University of Mumbai

Practical Journal of Advanced Artificial Intelligence & Machine Learning

M.Sc. (Information Technology) Part-II

Submitted by
SINGH MANASI RAKESH

Seat No: 1313421

DEPARTMENT OF INFORMATION TECHNOLOGY
PILLAI HOC COLLEGE OF ARTS, SCIENCE &COMMERCE, RASAYANI
(Affiliated to Mumbai University)
RASAYANI, 410207
MAHARASHTRA
2024-2025

University of Mumbai

Practical Journal of
Advanced Artificial Intelligence

M.Sc. (Information Technology)
Part-II

Submitted by
SINGH MANASI RAKESH

Seat No: 1313421

DEPARTMENT OF INFORMATION TECHNOLOGY
PILLAI HOC COLLEGE OF ARTS, SCIENCE &COMMERCE, RASAYANI
(Affiliated to Mumbai University)
RASAYANI, 410207
MAHARASHTRA
2024-2025

Mahatma Education Society's
Pillai Hoc College of Arts, Science & Commerce, Rasayani
(Affiliated to Mumbai University)
RASAYANI – MAHARASHTRA - 410207

DEPARTMENT OF INFORMATION TECHNOLOGY

CERTIFICATE

This is to certify that the experiment work entered in this journal is as per the syllabus in **M.Sc. (Information Technology) Part-II, Semester-III**; class prescribed by University of Mumbai for the subject **Advanced Artificial Intelligence** was done in computer lab of Mahatma Education Society's Pillai HOC College of Arts, Science & Commerce, Rasayani by **MANASI SINGH** during Academic year 2024-2025.

Exam Seat No: 1313421

Subject Incharge **Coordinator**

External Examiner **Principal**

Date: **College Seal**

ADVANCED ARTIFICIAL INTELLIGENCE

INDEX

Sr.No.	Practicals	Date	Sign
1	Implementing advanced deep learning algorithms such as convolutional neural networks (CNNs) or recurrent neural networks (RNNs) using Python libraries like TensorFlow or PyTorch.		
2	Building a natural language processing (NLP) model for sentiment analysis or text classification.		
3	Creating a chatbot using advanced techniques like transformer models.		
4	Developing a recommendation system using collaborative filtering or deep learning approaches.		
5	Implementing a computer vision project, such as object detection or image segmentation.		
6	Applying reinforcement learning algorithms to solve complex decision-making problems.		
7	Building a deep learning model for time series forecasting or anomaly detection.		
8	Implementing a machine learning pipeline for automated feature engineering and model selection.		
9	Using advanced optimization techniques like evolutionary algorithms or Bayesian optimization for hyperparameter tuning.		
10	Deploying a machine learning model in a production environment using containerization and cloud services.		

PRACTICAL NO – 1

Aim - Implementing advanced deep learning algorithms such as convolutional neural networks (CNNs) or recurrent neural networks (RNNs) using Python libraries like TensorFlow or PyTorch.

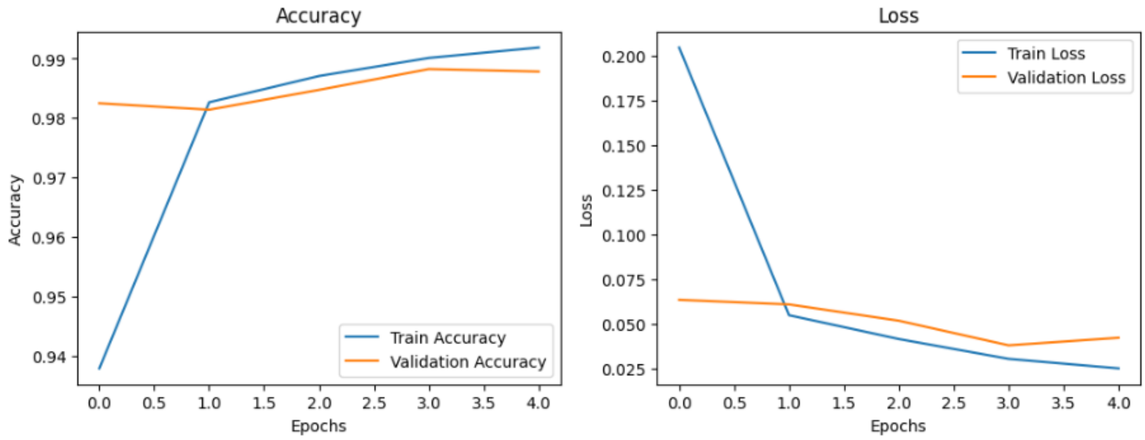
Code-

```
# Import necessary libraries
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt

# Load and preprocess the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape((x_train.shape[0], 28, 28, 1)).astype("float32") / 255.0
x_test = x_test.reshape((x_test.shape[0], 28, 28, 1)).astype("float32") / 255.0

# One-hot encode the labels
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)

# Build the CNN model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.Flatten(),
```

```
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
history = model.fit(x_train, y_train, epochs=5, batch_size=64, validation_split=0.2)

# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {test_acc:.4f}")

# Plot training and validation accuracy and loss
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
```

```
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

Output –



PRACTICAL NO – 2

Aim - Building a natural language processing (NLP) model for sentiment analysis or text classification.

Code-

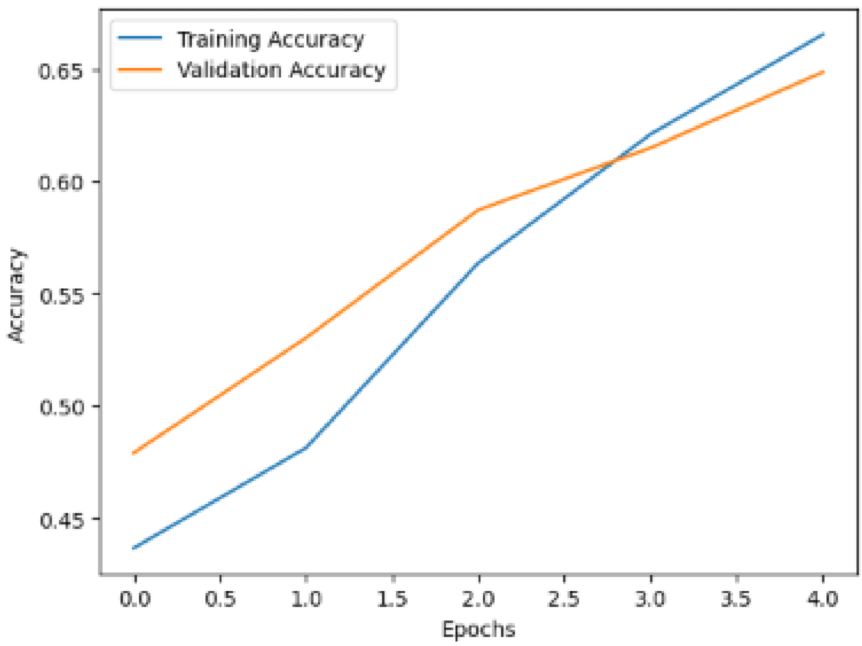
```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import reuters
from tensorflow.keras.preprocessing.sequence import pad_sequences
import matplotlib.pyplot as plt

# Custom callback for cleaner logs
class ClearLogsCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs=None):
        print(f"Epoch {epoch + 1}: Accuracy = {logs['accuracy']:.4f}, Loss = {logs['loss']:.4f}")

# Load the Reuters dataset
vocab_size = 10000  # Top 10,000 words
maxlen = 200  # Maximum sequence length

(train_data, train_labels), (test_data, test_labels) = reuters.load_data(num_words=vocab_size)

# Pad sequences to ensure uniform input length
train_data = pad_sequences(train_data, maxlen=maxlen)
test_data = pad_sequences(test_data, maxlen=maxlen)

# Convert labels to one-hot encoding
num_classes = max(train_labels) + 1
train_labels = tf.keras.utils.to_categorical(train_labels, num_classes)
```

```
test_labels = tf.keras.utils.to_categorical(test_labels, num_classes)

# Build the model
model = models.Sequential()

# Add an embedding layer
model.add(layers.Embedding(input_dim=vocab_size, output_dim=128))

# Add an LSTM layer
model.add(layers.LSTM(64))

# Add dense layers for classification
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(num_classes, activation='softmax'))  # Multi-class classification

# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Train the model with custom callback
history = model.fit(
    train_data,
    train_labels,
    epochs=5,
    batch_size=64,
    validation_data=(test_data, test_labels),
    verbose=0,  # Suppress dynamic progress bar
    callbacks=[ClearLogsCallback()]  # Use custom callback for logs
```

```
)

# Evaluate the model on the test data
test_loss, test_acc = model.evaluate(test_data, test_labels, verbose=0)
print(f"\nTest accuracy: {test_acc}")

# Plot training and validation accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

Output –

```
Epoch 1: Accuracy = 0.4367, Loss = 2.3768
Epoch 2: Accuracy = 0.4813, Loss = 2.0012
Epoch 3: Accuracy = 0.5637, Loss = 1.6856
Epoch 4: Accuracy = 0.6210, Loss = 1.4686
Epoch 5: Accuracy = 0.6654, Loss = 1.2886

Test accuracy: 0.6487088203430176
```



PRACTICAL NO – 3

Aim - Creating a chatbot using advanced techniques like transformer models.

Code-

```
from transformers import GPT2LMHeadModel, GPT2Tokenizer
import os

# Suppress TensorFlow logs
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

# Suppress Hugging Face symlink warnings
os.environ["HF_HUB_DISABLE_SYMLINKS_WARNING"] = "1"

def chatbot():
    # Load pre-trained GPT-2 model and tokenizer
    print("Loading model...")
    model_name = "gpt2"  # You can replace this with 'gpt2-medium' or another variant

    # Load the tokenizer and model
    tokenizer = GPT2Tokenizer.from_pretrained(model_name)
    model = GPT2LMHeadModel.from_pretrained(model_name)

    # Set a padding token (GPT-2 doesn’t have one by default)
    tokenizer.pad_token = tokenizer.eos_token

    print("Model loaded successfully!")

    # Instructions
```

```
    print("\nChatbot ready! Type 'exit' to quit.\n")

    # Chat loop
    while True:
        user_input = input("You: ")
        if user_input.lower() == "exit":
            print("Chatbot: Goodbye!")
            break

        # Tokenize and generate a response with attention mask
        inputs = tokenizer(user_input, return_tensors="pt", padding=True, truncation=True)
        input_ids = inputs["input_ids"]
        attention_mask = inputs["attention_mask"]

        # Generate the response
        output = model.generate(
            input_ids,
            attention_mask=attention_mask,  # Include attention mask
            max_length=100,
            num_return_sequences=1,
            pad_token_id=tokenizer.pad_token_id  # Use explicitly defined padding token
        )

        # Decode and print the response
        response = tokenizer.decode(output[0], skip_special_tokens=True)
        print(f"Chatbot: {response[len(user_input):].strip()}")  # Remove user's input from
response

# Run the chatbot
```

```python
if __name__ == "__main__":
    chatbot()
```

Output –

```
You: Hello
Chatbot: , I'm sorry, but I'm not sure if
You: How are you ?
Chatbot: I am a professional photographer.
You: Exit
Chatbot: Goodbye!
```

PRACTICAL NO – 4

Aim - Developing a recommendation system using collaborative filtering or deep learning approaches.

Code-

```
import numpy as np
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, Flatten, Dense, Concatenate
from sklearn.model_selection import train_test_split

# Sample data (user, item, rating)
data = np.array([
    [1, 101, 5], [1, 102, 3], [2, 101, 4], [2, 103, 2],
    [3, 102, 4], [3, 104, 1]
])

users = data[:, 0]
items = data[:, 1]
ratings = data[:, 2]

# Normalize ratings
ratings = (ratings - ratings.min()) / (ratings.max() - ratings.min())

# Train-test split
train_users, test_users, train_items, test_items, train_ratings, test_ratings = train_test_split(
    users, items, ratings, test_size=0.2, random_state=42
)

# Number of unique users and items
```

```
num_users = len(np.unique(users))
num_items = len(np.unique(items))

# Model architecture
user_input = Input(shape=(1,))
item_input = Input(shape=(1,))

user_embedding = Embedding(num_users + 1, 50)(user_input)
item_embedding = Embedding(num_items + 1, 50)(item_input)

user_vec = Flatten()(user_embedding)
item_vec = Flatten()(item_embedding)

concat = Concatenate()([user_vec, item_vec])
dense = Dense(128, activation='relu')(concat)
output = Dense(1, activation='sigmoid')(dense)

model = Model(inputs=[user_input, item_input], outputs=output)
model.compile(optimizer='adam', loss='mse', metrics=['mae'])

# Train the model
model.fit([train_users, train_items], train_ratings, epochs=10, batch_size=16, verbose=1)

# Test the model
predicted_ratings = model.predict([test_users, test_items])
print(f"Predicted ratings: {predicted_ratings.flatten()}")
```

Output –

```
Epoch 1/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step - loss: 0.1080 - mae: 0.3105
Epoch 2/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step - loss: 0.1054 - mae: 0.3071
Epoch 3/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - loss: 0.1030 - mae: 0.3038
Epoch 4/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step - loss: 0.1009 - mae: 0.3008
Epoch 5/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - loss: 0.0988 - mae: 0.2977
Epoch 6/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - loss: 0.0967 - mae: 0.2947
Epoch 7/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step - loss: 0.0946 - mae: 0.2916
Epoch 8/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step - loss: 0.0925 - mae: 0.2885
Epoch 9/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - loss: 0.0904 - mae: 0.2853
Epoch 10/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - loss: 0.0883 - mae: 0.2820
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 217ms/step
Predicted ratings: [0.5275267  0.51566106]
```

PRACTICAL NO – 5

Aim - Implementing a computer vision project, such as object detection or image segmentation.

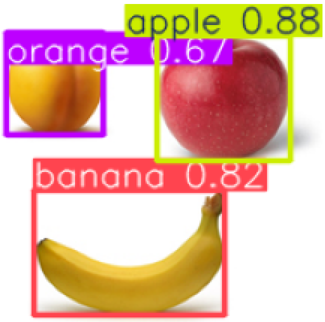
Code-

```
# Install required libraries

# pip install ultralytics requests


from ultralytics import YOLO

import cv2

import numpy as np

import requests

from matplotlib import pyplot as plt


# Load the pre-trained YOLO model

model = YOLO('yolov8n.pt')  # Replace with the appropriate model version


# Download the image from a URL

image_url = 'https://i.sstatic.net/UYYqo.jpg'

response = requests.get(image_url, stream=True)

image_array = np.asarray(bytearray(response.content), dtype=np.uint8)

image = cv2.imdecode(image_array, cv2.IMREAD_COLOR)


# Convert BGR to RGB for visualization

image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)


# Perform object detection

results = model(image)
```

```
# Show the detected results
results_image = results[0].plot()  # Annotated image with bounding boxes and labels

# Display the output using matplotlib
plt.figure(figsize=(10, 10))
plt.imshow(cv2.cvtColor(results_image, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()

# Print detected objects and their confidence scores
for result in results[0].boxes:
    print(f"Object: {result.data[0][5]}, Confidence: {result.data[0][4]:.2f}")
```

Output –

```
0: 640x640 1 banana, 1 apple, 1 orange, 998.9ms
Speed: 20.4ms preprocess, 998.9ms inference, 2.2ms postprocess per image at shape (1, 3, 640, 640)
```




```
Object: 47.0, Confidence: 0.88
Object: 46.0, Confidence: 0.82
Object: 49.0, Confidence: 0.67
```

PRACTICAL NO – 6

Aim - Applying reinforcement learning algorithms to solve complex decision-making problems.

Code-

```
import numpy as np

import random

# Define the environment

grid_size = 5

goal = (4, 4)

obstacles = [(1, 1), (2, 3), (3, 1)]

# Q-learning parameters

alpha = 0.1  # Learning rate

gamma = 0.9  # Discount factor

epsilon = 0.1  # Exploration rate

episodes = 1000

# Initialize the Q-table

q_table = np.zeros((grid_size, grid_size, 4))  # 4 actions: up, down, left, right

# Define actions

actions = {

    0: (-1, 0),  # Up

    1: (1, 0),   # Down

    2: (0, -1),  # Left

    3: (0, 1)    # Right

}

def is_valid_position(pos):

    """Check if the position is valid (not out of bounds or in an obstacle)."""

    x, y = pos

    if x < 0 or x >= grid_size or y < 0 or y >= grid_size:
```

```
        return False
    if pos in obstacles:
        return False
    return True
def get_next_state(state, action):
    """Get the next state after taking an action."""
    x, y = state
    dx, dy = actions[action]
    next_state = (x + dx, y + dy)
    if is_valid_position(next_state):
        return next_state
    return state  # Stay in the same position if invalid move
def get_reward(state):
    """Return the reward for a given state."""
    if state == goal:
        return 10
    if state in obstacles:
        return -10
    return -1
# Training the agent
for episode in range(episodes):
    state = (0, 0)  # Start position
    total_reward = 0
    while state != goal:
        # Choose an action (epsilon-greedy strategy)
        if random.uniform(0, 1) < epsilon:
            action = random.choice(list(actions.keys()))  # Explore
        else:
            action = np.argmax(q_table[state[0], state[1]])  # Exploit
```

```
        # Take the action
        next_state = get_next_state(state, action)
        reward = get_reward(next_state)
        # Update Q-value using the Q-learning formula
        old_value = q_table[state[0], state[1], action]
        next_max = np.max(q_table[next_state[0], next_state[1]])
        q_table[state[0], state[1], action] = old_value + alpha * (reward + gamma * next_max - old_value)
        state = next_state
        total_reward += reward
    if (episode + 1) % 100 == 0:
        print(f"Episode {episode + 1}, Total Reward: {total_reward}")
# Test the agent
state = (0, 0)
path = [state]
while state != goal:
    action = np.argmax(q_table[state[0], state[1]])
    state = get_next_state(state, action)
    path.append(state)
print("Optimal Path:", path)
```

Output –

```
Python 3.12.5 (tags/v3.12.5:ff3bc82, Aug  6 2024, 20:45:27) [MSC v.1940 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

= RESTART: C:/Users/Guest Login/Documents/advanced ai/REINFORCEMENT LEARNING ALGORITHM.py
Episode 100, Total Reward: 3
Episode 200, Total Reward: 1
Episode 300, Total Reward: 3
Episode 400, Total Reward: 3
Episode 500, Total Reward: 2
Episode 600, Total Reward: 1
Episode 700, Total Reward: 3
Episode 800, Total Reward: 1
Episode 900, Total Reward: 3
Episode 1000, Total Reward: 0
Optimal Path: [(0, 0), (0, 1), (0, 2), (1, 2), (2, 2), (3, 2), (4, 2), (4, 3), (4, 4)]
```

PRACTICAL NO – 7

Aim - Building a deep learning model for time series forecasting or anomaly detection

Code-

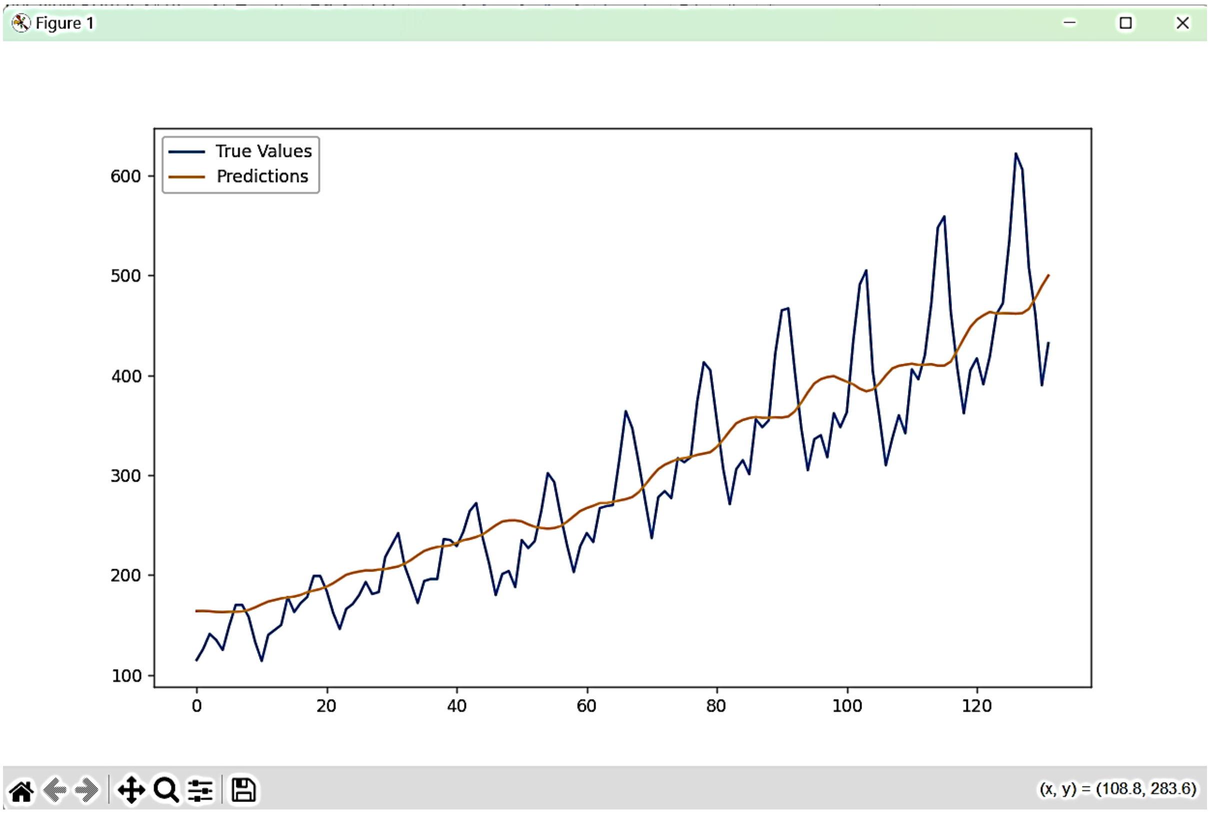
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.optimizers import Adam

# Step 1: Load dataset (using a new example dataset)
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/airline-passengers.csv'
data = pd.read_csv(url, header=0, index_col=0, parse_dates=True)

# Show the first few rows of the data
print(data.head())

# Step 2: Preprocessing - Using the dataset's values
data = data.values
data = data.reshape(-1, 1)

# Normalize the data using MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(data)

# Step 3: Create sequences of data for training
def create_sequences(data, time_step=12):
```

```
    X, y = [], []
    for i in range(len(data) - time_step):
        X.append(data[i:i + time_step])
        y.append(data[i + time_step])
    return np.array(X), np.array(y)

time_step = 12  # Using 12 previous months to predict the next one
X, y = create_sequences(scaled_data, time_step)

# Reshape the data for LSTM (samples, time steps, features)
X = X.reshape(X.shape[0], X.shape[1], 1)

# Step 4: Build the LSTM model
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(X.shape[1], 1)))
model.add(LSTM(units=50, return_sequences=False))
model.add(Dense(units=1))

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error')

# Step 5: Train the model
model.fit(X, y, epochs=10, batch_size=32)

# Step 6: Make predictions
predictions = model.predict(X)

# Step 7: Inverse transform the predictions
predictions = scaler.inverse_transform(predictions)
```

```
# Plot the predictions and the true values
plt.figure(figsize=(10, 6))
plt.plot(data[time_step:], label="True Values")
plt.plot(predictions, label="Predictions")
plt.legend()
plt.show()
```

OUTPUT-

```
========= RESTART: C:/Users/
            Passengers
Month
1949-01-01         112
1949-02-01         118
1949-03-01         132
1949-04-01         129
1949-05-01         121
```



PRACTICAL NO – 8

Aim - Implementing a machine learning pipeline for automated feature engineering and model selection

Code-

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from tpot import TPOTClassifier
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score

# Step 1: Load and Prepare Data
# Here we use the Iris dataset as an example
data = load_iris()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.Series(data.target)

# Step 2: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 3: Preprocessing and Feature Engineering
# Define which columns are numeric
numeric_features = X.columns.tolist()
```

```
# We can apply different transformations to numeric features
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),  # Handle missing values
    ('scaler', StandardScaler())  # Normalize the data
])

# Combine everything into a preprocessor
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features)
    ]
)

# Step 4: Define a Model Selection Pipeline using TPOT
# Use TPOT to automatically select the best model and parameters
tpot = TPOTClassifier( generations=5, population_size=20, random_state=42, verbosity=2)

# Step 5: Build the full pipeline
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),  # Apply preprocessing
    ('model', tpot)  # Automated model selection with TPOT
])

# Step 6: Train the model using the training data
pipeline.fit(X_train, y_train)

# Step 7: Evaluate the model on the test set
y_pred = pipeline.predict(X_test)
```

print(f'Accuracy: {accuracy_score(y_test, y_pred):.4f}')

Optional: Export the best model pipeline found by TPOT

tpot.export('best_model_pipeline.py')

OUTPUT-

```
0:23,  1.28pipeline/s]Optimization Progress:  76%|███████   | 91/120 [01:14<00:22,  1.26pipeline/s]Optimization Progress:  77%|███████   | 92/120 [01:15<00:20,  1.39pip
eline/s]Optimization Progress:  78%|███████   | 93/120 [01:15<00:16,  1.64pipeline/s]Optimization Progress:  78%|███████   | 94/120 [01:16<00:16,  1.56pipeline/s]Optim
ization Progress:  79%|███████   | 95/120 [01:16<00:12,  1.99pipeline/s]Optimization Progress:  80%|████████  | 96/120 [01:17<00:11,  2.17pipeline/s]Optimization Progre
ss:  81%|████████  | 97/120 [01:18<00:17,  1.35pipeline/s]Optimization Progress:  82%|████████  | 98/120 [01:19<00:17,  1.26pipeline/s]Optimization Progress:  82%|████████
██  | 99/120 [01:20<00:15,  1.33pipeline/s]Optimization Progress:  83%|████████  | 100/120 [01:20<00:13,  1.48pipeline/s]

Generation 4 - Current best internal CV score: 0.9583333333333334
Optimization Progress:  83%|████████  | 100/120 [01:22<00:13,  1.48pipeline/s]Optimization Progress:  84%|████████  | 101/120 [01:25<00:34,  1.83s/pipeline]Optimizatio
n Progress:  86%|████████  | 103/120 [01:25<00:19,  1.12s/pipeline]Optimization Progress:  87%|████████  | 104/120 [01:26<00:16,  1.01s/pipeline]Optimization Progress:
  88%|████████  | 105/120 [01:27<00:15,  1.06s/pipeline]Optimization Progress:  88%|████████  | 106/120 [01:28<00:13,  1.05pipeline/s]Optimization Progress:  89%|████████
██  | 107/120 [01:29<00:12,  1.08pipeline/s]Optimization Progress:  90%|█████████  | 108/120 [01:29<00:08,  1.34pipeline/s]Optimization Progress:  91%|█████████  | 109/12
0 [01:29<00:07,  1.46pipeline/s]Optimization Progress:  92%|█████████  | 110/120 [01:30<00:05,  1.84pipeline/s]Optimization Progress:  95%|█████████  | 114/120 [01:30<00:
01,  3.54pipeline/s]Optimization Progress:  96%|█████████  | 115/120 [01:31<00:01,  2.73pipeline/s]Optimization Progress:  97%|█████████| 116/120 [01:31<00:01,  2.48pip
eline/s]Optimization Progress:  98%|█████████| 117/120 [01:32<00:01,  2.47pipeline/s]Optimization Progress:  98%|█████████| 118/120 [01:32<00:00,  2.25pipeline/s]Opt
imization Progress:  99%|█████████| 119/120 [01:33<00:00,  2.08pipeline/s]Optimization Progress: 100%|██████████| 120/120 [01:34<00:00,  1.77pipeline/s]

Generation 5 - Current best internal CV score: 0.9583333333333334
Optimization Progress: 100%|██████████| 120/120 [01:36<00:00,  1.77pipeline/s]
Best pipeline: KNeighborsClassifier(input_matrix, n_neighbors=10, p=1, weights=uniform)
Accuracy: 1.0000
>>>
```

PRACTICAL NO – 9

Aim – Using advanced optimization techniques like evolutionary algorithms or Bayesian optimization for hyperparameter tuning.

1. **Bayesian Optimization with Optuna**

Code-

```
import optuna
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Load the Iris dataset
data = load_iris()
X = data.data
y = data.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Define an objective function for Optuna optimization
def objective(trial):
    # Define hyperparameters to tune
    n_estimators = trial.suggest_int('n_estimators', 10, 100)
    max_depth = trial.suggest_int('max_depth', 3, 10)
    min_samples_split = trial.suggest_int('min_samples_split', 2, 10)
    min_samples_leaf = trial.suggest_int('min_samples_leaf', 1, 5)

    # Create a Random Forest Classifier model
    model = RandomForestClassifier(n_estimators=n_estimators,
                                   max_depth=max_depth,
                                   min_samples_split=min_samples_split,
                                   min_samples_leaf=min_samples_leaf,
                                   random_state=42)

    # Train the model
    model.fit(X_train, y_train)

    # Evaluate the model
```

```
    y_pred = model.predict(X_test)
    return accuracy_score(y_test, y_pred)

# Set up the Optuna study for optimization
study = optuna.create_study(direction="maximize")

# Optimize hyperparameters
study.optimize(objective, n_trials=50)

# Print the best hyperparameters found by Optuna
print(f"Best hyperparameters: {study.best_params}")
print(f"Best accuracy: {study.best_value}")
```

OUTPUT -

```
: 1}. Best is trial 0 with value: 1.0.[0m
[32m[I 2025-01-17 15:06:51,972][0m Trial 47 finished with value: 1.0 and parameters: {'n_estimators': 60, 'max
2}. Best is trial 0 with value: 1.0.[0m
[32m[I 2025-01-17 15:06:52,100][0m Trial 48 finished with value: 1.0 and parameters: {'n_estimators': 52, 'max
: 5}. Best is trial 0 with value: 1.0.[0m
[32m[I 2025-01-17 15:06:52,258][0m Trial 49 finished with value: 1.0 and parameters: {'n_estimators': 87, 'max
4}. Best is trial 0 with value: 1.0.[0m
Best hyperparameters: {'n_estimators': 65, 'max_depth': 7, 'min_samples_split': 8, 'min_samples_leaf': 5}
Best accuracy: 1.0
>>>
```

2. Evolutionary Algorithms with TPOT

Code-

```
from tpot import TPOTClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load the Iris dataset
data = load_iris()
X = data.data
y = data.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Initialize the TPOTClassifier (using evolutionary algorithms for hyperparameter optimization)

tpot = TPOTClassifier(generations=5, population_size=20, random_state=42, verbosity=2)

Train the model and optimize hyperparameters

tpot.fit(X_train, y_train)

Evaluate the model on the test set

y_pred = tpot.predict(X_test)

print(f'Accuracy: {accuracy_score(y_test, y_pred):.4f}')

Export the best model pipeline found by TPOT

tpot.export('best_model_pipeline.py')

OUTPUT-

```
peline/s]Optimization Progress:  59%|      | 71/120 [00:32<00:24,  2.00pipeline/s]Optimization Progress:  60%|      | 72/120 [00:32<00:20,  2.30pipeline/s]Optim
ization Progress:  61%|      | 73/120 [00:32<00:18,  2.56pipeline/s]Optimization Progress:  62%|      | 74/120 [00:33<00:18,  2.46pipeline/s]Optimization Progr
ess:  62%|      | 75/120 [00:33<00:17,  2.56pipeline/s]Optimization Progress:  63%|      | 76/120 [00:34<00:18,  2.34pipeline/s]Optimization Progress:  64%|
      | 77/120 [00:34<00:18,  2.39pipeline/s]Optimization Progress:  65%|      | 78/120 [00:35<00:31,  1.35pipeline/s]Optimization Progress:  66%|      | 79/120
 [00:37<00:42,  1.04s/pipeline]Optimization Progress:  67%|      | 80/120 [00:38<00:36,  1.09pipeline/s]

Generation 3 - Current best internal CV score: 0.975
Optimization Progress:  67%|      | 80/120 [00:41<00:36,  1.09pipeline/s]Optimization Progress:  68%|      | 82/120 [00:41<00:48,  1.29s/pipeline]Optimization
Progress:  72%|      | 87/120 [00:41<00:16,  2.00pipeline/s]Optimization Progress:  74%|      | 89/120 [00:42<00:12,  2.52pipeline/s]Optimization Progress:  76
%|      | 91/120 [00:42<00:09,  2.93pipeline/s]Optimization Progress:  78%|      | 93/120 [00:43<00:09,  2.91pipeline/s]Optimization Progress:  79%|      |
95/120 [00:43<00:06,  3.80pipeline/s]Optimization Progress:  81%|      | 97/120 [00:43<00:04,  4.70pipeline/s]Optimization Progress:  82%|      | 99/120 [00:43<
00:04,  4.29pipeline/s]Optimization Progress:  83%|      | 100/120 [00:44<00:04,  4.38pipeline/s]

Generation 4 - Current best internal CV score: 0.975
Optimization Progress:  83%|      | 100/120 [00:44<00:04,  4.38pipeline/s]Optimization Progress:  84%|      | 101/120 [00:45<00:09,  2.10pipeline/s]Optimizatio
n Progress:  86%|      | 103/120 [00:46<00:08,  1.99pipeline/s]Optimization Progress:  87%|      | 104/120 [00:46<00:06,  2.33pipeline/s]Optimization Progress:
  88%|      | 105/120 [00:47<00:06,  2.21pipeline/s]Optimization Progress:  88%|      | 106/120 [00:47<00:05,  2.62pipeline/s]Optimization Progress:  91%|
      | 109/120 [00:48<00:03,  2.77pipeline/s]Optimization Progress:  92%|      | 110/120 [00:48<00:03,  3.09pipeline/s]Optimization Progress:  93%|      | 112/12
0 [00:49<00:02,  3.21pipeline/s]Optimization Progress:  94%|      | 113/120 [00:49<00:01,  3.53pipeline/s]Optimization Progress:  98%|      | 118/120 [00:49<00:
00,  7.56pipeline/s]Optimization Progress: 100%|      | 120/120 [00:50<00:00,  4.48pipeline/s]

Generation 5 - Current best internal CV score: 0.975
Optimization Progress: 100%|      | 120/120 [00:51<00:00,  4.48pipeline/s]
Best pipeline: MLPClassifier(input_matrix, alpha=0.0001, learning_rate_init=0.001)
Accuracy: 1.0000
>>>
```

```
[I 2025-01-26 08:40:07,557] Trial 47 finished with value: 1.0 and parameters: {'n_estimators': 128, 'max_depth
[I 2025-01-26 08:40:07,629] Trial 48 finished with value: 1.0 and parameters: {'n_estimators': 27, 'max_depth'
[I 2025-01-26 08:40:07,765] Trial 49 finished with value: 1.0 and parameters: {'n_estimators': 66, 'max_depth'
Best Hyperparameters: {'n_estimators': 182, 'max_depth': 48, 'min_samples_split': 10, 'min_samples_leaf': 7}
Best Accuracy: 1.0000
```

Practical 10

Q10. Deploying a machine learning model in a production environment using containerization and cloud services.

Aim: Deploying a machine learning model in a production environment using containerization and cloud services.

Code:-

```
!pip install transformers datasets

from transformers import GPT2LMHeadModel, GPT2Tokenizer, Trainer,
TrainingArguments
from datasets import load_dataset

# Load a pre-trained GPT-2 model and tokenizer
model_name = "gpt2"
model = GPT2LMHeadModel.from_pretrained(model_name)
tokenizer = GPT2Tokenizer.from_pretrained(model_name)

# Define the padding token (usually the EOS token for GPT-2)
tokenizer.pad_token = tokenizer.eos_token # This line is crucial to fix
the error

# Load a text dataset (You can replace this with your own dataset)
dataset = load_dataset("wikitext", "wikitext-103-raw-v1")

# Preprocess the dataset (Tokenization)
def encode(example):
    return tokenizer(example['text'], truncation=True,
padding="max_length", max_length=512)

train_data = dataset['train'].map(encode, batched=True)
val_data = dataset['validation'].map(encode, batched=True)

# Set up training arguments
training_args = TrainingArguments(
    output_dir="./results",
    num_train_epochs=1,
    per_device_train_batch_size=4,
    per_device_eval_batch_size=4,
    logging_dir="./logs",
    logging_steps=10,
    save_steps=10,
    warmup_steps=100,
    weight_decay=0.01,
    evaluation_strategy="steps",
    save_total_limit=2,
)

# Set up Trainer
trainer = Trainer(
    model=model,
    args=training_args,
```

```
    train_dataset=train_data,
    eval_dataset=val_data,
)

# Train the model
trainer.train()
```

OUTPUT:-

```
Map: 100%  1801350/1801350 [26:48<00:00, 1354.28 examples/s]
Map: 100%  3760/3760 [00:04<00:00, 723.60 examples/s]
/usr/local/lib/python3.10/dist-packages/transformers/training_args.py:1575: FutureWarning: `evaluation_strategy` is deprecated and will
  warnings.warn(
wandb: WARNING The `run_name` is currently set to the same value as `TrainingArguments.output_dir`. If this was not intended, please sp
wandb: Using wandb-core as the SDK backend.  Please refer to https://wandb.me/wandb-core for more information.
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: https://wandb.me/wandb-server)
wandb: You can find your API key in your browser here: https://wandb.ai/authorize
wandb: Paste an API key from your profile and hit enter, or press ctrl+c to quit:
```

University of Mumbai

Practical Journal of
Machine Learning

M.Sc. (Information Technology)
Part-II

Submitted by
SINGH MANASI RAKESH

Seat No: 1313421

DEPARTMENT OF INFORMATION TECHNOLOGY
PILLAI HOC COLLEGE OF ARTS, SCIENCE &COMMERCE, RASAYANI
(Affiliated to Mumbai University)
RASAYANI, 410207
MAHARASHTRA
2024-2025

Mahatma Education Society's
Pillai Hoc College of Arts, Science & Commerce, Rasayani
(Affiliated to Mumbai University)
RASAYANI – MAHARASHTRA - 410207

DEPARTMENT OF INFORMATION TECHNOLOGY

CERTIFICATE

This is to certify that the experiment work entered in this journal is as per the syllabus in **M.Sc. (Information Technology) Part-II, Semester-III**; class prescribed by University of Mumbai for the subject **Machine Learning** was done in computer lab of Mahatma Education Society's Pillai HOC College of Arts, Science & Commerce, Rasayani by **MANASI SINGH** during Academic year 2024-2025.

Exam Seat No: 1313421

Subject Incharge **Coordinator**

External Examiner **Principal**

Date: **College Seal**

MACHINE LEARNING

INDEX

Sr.No.	Practicals	Date	Sign
1	**Data Pre-processing and Exploration**		
a.	Load a CSV dataset. Handle missing values, inconsistent formatting, and outliers.		
b.	Load a dataset, calculate descriptive summary statistics, create visualizations using different graphs, and identify potential features and target variables **Note**: Explore Univariate and Bivariate graphs (Matplotlib) and Seaborn for visualization.		
c	Create or Explore datasets to use all pre-processing routines like label encoding, scaling, and binarization. .		
2	**Testing Hypothesis**		
a	Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a. CSV file and generate the final specific hypothesis. (Create your dataset)		
3	**Linear Models**		
a	**Simple Linear Regression** Fit a linear regression model on a dataset. Interpret coefficients, make predictions, and evaluate performance using metrics like R-squared and MSE.		
b	**Multiple Linear Regression** Extend linear regression to multiple features. Handle feature selection and potential multicollinearity.		
c	**Regularized Linear Models** (Ridge, Lasso, ElasticNet) Implement regression variants like LASSO and Ridge on any generated dataset.		
4	**Discriminative Models**		
a	**Logistic Regression** Perform binary classification using logistic regression. Calculate accuracy, precision, recall, and understand the ROC curve**.**		
b	Implement and demonstrate k-nearest Neighbor algorithm. Read the training data from a .CSV file and build the model to classify a testsample. Print both correct and wrong predictions.		

c	Build a decision tree classifier or regressor. Control hyperparameters like tree depth to avoid overfitting. Visualize the tree.		
d	Implement a Support Vector Machine for any relevant dataset.		
e	Implement a gradient boosting machine (e.g., XGBoost). Tune hyperparameters and explore feature importance.		
5	**Probabilistic Models**		
a	Implement Bayesian Linear Regression to explore prior and posterior Distribution.		
b	Implement Gaussian Mixture Models for density estimation and unsupervised clustering.		
6	**Bayesian Learning**		
a	Implement Bayesian Learning using inferences		

Practical 1

Q1a. Load a CSV dataset. Handle missing values, inconsistent formatting, and outliers.

Aim:- Load a CSV dataset. Handle missing values, inconsistent formatting, and outliers.

Code:-

```
# Import the necessary module
from google.colab import files

# Upload the file
uploaded = files.upload()
```

https://drive.google.com/drive/folders/1RIVpwC7y7LO-DcMhWnDl9W9wfjO_o94b

```
# Corrected indentation:
print(f"Data from netflix_titles.csv:")
df = pd.read_csv('netflix_titles.csv')
print(df)
```

```
df = pd.read_csv('netflix_titles.csv')
print(f"Data from netflix_titles.csv:") # Removed the extra indent from this line and the curly brackets
since we are not using f-string formatting for the file name anymore
# Step 4: Check for missing values
print("\nMissing values in each column:")
missing_values = df.isnull().sum()
print(missing_values)



# Step 5: Check the percentage of missing values
missing_percentage = (missing_values / len(df)) * 100
print("\nPercentage of missing values in each column:")
print(missing_percentage)
```

```
import matplotlib.pyplot as plt # Import the pyplot module from matplotlib and alias it as plt

plt.figure(figsize=(10, 6))
df.plot(kind='bar')
plt.title('CSV Data Visualization')
plt.xlabel('Index')
plt.ylabel('Values')
plt.show()
```

OUTPUT:

```
# Import the necessary module
from google.colab import files

# Upload the file
uploaded = files.upload()
```

Choose Files netflix_titles.csv

- **netflix_titles.csv**(text/csv) - 3399671 bytes, last modified: 1/11/2025 - 100% done

```
Saving netflix_titles.csv to netflix_titles.csv
```

```
# Corrected indentation:
print(f"Data from netflix_titles.csv:")
df = pd.read_csv('netflix_titles.csv')
print(df)
```

```
Data from netflix_titles.csv:
     show_id     type                  title         director  \
0         s1    Movie   Dick Johnson Is Dead  Kirsten Johnson
1         s2  TV Show          Blood & Water              NaN
2         s3  TV Show              Ganglands  Julien Leclercq
3         s4  TV Show  Jailbirds New Orleans              NaN
4         s5  TV Show           Kota Factory              NaN
...      ...      ...                    ...              ...
8802   s8803    Movie                 Zodiac    David Fincher
8803   s8804  TV Show            Zombie Dumb              NaN
8804   s8805    Movie             Zombieland  Ruben Fleischer
8805   s8806    Movie                   Zoom     Peter Hewitt
8806   s8807    Movie                 Zubaan      Mozez Singh

                                                   cast        country  \
0                                                   NaN  United States
1     Ama Qamata, Khosi Ngema, Gail Mabalane, Thaban...   South Africa
2     Sami Bouajila, Tracy Gotoas, Samuel Jouy, Nabi...            NaN
3                                                   NaN            NaN
4     Mayur More, Jitendra Kumar, Ranjan Raj, Alam K...          India
...                                                 ...            ...
8802  Mark Ruffalo, Jake Gyllenhaal, Robert Downey J...  United States
8803                                                NaN            NaN
```

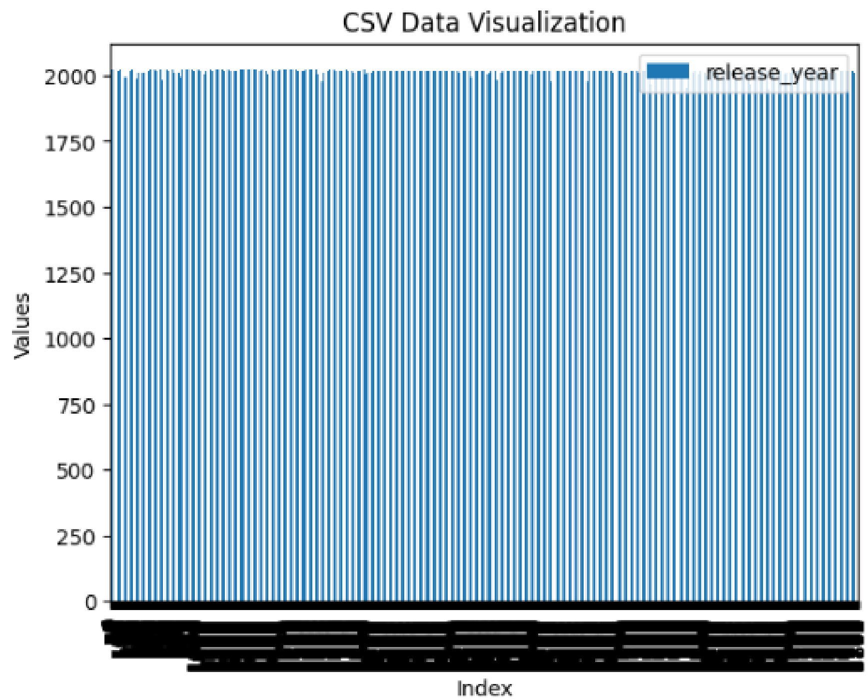
```
...                                              ...
8802                 Cult Movies, Dramas, Thrillers
8803         Kids' TV, Korean TV Shows, TV Comedies
8804                        Comedies, Horror Movies
8805             Children & Family Movies, Comedies
8806     Dramas, International Movies, Music & Musicals

                                        description
0     As her father nears the end of his life, filmm...
1     After crossing paths at a party, a Cape Town t...
2     To protect his family from a powerful drug lor...
3     Feuds, flirtations and toilet talk go down amo...
4     In a city of coaching centers known to train I...
...                                             ...
8802  A political cartoonist, a crime reporter and a...
8803  While living alone in a spooky town, a young g...
8804  Looking to survive in a world taken over by zo...
8805  Dragged from civilian life, a former superhero...
8806  A scrappy but poor boy worms his way into a ty...

[8807 rows x 12 columns]
```

```
df = pd.read_csv('netflix_titles.csv')
print(f"Data from netflix_titles.csv:") # Removed the extra indent from this line and the curly brackets sinc
# Step 4: Check for missing values
print("\nMissing values in each column:")
missing_values = df.isnull().sum()
print(missing_values)



# Step 5: Check the percentage of missing values
missing_percentage = (missing_values / len(df)) * 100
print("\nPercentage of missing values in each column:")
print(missing_percentage)
```

```
Data from netflix_titles.csv:

Missing values in each column:
show_id            0
type               0
title              0
director        2634
cast             825
country          831
date_added        10
release_year       0
rating             4
duration           3
listed_in          0
description        0
dtype: int64
```

```
dtype: int64

Percentage of missing values in each column:
show_id          0.000000
type             0.000000
title            0.000000
director        29.908028
cast             9.367549
country          9.435676
date_added       0.113546
release_year     0.000000
rating           0.045418
duration         0.034064
listed_in        0.000000
description      0.000000
dtype: float64
```

```
<Figure size 1000x600 with 0 Axes>
```



Q1b. Load a dataset, calculate descriptive summary statistics, create visualizations using different graphs, and identify potential features and target variables Note: Explore Univariate and Bivariate graphs (Matplotlib) and Seaborn for visualization.

Aim:- Load a dataset, calculate descriptive summary statistics, create visualizations using different graphs, and identify potential features and target variables **Note**: Explore Univariate and Bivariate graphs (Matplotlib) and Seaborn for visualization.

Code:-

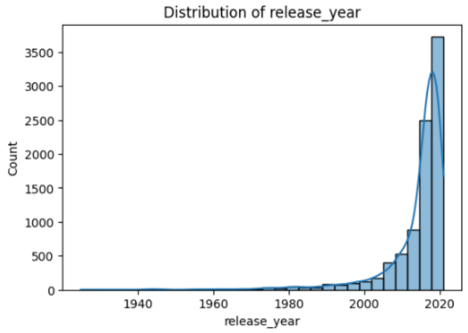
```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
import seaborn as sns

# Load dataset
# Replace 'your_dataset.csv' with the actual file path or URL to your dataset
data = pd.read_csv('/content/netflix_titles.csv')

# Display the first few rows of the dataset
print("First 5 rows of the dataset:")
print(data.head())

# Display the shape of the dataset
print("\nShape of the dataset:", data.shape)

# Check for missing values
print("\nMissing values:")
print(data.isnull().sum())

# Descriptive summary statistics
print("\nDescriptive Statistics:")
print(data.describe())

# Univariate Analysis - Numerical Variables
numeric_columns = data.select_dtypes(include=[np.number]).columns.tolist()
print("\nNumeric Columns:", numeric_columns)

for col in numeric_columns:
    plt.figure(figsize=(6, 4))
    sns.histplot(data[col], kde=True, bins=30)
    plt.title(f'Distribution of {col}')
    plt.show()

# Univariate Analysis - Categorical Variables
categorical_columns = data.select_dtypes(include=['object']).columns.tolist()
print("\nCategorical Columns:", categorical_columns)

for col in categorical_columns:
    plt.figure(figsize=(6, 4))
    sns.countplot(y=data[col], order=data[col].value_counts().index)
    plt.title(f'Count Plot of {col}')
    plt.show()

# Bivariate Analysis
# Pairplot for numerical columns
sns.pairplot(data[numeric_columns])
plt.show()

# Heatmap to visualize correlations
plt.figure(figsize=(10, 8))
sns.heatmap(data[numeric_columns].corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix')
plt.show()

# Scatter plots for specific relationships
# Replace 'feature1' and 'feature2' with your column names
for col1 in numeric_columns:
    for col2 in numeric_columns:
```

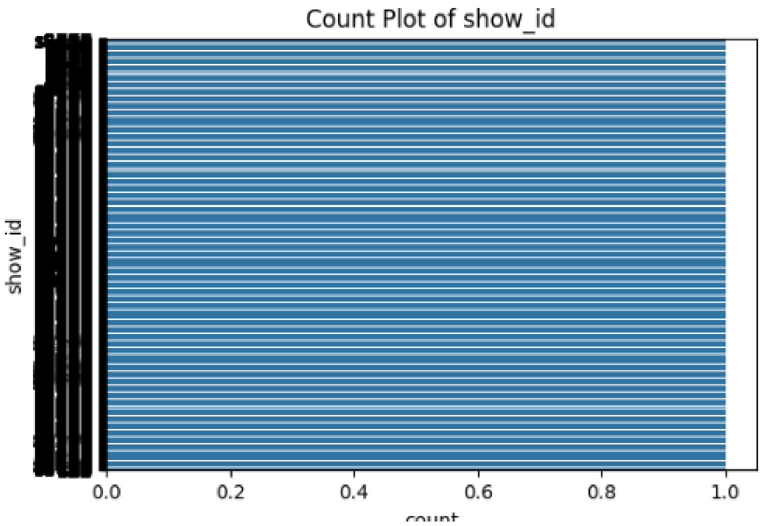
```
        if col1 != col2:
            plt.figure(figsize=(6, 4))
            sns.scatterplot(x=data[col1], y=data[col2])
            plt.title(f'{col1} vs {col2}')
            plt.show()

# Identify potential features and target variables
# Hypothetical target variable
target_variable = 'target_column'  # Replace with your actual target column
features = [col for col in data.columns if col != target_variable]
print("\nPotential Features:", features)
print("Target Variable:", target_variable)
```

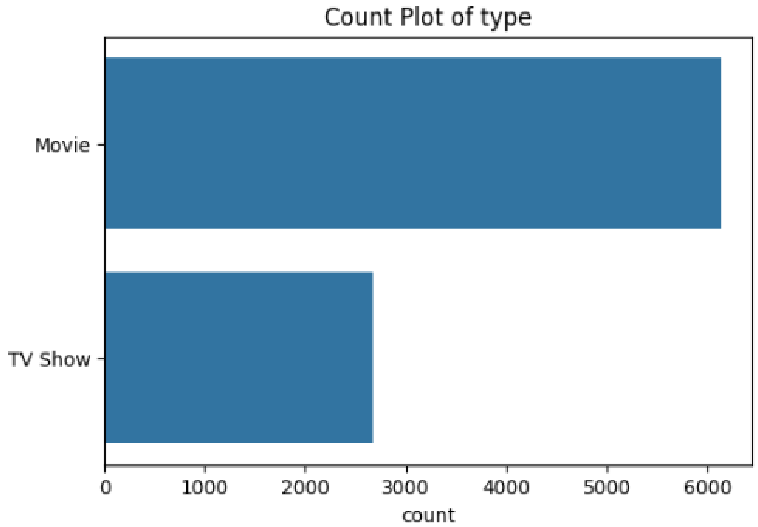
OUTPUT:-

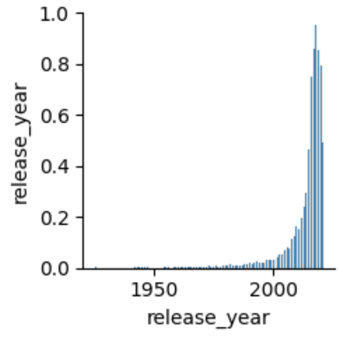
```
Numeric Columns: ['release_year']
```

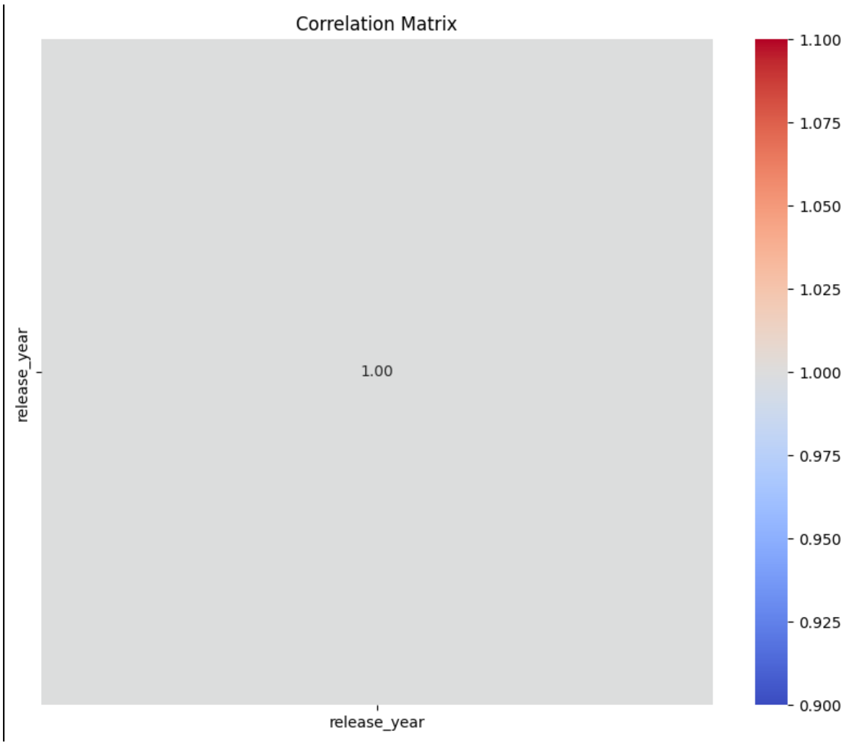



```
Categorical Columns: ['show_id', 'type', 'title', 'director', 'cast', 'country', 'date_added', 'rating', 'duration', 'listed_in', 'de
```


Count Plot of show_id
show_id
0.0
0.2
0.4
0.6
0.8
1.0


Count Plot of type
Movie
TV Show
0
1000
2000
3000
4000
5000
6000
count






```
Potential Features: ['show_id', 'type', 'title', 'director', 'cast', 'country', 'date_added', 'release_year', 'rating', 'duration', 'listed_in', 'description']
Target Variable: target_column
```

Q1c. Create or Explore datasets to use all pre-processing routines like label encoding, scaling, and binarization.

Aim:- Create or Explore datasets to use all pre-processing routines like label encoding, scaling, and binarization.

Code:-

```
# Import necessary libraries
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler, Binarizer
from sklearn.datasets import make_classification

# Generate a synthetic dataset
# Alternatively, load your dataset using pd.read_csv('your_dataset.csv')
data, target = make_classification(
    n_samples=1000,
    n_features=10,
    n_informative=5,
    n_redundant=2,
    random_state=42
)

# Convert the dataset into a DataFrame
```

```
columns = [f'Feature_{i}' for i in range(1, 11)]
df = pd.DataFrame(data, columns=columns)
df['Target'] = target

# Add a categorical column for demonstration
df['Category'] = np.random.choice(['A', 'B', 'C'], size=df.shape[0])

# Display the first few rows
print("Original Dataset:")
print(df.head())

# Step 1: Label Encoding
label_encoder = LabelEncoder()
df['Category_encoded'] = label_encoder.fit_transform(df['Category'])
print("\nLabel Encoded Column:")
print(df[['Category', 'Category_encoded']].head())

# Step 2: Feature Scaling (Standardization)
scaler = StandardScaler()
scaled_features = scaler.fit_transform(df[columns])
scaled_df = pd.DataFrame(scaled_features, columns=columns)
print("\nStandardized Features (First 5 rows):")
print(scaled_df.head())

# Step 3: Min-Max Scaling
minmax_scaler = MinMaxScaler()
minmax_scaled_features = minmax_scaler.fit_transform(df[columns])
minmax_df = pd.DataFrame(minmax_scaled_features, columns=columns)
print("\nMin-Max Scaled Features (First 5 rows):")
print(minmax_df.head())

# Step 4: Binarization
binarizer = Binarizer(threshold=0.5)  # Set the threshold as required
binarized_target = binarizer.fit_transform(df[['Target']])
df['Target_binarized'] = binarized_target
print("\nBinarized Target Column:")
print(df[['Target', 'Target_binarized']].head())

# Step 5: Display all processed results in Colab
from google.colab.data_table import DataTable
print("\nFinal Processed Dataset:")
DataTable(df)

# Step 6: Save to a CSV (Optional)
df.to_csv('processed_dataset.csv', index=False)
```

OUTPUT:-

```
2   0.516313   2.165426  -0.628486  -0.386923   0.492518   1.442381
3   0.537282   0.966618  -0.115420   0.670755  -0.958516   0.871440
4   0.278385   1.065828  -1.724917  -2.235667   0.715107   0.731249

   Feature_7  Feature_8  Feature_9  Feature_10  Target Category
0   1.419515   1.357325   0.966041   -1.981139       1        C
1   3.190292  -0.890254   1.438826   -3.828748       0        B
2   1.332905  -1.958175  -0.348803   -1.804124       0        A
3   0.508186  -1.034471  -1.654176   -1.910503       1        C
4  -0.674119   0.598330  -0.524283    1.047610       0        A

Label Encoded Column:
  Category  Category_encoded
0        C                 2
1        B                 1
2        A                 0
3        C                 2
4        A                 0

Standardized Features (First 5 rows):
   Feature_1  Feature_2  Feature_3  Feature_4  Feature_5  Feature_6  \
0   0.407215   0.735075   0.552800   0.845310  -0.727544   0.522771
1  -0.660203   2.256028  -1.416491  -0.691132   0.770728   2.641987
2   0.022641   1.345373  -0.543244   0.069507   0.015755   0.581616
3   0.035887   0.604333  -0.042049   0.766185  -0.958727   0.229302
4  -0.127660   0.665659  -1.614308  -1.148235   0.165241   0.142794

   Feature_7  Feature_8  Feature_9  Feature_10
0   0.482034   1.303030   0.979194   -0.931480
1   1.367794  -0.882197   1.431240   -1.590225
2   0.438711  -1.920492  -0.277973   -0.868367
3   0.026179  -1.022413  -1.526085   -0.906295
```

```
   Feature_7  Feature_8  Feature_9  Feature_10
0   0.482034   1.303030   0.979194   -0.931480
1   1.367794  -0.882197   1.431240   -1.590225
2   0.438711  -1.920492  -0.277973   -0.868367
3   0.026179  -1.022413  -1.526085   -0.906295
4  -0.565222   0.565091  -0.445755    0.148388

Min-Max Scaled Features (First 5 rows):
   Feature_1  Feature_2  Feature_3  Feature_4  Feature_5  Feature_6  \
0   0.594853   0.671057   0.558804   0.507749   0.329940   0.571715
1   0.421140   0.907193   0.239200   0.273170   0.554699   0.916873
2   0.532267   0.765809   0.380923   0.389302   0.441443   0.581299
3   0.534422   0.650759   0.462264   0.495668   0.295259   0.523917
4   0.507807   0.660280   0.207096   0.203381   0.463868   0.509827

   Feature_7  Feature_8  Feature_9  Feature_10
0   0.590821   0.726913   0.692423    0.381720
1   0.710832   0.372649   0.764402    0.266508
2   0.584951   0.204323   0.492245    0.392758
3   0.529057   0.349918   0.293510    0.386125
4   0.448928   0.607280   0.465529    0.570585

Binarized Target Column:
   Target  Target_binarized
0       1                 1
1       0                 0
2       0                 0
3       1                 1
4       0                 0

Final Processed Dataset:
```

Practical 2

Q2a. Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a. CSV file and generate the final specific hypothesis. (Create your dataset).

Aim:- Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a. CSV file and generate the final specific hypothesis. (Create your dataset)

Code:

```
import pandas as pd

# Creating a sample dataset for the Find-S algorithm
data = {
    'Sky': ['Sunny', 'Sunny', 'Overcast', 'Rain', 'Rain'],
    'AirTemp': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool'],
    'Humidity': ['High', 'High', 'High', 'High', 'High'],
    'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak'],
    'PlayTennis': ['Yes', 'No', 'Yes', 'Yes', 'Yes']
}

df = pd.DataFrame(data)

# Display the dataset
print("Dataset:")
print(df)

# Function for the Find-S Algorithm to find the most specific hypothesis
def find_s_algorithm(df):
    # Initialize the most specific hypothesis (taking the first positive example)
    hypothesis = df.iloc[0, :-1].values.tolist()

    # Iterate through each row in the dataset
    for i in range(1, len(df)):
        if df.iloc[i, -1] == 'Yes':  # Only consider positive examples (PlayTennis == Yes)
            for j in range(len(hypothesis)):
                # If the hypothesis value doesn't match the current example, generalize it
                if hypothesis[j] != df.iloc[i, j]:
                    hypothesis[j] = '?'  # '?' means the attribute is generalized

    return hypothesis

# Run the Find-S Algorithm
hypothesis = find_s_algorithm(df)

# Output the most specific hypothesis found
print("\nMost Specific Hypothesis:")
print(hypothesis)
```

OUTPUT:

```
Dataset:
        Sky AirTemp Humidity    Wind PlayTennis
0     Sunny     Hot     High    Weak        Yes
1     Sunny     Hot     High  Strong         No
2  Overcast     Hot     High    Weak        Yes
3      Rain    Mild     High    Weak        Yes
4      Rain    Cool     High    Weak        Yes

Most Specific Hypothesis:
['?', '?', 'High', 'Weak']
```

Steps to Follow:

1. First, you'll load the dataset from the find_s_dataset.csv.

Then, apply the **Find-S Algorithm** to extract the most specific hypothesis from the dataset.

Code for Find-S Algorithm Implementation:

```
# Import necessary libraries
import pandas as pd

# Load the dataset from CSV (you can upload the CSV file manually or use the code below)
from google.colab import files

# Assuming the 'find_s_dataset.csv' file is uploaded already
uploaded = files.upload()

# Read the CSV into a pandas DataFrame
df = pd.read_csv('find_s_dataset.csv')

# Display the dataset to verify it's loaded correctly
```

```
print("Dataset:")
print(df)

# Function to implement Find-S Algorithm
def find_s_algorithm(df):
    # Initialize hypothesis with the first positive example
    # The hypothesis is the attributes of the first row where PlayTennis = 'Yes'
    hypothesis = df[df['PlayTennis'] == 'Yes'].iloc[0, :-1].values.tolist()

    # Iterate through the rest of the rows
    for i in range(1, len(df)):
        if df.iloc[i, -1] == 'Yes':  # Only consider positive examples
            for j in range(len(hypothesis)):
                # If the attribute value doesn't match, generalize it
                if hypothesis[j] != df.iloc[i, j]:
                    hypothesis[j] = '?'  # '?' means the attribute is generalized

    return hypothesis

# Run the Find-S Algorithm
hypothesis = find_s_algorithm(df)

# Output the most specific hypothesis
print("\nMost Specific Hypothesis:")
print(hypothesis)
```

OUTPUT:

```
Choose Files find_s_dataset.csv
• find_s_dataset.csv(text/csv) - 161 bytes, last modified: 1/11/2025 - 100% done
Saving find_s_dataset.csv to find_s_dataset.csv
Dataset:
        Sky AirTemp Humidity    Wind PlayTennis
0     Sunny     Hot     High    Weak        Yes
1     Sunny     Hot     High  Strong         No
2  Overcast     Hot     High    Weak        Yes
3      Rain    Mild     High    Weak        Yes
4      Rain    Cool     High    Weak        Yes

Most Specific Hypothesis:
['?', '?', 'High', 'Weak']
```

Practical 3

Q3a. Simple Linear Regression: Fit a linear regression model on a dataset. Interpret coefficients, make predictions, and evaluate performance using metrics like R-squared and MSE.

Aim:- Simple Linear Regression :Fit a linear regression model on a dataset. Interpret coefficients, make predictions, and evaluate performance using metrics like R-squared and MSE.

Code:-

```
# Install required libraries
!pip install scikit-learn matplotlib

# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Sample dataset: number of rooms (X) and corresponding house prices (y)
# X = number of rooms, y = house price in thousands of dollars
X = np.array([1, 2, 3, 4, 5]).reshape(-1, 1)  # Number of rooms
y = np.array([100, 150, 200, 250, 300])  # House prices in thousands

# Initialize and train the linear regression model
model = LinearRegression()
model.fit(X, y)

# Predict house prices for the given number of rooms
predicted_prices = model.predict(X)

# Visualize the data and the regression line
plt.scatter(X, y, color='blue', label='Actual data')
plt.plot(X, predicted_prices, color='red', label='Regression line')
plt.xlabel('Number of Rooms')
plt.ylabel('House Price (in thousands)')
plt.title('House Price Prediction')
plt.legend()
plt.show()

# Print the model's coefficients and intercept
print(f'Coefficient: {model.coef_}')
print(f'Intercept: {model.intercept_}')

# Predict the price for a house with 6 rooms
predicted_price_for_6_rooms = model.predict([[6]])
print(f'Predicted price for a house with 6 rooms: ${predicted_price_for_6_rooms[0]:.2f} thousand')
```

OUTPUT:


```
Coefficient: [50.]
Intercept: 50.0
Predicted price for a house with 6 rooms: $350.00 thousand
```

Q3b. Multiple Linear Regression: Extend linear regression to multiple features. Handle feature selection and potential multicollinearity.

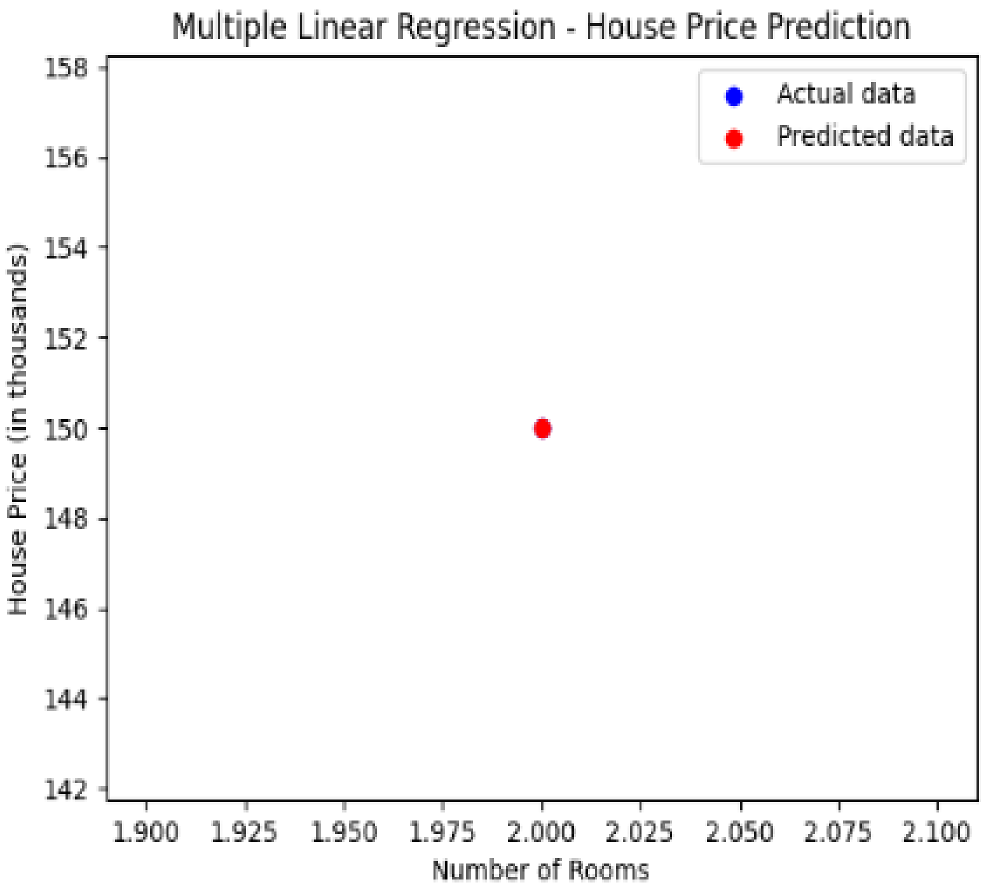
Aim:- **Multiple Linear Regression** :Extend linear regression to multiple features. Handle feature selection and potential multicollinearity.

Code:

```
# Install required libraries
!pip install scikit-learn matplotlib

# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

# Sample dataset: features (X) = [number of rooms, house age], target (y) = house price
# X = number of rooms, house age, y = house price in thousands of dollars
X = np.array([[1, 10], [2, 15], [3, 20], [4, 25], [5, 30]])  # Features: [rooms, age]
y = np.array([100, 150, 200, 250, 300])  # House prices in thousands
```

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train the multiple linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Visualize the actual vs predicted prices (for simplicity, we only plot one feature vs price)
plt.scatter(X_test[:, 0], y_test, color='blue', label='Actual data')
plt.scatter(X_test[:, 0], y_pred, color='red', label='Predicted data')
plt.xlabel('Number of Rooms')
plt.ylabel('House Price (in thousands)')
plt.title('Multiple Linear Regression - House Price Prediction')
plt.legend()
plt.show()
```

OUTPUT:




```
Coefficients: [1.92307692 9.61538462]
Intercept: 1.9230769230768487
Predicted price for a house with 6 rooms and 40 years old: $398.08 thousand
```

Q3c. Regularized Linear Models (Ridge, Lasso, ElasticNet) : Implement regression variants like LASSO and Ridge on any generated dataset.

Aim:- Regularized Linear Models (Ridge, Lasso, ElasticNet) : Implement regression variants like LASSO and Ridge on any generated dataset.

RIDGE REGRESSION:

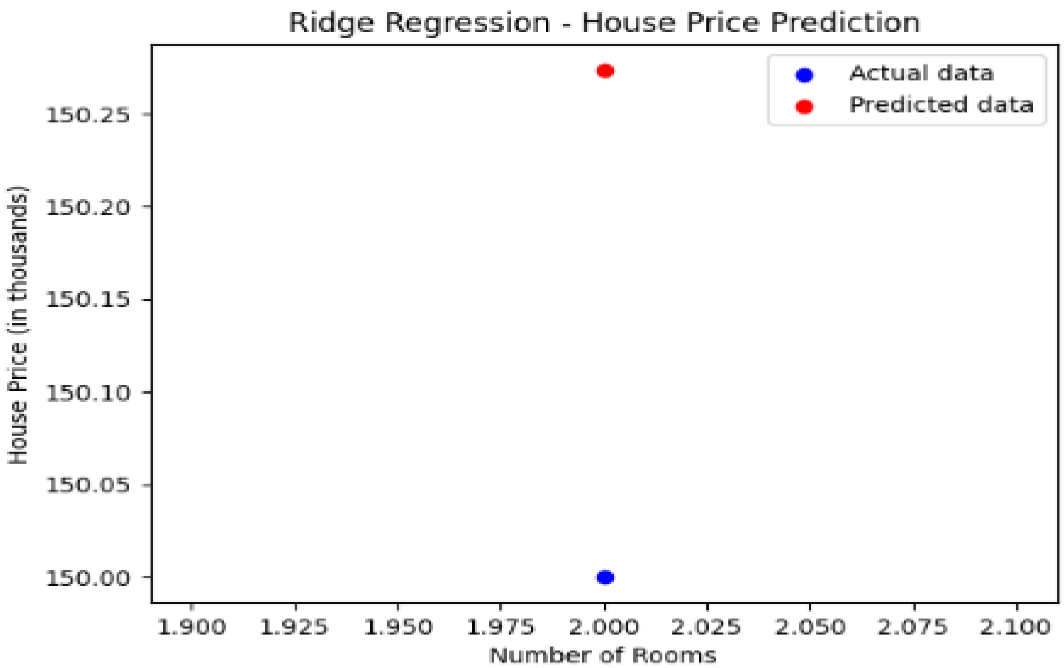
Code:-

```
# Install required libraries
!pip install scikit-learn matplotlib

# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import Ridge
from sklearn.model_selection import train_test_split
```

```
# Print the model's coefficients and intercept
print(f'Coefficients: {model.coef_}')
print(f'Intercept: {model.intercept_}')

# Predict the price for a new house with 6 rooms and 40 years old
predicted_price_for_new_house = model.predict([[6, 40]])
print(f'Predicted price for a house with 6 rooms and 40 years old:
${predicted_price_for_new_house[0]:.2f} thousand')

# Sample dataset: features (X) = [number of rooms, house age], target (y) = house price
# X = number of rooms, house age, y = house price in thousands of dollars
X = np.array([[1, 10], [2, 15], [3, 20], [4, 25], [5, 30]])  # Features: [rooms, age]
y = np.array([100, 150, 200, 250, 300])  # House prices in thousands
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train the Ridge regression model (regularized linear model)
ridge_model = Ridge(alpha=1.0)  # alpha is the regularization strength
ridge_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = ridge_model.predict(X_test)

# Visualize the actual vs predicted prices (for simplicity, we only plot one feature vs price)
plt.scatter(X_test[:, 0], y_test, color='blue', label='Actual data')
plt.scatter(X_test[:, 0], y_pred, color='red', label='Predicted data')
plt.xlabel('Number of Rooms')
plt.ylabel('House Price (in thousands)')
plt.title('Ridge Regression - House Price Prediction')
plt.legend()
plt.show()

# Print the model's coefficients and intercept
print(f'Coefficients: {ridge_model.coef_}')
print(f'Intercept: {ridge_model.intercept_}')
```

```
# Predict the price for a new house with 6 rooms and 40 years old
predicted_price_for_new_house = ridge_model.predict([[6, 40]])
print(f'Predicted price for a house with 6 rooms and 40 years old:
${predicted_price_for_new_house[0]:.2f} thousand')
```

OUTPUT:




```
Coefficients: [1.91466083 9.57330416]
Intercept: 2.8446389496717472
Predicted price for a house with 6 rooms and 40 years old: $397.26 thousand
```

LASSO REGRESSION:

Code:

```
# Install required libraries
!pip install scikit-learn matplotlib

# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import Lasso
from sklearn.model_selection import train_test_split

# Sample dataset: features (X) = [number of rooms, house age], target (y) = house price
# X = number of rooms, house age, y = house price in thousands of dollars
X = np.array([[1, 10], [2, 15], [3, 20], [4, 25], [5, 30]])  # Features: [rooms, age]
y = np.array([100, 150, 200, 250, 300])  # House prices in thousands

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train the Lasso regression model (regularized linear model)
```

```
lasso_model = Lasso(alpha=0.1)  # alpha is the regularization strength
lasso_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = lasso_model.predict(X_test)

# Visualize the actual vs predicted prices (for simplicity, we only plot one feature vs price)
plt.scatter(X_test[:, 0], y_test, color='blue', label='Actual data')
plt.scatter(X_test[:, 0], y_pred, color='red', label='Predicted data')
plt.xlabel('Number of Rooms')
plt.ylabel('House Price (in thousands)')
plt.title('Lasso Regression - House Price Prediction')
plt.legend()
plt.show()

# Print the model's coefficients and intercept
print(f'Coefficients: {lasso_model.coef_}')
print(f'Intercept: {lasso_model.intercept_}')

# Predict the price for a new house with 6 rooms and 40 years old
predicted_price_for_new_house = lasso_model.predict([[6, 40]])
print(f'Predicted price for a house with 6 rooms and 40 years old:
${predicted_price_for_new_house[0]:.2f} thousand')
```

NOTE

- Coefficient: This is a number that shows how strong the relationship is between the number of rooms and the house price.

- Intercept: This is the starting point or the base price of the house.

For example:

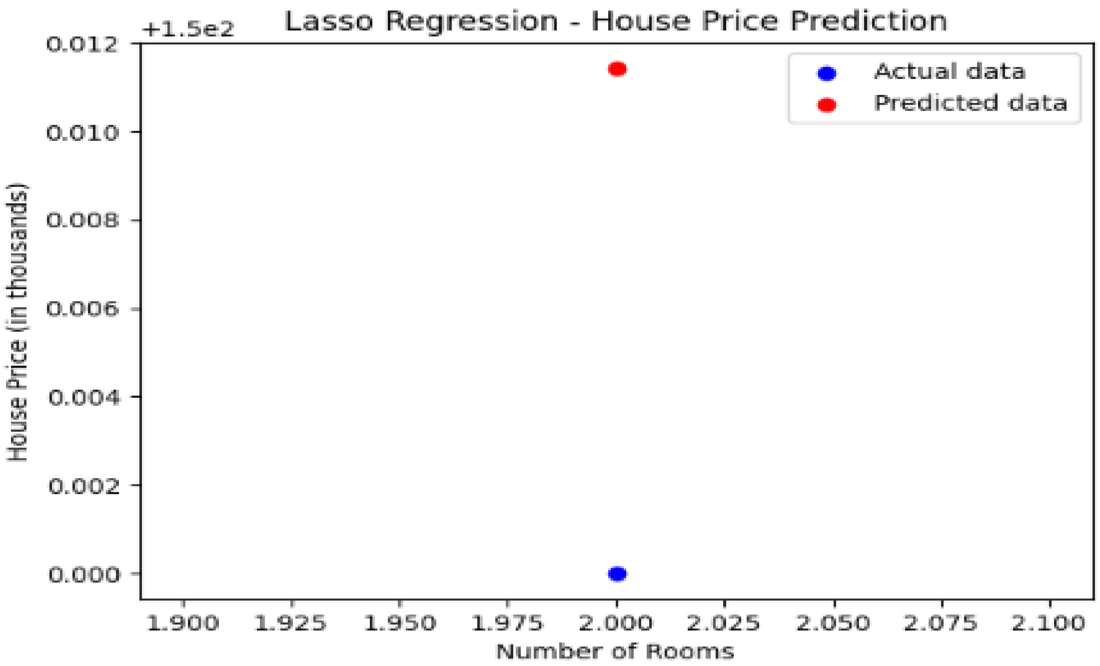
- Coefficient: 50
- Intercept: 100

This means that for every extra room, the price increases by 50. And the starting price (intercept) is 100.

So, if you have:

- 1 room: Price = 100 + (50 x 1) = 150
- 2 rooms: Price = 100 + (50 x 2) = 200
- 3 rooms: Price = 100 + (50 x 3) = 250

OUTPUT:




```
Coefficients: [13.41942857  7.31428571]
Intercept: 13.458285714284898
Predicted price for a house with 6 rooms and 40 years old: $386.55 thousand
```

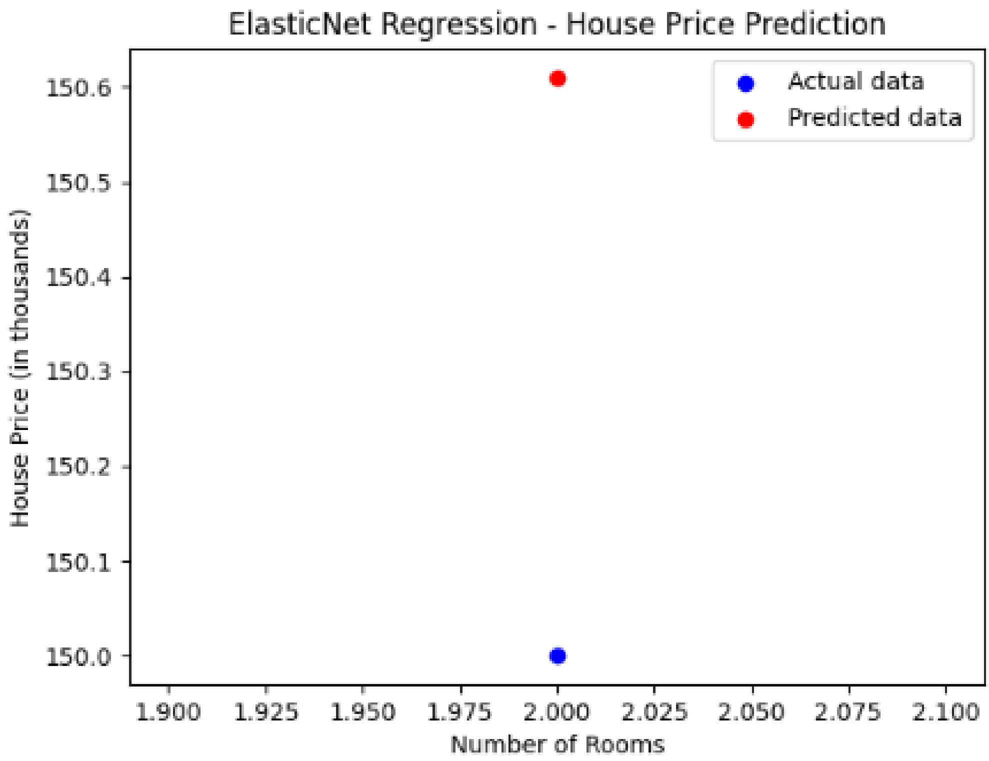
ELASTICNET REGRESSION:

Code:

```
# Install required libraries
!pip install scikit-learn matplotlib

# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import ElasticNet
from sklearn.model_selection import train_test_split

# Sample dataset: features (X) = [number of rooms, house age], target (y) = house price
# X = number of rooms, house age, y = house price in thousands of dollars
X = np.array([[1, 10], [2, 15], [3, 20], [4, 25], [5, 30]])  # Features: [rooms, age]
y = np.array([100, 150, 200, 250, 300])  # House prices in thousands

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train the ElasticNet regression model (combination of Lasso and Ridge)
elasticnet_model = ElasticNet(alpha=1.0, l1_ratio=0.5)  # alpha controls the strength, l1_ratio controls the
mix
elasticnet_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = elasticnet_model.predict(X_test)

# Visualize the actual vs predicted prices (for simplicity, we only plot one feature vs price)
```

```
plt.scatter(X_test[:, 0], y_test, color='blue', label='Actual data')
plt.scatter(X_test[:, 0], y_pred, color='red', label='Predicted data')
plt.xlabel('Number of Rooms')
plt.ylabel('House Price (in thousands)')
plt.title('ElasticNet Regression - House Price Prediction')
plt.legend()
plt.show()

# Print the model's coefficients and intercept
print(f'Coefficients: {elasticnet_model.coef_}')
print(f'Intercept: {elasticnet_model.intercept_}')

# Predict the price for a new house with 6 rooms and 40 years old
predicted_price_for_new_house = elasticnet_model.predict([[6, 40]])
print(f'Predicted price for a house with 6 rooms and 40 years old:
${predicted_price_for_new_house[0]:.2f} thousand')
```

OUTPUT:




```
Coefficients: [1.1388874  9.67462594]
Intercept: 3.2128147279466646
Predicted price for a house with 6 rooms and 40 years old: $397.03 thousand
```

Practical 4

Q4A. Logistic Regression :Perform binary classification using logistic regression. Calculate accuracy, precision, recall, and understand the ROC curve**.**

Aim: Logistic Regression Perform binary classification using logistic regression. Calculate accuracy, precision, recall, and understand the ROC curve**.**

Code:

```
# Install necessary libraries
!pip install numpy pandas scikit-learn

# Import libraries
import numpy as np
import pandas as pd
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score, recall_score, classification_report

# Generate synthetic dataset
X, y = make_classification(n_samples=1000, n_features=10, n_classes=2, random_state=42)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create and train the logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate precision and recall
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the precision, recall, and classification report
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

OUTPUT:

```
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (1.26.4)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (2.2.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.6.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas) (2024.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.5.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
Precision: 0.8888888888888888
Recall: 0.8242424242424242

Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.87      0.84       135
           1       0.89      0.82      0.86       165

    accuracy                           0.85       300
   macro avg       0.85      0.85      0.85       300
weighted avg       0.85      0.85      0.85       300
```

Q4B. Implement and demonstrate k-nearest Neighbor algorithm. Read the training data from a .CSV file and build the model to classify a test sample. Print both correct and wrong predictions.

Aim: Implement and demonstrate k-nearest Neighbor algorithm. Read the training data from a .CSV file and build the model to classify a test sample. Print both correct and wrong predictions.

Code:

```
# Install necessary libraries
!pip install numpy pandas scikit-learn tensorflow

# Import libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, Flatten, Dense, MaxPooling1D
from tensorflow.keras.utils import to_categorical
from google.colab import files

# Upload the CSV file
uploaded = files.upload()

# Load the dataset
df = pd.read_csv(next(iter(uploaded.keys())))

# Split features and target
X = df[['feature1', 'feature2']].values
y = df['label'].values

# Standardize the features
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

```
# Reshape data for CNN (reshape to 2D for each sample [samples, timesteps, features])
X = X.reshape(X.shape[0], X.shape[1], 1)

# Convert labels to categorical (for binary classification)
y = to_categorical(y)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create the CNN model
model = Sequential()
model.add(Conv1D(64, kernel_size=2, activation='relu', input_shape=(X.shape[1], 1)))
# Adjust the pool_size to 1 to avoid negative dimension
model.add(MaxPooling1D(pool_size=1))  # Changed pool_size to 1
model.add(Flatten())
model.add(Dense(2, activation='softmax'))  # 2 classes for binary classification

# Compile the model
# ... (rest of the code remains the same)
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=2, validation_split=0.2)

# Make predictions on the test set
y_pred_prob = model.predict(X_test)
y_pred = np.argmax(y_pred_prob, axis=1)

# Calculate accuracy
accuracy = accuracy_score(np.argmax(y_test, axis=1), y_pred)
print(f"Accuracy: {accuracy}")

# Print both correct and incorrect predictions
print("\nCorrect Predictions:")
for i in range(len(y_test)):
    if np.argmax(y_test[i]) == y_pred[i]:
        print(f"Actual: {np.argmax(y_test[i])}, Predicted: {y_pred[i]} (Correct)")

print("\nIncorrect Predictions:")
for i in range(len(y_test)):
    if np.argmax(y_test[i]) != y_pred[i]:
        print(f"Actual: {np.argmax(y_test[i])}, Predicted: {y_pred[i]} (Incorrect)")
```

OUTPUT:

```
Saving knn_dataset.csv to knn_dataset.csv
Epoch 1/10
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 100ms/step - accuracy: 0.6750 - loss: 0.7047 - val_accuracy: 0.5000 - val_loss: 0.7398
Epoch 2/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.7375 - loss: 0.6579 - val_accuracy: 0.5000 - val_loss: 0.7243
Epoch 3/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.7375 - loss: 0.6145 - val_accuracy: 0.5000 - val_loss: 0.7079
Epoch 4/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.7375 - loss: 0.5869 - val_accuracy: 0.5000 - val_loss: 0.6914
Epoch 5/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.6125 - loss: 0.5988 - val_accuracy: 0.5000 - val_loss: 0.6750
Epoch 6/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.7375 - loss: 0.5361 - val_accuracy: 0.5000 - val_loss: 0.6589
Epoch 7/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.5500 - loss: 0.5805 - val_accuracy: 0.5000 - val_loss: 0.6433
Epoch 8/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.7125 - loss: 0.5854 - val_accuracy: 0.5000 - val_loss: 0.6287
Epoch 9/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 1.0000 - loss: 0.5366 - val_accuracy: 1.0000 - val_loss: 0.6144
Epoch 10/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 1.0000 - loss: 0.4911 - val_accuracy: 1.0000 - val_loss: 0.6010
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step
Accuracy: 0.6666666666666666

Correct Predictions:
Actual: 0, Predicted: 0 (Correct)
Actual: 1, Predicted: 1 (Correct)

Incorrect Predictions:
Actual: 0, Predicted: 1 (Incorrect)
```

Q4C. Build a decision tree classifier or regressor. Control hyperparameters like tree depth to avoid overfitting. Visualize the tree.

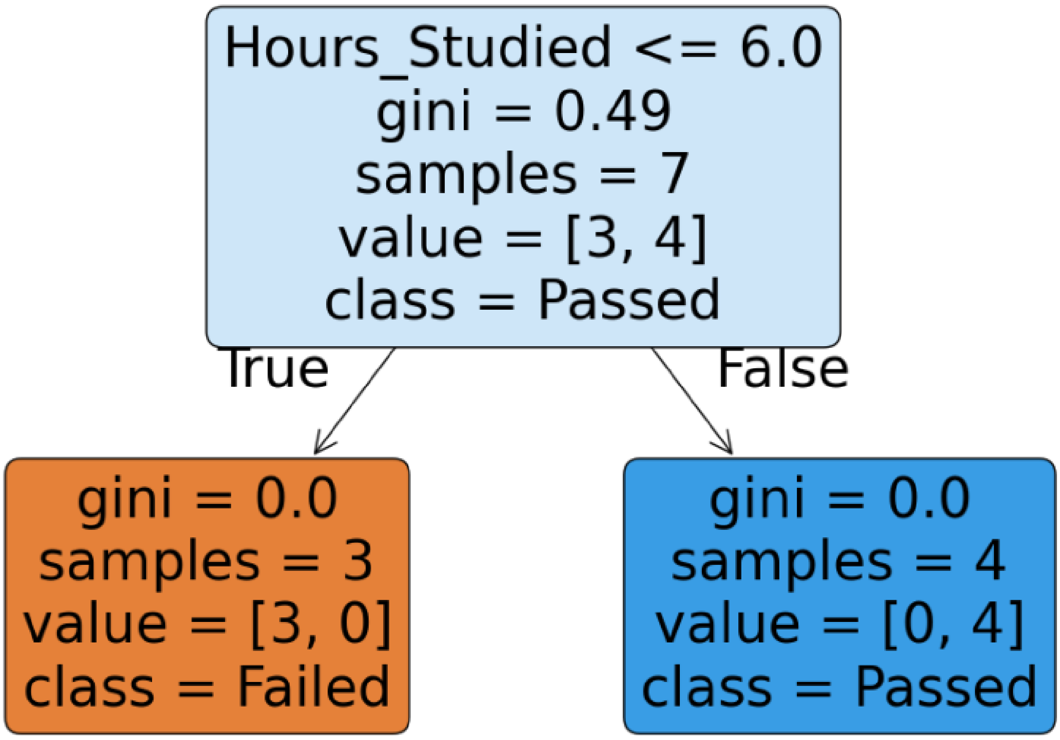
Aim: Build a decision tree classifier or regressor. Control hyperparameters like tree depth to avoid overfitting. Visualize the tree.

Code:

```
# Install necessary libraries
!pip install numpy pandas scikit-learn matplotlib

# Import libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
import matplotlib.pyplot as plt

# Create a simple dataset
data = {
    'Age': [15, 16, 17, 15, 18, 16, 19, 18, 17, 16],
    'Hours_Studied': [5, 6, 7, 4, 9, 5, 10, 8, 6, 5],
    'Passed': [0, 0, 1, 0, 1, 0, 1, 1, 0, 0]  # 0 = Failed, 1 = Passed
}

# Create DataFrame
df = pd.DataFrame(data)

# Split features and target
X = df[['Age', 'Hours_Studied']]
y = df['Passed']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
# Create and train the decision tree classifier
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)

# Visualize the decision tree
plt.figure(figsize=(12,8))
tree.plot_tree(clf, filled=True, feature_names=X.columns, class_names=['Failed', 'Passed'], rounded=True)
plt.show()

# Print the accuracy of the model
accuracy = clf.score(X_test, y_test)
print(f"Accuracy: {accuracy}")
```

OUTPUT:



Accuracy: 1.0

Q4D. Implement a Support Vector Machine for any relevant dataset.

Aim: Implement a Support Vector Machine for any relevant dataset.

Code:

```
# Install necessary libraries
!pip install numpy pandas scikit-learn

# Import libraries
import pandas as pd
from sklearn.model_selection import train_test_split
```

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Create a simple student-related dataset
data = {
    'marks_math': [85, 78, 92, 56, 65, 88, 73, 45, 91, 60],
    'marks_english': [80, 75, 88, 58, 64, 85, 70, 40, 89, 58],
    'passed': [1, 1, 1, 0, 0, 1, 1, 0, 1, 0]  # 1: Passed, 0: Failed
}

# Load data into a DataFrame
df = pd.DataFrame(data)

# Split features and target variable
X = df[['marks_math', 'marks_english']]
y = df['passed']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create and train the SVM model
model = SVC(kernel='linear')  # Linear kernel
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")

# Print predictions (both correct and incorrect)
print("\nCorrect Predictions:")
for i in range(len(y_test)):
    if y_test.iloc[i] == y_pred[i]:
        print(f"Actual: {y_test.iloc[i]}, Predicted: {y_pred[i]} (Correct)")

print("\nIncorrect Predictions:")
for i in range(len(y_test)):
    if y_test.iloc[i] != y_pred[i]:
        print(f"Actual: {y_test.iloc[i]}, Predicted: {y_pred[i]} (Incorrect)")
```

OUTPUT:

```
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (1.26.4)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (2.2.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.6.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas) (2024.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.5.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
Accuracy: 1.0

Correct Predictions:
Actual: 1, Predicted: 1 (Correct)
Actual: 1, Predicted: 1 (Correct)
Actual: 1, Predicted: 1 (Correct)

Incorrect Predictions:
```

Q4E. Implement a gradient boosting machine (e.g., XGBoost). Tune hyperparameters and explore feature importance.

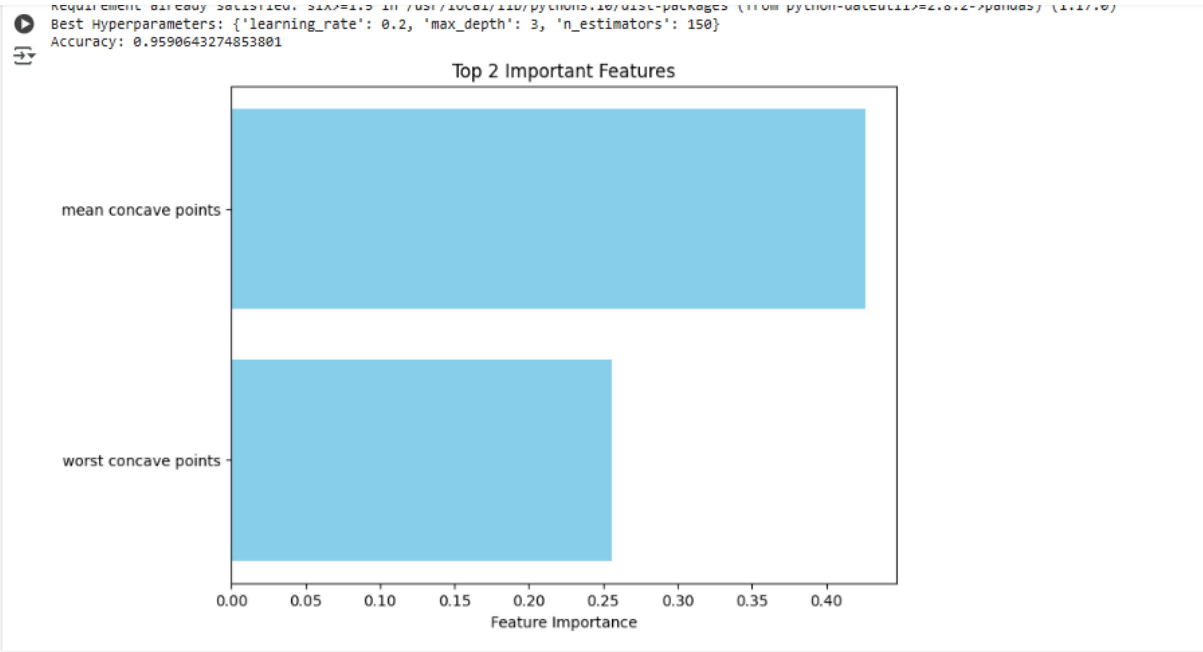
Aim: Implement a gradient boosting machine (e.g., XGBoost). Tune hyperparameters and explore feature importance.

Code:

```
# Install necessary libraries
!pip install numpy pandas scikit-learn

# Import libraries
import numpy as np
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Load the dataset
data = load_breast_cancer()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = data.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create the Gradient Boosting model
gbm = GradientBoostingClassifier()

# Hyperparameter tuning using GridSearchCV
param_grid = {
    'n_estimators': [50, 100, 150],
    'learning_rate': [0.05, 0.1, 0.2],
    'max_depth': [3, 5, 7]
}

grid_search = GridSearchCV(gbm, param_grid, cv=3, n_jobs=-1)
grid_search.fit(X_train, y_train)

# Best hyperparameters
print(f"Best Hyperparameters: {grid_search.best_params_}")

# Train the model with best hyperparameters
best_gbm = grid_search.best_estimator_

# Make predictions on the test set
y_pred = best_gbm.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
```

```
print(f"Accuracy: {accuracy}")

# Feature importance (Explore 2 important features)
importances = best_gbm.feature_importances_

# Get indices of top 2 important features
top_2_features_idx = np.argsort(importances)[-2:]

# Plot the feature importances
plt.figure(figsize=(8, 6))
plt.barh(X.columns[top_2_features_idx], importances[top_2_features_idx], color='skyblue')
plt.xlabel('Feature Importance')
plt.title('Top 2 Important Features')
plt.show()
```

OUTPUT:



Practical 5

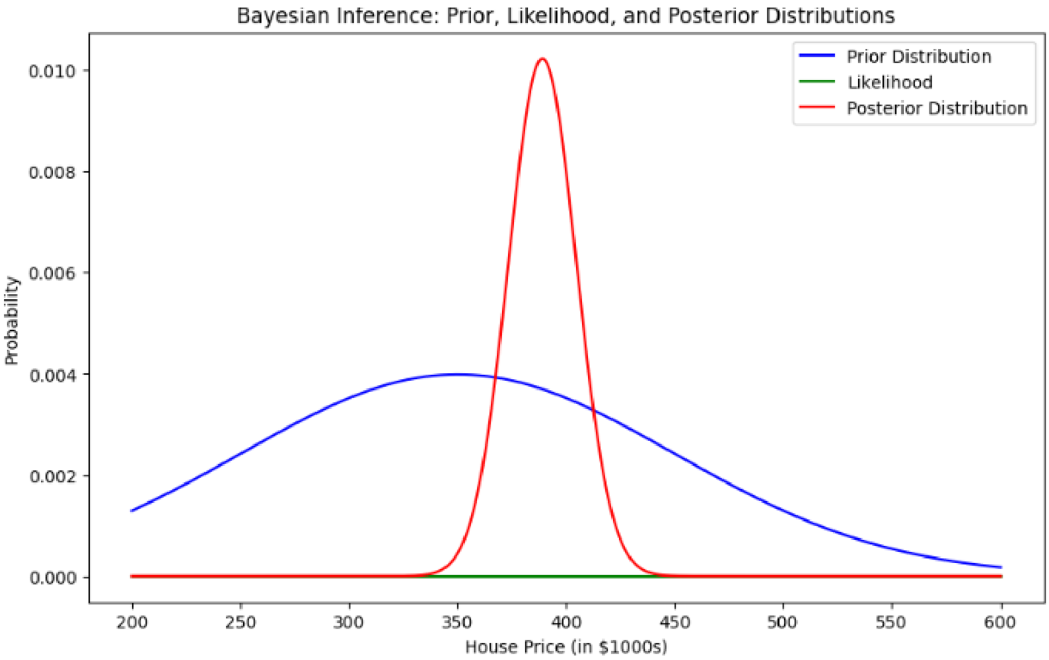
Q5a. Implement Bayesian Linear Regression to explore prior and posterior distribution.

Aim: Implement Bayesian Linear Regression to explore prior and posterior distribution.

Code:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

# Define the prior distribution
def prior_distribution(x):
    return norm.pdf(x, loc=350, scale=100)  # Prior: N(350, 100^2)

# Define the likelihood
def likelihood(x, observed_data):
    return np.prod([norm.pdf(d, loc=x, scale=50) for d in observed_data])

# Define the posterior distribution
def posterior_distribution(x, observed_data):
    return prior_distribution(x) * likelihood(x, observed_data)

# Observed data (house prices in thousands of dollars)
observed_data = [300, 320, 340, 360, 380, 400, 420, 440, 460, 480]

# Generate values for x (house prices)
x_values = np.linspace(200, 600, 1000)

# Calculate prior, likelihood, and posterior distributions
prior_values = prior_distribution(x_values)
likelihood_values = [likelihood(x, observed_data) for x in x_values]
posterior_values = [posterior_distribution(x, observed_data) for x in x_values]

# Normalize posterior distribution
posterior_values /= np.sum(posterior_values)

# Plot the distributions
plt.figure(figsize=(10, 6))
plt.plot(x_values, prior_values, label='Prior Distribution', color='blue')
plt.plot(x_values, likelihood_values, label='Likelihood', color='green')
plt.plot(x_values, posterior_values, label='Posterior Distribution', color='red')
plt.xlabel('House Price (in $1000s)')
plt.ylabel('Probability')
plt.title('Bayesian Inference: Prior, Likelihood, and Posterior Distributions')
plt.legend()
plt.show()
```

OUTPUT:



Q5b. Implement Gaussian Mixture Models for density estimation and unsupervised clustering

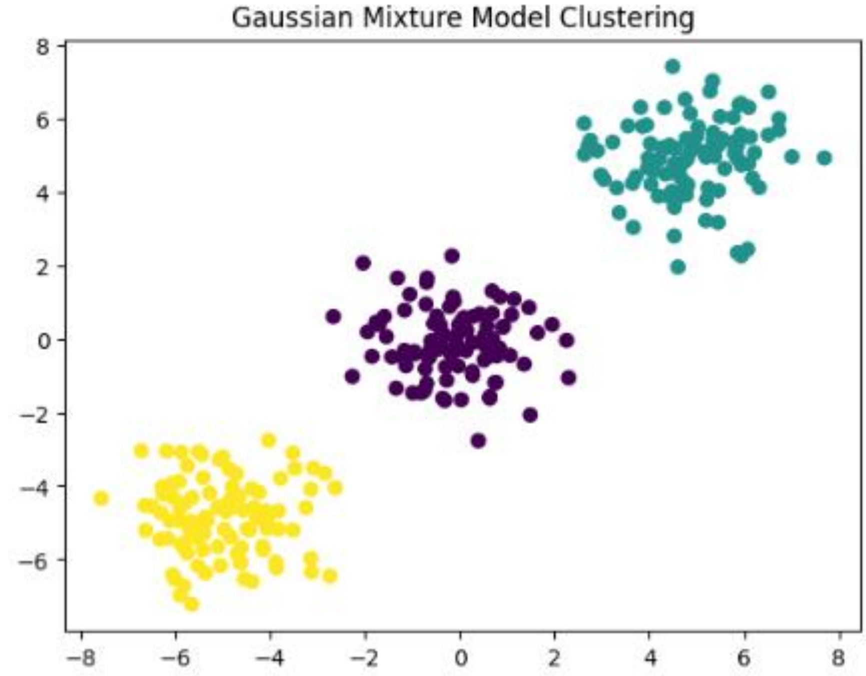
Aim: Implement Gaussian Mixture Models for density estimation and unsupervised clustering

Code:

```
# Importing necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture

# Generating synthetic data
np.random.seed(0)
n_samples = 300
X = np.vstack([
    np.random.normal(loc=-5, scale=1, size=(n_samples//3, 2)),
    np.random.normal(loc=0, scale=1, size=(n_samples//3, 2)),
    np.random.normal(loc=5, scale=1, size=(n_samples//3, 2))
])

# Fitting Gaussian Mixture Model
gmm = GaussianMixture(n_components=3, covariance_type='full', random_state=0)
gmm.fit(X)

# Predicting the clusters
labels = gmm.predict(X)

# Plotting the results
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', marker='o')
plt.title('Gaussian Mixture Model Clustering')
```

```
plt.show()
```

OUTPUT:



Practical 6

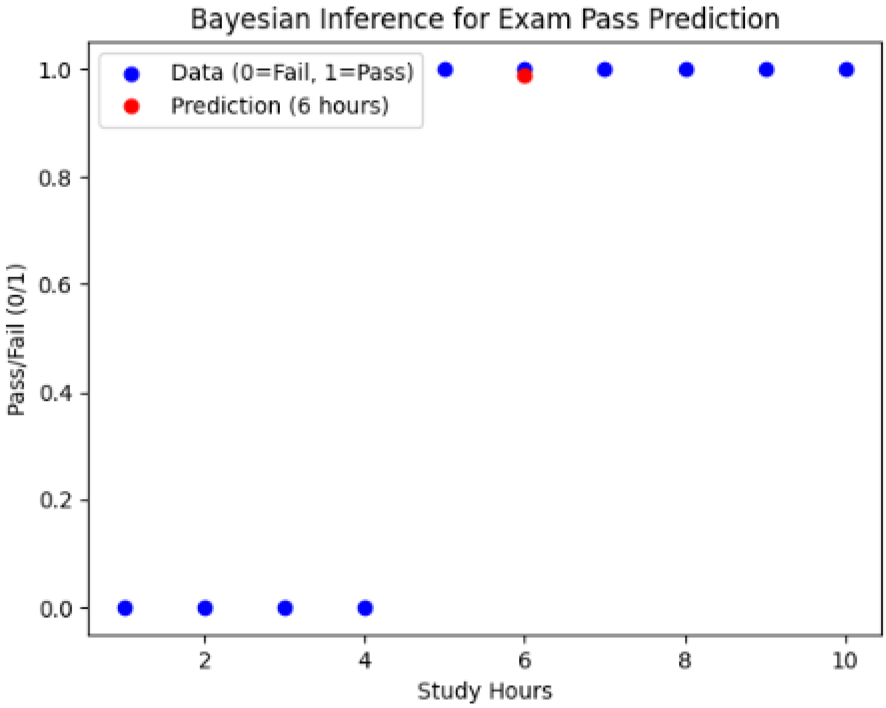
Q6. Implement Bayesian Learning using inferences

Aim: Implement Bayesian Learning using inferences

Code:

```
# Install required libraries
!pip install scikit-learn matplotlib

# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB

# Sample dataset: study hours (X) and pass/fail (y)
# X = number of study hours, y = outcome (0 = fail, 1 = pass)
X = np.array([[1], [2], [3], [4], [5], [6], [7], [8], [9], [10]])  # Study hours
y = np.array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1])  # Pass (1) or Fail (0)

# Initialize the Gaussian Naive Bayes model
model = GaussianNB()

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

# Predict the probability of passing for 6 study hours
study_hours = np.array([[6]])
predicted_prob = model.predict_proba(study_hours)

# Output the result
print(f'Probability of passing with 6 hours of study: {predicted_prob[0][1]:.2f}')

# Plot the data and predicted probability
plt.scatter(X, y, color='blue', label='Data (0=Fail, 1=Pass)')
plt.plot(study_hours, predicted_prob[0][1], 'ro', label='Prediction (6 hours)')
plt.xlabel('Study Hours')
plt.ylabel('Pass/Fail (0/1)')
plt.title('Bayesian Inference for Exam Pass Prediction')
plt.legend()
plt.show()
```

OUTPUT:


Bayesian Inference for Exam Pass Prediction
Data (0=Fail, 1=Pass)
Prediction (6 hours)
Pass/Fail (0/1)
1.0
0.8
0.6
0.4
0.2
0.0
2
4
6
8
10
Study Hours