MANAV RACHNA UNIVERSITY

SCHOOL OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE & TECHNOLOGY

LAB FILE

Supervised Learning (CSH212B-T)

Submitted to:

Dr.Roshi Saxena

Manager, Xebia Academy

Submitted by:

CH.Harsha Luke Chowdary

2K23CSUN01246

CSE AIML-3B

MANAV RACHNA UNIVERSITY

SCHOOL OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE & TECHNOLOGY

Supervised Learning Projects

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.	

LAB-1

```
import numpy as np
import pandas as pd
```

```
emp_data = pd.read_csv("Employee.csv")
```

```
print(emp_data.shape)
print(emp_data.columns)
print(emp_data.describe())
print(emp_data['Age'].mean())
```

```
(4653, 9)
Index(['Education', 'JoiningYear', 'City', 'PaymentTier', 'Age', 'Gender',
       'EverBenched', 'ExperienceInCurrentDomain', 'LeaveOrNot'],
      dtype='object')
       JoiningYear  PaymentTier          Age  ExperienceInCurrentDomain  \
count  4653.000000  4653.000000  4653.000000                4653.000000
mean   2015.062970     2.698259    29.393295                   2.905652
std       1.863377     0.561435     4.826087                   1.558240
min    2012.000000     1.000000    22.000000                   0.000000
25%    2013.000000     3.000000    26.000000                   2.000000
50%    2015.000000     3.000000    28.000000                   3.000000
75%    2017.000000     3.000000    32.000000                   4.000000
max    2018.000000     3.000000    41.000000                   7.000000

        LeaveOrNot
count  4653.000000
mean      0.343864
std       0.475047
min       0.000000
25%       0.000000
50%       0.000000
75%       1.000000
max       1.000000
29.393294648613796
```

```
ages = emp_data['Age'].values
print(ages)
print(np.mean(ages))
print(np.std(ages))
print(np.unique(ages))
```

```
[34 28 38 ... 27 30 33]
29.393294648613796
4.825568381752676
[22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41]
```

LAB-2

```
import numpy as np


def mae(List_X, List_Y, W):
    b = 0
    error = 0
    for i in range(len(List_X)):
        predicted_output = W * List_X[i] + b
        error += abs(List_Y[i] - predicted_output)
    return error / len(List_X)


def mse(List_X, List_Y, W):
    b = 0
    error = 0
    for i in range(len(List_X)):
        predicted_output = W * List_X[i] + b
        error += (List_Y[i] - predicted_output) ** 2
    return error / len(List_X)
List_X = [2, 7, 8, 9]
List_Y = [13, 15, 19, 17]
W = 2


mae_result = mae(List_X, List_Y, W)
print("Mean Absolute Error is: ", mae_result)
```

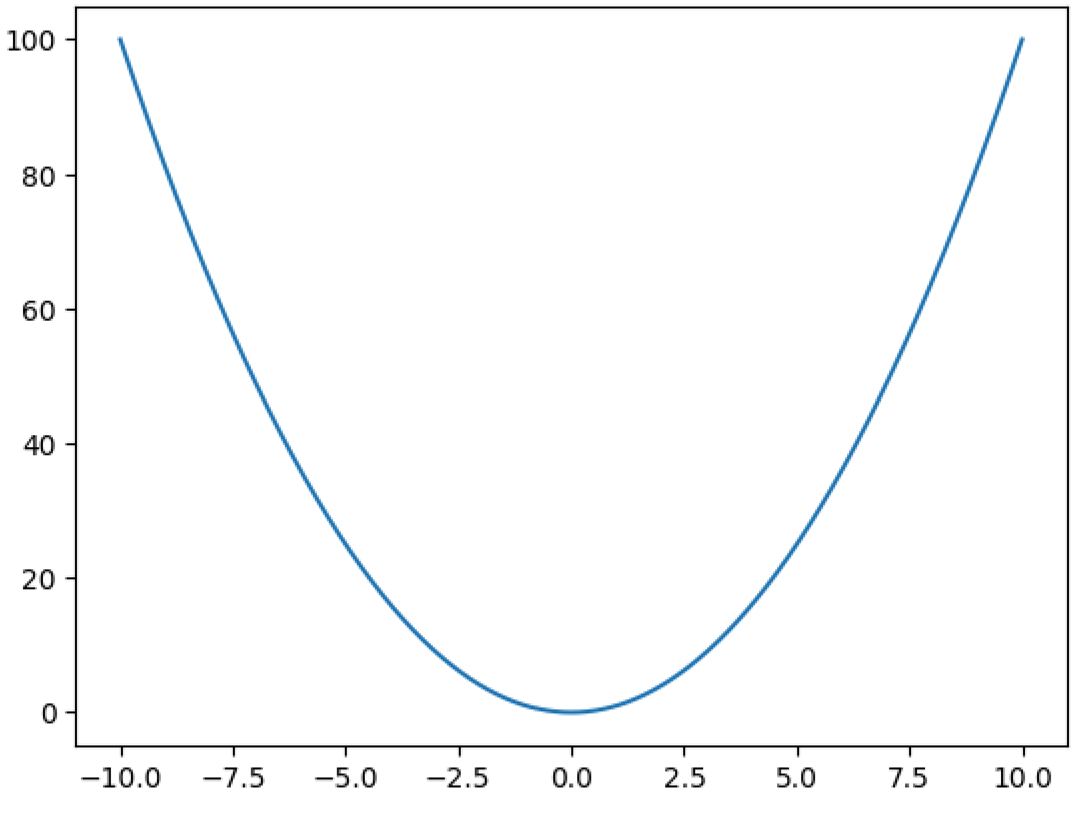
```
Mean Absolute Error is:  3.5
```

```
mse_result = mse(List_X, List_Y, W)
print("Mean Square Error is: ", mse_result)
```

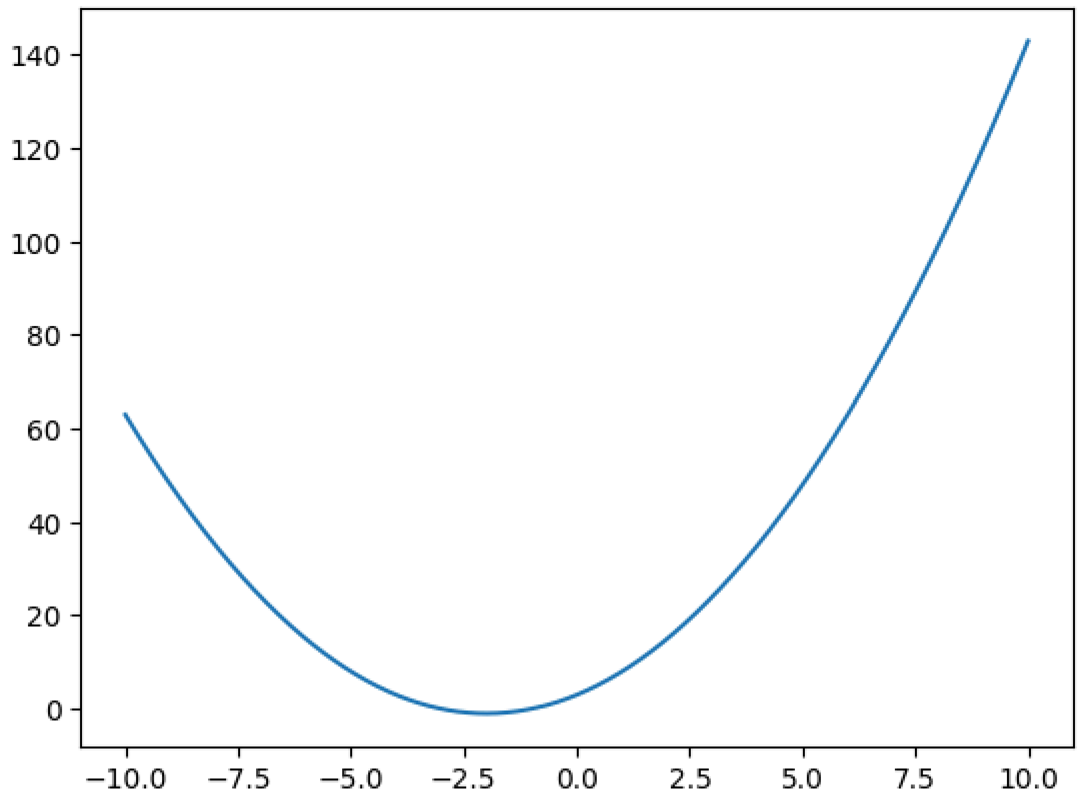
```
Mean Square Error is:  23.0
```

LAB-3

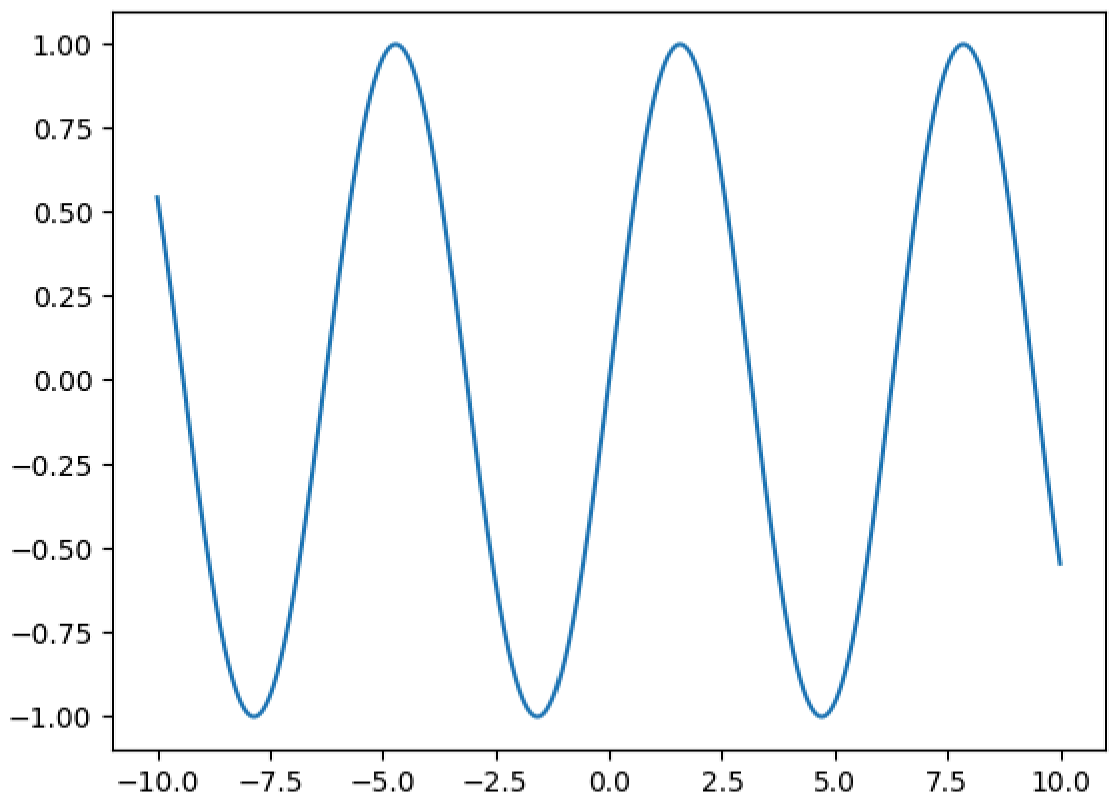
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import math

def plot_function(func):
  x_values = np.linspace(-10, 10, 1000)  #linspace(start, end, no.of points (or)
  y_values = [func(x) for x in x_values]  #list comprehension is used to make our
  # print("x values are : ", x_values)
  # print("y values are : ", y_values)
  plt.plot(x_values,y_values)

plot_function(lambda x: x**2) # lambda funtions are the anonymus functions with c
```



```
plot_function(lambda x: x**2 + 4*x +3)
```

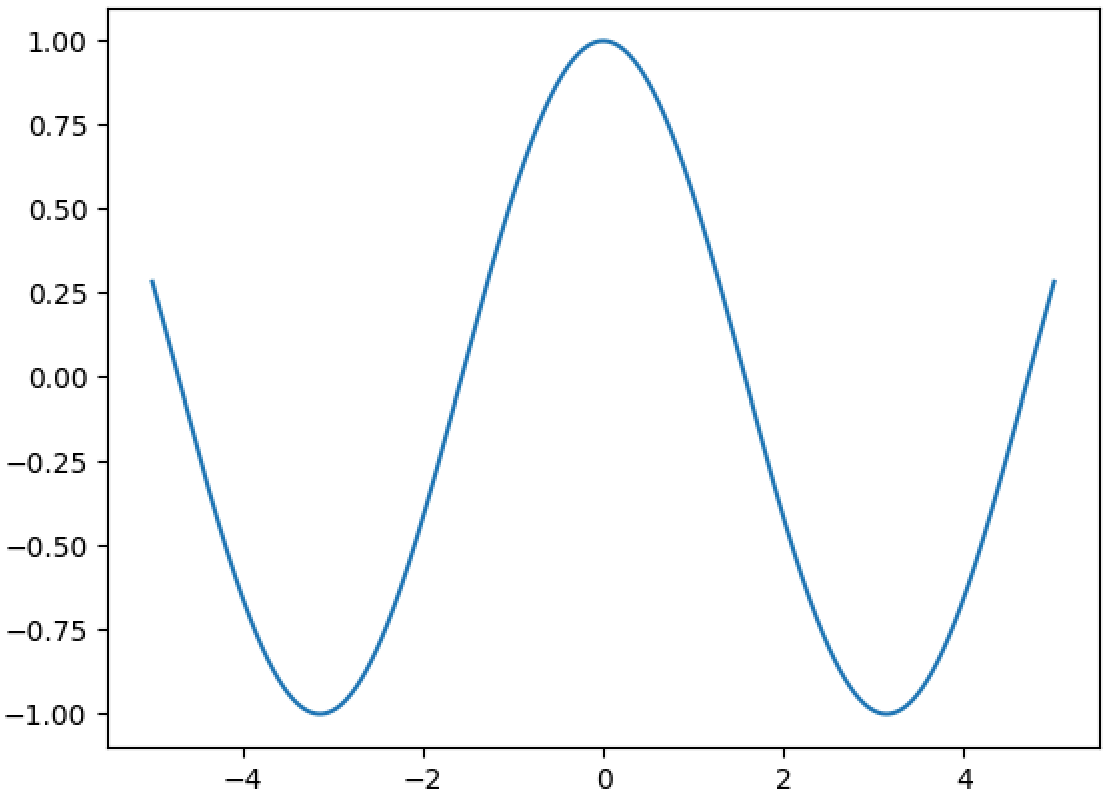



```
plot_function(lambda x: np.sin(x))
```

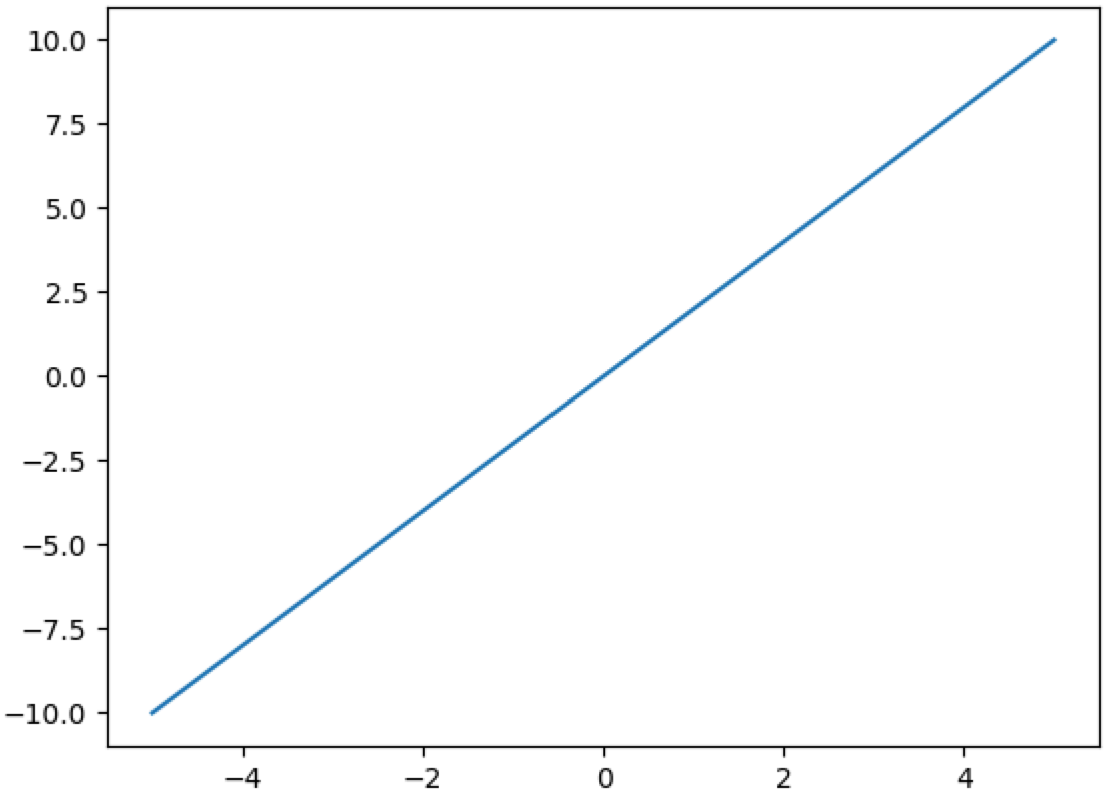



```
def plot_derivative(func):
  x_values = np.linspace(-5, 5, 1000)
  delta_x = 0.0001
  y_values = ((func(x_values + delta_x)) - func(x_values)) / delta_x
  plt.plot(x_values, y_values)
```

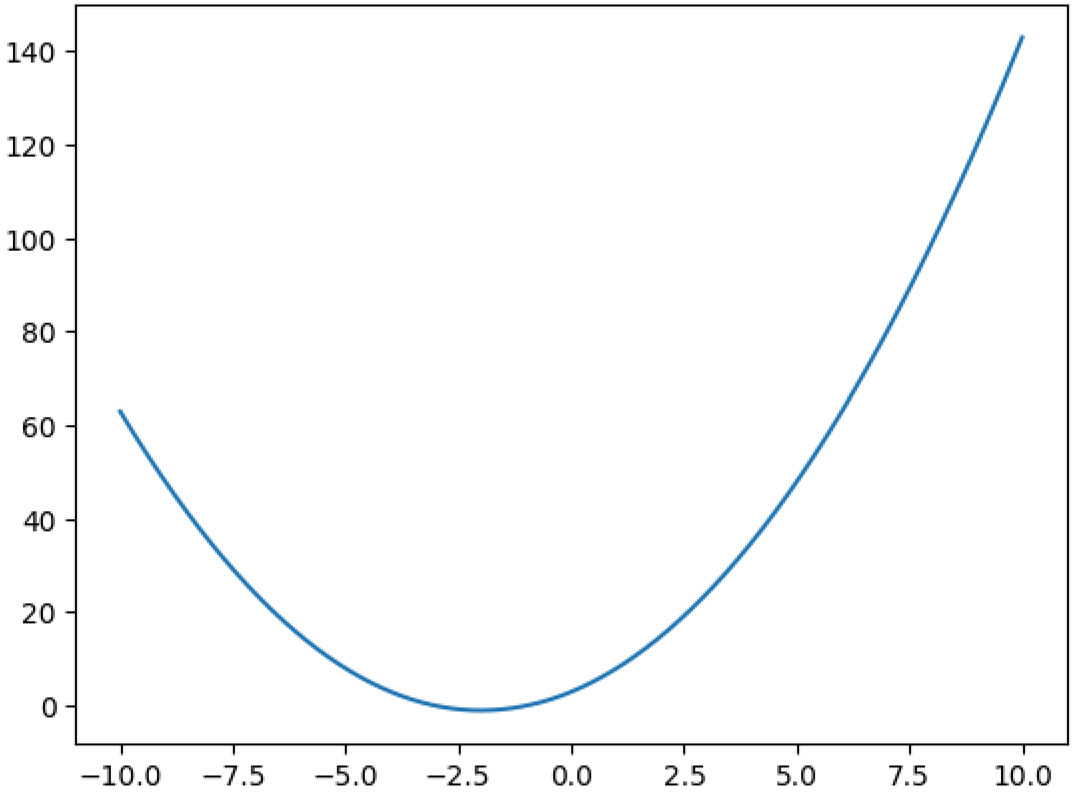
```
plot_derivative(lambda x : np.sin(x))
```



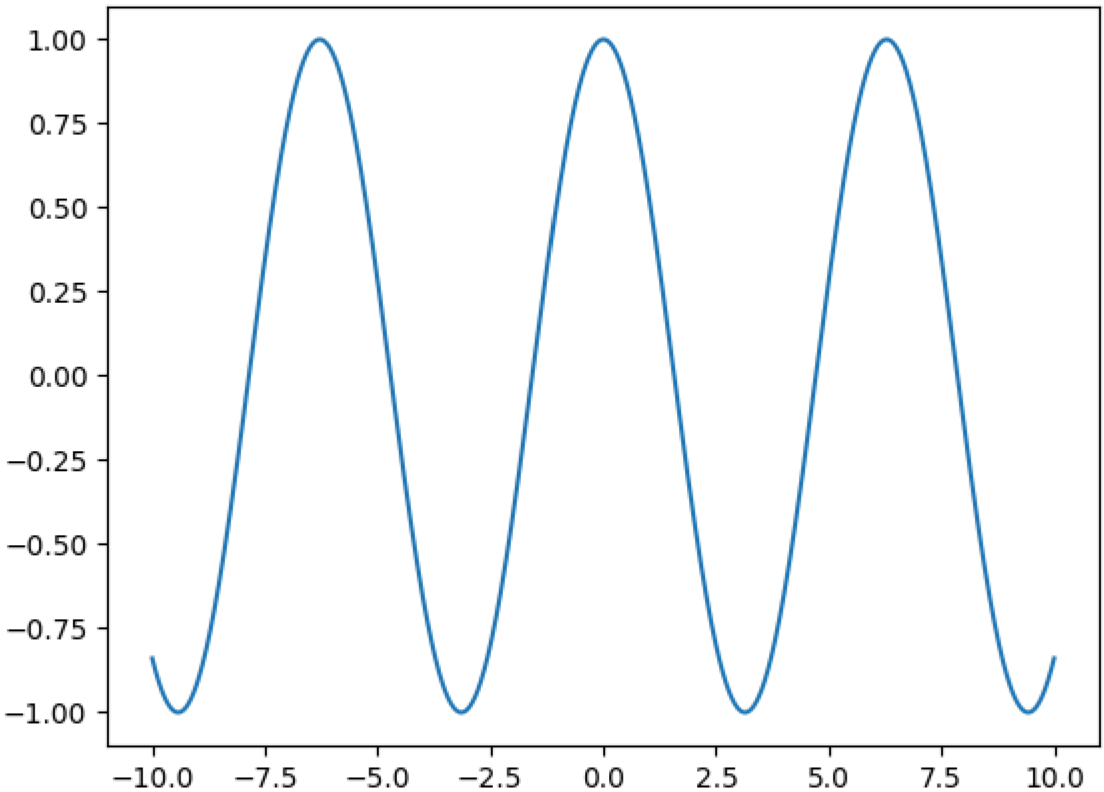

```
plot_derivative(lambda x : x**2)
```




```
plot_function(lambda x: x**2 + 4*x +3)
```




```
plot_function(lambda x: np.cos(x))
```

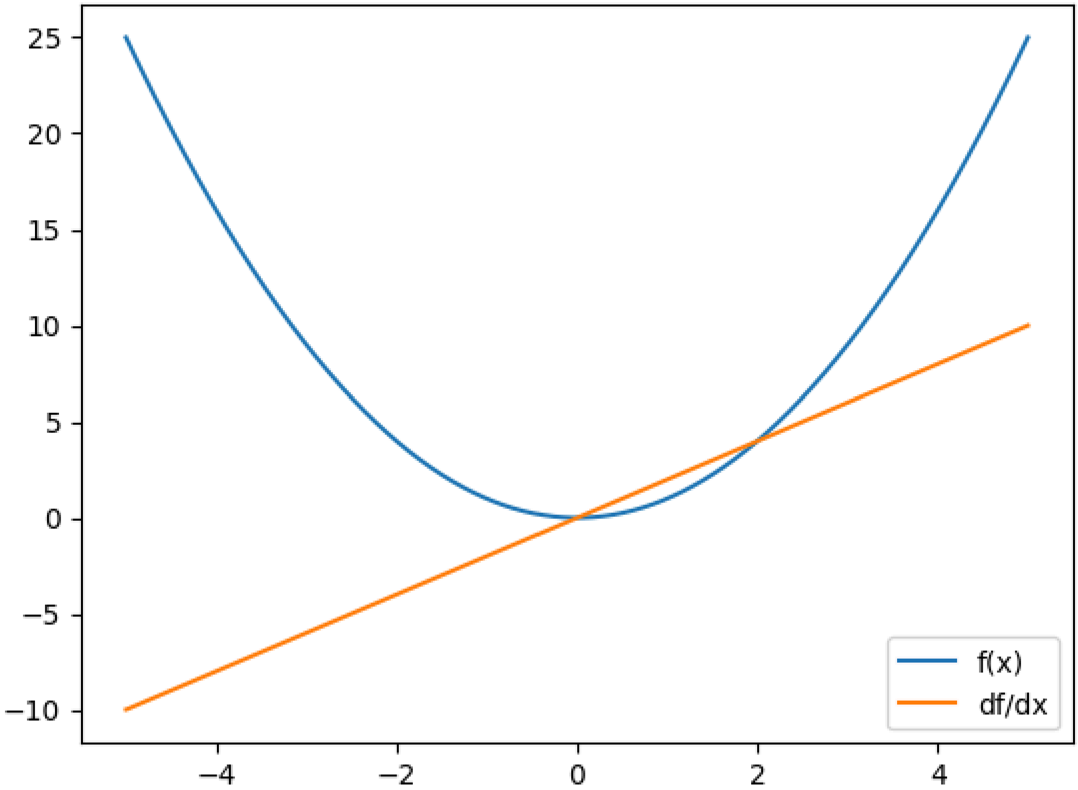



```
# GIven a function f and a starting point w(weight), try to find the minima or gr
def gradient_descent(func, w):
  list_of_weights = []
```

```
  weight = w
  delta = 0.0001
  learning_rate = 0.1
  for i in range(1000):
    derivative = (func(weight+delta) - func(weight))/delta
    weight = weight - (learning_rate * derivative)
    list_of_weights.append(weight)
  return list_of_weights

gradient_descent(lambda x: x**2+4*x+3, 10)
```

Show hidden output

```
def f(x):
  return x **2

x = np.linspace(-5,5,1000)
dfdx = np.gradient(f(x), x)
plt.plot(x, f(x), label = 'f(x)')
plt.plot(x, dfdx, label = 'df/dx')
plt.legend()
plt.show()
```



LAB-4

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

```
df = pd.read_csv("salary_data.csv") #divide your data frame into 2 data frames ,
```

Show hidden output

```
x = pd.DataFrame(df["Salary"])
y = df["YearsExperience"]
x,y
#Again we have to divide both the data frame into train data and test data
#Train is the one from which we will be training our model the test data is the c
#We will be using 80% of the dat for training purpose and 20% of the data for tes
```

Show hidden output

```
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 =
```

```
x_train, y_train, x_test, y_test
```

Show hidden output

```
#After splitting the data frame now we prepare a linear regression model on our trained d
#For building a model we will import linear regression library

from sklearn.linear_model import LinearRegression
```

```
model = LinearRegression()
model.fit(x_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
x_pred = model.predict(x_train)
y_pred = model.predict(x_test)
```

```
x_pred
```

```
array([ 8.05410626,  9.36023523,  1.64238074,  1.41686206,  8.50096733,
        3.75755794,  5.97233923,  3.4371335 ,  3.16191719,  9.23734844,
```

```
       2.30160522,  4.37136547,  1.58516581,  4.57067804, 10.20175398,
      10.2559411 ,  3.44840943,  4.20598511,  3.42418705,  3.30276195,
       3.39129891])
```

```
#After buliding a linear regression model, we will make predictions on test data and then
model.coef_
```

```
array([0.00010441])
```

```
model.intercept_
```

```
-2.522510616511819
```

```
model.score(x_test,y_test) #This will give the accuracy of the test
```

```
0.9242662549548135
```

```
plt.scatter(x_train, y_train, color ="Green")
plt.plot(x_train, x_pred, color = "red")
plt.title("Salary vs Experience")
plt.xlabel("Years of experience")
plt.ylabel("Salary")
plt.show()
```

LAB-5

```
# Question
# Prepare a machine learning model for prediction of presale price of used cars

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

```
df = pd.read_csv("cars24-car-price-cleaned.csv")
```

```
df.info() #info command is used to share the structure of the data frame i.e.; hc
```

Show hidden output

```
df.head()
```

Show hidden output

Next steps: View recommended plots New interactive sheet

```
df.describe()
```

Show hidden output

```
df["make"].nunique() # nunique will give me the unique values present in a particular prc
```

```
41
```

```
df["model"].nunique()
```

```
3233
```

```
df["make"].value_counts() # .value_count will return the total value associated with the
```

Show hidden output

```
df["model"].value_counts()
```

Show hidden output

Steps for multiple Linear Regression

Before dividing data frame into two parts i.e.; target and input variable, we have to check wether our df contains any categorical data or not.

If out df contains the categorical data then we have to convert the categorical into continuous data by encoding.

When the no.of values in the categorical column are limited or very less we make use one hot encoding but if the categorical data has large no.of values then we make use of traget variable encoding.

Because the machine learning model is a mathematical model, it only understands ditgits not letters.

Target variable encoding is replacing the categorical column by the mean or avg of the target variable.

```
df["make"] = df.groupby("make")["selling_price"].transform("mean")
df["make"]
```

Show hidden output

```
df["model"] = df.groupby("model")["selling_price"].transform("mean")
df["model"]
```

Show hidden output

Step 2

Divide the dataframe into target features and independent features

Step 3

Feature scaling or normalization

Features scaling

It means we have to scale all the feature in same range , that means we will be keeping all the features in the range of 1 so that our machine learning model does not create perception about any other feature because every features are important to us.

```
#Normalization(Scaling)
from sklearn.preprocessing import MinMaxScaler  # MinMacScaler is the lib used for featur
scaler = MinMaxScaler()
df1 = pd.DataFrame(scaler.fit_transform(df), columns = df.columns)
df1
```

Show hidden output

Next steps: View recommended plots New interactive sheet

```
y = df1["selling_price"]
x = df1.drop("selling_price", axis =1) # Axis = 0 means rows are dropped, axis = 1 means
y.shape, x.shape # It is used to tell the shape of df i.e.; how many rows and columns are
```

```
((19820,), (19820, 17))
```

```
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_s
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

```
((13874, 17), (13874,), (5946, 17), (5946,))
```

```
from sklearn.linear_model import LinearRegression
```

```
model = LinearRegression()
model.fit(x_train, y_train)
```

```
▾ LinearRegression ⓘ ?
LinearRegression()
```

```
x_pred = model.predict(x_train)
y_pred = model.predict(x_test)
x_pred
```

```
array([0.12597656, 1.00378418, 0.35705566, ..., 0.14587402, 0.25183105,
       0.08837891])
```

```
model.coef_
```

```
array([ 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.score(x_test,y_test)
```

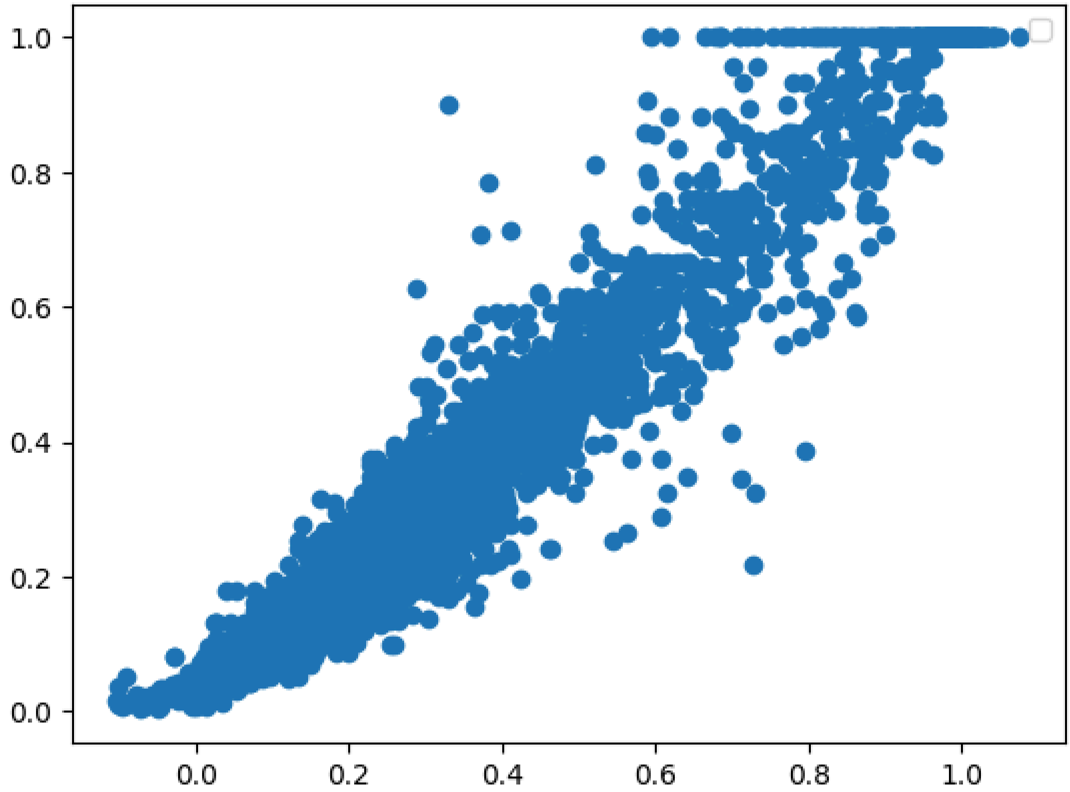
```
0.9459835819294395
```

```
y_test_predict = model.predict(x_test)
y_test_predict
```

```
array([0.04589844, 0.21557617, 0.27368164, ..., 0.04516602, 0.13549805,
       0.50073242])
```

```
fig = plt.figure()
plt.scatter(y_test_predict, y_test)
plt.legend()
plt.show()
```

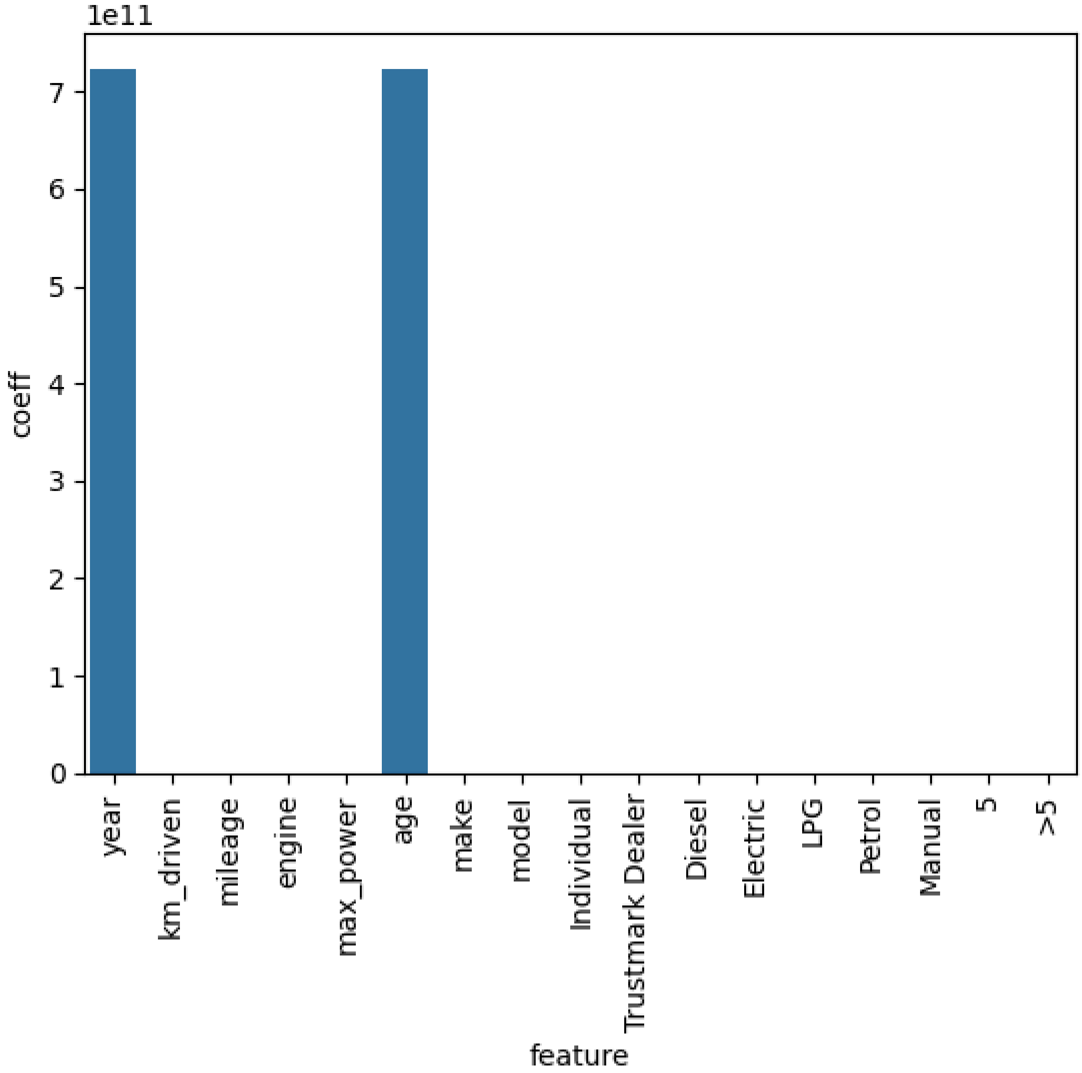
```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that a
```



Double-click (or enter) to edit

```
import seaborn as sns

imp = pd.DataFrame(list(zip(x_test.columns, np.abs(model.coef_))), columns = ['feature',
# Zip command is used to pack different data types together
sns.barplot(x = 'feature', y = 'coeff', data = imp)
plt.xticks(rotation = 90)
```

```
([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16],
 [Text(0, 0, 'year'),
  Text(1, 0, 'km_driven'),
  Text(2, 0, 'mileage'),
  Text(3, 0, 'engine'),
  Text(4, 0, 'max_power'),
  Text(5, 0, 'age'),
  Text(6, 0, 'make'),
  Text(7, 0, 'model'),
  Text(8, 0, 'Individual'),
  Text(9, 0, 'Trustmark Dealer'),
  Text(10, 0, 'Diesel'),
  Text(11, 0, 'Electric'),
  Text(12, 0, 'LPG'),
  Text(13, 0, 'Petrol'),
  Text(14, 0, 'Manual'),
  Text(15, 0, '5'),
  Text(16, 0, '>5')])
```



LAB-6

```
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
```

```
df = pd.read_csv("Housing_2.csv")
```

```
df.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population
0	-122.23	37.88	41.0	880.0	129.0	322.0
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0
2	-122.24	37.85	52.0	1467.0	190.0	496.0
3	-122.25	37.85	52.0	1274.0	235.0	558.0
4	-122.25	37.85	52.0	1627.0	280.0	565.0

```
df["ocean_proximity"] = df.groupby("ocean_proximity")["median_house_value"].transform("me
```

```
df.head()
```

Show hidden output

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

Show hidden output

```
df1.fillna(999,inplace = True)  # inplace = true makes the changes Permanent
```

```
y = df1["median_house_value"]
x = df1.drop("median_house_value", axis = 1)
```

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

```
linear_model = LinearRegression()
lasso_model = Lasso(alpha = 0.1)
ridge_model = Ridge(alpha = 0.1)
# Alpha is the regularisation parameter which we are adding as an error term to our model
# We are not taking the errors added in the feature we are only teking the regularization
```

```
linear_model.fit(x_train, y_train)
```

```
LinearRegression
LinearRegression()
```

```
lasso_model.fit(x_train, y_train)
```

```
Lasso
Lasso(alpha=0.1)
```

```
ridge_model.fit(x_train, y_train)
```

```
Ridge
Ridge(alpha=0.1)
```

```
print(linear_model.coef_)
print(lasso_model.coef_)
print(ridge_model.coef_)

print(linear_model.intercept_)
print(lasso_model.intercept_)
print(ridge_model.intercept_)
```

```
[-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]
[-0. -0.  0.  0.  0. -0.  0.  0.  0.]
[-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]
0.40752917375286374
0.3972234099154518
0.4050122758456162
```

```
linear_model.score(x_test, y_test)
```

```
0.6363169885864803
```

```
lasso_model.score(x_test, y_test)
```

```
-0.0005372966032284321
```

```
ridge_model.score(x_test, y_test)
```

```
0.636010311671446
```

```
linear_train_mse = mean_squared_error(y_train,linear_model.predict(x_train))
linear_test_mse = mean_squared_error(y_test,linear_model.predict(x_test))
```

LAB-7

```
# Prepare a decision tree model using the GINI Index as the criteria on the iris.csv dataset

from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier,plot_tree
from sklearn.metrics import accuracy_score

import pandas as pd
import matplotlib.pyplot as plt
```

```
df = pd.read_csv('Iris.csv')
```

```
df.head()
```

Show hidden output

Next steps: View recommended plots New interactive sheet

```
df.info()
```

Show hidden output

```
x = df.drop(['Species','Id'],axis = 1)
```

```
y = df['Species']
```

```
# If we do not specify the criteria for splitting the root node, by default the criteria used is ENTROPY
model = DecisionTreeClassifier(criterion='gini')
model
```

```
DecisionTreeClassifier
DecisionTreeClassifier()
```

```
# initialising a dictionary to hold gini impurities for each feature
gini_impurities = {}
```

```
# NOT A PART OF THE DESISION MODEL TREE
import numpy as np

# Original array
arr = np.array([1,2,3,4,5,6])
print('Original array shape: ', arr.shape)

# Reshape array
# Reshape cmd is used to reshape the numpy array or we can convert a 1-D array to a multi dimensional array
reshaped_arr = arr.reshape(3, 2)
print('Reshaped array shape: ', reshaped_arr.shape)
print(reshaped_arr)

# While reshaping a 1-D array, make sure that the no.of row and columns shall be the factor of total number of element
```

```
Original array shape:  (6,)
Reshaped array shape:  (3, 2)
[[1 2]
 [3 4]
 [5 6]]
```

```
x
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

Next steps: View recommended plots New interactive sheet

```
x.shape[1]
```

```
4
```

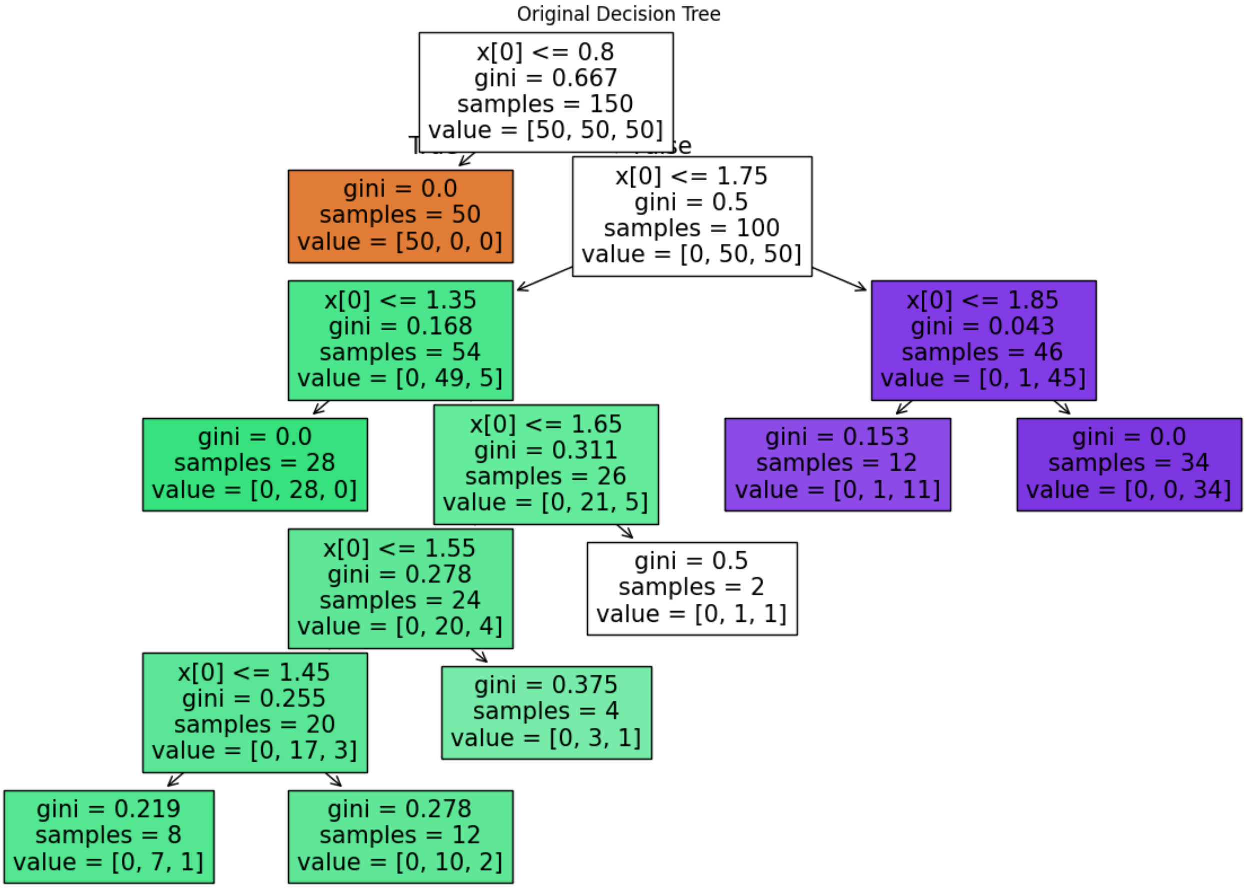
```
# Loop through each feature column
for i in range(x.shape[1]):
    # Fit the classifier with the current feature only
    model = model.fit(x.iloc[:,i].values.reshape(-1,1),y)
    prob = model.predict_proba(x.iloc[:, i].values.reshape(-1, 1))
    gini_impurities[i] = 1 - ((prob[:, 0]**2 + prob[:, 1]**2 + prob[:, 2]**2).sum())

gini_impurities

# when we are writing the simple colon(:) then it takes the entire row or entire column
```

```
{0: -101.0888888888889,
 1: -78.45482295482296,
 2: -139.6,
 3: -139.58333333333331}
```

```
# Find the feature with the lowest gini impurity
best_feature = min(gini_impurities, key = gini_impurities.get)
print(f"Best Feature: {best_feature}")
```

```
Best Feature: 2
```

```
plt.figure(figsize = (15,10))
plot_tree(model, filled = True)
plt.title("Original Decision Tree")
plt.show()
```


Original Decision Tree
x[0] <= 0.8
gini = 0.667
samples = 150
value = [50, 50, 50]
True
False
gini = 0.0
samples = 50
value = [50, 0, 0]
x[0] <= 1.75
gini = 0.5
samples = 100
value = [0, 50, 50]
x[0] <= 1.35
gini = 0.168
samples = 54
value = [0, 49, 5]
x[0] <= 1.85
gini = 0.043
samples = 46
value = [0, 1, 45]
gini = 0.0
samples = 28
value = [0, 28, 0]
x[0] <= 1.65
gini = 0.311
samples = 26
value = [0, 21, 5]
gini = 0.153
samples = 12
value = [0, 1, 11]
gini = 0.0
samples = 34
value = [0, 0, 34]
x[0] <= 1.55
gini = 0.278
samples = 24
value = [0, 20, 4]
gini = 0.5
samples = 2
value = [0, 1, 1]
x[0] <= 1.45
gini = 0.255
samples = 20
value = [0, 17, 3]
gini = 0.375
samples = 4
value = [0, 3, 1]
gini = 0.219
samples = 8
value = [0, 7, 1]
gini = 0.278
samples = 12
value = [0, 10, 2]

LAB-8

```
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
```

```
df = pd.read_csv('Iris.csv')
```

```
x = df.drop(['Species','Id'],axis = 1)
```

```
y = df['Species']
```

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

```
# build decision tree
model = tree.DecisionTreeClassifier(criterion = 'entropy', max_depth = 4)

# max_depth represents max level allowed in each tree
```

```
# fit the tree to iris dataset
model.fit(x_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

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

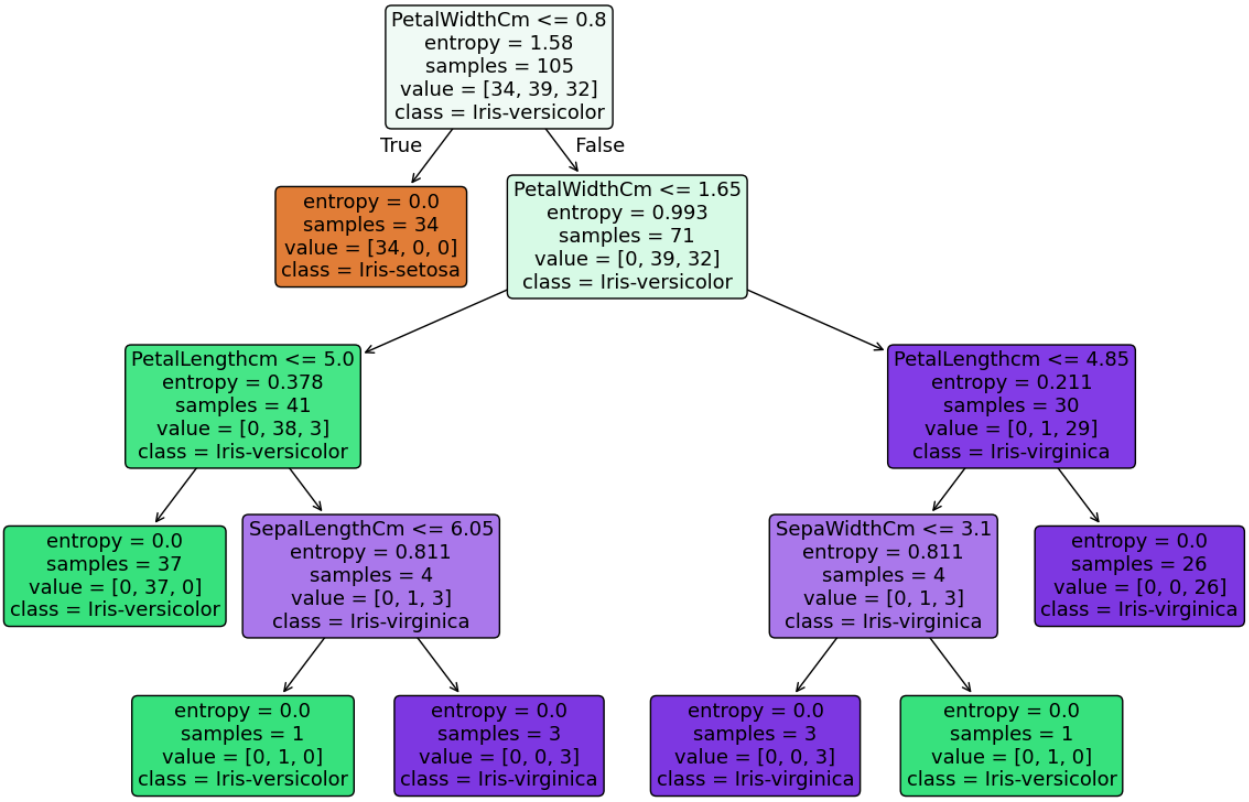
```
print("Accuracy: ", accuracy_score(y_test, y_pred)*100)
```

```
Accuracy:  95.55555555555556
```

+ Code + Text

```
# Function to plot the decision tree
def plot_decision_tree(model, feature_names, class_names):
  plt.figure(figsize = (15, 10))
  plot_tree(model, filled = True, feature_names = feature_names, class_names = cl
  plt.show()
```

```
plot_decision_tree(model, ['SepalLengthCm', 'SepaWidthCm', 'PetalLengthcm', 'Peta
```


PetalWidthCm <= 0.8
entropy = 1.58
samples = 105
value = [34, 39, 32]
class = Iris-versicolor
True
False
entropy = 0.0
samples = 34
value = [34, 0, 0]
class = Iris-setosa
PetalWidthCm <= 1.65
entropy = 0.993
samples = 71
value = [0, 39, 32]
class = Iris-versicolor
PetalLengthcm <= 5.0
entropy = 0.378
samples = 41
value = [0, 38, 3]
class = Iris-versicolor
PetalLengthcm <= 4.85
entropy = 0.211
samples = 30
value = [0, 1, 29]
class = Iris-virginica
entropy = 0.0
samples = 37
value = [0, 37, 0]
class = Iris-versicolor
SepalLengthCm <= 6.05
entropy = 0.811
samples = 4
value = [0, 1, 3]
class = Iris-virginica
SepaWidthCm <= 3.1
entropy = 0.811
samples = 4
value = [0, 1, 3]
class = Iris-virginica
entropy = 0.0
samples = 26
value = [0, 0, 26]
class = Iris-virginica
entropy = 0.0
samples = 1
value = [0, 1, 0]
class = Iris-versicolor
entropy = 0.0
samples = 3
value = [0, 0, 3]
class = Iris-virginica
entropy = 0.0
samples = 3
value = [0, 0, 3]
class = Iris-virginica
entropy = 0.0
samples = 1
value = [0, 1, 0]
class = Iris-versicolor

LAB-9

```
# Prepare a Naive Bayes classification model for predicting the purchase power of
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
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 accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
```

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

```
df.head()
```

Show hidden output

```
df.drop(columns = ['User ID'],axis = 1, inplace = True)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   User ID          400 non-null    int64
 1   Gender           400 non-null    object
 2   Age              400 non-null    int64
 3   EstimatedSalary  400 non-null    int64
 4   Purchased        400 non-null    int64
dtypes: int64(4), object(1)
memory usage: 15.8+ KB
```

```
# Label Encoding (Also known as One hot Encoding)
le = LabelEncoder()
df['Gender'] = le.fit_transform(df['Gender'])

# Label Encoder is a class which is used to convert a categorical variable into numerical
# Since a ML model is a mathematical model, so it understands only numerical values
```

```
# Split data into dependant/independent variable
x = df.iloc[:, :-1].values
y = df.iloc[:, -1].values


# Split data into test/train set
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.25, random_state =


# Scale dataset
sc = StandardScaler()
X_train = sc.fit_transform(x_train)
X_test = sc.transform(x_test)


# Classifier
classifier = GaussianNB()
classifier.fit(X_train, y_train)
```

```
GaussianNB

GaussianNB()
```

```
# Prediction
y_pred = classifier.predict(X_test)
# Accuracy
accuracy_score(y_test, y_pred)
```

```
0.87
```

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

```
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
```

```
# Confusion matrix
cf_matrix = confusion_matrix(y_test, y_pred)
print(cf_matrix)
```

```
[[51  7]
 [ 6 36]]
```

LAB-10

```
# Prepare a multinomial Naive Bayes classification model for email classification
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB, MultinomialNB
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import accuracy_score, f1_score
from wordcloud import WordCloud
# A wordcloud in python is a visual reresentation of text data that uses --
# --the size and colour of words to show their sequences
```

```
# Latin-1 encoding is use for assigning a unique numerical --
# --values to each charater which include ,many non ascai characters as well

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

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

Show hidden output

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

```
df.head()
```

Show hidden output

```
x = df['text']
y = df['label']
```

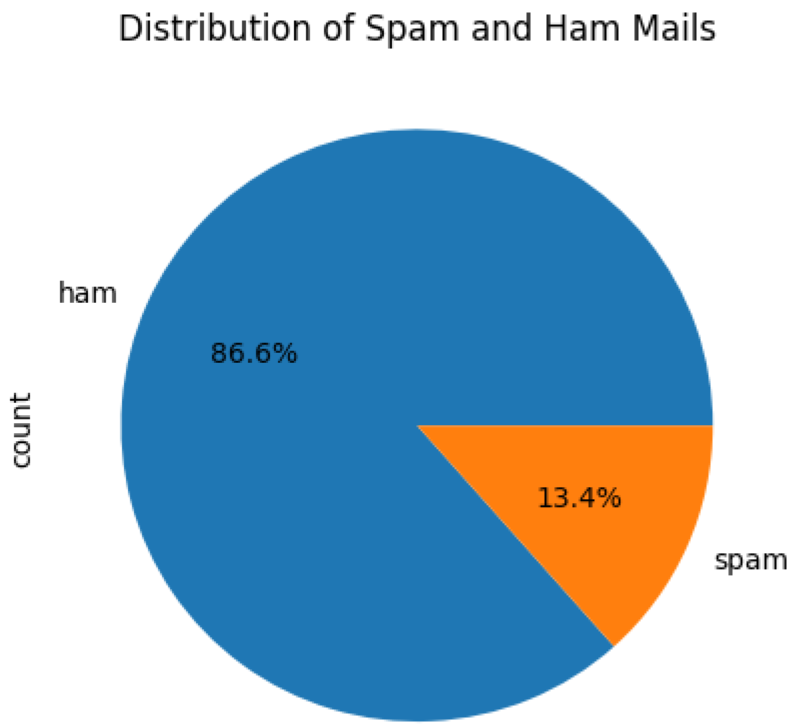
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state =
```

```
# Value_counts is used to count the numberof unique values in a dataset
distribution = y.value_counts()
distribution
```

	count
label	
ham	4825
spam	747

dtype: int64

```
distribution.plot(kind = 'pie', autopct = '%1.1f%%')
plt.title("Distribution of Spam and Ham Mails")
plt.show()
```




```
# Generate Word Cloud for Spam Mails
spam_text = ''.join(df[df['label'] == 'spam']['text'])
spam_text
```

```
'Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 8712
1 to receive entry question(std txt rate)T&C\'s apply 08452810075over18\'sFreeMsg He
y there darling it\'s been 3 week\'s now and no word back! I\'d like some fun you up
for it still? Tb ok! XxX std chgs to send, å£1.50 to rcvWINNER!! As a valued network
customer you have been selected to receivea å£900 prize reward! To claim call 090617
01461. Claim code KL341. Valid 12 hours only.Had your mobile 11 months or more? U R
entitled to Update to the latest colour mobiles with camera for Free! Call The Mobil
e Update Co FREE on 08002986030SIX chances to win CASH! From 100 to 20,000 pounds tx
t> CSH11 and send to 87575. Cost 150p/day, 6days, 16+ TsandCs apply Reply HL 4 infoU
RGENT! You have won a 1 week FREE membership in our å£100,000 Prize Jackpot! Txt the
word: CLAIM to No: 81010 T&C www.dbuk.net LCCLTD POBOX 4403LDNW1A7RW18XXXMobileMovie
Club: To use your credit, click the WAP link in the next txt message or cl…'
```

```
spam_wordcloud = WordCloud(width = 800, height = 400, max_words = 100, background_color =
```

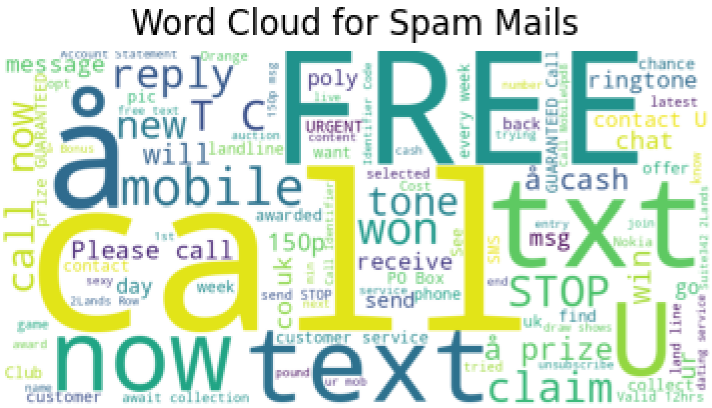
```
<wordcloud.wordcloud.WordCloud at 0x795c061fca90>
```

```
spam_wordcloud
```

```
<wordcloud.wordcloud.WordCloud at 0x795c061fca90>
```

```
# Plot the wordcloud for spam mails
plt.figure(figsize = (10,4))
plt.subplot(1, 2, 1)
plt.imshow(spam_wordcloud)
plt.title("Word Cloud for Spam Mails")
plt.axis("off")
```

```
(-0.5, 799.5, 399.5, -0.5)
```




```
# Generate Word cloud for Ham Mails
ham_text = ' '.join(df[df['label'] == 'ham']['text'])
ham_text
```

Show hidden output

```
ham_wordcloud = WordCloud(width = 800, height = 400, max_words = 100, background_color =
```

```
# Plot the wordcloud for ham mails
plt.subplot(1, 2, 2)
plt.imshow(ham_wordcloud)
plt.title("Word Cloud for Ham Mails")
plt.axis("off")
```

Show hidden output

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

```
# CountVectorizer is a text processing technique used in
# natural language processing task for converting a collection of text document into a nu
```

```
# Train a Multinomial Naive Bayes classifier
model_multinomial = MultinomialNB(alpha = 0.8, fit_prior = True, force_alpha = True)
model_multinomial.fit(x_train, y_train)
```

```
▾ MultinomialNB
MultinomialNB(alpha=0.8)
```

```
# Train a Guassian Naive Bayes classifier

model_gaussian = GaussianNB()
model_gaussian.fit(x_train.toarray(), y_train)
```

```
▾ GaussianNB
GaussianNB()
```

```
# Calculating the Accuracy
y_pred_multinomial = model_multinomial.predict(x_test)
accuracy_multinomial = accuracy_score(y_test, y_pred_multinomial)
print("Accuracy for multinomial model is : ", accuracy_multinomial)
```

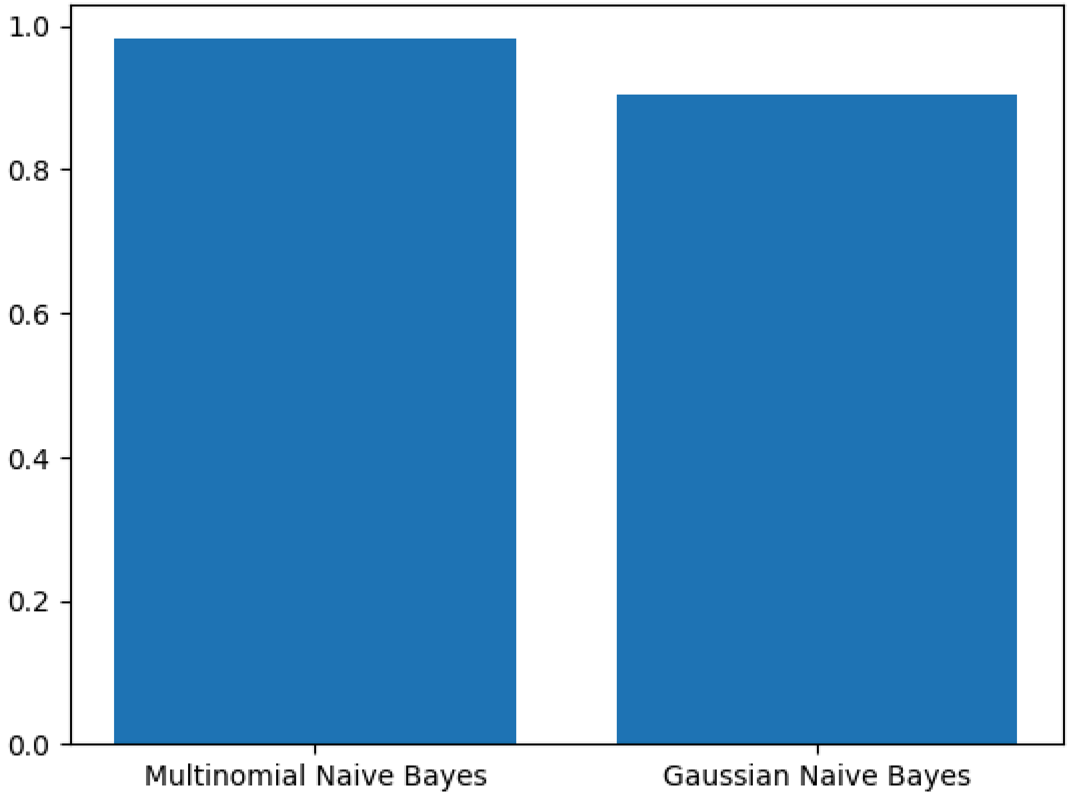
```
Accuracy for multinomial model is :  0.9811659192825112
```

```
y_pred_gaussian = model_gaussian.predict(x_test.toarray())
accuracy_gaussian = accuracy_score(y_test, y_pred_gaussian)
print("Accuracy for Gaussian model is : ",accuracy_gaussian)
```

```
Accuracy for Gaussian model is :  0.9031390134529148
```

```
methods = ["Multinomial Naive Bayes", "Gaussian Naive Bayes"]
scores = [accuracy_multinomial, accuracy_gaussian]
plt.bar(methods, scores)
```

```
<BarContainer object of 2 artists>
```



LAB-11

LAB-11

```
# Prepare a model for predictioon of prostate cancer using KNN Classifier
```

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
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
```

```
df = pd.read_csv('prostate.csv')
df
```

Show hidden output

```
df.shape
```

```
(97, 9)
```

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

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

Show hidden output

```
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)
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=1)
```

```
y_pred = knn_model.predict(x_test)
```

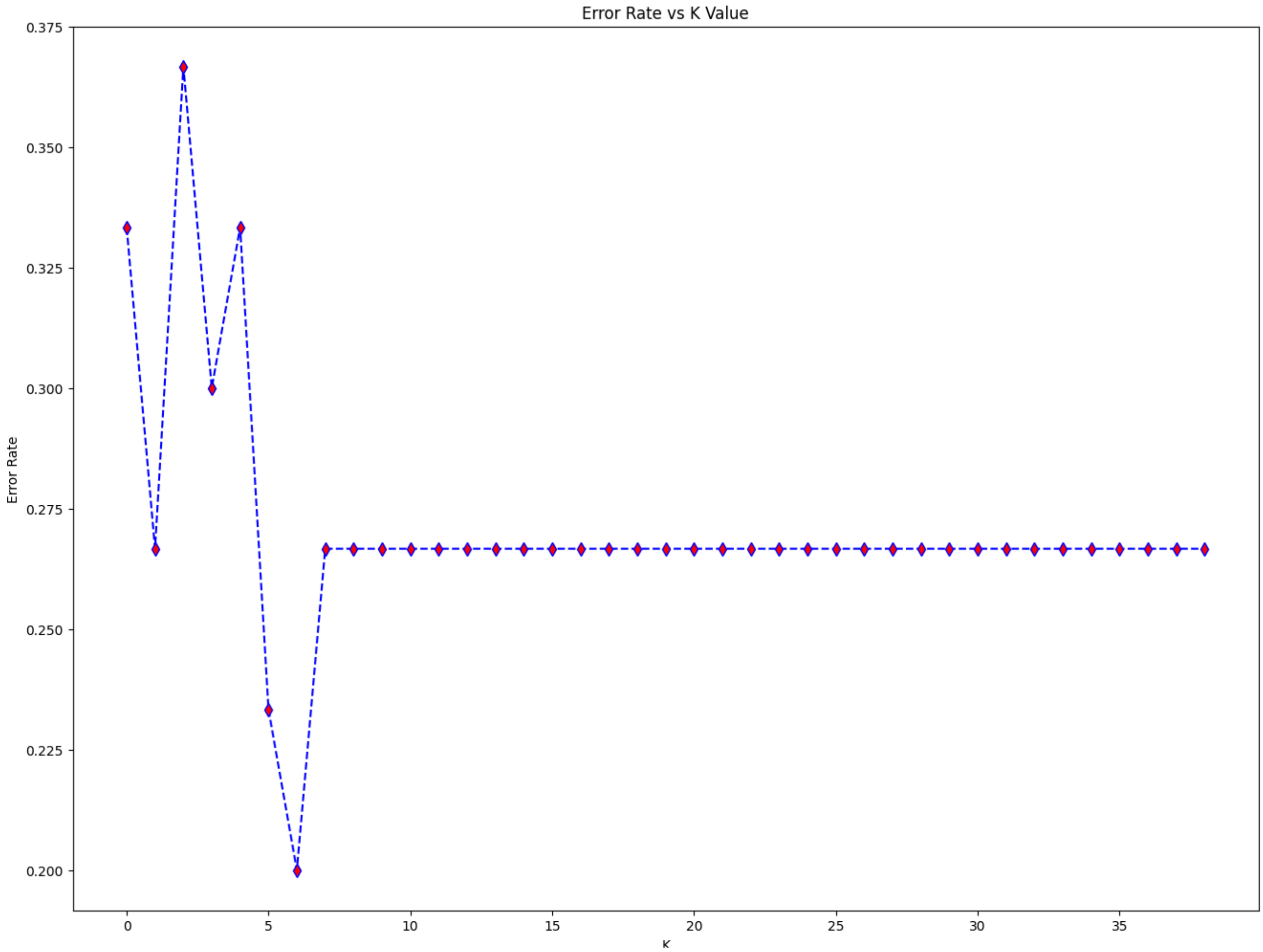
```
print(confusion_matrix(y_test, y_pred))
```

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

```
print(classification_report(y_test, y_pred))
```

Show hidden output

```
# Elbow method for calculating 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))

plt.figure(figsize = (16, 12))
plt.plot(error_rate, color = 'Blue', linestyle = 'dashed', marker = 'd', markerfacecolor
plt.title('Error Rate vs K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
plt.show()
```



LAB-12

```
# Prepare a model for prediction of survival from Titanic Ship using Random Fores
# and compare the accuracy with other classifiers too
```

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score,classification_report
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import LabelEncoder
```

```
df = pd.read_csv("Titanic-Dataset.csv")
df
```

Show hidden output

```
df.info()
```

Show hidden output

```
# dropping those rows where target variable is missing
df = df.dropna(subset = ['Survived'])
df.shape
```

```
(891, 12)
```

```
x = df[['Pclass','Sex','Age','SibSp','Parch','Fare']]
y = df['Survived']
```

```
le = LabelEncoder()
x['Sex'] = le.fit_transform(x['Sex'])
x.head()
```

Show hidden output

```
x['Age'] = x['Age'].fillna(x['Age'].mean())
x['Age']
```

Show hidden output

```
x_train , x_test , y_train , y_test = train_test_split(x, y, random_state = 42 , test_siz
```

```
# create a random forest classifier
# n_estimators = 100 decision Trees
```

```
model = RandomForestClassifier(n_estimators = 100 , random_state = 42)

# train the classifier
model.fit(x_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
# make Predictions on the test set
y_pred = model.predict(x_test)

#evaluate the model
accuracy = accuracy_score(y_test,y_pred)
classification_report = classification_report(y_test , y_pred)
accuracy
```

```
0.8156424581005587
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

```
model1 = KNeighborsClassifier(n_neighbors = 9)
model2 = GaussianNB()
model3 = DecisionTreeClassifier(criterion = "entropy")
model4 = RandomForestClassifier(n_estimators = 100)
modelList = [model1,model2,model3,model4]
```

```
model1
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=9)
```

```
model2
```

```
GaussianNB
GaussianNB()
```

```
model3
```

```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy')
```

```
model4
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

```
from sklearn.metrics import confusion_matrix, accuracy_score, classification_repc

for i in modelList:
  i.fit(x_train, y_train)
  y_pred  = i.predict(x_test)
  print('the classification details of model', i, 'is below')
  print('the confusion matrix of ', i, 'is')
  print(confusion_matrix(y_test, y_pred))
  print('accuracy score of ', i, 'is')
  print(accuracy_score(y_test, y_pred))
  print('the classification report of ', i, 'is')
  print(classification_report(y_test, y_pred))
```

```
the classification details of model KNeighborsClassifier(n_neighbors=9) is below
the confusion matrix of  KNeighborsClassifier(n_neighbors=9) is
[[85 20]
 [34 40]]
accuracy score of  KNeighborsClassifier(n_neighbors=9) is
0.6983240223463687
the classification report of  KNeighborsClassifier(n_neighbors=9) is
              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

the classification details of model GaussianNB() is below
the confusion matrix of  GaussianNB() is
[[85 20]
 [21 53]]
accuracy score of  GaussianNB() is
0.770949720670391
the classification report of  GaussianNB() is
              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

the classification details of model DecisionTreeClassifier(criterion='entropy') is
the confusion matrix of  DecisionTreeClassifier(criterion='entropy') is
[[84 21]
 [21 53]]
accuracy score of  DecisionTreeClassifier(criterion='entropy') is
0.7653631284916201
the classification report of  DecisionTreeClassifier(criterion='entropy') is
```

```
              precision    recall  f1-score   support

           0       0.80      0.80      0.80       105
           1       0.72      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

the classification details of model RandomForestClassifier() is below
the confusion matrix of  RandomForestClassifier() is
[[88 17]
 [20 54]]
accuracy score of  RandomForestClassifier() is
0.7932960893854749
the classification report of  RandomForestClassifier() is
              precision    recall  f1-score   support
```