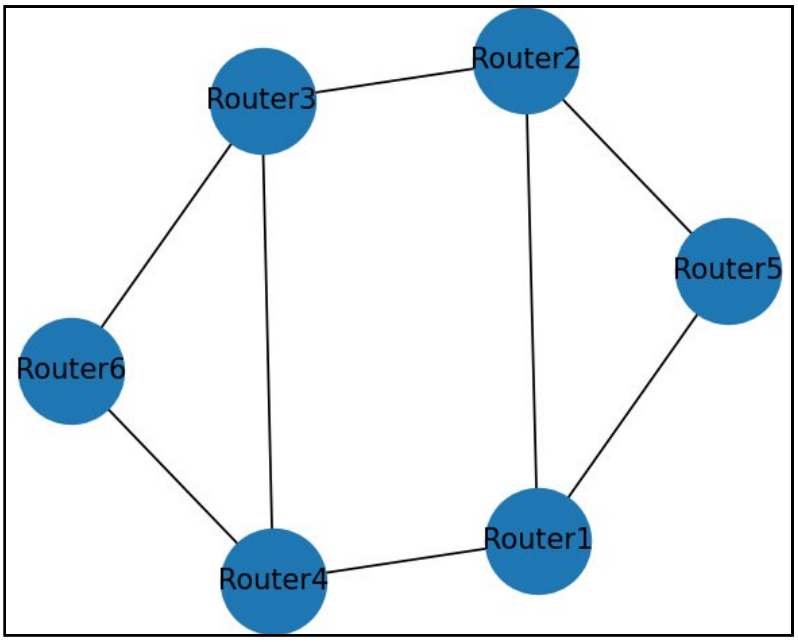
Code:

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

G = nx.Graph()
routers = [
   ("Router1", "Router2"),
   ("Router2", "Router3"),
   ("Router3", "Router4"),
   ("Router4", "Router1"),
   ("Router1", "Router5"),
   ("Router5", "Router2"),
   ("Router3", "Router6"),
   ("Router6", "Router4")]

G.add_edges_from(routers)
print("Routers:", G.nodes())
print("Connections:", G.edges())
nx.draw(G, with_labels=True, node_size=2000)
plt.show()
```

Output:



```
Routers: ['Router1', 'Router2', 'Router3', 'Router4', 'Router5', 'Router6']
Connections: [('Router1', 'Router2'), ('Router1', 'Router4'), ('Router1', 'Router5'),
), ('Router3', 'Router4'), ('Router3', 'Router6'), ('Router4', 'Router6')]
```

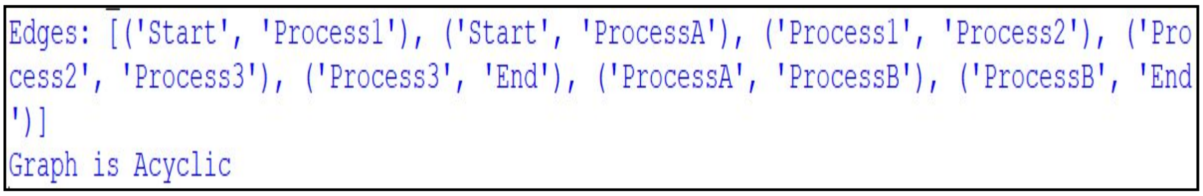


c) Acyclic Graph:

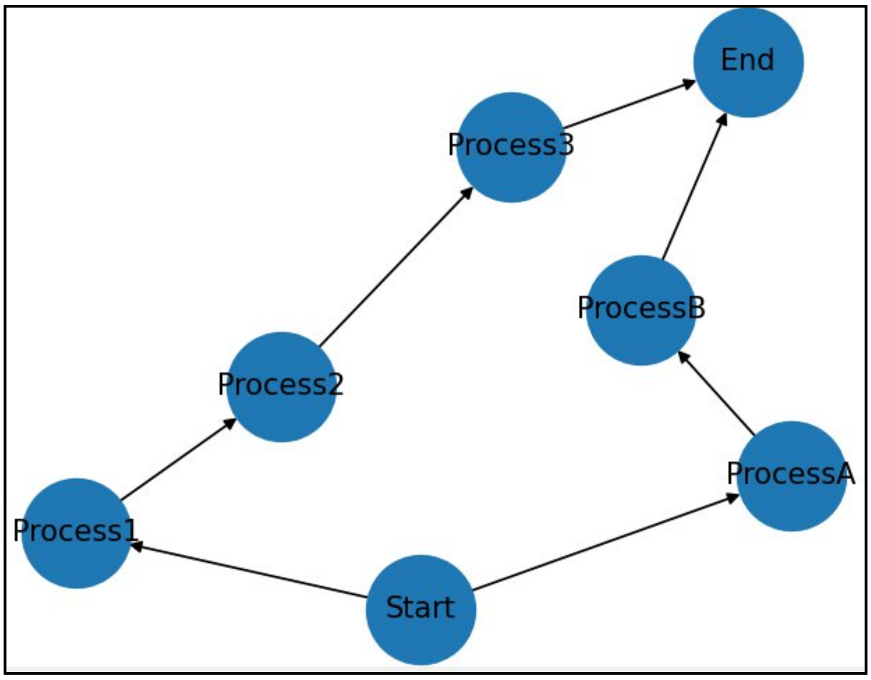
Code:

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

G = nx.DiGraph()
G.add_edge("Start", "Process1")
G.add_edge("Process1", "Process2")
G.add_edge("Process2", "Process3")
G.add_edge("Process3", "End")
G.add_edge("Start", "ProcessA")
G.add_edge("ProcessA", "ProcessB")
G.add_edge("ProcessB", "End")
print("Edges:", G.edges())
if nx.is_directed_acyclic_graph(G):
   print("Graph is Acyclic")
else:
   print("Graph has a Cycle")
nx.draw(G, with_labels=True, node_size=2000)
plt.show()
```

Output:



```
Edges: [('Start', 'Process1'), ('Start', 'ProcessA'), ('Process1', 'Process2'), ('Pro
cess2', 'Process3'), ('Process3', 'End'), ('ProcessA', 'ProcessB'), ('ProcessB', 'End
')]
Graph is Acyclic
```

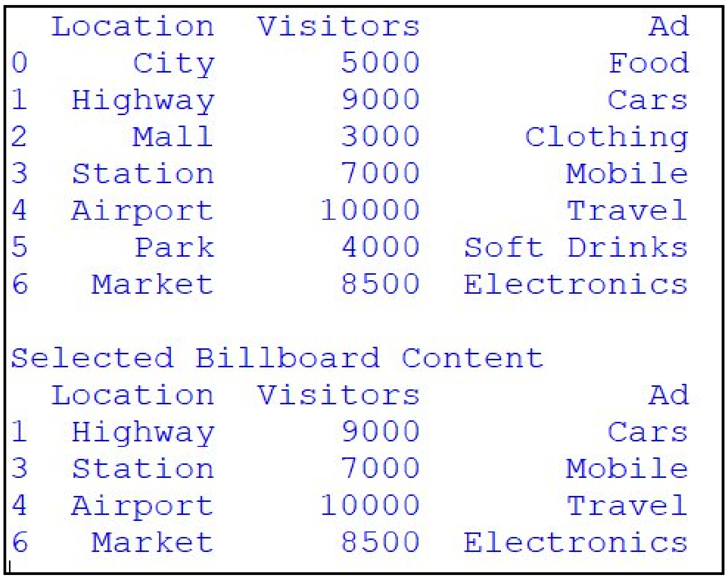


d) Picking the content for BillBoards from the given data

Code:

```
import pandas as pd

data = {
   "Location": ["City", "Highway", "Mall", "Station", "Airport", "Park", "Market"],
   "Visitors": [5000, 9000, 3000, 7000, 10000, 4000, 8500],
   "Ad": ["Food", "Cars", "Clothing", "Mobile", "Travel", "Soft Drinks", "Electronics"]
}

df = pd.DataFrame(data)
print(df)
selected = df[df["Visitors"] > 6000]
print("\nSelected Billboard Content")
print(selected)
```

Output:



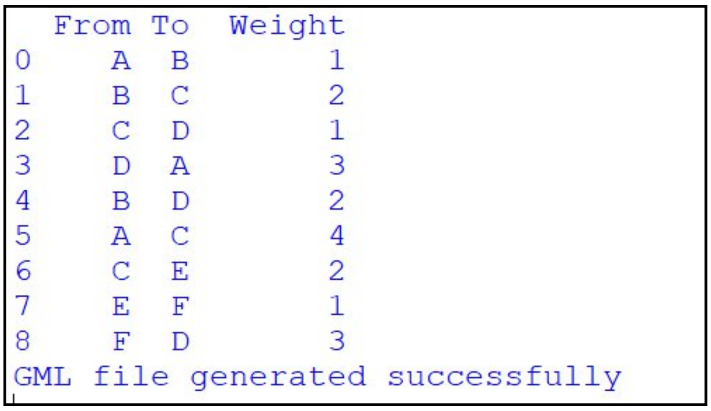
```
  Location  Visitors           Ad
0     City      5000         Food
1  Highway      9000         Cars
2     Mall      3000     Clothing
3  Station      7000       Mobile
4  Airport     10000       Travel
5     Park      4000  Soft Drinks
6   Market      8500  Electronics

Selected Billboard Content
  Location  Visitors           Ad
1  Highway      9000         Cars
3  Station      7000       Mobile
4  Airport     10000       Travel
6   Market      8500  Electronics
```

e) Generating GML file from given csv file:

Code:

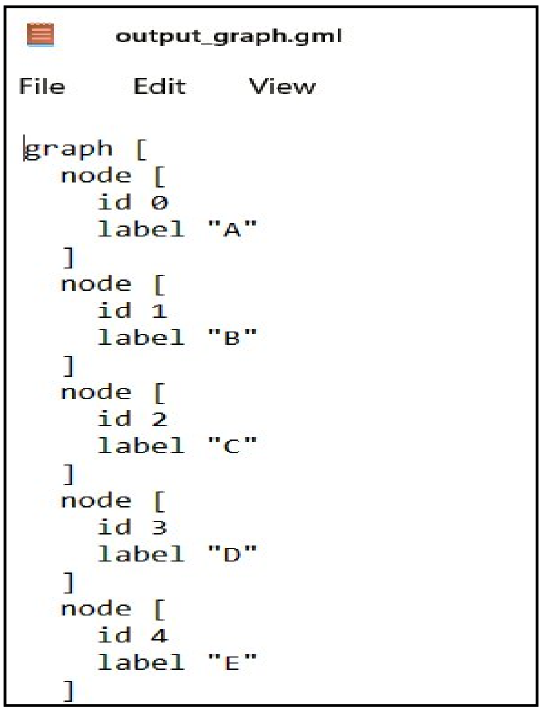
```
import pandas as pd
import networkx as nx

df = pd.read_csv("edges.csv")
print(df)

G = nx.from_pandas_edgelist(df, "From", "To")
nx.write_gml(G, "output_graph.gml")
print("GML file generated successfully")
```

Output:



```
  From To  Weight
0    A  B       1
1    B  C       2
2    C  D       1
3    D  A       3
4    B  D       2
5    A  C       4
6    C  E       2
7    E  F       1
8    F  D       3
GML file generated successfully
```

Output_graph.gml file:



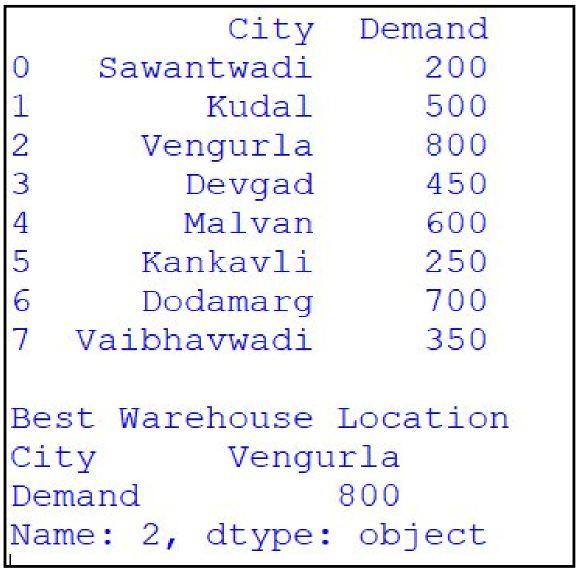
```
graph [
  node [
    id 0
    label "A"
  ]
  node [
    id 1
    label "B"
  ]
  node [
    id 2
    label "C"
  ]
  node [
    id 3
    label "D"
  ]
  node [
    id 4
    label "E"
  ]
```

f) Planning location of Warehouse:

Code:

```
import pandas as pd

data = {
    "City": ["Sawantwadi", "Kudal", "Vengurla", "Devgad", "Malvan", "Kankavli",
"Dodamarg", "Vaibhavwadi"],
    "Demand": [200, 500, 800, 450, 600, 250, 700, 350]
}
df = pd.DataFrame(data)
print(df)

best = df.loc[df["Demand"].idxmax()]
print("\nBest Warehouse Location")
print(best)
```

Output:



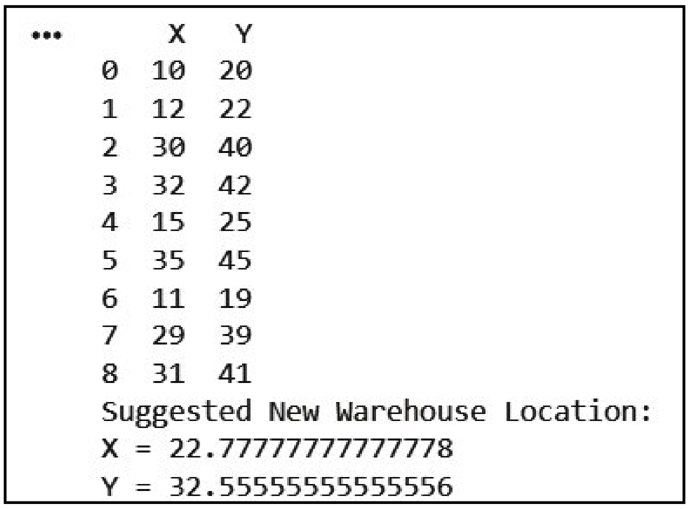
```
          City  Demand
0   Sawantwadi     200
1        Kudal     500
2     Vengurla     800
3       Devgad     450
4       Malvan     600
5     Kankavli     250
6     Dodamarg     700
7  Vaibhavwadi     350

Best Warehouse Location
City      Vengurla
Demand         800
Name: 2, dtype: object
```

g) Clustering to determine new warehouse using the given data

Code:

```
import pandas as pd
from sklearn.cluster import KMeans

data = {
  "X": [10, 12, 30, 32, 15, 35, 11, 29, 31],
  "Y": [20, 22, 40, 42, 25, 45, 19, 39, 41]
}

df = pd.DataFrame(data)
kmeans = KMeans(n_clusters=1, n_init=10)
kmeans.fit(df)
warehouse_location = kmeans.cluster_centers_

print(df)
print("Suggested New Warehouse Location:")
print("X =", warehouse_location[0][0])
print("Y =", warehouse_location[0][1])
```

Output:



```
•••      X   Y
     0  10  20
     1  12  22
     2  30  40
     3  32  42
     4  15  25
     5  35  45
     6  11  19
     7  29  39
     8  31  41
     Suggested New Warehouse Location:
     X = 22.77777777777778
     Y = 32.55555555555556
```

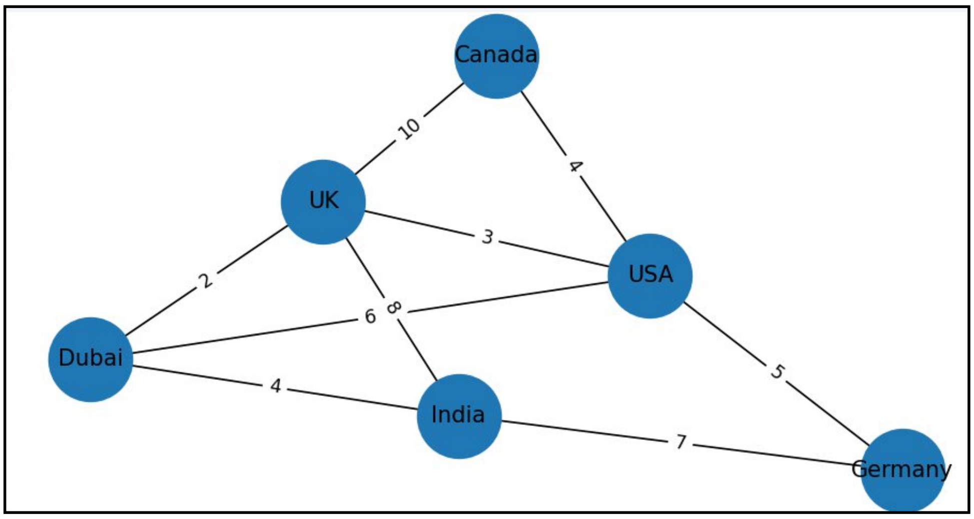
h) Planning the shipping routers from best-fit international logistics:

Code:

```
import networkx as nx
import matplotlib.pyplot as plt
G = nx.Graph()
G.add_weighted_edges_from([
    ("India", "Dubai", 4),
    ("Dubai", "USA", 6),
    ("India", "UK", 8),
    ("UK", "USA", 3),
    ("Dubai", "UK", 2),
    ("India", "Germany", 7),
    ("Germany", "USA", 5),("UK", "Canada", 10),("Canada", "USA", 4)])

# Find shortest path
route = nx.shortest_path(G, "India", "USA", weight="weight")
print("Best Shipping Route:", route)
# Position nodes
pos = nx.spring_layout(G)
nx.draw(G, pos, with_labels=True, node_size=2000)
edge_labels = nx.get_edge_attributes(G, "weight")
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels)
plt.show()
```

Output:

```
Best Shipping Route: ['India', 'Dubai', 'UK', 'USA']
```

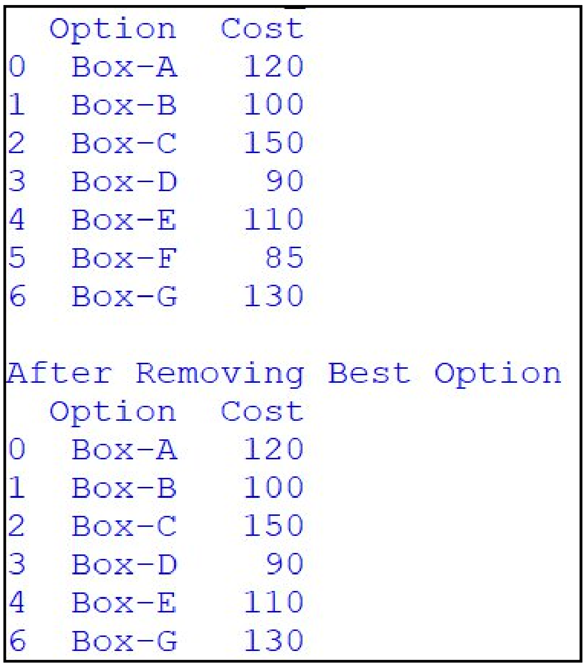


i) Program to delete the best packing option to ship in container from the given data:

Code:

```
import pandas as pd

data = {
   "Option": ["Box-A", "Box-B", "Box-C", "Box-D", "Box-E", "Box-F", "Box-G"],
   "Cost": [120, 100, 150, 90, 110, 85, 130]}
df = pd.DataFrame(data)
print(df)
best_index = df["Cost"].idxmin()
df.drop(best_index, inplace=True)

print("\nAfter Removing Best Option")
print(df)
```

Output:



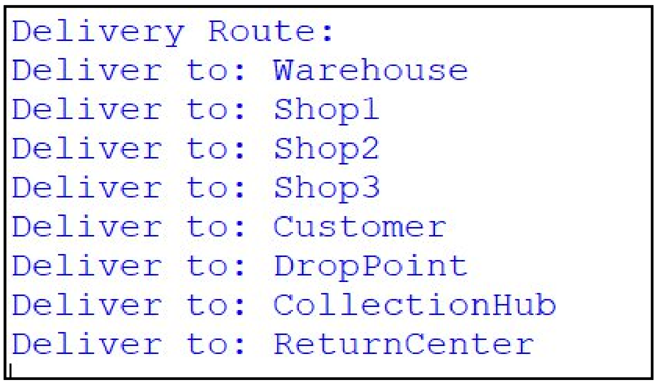
```
  Option  Cost
0  Box-A   120
1  Box-B   100
2  Box-C   150
3  Box-D    90
4  Box-E   110
5  Box-F    85
6  Box-G   130

After Removing Best Option
  Option  Cost
0  Box-A   120
1  Box-B   100
2  Box-C   150
3  Box-D    90
4  Box-E   110
6  Box-G   130
```

j) Program to create delivery route using the given data:

Code:

```
delivery_points = ["Warehouse", "Shop1", "Shop2", "Shop3", "Customer", "DropPoint",
"CollectionHub", "ReturnCenter"]

print("Delivery Route:")
for point in delivery_points:
    print("Deliver to:", point)
```

Output:



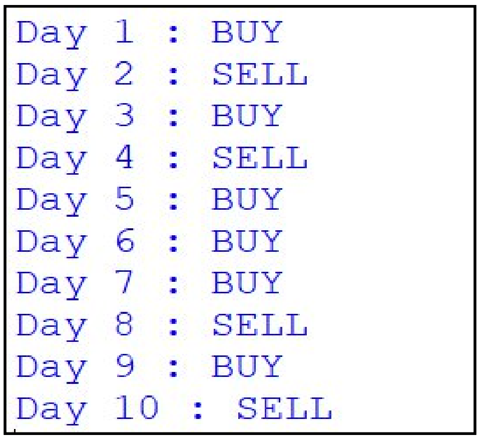
```
Delivery Route:
Deliver to: Warehouse
Deliver to: Shop1
Deliver to: Shop2
Deliver to: Shop3
Deliver to: Customer
Deliver to: DropPoint
Deliver to: CollectionHub
Deliver to: ReturnCenter
```

k) Program for forex planner:

Code:

```
prices = [74, 76, 75, 78, 77, 79, 80, 82, 81, 83, 80]
for i in range(1, len(prices)):
    if prices[i] > prices[i-1]:
        print("Day", i, ": BUY")
    else:
        print("Day", i, ": SELL")
```

Output:



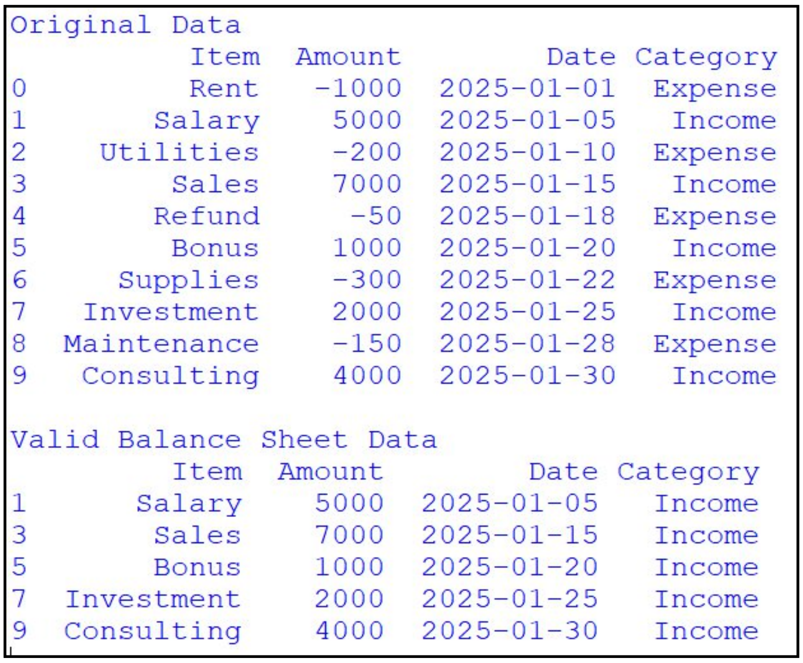
```
Day 1 : BUY
Day 2 : SELL
Day 3 : BUY
Day 4 : SELL
Day 5 : BUY
Day 6 : BUY
Day 7 : BUY
Day 8 : SELL
Day 9 : BUY
Day 10 : SELL
```

l) Program to process the balance sheet to ensure the only good data is processing:

Code:

```
import pandas as pd
df = pd.read_csv("balance.csv")
print("Original Data")
print(df)
valid = df[df["Amount"] > 0]
print("\nValid Balance Sheet Data")
print(valid)
```

Output:



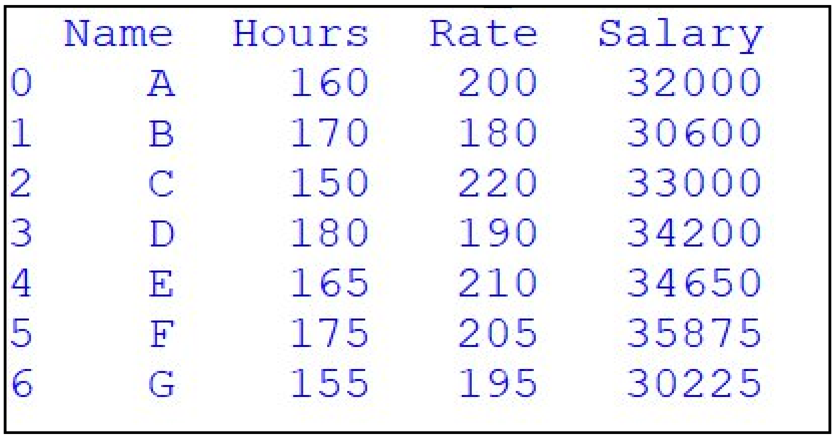
```
Original Data
          Item  Amount        Date Category
0         Rent   -1000  2025-01-01  Expense
1       Salary    5000  2025-01-05   Income
2    Utilities    -200  2025-01-10  Expense
3        Sales    7000  2025-01-15   Income
4       Refund     -50  2025-01-18  Expense
5        Bonus    1000  2025-01-20   Income
6     Supplies    -300  2025-01-22  Expense
7   Investment    2000  2025-01-25   Income
8  Maintenance    -150  2025-01-28  Expense
9   Consulting    4000  2025-01-30   Income

Valid Balance Sheet Data
         Item  Amount        Date Category
1      Salary    5000  2025-01-05   Income
3       Sales    7000  2025-01-15   Income
5       Bonus    1000  2025-01-20   Income
7  Investment    2000  2025-01-25   Income
9  Consulting    4000  2025-01-30   Income
```

m) Program to generate payroll from the given data:

Code:

```
import pandas as pd

data = {
    "Name": ["A", "B", "C", "D", "E", "F", "G"],
    "Hours": [160, 170, 150, 180, 165, 175, 155],
    "Rate": [200, 180, 220, 190, 210, 205, 195]
}
df = pd.DataFrame(data)
df["Salary"] = df["Hours"] * df["Rate"]

print(df)
```

Output:



```
  Name  Hours  Rate  Salary
0    A    160   200   32000
1    B    170   180   30600
2    C    150   220   33000
3    D    180   190   34200
4    E    165   210   34650
5    F    175   205   35875
6    G    155   195   30225
```

K. M. S. P Mandal's **Sant Rawool Maharaj Mahavidyalaya, Kudal** **Department of Information Technology**	**Date: 14/11/2025** **Roll No: 20** **Expt. No: 06**	**Signature**
Title: Build the time hub, links and satellites		

Time Vault:

A Time Vault is an extension of the Data Vault model that focuses on tracking data changes over time. It stores historical data so we can analyse what changed, when it changed, and how it changed.

Time Hub:

Time Hub is a core component of the Time Vault (Temporal Data Vault) model. It represents time as a separate entity using dates or timestamps. Time Hub helps in tracking when data changes occur and supports historical and trend analysis. It enables accurate time-based queries and auditing of data.

Time Link:

Link is a component of the Time Vault / Data Vault model. It represents the relationship between two or more hubs, often including a time hub. Links capture events or interactions between entities. They help answer how entities are connected and when the connection occurred.

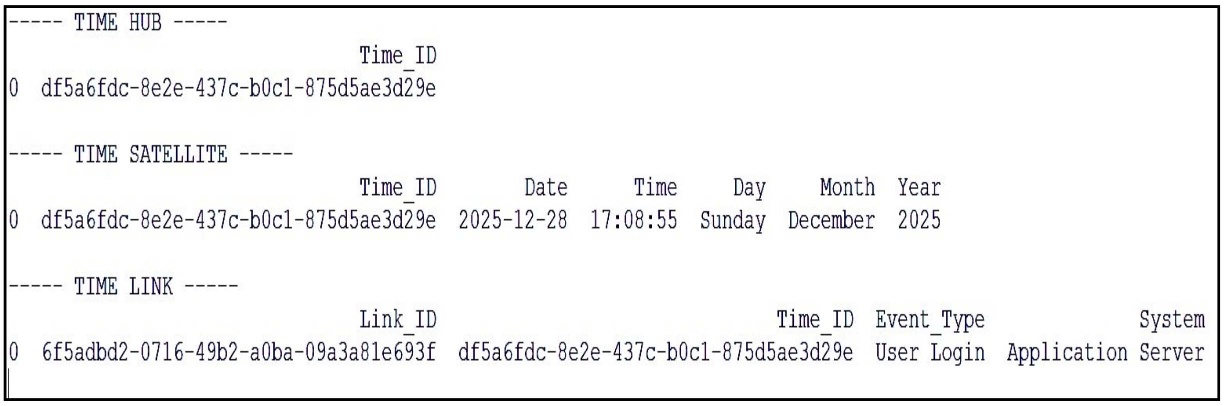
Time Satellite:

Time Satellite is a component of the Time Vault (Temporal Data Vault) model. It stores time-dependent descriptive attributes related to a hub or link. Time Satellites keep historical changes by recording data with timestamps. They help analyse how details change over time.

Code:

```
import pandas as pd
from datetime import datetime
import uuid

pd.set_option("display.max_columns", None)
pd.set_option("display.width", None)
time_id = str(uuid.uuid4())
time_hub = pd.DataFrame({
    "Time_ID": [time_id]
})

print("----- TIME HUB -----")
print(time_hub)
current_time = datetime.now()
time_satellite = pd.DataFrame({
    "Time_ID": [time_id],
    "Date": [current_time.strftime("%Y-%m-%d")],
    "Time": [current_time.strftime("%H:%M:%S")],
    "Day": [current_time.strftime("%A")],
    "Month": [current_time.strftime("%B")],
    "Year": [current_time.year]
})

print("\n----- TIME SATELLITE -----")
print(time_satellite)
time_link = pd.DataFrame({
    "Link_ID": [str(uuid.uuid4())],
    "Time_ID": [time_id],
    "Event_Type": ["User Login"],
    "System": ["Application Server"]
})
print("\n----- TIME LINK -----")
print(time_link)
```

Output:



```
----- TIME HUB -----
                                Time_ID
0  df5a6fdc-8e2e-437c-b0c1-875d5ae3d29e

----- TIME SATELLITE -----
                                Time_ID        Date      Time     Day     Month  Year
0  df5a6fdc-8e2e-437c-b0c1-875d5ae3d29e  2025-12-28  17:08:55  Sunday  December  2025

----- TIME LINK -----
                                Link_ID                               Time_ID  Event_Type              System
0  6f5adbd2-0716-49b2-a0ba-09a3a81e693f  df5a6fdc-8e2e-437c-b0c1-875d5ae3d29e  User Login  Application Server
```

K. M. S. P Mandal’s	**Date: 21/11/2025**	
Sant Rawool Maharaj Mahavidyalaya, Kudal	**Roll No: 20**	
Department of Information Technology	**Expt. No: 07**	**Signature**

Title: Transforming data

Transforming Data:

Transforming data is the process of converting raw data into a suitable format for analysis. It includes cleaning data, handling missing values, normalization, and feature creation. Data transformation improves data quality and consistency. It helps make data ready for accurate analysis and modelling.

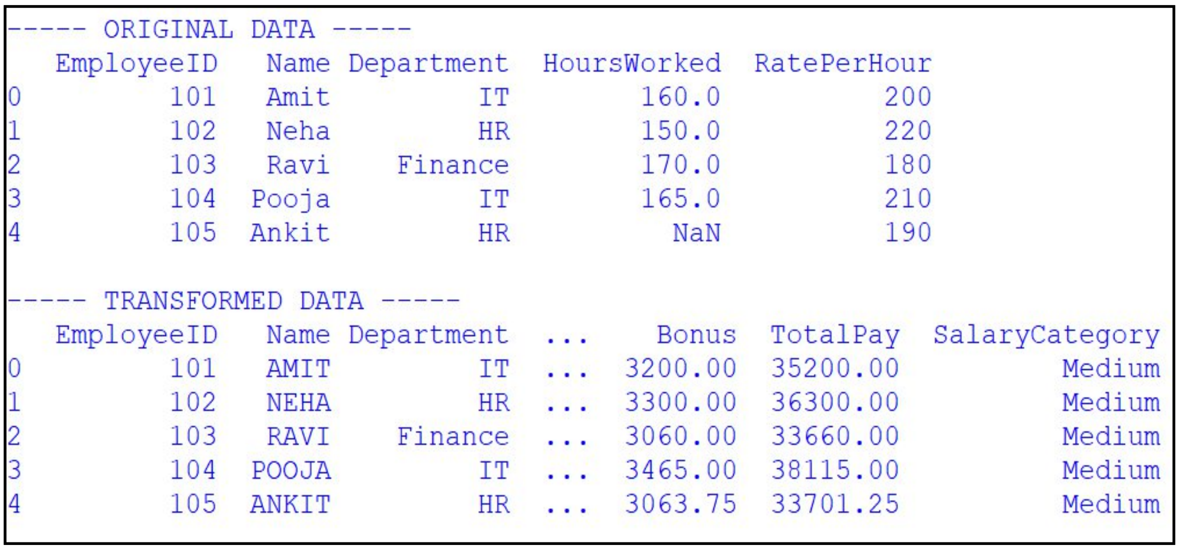
Benefits of transforming the Data:

1. **Improves data accuracy and quality:**
   Cleaning and transforming data removes errors, duplicates, and missing values, making the data reliable.
2. **Makes data suitable for analysis:**
   Raw data is converted into a structured format that can be easily used for analysis and modelling.
3. **Enhances model performance:**
   Properly transformed data helps machine learning models learn better and give accurate results.
4. **Reduces data complexity:**
   Transformation simplifies large and complex datasets, making them easier to understand and process.
5. **Supports better decision-making:**
   High-quality and well-prepared data leads to correct insights and confident decisions.

Code:

```
import pandas as pd

df = pd.read_csv("employees-2.csv")

print("----- ORIGINAL DATA -----")
print(df)
df["HoursWorked"] = df["HoursWorked"].fillna(df["HoursWorked"].mean())
df["Salary"] = df["HoursWorked"] * df["RatePerHour"]

# Add Bonus (10% of Salary)
df["Bonus"] = df["Salary"] * 0.10
df["TotalPay"] = df["Salary"] + df["Bonus"]
df["Name"] = df["Name"].str.upper()
df["SalaryCategory"] = df["Salary"].apply(
    lambda x: "High" if x >= 35000 else "Medium")
print("\n----- TRANSFORMED DATA -----")
print(df)
```

Output:



```
----- ORIGINAL DATA -----
   EmployeeID   Name Department  HoursWorked  RatePerHour
0         101   Amit         IT        160.0          200
1         102   Neha         HR        150.0          220
2         103   Ravi    Finance        170.0          180
3         104  Pooja         IT        165.0          210
4         105  Ankit         HR          NaN          190

----- TRANSFORMED DATA -----
   EmployeeID   Name Department  ...    Bonus  TotalPay SalaryCategory
0         101   AMIT         IT  ...  3200.00  35200.00         Medium
1         102   NEHA         HR  ...  3300.00  36300.00         Medium
2         103   RAVI    Finance  ...  3060.00  33660.00         Medium
3         104  POOJA         IT  ...  3465.00  38115.00         Medium
4         105  ANKIT         HR  ...  3063.75  33701.25         Medium
```

K. M. S. P Mandal's **Sant Rawool Maharaj Mahavidyalaya, Kudal** **Department of Information Technology**	**Date: 28/11/2025** **Roll No: 20** **Expt. No: 08**	**Signature**
Title: Organizing data		

Organizing Data:

Organizing data means arranging data in a clear and structured way. It helps store data in proper tables, rows, and columns. Organized data is easy to understand, access, and analyse. It reduces confusion and saves time during data processing.

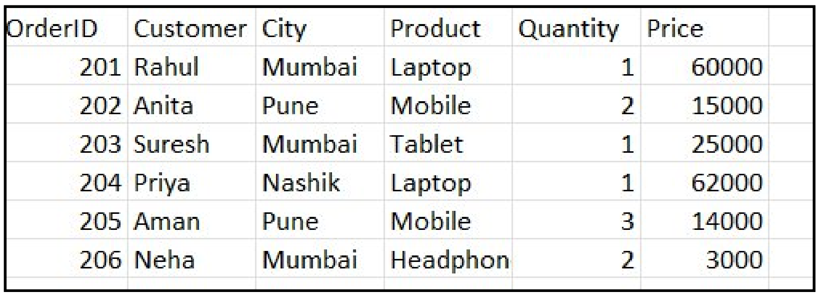
Benefits of Organizing the Data:

1. **Makes data easy to understand:**
   Well-organized data is clear and readable, so users can easily interpret the information.
2. **Saves time while searching and accessing data:**
   Proper arrangement helps find required data quickly without wasting time.
3. **Reduces errors and duplication:**
   Organized data avoids repeated or misplaced information, improving accuracy.
4. **Improves efficiency of data analysis:**
   Structured data can be analysed faster and gives better results.
5. **Supports better decision-making:**
   Clear and reliable data helps in making correct and confident decisions.

Code:

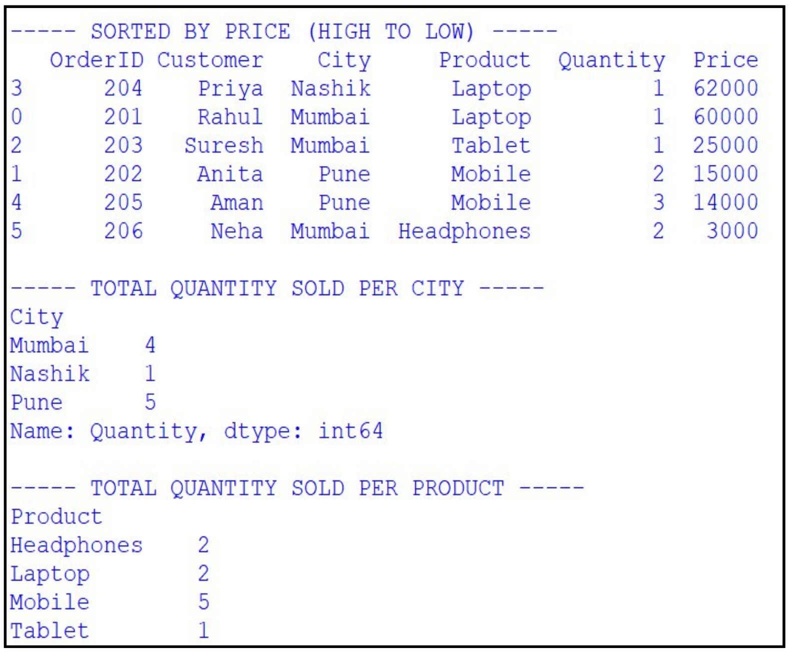
```
import pandas as pd
df = pd.read_csv("sales_data.csv")

sorted_data = df.sort_values(by="Price", ascending=False)
print("\n----- SORTED BY PRICE (HIGH TO LOW) -----")
print(sorted_data)
city_group = df.groupby("City")["Quantity"].sum()

print("\n----- TOTAL QUANTITY SOLD PER CITY -----")
print(city_group)

product_group = df.groupby("Product")["Quantity"].sum()
print("\n----- TOTAL QUANTITY SOLD PER PRODUCT -----")
print(product_group)
```

sales_data.csv:



OrderID	Customer	City	Product	Quantity	Price
201	Rahul	Mumbai	Laptop	1	60000
202	Anita	Pune	Mobile	2	15000
203	Suresh	Mumbai	Tablet	1	25000
204	Priya	Nashik	Laptop	1	62000
205	Aman	Pune	Mobile	3	14000
206	Neha	Mumbai	Headphon	2	3000

Output:



```
----- SORTED BY PRICE (HIGH TO LOW) -----
   OrderID Customer    City     Product  Quantity  Price
3      204    Priya  Nashik      Laptop         1  62000
0      201    Rahul  Mumbai      Laptop         1  60000
2      203   Suresh  Mumbai      Tablet         1  25000
1      202    Anita    Pune      Mobile         2  15000
4      205     Aman    Pune      Mobile         3  14000
5      206     Neha  Mumbai  Headphones         2   3000

----- TOTAL QUANTITY SOLD PER CITY -----
City
Mumbai    4
Nashik    1
Pune      5
Name: Quantity, dtype: int64

----- TOTAL QUANTITY SOLD PER PRODUCT -----
Product
Headphones    2
Laptop        2
Mobile        5
Tablet        1
```

K. M. S. P Mandal's **Sant Rawool Maharaj Mahavidyalaya, Kudal** **Department of Information Technology**	**Date: 05/12/2025** **Roll No: 20** **Expt. No: 09**	**Signature**
Title: Generating data		

Generating Data:

Generating data means creating new data either manually or automatically. It can be generated through surveys, sensors, software systems, or simulations. Generated data is used for analysis, testing, and decision-making. It helps in building and training data science models.

The random module is used to create random names, departments, ages, and salaries. Pandas is used to organize the generated data into a DataFrame. The DataFrame is then saved as a CSV file for further analysis. This mechanism is useful for testing, learning, and data science practice.

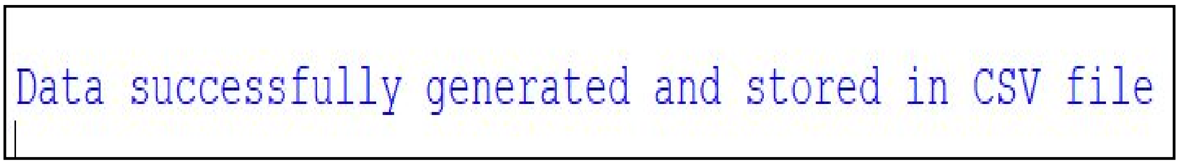
Benefits of generating data using the random library:

1. **Quick data creation:** Instantly generates sample data without manual entry.
2. **Variety and randomness:** Produces diverse values for testing different scenarios.
3. **Testing and simulation:** Useful for testing programs or models under various conditions.
4. **Practice and experimentation:** Helps learners and developers experiment with datasets.
5. **Reproducibility:** With a seed, you can generate the same random data for consistent testing.

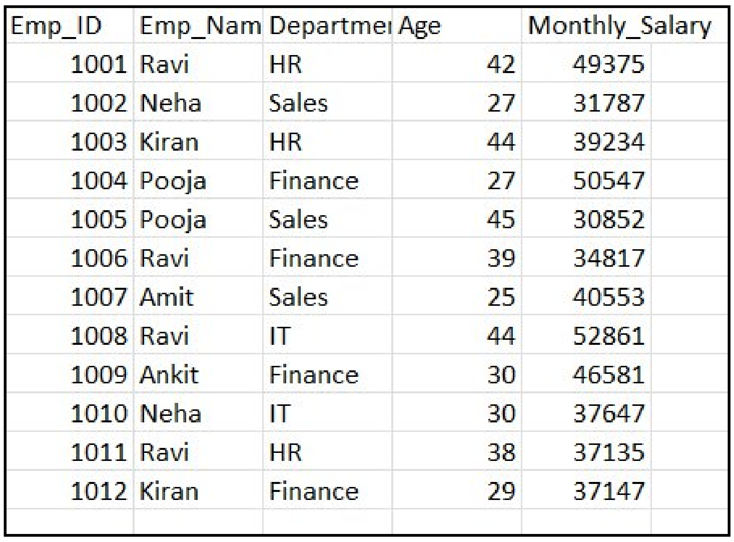
Code:

```
import pandas as pd
import random
record_count = 12
names = ["Amit", "Neha", "Ravi", "Pooja", "Ankit", "Kiran"]
departments = ["IT", "HR", "Finance", "Sales"]

data = {
    "Emp_ID": [],
    "Emp_Name": [],
    "Department": [],
    "Age": [],
    "Monthly_Salary": []
}
for i in range(record_count):
    data["Emp_ID"].append(1001 + i)
    data["Emp_Name"].append(random.choice(names))
    data["Department"].append(random.choice(departments))
    data["Age"].append(random.randint(22, 45))
    data["Monthly_Salary"].append(random.randint(25000, 60000))

df = pd.DataFrame(data)
df.to_csv("employee_generated_data.csv", index=False)
print("\nData successfully generated and stored in CSV file")
```

Output:



```
Data successfully generated and stored in CSV file
```



Emp_ID	Emp_Nam	Departme	Age	Monthly_Salary
1001	Ravi	HR	42	49375
1002	Neha	Sales	27	31787
1003	Kiran	HR	44	39234
1004	Pooja	Finance	27	50547
1005	Pooja	Sales	45	30852
1006	Ravi	Finance	39	34817
1007	Amit	Sales	25	40553
1008	Ravi	IT	44	52861
1009	Ankit	Finance	30	46581
1010	Neha	IT	30	37647
1011	Ravi	HR	38	37135
1012	Kiran	Finance	29	37147

K. M. S. P Mandal's **Sant Rawool Maharaj Mahavidyalaya, Kudal** **Department of Information Technology**	**Date: 16/12/2025** **Roll No: 20** **Expt. No: 10**	**Signature**
Title: Data visualisation using power Bi		

Data Visualization:

Data Visualization is the process of turning raw data into visual forms like charts, graphs, and dashboards. It helps people understand patterns, trends, and relationships in the data quickly. By using visuals, complex data becomes easy to interpret and analyse. It also helps in communicating insights effectively to others. Tools like Power BI, Tableau, and Excel are commonly used for data visualization.

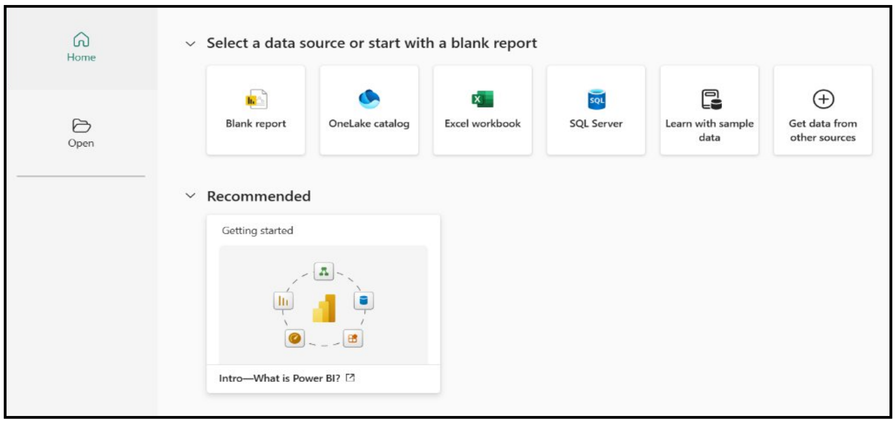
Power BI Tool:

Power BI is a Microsoft tool used to visualize and analyse data easily. It can connect to various data sources like Excel, CSV files, databases, and online services. You can clean and transform data before creating visuals. It allows you to make interactive charts, graphs, tables, and dashboards. Dashboards can be shared online for teams to collaborate and make decisions. Power BI helps turn raw data into clear insights quickly and effectively.

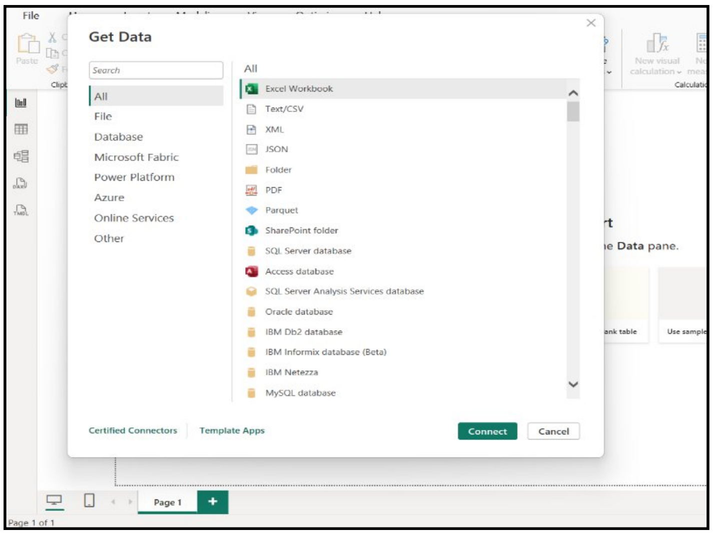
Steps to Visualize data in Power BI:

1. **Open Power BI Desktop:-** Launch the Power BI Desktop application on your computer.
2. **Import Data:-** Click "Get Data" and then choose the data source
3. **Explore Data:-** Go to Data View to check your dataset
4. **Create Visuals:-** Go to report view, select type of visual (Bar chart, Pie Chart, Table, etc)
5. **Customize Visuals:-** Use the Format panel to change colours, labels, titles, and layout.

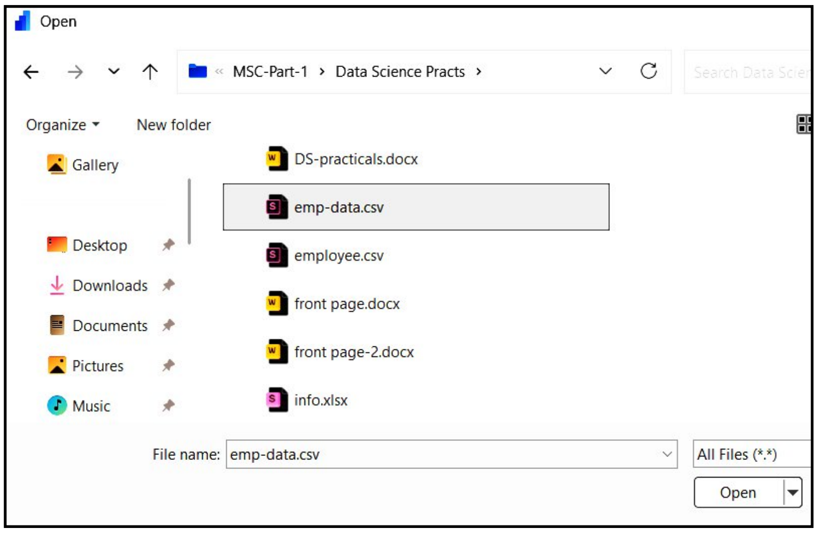
Open Power BI Desktop:



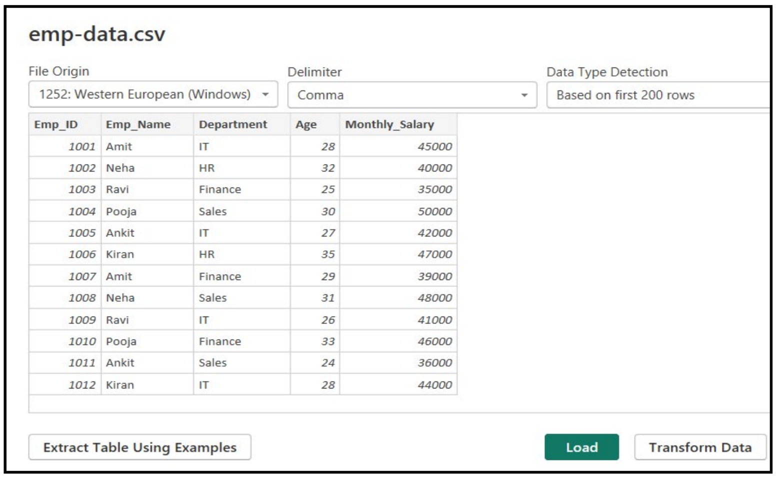
Click on "Get Data from other sources":



Select the data source file(emp-data.csv):



Load the data:

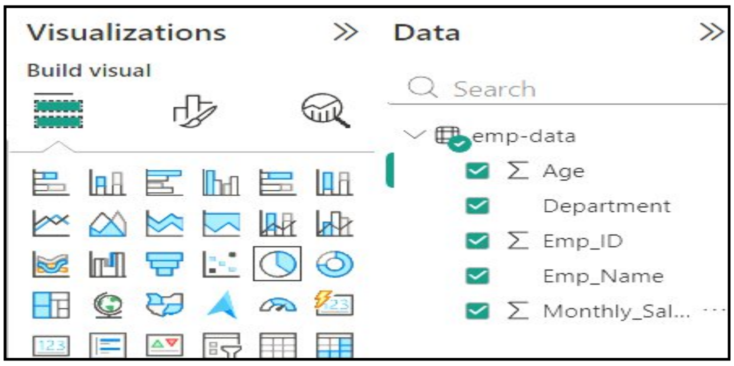


emp-data.csv

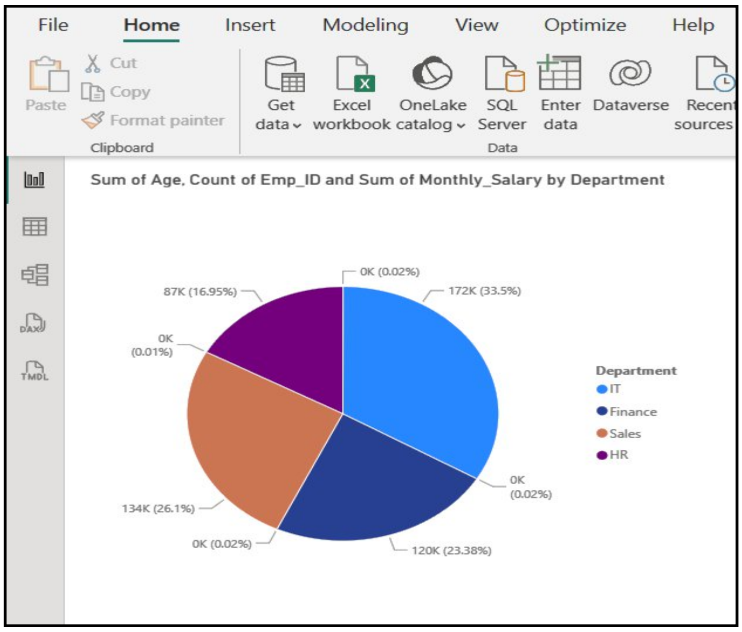
File Origin	Delimiter	Data Type Detection
1252: Western European (Windows)	Comma	Based on first 200 rows

Emp_ID	Emp_Name	Department	Age	Monthly_Salary
1001	Amit	IT	28	45000
1002	Neha	HR	32	40000
1003	Ravi	Finance	25	35000
1004	Pooja	Sales	30	50000
1005	Ankit	IT	27	42000
1006	Kiran	HR	35	47000
1007	Amit	Finance	29	39000
1008	Neha	Sales	31	48000
1009	Ravi	IT	26	41000
1010	Pooja	Finance	33	46000
1011	Ankit	Sales	24	36000
1012	Kiran	IT	28	44000

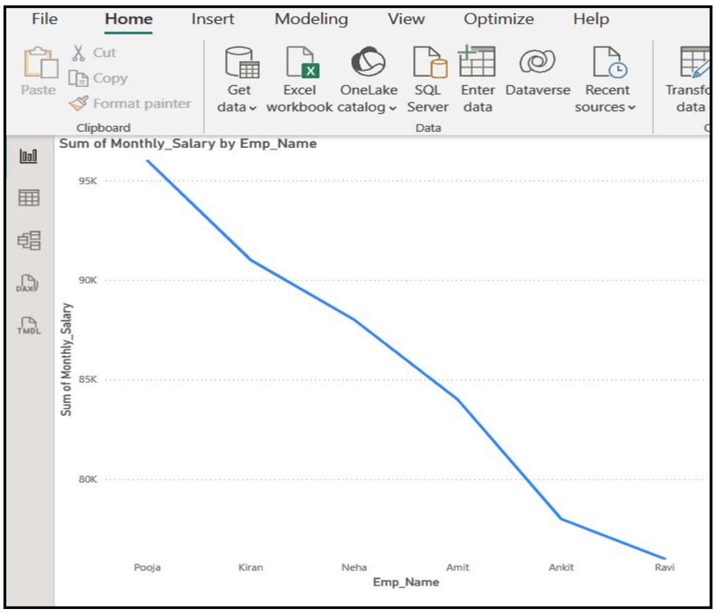
Select the data columns to be displayed on the visuals:



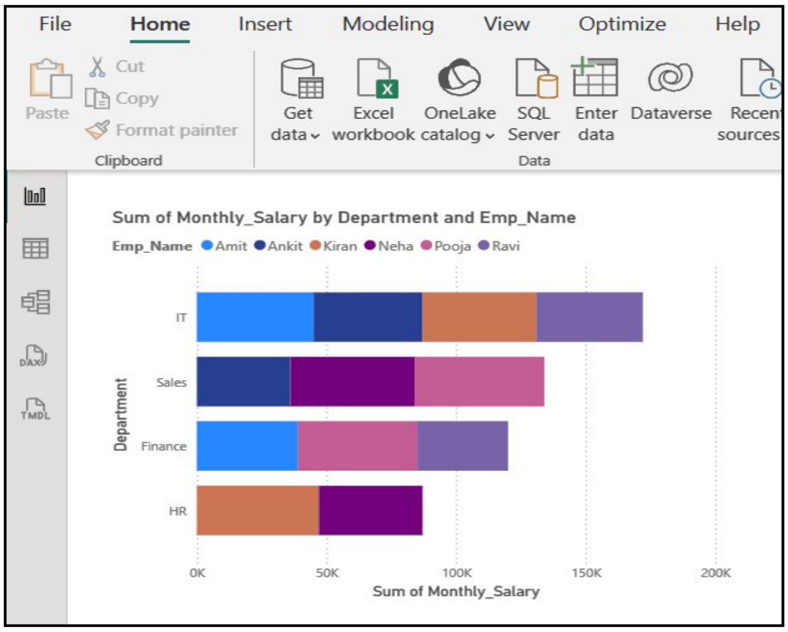
Pie Chart Visualization:



Line Chart:



Stacked Bar Chart:



SOFT COMPUTING

UNIVERSITY OF MUMBAI

K.M.S.P. Mandal’s

Sant Rawool Maharaj Mahavidyalaya

Kudal, Dist-Sindhudurg

DEPARTMEMT OF INFORMATION TECHNOLOGY

CERTIFICATE

This to certify that,

Mr / Miss. Shravani Dilip Sutar,

Exam Seat No. _________________ student of the M.Sc. (Information Technology, Part-I). He / She has been successfully completed practical as prescribed by University of Mumbai in the Sant Rawool Maharaj Mahavidyalaya during the semester I of the academic year 2025-26

In the following topics

____

____

Teacher In -Charge: **External Examiner:**

Date:

INDEX

Sr no.	Title	Sign
1.	Design simple neural network model and calculate the output of neural net using both binary and bipolar sigmoidal function.	
2.	Generate AND, OR, XOR function using McCulloch-Pitts neural net.	
3.	Implement the following a) Hebb's Rule b) Delta Rule	
4.	Write a Program to implement Hopfield Algorithm and Radial Basis Function	
5.	Write a program for Hopfield network model for associative memory and Linear Separation	
6.	Write a program for Kohonen Self organizing map and Adaptive resonance theory	
7.	Write a program for Back Propagation and Error Back Propagation	
8.	Finding ratios using fuzzy logic and solving tipping problem by using Fuzzy Logic	
9.	Implementation of Simple genetic algorithm (Creating two classes: City and Fitness using Genetic algorithm)	
10.	Implementation of Membership and Identity operators (in, not in, is, is not)	

K. M. S. P Mandal's	**Date: 01/08/2025**	
Sant Rawool Maharaj Mahavidyalaya, Kudal	**Roll No: 20**	
Department of Information Technology	**Expt. No: 01**	**Signature**

Title: Design simple neural network model and calculate the output of neural net using both binary and bipolar sigmoidal function.

Linear Neural Network:

Neural networks are computer models inspired by how the human brain works. They are made up of connected units called neurons that process information. These neurons learn patterns from data. Neural networks help computers recognize patterns and make decisions.

A linear neural network is a simple neural model that produces output by multiplying inputs with weights and adding a bias.

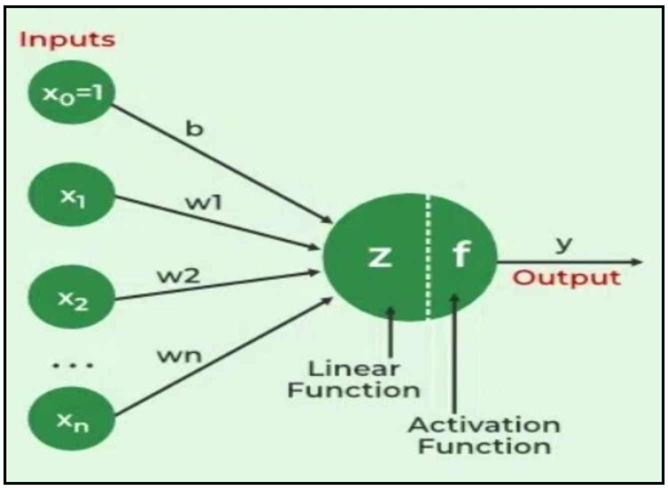
Formula to calculate the output:

$z = w_1x_1 + w_2x_2 + \ldots. + w_nx_n + b$, where, w= weight, x=inputs, b=bias value

Steps to calculate the output:

1. Take input keywords from the user.
2. Assign random weights and bias for scoring.
3. Calculate a score using the formula (weight × input) + bias.
4. Compare the keywords with each course in the dataset.
5. Store courses that match the keywords along with their scores.
6. Compute the average score for all matching courses.
7. If no exact match is found, select the least relevant course with the lowest score.
8. Display the matching or least relevant courses along with their scores.

Analogy of Inputs received and outputs generated:

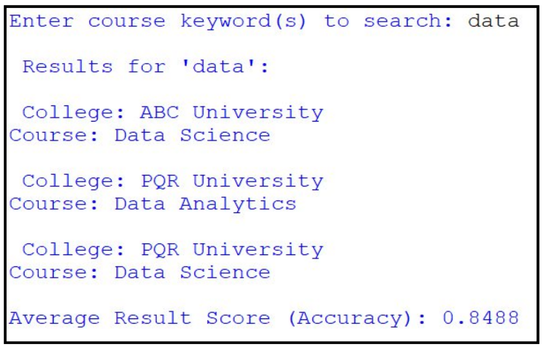


Code:

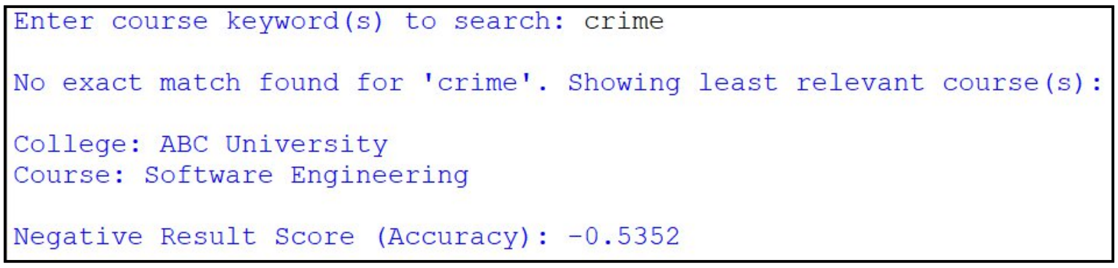
```
import random
dataset = [
   {
      "college": "ABC University",
      "courses": ["Computer Science","Information Technology","Data Science",
         "Artificial Intelligence","Software Engineering"]},
   {
      "college": "XYZ Institute of Technology",
      "courses": ["Mechanical Engineering","Civil Engineering","Electrical
Engineering","Robotics","Aerospace Engineering"]},
   {
      "college": "LMN College",
      "courses": ["Business Administration","Marketing Management","Finance and
Accounting","Human Resource Management","Entrepreneurship" ]},
   {
      "college": "PQR University",
      "courses": ["Cyber Security","Network Engineering","Cloud
Computing","Blockchain Technology","Data Analytics","Data Science"]},
# --- Formula: (wi * xi) + bi ---
def calculate_accuracy(wi, xi, bi):
   return (wi * xi) + bi
ignore_words = {"course", "courses", "program", "programs"}
def generate_score():
   wi = random.uniform(0.7, 1.0)
   xi = random.uniform(0.6, 1.0)
   bi = random.uniform(0.01, 0.1)
   return calculate_accuracy(wi, xi, bi)
def search_course(search_phrase):
   keywords = [word.lower() for word in search_phrase.split() if word.lower() not in
ignore_words]
   results = []
   accuracies = []
   for college in dataset:
      for course in college["courses"]:
         course_lower = course.lower()
         if all(keyword in course_lower for keyword in keywords):
            score = generate_score()
            results.append({"college": college["college"], "course": course})
            accuracies.append(score)
   if results:
      avg_accuracy = sum(accuracies) / len(accuracies)
      print(f"\n Results for '{search_phrase}':\n")
      for res in results:
         print(f" College: {res['college']}")
         print(f"Course: {res['course']}\n")
      print(f"Average Result Score (Accuracy): {avg_accuracy:.4f}")
   else:
      least_relevant = []
```

```
        lowest_score = float('inf')
        for college in dataset:
            for course in college["courses"]:
                score = generate_score()
                if score < lowest_score:
                    lowest_score = score
                    least_relevant = [{"college": college["college"], "course": course}]
                elif score == lowest_score:
                    least_relevant.append({"college": college["college"], "course": course})
        print(f"\nNo exact match found for '{search_phrase}'. Showing least relevant
course(s):\n")
        for res in least_relevant:
            print(f"College: {res['college']}")
            print(f"Course: {res['course']}\n")
        print(f"Negative Result Score (Accuracy): {-lowest_score:.4f}")
if __name__ == "__main__":
    search_phrase = input("Enter course keyword(s) to search: ")
    search_course(search_phrase)
```

Output:



```
Enter course keyword(s) to search: data

 Results for 'data':

 College: ABC University
Course: Data Science

 College: PQR University
Course: Data Analytics

 College: PQR University
Course: Data Science

Average Result Score (Accuracy): 0.8488
```



```
Enter course keyword(s) to search: crime

No exact match found for 'crime'. Showing least relevant course(s):

College: ABC University
Course: Software Engineering

Negative Result Score (Accuracy): -0.5352
```

K. M. S. P Mandal’s **Sant Rawool Maharaj Mahavidyalaya, Kudal** **Department of Information Technology**	**Date: 05/08/2025** **Roll No: 20** **Expt. No: 02**	**Signature**

Title: Generate AND, OR, XOR function using McCulloch-Pitts neural net

McCulloch-Pitts Neural Net:

The McCulloch-Pitts Neural Network is one of the earliest models of artificial neurons. It works like a simple decision-making unit: it takes multiple binary inputs, applies weights, and outputs a binary signal (0 or 1) based on a threshold. If the weighted sum of inputs exceeds the threshold, the neuron “fires” (outputs 1); otherwise, it stays 0. This model laid the foundation for understanding how neurons can perform logic operations and inspired modern neural networks.

Formula to calculate the output:

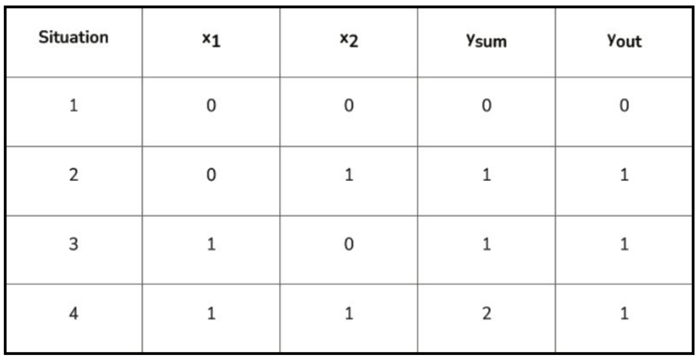
ysum=$w_1x_1 + w_2x_2 + \ldots + w_nx_n$
yout= f(ysum)= 1 if x >=1, otherwise 0

w=weight, x=input, n=number of inputs

Steps to calculate the Output:

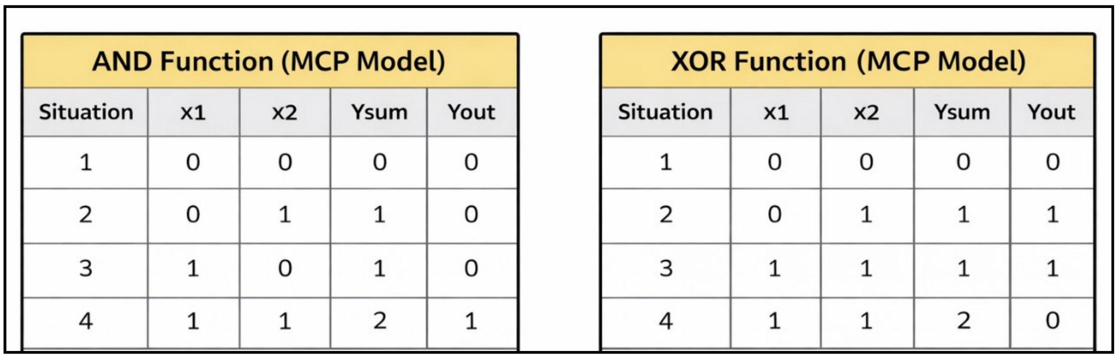
1. Define logical (AND / OR/ XOR) neuron.
2. Take user input, get two course names from the user.
3. Check database, see if each course exists in the CSV and set x1 and x2 to 1 or 0.
4. Calculate weighted sum, compute sum_value = w1*x1 + w2*x2.
5. Compute logical output, apply logic to x1 and x2 to get the final output.

Truth table for OR function:



Situation	x_1	x_2	y_{sum}	y_{out}
1	0	0	0	0
2	0	1	1	1
3	1	0	1	1
4	1	1	2	1

Truth table for XOR and AND function:



AND Function (MCP Model)

Situation	x1	x2	Ysum	Yout
1	0	0	0	0
2	0	1	1	0
3	1	0	1	0
4	1	1	2	1

XOR Function (MCP Model)

Situation	x1	x2	Ysum	Yout
1	0	0	0	0
2	0	1	1	1
3	1	1	1	1
4	1	1	2	0

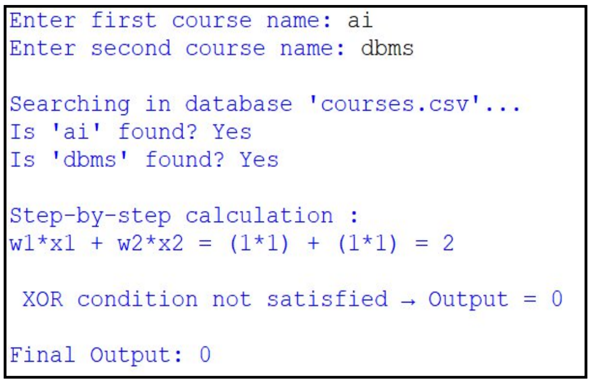
XOR

Code:

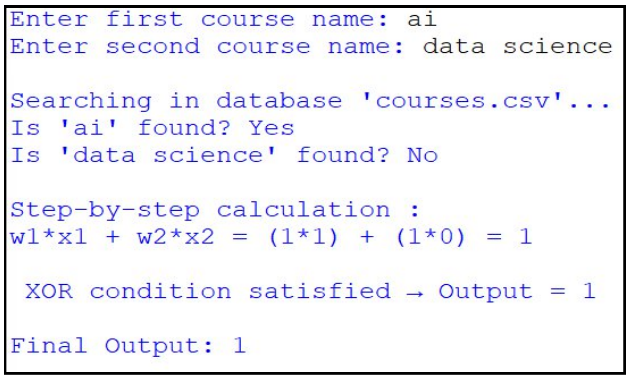
```
import csv
import os

filename = "courses.csv"
if not os.path.exists(filename):
    with open(filename, mode="w", newline="") as file:
        writer = csv.writer(file)
        writer.writerow(["course_name"])
        writer.writerow(["DBMS"])
        writer.writerow(["Python"])
        writer.writerow(["SQL"])
        writer.writerow(["SQL"])
        writer.writerow(["AI"])
        writer.writerow(["DBMS"])

def xor_neuron(x1, x2):
    return 1 if ((x1 or x2) and not (x1 and x2)) else 0

word1 = input("Enter first course name: ").strip().lower()
word2 = input("Enter second course name: ").strip().lower()
courses = []
with open(filename, mode="r") as file:
    reader = csv.DictReader(file)
    for row in reader:
        courses.append(row["course_name"].strip().lower())

x1 = 1 if word1 in courses else 0
x2 = 1 if word2 in courses else 0

print(f"\nSearching in database '{filename}'...")
print(f"Is '{word1}' found? {'Yes' if x1 else 'No'}")
print(f"Is '{word2}' found? {'Yes' if x2 else 'No'}")
```

```
w1, w2 = 1, 1
sum_value = w1*x1 + w2*x2
print(f"\nStep-by-step calculation :")
print(f"w1*x1 + w2*x2 = ({w1}*{x1}) + ({w2}*{x2}) = {sum_value}")

y = xor_neuron(x1, x2)
if y == 1:
    print("\n XOR condition satisfied → Output = 1")
else:
    print("\n XOR condition not satisfied → Output = 0")

print("\nFinal Output:", y)
```

Output:



```
Enter first course name: ai
Enter second course name: dbms

Searching in database 'courses.csv'...
Is 'ai' found? Yes
Is 'dbms' found? Yes

Step-by-step calculation :
w1*x1 + w2*x2 = (1*1) + (1*1) = 2

 XOR condition not satisfied → Output = 0

Final Output: 0
```



```
Enter first course name: ai
Enter second course name: data science

Searching in database 'courses.csv'...
Is 'ai' found? Yes
Is 'data science' found? No

Step-by-step calculation :
w1*x1 + w2*x2 = (1*1) + (1*0) = 1

 XOR condition satisfied → Output = 1

Final Output: 1
```

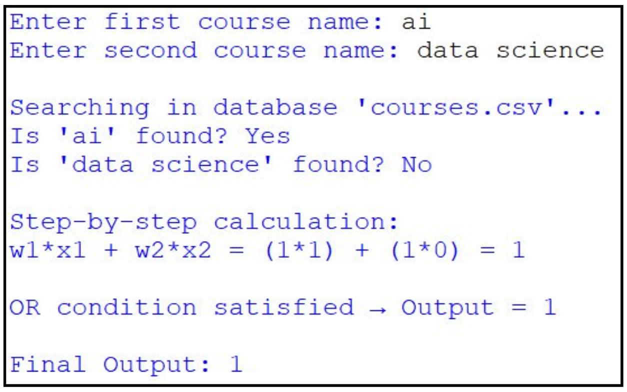
OR

Code:

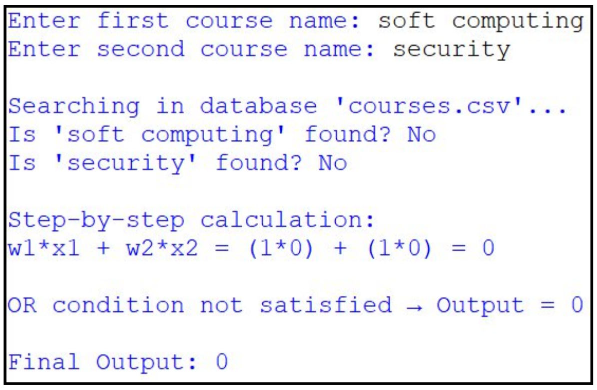
```
import csv
import os

filename = "courses.csv"

if not os.path.exists(filename):
    with open(filename, mode="w", newline="") as file:
        writer = csv.writer(file)
        writer.writerow(["course_name"])
        writer.writerow(["DBMS"])
        writer.writerow(["Python"])
        writer.writerow(["SQL"])
        writer.writerow(["NoSQL"])
        writer.writerow(["AI"])
        writer.writerow(["DBMS"])

def or_neuron(x1, x2):
    # OR logic: output 1 if any input is 1
    return 1 if (x1 == 1 or x2 == 1) else 0
```

```
word1 = input("Enter first course name: ").strip().lower()
word2 = input("Enter second course name: ").strip().lower()

courses = []
with open(filename, mode="r") as file:
    reader = csv.DictReader(file)
    for row in reader:
        courses.append(row["course_name"].strip().lower())

x1 = 1 if word1 in courses else 0
x2 = 1 if word2 in courses else 0

print(f"\nSearching in database '{filename}'...")
print(f"Is '{word1}' found? {'Yes' if x1 else 'No'}")
print(f"Is '{word2}' found? {'Yes' if x2 else 'No'}")

w1, w2 = 1, 1
sum_value = w1*x1 + w2*x2

print(f"\nStep-by-step calculation (without threshold):")
print(f"w1*x1 + w2*x2 = ({w1}*{x1}) + ({w2}*{x2}) = {sum_value}")
y = or_neuron(x1, x2)
if y == 1:
    print("\nOR condition satisfied → Output = 1")
else:
    print("\nOR condition not satisfied → Output = 0")

print("\nFinal Output:", y)
```

Output:



```
Enter first course name: ai
Enter second course name: data science

Searching in database 'courses.csv'...
Is 'ai' found? Yes
Is 'data science' found? No

Step-by-step calculation:
w1*x1 + w2*x2 = (1*1) + (1*0) = 1

OR condition satisfied → Output = 1

Final Output: 1
```



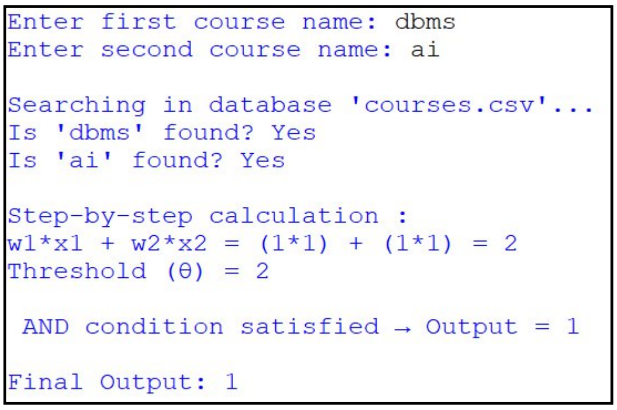
```
Enter first course name: soft computing
Enter second course name: security

Searching in database 'courses.csv'...
Is 'soft computing' found? No
Is 'security' found? No

Step-by-step calculation:
w1*x1 + w2*x2 = (1*0) + (1*0) = 0

OR condition not satisfied → Output = 0

Final Output: 0
```

AND

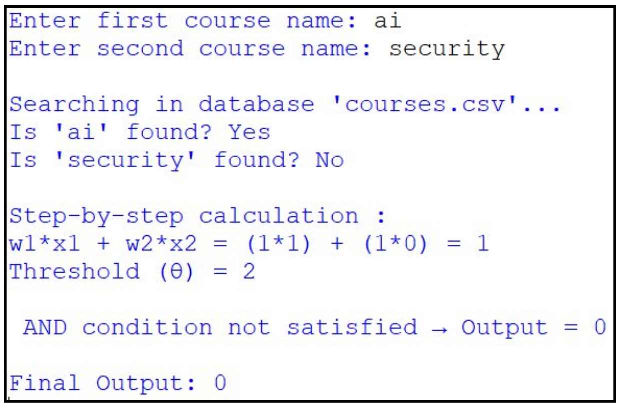
Code:

```
import csv
import os
filename = "courses.csv"
if not os.path.exists(filename):
    with open(filename, mode="w", newline="") as file:
        writer = csv.writer(file)
        writer.writerow(["course_name"])
        writer.writerow(["DBMS"])
        writer.writerow(["Python"])
        writer.writerow(["SQL"])
        writer.writerow(["NoSql"])
        writer.writerow(["AI"])
        writer.writerow(["DBMS"])

def and_neuron(x1, x2):
    return 1 if (x1 == 1 and x2 == 1) else 0
word1 = input("Enter first course name: ").strip().lower()
word2 = input("Enter second course name: ").strip().lower()
courses = []
with open(filename, mode="r") as file:
    reader = csv.DictReader(file)
    for row in reader:
        courses.append(row["course_name"].strip().lower())
x1 = 1 if word1 in courses else 0
x2 = 1 if word2 in courses else 0
print(f"\nSearching in database '{filename}'...")
print(f"Is '{word1}' found? {'Yes' if x1 else 'No'}")
print(f"Is '{word2}' found? {'Yes' if x2 else 'No'}")
w1, w2 = 1, 1
theta = 2
sum_value = w1*x1 + w2*x2
print(f"\nStep-by-step calculation :")
print(f"w1*x1 + w2*x2 = ({w1}*{x1}) + ({w2}*{x2}) = {sum_value}")
print(f"Threshold (θ) = {theta}")

# AND output using threshold
y = 1 if sum_value >= theta else 0
if y == 1:
    print("\n AND condition satisfied → Output = 1")
else:
    print("\n AND condition not satisfied → Output = 0")
print("\nFinal Output:", y)
```

Output:



```
Enter first course name: dbms
Enter second course name: ai

Searching in database 'courses.csv'...
Is 'dbms' found? Yes
Is 'ai' found? Yes

Step-by-step calculation :
w1*x1 + w2*x2 = (1*1) + (1*1) = 2
Threshold (θ) = 2

 AND condition satisfied → Output = 1

Final Output: 1
```



```
Enter first course name: ai
Enter second course name: security

Searching in database 'courses.csv'...
Is 'ai' found? Yes
Is 'security' found? No

Step-by-step calculation :
w1*x1 + w2*x2 = (1*1) + (1*0) = 1
Threshold (θ) = 2

 AND condition not satisfied → Output = 0

Final Output: 0
```

K. M. S. P Mandal’s **Sant Rawool Maharaj Mahavidyalaya, Kudal** **Department of Information Technology**	**Date: 09/08/2025** **Roll No: 20** **Expt. No: 03**	**Signature**

Title: Write a Program to implement Hebb’s Rule and Delta Rule

Hebb’ s Rule:
Hebb’s Rule is a basic learning principle in neural networks, often summarized as: “Neurons that fire together, wire together.” It means that if a neuron repeatedly helps activate another neuron, the connection (weight) between them is strengthened. In simple terms, the more two neurons are active at the same time, the stronger their link becomes. This rule is used to adjust weights in neural networks based on experience, helping the network learn patterns.

Weight update and Bias update formula:
w(new)= w(old) + x*y
b(new)= b(old) + y
here, w=weight, x=input, y=output, b=bias value

Steps to calculate Output:
1. Start the network with zero weights and zero bias, meaning it has no prior knowledge.
2. For each college–course pair, the input vector (college) is given to the neuron.
3. The neuron calculates its output using the current weights and bias.
4. According to Hebb’s rule, the weights are updated by adding the input multiplied by the output, strengthening the connection when both are active.
5. This process is repeated for all colleges and their courses to complete training.
6. During testing, the trained weights are used to predict the most related course for the given college input.

Delta Rule:
The Delta Rule is a learning rule used to reduce errors in neural networks. It works by comparing the actual output with the desired (target) output and calculating the error. The weights are then adjusted slightly in the direction that reduces this error. This process is repeated many times so the network gradually learns the correct output. In simple words, the delta rule learns by correcting mistakes step by step.

Formula to adjust the weight according to desired output:
cw=n*(d−y) *x
cb=n*(d-y)
here, cw= change in weight, n=learning rate, d=desired output, y= actual output, x=input, cb=change in bias

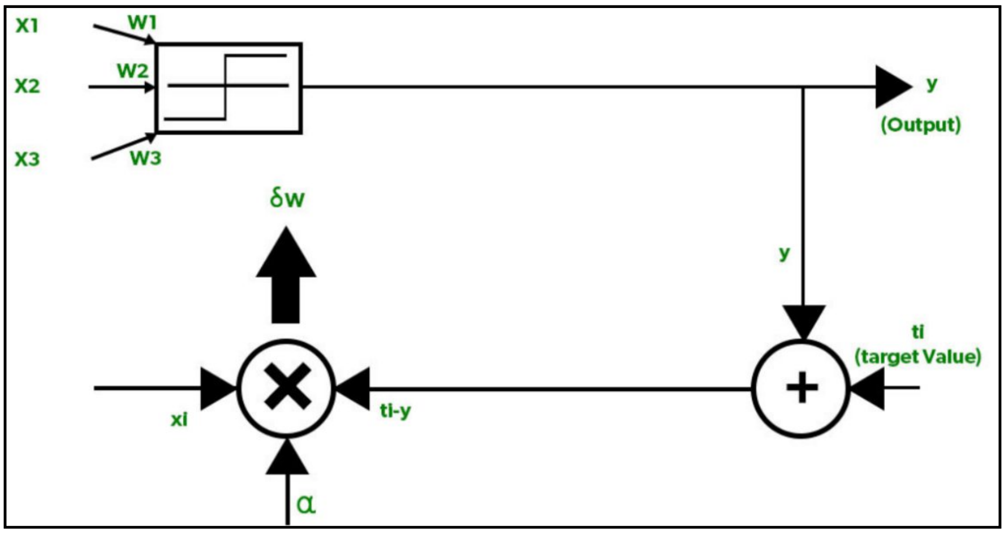
Steps to calculate the output:

1. Weights and bias are initialized to zero.
2. During training, the network calculates the output using the sigmoid function and compares it with the desired output.
3. The error (desired − output) is computed, and weights and bias are updated using the Delta Rule formula.
4. After training, the user enters a college and course name.
5. The trained model predicts the most relevant course and gives a confidence value for the user's input.

Truth table for AND function in Hebb's Rule:

INPUT				TARGET	
	x_1	x_2	b		y
X_1	-1	-1	1	Y_1	-1
X_2	-1	1	1	Y_2	-1
X_3	1	-1	1	Y_3	-1
X_4	1	1	1	Y_4	1

Analogy of Delta Rule:

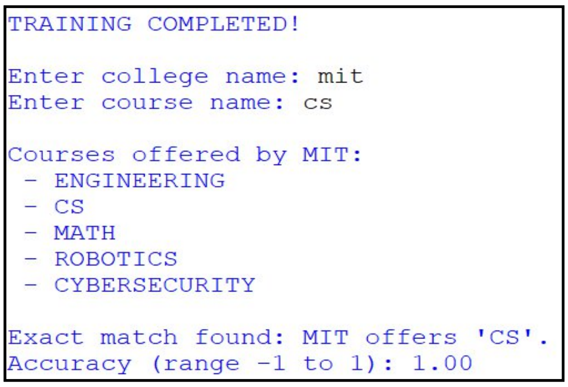


Hebb’s Rule:

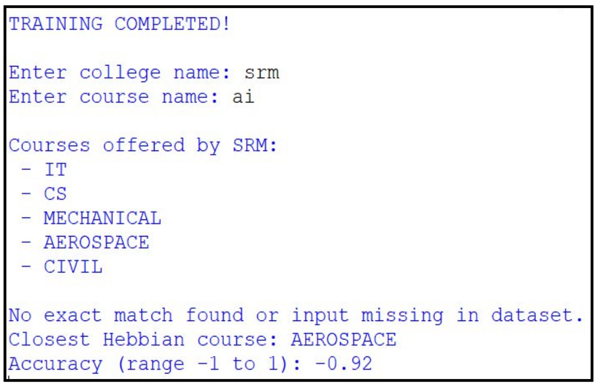
Code:

```
import numpy as np
dataset = {
    "Harvard": ["CS", "Biology", "Law", "AI", "Data Science"],
    "MIT": ["Engineering", "CS", "Math", "Robotics", "Cybersecurity"],
    "Stanford": ["Business", "CS", "AI", "Economics", "Psychology"],
    "IIT Delhi": ["CS", "Mechanical", "Electrical", "AI"],
    "IIT Bombay": ["CS", "Chemical", "Civil", "Data Science"],
    "SRM": ["IT", "CS", "Mechanical", "Aerospace", "Civil"],
    "VIT": ["IT", "Data Science", "Robotics", "CS", "Mechanical"]}
dataset = {k.upper(): [c.upper() for c in v] for k, v in dataset.items()}
colleges = list(dataset.keys())
courses = sorted({c for clist in dataset.values() for c in clist})
COL_SIZE = len(colleges)
COURSE_SIZE = len(courses)
college_to_id = {name: i for i, name in enumerate(colleges)}
course_to_id = {name: i for i, name in enumerate(courses)}
def bipolar_vector(size, index):
    v = -1 * np.ones(size)
    v[index] = 1
    return v
W = np.zeros((COURSE_SIZE, COL_SIZE))   # weights
b = np.zeros(COURSE_SIZE)               # bias
for college, offered_courses in dataset.items():
    x = bipolar_vector(COL_SIZE, college_to_id[college])
    for course in offered_courses:
        idx = course_to_id[course]
        y_raw = np.dot(W[idx], x) + b[idx]
        y = 1 if y_raw >= 0 else -1
        W[idx] += x * y
        b[idx] += y
print("TRAINING COMPLETED!")
def similarity_score(s1, s2):
    score = 0
    for c1, c2 in zip(s1, s2):
        if c1 == c2:
            score += 1
        else:
            break
    return score
user_college = input("\nEnter college name: ").strip().upper()
user_course = input("Enter course name: ").strip().upper()
college_exists = user_college in college_to_id
course_exists = user_course in course_to_id
if not college_exists:
    best_college = max(colleges, key=lambda c: similarity_score(c, user_college))
    college_idx = college_to_id[best_college]
    print(f"\nCollege '{user_college}' not found. Using closest college '{best_college}'.")
else:
    best_college = user_college
    college_idx = college_to_id[user_college]
if not course_exists:
    best_course = max(courses, key=lambda c: similarity_score(c, user_course))
    course_idx = course_to_id[best_course]
    print(f"Course '{user_course}' not found. Using closest course '{best_course}'.")
else:
    best_course = user_course
    course_idx = course_to_id[user_course]
print(f"\nCourses offered by {best_college}:")
for c in dataset[best_college]:
    print(" -", c)
x_user = bipolar_vector(COL_SIZE, college_idx)
net = W @ x_user + b
predicted_vector = np.where(net >= 0, 1, -1)
best_heb_course = None
best_accuracy = -1
for cname, idx in course_to_id.items():
    y_vec = bipolar_vector(COURSE_SIZE, idx)
    accuracy = (np.sum(predicted_vector == y_vec) -
                np.sum(predicted_vector != y_vec)) / COURSE_SIZE
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_heb_course = cname
if college_exists and course_exists and user_course in dataset[best_college]:
    print(f"\nExact match found: {best_college} offers '{user_course}'.")
    print("Accuracy (range -1 to 1): 1.00")
else:
    print("\nNo exact match found or input missing in dataset.")
    print(f"Closest Hebbian course: {best_heb_course}")
    print(f"Accuracy (range -1 to 1): {best_accuracy:.2f}")
```

Output:



```
TRAINING COMPLETED!

Enter college name: mit
Enter course name: cs

Courses offered by MIT:
 - ENGINEERING
 - CS
 - MATH
 - ROBOTICS
 - CYBERSECURITY

Exact match found: MIT offers 'CS'.
Accuracy (range -1 to 1): 1.00
```



```
TRAINING COMPLETED!

Enter college name: srm
Enter course name: ai

Courses offered by SRM:
 - IT
 - CS
 - MECHANICAL
 - AEROSPACE
 - CIVIL

No exact match found or input missing in dataset.
Closest Hebbian course: AEROSPACE
Accuracy (range -1 to 1): -0.92
```

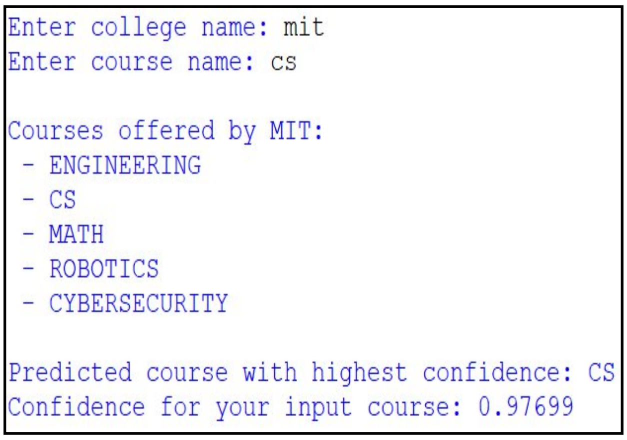
Delta Rule:

Code:

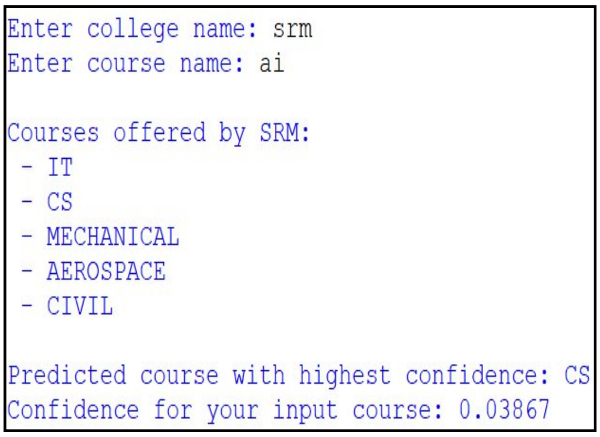
```
import numpy as np
dataset = {
   "Harvard": ["CS", "Biology", "Law", "AI", "Data Science"],
   "MIT": ["Engineering", "CS", "Math", "Robotics", "Cybersecurity"],
   "Stanford": ["Business", "CS", "AI", "Economics", "Psychology"],
   "UCLA": ["Medicine", "Biology", "Nursing", "Pharmacy", "Biotech"],
   "IIT Delhi": ["CS", "Mechanical", "Electrical", "AI"],
   "IIT Bombay": ["CS", "Chemical", "Civil", "Data Science"],
   "IIT Madras": ["Mechanical", "AI", "Robotics", "Physics"],
   "IISc Bangalore": ["Research", "Physics", "Biotech", "CS"],
   "SRM": ["IT", "CS", "Mechanical", "Aerospace", "Civil"],
   "VIT": ["IT", "Data Science", "Robotics", "CS", "Mechanical"]
}
dataset = {k.upper(): [c.upper() for c in v] for k, v in dataset.items()}

colleges = list(dataset.keys())
courses = sorted({c for clist in dataset.values() for c in clist})

COL_SIZE = len(colleges)
COURSE_SIZE = len(courses)
college_to_id = {name: i for i, name in enumerate(colleges)}
course_to_id = {name: i for i, name in enumerate(courses)}
def bipolar_vector(size, index):
   v = -1 * np.ones(size)
   v[index] = 1
   return v
def sigmoid(x):
   return 1 / (1 + np.exp(-x))
def similarity_score(s1, s2):
   score = 0
   for c1, c2 in zip(s1, s2):
      if c1 == c2:
         score += 1
      else:
         break
   return score
W = np.zeros((COURSE_SIZE, COL_SIZE))
b = np.zeros(COURSE_SIZE)
learning_rate = 0.1
epochs = 50
for epoch in range(epochs):
   for college, offered_courses in dataset.items():
      x = bipolar_vector(COL_SIZE, college_to_id[college])
      for course_idx in range(COURSE_SIZE):
         desired = 1.0 if courses[course_idx] in offered_courses else 0.0
         net = np.dot(W[course_idx], x) + b[course_idx]
```

```
            output = sigmoid(net)
            error = desired - output

            W[course_idx] += learning_rate * error * x
            b[course_idx] += learning_rate * error
user_college = input("\nEnter college name: ").strip().upper()
user_course = input("Enter course name: ").strip().upper()
if user_college in college_to_id:
    selected_college = user_college
    college_idx = college_to_id[user_college]
else:
    selected_college = max(colleges, key=lambda c: similarity_score(c, user_college))
    college_idx = college_to_id[selected_college]
    print(f"\nCollege '{user_college}' not found. Using closest match '{selected_college}'.")
if user_course in course_to_id:
    course_idx = course_to_id[user_course]
else:
    best_course = max(courses, key=lambda c: similarity_score(c, user_course))
    course_idx = course_to_id[best_course]
    print(f"Course '{user_course}' not found. Using closest match '{best_course}'.")
print(f"\nCourses offered by {selected_college}:")
for c in dataset[selected_college]:
    print(" -", c)
x_user = bipolar_vector(COL_SIZE, college_idx)
net = W @ x_user + b
y_pred = sigmoid(net)

best_course_pred = courses[np.argmax(y_pred)]
confidence = y_pred[course_idx]
print(f"\nPredicted course with highest confidence: {best_course_pred}")
print(f"Confidence for your input course: {confidence:.5f}")
```

Output:



```
Enter college name: mit
Enter course name: cs

Courses offered by MIT:
 - ENGINEERING
 - CS
 - MATH
 - ROBOTICS
 - CYBERSECURITY

Predicted course with highest confidence: CS
Confidence for your input course: 0.97699
```



```
Enter college name: srm
Enter course name: ai

Courses offered by SRM:
 - IT
 - CS
 - MECHANICAL
 - AEROSPACE
 - CIVIL

Predicted course with highest confidence: CS
Confidence for your input course: 0.03867
```

K. M. S. P Mandal's **Sant Rawool Maharaj Mahavidyalaya, Kudal** **Department of Information Technology**	**Date: 12/08/2025** **Roll No: 20** **Expt. No: 04**	**Signature**

Title: Write a Program to implement Hopfield Algorithm and Radial Basis Function

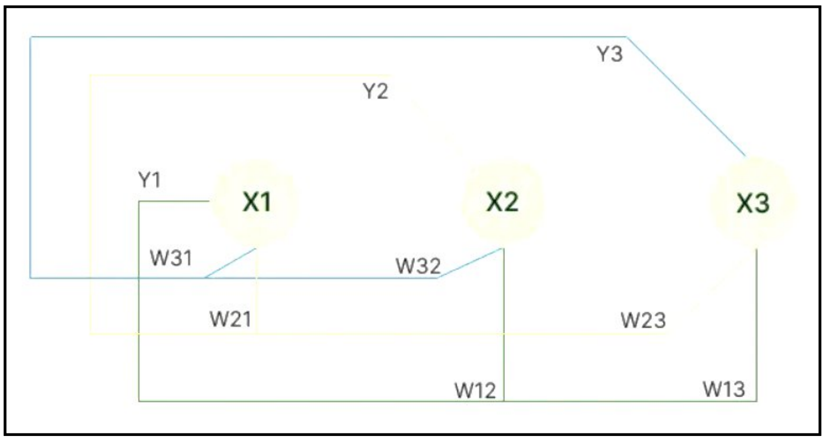
Hopfield Network:

A Hopfield Neural Network is a type of neural network used mainly for memory storage and pattern recall. It stores patterns like memories, and when you give it an incomplete or noisy input, it tries to remember the closest stored pattern. All neurons are connected to each other, and the connections are symmetric. The network works by minimizing an energy function, so it slowly settles into a stable state. In simple words, it behaves like a brain that recalls memories from partial information.

Formula to calculate the Output:

b(new)=b(old)+n
n(new)=0.5*n(old), where, b=bias value, n= (xi-wi), xi=input, wi=weight

Analogy of Hopfield Network:



Radial Basis Function:

A Radial Basis Function (RBF) is a function that measures how close one data point is to another. It gives a high value when points are near and a low value when they are far apart. RBF is used when data cannot be separated by straight lines and needs non-linear separation. It is commonly used in RBF neural networks for similarity-based matching.

Formula to calculate output:

$$\boxed{k(x,y) = \exp\left(-\frac{\|x-y\|^2}{2\sigma^2}\right)}$$

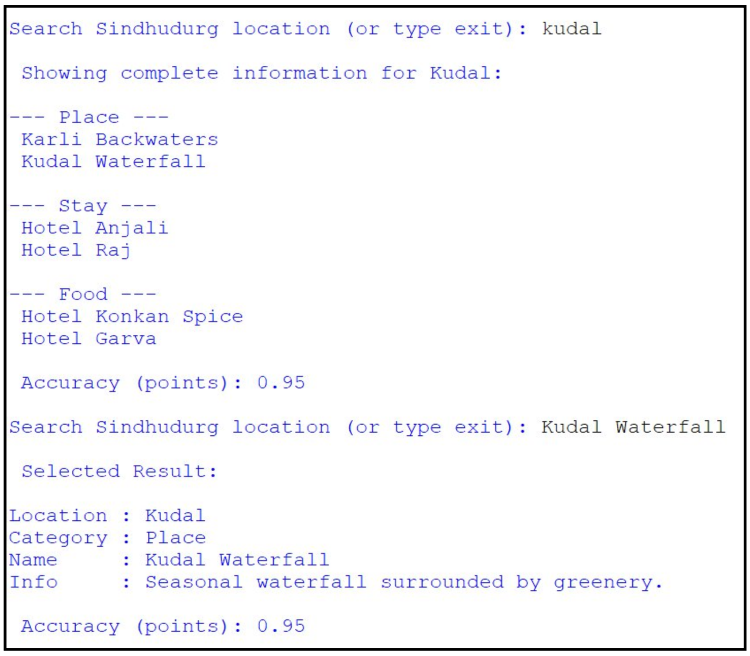
x= input data point
y= Center point (Stored data)
||x-y||= Distance between input and center points
σ→ **Spread parameter (controls how fast similarity decreases)**
k(x,y)= Output similarity value (ranges from 0 to 1)

Hopfield Algorithm:
Code:

```
import pandas as pd
import difflib

FILE = "sindhudurg_places.csv"
df = pd.read_csv(FILE)
if "info" not in df.columns:
    df["info"] = "Information not available."

class LearningModel:
    def __init__(self):
        self.memory = set()
    def learn(self, text):
        self.memory.add(text.lower())
model = LearningModel()
for col in ["location", "name"]:
    for val in df[col]:
        model.learn(str(val))
def calculate_accuracy(user_input, matched_output, iterations=6):
    user_input = user_input.lower().strip()
    matched_output = matched_output.lower().strip()

    if user_input == matched_output:
        return 0.95
    b = 0.0
    w = 0.0
    xi = difflib.SequenceMatcher(None, user_input, matched_output).ratio()
    n = xi - w
    for _ in range(iterations):
        b = b + n
        n = 0.5 * n
    accuracy = b / (b + 1)
    accuracy = min(max(accuracy, 0), 0.94)
    return round(accuracy, 3)
def smart_search(user_input, df):
    query = user_input.lower().strip()

    loc_exact = df[df["location"].str.lower() == query]

    if not loc_exact.empty:
        location_name = loc_exact.iloc[0]["location"]
        print(f"\n Showing complete information for {location_name}:\n")

        for category in ["Place", "Stay", "Food"]:
            items = loc_exact[loc_exact["category"] == category]
            if not items.empty:
                print(f"--- {category} ---")
                for _, row in items.iterrows():
```

```
                print(f"  {row['name']}")
            print()
    acc = calculate_accuracy(query, location_name)
    print(f" Accuracy (points): {acc}")
    return df
locations = df["location"].str.lower().unique().tolist()
loc_match = difflib.get_close_matches(query, locations, n=1, cutoff=0.7)

if loc_match:
    loc = loc_match[0]
    loc_df = df[df["location"].str.lower() == loc]
    print(f"\n Showing complete information for {loc.capitalize()}:\n")
    for category in ["Place", "Stay", "Food"]:
        items = loc_df[loc_df["category"] == category]
        if not items.empty:
            print(f"--- {category} ---")
            for _, row in items.iterrows():
                print(f"• {row['name']}")
            print()
    acc = calculate_accuracy(query, loc)
    print(f"Accuracy (points): {acc}")
    return df
exact = df[df["name"].str.lower() == query]

if not exact.empty:
    row = exact.iloc[0]
    print("\n Selected Result:\n")
    print(f"Location : {row['location']}")
    print(f"Category : {row['category']}")
    print(f"Name     : {row['name']}")
    print(f"Info     : {row['info']}")

    acc = calculate_accuracy(query, row["name"])
    print(f"\n Accuracy (points): {acc}")
    return df

names = df["name"].str.lower().tolist()
best = difflib.get_close_matches(query, names, n=1, cutoff=0.75)

if best:
    row = df[df["name"].str.lower() == best[0]].iloc[0]
    print("\n Auto-corrected Result:\n")
    print(f"Location : {row['location']}")
    print(f"Category : {row['category']}")
    print(f"Name     : {row['name']}")
    print(f"Info     : {row['info']}")

    acc = calculate_accuracy(query, row["name"])
    print(f"\n Accuracy (points): {acc}")
    return df
    partial = df[df["name"].str.lower().str.contains(query)]
    if not partial.empty:
        print("\n Related Results Found:\n")

        for _, row in partial.iterrows():
            print(f"Location : {row['location']}")
            print(f"Category : {row['category']}")
            print(f"Name     : {row['name']}")
            print(f"Info     : {row['info']}")
            print("-" * 30)
        acc = calculate_accuracy(query, partial.iloc[0]["name"])
        print(f"\n Accuracy (points): {acc}")
        return df
    print("\n No result found.")
    return df
while True:
    user_search = input("\nSearch Sindhudurg location (or type exit): ")
    if user_search.lower() == "exit":
        break
    df = smart_search(user_search, df)
```

Output:



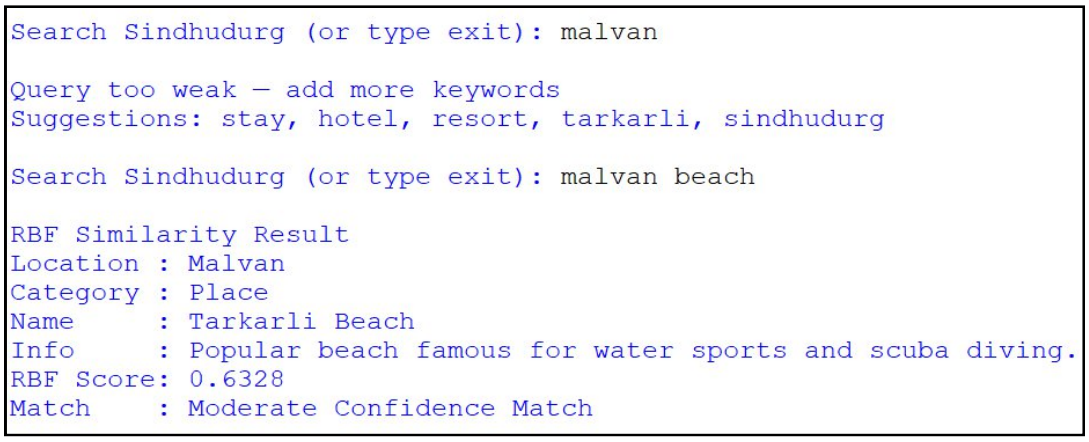
```
Search Sindhudurg location (or type exit): kudal

 Showing complete information for Kudal:

--- Place ---
 Karli Backwaters
 Kudal Waterfall

--- Stay ---
 Hotel Anjali
 Hotel Raj

--- Food ---
 Hotel Konkan Spice
 Hotel Garva

 Accuracy (points): 0.95

Search Sindhudurg location (or type exit): Kudal Waterfall

 Selected Result:

Location : Kudal
Category : Place
Name     : Kudal Waterfall
Info     : Seasonal waterfall surrounded by greenery.

 Accuracy (points): 0.95
```

Radial Basis Function:

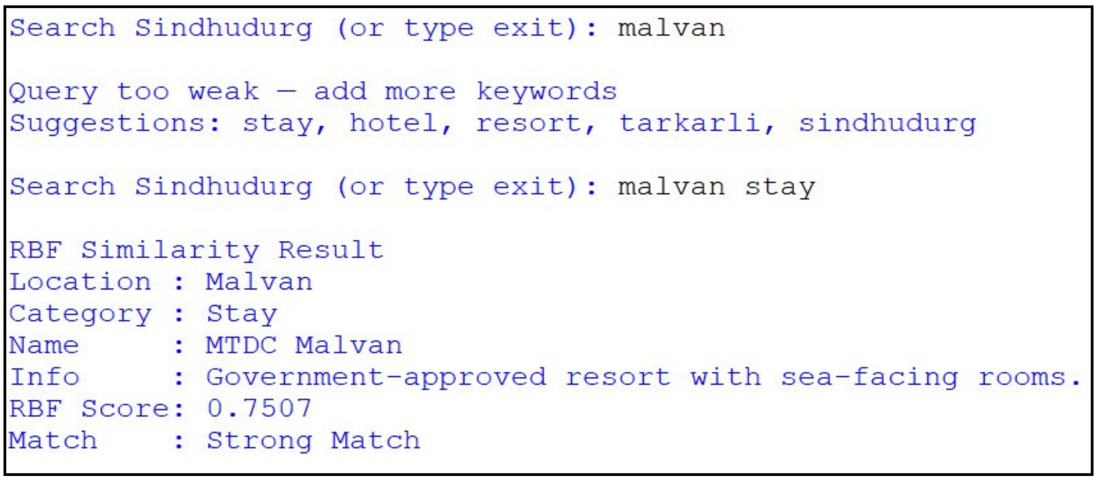
Code:

```
import pandas as pd
import numpy as np
FILE = "sindhudurg_places-4.csv"
df = pd.read_csv(FILE)
vocab = set()
for _, row in df.iterrows():
    text = f"{row['location']} {row['name']} {row['category']}".lower()
    vocab.update(text.split())

vocab = sorted(vocab)
index = {word: i for i, word in enumerate(vocab)}
N = len(vocab)
patterns = []
rows = []
for _, row in df.iterrows():
    vec = np.zeros(N)
    text = f"{row['location']} {row['name']} {row['category']}".lower()
    for word in text.split():
        vec[index[word]] = 1

    vec = vec / (np.linalg.norm(vec) + 1e-9)
    patterns.append(vec)
    rows.append(row)

patterns = np.array(patterns)
def gaussian_rbf(x, c, sigma=0.8):
    dist = np.linalg.norm(x - c)
    return np.exp(-(dist ** 2) / (2 * sigma ** 2))

def interpret_score(score):
    if score >= 0.9:
        return "Exact Match"
    elif score >= 0.75:
        return "Strong Match"
    elif score >= 0.55:
        return "Moderate Confidence Match"
    elif score >= 0.35:
        return "Low Confidence Match"
    else:
        return "No Reliable Match"

def suggest_keywords(query, max_suggestions=5):
    suggestions = set()
    query_words = query.lower().split()

    for _, row in df.iterrows():
        text = f"{row['location']} {row['name']} {row['category']}".lower()
        if any(word in text for word in query_words):
            suggestions.update(text.split())

    suggestions = [w for w in suggestions if w not in query_words]
    return suggestions[:max_suggestions]
def rbf_search(query, sigma=0.8):
    q_vec = np.zeros(N)
    valid_words = 0
    for word in query.lower().split():
        if word in index:
            q_vec[index[word]] = 1
            valid_words += 1

    if valid_words < 2:
        suggestions = suggest_keywords(query)
        return None, 0.0, suggestions
    q_vec = q_vec / (np.linalg.norm(q_vec) + 1e-9)
    scores = [gaussian_rbf(q_vec, p, sigma) for p in patterns]
    best = np.argmax(scores)
    return rows[best], scores[best], interpret_score(scores[best])

while True:
    query = input("\nSearch Sindhudurg (or type exit): ")
    if query.lower() == "exit":
        break
    result = rbf_search(query)
    if result[0] is None:
        print("\nQuery too weak — add more keywords")
        if result[2]:
            print("Suggestions:", ", ".join(result[2]))
        else:
            print("Suggestions: malvan beach, malvan stay, sindhudurg fort")
        continue
    row, score, label = result
    print("\nRBF Similarity Result")
    print(f"Location : {row['location']}")
    print(f"Category : {row['category']}")
    print(f"Name     : {row['name']}")
    print(f"Info     : {row['info']}")
    print(f"RBF Score: {round(score, 4)}")
    print(f"Match    : {label}")
```

Output:



```
Search Sindhudurg (or type exit): malvan

Query too weak — add more keywords
Suggestions: stay, hotel, resort, tarkarli, sindhudurg

Search Sindhudurg (or type exit): malvan beach

RBF Similarity Result
Location : Malvan
Category : Place
Name     : Tarkarli Beach
Info     : Popular beach famous for water sports and scuba diving.
RBF Score: 0.6328
Match    : Moderate Confidence Match
```



```
Search Sindhudurg (or type exit): malvan

Query too weak — add more keywords
Suggestions: stay, hotel, resort, tarkarli, sindhudurg

Search Sindhudurg (or type exit): malvan stay

RBF Similarity Result
Location : Malvan
Category : Stay
Name     : MTDC Malvan
Info     : Government-approved resort with sea-facing rooms.
RBF Score: 0.7507
Match    : Strong Match
```

K. M. S. P Mandal's	**Date: 14/08/2025**	
Sant Rawool Maharaj Mahavidyalaya, Kudal	**Roll No: 20**	
Department of Information Technology	**Expt. No: 05**	**Signature**

Title: Write a program for Hopfield network model for associative memory and Linear Separation

Hopfield Network for Associative Memory:

A Hopfield network is a simple neural network used to store and remember patterns. It works like a memory that can recognize a pattern even if it is incomplete or noisy. All neurons are connected to each other and update their values step by step. The network finally settles into a stable pattern, which is the recalled memory. So, it acts like associative memory, recalling the closest stored pattern.

Formula to calculate the Output:

$$w_{ij} = \sum_{p=1}^{P} x_i^{(p)} x_j^{(p)}, \ for\ i \neq j$$

$$w_{ii} = 0$$

a) **w_{ij}= weight between neuron i and neuron j**
b) **P= number of stored patterns**
c) **$x_i^{(p)}$= state of neuron iin pattern p(+ 1 or − 1)**
d) **$x_j^{(p)}$= state of neuron jin pattern p**

Linear Separation:

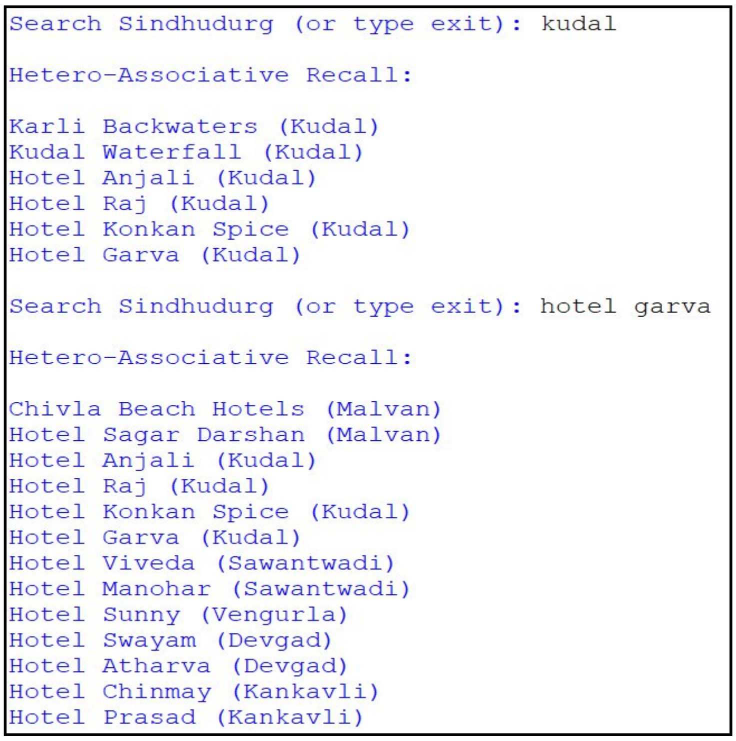
Linear separability means that different groups of data can be separated using a straight line (in 2D) or a flat plane (in higher dimensions). If one straight line can correctly divide the data into classes without overlap, the data is said to be linearly separable. It helps us understand whether simple models like the Perceptron can classify the data correctly.

Hopfield network model for associative memory

Code:

```
import pandas as pd
import numpy as np

FILE = "sindhudurg_places.csv"
df = pd.read_csv(FILE)

if "info" not in df.columns:
    df["info"] = "Information not available."

vocab = set()
for _, row in df.iterrows():
    text = f"{row['location']} {row['name']} {row['category']}".lower()
    vocab.update(text.split())
vocab = sorted(list(vocab))
index = {word: i for i, word in enumerate(vocab)}
N = len(vocab)  # number of neurons

patterns = []
rows = []
for _, row in df.iterrows():
    vec = np.full(N, -1)
    text = f"{row['location']} {row['name']} {row['category']}".lower()
    for word in text.split():
        vec[index[word]] = 1
    patterns.append(vec)
    rows.append(row)

patterns = np.array(patterns)
W = np.zeros((N, N))
for p in patterns:
    W += np.outer(p, p)
np.fill_diagonal(W, 0)

def hopfield_recall(query):
    state = np.full(N, -1)
    for word in query.lower().split():
        if word in index:
            state[index[word]] = 1
    # One-step update (synchronous)
    new_state = np.sign(W @ state)
    new_state[new_state == 0] = 1
    similarities = [np.dot(new_state, p) for p in patterns]
    best = np.argmax(similarities)
    return rows[best]
def hetero_recall(query):
    results = []
    for _, row in df.iterrows():
        text = f"{row['location']} {row['name']} {row['category']}".lower()
        if any(word in text for word in query.lower().split()):
            results.append(row)
    return results
def associative_search(user_input):
    print("\nHetero-Associative Recall:\n")
    hetero = hetero_recall(user_input)
    seen = set()
    for r in hetero:
        key = (r['name'], r['location'])
        if key not in seen:
            print(f"{r['name']} ({r['location']})")
            seen.add(key)
    auto = hopfield_recall(user_input)
while True:
    user_input = input("\nSearch Sindhudurg (or type exit): ")
    if user_input.lower() == "exit":
        break
    associative_search(user_input)
```

Output:



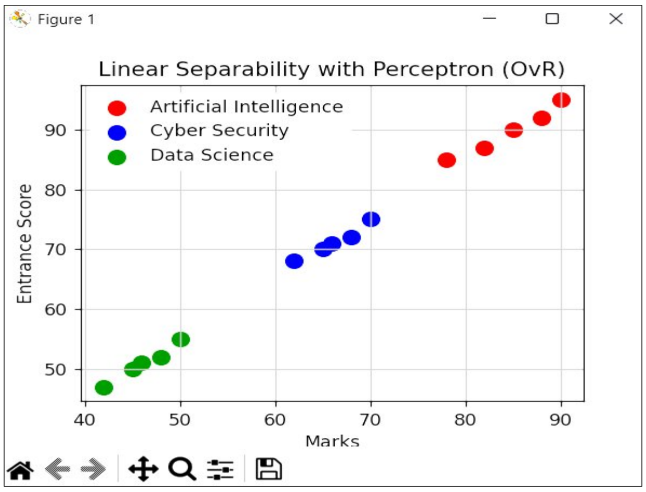
```
Search Sindhudurg (or type exit): kudal

Hetero-Associative Recall:

Karli Backwaters (Kudal)
Kudal Waterfall (Kudal)
Hotel Anjali (Kudal)
Hotel Raj (Kudal)
Hotel Konkan Spice (Kudal)
Hotel Garva (Kudal)

Search Sindhudurg (or type exit): hotel garva

Hetero-Associative Recall:

Chivla Beach Hotels (Malvan)
Hotel Sagar Darshan (Malvan)
Hotel Anjali (Kudal)
Hotel Raj (Kudal)
Hotel Konkan Spice (Kudal)
Hotel Garva (Kudal)
Hotel Viveda (Sawantwadi)
Hotel Manohar (Sawantwadi)
Hotel Sunny (Vengurla)
Hotel Swayam (Devgad)
Hotel Atharva (Devgad)
Hotel Chinmay (Kankavli)
Hotel Prasad (Kankavli)
```

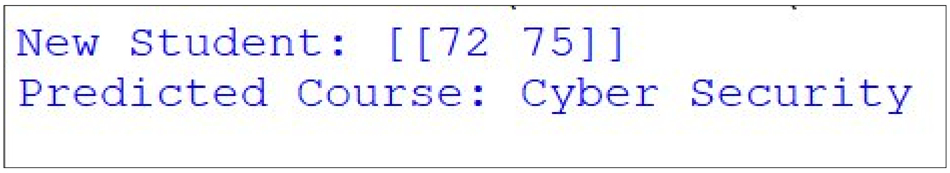
Linear Separation:

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
df = pd.read_csv("college_courses.csv")
label_map = {
    "Artificial Intelligence": 0,
    "Cyber Security": 1,
    "Data Science": 2
}
df["Label"] = df["Course"].map(label_map)

X = df[["Marks", "Entrance_Score"]].values
y = df["Label"].values
class Perceptron:
    def __init__(self, lr=0.01, epochs=1000):
        self.lr = lr
        self.epochs = epochs

    def fit(self, X, y):
        self.w = np.zeros(X.shape[1])
        self.b = 0
        for _ in range(self.epochs):
            for xi, target in zip(X, y):
                y_pred = self.predict_single(xi)
                update = self.lr * (target - y_pred)
                self.w += update * xi
                self.b += update
    def predict_single(self, x):
        return 1 if np.dot(x, self.w) + self.b >= 0 else 0
    def score(self, X):
        return np.dot(X, self.w) + self.b
models = {}
for course, label in label_map.items():
    y_binary = (y == label).astype(int)
    p = Perceptron()
    p.fit(X, y_binary)
    models[course] = p
course_colors = {
    "Artificial Intelligence": "red",
    "Cyber Security": "blue",
    "Data Science": "green"
}
plt.figure(figsize=(8, 6))
for course, color in course_colors.items():
    subset = df[df["Course"] == course]
    plt.scatter(
        subset["Marks"],
        subset["Entrance_Score"],
        color=color,
        label=course,
        s=80)
plt.xlabel("Marks")
plt.ylabel("Entrance Score")
plt.title("Linear Separability with Perceptron (OvR)")
plt.legend()
plt.grid(True)
plt.show()
new_student = np.array([[72, 75]])
scores = {}
for course, model in models.items():
    scores[course] = model.score(new_student)[0]
predicted_course = max(scores, key=scores.get)
print("New Student:", new_student)
print("Predicted Course:", predicted_course)
```

Output:





```
New Student: [[72 75]]
Predicted Course: Cyber Security
```

K. M. S. P Mandal’s	**Date: 13/09/2025**	
Sant Rawool Maharaj Mahavidyalaya, Kudal	**Roll No: 20**	
Department of Information Technology	**Expt. No: 06**	**Signature**

Title: Write a program for Kohonen Self organizing map and Adaptive resonance theory

Kohonen’ s Self Organizing Map:

Kohonen’s Self-Organizing Map (SOM) is an unsupervised neural network that groups similar data together. It maps high-dimensional data onto a 2D grid, preserving the relationships between data points. Each neuron in the grid has weights that adjust to match the input data. Similar inputs activate nearby neurons, forming clusters.

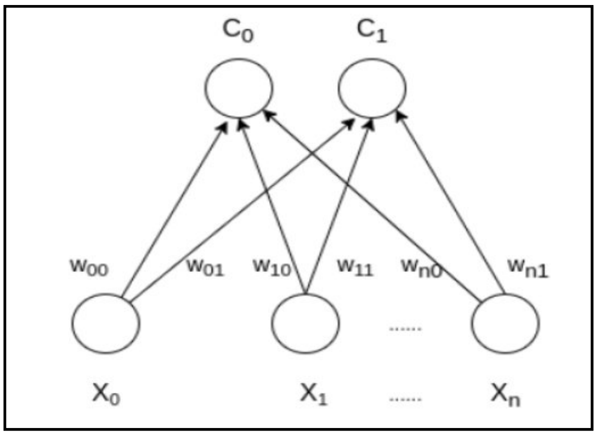
Formula for Best Matching Unit (BMU):
||X−Wi||=min_j(||X−Wj||)

1. X → input vector (the data point you want to train with).
2. Wi → weight vector of the i-th neuron.
3. ||X - Wi|| → distance between the input and neuron i (usually Euclidean distance).
4. min_j (||X - Wj||) → we look for the neuron whose weight is closest to X.

Weight update Formula:
Wi(t+1)=Wi(t)+η(t)·hci(t)·(X−Wi(t))

1. Wi(t) → current weight of neuron i at time t.
2. η(t) → learning rate (a small number that decreases over time). It controls how fast weights change.
3. hci(t) → neighborhood function. It determines how strongly neighboring neurons around the BMU are updated. Neurons closer to the BMU get updated more.
4. (X - Wi(t)) → direction and amount by which the weight should move toward the input X.

Analogy of Kohen’ s Self Orgainzing Map:



Adaptive Resonance Theory:

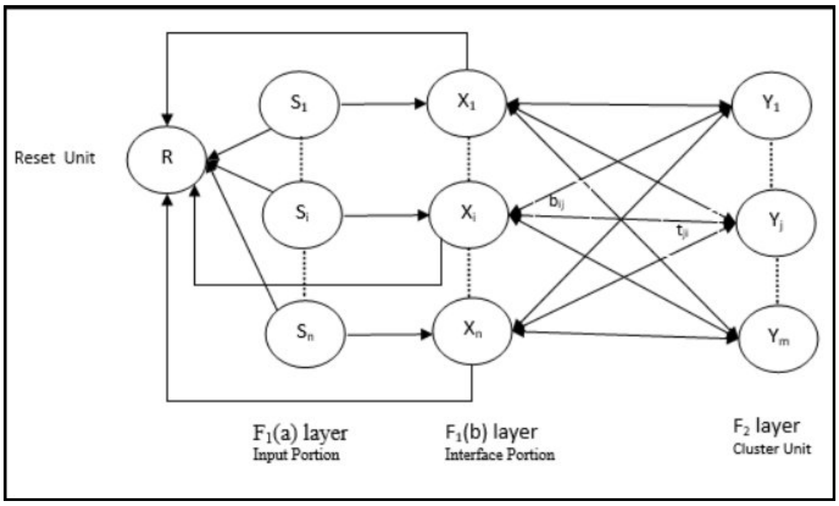
Adaptive Resonance Theory (ART) in Soft Computing is a neural network model used for pattern recognition and clustering. It learns new patterns without forgetting old ones. ART compares an input pattern with stored patterns; if the similarity is high enough (called the vigilance test), it matches and updates the category.

Formula used to calculate the output:

Wij(new) = Wij(old) * Xj / (α + |X|)

- W_{ij}^{old}: **Old weight between input node *j* and category (neuron) *i***
- X_j: **j-th input value (usually 0 or 1 in ART-1)**
- $|X|$: **Total number of active inputs (sum of all X_j)**
- α: **Small positive constant (prevents division by zero)**

Analogy of ART (Adaptive Resonance Theory):

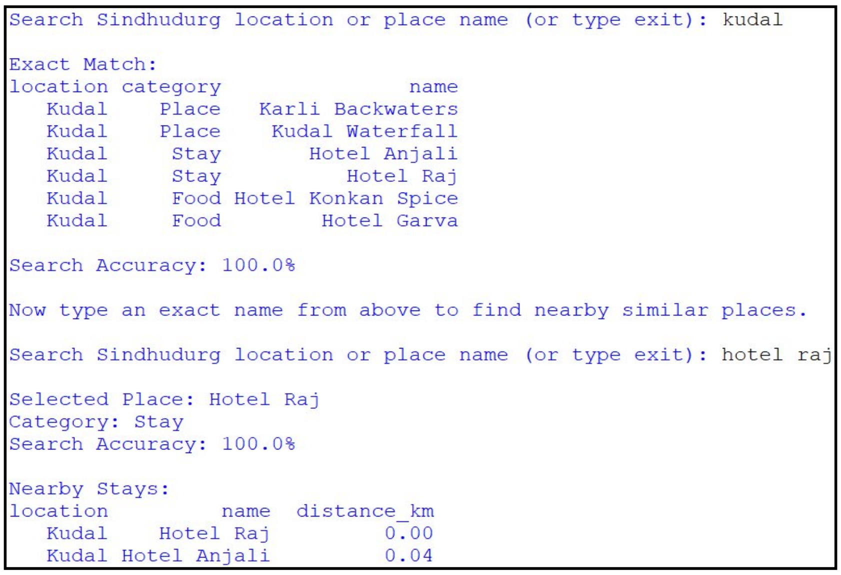


Kohonen’ s Self Organizing Map:

Code:

```
import pandas as pd
from math import radians, cos, sin, sqrt, atan2
import os

LOCATION_RADIUS_KM = 5
FILE = "sindhudurg_places-2.csv"
if not os.path.isfile(FILE):
    print(f"CSV file not found: {FILE}")
    exit(1)
for sep in [",", ";", "\t", "|"]:
    try:
        df = pd.read_csv(FILE, sep=sep)
        if df.shape[1] > 1:
            break
    except:
        pass
df.columns = (
    df.columns
    .str.strip()
    .str.lower()
    .str.replace('\ufeff', '', regex=False))
if len(df.columns) == 1:
    df = df[df.columns[0]].str.split(",", expand=True)
    df.columns = ["location", "category", "name", "info", "latitude", "longitude"]

df = df.rename(columns={
    "lat": "latitude",
    "lan": "latitude",
    "lon": "longitude",
    "log": "longitude"})

required = {"location", "category", "name", "latitude", "longitude"}
if not required.issubset(df.columns):
    print("CSV must contain location, category, name, latitude, longitude.")
    exit(1)
df["latitude"] = pd.to_numeric(df["latitude"], errors="coerce")
df["longitude"] = pd.to_numeric(df["longitude"], errors="coerce")
df = df.dropna(subset=["latitude", "longitude"])

if "info" not in df.columns:
    df["info"] = "Information not available."
print("CSV Loaded with", len(df), "records")
def haversine(lat1, lon1, lat2, lon2):
    R = 6371
    dlat = radians(lat2 - lat1)
    dlon = radians(lon2 - lon1)
    a = sin(dlat / 2)**2 + cos(radians(lat1)) * cos(radians(lat2)) * sin(dlon / 2)**2
    return 2 * R * atan2(sqrt(a), sqrt(1 - a))
def compute_accuracy(user_input, df, column):

    user_vec = user_input.lower().strip()
    df_vecs = df[column].str.lower().str.strip()
    distances = df_vecs.apply(lambda x: sum(1 for a, b in zip(x, user_vec) if a != b) +
abs(len(x) - len(user_vec)))
    min_dist = distances.min()
    max_len = max(df_vecs.str.len().max(), len(user_vec))
    similarity = 100 * (1 - min_dist / max_len)
    return round(similarity, 2)

def smart_search(user_input):
    q = user_input.lower().strip()
    location_data = df[df["location"].str.lower().str.strip() == q]
    if not location_data.empty:
        accuracy = compute_accuracy(user_input, df, "location")
        print("\nExact Match:")
        print(location_data[["location", "category", "name"]].to_string(index=False))
        print(f"\nSearch Accuracy: {accuracy}%")
        print("\nNow type an exact name from above to find nearby similar places.")
        return

    name_match = df[df["name"].str.lower().str.strip() == q]
    if not name_match.empty:
        selected = name_match.iloc[0]
        base_lat = selected["latitude"]
        base_lon = selected["longitude"]
        category = selected["category"]
        accuracy = compute_accuracy(user_input, df, "name")

        print(f"\nSelected Place: {selected['name']}")
        print(f"Category: {category}")
        print(f"Search Accuracy: {accuracy}%")

        df["distance_km"] = df.apply(
            lambda r: haversine(base_lat, base_lon, r["latitude"], r["longitude"]),
            axis=1)
        nearby = df[
            (df["category"].str.lower().str.strip() == category.lower().strip()) &
            (df["distance_km"] <= LOCATION_RADIUS_KM)
        ].sort_values("distance_km")
        if nearby.empty:
            print("\nNo nearby similar places found.")
        else:
            print(f"\nNearby {category}s:")
            print(nearby[["location", "name",
"distance_km"]].round(2).to_string(index=False))
        return
    print("\nNo matching place found.")
while True:
    user_input = input("\nSearch Sindhudurg location or place name (or type exit): ")
    if user_input.lower() == "exit":
        print("Exiting search.")
        break
    smart_search(user_input)
```

Output:



```
Search Sindhudurg location or place name (or type exit): kudal

Exact Match:
location category                name
   Kudal    Place   Karli Backwaters
   Kudal    Place    Kudal Waterfall
   Kudal     Stay       Hotel Anjali
   Kudal     Stay          Hotel Raj
   Kudal     Food Hotel Konkan Spice
   Kudal     Food        Hotel Garva

Search Accuracy: 100.0%

Now type an exact name from above to find nearby similar places.

Search Sindhudurg location or place name (or type exit): hotel raj

Selected Place: Hotel Raj
Category: Stay
Search Accuracy: 100.0%

Nearby Stays:
location         name  distance_km
   Kudal    Hotel Raj         0.00
   Kudal Hotel Anjali         0.04
```

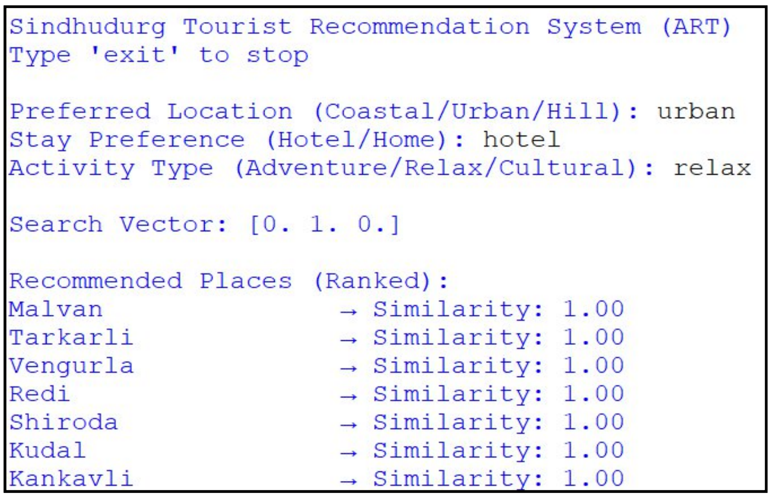
Adaptive Resonance Theory:

Code:

```
import pandas as pd
import numpy as np

df = pd.read_csv("sindhudurg_places3.csv")
def encode_place(row):
   location = 1 if row["Location_Type"].lower() == "coastal" else 0
   stay = 1 if row["Stay_Type"].lower() == "hotel" else 0
   activity = 1 if row["Activity_Type"].lower() == "adventure" else 0
   return np.array([location, stay, activity], dtype=float)
categories = []
place_names = []
for _, row in df.iterrows():
   categories.append(encode_place(row))
   place_names.append(row["Place"])

vigilance = 0.6     # ρ
alpha = 0.5         # learning constant
def similarity(x, w):
   return np.sum(x * w) / np.sum(x) if np.sum(x) != 0 else 0
def update_weight(x, w):
   return (w * x) / (alpha + np.sum(x))
def encode_user_input(location, stay, activity):
   loc = 1 if location.lower() == "coastal" else 0
   sty = 1 if stay.lower() == "hotel" else 0
   act = 1 if activity.lower() == "adventure" else 0
   return np.array([loc, sty, act], dtype=float)

print("\nSindhudurg Tourist Recommendation System (ART)")
print("Type 'exit' to stop\n")

while True:
   loc = input("Preferred Location (Coastal/Urban/Hill): ")
   if loc.lower() == "exit":
      break

   stay = input("Stay Preference (Hotel/Home): ")
   act = input("Activity Type (Adventure/Relax/Cultural): ")

   x = encode_user_input(loc, stay, act)
   print(f"\nSearch Vector: {x}")
   results = []
   for i, w in enumerate(categories):
      sim = similarity(x, w)

      if sim >= vigilance:
         results.append((place_names[i], sim))
```

```
            categories[i] = update_weight(x, w)  # ART learning
    if results:
        results.sort(key=lambda r: r[1], reverse=True)
        print("\nRecommended Places (Ranked):")
        for place, score in results:
            print(f"{place:<20} → Similarity: {score:.2f}")
    else:
        print("\nNo close match found. Learning new preference.")
        categories.append(x)
        place_names.append("New Preference Pattern")
    print("-" * 70)
```

Output:



```
Sindhudurg Tourist Recommendation System (ART)
Type 'exit' to stop

Preferred Location (Coastal/Urban/Hill): urban
Stay Preference (Hotel/Home): hotel
Activity Type (Adventure/Relax/Cultural): relax

Search Vector: [0. 1. 0.]

Recommended Places (Ranked):
Malvan               → Similarity: 1.00
Tarkarli             → Similarity: 1.00
Vengurla             → Similarity: 1.00
Redi                 → Similarity: 1.00
Shiroda              → Similarity: 1.00
Kudal                → Similarity: 1.00
Kankavli             → Similarity: 1.00
```

K. M. S. P Mandal's	**Date: 11/10/2025**	
Sant Rawool Maharaj Mahavidyalaya, Kudal	**Roll No: 20**	
Department of Information Technology	**Expt. No: 07**	**Signature**

Title: Write a program for Back Propagation Algorithm and Error Backpropagation algorithm.

Back Propagation Algorithm:

Backpropagation is a method used to train neural networks. It works by comparing the network's predicted output with the actual output to calculate an error. This error is then sent backward through the network to adjust the weights, so the predictions get closer to the correct values. The process repeats many times, gradually improving the network's accuracy. Essentially, it's like learning from mistakes and correcting them step by step.

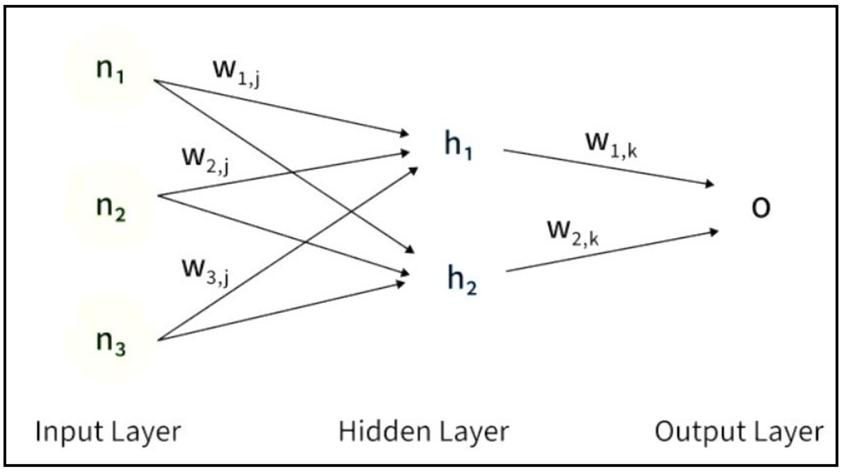
Formula used to calculate the error:

Calculating the weighted sum:

aj=∑(wi,j∗xi) + b

aj is the weighted sum of all the inputs and weights at each node
$w_{i,j}$ represents the weights between the ith input and the jth neuron
x_i represents the value of the ith input
B represents the bias value

Analogy of Backpropagation algorithm:



Error Backpropagation algorithm:

Backpropagation is a method used to train neural networks by adjusting the weights to reduce prediction errors. The error is the difference between what the network predicts and the actual target. During training, this error is calculated and sent backward through the network. Each weight is updated slightly, based on how much it contributed to the error, using the derivative of the activation function. By repeating this process over many iterations, the network gradually “learns” to make more accurate predictions. Essentially, the network is learning from its mistakes step by step.

Formula used to calculate the output:

$$\mathbf{Error_j = y_{target} - y_j}$$

$Error_j$ → This is the error term for the j-th neuron (or output unit)
y_{target}→ The desired output (what the neuron should have predicted).
y_j→ The output produced by the neuron (what the neuron actually predicted).

Weight update formula:

$$\mathbf{w_{ij} = \eta * \delta_j * O_j}$$

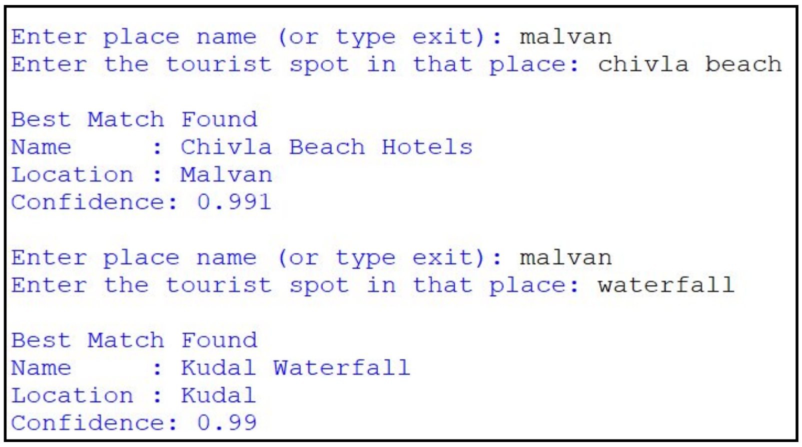
δ_j is the error term for each unit,
η is the learning rate.

Back Propagation Algorithm:

Code:

```
import pandas as pd
import numpy as np
import difflib
import math
FILE = "sindhudurg_places.csv"
df = pd.read_csv(FILE)
X = []
Y = []
for _, row in df.iterrows():
    name = str(row["name"]).lower()
    location = str(row["location"]).lower()
    x1 = len(name) / 20
    x2 = difflib.SequenceMatcher(None, name, location).ratio()
    X.append([x1, x2])
    Y.append(1 if x2 >= 0.6 else 0)
X = np.array(X)
Y = np.array(Y)
w1 = np.random.rand()
w2 = np.random.rand()
b = np.random.rand()
learning_rate = 0.1
epochs = 1200
def sigmoid(a):
    return 1 / (1 + math.exp(-a))
for _ in range(epochs):
    for i in range(len(X)):
        x1_i, x2_i = X[i]
        target = Y[i]

        a = (w1 * x1_i) + (w2 * x2_i) + b
        y = sigmoid(a)
        delta = (target - y) * y * (1 - y)
        w1 += learning_rate * delta * x1_i
        w2 += learning_rate * delta * x2_i
        b  += learning_rate * delta
def search_place(user_input):
    user_input = user_input.lower()
    best_match = None
    best_score = -1

    for _, row in df.iterrows():
        place_name = str(row["name"]).lower()
        place_location = str(row["location"]).lower()
        x1 = len(place_name) / 20
        x2 = difflib.SequenceMatcher(None, user_input, place_name).ratio()
        if user_input in place_name:
            x2 += 0.2
        x2 = min(x2, 1.0)
        a = (w1 * x1) + (w2 * x2) + b
        y = sigmoid(a)
        if y > best_score:
            best_score = y
            best_match = row
    return best_match, best_score
while True:
    query = input("\nEnter place name (or type exit): ")
    if query.lower() == "exit":
        break
    target_place = input("Enter the tourist spot in that place: ").lower()
    x1_user = len(target_place) / 20
    x2_user = difflib.SequenceMatcher(None, query.lower(), target_place).ratio()
    a = (w1 * x1_user) + (w2 * x2_user) + b
    y = sigmoid(a)
  threshold = 0.6
    target_label = 1 if x2_user >= threshold else 0
    if target_label == 0:
        delta = (target_label - y) * y * (1 - y)
        w1 += learning_rate * delta * x1_user
        w2 += learning_rate * delta * x2_user
        b  += learning_rate * delta
        a_updated = (w1 * x1_user) + (w2 * x2_user) + b
        y_updated = sigmoid(a_updated)
        print(f"Updated Confidence: {round(y_updated, 3)}")
    best_match, best_score = search_place(target_place)
    print("\nBest Match Found")
    print("Name     :", best_match["name"])
    print("Location :", best_match["location"])
    print("Confidence:", round(best_score, 3))
```

Output:



```
Enter place name (or type exit): malvan
Enter the tourist spot in that place: chivla beach

Best Match Found
Name      : Chivla Beach Hotels
Location : Malvan
Confidence: 0.991

Enter place name (or type exit): malvan
Enter the tourist spot in that place: waterfall

Best Match Found
Name      : Kudal Waterfall
Location : Kudal
Confidence: 0.99
```

Error Backpropagation algorithm:

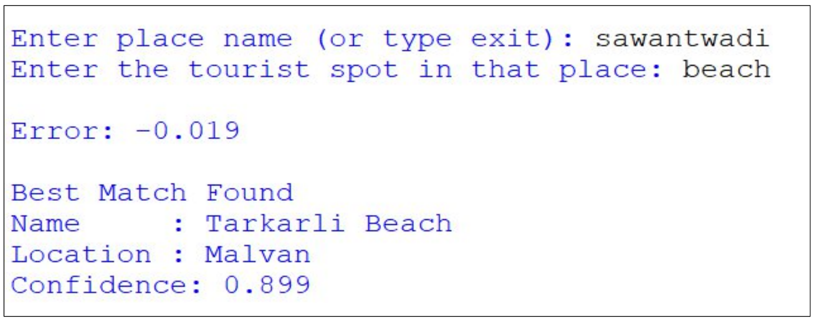
Code:

```
import pandas as pd
import numpy as np
import difflib
import math
FILE = "sindhudurg_places.csv"
df = pd.read_csv(FILE)

X = []
Y = []

for _, row in df.iterrows():
   name = str(row["name"]).lower()
   location = str(row["location"]).lower()
   x1 = len(name) / 20
   x2 = difflib.SequenceMatcher(None, name, location).ratio()
   X.append([x1, x2])
   Y.append(1 if x2 >= 0.6 else 0)

X = np.array(X)
Y = np.array(Y)
w1 = np.random.rand()
w2 = np.random.rand()
b = np.random.rand()

learning_rate = 0.1
epochs = 1200
def sigmoid(a):
   return 1 / (1 + math.exp(-a))
def sigmoid_derivative(y):
   return y * (1 - y)
for _ in range(epochs):
   for i in range(len(X)):
      x1_i, x2_i = X[i]
      target = Y[i]
      a = (w1 * x1_i) + (w2 * x2_i) + b
      y = sigmoid(a)
      error = target - y
      delta = error * sigmoid_derivative(y)  # δj = Error * derivative of output
      w1 += learning_rate * delta * x1_i
      w2 += learning_rate * delta * x2_i
      b  += learning_rate * delta
```

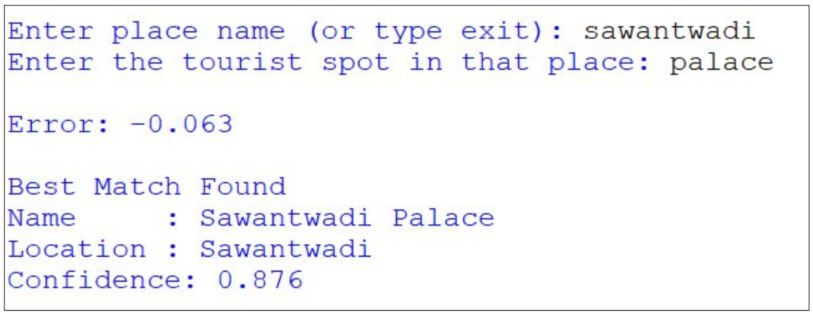
```
def search_place(user_input):
    user_input = user_input.lower()
    best_match = None
    best_score = -1

    for _, row in df.iterrows():
        place_name = str(row["name"]).lower()
        place_location = str(row["location"]).lower()
        x1 = len(place_name) / 20
        x2 = difflib.SequenceMatcher(None, user_input, place_name).ratio()
        if user_input in place_name:
            x2 += 0.2
        x2 = min(x2, 1.0)
        a = (w1 * x1) + (w2 * x2) + b
        y = sigmoid(a)
        if y > best_score:
            best_score = y
            best_match = row
    return best_match, best_score
while True:
    query = input("\nEnter place name (or type exit): ")
    if query.lower() == "exit":
        break

    target_place = input("Enter the tourist spot in that place: ").lower()
    x1_user = len(target_place) / 20
    x2_user = difflib.SequenceMatcher(None, query.lower(), target_place).ratio()
    a = (w1 * x1_user) + (w2 * x2_user) + b
    y = sigmoid(a)
    threshold = 0.6
    target_label = 1 if x2_user >= threshold else 0
    error = target_label - y
    delta = error * sigmoid_derivative(y)

    print(f"\nError: {round(error, 3)}")
    print(f"Initial Confidence: {round(y, 3)}")

    if target_label == 0:  # mismatch detected
        w1 += learning_rate * delta * x1_user
        w2 += learning_rate * delta * x2_user
        b  += learning_rate * delta
        a_updated = (w1 * x1_user) + (w2 * x2_user) + b
        y_updated = sigmoid(a_updated)
        print(f"Updated Confidence: {round(y_updated, 3)}")
```

```
best_match, best_score = search_place(target_place)
print("\nBest Match Found")
print("Name     :", best_match["name"])
print("Location :", best_match["location"])
print("Confidence:", round(best_score, 3))
```

Output:



```
Enter place name (or type exit): sawantwadi
Enter the tourist spot in that place: beach

Error: -0.019

Best Match Found
Name     : Tarkarli Beach
Location : Malvan
Confidence: 0.899
```



```
Enter place name (or type exit): sawantwadi
Enter the tourist spot in that place: palace

Error: -0.063

Best Match Found
Name     : Sawantwadi Palace
Location : Sawantwadi
Confidence: 0.876
```

K. M. S. P Mandal's **Sant Rawool Maharaj Mahavidyalaya, Kudal** **Department of Information Technology**	**Date: 10/11/2025** **Roll No: 20** **Expt. No: 08**	**Signature**
Title: Finding ratios using fuzzy logic and solving Tipping problem using fuzzy logic		

Fuzzy Logic:

Fuzzy Logic is a technique in soft computing that works with uncertain and vague information. Instead of using only true or false, it allows partial values between 0 and 1. It works like human thinking by using IF–THEN rules to make decisions. Fuzzy logic is useful when exact data is not available or is hard to define.

Applications of Fuzzy Logic:

1. **Home appliances – washing machines, air conditioners, refrigerators**
2. **Traffic control systems – managing traffic lights and congestion**
3. **Medical diagnosis – disease detection based on symptoms**
4. **Industrial control systems – speed, pressure, and temperature control**
5. **Decision support systems – recommendations and expert systems**
6. **Pattern recognition – image and speech recognition**
7. **Tourism & business analysis – popularity ranking, customer satisfaction analysis**

Fuzzy Logic System Architecture:

1. **Fuzzification – Converts crisp inputs into fuzzy sets.**
2. **Inference Engine – Applies fuzzy rules to inputs.**
3. **Defuzzification – Converts fuzzy outputs back to crisp values.**

Tipping Problem using Fuzzy Logic:

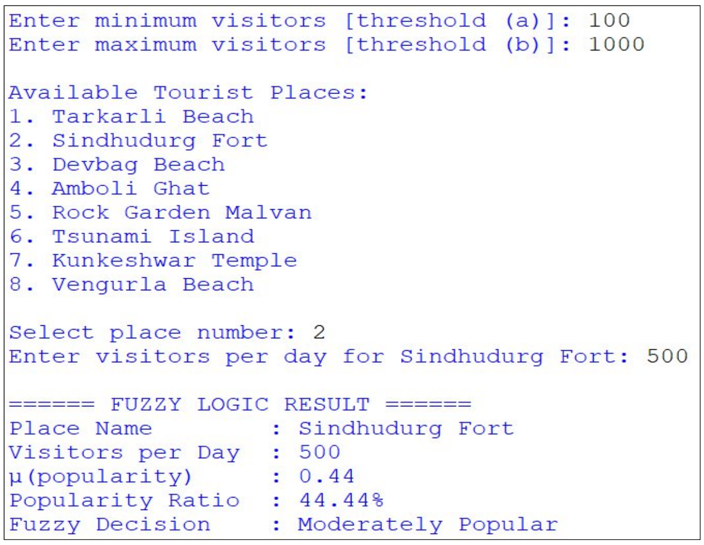
The tipping problem uses fuzzy logic to decide how much tip to give in a restaurant. The decision depends on service quality and food quality, which are not exact values. Fuzzy logic uses simple IF–THEN rules like “IF service is good, THEN tip is high.” Each input is given a degree between 0 and 1 instead of yes or no.
This helps in making decisions similar to human thinking.

Fuzzy Logic:

Code:

```
import pandas as pd

a = int(input("Enter minimum visitors [threshold (a)]: "))
b = int(input("Enter maximum visitors [threshold (b)]: "))

def popularity_membership(x, a, b):
    if x <= a:
        return 0.0
    elif x >= b:
        return 1.0
    else:
        return (x - a) / (b - a)

df = pd.read_csv("sindhudurg_places5.csv", encoding="utf-8-sig")
df.columns = df.columns.str.strip()

print("\nAvailable Tourist Places:")
for i, place in enumerate(df["Place"], start=1):
    print(f"{i}. {place}")

choice = int(input("\nSelect place number: ")) - 1
selected_place = df.loc[choice, "Place"]

visitors = int(input(f"Enter visitors per day for {selected_place}: "))

mu = popularity_membership(visitors, a, b)
ratio = mu * 100

if mu == 0:
    status = "Not Popular"
elif mu < 0.4:
    status = "Low Popularity"
elif mu < 0.7:
    status = "Moderately Popular"
else:
    status = "Highly Popular"

print("\n====== FUZZY LOGIC RESULT ======")
print(f"Place Name        : {selected_place}")
print(f"Visitors per Day  : {visitors}")
print(f"μ(popularity)     : {mu:.2f}")
print(f"Popularity Ratio  : {ratio:.2f}%")
print(f"Fuzzy Decision    : {status}")
```

Output:



```
Enter minimum visitors [threshold (a)]: 100
Enter maximum visitors [threshold (b)]: 1000

Available Tourist Places:
1. Tarkarli Beach
2. Sindhudurg Fort
3. Devbag Beach
4. Amboli Ghat
5. Rock Garden Malvan
6. Tsunami Island
7. Kunkeshwar Temple
8. Vengurla Beach

Select place number: 2
Enter visitors per day for Sindhudurg Fort: 500

====== FUZZY LOGIC RESULT ======
Place Name          : Sindhudurg Fort
Visitors per Day    : 500
μ(popularity)       : 0.44
Popularity Ratio    : 44.44%
Fuzzy Decision      : Moderately Popular
```

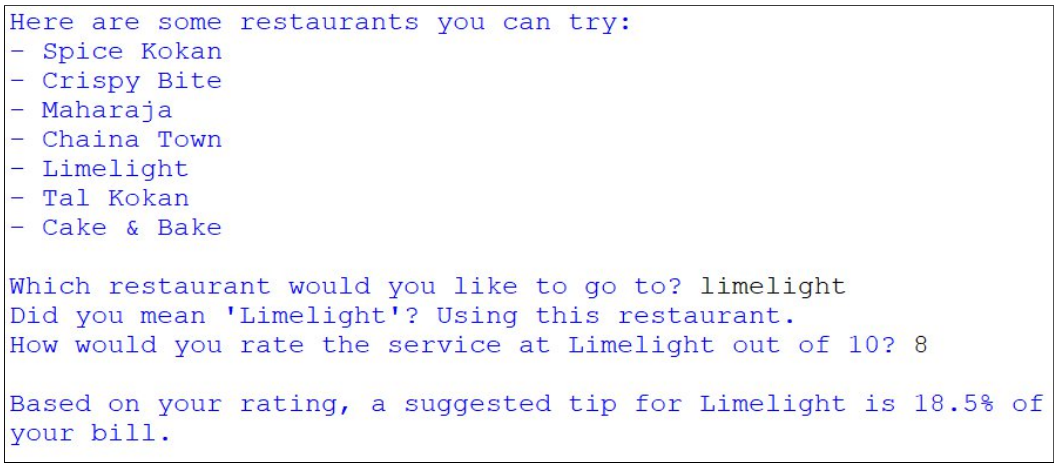
Tipping problem by using Fuzzy Logic:

Code:

```
import pandas as pd
from difflib import get_close_matches

restaurants = {
   "Spice Kokan": {"Food": 3, "Service": 2},
   "Crispy Bite": {"Food": 7, "Service": 6},
   "Maharaja": {"Food": 9, "Service": 8},
   "Chaina Town": {"Food": 8, "Service": 7},
   "Limelight": {"Food": 9, "Service": 9},
   "Tal Kokan": {"Food": 8, "Service": 6},
   "Cake & Bake ": {"Food": 7, "Service": 8}
}

print("Here are some restaurants you can try:")
for name in restaurants.keys():
   print("-", name)

user_choice = input("\nWhich restaurant would you like to go to? ").strip()
matches = get_close_matches(user_choice, restaurants.keys(), n=1, cutoff=0.6)

if matches:
   chosen_restaurant = matches[0]
   print(f"Did you mean '{chosen_restaurant}'? Using this restaurant.")

   service_rating = float(input(f"How would you rate the service at {chosen_restaurant}
out of 10? "))

   food_rating = restaurants[chosen_restaurant]["Food"]
   tip_percent = 10 + 0.5 * food_rating + 0.5 * service_rating
   tip_percent = round(tip_percent, 2)

   print(f"\nBased on your rating, a suggested tip for {chosen_restaurant} is
{tip_percent}% of your bill.")
else:
   print("Sorry, we could not find a restaurant matching your input.")

data = []
for name, ratings in restaurants.items():
   for user_service in range(5, 11):  # ratings 5 to 10
      tip = 10 + 0.5 * ratings["Food"] + 0.5 * user_service
      data.append([name, ratings["Food"], user_service, round(tip, 2)])
```

```
df = pd.DataFrame(data, columns=["Restaurant", "Food_Rating", "Service_Rating",
"Suggested_Tip"])
df.to_csv("restaurant_tips.csv", index=False)
print("\nA CSV file 'restaurant_tips.csv' has been created with all restaurants and
calculated tips.")
```

Output:



```
Here are some restaurants you can try:
- Spice Kokan
- Crispy Bite
- Maharaja
- Chaina Town
- Limelight
- Tal Kokan
- Cake & Bake

Which restaurant would you like to go to? limelight
Did you mean 'Limelight'? Using this restaurant.
How would you rate the service at Limelight out of 10? 8

Based on your rating, a suggested tip for Limelight is 18.5% of
your bill.
```

K. M. S. P Mandal's	**Date: 26/11/2025**	
Sant Rawool Maharaj Mahavidyalaya, Kudal	**Roll No: 20**	
Department of Information Technology	**Expt. No: 09**	**Signature**

Title: Implementation of Simple genetic algorithm (Creating two classes: City and Fitness using Genetic algorithm)

Genetic Algorithm:

A Genetic Algorithm is a soft computing technique inspired by natural evolution. It works by creating many possible solutions and improving them over generations. The best solutions are selected and combined using selection, crossover, and mutation. Over time, weak solutions are removed and better ones survive. Genetic algorithms are used to solve optimization and search problems.

Steps in Genetic Algorithm:

1. **Initialization – Generate random population.**
2. **Selection – Choose best candidates based on fitness.**
3. **Crossover – Combine parts of two solutions.**
4. **Mutation – Introduce random changes for diversity.**
5. **Repeat until optimal solution is found.**

Creating City and Fitness class for the application of Genetic Algorithm:

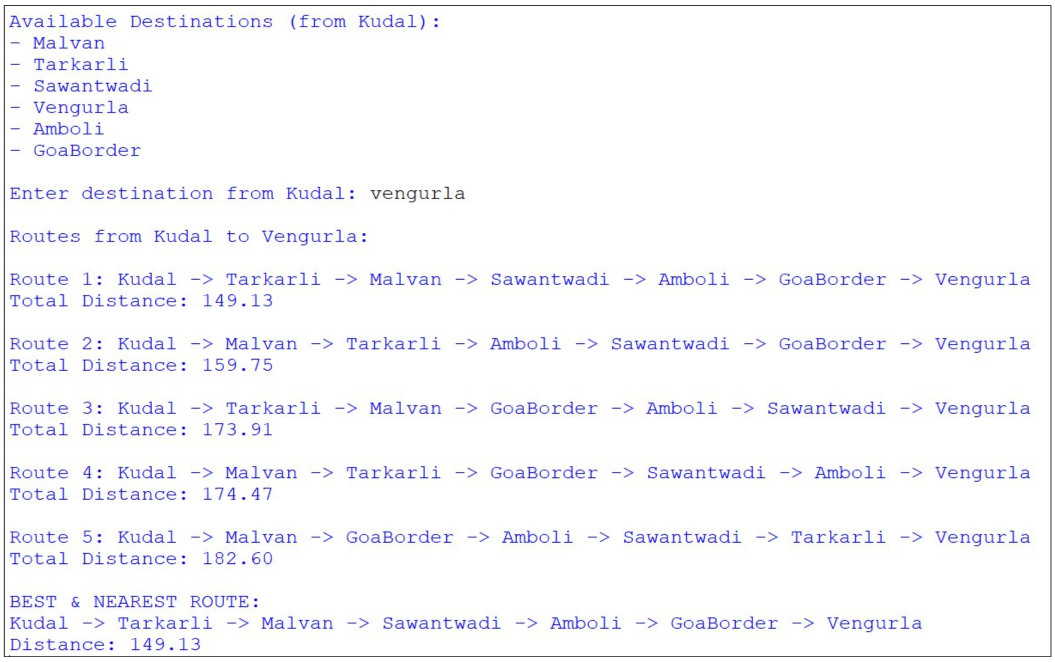
The program uses a Genetic Algorithm to find the shortest travel route from Kudal to a selected destination. It treats each place as a city with coordinates and calculates distances between them. The algorithm creates many random routes, then improves them using crossover and mutation. After several generations, weak routes are removed and better routes survive. Finally, it shows the best and nearest route with the minimum distance.

Code:

```
import random
import math
import csv

class City:
    def __init__(self, name, x, y):
        self.name = name
        self.x = x
        self.y = y

    def distance(self, city):
        return math.sqrt((self.x - city.x) ** 2 + (self.y - city.y) ** 2)
    def __repr__(self):
        return self.name
class Fitness:
    def __init__(self, route):
        self.route = route

    def route_distance(self):
        return sum(
            self.route[i].distance(self.route[i + 1])
            for i in range(len(self.route) - 1))

def load_cities(filename):
    cities = {}
    with open(filename, "r") as file:
        reader = csv.DictReader(file)
        for row in reader:
            cities[row["Place"]] = City(
                row["Place"], float(row["X"]), float(row["Y"]))
    return cities

def create_route(start, destination, mid_cities):
    return [start] + random.sample(mid_cities, len(mid_cities)) + [destination]


def initial_population(size, start, destination, mid_cities):
    return [create_route(start, destination, mid_cities) for _ in range(size)]

def crossover(parent1, parent2, start, destination):
    mid1 = parent1[1:-1]
    mid2 = parent2[1:-1]

    a, b = sorted(random.sample(range(len(mid1)), 2))
    child_mid = mid1[a:b]

    for city in mid2:
        if city not in child_mid:
            child_mid.append(city)

    return [start] + child_mid + [destination]
def mutate(route, rate=0.15):
    mid = route[1:-1]
    for i in range(len(mid)):
        if random.random() < rate:
            j = random.randint(0, len(mid) - 1)
            mid[i], mid[j] = mid[j], mid[i]
    return [route[0]] + mid + [route[-1]]
def genetic_algorithm(start, destination, mid_cities,
                      pop_size=60, generations=200, required_routes=5):

    population = initial_population(pop_size, start, destination, mid_cities)
    for _ in range(generations):
        new_population = []
        for _ in range(pop_size):
            p1 = random.choice(population)
            p2 = random.choice(population)
            child = crossover(p1, p2, start, destination)
            child = mutate(child)
            new_population.append(child)
        population = new_population
    unique_routes = {}
    for route in population:
        key = tuple(city.name for city in route)
        dist = Fitness(route).route_distance()
        if key not in unique_routes:
            unique_routes[key] = (route, dist)
    sorted_routes = sorted(unique_routes.values(), key=lambda x: x[1])

    return sorted_routes[:required_routes]
cities = load_cities("sindhudurg_places6.csv")

start_city = cities["Kudal"]
city_lookup = {
    name.lower(): name
    for name in cities
    if name != "Kudal"
}
print("\nAvailable Destinations (from Kudal):")
for place in city_lookup.values():
    print("-", place)
destination_input = input("\nEnter destination from Kudal: ").strip().lower()
if destination_input not in city_lookup:
    print("Invalid destination! Please choose from the list above.")
    exit()
destination_name = city_lookup[destination_input]
destination_city = cities[destination_name]
```

```
mid_cities = [
    city for name, city in cities.items()
    if name not in ["Kudal", destination_name]]
print(f"\nRoutes from Kudal to {destination_name}:\n")

routes = genetic_algorithm(start_city, destination_city, mid_cities)
for i, (route, dist) in enumerate(routes, start=1):
    print(f"Route {i}: {' -> '.join(city.name for city in route)}")
    print(f"Total Distance: {dist:.2f}\n")
best_route, best_dist = routes[0]
print("BEST & NEAREST ROUTE:")
print(" -> ".join(city.name for city in best_route))
print(f"Distance: {best_dist:.2f}")
```

Output:



```
Available Destinations (from Kudal):
- Malvan
- Tarkarli
- Sawantwadi
- Vengurla
- Amboli
- GoaBorder

Enter destination from Kudal: vengurla

Routes from Kudal to Vengurla:

Route 1: Kudal -> Tarkarli -> Malvan -> Sawantwadi -> Amboli -> GoaBorder -> Vengurla
Total Distance: 149.13

Route 2: Kudal -> Malvan -> Tarkarli -> Amboli -> Sawantwadi -> GoaBorder -> Vengurla
Total Distance: 159.75

Route 3: Kudal -> Tarkarli -> Malvan -> GoaBorder -> Amboli -> Sawantwadi -> Vengurla
Total Distance: 173.91

Route 4: Kudal -> Malvan -> Tarkarli -> GoaBorder -> Sawantwadi -> Amboli -> Vengurla
Total Distance: 174.47

Route 5: Kudal -> Malvan -> GoaBorder -> Amboli -> Sawantwadi -> Tarkarli -> Vengurla
Total Distance: 182.60

BEST & NEAREST ROUTE:
Kudal -> Tarkarli -> Malvan -> Sawantwadi -> Amboli -> GoaBorder -> Vengurla
Distance: 149.13
```

K. M. S. P Mandal's **Sant Rawool Maharaj Mahavidyalaya, Kudal** **Department of Information Technology**	**Date: 17/12/2025** **Roll No: 20** **Expt. No: 10**	**Signature**

Title: Program of implementation Membership and Identity Operators is, is not, in, not in

Membership Operators:

Membership operators are used in fuzzy logic to show how much an element belongs to a fuzzy set. The value lies between 0 and 1, not just true or false. A value close to 1 means strong membership, while close to 0 means weak membership. They help represent vague terms like hot, cold, tall, or short.

Identity Operators:

Identity operators are used to check whether two values or fuzzy sets are the same. If both values are exactly equal, the result is true (1). If they are different, the result is false (0). They help in comparing and validating fuzzy data in a system.

IN
Used to check whether an element belongs to a set or group. It returns true if the element is part of that set.

NOT IN
Used to check whether an element does not belong to a set. It returns true when the element is absent from the set.

IS
Used to check whether two values are exactly the same. It confirms equality between them.

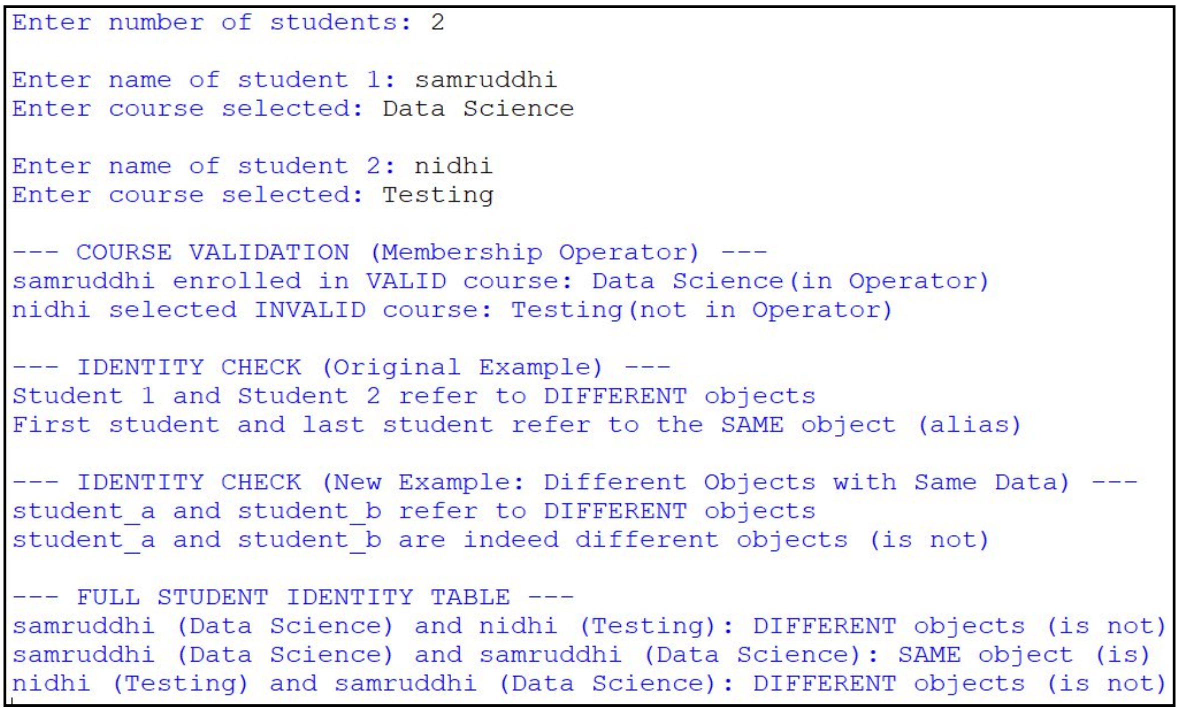
IS NOT
Used to check whether two values are different. It confirms inequality between them.

Code:

```
class Student:
    def __init__(self, name, course):
        self.name = name
        self.course = course

    def __repr__(self):
        return f"{self.name} ({self.course})"

available_courses = [
    "Data Science", "Artificial Intelligence", "Machine Learning",
    "Cyber Security", "Cloud Computing", "Blockchain",
    "Internet of Things", "Big Data Analytics",
    "Python Programming", "Web Development"
]

students = []
n = int(input("Enter number of students: "))

for i in range(n):
    name = input(f"\nEnter name of student {i+1}: ")
    course = input("Enter course selected: ")
    students.append(Student(name, course))

print("\n--- COURSE VALIDATION (Membership Operator) ---")
for student in students:
    if student.course in available_courses:
        print(f"{student.name} enrolled in VALID course: {student.course}(in Operator)")
    else:
        print(f"{student.name} selected INVALID course: {student.course}(not in
Operator)")
print("\n--- IDENTITY CHECK (Original Example) ---")

if len(students) >= 2:
    if students[0] is students[1]:
        print("Student 1 and Student 2 refer to the SAME object")
    else:
        print("Student 1 and Student 2 refer to DIFFERENT objects")
students.append(students[0])
if students[0] is students[-1]:
    print("First student and last student refer to the SAME object (alias)")
print("\n--- IDENTITY CHECK (New Example: Different Objects with Same Data) ---")
student_a = Student("Alice", "Python Programming")
student_b = Student("Alice", "Python Programming")

if student_a is student_b:
    print("student_a and student_b refer to the SAME object")
else:
    print("student_a and student_b refer to DIFFERENT objects")
```

```
if student_a is not student_b:
    print("student_a and student_b are indeed different objects (is not)")
print("\n--- FULL STUDENT IDENTITY TABLE ---")
for i in range(len(students)):
    for j in range(i + 1, len(students)):
        if students[i] is students[j]:
            print(f"{students[i]} and {students[j]}: SAME object (is)")
        else:
            print(f"{students[i]} and {students[j]}: DIFFERENT objects (is not)")
```

Output:



```
Enter number of students: 2

Enter name of student 1: samruddhi
Enter course selected: Data Science

Enter name of student 2: nidhi
Enter course selected: Testing

--- COURSE VALIDATION (Membership Operator) ---
samruddhi enrolled in VALID course: Data Science(in Operator)
nidhi selected INVALID course: Testing(not in Operator)

--- IDENTITY CHECK (Original Example) ---
Student 1 and Student 2 refer to DIFFERENT objects
First student and last student refer to the SAME object (alias)

--- IDENTITY CHECK (New Example: Different Objects with Same Data) ---
student_a and student_b refer to DIFFERENT objects
student_a and student_b are indeed different objects (is not)

--- FULL STUDENT IDENTITY TABLE ---
samruddhi (Data Science) and nidhi (Testing): DIFFERENT objects (is not)
samruddhi (Data Science) and samruddhi (Data Science): SAME object (is)
nidhi (Testing) and samruddhi (Data Science): DIFFERENT objects (is not)
```