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

import numpy as np

# Create an array of zeros
a = np.zeros(3)
print("a:", a, "\n")

# Check the type of 'a' and elements of 'a'
print("type(a):", type(a), "\n")
print("type(a[0]):", type(a[0]), "\n")

# Create an integer array of zeros
a = np.zeros(3, dtype=int)
print("Integer array a:", a, "\n")

# Create a 1D array of zeros and check its shape
z = np.zeros(10)
print("z:", z, "\n")
print("z.shape:", z.shape, "\n")

# Reshape z to 10x1
z.shape = (10, 1)
print("z reshaped to (10,1):", z, "\n")

# Create a 2x2 zero matrix
z = np.zeros(4)
z.shape = (2, 2)
print("2x2 zero matrix z:", z, "\n")

# Linearly spaced vector from 2 to 4 with 5 elements
z = np.linspace(2, 4, 5)
print("Linearly spaced z:", z, "\n")

# Identity matrix
z = np.identity(2)
print("Identity matrix z:", z, "\n")

# Array with specific values
z = np.array([10, 20])
print("Array z:", z, "\n")
print("type(z):", type(z), "\n")

# 2x2 matrix with specified values
z = np.array([[1, 2], [3, 4]])
print("2x2 matrix z:", z, "\n")

# Linearly spaced vector from 1 to 2 with 5 elements
z = np.linspace(1, 2, 5)
print("Linearly spaced z:", z, "\n")

# Access elements of z
print("z[0]:", z[0], "\n")
print("z[0:2]:", z[0:2], "\n")
print("z[-1]:", z[-1], "\n")

# 2x2 matrix x
x = np.array([[1, 2], [3, 4]])
print("Matrix x:", x, "\n")
print("x[0,0]:", x[0, 0], "\n")
print("x[0, : ]:", x[0, :], "\n")

# 3x3 matrix y
y = np.array([[7, 5, 21], [11, 8, 3], [5, 15, 12]])
print("Matrix y:", y, "\n")
print("y[0, : ]:", y[0, :], "\n")
print("y[ : ,0]:", y[:, 0], "\n")
print("y[1, : ]:", y[1, :], "\n")
print("y[2, : ]:", y[2, :], "\n")
print("y[ : ,1]:", y[:, 1], "\n")
print("y[ : ,2]:", y[:, 2], "\n")

# Linearly spaced vector and selected indices
z = np.linspace(2, 4, 5)
print("Linearly spaced z:", z, "\n")
indices = np.array((0, 2, 3))
```

```
indices = np.array((0, 2, 3))
print("z[indices]:", z[indices], "\n")

# Boolean array
d = np.array([0, 5, 1, 0, 0], dtype=bool)
print("Boolean array d:", d, "\n")

# Array with integers
a = np.array((11, 23, 15, 92, 45, 76, 23, 87, 15, 12))
print("Array a:", a, "\n")

# Sort, sum, max, min, argmax, argmin, cumulative sum, cumulative product, variance, and standard deviation of a
a.sort()
print("Sorted a:", a, "\n")
print("Sum of a:", a.sum(), "\n")
print("Max of a:", a.max(), "\n")
print("Min of a:", a.min(), "\n")
print("Argmax of a:", a.argmax(), "\n")
print("Argmin of a:", a.argmin(), "\n")
print("Cumulative sum of a:", a.cumsum(), "\n")
print("Cumulative product of a:", a.cumprod(), "\n")
print("Variance of a:", a.var(), "\n")
print("Standard deviation of a:", a.std(), "\n")

# Search sorted and other numpy functions on a
print("Searchsorted 76 in a:", a.searchsorted(76), "\n")
print("np.sum(a):", np.sum(a), "\n")
print("np.sort(a):", np.sort(a), "\n")
print("np.min(a):", np.min(a), "\n")
print("np.max(a):", np.max(a), "\n")
print("np.mean(a):", np.mean(a), "\n")
print("np.var(a):", np.var(a), "\n")
print("np.argmax(a):", np.argmax(a), "\n")

# Array operations
a = np.array([2, 2, 2, 2])
b = np.array([3, 3, 3, 3])
print("a + b:", a + b, "\n")
print("a * b:", a * b, "\n")
print("a + 10:", a + 10, "\n")
print("a * 10:", a * 10, "\n")

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

```
np.argmax(a): 9

a + b: [5 5 5 5]

a * b: [6 6 6 6]

a + 10: [12 12 12 12]

a * 10: [20 20 20 20]

X + Y: [[ 6  8 12]
 [ 8 10 12]
 [ 8  6  8]]

X + 10: [[11 12 13]
 [14 13 14]
 [14 13 14]]

X @ Y: [[25 29 37]
 [48 57 76]
 [48 57 76]]

X.T (transpose of X): [[1 4 4]
 [2 3 3]
 [3 4 4]]

X == Y: [[False False False]
 [ True False False]
 [ True  True  True]]

X > 5: [[False False False]
 [False False False]
 [False False False]]
```

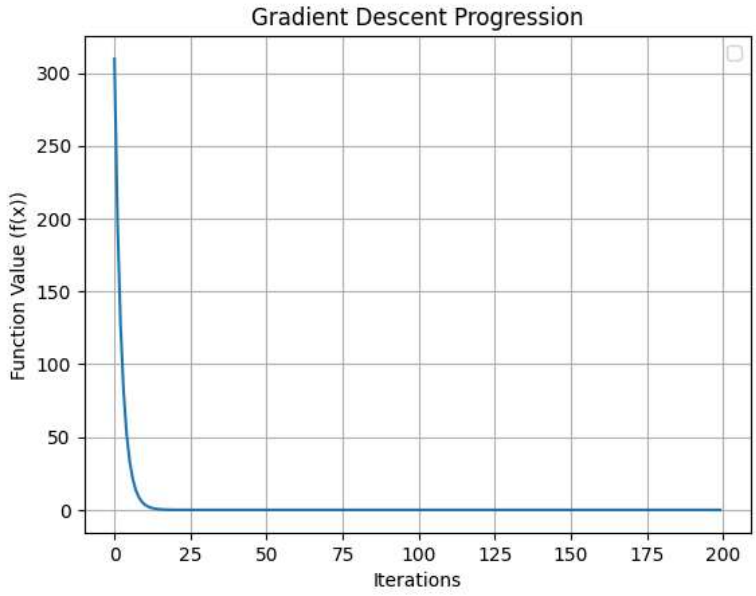
```
# 2. Write a python program to calculate mean square and mean absolute error.

overs = [3,6,9,12,15,18,20]
actual_run = [15,28,63,90,120,152,190]

w = float(input("Enter the value of w : "))
b = float(input("Enter the value of b : "))

a = []

sum = 0
sum_square = 0
for i in range(len(overs)) :
    y = w*overs[i] + b
    a.append(y)
    sum = sum + abs(actual_run[i] - y)
    sum_square = sum_square + (actual_run[i] - y)**2

print("Actual Output : " , actual_run)
print("Predicted Output : " , a)
MAE = sum/len(overs)
MSE = (sum_square/len(overs))

print("Mean Square Error : ",MSE)
print("Mean Absolute Error : ",MAE)
```

```
Enter the value of w : 2
Enter the value of b : 0
Actual Output :  [15, 28, 63, 90, 120, 152, 190]
Predicted Output :  [6.0, 12.0, 18.0, 24.0, 30.0, 36.0, 40.0]
Mean Square Error :  7253.428571428572
Mean Absolute Error :  70.28571428571429
```

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

import matplotlib.pyplot as plt

def gradient_descent(func, w):
    learning_rate = float(input("Enter the Learning Rate : "))
    list_of_weights = []
    weight = w
    delta = 0.0001
    for i in range(200):
        derivative = (func(weight + delta) - func(weight)) / delta
```

```
        weight = (weight - derivative * learning_rate)
        list_of_weights.append(weight)

    iterations = range(len(list_of_weights))
    func_values = [func(w) for w in list_of_weights]

    plt.plot(iterations, func_values)
    plt.title("Gradient Descent Progression")
    plt.xlabel("Iterations")
    plt.ylabel("Function Value (f(x))")
    plt.grid(True)
    plt.legend()
    plt.show()

    return weight, list_of_weights

l = []
min, l = gradient_descent(lambda x: x**2 + 4*x + 4, 20)
print("Minimum Value : ", min)
print("List of Weights : ", l)
```

```
Enter the Learning Rate : 0.1
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ig
```

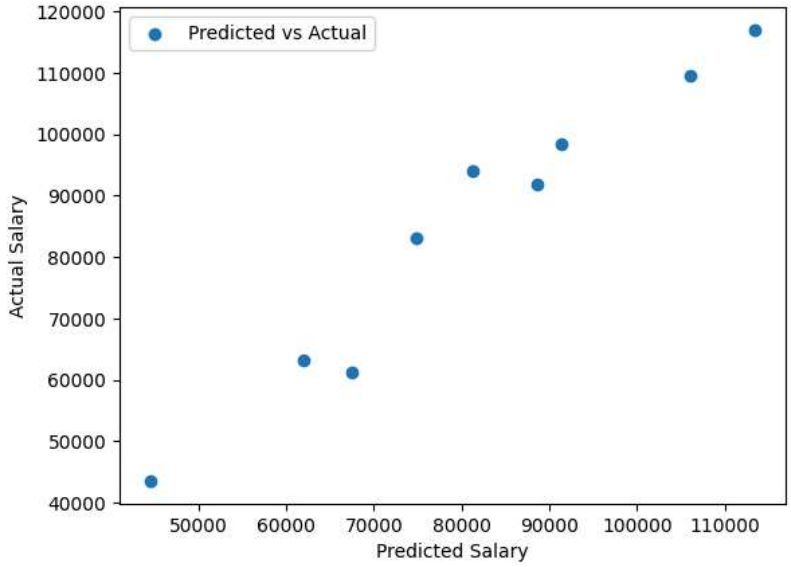



```
Minimum Value :  -2.0000499999977173
List of Weights :  [15.599990000016533, 12.079982000063865, 9.263975600067624, 7.011170480059263, 5.208926384061101, 3.7671311072497815
```

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

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv("Salary_Data.csv")

# Print the shape of the dataset
print("Shape of df:", df.shape, "\n")

# Print the first few rows of the dataset
print("First 5 rows of df:\n", df.head(), "\n")

# Split the dataset into features (x) and target (y)
from sklearn.model_selection import train_test_split
y = df["Salary"]
x = df.drop("Salary", axis=1)
print("Shape of y:", y.shape)
print("Shape of x:", x.shape, "\n")

# Split the 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)
print("Shape of x_train:", x_train.shape)
```

```
print("Shape of y_train:", y_train.shape)
print("Shape of x_test:", x_test.shape)
print("Shape of y_test:", y_test.shape, "\n")

# Create and fit the model
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(x_train, y_train)

# Print model coefficients and intercept
print("Model coefficients:", model.coef_)
print("Model intercept:", model.intercept_, "\n")

# Predict on the test set
y_test_predict = model.predict(x_test)
print("Predicted y values for test set:\n", y_test_predict, "\n")

# Print the model score (R^2)
print("Model score (R^2) on test set:", model.score(x_test, y_test), "\n")

# Plot predicted vs actual values
fig = plt.figure()
plt.scatter(y_test_predict, y_test, label="Predicted vs Actual")
plt.xlabel("Predicted Salary")
plt.ylabel("Actual Salary")
plt.legend()
plt.show()
```

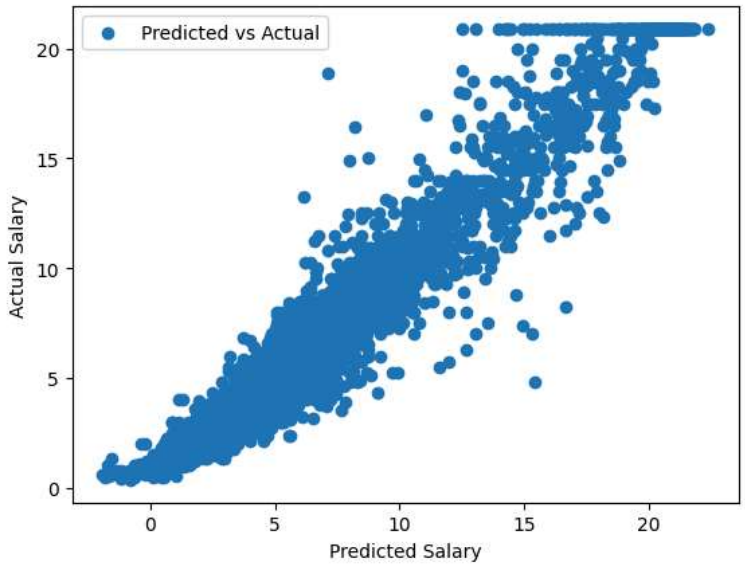
```
Shape of df: (30, 2)

First 5 rows of df:
    YearsExperience   Salary
0              1.1  39343.0
1              1.3  46205.0
2              1.5  37731.0
3              2.0  43525.0
4              2.2  39891.0

Shape of y: (30,)
Shape of x: (30, 1)

Shape of x_train: (21, 1)
Shape of y_train: (21,)
Shape of x_test: (9, 1)
Shape of y_test: (9,)

Model coefficients: [9202.23359825]
Model intercept: 26049.577715443353

Predicted y values for test set:
 [ 74821.41578619  91385.43626305  61938.28874864  81262.97930497
  67459.62890759  88624.76618357 113470.79689886  44454.04491195
 106109.01002026]

Model score (R^2) on test set: 0.9248580247217075
```




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

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv("cars24-car-price-cleaned.csv")

# Print the shape of the dataset
print("Shape of df:", df.shape, "\n")

# Print the first few rows of the dataset
print("First 5 rows of df:\n", df.head(), "\n")

# target variable encoding
df["make"] = df.groupby("make")["selling_price"].transform("mean")
df["model"] = df.groupby("model")["selling_price"].transform("mean")

# Normalization
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df1 = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)
df1
```

```
# Split the dataset into features (x) and target (y)
from sklearn.model_selection import train_test_split
y = df["selling_price"]
x = df.drop("selling_price", axis=1)
print("Shape of y:", y.shape)
print("Shape of x:", x.shape, "\n")

# Split the 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)
print("Shape of x_train:", x_train.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of x_test:", x_test.shape)
print("Shape of y_test:", y_test.shape, "\n")

# Create and fit the model
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(x_train, y_train)

# Print model coefficients and intercept
print("Model coefficients:", model.coef_)
print("Model intercept:", model.intercept_, "\n")

# Predict on the test set
y_test_predict = model.predict(x_test)
print("Predicted y values for test set:\n", y_test_predict, "\n")

# Print the model score (R^2)
print("Model score (R^2) on test set:", model.score(x_test, y_test), "\n")

# Plot predicted vs actual values
fig = plt.figure()
plt.scatter(y_test_predict, y_test, label="Predicted vs Actual")
plt.xlabel("Predicted Salary")
plt.ylabel("Actual Salary")
plt.legend()
plt.show()
```

```
Shape of df: (19820, 18)

First 5 rows of df:
    selling_price    year  km_driven  mileage  engine  max_power   age  \
0           1.20  2012.0     120000    19.70   796.0      46.30  11.0
1           5.50  2016.0      20000    18.90  1197.0      82.00   7.0
2           2.15  2010.0      60000    17.00  1197.0      80.00  13.0
3           2.26  2012.0      37000    20.92   998.0      67.10  11.0
4           5.70  2015.0      30000    22.77  1498.0      98.59   8.0

      make                                      model  Individual  \
0   MARUTI                                  ALTO STD           1
1  HYUNDAI                                GRAND I10 ASTA           1
2  HYUNDAI                                     I20 ASTA           1
3   MARUTI                       ALTO K10 2010-2014 VXI           1
4     FORD  ECOSPORT 2015-2021 1.5 TDCI TITANIUM BSIV           0

   Trustmark Dealer  Diesel  Electric  LPG  Petrol  Manual  5  >5
0                 0       0         0    0       1       1  1   0
1                 0       0         0    0       1       1  1   0
2                 0       0         0    0       1       1  1   0
3                 0       0         0    0       1       1  1   0
4                 0       1         0    0       0       1  1   0

Shape of y: (19820,)
Shape of x: (19820, 17)

Shape of x_train: (13874, 17)
Shape of y_train: (13874,)
Shape of x_test: (5946, 17)
Shape of y_test: (5946,)

Model coefficients: [ 8.94320251e-02 -1.35638241e-06 -4.05906554e-02  2.29106553e-04
  1.50304468e-03 -8.94320251e-02  6.61474952e-02  8.61386888e-01
 -1.44630798e-01 -1.44855016e-01  1.38520375e-01  2.66216225e+00
  3.30456609e-01 -1.36368445e-01 -8.04585280e-02 -3.35811569e-01
 -4.86084483e-01]
Model intercept: -178.08232225367115

Predicted y values for test set:
 [ 1.135053    4.78976725  5.96846351 ...  1.20471131  3.08615105
 10.67406941]

Model score (R^2) on test set: 0.9458843076992299
```




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

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, r2_score
```

```
# Load the dataset
df = pd.read_csv("housing_data.csv")

# Display the first few rows of the dataset
print("First 5 rows of df:\n", df.head(), "\n")

# Replace categorical feature with mean house value based on "ocean_proximity" groups
df["ocean_proximity"] = df.groupby("ocean_proximity")["median_house_value"].transform("mean")

# Scale the features
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df1 = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)
print("Scaled DataFrame:\n", df1.head(), "\n")

# Split the data into features and target variable
y = df1["median_house_value"]
x = df1.drop("median_house_value", axis=1)
print("Shape of y:", y.shape)
print("Shape of x:", x.shape, "\n")

# Split the 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)
print("Shape of x_train:", x_train.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of x_test:", x_test.shape)
print("Shape of y_test:", y_test.shape, "\n")

# Handle missing values by filling with a placeholder
x_train.fillna(999, inplace=True)
x_test.fillna(999, inplace=True)

# Initialize models
linear_model = LinearRegression()
lasso_model = Lasso(alpha=0.1)
ridge_model = Ridge(alpha=0.1)

# Train models
linear_model.fit(x_train, y_train)
lasso_model.fit(x_train, y_train)
ridge_model.fit(x_train, y_train)

# Print coefficients and intercepts
print("Linear Regression Coefficients:", linear_model.coef_)
print("Linear Regression Intercept:", linear_model.intercept_, "\n")

print("Lasso Regression Coefficients:", lasso_model.coef_)
print("Lasso Regression Intercept:", lasso_model.intercept_, "\n")

print("Ridge Regression Coefficients:", ridge_model.coef_)
print("Ridge Regression Intercept:", ridge_model.intercept_, "\n")

# Predictions
y_test_predict_linear = linear_model.predict(x_test)
y_test_predict_lasso = lasso_model.predict(x_test)
y_test_predict_ridge = ridge_model.predict(x_test)

# Model evaluation
linear_mse = mean_squared_error(y_test, y_test_predict_linear)
lasso_mse = mean_squared_error(y_test, y_test_predict_lasso)
ridge_mse = mean_squared_error(y_test, y_test_predict_ridge)

linear_r2 = r2_score(y_test, y_test_predict_linear)
lasso_r2 = r2_score(y_test, y_test_predict_lasso)
ridge_r2 = r2_score(y_test, y_test_predict_ridge)

print("Linear Regression MSE:", linear_mse)
print("Linear Regression R^2:", linear_r2, "\n")

print("Lasso Regression MSE:", lasso_mse)
print("Lasso Regression R^2:", lasso_r2, "\n")

print("Ridge Regression MSE:", ridge_mse)
print("Ridge Regression R^2:", ridge_r2, "\n")

# Comparison
print("Comparison of Model Performance:\n")
```

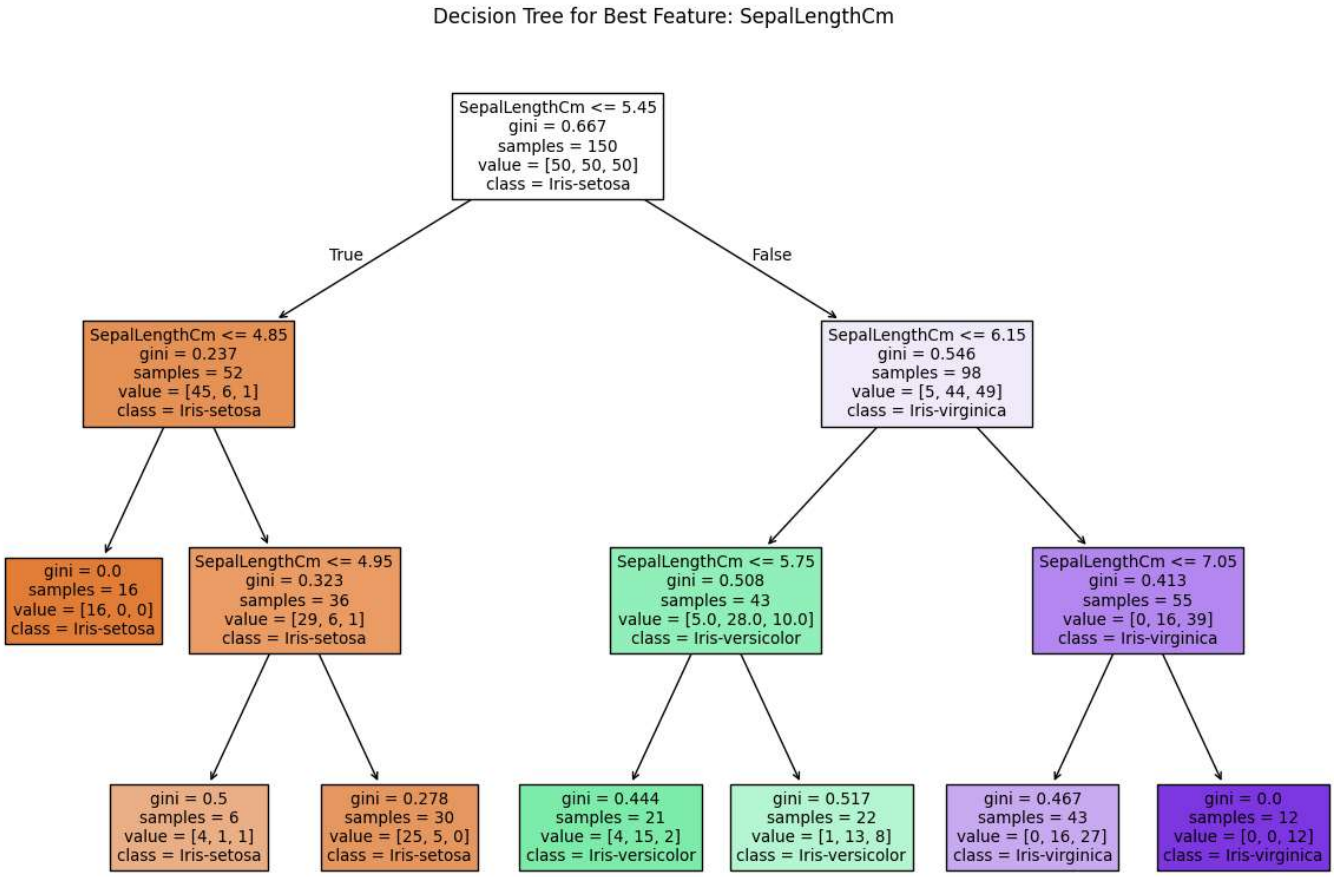
```
print("Model\t\tMSE\t\tR^2")
print(f"Linear\t\t{linear_mse:.4f}\t\t{linear_r2:.4f}")
print(f"Lasso\t\t{lasso_mse:.4f}\t\t{lasso_r2:.4f}")
print(f"Ridge\t\t{ridge_mse:.4f}\t\t{ridge_r2:.4f}")
```

```
   population  households  median_income  median_house_value ocean_proximity
0         322         126         8.3252              452600        NEAR BAY
1        2401        1138         8.3014              358500        NEAR BAY
2         496         177         7.2574              352100        NEAR BAY
3         558         219         5.6431              341300        NEAR BAY
4         565         259         3.8462              342200        NEAR BAY

Scaled DataFrame:
    longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0   0.211155  0.567481            0.784314     0.022331        0.019863
1   0.212151  0.565356            0.392157     0.180503        0.171477
2   0.210159  0.564293            1.000000     0.037260        0.029330
3   0.209163  0.564293            1.000000     0.032352        0.036313
4   0.209163  0.564293            1.000000     0.041330        0.043296

   population  households  median_income  median_house_value  ocean_proximity
0    0.008941    0.020556       0.539668            0.902266         0.525777
1    0.067210    0.186976       0.538027            0.708247         0.525777
2    0.013818    0.028943       0.466028            0.695051         0.525777
3    0.015555    0.035849       0.354699            0.672783         0.525777
4    0.015752    0.042427       0.230776            0.674638         0.525777

Shape of y: (20640,)
Shape of x: (20640, 9)

Shape of x_train: (14448, 9)
Shape of y_train: (14448,)
Shape of x_test: (6192, 9)
Shape of y_test: (6192,)

Linear Regression Coefficients: [-5.21369632e-01 -4.80885402e-01  1.05448058e-01 -9.95155295e-02
  9.05811317e-07 -2.96900236e+00  1.71684723e+00  1.14140894e+00
  1.78608947e-01]
Linear Regression Intercept: 0.40752917375286374

Lasso Regression Coefficients: [-0. -0.  0.  0.  0. -0.  0.  0.  0.]
Lasso Regression Intercept: 0.3972234099154518

Ridge Regression Coefficients: [-5.19071574e-01 -4.77794156e-01  1.05589015e-01 -8.48795182e-02
  8.27450951e-07 -2.83592891e+00  1.64191665e+00  1.13973254e+00
  1.80420220e-01]
Ridge Regression Intercept: 0.4050122758456162

Linear Regression MSE: 0.0203262390236933
Linear Regression R^2: 0.6363169885864803

Lasso Regression MSE: 0.05592001717053842
Lasso Regression R^2: -0.0005372966032284321

Ridge Regression MSE: 0.020343379192693252
Ridge Regression R^2: 0.636010311671446

Comparison of Model Performance:

Model           MSE             R^2
Linear          0.0203          0.6363
Lasso           0.0559          -0.0005
Ridge           0.0203          0.6360
```

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

from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier, plot_tree
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('Iris.csv')

# Display the first few rows of the dataframe
print("First 5 rows of df:\n", df.head(), "\n")

# Prepare the features and target
x = df.drop(['Species', 'Id'], axis=1)
y = df['Species']
```

```
# Initialize the Decision Tree model with the Gini criterion
model = DecisionTreeClassifier(criterion='gini')
print("Decision Tree Model Initialized:\n", model, "\n")

# Dictionary to store Gini importances for each feature
gini_importances = {}

# Display shapes of x and number of samples
print("Shape of x:", x.shape)
print("Number of samples:", x.shape[0], "\n")

# Calculate Gini importance for each feature
for i in range(x.shape[1]):
    # Initialize and fit the model on the individual feature
    model = DecisionTreeClassifier(criterion='gini', max_depth=1)
    model.fit(x.iloc[:, i].values.reshape(-1, 1), y)

    # Retrieve the Gini importance (feature importance)
    gini_importances[i] = model.feature_importances_[0]
    print(f"Gini importance for feature {x.columns[i]}:", gini_importances[i])

# Identify the feature with the highest Gini importance
best_feature = max(gini_importances, key=gini_importances.get)
print(f"Best feature: {x.columns[best_feature]}")

# Plot the decision tree for the best feature
plt.figure(figsize=(15, 10))
model = DecisionTreeClassifier(criterion='gini', max_depth=3)
model.fit(x.iloc[:, best_feature].values.reshape(-1, 1), y)
plot_tree(model, filled=True, feature_names=[x.columns[best_feature]], class_names=y.unique())
plt.title(f"Decision Tree for Best Feature: {x.columns[best_feature]}")
plt.show()
```

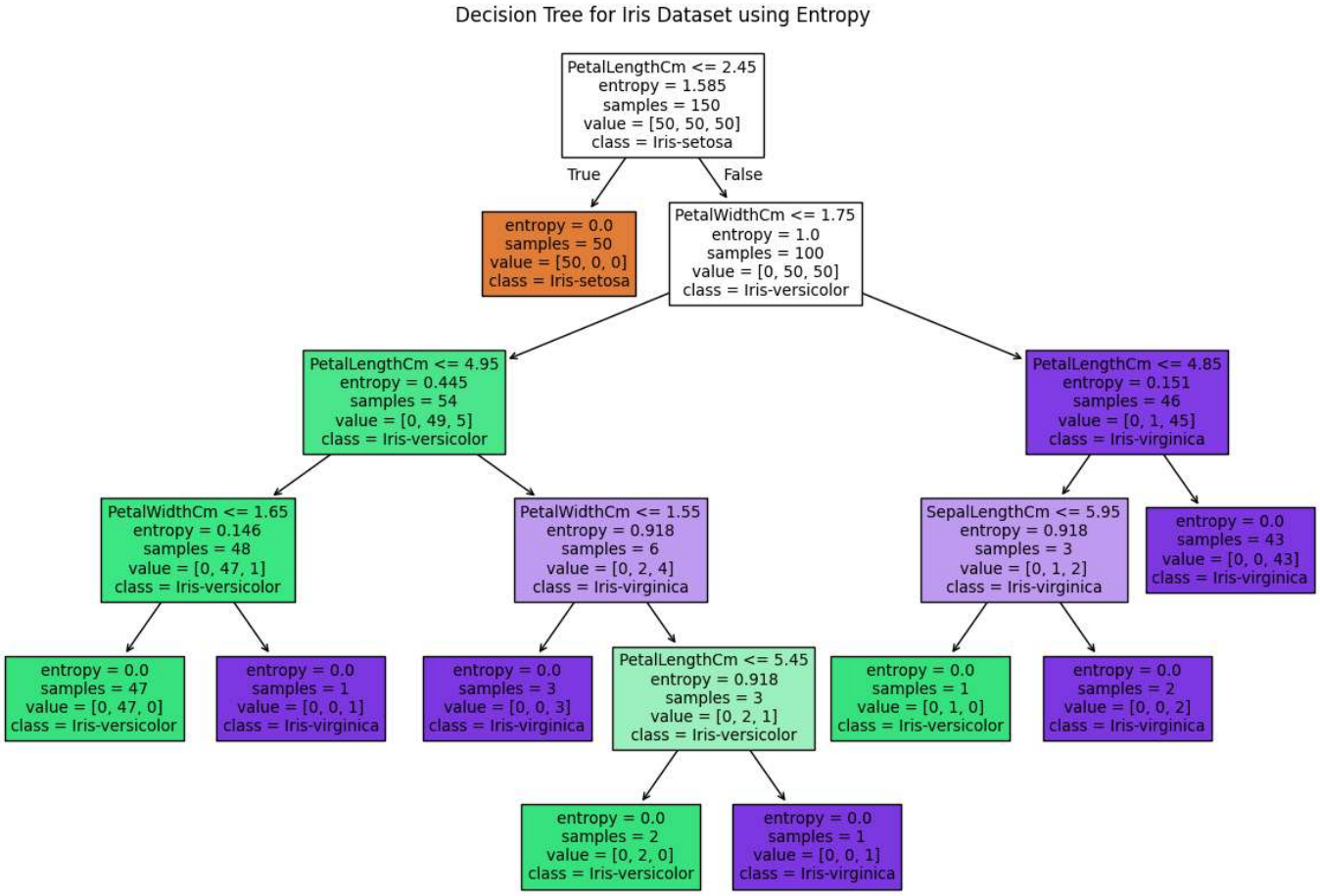
```
First 5 rows of df:
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
0   1            5.1           3.5            1.4           0.2  Iris-setosa
1   2            4.9           3.0            1.4           0.2  Iris-setosa
2   3            4.7           3.2            1.3           0.2  Iris-setosa
3   4            4.6           3.1            1.5           0.2  Iris-setosa
4   5            5.0           3.6            1.4           0.2  Iris-setosa

Decision Tree Model Initialized:
 DecisionTreeClassifier()

Shape of x: (150, 4)
Number of samples: 150

Gini importance for feature SepalLengthCm: 1.0
Gini importance for feature SepalWidthCm: 1.0
Gini importance for feature PetalLengthCm: 1.0
Gini importance for feature PetalWidthCm: 1.0
Best feature: SepalLengthCm
```




```
# 8. Prepare a decision tree model for Iris Dataset using entropy.

import pandas as pd
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier, plot_tree

# Load the dataset
df = pd.read_csv('Iris.csv')

# Display the first few rows of the dataframe
print("First 5 rows of df:\n", df.head(), "\n")

# Prepare the features and target
x = df.drop(['Species', 'Id'], axis=1)
y = df['Species']

# Initialize the Decision Tree model with the Entropy criterion
```

```
model = DecisionTreeClassifier(criterion='entropy', random_state=42)

# Fit the model to the data
model.fit(x, y)

# Print the model details
print("Decision Tree Model Initialized and Fitted:\n", model, "\n")

# Plot the decision tree
plt.figure(figsize=(15, 10))
plot_tree(model, filled=True, feature_names=x.columns, class_names=y.unique())
plt.title("Decision Tree for Iris Dataset using Entropy")
plt.show()
```

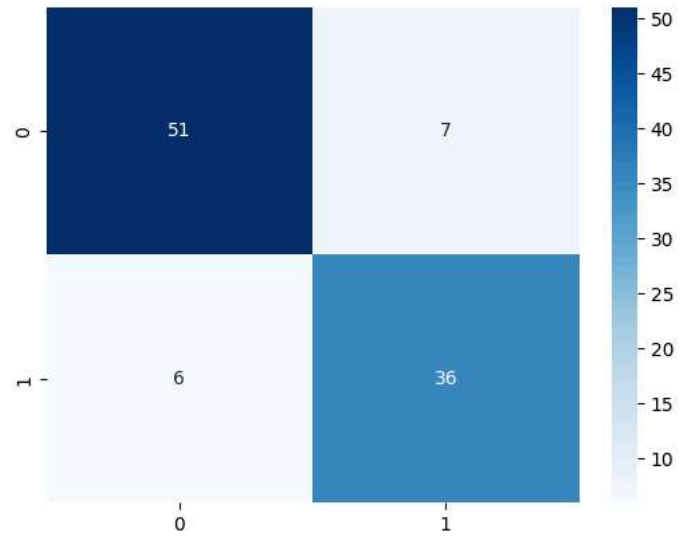
```
First 5 rows of df:
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
0   1            5.1           3.5            1.4           0.2  Iris-setosa
1   2            4.9           3.0            1.4           0.2  Iris-setosa
2   3            4.7           3.2            1.3           0.2  Iris-setosa
3   4            4.6           3.1            1.5           0.2  Iris-setosa
4   5            5.0           3.6            1.4           0.2  Iris-setosa

Decision Tree Model Initialized and Fitted:
 DecisionTreeClassifier(criterion='entropy', random_state=42)
```




```
# 9. Prepare a naïve bayes classification model for prediction of purchase power of a user.

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
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

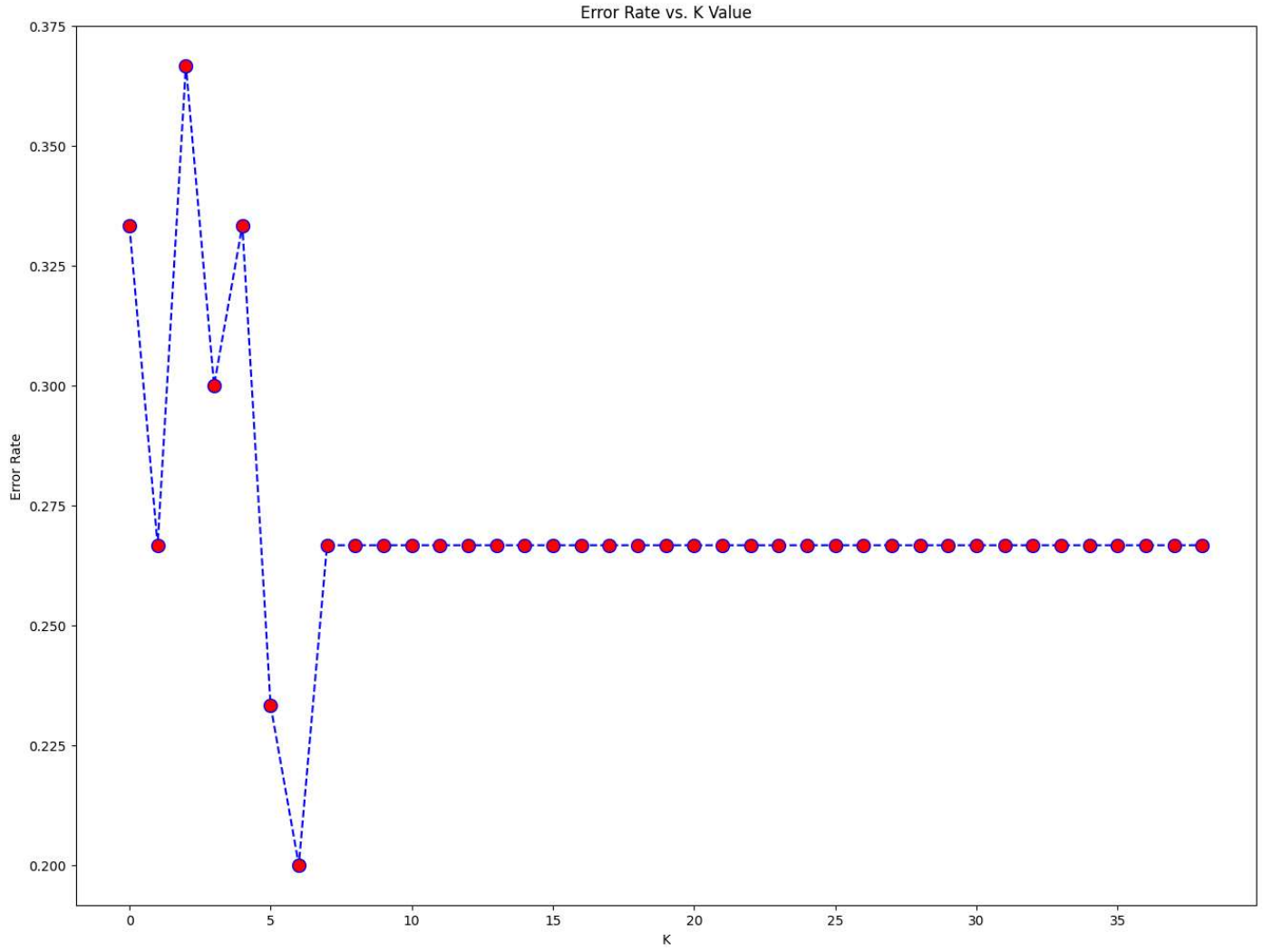
```
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score

df = pd.read_csv("User_Data.csv")

df.head()

df.drop(["User ID"], axis=1, inplace=True)

le = LabelEncoder()
df["Gender"] = le.fit_transform(df["Gender"])

X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state = True)

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

classifier = GaussianNB()
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

accuracy_score(y_test, y_pred)

print(f'Classification Report : \n {classification_report(y_test, y_pred)}')

cf_matrix=confusion_matrix(y_test,y_pred)

print(cf_matrix)
sns.heatmap(cf_matrix,annot=True,fmt='d',cmap='Blues',cbar='False')
```

```
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

[[51  7]
 [ 6 36]]
<Axes: >
```



```
# 10. Prepare a model for prediction of prostate cancer using KNN classifier

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix

df = pd.read_csv("prostate.csv")

df

x = df.drop("Target", axis = 1)
y = df["Target"]

scaler = StandardScaler()
df1 = pd.DataFrame(scaler.fit_transform(x), columns = df.columns[:-1])
df1.head()

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, random_state = 1)

knn_model = KNeighborsClassifier(n_neighbors = 1)
knn_model.fit(x_train, y_train)

y_pred = knn_model.predict(x_test)

print(confusion_matrix(y_test, y_pred))

print(classification_report(y_test, y_pred))

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))

plt.figure(figsize = (16,12))
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()
```

```
[[18  4]
 [ 6  2]]
              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
```




```
# 11. Prepare a multi-nomial name base classificatioin model for text classification of email messages

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

df = pd.read_csv("spam.csv", encoding = "latin-1")

df.head()

df = df[['v1','v2']]
df.head()

df = df.rename(columns = {'v1':'label', 'v2':'text'})
df.head()

x = df['text']
```

```
y = df['label']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 42)

distribution = y.value_counts()
distribution

distribution.plot(kind = 'pie', autopct = '%1.1f%%')
plt.title("Distribution of Spam and Non-Spam Emails")
plt.show()

spam_text = ''.join(df[df['label'] == 'spam']['text'])
spam_text

spam_wordcloud = WordCloud(width = 800, height = 800, max_words = 100 ,background_color = 'white', random_state = 42).generate(spam_text)

spam_wordcloud

ham_text = ''.join(df[df['label'] == 'ham']['text'])
ham_text

ham_wordcloud = WordCloud(width = 800, height = 800, max_words = 100, background_color = 'white', random_state = 42).generate(ham_text)

# plot the wordcloud for spam messages

plt.figure(figsize = (10,4))
plt.subplot(1,2,1)
plt.imshow(spam_wordcloud)
plt.title('Wordcloud for Spam Emails')
plt.axis('off')

# plot the wordcloud for ham messages
plt.subplot(1,2,2)
plt.imshow(ham_wordcloud)
plt.title('Wordcloud for Ham Emails')
plt.axis('off')

vectorizer = CountVectorizer()

x_train = vectorizer.fit_transform(x_train)
x_test = vectorizer.transform(x_test)

model_multinomial = MultinomialNB(alpha = 0.8, fit_prior = True, force_alpha = True)

model_multinomial.fit(x_train, y_train)

model_guassain = GaussianNB()

model_guassain.fit(x_train.toarray(), y_train)

y_pred_multinomial = model_multinomial.predict(x_test)

accuracy_multinomial = accuracy_score(y_test, y_pred_multinomial)

print("Accuracy of Multinomial model is :", accuracy_multinomial)

y_pred_guassian = model_guassain.predict(x_test.toarray())

accuracy_guassain = accuracy_score(y_test, y_pred_guassian)

print("Accuracy of Guassain model is :", accuracy_guassain)

methods = ["Multinomial naiye bayes", "Guassain naive bayes"]
scores = [accuracy_multinomial, accuracy_guassain]
plt.bar(methods, scores)
```