Government Engineering College, Modasa
B.E. – Computer Engineering (Semester – VII)
3170724 – Machine Learning

LIST OF PRACTICALS

<table>
<tr><th>Sr No.</th><th>Practicals</th></tr>
<tr><td>1.</td><td>Given the following vectors:
A = [1, 2, 3, 4, 5, 6, 7, 8, 9 10]
B = [4, 8, 12, 16, 20, 24, 28, 32, 36, 40]
C = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
Ex. 1: Find the arithmetic mean of vector A, B and C
Ex. 2: Find the variance of the vector A, B and C
Ex. 3: Find the Euclidean distance between vector A and B
Ex. 4: Find the correlation between vectors A & B and A & C</td></tr>
<tr><td>2.</td><td>Load breast cancer dataset and perform classification using Euclidean distance. Use 70% data as training and 30% for testing.</td></tr>
<tr><td>3.</td><td>Repeat the above experiment with 10-fold cross validation and find the standard deviation in accuracy.</td></tr>
<tr><td>4.</td><td>Repeat the experiment 2 and build the confusion matrix. Also derive Precision, Recall and Specificity of the algorithm.</td></tr>
<tr><td>5.</td><td>Predict the class for X = < Sunny, Cool, High, Strong > using Naïve Bayes Classifier for given data
$$P(C \mid X) = \frac{P(X \mid C).\,P(C)}{P(X)}$$
<table>
<tr><th>#</th><th>Outlook</th><th>Temp.</th><th>Humidity</th><th>Windy</th><th>Play</th></tr>
<tr><td>D1</td><td>Sunny</td><td>Hot</td><td>High</td><td>False</td><td>No</td></tr>
<tr><td>D2</td><td>Sunny</td><td>Hot</td><td>High</td><td>True</td><td>No</td></tr>
<tr><td>D3</td><td>Overcast</td><td>Hot</td><td>High</td><td>False</td><td>Yes</td></tr>
<tr><td>D4</td><td>Rainy</td><td>Mild</td><td>High</td><td>False</td><td>Yes</td></tr>
<tr><td>D5</td><td>Rainy</td><td>Cool</td><td>Normal</td><td>False</td><td>Yes</td></tr>
<tr><td>D6</td><td>Rainy</td><td>Cool</td><td>Normal</td><td>True</td><td>No</td></tr>
<tr><td>D7</td><td>Overcast</td><td>Cool</td><td>Normal</td><td>True</td><td>Yes</td></tr>
</table></td></tr>
</table>

D8	Sunny	Mild	High	False	No
D9	Sunny	Cool	Normal	False	Yes
D10	Rainy	Mild	Normal	False	Yes
D11	Sunny	Mild	Normal	True	Yes
D12	Overcast	Mild	High	True	Yes
D13	Overcast	Hot	Normal	False	Yes
D14	Rainy	Mild	High	True	No

Ans: Label = NO

6. For the data given in Exercise 5, find the splitting attribute at first level:

Information Gain: $I(P, N) = -\frac{P}{S}\log_2\frac{P}{S} \; -\frac{N}{S}\log_2\frac{N}{S} = 0.940$

$$Entropy: E(Outlook) = \sum_{i=1}^{v}\frac{P_i + N_i}{P + N}I(P_i,\ N_i)) = \ 0.694$$

$Gain\ (Outlook) = I(P,\ N) - E(Outlook) = \ 0.246$

Ans:

Attribute	Gain
Outlook	**0.246**
Temperature	0.029
Humidity	0.151
Windy	0.048

7. Generate and test decision tree for the dataset in exercise 5

8. Find the clusters for following data with k = 2: Start with points 1 and 4 as two separate clusters.

i	A	B
1	1.0	1.0
2	1.5	2.0
3	3.0	4.0
4	5.0	7.0
5	3.5	5.0
6	4.5	5.0
7	3.5	4.5

Ans:

i	Point
C_1	1, 2
C_2	3, 4, 5, 6, 7

9.	Find following statistics for the data given in Exercise 1 $$Within\ Class\ Scatter: S_W = \sum_{i=1}^{C} \sum_{x \in w_i} (x - m_i)(x - m_i)^T$$ $$Between\ Class\ Scatter: S_B = \sum_{i=1}^{C} n_i (m_i - m)(m_i - m)^T$$ $$Total\ Scatter: S_T = \sum_{i=1}^{M} (x_i - m)(x_i - m)^T$$
10.	Given the following vectors: X = [340, 230, 405, 325, 280, 195, 265, 300, 350, 310]; %sale Y = [71, 65, 83, 74, 67, 56, 57, 78, 84, 65]; **Ex. 1:** Find the Linear Regression model for independent variable X and dependent variable Y. **Ex. 2:** Predict the value of y for x = 250. Also find the residual for y4.

Practical-1

AIM: Given the following vectors:

A = [1, 2, 3, 4, 5, 6, 7, 8, 9 10]

B = [4, 8, 12, 16, 20, 24, 28, 32, 36, 40]

C = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

Ex. 1: Find the arithmetic mean of vector A, B and C

```
A = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
B = [4, 8, 12, 16, 20, 24, 28, 32, 36, 40]
C = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

```
meanA= sum(A)/len(A)
meanB= sum(B)/len(B)
meanC= sum(C)/len(C)
print("Mean of vector A: ",meanA)
print("Mean of vector B: ",meanB)
print("Mean of vector C: ",meanC)
```

In [1]:
```
A = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
B = [4, 8, 12, 16, 20, 24, 28, 32, 36, 40]
C = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

meanA= sum(A)/len(A)
meanB= sum(B)/len(B)
meanC= sum(C)/len(C)
print("Mean of vector A: ",meanA)
print("Mean of vector B: ",meanB)
print("Mean of vector C: ",meanC)
```

```
Mean of vector A:  5.5
Mean of vector B:  22.0
Mean of vector C:  5.5
```

Ex. 2: Find the variance of the vector A, B and C

```
def sumSquaredDeviation(X,m):
    s= []
    for i in X:
        s.append((i-m)**2)

    return sum(s)

varA= sumSquaredDeviation(A,meanA)/len(A)
varB= sumSquaredDeviation(B,meanB)/len(B)
varC= sumSquaredDeviation(C,meanC)/len(C)
print("Variance of vector A: ",varA)
print("Variance of vector B: ",varB)
print("Variance of vector C: ",varC)
```

```
def sumSquaredDeviation(X,m):
    s= []
    for i in X:
        s.append((i-m)**2)

    return sum(s)

varA= sumSquaredDeviation(A,meanA)/len(A)
varB= sumSquaredDeviation(B,meanB)/len(B)
varC= sumSquaredDeviation(C,meanC)/len(C)
print("Variance of vector A: ",varA)
print("Variance of vector B: ",varB)
print("Variance of vector C: ",varC)
```

```
Variance of vector A:  8.25
Variance of vector B:  132.0
Variance of vector C:  8.25
```

Ex. 3: Find the Euclidean distance between vector A and B

```
def euclidDistance(x,y):
    s=[]
    for i in range(len(x)):
        s.append((x[i]-y[i])**2)

    return sum(s)**0.5

ed= euclidDistance(A,B)
print("Distance between vector A and vector B: ",ed)
```

```
def euclidDistance(x,y):
    s=[]
    for i in range(len(x)):
        s.append((x[i]-y[i])**2)

    return sum(s)**0.5

ed= euclidDistance(A,B)
print("Distance between vector A and vector B: ",ed)
```

```
Distance between vector A and vector B:  58.86425061104575
```

Ex. 4: Find the correlation between vectors A & B and A & C

```
def correlation(x,y,mx,my):
    s=[]
    for i in range(len(x)):
        s.append((x[i]-mx)*(y[i]-my))

    return sum(s)

stdA= sumSquaredDeviation(A,meanA)**0.5
stdB= sumSquaredDeviation(B,meanB)**0.5
stdC= sumSquaredDeviation(C,meanC)**0.5

rAB= correlation(A,B,meanA,meanB)/(stdA*stdB)
rAC= correlation(A,C,meanA,meanC)/(stdA*stdC)
print("Correlation of vector A & B :",rAB)
print("Correlation of vector A & C :",rAC)
```

```
In [4]: def correlation(x,y,mx,my):
            s=[]
            for i in range(len(x)):
                s.append((x[i]-mx)*(y[i]-my))

            return sum(s)

        stdA= sumSquaredDeviation(A,meanA)**0.5
        stdB= sumSquaredDeviation(B,meanB)**0.5
        stdC= sumSquaredDeviation(C,meanC)**0.5

        rAB= correlation(A,B,meanA,meanB)/(stdA*stdB)
        rAC= correlation(A,C,meanA,meanC)/(stdA*stdC)
        print("Correlation of vector A & B :",rAB)
        print("Correlation of vector A & C :",rAC)
```

```
Correlation of vector A & B : 1.0
Correlation of vector A & C : -1.0
```

Practical-2

AIM: Load breast cancer dataset and perform classification using Euclidean distance. Use 70% data as training and 30% for testing.

Program:

```
from sklearn.datasets import load_breast_cancer

import pandas as pd
d = load_breast_cancer(as_frame=True)
df= pd.DataFrame(d['data'])
df
```

```
import pandas as pd
d = load_breast_cancer(as_frame=True)
df= pd.DataFrame(d['data'])
df
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	wor smoothne
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	...	25.380	17.33	184.60	2019.0	0.162
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	...	24.990	23.41	158.80	1956.0	0.123
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	...	23.570	25.53	152.50	1709.0	0.144
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	...	14.910	26.50	98.87	567.7	0.209
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	...	22.540	16.67	152.20	1575.0	0.137
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	...	25.450	26.40	166.10	2027.0	0.141
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	...	23.690	38.25	155.00	1731.0	0.116
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	...	18.980	34.12	126.70	1124.0	0.113
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	...	25.740	39.42	184.60	1821.0	0.165
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	...	9.456	30.37	59.16	268.6	0.089

569 rows × 30 columns

```
data = load_breast_cancer()
features= data['data']
target= data['target']
print(features)
print(target)
```

```
data = load_breast_cancer()
features= data['data']
target= data['target']
print(features)
print(target)
```

```
[[1.799e+01 1.038e+01 1.228e+02 ... 2.654e-01 4.601e-01 1.189e-01]
 [2.057e+01 1.777e+01 1.329e+02 ... 1.860e-01 2.750e-01 8.902e-02]
 [1.969e+01 2.125e+01 1.300e+02 ... 2.430e-01 3.613e-01 8.758e-02]
 ...
 [1.660e+01 2.808e+01 1.083e+02 ... 1.418e-01 2.218e-01 7.820e-02]
 [2.060e+01 2.933e+01 1.401e+02 ... 2.650e-01 4.087e-01 1.240e-01]
 [7.760e+00 2.454e+01 4.792e+01 ... 0.000e+00 2.871e-01 7.039e-02]]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 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 1 1 1 1 1 0 0 1 0 0 1 1 1 1 0 1 0 0 1 1 1 1 0 1 0 0
 1 0 1 0 0 1 1 1 0 0 1 0 0 0 1 1 1 0 1 1 0 0 1 1 1 0 0 1 1 1 1 0 1 1 0 1 1
 1 1 1 1 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 1 1 0 1
 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 1 1 1 1 0 1 1 0 0 0 1 0
 1 0 1 1 1 0 1 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 1 1 0 0 1 1
 1 0 1 1 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 1 1 1 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 1 0 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1
 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 0 0 0 1 1
 1 1 0 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0
 0 1 0 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1
 1 0 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 1 1 1 1 0 1 1
 0 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1
 1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1 0 0 1 0 1 0 1 1 1 1 1 0 1 1 0 1 0 1 0 0
 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 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 0 0 0 0 0 0 1]
```

```
import random
# train = 70% of total= 569
# test= 30% of total =170

r= random.sample(range(569),170)
random.seed(42)

train= []
test=[]

for i in range(569):
   if i in r:
      test.append((i,features[i]))
   else:
      train.append((i,features[i]))

def euclidDistance(x,y):
   s=[]
   for i in range(len(x)):
      s.append((x[i]-y[i])**2)

   return sum(s)**0.5

res=[]
for i in test:
   m= 100000
   for j in train:
      dij= euclidDistance(i[1],j[1])
      if dij < m:
         m= dij
         ind= j[0]

   res.append((i[0],target[ind]))

c=0
for i in res:
```

```
    if target[i[0]]==i[1]:
        c+=1
acc= (c/170)*100
print("Correct predictions: ",c)
print("Accuracy: ",acc)
```

```
import random
# train = 70% of total= 569
# test= 30% of total =170

r= random.sample(range(569),170)
random.seed(42)

train= []
test=[]

for i in range(569):
    if i in r:
        test.append((i,features[i]))
    else:
        train.append((i,features[i]))

def euclidDistance(x,y):
    s=[]
    for i in range(len(x)):
        s.append((x[i]-y[i])**2)

    return sum(s)**0.5

res=[]
for i in test:
    m= 100000
    for j in train:
        dij= euclidDistance(i[1],j[1])
        if dij < m:
            m= dij
            ind= j[0]

    res.append((i[0],target[ind]))

c=0
for i in res:
    if target[i[0]]==i[1]:
        c+=1
acc= (c/170)*100
print("Correct predictions: ",c)
print("Accuracy: ",acc)
```

```
Correct predictions:  160
Accuracy:  94.11764705882352
```

Practical-3

AIM: Repeat the above experiment with 10-fold cross validation and find the standard deviation in accuracy.

Program:

```
from sklearn.datasets import load_breast_cancer
import pandas as pd

d = load_breast_cancer(as_frame=True)
X= pd.DataFrame(d['data'])
y= pd.DataFrame(d['target'])
X
```

```
d = load_breast_cancer(as_frame=True)
X= pd.DataFrame(d['data'])
y= pd.DataFrame(d['target'])
X
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	wor smoothne
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	...	25.380	17.33	184.60	2019.0	0.162
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	...	24.990	23.41	158.80	1956.0	0.123
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	...	23.570	25.53	152.50	1709.0	0.144
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	...	14.910	26.50	98.87	567.7	0.209
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	...	22.540	16.67	152.20	1575.0	0.137
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	...	25.450	26.40	166.10	2027.0	0.141
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	...	23.690	38.25	155.00	1731.0	0.116
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	...	18.980	34.12	126.70	1124.0	0.113
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	...	25.740	39.42	184.60	1821.0	0.165
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	...	9.456	30.37	59.16	268.6	0.089

569 rows × 30 columns

```
def euclidDistance(x,y):
    s=[]

    for i in range(len(x)):
        s.append((x[i]-y[i])**2)

    return sum(s)**0.5

# to store accuracies of different folds
accuracy=[]

start=0
end=569

# 569/10= 56.9 => 57
# each fold contains 57 or 56 rows
begin=0
```

```
finish=57

# generating 10 folds
for fold in range(10):

    if finish > end:
        finish= finish - (finish-end)
    X_test= X.iloc[begin:finish]
    X_train= pd.concat( [X.iloc[start:begin] , X.iloc[finish:end]])
    y_test= y.iloc[begin:finish]
    y_train= pd.concat( [y.iloc[start:begin] , y.iloc[finish:end]] )

    print("FOR fold=",fold+1)

    print("X_train :")
    display(X_train)
    print("y_train :")
    display(y_train)

    print("X_test :")
    display(X_test)
    print("y_test :")
    display(y_test)


    res=[]
    for i in range(len(X_test)):
        m= 100000
        for j in range(len(X_train)):
            dij= euclidDistance(X_test.iloc[i],X_train.iloc[j])
            if dij < m:
                m= dij
                ind=j
        res.append(y_train.iloc[ind][0])

    c=0
    for i in range(len(res)):
        if y_test.iloc[i][0]==res[i]:
            c+=1
            acc= (c/len(res))*100

    print("Correct predictions for Fold",fold+1," : ",c)
    print("Accuracy for Fold",fold+1," : ",acc)
    accuracy.append(acc)

    begin= finish
   finish+= 57
```

```
FOR fold= 1
X_train :
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	smootl
57	14.710	21.59	95.55	656.9	0.11370	0.13650	0.129300	0.081230	0.2027	0.06758	...	17.870	30.70	115.70	985.5	0.
58	13.050	19.31	82.61	527.2	0.08060	0.03789	0.000692	0.004167	0.1819	0.05501	...	14.230	22.25	90.24	624.1	0.
59	8.618	11.79	54.34	224.5	0.09752	0.05272	0.020610	0.007799	0.1683	0.07187	...	9.507	15.40	59.90	274.9	0.
60	10.170	14.88	64.55	311.9	0.11340	0.08061	0.010840	0.012900	0.2743	0.06960	...	11.020	17.45	69.86	368.6	0.
61	8.598	20.98	54.66	221.8	0.12430	0.08963	0.030000	0.009259	0.1828	0.06757	...	9.565	27.04	62.06	273.9	0.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
564	21.560	22.39	142.00	1479.0	0.11100	0.11590	0.243900	0.138900	0.1726	0.05623	...	25.450	26.40	166.10	2027.0	0.
565	20.130	28.25	131.20	1261.0	0.09780	0.10340	0.144000	0.097910	0.1752	0.05533	...	23.690	38.25	155.00	1731.0	0.

```
for i in range(10):
    print("Accuracy of fold ",i+1," : ",accuracy[i])

mean_acc= sum(accuracy)/10
print("Mean of accuracies : ",mean_acc)

def sumSquaredDeviation(X,m):
    s= []
    for i in X:
        s.append((i-m)**2)

    return sum(s)

var_acc= sumSquaredDeviation(accuracy,mean_acc)/10
print("Variance of accuracies : ",var_acc)

std_acc= var_acc**0.5
print("Standard Deviation of accuracies : ",std_acc)
```

```
Accuracy of fold  1  :  80.7017543859649
Accuracy of fold  2  :  91.22807017543859
Accuracy of fold  3  :  92.98245614035088
Accuracy of fold  4  :  85.96491228070175
Accuracy of fold  5  :  94.73684210526315
Accuracy of fold  6  :  96.49122807017544
Accuracy of fold  7  :  91.22807017543859
Accuracy of fold  8  :  94.73684210526315
Accuracy of fold  9  :  87.71929824561403
Accuracy of fold  10  :  96.42857142857143
Mean of accuracies :  91.2218045112782
Variance of accuracies :  23.32621183284026
Standard Deviation of accuracies :  4.829721713809219
```

Practical-4

AIM: Repeat the experiment 2 and build the confusion matrix. Also derive Precision, Recall and Specificity of the algorithm.

Program:

from sklearn.datasets import load_breast_cancer

import pandas as pd
d = load_breast_cancer(as_frame=True)
df= pd.DataFrame(d['data'])
df

```
import pandas as pd
d = load_breast_cancer(as_frame=True)
df= pd.DataFrame(d['data'])
df
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	wo smoothne
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	...	25.380	17.33	184.60	2019.0	0.162
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	...	24.990	23.41	158.80	1956.0	0.123
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	...	23.570	25.53	152.50	1709.0	0.144
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	...	14.910	26.50	98.87	567.7	0.209
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	...	22.540	16.67	152.20	1575.0	0.137
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	...	25.450	26.40	166.10	2027.0	0.141
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	...	23.690	38.25	155.00	1731.0	0.116
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	...	18.980	34.12	126.70	1124.0	0.113
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	...	25.740	39.42	184.60	1821.0	0.165
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	...	9.456	30.37	59.16	268.6	0.089

569 rows × 30 columns

data = load_breast_cancer()
features= data['data']
target= data['target']
print(features)
print(target)

```
data = load_breast_cancer()
features= data['data']
target= data['target']
print(features)
print(target)
```

```
[[1.799e+01 1.038e+01 1.228e+02 ... 2.654e-01 4.601e-01 1.189e-01]
 [2.057e+01 1.777e+01 1.329e+02 ... 1.860e-01 2.750e-01 8.902e-02]
 [1.969e+01 2.125e+01 1.300e+02 ... 2.430e-01 3.613e-01 8.758e-02]
 ...
 [1.660e+01 2.808e+01 1.083e+02 ... 1.418e-01 2.218e-01 7.820e-02]
 [2.060e+01 2.933e+01 1.401e+02 ... 2.650e-01 4.087e-01 1.240e-01]
 [7.760e+00 2.454e+01 4.792e+01 ... 0.000e+00 2.871e-01 7.039e-02]]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 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 1 1 1 1 1 0 0 1 0 0 1 1 1 1 0 1 0 0 1 1 1 1 0 1 0 0
 1 0 1 0 0 1 1 1 0 0 1 0 0 0 1 1 1 0 1 1 0 0 1 1 1 0 0 1 1 1 1 0 1 1 0 1 1
 1 1 1 1 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 1 1 0 1
 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 1 1 1 1 0 1 1 0 0 0 1 0
 1 0 1 1 1 0 1 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 1 1 0 0 1 1
 1 0 1 1 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 1 1 1 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 1 0 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1
 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 0 0 0 1 1
 1 1 0 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0
 0 1 0 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1
 1 0 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 1 1 1 1 0 1 1
 0 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1
 1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1 0 0 1 0 1 0 1 1 1 1 1 0 1 1 0 1 0 1 0 0
 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 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 0 0 0 0 0 0 1]
```

```
import random

# train = 80% of total= 569
# test= 20% of total =114
# random.seed(42)

r= random.sample(range(569),114)


train= []
test=[]

for i in range(569):
    if i in r:
        test.append((i,features[i]))
    else:
        train.append((i,features[i]))

def euclidDistance(x,y):
    s=[]
    for i in range(len(x)):
        s.append((x[i]-y[i])**2)

    return sum(s)**0.5

res=[]
for i in test:
    m= 100000
    for j in train:
        dij= euclidDistance(i[1],j[1])
        if dij < m:
            m= dij
            ind= j[0]
```

```
    # i[0] represents the index of target variable in dataset
    # target[ind] represents the predicted value of target variable
    res.append((i[0],target[ind]))

c=0
tp=0
tn=0
fp=0
fn=0
for i in res:
    if target[i[0]]==i[1] and i[1]==0:
        tp+=1
        c+=1
    elif target[i[0]]==i[1] and i[1]==1:
        tn+=1
        c+=1
    elif target[i[0]]!=i[1] and i[1]==0:
        fp+=1
    else:
        fn+=1
acc= (c/114)*100

print("Correct predictions: ",c)

print("Accuracy: ",acc)
```

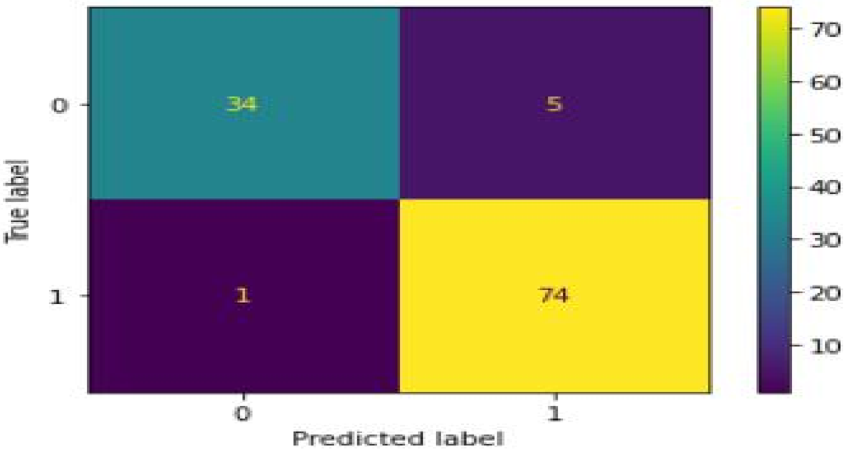
```
Correct predictions:  108
Accuracy:  94.73684210526315
```

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

y_test= []
y_predict=[]
for value in res:
    y_predict.append(value[1])
    y_test.append(target[value[0]])


cm = confusion_matrix(y_test,y_predict)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
```

```
print("True positive: ",tp)
print("True negative: ",tn)
print("False positive: ",fp)
print("False negative: ",fn)


print("\n\n")
print("Precision: ",(tp/(tp+fp))*100)
print("Recall: ",(tp/(tp+fn))*100)
print("Specificity: ",(tn/(tn+fp))*100)
```

```
True positive:  34
True negative:  74
False positive:  1
False negative:  5
```

```
Precision:  97.14285714285714
Recall:  87.17948717948718
Specificity:  98.66666666666667
```



Practical-5

AIM: Predict the class for X = < Sunny, Cool, High, Strong > using Naïve Bayes Classifier for given data.

Program:

```
import pandas as pd

data=
{'outlook':['Sunny','Sunny','Overcast','Rainy','Rainy','Rainy','Overcast','Sunny','Sunny',
'Rainy','Sunny','Overcast','Overcast','Rainy'],
'temp':['Hot','Hot','Hot','Mild','Cool','Cool','Cool','Mild','Cool','Mild','Mild','Mild','Hot','
Mild'],
'humidity':['High','High','High','High','Normal','Normal','Normal','High','Normal','Nor
mal','Normal','High','Normal','High'],
'windy':['False','True','False','False','False','True','True','False','False','False','True','True
','False','True'],
'play':['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Yes','Yes','No']}

df= pd.DataFrame(data)
df
```

	outlook	temp	humidity	windy	play
0	Sunny	Hot	High	False	No
1	Sunny	Hot	High	True	No
2	Overcast	Hot	High	False	Yes
3	Rainy	Mild	High	False	Yes
4	Rainy	Cool	Normal	False	Yes
5	Rainy	Cool	Normal	True	No
6	Overcast	Cool	Normal	True	Yes
7	Sunny	Mild	High	False	No
8	Sunny	Cool	Normal	False	Yes
9	Rainy	Mild	Normal	False	Yes
10	Sunny	Mild	Normal	True	Yes
11	Overcast	Mild	High	True	Yes
12	Overcast	Hot	Normal	False	Yes
13	Rainy	Mild	High	True	No

```
# counting no. of rows with given feature having specific value and target
# ex: temp = Hot , play= Yes  count= 2
# ex: outlook= Sunny, play= No  count= 3
def countYesNo(feature,value,target):
    c= df.loc[(df[feature]==value) & (df['play']==target)]['play'].count()
    return c

print(countYesNo('outlook','Sunny','Yes'))
print(countYesNo('outlook','Sunny','No'))
```

```
# counting no. of rows with given feature having specific value and target
# ex: temp = Hot , play= Yes  count= 2
# ex: outlook= Sunny, play= No  count= 3

def countYesNo(feature,value,target):
    c= df.loc[(df[feature]==value) & (df['play']==target)]['play'].count()
    return c

print(countYesNo('outlook','Sunny','Yes'))
print(countYesNo('outlook','Sunny','No'))
```

```
2
3
```

```
# Calculating probability of given feature with specific value and target(yes/no)
def probability(feature,value,target):
    c= countYesNo(feature,value,target)
    if target=='Yes':
        prob= c/num_yes
    else:
        prob= c/num_no
    return prob

# number of yes/no in dataset
num_yes= df.loc[(df['play']=='Yes')]['play'].count()
num_no= df.loc[(df['play']=='No')]['play'].count()

print(probability('outlook','Sunny','Yes'))
print(probability('outlook','Sunny','No'))
```

```
# Calculating probability of given feature with specific value and target(yes/no)

def probability(feature,value,target):
    c= countYesNo(feature,value,target)
    if target=='Yes':
        prob= c/num_yes
    else:
        prob= c/num_no
    return prob

# number of yes/no in dataset
num_yes= df.loc[(df['play']=='Yes')]['play'].count()
num_no= df.loc[(df['play']=='No')]['play'].count()

print(probability('outlook','Sunny','Yes'))
print(probability('outlook','Sunny','No'))
```

```
0.2222222222222222
0.6
```

```
outlook= input("What kind of outlook (Sunny,Overcast,Rainy) : ")
temp= input("How's temperatue (Hot,Mild,Cool) : ")
humidity= input("How's humidity (High,Normal) : ")
windy= input("Is it Windy ?? (True,False) : ")
```

```
outlook= input("What kind of outlook (Sunny,Overcast,Rainy) : ")
temp= input("How's temperatue (Hot,Mild,Cool) : ")
humidity= input("How's humidity (High,Normal) : ")
windy= input("Is it Windy ?? (True,False) : ")
```

```
What kind of outlook (Sunny,Overcast,Rainy) : Sunny
How's temperatue (Hot,Mild,Cool) : Cool
How's humidity (High,Normal) : High
Is it Windy ?? (True,False) : True
```

```
prob_yes= num_yes/14
prob_no= num_no/14


yes_today= probability('outlook',outlook,'Yes')*probability('temp',temp,'Yes')*probability('humidity',humidity,'Yes')*probability('windy',windy,'Yes')*prob_yes
no_today= probability('outlook',outlook,'No')*probability('temp',temp,'No')*probability('humidity',humidity,'No')*probability('windy',windy,'No')*prob_no

prob_play= yes_today/(yes_today+no_today)
prob_noPlay= no_today/(yes_today+no_today)

print("Possibilty of playing today: ",prob_play*100,"%")
print("Possibilty of not playing today: ",prob_noPlay*100,"%")

if prob_play > prob_noPlay:
    print("Game can be Played today for given weather conditions",(outlook,temp,humidity,windy))
else:
   print("Game cannot be Played today for given weather condtions",(outlook,temp,humidity,windy))
```

```
Possibilty of playing today:  20.458265139116204 %
Possibilty of not playing today:  79.54173486088379 %
Game cannot be Played today for given weather condtions ('Sunny', 'Cool', 'High', 'True')
```

Practical-6

AIM: For the data given in Exercise 5, find the splitting attribute at first level:

Program:

```
import pandas as pd

data=
{'outlook':['Sunny','Sunny','Overcast','Rainy','Rainy','Rainy','Overcast','Sunny','Sunny',
'Rainy','Sunny','Overcast','Overcast','Rainy'],
'temp':['Hot','Hot','Hot','Mild','Cool','Cool','Cool','Mild','Cool','Mild','Mild','Mild','Hot','
Mild'],
'humidity':['High','High','High','High','Normal','Normal','Normal','High','Normal','Nor
mal','Normal','High','Normal','High'],
'windy':['False','True','False','False','False','True','True','False','False','False','True','True
','False','True'],
'play':['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Yes','Yes','No']}

df= pd.DataFrame(data)
df
```

	outlook	temp	humidity	windy	play
0	Sunny	Hot	High	False	No
1	Sunny	Hot	High	True	No
2	Overcast	Hot	High	False	Yes
3	Rainy	Mild	High	False	Yes
4	Rainy	Cool	Normal	False	Yes
5	Rainy	Cool	Normal	True	No
6	Overcast	Cool	Normal	True	Yes
7	Sunny	Mild	High	False	No
8	Sunny	Cool	Normal	False	Yes
9	Rainy	Mild	Normal	False	Yes
10	Sunny	Mild	Normal	True	Yes
11	Overcast	Mild	High	True	Yes
12	Overcast	Hot	Normal	False	Yes
13	Rainy	Mild	High	True	No

```
import math

def entropy_target():
    p= df.loc[(df['play']=='Yes')]['play'].count()
    n= df.loc[(df['play']=='No')]['play'].count()
    total= df['play'].count()

    e= -((p/total)*math.log2((p/total))-((n/total)*math.log2((n/total)))

    return e

E_play= entropy_target()
E_play
```

```
import math

def entropy_target():
    p= df.loc[(df['play']=='Yes')]['play'].count()
    n= df.loc[(df['play']=='No')]['play'].count()
    total= df['play'].count()

    e= -((p/total)*math.log2((p/total))-((n/total)*math.log2((n/total)))

    return e


E_play= entropy_target()
E_play
```

```
0.9402859586706311
```

```
def _count(feature,value):
    c= df.loc[(df[feature]==value)][feature].count()
    return c

def probability(feature,value):
    c= _count(feature,value)
    total= df[feature].count()
    prob= c/total
    return prob

def entropy_attribute(attribute,value):
    p= df.loc[(df[attribute]==value) & (df['play']=='Yes')]['play'].count()
    n= df.loc[(df[attribute]==value) & (df['play']=='No')]['play'].count()
    total= df.loc[df[attribute]==value][attribute].count()

    if p==0 or n==0:
        e=0
```

```
    else:
        e= -((p/total)*math.log2((p/total)))-((n/total)*math.log2((n/total)))


    return e

E_outlook= probability('outlook','Sunny')*entropy_attribute('outlook','Sunny') +
probability('outlook','Overcast')*entropy_attribute('outlook','Overcast') +
probability('outlook','Rainy')*entropy_attribute('outlook','Rainy')
E_temp= probability('temp','Hot')*entropy_attribute('temp','Hot') +
probability('temp','Mild')*entropy_attribute('temp','Mild') +
probability('temp','Cool')*entropy_attribute('temp','Cool')
E_humidity= probability('humidity','High')*entropy_attribute('humidity','High') +
probability('humidity','Normal')*entropy_attribute('humidity','Normal')
E_windy= probability('windy','True')*entropy_attribute('windy','True') +
probability('windy','False')*entropy_attribute('windy','False')

print(E_outlook,E_temp,E_humidity,E_windy)
```

```
def _count(feature,value):
    c= df.loc[(df[feature]==value)][feature].count()
    return c

def probability(feature,value):
    c= _count(feature,value)
    total= df[feature].count()
    prob= c/total
    return prob

def entropy_attribute(attribute,value):
    p= df.loc[(df[attribute]==value) & (df['play']=='Yes')]['play'].count()
    n= df.loc[(df[attribute]==value) & (df['play']=='No')]['play'].count()
    total= df.loc[df[attribute]==value][attribute].count()

    if p==0 or n==0:
        e=0
    else:
        e= -((p/total)*math.log2((p/total)))-((n/total)*math.log2((n/total)))


    return e

E_outlook= probability('outlook','Sunny')*entropy_attribute('outlook','Sunny') + probability('outlook','Overcast')*entropy_attrib
E_temp= probability('temp','Hot')*entropy_attribute('temp','Hot') + probability('temp','Mild')*entropy_attribute('temp','Mild') +
E_humidity= probability('humidity','High')*entropy_attribute('humidity','High') + probability('humidity','Normal')*entropy_attrib
E_windy= probability('windy','True')*entropy_attribute('windy','True') + probability('windy','False')*entropy_attribute('windy',

print(E_outlook,E_temp,E_humidity,E_windy)
```

```
0.6935361388961918 0.9110633930116763 0.7884504573082896 0.8921589282623617
```

```
gain_outlook= E_play - E_outlook
gain_temp= E_play - E_temp
gain_humidity= E_play - E_humidity
gain_windy= E_play - E_windy
```

gain=
{gain_outlook:'outlook',gain_temp:'temp',gain_humidity:'humidity',gain_windy:'wind
y'}
gain

```
gain_outlook= E_play - E_outlook
gain_temp= E_play - E_temp
gain_humidity= E_play - E_humidity
gain_windy= E_play - E_windy
gain= {gain_outlook:'outlook',gain_temp:'temp',gain_humidity:'humidity',gain_windy:'windy'}
gain
```

```
{0.24674981977443933: 'outlook',
 0.02922256565895487: 'temp',
 0.15183550136234159: 'humidity',
 0.04812703040826949: 'windy'}
```

print("Splliting attribute at first level is : ",gain[max(gain)])

The attribute with the largest information gain is used for the split.

```
print("Splliting attribute at first level is : ",gain[max(gain)])
```

```
Splliting attribute at first level is :  outlook
```

Practical-7

AIM: Generate and test decision tree for the dataset in exercise 5

Program:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from matplotlib import pyplot as plt


data=
{'outlook':['Sunny','Sunny','Overcast','Rainy','Rainy','Rainy','Overcast','Sunny','Sunny',
'Rainy','Sunny','Overcast','Overcast','Rainy'],
'temp':['Hot','Hot','Hot','Mild','Cool','Cool','Cool','Mild','Cool','Mild','Mild','Mild','Hot','
Mild'],
'humidity':['High','High','High','High','Normal','Normal','Normal','High','Normal','Nor
mal','Normal','High','Normal','High'],
'windy':['False','True','False','False','False','True','True','False','False','False','True','True
','False','True'],
'play':['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Yes','Yes','No']}

df= pd.DataFrame(data)
df
```

	outlook	temp	humidity	windy	play
0	Sunny	Hot	High	False	No
1	Sunny	Hot	High	True	No
2	Overcast	Hot	High	False	Yes
3	Rainy	Mild	High	False	Yes
4	Rainy	Cool	Normal	False	Yes
5	Rainy	Cool	Normal	True	No
6	Overcast	Cool	Normal	True	Yes
7	Sunny	Mild	High	False	No
8	Sunny	Cool	Normal	False	Yes
9	Rainy	Mild	Normal	False	Yes
10	Sunny	Mild	Normal	True	Yes
11	Overcast	Mild	High	True	Yes
12	Overcast	Hot	Normal	False	Yes
13	Rainy	Mild	High	True	No

converting categorical data into numerical data

one_hot_data = pd.get_dummies(df[['outlook', 'temp', 'humidity', 'windy']])
one_hot_data

```
# converting categorical data into numerical data
|
one_hot_data = pd.get_dummies(df[ ['outlook', 'temp', 'humidity', 'windy'] ])
one_hot_data
```

	outlook_Overcast	outlook_Rainy	outlook_Sunny	temp_Cool	temp_Hot	temp_Mild	humidity_High	humidity_Normal	windy_False	windy_True
0	0	0	1	0	1	0	1	0	1	0
1	0	0	1	0	1	0	1	0	0	1
2	1	0	0	0	1	0	1	0	1	0
3	0	1	0	0	0	1	1	0	1	0
4	0	1	0	1	0	0	0	1	1	0
5	0	1	0	1	0	0	0	1	0	1
6	1	0	0	1	0	0	0	1	0	1
7	0	0	1	0	0	1	1	0	1	0
8	0	0	1	1	0	0	0	1	1	0
9	0	1	0	0	0	1	0	1	1	0
10	0	0	1	0	0	1	0	1	0	1
11	1	0	0	0	0	1	1	0	0	1
12	1	0	0	0	1	0	0	1	1	0
13	0	1	0	0	0	1	1	0	0	1

target= pd.DataFrame(df['play'])

"splitter= best" means strategy used to choose the split at each node.
"criterion= entropy" means function to measure the quality of a split. “entropy” for the information gain.

clf = tree.DecisionTreeClassifier(splitter='best',criterion='entropy')
clf_train = clf.fit(one_hot_data, target)

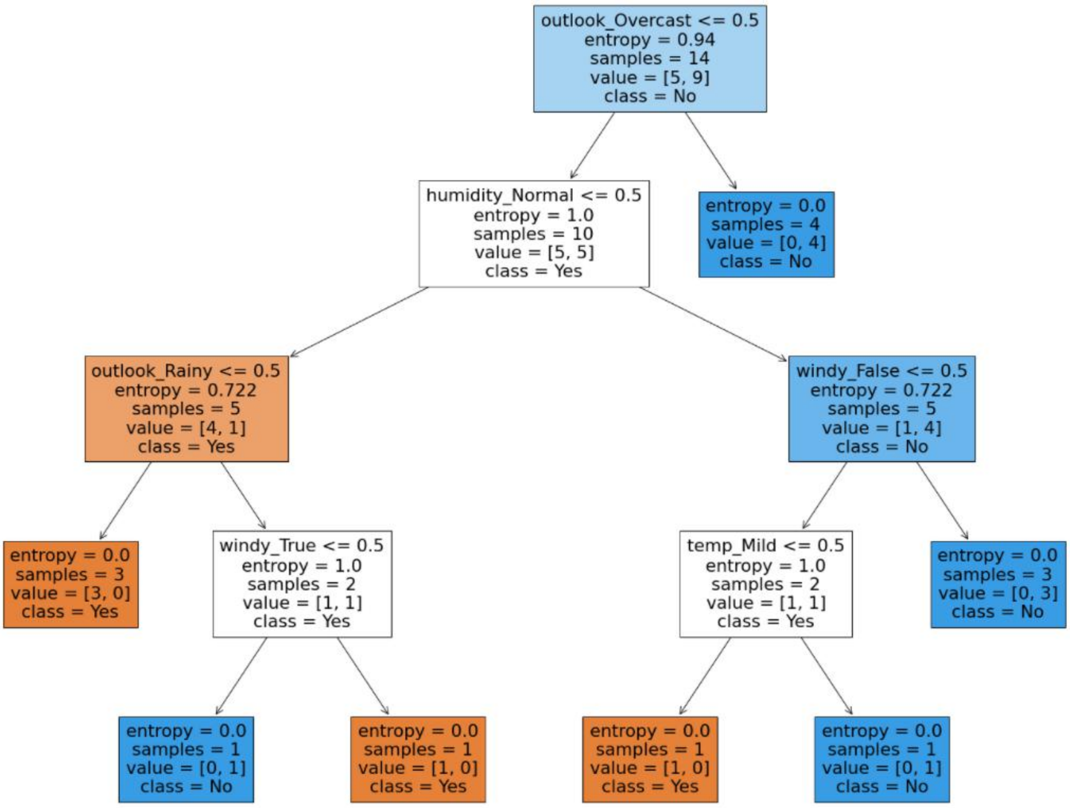
prediction = clf_train.predict([[0,0,1,1,0,0,1,0,0,1]])
print("Is there possiblity of playing game: ",prediction[0])

#Predicting the output from decision tree classifier

```
prediction = clf_train.predict([[0,0,1,1,0,0,1,0,0,1]])
print("Is there possiblity of playing game:  ",prediction[0])
```

```
Is there possiblity of playing game:   No
```

```
fn=list(one_hot_data.columns.values)
cn=['Yes', 'No']

fig = plt.figure(figsize=(25,20))
_ = tree.plot_tree(clf, feature_names=fn,class_names=cn,filled=True)
```



Practical-8

AIM: Find the clusters for following data with k = 2: Start with points 1 and 4 as two separate clusters.

Program:

```
# cordinates of points
data= [(1.0,1.0), (1.5,2.0), (3.0,4.0), (5.0,7.0), (3.5,5.0), (4.5,5.0), (3.5,4.5)]

# starting point for cluster 1 "point 1"
current_cluster1= [(data[0][0],data[0][1])]

# starting point for cluster 2 "point 4"
current_cluster2= [(data[3][0],data[3][1])]
print(current_cluster1,current_cluster2)
```

```
# cordinates of points
data= [(1.0,1.0), (1.5,2.0), (3.0,4.0), (5.0,7.0), (3.5,5.0), (4.5,5.0), (3.5,4.5)]

# starting point for cluster 1 "point 1"
current_cluster1= [(data[0][0],data[0][1])]

# starting point for cluster 2 "point 4"
current_cluster2= [(data[3][0],data[3][1])]
print(current_cluster1,current_cluster2)
```

```
[(1.0, 1.0)] [(5.0, 7.0)]
```

```
# function to calculate mean/centroid of cluster
def mean_cluster(c):
    l= len(c)
    sum_x=0
    sum_y=0

    for i in range(l):
        sum_x += c[i][0]
        sum_y += c[i][1]

    mean_x= sum_x/l
    mean_y= sum_y/l

   return (mean_x,mean_y)
```

```
import math
# function to calculate distance between two points
def distance(p1,p2):
  return math.sqrt((p1[0]-p2[0])**2 + (p1[1]-p2[1])**2)

# function to add data points to appropriate clusters
def add_points(data,centroid1,centroid2):
   cluster1=[]
   cluster2=[]
   for point in range(len(data)):
      if distance(data[point], centroid1) > distance(data[point], centroid2):
         cluster2.append(data[point])
      else:
         cluster1.append(data[point])

  return cluster1,cluster2




  i=1
  while True:
     # centroid of c1
     centroid1= mean_cluster(current_cluster1)

     # centroid of c2
     centroid2= mean_cluster(current_cluster2)

     new_cluster1,new_cluster2= add_points(data,centroid1,centroid2)
     print("Iteration number: ",i)
     print("Current cluster1 : ",current_cluster1)
     print("Current cluster2 : ",current_cluster2)
     print("New cluster1 : ",new_cluster1)
     print("New cluster2 : ",new_cluster2)
     print()

     if (current_cluster1 != new_cluster1) or (current_cluster2 != new_cluster2):
        current_cluster1= new_cluster1
        current_cluster2= new_cluster2
        i+=1
     else:
        Break



  print("Final clusters for given data: \n")
  print("Cluster 1: ",current_cluster1)
  print("Cluster 2: ",current_cluster2)
```

```
Iteration number:  1
Current cluster1 :  [(1.0, 1.0)]
Current cluster2 :  [(5.0, 7.0)]
New cluster1 :  [(1.0, 1.0), (1.5, 2.0), (3.0, 4.0)]
New cluster2 :  [(5.0, 7.0), (3.5, 5.0), (4.5, 5.0), (3.5, 4.5)]

Iteration number:  2
Current cluster1 :  [(1.0, 1.0), (1.5, 2.0), (3.0, 4.0)]
Current cluster2 :  [(5.0, 7.0), (3.5, 5.0), (4.5, 5.0), (3.5, 4.5)]
New cluster1 :  [(1.0, 1.0), (1.5, 2.0)]
New cluster2 :  [(3.0, 4.0), (5.0, 7.0), (3.5, 5.0), (4.5, 5.0), (3.5, 4.5)]

Iteration number:  3
Current cluster1 :  [(1.0, 1.0), (1.5, 2.0)]
Current cluster2 :  [(3.0, 4.0), (5.0, 7.0), (3.5, 5.0), (4.5, 5.0), (3.5, 4.5)]
New cluster1 :  [(1.0, 1.0), (1.5, 2.0)]
New cluster2 :  [(3.0, 4.0), (5.0, 7.0), (3.5, 5.0), (4.5, 5.0), (3.5, 4.5)]

Final clusters for given data:

Cluster 1:  [(1.0, 1.0), (1.5, 2.0)]
Cluster 2:  [(3.0, 4.0), (5.0, 7.0), (3.5, 5.0), (4.5, 5.0), (3.5, 4.5)]
```

Practical-9

AIM: Find following statistics for the data given in Exercise 1

$$Within\ Class\ Scatter: S_W = \sum_{i=1} \sum_{x \in w_i} (x - m_i)(x - m_i)^T$$

$$Between\ Class\ Scatter: S_B = \sum_{i=1}^{C} n_i (m_i - m)(m_i - m)^T$$

$$Total\ Scatter: S_T = \sum_{i=1}^{M} (x_i - m)(x_i - m)^T$$

Program:

```
A = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
B = [4, 8, 12, 16, 20, 24, 28, 32, 36, 40]
C = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

meanA= sum(A)/len(A)
meanB= sum(B)/len(B)
meanC= sum(C)/len(C)



# class specific covariance
def classSpecificCovariance(X,meanX):
   s=0
   for i in range(len(X)):
      temp= X[i]-meanX

      # transpose of 1X1 matrix is same as matrix
      temp= temp*temp

      s+= temp

   return s/len(X)

SA= classSpecificCovariance(A,meanA)
SB= classSpecificCovariance(B,meanB)
SC= classSpecificCovariance(C,meanC)
```

within class scatter Sw

Sw= SA+SB+SC
print("Within class scatter : ",Sw)

Within class scatter

```
# within class scatter Sw

Sw= SA+SB+SC
print("Within class scatter : ",Sw)
```

```
Within class scatter :  148.5
```

between class scatter matrix Sb

total_mean= (sum(A)+sum(B)+sum(C))/(len(A)+len(B)+len(C))

# N= 3   no. of classes
Sb= len(A)*((meanA-total_mean)*(meanA-total_mean))+len(B)*((meanB-total_mean)*(meanB-total_mean))+len(C)*((meanC-total_mean)*(meanC-total_mean))

print("Between class scatter : ",Sb)

Between class scatter

```
# between class scatter matrix Sb

total_mean= (sum(A)+sum(B)+sum(C))/(len(A)+len(B)+len(C))

# N= 3     no. of classes
Sb= len(A)*((meanA-total_mean)*(meanA-total_mean))+len(B)*((meanB-total_mean)*(meanB-total_mean))+len(C)*((meanC-total_mean)*(mea

print("Between class scatter : ",Sb)
```

```
Between class scatter :  1815.0
```

```
# total scatter

St=0
for i in range(10):
    St+= ((A[i]-total_mean)*(A[i]-total_mean)) + ((B[i]-total_mean)*(B[i]-total_mean))
+ ((C[i]-total_mean)*(C[i]-total_mean))

print("Total scatter : ",St)
```

Total class scatter

```
# total scatter

St=0
for i in range(10):
    St+= ((A[i]-total_mean)*(A[i]-total_mean)) + ((B[i]-total_mean)*(B[i]-total_mean)) + ((C[i]-total_mean)*(C[i]-total_mean))

print("Total scatter : ",St)
```

```
Total scatter :  3300.0
```

Practical-10

AIM: Given the following vectors:
X = [340, 230, 405, 325, 280, 195, 265, 300, 350, 310]; %sale
Y = [71, 65, 83, 74, 67, 56, 57, 78, 84, 65];
Ex. 1: Find the Linear Regression model for independent variable X and dependent variable Y.
Ex. 2: Predict the value of y for x = 250. Also find the residual for y4.

Program:

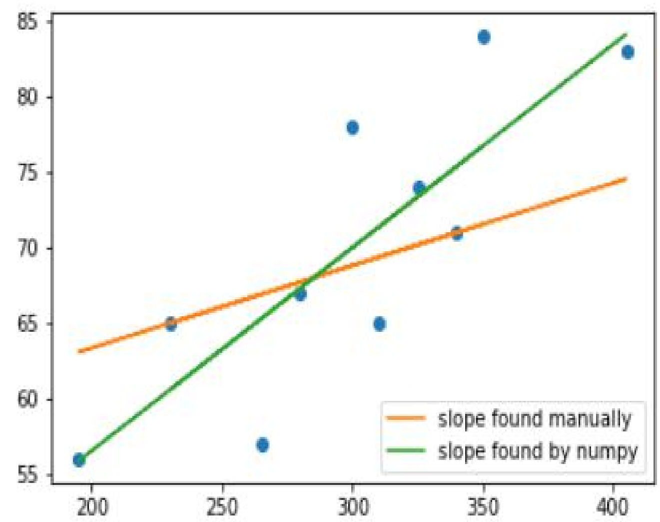
```
import numpy as np

X = np.array([340, 230, 405, 325, 280, 195, 265, 300, 350, 310])
Y = np.array([71, 65, 83, 74, 67, 56, 57, 78, 84, 65])


m= (Y[1]-Y[0])/(X[1]-X[0])
c= -m*X[0]+Y[0]

x= int(input("Enter % of sale : "))

# equation of line y=mx+c
y= m*x + c

print("Profit as per sale : ",y)
```

slope m= (y2-y1)/(x2-x1)

```
m= (Y[1]-Y[0])/(X[1]-X[0])
c= -m*X[0]+Y[0]

x= int(input("Enter % of sale : "))

# equation of line y=mx+c
y= m*x + c

print("Profit as per sale : ",y)
```

```
Enter % of sale : 300
Profit as per sale :  68.81818181818181
```

```
import matplotlib.pyplot as plt

# basic scatter plot
plt.plot(X,Y,'o')

# plotting linear regression in scatter plot
line1,=plt.plot(X,(m*X)+c, label="slope found manually")

# best fitting line
m1,c1= np.polyfit(X,Y,1)
line2,= plt.plot(X,m1*X+c1, label="slope found by numpy")
leg = plt.legend(loc='lower right')
plt.show()

print("Slope and intercept found manually: ",m,c)
print("Slope and intercept found by numpy: ",m1,c1)
```




```
Slope and intercept found manually:  0.05454545454545454 52.45454545454545
Slope and intercept found by numpy:  0.13443113772455093 29.670658682634706
```

```
print("Residual for ",Y[3],"= ",Y[3]-(m*X[3]+c))
```

residual= actual value - predicted value

residual for y4= Y[3] - predicted value using X[3]

```
print("Residual for ",Y[3],"= ",Y[3]-(m*X[3]+c))
```

```
Residual for  74 =  3.8181818181818181813
```