MANAV RACHNA UNIVERSITY

SCHOOL OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE & TECHNOLOGY

LAB FILE

Supervised Learning (CSH212B-P/ CSH212B-T)

Submitted to:

Dr. ROSHI SAXENA

Manager - Xebia Academy

Submitted by:

Devanaboina Tharun

2K23CSUN01247 AIML-3B

Supervised Learning Projects: - 3B

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.	

[redacted]

LAB-01

```
#question-1)Write a python code to demonstrate commands for numpy an
import numpy as np
import pandas as pd

# NumPy
arr = np.array([1, 2, 3, 4, 5])
matrix = np.array([[1, 2, 3], [4, 5, 6]])
mean = np.mean(arr)
std_dev = np.std(arr)

print("NumPy  Array:",  arr)  print("Matrix:\n",
matrix)   print("Mean   of   Array:",   mean)
print("Standard Deviation of Array:", std_dev)



# Pandas
data = pd.Series([10, 20, 30, 40], index=['a', 'b', 'c', 'd'])
df = pd.DataFrame({'Name': ['Alice', 'Bob', 'Charlie'], 'Age': [25,
df['Pass'] = df['Score'] > 50

print("\nPandas Series:\n", data)
print("\nPandas DataFrame:\n", df)
```

```
NumPy Array: [1 2 3 4 5] Matrix:  [[1 2 3]  [4
5 6]]
Mean of Array: 3.0
Standard Deviation of Array: 1.4142135623730951



Pandas Series:
 a    10
b    20
c    30
d    40
dtype: int64

Pandas DataFrame:
       Name Age Score Pass
0    Alice   25     85 True
1      Bob   30     90 True
2 Charlie    35     95 True
```

LAB-02

```
#question-2)Write a python program to calculate mean square and mean
def calculate_mse_and_mae(actual, predicted):
    n = len(actual)


    sum_squared_error = 0
    sum_absolute_error = 0
```

```
    for i in range(n):
        sum_squared_error += (actual[i] - predicted[i]) ** 2
        sum_absolute_error += abs(actual[i] - predicted[i])


    mse = sum_squared_error / n
    mae = sum_absolute_error / n

    return mse, mae


actual = [3, -0.5, 2, 7]
predicted = [2.5, 0.0, 2, 8]

mse, mae = calculate_mse_and_mae(actual, predicted)
print(f"Mean Squared Error: {mse}")
print(f"Mean Absolute Error: {mae}")
```

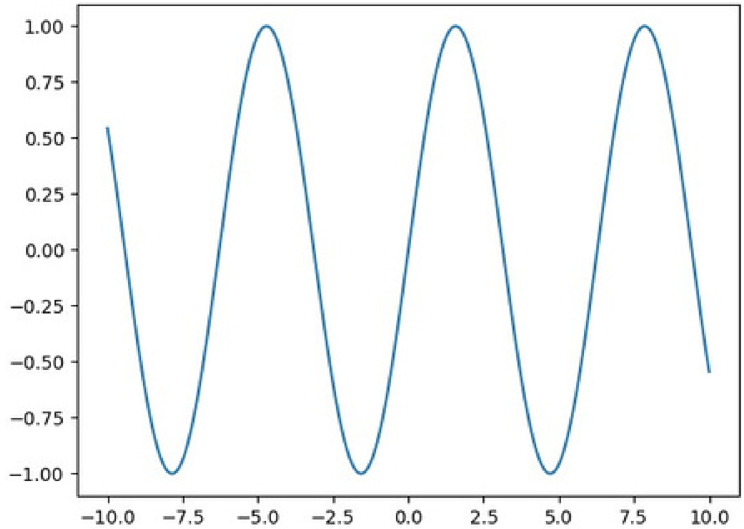
LAB-03

```
#question-3)Write a python program to calculate gradient descent of
import numpy as np
import matplotlib.pyplot as plt

def plot_function(func):
    x_values  =  np.linspace(-10,  10,  1000)
    y_values = [func(x) for x in x_values]
    print("x   values   are:   ",   x_values)
    print("y   values   are:   ",   y_values)
    plt.plot(x_values,            y_values)
    plt.show()
plot_function(lambda x: np.sin(x))
```

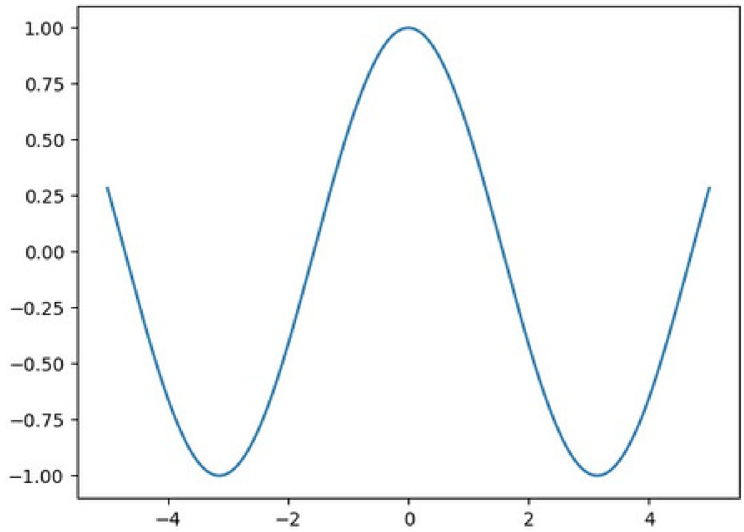
```
x values are: [-10.          -9.97997998 -9.95995996 -9.93993994 -9.91991992
  -9.8998999   -9.87987988 -9.85985986 -9.83983984 -9.81981982
  -9.7997998   -9.77977978 -9.75975976 -9.73973974 -9.71971972
  -9.6996997   -9.67967968 -9.65965966 -9.63963964 -9.61961962
  -9.5995996   -9.57957958 -9.55955956 -9.53953954 -9.51951952
  -9.4994995   -9.47947948 -9.45945946 -9.43943944 -9.41941942
  -9.3993994   -9.37937938 -9.35935936 -9.33933934 -9.31931932
  -9.2992993   -9.27927928 -9.25925926 -9.23923924 -9.21921922
  -9.1991992   -9.17917918 -9.15915916 -9.13913914 -9.11911912
  -9.0990991   -9.07907908 -9.05905906 -9.03903904 -9.01901902
  -8.998999    -8.97897898 -8.95895896 -8.93893894 -8.91891892
  -8.8988989   -8.87887888 -8.85885886 -8.83883884 -8.81881882
  -8.7987988   -8.77877878 -8.75875876 -8.73873874 -8.71871872
  -8.6986987   -8.67867868 -8.65865866 -8.63863864 -8.61861862
  -8.5985986   -8.57857858 -8.55855856 -8.53853854 -8.51851852
  -8.4984985   -8.47847848 -8.45845846 -8.43843844 -8.41841842
  -8.3983984   -8.37837838 -8.35835836 -8.33833834 -8.31831832
  -8.2982983   -8.27827828 -8.25825826 -8.23823824 -8.21821822
  -8.1981982   -8.17817818 -8.15815816 -8.13813814 -8.11811812
  -8.0980981   -8.07807808 -8.05805806 -8.03803804 -8.01801802
  -7.997998    -7.97797798 -7.95795796 -7.93793794 -7.91791792
  -7.8978979   -7.87787788 -7.85785786 -7.83783784 -7.81781782
  -7.7977978   -7.77777778 -7.75775776 -7.73773774 -7.71771772
  -7.6976977   -7.67767768 -7.65765766 -7.63763764 -7.61761762
  -7.5975976   -7.57757758 -7.55755756 -7.53753754 -7.51751752
  -7.4974975   -7.47747748 -7.45745746 -7.43743744 -7.41741742
  -7.3973974   -7.37737738 -7.35735736 -7.33733734 -7.31731732
  -7.2972973   -7.27727728 -7.25725726 -7.23723724 -7.21721722
  -7.1971972   -7.17717718 -7.15715716 -7.13713714 -7.11711712
  -7.0970971   -7.07707708 -7.05705706 -7.03703704 -7.01701702
  -6.996997    -6.97697698 -6.95695696 -6.93693694 -6.91691692
  -6.8968969   -6.87687688 -6.85685686 -6.83683684 -6.81681682
  -6.7967968   -6.77677678 -6.75675676 -6.73673674 -6.71671672
  -6.6966967   -6.67667668 -6.65665666 -6.63663664 -6.61661662
  -6.5965966   -6.57657658 -6.55655656 -6.53653654 -6.51651652
  -6.4964965   -6.47647648 -6.45645646 -6.43643644 -6.41641642
  -6.3963964   -6.37637638 -6.35635636 -6.33633634 -6.31631632
  -6.2962963   -6.27627628 -6.25625626 -6.23623624 -6.21621622
  -6.1961962   -6.17617618 -6.15615616 -6.13613614 -6.11611612
  -6.0960961   -6.07607608 -6.05605606 -6.03603604 -6.01601602
  -5.995996    -5.97597598 -5.95595596 -5.93593594 -5.91591592
  -5.8958959   -5.87587588 -5.85585586 -5.83583584 -5.81581582
  -5.7957958   -5.77577578 -5.75575576 -5.73573574 -5.71571572
  -5.6956957   -5.67567568 -5.65565566 -5.63563564 -5.61561562
  -5.5955956   -5.57557558 -5.55555556 -5.53553554 -5.51551552
  -5.4954955   -5.47547548 -5.45545546 -5.43543544 -5.41541542
  -5.3953954   -5.37537538 -5.35535536 -5.33533534 -5.31531532
  -5.2952953   -5.27527528 -5.25525526 -5.23523524 -5.21521522
  -5.1951952   -5.17517518 -5.15515516 -5.13513514 -5.11511512
  -5.0950951   -5.07507508 -5.05505506 -5.03503504 -5.01501502
  -4.99499499 -4.97497497 -4.95495495 -4.93493493 -4.91491491
  -4.89489489 -4.87487487 -4.85485485 -4.83483483 -4.81481481
  -4.79479479 -4.77477477 -4.75475475 -4.73473473 -4.71471471
  -4.69469469 -4.67467467 -4.65465465 -4.63463463 -4.61461461
  -4.59459459 -4.57457457 -4.55455455 -4.53453453 -4.51451451
  -4.49449449 -4.47447447 -4.45445445 -4.43443443 -4.41441441
  -4.39439439 -4.37437437 -4.35435435 -4.33433433 -4.31431431
  -4.29429429 -4.27427427 -4.25425425 -4.23423423 -4.21421421
  -4.19419419 -4.17417417 -4.15415415 -4.13413413 -4.11411411
  -4.09409409 -4.07407407 -4.05405405 -4.03403403 -4.01401401
  -3.99399399 -3.97397397 -3.95395395 -3.93393393 -3.91391391
  -3.89389389 -3.87387387 -3.85385385 -3.83383383 -3.81381381
  -3.79379379 -3.77377377 -3.75375375 -3.73373373 -3.71371371
  -3.69369369 -3.67367367 -3.65365365 -3.63363363 -3.61361361
  -3.59359359 -3.57357357 -3.55355355 -3.53353353 -3.51351351
  -3.49349349 -3.47347347 -3.45345345 -3.43343343 -3.41341341
  -3.39339339 -3.37337337 -3.35335335 -3.33333333 -3.31331331
  -3.29329329 -3.27327327 -3.25325325 -3.23323323 -3.21321321
  -3.19319319 -3.17317317 -3.15315315 -3.13313313 -3.11311311
  -3.09309309 -3.07307307 -3.05305305 -3.03303303 -3.01301301
  -2.99299299 -2.97297297 -2.95295295 -2.93293293 -2.91291291
  -2.89289289 -2.87287287 -2.85285285 -2.83283283 -2.81281281
  -2.79279279 -2.77277277 -2.75275275 -2.73273273 -2.71271271
  -2.69269269 -2.67267267 -2.65265265 -2.63263263 -2.61261261
  -2.59259259 -2.57257257 -2.55255255 -2.53253253 -2.51251251
  -2.49249249 -2.47247247 -2.45245245 -2.43243243 -2.41241241
  -2.39239239 -2.37237237 -2.35235235 -2.33233233 -2.31231231
  -2.29229229 -2.27227227 -2.25225225 -2.23223223 -2.21221221
  -2.19219219 -2.17217217 -2.15215215 -2.13213213 -2.11211211
  -2.09209209 -2.07207207 -2.05205205 -2.03203203 -2.01201201
  -1.99199199 -1.97197197 -1.95195195 -1.93193193 -1.91191191
  -1.89189189 -1.87187187 -1.85185185 -1.83183183 -1.81181181
  -1.79179179 -1.77177177 -1.75175175 -1.73173173 -1.71171171
  -1.69169169 -1.67167167 -1.65165165 -1.63163163 -1.61161161
  -1.59159159 -1.57157157 -1.55155155 -1.53153153 -1.51151151
  -1.49149149 -1.47147147 -1.45145145 -1.43143143 -1.41141141
  -1.39139139 -1.37137137 -1.35135135 -1.33133133 -1.31131131
  -1.29129129 -1.27127127 -1.25125125 -1.23123123 -1.21121121
  -1.19119119 -1.17117117 -1.15115115 -1.13113113 -1.11111111
  -1.09109109 -1.07107107 -1.05105105 -1.03103103 -1.01101101
```

-0.99099099	-0.97097097	-0.95095095	-0.93093093	-0.91091091
-0.89089089	-0.87087087	-0.85085085	-0.83083083	-0.81081081
-0.79079079	-0.77077077	-0.75075075	-0.73073073	-0.71071071
-0.69069069	-0.67067067	-0.65065065	-0.63063063	-0.61061061
-0.59059059	-0.57057057	-0.55055055	-0.53053053	-0.51051051
-0.49049049	-0.47047047	-0.45045045	-0.43043043	-0.41041041
-0.39039039	-0.37037037	-0.35035035	-0.33033033	-0.31031031
-0.29029029	-0.27027027	-0.25025025	-0.23023023	-0.21021021
-0.19019019	-0.17017017	-0.15015015	-0.13013013	-0.11011011
-0.09009009	-0.07007007	-0.05005005	-0.03003003	-0.01001001
0.01001001	0.03003003	0.05005005	0.07007007	0.09009009
0.11011011	0.13013013	0.15015015	0.17017017	0.19019019
0.21021021	0.23023023	0.25025025	0.27027027	0.29029029
0.31031031	0.33033033	0.35035035	0.37037037	0.39039039
0.41041041	0.43043043	0.45045045	0.47047047	0.49049049
0.51051051	0.53053053	0.55055055	0.57057057	0.59059059
0.61061061	0.63063063	0.65065065	0.67067067	0.69069069
0.71071071	0.73073073	0.75075075	0.77077077	0.79079079
0.81081081	0.83083083	0.85085085	0.87087087	0.89089089
0.91091091	0.93093093	0.95095095	0.97097097	0.99099099
1.01101101	1.03103103	1.05105105	1.07107107	1.09109109
1.11111111	1.13113113	1.15115115	1.17117117	1.19119119
1.21121121	1.23123123	1.25125125	1.27127127	1.29129129
1.31131131	1.33133133	1.35135135	1.37137137	1.39139139
1.41141141	1.43143143	1.45145145	1.47147147	1.49149149
1.51151151	1.53153153	1.55155155	1.57157157	1.59159159
1.61161161	1.63163163	1.65165165	1.67167167	1.69169169
1.71171171	1.73173173	1.75175175	1.77177177	1.79179179
1.81181181	1.83183183	1.85185185	1.87187187	1.89189189
1.91191191	1.93193193	1.95195195	1.97197197	1.99199199
2.01201201	2.03203203	2.05205205	2.07207207	2.09209209
2.11211211	2.13213213	2.15215215	2.17217217	2.19219219
2.21221221	2.23223223	2.25225225	2.27227227	2.29229229
2.31231231	2.33233233	2.35235235	2.37237237	2.39239239
2.41241241	2.43243243	2.45245245	2.47247247	2.49249249
2.51251251	2.53253253	2.55255255	2.57257257	2.59259259
2.61261261	2.63263263	2.65265265	2.67267267	2.69269269
2.71271271	2.73273273	2.75275275	2.77277277	2.79279279
2.81281281	2.83283283	2.85285285	2.87287287	2.89289289
2.91291291	2.93293293	2.95295295	2.97297297	2.99299299
3.01301301	3.03303303	3.05305305	3.07307307	3.09309309
3.11311311	3.13313313	3.15315315	3.17317317	3.19319319
3.21321321	3.23323323	3.25325325	3.27327327	3.29329329
3.31331331	3.33333333	3.35335335	3.37337337	3.39339339
3.41341341	3.43343343	3.45345345	3.47347347	3.49349349
3.51351351	3.53353353	3.55355355	3.57357357	3.59359359
3.61361361	3.63363363	3.65365365	3.67367367	3.69369369
3.71371371	3.73373373	3.75375375	3.77377377	3.79379379
3.81381381	3.83383383	3.85385385	3.87387387	3.89389389
3.91391391	3.93393393	3.95395395	3.97397397	3.99399399
4.01401401	4.03403403	4.05405405	4.07407407	4.09409409
4.11411411	4.13413413	4.15415415	4.17417417	4.19419419
4.21421421	4.23423423	4.25425425	4.27427427	4.29429429
4.31431431	4.33433433	4.35435435	4.37437437	4.39439439
4.41441441	4.43443443	4.45445445	4.47447447	4.49449449
4.51451451	4.53453453	4.55455455	4.57457457	4.59459459
4.61461461	4.63463463	4.65465465	4.67467467	4.69469469
4.71471471	4.73473473	4.75475475	4.77477477	4.79479479
4.81481481	4.83483483	4.85485485	4.87487487	4.89489489
4.91491491	4.93493493	4.95495495	4.97497497	4.99499499
5.01501502	5.03503504	5.05505506	5.07507508	5.0950951
5.11511512	5.13513514	5.15515516	5.17517518	5.1951952
5.21521522	5.23523524	5.25525526	5.27527528	5.2952953
5.31531532	5.33533534	5.35535536	5.37537538	5.3953954
5.41541542	5.43543544	5.45545546	5.47547548	5.4954955
5.51551552	5.53553554	5.55555556	5.57557558	5.5955956
5.61561562	5.63563564	5.65565566	5.67567568	5.6956957
5.71571572	5.73573574	5.75575576	5.77577578	5.7957958
5.81581582	5.83583584	5.85585586	5.87587588	5.8958959
5.91591592	5.93593594	5.95595596	5.97597598	5.995996
6.01601602	6.03603604	6.05605606	6.07607608	6.0960961
6.11611612	6.13613614	6.15615616	6.17617618	6.1961962
6.21621622	6.23623624	6.25625626	6.27627628	6.2962963
6.31631632	6.33633634	6.35635636	6.37637638	6.3963964
6.41641642	6.43643644	6.45645646	6.47647648	6.4964965
6.51651652	6.53653654	6.55655656	6.57657658	6.5965966
6.61661662	6.63663664	6.65665666	6.67667668	6.6966967
6.71671672	6.73673674	6.75675676	6.77677678	6.7967968
6.81681682	6.83683684	6.85685686	6.87687688	6.8968969
6.91691692	6.93693694	6.95695696	6.97697698	6.996997
7.01701702	7.03703704	7.05705706	7.07707708	7.0970971
7.11711712	7.13713714	7.15715716	7.17717718	7.1971972
7.21721722	7.23723724	7.25725726	7.27727728	7.2972973
7.31731732	7.33733734	7.35735736	7.37737738	7.3973974
7.41741742	7.43743744	7.45745746	7.47747748	7.4974975
7.51751752	7.53753754	7.55755756	7.57757758	7.5975976
7.61761762	7.63763764	7.65765766	7.67767768	7.6976977
7.71771772	7.73773774	7.75775776	7.77777778	7.7977978
7.81781782	7.83783784	7.85785786	7.87787788	7.8978979
7.91791792	7.93793794	7.95795796	7.97797798	7.997998

```
 8.01801802   8.03803804   8.05805806   8.07807808   8.0980981
 8.11811812   8.13813814   8.15815816   8.17817818   8.1981982
 8.21821822   8.23823824   8.25825826   8.27827828   8.2982983
 8.31831832   8.33833834   8.35835836   8.37837838   8.3983984
 8.41841842   8.43843844   8.45845846   8.47847848   8.4984985
 8.51851852   8.53853854   8.55855856   8.57857858   8.5985986
 8.61861862   8.63863864   8.65865866   8.67867868   8.6986987
 8.71871872   8.73873874   8.75875876   8.77877878   8.7987988
 8.81881882   8.83883884   8.85885886   8.87887888   8.8988989
 8.91891892   8.93893894   8.95895896   8.97897898   8.998999
 9.01901902   9.03903904   9.05905906   9.07907908   9.0990991
 9.11911912   9.13913914   9.15915916   9.17917918   9.1991992
 9.21921922   9.23923924   9.25925926   9.27927928   9.2992993
 9.31931932   9.33933934   9.35935936   9.37937938   9.3993994
 9.41941942   9.43943944   9.45945946   9.47947948   9.4994995
 9.51951952   9.53953954   9.55955956   9.57957958   9.5995996
 9.61961962   9.63963964   9.65965966   9.67967968   9.6996997
 9.71971972   9.73973974   9.75975976   9.77977978   9.7997998
 9.81981982   9.83983984   9.85985986   9.87987988   9.8998999
 9.91991992   9.93993994   9.95995996   9.97997998 10.
                                                                  ]
y values are: [0.5440211108893698, 0.5271149856654235, 0.5099975991781289, 0.49267581186741427, 0.4751565660945675, 0.4574468833598
```



```
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)
    plt.show()

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




```
#try to find the minima or gradiant descent
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 - derivative
      list_of_weights.append(weight)
    return list_of_weights
gradient_descent(lambda x:x**2+4*x+3,10)
```

```
[-14.000099999739177,
 9.999999999431566,
 -14.000099999170743,
 9.999999999431566,
 -14.000099999170743,
 9.999999999431566,
 -14.000099999170743,
 9.999999999431566,
 -14.000099999170743,
 9.999999999431566,
 -14.000099999170743,
 9.999999999431566,
 -14.000099999170743,
 9.999999999431566,
 -14.000099999170743,
 9.999999999431566,
 -14.000099999170743,
 9.999999999431566,
 -14.000099999170743,
 9.999999999431566,
 -14.000099999170743,
 9.999999999431566,
 -14.000099999170743,
 9.999999999431566,
```

```
 -14.000099999170743,
9.999999999431566,
-14.000099999170743,
 9.999999999431566,
 -14.000099999170743,
9.999999999431566,
-14.000099999170743,
9.999999999431566,
 -14.000099999170743,
 9.999999999431566,
-14.000099999170743,
9.999999999431566,
-14.000099999170743,
9.999999999431566,
 -14.000099999170743,
 9.999999999431566,
-14.000099999170743,
9.999999999431566,
-14.000099999170743,
9.999999999431566,
 -14.000099999170743,
 9.999999999431566,
 -14.000099999170743,
 9.999999999431566,
 -14.000099999170743,
 9.999999999431566,
 -14.000099999170743,
 9.999999999431566,
 -14.000099999170743,
 9.999999999431566,
 -14.000099999170743,
 9.999999999431566,
-14.000099999170743,
9.999999999431566,
```

LAB-04

```
#question-4)Prepare a linear regression model for predicting the sal
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
df=pd.read_csv("salary_data.csv")
```

```
x=pd.DataFrame(df["Salary"])
y=df["YearsExperience"]
x,y
```

```
43525.0
39891.0
56642.0
60150.0
54445.0
64445.0
57189.0
63218.0
55794.0
56957.0
57081.0
61111.0
67938.0
66029.0
83088.0
81363.0
93940.0
91738.0
98273.0
 22 101302.0
  23  113812.0
24    109431.0
25    105582.0
26    116969.0
27 112635.0
 28 122391.0
 29 121872.0,

 0      1.1
```

```
5 2.9  6 3.0  7 3.2
8 3.2
9 3.7
10 3.9
11 4.0  12 4.0  13 4.1
14 4.5
15 4.9
16 5.1
17 5.3  18 5.9  19 6.0
20 6.8
21 7.1
22 7.9
23 8.2
24 8.7
25 9.0
26 9.5
27 9.6
28 10.3
29 10.5
Name: YearsExperience, dtype: float64)
```

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

```
  6   60150.0
18    81363.0
13    57081.0
7     54445.0
  27 112635.0
1     46205.0
16    66029.0
0     39343.0
15    67938.0
  29 121872.0
  28 122391.0
9      57189.0
8      64445.0
12     56957.0
11     55794.0
5     56642.0,
        Salary
 17    83088.0
21     98273.0
10     63218.0
19     93940.0
 14    61111.0
20     91738.0
   26 116969.0
3    43525.0
24 109431.0,
22     7.9
 23    8.2
4  2   2.2
25     1.5
 6     9.0
  18   3.0
13     5.9
7      4.1
27     3.2
1      9.6
 16    1.3
   0   5.1
15     1.1
29     4.9
28    10.5
9     10.3
 8     3.7
12     3.2
11     4.0
5      4.0
       2.9
Name: YearsExperience, dtype: float64,
17    5.3
21    7.1
```

```
24    8.7
Name: YearsExperience, dtype: float64)
```

```
from sklearn.linear_model import LinearRegression
model=LinearRegression()
model.fit(x_train,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
model.coef_
```

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

```
model.intercept_
```

```
-2.522510616511819
```

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

```
0.9242662549548135
```

```
x_pred=model.predict(x_train)
y_pred=model.predict(x_test)
x_pred,y_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]),
 array([6.15244094, 7.73785808, 4.07787798, 7.28546345, 3.85789287,
        7.0555597 , 9.68984747, 2.02179502, 8.90282907]))
```

```
plt.scatter(x_train,y_train,color="green")
plt.plot(x_train,x_pred,color="blue")
plt.ylabel("Years Of Experience")
plt.xlabel("Salary")
plt.title("Salary vs Experience")
plt.show()
```

LAB-05

```
#QUESTION-5)Prepare a linear regression model for prediction of resal
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
df3=pd.read_csv("/content/cars24-car-price-cleaned.csv")

df3.info()
df3.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19820 entries, 0 to 19819
Data columns (total 18 columns):
 #  Column Non-Null Count Dtype
--------------- -----
 0  selling_price 19820 non-null float64
 1  year 19820 non-null float64
 2  km_driven 19820 non-null int64
 3  mileage 19820 non-null float64
 4  engine 19820 non-null float64
 5  max_power 19820 non-null float64
 6  age 19820 non-null float64
 7  make 19820 non-null object
 8  model 19820 non-null object
 9  Individual 19820 non-null int64
 10 Trustmark Dealer 19820 non-null int64

 11  Diesel  12         19820  non-null  int64
 Electric   13        19820  non-null  int64
 LPG 14 Petrol        19820  non-null  int64
 15 Manual            19820 non-null int64
 16 5                 19820 non-null int64
 17 >5                19820 non-null int64
                      19820 non-null int64

dtypes: float64(6), int64(10), object(2)
memory usage: 2.7+ MB
```

	selling_price	year	km_driven	mileage	engine	max_power	age	Individual	Trustmark Dealer	
count	19820.000000	19820.000000	1.982000e+04	19820.000000	19820.000000	19820.000000	19820.000000	19820.000000	19820.000000	1
mean	6.585509	2014.561453	5.815856e+04	19.503402	1475.702381	98.122907	8.438547	0.390666	0.009586	
std	4.847364	3.196636	5.171563e+04	4.297784	518.571223	44.761727	3.196636	0.487912	0.097442	
min	0.300000	1992.000000	1.000000e+02	4.000000	0.000000	5.000000	2.000000	0.000000	0.000000	
25%	3.410000	2013.000000	3.100000e+04	16.950000	1197.000000	73.900000	6.000000	0.000000	0.000000	
50%	5.200000	2015.000000	5.200000e+04	19.300000	1248.000000	86.800000	8.000000	0.000000	0.000000	
75%	7.850000	2017.000000	7.400000e+04	22.320000	1582.000000	112.000000	10.000000	1.000000	0.000000	
max	20.902500	2021.000000	3.800000e+06	120.000000	6752.000000	626.000000	31.000000	1.000000	1.000000	

```
df3["make"]=df3.groupby("make")["selling_price"].transform("mean")

df3["model"]=df3.groupby("model")["selling_price"].transform("mean")

from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
df4=pd.DataFrame(scaler.fit_transform(df3),columns=df3.columns)
df4
```

	selling_price	year	km_driven	mileage	engine	max_power	age	make	model	Individual	Trustmark Dealer	Diesel
0	0.043684	0.689655	0.031553	0.135345	0.117891	0.066506	0.310345	0.194048	0.041550	1.0	0.0	0.0
1	0.252397	0.827586	0.005237	0.128448	0.177281	0.123994	0.172414	0.232517	0.218382	1.0	0.0	0.0
2	0.089795	0.620690	0.015764	0.112069	0.177281	0.120773	0.379310	0.232517	0.149143	1.0	0.0	0.0
3	0.095134	0.689655	0.009711	0.145862	0.147808	0.100000	0.310345	0.194048	0.093193	1.0	0.0	0.0
4	0.262104	0.793103	0.007869	0.161810	0.221860	0.150709	0.206897	0.252367	0.313574	0.0	0.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...
19815	0.300934	0.862069	0.018258	0.168879	0.202014	0.099919	0.137931	0.484670	0.328028	0.0	0.0	1.0
19816	0.434413	0.931034	0.004711	0.116379	0.203347	0.138647	0.068966	0.194048	0.330632	0.0	0.0	0.0
19817	0.191724	0.793103	0.017606	0.147759	0.221860	0.158647	0.206897	0.318156	0.200656	0.0	0.0	1.0
19818	0.580027	0.827586	1.000000	0.103448	0.322719	0.217391	0.172414	0.324782	0.377671	0.0	0.0	1.0
19819	0.567892	0.931034	0.003395	0.120690	0.221712	0.181320	0.068966	0.258412	0.519465	0.0	0.0	0.0

19820 rows × 18 columns

Next steps: Generate code with df4 | View recommended plots | New interactive sheet

```
from sklearn.model_selection import train_test_split
y=df3["selling_price"]
x=df3.drop("selling_price",axis=1)
y.shape,x.shape
```

```
((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,ran
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

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

```
from sklearn.linear_model import LinearRegression
model=LinearRegression()
model.fit(x_train,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
from sklearn.linear_model import LinearRegression
model= LinearRegression()
model.fit(x_train,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
model.coef_
```

```
array([ 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
```

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

```
array([ 2.95898276, 20.91025983, 7.69454424, ..., 3.24968466,
        5.44348659, 2.19881995])
```

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

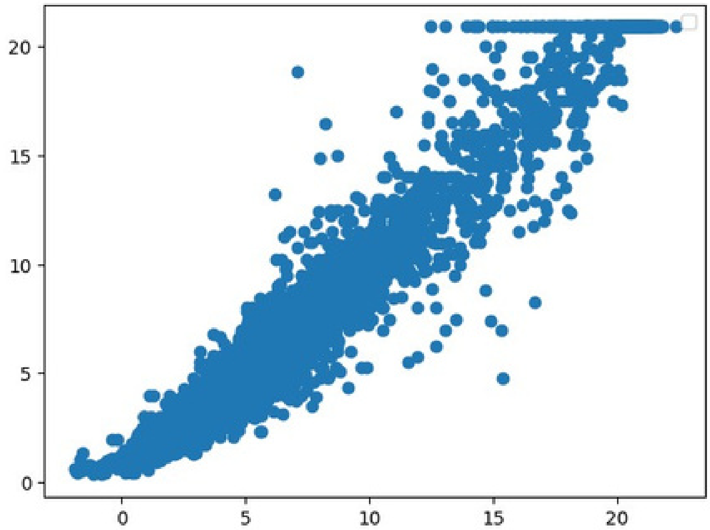
```
array([ 1.135053 , 4.78976725, 5.96846351, ..., 1.20471131,
        3.08615105, 10.67406941])
```

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

```
0.9458843076992299
```

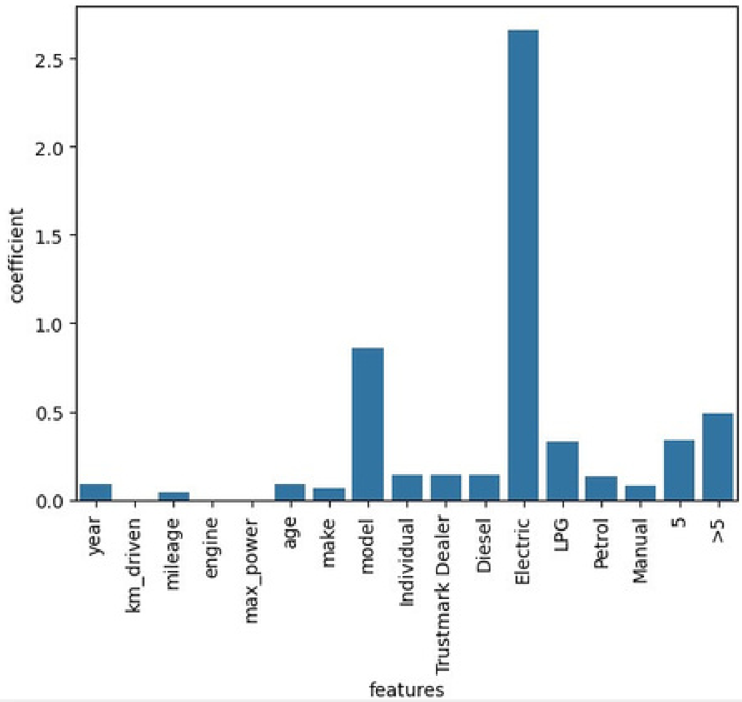
```
import matplotlib.pyplot as plt
fig=plt.figure()
plt.scatter(y_test_predict,y_test)
plt.legend()
plt.show()
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend. Note that artists whose label start with an underscore are
```




```
import seaborn as sns
imp=pd.DataFrame(list(zip(x.columns,np.abs(model.coef_))),columns=["
sns.barplot(x="features",y="coefficient",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-06

```
#QUESTION-6)prepare a l1 and l2 regression models for prediction of
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
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.csv")
df.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	o
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	

```
df.shape
```

```
(15210, 10)
```

```
df["ocean_proximity"]=df.groupby("ocean_proximity")["median_house_va
df.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	o
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	

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

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_val
0	0.211155	0.567481	0.784314	0.022331	0.019711	0.011168	0.020395	0.539668	0.9051
1	0.212151	0.565356	0.392157	0.180503	0.171349	0.083955	0.186842	0.538027	0.7169
2	0.210159	0.564293	1.000000	0.037260	0.029179	0.017260	0.028783	0.466028	0.7041
3	0.209163	0.564293	1.000000	0.032352	0.036163	0.019431	0.035691	0.354699	0.6825
4	0.209163	0.564293	1.000000	0.041330	0.043148	0.019676	0.042270	0.230776	0.6843
...	...	...	...	...	...	...	...	...	
15205	0.725100	0.049947	0.078431	0.149245	0.151327	0.067010	0.138158	0.324127	0.5881
15206	0.726096	0.049947	0.156863	0.062770	0.064411	0.031544	0.069901	0.270479	0.4183
15207	0.725100	0.048884	0.058824	0.158706	0.194940	0.077303	0.183224	0.258624	0.5631
15208	0.725100	0.048884	0.058824	0.232743	0.289306	0.132059	0.278947	0.307244	0.4829
15209	0.724104	0.049947	0.078431	0.143878	0.144187	0.089696	0.148026	0.392753	0.0000

15210 rows × 10 columns

```
df1.fillna(999,inplace=True)
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,ran
```

```
linear_model=LinearRegression()
lasso_model=Lasso(alpha=0.1)
ridge_model=Ridge(alpha=0.1)
```

```
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_)
```

```
[-3.66169442e-01  -3.91829113e-01  8.11385537e-02  6.36677925e-02
-9.63570893e-06  -2.76752624e+00  1.75536617e+00  1.08679972e+00
2.03408696e-01]
[ 0.00000000e+00 -0.00000000e+00 0.00000000e+00 0.00000000e+00
  -1.36304348e-05  -0.00000000e+00  0.00000000e+00  0.00000000e+00
0.00000000e+00]  [-3.65123858e-01  -3.89548815e-01  8.13405031e-02
7.73727816e-02    -9.92818475e-06  -2.63029307e+00  1.66464762e+00
1.08561983e+00
 2.04808328e-01]
```

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

```
0.34227829647618624
0.39767141229141667
0.34050258078362833
```

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

```
-184.32628077070459
```

```
linear_train_mse=mean_squared_error(y_train,linear_model.predict(x_t
linear_test_mse=mean_squared_error(y_test,linear_model.predict(x_tes
print(f"Linear    Model    Training    MSE:    {linear_train_mse}")
print(f"Linear Model Testing MSE: {linear_test_mse}")
```

```
Linear Model Training MSE: 0.018084107130981655
Linear Model Testing MSE: 9.113051321793758
```

LAB-07

```
#QUESTION-7)Prepare a decision tree model for Iris Dataset using Gin
#prepare a decision tree model using the gini index as criteria on t
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
```

	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
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

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

```
model = DecisionTreeClassifier(criterion='gini')
model
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

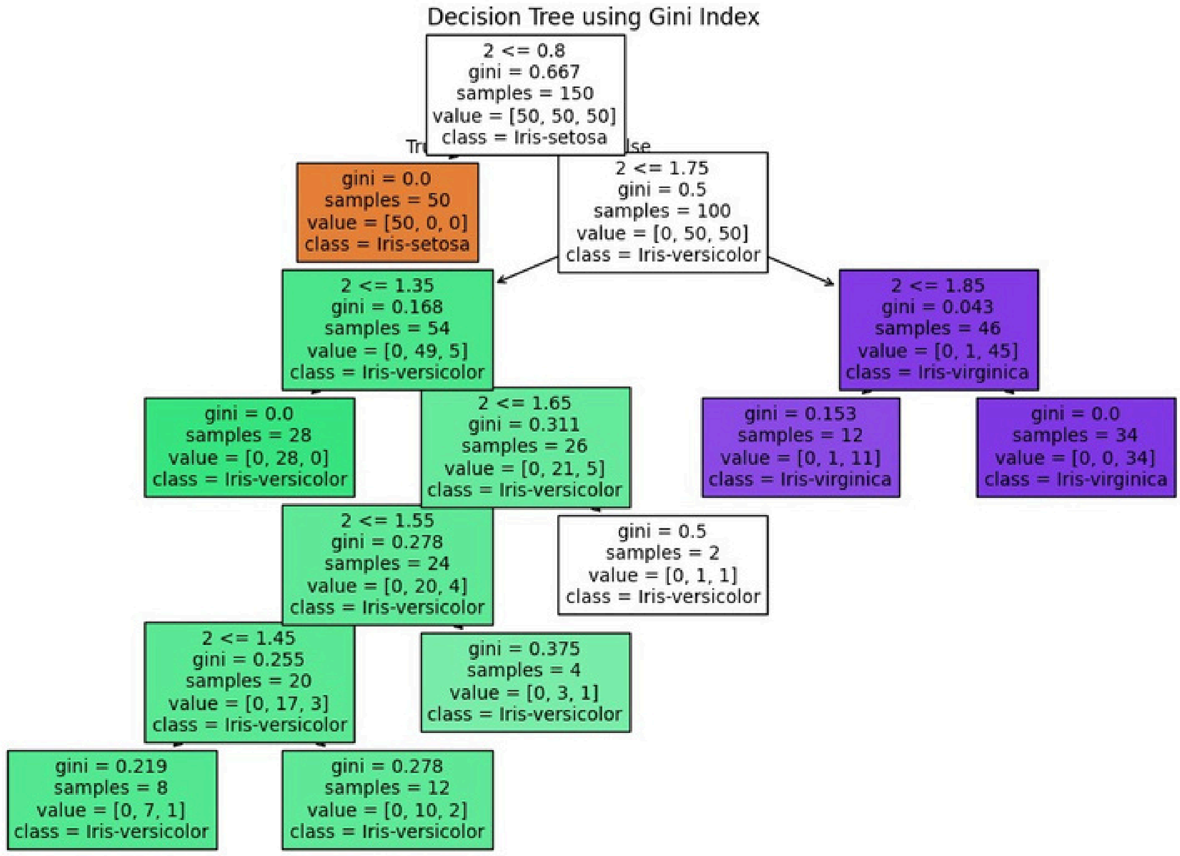
```
gini_impurities={}    import    numpy    as    np
arr=np.array([1,2,3,4,5,6])  print("orignal  array
shape:  ",arr.shape) reshaped_arr=arr.reshape(-1,1)
print("Reshaped array shape: ",reshaped_arr.shape)
print(reshaped_arr)
```

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

```
for i in range(x.shape[1]):
#fit the classifier with only the current feature
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).su

best_feature = min(gini_impurities, key=gini_impurities.get)
print(f"Best feature: {best_feature}")
```

```
Best feature: 2
```

```
plt.figure(figsize=(12, 8))
plot_tree(model, filled=True, feature_names=[best_feature], class_na
plt.title("Decision Tree using Gini Index")
plt.show()
```



LAB-08

```
#question-8)Prepare a decision tree model for Iris Dataset using ent
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
```

```
df =pd.read_csv("Iris.csv")
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
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

```
y = df['Species']

x_train , x_test , y_train , y_test = train_test_split(x,y,test_size

from sklearn import tree
model = tree.DecisionTreeClassifier(criterion = 'entropy',max_depth

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

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

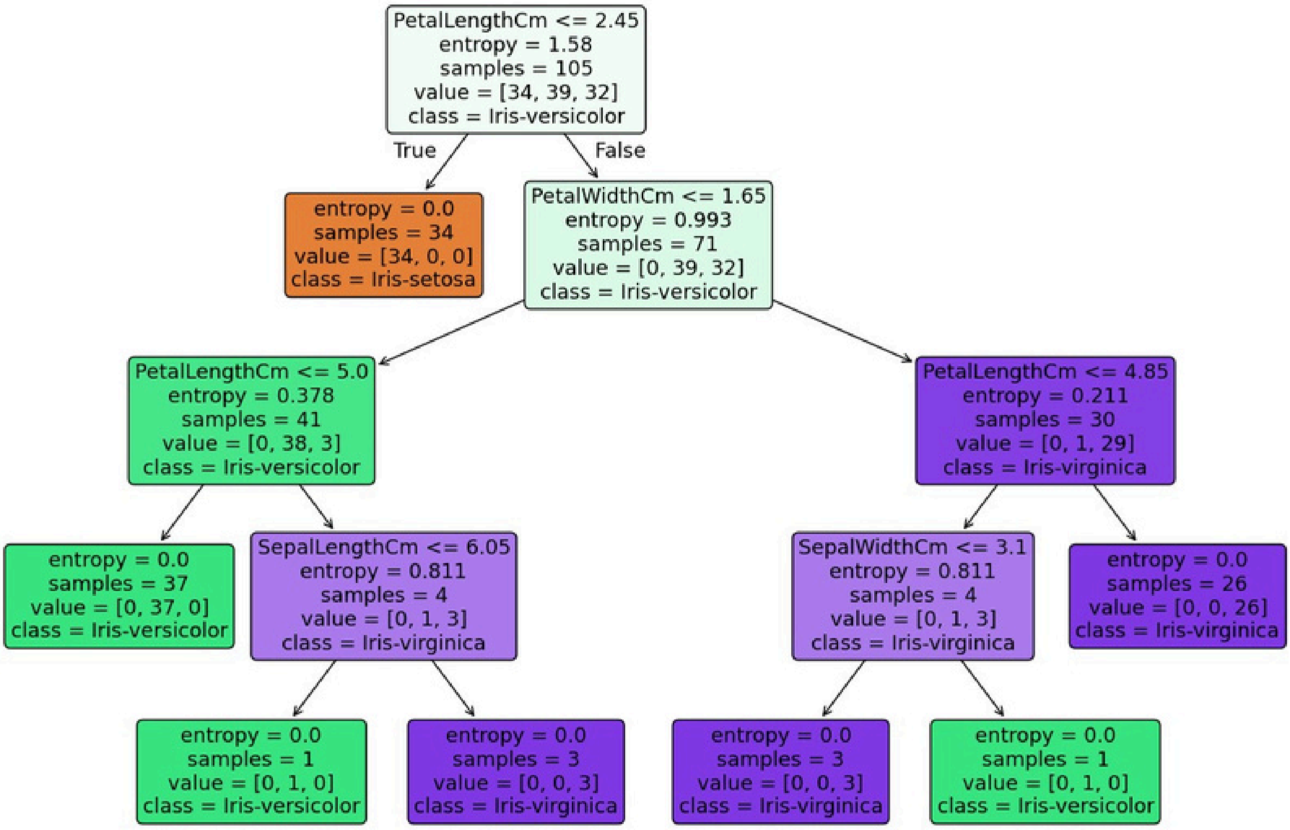
```
y_pred = model.predict(x_test)
print("Accuracy: " , accuracy_score(y_test,y_pred)*100)
```

```
Accuracy: 95.55555555555556
```

```
def plot_decision_tree(model, features_names,class_names):
plt.figure(figsize=(15,10))
plot_tree(model,feature_names=features_names,class_names=class_nam
plt.show()

model=tree.DecisionTreeClassifier(criterion='entropy',max_depth=4) #
model.fit(x_train,y_train)
y_pred=model.predict(x_test)
print(accuracy_score(y_test,y_pred)*100)
```

```
95.55555555555556
```

```
plot_decision_tree(model,x.columns,y.unique())
```



LAB-09

```
#question-9)Prepare a naïve bayes classification model for predictio
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.colors import ListedColormap
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.preprocessing import LabelEncoder
from sklearn import metrics
from sklearn.metrics import classification_report
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()
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

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

#label encoder is the class which is use to convert a categorical va
#since a ml model is the mathematical model so it understands numeri
le=LabelEncoder()
df['Gender']=le.fit_transform(df['Gender'])

#split data into dependent and independent variables
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,ra

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

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
```

```
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

```
#question-10)prepare a naiva bayes model for email classification in
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()
```

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN

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

	v1	v2
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

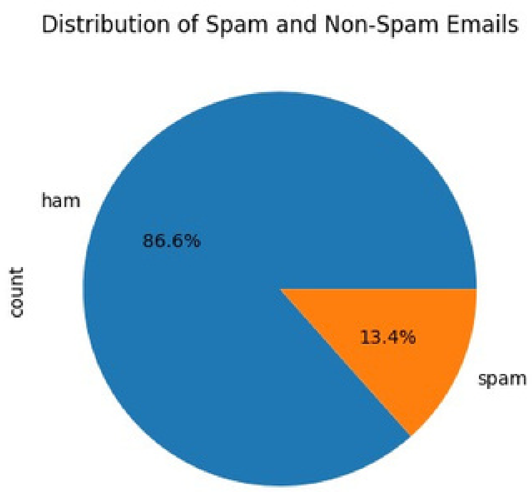
```
df=df.rename(columns={
    'v1':'label',
    'v2':'text'
})
df.head()
```

	label	text
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

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

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

#value_count is used to count the number of unique values in a datas
distribution=y.value_counts()
print(distribution)
```

```
label
ham     4825
spam     747
Name: count, dtype: int64
```

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




```
#generate a word cloud for spam messages
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 87121 to receive entry question(std txt rate)T&C\'s
apply 08452810075over18\'s FreeMsg Hey there darling it\'s been 3 week\'s now and no word back! I\'d like some fun you up for it st
ill? Tb ok! XxX std chgs to send, â£1.50 to rcv WINNER!! As a valued network customer you have been selected to receivea â£900 priz
e reward! To claim call 09061701461. Claim code KL341. Valid 12 hours only. Had your mobile 11 months or more? U R entitled to Upda
te to the latest colour mobiles with camera for Free! Call The Mobile Update Co FREE on 08002986030 SIX chances to win CASH! From 1 00
to 20,000 pounds txt> CSH11 and send to 87575. Cost 150p/day, 6days, 16+ TsandCs apply Reply HL 4 info URGENT! You have won a 1 week
FREE membership in our â£100,000 Prize Jackpot! Txt the word: CLAIM to No: 81010 T&C www.dbuk.net LCCLTD POBOX 4403LDNW1A7RW18
```

```
spam_wordcloud=WordCloud(width=800,height=400,max_words=100,backgrou
print(spam_wordcloud)
plt.figure(figsize=(10,4))
```

```
plt.imshow(spam_wordcloud)
plt.title('Word Cloud for Spam Messages')
plt.axis('off')
plt
```

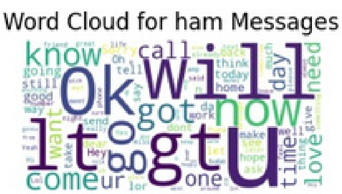
```
<wordcloud.wordcloud.WordCloud object at 0x7bc0830e8d30>
<module 'matplotlib.pyplot' from '/usr/local/lib/python3.10/dist-packages/matplotlib/pyplot.py'>
```


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

```
'Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat... Ok lar... Joking w if
u oni... U dun say so early hor... U c already then say... Nah I don\'t think he goes to usf, he lives around here though Even m y
brother is not like to speak with me. They treat me like aids patent. As per your request \'Melle Melle (Oru Minnaminunginte Nuru
ngu Vettam)\' has been set as your callertune for all Callers. Press *9 to copy your friends Callertune I\'m gonna be home soon and
i don\'t want to talk about this stuff anymore tonight, k? I\'ve cried enough today. I\'ve been searching for the right words to th
ank you for this breather. I promise i wont take your help for granted and will fulfil my promise. You have been wonderful and a bl
essing at all times. I HAVE A DATE ON SUNDAY WITH WILL!! Oh k...i\'m watching here:) Eh u remember how 2 spell his name... Yes i di
```

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

plt.subplot(1,2,2)
plt.imshow(ham_wordcloud)
plt.title("Word Cloud for ham Messages")
plt.axis("off")
```

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




```
#count vectorizer is a text processing techqnique used in natural la
vectorizer=CountVectorizer()
x_train=vectorizer.fit_transform(x_train)
x_test=vectorizer.transform(x_test)

#alpha is the regularization parametre and prevent overfitting
model_multinomial=MultinomialNB(alpha=0.8,fit_prior=True,force_alpha
model_multinomial.fit(x_train,y_train)
```

MultinomialNB

MultinomialNB(alpha=0.8)

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

GaussianNB

GaussianNB()

```
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)
```

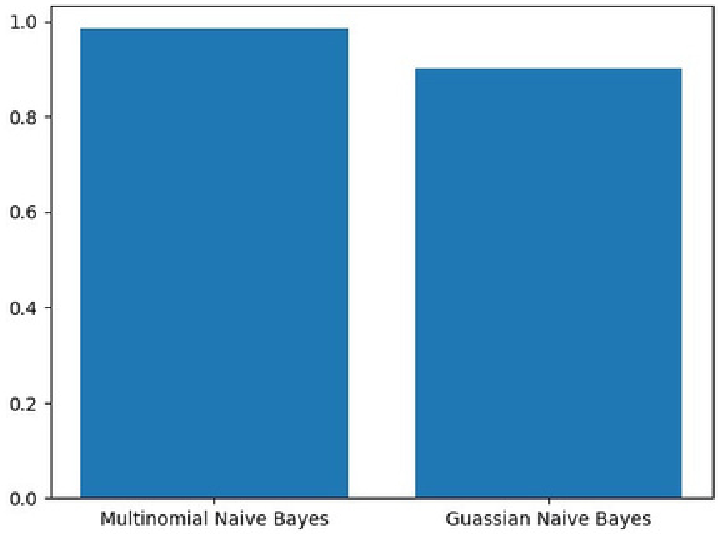
```
Accuracy for multinomial model is: 0.9838565022421525
```

```
y_pred_guassion=model_guassian.predict(x_test.toarray())
accuracy_guassian=accuracy_score(y_test,y_pred_guassion)
print("Accuracy for guassian model is:",accuracy_guassian)
```

```
Accuracy for guassian model is: 0.9004484304932735
```

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

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



LAB-11

```
#Question-11)prepare a model for prediction of prostate cancer using
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.model_selection import train_test_split

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

	lcavol	lweight	age	lbph	lcp	gleason	pgg45	lpsa	Target
0	-0.579818	2.769459	50	-1.386294	-1.386294	6	0	-0.430783	0
1	-0.994252	3.319626	58	-1.386294	-1.386294	6	0	-0.162519	0
2	-0.510826	2.691243	74	-1.386294	-1.386294	7	20	-0.162519	0
3	-1.203973	3.282789	58	-1.386294	-1.386294	6	0	-0.162519	0
4	0.751416	3.432373	62	-1.386294	-1.386294	6	0	0.371564	0
...	...	...	...	...	...	...	...	...	...
92	2.830268	3.876396	68	-1.386294	1.321756	7	60	4.385147	1
93	3.821004	3.896909	44	-1.386294	2.169054	7	40	4.684443	1
94	2.907447	3.396185	52	-1.386294	2.463853	7	10	5.143124	1
95	2.882564	3.773910	68	1.558145	1.558145	7	80	5.477509	1
96	3.471966	3.974998	68	0.438255	2.904165	7	20	5.582932	1

97 rows × 9 columns

```
df.shape
```

```
(97, 9)
```

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

#feature scaling(to convert the values between 0 and 1)
scaler=StandardScaler()
df1=pd.DataFrame(scaler.fit_transform(x),columns=df.columns[:-1])
df1.head()
```

	lcavol	lweight	age	lbph	lcp	gleason	pgg45	lpsa
0	-1.645861	-2.016634	-1.872101	-1.030029	-0.867655	-1.047571	-0.868957	-2.533318
1	-1.999313	-0.725759	-0.791989	-1.030029	-0.867655	-1.047571	-0.868957	-2.299712
2	-1.587021	-2.200154	1.368234	-1.030029	-0.867655	0.344407	-0.156155	-2.299712
3	-2.178174	-0.812191	-0.791989	-1.030029	-0.867655	-1.047571	-0.868957	-2.299712
4	-0.510513	-0.461218	-0.251933	-1.030029	-0.867655	-1.047571	-0.868957	-1.834631

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

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

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

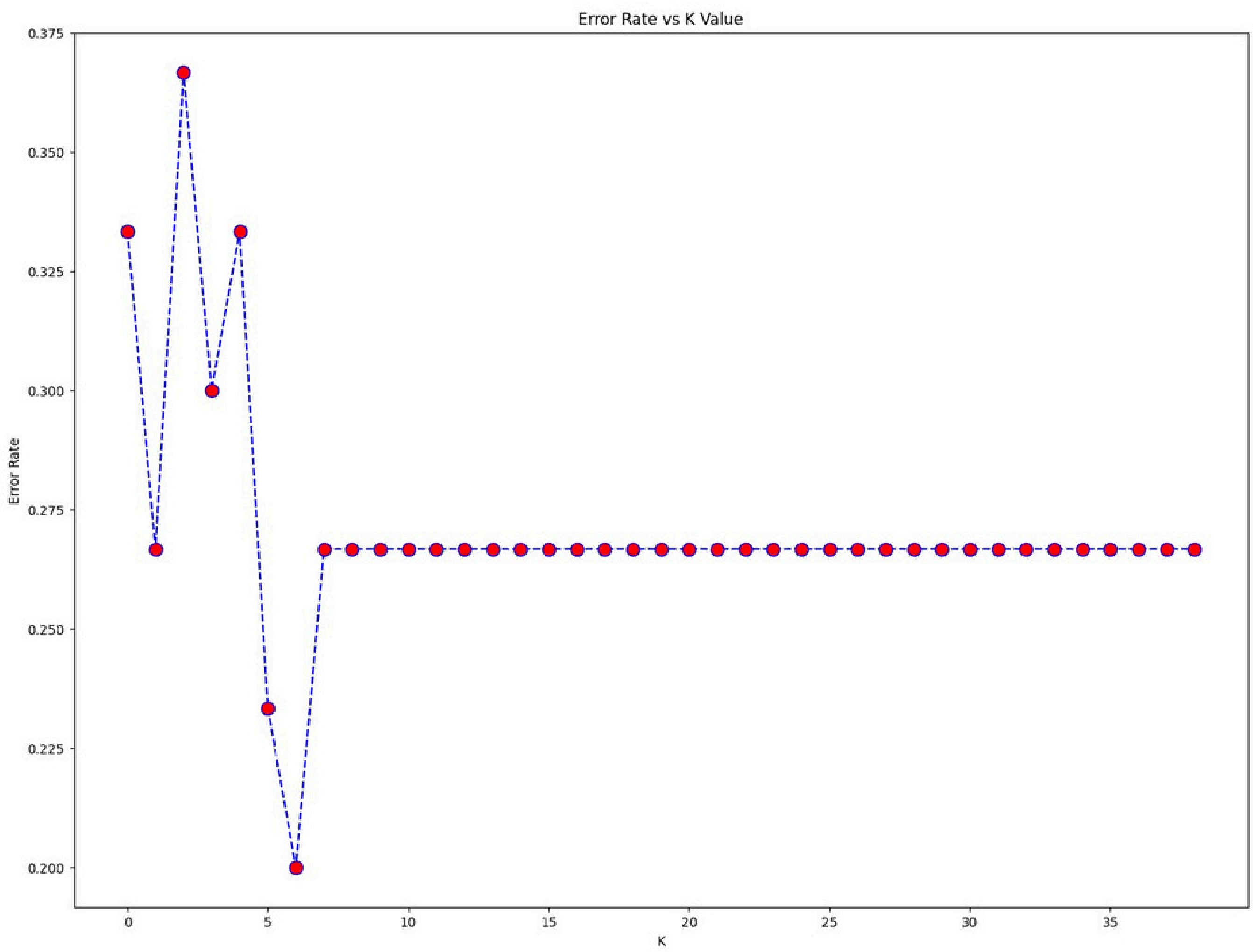
```
y_pred=knn_model.predict(x_test)

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

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

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

```
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
```

```
#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='o',marke

plt.title('Error Rate vs K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
plt.show()
```



 LAB-12

```
#Question-12)prepare a model for prediction of survival from Titanic import pandas as pd import numpy as np import warnings warnings.filterwarnings('ignore') import matplotlib.pyplot as plt from sklearn.preprocessing import LabelEncoder from sklearn.model_selection import train_test_split from sklearn.ensemble import RandomForestClassifier from sklearn.linear_model import LogisticRegression from sklearn.neighbors import KNeighborsClassifier from sklearn.svm import SVC from sklearn.metrics import accuracy_score, confusion_matrix, classi
```

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

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count Dtype
--- ------       -------------- -----
 0  PassengerId 891 non-null int64
 1  Survived    891 non-null int64
 2  Pclass      891 non-null int64
 3  Name        891 non-null object
 4  Sex   Age   891 non-null object
 5  SibSp       714 non-null float64
 6  Parch       891 non-null int64
 7  Ticket      891 non-null int64
 8  Fare        891 non-null object
 9              891 non-null float64
 10 Cabin       204 non-null object
 11 Embarked    889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

```python
df.dropna(subset=['Survived'])
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C

```python
df.shape
```

```
(891, 12)
```

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

```python
x.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 6 columns):
 #      Column Non-Null Count Dtype
    --- ------ -------------- -----
 0  Pclass 891 non-null      int64
```

```
 1  Sex      891  non-null  object
 2  Age      714  non-null  float64
 3  SibSp    891 non-null   int64
 4  Parch    891 non-null   int64
 5  Fare     891 non-null   float64
dtypes: float64(2), int64(3), object(1)
memory usage: 41.9+ KB
```

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

```
x.head()
```

	Pclass	Sex	Age	SibSp	Parch	Fare
0	3	1	22.0	1	0	7.2500
1	1	0	38.0	1	0	71.2833
2	3	0	26.0	0	0	7.9250
3	1	0	35.0	1	0	53.1000
4	3	1	35.0	0	0	8.0500

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

```
x['Age']
```

	Age
0	22.000000
1	38.000000
2	26.000000
3	35.000000
4	35.000000
...	...
886	27.000000
887	19.000000
888	29.699118
889	26.000000
890	32.000000

891 rows × 1 columns

dtype: float64

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

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

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

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

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