MANAV RACHNA UNIVERSITY

MANAV RACHNA
vidyanatariksha

SUPERVISED LEARNING

python™

Submitted to:-
Dr. Roshi Saxena

Submitted by:-
Aniket Hans
2K23CSUN01198
AIML-3A

INDEX

Program 1

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

```
# Demonstrate numpy com
mands
# 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 operation
s 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

```
Array a: [0. 0. 0.]
Type of array a: <class 'numpy.ndarray'>
Type of elements in array a: <class 'numpy.float64'>
Array b: [0 0 0]
Type of array b: <class 'numpy.ndarray'>
Type of elements in array b: <class 'numpy.int64'>
Original Array:  [0. 0. 0.]
Shape of Array:  (3,)
Reshaped Array:
 [[0.]
 [0.]
 [0.]]
Shape of Reshaped Array:  (3, 1)
Array created using linspace:  [1.   1.25 1.5  1.75 2.  ]
Element at index 0:  1.0
Element at index -3:  1.5
Array elements from index 0 to 2:  [1.   1.25]
Identity Matrix:
 [[1 0]
 [0 1]]
2-D Array (method 1):
 [[0. 0.]
 [0. 0.]]
2-D Array (method 2):
 [[1 2]
 [3 4]]
Element at (0,1):  2
Element at (0,0):  1
Second row:  [3 4]
First column:  [1 3]
2-D Array:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
First row: [1 2 3]
Third row: [7 8 9]
First column across rows:  [1 4 7]
Array x: [2.  2.5 3.  3.5 4. ]
```

```
Array x: [2.  2.5 3.  3.5 4. ]
Elements at specified indices(0,2,3):  [2.  3.  3.5]
Boolean Array d: [False  True  True False False]
Original Array: [11 15 17 19 24 26 28 35 37 40]
Sorted Array: [11 15 17 19 24 26 28 35 37 40]
Sum: 252
Min: 11
Max: 40
Argmin (index of min): 0
Argmax (index of max): 9
Cumulative Sum: [ 11  26  43  62  86 112 140 175 212 252]
Cumulative Product: [            11            165           2805          53295
        1279080       33256080      931170240    32590958400
  1205865460800 48234618432000]
Mean: 25.2
Median: 25.0
Variance: 87.55999999999999
Standard Deviation: 9.357350052231668
Searchsorted (insert position for 25): 5
a + b: [ 6  8 10 12]
a * b: [ 5 12 21 32]
a + 10: [11 12 13 14]
a * 10: [10 20 30 40]
X:
 [[1 2 3]
 [4 5 6]
 [5 6 7]]
Y:
 [[7 8 9]
 [4 8 9]
 [6 3 5]]
X + Y:
 [[ 8 10 12]
 [ 8 13 15]
 [11  9 12]]
X + 10:
 [[11 12 13]
 [14 15 16]
 [15 16 17]]
```

```
X * Y:
 [[ 33  33  42]
 [ 84  90 111]
 [101 109 134]]
Transpose of X:
 [[1 4 5]
 [2 5 6]
 [3 6 7]]
[2 3]
X == Z: [ True  True]
X == Z after modifying X: [ True False]
```

```
  print("ser_2[1] == ser_2[b]", ser_2[1] == ser_2["b"])
Series with default index:
 0     1
1     1
2     2
3    -3
4    -5
5     8
6    13
dtype: int64
Values in series:  [ 1  1  2 -3 -5  8 13]
Series 2:
 a    1
b    1
c    2
d   -3
e   -5
dtype: int64
ser_2[1] == ser_2[b] True
c    2
a    1
b    1
dtype: int64
Series 3:
 foo    100
bar    200
baz    300
dtype: int64
Series 4:
 foo    100.0
bar    200.0
baz    300.0
qux      NaN
dtype: float64
Null values in ser_4:
 foo    False
bar    False
```

```
baz    False
qux     True
dtype: bool
Sum of series 3 and 4:
 bar    400.0
baz    600.0
foo    200.0
qux      NaN
dtype: float64
Series 4 after setting names for series and index:
 label
foo    100.0
bar    200.0
baz    300.0
qux      NaN
Name: foobarbaz, dtype: float64
Accessing elements by label or position:
0    12
1    20
dtype: int64
5    10
dtype: int64
10
12
Series of Primes:
 0       2
1       3
2       5
3       7
4      11
      ...
57    271
58    277
59    281
60    283
61    293
```

```
Length: 62, dtype: int64
Fibonacci Series:
 0      0
1      1
2      1
3      2
4      3
5      5
6      8
7     13
8     21
9     34
10    55
11    89
dtype: int64
Enter 20 numbers: 1 2 3 4 5 6 7 8 0  9 11 23 44 2 21 34 5 12 23 21
Sum: 241
Min: 0
Max: 44
Enter number: 1
Enter number: 2
Enter number: 3
Enter number: 4
Enter number: 5
Enter number: 6
Enter number: 3
Enter number: 62
Enter number: 47
Enter number: 34
Enter number: 67
Enter number: 433
Enter number: 33
Enter number: 25
Enter number: 24
Enter number: 54
Enter number: 53
Enter number: 2
```

Program 2

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)
```

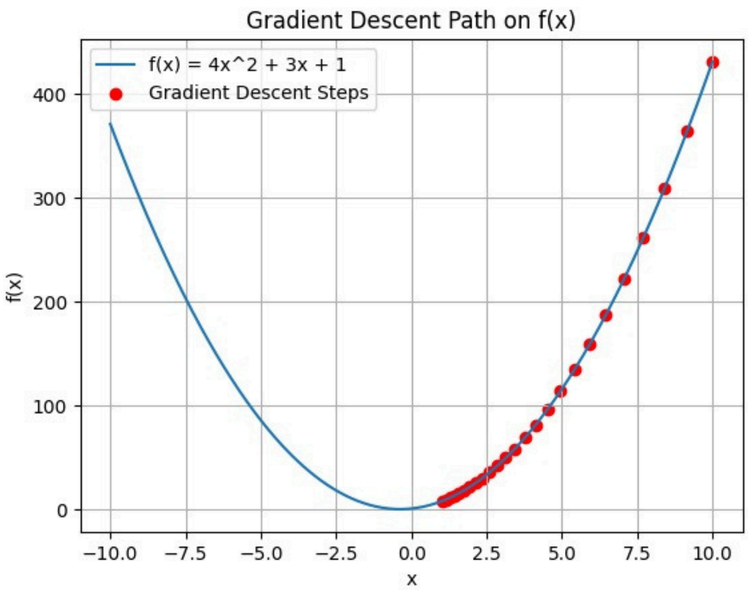
```
Enter the values of x(input) separated by ',': 3, 6, 9, 12, 15, 18, 20
Enter the values of y(output) separated by ',': 15, 28, 63, 90, 120, 152, 190
Enter the value of w: 2.5
Enter the value of b: 0
Predicted Output: [7.5, 15.0, 22.5, 30.0, 37.5, 45.0, 50.0]
Mean Absolute Error: 64.35714285714286
Mean Square Error: 6188.678571428572
```

Program 3

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

# 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)
```

```
For function: 4x^2+3x+1:
Minimum value of x: 0.9152794755738098
```

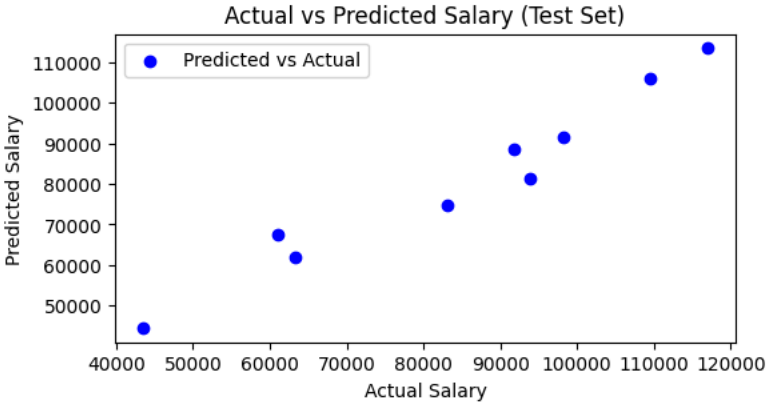


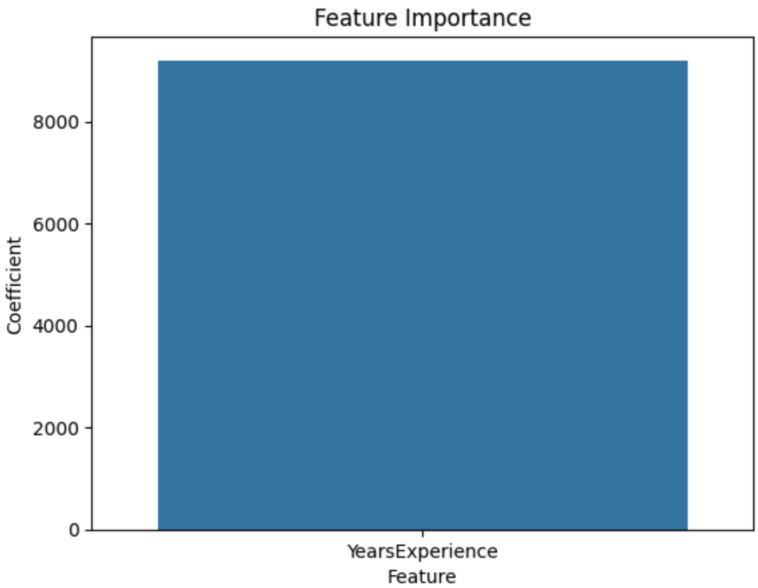
□ Program 4

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
```



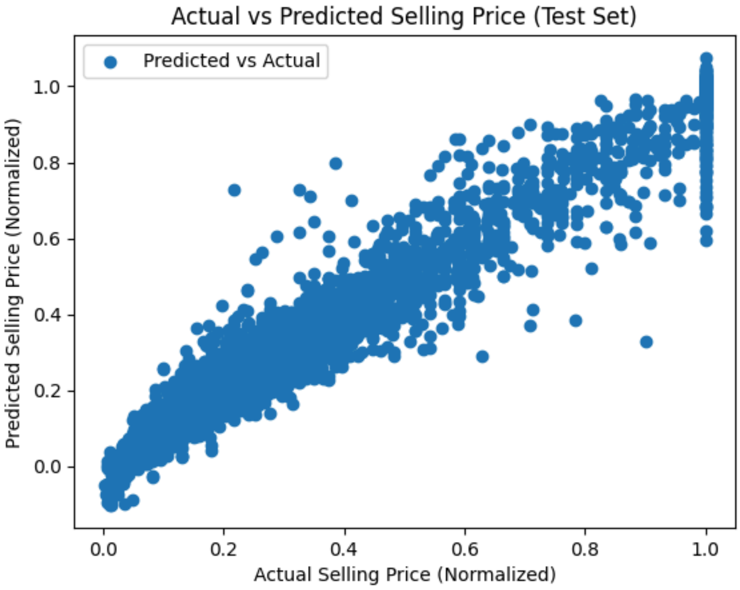


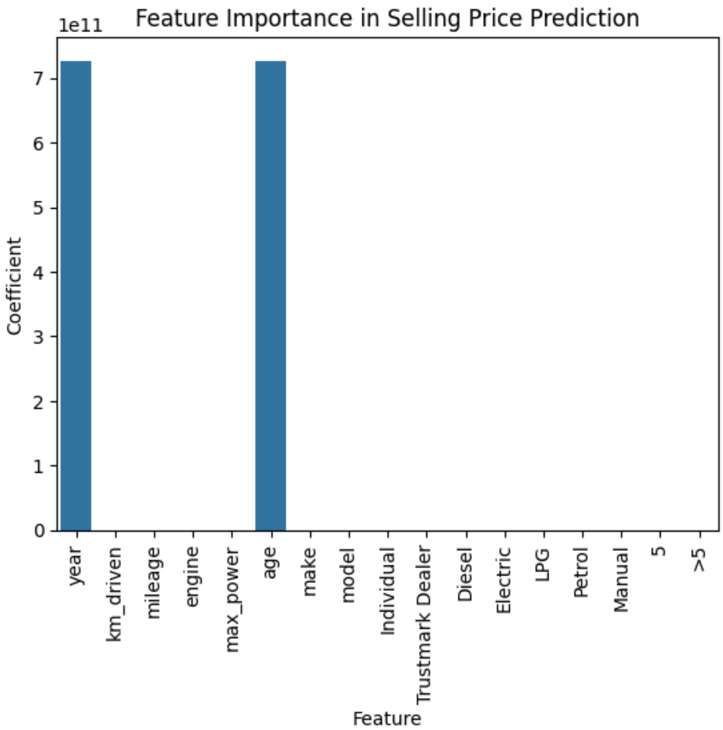
Program 5

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





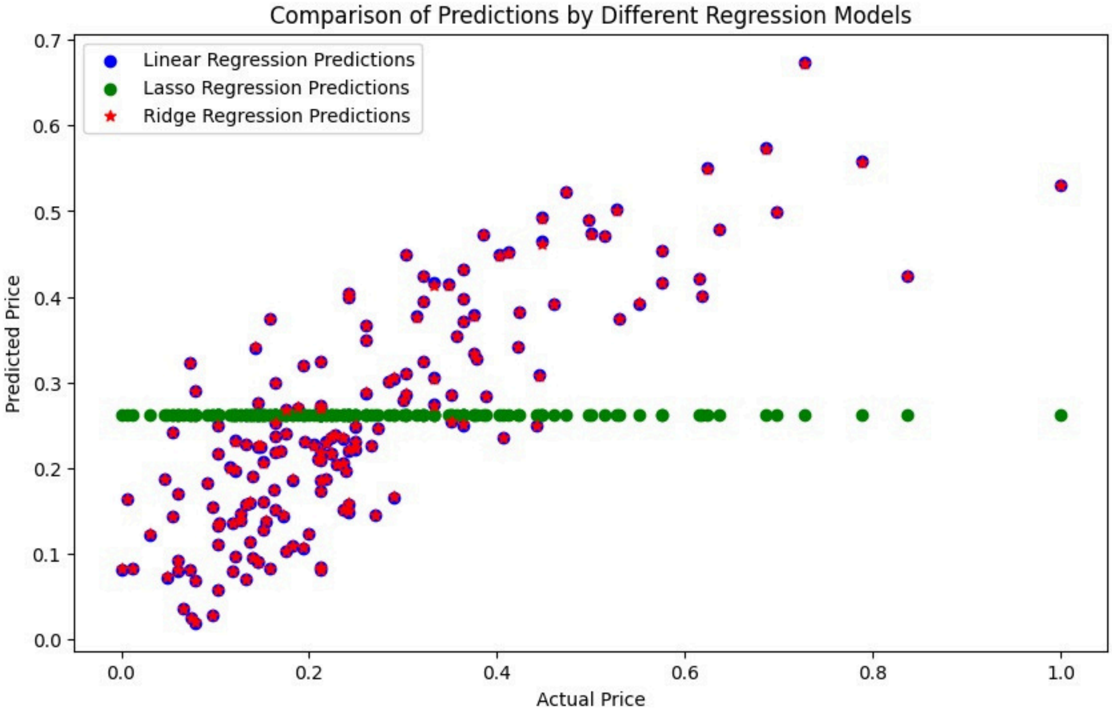
Program 6

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, Rid
ge 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 Regressio
n model = LinearRegression() lasso_model = Lasso(alpha=0.1)
ridge_model = Ridge(alpha=0.1)


# Fit each model to the training dat
a 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 mode
l 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 s
et 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: -0.005042772567566744 5
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
```



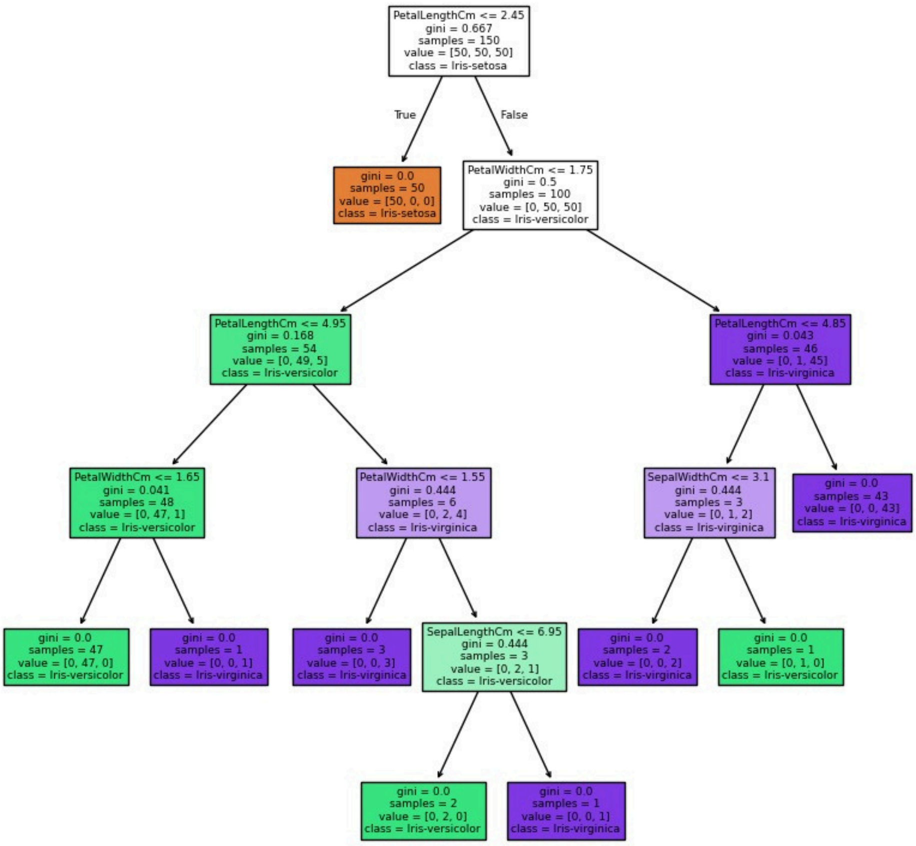
Program 7

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

```
# Import necessary libraries
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier, plot_tre
e 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")
plt.show()
```

```
Best feature: PetalLengthCm
```

Original Decision Tree

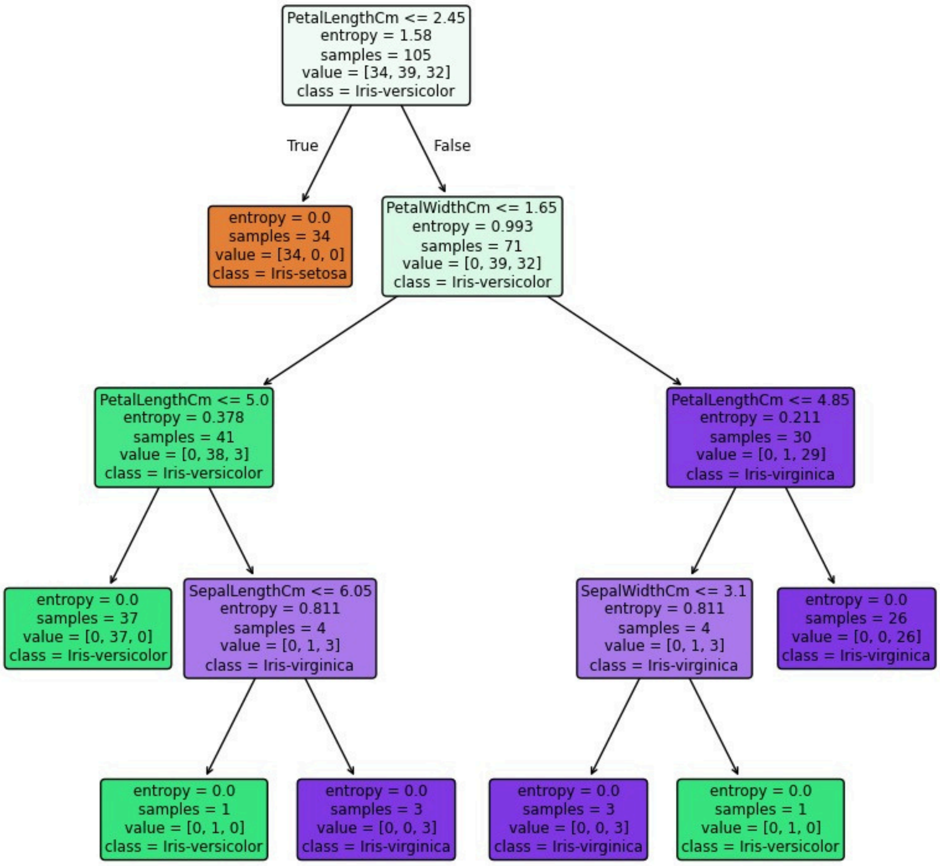


Program 8

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
```

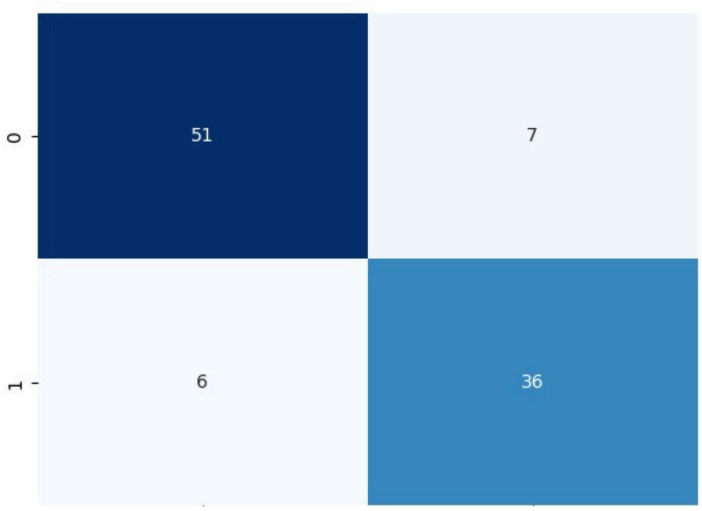


□ Program 9

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: >
```



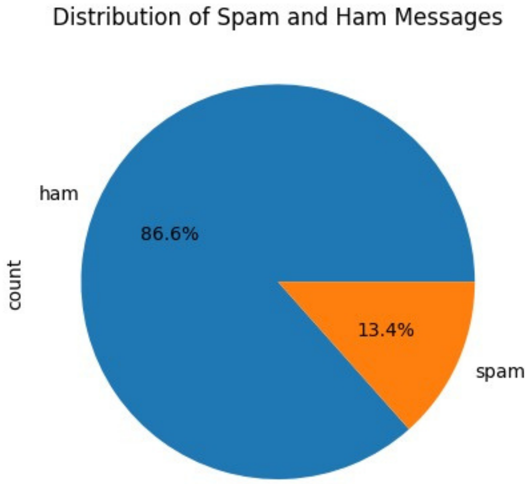
Program 10

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

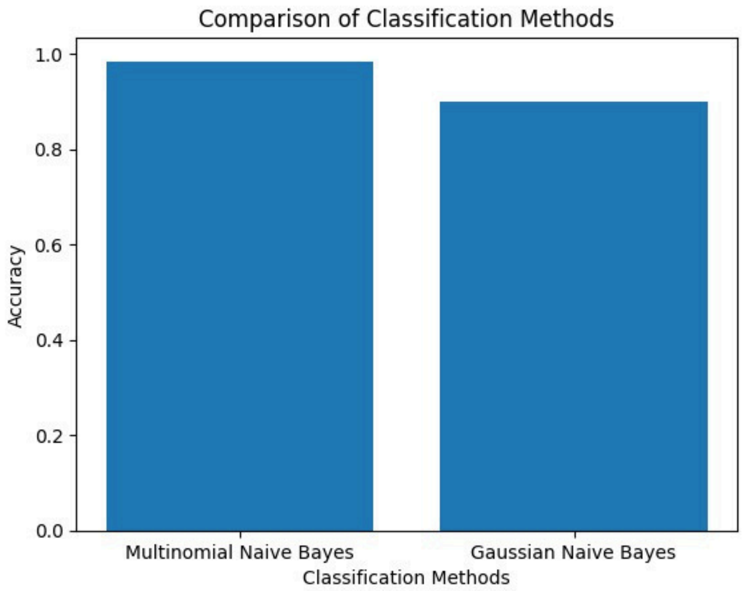
```
# Import libraries import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB, Gaussian
NB
from sklearn.feature_extraction.text import CountVectoriz
er from sklearn.metrics import accuracy_score, f1_score
import matplotlib.pyplot as plt
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(sp

# 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(han

# 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
```



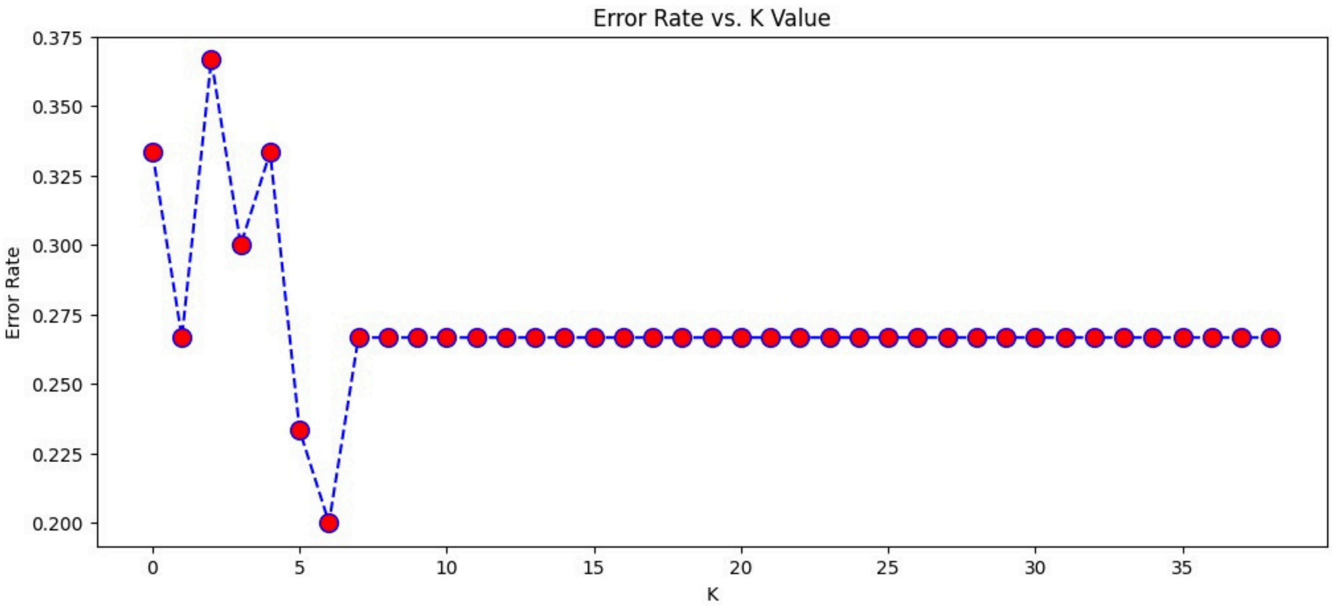
Program 11

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 se
t
y_pred = knn_model.predict(x_tes
t)
# 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