ITA6016 MACHINE LEARNING LAB EXERCISE 1

Name: SIVAA V

Reg: 22MCA1127

1. Create two random numpy array of integers X and Y; check whether they are equal or not?

   Code:
   import numpy as np
   X=np.array(["22MCA1127","SIVAA V","MCA"])
   Y=np.array(["22MCA1127","SIVAA V","MCA"])
   compare=X==Y
   print(compare.all())
   Screenshot:

```
In [1]: import numpy as np
        X=np.array(["22MCA1127","SIVAA V","MCA"])
        Y=np.array(["22MCA1127","SIVAA V","MCA"])
        compare=X==Y
        print(compare.all())

        True
```

2. Create a numpy array with zeros and make it immutable and check whether able to change the values or not.

   Code:
   import numpy as np
   arr=np.zeros(3,dtype=int)
   arr.flags.writeable=False
   print("Trying to change first value of array")
   arr[0]=1

   Screenshot:

```
In [10]: import numpy as np
         arr=np.zeros(3,dtype=int)
         arr.flags.writeable=False
         print("Trying to change first value of array")
         arr[0]=1

Trying to change first value of array

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_2656/1403889979.py in <module>
      3 arr.flags.writeable=False
      4 print("Trying to change first value of array")
----> 5 arr[0]=1

ValueError: assignment destination is read-only
```

3. Create a random numpy array Y of integers with size `10` and replace the maximum value by 100 and minimum value by 0.

   Code:

   import numpy as np

   a=np.array([1,2,3])

   print(a)

   a[a.argmax()]=100

   a[a.argmin()]=0

   print("After replacing values")

   print(a)

   Screenshot:

```
In [3]: import numpy as np
        a=np.array([1,2,3])
        print(a)
        a[a.argmax()]=100
        a[a.argmin()]=0
        print("After replacing values")
        print(a)

[1 2 3]
After replacing values
[  0   2 100]
```

4. Write a program to find the closest value to a given value from a numpy array.

Code:

```
def find_closest(arr, val):

    idx = np.abs(arr - val).argmin()

    return arr[idx]

arr = np.array([1,7,10,12])

n=int(input("Enter a number "))

find_closest(arr, n)
```

```
In [7]: def find_closest(arr, val):
               idx = np.abs(arr - val).argmin()
               return arr[idx]

        arr = np.array([1,7,10,12])
        n=int(input("Enter a number "))
        find_closest(arr, n)

        Enter a number 11

Out[7]: 10
```

5. Write a program to subtract the mean of each row of a matrix

Code:

```
import numpy as np

Y = np.array([1,2,3])
```

Y_avg = Y.mean(axis=0)

print(Y - Y_avg)

```
In [17]: import numpy as np

         Y = np.array([1,2,3])
         Y_avg = Y.mean(axis=0)
         print(Y - Y_avg)

         [-1.  0.  1.]
```

6. Write a program to convert the index of a series into a column of a dataframe.

Code:

import pandas as pd

d = pd.Series([10, 20, 30, 40, 50], index=['a', 'b', 'c', 'd', 'e'])

df = pd.DataFrame({'index_col': d.index, 'values': d.values})

print(df)

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

        d = pd.Series([10, 20, 30, 40, 50], index=['a', 'b', 'c', 'd', 'e'])

        df = pd.DataFrame({'index_col': d.index, 'values': d.values})

        print(df)

          index_col  values
        0         a      10
        1         b      20
        2         c      30
        3         d      40
        4         e      50
```

Screenshot:

7. Write a program to find out the items of series X not present in series Y?

Code:

import pandas as pd

X = pd.Series([1, 2, 3, 4, 5, 8])

Y = pd.Series([3, 4, 5, 6, 7, 9])

res = X[~X.isin(Y)]

print(res)

Screenshot:

```
In [4]: import pandas as pd

        X = pd.Series([1, 2, 3, 4, 5, 8])
        Y = pd.Series([3, 4, 5, 6, 7, 9])

        result = X[~X.isin(Y)]

        print(result)

        0    1
        1    2
        5    8
        dtype: int64
```

8. Write a program to find out the minimum, 25th percentile, median, 75th, and max of a numeric series?

Code:

import pandas as pd

import numpy as np

data = pd.Series(np.array([1,5,7,9,11,13,15]))

minimum = data.min()

q_25 = data.quantile(0.25)

median = data.median()

q_75 = data.quantile(0.75)

maximum = data.max()

print("Minimum:", minimum)

print("25th Percentile:", q_25)

print("Median:", median)

print("75th Percentile:", q_75)

print("Maximum:", maximum)

Screenshot:

```
In [9]: import pandas as pd
        import numpy as np

        data = pd.Series(np.array([1,5,7,9,11,13,15]))

        minimum = data.min()
        q_25 = data.quantile(0.25)
        median = data.median()
        q_75 = data.quantile(0.75)
        maximum = data.max()

        print("Minimum:", minimum)
        print("25th Percentile:", q_25)
        print("Median:", median)
        print("75th Percentile:", q_75)
        print("Maximum:", maximum)

        Minimum: 1
        25th Percentile: 6.0
        Median: 9.0
        75th Percentile: 12.0
        Maximum: 15
```

9. Write a program to keep only the top 2 most frequent values as it is and replace everything else as ‘Other’?

Code:

import pandas as pd

data = pd.Series(['SIVAA V', '22MCA1127', 'SIVAA V', 'Arrange', 'New', 'Python'])

value_counts = data.value_counts()

top_values = value_counts.index[:2]

data[~data.isin(top_values)] = 'Other'

print(data)

Screenshot:

```
In [3]: import pandas as pd

        data = pd.Series(['SIVAA V', '22MCA1127', 'SIVAA V', 'Arrange', 'New', 'Python'])
        value_counts = data.value_counts()

        top_values = value_counts.index[:2]
        data[~data.isin(top_values)] = 'Other'

        print(data)

        0      SIVAA V
        1    22MCA1127
        2      SIVAA V
        3        Other
        4        Other
        5        Other
        dtype: object
```

10. Write a program to convert the first character of each element in a series to uppercase

Code:

import pandas as pd

data = pd.Series([sivaa v, 'mca', 'vit'])

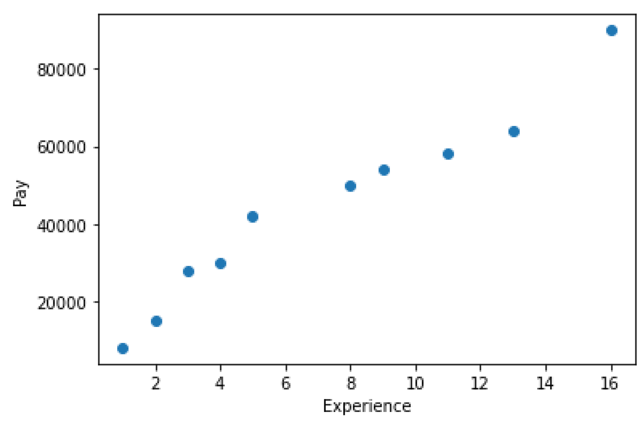
data = data.str.capitalize()

print(data)

Screenshot:

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

        data = pd.Series(['sivaa v', 'mca', 'vit'])
        data = data.str.capitalize()
        print(data)

        0    Sivaa v
        1        Mca
        2        Vit
        dtype: object
```

In [3]:
```
#1
import matplotlib.pyplot as plt

experience=[2,3,5,13,8,16,1,9,4]
pay=[15000,28000,42000,64000,50000,90000,58000,8000,54000,30000]
plt.scatter(experience,pay)
plt.xlabel("Experience")
plt.ylabel("Pay")
plt.show()
```



In [36]:
```
#2
import numpy as np
import pandas as pd
x=[2,3,5,13,8,16,11,1,9,4]
y=[15,28,42,64,50,90,58,8,54,30]
xmean=np.mean(x)
ymean=np.mean(y)
x_xmean=sum(x-xmean)
y_ymean=sum(y-ymean)
prod_x_y=x_xmean*y_ymean
data={
    "x":x,
    "y":y,
    "x-xbar":x-xmean,
    "y-ybar":y-ymean,
    "(x-xbar)*(y-ybar)":prod_x_y,
    "(x-xbar)^2":(x-xmean)**2
}
df=pd.DataFrame(data)
print(df)
print("Mean of x: ",xmean)
print("Mean of y: ",ymean)
print("Sumation of x-xbar: ",x_xmean)
print("Sumation of y-ybar: ",y_ymean)
print("Summation of (x-xbar)*(y-ybar): ", prod_x_y)
print("Sumation of x-xbar^2: ",sum(x-xmean)**2)
```

```
    x   y  x-xbar  y-ybar  (x-xbar)*(y-ybar)  (x-xbar)^2
0   2  15    -5.2   -28.9      -2.524355e-29       27.04
1   3  28    -4.2   -15.9      -2.524355e-29       17.64
2   5  42    -2.2    -1.9      -2.524355e-29        4.84
3  13  64     5.8    20.1      -2.524355e-29       33.64
4   8  50     0.8     6.1      -2.524355e-29        0.64
5  16  90     8.8    46.1      -2.524355e-29       77.44
6  11  58     3.8    14.1      -2.524355e-29       14.44
7   1   8    -6.2   -35.9      -2.524355e-29       38.44
8   9  54     1.8    10.1      -2.524355e-29        3.24
9   4  30    -3.2   -13.9      -2.524355e-29       10.24
Mean of x:  7.2
Mean of y:  43.9
Sumation of x-xbar:  -1.7763568394002505e-15
Sumation of y-ybar:  1.4210854715202004e-14
Summation of (x-xbar)*(y-ybar):  -2.524354896707238e-29
Sumation of x-xbar^2:  3.1554436208840472e-30
```

In [17]:
```
#3
import numpy as np
import pandas as pd
x=[2,3,5,13,8,16,11,1,9,4]
y=[15,28,42,64,50,90,58,8,54,30]
xmean=np.mean(x)
ymean=np.mean(y)
prod=(x-xmean)*(y-ymean)
sum_prod=sum(prod)
sq_x=(x-xmean)**2
sq1=sum(sq_x)
w1=sum_prod/sq1
print("w1: ",w1)
w0=ymean-w1*xmean
print("w0: ",w0)
```

```
w1:  4.776801405975396
w0:  9.507029876977143
```

In [19]:
```
#4
import numpy as np
import pandas as pd
x=[]
y=[]
n=int(input("Enter number of elements for x: "))
for i in range(0,n):
    ele=int(input())
    x.append(ele)
n2=int(input("Enter number of elements for y: "))
for i in range(0,n2):
    ele1=int(input())
    y.append(ele1)

def calc(x,y):
    xmean=np.mean(x)
    ymean=np.mean(y)
    prod=(x-xmean)*(y-ymean)
    sum_prod=sum(prod)
    sq_x=(x-xmean)**2
    sq1=sum(sq_x)
    w1=sum_prod/sq1
    print("w1: ",w1)
    w0=ymean-w1*xmean
    print("w0: ",w0)

calc(x,y)
```

```
Enter number of elements for x: 3
1
2
3
Enter number of elements for y: 3
1
2
3
w1:  1.0
w0:  0.0
```

In [31]:
```
#5
import numpy as np
import pandas as pd
from sympy import symbols, Eq, solve

x=[]
y=[]
n=int(input("Enter number of elements for x: "))
for i in range(0,n):
    ele=int(input())
    x.append(ele)
n2=int(input("Enter number of elements for y: "))
for i in range(0,n2):
    ele1=int(input())
    y.append(ele1)

def calc(x,y):
    b0, b1 = symbols('w0,w1')
    sum_x=sum(x)
    sum_y=sum(y)
    c=[]
    x_2=[]
    for i in range(0,n):
        c.append(x[i]*y[i])
        x_2.append(x[i]**2)
    calc1=sum(c)
    eq1=Eq(n*b0+b1*sum_x,sum_y)
    eq2=Eq(b0*sum(x)+b1*sum(x_2),calc1)
    print(solve((eq1,eq2),(b0,b1)))


calc(x,y)
```

```
Enter number of elements for x: 3
1
2
3
Enter number of elements for y: 3
1
2
3
{w0: 0, w1: 1}
```

In [30]:
```
11/6+1/12
```

Out[30]:
```
1.9166666666666665
```

In [49]:
```
#6
import numpy as num
import random as rand

def gradientDescent(X, Y, theta, alpha, a, numIterations):
    Xtrans = X.transpose()
    for i in range(0, numIterations):
        hypothesis = num.dot(X, theta)
        loss = hypothesis - Y

        cst = num.sum(loss ** 2) / (2 * a)
        gradient = num.dot(Xtrans, loss) / a
        theta = theta - alpha * gradient
    return theta


def genData(numPoints, bias, variance):
    X = num.zeros(shape=(numPoints, 3))
    Y = num.zeros(shape=numPoints)
    for i in range(0, numPoints):
        X[i][0] = 2
        X[i][1] = i
        Y[i] = (i + bias) + rand.uniform(0, 2) * variance
    return X, Y
X,Y = genData(90, 20, 9)
a, b = num.shape(X)
numIterations= 1
alpha = 0.0004
theta = num.ones(b)
theta = gradientDescent(X,Y, theta, alpha,a, numIterations)
print("w0: ",theta[0])
print("w1: ",theta[1])
```

```
w0:  1.0215475744298648
w1:  1.475157085335912
```

In [54]:
```
#7
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
import math

data = {'X': [2,3,5,13,8,16,11,1,9,4],
        'Y': [15,28,42,64,50,90,58,8,54,30]}
df = pd.DataFrame(data)

X_train, X_test, y_train, y_test = train_test_split(df['X'], df['Y'], test_size=0.2, random_state=0)

X_train = np.array(X_train).reshape((-1, 1))
X_test = np.array(X_test).reshape((-1, 1))

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print('Coefficients:', model.coef_)
print('Intercept:', model.intercept_)

df_result = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print(df_result)

print("MSE: ",mean_squared_error(y_test, y_pred))
print("RMSE: ",math.sqrt(mean_squared_error(y_test, y_pred)))
print("MAE: ",mean_absolute_error(y_test, y_pred))
```

```
Coefficients: [4.85307517]
Intercept: 7.69020501138953
   Actual  Predicted
2      42  31.955581
8      54  51.367882
MSE:  53.909201643827025
RMSE:  7.3422885835294585
6.3382687927107035
```

In [ ]:

Exercise_3_DecisionTree-Qn

In [ ]:

```
#Name: SIVAA V
#Reg: 22MCA1127
```

In [1]:

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

In [2]:

```
data = pd.read_csv("/home/sivaa/Machine Learning/dataset.csv")

print(data)
```

```
  Age Group Certified  Skill Type    Status
0       Old       Yes  Soft Skill  Rejected
1    Middle        No  Hard Skill  Selected
2    Middle       Yes  Soft Skill  Rejected
3     Young        No  Hard Skill  Selected
4    Middle       Yes  Hard Skill  Rejected
5     Young        No  Soft Skill  Selected
6     Young       Yes  Soft Skill  Selected
7       Old        No  Soft Skill  Rejected
8       Old        No  Hard Skill  Rejected
9    Middle        No  Soft Skill  Selected
```

1. Consider the following dataset and calculate the entropy and information gain w.r.t the target attribute named “Status”.

In [3]:

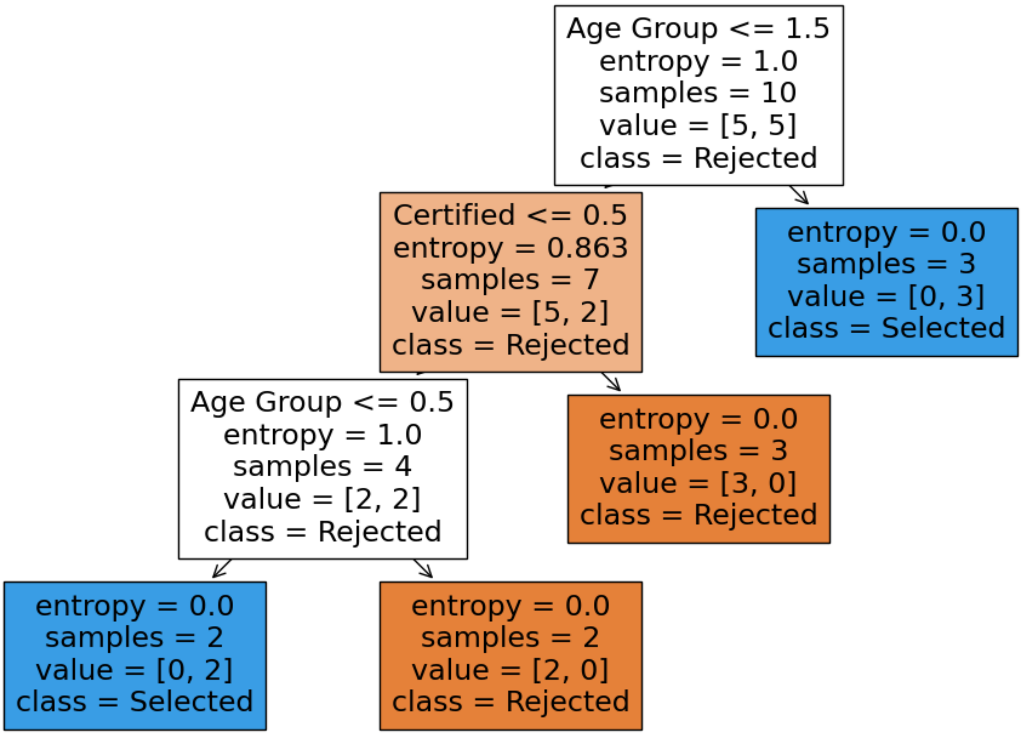
```
# Calculate the entropy of the target attribute "Status"
status_counts = data['Status'].value_counts()
num_instances = len(data)
entropy_status = 0
for count in status_counts:
    probability = count / num_instances
    entropy_status += -probability * math.log2(probability)
print(f"Entropy of Status: {entropy_status:.3f}")

# Calculate the information gain of each attribute w.r.t. the "Status" attribute
for attribute in ['Age Group', 'Certified', 'Skill Type']:
    entropy_attribute = 0
    attribute_value_counts = data[attribute].value_counts()
    for value, count in attribute_value_counts.items():
        value_subset = data[data[attribute] == value]
        value_subset_size = len(value_subset)
        value_status_counts = value_subset['Status'].value_counts()
        value_entropy_status = 0
        for value_count in value_status_counts:
            value_probability = value_count / value_subset_size
            value_entropy_status += -value_probability * math.log2(value_probability)
        entropy_attribute += count / num_instances * value_entropy_status
    information_gain = entropy_status - entropy_attribute
    print(f"Information gain of {attribute}: {information_gain:.3f}")
```

```
Entropy of Status: 1.000
Information gain of Age Group: 0.600
Information gain of Certified: 0.125
Information gain of Skill Type: 0.000
```

2. From the above calculated values of gain, design a decision tree for the above given data set.

In [4]:

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

data = pd.read_csv("/home/sivaa/Machine Learning/dataset.csv")


le = LabelEncoder()
data['Age Group'] = le.fit_transform(data['Age Group'])
data['Certified'] = le.fit_transform(data['Certified'])
data['Skill Type'] = le.fit_transform(data['Skill Type'])
data['Status'] = le.fit_transform(data['Status'])

dt = DecisionTreeClassifier(criterion='entropy')
X = data.drop('Status', axis=1)
y = data['Status']
dt.fit(X, y)

plt.figure(figsize=(12, 8))
plot_tree(dt, feature_names=X.columns, class_names=['Rejected', 'Selected'], filled=True)
plt.show()
```



3. Transform the designed decision tree into decision rules.

In [5]:

```
# Transform the decision tree into decision rules
def tree_to_rules(tree, feature_names):
    rules = []
    for feature in feature_names:
        for value in np.unique(data[feature]):
            threshold = np.mean(data[data[feature] == value]['Status'])
            rule = f"{feature} == {value} => Status = {'Selected' if threshold > 0.5 else
            rules.append(rule)
    return rules

# Print the decision rules
rules = tree_to_rules(dt, data.columns[:-1])
for rule in rules:
    print(rule)
```

```
Age Group == 0 => Status = Rejected
Age Group == 1 => Status = Rejected
Age Group == 2 => Status = Selected
Certified == 0 => Status = Selected
Certified == 1 => Status = Rejected
Skill Type == 0 => Status = Rejected
Skill Type == 1 => Status = Rejected
```

4. Use the designed decision tree or rules to predict the ‘Status’ of the given employee. ▪ Young No Hard Skill ▪ Old Yes Soft Skill ▪ Middle Yes Hard Skill

In [6]:

```
from sklearn.tree import DecisionTreeClassifier, export_text
from sklearn.preprocessing import LabelEncoder
import pandas as pd

data = pd.read_csv("/home/sivaa/Machine Learning/dataset.csv")


le = LabelEncoder()
data['Age Group'] = le.fit_transform(data['Age Group'])
data['Certified'] = le.fit_transform(data['Certified'])
data['Skill Type'] = le.fit_transform(data['Skill Type'])
data['Status'] = le.fit_transform(data['Status'])

dt = DecisionTreeClassifier(criterion='entropy')
X = data.drop('Status', axis=1)
y = data['Status']
dt.fit(X, y)

tree_rules = export_text(dt, feature_names=X.columns.tolist())

print(tree_rules)
```

```
|--- Age Group <= 1.50
|   |--- Age Group <= 0.50
|   |   |--- Certified <= 0.50
|   |   |   |--- class: 1
|   |   |--- Certified >  0.50
|   |   |   |--- class: 0
|   |--- Age Group >  0.50
|   |   |--- class: 0
|--- Age Group >  1.50
|   |--- class: 1
```

5. Design a function named find_entropy in python for finding the entropy of the attributes given in the above dataset.

In [7]:

```
import pandas as pd
import math

data = pd.read_csv("/home/sivaa/Machine Learning/dataset.csv")


def find_entropy(df, attribute):
    value_counts = df[attribute].value_counts()

    total_instances = len(df)
    entropy = 0

    for value_count in value_counts:
        probability = value_count / total_instances
        entropy -= probability * math.log2(probability)

    return entropy

entropy = find_entropy(data, 'Age Group')
print(f"Entropy of 'Age Group': {entropy}")
```

```
Entropy of 'Age Group': 1.5709505944546684
```

6. Design a function named find_gain in python for finding the information gain of the attributes given in the above dataset w.r.t to the 'Status' attribute

In [8]:

```
import math

def find_entropy(df):
    entropy = 0
    num_records = len(df)
    classes = df['Status'].unique()
    for c in classes:
        num_c = len(df[df['Status']==c])
        p = num_c/num_records
        entropy -= p * math.log2(p)
    return entropy

def find_gain(df, attribute):
    total_entropy = find_entropy(df)
    values = df[attribute].unique()
    entropy = 0
    for v in values:
        num_v = len(df[df[attribute]==v])
        df_v = df[df[attribute]==v]
        entropy += (num_v/len(df))*find_entropy(df_v)
    gain = total_entropy - entropy
    return gain
find_entropy(data)
find_gain(data, 'Status')
```

Out[8]:

```
1.0
```

7. Load the above dataset as data frame in python

In [9]:

```
import pandas as pd

data = pd.DataFrame({
    'Age Group': ['Old', 'Middle', 'Middle', 'Young', 'Middle', 'Young', 'Young', 'Old', 
    'Certified': ['Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'No', 'No'],
    'Skill Type': ['Soft Skill', 'Hard Skill', 'Soft Skill', 'Hard Skill', 'Hard Skill', 
    'Status': ['Rejected', 'Selected', 'Rejected', 'Selected', 'Rejected', 'Selected', 'S
})
```

8. Design and visualize the decision tree using scikit learn package for the given dataset.

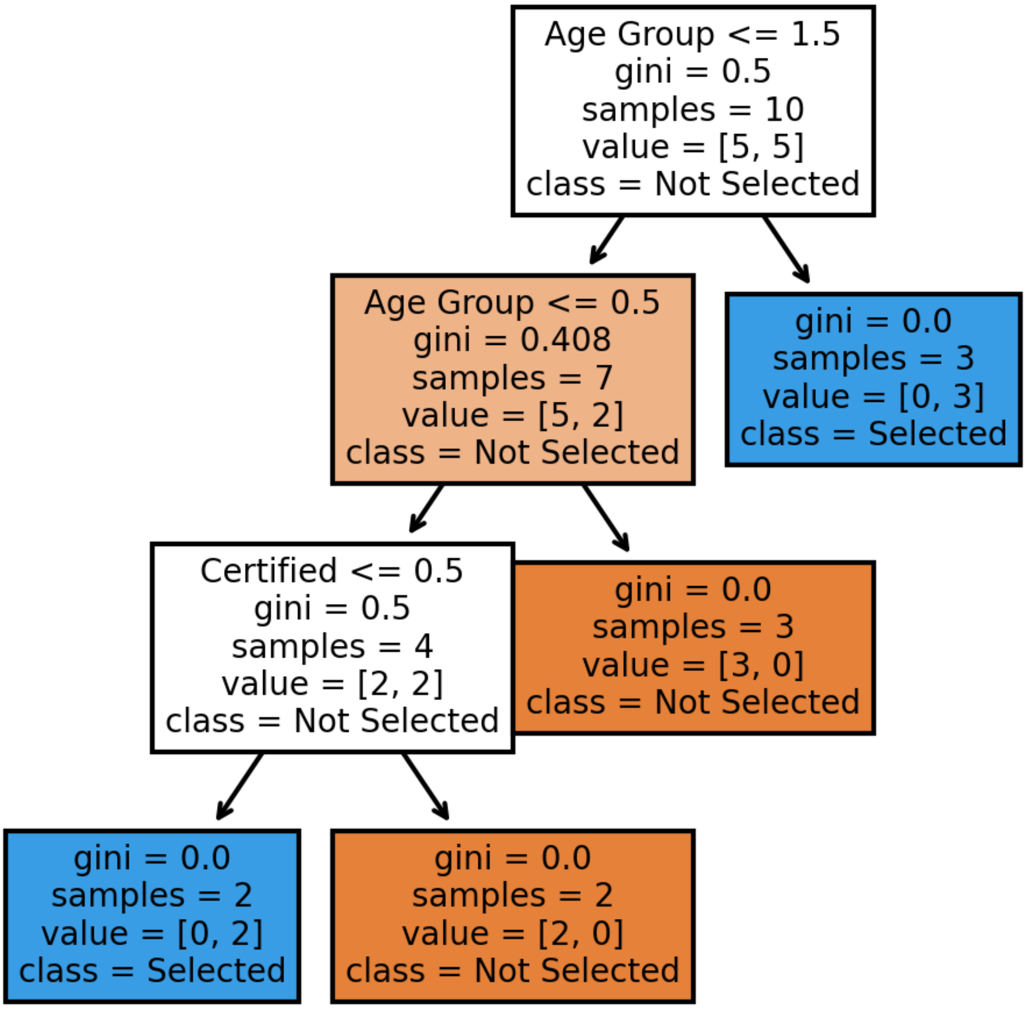
In [10]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
import matplotlib.pyplot as plt

model = DecisionTreeClassifier()

model.fit(X, y)

fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(4,4), dpi=300)
tree.plot_tree(model, feature_names=X.columns, class_names=['Not Selected', 'Selected'],
```

Out[10]:

```
[Text(0.6666666666666666, 0.875, 'Age Group <= 1.5\ngini = 0.5\nsamples =
10\nvalue = [5, 5]\nclass = Not Selected'),
 Text(0.5, 0.625, 'Age Group <= 0.5\ngini = 0.408\nsamples = 7\nvalue =
[5, 2]\nclass = Not Selected'),
 Text(0.3333333333333333, 0.375, 'Certified <= 0.5\ngini = 0.5\nsamples =
4\nvalue = [2, 2]\nclass = Not Selected'),
 Text(0.16666666666666666, 0.125, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]
\nclass = Selected'),
 Text(0.5, 0.125, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]\nclass = Not Se
lected'),
 Text(0.6666666666666666, 0.375, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]
\nclass = Not Selected'),
 Text(0.8333333333333334, 0.625, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]
\nclass = Selected')]
```

taset from



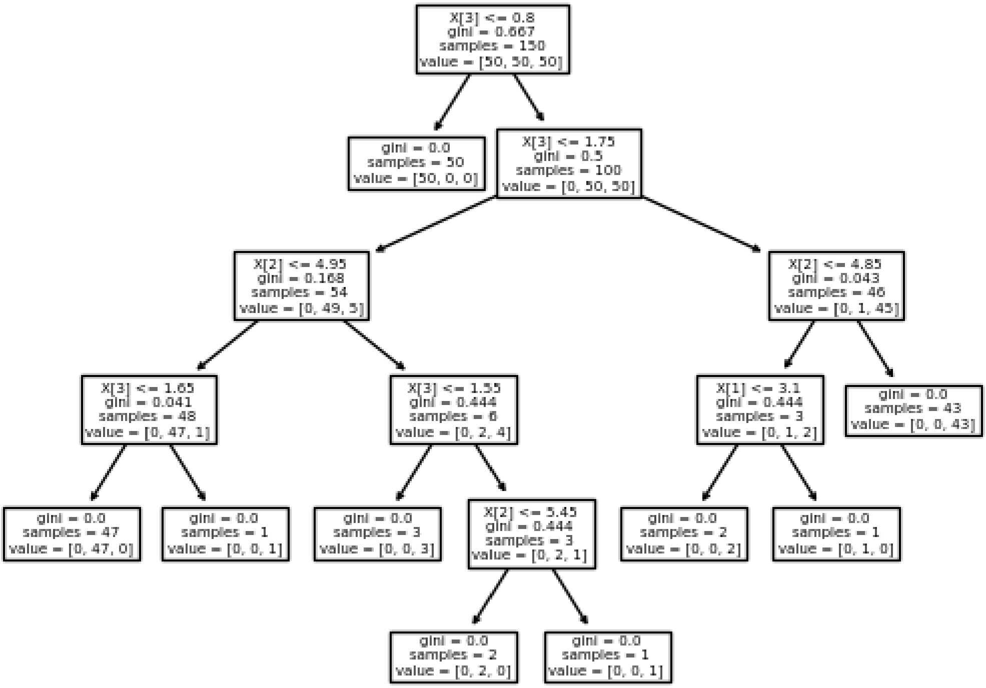
Age Group <= 1.5
gini = 0.5
samples = 10
value = [5, 5]
class = Not Selected
Age Group <= 0.5
gini = 0.408
samples = 7
value = [5, 2]
class = Not Selected
gini = 0.0
samples = 3
value = [0, 3]
class = Selected
Certified <= 0.5
gini = 0.5
samples = 4
value = [2, 2]
class = Not Selected
gini = 0.0
samples = 3
value = [3, 0]
class = Not Selected
gini = 0.0
samples = 2
value = [0, 2]
class = Selected
gini = 0.0
samples = 2
value = [2, 0]
class = Not Selected

In [11]:

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree

iris = load_iris()

iris_df = pd.DataFrame(iris.data, columns=iris.feature_names)

iris_df['target'] = iris.target

X = iris_df.iloc[:, :-1]
y = iris_df.iloc[:, -1]

tree = DecisionTreeClassifier()

tree.fit(X, y)

plot_tree(tree)
```

Out[11]:

```
[Text(0.5, 0.9166666666666666, 'X[3] <= 0.8\ngini = 0.667\nsamples = 150\n
value = [50, 50, 50]'),
 Text(0.4230769230769231, 0.75, 'gini = 0.0\nsamples = 50\nvalue = [50, 0,
0]'),
 Text(0.5769230769230769, 0.75, 'X[3] <= 1.75\ngini = 0.5\nsamples = 100\n
value = [0, 50, 50]'),
 Text(0.3076923076923077, 0.5833333333333334, 'X[2] <= 4.95\ngini = 0.168
\nsamples = 54\nvalue = [0, 49, 5]'),
 Text(0.15384615384615385, 0.4166666666666667, 'X[3] <= 1.65\ngini = 0.041
\nsamples = 48\nvalue = [0, 47, 1]'),
 Text(0.07692307692307693, 0.25, 'gini = 0.0\nsamples = 47\nvalue = [0, 4
7, 0]'),
 Text(0.23076923076923078, 0.25, 'gini = 0.0\nsamples = 1\nvalue = [0, 0,
1]'),
 Text(0.46153846153846156, 0.4166666666666667, 'X[3] <= 1.55\ngini = 0.444
\nsamples = 6\nvalue = [0, 2, 4]'),
 Text(0.38461538461538464, 0.25, 'gini = 0.0\nsamples = 3\nvalue = [0, 0,
3]'),
 Text(0.5384615384615384, 0.25, 'X[2] <= 5.45\ngini = 0.444\nsamples = 3\n
value = [0, 2, 1]'),
 Text(0.46153846153846156, 0.08333333333333333, 'gini = 0.0\nsamples = 2\n
value = [0, 2, 0]'),
 Text(0.6153846153846154, 0.08333333333333333, 'gini = 0.0\nsamples = 1\nv
alue = [0, 0, 1]'),
 Text(0.8461538461538461, 0.5833333333333334, 'X[2] <= 4.85\ngini = 0.043
\nsamples = 46\nvalue = [0, 1, 45]'),
 Text(0.7692307692307693, 0.4166666666666667, 'X[1] <= 3.1\ngini = 0.444\n
samples = 3\nvalue = [0, 1, 2]'),
 Text(0.6923076923076923, 0.25, 'gini = 0.0\nsamples = 2\nvalue = [0, 0,
2]'),
 Text(0.8461538461538461, 0.25, 'gini = 0.0\nsamples = 1\nvalue = [0, 1,
0]'),
 Text(0.9230769230769231, 0.4166666666666667, 'gini = 0.0\nsamples = 43\nv
alue = [0, 0, 43]')]
```



10.Evaluate the designed model on the iris dataset itself with various metrics.

In [12]:

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.model_selection import train_test_split

df = pd.read_csv('iris.csv')

X = df.drop(['Species'], axis=1)
y = df['Species']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

tree = DecisionTreeClassifier(random_state=42)
tree.fit(X_train, y_train)

y_pred = tree.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1 Score: 1.0
```

In [ ]:

```
#Name: SIVAA V
#Reg: 22MCA1127
#Exercise: 4
```

In [4]:

```
#1. Load the Titanic dataset from Kaggle to working environment.

import pandas as pd
train_data = pd.read_csv("/home/sivaa/Machine Learning/train.csv")
```

In [5]:

```
#2. Perform exploratory analysis on the loaded dataset and draw your inferences.
train_data.head()
```

Out[5]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500

In [6]:

```
#2. Perform exploratory analysis on the loaded dataset and draw your inferences.

train_data.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          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         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
```

In [7]:

```
#2. Perform exploratory analysis on the loaded dataset and draw your inferences.

train_data.describe()
```

Out[7]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

In [8]:

```
#3. From the above analysis if any attributes are not relevant in accessing the survival
#of the passenger then drop those columns.

train_data = train_data.drop(['PassengerId','Name','Ticket','Cabin'],axis=1)
train_data.head()
```

Out[8]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	male	22.0	1	0	7.2500	S
1	1	1	female	38.0	1	0	71.2833	C
2	1	3	female	26.0	0	0	7.9250	S
3	1	1	female	35.0	1	0	53.1000	S
4	0	3	male	35.0	0	0	8.0500	S

In [10]:

```
#4.Check for the missing values in the modified dataset and fill the missing
# values with appropriate methods.

train_data.isnull().sum()
median = train_data['Age'].median()
train_data['Age'].fillna(median,inplace=True)
mode_embarked = train_data['Embarked'].mode()[0]
train_data['Embarked'].fillna(mode_embarked, inplace=True)
train_data.isnull().sum()
```

Out[10]:

```
Survived    0
Pclass      0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
Embarked    0
dtype: int64
```

In [24]:

```
#5. Split the modified dataset into 80-20 ratio for training and testing.

from sklearn.model_selection import train_test_split

x = train_data.drop('Survived', axis=1)
y = train_data['Survived']
features = ["Pclass", "Sex", "SibSp", "Parch"]
x = pd.get_dummies(train_data[features])

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

```
     Pclass  SibSp  Parch  Sex_female  Sex_male
0         3      1      0           0         1
1         1      1      0           1         0
2         3      0      0           1         0
3         1      1      0           1         0
4         3      0      0           0         1
..      ...    ...    ...         ...       ...
886       2      0      0           0         1
887       1      0      0           1         0
888       3      1      2           1         0
889       1      0      0           0         1
890       3      0      0           0         1

[891 rows x 5 columns]
```

In [25]:

```
#6. Apply Logistic Regression and design a model on the training data.

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score

model = LogisticRegression()

model.fit(x_train, y_train)
```

```
Accuracy of the logistic regression model: 79.33%
```

In [26]:

```
#7 Fit the created model on the test data.

y_pred = model.predict(x_test)

accuracy = accuracy_score(y_test, y_pred)

print("Accuracy of the logistic regression model: {:.2f}%".format(accuracy * 100))
```

```
Accuracy of the logistic regression model: 79.33%
```

In [ ]:

```

```

In [ ]:

```
#Name: SIVAA V
#Reg: 22MCA1127
#Lab Exercise: 5
```

In [5]:

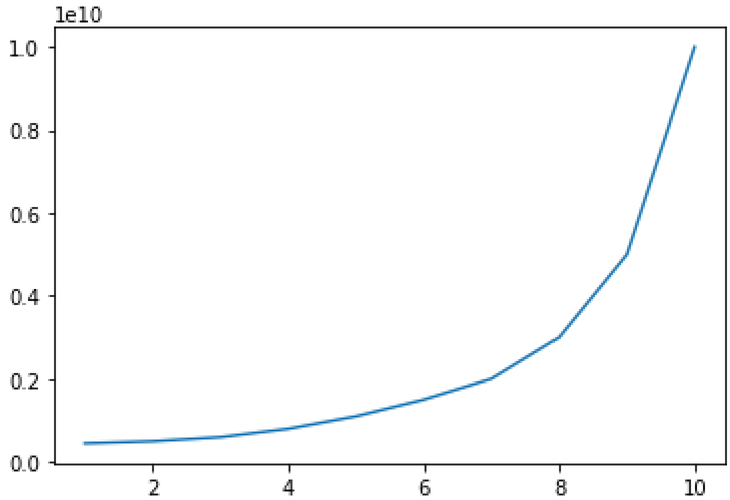
```
#1. Load the salary dataset to working environment.

import pandas as pd
train_data = pd.read_csv("C:/Users/student/Downloads/position_salaries.csv")
```

In [14]:

```
# 2. Perform exploratory analysis on the loaded dataset and draw your inferences

import matplotlib.pyplot as plt

x = train_data["Years of Experience"]
y = train_data["Salary"]
y_val = y*10000
plt.plot(x,y_val)
```

Out[14]:

```
[<matplotlib.lines.Line2D at 0x1ce853269d0>]
```



In [ ]:

```
# 3. Apply Linear Regression and design a model on the training data.
import LinearRegression
```

In [ ]:

```
#Name: SIVAA V
#Reg: 22MCA1127
#Lab Exercise: 6
```

In [23]:

```
#Load the breast cancer dataset form sklearn.
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
```

In [24]:

```
#If require perform data preprocessing.
from sklearn.preprocessing import StandardScaler

x= data.data
y = data.target
scaler = StandardScaler()
X = scaler.fit_transform(x)
```

In [25]:

```
#Split the dataset into training and testing by 90:10 ratios
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=10)
```

In [26]:

```
#Apply linear SVM on the training dataset.
from sklearn.svm import LinearSVC
svm = LinearSVC()
svm.fit(X_train, y_train)
```

Out[26]:

```
LinearSVC()
```

In [27]:

```
#Use the above model and fit the test dataset.
y_predicted = svm.predict(X_test)
```

In [28]:

```
#Display  the  accuracy  and  confusion  matrix  of  evaluated  model  on  test data.
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(y_test, y_predicted)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9473684210526315
```

In [53]:

```
#Apply SVM on the training dataset using polynomial kernel.
from sklearn.svm import SVC

svm = SVC(kernel='poly', degree=3)
svm.fit(X_train, y_train)
```

Out[53]:

```
SVC(kernel='poly')
```

In [54]:

```
#Use the above model and fit the test dataset.
y_predicted_poly = svm.predict(X_test)
```

In [55]:

```
#Display  the  accuracy  and  confusion  matrix  of  evaluated  model  on  test data

accuracy = accuracy_score(y_test, y_predicted_poly)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.8771929824561403
```

In [56]:

```
#Apply SVM on the training dataset using RBF kernel.

svm = SVC(kernel='rbf')
svm.fit(X_train, y_train)
```

Out[56]:

```
SVC()
```

In [57]:

```
#Use the above model and fit the test dataset.
y_predicted_rbf = svm.predict(X_test)
```

In [58]:

```
#Display the accuracy and confusion matrix of evaluated model on test data

accuracy = accuracy_score(y_test, y_predicted_rbf)
print("Accuracy:", accuracy)
```

Accuracy: 0.9649122807017544

In [ ]:

```

```

In [ ]:

```
#Name: SIVAA V
#Reg; 22MCA1127
#Lab Exercise: 7
```

In [79]:

```

#1.Load the dataset (iris.csv).
import pandas as pd
data = pd.read_csv("C:/Users/student/Downloads/archive/Iris.csv")
print(data)
```

```
      Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  \
0      1            5.1           3.5            1.4           0.2
1      2            4.9           3.0            1.4           0.2
2      3            4.7           3.2            1.3           0.2
3      4            4.6           3.1            1.5           0.2
4      5            5.0           3.6            1.4           0.2
..   ...            ...           ...            ...           ...
145  146            6.7           3.0            5.2           2.3
146  147            6.3           2.5            5.0           1.9
147  148            6.5           3.0            5.2           2.0
148  149            6.2           3.4            5.4           2.3
149  150            5.9           3.0            5.1           1.8

            Species
0       Iris-setosa
1       Iris-setosa
2       Iris-setosa
3       Iris-setosa
4       Iris-setosa
..              ...
145  Iris-virginica
146  Iris-virginica
147  Iris-virginica
148  Iris-virginica
149  Iris-virginica

[150 rows x 6 columns]
```

In [80]:

```
#2.Split dataset into test and train (20:80)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data.drop('Species', axis=1), data['S
```

In [81]:

```
#3.Build KNN classifier with k value as 2 for identifying the flower Species
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=2)
knn.fit(X_train, y_train)
y_pred_2 = knn.predict(X_test)
```

In [82]:

```
#4.Build KNN classifier with k value as 4 for identifying the flower Species.
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=4)
knn.fit(X_train, y_train)
y_pred_4 = knn.predict(X_test)
```

In [83]:

```
#5. Evaluate the step-3 and step-4.
print("Predicted values for Step-3")
print(y_pred_2)
print("\nPredicted values for Step-4")
print(y_pred_4)
```

```
Predicted values for Step-3
['Iris-virginica' 'Iris-versicolor' 'Iris-setosa' 'Iris-virginica'
 'Iris-setosa' 'Iris-virginica' 'Iris-setosa' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-virginica' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-setosa'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-setosa' 'Iris-setosa'
 'Iris-virginica' 'Iris-versicolor' 'Iris-setosa' 'Iris-setosa'
 'Iris-virginica' 'Iris-setosa' 'Iris-setosa' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-setosa']

Predicted values for Step-4
['Iris-virginica' 'Iris-versicolor' 'Iris-setosa' 'Iris-virginica'
 'Iris-setosa' 'Iris-virginica' 'Iris-setosa' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-virginica' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-setosa'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-setosa' 'Iris-setosa'
 'Iris-virginica' 'Iris-versicolor' 'Iris-setosa' 'Iris-setosa'
 'Iris-virginica' 'Iris-setosa' 'Iris-setosa' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-setosa']
```

In [84]:

```
#6.Design a method for calculating the distance between data points for the given dataset
import numpy as np

def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((x1 - x2)**2))
d = euclidean_distance(X_train['SepalLengthCm'],X_train['SepalWidthCm'])
print(d)
```

```
32.75561020649745
```

In [89]:

```
#7. Design a method for finding the nearest neighbours of a given data point using the ab
import numpy as np

def find_nearest_neighbor(X, y, x_query):
    distances = []
    for i, x in enumerate(X):
        distance = euclidean_distance(x, x_query)
        distances.append((distance, y[i]))
    distances.sort()
    return distances[0][1]

y_pred=[]
for i, x in enumerate(X_test):
    y_pred.append(find_nearest_neighbor(X_train, y_train, x))
print(y_pred)
```

```
[51.732763231999996, 22.89749796, 59.09498796, 35.3011712, 10.535882756, 4
4.27814872, 23.696600644, 51.0556978, 26.310404159999997, 48.79421652, 27.
827251359999998, 18.28490352, 54.896079119999996, 28.296095039999997, 38.2
0386516, 39.699338604, 40.062002979999995, 9.731264264, 37.171030112, 31.8
12422639999998, 32.051670812, 36.56291228, 45.08483564, 15.74832131599999
8, 18.01600788, 50.60064364, 17.3403214, 41.053469468, 59.30183076, 20.966
96516, 37.437857324, 27.742445811999996, 41.684340008, 46.2293658, 19.4156
4416, 37.721921436, 26.310404159999997, 33.798802996, 27.874825204, 47.401
475, 26.848195439999998, 10.089791784, 43.25082948, 46.68441996, 13.355150
12, 77.29715436, 59.494884039999995, 41.244454319999996, 27.53077668, 10.3
938507, 57.21823428799999, 39.700028079999996, 24.3385028, 27.935499091999
997, 69.83702404, 46.20178676, 26.310404159999997, 11.483912255999998, 38.
56239268, 19.691434559999998, 33.798802996, 15.615941924, 40.85834776, 36.
3009114, 15.030576799999999, 18.033934256, 33.398217439999996, 12.18097249
2, 39.64487, 13.12072828, 14.596896395999998, 10.787541496, 11.36256448, 2
7.827251359999998, 26.147687824, 24.44881896, 56.141962252, 51.0556978, 5
7.915983999999995, 8.06342182, 39.358047983999995, 28.096147, 34.6806428,
39.64487, 43.698299404, 35.3011712, 44.611855103999999, 33.7153764, 7.31534
03599999995, 22.31833812, 31.84000168, 73.69808963999999, 9.4458212, 56.61
9079643999996, 46.234192132, 46.931941843999994, 19.519065559999998, 40.23
09246, 39.58281716, 37.231704, 24.065470304, 43.798273423999994, 33.116911
232, 24.00065956, 69.29923276, 12.052040479999999, 50.93848688, 30.1232064
4, 45.939785879999995, 67.69964844, 44.296075096, 45.08483564, 32.82319445
6, 44.27814872, 35.75425543054136, 55.25460664, 22.53207568, 33.75674496,
55.50971276, 46.234192132, 17.236210524, 21.946021079999998, 46.6430514, 2
7.337723399999998, 24.579819399999998, 31.2677366, 52.61391356, 49.2010073
6, 42.13387836, 32.39847724, 11.483912255999998, 23.13881456, 7.8393421199
999995, 65.99664272, 10.222171176, 41.15137506, 17.43684804, 45.83636448,
22.94576128, 37.363393916, 21.29101888, 31.178794196000002, 38.20386516, 3
4.6806428, 46.234192132, 43.57833058, 26.8550902, 13.19657064, 27.87482520
4, 14.2032056, 34.73580088, 15.520104759999999, 45.89841732, 21.29101888,
48.67011084, 42.347615919999996, 23.835874796, 61.921839559999995, 22.8974
9796, 41.053469468, 9.131420144, 45.69846928, 34.67374804, 55.064311264, 3
3.398217439999996, 68.75060685418796, 37.427515183999994, 17.22311048, 38.
01770664, 33.35684888, 38.76923548, 9.694722036, 24.986610239999997, 40.67
9083999999996, 36.349864196, 17.596806471999997, 17.165883972, 42.03045695
9999995, 41.053469468, 29.58541516, 30.647208199999998, 26.91714304, 67.69
964844, 29.58541516, 42.21661548, 51.732763231999996, 22.435549039999998,
38.500339839999995, 35.101223159999996, 11.85209244, 25.510612, 35.8527519
99999995, 40.56876784, 32.109882284729316, 33.70158688, 15.5201047599999
9, 37.2661778, 15.520104759999999, 19.98790924, 18.28490352, 30.44726016,
55.25460664, 10.222171176, 24.40055564, 65.69721314581203, 31.35047372]
```

In [90]:

```
#8.Design a method predicting the data point using the above two methods.
import numpy as np

def predict_knn(X_train, y_train, x_query, k):
    distances = []
    for i, x in enumerate(X_train):
        distance = euclidean_distance(x, x_query)
        distances.append((distance, y_train[i]))
    distances.sort()
    neighbors = [distances[i][1] for i in range(k)]
    counts = np.bincount(neighbors)
    return np.argmax(counts)

y_pred = []
for i, x in enumerate(X_test):
    y_pred.append(predict_knn_regression(X_train, y_train, x, 5))
print(y_pred)
```

[52.0659180352, 44.13887456799999, 64.33776346399999, 39.119489288, 12.211
171540799999, 38.695047862399996, 34.0528059544, 48.29917275199999, 25.524
401520000005, 38.827151464, 32.1717775312, 18.2195411952, 51.5783206079999
9, 49.266369684800004, 40.473620152, 32.8380871376, 32.9826013072, 20.5072
22563199996, 37.3780108072, 34.6116952, 31.55386914, 41.0927696, 41.295475
544, 24.337399638399997, 32.551540912, 34.175946368, 14.167352847999998, 3
7.6470443424, 50.528938136, 24.670830231999997, 36.8926197032, 29.15973267
7599997, 29.659602777599996, 47.150505736, 29.479235855999995, 35.25276998
48, 25.56577008, 40.1717675592, 13.775730479999998, 46.924357607999994, 1
8.469545192800002, 12.617410800000002, 35.1216316496, 39.771733584, 18.100
123951999997, 74.05661716, 42.76701432232481, 38.816119848, 20.830448912,
21.9890443824, 56.914864848, 40.57000889679999, 30.836124624, 26.174853178
399996, 65.49194628799998, 39.119489288, 18.12494508799997, 16.091818259
2, 36.220932184000006, 18.4079060384, 40.1717675592, 22.2316020392, 42.926
77576, 51.638994495999995, 22.887983191199996, 15.9664715224, 44.425834479
19999, 14.916951155200001, 32.097865704, 22.67686564, 16.6036852416, 14.45
85875104, 12.384367912, 32.1717775312, 29.214477072, 19.135716904, 51.8308
067192, 48.121287943999995, 48.10552849316241, 17.7847576296, 39.461745174
4, 46.478956112000006, 32.774241659999994, 31.652464207999998, 44.73706394
5600006, 39.119489288, 30.535513088, 35.501808716, 20.121667584, 20.383668
464, 28.074773244, 65.578820264, 17.038330912, 57.3157261944, 39.821375855
999996, 45.9520585528, 24.425376775999997, 23.926196152000003, 21.49827536
56, 35.545245703999996, 29.6455374672, 40.8994405296, 42.5670071832, 32.07
0700349599996, 66.357928144, 15.2385227616, 52.644250504, 33.6435330008, 3
6.62220721600001, 49.392819583199994, 43.30474650319999, 43.486630272, 29.
4061514, 42.45793207999999, 41.388062322324814, 52.498081592000005, 25.983
592535999996, 33.70379320319999, 44.31262252, 39.821375855999996, 31.55386
914, 26.6152904472, 43.91134748799999, 29.912226783999994, 26.8390943568,
34.74269564, 50.476537959999995, 45.09862516, 43.6101843712, 38.6950478623
99996, 16.0918182592, 32.704604584, 15.272169190400001, 59.15704079999999,
15.7437707744, 43.424163746400005, 20.829069959999998, 42.860586064, 31.08
7093888000005, 32.83850082319999, 23.851456953599996, 35.113220042399995,
40.473620152, 31.117430831999997, 39.821375855999996, 39.290203545599994,
28.370006867199997, 14.916951155200001, 10.3209041392, 18.9576942008, 36.6
18070360000004, 12.211171540799999, 35.536971992, 21.987113849599996, 45.0
95867256, 40.312282768, 32.9826013072, 63.11463304, 44.13887456799999, 41.
65386516880001, 15.304160876800001, 49.618829816, 31.087093888000005, 51.1
6449711279999, 45.09862516, 50.11485855010827, 39.4265818984, 22.67686564,
32.209560816, 29.346856463999995, 40.109576824, 22.7434690216, 22.52104406
4, 37.0022463872, 30.2311783816, 16.672494946399997, 15.304160876800001, 4
2.601343088, 41.65386516880001, 33.42855438400001, 32.687919264799994, 22.
483674464799996, 49.392819583199994, 34.173188464000006, 29.12622414400000
2, 51.7131821136, 21.7962668928, 27.334965496000002, 35.699688328, 17.5375
11536, 35.536971992, 40.040629224, 41.347875720000005, 54.19261660705414,
35.407350504, 28.021683592, 41.15344348799999, 12.211171540799999, 25.0606
59962400003, 20.4640613656, 41.72708752, 52.49808159199999, 20.8639574456,
56.56598999199999, 54.19261660705414, 34.203525408000004]

In [87]:

```
#9.Choose any dataset from Kaggle or UCI repository suitable for regression and apply KNN

import pandas as pd
from sklearn.model_selection import train_test_split
import numpy as np

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/concrete/compressive/Con
df = pd.read_excel(url)
def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((x1 - x2)**2))

def predict_knn_regression(X_train, y_train, x_query, k):
    distances = []
    for i, x in enumerate(X_train):
        distance = euclidean_distance(x, x_query)
        distances.append((distance, y_train[i]))
    distances.sort()
    neighbors = [distances[i][1] for i in range(k)]
    return np.mean(neighbors)

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=0.2, random_state=42)
y_pred = []
for i, x in enumerate(X_test):
    y_pred.append(predict_knn_regression(X_train, y_train, x, k=5))
```

In [88]:

```
#10.Evaluate the designed regression model with appropriate metric
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)
print(f'Mean Absolute Error: {mae}')
print(f'Mean Squared Error: {mse}')
print(f'Root Mean Squared Error: {rmse}')
print(f'R-squared: {r2}')
```

```
Mean Absolute Error: 6.459309144164602
Mean Squared Error: 67.95464317359179
Root Mean Squared Error: 8.243460630923872
R-squared: 0.7362839326848325
```

In [ ]:

In [ ]:

```

```

In [ ]:

```
#Name: SIVAA V
#Reg: 22MCA1127
#Lab Exercise: 8
```

In [1]:

```
#Load the dataset (titanic).
import pandas as pd
data = pd.read_csv("C:/Users/student/Downloads/titanic/train.csv")
print(data)
```

```
     PassengerId  Survived  Pclass  \
0              1         0       3   
1              2         1       1   
2              3         1       3   
3              4         1       1   
4              5         0       3   
..           ...       ...     ...   
886          887         0       2   
887          888         1       1   
888          889         0       3   
889          890         1       1   
890          891         0       3   

                                                  Name     Sex   Age  SibS
p  \
0                              Braund, Mr. Owen Harris    male  22.0      
1   
1    Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      
1   
2                               Heikkinen, Miss. Laina  female  26.0      
0   
3         Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      
1   
4                             Allen, Mr. William Henry    male  35.0      
0   
..                                                 ...     ...   ...      
...   
886                              Montvila, Rev. Juozas    male  27.0      
0   
887                       Graham, Miss. Margaret Edith  female  19.0      
0   
888           Johnston, Miss. Catherine Helen "Carrie"  female   NaN      
1   
889                              Behr, Mr. Karl Howell    male  26.0      
0   
890                                Dooley, Mr. Patrick    male  32.0      
0   

     Parch            Ticket     Fare Cabin Embarked  
0        0         A/5 21171   7.2500   NaN        S  
1        0          PC 17599  71.2833   C85        C  
2        0  STON/O2. 3101282   7.9250   NaN        S  
3        0            113803  53.1000  C123        S  
4        0            373450   8.0500   NaN        S  
..     ...               ...      ...   ...      ...  
886      0            211536  13.0000   NaN        S  
887      0            112053  30.0000   B42        S  
888      2        W./C. 6607  23.4500   NaN        S  
889      0            111369  30.0000  C148        C  
890      0            370376   7.7500   NaN        Q  

[891 rows x 12 columns]
```

In [17]:

```
#2.Split dataset into test and train (20:80)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data, data, test_size=0.2, random_sta

X_train = X_train.drop(['Pclass','Name','Sex','Age','SibSp','Parch','Ticket','Fare','Cabi
X_test = X_test.drop(['Pclass','Name','Sex','Age','SibSp','Parch','Ticket','Fare','Cabin'

y_train = y_train.drop(['Pclass','Name','Sex','Age','SibSp','Parch','Ticket','Fare','Cabi
y_test = y_test.drop(['Pclass','Name','Sex','Age','SibSp','Parch','Ticket','Fare','Cabin'

print(y_train)
```

```
     Survived
140         0
439         0
817         0
378         0
491         0
..        ...
835         1
192         1
629         0
559         1
684         0

[712 rows x 1 columns]
```

In [18]:

```
#3.Build  KNN  classifier  with  k  value  as  2  for  identifying  the Passenger Survive
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=2)
knn.fit(X_train, y_train.values.reshape(-1,))
y_pred_2 = knn.predict(X_test)
```

In [20]:

```
#4.Build KNN classifier with k value as 4 for identifying the the Passenger Survived.
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=4)
knn.fit(X_train, y_train.values.reshape(-1,))
y_pred_4 = knn.predict(X_test)
```

In [22]:

```python
#5. Evaluate the step-3 and step-4.
print("Predicted values for Step-3")
print(X_test)
print(y_pred_2)
print(X_test)
print("\nPredicted values for Step-4")
print(y_pred_4)
```

```
Predicted values for Step-3
     PassengerId
495          496
648          649
278          279
31            32
255          256
..           ...
780          781
837          838
215          216
833          834
372          373

[179 rows x 1 columns]
[0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 1 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 1 0 0 0 0 0 0 0 1 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 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 1 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 1 0 0 0 0]
     PassengerId
495          496
648          649
278          279
31            32
255          256
..           ...
780          781
837          838
215          216
833          834
372          373

[179 rows x 1 columns]

Predicted values for Step-4
[0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 1 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 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 1 0 0 0 0 0 0 0 0 0 0 1 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 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
```

In [25]:

```
#6.Design a method for calculating the distance between data points for the given dataset
import numpy as np

def euclidean_distance(x1, y1):
    return np.sqrt(np.sum((x1 - y1)**2))
d = euclidean_distance(X_test,y_test)
print(d)
```

```
PassengerId    0.0
Survived       0.0
dtype: float64
```

In [27]:

```
#7. Design a method for finding the nearest neighbours of a given data point using the ab
import numpy as np

def find_nearest_neighbor(X, y, x_query):
    distances = []
    for i, x in enumerate(X):
        distance = euclidean_distance(x, x_query)
        distances.append((distance, y[i]))
    distances.sort()
    return distances[0][1]
```

In [33]:

```
#8.Design a method predicting the data point using the above two methods.
import numpy as np
import pandas as pd
def predict_knn_regression(X_train, y_train, x_query, k):
    distances = []
    for i, x in enumerate(X_train):
        distance = euclidean_distance(x, x_query)
        distances.append((distance, y_train[i]))
    distances.sort()
    neighbors = [distances[i][1] for i in range(k)]
    return np.mean(neighbors)

y_pred = []
for i, x in enumerate(X_test):
    y_pred.append(predict_knn_regression(X_train, y_train, x, k=5))
```

```
[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, 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, 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, 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, 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]
```

In [ ]:

```

```

In [ ]:

```
#Name: SIVAA V
#Reg: 22MCA1127
#Lab Exercise: 9
```

In [1]:

```
#1.Load the dataset (iris.csv)

import pandas as pd
data = pd.read_csv("Iris.csv")
print(data)
```

```
      Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  \
0      1            5.1           3.5            1.4           0.2
1      2            4.9           3.0            1.4           0.2
2      3            4.7           3.2            1.3           0.2
3      4            4.6           3.1            1.5           0.2
4      5            5.0           3.6            1.4           0.2
..   ...            ...           ...            ...           ...
145  146            6.7           3.0            5.2           2.3
146  147            6.3           2.5            5.0           1.9
147  148            6.5           3.0            5.2           2.0
148  149            6.2           3.4            5.4           2.3
149  150            5.9           3.0            5.1           1.8

            Species
0       Iris-setosa
1       Iris-setosa
2       Iris-setosa
3       Iris-setosa
4       Iris-setosa
..              ...
145  Iris-virginica
146  Iris-virginica
147  Iris-virginica
148  Iris-virginica
149  Iris-virginica

[150 rows x 6 columns]
```

In [2]:

```
#2.Load the dataset (Churnprediction.csv)

import pandas as pd
train_data = pd.read_csv("C:/users/student/Machine Learning/customer-churn-prediction-202
print(train_data)
```

```
     state  account_length      area_code international_plan voice_mail_pl
an  \
0       OH             107  area_code_415                 no             y
es
1       NJ             137  area_code_415                 no
no
2       OH              84  area_code_408                yes
no
3       OK              75  area_code_415                yes
no
4       MA             121  area_code_510                 no             y
es
...    ...             ...            ...                ...
...
4245    MT              83  area_code_415                 no
no
4246    WV              73  area_code_408                 no
no
4247    NC              75  area_code_408                 no
no
4248    HI              50  area_code_408                 no             y
es
4249    VT              86  area_code_415                 no             y
es

      number_vmail_messages  total_day_minutes  total_day_calls  \
0                        26              161.6              123
1                         0              243.4              114
2                         0              299.4               71
3                         0              166.7              113
4                        24              218.2               88
...                     ...                ...              ...
4245                      0              188.3               70
4246                      0              177.9               89
4247                      0              170.7              101
4248                     40              235.7              127
4249                     34              129.4              102

      total_day_charge  total_eve_minutes  total_eve_calls  total_eve_char
ge  \
0                27.47              195.5              103             16.
62
1                41.38              121.2              110             10.
30
2                50.90               61.9               88              5.
26
3                28.34              148.3              122             12.
61
4                37.09              348.5              108             29.
62
...                ...                ...              ...
...
4245             32.01              243.8               88             20.
72
4246             30.24              131.2               82             11.
15
4247             29.02              193.1              126             16.
41
4248             40.07              223.0              126             18.
96
4249             22.00              267.1              104             22.
```

```
70

      total_night_minutes  total_night_calls  total_night_charge  \
0                   254.4                103               11.45   
1                   162.6                104                7.32   
2                   196.9                 89                8.86   
3                   186.9                121                8.41   
4                   212.6                118                9.57   
...                   ...                ...                 ...   
4245                213.7                 79                9.62   
4246                186.2                 89                8.38   
4247                129.1                104                5.81   
4248                297.5                116               13.39   
4249                154.8                100                6.97   

      total_intl_minutes  total_intl_calls  total_intl_charge  \
0                   13.7                 3               3.70   
1                   12.2                 5               3.29   
2                    6.6                 7               1.78   
3                   10.1                 3               2.73   
4                    7.5                 7               2.03   
...                  ...               ...                ...   
4245                10.3                 6               2.78   
4246                11.5                 6               3.11   
4247                 6.9                 7               1.86   
4248                 9.9                 5               2.67   
4249                 9.3                16               2.51   

      number_customer_service_calls churn  
0                                 1    no  
1                                 0    no  
2                                 2    no  
3                                 3    no  
4                                 3    no  
...                             ...   ...  
4245                              0    no  
4246                              3    no  
4247                              1    no  
4248                              2    no  
4249                              0    no  

[4250 rows x 20 columns]
```

In [7]:

```
#3.Drop columns that are not required for classification of Churn Risk.
train_data.drop(columns=['state', 'area_code'],axis=1,inplace=True)
train_data.drop(columns=['total_day_minutes', 'total_eve_minutes', 'total_night_minutes',

print(train_data)
```

```
      account_length international_plan voice_mail_plan  \
0                107                 no             yes   
1                137                 no              no   
2                 84                yes              no   
3                 75                yes              no   
4                121                 no             yes   
...              ...                ...             ...   
4245              83                 no              no   
4246              73                 no              no   
4247              75                 no              no   
4248              50                 no             yes   
4249              86                 no             yes   

      number_vmail_messages  total_day_calls  total_day_charge  \
0                        26              123             27.47   
1                         0              114             41.38   
2                         0               71             50.90   
3                         0              113             28.34   
4                        24               88             37.09   
...                     ...              ...               ...   
4245                      0               70             32.01   
4246                      0               89             30.24   
4247                      0              101             29.02   
4248                     40              127             40.07   
4249                     34              102             22.00   

      total_eve_calls  total_eve_charge  total_night_calls  \
0                 103             16.62                103   
1                 110             10.30                104   
2                  88              5.26                 89   
3                 122             12.61                121   
4                 108             29.62                118   
...               ...               ...                ...   
4245               88             20.72                 79   
4246               82             11.15                 89   
4247              126             16.41                104   
4248              126             18.96                116   
4249              104             22.70                100   

      total_night_charge  total_intl_calls  total_intl_charge  \
0                  11.45                 3               3.70   
1                   7.32                 5               3.29   
2                   8.86                 7               1.78   
3                   8.41                 3               2.73   
4                   9.57                 7               2.03   
...                  ...               ...                ...   
4245                9.62                 6               2.78   
4246                8.38                 6               3.11   
4247                5.81                 7               1.86   
4248               13.39                 5               2.67   
4249                6.97                16               2.51   

      number_customer_service_calls churn  
0                                 1    no  
1                                 0    no  
2                                 2    no  
3                                 3    no  
4                                 3    no  
...                             ...   ...  
4245                              0    no  
4246                              3    no  
```

```
4247                                 1    no
4248                                 2    no
4249                                 0    no

[4250 rows x 14 columns]
```

In [8]:

```
#4.If require perform data preprocessing.
train_data['international_plan'].replace({'yes':1,'no':0},inplace=True)
train_data['voice_mail_plan'].replace({'yes':1,'no':0},inplace=True)
train_data['churn'].replace({'yes':1,'no':0},inplace=True)
```

In [23]:

```
#5.Split dataset into test and train (20:80)
X = train_data.drop("churn", axis=1)
y = train_data["churn"]

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state =
```

In [13]:

```
#6.Build any three classification models for identifying Churn Risk.

#Random Forest
X = train_data.drop("churn", axis=1)
y = train_data["churn"]

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state =
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(X_train)
x_test= st_x.transform(X_test)

from sklearn.ensemble import RandomForestClassifier
classifier= RandomForestClassifier(n_estimators= 10, criterion="entropy")
classifier.fit(x_train, y_train)
y_pred= classifier.predict(x_test)
print(y_pred)
```

```
[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 1 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 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 1 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 1 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 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 1 0 0 0 1 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 1
 0 0 0 0 0 0 1 0 1 0 0 0 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 1 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 1 0 0 0 0 0 1 0 1
 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 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 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 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 0 1 0 0 0 0 0 0
 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1
 0 0 0 0 0 0 0 1 0 0 0 0 0 1 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 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0
 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0
 0 0 0 0 0 1 1 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 1 0 1 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 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 0 0 1 0 0 0 0 0 0 0 0 0
 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0
 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0
 1 1 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 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 1 0 0 0 0]
```

In [20]:

```python
#Naive Bayes
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import StandardScaler
X = train_data.drop("churn", axis=1)
y = train_data["churn"]

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state =
sc = StandardScaler()
x_train = sc.fit_transform(X_train)
x_test = sc.transform(X_test)

model = GaussianNB()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
print(y_pred)
```

```
[1 1 0 1 0 0 1 0 0 1 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 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0 1 0 0 0 1 0 0 0
 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 1
 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 1 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0
 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 1 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 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 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 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 0 0 0 0 0 0 1
 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 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 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 1 0 0 0 0 0 0
 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 1 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 1 1
 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 1 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 1 0 0 0 0 1 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0
 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0
 0 0 0 0 0 1 0 0 0 1 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 1 0 0 0 0 1 0 0 1 0 0 0 0 0 1 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 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 0
 0 0 1 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 1 0 0 0 0 0
 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0
 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 1 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 1 0 0 0 0 0 0 0 0 1 0 0 0]
```

In [22]:

```python
#Logistic Regression
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
X = train_data.drop("churn", axis=1)
y = train_data["churn"]

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state =
st_x= StandardScaler()
x_train= st_x.fit_transform(X_train)
x_test= st_x.transform(X_test)

classifier = LogisticRegression(random_state=0)
classifier.fit(x_train, y_train)
y_pred = classifier.predict(x_test)
print(y_pred)
```

```
[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 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 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
 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 1 0 0 0 1 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 1 0 0 1 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 0 0 0 0 1 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 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 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 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 0 0 0 0 0 0 1 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 1 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 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 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 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 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 1 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 1 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 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 1 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 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 1 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

In [30]:

```
#7.Build Voting ensemble classifier on the training dataset.

from sklearn.ensemble import VotingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

X = train_data.drop("churn", axis=1)
y = train_data["churn"]

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state =
classifier1 = DecisionTreeClassifier()
classifier2 = LogisticRegression()
classifier3 = SVC()

voting_classifier = VotingClassifier(
    estimators=[('dt', classifier1), ('lr', classifier2), ('svc', classifier3)],
    voting='hard'
)

voting_classifier.fit(X_train, y_train)

y_pred = voting_classifier.predict(X_test)
```

```
/home/sivaa/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_lo
gistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sc
ikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression (https://scikit-learn.org/stable/modules/linear_model.html#logisti
c-regression)
  n_iter_i = _check_optimize_result(
```

In [31]:

```
#8.Build Bagging ensemble classifier on the training dataset.

from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score


X = train_data.drop("churn", axis=1)
y = train_data["churn"]

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state =
base_classifier = DecisionTreeClassifier()

bagging_classifier = BaggingClassifier(
    base_estimator=base_classifier,
    n_estimators=10,
    random_state=42
)

bagging_classifier.fit(X_train, y_train)

y_pred = bagging_classifier.predict(X_test)
```

In [32]:

```
#9.Build Boosting ensemble classifier on the training dataset
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

X = train_data.drop("churn", axis=1)
y = train_data["churn"]

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state =

boosting_classifier = GradientBoostingClassifier(
    n_estimators=100,
    learning_rate=0.1
)

boosting_classifier.fit(X_train, y_train)

y_pred = boosting_classifier.predict(X_test)
```

In [ ]:

```
#10.Fit the models designed from step-5 to step-8 on the test dataset.

#Random Forest
X = train_data.drop("churn", axis=1)
y = train_data["churn"]

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state =
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(X_train)
x_test= st_x.transform(X_test)

from sklearn.ensemble import RandomForestClassifier
classifier= RandomForestClassifier(n_estimators= 10, criterion="entropy")
classifier.fit(x_test, y_test)
y_pred= classifier.predict(x_train)
print(y_pred)

#Naive Bayes
from sklearn.naive_bayes import GaussianNB

x_train = sc.fit_transform(X_train)
x_test = sc.transform(X_test)

model = GaussianNB()
model.fit(x_test, y_test)
y_pred = model.predict(x_train)


#Logistic Regression
classifier = LogisticRegression(random_state=0)
classifier.fit(x_test, y_test)
y_pred = classifier.predict(x_train)
print(y_pred)
```

In [39]:

```
#11.Evaluate  the  designed  models  from  step-5  to  step-8  with  appropriate classifi

X = train_data.drop("churn", axis=1)
y = train_data["churn"]

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state =
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(X_train)
x_test= st_x.transform(X_test)

from sklearn.ensemble import RandomForestClassifier
classifier= RandomForestClassifier(n_estimators= 10, criterion="entropy")
classifier.fit(x_train, y_train)
y_pred= classifier.predict(x_test)

from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print("Random Forest Accuracy: ",accuracy)

from sklearn.metrics import precision_score
precision = precision_score(y_test, y_pred)
print("Random Forest Precision: ",precision)

from sklearn.metrics import recall_score
recall = recall_score(y_test, y_pred)
print("Random Forest Recall: ",recall)

from sklearn.metrics import f1_score
f1 = f1_score(y_test, y_pred)
print("Random Forest F1 score: ",f1)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

from sklearn.metrics import classification_report
report = classification_report(y_test, y_pred)
print("Random Forest Classification")
print(report)


#Naive Bayes
from sklearn.naive_bayes import GaussianNB

x_train = sc.fit_transform(X_train)
x_test = sc.transform(X_test)

model = GaussianNB()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)


from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print("Naive Bayes Accuracy: ",accuracy)

from sklearn.metrics import precision_score
precision = precision_score(y_test, y_pred)
```

```
print("Naive Bayes Precision: ",precision)

from sklearn.metrics import recall_score
recall = recall_score(y_test, y_pred)
print("Naive Bayes Recall: ",recall)

from sklearn.metrics import f1_score
f1 = f1_score(y_test, y_pred)
print("Naive Bayes F1 score: ",f1)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

from sklearn.metrics import classification_report
report = classification_report(y_test, y_pred)
print("Naive Bayes Classification")
print(report)

#Logistic Regression
classifier = LogisticRegression(random_state=0)
classifier.fit(x_train, y_train)
y_pred = classifier.predict(x_test)

from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print("Logistic Regression Accuracy: ",accuracy)

from sklearn.metrics import precision_score
precision = precision_score(y_test, y_pred)
print("Logistic Regression Precision: ",precision)

from sklearn.metrics import recall_score
recall = recall_score(y_test, y_pred)
print("Logistic Regression Recall: ",recall)

from sklearn.metrics import f1_score
f1 = f1_score(y_test, y_pred)
print("Logistic Regression F1 score: ",f1)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

from sklearn.metrics import classification_report
report = classification_report(y_test, y_pred)
print("Logistic Regression Classification")
print(report)
```

```
Random Forest Accuracy:  0.9423529411764706
Random Forest Precision:  0.9294117647058824
Random Forest Recall:  0.6475409836065574
Random Forest F1 score:  0.7632850241545894
Random Forest Classification
              precision    recall  f1-score   support

           0       0.94      0.99      0.97       728
           1       0.93      0.65      0.76       122

    accuracy                           0.94       850
   macro avg       0.94      0.82      0.87       850
weighted avg       0.94      0.94      0.94       850

Naive Bayes Accuracy:  0.8470588235294118
Naive Bayes Precision:  0.46296296296296297
Naive Bayes Recall:  0.4098360655737705
Naive Bayes F1 score:  0.43478260869565216
Naive Bayes Classification
              precision    recall  f1-score   support

           0       0.90      0.92      0.91       728
           1       0.46      0.41      0.43       122

    accuracy                           0.85       850
   macro avg       0.68      0.67      0.67       850
weighted avg       0.84      0.85      0.84       850

Logistic Regression Accuracy:  0.8623529411764705
Logistic Regression Precision:  0.5609756097560976
Logistic Regression Recall:  0.1885245901639344
Logistic Regression F1 score:  0.2822085889570552
Logistic Regression Classification
              precision    recall  f1-score   support

           0       0.88      0.98      0.92       728
           1       0.56      0.19      0.28       122

    accuracy                           0.86       850
   macro avg       0.72      0.58      0.60       850
weighted avg       0.83      0.86      0.83       850
```

In [ ]:

```

```

In [ ]:

```
#Name: SIVAA V
#Reg: 22MCA1127
#Lab Exercise: 10
```

In [1]:

```
#1.Load the dataset (iris.csv).
import pandas as pd
data = pd.read_csv("C:/Users/student/Downloads/archive (2)/pima-indians-diabetes.csv")
print(data)
```

```
      6  148  72  35    0  33.6  0.627  50  1
0     1   85  66  29    0  26.6  0.351  31  0
1     8  183  64   0    0  23.3  0.672  32  1
2     1   89  66  23   94  28.1  0.167  21  0
3     0  137  40  35  168  43.1  2.288  33  1
4     5  116  74   0    0  25.6  0.201  30  0
..   ..  ...  ..  ..  ...   ...    ...  .. ..
762  10  101  76  48  180  32.9  0.171  63  0
763   2  122  70  27    0  36.8  0.340  27  0
764   5  121  72  23  112  26.2  0.245  30  0
765   1  126  60   0    0  30.1  0.349  47  1
766   1   93  70  31    0  30.4  0.315  23  0

[767 rows x 9 columns]
```

In [2]:

```
#2.check if there are missing values are present in the dataset.
data.isnull()
```

Out[2]:

	6	148	72	35	0	33.6	0.627	50	1
0	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
762	False	False	False	False	False	False	False	False	False
763	False	False	False	False	False	False	False	False	False
764	False	False	False	False	False	False	False	False	False
765	False	False	False	False	False	False	False	False	False
766	False	False	False	False	False	False	False	False	False

767 rows × 9 columns

In [5]:

```
#3. Perform data preprocessing.
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class']
data=pd.read_csv("C:/Users/student/Downloads/archive (2)/pima-indians-diabetes.csv",names
print(data)
```

```
     preg  plas  pres  skin  test  mass   pedi  age  class
0       6   148    72    35     0  33.6  0.627   50      1
1       1    85    66    29     0  26.6  0.351   31      0
2       8   183    64     0     0  23.3  0.672   32      1
3       1    89    66    23    94  28.1  0.167   21      0
4       0   137    40    35   168  43.1  2.288   33      1
..    ...   ...   ...   ...   ...   ...    ...  ...    ...
763    10   101    76    48   180  32.9  0.171   63      0
764     2   122    70    27     0  36.8  0.340   27      0
765     5   121    72    23   112  26.2  0.245   30      0
766     1   126    60     0     0  30.1  0.349   47      1
767     1    93    70    31     0  30.4  0.315   23      0

[768 rows x 9 columns]
```

In [6]:

```
#4.Split dataset into test and train (20:80).
from sklearn.model_selection import train_test_split

X=data.iloc[:,:-1]
y=data.iloc[:, -1:]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state =
```

In [7]:

```
#5.Build any three classification models for identifying diabetes.

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state =
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(X_train)
x_test= st_x.transform(X_test)

from sklearn.ensemble import RandomForestClassifier
classifier= RandomForestClassifier(n_estimators= 10, criterion="entropy")
classifier.fit(x_train, y_train)
y_pred= classifier.predict(x_test)
print(y_pred)
```

```
[1 1 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0
 0 0 0 0 1 0 0 0 0 0 1 0 1 0 1 0 0 1 1 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 1 0
 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 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 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 1 0 0 0 0
 0 0 1 0 0 0]

C:\Users\student\AppData\Local\Temp/ipykernel_1976/2771513193.py:12: DataC
onversionWarning: A column-vector y was passed when a 1d array was expecte
d. Please change the shape of y to (n_samples,), for example using ravel
().
  classifier.fit(x_train, y_train)
```

In [8]:

```python
#Naive Bayes
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class']
data=pd.read_csv("C:/Users/student/Downloads/archive (2)/pima-indians-diabetes.csv",names
X=data.iloc[:,:-1]
y=data.iloc[:, -1:]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state =
sc = StandardScaler()
x_train = sc.fit_transform(X_train)
x_test = sc.transform(X_test)

model = GaussianNB()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
print(y_pred)
```

```
[1 1 0 1 1 0 1 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0
 0 0 0 0 1 0 1 0 0 0 1 0 0 1 1 0 0 1 1 0 1 1 0 0 0 0 1 0 0 1 0 0 1 0 1 1 0
 1 0 1 1 0 1 1 1 0 1 1 1 0 0 0 0 1 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 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 1 0 0 0 0 1 1 1 1 0 0 0 0
 1 0 1 0 0 0]

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was ex
pected. Please change the shape of y to (n_samples, ), for example using r
avel().
  return f(*args, **kwargs)
```

In [10]:

```
#Logistic Regression
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class']
data=pd.read_csv("C:/Users/student/Downloads/archive (2)/pima-indians-diabetes.csv",names
X=data.iloc[:,:-1]
y=data.iloc[:, -1:]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state =
sc = StandardScaler()
x_train = sc.fit_transform(X_train)
x_test = sc.transform(X_test)

classifier = LogisticRegression(random_state=0)
classifier.fit(x_train, y_train)
y_pred = classifier.predict(x_test)
print(y_pred)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was ex
pected. Please change the shape of y to (n_samples, ), for example using r
avel().
  return f(*args, **kwargs)

[1 1 0 1 1 0 1 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0
 0 0 1 0 0 0 1 0 0 0 1 0 0 1 1 0 0 1 1 0 1 1 0 0 0 0 1 0 0 0 0 0 1 0 1 1 0
 0 0 1 0 0 1 1 1 0 1 1 0 0 0 0 0 0 0 1 1 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 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 1 0 1 0 0 0 0
 1 0 1 0 0 0]
```

In [19]:

```
#6.Build Voting ensemble classifier on the training dataset.
from sklearn.ensemble import VotingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class']
data=pd.read_csv("C:/Users/student/Downloads/archive (2)/pima-indians-diabetes.csv",names
X=data.iloc[:,:-1]
y=data.iloc[:, -1:]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state =
classifier1 = DecisionTreeClassifier()
classifier2 = LogisticRegression(solver='lbfgs',max_iter=100)
classifier3 = SVC()

voting_classifier = VotingClassifier(
    estimators=[('dt', classifier1), ('lr', classifier2), ('svc', classifier3)],
    voting='hard'
)

voting_classifier.fit(X_train, y_train)

y_pred = voting_classifier.predict(X_test)
print(y_pred)
```

```
[0 1 0 1 1 0 1 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0
 0 0 1 0 0 0 1 0 0 0 1 0 0 1 1 0 0 1 1 0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 1 1 0
 0 0 1 0 0 1 1 1 0 1 1 0 0 0 0 0 0 0 1 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 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 1 0 0 0 0
 1 0 1 0 0 0]

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was ex
pected. Please change the shape of y to (n_samples, ), for example using r
avel().
  return f(*args, **kwargs)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.
py:763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sc
ikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression (https://scikit-learn.org/stable/modules/linear_model.html#logisti
c-regression)
  n_iter_i = _check_optimize_result(
```

In [18]:

```
#7.Build Bagging ensemble classifier on the training dataset.
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class']
data=pd.read_csv("C:/Users/student/Downloads/archive (2)/pima-indians-diabetes.csv",names
X=data.iloc[:,:-1]
y=data.iloc[:, -1:]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state =
base_classifier = DecisionTreeClassifier()

bagging_classifier = BaggingClassifier(
    base_estimator=base_classifier,
    n_estimators=10,
    random_state=42
)

bagging_classifier.fit(X_train, y_train)

y_pred = bagging_classifier.predict(X_test)
print(y_pred)
```

```
[0 1 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0 0 0 1 1
 0 0 1 0 1 0 1 0 0 0 1 0 1 0 1 0 0 1 0 0 0 1 1 0 1 0 1 0 0 0 0 0 1 0 1 1 0
 0 0 1 0 0 0 1 0 0 1 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0
 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 1 1 0 0 0 0
 1 0 0 0 0 0]

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was ex
pected. Please change the shape of y to (n_samples, ), for example using r
avel().
  return f(*args, **kwargs)
```

In [20]:

```
#8.Build Boosting ensemble classifier on the training dataset.
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class']
data=pd.read_csv("C:/Users/student/Downloads/archive (2)/pima-indians-diabetes.csv",names
X=data.iloc[:,:-1]
y=data.iloc[:, -1:]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state =
boosting_classifier = GradientBoostingClassifier(
    n_estimators=100,
    learning_rate=0.1
)

boosting_classifier.fit(X_train, y_train)

y_pred = boosting_classifier.predict(X_test)

print(y_pred)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was ex
pected. Please change the shape of y to (n_samples, ), for example using r
avel().
  return f(*args, **kwargs)

[1 1 0 1 1 0 1 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0
 0 0 1 0 1 0 1 0 0 0 1 0 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 0 0 0 0 0 1 0 0 1 0
 0 1 1 0 0 1 1 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0
 0 0 0 0 0 0 1 1 1 1 0 0 1 0 0 0 1 0 1 1 0 0 1 0 1 0 0 0 0 1 0 1 1 0 0 0 0
 1 0 1 0 0 0]
```

In [23]:

```
#9.Fit the models designed from step-5 to step-8 on the test dataset.
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state =
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(X_train)
x_test= st_x.transform(X_test)

from sklearn.ensemble import RandomForestClassifier
classifier= RandomForestClassifier(n_estimators= 10, criterion="entropy")
classifier.fit(x_test, y_test)
y_pred= classifier.predict(x_train)
print("Random Forest")
print(y_pred)

#Naive Bayes
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class']
data=pd.read_csv("C:/Users/student/Downloads/archive (2)/pima-indians-diabetes.csv",names
X=data.iloc[:,:-1]
y=data.iloc[:, -1:]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state =
sc = StandardScaler()
x_train = sc.fit_transform(X_train)
x_test = sc.transform(X_test)

model = GaussianNB()
model.fit(x_test, y_test)
y_pred = model.predict(x_train)
print("Naive Bayes")
print(y_pred)


#Logistic Regression
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class']
data=pd.read_csv("C:/Users/student/Downloads/archive (2)/pima-indians-diabetes.csv",names
X=data.iloc[:,:-1]
y=data.iloc[:, -1:]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state =
sc = StandardScaler()
x_train = sc.fit_transform(X_train)
x_test = sc.transform(X_test)

classifier = LogisticRegression(random_state=0)
classifier.fit(x_test, y_test)
y_pred = classifier.predict(x_train)
print("Logistic Regression")
```

```python
print(y_pred)
```

```
Random Forest
[1 0 0 0 0 1 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 1 0 0 0 1 1 0 0 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 1 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 0 0 0 0 1
 0 0 1 1 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 0 1 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 1 0 0 0 0 1 1 1 0 0 0 0 0 1 0 0 0 0 0
 0 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 1 1 0 1 1
 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 1 0 1 0 1 1 1
 0 0 0 0 1 0 0 0 0 0 0 0 1 1 0 1 0 0 1 0 0 0 0 1 1 1 0 0 0 0 0 1 0 0 0 1 0
 0 0 0 1 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 1 0 0 1
 1 1 1 0 0 1 0 1 0 0 0 0 0 1 1 0 1 0 0 1 0 0 0 1 0 0 0 0 1 1 0 1 0 0 0 1 0
 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0 0 0 0 0
 1 1 0 0 0 1 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0 0 1
 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 1 1 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1
 0 0 0 1 1 1 1 1 1 1 0 0 0 0 1 0 1 1 1 0 0 1 0 0 0 0 0 1 1 0 1 0 0 1 1 0 0
 1 0 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 1 0 1 0 0 0 0 1 0
 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0]
Naive Bayes
[1 0 0 0 0 0 0 0 1 1 1 0 1 0 1 1 0 0 0 0 1 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 1 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 1
 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 1 0 1 0 0 0 0 1 0 1 1 0 0 0 1 0 0 0 0 0 1
 0 0 1 0 0 1 0 0 1 0 0 1 1 1 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 1 0 1 1 1 0 0 0
 0 0 0 0 1 0 0 0 1 0 1 0 1 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0
 0 0 1 0 1 1 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 1 1 0 0 0
 0 0 0 1 0 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 1 0 1
 0 0 0 0 1 0 0 0 0 0 0 0 1 1 1 0 0 1 0 1 1 0 1 1 0 1 0 0 0 0 0 1 0 0 0 1 0
 1 0 0 1 1 0 1 0 0 1 0 1 1 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 1
 0 1 1 0 0 1 0 1 0 0 0 0 0 1 1 0 1 0 0 1 0 0 0 1 0 1 0 0 1 0 0 1 0 0 0 1 0
 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0
 1 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 1 0 0 1
 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 1 0 0 1 0 1 1 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 1 0 0 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0
 0 0 1 1 1 1 0 1 1 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 1 0 1 1 1 1 1 0 0 0 1 0 1
 1 0 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 1 1 0 1 1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0
 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 1 0 0 1 0 0 0]
Logistic Regression
[1 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 1 0 0 0 1 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 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0
 0 0 0 1 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 1 1 0 0 0 1 0 0 0 0 0 1
 0 0 1 0 0 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 1 0 1 1 0 0 0 0
 0 1 0 0 1 0 0 0 1 0 1 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0
 0 0 1 0 1 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0
 0 0 0 1 0 0 0 0 0 0 1 1 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 1
 0 0 0 0 1 0 0 1 0 0 0 0 1 0 1 0 0 0 0 1 1 0 0 1 0 1 0 0 0 0 0 1 0 0 0 1 0
 0 0 0 1 1 0 1 0 0 1 0 1 0 1 1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 1 0 0 1
 0 0 1 0 0 1 0 1 0 0 0 0 0 1 1 0 1 0 0 1 0 0 0 1 0 1 0 0 1 0 0 1 0 0 0 1 0
 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0
 1 1 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 0 0 1 0 1 0 0 1
 0 1 1 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 1 0 0 0 1 1 1 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 1 1 1 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0
 0 0 0 1 1 1 0 1 1 0 0 0 0 0 1 0 1 0 1 0 0 1 0 0 1 0 0 1 1 1 1 0 0 0 0 0 1
 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0
 0 0 0 0 1 0 0 1 0 0 1 1 0 0 0 1 0 0 1 0 0 0]
```

```
C:\Users\student\AppData\Local\Temp/ipykernel_1976/3890763543.py:10: DataC
onversionWarning: A column-vector y was passed when a 1d array was expecte
d. Please change the shape of y to (n_samples,), for example using ravel
().
  classifier.fit(x_test, y_test)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was ex
pected. Please change the shape of y to (n_samples, ), for example using r
avel().
  return f(*args, **kwargs)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was ex
pected. Please change the shape of y to (n_samples, ), for example using r
avel().
  return f(*args, **kwargs)
```

In [25]:

```
#10.Evaluate  the  designed  models  from  step-5  to  step-8  with  appropriate classifi
import pandas as pd
from sklearn.model_selection import train_test_split

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class']
data=pd.read_csv("C:/Users/student/Downloads/archive (2)/pima-indians-diabetes.csv",names
X=data.iloc[:,:-1]
y=data.iloc[:, -1:]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state =

from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(X_train)
x_test= st_x.transform(X_test)

from sklearn.ensemble import RandomForestClassifier
classifier= RandomForestClassifier(n_estimators= 10, criterion="entropy")
classifier.fit(x_train, y_train)
y_pred= classifier.predict(x_test)

from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print("Random Forest Accuracy: ",accuracy)

from sklearn.metrics import precision_score
precision = precision_score(y_test, y_pred)
print("Random Forest Precision: ",precision)

from sklearn.metrics import recall_score
recall = recall_score(y_test, y_pred)
print("Random Forest Recall: ",recall)

from sklearn.metrics import f1_score
f1 = f1_score(y_test, y_pred)
print("Random Forest F1 score: ",f1)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

from sklearn.metrics import classification_report
report = classification_report(y_test, y_pred)
print("Random Forest Classification")
print(report)


#Naive Bayes
from sklearn.naive_bayes import GaussianNB

x_train = sc.fit_transform(X_train)
x_test = sc.transform(X_test)

model = GaussianNB()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)


from sklearn.metrics import accuracy_score
```

```
accuracy = accuracy_score(y_test, y_pred)
print("Naive Bayes Accuracy: ",accuracy)

from sklearn.metrics import precision_score
precision = precision_score(y_test, y_pred)
print("Naive Bayes Precision: ",precision)

from sklearn.metrics import recall_score
recall = recall_score(y_test, y_pred)
print("Naive Bayes Recall: ",recall)

from sklearn.metrics import f1_score
f1 = f1_score(y_test, y_pred)
print("Naive Bayes F1 score: ",f1)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

from sklearn.metrics import classification_report
report = classification_report(y_test, y_pred)
print("Naive Bayes Classification")
print(report)

#Logistic Regression
classifier = LogisticRegression(random_state=0)
classifier.fit(x_train, y_train)
y_pred = classifier.predict(x_test)

from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print("Logistic Regression Accuracy: ",accuracy)

from sklearn.metrics import precision_score
precision = precision_score(y_test, y_pred)
print("Logistic Regression Precision: ",precision)

from sklearn.metrics import recall_score
recall = recall_score(y_test, y_pred)
print("Logistic Regression Recall: ",recall)

from sklearn.metrics import f1_score
f1 = f1_score(y_test, y_pred)
print("Logistic Regression F1 score: ",f1)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

from sklearn.metrics import classification_report
report = classification_report(y_test, y_pred)
print("Logistic Regression Classification")
print(report)
```

```
C:\Users\student\AppData\Local\Temp/ipykernel_1976/3442923170.py:19: DataC
onversionWarning: A column-vector y was passed when a 1d array was expecte
d. Please change the shape of y to (n_samples,), for example using ravel
().
  classifier.fit(x_train, y_train)
```

```
Random Forest Accuracy:  0.8116883116883117
Random Forest Precision:  0.7631578947368421
Random Forest Recall:  0.5918367346938775
Random Forest F1 score:  0.6666666666666667
Random Forest Classification
              precision    recall  f1-score   support

           0       0.83      0.91      0.87       105
           1       0.76      0.59      0.67        49

    accuracy                           0.81       154
   macro avg       0.80      0.75      0.77       154
weighted avg       0.81      0.81      0.80       154

Naive Bayes Accuracy:  0.8051948051948052
Naive Bayes Precision:  0.7021276595744681
Naive Bayes Recall:  0.673469387755102
Naive Bayes F1 score:  0.6875000000000001
Naive Bayes Classification
              precision    recall  f1-score   support

           0       0.85      0.87      0.86       105
           1       0.70      0.67      0.69        49

    accuracy                           0.81       154
   macro avg       0.78      0.77      0.77       154
weighted avg       0.80      0.81      0.80       154

Logistic Regression Accuracy:  0.8051948051948052
Logistic Regression Precision:  0.7209302325581395
Logistic Regression Recall:  0.6326530612244898
Logistic Regression F1 score:  0.6739130434782609
Logistic Regression Classification
              precision    recall  f1-score   support

           0       0.84      0.89      0.86       105
           1       0.72      0.63      0.67        49

    accuracy                           0.81       154
   macro avg       0.78      0.76      0.77       154
weighted avg       0.80      0.81      0.80       154


C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was ex
pected. Please change the shape of y to (n_samples, ), for example using r
avel().
  return f(*args, **kwargs)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was ex
pected. Please change the shape of y to (n_samples, ), for example using r
avel().
  return f(*args, **kwargs)
```

In [ ]:

In [ ]:

```
#Name: SIVAA V
#Reg: 22MCA1127
#Lab Exercise: 11
```

In [3]:

```
#1.Load the dataset (Mall Customers.csv)

import pandas as pd
data = pd.read_csv("Mall_Customers.csv")
print(data)
```

```
     CustomerID  Gender  Age  Annual Income (k$)  Spending Score (1-100)
0             1    Male   19                  15                      39
1             2    Male   21                  15                      81
2             3  Female   20                  16                       6
3             4  Female   23                  16                      77
4             5  Female   31                  17                      40
..          ...     ...  ...                 ...                     ...
195         196  Female   35                 120                      79
196         197  Female   45                 126                      28
197         198    Male   32                 126                      74
198         199    Male   32                 137                      18
199         200    Male   30                 137                      83

[200 rows x 5 columns]
```

In [4]:

```
#2. If require perform data preprocessing.
data.isnull().sum()
```

Out[4]:

```
CustomerID                0
Gender                    0
Age                       0
Annual Income (k$)        0
Spending Score (1-100)    0
dtype: int64
```

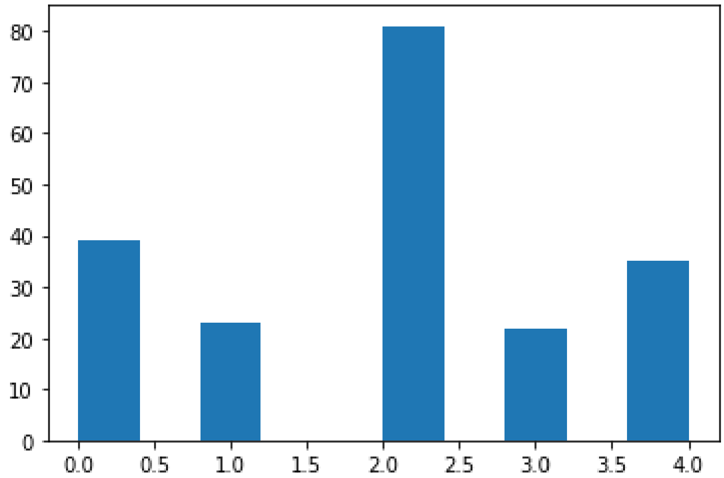
In [5]:

```python
#3.  Perform  k-means  clustering  using  sklearn  with  arbitrary  number  of clusters
from sklearn.cluster import KMeans
import numpy as np

X= data.iloc[:, [3,4]].values

n_clusters = 5

kmeans = KMeans(n_clusters=n_clusters)

kmeans.fit(X)

labels = kmeans.labels_

centers = kmeans.cluster_centers_

print("Cluster Labels:")
print(labels)

print("Cluster Centers:")
print(centers)
```

```
Cluster Labels:
[1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1
 3 1 3 1 3 1 2 1 3 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 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 0 4 0 2 0 4 0 4 0 2 0 4 0 4 0 4 0 4 0 2 0 4 0 4 0
 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4
 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0]
Cluster Centers:
[[86.53846154 82.12820513]
 [26.30434783 20.91304348]
 [55.2962963  49.51851852]
 [25.72727273 79.36363636]
 [88.2        17.11428571]]
```

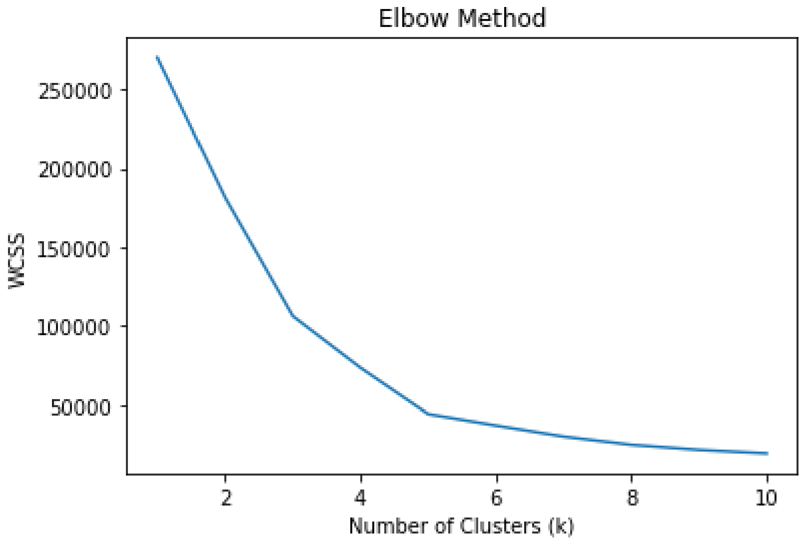
In [6]:

```
#4.Draw the inferences you find out from the clustering process.

import matplotlib.pyplot as plt
plt.hist(labels)
plt.show()
```



In [7]:

```
from sklearn.cluster import KMeans
import numpy as np
import matplotlib.pyplot as plt

X= data.iloc[:, [3,4]].values

wcss = []

max_clusters = 10

for k in range(1, max_clusters+1):
    kmeans = KMeans(n_clusters=k)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)

plt.plot(range(1, max_clusters+1), wcss)
plt.xlabel('Number of Clusters (k)')
plt.ylabel('WCSS')
plt.title('Elbow Method')
plt.show()
```



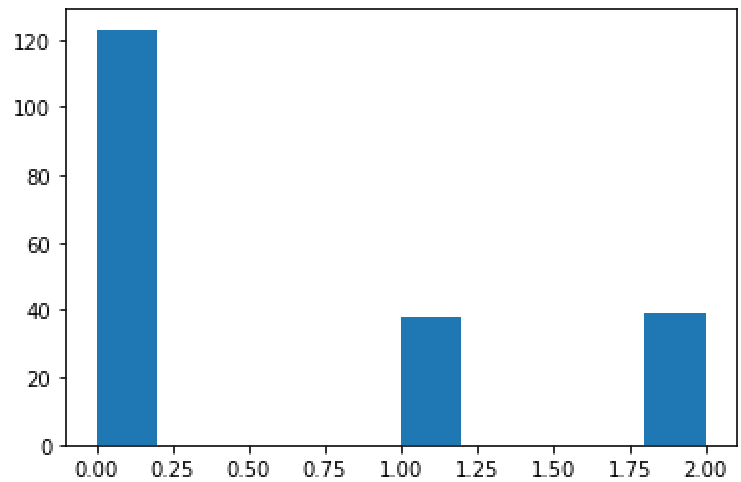
In [8]:

```
from sklearn.cluster import KMeans
import numpy as np

X= data.iloc[:, [3,4]].values

optimal_clusters = 3

kmeans = KMeans(n_clusters=optimal_clusters)

kmeans.fit(X)

labels = kmeans.labels_

centers = kmeans.cluster_centers_

print("Cluster Labels:")
print(labels)

print("Cluster Centers:")
print(centers)
```

```
Cluster Labels:
[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 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 0 0 0 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1
 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2]
Cluster Centers:
[[44.15447154 49.82926829]
 [87.         18.63157895]
 [86.53846154 82.12820513]]
```

In [9]:

```
import matplotlib.pyplot as plt
plt.hist(labels)
plt.show()
```



In [13]:

```
#8. Which attributes are strongly correlated with Spending Score?
import pandas as pd

correlation_matrix = data.corr()
spending_score_corr = correlation_matrix['Spending Score (1-100)']

spending_score_corr = spending_score_corr.sort_values(ascending=False)

print(spending_score_corr)
```

```
Spending Score (1-100)    1.000000
CustomerID                0.013835
Annual Income (k$)        0.009903
Age                      -0.327227
Name: Spending Score (1-100), dtype: float64
```

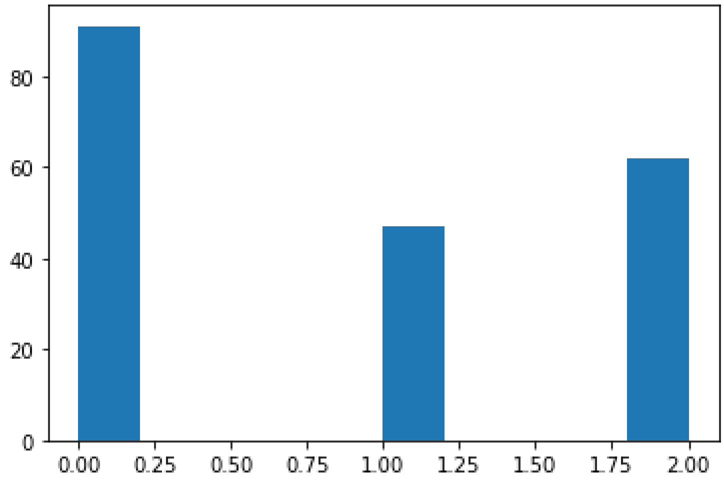
In [14]:

```
#9. Apply K-means clustering using sklearn with optimal number of clusters along with hig
from sklearn.cluster import KMeans
import pandas as pd
import numpy as np

optimal_clusters = 3

correlation_matrix = data.corr()
highly_correlated_features = correlation_matrix['Spending Score (1-100)'].abs().nlargest(

X = data[highly_correlated_features].values

kmeans = KMeans(n_clusters=optimal_clusters)
kmeans.fit(X)

labels = kmeans.labels_

centers = kmeans.cluster_centers_

print("Cluster Labels:")
print(labels)

print("Cluster Centers:")
print(centers)
```

```
Cluster Labels:
[0 2 1 2 0 2 1 2 1 2 1 2 1 2 1 2 0 2 1 2 0 2 1 2 1 2 0 0 1 2 1 2 1 2 1 2 1
 2 1 2 0 2 0 0 1 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 2 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 2 1 2 0 2 1 2 1 2 0 2 1 2 1 2 1 2 1 2 0 2 1 2 0 2
 1 2 1 2 1 2 1 2 1 2 1 2 0 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 0 2 1 2 0
 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2]
Cluster Centers:
[[47.78021978 43.05494505]
 [14.59574468 42.95744681]
 [80.74193548 29.56451613]]
```

In [15]:

```
#10. Draw the inferences you find out from the clustering process
import matplotlib.pyplot as plt
plt.hist(labels)
plt.show()
```



In [ ]: