**MANAV RACHNA UNIVERSITY**

**SUPERVISED LEARNING**

Submitted by:



Rashi Bhati

2K23CSUN01270

A

AIML 3B

Subject code : CSH212B-T CCSHCSH212B-T

Faculty Name: Dr. ROSHI SAXENA

Designation: Manager - Xebia Academy

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2     |  | | --- | | **INDEX** |      |  |  |  | | --- | --- | --- | | S. No. | Name of the Program | Date | | 1. | Write a python code to demonstrate commands for numpy and pandas. |  | | 2. | Write a python program to calculate mean square and mean absolute error. |  | | 3. | Write a python program to calculate gradient descent of a machine learning model. |  | | 4. | Prepare a linear regression model for predicting the salary of user based on number of years of experience. |  | | 5. | Prepare a linear regression model for prediction of resale car price. |  | | 6. | Prepare a Lasso and Ridge regression model for prediction of house price and compare it with linear regression model. |  | | 7. | Prepare a decision tree model for Iris Dataset using Gini Index. |  | | 8. | Prepare a decision tree model for Iris Dataset using entropy. |  | | 9. | Prepare a naïve bayes classification model for prediction of purchase power of a user. |  | | 10. | Prepare a naïve bayes classification model for classification of email messages into spam or not spam. |  | | 11. | Prepare a model for prediction of prostate cancer using KNN Classifier. |  | | 12. | Prepare a model for prediction of survival from Titanic Ship using Random Forest and compare the accuracy with other classifiers also. |  | |

#  Program 1

Write a python code to demonstrate commands for numpy and pandas.

# Demonstrate numpy commands # Import necessary libraries import numpy as np

# Creating arrays with zeros a = np.zeros(3) # 1D array of zeros print("Array a:", a) print("Type of array a:", type(a)) print("Type of elements in array a:", type(a[0]))

b = np.zeros(3, dtype=int) # 1D array of zeros with integer type print("Array b:", b) print("Type of array b:", type(b)) print("Type of elements in array b:", type(b[0]))

# Reshape example z = np.zeros(3) print("Original Array: ", z) print("Shape of Array: ", z.shape)

z.shape = (3, 1) # Reshape array to 5x1 print("Reshaped Array:\n", z) print("Shape of Reshaped Array: ", z.shape)

# Creating an array using linspace z = np.linspace(1, 2, 5) print("Array created using linspace: ", z)

# Accessing array elements with positive and negative indexing print("Element at index 0: ", z[0]) print("Element at index -3: ", z[-3]) print("Array elements from index 0 to 2: ", z[0:2])

# Identity matrix i = np.identity(2, dtype=int) print("Identity Matrix:\n", i)

# Creating a 2D matrix in two different ways z = np.zeros((2, 2)) # 2D array of zeros print("2-D Array (method 1):\n", z)

y = np.array([[1, 2], [3, 4]]) # Manually defined 2D array print("2-D Array (method 2):\n", y)

# Accessing elements with index print("Element at (0,1): ", y[0, 1]) print("Element at (0,0): ", y[0, 0])

# Slicing in 2D arrays print("Second row: ", y[1, :]) print("First column: ", y[:, 0])

H = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) print("2-D Array:\n", H) print("First row:", H[0, :]) print("Third row:", H[2, :]) print("First column across rows: ", H[:, 0])

# Access elements at specified indices x = np.linspace(2, 4, 5) indices = np.array((0, 2, 3)) print("Array x:", x) print("Elements at specified indices(0,2,3): ", x[indices])

# Boolean array d = np.array([0, 1, 2, 0, 0], dtype=bool) # Every non-zero is True, 0 is False print("Boolean Array d:", d)

# Sorting and basic array statistics a = np.array([17, 11, 15, 19, 24, 28, 26, 37, 35, 40])

a.sort() print("Original Array:", a) print("Sorted Array:", a) print("Sum:", a.sum()) print("Min:", a.min()) print("Max:", a.max()) print("Argmin (index of min):", a.argmin()) print("Argmax (index of max):", a.argmax()) print("Cumulative Sum:", a.cumsum()) print("Cumulative Product:", a.cumprod()) print("Mean:", a.mean()) print("Median:", np.median(a)) print("Variance:", a.var()) print("Standard Deviation:", a.std()) print("Searchsorted (insert position for 25):", a.searchsorted(25))

# Array arithmetic operations a = np.array([1, 2, 3, 4]) b = np.array([5, 6, 7, 8]) print("a + b:", a + b) print("a \* b:", a \* b) print("a + 10:", a + 10) print("a \* 10:", a \* 10)

# Matrix operations

X = np.array([[1, 2, 3], [4, 5, 6], [5, 6, 7]]) Y = np.array([[7, 8, 9], [4, 8, 9], [6, 3, 5]]) print("X:\n", X) print("Y:\n", Y) print("X + Y:\n", X + Y) print("X + 10:\n", X + 10) print("X \* Y:\n", X @ Y) # Matrix multiplication print("Transpose of X:\n", X.T)

# Comparison and modifying elements

Z = np.array([2, 3]) X = np.array([2, 3]) print("X == Z:", X == Z) X[0] = 5

print("X == Z after modifying X:", X == Z)

Show hidden output

# Impoer neccessary libraries from pandas import DataFrame, Series # Import Series and DataFrame for convenience import pandas as pd import numpy as np

# Creating a Series with default index ser\_1 = Series([1, 1, 2, -3, -5, 8, 13]) print("Series with default index:\n", ser\_1) print("Values in series: ", ser\_1.values) # Display only the values of the series

# Creating a Series with a custom index ser\_2 = Series([1, 1, 2, -3, -5], index=['a', 'b', 'c', 'd', 'e']) print("Series 2:\n", ser\_2)

# Accessing elements in a Series using index and labels print("ser\_2[1] == ser\_2[b]", ser\_2[1] == ser\_2["b"]) print(ser\_2[['c', 'a', 'b']]) # Filter Series for values greater than 0 ser\_2[ser\_2 > 0]

# Apply an operation on Series elements ser\_2 \* 2 np.exp(ser\_2)

# Create a Series from a dictionary dict\_1 = {'foo': 100, 'bar': 200, 'baz': 300} ser\_3 = Series(dict\_1) # Custom index on Series index = ['foo', 'bar', 'baz', 'qux'] ser\_4 = Series(dict\_1, index=index) # Missing values become NaN

# Print Series print("Series 3:\n", ser\_3) print("Series 4:\n", ser\_4) # Check for null values in Series

print("Null values in ser\_4:\n", pd.isnull(ser\_4)) # Arithmetic operations between Series print("Sum of series 3 and 4:\n", ser\_3 + ser\_4) # Setting names for the Series and index ser\_4.name = 'foobarbaz' ser\_4.index.name = 'label' print("Series 4 after setting names for series and index:\n", ser\_4)

# Create another Series with custom index ser = Series([10, 15, 18, 12, 20, 9], index=[5, 8, 12, 0, 1, 7]) # Access elements by label or position using loc and iloc print("Accessing elements by label or position: ") print(ser.loc[0:1]) print(ser.iloc[0:1]) print(ser.iloc[0]) print(ser.loc[0])

# Create a DataFrame with dictionaries data\_1 = {'state': ['VA', 'VA', 'VA', 'MD', 'MD'], 'year': [2012, 2013, 2014, 2015, 2016], 'pop': [5.0, 5.1, 5.2, 4.0, 4.1]} df\_1 = DataFrame(data\_1)

# Access a column of the DataFrame df\_1['state']

# Find and print the series of prime numbers from 1 to 300 primes = [] for i in range(1, 301):

if i > 1:

for j in range(2, i // 2 + 1):

if i % j == 0: break

else:

primes.append(i) primes\_series = pd.Series(primes) print("Series of Primes:\n", primes\_series)

# Generate Fibonacci numbers up to 100 a, b = 0, 1 fibonacci\_nums = [] while a < 100:

fibonacci\_nums.append(a) a, b = b, a + b fibonacci\_series = Series(fibonacci\_nums) print("Fibonacci Series:\n", fibonacci\_series)

# Prompt user for a list of 20 numbers l = [int(x) for x in input("Enter 20 numbers: ").split()]

# Initialize min, max, and sum variables min\_val = l[0] max\_val = l[0] sum\_val = 0

# Calculate sum, min, and max manually for i in l:

sum\_val += i if i < min\_val:

min\_val = i

if i > max\_val: max\_val = i

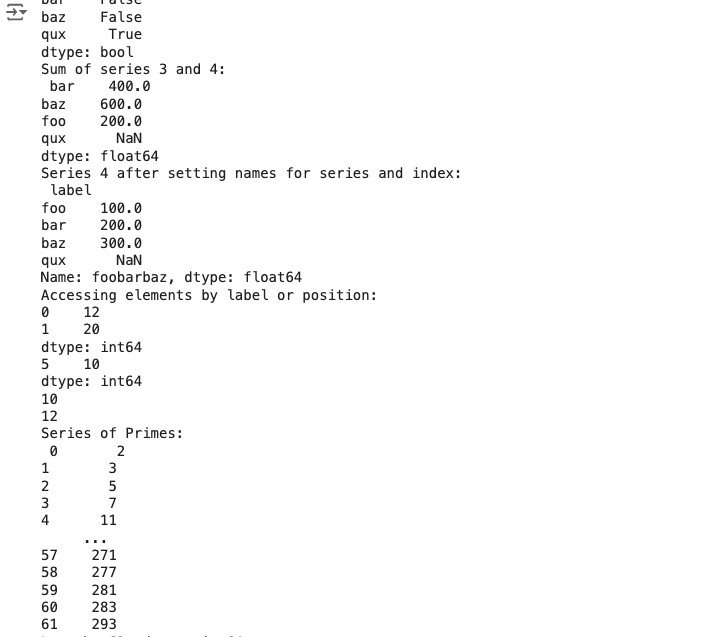
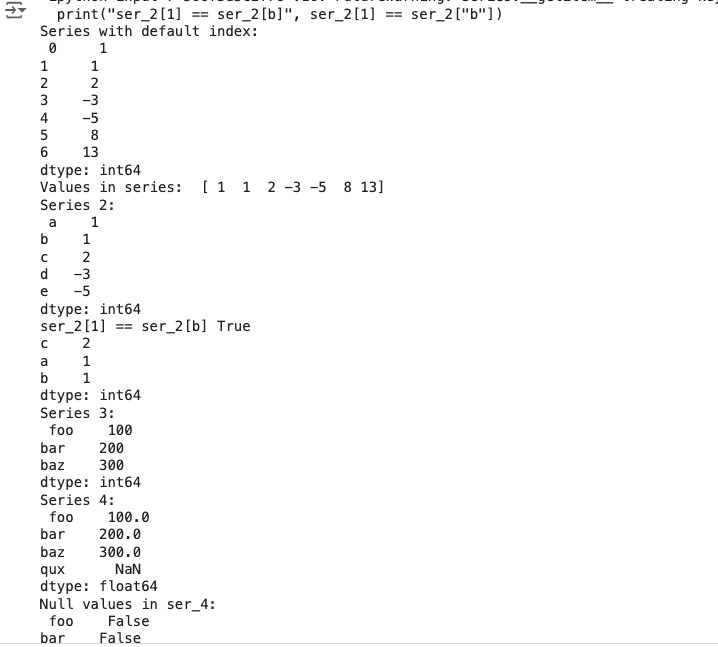
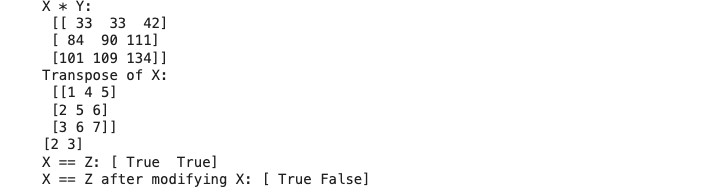
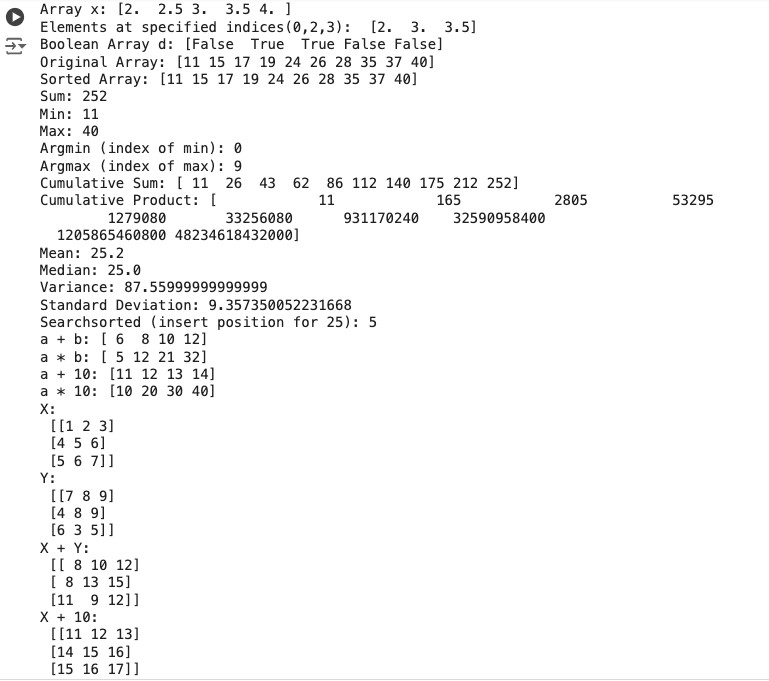
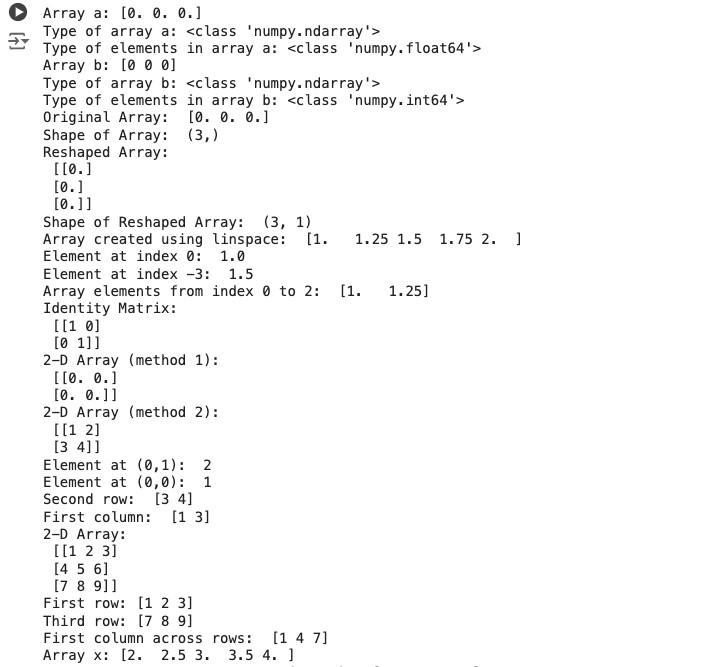
print("Sum:", sum\_val) print("Min:", min\_val) print("Max:", max\_val)

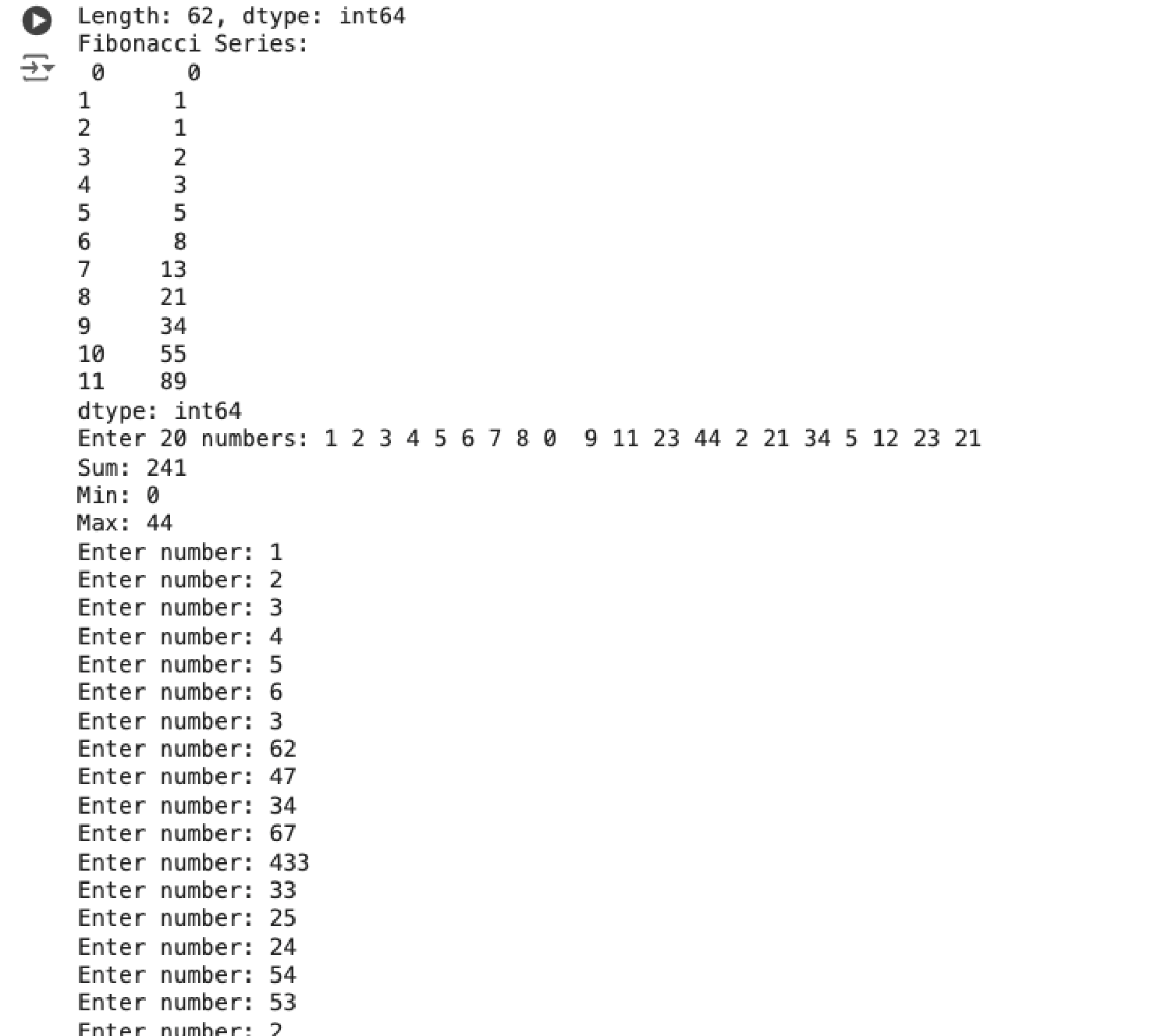
# Manually inputing values in a list one by one and finding the sum l = [] sum\_val = 0 for i in range(1, 21):

num = int(input("Enter number: "))

l.append(num) sum\_val += num print("Sum:", sum\_val)

Show hidden output





Write a python program to calculate mean absolute error and mean square error.

#function to calculate the predicted values def predicted\_output(x,w,b):

y\_hat=[]

for i in range(len(x)):

y\_hat.append(w\*x[i]+b)

return y\_hat

#function to calculate mean absolute error def MAE(y, y\_hat):

sum=0

for i in range(len(y)):

sum+=abs(y\_hat[i]-y[i])

return sum/len(y)

#function to calculate mean square error def MSE(y, y\_hat):

sq\_sum=0

for i in range(len(y)):

sq\_sum+=(y\_hat[i]-y[i])\*\*2

return sq\_sum/len(y)

#taking inputs x=[eval(x) for x in input("Enter the values of x(input) separated by ',': ").split(",")] y=[eval(x) for x in input("Enter the values of y(output) separated by ',': ").split(",")] w=eval(input("Enter the value of w: ")) b=eval(input("Enter the value of b: "))

#calling functions

y\_hat=predicted\_output(x, w, b) MAE\_value=MAE(y, y\_hat)

MSE\_value=MSE(y, y\_hat) #printing the values print("Predicted Output: ",y\_hat) print("Mean Absolute Error: ",MAE\_value) print("Mean Square Error: ",MSE\_value)

Enter the values of x(input) separated by ',': 3, 6, 9, 12, 15, 18, 20

Enter the values of y(output) separated by ',': 15, 28, 63, 90, 120, 152, 190

Enter the value of w: 2.5

Enter the value of b: 0

Predicted Output: [7.5, 15.0, 22.5, 30.0, 37.5, 45.0, 50.0]

Mean Absolute Error: 64.35714285714286

Mean Square Error: 6188.678571428572



Write a python program to calculate gradient descent of a machine learning model.

# Import neccessary libraries import numpy as np import matplotlib.pyplot as plt

# Function to perform gradient descent def gradient\_descent(func, x, learning\_rate, num\_iterations):

x\_values=[]

for i in range(num\_iterations):

gradient=func(x)

x\_values.append(x)

x-=(learning\_rate\*gradient)

return x,x\_values

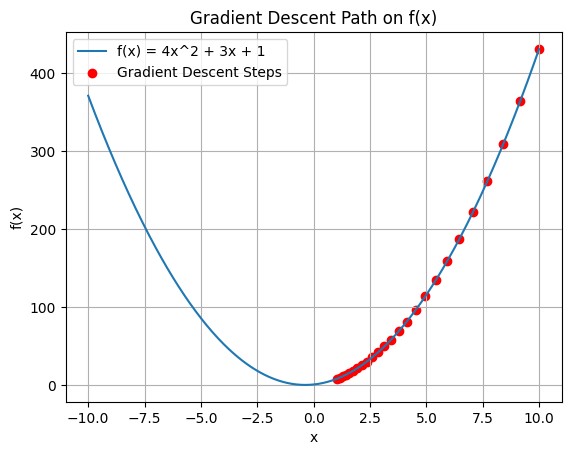
# Define the original function def function(x):

return 4\*x\*\*2+3\*x+1

# Define the derivative of the function def derivative\_f(x): return 8\*x+3

For function: 4x^2+3x+1:

Minimum value of x: 0.9152794755738098



# Plotting the gradient descent steps on the function curve def plot\_gradient\_descent(func, x, learning\_rate, num\_iterations, x\_values):

x\_range = np.linspace(-10, 10, 400)

y\_range = func(x\_range)

plt.plot(x\_range, y\_range, label="f(x) = 4x^2 + 3x + 1")

plt.scatter(x\_values, [func(x) for x in x\_values], color='red', label="Gradient Descent Steps")

plt.xlabel("x")

plt.ylabel("f(x)")

plt.legend()

plt.grid(True)

plt.title("Gradient Descent Path on f(x)")

plt.show()

# Set parameters for gradient descent initial\_x=10 learning\_rate=0.01 num\_iterations=25 # Perform gradient descent min\_x, x\_values=gradient\_descent(derivative\_f, initial\_x, learning\_rate, num\_iterations)

# Print results print("For function: 4x^2+3x+1: ") print("Minimum value of x:", min\_x)

# Call the plot function to visualize gradient descent plot\_gradient\_descent(function, x, learning\_rate, num\_iterations, x\_values)

Prepare a linear regression model for predicting the salary of user based on number of years of experience.

# importing neccessary libraries import numpy as np import pandas as pd import matplotlib.pyplot as plt import seaborn as sns

# loading the dataset df = pd.read\_csv('Salary\_Data.csv')

# defining the feature variable 'x' by dropping Salary and target variable 'y' as the Salary column x = df.drop('Salary', axis=1) y = df['Salary']

# split the dataset into training and testing sets from sklearn.model\_selection import train\_test\_split x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.3, random\_state=1)

# initialize and train the Linear Regression model on the training data from sklearn.linear\_model import LinearRegression model = LinearRegression() model.fit(x\_train, y\_train)

# Predict the target variable for the test set y\_test\_predict = model.predict(x\_test)

# Display the model's coefficient and intercept print("Model coefficient(s):", model.coef\_) print("Model intercept:", model.intercept\_) print("Model R^2 score on test set:", model.score(x\_test, y\_test))

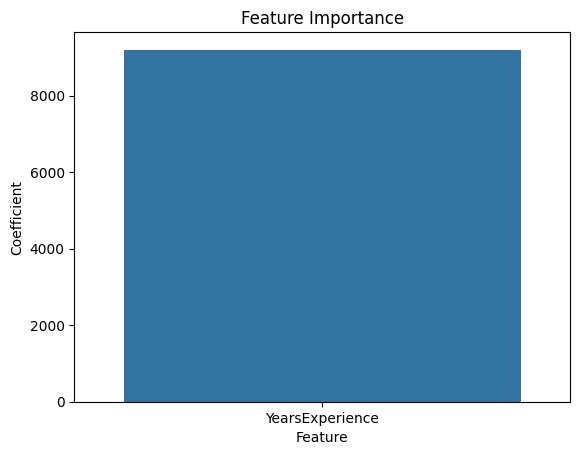
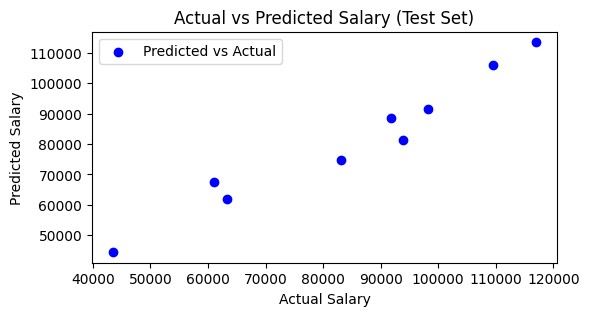
# scatter plot to visualize the relationship between predicted and actual values in the test set plt.figure(figsize=(6, 3)) plt.scatter(y\_test, y\_test\_predict, color='blue', label="Predicted vs Actual") plt.xlabel("Actual Salary") plt.ylabel("Predicted Salary") plt.title("Actual vs Predicted Salary (Test Set)") plt.legend() plt.show()

# bar plot to display the importance of each feature based on model coefficients imp=pd.DataFrame(list(zip(x\_test.columns,np.abs(model.coef\_))),columns=['Feature','Coefficient']) sns.barplot(x='Feature', y='Coefficient', data=imp) plt.title("Feature Importance") plt.show()

Model coefficient(s): [9202.23359825]

Model intercept: 26049.577715443353

Model R^2 score on test set: 0.9248580247217075



# Program 5

Prepare a linear regression model for prediction of resale car price.

# import necessary libraries import numpy as np import pandas as pd import matplotlib.pyplot as plt import seaborn as sns

# load the dataset df = pd.read\_csv('cars24-car-price-cleaned.csv')

# replace 'make' and 'model' columns with the mean selling price for each group df['make'] = df.groupby('make')['selling\_price'].transform('mean') df['model'] = df.groupby('model')['selling\_price'].transform('mean')

# normalize the dataset using MinMaxScaler to scale features between 0 and 1 from sklearn.preprocessing import MinMaxScaler scaler = MinMaxScaler() df\_normalized = pd.DataFrame(scaler.fit\_transform(df), columns=df.columns)

# define target variable 'y' as the selling price and features 'x' by dropping the selling price y = df\_normalized['selling\_price'] x = df\_normalized.drop('selling\_price', axis=1)

# split the dataset into training and testing sets from sklearn.model\_selection import train\_test\_split x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.3, random\_state=1)

# initialize and train the Linear Regression model on the training data from sklearn.linear\_model import LinearRegression model = LinearRegression() model.fit(x\_train, y\_train)

# predict the target variable for the test set y\_test\_predict = model.predict(x\_test)

# Display model's coefficient, intercept, and R^2 score on test set print("Model coefficients:", model.coef\_) print("Model intercept:", model.intercept\_) print("Model R^2 score on test set:", model.score(x\_test, y\_test))

# Scatter plot to visualize the relationship between predicted and actual values in the test set #plt.figure(figsize=(8, 6)) plt.scatter(y\_test, y\_test\_predict, label="Predicted vs Actual") plt.xlabel("Actual Selling Price (Normalized)") plt.ylabel("Predicted Selling Price (Normalized)") plt.title("Actual vs Predicted Selling Price (Test Set)") plt.legend() plt.show()

# Bar plot to display the importance of each feature based on model coefficients imp = pd.DataFrame(list(zip(x\_test.columns, np.abs(model.coef\_))), columns=['Feature', 'Coefficient'])

#plt.figure(figsize=(8, 6)) sns.barplot(x='Feature', y='Coefficient', data=imp) plt.xticks(rotation=90) plt.title("Feature Importance in Selling Price Prediction") plt.show()

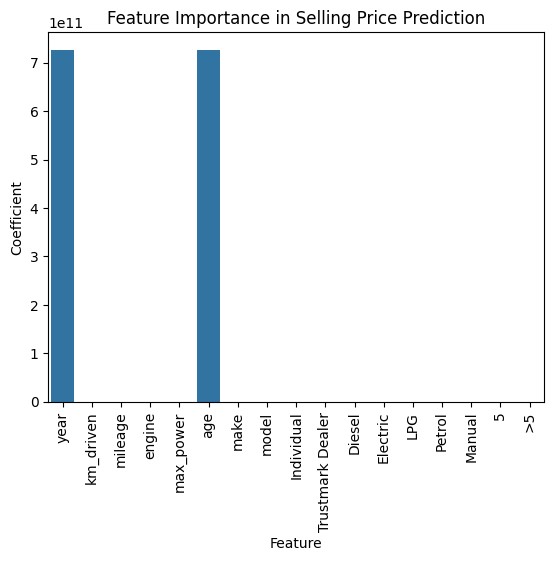
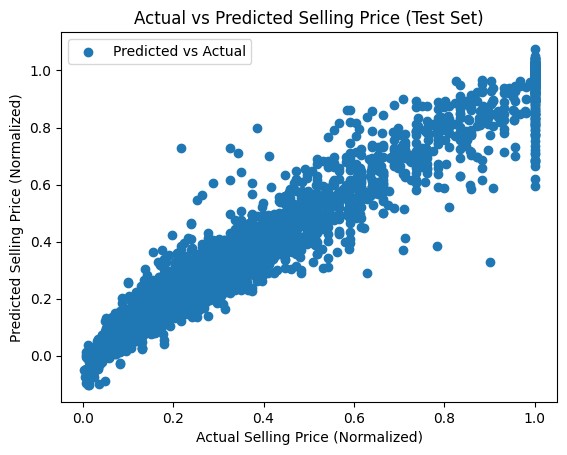
Model coefficients: [ 7.26831852e+11 -2.50610352e-01 -2.32537818e-01 7.38776447e-02

4.70141495e-02 7.26831852e+11 6.62815814e-02 8.59178586e-01 -7.22882618e-03 -7.02099753e-03 7.03528760e-03 1.32983308e-01

1.49877118e-02 -6.86552095e-03 -3.59124005e-03 -1.61993065e-02 -2.35818239e-02]

Model intercept: -726831852169.8219

Model R^2 score on test set: 0.9459835819294395



# Program 6

Prepare a Lasso and Ridge regression model for prediction of house price and compare it with linear regression model.

# Import necessary libraries import numpy as np import pandas as pd import matplotlib.pyplot as plt from sklearn.linear\_model import LinearRegression, Lasso, Ridge from sklearn.model\_selection import train\_test\_split from sklearn.metrics import mean\_squared\_error from sklearn.preprocessing import MinMaxScaler

# Load the housing dataset df = pd.read\_csv('Housing.csv')

# convert categorical variables into numerical features that can be used by the model (target variable encoding) df['mainroad']=df.groupby('mainroad')['price'].transform('mean') df['guestroom']=df.groupby('guestroom')['price'].transform('mean') df['basement']=df.groupby('basement')['price'].transform('mean') df['hotwaterheating']=df.groupby('hotwaterheating')['price'].transform('mean') df['airconditioning']=df.groupby('airconditioning')['price'].transform('mean') df['prefarea']=df.groupby('prefarea')['price'].transform('mean') df['furnishingstatus']=df.groupby('furnishingstatus')['price'].transform('mean')

# Normalize the dataset to bring all features to the same scale scaler = MinMaxScaler() df\_normalized = pd.DataFrame(scaler.fit\_transform(df), columns=df.columns)

# Define the target variable 'y' as 'median\_house\_value' and features 'x' by dropping the target column y = df\_normalized['price'] x = df\_normalized.drop('price', axis=1)

# Split the dataset into training and testing sets x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.3, random\_state=1)

# Initialize models: Linear Regression, Lasso Regression, and Ridge Regression model = LinearRegression() lasso\_model = Lasso(alpha=0.1) ridge\_model = Ridge(alpha=0.1)

# Fit each model to the training data model.fit(x\_train, y\_train) lasso\_model.fit(x\_train, y\_train) ridge\_model.fit(x\_train, y\_train)

# Display model coefficients, intercepts and R^2 scores print("Linear Regression Coefficients:", model.coef\_) print("Lasso Regression Coefficients:", lasso\_model.coef\_) print("Ridge Regression Coefficients:", ridge\_model.coef\_)

print("Linear Regression Intercept:", model.intercept\_) print("Lasso Regression Intercept:", lasso\_model.intercept\_) print("Ridge Regression Intercept:", ridge\_model.intercept\_)

print("Linear Regression R^2 Score (Train):", model.score(x\_train, y\_train)) print("Lasso Regression R^2 Score (Train):", lasso\_model.score(x\_train, y\_train)) print("Ridge Regression R^2 Score (Train):", ridge\_model.score(x\_train, y\_train))

# Predict the target values on the test set using each model y\_pred = model.predict(x\_test) y\_pred\_lasso = lasso\_model.predict(x\_test) y\_pred\_ridge = ridge\_model.predict(x\_test)

# Calculate Mean Squared Error (MSE) for each model on the test set mse = mean\_squared\_error(y\_test, y\_pred) mse\_lasso = mean\_squared\_error(y\_test, y\_pred\_lasso) mse\_ridge = mean\_squared\_error(y\_test, y\_pred\_ridge)

# Display the MSE results to compare model performance, with lower MSE indicating better fit print('MSE without regularization (Linear Regression):', mse) print('MSE with Lasso regularization:', mse\_lasso) print('MSE with Ridge regularization:', mse\_ridge)

# Visualize the comparison of actual vs predicted values for each model plt.figure(figsize=(10, 6)) plt.scatter(y\_test, y\_pred, color='blue', label="Linear Regression Predictions") plt.scatter(y\_test, y\_pred\_lasso, color='green', label="Lasso Regression Predictions") plt.scatter(y\_test, y\_pred\_ridge, color='red', label="Ridge Regression Predictions", marker='\*') plt.xlabel("Actual Price") plt.ylabel("Predicted Price") plt.title("Comparison of Predictions by Different Regression Models") plt.legend() plt.show()

Linear Regression Coefficients: [0.31039697 0.01959006 0.26477477 0.13658528 0.04098972 0.02376751

0.04792801 0.07098812 0.05282266 0.07096655 0.04358941 0.03623753] Lasso Regression Coefficients: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

Ridge Regression Coefficients: [0.30639084 0.02106921 0.26241647 0.13615958 0.04133038 0.02401481

0.04774817 0.07051319 0.0530351 0.0713936 0.04377635 0.03640865] Linear Regression Intercept: -0.0050427725675667445

Lasso Regression Intercept: 0.26192224608287595

Ridge Regression Intercept: -0.0048457449783638196

Linear Regression R^2 Score (Train): 0.6806547764599723

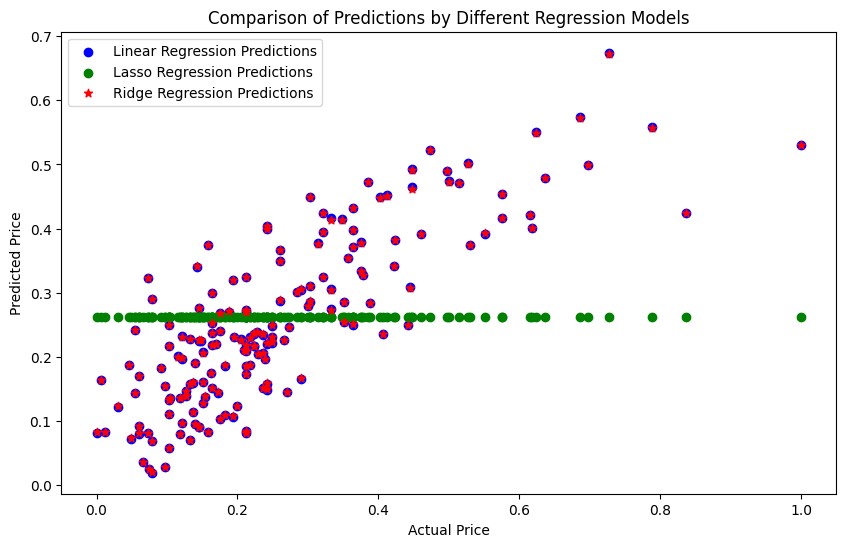
Lasso Regression R^2 Score (Train): 0.0

Ridge Regression R^2 Score (Train): 0.6806349211986238

MSE without regularization (Linear Regression): 0.010274158458096141

MSE with Lasso regularization: 0.03051838551799671

MSE with Ridge regularization: 0.010266744866035897



**Program 7**

Prepare a decision tree model for Iris Dataset using Gini Index.

# Import necessary libraries from sklearn import datasets

from sklearn.tree import DecisionTreeClassifier, plot\_tree from sklearn.metrics import accuracy\_score import matplotlib.pyplot as plt import pandas as pd

# Load the Iris dataset df = pd.read\_csv("Iris.csv")

# Define feature matrix 'x' by dropping 'Species' and 'Id' columns and target variable 'y' as 'Species' x = df.drop(['Species', 'Id'], axis=1) y = df['Species']

# Initialize DecisionTreeClassifier with Gini impurity criterion model = DecisionTreeClassifier(criterion='gini')

# Dictionary to store Gini impurity for each feature gini\_impurities = {}

#loop through each feature for i in range(x.shape[1]):

#fit classifier with only the current feature

model.fit(x.iloc[:, i].values.reshape(-1, 1), y)

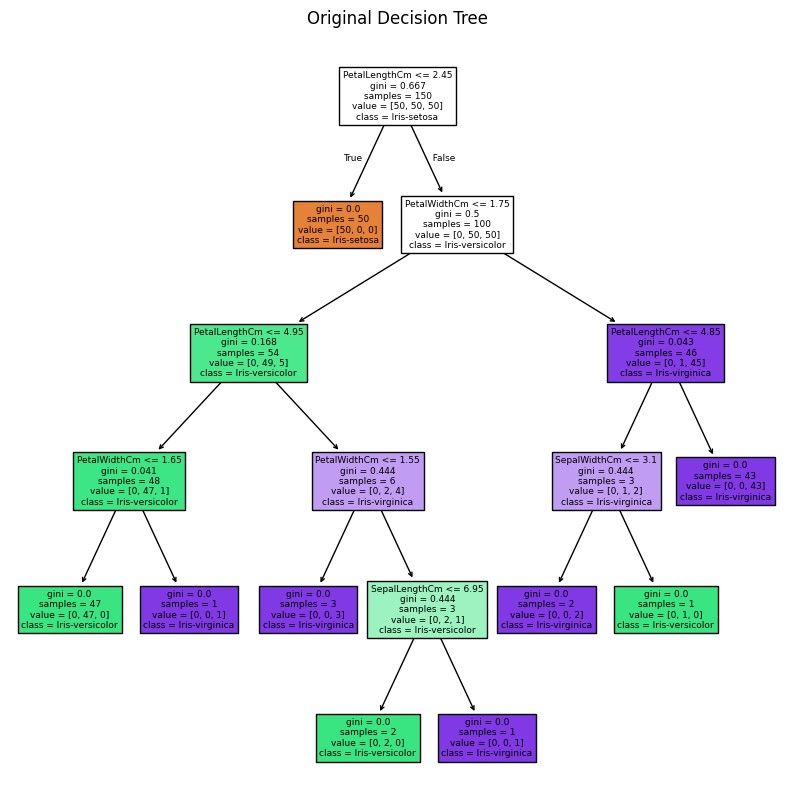
prob=model.predict\_proba(x.iloc[:, i].values.reshape(-1,1))

gini\_impurities[i] = 1 - (prob[:, 0]\*\*2 + prob[:, 1]\*\*2 + prob[:, 2]\*\*2).sum()

# Find the feature with the lowest Gini impurity (best feature) best\_feature = min(gini\_impurities, key=gini\_impurities.get) print(f"Best feature: {x.columns[best\_feature]}") model.fit(x, y)

#plot original tree plt.figure(figsize=(10, 10)) plot\_tree(model, filled=True, feature\_names=x.columns, class\_names=model.classes\_) plt.title("Original Decision Tree")

Best feature: PetalLengthCm



plt.show

(

)

**Program 8**

Prepare a decision tree model for Iris Dataset using entropy.

# Import necessary libraries import numpy as np import pandas as pd from sklearn.metrics import confusion\_matrix, accuracy\_score from sklearn.model\_selection import train\_test\_split from sklearn.tree import DecisionTreeClassifier, plot\_tree import matplotlib.pyplot as plt from sklearn import tree

# Load the Iris dataset df=pd.read\_csv("Iris.csv")

# Define feature matrix 'x' by dropping 'Species' and 'Id' columns and target variable 'y' as 'Species' x=df.drop(["Species", "Id"], axis=1) y=df["Species"]

# Splitting the dataset into train and test x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.3, random\_state=100)

# Build decision tree

model = tree.DecisionTreeClassifier(criterion='entropy', max\_depth=4)

# Fit the tree to iris dataset model.fit(x\_train, y\_train)

# Find the accuracy of the model y\_pred = model.predict(x\_test)

print("Accuracy: ", accuracy\_score(y\_test, y\_pred)\*100)

# Function to plot the decision tree def plot\_decision\_tree(model, feature\_names, class\_names):

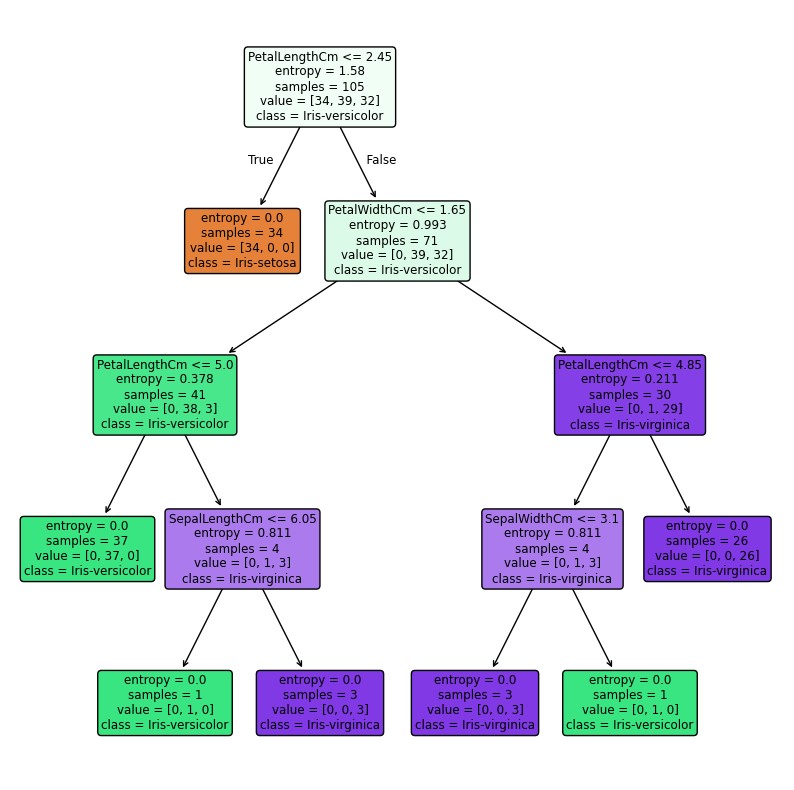
plt.figure(figsize=(10, 10))

plot\_tree(model, filled=True, feature\_names=feature\_names, class\_names=class\_names, rounded=True)

plt.show() plot\_decision\_tree(model, ["SepalLengthCm", "SepalWidthCm", "PetalLengthCm", "PetalWidthCm"],

["Iris-setosa", "Iris-versicolor", "Iris-virginica"])

Accuracy: 95.55555555555556



##  Program 9

Prepare a naïve bayes classi cation model for prediction of purchase power of a user.

# Import libraries import pandas as pd import numpy as np import matplotlib.pyplot as plt from matplotlib.colors import ListedColormap import seaborn as sns from sklearn.preprocessing import LabelEncoder, StandardScaler from sklearn.model\_selection import train\_test\_split from sklearn.naive\_bayes import GaussianNB from sklearn import metrics from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report, precision\_recall\_curve, f1\_score

# Load User\_Data dataset df = pd.read\_csv('User\_Data.csv')

# Drop User ID column as it does not contribute towards prediction purpose df.drop(['User ID'], axis=1, inplace=True)

# Label Encoding le=LabelEncoder() df['Gender']=le.fit\_transform(df['Gender'])

# Split data into dependent/independent variables x = df.iloc[:, :-1].values y = df.iloc[:, -1].values

# Split the dataset into training and testing sets x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.25, random\_state=True)

# Scale dataset sc = StandardScaler() x\_train = sc.fit\_transform(x\_train) x\_test = sc.transform(x\_test)

# Create naive-bayes classifier model classifier=GaussianNB() classifier.fit(x\_train, y\_train)

# Predict the values y\_pred=classifier.predict(x\_test) # Print accuracy of classifier print("Accuracy of classifier: ", accuracy\_score(y\_test, y\_pred))

# Print the classification report print(f'Classification report:\n{classification\_report(y\_test, y\_pred)}')

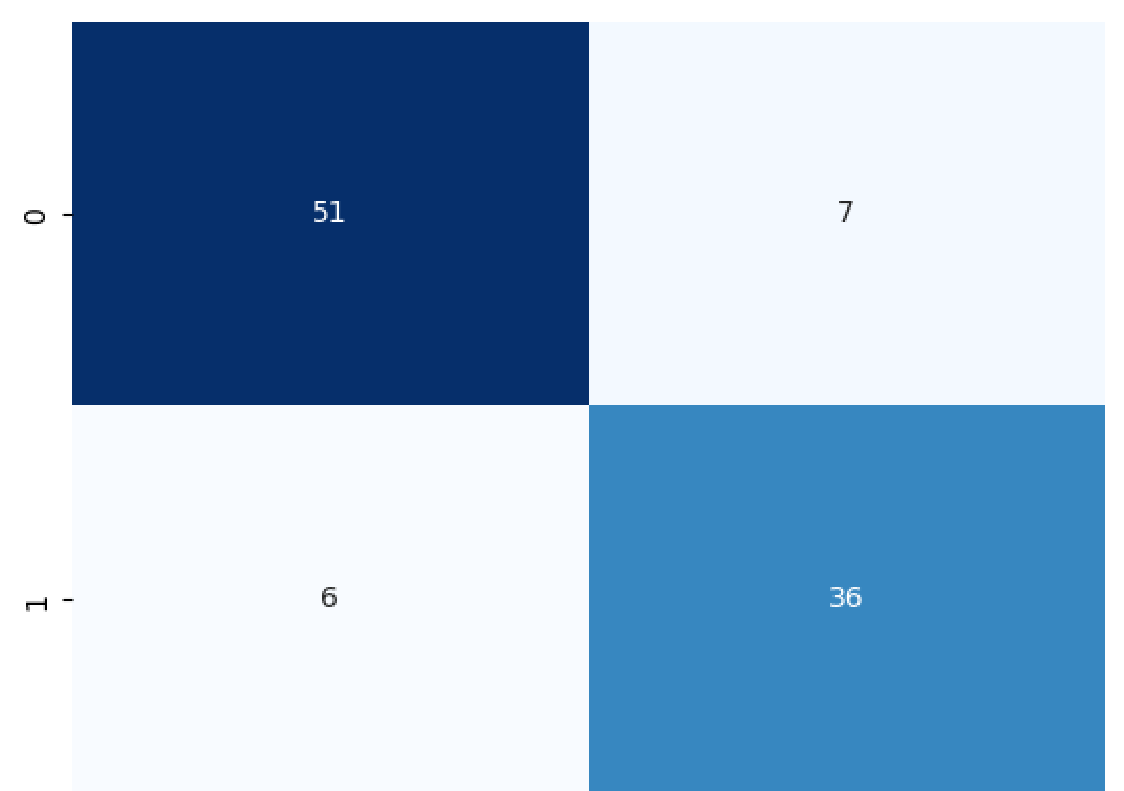
# Print the confusion matrix cf\_matrix=confusion\_matrix(y\_test, y\_pred) sns.heatmap(cf\_matrix, annot=True, fmt='d', cmap='Blues', cbar=False) Accuracy of classifier: 0.87

Classification report: precision recall f1-score support

0 0.89 0.88 0.89 58 1 0.84 0.86 0.85 42

accuracy 0.87 100 macro avg 0.87 0.87 0.87 100 weighted avg 0.87 0.87 0.87 100

<Axes: >



## Program 10

Prepare a naïve bayes classi cation model for classi cation of email messages into spam or not spam.

# Import libraries import pandas as pd from sklearn.model\_selection import train\_test\_split from sklearn.naive\_bayes import MultinomialNB, GaussianNB from sklearn.feature\_extraction.text import CountVectorizer from sklearn.metrics import accuracy\_score, f1\_score import matplotlib.pyplot as plt from wordcloud import WordCloud

# Load the dataset into a DataFrame with 'latin-1' encoding to avoid encoding issues df = pd.read\_csv('spam.csv', encoding='latin-1')

# Select only the relevant columns ('v1' as labels and 'v2' as messages) and rename them df = df[['v1', 'v2']] df = df.rename(columns={'v1': 'label', 'v2': 'text'})

# Define feature matrix 'x' as 'text' and target variable 'y' as 'label' x=df['text'] y=df['label']

# 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)

# Find and plot the distribution of spam and ham messages distribution = y.value\_counts() print("Distribution of spam and ham messages:\n", distribution) distribution.plot(kind='pie', autopct='%1.1f%%') plt.title("Distribution of Spam and Ham Messages") plt.show()

# Generate a Wordcloud for the Spam emails spam\_text = ' '.join(df[df['label'] == 'spam']['text']) spam\_wordcloud = WordCloud(width=800, height=400, max\_words=100, background\_color='white', random\_state=42).generate(spam\_tex

# Generate a Wordcloud for the Ham emails ham\_text = ' '.join(df[df['label'] == 'ham']['text']) ham\_wordcloud = WordCloud(width=800, height=400, max\_words=100, background\_color='white', random\_state=42).generate(ham\_text)

# Plot the word clouds for spam messages plt.figure(figsize=(10, 4)) plt.subplot(1, 2, 1) plt.imshow(spam\_wordcloud) plt.title('Word Cloud for Spam Messages') plt.axis('off')

# Plot the wordcloud for ham messages plt.subplot(1, 2, 2) plt.imshow(ham\_wordcloud) plt.title('Word Cloud for Ham Messages') plt.axis('off')

# Show both plots side by side plt.tight\_layout() plt.show()

# Vectorize the text data to convert it into numerical features vectorizer = CountVectorizer() x\_train = vectorizer.fit\_transform(x\_train) x\_test = vectorizer.transform(x\_test)

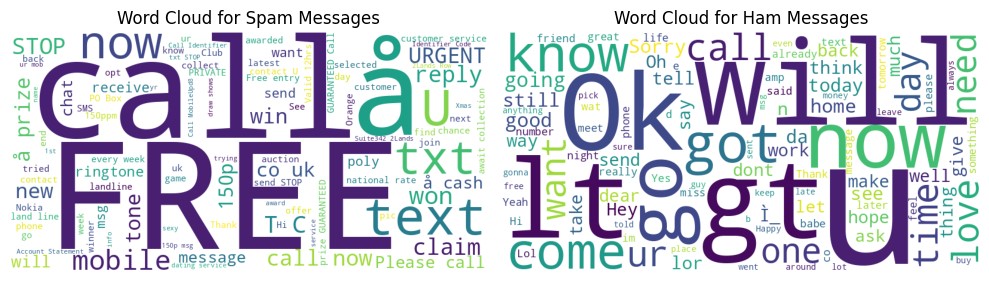
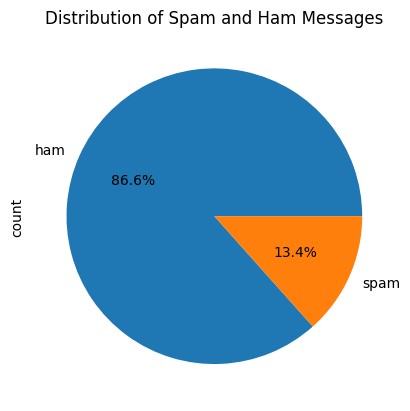
# Train a Multinomial Naive Bayes classifier on the vectorized data model\_multinomial = MultinomialNB(alpha = 0.8, fit\_prior = True, force\_alpha = True) model\_multinomial.fit(x\_train, y\_train)

# Train a Gaussian Naive Bayes classifier on the vectorized data model\_gaussian = GaussianNB() model\_gaussian.fit(x\_train.toarray(), y\_train)

# Calculate and print the accuracy of both models on the test data y\_pred\_multinomial = model\_multinomial.predict(x\_test) accuracy\_multinomial = accuracy\_score(y\_test, y\_pred\_multinomial) print("Accuracy for Multinomial Naive Bayes Model: ", accuracy\_multinomial)

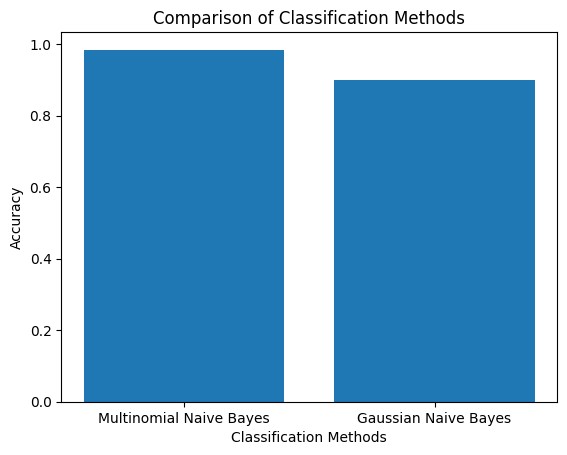
y\_pred\_gaussian = model\_gaussian.predict(x\_test.toarray()) accuracy\_gaussian = accuracy\_score(y\_test, y\_pred\_gaussian) print("Accuracy for Gaussian Naive Bayes Model: ", accuracy\_gaussian) # Plot a comparison of the accuracy scores for the two classification methods methods = ["Multinomial Naive Bayes", "Gaussian Naive Bayes"] scores = [accuracy\_multinomial, accuracy\_gaussian] plt.bar(methods, scores) plt.xlabel("Classification Methods") plt.ylabel("Accuracy") plt.title("Comparison of Classification Methods") plt.show()

Distribution of spam and ham messages: label ham 4825 spam 747 Name: count, dtype: int64



Accuracy for Multinomial Naive Bayes Model: 0.9838565022421525

Accuracy for Gaussian Naive Bayes Model: 0.9004484304932735



## Program 11

Prepare a model for prediction of prostate cancer using KNN Classi er.

# Import necessary libraries import pandas as pd import numpy as np import matplotlib.pyplot as plt import seaborn as sns from sklearn.preprocessing import StandardScaler from sklearn.metrics import classification\_report, confusion\_matrix from sklearn.neighbors import KNeighborsClassifier from sklearn.model\_selection import train\_test\_split

# Load the dataset df = pd.read\_csv('prostate.csv')

# Define feature matrix 'x' and target vector 'y' x=df.drop('Target', axis = 1) y=df['Target']

# Feature scaling using StandardScaler scaler=StandardScaler() df1=pd.DataFrame(scaler.fit\_transform(x),columns=x.columns[::-1])

# 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=1)

# Initialize K-Nearest Neighbors classifier with 1 neighbor knn\_model = KNeighborsClassifier(n\_neighbors=1) knn\_model.fit(x\_train,y\_train)

# Make predictions on the test set y\_pred = knn\_model.predict(x\_test)

# Display the confusion matrix to evaluate model performance print("Confusion Matrix:\n", confusion\_matrix(y\_test,y\_pred))

# Display classification report with precision, recall, F1-score, and accuracy print("Classification Report:\n", classification\_report(y\_test,y\_pred))

# Elbow method for determining the optimal number of neighbors 'K' error\_rate = [] for i in range(1,40):

knn = KNeighborsClassifier(n\_neighbors=i)

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

new\_y\_pred = knn.predict(x\_test)

error\_rate.append(np.mean(new\_y\_pred != y\_test))

# Plot the error rate for different values of K plt.figure(figsize=(12,5)) plt.plot(error\_rate,color='blue', linestyle='dashed', marker='o',

markerfacecolor='red', markersize=10) plt.title('Error Rate vs. K Value') plt.xlabel('K') plt.ylabel('Error Rate') plt.show()

Confusion Matrix:

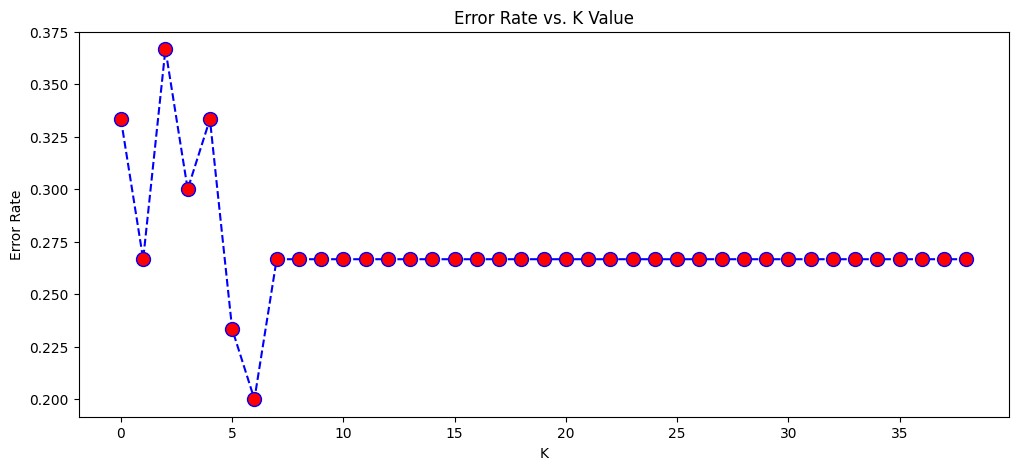
[[18 4]

[ 6 2]]

Classification Report: precision recall f1-score support

0 0.75 0.82 0.78 22 1 0.33 0.25 0.29 8

accuracy 0.67 30 macro avg 0.54 0.53 0.53 30 weighted avg 0.64 0.67 0.65 30



## Program 12

Prepare a model for prediction of survival from Titanic Ship using Random Forest and compare the accuracy with other classi ers also.

# Import necessary libraries import pandas as pd from sklearn.model\_selection import train\_test\_split from sklearn.ensemble import RandomForestClassifier from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix from sklearn.preprocessing import LabelEncoder from sklearn.neighbors import KNeighborsClassifier from sklearn.naive\_bayes import GaussianNB from sklearn.tree import DecisionTreeClassifier import warnings warnings.filterwarnings('ignore')

# Load the dataset df = pd.read\_csv("titanic.csv")

# Drop rows where the target variable is missing df = df.dropna(subset=['Survived'])

# Select features 'x' and target variable 'y' x = df[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare']] y = df["Survived"]

# Encode categorical feature 'Sex' to numeric le = LabelEncoder() x['Sex'] = le.fit\_transform(x['Sex'])

# Fill missing values in 'Age' with the mean x['Age'] = x['Age'].fillna(x['Age'].mean())

# 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)

# Create a Random Forest Classifier with 100 decision trees rf\_model = RandomForestClassifier(n\_estimators=100, random\_state=42)

# Train the Random Forest Classifier rf\_model.fit(x\_train, y\_train)

# Make predictions using the Random Forest Classifier y\_pred\_rf = rf\_model.predict(x\_test)

# Evaluate the Random Forest Classifier rf\_accuracy = accuracy\_score(y\_test, y\_pred\_rf) rf\_classification\_report = classification\_report(y\_test, y\_pred\_rf)

print("Accuracy of Random Forest Classifier: ", rf\_accuracy) print("Classification Report:\n", rf\_classification\_report)

# Comparison with other Models

# Initialize models model1 = KNeighborsClassifier(n\_neighbors=9) model2 = GaussianNB() model3 = DecisionTreeClassifier(criterion='entropy') model4 = RandomForestClassifier(n\_estimators=100)

# List of models for comparison modellist = [model1, model2, model3, model4]

# Evaluate each model print("\n=== Model Comparison Results ===") for model in modellist: model.fit(x\_train, y\_train)

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

# Calculate performance metrics

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

model\_confusion\_matrix = confusion\_matrix(y\_test, y\_pred)

model\_classification\_report = classification\_report(y\_test, y\_pred)

# Display results for each model

print(f"\nModel: {model.\_\_class\_\_.\_\_name\_\_}")

print("Confusion Matrix:")

print(model\_confusion\_matrix)

print(f"Accuracy: {model\_accuracy:.2f}")

print("Classification Report:")

print(model\_classification\_report)

Classification Report: precision recall f1-score support

0 0.71 0.81 0.76 105 1 0.67 0.54 0.60 74

accuracy 0.70 179 macro avg 0.69 0.68 0.68 179 weighted avg 0.69 0.70 0.69 179

Model: GaussianNB Confusion Matrix:

[[85 20]

[21 53]]

Accuracy: 0.77

Classification Report: precision recall f1-score support

0 0.80 0.81 0.81 105 1 0.73 0.72 0.72 74

accuracy 0.77 179 macro avg 0.76 0.76 0.76 179 weighted avg 0.77 0.77 0.77 179

Model: DecisionTreeClassifier Confusion Matrix:

[[83 22]

[21 53]]

Accuracy: 0.76

Classification Report: precision recall f1-score support

0 0.80 0.79 0.79 105 1 0.71 0.72 0.71 74

accuracy 0.76 179 macro avg 0.75 0.75 0.75 179 weighted avg 0.76 0.76 0.76 179

Model: RandomForestClassifier Confusion Matrix:

[[91 14]

[20 54]]

Accuracy: 0.81

Classification Report: precision recall f1-score support

0 0.82 0.87 0.84 105 1 0.79 0.73 0.76 74

accuracy 0.81 179 macro avg 0.81 0.80 0.80 179 weighted avg 0.81 0.81 0.81 179