Deccan Education Society's

Kirti M. Doongursee College of Arts, Science and Commerce

[Autonomous], Dadar (West), Mumbai-400 028.

M.Sc. [Computer Science]

Practical journal

Seat Number [   ]

Department of Computer Science and Information Technology

Department of Computer Science and Information Technology
Deccan Education Society's

Kirti M. Doongursee College of Arts, Science and Commerce
[Autonomous], Dadar (West), Mumbai-400 028.

C E R T I F I C A T E

This is to certify that ____

of M.Sc. (Computer Science) with Seat No._____ has completed **Practical journal** of Paper- **Applied Machine Learning and Deep Learning** under my supervision in this College during the year **2023-2024.**

Lecturer-In-Charge | **H.O.D.**
Department of
Computer Science & IT

Date: / /2024 | Date:

Examined by: | **Remarks:**

Date:

____

INDEX

Sr. No.	Date	Title	Sign
1.	14/05/24	Practical 1.1 linear regression on Study dataset	
2.	14/05/24	Practical 1.2 linear regression on Advertising dataset	
3.	21/05/24	Practical-2.1 Logistic Regression on Placement dataset	
4.	21/05/24	Practical-2.2 Logistic Regression on iris dataset with Cross Validation and Kfold	
5.	21/05/24	Practical-3.1_Lasso_Regression on Boston Housing dataset	
6.	21/05/24	Practical-3.2_Linear_Regression_standard_scalar_california_housing	
7.	27/05/24	Practical-3.3_ridge_Regression_standard_scalar_california_housing	
8.	27/05/24	Practical-3.4_Lasso_Regression_standard_scalar_california_housing	
9.	28/05/24	Practical-4_KNN_Regressor on Boston Housing dataset	
10.	28/05/24	Practical-5_KNN_Classifier on iris dataset	
11.	30/05/24	Practical-6_K-means on Student Results dataset	
12.	30/05/24	Practical-7_SVM_classifier on diabetes dataset	
13	06/06/24	Practical-8_Ensemble_Bagging on Breast Cancer dataset	

Practical-1.1_linear_regression_study_dataset

[ ]:
```
!pip install scikit-learn pandas numpy
```

[1]:
```
import pandas as pd
df = pd.read_csv("./dataset/Study.csv")
df
```

[1]:

	Hours	Scores
0	2.5	21
1	5.1	47
2	3.2	27
3	8.5	75
4	3.5	30
5	1.5	20
6	9.2	88
7	5.5	60
8	8.3	81
9	2.7	25
10	7.7	85
11	5.9	62
12	4.5	41
13	3.3	42
14	1.1	17
15	8.9	95
16	2.5	30
17	1.9	24
18	6.1	67
19	7.4	69
20	2.7	30
21	4.8	54
22	3.8	35
23	6.9	76
24	7.8	86

[2]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25 entries, 0 to 24
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
```

```
---  ------  --------------  -----
 0   Hours   25 non-null     float64
 1   Scores  25 non-null     int64
dtypes: float64(1), int64(1)
memory usage: 532.0 bytes
```

[3]:
```
x = df.iloc[:,:-1]
y = df.iloc[:,-1]
```

[4]:
```
x
```

[4]:

	Hours
0	2.5
1	5.1
2	3.2
3	8.5
4	3.5
5	1.5
6	9.2
7	5.5
8	8.3
9	2.7
10	7.7
11	5.9
12	4.5
13	3.3
14	1.1
15	8.9
16	2.5
17	1.9
18	6.1
19	7.4
20	2.7
21	4.8
22	3.8
23	6.9
24	7.8

[5]:
```
y
```

[5]:
```
0     21
1     47
2     27
3     75
4     30
5     20
6     88
7     60
```

```
8     81
9     25
10    85
11    62
12    41
13    42
14    17
15    95
16    30
17    24
18    67
19    69
20    30
21    54
22    35
23    76
24    86
Name: Scores, dtype: int64
```

[6]:
```python
from sklearn.model_selection import train_test_split
```

[7]:
```python
xtrain, xtest, ytrain, ytest = train_test_split(x,y,test_size=0.
↪2,random_state=1)
```

[8]:
```python
ytrain
```

[8]:
```
10    85
18    67
19    69
4     30
2     27
20    30
6     88
7     60
22    35
1     47
16    30
0     21
15    95
24    86
23    76
9     25
8     81
12    41
11    62
5     20
Name: Scores, dtype: int64
```

[9]:
```
from sklearn.linear_model import LinearRegression
```

[10]:
```
lr = LinearRegression()
```

[11]:
```
lr.fit(xtrain, ytrain)
```

[11]:
```
LinearRegression()
```

[12]:
```
predictions = lr.predict(xtest)
```

Beta 0

[14]:
```
lr.intercept_
```

[14]:
```
-1.5369573315500702
```

Beta 1

[15]:
```
lr.coef_
```

[15]:
```
array([10.46110829])
```

[16]:
```
print(ytest)
print(predictions)
```

```
14    17
13    42
17    24
3     75
21    54
Name: Scores, dtype: int64
[ 9.97026179 32.98470004 18.33914843 87.38246316 48.67636248]
```

[17]:
```
from sklearn.metrics import mean_absolute_error,r2_score
```

[18]:
```
mean_absolute_error(ytest,predictions)
```

[18]:
```
7.882398086270432
```

R2 Score should be close to 1

[19]:
```
r2_score(ytest,predictions)
```

[19]:
```
0.8421031525243527
```

Practical-1.2_linear_regression_Advertising_dataset

[ ]:
```
!pip install numpy pandas scikit-learn matplotlib
```

[2]:
```
import pandas as pd
```

[3]:
```
df = pd.read_csv('./dataset/Advertising.csv')
```

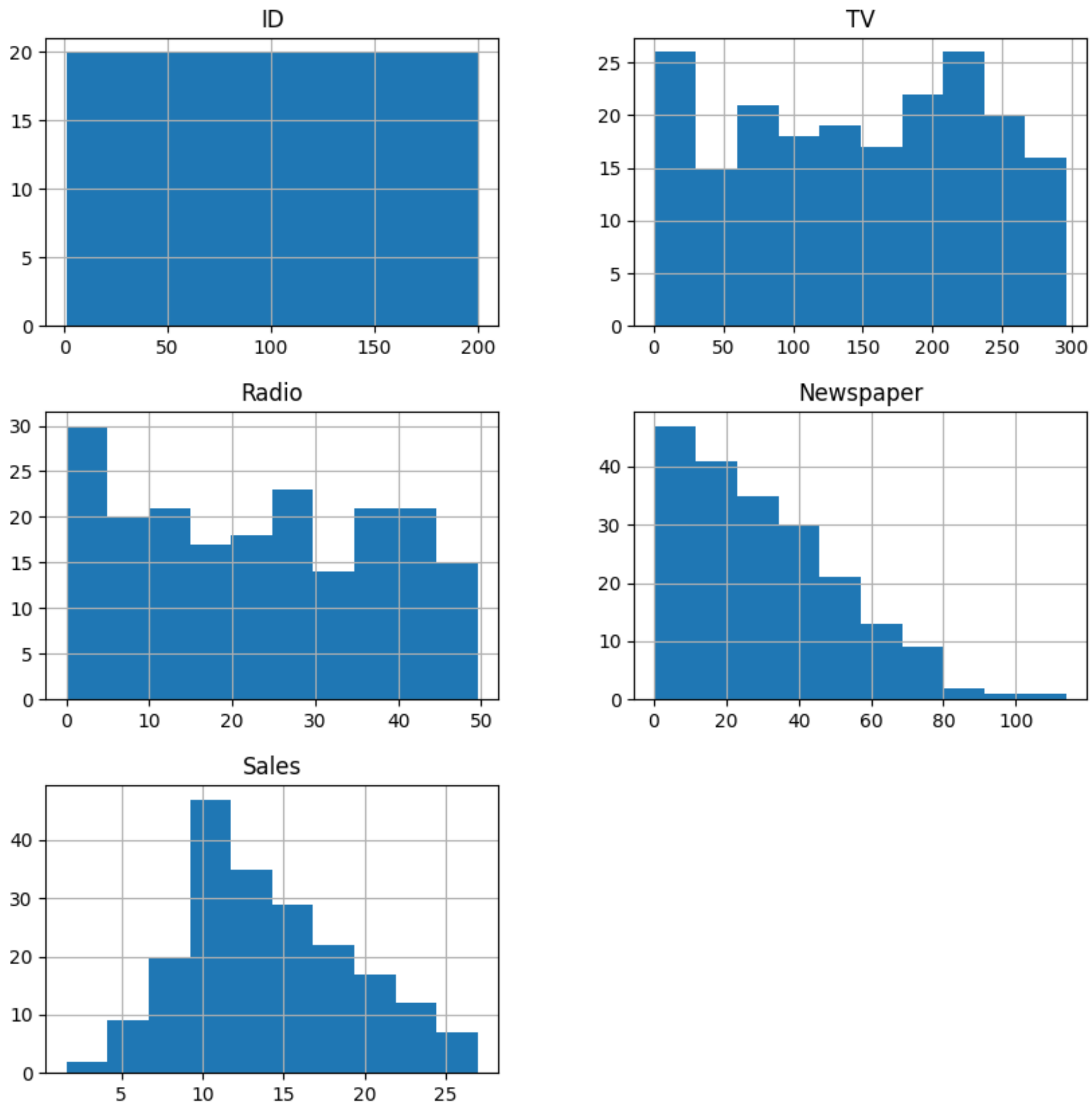
[4]:
```
df
```

[4]:
```
      ID     TV  Radio  Newspaper  Sales
0      1  230.1   37.8       69.2   22.1
1      2   44.5   39.3       45.1   10.4
2      3   17.2   45.9       69.3    9.3
3      4  151.5   41.3       58.5   18.5
4      5  180.8   10.8       58.4   12.9
..   …     …     …        …     …
195  196   38.2    3.7       13.8    7.6
196  197   94.2    4.9        8.1    9.7
197  198  177.0    9.3        6.4   12.8
198  199  283.6   42.0       66.2   25.5
199  200  232.1    8.6        8.7   13.4

[200 rows x 5 columns]
```

[5]:
```
df.hist(figsize = (10,10))
```

[5]:
```
array([[<Axes: title={'center': 'ID'}>, <Axes: title={'center': 'TV'}>],
       [<Axes: title={'center': 'Radio'}>,
        <Axes: title={'center': 'Newspaper'}>],
       [<Axes: title={'center': 'Sales'}>, <Axes: >]], dtype=object)
```



[6]:
```
x = df.iloc[:,:-1]
x
```

[6]:
```
      ID     TV  Radio  Newspaper
0      1  230.1   37.8       69.2
1      2   44.5   39.3       45.1
2      3   17.2   45.9       69.3
3      4  151.5   41.3       58.5
4      5  180.8   10.8       58.4
..   …      …      …        …
195  196   38.2    3.7       13.8
196  197   94.2    4.9        8.1
```

```
197  198  177.0    9.3        6.4
198  199  283.6   42.0       66.2
199  200  232.1    8.6        8.7

[200 rows x 4 columns]
```

[7]:
```
y = df.iloc[:,-1]
y
```

[7]:
```
0      22.1
1      10.4
2       9.3
3      18.5
4      12.9
       …
195     7.6
196     9.7
197    12.8
198    25.5
199    13.4
Name: Sales, Length: 200, dtype: float64
```

[8]:
```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest = train_test_split(x,y,test_size=0.3,random_state=2)
```

[9]:
```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
```

[10]:
```
lr.fit(xtrain,ytrain)
```

[10]:
```
LinearRegression()
```

[11]:
```
predict1 = lr.predict(xtest)
```

[12]:
```
predict1
```

[12]:
```
array([13.9532874 ,  9.66868515,  6.8184482 , 15.17477774, 18.17445759,
       15.71712804,  7.57102926, 20.49761622, 13.19168021, 17.33321225,
       11.14787534, 19.48405747,  9.21291503, 10.79916797, 13.87449153,
       12.82277648,  9.32688188, 18.03502362, 16.4741683 , 18.82432455,
       16.92036691, 16.04687481, 12.02316254, 12.14339122, 14.87243899,
       12.08358743, 15.49759505,  8.08839314, 16.76959963, 13.87908137,
       16.20151428, 17.08595853, 13.05758661, 13.02632284,  8.91762832,
       11.01315087, 21.97837505, 19.89971271, 15.98938391, 20.23559933,
       21.12563546, 17.17066585, 21.13093152, 15.04421578, 19.66106778,
       18.74322737, 17.58850441, 10.34247317,  9.60222257, 13.00297864,
       12.59823575, 14.39336676, 17.46356839, 16.99371718,  8.55587416,
       17.10336367,  9.13263193,  4.15865759,  7.67835429, 24.67819442])
```

[13]:
```python
from sklearn.metrics import r2_score,mean_absolute_error
```

[14]:
```python
print(r2_score(ytest,predict1))
```

```
0.8021354391516504
```

[15]:
```python
print(mean_absolute_error(ytest,predict1))
```

```
1.5264827355298254
```

Practical-2.1_Logistic_Regression_placement_dataset

[1]:
```python
import pandas as pd
```

[2]:
```python
df = pd.read_csv("./dataset/placement.csv")
```

[3]:
```python
df
```

[3]:
```
     cgpa  placement_exam_marks  placed
0    7.19                  26.0       1
1    7.46                  38.0       1
2    7.54                  40.0       1
3    6.42                   8.0       1
4    7.23                  17.0       0
..    …                     …      …
995  8.87                  44.0       1
996  9.12                  65.0       1
997  4.89                  34.0       0
998  8.62                  46.0       1
999  4.90                  10.0       1

[1000 rows x 3 columns]
```

[4]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 3 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   cgpa                  1000 non-null   float64
 1   placement_exam_marks  1000 non-null   float64
 2   placed                1000 non-null   int64
dtypes: float64(2), int64(1)
memory usage: 23.6 KB
```

[5]:
```python
df.shape
```

[5]: (1000, 3)

[6]:
```python
df.head()
```

[6]:
```
   cgpa  placement_exam_marks  placed
0  7.19                  26.0       1
1  7.46                  38.0       1
2  7.54                  40.0       1
3  6.42                   8.0       1
4  7.23                  17.0       0
```

[7]:
```
df['placed'].unique()
```

[7]:
```
array([1, 0])
```

[8]:
```
X = df.iloc[:,:-1]
```

[9]:
```
y = df.iloc[:,-1]
```

[10]:
```
from sklearn.model_selection import train_test_split
```

[11]:
```
xtrain,xtest,ytrain,ytest = train_test_split(X,y,test_size=0.25, random_state=1)
```

[12]:
```
from sklearn.linear_model import LogisticRegression
```

[13]:
```
classifier = LogisticRegression()
```

[14]:
```
classifier.fit(xtrain, ytrain)
```

[14]:
```
LogisticRegression()
```

[15]:
```
predictions = classifier.predict(xtest)
```

[16]:
```
probability = classifier.predict_proba(xtest)
```

[17]:
```
probability[:10]
```

[17]:
```
array([[0.57188237, 0.42811763],
       [0.58207783, 0.41792217],
       [0.54254942, 0.45745058],
       [0.52320624, 0.47679376],
       [0.55570161, 0.44429839],
       [0.51994635, 0.48005365],
       [0.49926819, 0.50073181],
       [0.51124366, 0.48875634],
       [0.55763857, 0.44236143],
       [0.497788  , 0.502212  ]])
```

[18]:
```
predictions
```

[18]:
```
array([0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0,
```

```
       0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0])
```

[19]:

```
from sklearn.metrics import accuracy_score
```

[20]:

```
accuracy_score(ytest, predictions)
```

[20]: 0.48

Practical-2.2_Logistic_Regression_iris_cv_kfold

[1]:
```
from sklearn.datasets import load_iris
```

[2]:
```
from sklearn.model_selection import cross_val_score, KFold
```

[3]:
```
from sklearn.linear_model import LogisticRegression
```

[4]:
```
iris = load_iris()
```

[5]:
```
x = iris.data
```

[14]:
```
x[:10]
```

[14]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1]])
```

[7]:
```
y = iris.target
```

[8]:
```
y
```

[8]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

[9]:
```
lr = LogisticRegression()
```

[10]:
```
k_fold = KFold(n_splits=7)
```

[11]:
```python
score = cross_val_score(lr, x, y, cv=k_fold)
```

[12]:
```python
print("Cross Validation Scores:{}".format(score))
```

```
Cross Validation Scores:[1.         1.         1.         0.80952381 0.95238095
0.85714286
 0.9047619 ]
```

[13]:
```python
print("Average Cross Validation score:{}".format(score.mean()))
```

```
Average Cross Validation score:0.9319727891156463
```

Practical-3.1_Lasso_Regression_housing-dataset

[ ]:
```
!pip install numpy pandas scikit-learn
```

[1]:
```
import pandas as pd
df = pd.read_csv("./dataset/boston-housing-dataset.csv")

# First 3
df.head(3)

# Random 3
df.sample(3)

# Last 3
df.tail(3)
```

[1]:
```
        CRIM   ZN  INDUS  CHAS     NX     RM   AGE     DIS  RAD    TAX  \
503  0.06076  0.0  11.93     0  0.573  6.976  91.0  2.1675    1  273.0
504  0.10959  0.0  11.93     0  0.573  6.794  89.3  2.3889    1  273.0
505  0.04741  0.0  11.93     0  0.573  6.030  80.8  2.5050    1  273.0

     PTRATIO       B  LSTAT  MEDV
503     21.0  396.90   5.64  23.9
504     21.0  393.45   6.48  22.0
505     21.0  396.90   7.88  11.9
```

[2]:
```
x = df.iloc[:,:-1]
x.shape
```

[2]: (506, 13)

[3]:
```
y = df.iloc[:,-1]
y
```

[3]:
```
0      24.0
1      21.6
2      34.7
3      33.4
4      36.2
       …
```

```
501    22.4
502    20.6
503    23.9
504    22.0
505    11.9
Name: MEDV, Length: 506, dtype: float64
```

[4]:
```
from sklearn.linear_model import Lasso
model = Lasso()
```

[5]:
```
from sklearn.model_selection import train_test_split
```

[6]:
```
xtrain, xtest, ytrain, ytest = train_test_split(x,y,test_size = 0.25,␣
 ↪random_state = 1)
model.fit(xtrain, ytrain)
```

[6]:
```
Lasso()
```

[7]:
```
from sklearn.model_selection import RepeatedKFold
cv = RepeatedKFold(n_splits = 10, n_repeats=3, random_state=1)
```

[8]:
```
from sklearn.metrics import r2_score
ypred = model.predict(xtest)
r2_score(ytest, ypred)
```

[8]:
```
0.662198077052326
```

[9]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_sc = sc.fit_transform(x)
xtrain, xtest, ytrain, ytest = train_test_split(x_sc,y,test_size = 0.25,␣
 ↪random_state=1)
model1 = Lasso()
params = {'alpha':[0.00001,0.0001,0.001,0.01]}
```

[10]:
```
from sklearn.model_selection import GridSearchCV
search = GridSearchCV(model1, params, cv=cv)
result = search.fit(x_sc,y)
result.best_params_
```

[10]:
```
{'alpha': 0.01}
```

[11]:
```
model2 = Lasso(alpha=0.01)
model2.fit(xtrain,ytrain)
```

[11]:
```
Lasso(alpha=0.01)
```

[12]:
```
ypred2 = model2.predict(xtest)
r2_score(ytest, ypred2)
```

[12]: 0.7787372388293925

Practical-3.2_Linear_Regression_standard_scalar_california_housing

[1]:
```
!pip install numpy pandas scikit-learn
```

[1]:
```
import pandas as pd
from sklearn.datasets import fetch_california_housing
```

[2]:
```
boston = fetch_california_housing()
boston_df = pd.DataFrame(boston.data, columns=boston.feature_names)
boston_df["Price_House"]=boston.target
boston_df
```

[2]:
```
       MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  Latitude  \
0      8.3252      41.0  6.984127   1.023810       322.0  2.555556     37.88
1      8.3014      21.0  6.238137   0.971880      2401.0  2.109842     37.86
2      7.2574      52.0  8.288136   1.073446       496.0  2.802260     37.85
3      5.6431      52.0  5.817352   1.073059       558.0  2.547945     37.85
4      3.8462      52.0  6.281853   1.081081       565.0  2.181467     37.85
…        …         …        …          …           …          …         …
20635  1.5603      25.0  5.045455   1.133333       845.0  2.560606     39.48
20636  2.5568      18.0  6.114035   1.315789       356.0  3.122807     39.49
20637  1.7000      17.0  5.205543   1.120092      1007.0  2.325635     39.43
20638  1.8672      18.0  5.329513   1.171920       741.0  2.123209     39.43
20639  2.3886      16.0  5.254717   1.162264      1387.0  2.616981     39.37

       Longitude  Price_House
0        -122.23        4.526
1        -122.22        3.585
2        -122.24        3.521
3        -122.25        3.413
4        -122.25        3.422
…          …            …
20635    -121.09        0.781
20636    -121.21        0.771
20637    -121.22        0.923
20638    -121.32        0.847
20639    -121.24        0.894
```

```
[20640 rows x 9 columns]
```

[3]:
```python
x = boston_df.iloc[:,:-1]
```

[4]:
```python
x
```

[4]:
```
         MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  Latitude  \
0        8.3252      41.0  6.984127   1.023810       322.0  2.555556     37.88
1        8.3014      21.0  6.238137   0.971880      2401.0  2.109842     37.86
2        7.2574      52.0  8.288136   1.073446       496.0  2.802260     37.85
3        5.6431      52.0  5.817352   1.073059       558.0  2.547945     37.85
4        3.8462      52.0  6.281853   1.081081       565.0  2.181467     37.85
…          …         …         …          …           …         …
20635    1.5603      25.0  5.045455   1.133333       845.0  2.560606     39.48
20636    2.5568      18.0  6.114035   1.315789       356.0  3.122807     39.49
20637    1.7000      17.0  5.205543   1.120092      1007.0  2.325635     39.43
20638    1.8672      18.0  5.329513   1.171920       741.0  2.123209     39.43
20639    2.3886      16.0  5.254717   1.162264      1387.0  2.616981     39.37

         Longitude
0          -122.23
1          -122.22
2          -122.24
3          -122.25
4          -122.25
…             …
20635      -121.09
20636      -121.21
20637      -121.22
20638      -121.32
20639      -121.24

[20640 rows x 8 columns]
```

[5]:
```python
y = boston_df.iloc[:,-1]
```

[6]:
```python
y
```

[6]:
```
0        4.526
1        3.585
2        3.521
3        3.413
4        3.422
         …
20635    0.781
20636    0.771
20637    0.923
```

```
20638    0.847
20639    0.894
Name: Price_House, Length: 20640, dtype: float64
```

[7]:
```python
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_sc= sc.fit_transform(x)
```

[8]:
```python
x_sc
```

[8]:
```
array([[ 2.34476576,  0.98214266,  0.62855945, …, -0.04959654,
         1.05254828, -1.32783522],
       [ 2.33223796, -0.60701891,  0.32704136, …, -0.09251223,
         1.04318455, -1.32284391],
       [ 1.7826994 ,  1.85618152,  1.15562047, …, -0.02584253,
         1.03850269, -1.33282653],
       …,
       [-1.14259331, -0.92485123, -0.09031802, …, -0.0717345 ,
         1.77823747, -0.8237132 ],
       [-1.05458292, -0.84539315, -0.04021111, …, -0.09122515,
         1.77823747, -0.87362627],
       [-0.78012947, -1.00430931, -0.07044252, …, -0.04368215,
         1.75014627, -0.83369581]])
```

[9]:
```python
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest = train_test_split(x_sc,y,test_size=0.
 ↪3,random_state=2)
```

[10]:
```python
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
```

[11]:
```python
lr.fit(xtrain,ytrain)
```

[11]:
```
LinearRegression()
```

[12]:
```python
predict1 = lr.predict(xtest)
```

[13]:
```python
from sklearn.metrics import r2_score,mean_absolute_error
```

[14]:
```python
print(r2_score(ytest,predict1))
```

```
0.6015507891610434
```

[15]:
```python
print(mean_absolute_error(ytest,predict1))
```

```
0.5362588391493065
```

Practical-3.3_ridge_Regression_standard_scalar_california_housing

[ ]:
```
!pip install numpy pandas scikit-learn
```

[1]:
```
import pandas as pd
from sklearn.datasets import fetch_california_housing
ds = fetch_california_housing()
df = pd.DataFrame(ds.data, columns=ds.feature_names)
df.head(3)
```

[1]:
```
   MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  Latitude  \
0  8.3252      41.0  6.984127   1.023810       322.0  2.555556     37.88
1  8.3014      21.0  6.238137   0.971880      2401.0  2.109842     37.86
2  7.2574      52.0  8.288136   1.073446       496.0  2.802260     37.85

   Longitude
0    -122.23
1    -122.22
2    -122.24
```

[2]:
```
df["HousePrice"]=ds.target
df.head(3)
```

[2]:
```
   MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  Latitude  \
0  8.3252      41.0  6.984127   1.023810       322.0  2.555556     37.88
1  8.3014      21.0  6.238137   0.971880      2401.0  2.109842     37.86
2  7.2574      52.0  8.288136   1.073446       496.0  2.802260     37.85

   Longitude  HousePrice
0    -122.23       4.526
1    -122.22       3.585
2    -122.24       3.521
```

[3]:
```
X = df.iloc[:,:-1]
y = df.iloc[:,-1]
```

[4]:
```
X.head(3)
```

[4]:
```
   MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  Latitude  \
0  8.3252      41.0  6.984127   1.023810       322.0  2.555556     37.88
1  8.3014      21.0  6.238137   0.971880      2401.0  2.109842     37.86
2  7.2574      52.0  8.288136   1.073446       496.0  2.802260     37.85

   Longitude
0    -122.23
1    -122.22
2    -122.24
```

[5]:
```python
y.head(3)
```

[5]:
```
0    4.526
1    3.585
2    3.521
Name: HousePrice, dtype: float64
```

[6]:
```python
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_sc = sc.fit_transform(X)
x_sc
```

[6]:
```
array([[ 2.34476576,  0.98214266,  0.62855945, …, -0.04959654,
         1.05254828, -1.32783522],
       [ 2.33223796, -0.60701891,  0.32704136, …, -0.09251223,
         1.04318455, -1.32284391],
       [ 1.7826994 ,  1.85618152,  1.15562047, …, -0.02584253,
         1.03850269, -1.33282653],
       …,
       [-1.14259331, -0.92485123, -0.09031802, …, -0.0717345 ,
         1.77823747, -0.8237132 ],
       [-1.05458292, -0.84539315, -0.04021111, …, -0.09122515,
         1.77823747, -0.87362627],
       [-0.78012947, -1.00430931, -0.07044252, …, -0.04368215,
         1.75014627, -0.83369581]])
```

[7]:
```python
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(x_sc, y, test_size = 0.3,␣
 ↪random_state=2)
```

[8]:
```python
from sklearn.linear_model import Ridge
rr = Ridge(alpha = 10)
```

[9]:
```python
rr.fit(xtrain,ytrain)
```

[9]:
```
Ridge(alpha=10)
```

[10]:
```python
predict = rr.predict(xtest)
```

[11]:
```
from sklearn.metrics import r2_score, mean_absolute_error
r2_score(ytest,predict)
```

[11]: 0.6014258698314037

[12]:
```
mean_absolute_error(ytest,predict)
```

[12]: np.float64(0.536239634635214)

Practical-3.4_Lasso_Regression_standard_scalar_california_housing

[ ]:
```
!pip install numpy pandas scikit-learn
```

[1]:
```
import pandas as pd
from sklearn.datasets import fetch_california_housing
```

[2]:
```
ds = fetch_california_housing()
df = pd.DataFrame(ds.data, columns = ds.feature_names)
df["HousePrice"] = ds.target

df.head(3)
```

[2]:
```
   MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  Latitude  \
0  8.3252      41.0  6.984127   1.023810       322.0  2.555556     37.88
1  8.3014      21.0  6.238137   0.971880      2401.0  2.109842     37.86
2  7.2574      52.0  8.288136   1.073446       496.0  2.802260     37.85

   Longitude  HousePrice
0    -122.23       4.526
1    -122.22       3.585
2    -122.24       3.521
```

[3]:
```
X = df.iloc[:,:-1]
y = df.iloc[:,-1]
```

[4]:
```
X.head(3)
```

[4]:
```
   MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  Latitude  \
0  8.3252      41.0  6.984127   1.023810       322.0  2.555556     37.88
1  8.3014      21.0  6.238137   0.971880      2401.0  2.109842     37.86
2  7.2574      52.0  8.288136   1.073446       496.0  2.802260     37.85

   Longitude
0    -122.23
1    -122.22
2    -122.24
```

[5]:
```
y.head(3)
```

[5]:
```
0    4.526
1    3.585
2    3.521
Name: HousePrice, dtype: float64
```

[6]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_sc = sc.fit_transform(X)

x_sc
```

[6]:
```
array([[ 2.34476576,  0.98214266,  0.62855945, …, -0.04959654,
         1.05254828, -1.32783522],
       [ 2.33223796, -0.60701891,  0.32704136, …, -0.09251223,
         1.04318455, -1.32284391],
       [ 1.7826994 ,  1.85618152,  1.15562047, …, -0.02584253,
         1.03850269, -1.33282653],
       …,
       [-1.14259331, -0.92485123, -0.09031802, …, -0.0717345 ,
         1.77823747, -0.8237132 ],
       [-1.05458292, -0.84539315, -0.04021111, …, -0.09122515,
         1.77823747, -0.87362627],
       [-0.78012947, -1.00430931, -0.07044252, …, -0.04368215,
         1.75014627, -0.83369581]])
```

[7]:
```
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(x_sc, y, test_size=0.3,␣
 ↪random_state=2)
```

[8]:
```
from sklearn.linear_model import Lasso
lr = Lasso(alpha=10)
```

[9]:
```
lr.fit(xtrain, ytrain)
```

[9]:
```
Lasso(alpha=10)
```

[10]:
```
predict = lr.predict(xtest)
```

[11]:
```
from sklearn.metrics import r2_score, mean_absolute_error
r2_score(ytest, predict)
```

[11]:
```
-0.00033321189864432554
```

[12]:
```
mean_absolute_error(ytest,predict)
```

[12]:
```
np.float64(0.9147942692371251)
```

Practical-4_KNN_Regressor-assignment-done

[1]:
```
import pandas as pd
data = pd.read_csv("./dataset/BostonHousing.csv")
data.head(3)
```

[1]:
```
      crim    zn  indus  chas    nox     rm   age     dis  rad  tax  ptratio  \
0  0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296     15.3
1  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242     17.8
2  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242     17.8

        b  lstat  medv
0  396.90   4.98  24.0
1  396.90   9.14  21.6
2  392.83   4.03  34.7
```

[2]:
```
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range=(0,1))
```

[3]:
```
x_og = data.iloc[:,:-1]
```

[4]:
```
x = sc.fit_transform(x_og)
x
```

[4]:
```
array([[0.00000000e+00, 1.80000000e-01, 6.78152493e-02, …,
         2.87234043e-01, 1.00000000e+00, 8.96799117e-02],
        [2.35922539e-04, 0.00000000e+00, 2.42302053e-01, …,
         5.53191489e-01, 1.00000000e+00, 2.04470199e-01],
        [2.35697744e-04, 0.00000000e+00, 2.42302053e-01, …,
         5.53191489e-01, 9.89737254e-01, 6.34657837e-02],
        …,
        [6.11892474e-04, 0.00000000e+00, 4.20454545e-01, …,
         8.93617021e-01, 1.00000000e+00, 1.07891832e-01],
        [1.16072990e-03, 0.00000000e+00, 4.20454545e-01, …,
         8.93617021e-01, 9.91300620e-01, 1.31070640e-01],
        [4.61841693e-04, 0.00000000e+00, 4.20454545e-01, …,
         8.93617021e-01, 1.00000000e+00, 1.69701987e-01]])
```

[5]:
```
x.shape
```

[5]: (506, 13)

[6]:
```
y = data.iloc[:,-1]
y.shape
```

[6]: (506,)

[7]:
```
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.25,␣
↪random_state = 1)
```

[8]:
```
from sklearn.neighbors import KNeighborsRegressor
model = KNeighborsRegressor(n_neighbors=3)
model.fit(xtrain,ytrain)
```

[8]: KNeighborsRegressor(n_neighbors=3)

[9]:
```
predictions = model.predict(xtest)
```

[10]:
```
from sklearn.metrics import r2_score
r2_score(ytest, predictions)
```

[10]: 0.8069614057252134

0.1 Testing with StandardScaler

[11]:
```
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
```

[12]:
```
ss.fit(x_og)
x = ss.transform(x_og)
```

[13]:
```
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.25,␣
↪random_state = 1)
```

0.2 Find best number of neighbors using gridSearchCV

[14]:
```
from sklearn.model_selection import GridSearchCV
```

[15]:
```
model2 = KNeighborsRegressor()
```

[16]:
```
param = {"n_neighbors":[1,3,5,7,9,11],
         "weights": ["uniform", "distance"]}

scoring = "neg_mean_squared_error"
```

[17]:
```
search = GridSearchCV(model2, param, scoring = scoring, cv = 5)
search.fit(xtrain, ytrain)
```

```python
print(f"Best Params: {search.best_params_}")
```

```
Best Params: {'n_neighbors': 3, 'weights': 'distance'}
```

[18]:
```python
final_model = search.best_estimator_
```

[19]:
```python
predictions2 = final_model.predict(xtest)

from sklearn.metrics import r2_score
r2_score(ytest, predictions2)
```

[19]:
```
0.8638489106891928
```

Practical-5_KNN_Classifier_assignment_done

[1]:
```python
from sklearn.datasets import load_iris
df = load_iris()
x = df.data
y = df.target
```

[2]:
```python
x
```

[2]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
       [5.1, 3.8, 1.5, 0.3],
       [5.4, 3.4, 1.7, 0.2],
       [5.1, 3.7, 1.5, 0.4],
       [4.6, 3.6, 1. , 0.2],
       [5.1, 3.3, 1.7, 0.5],
       [4.8, 3.4, 1.9, 0.2],
       [5. , 3. , 1.6, 0.2],
       [5. , 3.4, 1.6, 0.4],
       [5.2, 3.5, 1.5, 0.2],
       [5.2, 3.4, 1.4, 0.2],
       [4.7, 3.2, 1.6, 0.2],
       [4.8, 3.1, 1.6, 0.2],
```

```
[5.4, 3.4, 1.5, 0.4],
[5.2, 4.1, 1.5, 0.1],
[5.5, 4.2, 1.4, 0.2],
[4.9, 3.1, 1.5, 0.2],
[5. , 3.2, 1.2, 0.2],
[5.5, 3.5, 1.3, 0.2],
[4.9, 3.6, 1.4, 0.1],
[4.4, 3. , 1.3, 0.2],
[5.1, 3.4, 1.5, 0.2],
[5. , 3.5, 1.3, 0.3],
[4.5, 2.3, 1.3, 0.3],
[4.4, 3.2, 1.3, 0.2],
[5. , 3.5, 1.6, 0.6],
[5.1, 3.8, 1.9, 0.4],
[4.8, 3. , 1.4, 0.3],
[5.1, 3.8, 1.6, 0.2],
[4.6, 3.2, 1.4, 0.2],
[5.3, 3.7, 1.5, 0.2],
[5. , 3.3, 1.4, 0.2],
[7. , 3.2, 4.7, 1.4],
[6.4, 3.2, 4.5, 1.5],
[6.9, 3.1, 4.9, 1.5],
[5.5, 2.3, 4. , 1.3],
[6.5, 2.8, 4.6, 1.5],
[5.7, 2.8, 4.5, 1.3],
[6.3, 3.3, 4.7, 1.6],
[4.9, 2.4, 3.3, 1. ],
[6.6, 2.9, 4.6, 1.3],
[5.2, 2.7, 3.9, 1.4],
[5. , 2. , 3.5, 1. ],
[5.9, 3. , 4.2, 1.5],
[6. , 2.2, 4. , 1. ],
[6.1, 2.9, 4.7, 1.4],
[5.6, 2.9, 3.6, 1.3],
[6.7, 3.1, 4.4, 1.4],
[5.6, 3. , 4.5, 1.5],
[5.8, 2.7, 4.1, 1. ],
[6.2, 2.2, 4.5, 1.5],
[5.6, 2.5, 3.9, 1.1],
[5.9, 3.2, 4.8, 1.8],
[6.1, 2.8, 4. , 1.3],
[6.3, 2.5, 4.9, 1.5],
[6.1, 2.8, 4.7, 1.2],
[6.4, 2.9, 4.3, 1.3],
[6.6, 3. , 4.4, 1.4],
[6.8, 2.8, 4.8, 1.4],
[6.7, 3. , 5. , 1.7],
```

```
[6. , 2.9, 4.5, 1.5],
[5.7, 2.6, 3.5, 1. ],
[5.5, 2.4, 3.8, 1.1],
[5.5, 2.4, 3.7, 1. ],
[5.8, 2.7, 3.9, 1.2],
[6. , 2.7, 5.1, 1.6],
[5.4, 3. , 4.5, 1.5],
[6. , 3.4, 4.5, 1.6],
[6.7, 3.1, 4.7, 1.5],
[6.3, 2.3, 4.4, 1.3],
[5.6, 3. , 4.1, 1.3],
[5.5, 2.5, 4. , 1.3],
[5.5, 2.6, 4.4, 1.2],
[6.1, 3. , 4.6, 1.4],
[5.8, 2.6, 4. , 1.2],
[5. , 2.3, 3.3, 1. ],
[5.6, 2.7, 4.2, 1.3],
[5.7, 3. , 4.2, 1.2],
[5.7, 2.9, 4.2, 1.3],
[6.2, 2.9, 4.3, 1.3],
[5.1, 2.5, 3. , 1.1],
[5.7, 2.8, 4.1, 1.3],
[6.3, 3.3, 6. , 2.5],
[5.8, 2.7, 5.1, 1.9],
[7.1, 3. , 5.9, 2.1],
[6.3, 2.9, 5.6, 1.8],
[6.5, 3. , 5.8, 2.2],
[7.6, 3. , 6.6, 2.1],
[4.9, 2.5, 4.5, 1.7],
[7.3, 2.9, 6.3, 1.8],
[6.7, 2.5, 5.8, 1.8],
[7.2, 3.6, 6.1, 2.5],
[6.5, 3.2, 5.1, 2. ],
[6.4, 2.7, 5.3, 1.9],
[6.8, 3. , 5.5, 2.1],
[5.7, 2.5, 5. , 2. ],
[5.8, 2.8, 5.1, 2.4],
[6.4, 3.2, 5.3, 2.3],
[6.5, 3. , 5.5, 1.8],
[7.7, 3.8, 6.7, 2.2],
[7.7, 2.6, 6.9, 2.3],
[6. , 2.2, 5. , 1.5],
[6.9, 3.2, 5.7, 2.3],
[5.6, 2.8, 4.9, 2. ],
[7.7, 2.8, 6.7, 2. ],
[6.3, 2.7, 4.9, 1.8],
[6.7, 3.3, 5.7, 2.1],
```

```
       [7.2, 3.2, 6. , 1.8],
       [6.2, 2.8, 4.8, 1.8],
       [6.1, 3. , 4.9, 1.8],
       [6.4, 2.8, 5.6, 2.1],
       [7.2, 3. , 5.8, 1.6],
       [7.4, 2.8, 6.1, 1.9],
       [7.9, 3.8, 6.4, 2. ],
       [6.4, 2.8, 5.6, 2.2],
       [6.3, 2.8, 5.1, 1.5],
       [6.1, 2.6, 5.6, 1.4],
       [7.7, 3. , 6.1, 2.3],
       [6.3, 3.4, 5.6, 2.4],
       [6.4, 3.1, 5.5, 1.8],
       [6. , 3. , 4.8, 1.8],
       [6.9, 3.1, 5.4, 2.1],
       [6.7, 3.1, 5.6, 2.4],
       [6.9, 3.1, 5.1, 2.3],
       [5.8, 2.7, 5.1, 1.9],
       [6.8, 3.2, 5.9, 2.3],
       [6.7, 3.3, 5.7, 2.5],
       [6.7, 3. , 5.2, 2.3],
       [6.3, 2.5, 5. , 1.9],
       [6.5, 3. , 5.2, 2. ],
       [6.2, 3.4, 5.4, 2.3],
       [5.9, 3. , 5.1, 1.8]])
```

[3]:

```python
y
```

[3]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

[4]:

```python
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(x,y,test_size=0.2, random_state␣
 ↪= 1)
```

[5]:

```python
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier()
model.fit(xtrain,ytrain)
```

[5]:

```
KNeighborsClassifier()
```

[6]:

```python
predictions = model.predict(xtest)
```

[7]:
```python
from sklearn.metrics import accuracy_score
accuracy_score(ytest, predictions)
```

[7]: 1.0

0.1 Find best number of neighbors using gridSearchCV

[8]:
```python
from sklearn.model_selection import GridSearchCV
```

[9]:
```python
model2 = KNeighborsClassifier()
```

[10]:
```python
param = {"n_neighbors" : [1,3,5],
         "weights" : ["uniform", "distance"]}
scoring = "neg_mean_squared_error"
```

[11]:
```python
search = GridSearchCV(model2, param, scoring = scoring, cv = 5)
search.fit(xtrain, ytrain)
print(f"Best Params: {search.best_params_}")
```

```
Best Params: {'n_neighbors': 3, 'weights': 'uniform'}
```

[12]:
```python
final_model = search.best_estimator_
```

[13]:
```python
predictions2 = final_model.predict(xtest)
```

[14]:
```python
from sklearn.metrics import r2_score
r2_score(ytest, predictions2)
```

[14]: 1.0

Practical-6_K-means-R

[1]:
```
library(ggplot2)
library(cluster)
```

[3]:
```
df = as.data.frame(read.csv("./dataset/marks.csv"))
df
```

A data.frame: 17 × 4

Roll_no <int>	English <int>	Maths <int>	Science <int>
1	99	100	100
2	98	99	97
3	92	9	96
4	95	92	94
5	90	100	96
6	80	75	82
7	75	83	80
8	72	73	74
9	71	82	76
10	73	74	76
11	34	32	28
12	26	28	30
13	32	30	31
14	98	97	98
15	30	29	29
16	78	75	78
17	100	99	100

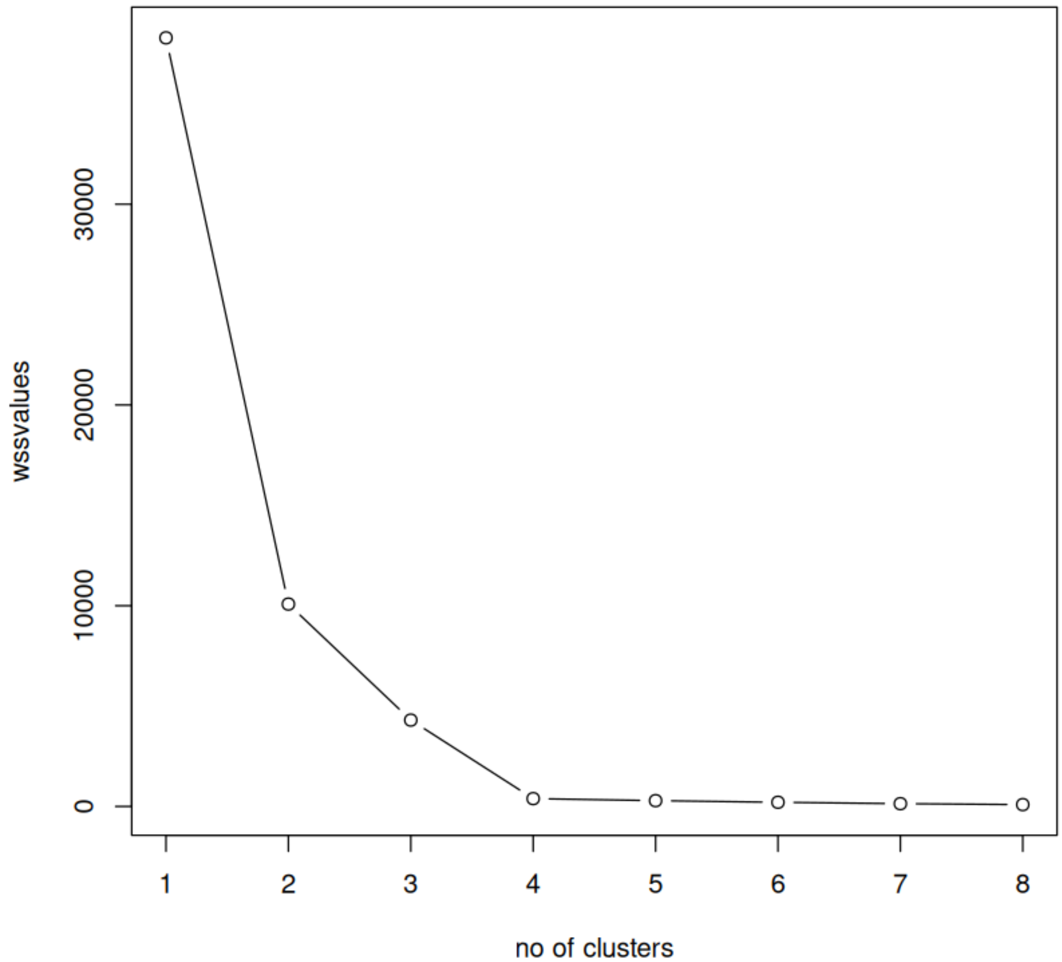
[4]:
```
kmdata = df[,2:4]
kmdata
```

A data.frame: 17 × 3

English <int>	Maths <int>	Science <int>
99	100	100
98	99	97
92	9	96
95	92	94
90	100	96
80	75	82
75	83	80
72	73	74
71	82	76
73	74	76
34	32	28
26	28	30
32	30	31
98	97	98
30	29	29
78	75	78
100	99	100

[5]:

```
wss = numeric(8)
for (k in 1:8) wss[k] = sum(kmeans(kmdata,centers = k, nstart = 25)$withinss)
plot(1:8,wss,type="b",xlab="no of clusters", ylab = "wssvalues")
```



[6]:
```
final_model = kmeans(kmdata,3,nstart = 25)
final_model
```

```
K-means clustering with 3 clusters of sizes 4, 1, 12

Cluster means:
  English    Maths  Science
1   30.50 29.75000 29.50000
2   92.00  9.00000 96.00000
3   85.75 87.41667 87.58333

Clustering vector:
 [1] 3 3 2 3 3 3 3 3 3 3 1 1 1 3 1 3 3
```

```
Within cluster sum of squares by cluster:
[1]   48.750    0.000 4254.083
 (between_SS / total_SS =  88.8 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
[6] "betweenss"    "size"         "iter"         "ifault"
```

Practical-7_SVM_classifier

[1]:
```
import pandas as pd
```

[4]:
```
df = pd.read_csv("./dataset/diabetes.csv")
```

[5]:
```
df.head(3)
```

[5]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
```

[6]:
```
df.sample(2)
```

[6]:
```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
442            4      117             64             27      120  33.2
311            0      106             70             37      148  39.4

     DiabetesPedigreeFunction  Age  Outcome
442                     0.230   24        0
311                     0.605   22        0
```

[7]:
```
df.shape
```

[7]:
```
(768, 9)
```

[8]:
```
df["Outcome"].value_counts()
```

[8]:
```
Outcome
0    500
1    268
Name: count, dtype: int64
```

[9]:
```
X = df.iloc[:,:-1]
X.shape
```

[9]:
```
(768, 8)
```

[10]:
```
y = df.iloc[:,-1]
y.shape
```

[10]:
```
(768,)
```

[11]:
```
X
```

[11]:
```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0              6      148             72             35        0  33.6
1              1       85             66             29        0  26.6
2              8      183             64              0        0  23.3
3              1       89             66             23       94  28.1
4              0      137             40             35      168  43.1
..           …        …            …            …      …    …
763           10      101             76             48      180  32.9
764            2      122             70             27        0  36.8
765            5      121             72             23      112  26.2
766            1      126             60              0        0  30.1
767            1       93             70             31        0  30.4

     DiabetesPedigreeFunction  Age
0                       0.627   50
1                       0.351   31
2                       0.672   32
3                       0.167   21
4                       2.288   33
..                        …  …
763                     0.171   63
764                     0.340   27
765                     0.245   30
766                     0.349   47
767                     0.315   23

[768 rows x 8 columns]
```

[12]:
```
y
```

[12]:
```
0      1
1      0
2      1
3      0
4      1
      ..
```

```
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

[13]:
```
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(X, y, test_size = 0.25,␣
↪random_state=1)
```

[14]:
```
xtrain.shape, xtest.shape, ytrain.shape, ytest.shape
```

[14]:
```
((576, 8), (192, 8), (576,), (192,))
```

[15]:
```
from sklearn.svm import SVC

model1 = SVC(kernel="linear", degree=2, coef0=1.5)
# default will be linear kernal

model1.fit(xtrain, ytrain)
```

[15]:
```
SVC(coef0=1.5, degree=2, kernel='linear')
```

[16]:
```
predictions = model1.predict(xtest)
```

[17]:
```
from sklearn.metrics import accuracy_score, r2_score, confusion_matrix

accuracy_score(ytest, predictions)
```

[17]:
```
0.78125
```

[18]:
```
confusion_matrix(ytest, predictions)
```

[18]:
```
array([[108,  15],
       [ 27,  42]])
```

[19]:
```
model2 = SVC(kernel="poly", degree=2, coef0=1.5)
model2.fit(xtrain, ytrain)
```

[19]:
```
SVC(coef0=1.5, degree=2, kernel='poly')
```

[20]:
```
predictions2 = model2.predict(xtest)
accuracy_score(ytest, predictions2)
```

[20]:
```
0.796875
```

[21]:
```
confusion_matrix(ytest, predictions2)
```

[21]:
```
array([[113,  10],
       [ 29,  40]])
```

[22]:
```
model3 = SVC(kernel="rbf", degree=2, coef0=1.5)
model3.fit(xtrain, ytrain)
```

[22]:
```
SVC(coef0=1.5, degree=2)
```

[23]:
```
predictions3 = model3.predict(xtest)
accuracy_score(ytest, predictions3)
```

[23]:
```
0.7708333333333334
```

Practical-8_Ensemble_Bagging

[1]:
```python
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
```

[2]:
```python
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import BaggingClassifier
```

[3]:
```python
df = load_breast_cancer()
x=df.data
y=df.target
```

[4]:
```python
xtrain, xtest,  ytrain, ytest = train_test_split(x,y,test_size=0.
 ↪25,random_state=1)
```

[5]:
```python
pipeline= make_pipeline(StandardScaler(),LogisticRegression(random_state=1))
```

[6]:
```python
pipeline.fit(xtrain,ytrain)
```

[6]:
```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('logisticregression', LogisticRegression(random_state=1))])
```

[7]:
```python
print("Score:",pipeline.score(xtest,ytest))
pred1=pipeline.predict(xtest)
from sklearn.metrics import accuracy_score
print("Accuracy Score:",accuracy_score(ytest,pred1))
```

```
Score: 0.9790209790209791
Accuracy Score: 0.9790209790209791
```

[8]:
```python
bgclassifier=␣
 ↪BaggingClassifier(estimator=pipeline,n_estimators=100,max_features=10,max_samples=100,␣
 ↪random_state=1)
```

[9]:
```python
bgclassifier.fit(xtrain,ytrain)
```

[9]:
```
BaggingClassifier(estimator=Pipeline(steps=[('standardscaler',
                                              StandardScaler()),
```

```
                                               ('logisticregression',
LogisticRegression(random_state=1))]),
                  max_features=10, max_samples=100, n_estimators=100,
                  random_state=1)
```

[10]:
```
print(bgclassifier.score(xtest,ytest))
```

```
0.958041958041958
```

[ ]:
```

```