**PRACTICAL NO : 1**

**Data Pre-processing and Exploration**

**Aim:[A] Load a CSV dataset. Handle missing values, inconsistent formatting, and outliers.**

**CODE:**

#Import necessary libraries

import pandas as pd

import numpy as np

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import StandardScaler

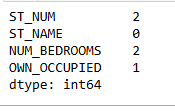
#Load the CSV dataset

df = pd.read\_csv('C:\\Users\\Lenovo\\Desktop\\MSC IT PROGRAM - Poornima\\SEM 3\\ML\\PRACTIAL 1 DATASETS.csv')

#Check for missing values

# Show the count of missing values in each column

print(df.isnull().sum())



df = df.dropna() # Removes all rows with missing data

# Impute missing numerical values with the mean

imputer = SimpleImputer(strategy='mean')

df['ST\_NUM'] = imputer.fit\_transform(df[['ST\_NUM']])

# Fill categorical missing values with mode (most frequent value)

df['OWN\_OCCUPIED'].fillna(df['OWN\_OCCUPIED'].mode()[0], inplace=True)

df['OWN\_OCCUPIED'] = df['OWN\_OCCUPIED'].str.strip()

df['OWN\_OCCUPIED'] = df['OWN\_OCCUPIED'].str.lower()

from scipy import stats

# Compute Z-scores and remove data points with Z-scores > 3 or < -3

z\_scores = np.abs(stats.zscore(df['ST\_NUM']))

df = df[(z\_scores < 3)]

Q1 = df['ST\_NUM'].quantile(0.25)

Q3 = df['ST\_NUM'].quantile(0.75)

IQR = Q3 - Q1

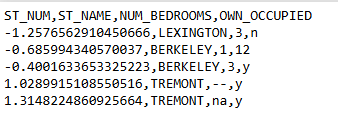
# Remove rows that have outliers (data points outside 1.5 times the IQR)

df = df[~((df['ST\_NUM'] < (Q1 - 1.5 \* IQR)) | (df['ST\_NUM'] > (Q3 + 1.5 \* IQR)))]

scaler = StandardScaler()

df['ST\_NUM'] = scaler.fit\_transform(df[['ST\_NUM']])

df.to\_csv('cleaned\_dataset.csv', index=False)



**Aim:[B] Load a dataset, calculate descriptive summary statistics, create visualizations using different graphs, and identify potential features and target variables**

**CODE:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

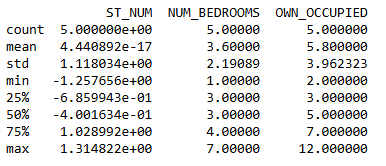
import seaborn as sns

# Load dataset (replace 'your\_dataset.csv' with your file path)

df = pd.read\_csv("C:\\Users\\Lenovo\\Desktop\\MSC IT PROGRAM - Poornima\\SEM 3\\ML\\PRACTIAL 1 ML\\cleaned\_dataset.csv")

# Descriptive statistics for numerical columns

print(df.describe())



# Descriptive statistics for categorical columns

print(df.describe(include='object'))

# Checking for unique values in each column

print(df.nunique())

# Checking for missing values in the dataset

print(df.isnull().sum())

# ⦁ Histogram (for numerical data):

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

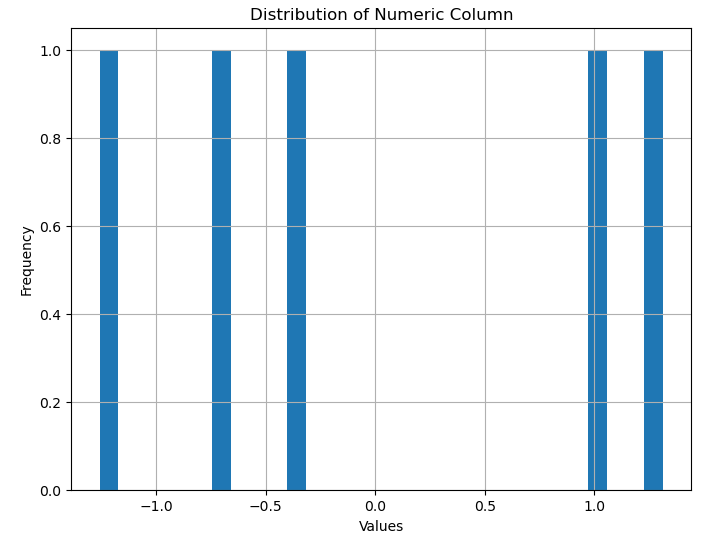
df['ST\_NUM'].hist(bins=30)

plt.title('Distribution of Numeric Column')

plt.xlabel('Values')

plt.ylabel('Frequency')

plt.show()



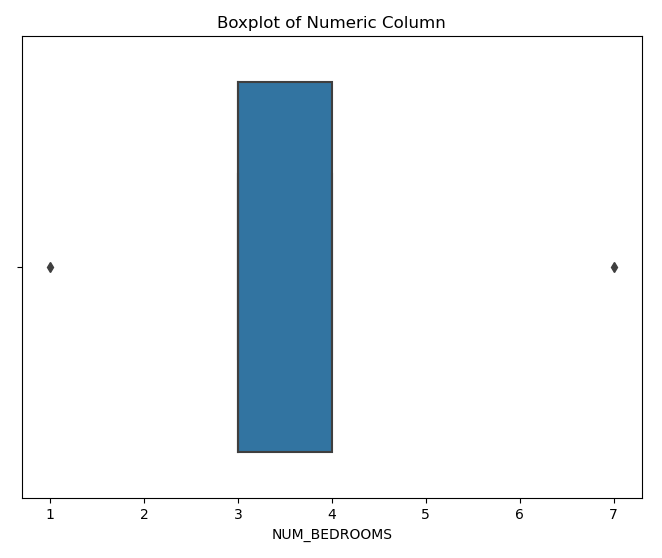
# ⦁ Boxplot (for detecting outliers):

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

sns.boxplot(x=df['NUM\_BEDROOMS'])

plt.title('Boxplot of Numeric Column')

plt.show()



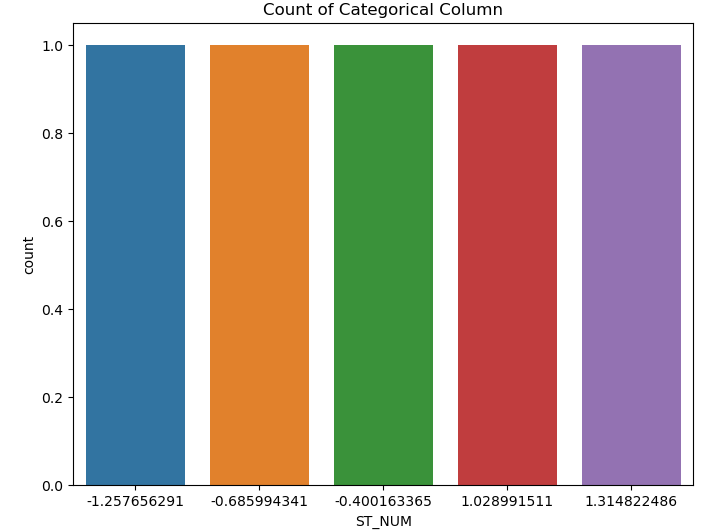
# ⦁ Count plot (for categorical variables):

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

sns.countplot(x=df['ST\_NUM'])

plt.title('Count of Categorical Column')

plt.show()



# Scatter Plot

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

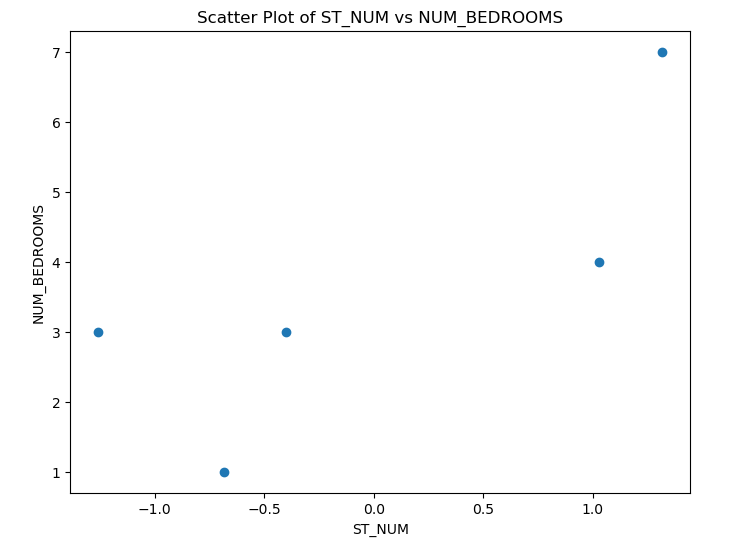
plt.scatter(df['ST\_NUM'], df['NUM\_BEDROOMS'])

plt.title('Scatter Plot of ST\_NUM vs NUM\_BEDROOMS')

plt.xlabel('ST\_NUM')

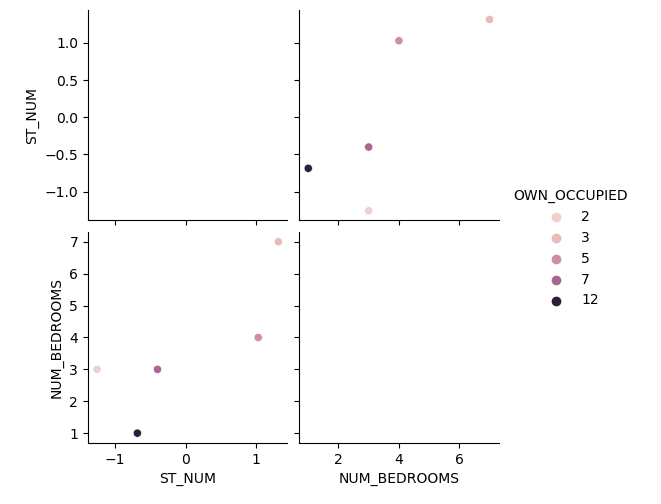
plt.ylabel('NUM\_BEDROOMS')

plt.show()

  
# Pairplot (with categorical hue)

sns.pairplot(df[['ST\_NUM', 'NUM\_BEDROOMS', 'OWN\_OCCUPIED']], hue='OWN\_OCCUPIED')

plt.show()



# Correlation Heatmap (excluding non-numeric columns)

plt.figure(figsize=(10, 8))

# Select only numeric columns for the correlation calculation

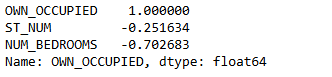
numeric\_df = df.select\_dtypes(include=[float, int])

correlation\_matrix = numeric\_df.corr()

# Correlation with target column

correlation = df.corr()['OWN\_OCCUPIED'].sort\_values(ascending=False)

print(correlation)

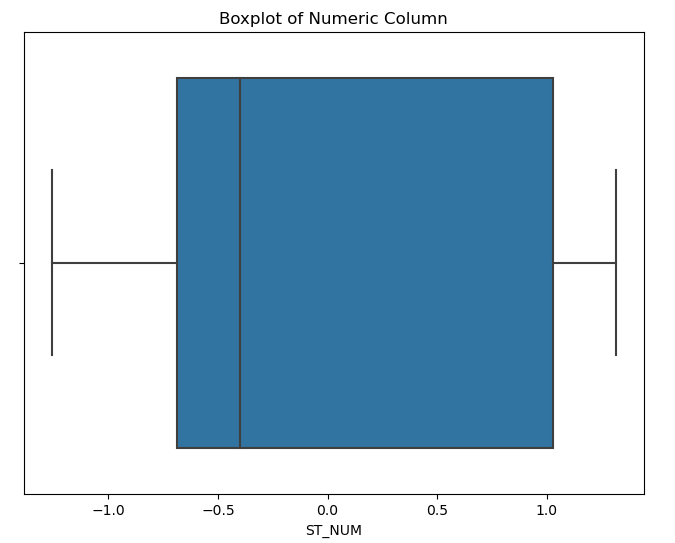


plt.figure(figsize=(8, 6))

sns.boxplot(x=df['ST\_NUM'])

plt.title('Boxplot of Numeric Column')

plt.savefig('boxplot.png')



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

**CODE:**

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler, Binarizer

# Create a sample dataset

data = {

'Age': [22, 25, 47, 52, 46, 56, 36],

'Salary': [21000, 25000, 47000, 52000, 46000, 56000, 36000],

'City': ['New York', 'Paris', 'Berlin', 'New York', 'Berlin', 'Paris', 'Berlin'],

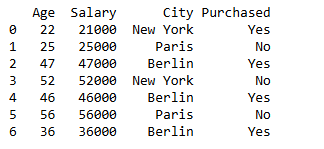
'Purchased': ['Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes']

}

# Convert into a pandas DataFrame

df = pd.DataFrame(data)

print(df)



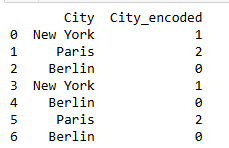
# Initialize the label encoder

label\_encoder = LabelEncoder()

# Encode the 'City' column

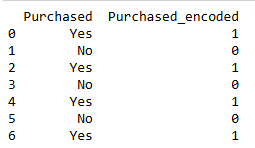
df['City\_encoded'] = label\_encoder.fit\_transform(df['City'])

print(df[['City', 'City\_encoded']])



df['Purchased\_encoded'] = label\_encoder.fit\_transform(df['Purchased'])

print(df[['Purchased', 'Purchased\_encoded']])



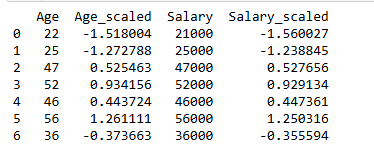
# Initialize the standard scaler

scaler = StandardScaler()

# Apply scaling to 'Age' and 'Salary' columns

df[['Age\_scaled', 'Salary\_scaled']] = scaler.fit\_transform(df[['Age', 'Salary']])

print(df[['Age', 'Age\_scaled', 'Salary', 'Salary\_scaled']])



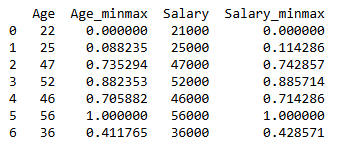
# Initialize the MinMaxScaler

minmax\_scaler = MinMaxScaler()

# Apply scaling to 'Age' and 'Salary'

df[['Age\_minmax', 'Salary\_minmax']] = minmax\_scaler.fit\_transform(df[['Age', 'Salary']])

print(df[['Age', 'Age\_minmax', 'Salary', 'Salary\_minmax']])



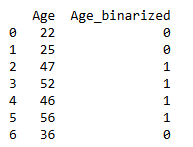
# Initialize the binarizer

binarizer = Binarizer(threshold=40)

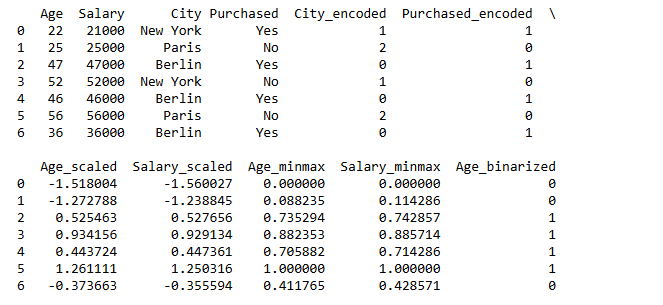
# Apply binarization to 'Age'

df['Age\_binarized'] = binarizer.fit\_transform(df[['Age']])

print(df[['Age', 'Age\_binarized']])



print(df)



**PRACTICAL NO : 2**

**Testing Hypothesis**

**Aim:[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)**

**CODE:**

import csv

num\_attributes = 6 # We have 6 attributes excluding the class

a = []

print("\nThe Given Training Dataset \n")

with open("Book1.csv", "r") as csvfile:

reader = csv.reader(csvfile)

count = 0

for row in reader:

if count == 0:

print(row)

count += 1

else:

a.append(row)

print(row)

count += 1

print("\nThe initial value of hypothesis: ")

hypothesis = ["0"] \* num\_attributes # Initialize hypothesis with placeholders

print(hypothesis)

# Set the hypothesis to the first row (excluding class)

for j in range(0, num\_attributes):

hypothesis[j] = a[0][j]

print(hypothesis)

print("\nFind S: Finding a Maximally Specific Hypothesis\n")

for i in range(0, len(a)):

if a[i][num\_attributes] == "Yes": # Check if the class label is 'Yes'

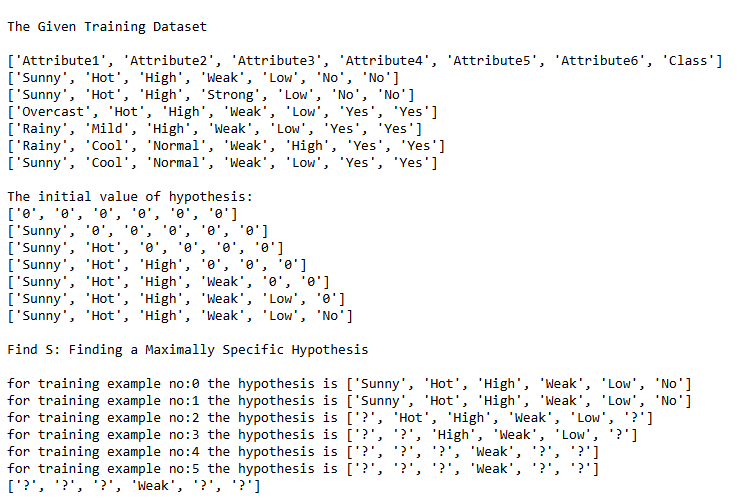
for j in range(0, num\_attributes):

if a[i][j] != hypothesis[j]: # If the attribute doesn't match, set to '?'

hypothesis[j] = "?"

print(f"for training example no:{i} the hypothesis is {hypothesis}")

print(hypothesis)



**PRACTICAL NO : 3**

**Linear Models**

**Aim:[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**

**CODE:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

np.random.seed(0)

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

y = 20 \* X + 1 + np.random.randn(100, 1)

data = pd.DataFrame({'X': X.flatten(), 'y': y.flatten()})

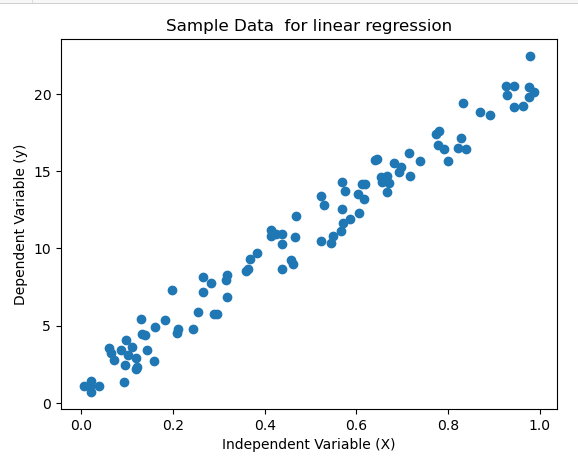
plt.scatter(X , y)

plt.xlabel('Independent Variable (X)')

plt.ylabel('Dependent Variable (y)')

plt.title('Sample Data for linear regression')

plt.show()



from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state= 42 )

model = LinearRegression()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

r\_squared = r2\_score(y\_test, y\_pred)

print(f'Mean square error (mse): {mse:.2f}')

print(f'R squared(r): {r\_squared:.2f}')



plt.scatter(X, y, label='Data')

plt.plot(X\_test, y\_pred, color='red', linewidth = 2, label= 'Regression Line')

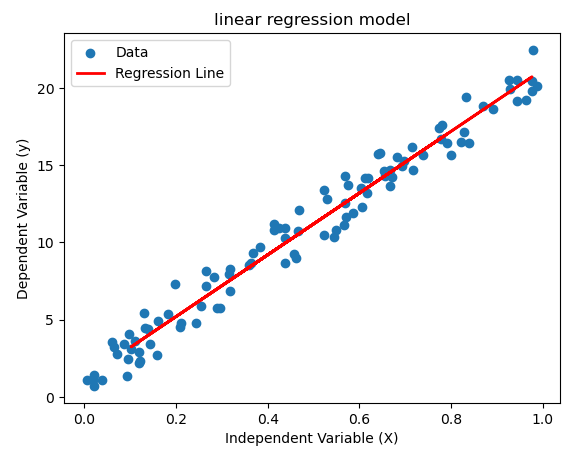
plt.xlabel('Independent Variable (X)')

plt.ylabel('Dependent Variable (y)')

plt.title('linear regression model')

plt.legend()

plt.show()



coefficients = model.coef\_

intercept = model.intercept\_

print(f'coefficients: {coefficients[0][0]:.2f}')

print(f'intercept: {intercept[0]:.2f}')



**Aim:[B] Multiple Linear Regression**

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

**CODE:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

from sklearn.decomposition import PCA

# Synthetic data generation

np.random.seed(0)

X1 = np.random.rand(100, 1) # Feature 1

X2 = 2 \* X1 + np.random.randn(100, 1) \* 0.1 # Feature 2 (correlated with X1)

X3 = np.random.rand(100, 1) # Feature 3 (independent)

y = 10 \* X1 + 5 \* X2 + 3 \* X3 + np.random.randn(100, 1) # Target variable with noise

# Combine features into a DataFrame

X = np.hstack([X1, X2, X3])

data = pd.DataFrame(X, columns=['X1', 'X2', 'X3'])

data['y'] = y

# Check multicollinearity using VIF

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

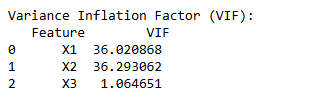
vif\_data = pd.DataFrame({

"Feature": ['X1', 'X2', 'X3'],

"VIF": [variance\_inflation\_factor(X\_scaled, i) for i in range(X\_scaled.shape[1])]

})

print("Variance Inflation Factor (VIF):\n", vif\_data)



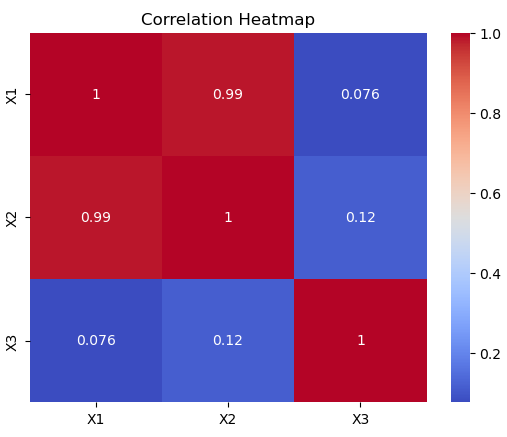
# Visualize correlation

corr = pd.DataFrame(X, columns=['X1', 'X2', 'X3']).corr()

sns.heatmap(corr, annot=True, cmap="coolwarm")

plt.title("Correlation Heatmap")

plt.show()



# Address multicollinearity with PCA if necessary

pca = PCA(n\_components=2) # Reduce to 2 components

X\_pca = pca.fit\_transform(X\_scaled)

# Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

X\_train\_pca, X\_test\_pca, y\_train, y\_test = train\_test\_split(X\_pca, y, test\_size=0.2, random\_state=42)

# Fit and evaluate model without PCA

model = LinearRegression()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

print(f"\nModel Without PCA: MSE = {mean\_squared\_error(y\_test, y\_pred):.2f}, R^2 = {r2\_score(y\_test, y\_pred):.2f}")

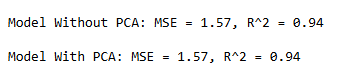
# Fit and evaluate model with PCA

model\_pca = LinearRegression()

model\_pca.fit(X\_train\_pca, y\_train)

y\_pred\_pca = model\_pca.predict(X\_test\_pca)

print(f"\nModel With PCA: MSE = {mean\_squared\_error(y\_test, y\_pred\_pca):.2f}, R^2 = {r2\_score(y\_test, y\_pred\_pca):.2f}")



# Visualization of predictions

plt.scatter(y\_test, y\_pred, label='Without PCA', alpha=0.7)

plt.scatter(y\_test, y\_pred\_pca, label='With PCA', alpha=0.7)

plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], color='red', linestyle='--', linewidth=2)

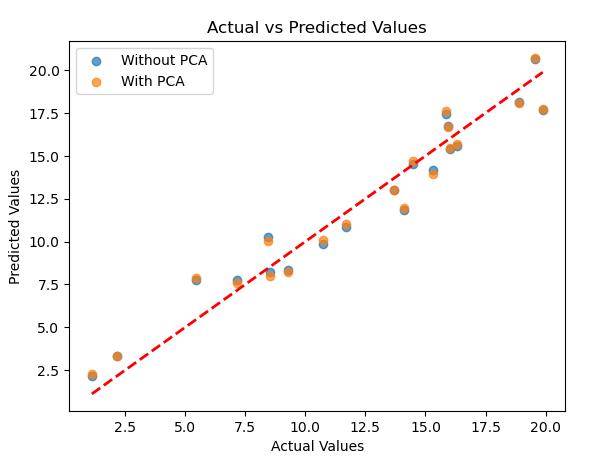
plt.xlabel('Actual Values')

plt.ylabel('Predicted Values')

plt.title('Actual vs Predicted Values')

plt.legend()

plt.show()



**Aim:[C] Regularized Linear Models**

**Implement regression variants like LASSO and Ridge on any generated dataset.**

**CODE:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import fetch\_california\_housing

from sklearn.model\_selection import train\_test\_split

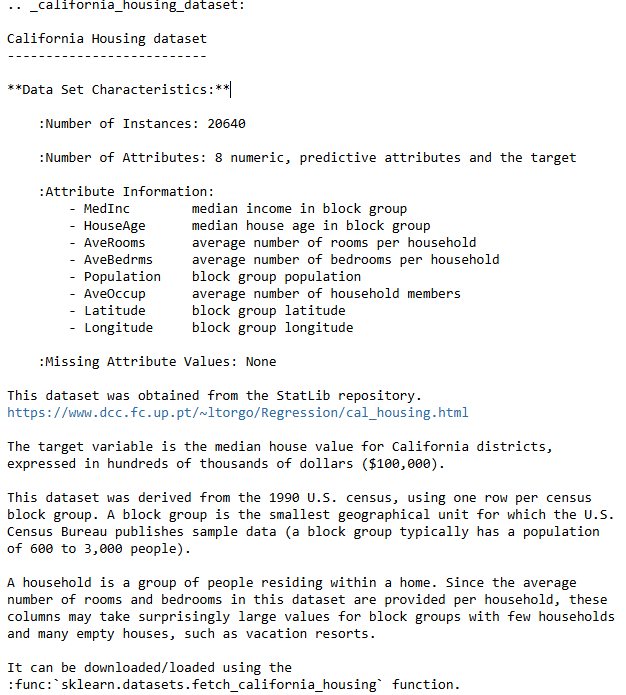
from sklearn.linear\_model import LinearRegression

# Loading California Housing Dataset

california\_dataset = fetch\_california\_housing()

# Display the dataset description

print(california\_dataset.DESCR)



# Convert the dataset into a Pandas DataFrame

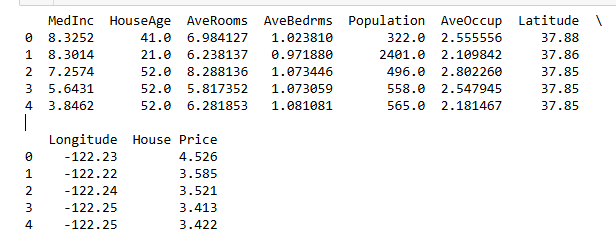
california\_df = pd.DataFrame(california\_dataset.data, columns=california\_dataset.feature\_names)

# Add the target variable (House Price) to the DataFrame

california\_df['House Price'] = california\_dataset.target

# Display the first few rows of the dataset

print(california\_df.head())



# Split the data into input (X) and output (y)

X = california\_df.iloc[:, :-1] # All columns except the target

y = california\_df.iloc[:, -1] # Target variable (House Price)

# Split the data into training and testing sets (75% for training, 25% for testing)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=42)

print(f"Training data shape: X = {X\_train.shape}, y = {y\_train.shape}")

print(f"Testing data shape: X = {X\_test.shape}, y = {y\_test.shape}")



# Create and train the linear regression model

lr\_model = LinearRegression()

lr\_model.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = lr\_model.predict(X\_test)

# Calculate Mean Squared Error

mse = np.mean((y\_pred - y\_test) \*\* 2)

print(f"Mean Squared Error: {mse}")

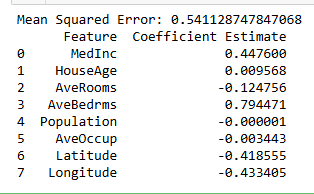
# Display the coefficients

coefficients = pd.DataFrame()

coefficients['Feature'] = X\_train.columns

coefficients['Coefficient Estimate'] = lr\_model.coef\_

print(coefficients)



# Plotting the coefficient scores

fig, ax = plt.subplots(figsize=(10, 6))

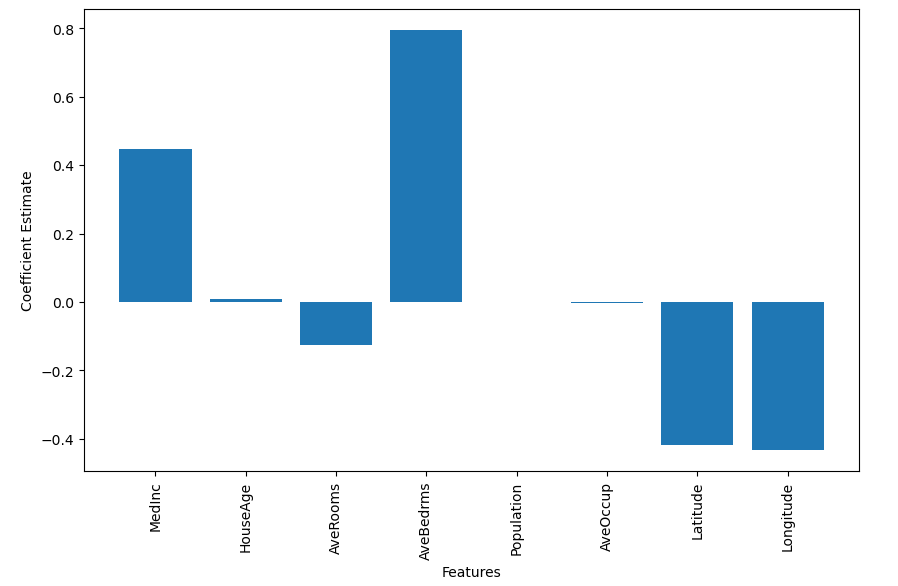
ax.bar(coefficients['Feature'], coefficients['Coefficient Estimate'])

ax.set\_xlabel('Features')

ax.set\_ylabel('Coefficient Estimate')

plt.xticks(rotation=90)

plt.show()



# Ridge Regression

from sklearn.linear\_model import Ridge

ridge\_model = Ridge(alpha=1)

ridge\_model.fit(X\_train, y\_train)

ridge\_pred = ridge\_model.predict(X\_test)

ridge\_mse = np.mean((ridge\_pred - y\_test) \*\* 2)

print(f"Ridge Regression MSE: {ridge\_mse}")



#Lasso

from sklearn.linear\_model import Lasso

lasso\_model = Ridge(alpha=1)

lasso\_model.fit(X\_train, y\_train)

lasso\_pred = Lasso\_model.predict(X\_test)

lasso\_mse = np.mean((lasso\_pred - y\_test) \*\* 2)

print(f"Lasso Regression MSE: {lasso\_mse}")



#ElasticNet

from sklearn.linear\_model import ElasticNet

ElasticNet\_model = ElasticNet(alpha=1)

ElasticNet\_model.fit(X\_train, y\_train)

ElasticNet\_pred = ElasticNet\_model.predict(X\_test)

ElasticNet\_mse = np.mean((ElasticNet\_pred - y\_test) \*\* 2)

print(f"ElasticNet Regression MSE: {ElasticNet\_mse}")



**PRACTICAL NO : 4**

**Discriminative Models**

**Aim:[A] Logistic Regression**

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

**CODE:**

import matplotlib.pyplot as plt

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\_recall\_curve, auc

# Generate a synthetic dataset

X, y = make\_classification(

n\_samples=1000, n\_features=20, 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=0.2, random\_state=42)

# Train a logistic regression model (you can replace this with your own classifier)

model = LogisticRegression()

model.fit(X\_train, y\_train)

# Predict probabilities for positive class

y\_scores = model.predict\_proba(X\_test)[:, 1]

# Calculate precision and recall

precision, recall, thresholds = precision\_recall\_curve(y\_test, y\_scores)

# Calculate Area Under the Curve (AUC) for precision-recall curve

auc\_score = auc(recall, precision)

# Plot precision-recall curve

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

plt.plot(recall, precision, label=f'Precision-Recall Curve (AUC = {auc\_score:.2f})')

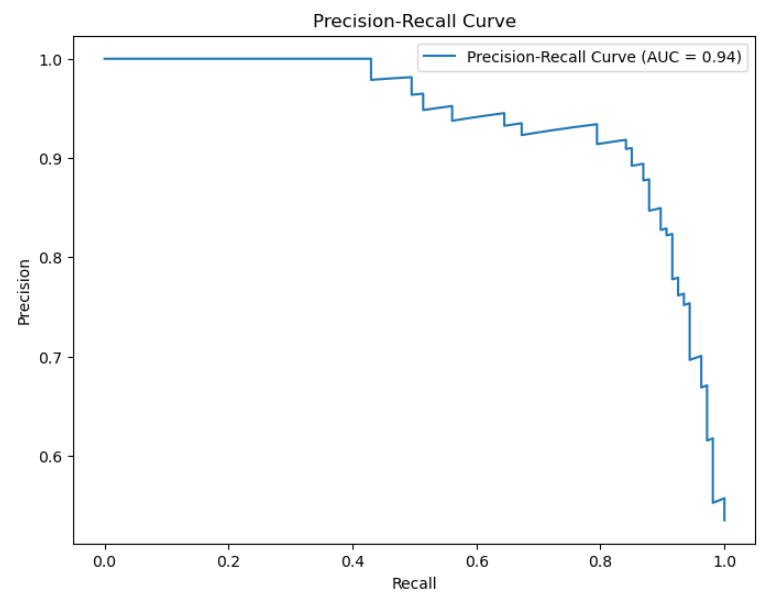
plt.xlabel('Recall')

plt.ylabel('Precision')

plt.title('Precision-Recall Curve')

plt.legend()

plt.show()



from sklearn.metrics import accuracy\_score

# Predict class labels

y\_pred = model.predict(X\_test)

# Calculate accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

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



from sklearn.metrics import roc\_curve, roc\_auc\_score

# Compute ROC curve

fpr, tpr, \_ = roc\_curve(y\_test, y\_scores)

# Compute AUC for ROC curve

roc\_auc = roc\_auc\_score(y\_test, y\_scores)

# Plot ROC curve

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

plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {roc\_auc:.2f})')

plt.plot([0, 1], [0, 1], linestyle='--', color='gray') # Random guess line

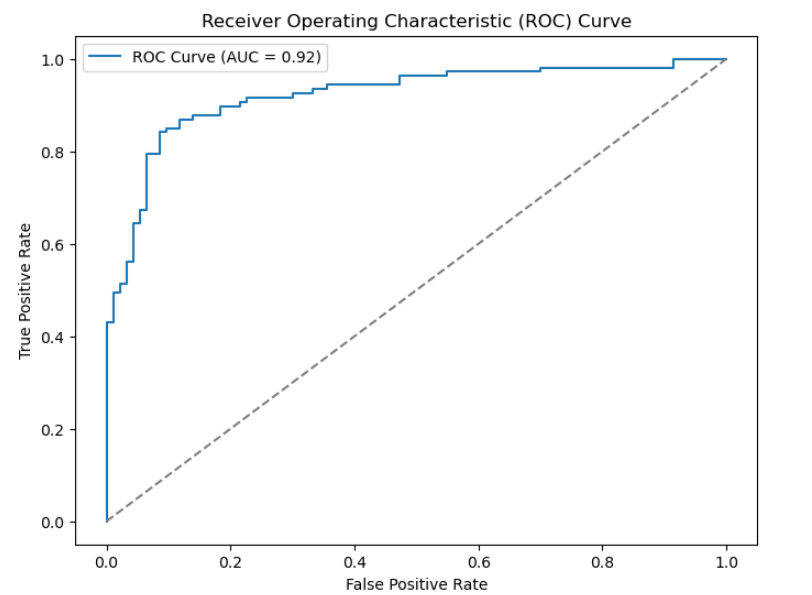
plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curve')

plt.legend()

plt.show()

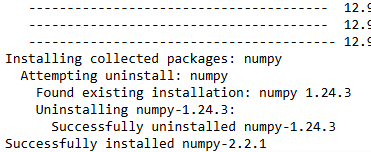


**Aim:[B]**

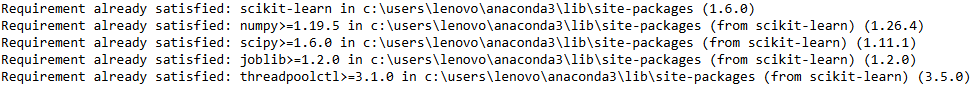
**Implement and demonstrate k-nearest Neighbor algorithm. Read the training data from a .CSV file and build the model to classify a test sample. Print both correct and wrong predictions.**

**CODE:**

pip install --upgrade numpy



pip install --upgrade scikit-learn



import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score

from sklearn.preprocessing import LabelEncoder

# Load the Iris dataset from CSV file

data = pd.read\_csv('iris.csv')

# Convert 'species' column to numeric values

label\_encoder = LabelEncoder()

data['label'] = label\_encoder.fit\_transform(data['species']) # Use 'species' for LabelEncoder

# Define features (X) and target (y)

X = data.iloc[:, :-2].values # Exclude the last column (label) and 'species' column

y = data['label'].values # Use the 'label' column for targets

# 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)

# Print the sizes of train and test sets

print(f'Training samples: {X\_train.shape[0]}')

print(f'Test samples: {X\_test.shape[0]}')

# Initialize the K-Nearest Neighbors classifier with n\_neighbors=4

knn = KNeighborsClassifier(n\_neighbors=10)

# Train the model

knn.fit(X\_train, y\_train)

# Predict on the test set

y\_pred = knn.predict(X\_test)

# Calculate accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

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

# Print correct and wrong predictions

correct\_predictions = [(x, y\_true, y\_pred) for x, y\_true, y\_pred in zip(X\_test, y\_test, y\_pred) if y\_true == y\_pred]

wrong\_predictions = [(x, y\_true, y\_pred) for x, y\_true, y\_pred in zip(X\_test, y\_test, y\_pred) if y\_true != y\_pred]

# Print correct predictions

print("\nCorrect Predictions:")

for sample, true\_label, pred\_label in correct\_predictions:

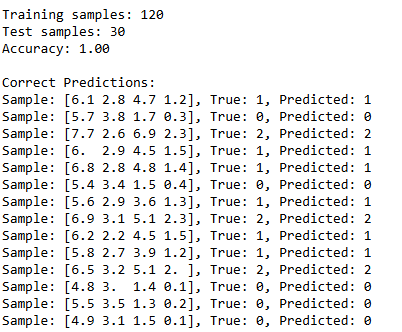
print(f"Sample: {sample}, True: {true\_label}, Predicted: {pred\_label}")

# Print wrong predictions

print("\nWrong Predictions:")

for sample, true\_label, pred\_label in wrong\_predictions:

print(f"Sample: {sample}, True: {true\_label}, Predicted: {pred\_label}")



from sklearn.model\_selection import cross\_val\_score

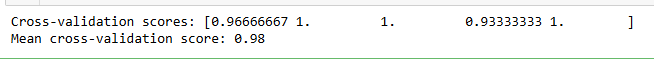
# Perform 5-fold cross-validation

cv\_scores = cross\_val\_score(knn, X, y, cv=5)

# Print the cross-validation scores

print(f'Cross-validation scores: {cv\_scores}')

print(f'Mean cross-validation score: {cv\_scores.mean():.2f}')



**Aim:[C]**

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

**CODE:**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier, plot\_tree

from sklearn.metrics import accuracy\_score

import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder

# Load the dataset

data = pd.read\_csv('iris.csv')

# Convert 'species' column to numeric values

label\_encoder = LabelEncoder()

data['species'] = label\_encoder.fit\_transform(data['species'])

# Define features (X) and target (y)

X = data.iloc[:, :-1].values # Features

y = data.iloc[:, -1].values # Target

# 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)

# Try adjusting the depth and other parameters

tree = DecisionTreeClassifier(max\_depth=3, min\_samples\_split=4, min\_samples\_leaf=2, random\_state=42)

tree.fit(X\_train, y\_train)

# Visualize the tree

plt.figure(figsize=(12, 8))

plot\_tree(tree, filled=True, feature\_names=data.columns[:-1], class\_names=label\_encoder.classes\_, rounded=True)

plt.title('Decision Tree Classifier')

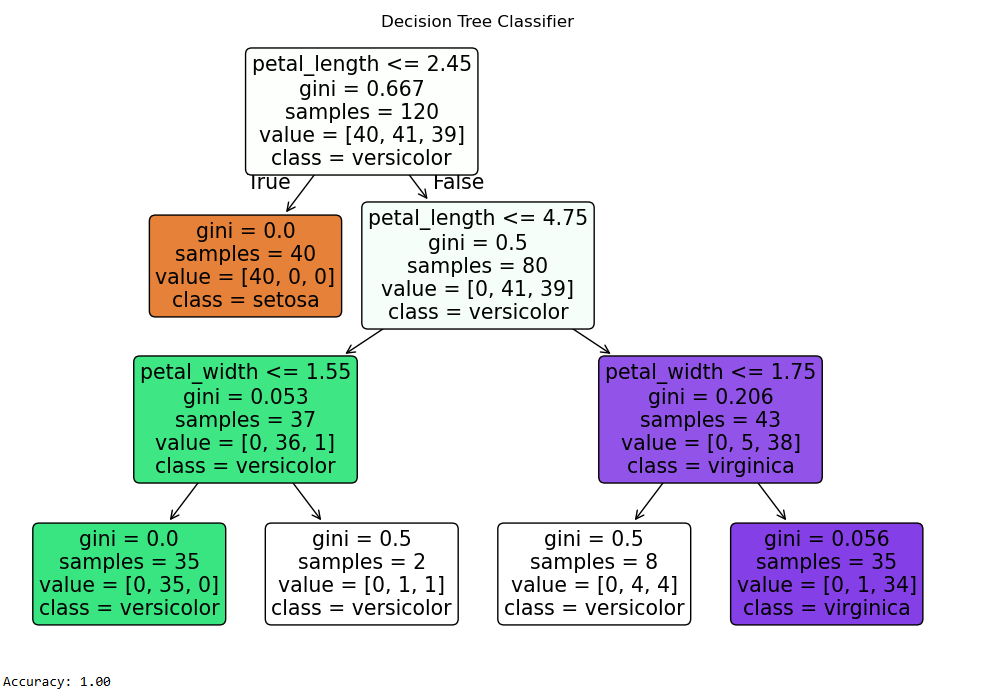
plt.show()

# Check accuracy again

y\_pred = tree.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

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



**Aim:[D]**

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

**CODE:**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score

from sklearn.preprocessing import LabelEncoder

# Load the Iris dataset from CSV file

data = pd.read\_csv('iris.csv')

# Convert the 'species' column to numeric values using LabelEncoder

label\_encoder = LabelEncoder()

data['label'] = label\_encoder.fit\_transform(data['species'])

# Ensure the features are numeric and the 'species' column is removed

# Features: sepal\_length, sepal\_width, petal\_length, petal\_width

X = data[['sepal\_length', 'sepal\_width', 'petal\_length', 'petal\_width']].values

y = data['label'].values # Target: species (encoded labels)

# Double-check that X contains only numeric values

print("Features (X):")

print(X[:5]) # Print the first 5 rows of X

# 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 the Support Vector Machine (SVM) model

svm\_model = SVC(kernel='linear', random\_state=42) # Using a linear kernel

# Train the SVM model on the training data

svm\_model.fit(X\_train, y\_train)

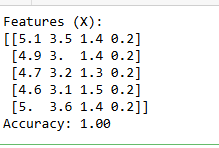
# Predict the labels on the test set

y\_pred = svm\_model.predict(X\_test)

# Calculate and print the accuracy of the model

accuracy = accuracy\_score(y\_test, y\_pred)

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



from sklearn.metrics import confusion\_matrix, classification\_report

# Confusion Matrix

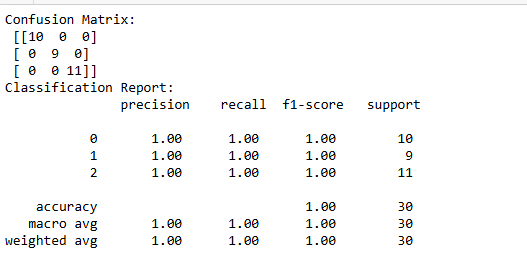
cm = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:\n", cm)

# Classification Report

cr = classification\_report(y\_test, y\_pred)

print("Classification Report:\n", cr)



from sklearn.model\_selection import cross\_val\_score

cross\_val\_scores = cross\_val\_score(svm\_model, X, y, cv=5)

print(f"Cross-validation scores: {cross\_val\_scores}")

print(f"Mean cross-validation score: {cross\_val\_scores.mean()}")



import numpy as np

import matplotlib.pyplot as plt

# Use only the first two features for visualization (e.g., sepal length and sepal width)

X\_train\_2d = X\_train[:, :2]

# Train the model again with only the 2D data

svm\_model.fit(X\_train\_2d, y\_train)

# Create a meshgrid for plotting

h = .02 # Step size in the mesh

x\_min, x\_max = X\_train\_2d[:, 0].min() - 1, X\_train\_2d[:, 0].max() + 1

y\_min, y\_max = X\_train\_2d[:, 1].min() - 1, X\_train\_2d[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h), np.arange(y\_min, y\_max, h))

# Predict the class for each point in the meshgrid

Z = svm\_model.predict(np.c\_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

# Plotting the decision boundary and the points

plt.contourf(xx, yy, Z, alpha=0.8)

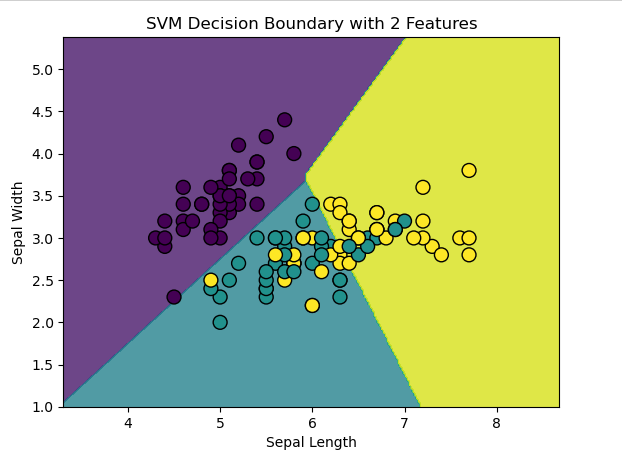
plt.scatter(X\_train\_2d[:, 0], X\_train\_2d[:, 1], c=y\_train, edgecolors='k', marker='o', s=100)

plt.title("SVM Decision Boundary with 2 Features")

plt.xlabel('Sepal Length')

plt.ylabel('Sepal Width')

plt.show()



**Aim:[E]**

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

**CODE:**

# Import necessary libraries

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score

from sklearn.datasets import load\_iris

# Load dataset (you can replace this with your own dataset)

iris = load\_iris()

X = iris.data # Features

y = iris.target # Labels

# Split the 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)

# Step 1: Train a Decision Tree Classifier

dt\_model = DecisionTreeClassifier(random\_state=42)

dt\_model.fit(X\_train, y\_train)

# Make predictions with the Decision Tree

y\_pred\_dt = dt\_model.predict(X\_test)

# Evaluate the Decision Tree

accuracy\_dt = accuracy\_score(y\_test, y\_pred\_dt)

print(f"Decision Tree Accuracy: {accuracy\_dt:.4f}")

# Step 2: Train a Random Forest Classifier with default parameters

rf\_model\_default = RandomForestClassifier(random\_state=42)

rf\_model\_default.fit(X\_train, y\_train)

# Make predictions with the Random Forest (default)

y\_pred\_rf\_default = rf\_model\_default.predict(X\_test)

# Evaluate the Random Forest (default)

accuracy\_rf\_default = accuracy\_score(y\_test, y\_pred\_rf\_default)

print(f"Random Forest (Default) Accuracy: {accuracy\_rf\_default:.4f}")

# Step 3: Train a Random Forest Classifier with custom parameters (more trees and feature sampling)

rf\_model\_custom = RandomForestClassifier(n\_estimators=200, max\_features='sqrt', random\_state=42)

rf\_model\_custom.fit(X\_train, y\_train)

# Make predictions with the Random Forest (custom)

y\_pred\_rf\_custom = rf\_model\_custom.predict(X\_test)

# Evaluate the Random Forest (custom)

accuracy\_rf\_custom = accuracy\_score(y\_test, y\_pred\_rf\_custom)

print(f"Random Forest (Custom) Accuracy: {accuracy\_rf\_custom:.4f}")

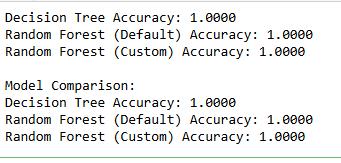
# Step 4: Compare the performance of the models

print("\nModel Comparison:")

print(f"Decision Tree Accuracy: {accuracy\_dt:.4f}")

print(f"Random Forest (Default) Accuracy: {accuracy\_rf\_default:.4f}")

print(f"Random Forest (Custom) Accuracy: {accuracy\_rf\_custom:.4f}")



from sklearn.model\_selection import cross\_val\_score

# Decision Tree with cross-validation

cv\_scores\_dt = cross\_val\_score(dt\_model, X, y, cv=5) # 5-fold cross-validation

print(f"Decision Tree Cross-Validation Scores: {cv\_scores\_dt}")

print(f"Mean Cross-Validation Score (Decision Tree): {cv\_scores\_dt.mean():.4f}")

# Random Forest (default) with cross-validation

cv\_scores\_rf\_default = cross\_val\_score(rf\_model\_default, X, y, cv=5)

print(f"Random Forest (Default) Cross-Validation Scores: {cv\_scores\_rf\_default}")

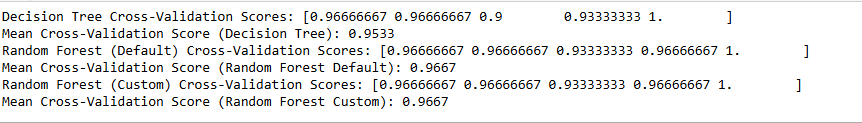
print(f"Mean Cross-Validation Score (Random Forest Default): {cv\_scores\_rf\_default.mean():.4f}")

# Random Forest (custom) with cross-validation

cv\_scores\_rf\_custom = cross\_val\_score(rf\_model\_custom, X, y, cv=5)

print(f"Random Forest (Custom) Cross-Validation Scores: {cv\_scores\_rf\_custom}")

print(f"Mean Cross-Validation Score (Random Forest Custom): {cv\_scores\_rf\_custom.mean():.4f}")



import matplotlib.pyplot as plt

from sklearn.tree import plot\_tree

import numpy as np

# Column names (replace these with actual names in your dataset)

feature\_names = ['sepal\_length', 'sepal\_width', 'petal\_length', 'petal\_width']

# Convert class labels to strings

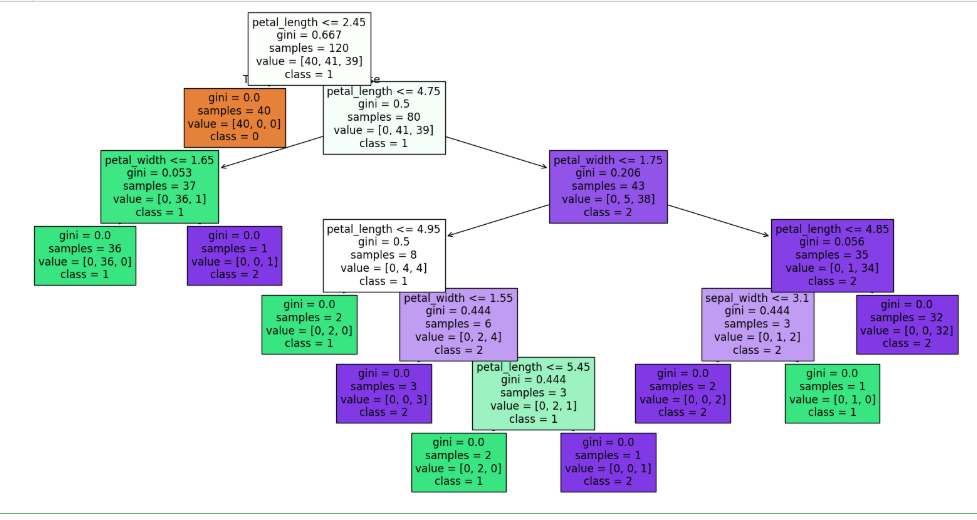
class\_names = [str(cls) for cls in np.unique(y\_train)]

# Plot the decision tree

plt.figure(figsize=(20, 10))

plot\_tree(dt\_model, filled=True, feature\_names=feature\_names, class\_names=class\_names)

plt.show()



import pandas as pd

# If X is not a DataFrame, convert it to one and define the column names

X = pd.DataFrame(X, columns=['sepal\_length', 'sepal\_width', 'petal\_length', 'petal\_width'])

# Get feature importances from the trained Random Forest model

feature\_importance = pd.Series(rf\_model.feature\_importances\_, index=X.columns)

# Sort the feature importances in descending order

feature\_importance = feature\_importance.sort\_values(ascending=False)

# Display the feature importances

print(feature\_importance)

import matplotlib.pyplot as plt

# Plot the feature importances

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

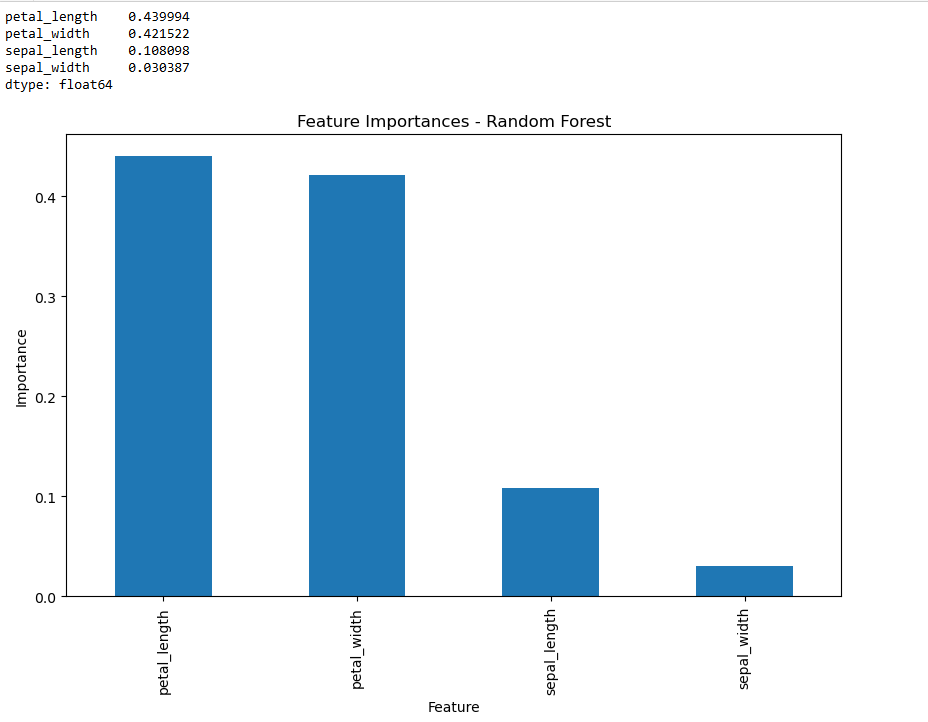
feature\_importance.plot(kind='bar')

plt.title('Feature Importances - Random Forest')

plt.xlabel('Feature')

plt.ylabel('Importance')

plt.show()

****

**PRACTICAL NO : 5**

**Generative Models**

**Aim:[A] Implement and demonstrate the working of a Naive Bayesian classifier using a sample data set. Build the model to classify a test sample.**

**CODE:**

# Import necessary libraries

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score, classification\_report

# Load the dataset from a CSV file

df = pd.read\_csv("iris.csv") # Ensure the file is in the same directory or provide the correct path

# Separate features and target

X = df[['sepal\_length', 'sepal\_width', 'petal\_length', 'petal\_width']] # Feature columns

y = df['species'] # Target column (species)

# Encode the target column into integers

from sklearn.preprocessing import LabelEncoder

label\_encoder = LabelEncoder()

y = label\_encoder.fit\_transform(y)

# Split 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)

# Create a Naive Bayes classifier (GaussianNB)

nb\_model = GaussianNB()

# Train the model

nb\_model.fit(X\_train, y\_train)

# Test the model on test data

y\_pred = nb\_model.predict(X\_test)

# Evaluate the model

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy: {accuracy:.4f}")

print("Classification Report:")

print(classification\_report(y\_test, y\_pred, target\_names=label\_encoder.classes\_))

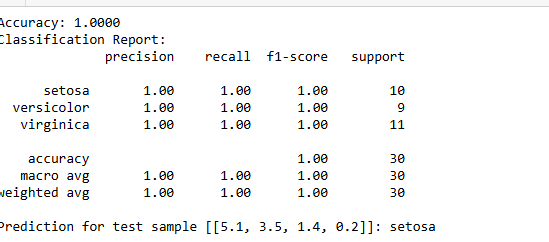
# Classify a new test sample

test\_sample = [[5.1, 3.5, 1.4, 0.2]] # Example sample resembling Setosa

prediction = nb\_model.predict(test\_sample)

predicted\_class = label\_encoder.inverse\_transform(prediction)[0]

print(f"Prediction for test sample {test\_sample}: {predicted\_class}")



from sklearn.metrics import confusion\_matrix

import seaborn as sns

import matplotlib.pyplot as plt

cm = confusion\_matrix(y\_test, y\_pred)

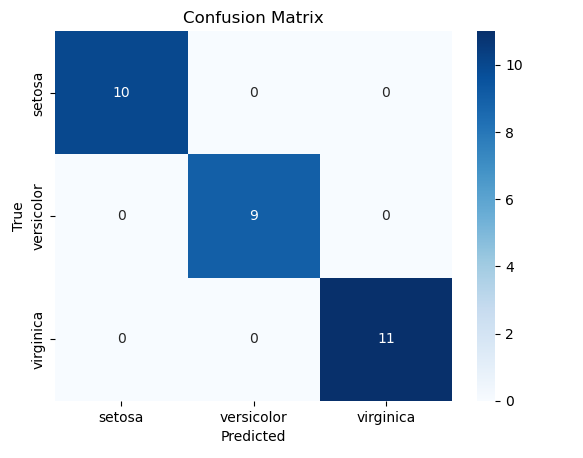
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=label\_encoder.classes\_, yticklabels=label\_encoder.classes\_)

plt.xlabel('Predicted')

plt.ylabel('True')

plt.title('Confusion Matrix')

plt.show()



from sklearn.model\_selection import cross\_val\_score

scores = cross\_val\_score(nb\_model, X, y, cv=5) # 5-fold cross-validation

print(f"Cross-validation scores: {scores}")



**Aim:[B] Implement Hidden Markov Models using hmmlearn.**

**CODE:**

pip install hmmlearn

import numpy as np

from hmmlearn import hmm

import matplotlib.pyplot as plt

# Step 1: Generate synthetic data for the example

n\_samples = 1000

n\_features = 1 # univariate data (1 feature per observation)

# Create the model with 3 hidden states

model = hmm.GaussianHMM(n\_components=3, covariance\_type="diag", n\_iter=1000)

# You need to first fit the model to some data before calling sample().

# You can use the same data to fit and then sample, or just fit a simple dataset.

X = np.random.randn(n\_samples, n\_features) # Use random data to fit (or use your own dataset)

# Step 2: Fit the model to the data

model.fit(X) # Fit the model first!

# Step 3: Now generate synthetic data using the model

# This can only be done after fitting the model

X\_generated, Z\_generated = model.sample(n\_samples)

# Step 4: Predict the hidden states for each data point in the original data

hidden\_states = model.predict(X) # Use predict to get the hidden states

# Step 5: Print the first few hidden states

print("First 10 hidden states:", hidden\_states[:10])

# Step 6: Plot the observed data and hidden states

plt.figure(figsize=(15, 8))

# Plot the observations

plt.subplot(2, 1, 1)

plt.plot(X, label="Observed Data")

plt.title("Observed Data")

plt.xlabel("Time Steps")

plt.ylabel("Observation Value")

# Plot the hidden states

plt.subplot(2, 1, 2)

plt.plot(hidden\_states, label="Hidden States", color='red')

plt.title("Hidden States")

plt.xlabel("Time Steps")

plt.ylabel("Hidden State")

plt.tight\_layout()

plt.show()

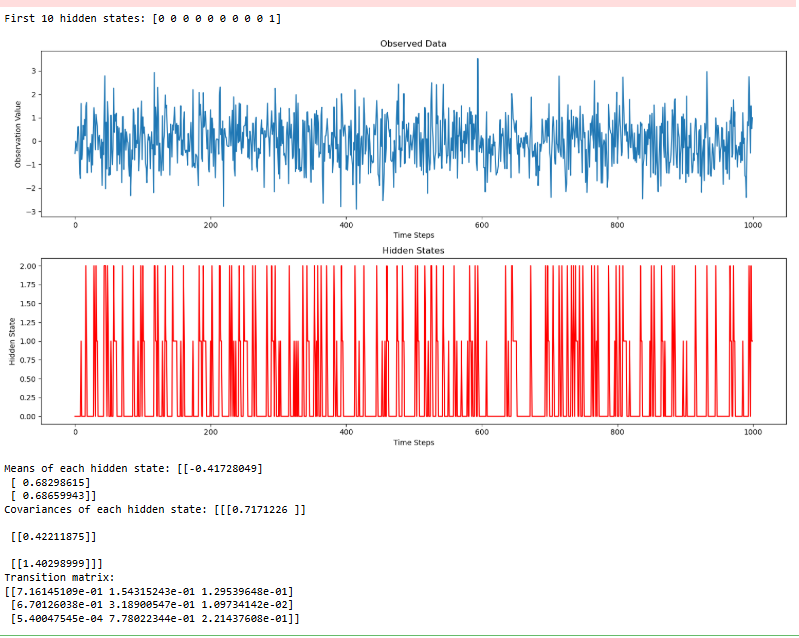
# Step 7: Print the model's learned parameters

print("Means of each hidden state:", model.means\_)

print("Covariances of each hidden state:", model.covars\_)

print("Transition matrix:")

print(model.transmat\_)



**PRACTICAL NO : 6**

**Probabilistic Models**

**Aim:[A] Implement Bayesian Linear Regression to explore prior and posterior distribution.**

**CODE:**

pip install pymc numpy scipy matplotlib seaborn arviz

import numpy as np

import matplotlib.pyplot as plt

import pymc as pm # Updated for PyMC v5

import seaborn as sns

import arviz as az # For better visualization of posterior distributions

from sklearn.linear\_model import BayesianRidge

from sklearn.preprocessing import StandardScaler

# Set random seed for reproducibility

np.random.seed(42)

# Generate synthetic data

N = 50 # Number of data points

X = np.linspace(0, 10, N)[:, None] # Feature values

true\_slope = 2.5

true\_intercept = 1.0

noise = np.random.normal(0, 2, size=N) # Add Gaussian noise

y = true\_slope \* X.flatten() + true\_intercept + noise # Linear relationship

# Bayesian Linear Regression with PyMC

with pm.Model() as model:

# Define priors for slope and intercept

alpha = pm.Normal("alpha", mu=0, sigma=10) # Intercept prior

beta = pm.Normal("beta", mu=0, sigma=10) # Slope prior

sigma = pm.HalfNormal("sigma", sigma=5) # Noise (observation standard deviation)

# Define the likelihood (observed data)

mu = alpha + beta \* X.flatten()

y\_obs = pm.Normal("y\_obs", mu=mu, sigma=sigma, observed=y)

# Sample from the posterior using Markov Chain Monte Carlo (MCMC)

trace = pm.sample(2000, return\_inferencedata=True, progressbar=True)

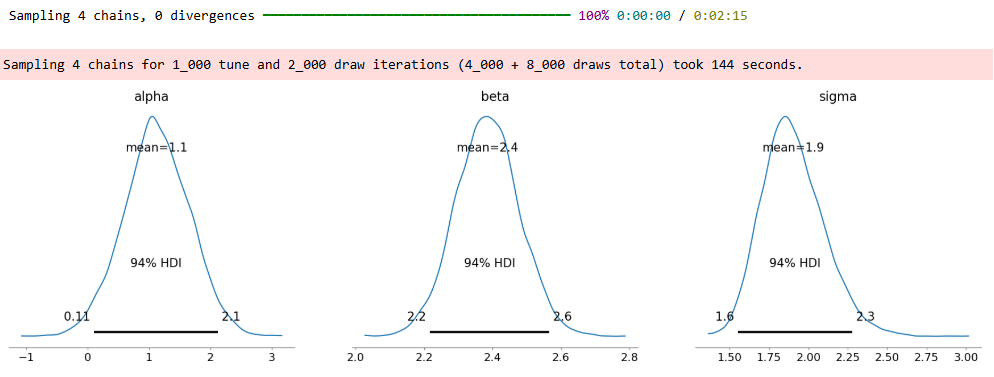
# Summary of posterior distributions

az.summary(trace)

# Plot posterior distributions

az.plot\_posterior(trace, var\_names=["alpha", "beta", "sigma"])

plt.show()



# Plot posterior distributions of alpha (intercept) and beta (slope)

sns.kdeplot(trace.posterior["alpha"].values.flatten(), label="Posterior of alpha (Intercept)", shade=True)

sns.kdeplot(trace.posterior["beta"].values.flatten(), label="Posterior of beta (Slope)", shade=True)

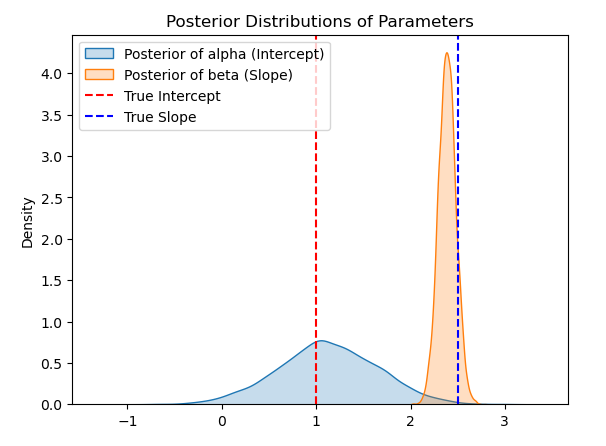
plt.axvline(true\_intercept, color="red", linestyle="--", label="True Intercept")

plt.axvline(true\_slope, color="blue", linestyle="--", label="True Slope")

plt.legend()

plt.title("Posterior Distributions of Parameters")

plt.show()



# Standardize features

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Train Bayesian Ridge regression

bayesian\_ridge = BayesianRidge()

bayesian\_ridge.fit(X\_scaled, y)

# Display learned parameters

print(f"Sklearn Bayesian Ridge: Intercept = {bayesian\_ridge.intercept\_:.3f}, Slope = {bayesian\_ridge.coef\_[0]:.3f}")



**Aim:[B] Implement Gaussian Mixture Models for density estimation and unsupervised clustering.**

**CODE:**

pip install numpy matplotlib seaborn scikit-learn

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.mixture import GaussianMixture

from sklearn.datasets import make\_blobs

# Set random seed for reproducibility

np.random.seed(42)

# Generate synthetic data (clusters)

X, \_ = make\_blobs(n\_samples=300, centers=3, cluster\_std=1.0, random\_state=42)

# Fit a Gaussian Mixture Model (GMM)

n\_components = 3 # Number of clusters

gmm = GaussianMixture(n\_components=n\_components, covariance\_type='full', random\_state=42)

gmm.fit(X)

# Predict cluster labels

labels = gmm.predict(X)

# Plot clusters

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

sns.scatterplot(x=X[:, 0], y=X[:, 1], hue=labels, palette="viridis", s=50, edgecolor='k')

plt.scatter(gmm.means\_[:, 0], gmm.means\_[:, 1], c='red', marker='x', s=200, label='Centroids') # Cluster centers

plt.title("Gaussian Mixture Model Clustering")

plt.legend()

plt.show()

# Density Estimation (Plot Probability Contours)

x, y = np.meshgrid(np.linspace(X[:, 0].min(), X[:, 0].max(), 100),

np.linspace(X[:, 1].min(), X[:, 1].max(), 100))

X\_grid = np.array([x.ravel(), y.ravel()]).T

log\_probs = gmm.score\_samples(X\_grid).reshape(100, 100)

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

plt.contourf(x, y, np.exp(log\_probs), levels=30, cmap="Blues")

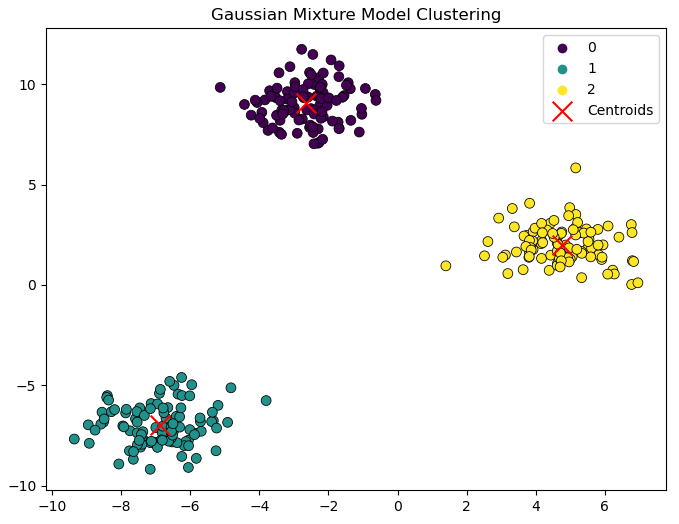
sns.scatterplot(x=X[:, 0], y=X[:, 1], hue=labels, palette="viridis", s=50, edgecolor='k')

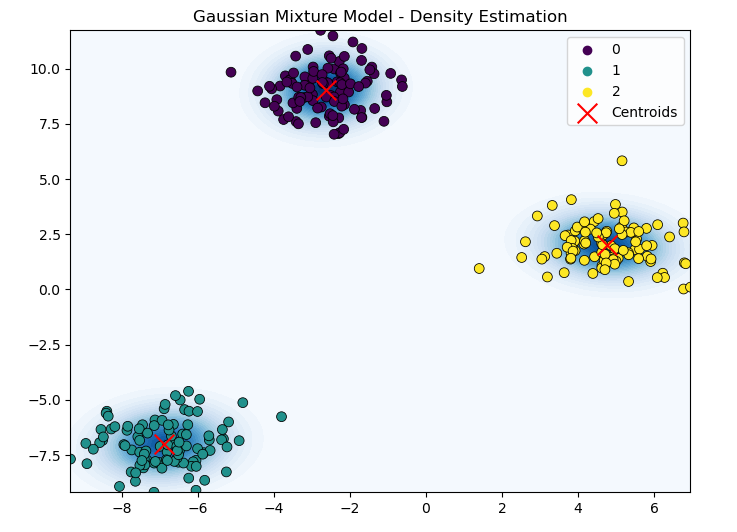
plt.scatter(gmm.means\_[:, 0], gmm.means\_[:, 1], c='red', marker='x', s=200, label='Centroids')

plt.title("Gaussian Mixture Model - Density Estimation")

plt.legend()

plt.show()





**PRACTICAL NO : 7**

**Model Evaluation and Hyperparameter Tuning**

**Aim:[A] Implement cross-validation techniques (k-fold, stratified, etc.) for robust model evaluation.**

**CODE:**