MACHINE LEARNING

INDEX

Sr.no.	Practical	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 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 feature. Handle feature selection and potential multicollinearity **c.** Regualarized Linear Models 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 test sample. 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.** Train a random forest ensemble. Experiment with the number of trees and feature sampling. Compare performance to a single decision tree. **f.** Implement a gradient boosting machine (e.g., XGBoost). Tune hyperparameters and explore feature importance.		
5.	**Generative Models** **a.** Implement and demonstrate the working of a Naive Bayesian classifier using a sample data set. Build the model to classify a test sample. **b.** Implement Hidden Markov Models using hmmlearn		
6.	**Probabilistic Models** **a.** Implement Bayesian Linear Regression to explore prior and posterior distribution. **b.** Implement Gaussian Mixture Models for density estimation and unsupervised clustering.		
7.	**Model Evaluation and Hyperparameter Tuning** **a.** Implement cross-validation techniques (k-fold, stratified, etc.) for robust model evaluation		

	b. Systematically explore combinations of hyperparameters to optimize model performance.(use grid and randomized search)		
8.	**Bayesian Learning** **a.** Implement Bayesian Learning using inferences		

Practical 1: Data Pre-processing and Exploration

1a. Load a CSV dataset. Handle missing values, inconsistent formatting, and outliers.

Code :

1. **Import Libraries**

 # Import necessary libraries

import pandas as pd import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

2. **Load the Dataset**

 # Load the Titanic dataset from a URL

url="https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv" data = pd.read_csv(url)

 # Display the first few rows

print(data.head())

3. **Handle Missing Values**

 # Check for missing values

print("Missing values in each column:")

print(data.isnull().sum())

 # Fill missing values in 'Age' with the mean

data['Age'].fillna(data['Age'].mean(), inplace=True)

 # Fill missing values in 'Embarked' with the most common value
data['Embarked'].fillna(data['Embarked'].mode()[0], inplace=True)

 # Drop rows where 'Cabin' is missing (too many NaNs)

data.drop(columns=['Cabin'], inplace=True)

 # Verify missing values are handled

print("\nAfter handling missing values:")

print(data.isnull().sum())

4. **Fix Inconsistent Formatting**

 # Fix inconsistent formatting in the 'Sex' column

data['Sex'] = data['Sex'].str.lower().str.strip()

 # Verify unique values

print("\nUnique values in 'Sex' column after formatting:")

print(data['Sex'].unique())

5. **Detect and Handle Outliers # Boxplot for the 'Fare' column** sns.boxplot(data['Fare'], color='skyblue') plt.title('Boxplot of Fare') plt.show()

 # Detect outliers using the IQR method

Q1 = data['Fare'].quantile(0.25)
Q3 = data['Fare'].quantile(0.75)
IQR = Q3 - Q1 lower_bound =
Q1 - 1.5 * IQR upper_bound =
Q3 + 1.5 * IQR

 # Capping outliers

data['Fare'] = np.where(data['Fare'] > upper_bound, upper_bound, np.where(data['Fare'] <
lower_bound, lower_bound, data['Fare']))

 # Verify with an updated boxplot

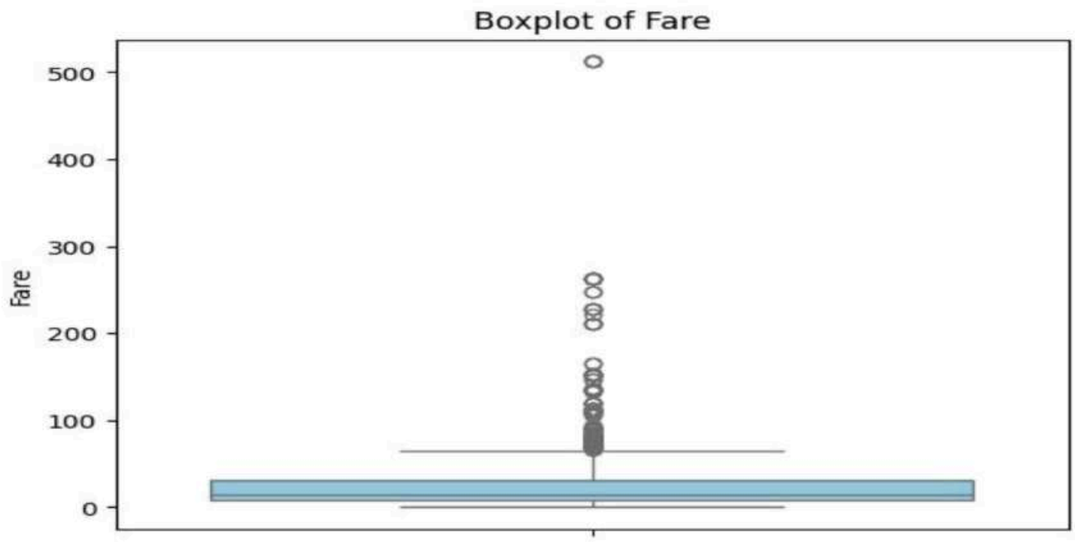
sns.boxplot(data['Fare'], color='lightgreen')

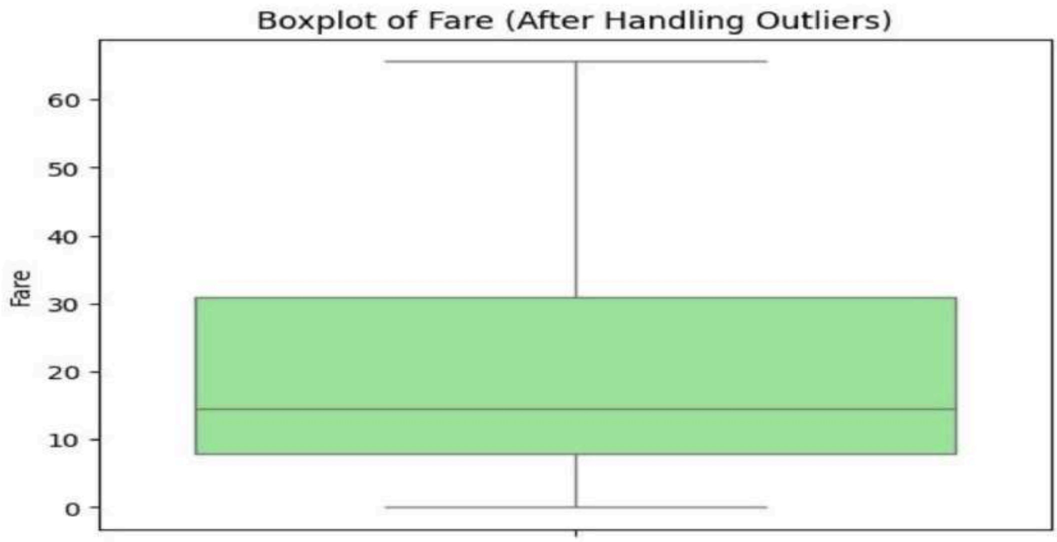
plt.title('Boxplot of Fare (After Handling Outliers)')

plt.show()

6. **Save the Cleaned Dataset # Save the cleaned dataset**
data.to_csv('cleaned_titanic.csv', index=False)
print("\nCleaned dataset saved as 'cleaned_titanic.csv'") .

Output :


Boxplot of Fare
Fare
0
100
200
300
400
500


Boxplot of Fare (After Handling Outliers)
Fare
0
10
20
30
40
50
60

1b. 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 :

1. Import Necessary Libraries # Import required libraries

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns
```

2. Load the Dataset

 # Load the dataset from the URL

```
url = "https://raw.githubusercontent.com/mwaskom/seaborn-data/master/iris.csv"

data = pd.read_csv(url)
```

 # Display the first few rows

```
print("First 5 rows of the dataset:")
print(data.head())
```

3. Calculate Descriptive Summary Statistics # Dataset information

```
print("\nDataset Info:")

print(data.info())
```

 # Summary statistics for numerical columns

```
print("\nDescriptive Statistics for Numerical Columns:")
print(data.describe())
```

 # Check unique values for categorical columns

```
print("\nUnique values in 'species' column:")
print(data['species'].value_counts())
```

4. Univariate Analysis

 # Histograms for numerical columns

data.hist(figsize=(10,8), color='skyblue', edgecolor='black')

plt.suptitle("Histograms of Numerical Features")

plt.show()

 # Bar plot for 'species' column
sns.countplot(x='species', data=data, palette='pastel')
plt.title("Count of Each Species") plt.show()

5. Bivariate Analysis

 # Scatter plot for two features

plt.figure(figsize=(8, 6))

plt.scatter(data['sepal_length'], data['sepal_width'], alpha=0.7, c='blue')

plt.title("Sepal Length vs Sepal Width")

plt.xlabel("Sepal Length")

plt.ylabel("Sepal Width")

plt.show()

 # Pairplot to visualize relationships between features
sns.pairplot(data, hue='species', palette='husl', diag_kind='kde')
plt.suptitle("Pairplot of Features by Species", y=1.02)

plt.show()

 # Boxplot for petal_length across species sns.boxplot(x='species',
y='petal_length', data=data, palette='Set3')

plt.title("Boxplot of Petal Length by Species")

plt.show()

6. Identify Potential Features and Target Variables

 # Separate features and target

features = data.drop(columns=['species'])

 # Drop the target

column target = data['species']

 # Target variable

print("\nFeatures:")

print(features.head())

print("\nTarget:")

print(target.head())

 # Visualize target distribution
sns.countplot(x=target, palette='viridis')
plt.title("Target Variable Distribution")

plt.show()

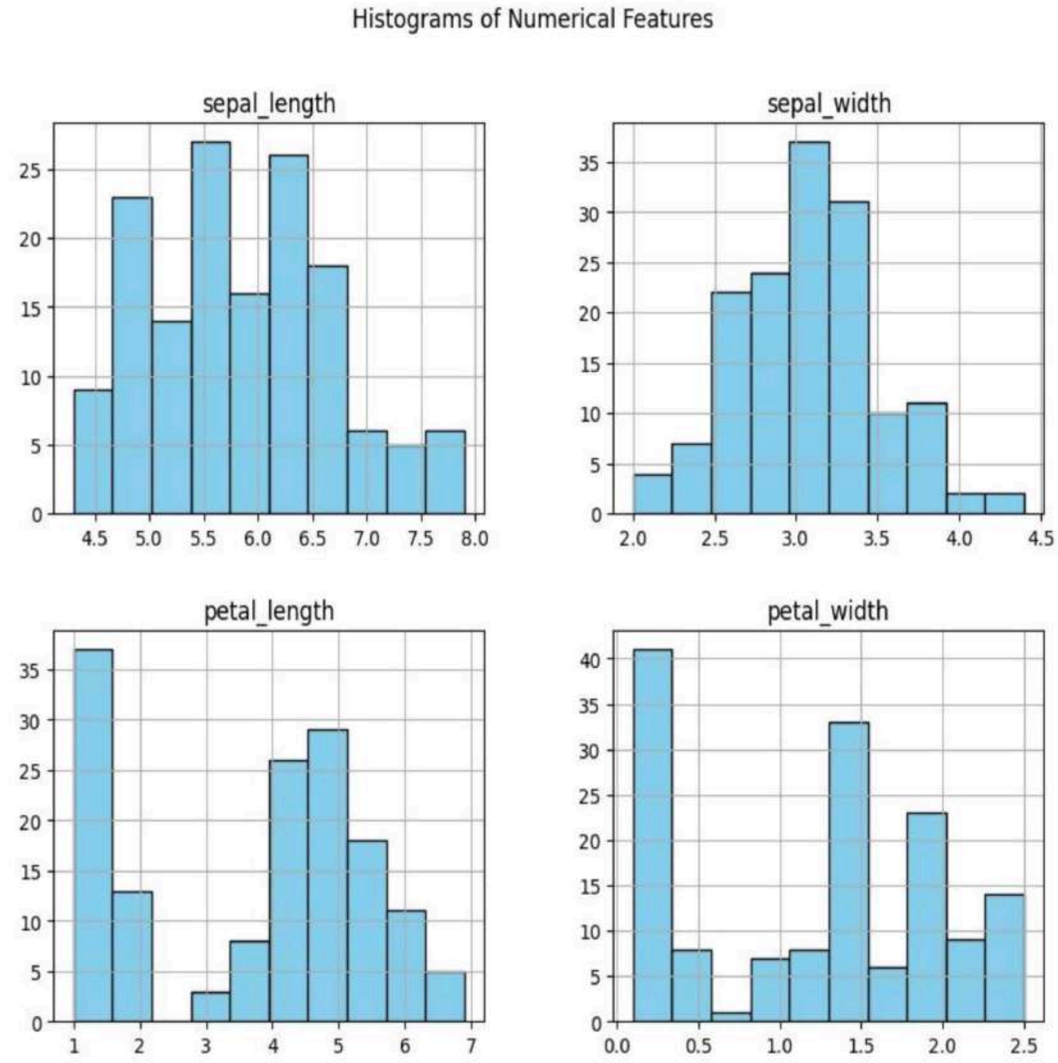
7. Save the Cleaned and Processed Dataset

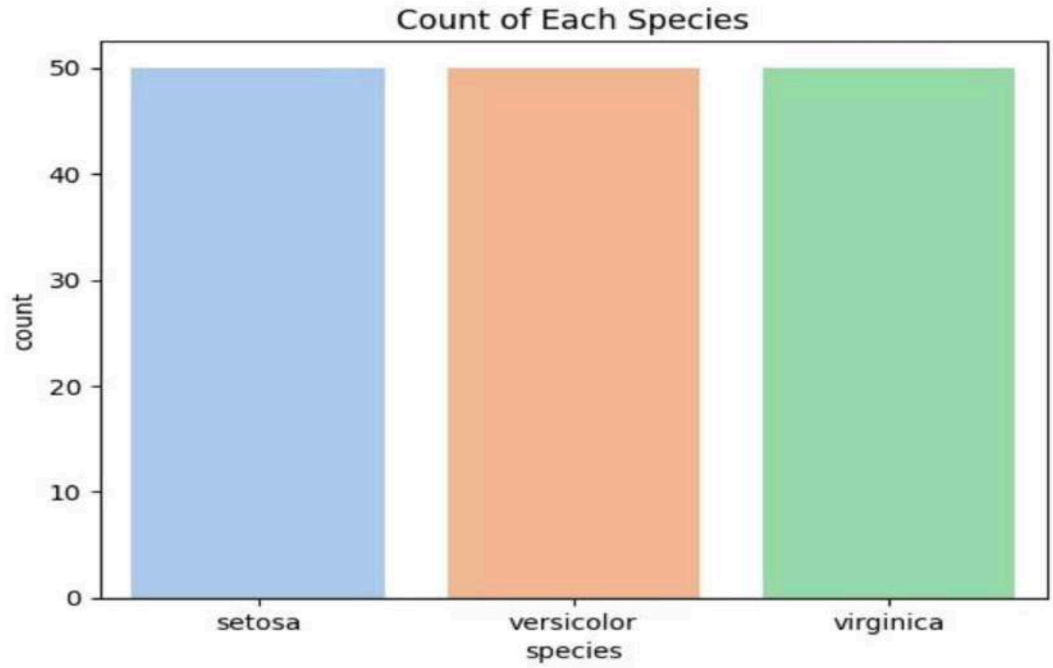
 # Save the dataset

data.to_csv('processed_iris.csv', index=False)

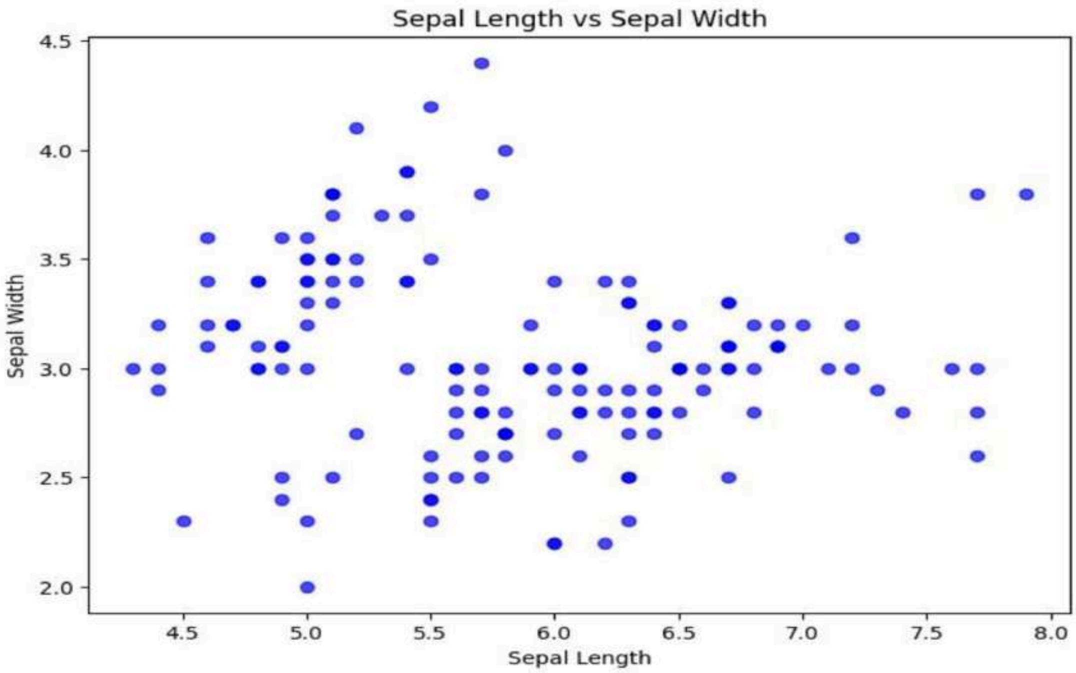
print("\nProcessed dataset saved as 'processed_iris.csv'")

Output :

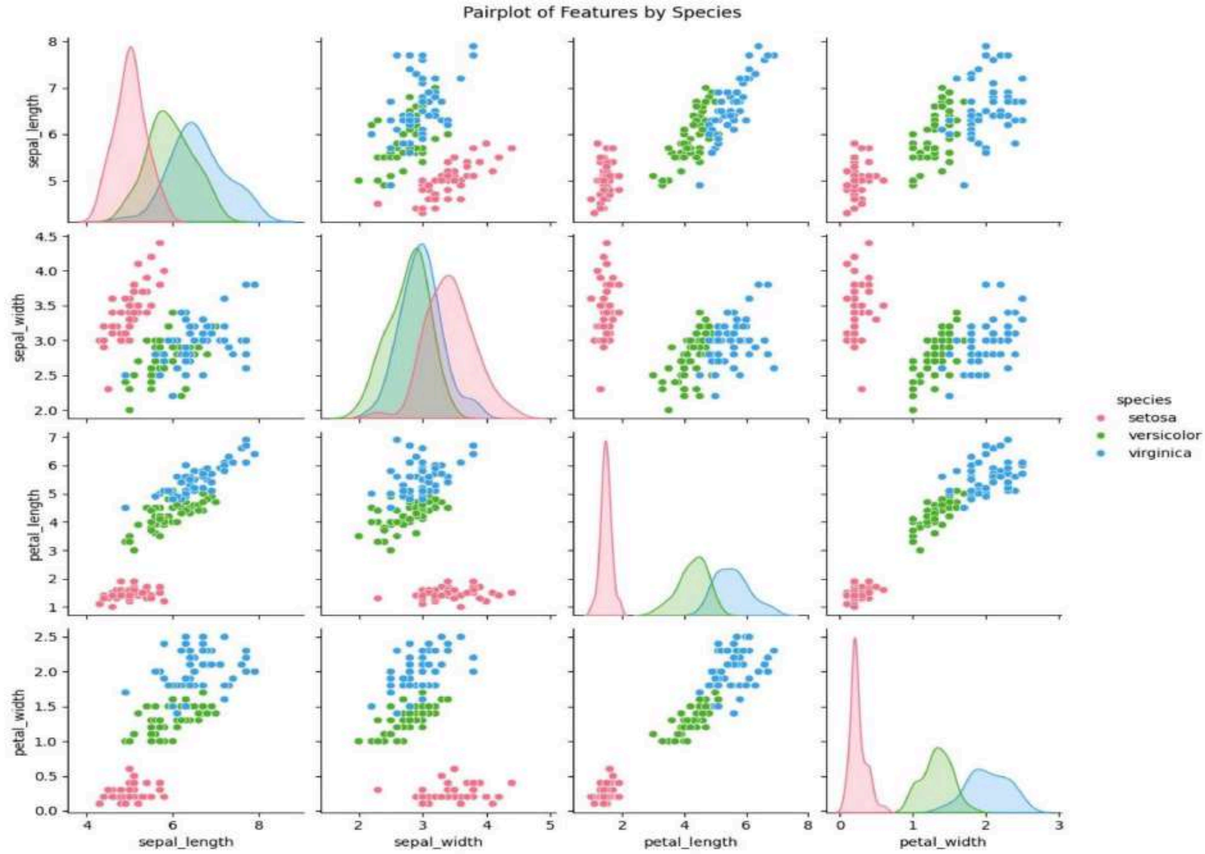


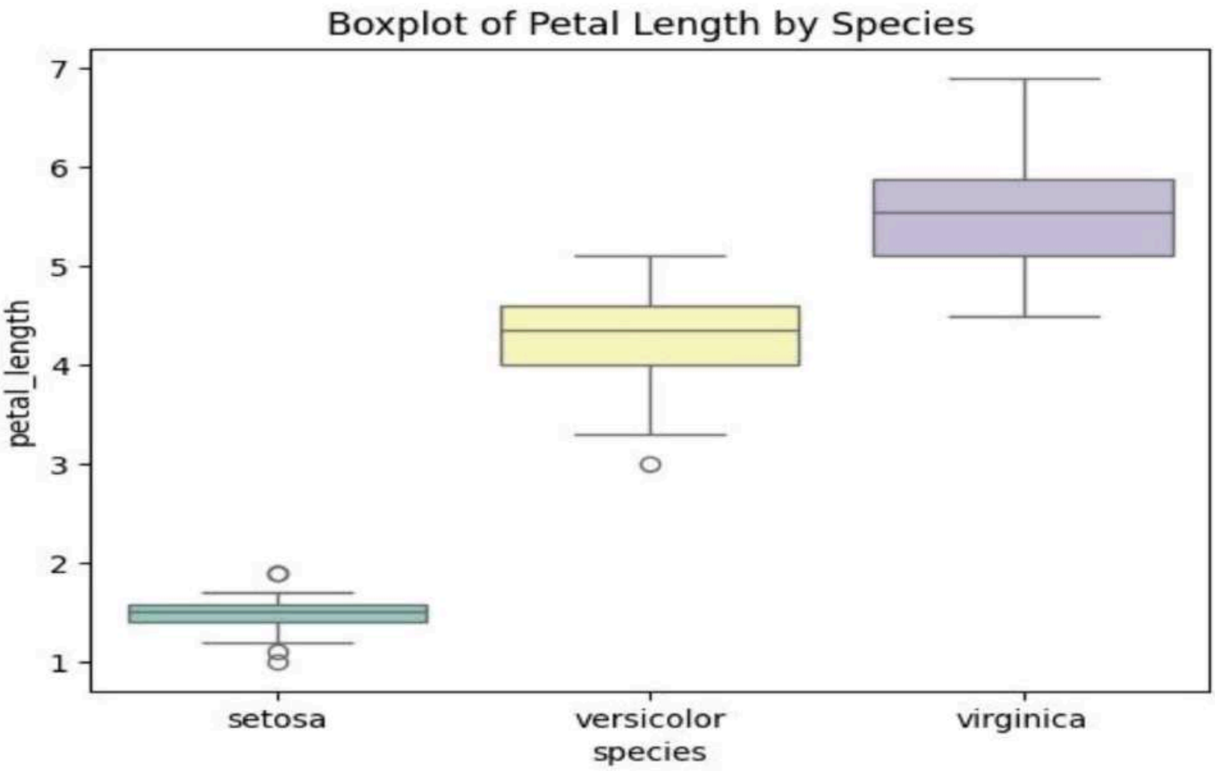


Count of Each Species
count
species
setosa
versicolor
virginica



Sepal Length vs Sepal Width
Sepal Width
Sepal Length


Pairplot of Features by Species
sepal_length
sepal_width
petal_length
petal_width
species
setosa
versicolor
virginica


Boxplot of Petal Length by Species
petal_length
setosa
versicolor
virginica
species

1c. Create or Explore datasets to use all pre-processing routines like label encoding, scaling, and binarization.

Code :

1. **Import Necessary Libraries # Import required libraries**

import pandas as pd

import numpy as np from sklearn.preprocessing

import LabelEncoder, MinMaxScaler, StandardScaler, Binarizer

2. **Create or Load a Dataset # Create a sample dataset**

data = pd.DataFrame({

'Category': ['A', 'B', 'C', 'A', 'B', 'C'],

 # Categorical variable

'Age': [23, 45, 31, 22, 35, 30],

 # Numerical variable

'Income': [50000, 60000, 70000, 80000, 90000, 100000],

 # Numerical variable 'Has_Car':

['Yes', 'No', 'Yes', 'No', 'Yes', 'No']

 # Binary categorical variable })

 # Display the dataset

print("Sample Dataset:")

print(data)

3. **Apply Pre-Processing Routines**

 # Label Encoding for 'Category' column

label_encoder = LabelEncoder()

data['Category_Encoded'] =

label_encoder.fit_transform(data['Category'])

 # Label Encoding for binary column 'Has_Car'

data['Has_Car_Encoded'] =
label_encoder.fit_transform(data['Has_Car']) print("\nAfter Label
Encoding:")

print(data)

 # Min-Max Scaling for 'Income'

min_max_scaler = MinMaxScaler()

data['Income_MinMax'] = min_max_scaler.fit_transform(data[['Income']])

 # Standard Scaling for 'Age'

standard_scaler = StandardScaler()

data['Age_Standardized'] =

standard_scaler.fit_transform(data[['Age']]) print("\nAfter Scaling:")

print(data)

 # Binarization for 'Income' with a threshold of 75,000

binarizer = Binarizer(threshold=75000)

data['Income_Binary'] =
binarizer.fit_transform(data[['Income']]) print("\nAfter
Binarization:")

print(data)

4. Save the Processed Dataset

 # Save the processed dataset

data.to_csv('processed_data.csv', index=False)
print("\nProcessed dataset saved as
'processed_data.csv'")

Output :

```
Sample Dataset:
  Category  Age  Income Has_Car
0        A   23   50000     Yes
1        B   45   60000      No
2        C   31   70000     Yes
3        A   22   80000      No
4        B   35   90000     Yes
5        C   30  100000      No
```

```
After Label Encoding:
  Category  Age  Income Has_Car  Category_Encoded  Has_Car_Encoded
0        A   23   50000     Yes                 0                1
1        B   45   60000      No                 1                0
2        C   31   70000     Yes                 2                1
3        A   22   80000      No                 0                0
4        B   35   90000     Yes                 1                1
5        C   30  100000      No                 2                0
```

```
After Scaling:
  Category  Age  Income Has_Car  Category_Encoded  Has_Car_Encoded  \
0        A   23   50000     Yes                 0                1
1        B   45   60000      No                 1                0
2        C   31   70000     Yes                 2                1
3        A   22   80000      No                 0                0
4        B   35   90000     Yes                 1                1
5        C   30  100000      No                 2                0

   Income_MinMax  Age_Standardized
0            0.0         -1.035676
1            0.2          1.812434
2            0.4          0.000000
3            0.6         -1.165136
4            0.8          0.517838
5            1.0         -0.129460
```

```
After Binarization:
  Category  Age  Income Has_Car  Category_Encoded  Has_Car_Encoded  \
0        A   23   50000     Yes                 0                1
1        B   45   60000      No                 1                0
2        C   31   70000     Yes                 2                1
3        A   22   80000      No                 0                0
4        B   35   90000     Yes                 1                1
5        C   30  100000      No                 2                0

   Income_MinMax  Age_Standardized  Income_Binary
0            0.0         -1.035676              0
1            0.2          1.812434              0
2            0.4          0.000000              0
3            0.6         -1.165136              1
4            0.8          0.517838              1
5            1.0         -0.129460              1
```

Practical 2 : Testing Hypothesis

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

# Step 1: Create the Dataset and Load It

data = {'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy', 'Rainy', 'Overcast', 'Sunny', 'Sunny', 'Rainy'],

'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool', 'Mild', 'Cool', 'Mild'],

'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal', 'Normal'],

'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak', 'Weak', 'Weak'],

'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes']

}


# Load dataset into a pandas DataFrame

df = pd.DataFrame(data)

    # Step 2: Implementing the FIND-S Algorithm

    def find_s_algorithm(data):

      # Get the positive examples (PlayTennis = 'Yes')

      positive_examples = data[data['PlayTennis'] == 'Yes']

      # Initialize hypothesis with the first positive example (most specific)

      hypothesis = positive_examples.iloc[0].drop('PlayTennis')

      # Loop through the rest of the positive examples and generalize the
    hypothesis

      for index, row in positive_examples.iterrows():

        for feature in hypothesis.index:

          if hypothesis[feature] != row[feature]:

            hypothesis[feature] = '?'
```

return hypothesis

Step 3: Apply FIND-S to the dataset

hypothesis = find_s_algorithm(df)

 # Display the final specific hypothesis

print("The most specific hypothesis is:")

print(hypothesis)

Output :

```
Dataset:
     Sky Temperature Humidity   Wind Water Forecast Condition
0  Sunny        Warm   Normal Strong  Warm     Same       Yes
1  Sunny        Cold     High Strong  Warm     Same        No
2  Rainy        Warm     High   Weak  Cool   Change        No
3  Sunny        Warm   Normal Strong  Warm     Same       Yes
4  Rainy        Cold   Normal   Weak  Cool   Change        No
```

```
Loaded Dataset:
     Sky Temperature Humidity   Wind Water Forecast Condition
0  Sunny        Warm   Normal Strong  Warm     Same       Yes
1  Sunny        Cold     High Strong  Warm     Same        No
2  Rainy        Warm     High   Weak  Cool   Change        No
3  Sunny        Warm   Normal Strong  Warm     Same       Yes
4  Rainy        Cold   Normal   Weak  Cool   Change        No
```

```
Final Specific Hypothesis:
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
```

Practical 3 : Linear Models

3a. 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 :

Step 1: Import Libraries # Import required libraries

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error, r2_score
```

Step 2: Create a Dataset and Save as CSV

 # Create a sample dataset

```
data = {

   'House_Size': [750, 800, 850, 900, 1000, 1100, 1200, 1300, 1400, 1500],

   'Price': [150000, 160000, 165000, 170000, 180000, 190000, 200000, 210000, 220000,
230000]

}
```

 # Convert the dataset into a DataFrame

```
df = pd.DataFrame(data)
```

 # Save to CSV file

```
df.to_csv('house_prices.csv',
index=False)
```

 # Display the dataset

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

Step 3: Load the Dataset

 # Load the dataset

dataset = pd.read_csv('house_prices.csv')
 # Display the first few

rows print("\nLoaded

Dataset:")

print(dataset.head())

Step 4: Split the Dataset into Training and Test Sets

 # Features and target variable

X = dataset[['House_Size']] **# Feature: House size**

y = dataset['Price'] **# Target: Price**

 # Split data into training and testing sets (80% train, 20% test)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print("\nTraining and Testing Data Sizes:")

print("Training Data Size:", X_train.shape[0])

print("Testing Data Size:", X_test.shape[0])

Step 5: Fit a Linear Regression Model

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

 # Display the coefficients

print("\nModel Coefficients:")

print("Slope (m):", model.coef_[0])
print("Intercept (b):", model.intercept_)

Step 6: Make Predictions

 # Predict on the test set

y_pred = model.predict(X_test)
 # Display predictions
print("\nPredictions on Test Data:")
print("Actual Prices:", y_test.values)
print("Predicted Prices:", y_pred)

Step 7: Evaluate the Model

 # Calculate evaluation metrics

mse = mean_squared_error(y_test, y_pred) r2 = r2_score(y_test, y_pred)

Display metrics

print("\nModel Performance Metrics:")
print("Mean Squared Error (MSE):", mse)
print("R-squared (R²):", r2)

Step 8: Visualize the Results # Scatter plot of the training data plt.scatter(X_train, y_train, color='blue', label='Training Data')

 # Plot the regression line

plt.plot(X_train, model.predict(X_train), color='red', label='Regression Line')
 # Scatter plot of the test data

plt.scatter(X_test, y_test, color='green', label='Test Data')

plt.title("Simple Linear Regression")

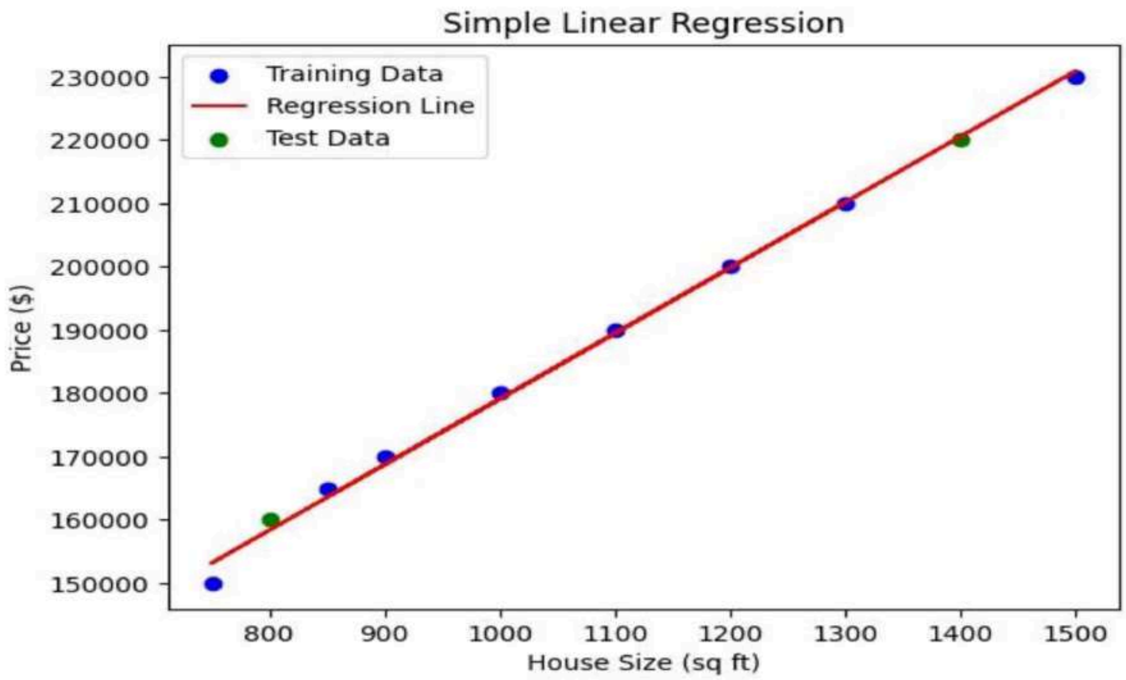
plt.xlabel("House Size (sq ft)")

plt.ylabel("Price ($)")

plt.legend()

plt.show()

Output :



3b. Multiple Linear Regression :

Extend linear regression to multiple feature. Handle feature selection and potential multicollinearity

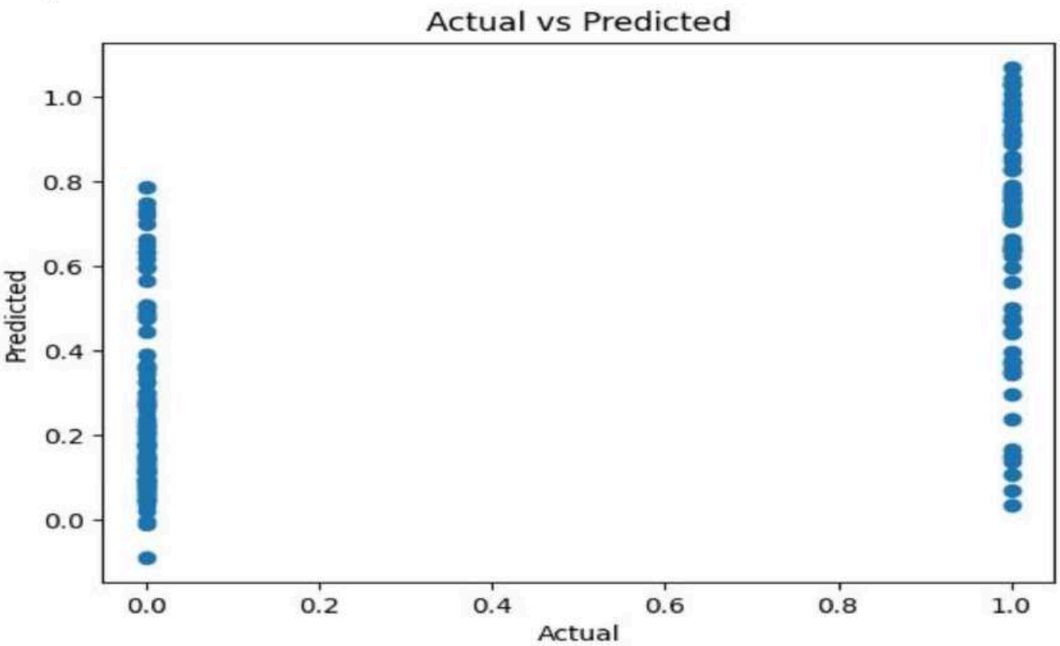
Code :

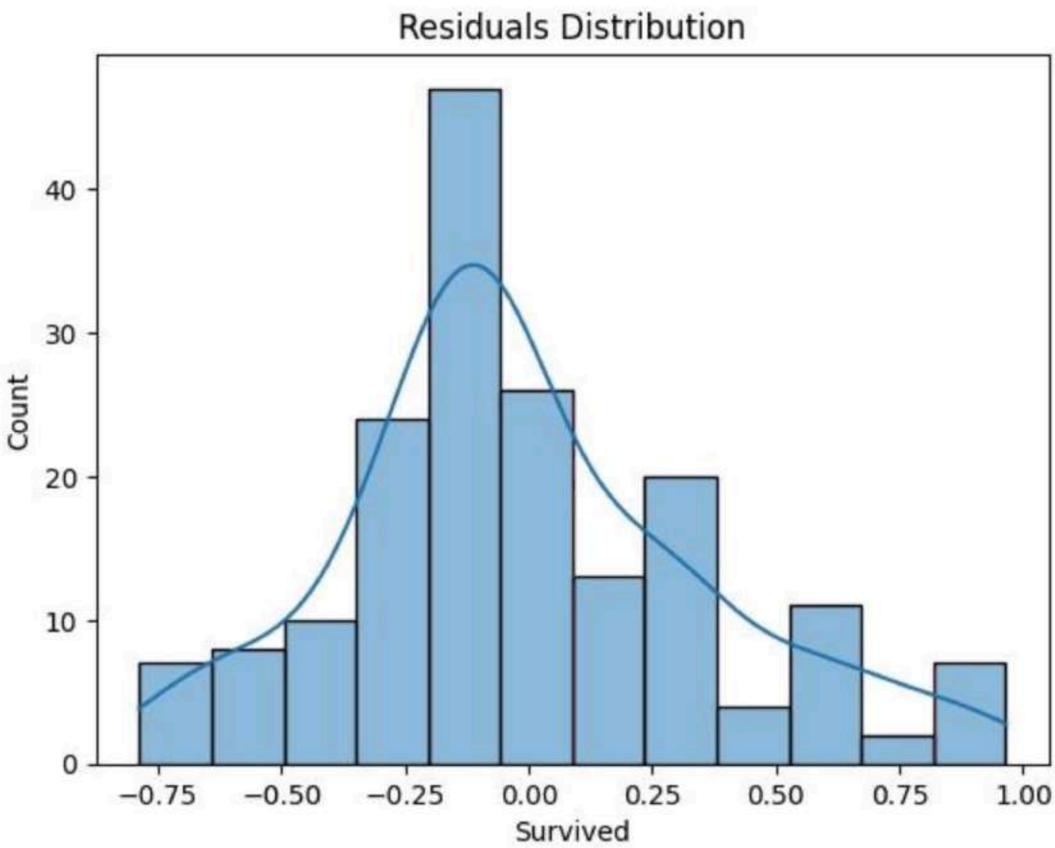
```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error, r2_score

from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.preprocessing import LabelEncoder

# Import LabelEncoder

from sklearn.impute import SimpleImputer


# Load dataset

from google.colab import files uploaded = files.upload()  # Upload your CSV file


# Read the CSV file

data =pd.read_csv(list(uploaded.keys())[0])

# Display the first few rows
print(data.head())

# Check for null values and basic statistics
print(data.info())
print(data.describe())

# Define a function to calculate VIF
def calculate_vif(df):

# Select only numeric features for VIF calculation
numeric_df = df.select_dtypes(include=np.number)

# Drop rows with infinite or missing values
```

```
numeric_df = numeric_df.replace([np.inf, -np.inf], np.nan).dropna()
vif_data = pd.DataFrame()
vif_data["feature"] = numeric_df.columns
vif_data["VIF"] = [variance_inflation_factor(numeric_df.values, i)
for i in range(numeric_df.shape[1])]
  return vif_data

# Selecting features and target variable
X = data.drop("Survived", axis=1)
# Changed 'y' to 'Survived' y = data["Survived"]

# Handle categorical features (e.g., using Label Encoding)
for col in X.select_dtypes(include=['object']).columns:
  le = LabelEncoder()
  X[col] = le.fit_transform(X[col])

# Impute missing values using the mean (you can choose other strategies)
imputer = SimpleImputer(strategy='mean')
# Create an imputer instance
X = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)
# Impute and update X

# Calculate VIF for initial features
print("VIF before handling multicollinearity:")
print(calculate_vif(X)) # Call the modified function

# Drop features based on VIF analysis (example: drop 'X1' if VIF is high)
# Check if the column exists before dropping
if 'X1' in X.columns:
  X = X.drop("X1", axis=1)  # Replace 'X1' with the actual high VIF feature name
else:
  print("Column 'X1' not found in the DataFrame.")

# Recalculate VIF
print("VIF after handling multicollinearity:")
print(calculate_vif(X))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and fit the model
model = LinearRegression()
model.fit(X_train, y_train)

# Get coefficients and intercept
print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)

# Predictions
y_pred = model.predict(X_test)

# Evaluation metrics
```

```
rmse = np.sqrt(mean_squared_error(y_test, y_pred)) r2 = r2_score(y_test, y_pred)
print(f"RMSE: {rmse}")
print(f"R^2: {r2}")
from sklearn.feature_selection import RFE

# Recursive Feature Elimination
rfe = RFE(estimator=LinearRegression(), n_features_to_select=5)

# Adjust features
rfe.fit(X_train, y_train)

# Selected features
print("Selected Features:", X.columns[rfe.support_])

# Scatter plot of actual vs predicted values
plt.scatter(y_test, y_pred) plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title("Actual vs Predicted")
plt.show()

# Residuals
residuals = y_test - y_pred sns.histplot(residuals, kde=True)
plt.title("Residuals Distribution")
plt.show()
```

Output :





Residuals Distribution
Count
Survived
40
30
20
10
0
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

3c. Regualarized Linear Models :

Implement Regression variants like LASSO and Ridge on any generated dataset

Code :

1. **Set Up the Environment**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge, Lasso, ElasticNet
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.datasets import make_regression
# Set random seed for reproducibility
np.random.seed(42)
```

2. **Generate a Synthetic Dataset**

```
# Generate synthetic data
X, y = make_regression(n_samples=1000,
# Number of samples
 n_features=10,
# Number of features
noise=15,
# Add some noise
 random_state=42
)
# Convert to DataFrame for exploration
data = pd.DataFrame(X, columns=[f"X{i}"
for i in range(1, 11)]) data["y"] = y
# Display the first few rows
```

print(data.head())

3. **Split the Dataset**

 # Split data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(data.drop("y", axis=1),

 # Features

data["y"],

 # Target variable

test_size=0.2,

 # 20% for testing

random_state=42

)

4. **Train and Evaluate Ridge Regression**

 # Initialize Ridge Regression with a regularization parameter (alpha)

ridge = Ridge(alpha=1.0)

 # Train the model

ridge.fit(X_train, y_train)

 # Predictions

ridge_pred = ridge.predict(X_test)

 # Evaluate Ridge Regression

ridge_rmse = np.sqrt(mean_squared_error(y_test, ridge_pred))

ridge_r2 = r2_score(y_test, ridge_pred)

print(f"Ridge RMSE: {ridge_rmse}")

print(f"Ridge R^2: {ridge_r2}")

5. **Train and Evaluate Lasso Regression**

 # Initialize Lasso Regression

lasso = Lasso(alpha=0.1)

 # Train the model

lasso.fit(X_train, y_train)

 # Predictions

lasso_pred = lasso.predict(X_test)

 # Evaluate Lasso Regression

lasso_rmse = np.sqrt(mean_squared_error(y_test, lasso_pred))

lasso_r2 = r2_score(y_test, lasso_pred)

print(f"Lasso RMSE: {lasso_rmse}")

print(f"Lasso R^2: {lasso_r2}")

 # Features shrunk to

zero print("Lasso Coefficients:", lasso.coef_)

6. Train and Evaluate ElasticNet Regression

 # Initialize ElasticNet

elastic_net = ElasticNet(alpha=0.1, l1_ratio=0.5) # l1_ratio balances L1 and L2 penalties

 # Train the model

elastic_net.fit(X_train, y_train)

 # Predictions

elastic_net_pred = elastic_net.predict(X_test)

 # Evaluate

ElasticNet Regression elastic_net_rmse = np.sqrt(mean_squared_error(y_test, elastic_net_pred)) elastic_net_r2 = r2_score(y_test, elastic_net_pred)

print(f"ElasticNet RMSE: {elastic_net_rmse}")

print(f"ElasticNet R^2: {elastic_net_r2}")

7. Compare Results

 # Collect metrics

metrics = pd.DataFrame({

"Model": ["Ridge", "Lasso", "ElasticNet"],

"RMSE": [ridge_rmse, lasso_rmse, elastic_net_rmse],

"R^2": [ridge_r2, lasso_r2, elastic_net_r2]

})

print(metrics)

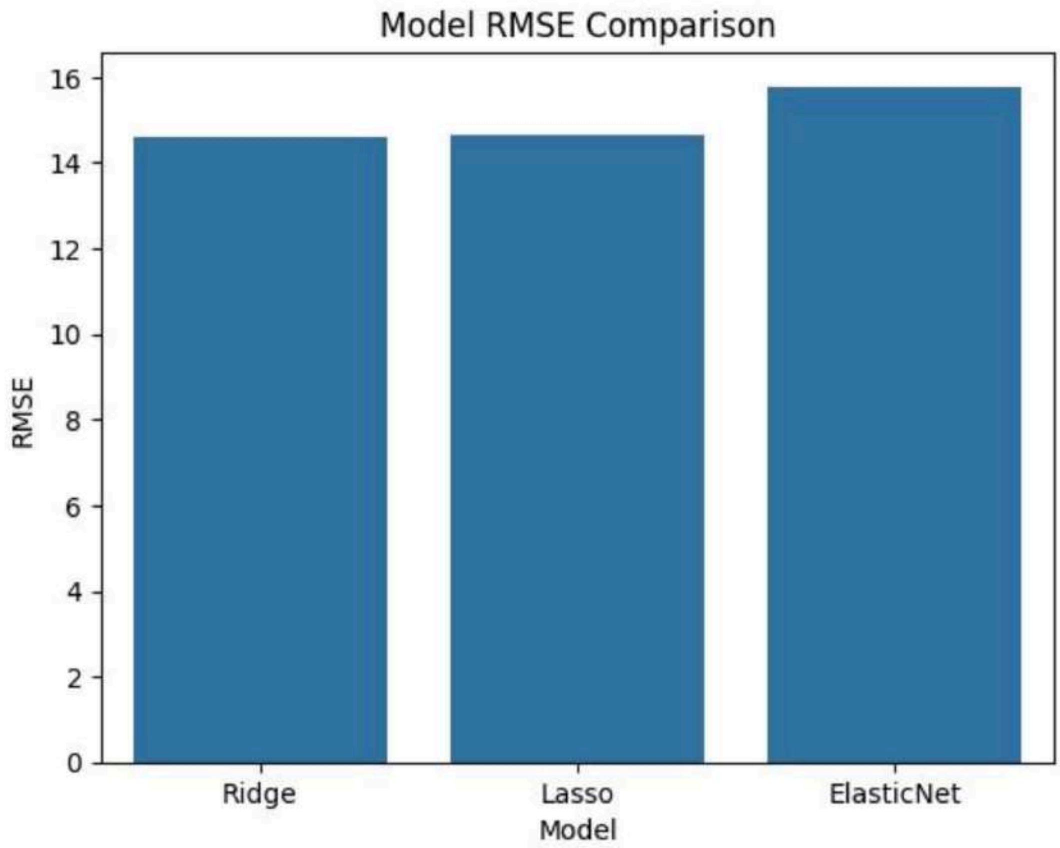
 # Plot RMSE comparison

sns.barplot(data=metrics, x="Model", y="RMSE")

plt.title("Model RMSE Comparison")

plt.show()

Output :



Practical 4 : Discriminative Models

4a. Logistic Regression :

Perform binary classification using logistic regression. Calculate accuracy, precision, recall, and understand the ROC curve."

Code :

Step 1: Import Required Libraries

```
# Import necessary libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_curve,
auc
import matplotlib.pyplot as plt
```

Step 2: Prepare the Dataset

```
from sklearn.datasets import make_classification
# Create a synthetic dataset
X, y = make_classification(n_samples=1000, n_features=10, n_classes=2, random_state=42)
# Split data 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)
```

Step 3: Train the Logistic Regression Model

```
# Initialize the logistic regression model
logreg = LogisticRegression()
# Train the model on the training data
logreg.fit(X_train, y_train)
```

Step 4: Make Predictions

 # Predict labels for the test set

y_pred = logreg.predict(X_test)

 # Predict probabilities for the ROC curve
y_prob = logreg.predict_proba(X_test)[:, 1]

Step 5: Evaluate the Model

 # Calculate metrics

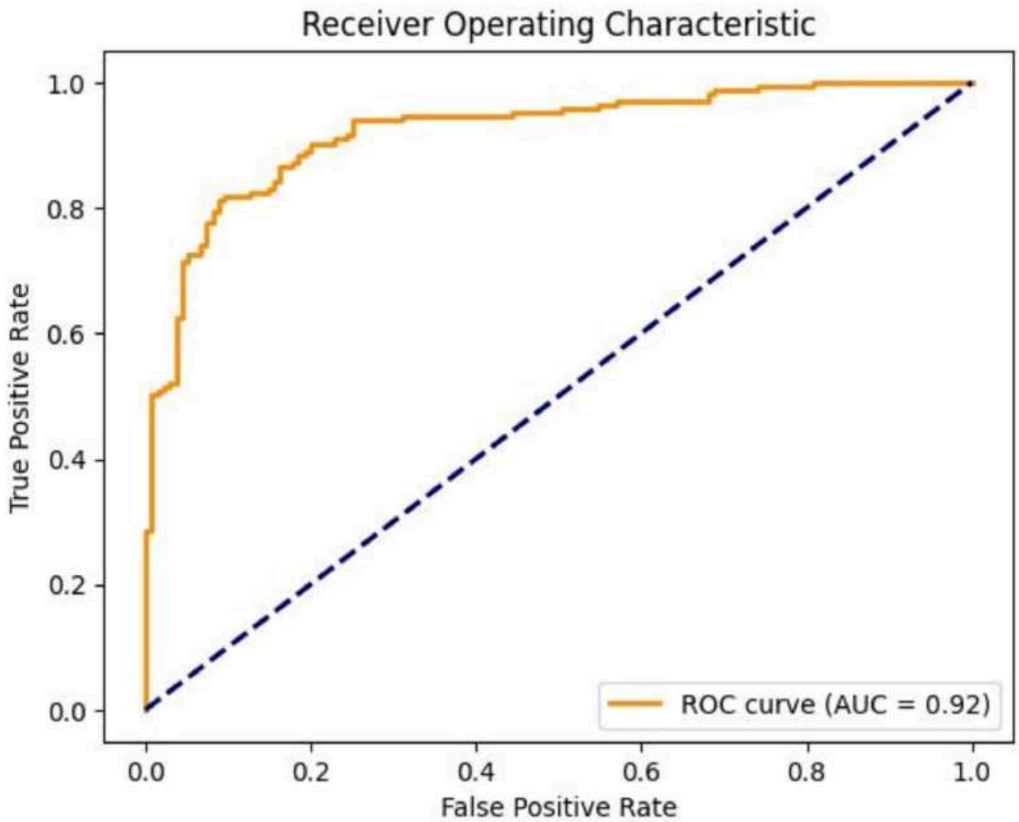
accuracy = accuracy_score(y_test, y_pred)

precision = precision_score(y_test, y_pred)

recall = recall_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.2f}")

Output :



4b .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 :

Step 1: Import Required Libraries # Import necessary libraries

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

from google.colab import files
```

Step 2: Create or Upload the CSV File

 # Check if the user wants to create a dataset or upload one

```
print("Do you have a CSV file to upload? (yes/no)")
response = input().lower()

if response == "yes":

   uploaded = files.upload()
filename = list(uploaded.keys())[0]
else:
```

 # Create a synthetic dataset

```
from sklearn.datasets import make_classification
```

 # Generate synthetic data

```
X,y=make_classification(n_samples=200,n_features=5, n_classes=2, random_state=42)
```

 # Combine features and target into a single DataFrame

```
data = pd.DataFrame(X, columns=[f"Feature_{i}"

for i in range(X.shape[1])])

   data['Target'] = y
```

 # Save the dataset to a CSV file

```
filename = "synthetic_data.csv"

data.to_csv(filename, index=False)

print(f"Synthetic dataset saved as {filename}.")
```

Step 3: Load the CSV File into a DataFrame

```
# Load the dataset into a DataFrame
data = pd.read_csv(filename)
```

 # Display the first few rows of the dataset

```
print("Loaded Dataset:")
print(data.head())
```

Step 4: Preprocess the Data

 # Separate features (X) and labels (y)

```
X = data.iloc[:, :-1].values  # All columns except the last one
y = data.iloc[:, -1].values   # Last column as the target
```

 # Split the dataset into training and testing sets (80% train, 20% test)

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Step 5: Train the k-NN Model #

Initialize the k-NN model with k=3 knn

= KNeighborsClassifier(n_neighbors=3) #

Train the model on the training data

```
knn.fit(X_train, y_train)
```

Step 6: Predict Test Samples #

Predict the labels for the test set

```
y_pred = knn.predict(X_test)
```

Step 7: Evaluate and Print Predictions #
Calculate and display the accuracy

```
accuracy = accuracy_score(y_test, y_pred)
print(f"\nModel Accuracy:
{accuracy:.2f}\n")
```

 # Display correct and incorrect predictions

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

for i in range(len(y_test)):
```

```
if y_pred[i] != y_test[i]:
print(f"Sample {i}: Predicted={y_pred[i]}, Actual={y_test[i]}")
```

Output :

```
Model Accuracy: 0.88

Correct Predictions:
Sample 0: Predicted=0, Actual=0
Sample 1: Predicted=1, Actual=1
Sample 2: Predicted=1, Actual=1
Sample 3: Predicted=0, Actual=0
Sample 4: Predicted=1, Actual=1
Sample 5: Predicted=1, Actual=1
Sample 6: Predicted=0, Actual=0
Sample 7: Predicted=0, Actual=0
Sample 9: Predicted=1, Actual=1
Sample 10: Predicted=1, Actual=1
Sample 11: Predicted=1, Actual=1
Sample 12: Predicted=0, Actual=0
Sample 13: Predicted=0, Actual=0
Sample 14: Predicted=0, Actual=0
Sample 15: Predicted=0, Actual=0
Sample 16: Predicted=0, Actual=0
Sample 17: Predicted=1, Actual=1
Sample 18: Predicted=1, Actual=1
Sample 19: Predicted=0, Actual=0
Sample 20: Predicted=0, Actual=0
Sample 22: Predicted=1, Actual=1
Sample 23: Predicted=1, Actual=1
Sample 24: Predicted=1, Actual=1
Sample 25: Predicted=1, Actual=1
Sample 26: Predicted=1, Actual=1
Sample 27: Predicted=0, Actual=0
Sample 28: Predicted=0, Actual=0
Sample 30: Predicted=1, Actual=1
Sample 31: Predicted=1, Actual=1
Sample 32: Predicted=1, Actual=1
Sample 34: Predicted=0, Actual=0
Sample 35: Predicted=1, Actual=1
Sample 36: Predicted=1, Actual=1
Sample 38: Predicted=1, Actual=1
Sample 39: Predicted=1, Actual=1
```

```
Incorrect Predictions:
Sample 8: Predicted=1, Actual=0
Sample 21: Predicted=1, Actual=0
Sample 29: Predicted=0, Actual=1
Sample 33: Predicted=1, Actual=0
Sample 37: Predicted=1, Actual=0
```

4c. Build a decision tree classifier or regressor. Control hyperparameters like tree depth to avoid overfitting. Visualize the tree.

Code :

Step 1: Import Required Libraries

 # Import necessary libraries

import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, plot_tree

from sklearn.metrics import accuracy_score, mean_squared_error

import matplotlib.pyplot as plt

from google.colab import files

Step 2: Create or Upload the CSV File

 # Check if the user wants to upload a file or generate one
print("Do you have a CSV file to upload? (yes/no)")

response = input().lower()

if response == "yes":

 # Upload the CSV file

uploaded = files.upload()

filename = list(uploaded.keys())[0]

else:

 # Generate synthetic data (classification or regression)

from sklearn.datasets import make_classification, make_regression
print("Choose a task: (1) Classification (2) Regression")

task = int(input())

if task == 1:

 # Generate synthetic classification data

X, y = make_classification(n_samples=200, n_features=5, random_state=42)

task_type = "classification"

else:

 # Generate synthetic regression data

X, y = make_regression(n_samples=200, n_features=5, random_state=42)

task_type = "regression"

 # Combine features and target into a single DataFrame

data = pd.DataFrame(X, columns=[f"Feature_{i}"

for i in range(X.shape[1])]) data['Target'] = y

 # Save the dataset to a CSV file

filename = "synthetic_data.csv"

data.to_csv(filename, index=False)

print(f"Synthetic {task_type} dataset saved as {filename}.")

Step 3: Load the Dataset

 # Load the dataset

data =pd.read_csv(filename)

 # Display the first few rows of the dataset

print("Dataset Preview:")

print(data.head())

Step 4: Preprocess the Data

 # Separate features and target

X = data.iloc[:, :-1].values # All columns except the last one
y = data.iloc[:, -1].values   # Last column as the target

 # Split 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 5: Build the Decision Tree

 # Define the tree depth to avoid overfitting

max_depth = 3

 # Initialize the model

```
if task_type =="classification":
   model = DecisionTreeClassifier(max_depth=max_depth, random_state=42)
else:
   model = DecisionTreeRegressor(max_depth=max_depth, random_state=42)
```

 # Train the model

```
model.fit(X_train, y_train)
```

Step 6: Make Predictions

 # Predict on the test set

```
y_pred = model.predict(X_test)
```

 # Evaluate the model

```
if task_type == "classification":
   accuracy = accuracy_score(y_test, y_pred)
   print(f"Accuracy: {accuracy:.2f}")
else:
   mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse:.2f}")
```

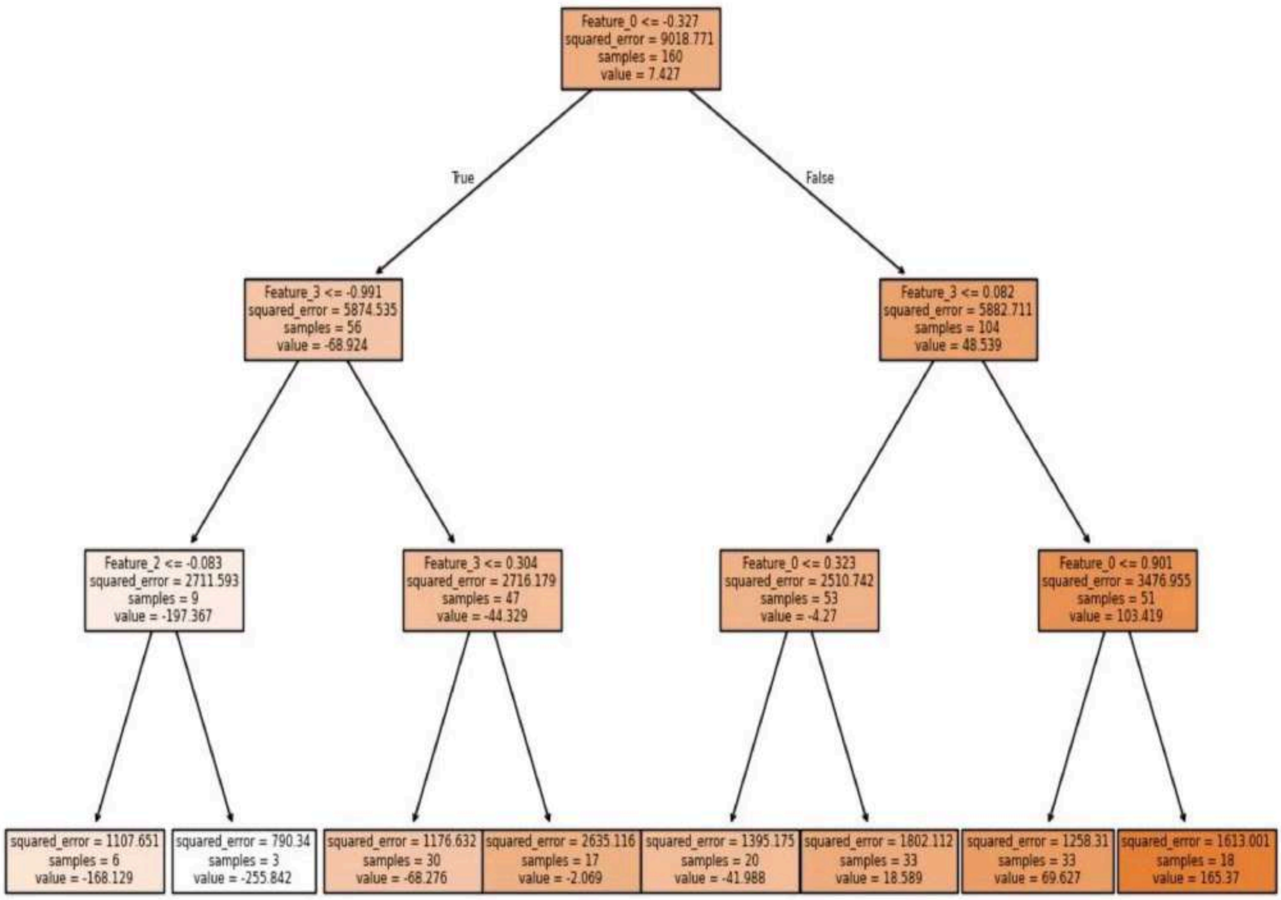
Step 7: Visualize the Tree

 # Visualize the decision tree

```
plt.figure(figsize=(12, 8))
plot_tree(model, feature_names=data.columns[:-1], class_names=str(np.unique(y))
if task_type == "classification" else None, filled=True)
plt.title("Decision Tree Visualization")
plt.show()
```

Output :

Decision Tree Visualization



4d. Implement a Support Vector Machine for any relevant dataset.

Code:

Step 1: Import Required Libraries

 # Import necessary libraries

import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.svm import SVC

from sklearn.metrics import accuracy_score, classification_report
from google.colab import files

Step 2: Create or Upload a Dataset

 # Check if the user wants to upload a file or generate one
print("Do you have a CSV file to upload? (yes/no)")

response = input().lower()

if response == "yes":

 # Upload the CSV file

uploaded = files.upload()

filename = list(uploaded.keys())[0]

else:

 # Generate synthetic classification data

from sklearn.datasets import make_classification

X, y = make_classification(n_samples=200, n_features=5, n_classes=2, random_state=42)

 # Combine features and target into a DataFrame

data = pd.DataFrame(X, columns=[f"Feature_{i}"

for i in range(X.shape[1])])

data['Target'] = y

#Save the synthetic dataset to a CSV file

```
filename="synthetic_data.csv"

data.to_csv(filename,index=False)

print(f"Synthetic dataset saved as {filename}.")
```

Step 3: Load the Dataset

 # Load the dataset into a DataFrame

```
data = pd.read_csv(filename)
```

 # Display the first few rows of the dataset

```
print("Dataset Preview:")
print(data.head())
```

Step 4: Preprocess the Data

 # Separate features (X) and target (y)

```
X = data.iloc[:, :-1].values  # All columns except the last one
y = data.iloc[:, -1].values   # Last column as the target
```

 # Split the dataset into training (80%) and testing (20%) sets

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Step 5: Train the Support Vector Machine

 # Initialize the SVM model (use RBF kernel as default)

```
svm_model = SVC(kernel='rbf', C=1.0, gamma='scale', random_state=42)
```

 # Train the SVM model on the training data

```
svm_model.fit(X_train, y_train)
```

Step 6: Make Predictions

 # Predict the labels for the test set

```
y_pred = svm_model.predict(X_test)
```

Step 7: Evaluate the Model

 # Calculate and print the accuracy

```
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.2f}")
```

 # Print a detailed classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

Step 8: Visualize the Decision Boundary (Optional for 2D Data)

import matplotlib.pyplot as plt

 # Generate 2D synthetic data

from sklearn.datasets import make_blobs

X, y = make_blobs(n_samples=100, centers=2, random_state=42, cluster_std=1.5)

 # Fit the SVM on this data

svm_model.fit(X, y)

#Plot the decision boundary
plt.figure(figsize=(8, 6))

plt.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm', edgecolor='k')
 # Create a grid to evaluate the model

xx, yy = np.meshgrid (np.linspace(X[:, 0].min(), X[:, 0].max(), 100), np.linspace(X[:, 1].min(), X[:, 1].max(), 100))
Z = svm_model.decision_function(np.c_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

 # Plot the decision boundary and margins

plt.contour(xx, yy, Z, levels=[-1, 0, 1], linestyles=['--', '-', '--'], colors='k')

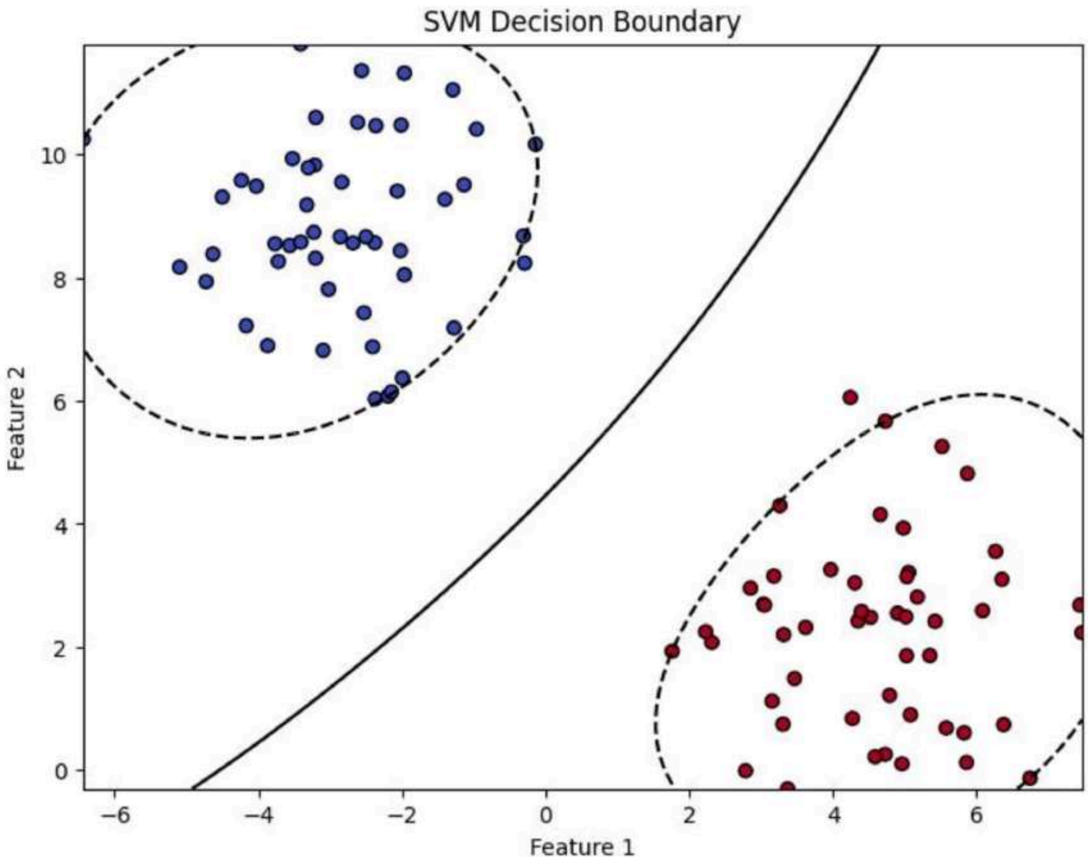
plt.title("SVM Decision Boundary")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.show()

Output :


SVM Decision Boundary
Feature 1
Feature 2
−6
−4
−2
0
2
4
6
0
2
4
6
8
10

4e. Train a random forest ensemble. Experiment with the number of trees and feature sampling. Compare performance to a single decision tree.

Code :

Step 1: Import Required Libraries # Import necessarylibraries

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, classification_report
from google.colab import files
```

Step 2: Create or Upload a Dataset

```
# Check if the user wants to upload a file or generate one
print("Do you have a CSV file to upload? (yes/no)")
response = input().lower()
if response == "yes":
# Upload the CSV file
uploaded = files.upload()
filename = list(uploaded.keys())[0]
else:
# Generate synthetic classification data

from sklearn.datasets import make_classification

 X, y = make_classification(n_samples=300, n_features=10, n_classes=2, random_state=42)

 # Combine features and target into a DataFrame

data = pd.DataFrame(X, columns=[f"Feature_{i}"

for i in range(X.shape[1])])

   data['Target'] = y

# Save the synthetic dataset to a CSV file
filename = "synthetic_data.csv"
data.to_csv(filename, index=False)
print(f"Synthetic dataset saved as {filename}.")
```

Step 3: Load the Dataset

```
# Load the dataset
```

data =pd.read_csv(filename)

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

Step 4: Preprocess the Data

 # Separate features (X) and target (y)

X = data.iloc[:, :-1].values # All columns except the last one
y = data.iloc[:, -1].values   # Last column as the target

 # Split the dataset into training (80%) and testing (20%) sets

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

Step 5: Train a Single Decision Tree Classifier
 # Initialize and train the Decision Tree model
decision_tree =DecisionTreeClassifier(random_state=42)
decision_tree.fit(X_train, y_train)
 # Predict and evaluate
y_pred_tree = decision_tree.predict(X_test)
accuracy_tree = accuracy_score(y_test, y_pred_tree)
print(f"Decision Tree Accuracy: {accuracy_tree:.2f}")

Step 6: Train a Random Forest Classifier

 # Initialize the Random Forest model with hyperparameter tuning

random_forest = RandomForestClassifier(n_estimators=100, max_features='sqrt', random_state=42)
 # Train the model
random_forest.fit(X_train, y_train)

 # Predict and evaluate

y_pred_rf = random_forest.predict(X_test)

accuracy_rf = accuracy_score(y_test, y_pred_rf)

print(f"Random Forest Accuracy (100 trees, sqrt features): {accuracy_rf:.2f}")

Step 7: Experiment with Random Forest Hyperparameters
 # Experiment with fewer trees and different feature sampling
rf_experiment = RandomForestClassifier(n_estimators=50, max_features=3, random_state=42)

rf_experiment.fit(X_train, y_train)

Predict and evaluate

y_pred_rf_exp = rf_experiment.predict(X_test)

accuracy_rf_exp = accuracy_score(y_test, y_pred_rf_exp)

print(f"Random Forest Accuracy (50 trees, max_features=3): {accuracy_rf_exp:.2f}")

Step 8: Compare the Models

print("\nModel Comparison:")

print(f"Decision Tree Accuracy: {accuracy_tree:.2f}")

print(f"Random Forest Accuracy (100 trees): {accuracy_rf:.2f}")

print(f"Random Forest Accuracy (50 trees, max_features=3): {accuracy_rf_exp:.2f}")

Step 9: Visualize Feature Importance (Optional)

import matplotlib.pyplot as plt

Extract feature importance from the Random Forest model

feature_importances = random_forest.feature_importances_

Plot the feature importance

plt.figure(figsize=(10, 6))

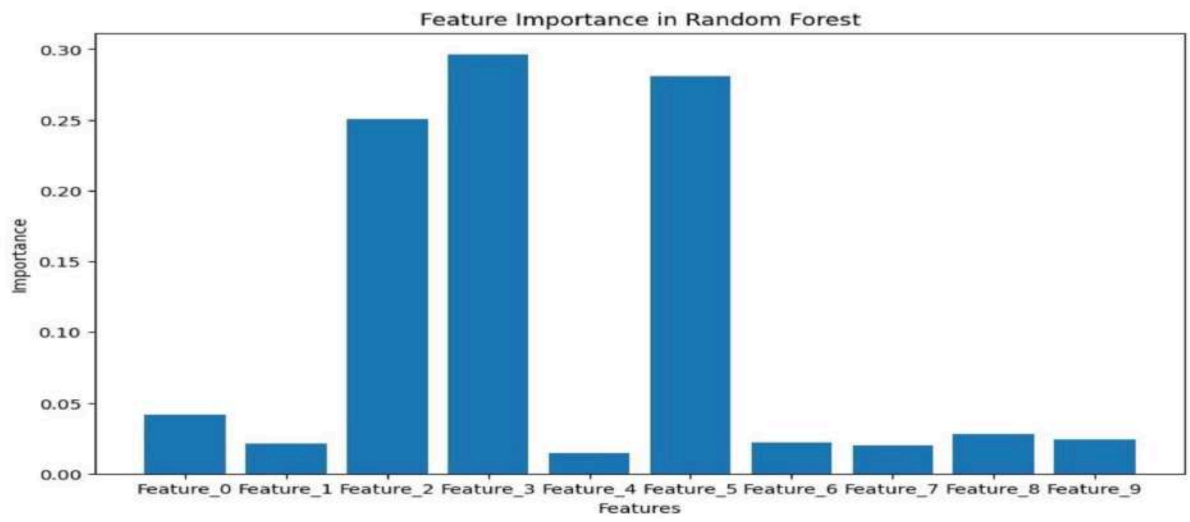
plt.bar(range(len(feature_importances)), feature_importances, tick_label=data.columns[:-1])
plt.title("Feature Importance in Random Forest")

plt.xlabel("Features")

plt.ylabel("Importance")

plt.show()

Output :



4f. Implement a gradient boosting machine (e.g., XGBoost). Tune hyperparameters and explore feature importance.

Code :

Step 1: Import Required Libraries

 # Import necessary libraries

import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.metrics import accuracy_score, classification_report
from xgboost import XGBClassifier, plot_importance

import matplotlib.pyplot as plt

from google.colab import files

Step 2: Create or Upload a Dataset

 # Check if the user wants to upload a file or generate

one print("Do you have a CSV file to upload? (yes/no)")

response = input().lower()

if response == "yes":

Upload the CSV file

uploaded=files.upload()

filename = list(uploaded.keys())[0]

else:

 # Generate synthetic classification data

from sklearn.datasets import make_classification

X, y = make_classification(n_samples=300, n_features=10, n_classes=2, random_state=42)

 # Combine features and target into a DataFrame

data = pd.DataFrame(X, columns=[f"Feature_{i}"

for i in range(X.shape[1])])data['Target'] = y

 # Save the synthetic dataset to a CSV file

filename="synthetic_data.csv"

data.to_csv(filename,index=False)

print(f"Synthetic dataset saved as {filename}.")

Step 3: Load the Dataset

 # Load the dataset

data = pd.read_csv(filename)
 # Display the first few rows of the dataset print("Dataset Preview:")
print(data.head())

Step 4: Preprocess the Data

 # Separate features (X) and target (y)

X = data.iloc[:, :-1].values # All columns except the last one
y = data.iloc[:, -1].values   # Last column as the target

 # Split the dataset into training (80%) and testing (20%) sets

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

Step 5: Train a Basic XGBoost Model

 # Initialize and train the XGBoost model with default parameters

xgb = XGBClassifier(random_state=42)

xgb.fit(X_train, y_train)

 # Predict and evaluate the model

y_pred = xgb.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print(f"XGBoost Accuracy (Default Parameters): {accuracy:.2f}")
Step 6: Tune Hyperparameters with GridSearchCV

 # Define a grid of hyperparameters
param_grid = { 'n_estimators': [50, 100, 150],'learning_rate': [0.01, 0.1, 0.2], 'max_depth': [3, 5, 7] }
 # Initialize GridSearchCV

grid_search =
GridSearchCV(estimator=XGBClassifier(random_state=42),param_grid=param_grid, scoring='accuracy', cv=3,verbose=1)
 # Fit the model with grid search
grid_search.fit(X_train, y_train)

 # Best parameters from GridSearch

print(f"Best Parameters: {grid_search.best_params_}")
 # Train the final model with the best parameters
best_xgb = grid_search.best_estimator_
 # Predict and evaluate

y_pred_best = best_xgb.predict(X_test)

accuracy_best = accuracy_score(y_test, y_pred_best)
print(f"XGBoost Accuracy (Tuned Parameters): {accuracy_best:.2f}")

Step 7: Explore Feature Importance

 # Plot feature importance for the tuned model

plt.figure(figsize=(10, 6))

plot_importance(best_xgb,importance_type='weight', xlabel="Importance",

ylabel="Features")

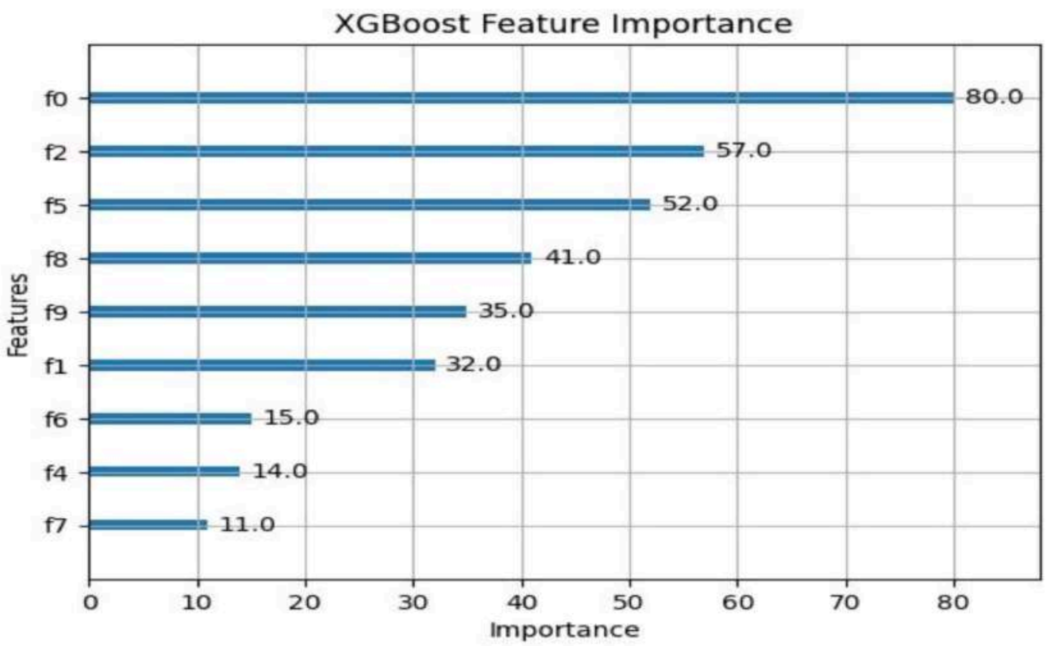
plt.title("XGBoost Feature Importance")

plt.show()

Step 8: Evaluate the Model

 # Print a detailed classification report
print("Classification Report:")
print(classification_report(y_test, y_pred_best))

Output :



Practical 5

5a. Implement and demonstrate the working of a Naive Bayesian classifier using a sample data set. Build the model to classify a test sample.

Step 1: Import Required Libraries

```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,classification_report
from sklearn.naive_bayes import GaussianNB
from google.colab import files
```

Step 2: Create or Upload a Dataset

```
# Ask if the user wants to upload a file or generate one
print("Do you have a CSV file to upload? (yes/no)")
response = input().lower()
if response == "yes":
# Upload the CSV file
uploaded =files.upload()
 filename = list(uploaded.keys())[0]
else:
   # Generate synthetic classification data
from sklearn.datasets import make_classification
   X, y = make_classification(n_samples=300, n_features=8, n_classes=2, random_state=42)
   # Combine features and target into a DataFrame
data = pd.DataFrame(X, columns=[f"Feature_{i}" for i in range(X.shape[1])])
 data['Target'] = y
   # Save the synthetic dataset to a CSV file
filename="synthetic_naive_bayes_data.csv"
data.to_csv(filename,index=False)
print(f"Synthetic dataset saved as {filename}.")
```

Step 3: Load the Dataset

 # Load the dataset

data =pd.read_csv(filename)

 # Display the first few rows of the dataset
print("Dataset Preview:")

print(data.head())

Step 4: Preprocess the Data

 # Separate features (X) and target (y)

X = data.iloc[:, :-1].values # All columns except the last one
y = data.iloc[:, -1].values # Last column as the target

 # Split the dataset into training (80%) and testing (20%) sets

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

Step 5: Train a Naive Bayes Classifier

 # Initialize the Gaussian Naive Bayes classifier
naive_bayes = GaussianNB()

 # Train the model

naive_bayes.fit(X_train, y_train)

Step 6: Make Predictions and Evaluate

 # Predict on the test set

y_pred =naive_bayes.predict(X_test)
 # Evaluate the model

accuracy = accuracy_score(y_test, y_pred)
print(f"Naive Bayes Accuracy:
{accuracy:.2f}")

 # Detailed classification report

print("Classification Report:")

print(classification_report(y_test, y_pred))

Step 7: Test the Model with a Custom Sample

 # Define a sample test input (replace with meaningful values based on your dataset)

test_sample = [X_test[0]]

 # Taking the first test sample for demonstration

 # Predict the class for the test sample

predicted_class = naive_bayes.predict(test_sample)

print(f"Test Sample: {test_sample}")

print(f"Predicted Class: {predicted_class[0]}")

Output :

```
Dataset Preview:
   Feature_0  Feature_1  Feature_2  Feature_3  Feature_4  Feature_5  \
0  -1.274158   1.317988  -2.423879   0.906946  -1.583903  -0.331811
1   1.607963  -1.649959   0.299293  -0.891720   1.301741   1.508502
2  -0.154167   0.161033   2.210523   0.139400  -0.557492   0.087713
3  -0.920991   0.949136  -1.613561   0.588410   1.471170  -0.529287
4   1.013304  -1.038578  -0.305225  -0.539334  -0.609512   1.048078

   Feature_6  Feature_7  Target
0  -0.452306   0.760415       1
1   0.742095   1.561511       0
2   0.963879  -1.369803       0
3  -1.371901  -0.209324       0
4  -1.065114  -0.186971       0
```

```
Test Sample: [array([-0.90320608,  0.9220511 , -1.32308979,  0.41081065,  1.64201516,
       -1.23559176, -0.63896175,  1.00981709])]
Predicted Class: 1
```

5b. Implement Hidden Markov Models using hmmlearn

Code :

Step 1: Install Required Libraries

 # Install hmmlearn

!pip install hmmlearn

Step 2: Import Required Libraries
 # Import necessary libraries
import numpy as np

import pandas as pd

from hmmlearn import hmm

import matplotlib.pyplot as plt

Step 3: Create or Load a Dataset

 # Generate synthetic observable data
np.random.seed(42)
 # Create a sequence of observations and hidden states

observations = np.random.choice(['A', 'B', 'C'], size=100, p=[0.5, 0.3, 0.2])

hidden_states = np.random.choice(['X', 'Y'], size=100, p=[0.6, 0.4])

 # Save the data in a DataFrame for analysis

data = pd.DataFrame({'Observations': observations, 'Hidden States': hidden_states})
print("Generated Data:")

print(data.head())

Step 4: Encode Observations

 # Encode the observations into integers

observation_mapping = {obs: idx for idx, obs in enumerate(np.unique(observations))}
encoded_observations = np.array([observation_mapping[obs] for obs in observations])
 # Print the mapping

print("Observation Encoding:")

print(observation_mapping)

Step 5: Initialize and Configure the HMM

 # Initialize the HMM model

n_states = 2 **# Number of hidden states**

n_observations = len(observation_mapping)
 # Number of unique observations
model = hmm.MultinomialHMM(n_components=n_states, random_state=42, n_iter=100, tol=0.01)

 # Define start probabilities (initial distribution of states)

start_probs = np.array([0.6, 0.4]) # Assumed probabilities

model.startprob_ = start_probs

 # Define transition probabilities between states

trans_probs = np.array([

[0.7, 0.3], # From state X

[0.4, 0.6], # From state Y])

model.transmat_ = trans_probs

 # Define emission probabilities (probability of observations given states)

emission_probs = np.array([

[0.5, 0.4, 0.1], # State X emits A, B, C

[0.2, 0.3, 0.5], # State Y emits A, B, C

])

model.emissionprob_ = emission_probs

 # Print the configured model parameters
print("Start Probabilities:", model.startprob_)
print("Transition Matrix:", model.transmat_)
print("Emission Probabilities:",
model.emissionprob_)

Step 6: Train the Model

 # Reshape the data for HMM (requires 2D array)
encoded_observations = encoded_observations.reshape(-1, 1)

 # Fit the model
model.fit(encoded_observations)

 # Predict hidden states for the observations
predicted_states = model.predict(encoded_observations)

 # Print the predicted states
print("Predicted States:")
print(predicted_states)

Step 7: Visualize the Results

 # Map predicted states back to their original labels

state_mapping = {0: 'X', 1: 'Y'}

predicted_state_labels = [state_mapping[state] for state in predicted_states]

 # Add predicted states to the DataFrame
data['Predicted States'] = predicted_state_labels

 # Display the first few rows with predicted states
print("Data with Predicted States:")
print(data.head())
 # Plot the observations and predicted states

plt.figure(figsize=(12, 6))

plt.plot(data['Observations'], label='Observations', marker='o', linestyle='-', alpha=0.7)
plt.plot(data['Predicted States'], label='Predicted States', marker='x', linestyle='--', alpha=0.7)
plt.legend()

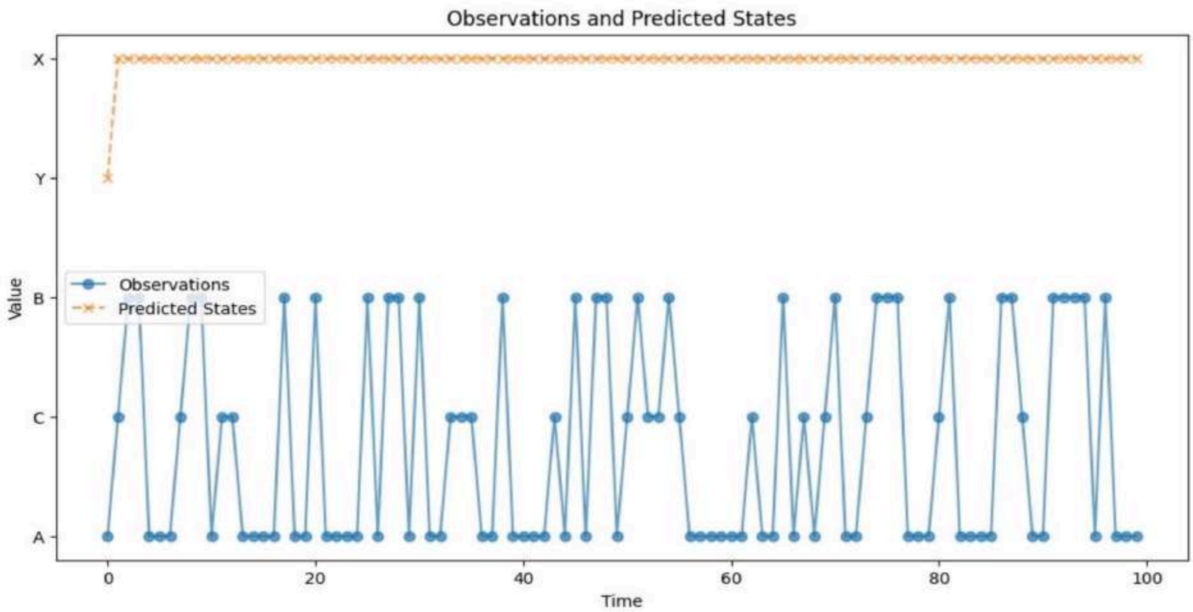
plt.title("Observations and Predicted States")

plt.xlabel("Time")

plt.ylabel("Value")

plt.show()

Output :



Practical 6 : Probabilistic Model

6a. Implement Bayesian Linear Regression to explore prior and posterior distribution.

Bayesian Linear Regression is a probabilistic approach to linear regression that incorporates uncertainty in the model parameters. Instead of estimating point values for parameters (as in traditional linear regression), we estimate distributions over the parameters.

Code :

Step 1: Install Required Libraries

 # Install necessary libraries

```
!pip install matplotlib seaborn scikit-learn
```

Step 2: Import Required Libraries

 # Import necessary libraries

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.linear_model import BayesianRidge
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from google.colab import files
```

Step 3: Create or Upload a Dataset

 # Upload a CSV file if you have one

```
print("Do you have a CSV file to upload? (yes/no)")
response = input().lower()

if response == "yes":

# Upload the CSV file

    uploaded = files.upload()
```

filename = list(uploaded.keys())[0]
else:

 # Generate synthetic data for demonstration
np.random.seed(42)

X = np.random.rand(100, 1) * 10

 # Random data between 0 and 10

y = 2 * X + 1 + np.random.randn(100, 1) * 2

 # y = 2x + 1 with some noise

 # Convert to a DataFrame

data = pd.DataFrame(np.hstack((X, y)), columns=["X", "y"])
 # Save to CSV for convenience

filename="synthetic_data.csv"

data.to_csv(filename,index=False)

print(f"Synthetic dataset saved as {filename}.")

Step 4: Load and Explore the Data
 # Load the dataset (for CSV file)
data = pd.read_csv(filename)
 # Display first few rows
print("Dataset Preview:")
print(data.head())

Step 5: Preprocess the Data

 # Separate features (X) and target (y)

X = data["X"].values.reshape(-1, 1) # Feature matrix
y = data["y"].values # Target vector

 # Split the dataset into training (80%) and testing (20%) sets

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

Step 6: Implement Bayesian Linear Regression Model

 # Initialize the BayesianRidge model (which implements Bayesian Linear Regression)

bayesian_regressor = BayesianRidge()

 # Fit the model on the training data

bayesian_regressor.fit(X_train, y_train)

 # Predict on the test data

y_pred = bayesian_regressor.predict(X_test)

Step 7: Visualize the Prior and Posterior Distributions

 # Plot the prior and posterior distributions of the parameters

fig, ax = plt.subplots(1, 2, figsize=(12, 6))

 # Plot prior distribution (assuming the model starts with a standard prior)

ax[0].set_title("Prior Distribution (Assumed)") ax[0].hist(np.random.normal(0, 1, 1000),

bins=50, alpha=0.7, color='blue', label="Prior") ax[0].legend()

 # Plot posterior distribution (after model fitting)

ax[1].set_title("Posterior Distribution (After Fitting)")
ax[1].hist(bayesian_regressor.coef_, bins=50, alpha=0.7, color='green',
label="Posterior") ax[1].legend()

plt.show()

Step 8: Evaluate the Model Performance #
Calculate the Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error (MSE):
{mse:.2f}")

Step 9: Visualize the Fit of the Model

 # Plot the true values and the predicted values

plt.figure(figsize=(8, 6))

plt.scatter(X_test, y_test, color="blue", label="True values")

plt.plot(X_test, y_pred, color="red", label="Predicted values",

linewidth=2)

plt.title("Bayesian Linear Regression: True vs Predicted Values")

plt.xlabel("X")

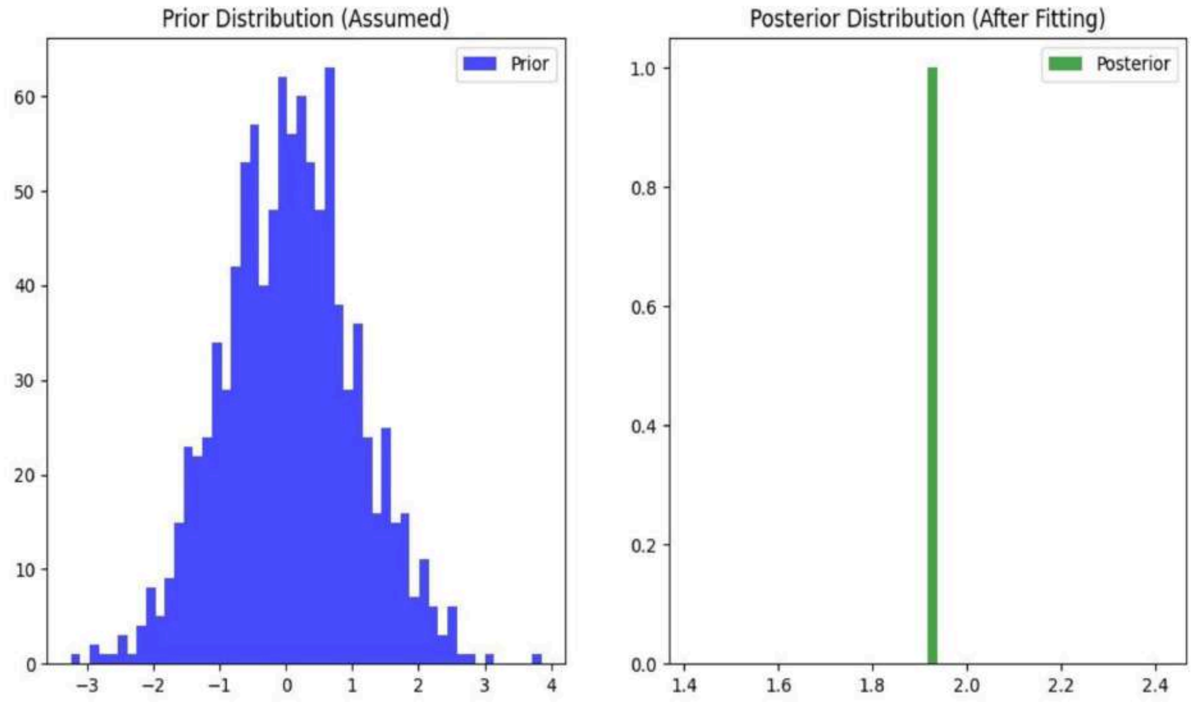
plt.ylabel("y")

plt.legend()

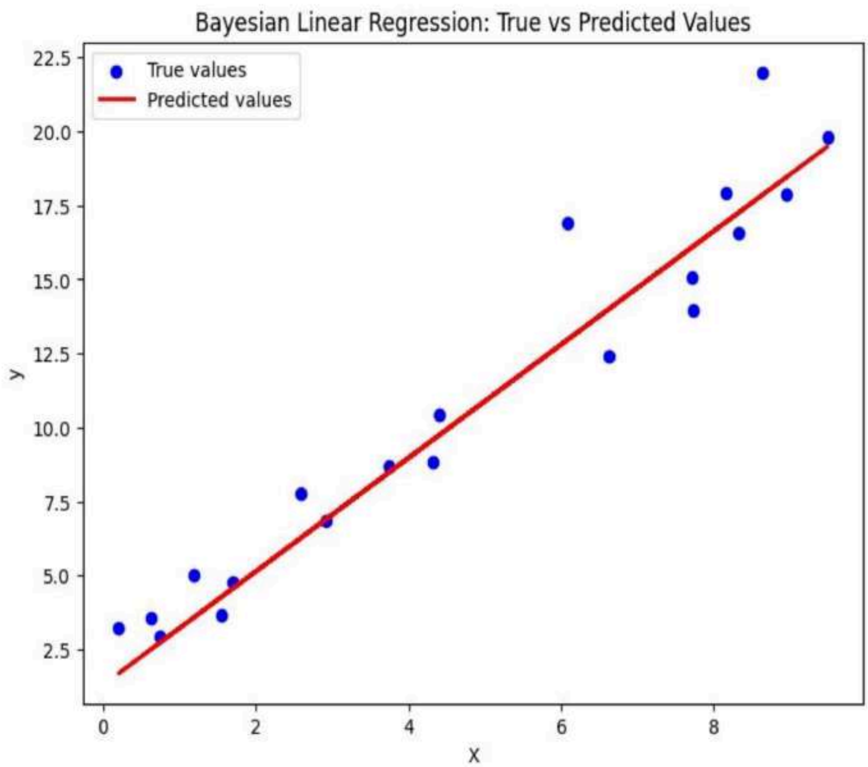
plt.show()

Mean Squared Error (MSE): 3.9

Output :



Prior Distribution (Assumed)
Prior
60
50
40
30
20
10
0
−3
−2
−1
0
1
2
3
4
Posterior Distribution (After Fitting)
Posterior
1.0
0.8
0.6
0.4
0.2
0.0
1.4
1.6
1.8
2.0
2.2
2.4



Bayesian Linear Regression: True vs Predicted Values
True values
Predicted values
22.5
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
y
0
2
4
6
8
X

6b. Implement Gaussian Mixture Models for density estimation and unsupervised clustering.

Code :

Step 1: Install Required Libraries

 # Install required libraries

!pip install matplotlib seaborn scikit-learn

Step 2: Import Required Libraries

 # Import necessary libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.mixture import GaussianMixture
from sklearn.model_selection import train_test_split
from google.colab import files

Step 3: Create or Upload a Dataset

#Ask if the user has a CSV file to upload
print("Do you have a CSV file to upload? (yes/no)")
response = input().lower()

if response == "yes":

 # Upload the CSV file

uploaded = files.upload()

filename = list(uploaded.keys())[0]

else:

 # Generate synthetic 2D data with two clusters for demonstration
np.random.seed(42)
 # Generate data for two Gaussian distributions

X1 = np.random.normal(loc=0, scale=1, size=(300, 2)) # Cluster 1: mean = 0, std = 1

X2 = np.random.normal(loc=5, scale=1, size=(300, 2)) # Cluster 2: mean = 5, std = 1 #
Stack the data to create a dataset

X = np.vstack([X1, X2])

 # Create DataFrame to simulate the CSV file for consistency
data = pd.DataFrame(X, columns=["Feature_1", "Feature_2"])
filename = "synthetic_data.csv"
data.to_csv(filename, index=False)
print(f"Synthetic dataset saved as {filename}.")

Step 4: Load and Explore the Dataset

 # Load the dataset (if CSV file is uploaded)
data = pd.read_csv(filename)
 # Display the first few rows
print("Dataset Preview:")
print(data.head())
 # Plot the data to visualize its structure
sns.scatterplot(data=data, x="Feature_1", y="Feature_2")
plt.title("Synthetic Data")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.show()

Step 5: Fit a Gaussian Mixture Model (GMM)

 # Define the GMM model

n_components = 2 **# Number of Gaussian distributions (clusters)**

gmm = GaussianMixture(n_components=n_components, covariance_type='full', random_state=42)

 # Fit the GMM model to the data

gmm.fit(data)

 # Predict the cluster labels for each data point

labels = gmm.predict(data)

 # Add the cluster labels to the dataset for visualization
data['Cluster'] = labels

 # Plot the clustered data

sns.scatterplot(data=data, x="Feature_1", y="Feature_2", hue="Cluster", palette="viridis", marker="o")

plt.title("Gaussian Mixture Model Clustering")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.legend()

plt.show()

Step 6: Visualize the Gaussian Mixture Model (GMM) Components
 # Extract the means and covariances of the Gaussian components

means = gmm.means_

covariances = gmm.covariances_

 # Plot the GMM components on top of the data

plt.figure(figsize=(8, 6))
 # Plot data points

sns.scatterplot(data=data, x="Feature_1", y="Feature_2", hue="Cluster", palette="viridis", marker="o", s=60, alpha=0.7)

 # Plot the GMM ellipses for mean, covar in zip(means, covariances):

 # Plot the Gaussian components as ellipses

v, w = np.linalg.eigh(covar)

v = 2.0 * np.sqrt(2.0) * np.sqrt(v)

 # Scaling factor for the ellipse
u = w[0] / np.linalg.norm(w[0])

 # Normalize the eigenvector

angle = np.arctan(u[1] / u[0])

 # Create the ellipse

angle = angle * 180.0 / np.pi **# Convert to degrees**
ellipse = plt.matplotlib.patches.Ellipse(mean, v[0], v[1], angle=angle, color='red', alpha=0.3)
plt.gca().add_patch(ellipse)

plt.title("GMM Clustering with Gaussian Components")
plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.legend()

plt.show()

Step 7: Model Evaluation (Optional)

 # Compute the log-likelihood of the data under the fitted GMM model
log_likelihood = gmm.score(data)
print(f"Log-Likelihood of the data: {log_likelihood:.2f}")

Step 8: Predict New Data Points

 # Example of predicting the cluster for new data points
new_data = np.array([[1.5, 2.5], [4.5, 5.5], [7.0, 8.0]])

new_labels = gmm.predict(new_data)

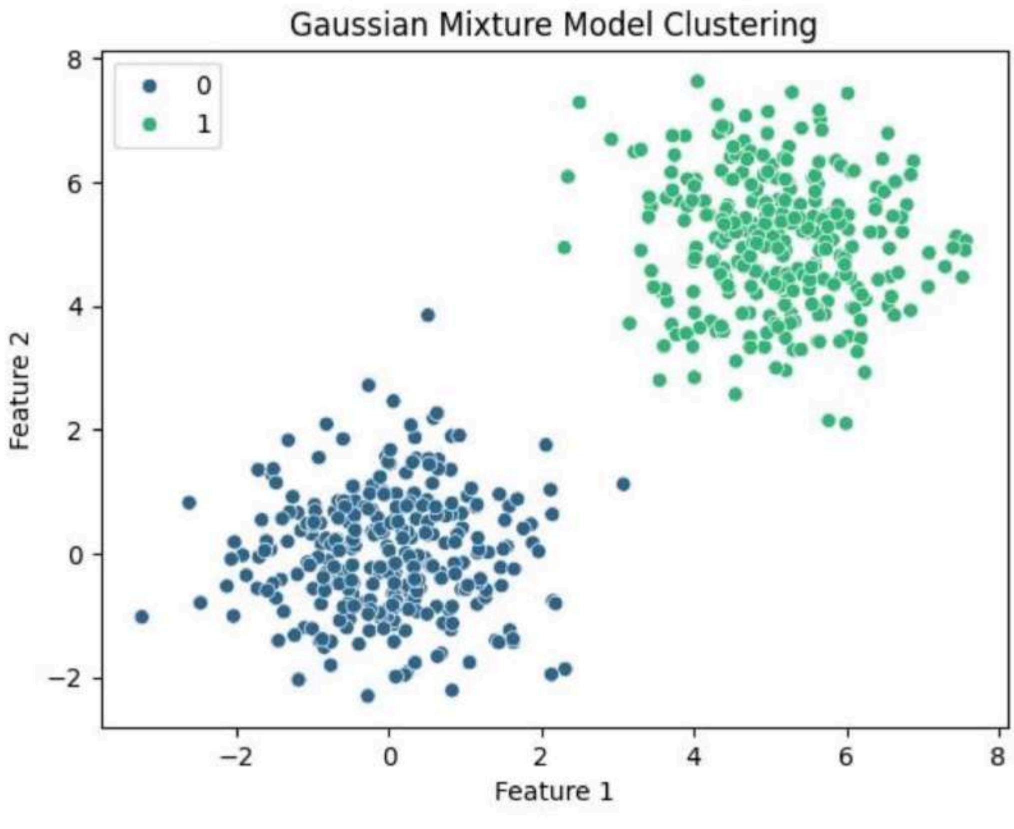
 # Print the predicted clusters for the new data points

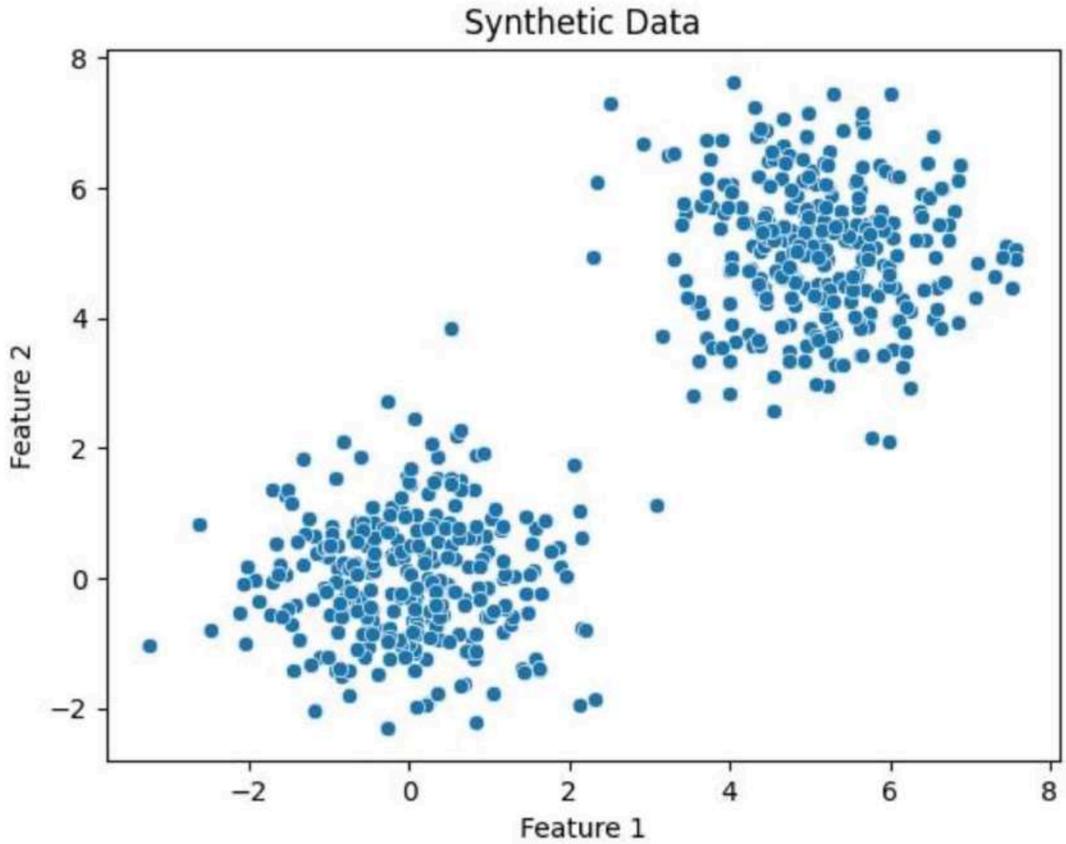
print("Predicted Clusters for New Data Points:")

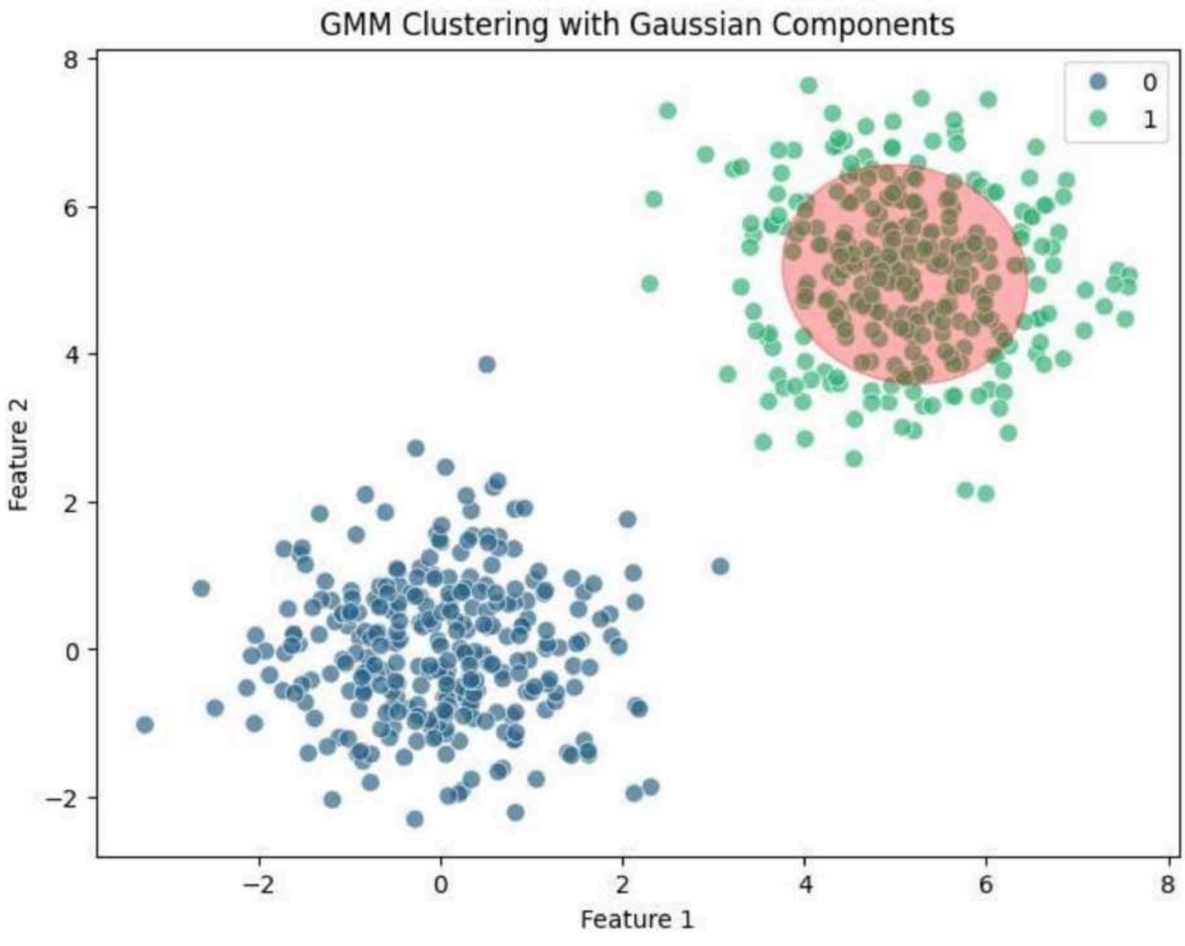
for i, label in enumerate(new_labels):

    print(f"Data point {new_data[i]} is in Cluster {label}")

Output :




Synthetic Data
Feature 2
Feature 1
8
6
4
2
0
−2
−2
0
2
4
6
8


GMM Clustering with Gaussian Components
Feature 2
Feature 1
0
1
8
6
4
2
0
−2
−2
0
2
4
6
8

Practical 7 : Model Evaluation and Hyperparameter Tuning

7a. Implement cross-validation techniques (k-fold, stratified, etc.) for robust model evaluation

Code :

1. Import Necessary Libraries

```
import numpy as np
import pandas as pd
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split, KFold, StratifiedKFold, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

2. Generate a Synthetic Dataset

```
# Create a synthetic dataset with 2 classes
X, y = make_classification(
    n_samples=1000, n_features=10, n_informative=8, n_redundant=2,
    n_clusters_per_class=1, random_state=42
)
# Convert to a DataFrame for visualization
df = pd.DataFrame(X, columns=[f'Feature_{i}' for i in range(1, 11)])
df['Target'] = y
# Display the first few rows
print(df.head())
```

3. Split Data into Train and Test Sets

```
# Split data into 80% training and 20% testing
```

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

4. Define k-Fold Cross-Validation

```
kf = KFold(n_splits=5, shuffle=True, random_state=42)
print("k-Fold Cross-Validation:")
for train_index, val_index in kf.split(X_train):
    print("TRAIN:", train_index, "VALIDATION:", val_index)
```

5. Define Stratified k-Fold Cross-Validation

```
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
print("\nStratified k-Fold Cross-Validation:")
for train_index, val_index in skf.split(X_train, y_train):
    print("TRAIN:", train_index, "VALIDATION:", val_index)
```

6. Train and Evaluate Using k-Fold Cross-Validation

```
# Initialize model
model = RandomForestClassifier(random_state=42)
# Perform k-Fold Cross-Validation
accuracies = []
for train_index, val_index in kf.split(X_train):
    X_kf_train, X_kf_val = X_train[train_index], X_train[val_index]
    y_kf_train, y_kf_val = y_train[train_index], y_train[val_index]
    # Train model
    model.fit(X_kf_train, y_kf_train)
    # Validate model
    y_pred = model.predict(X_kf_val)
    accuracy = accuracy_score(y_kf_val, y_pred)
    accuracies.append(accuracy)
print(f"Average Accuracy from k-Fold: {np.mean(accuracies):.2f}")
```

7. Hyperparameter Tuning Using GridSearchCV

```
# Define parameter grid
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
}
# Perform GridSearchCV with Stratified k-Fold
grid_search = GridSearchCV(
    estimator=RandomForestClassifier(random_state=42),
    param_grid=param_grid,
    cv=StratifiedKFold(n_splits=5, shuffle=True, random_state=42),
    scoring='accuracy',
    n_jobs=-1,
    verbose=1
)
# Fit to training data
grid_search.fit(X_train, y_train)
print("Best Parameters:", grid_search.best_params_)
print("Best Cross-Validation Accuracy:", grid_search.best_score_)
```

8. Evaluate the Final Model

```
# Use the best model for evaluation
best_model = grid_search.best_estimator_
# Predict on test data
y_test_pred = best_model.predict(X_test)
# Evaluate performance
print("\nTest Accuracy:", accuracy_score(y_test, y_test_pred))
print("\nClassification Report:\n", classification_report(y_test, y_test_pred))
```

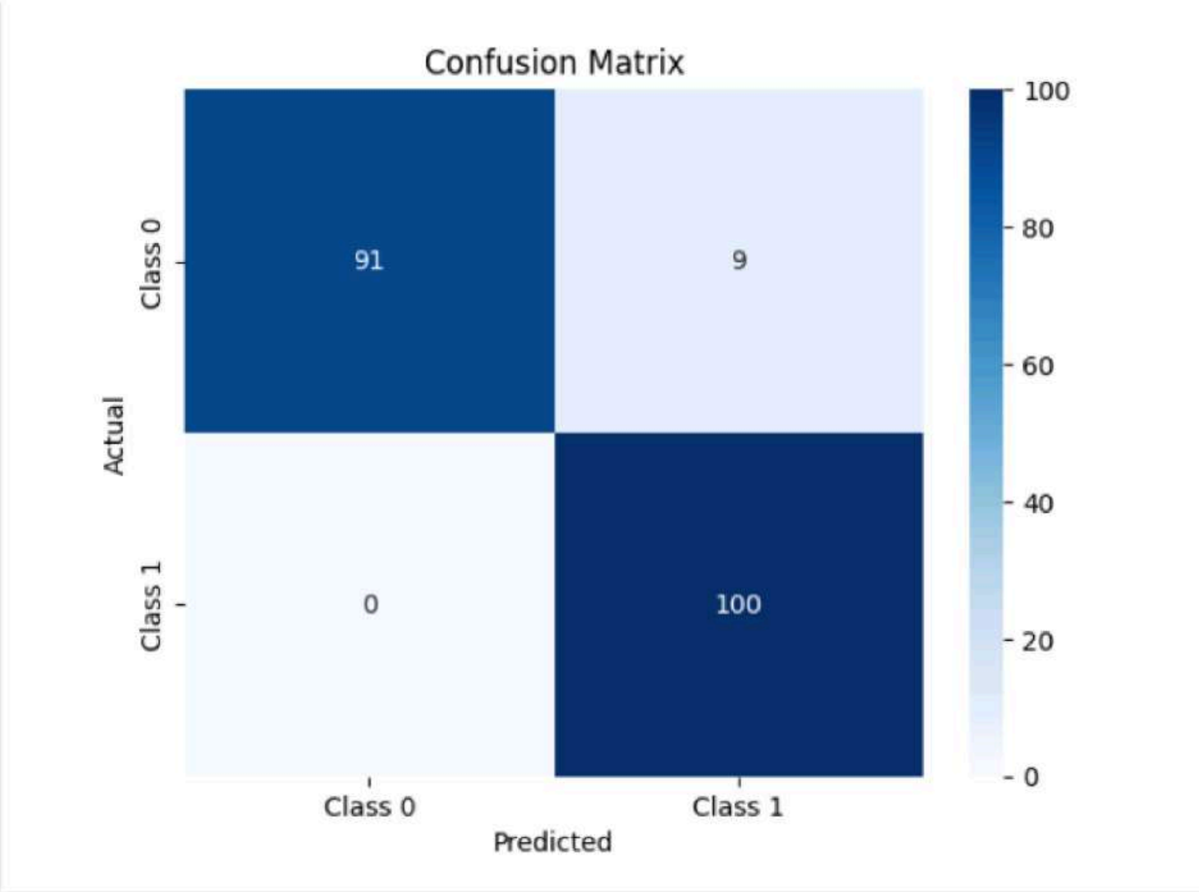
```
# Confusion matrix

conf_matrix = confusion_matrix(y_test, y_test_pred)

sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Class 0', 'Class 1'], yticklabels=['Class 0', 'Class 1'])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()
```

Output :

```
   Feature_1  Feature_2  Feature_3  Feature_4  Feature_5  Feature_6  \
0  -3.358483   3.159918   0.827163   0.069638  -6.715639  -2.708559
1   2.071819  -4.055419  -2.615940  -2.599432   3.053752   0.366795
2  -0.633460   0.712482   2.024390  -0.432639  -1.307929   0.419320
3  -0.464478   0.892442   2.521010   2.766580  -1.933734  -1.418018
4   1.042426  -1.192605  -2.071386  -0.131231   0.545377   0.379060

   Feature_7  Feature_8  Feature_9  Feature_10  Target
0   0.183206   1.113502   1.730759    1.228394       1
1  -0.392171  -1.191720  -1.220516    1.899925       0
2  -1.469510  -0.719051   1.155005    2.018026       0
3   1.391760  -2.430279   1.308295   -0.270896       1
4  -0.062978  -1.325591   2.037936    0.115414       0
k-Fold Cross-Validation:
TRAIN: [  0   1   3   4   5   6   8   9  11  12  13  14  15  16  17  18  19  20
  21  22  24  25  26  27  28  32  34  35  36  37  38  40  41  42  43  44
  45  46  47  48  50  51  52  53  55  56  57  58  59  60  61  62  64  68
  69  70  71  73  74  75  79  80  82  83  85  87  88  89  90  91  92  93
  94  95  98  99 100 102 103 104 105 106 107 108 111 112 113 114 115 116
 117 119 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136
 138 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 156 157
 158 159 160 161 162 163 164 165 166 167 169 170 171 172 173 175 176 177
 178 179 180 181 182 183 184 185 186 187 188 189 190 191 193 194 195 196
 197 200 201 202 203 205 206 207 212 213 214 216 217 219 220 221 222 223
 224 225 226 227 228 229 230 232 233 234 236 237 238 239 240 241 242 243
 245 246 247 248 249 251 252 253 255 256 257 258 259 261 262 263 264 267
 268 269 270 271 272 273 274 276 277 278 279 280 282 283 284 285 287 288
 289 290 291 292 293 295 297 298 299 300 301 303 304 305 307 308 309 310
 311 312 313 315 317 318 319 320 321 322 324 325 328 329 330 331 332 334
```

```
 674 675 676 677 678 679 680 681 682 683 684 685 686 687 689 690 691 693
 694 695 697 698 699 700 701 702 703 704 706 707 708 709 710 711 712 713
 714 716 717 718 722 724 725 726 727 728 729 730 731 732 733 734 735 736
 737 739 740 742 743 745 747 748 749 751 752 753 755 756 757 759 761 762
 763 765 766 767 768 769 770 771 772 773 774 775 778 779 780 781 782 784
 785 788 789 790 791 792 793 794 797 799] VALIDATION: [  2   7  10  23  29  30  31  33  39  49  54  63  65  66  67  72  76  77
  78  81  84  86  96  97 101 109 110 118 120 137 139 155 168 174 192 198
 199 204 208 209 210 211 215 218 231 235 244 250 254 260 265 266 275 281
 286 294 296 302 306 314 316 323 326 327 333 336 346 352 357 360 361 365
 367 368 377 383 388 393 395 398 409 422 423 425 428 432 446 456 464 481
 483 486 490 506 512 513 515 519 521 525 526 529 532 533 534 537 545 568
 570 578 589 591 594 595 596 604 608 610 621 622 628 635 637 640 641 644
 652 655 656 658 662 666 667 670 688 692 696 705 715 719 720 721 723 738
 741 744 746 750 754 758 760 764 776 777 783 786 787 795 796 798]
TRAIN: [  0   1   2   3   4   5   7   8   9  10  12  13  14  15  16  17  19  20
  21  22  23  25  26  27  29  30  31  32  33  34  35  36  37  38  39  40
  45  46  47  48  49  50  52  53  54  57  58  59  62  63  64  65  66  67
  68  71  72  75  76  77  78  80  81  84  85  86  87  88  91  93  94  95
  96  97  98  99 100 101 102 103 105 106 107 109 110 111 112 113 115 116
 117 118 119 120 121 122 123 124 125 126 127 128 129 130 134 137 138 139
 141 142 143 144 146 147 149 150 151 152 153 154 155 156 157 159 160 161
 162 166 168 169 170 171 172 173 174 175 179 180 183 184 185 186 187 188
 189 190 191 192 194 195 197 198 199 200 201 202 203 204 205 206 207 208
 209 210 211 214 215 216 217 218 219 221 222 224 225 226 228 229 230 231
 232 233 235 236 237 238 240 241 242 243 244 245 246 249 250 251 252 253
 254 255 256 257 258 260 261 262 263 265 266 267 268 269 270 271 272 273
 274 275 276 277 278 279 280 281 282 283 284 286 287 288 289 293 294 295
 296 297 298 301 302 303 304 305 306 307 308 310 312 313 314 315 316 317
 318 320 321 322 323 324 325 326 327 330 333 335 336 337 338 339 340 341
```

```
 450 455 459 460 463 470 472 475 480 481 483 494 496 502 503 505 514 519
 521 523 529 533 553 555 560 563 565 579 585 590 594 600 603 620 630 631
 634 635 637 638 644 646 648 649 651 673 675 686 691 693 708 709 715 720
 729 730 738 747 753 759 767 771 774 776 777 780 782 783 784 796]
Average Accuracy from k-Fold: 0.95
Fitting 5 folds for each of 36 candidates, totalling 180 fits
Best Parameters: {'max_depth': None, 'min_samples_split': 5, 'n_estimators': 50}
Best Cross-Validation Accuracy: 0.96

Test Accuracy: 0.955

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.91      0.95       100
           1       0.92      1.00      0.96       100

    accuracy                           0.95       200
   macro avg       0.96      0.96      0.95       200
weighted avg       0.96      0.95      0.95       200
```


Confusion Matrix
Class 0
Class 1
Actual
91
9
0
100
Class 0
Class 1
Predicted
100
80
60
40
20
0

7b. Systematically explore combinations of hyperparameters to optimize model performance.(use grid and randomized search)

Code :

1. Import Necessary Libraries

```
import numpy as np

import pandas as pd

from sklearn.datasets import make_classification

from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV,
StratifiedKFold

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

import matplotlib.pyplot as plt

import seaborn as sns
```

2. Generate a Synthetic Dataset

```
# Generate a binary classification dataset

X, y = make_classification(

    n_samples=1000, n_features=12, n_informative=8, n_redundant=2,

    n_clusters_per_class=1, flip_y=0.03, random_state=42

)

# Convert to a DataFrame for visualization

df = pd.DataFrame(X, columns=[f'Feature_{i}' for i in range(1, 13)])

df['Target'] = y

# Display the first few rows

print(df.head())
```

3. Split Data into Train and Test Sets

```
# Split 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,
stratify=y)
```

4. Define the Model

 # Initialize a Random Forest classifier

model = RandomForestClassifier(random_state=42)

5. Hyperparameter Tuning Using Grid Search

```
# Define a parameter grid for Grid Search
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

 # GridSearchCV with 5-fold cross-validation

```
grid_search = GridSearchCV(
    estimator=model,
    param_grid=param_grid,
    scoring='accuracy',
    cv=StratifiedKFold(n_splits=5, shuffle=True, random_state=42),
    verbose=1,
    n_jobs=-1
)
```

 # Fit the model

```
grid_search.fit(X_train, y_train)
```

 # Best parameters and score from Grid Search

```
print("Best Parameters from Grid Search:", grid_search.best_params_)
print("Best Cross-Validation Accuracy from Grid Search:", grid_search.best_score_)
```

6. Hyperparameter Tuning Using Randomized Search

```
from scipy.stats import randint
```

```
# Define a parameter distribution for Randomized Search
param_dist = {
    'n_estimators': randint(50, 300),
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': randint(2, 15),
    'min_samples_leaf': randint(1, 10)
}
# RandomizedSearchCV with 5-fold cross-validation
random_search = RandomizedSearchCV(
    estimator=model,
    param_distributions=param_dist,
    n_iter=50,  # Number of random combinations to try
    scoring='accuracy',
    cv=StratifiedKFold(n_splits=5, shuffle=True, random_state=42),
    verbose=1,
    n_jobs=-1,
    random_state=42
)
# Fit the model
random_search.fit(X_train, y_train)
# Best parameters and score from Randomized Search
print("Best Parameters from Randomized Search:", random_search.best_params_)
print("Best Cross-Validation Accuracy from Randomized Search:",
random_search.best_score_)
```

7. Evaluate the Best Model

```
# Select the best model from Grid Search and Randomized Search
best_model = random_search.best_estimator_  # Or use grid_search.best_estimator_
# Predict on test data
y_test_pred = best_model.predict(X_test)
# Evaluate the performance
```

print("\nTest Accuracy:", accuracy_score(y_test, y_test_pred))

print("\nClassification Report:\n", classification_report(y_test, y_test_pred))

 # Confusion Matrix

conf_matrix = confusion_matrix(y_test, y_test_pred)

sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Class 0', 'Class 1'], yticklabels=['Class 0', 'Class 1'])

plt.xlabel('Predicted')

plt.ylabel('Actual')

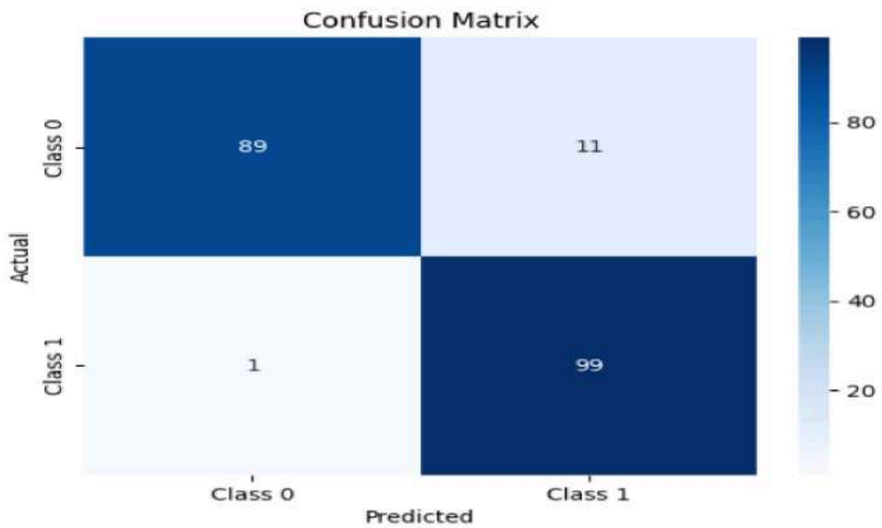
plt.title('Confusion Matrix')

plt.show()

Output :

```
   Feature_1  Feature_2  Feature_3  Feature_4  Feature_5  Feature_6  \
0   0.013650   0.607473  -2.096916   2.867232   2.504360   0.784101
1   0.107199   0.105735  -3.843343   1.524052  -1.619824   0.778334
2  -1.779086  -5.219831  -0.738488   2.108084  -0.803833  -3.431122
3  -4.310656  -2.268569   1.864943  -1.246116   1.260794  -2.007664
4  -3.195179  -0.671327   3.720485   0.356661   0.819486   2.670230

   Feature_7  Feature_8  Feature_9  Feature_10  Feature_11  Feature_12  Target
0  -0.497744  -0.482072   1.112773    1.641637   -2.689832   -0.480311       1
1   0.551177  -1.843583  -0.110132   -0.494739   -0.985276   -0.978400       0
2   1.346120  -0.858351  -0.792415   -2.260815    0.238780    3.029952       0
3  -0.824133  -2.277449   0.936206    1.255903    1.386278   -0.321200       0
4   1.857477  -3.410944  -1.773719    0.656476    3.534189   -1.704889       0
Fitting 5 folds for each of 81 candidates, totalling 405 fits
Best Parameters from Grid Search: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
Best Cross-Validation Accuracy from Grid Search: 0.9487499999999999
Fitting 5 folds for each of 50 candidates, totalling 250 fits
Best Parameters from Randomized Search: {'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 9, 'n_estimators': 285}
Best Cross-Validation Accuracy from Randomized Search: 0.9487499999999999

Test Accuracy: 0.94
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.89      0.94       100
           1       0.90      0.99      0.94       100

    accuracy                           0.94       200
   macro avg       0.94      0.94      0.94       200
weighted avg       0.94      0.94      0.94       200
```



Practical 8 : Bayesian Learning

Implement Bayesian Learning using inferences

Code :

1. Import Necessary 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.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

2. Generate a Synthetic Dataset

We create a dataset suitable for classification problems.

```
# Generate a dataset with 2 classes
X, y = make_classification(
    n_samples=1000, n_features=8, n_informative=6, n_redundant=2,
    n_classes=2, random_state=42)
# Convert to DataFrame for visualization
df = pd.DataFrame(X, columns=[f'Feature_{i}' for i in range(1, 9)])
df['Target'] = y
# Display the first few rows
print(df.head())
```

3. Split the Dataset

Divide the data into training and testing sets.

 # Split data into 80% training and 20% testing

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

4. Bayesian Learning with Naive Bayes

Here, we implement Bayesian Learning using the Gaussian Naive Bayes classifier.

 # Initialize the Gaussian Naive Bayes model

model = GaussianNB()

 # Fit the model to the training data

model.fit(X_train, y_train)

 # Predict on the test data

y_pred = model.predict(X_test)

5. Evaluate the Model

We evaluate the model's performance using accuracy, classification report, and confusion matrix.

 # Calculate accuracy

accuracy = accuracy_score(y_test, y_pred)

print(f"Test Accuracy: {accuracy:.2f}")

 # Print classification report

print("\nClassification Report:\n", classification_report(y_test, y_pred))

 # Generate and plot confusion matrix

conf_matrix = confusion_matrix(y_test, y_pred)

sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Class 0', 'Class 1'], yticklabels=['Class 0', 'Class 1'])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

6. Understanding Bayesian Inference

In Bayesian Learning, the model predicts based on the probabilities:

- **Prior Probability (P(C)P(C)P(C)):** The likelihood of each class based on historical data.
- **Likelihood (P(X|C)P(X|C)P(X|C)):** The probability of the data given a class.
- **Posterior Probability (P(C|X)P(C|X)P(C|X)):** Calculated using Bayes' theorem: P(C|X)=P(X|C)·P(C)P(X)P(C|X) = \frac{P(X|C) \cdot P(C)}{P(X)}P(C|X)=P(X)P(X|C)·P(C)

 # Example: Compute posterior probabilities for the first test sample

sample = X_test[0].reshape(1, -1)

posterior_probs = model.predict_proba(sample)

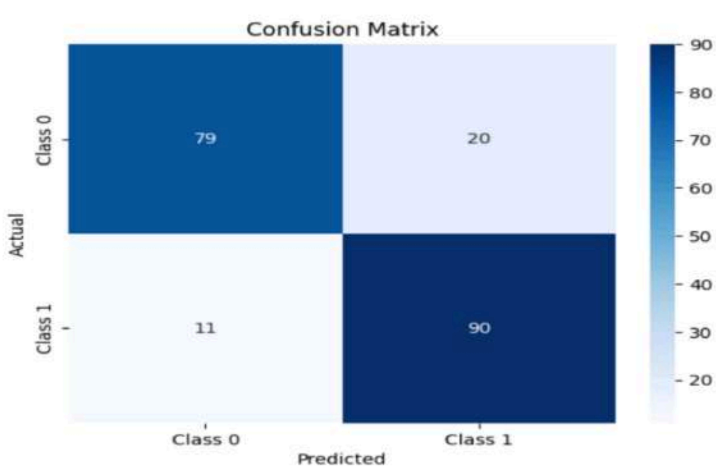
print(f"Sample Features: {sample}")

print(f"Posterior Probabilities: {posterior_probs}")

print(f"Predicted Class: {model.predict(sample)}")

Output :

```
   Feature_1  Feature_2  Feature_3  Feature_4  Feature_5  Feature_6  \
0  -1.732538   5.260112  -2.952194  -4.603768   2.235848   1.928893
1   2.072914   2.240572  -1.385104  -2.514962  -0.984756   1.436260
2  -0.263106   1.527781  -1.872414  -0.028009   1.612809   3.264194
3  -0.164349  -0.550131  -0.019503  -0.765000   2.273523   2.084217
4  -1.419423   1.015324  -0.864441  -0.009297   0.385404   0.449093

   Feature_7  Feature_8  Target
0  -0.101845   3.193487       0
1  -1.255271   2.089872       0
2  -1.296421   1.537870       0
3  -0.321931   0.426253       0
4  -0.029007  -1.902917       1
Test Accuracy: 0.84

Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.80      0.84        99
           1       0.82      0.89      0.85       101

    accuracy                           0.84       200
   macro avg       0.85      0.84      0.84       200
weighted avg       0.85      0.84      0.84       200
```




```
Sample Features: [[ 1.94397502  0.33753289 -0.77968371  2.63667244 -1.62323864  1.87859635
  -2.20522563  2.09632056]]
Posterior Probabilities: [[0.07067404 0.92932596]]
Predicted Class: [1]
```