Department of Information and Communication Engineering

Pabna University of Science and Technology

B.Sc. (Engineering) 3rd Year 1st Semester Examination -2021

Session: 2018-2019

Course Code: ICE-3102

Course Title: Artificial Intelligence and Robotics Sessional

Lab Report

Submitted by	Submitted to
Md. Sha Alam **Roll No:** 190610 **Registration No:** 1065335 Department of Information and Communication Engineering Pabna University of Science and Technology Pabna-6600, Bangladesh Email: shaalam.ice@pust.ac.bd	**Tarun Debnath** **Lecturer,** Department of Information and Communication Engineering Pabna University of Science and Technology Pabna-6600, Bangladesh

INDEX

Experiment Number : 01

Experiment Name : Write a program to implement Tower of Hanoi for n disk.

Objectives:

1. To learn how to do python code.
2. To learn about tower of hanoi for n disk problem and solution.

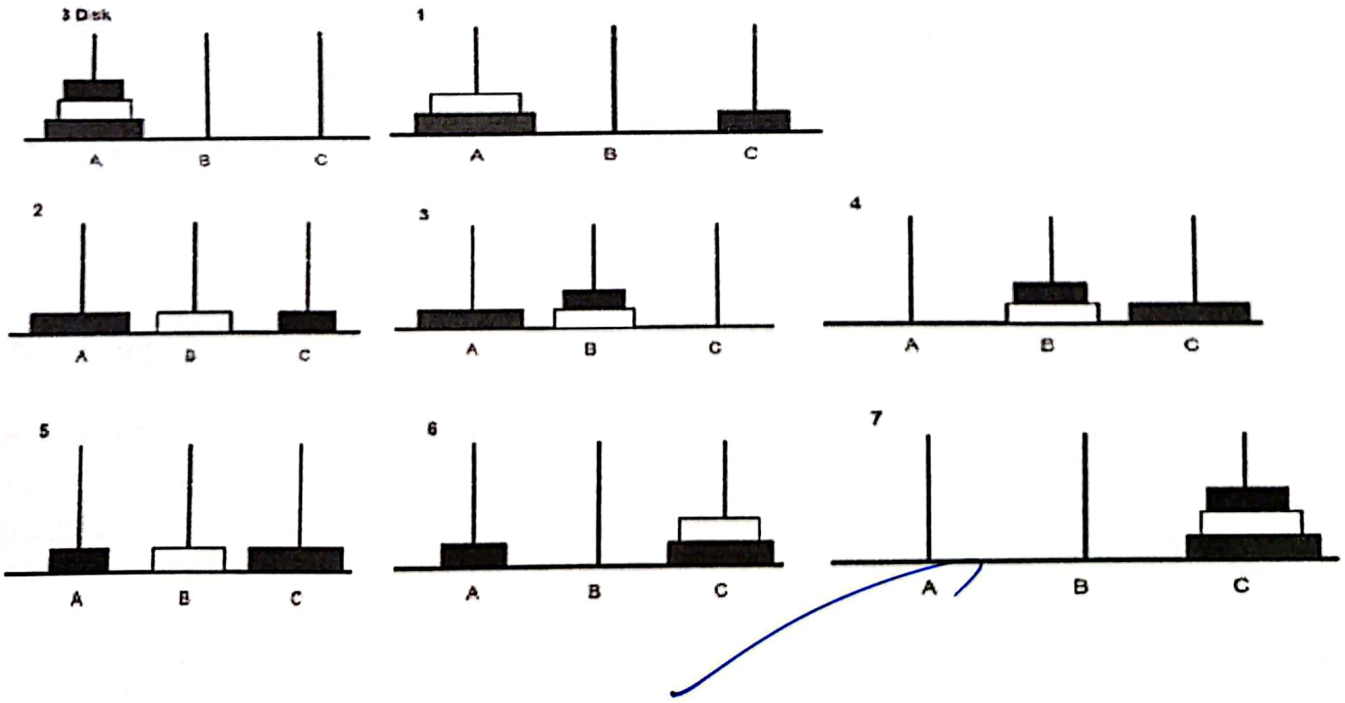
Theory:

Tower of Hanoi is a mathematical puzzle where we have three rods (A, B, and C) and N disks. Initially, all the disks are stacked in decreasing value of diameter i.e., the smallest disk is placed on the top and they are on rod A. The objective of the puzzle is to move the entire stack to another rod (here considered C), obeying the following simple rules

Only one disk can be moved at a time.

Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.

No disk may be placed on top of a smaller disk.



Source Code:

```
def tower_of_hanoi(n, from_rod, to_rod, aux_rod):
    if n == 1:
        print("Move disk 1 from rod", from_rod, "to rod", to_rod)
        return
    tower_of_hanoi(n - 1, from_rod, aux_rod, to_rod)
    print("Move disk", n, "from rod", from_rod, "to rod", to_rod)
    tower_of_hanoi(n - 1, aux_rod, to_rod, from_rod)


n = int(input("Enter number of disks: "))
tower_of_hanoi(n, 'A', 'C', 'B')
```

Output:

```
Enter number of disks: 3
Move disk 1 from rod A to rod C
Move disk 2 from rod A to rod B
Move disk 1 from rod C to rod B
Move disk 3 from rod A to rod C
Move disk 1 from rod B to rod A
Move disk 2 from rod B to rod C
Move disk 1 from rod A to rod C
```

Experiment Number : 02

Experiment Name : Write a Program to Implement Breadth First Search algorithm.

Objectives:

1. To learn how to do python code.
2. Learn how to implement Breadth First Search algorithm.

Theory:

Breadth-First Traversal (or Search) for a graph is similar to the Breadth-First Traversal of a tree (See method 2 of this post).

The only catch here is, that, unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we divide the vertices into two categories:

1. Visited and
2. Not visited.

A boolean visited array is used to mark the visited vertices. For simplicity, it is assumed that all vertices are reachable from the starting vertex. BFS uses a queue data structure for traversal.

- Follow the below method to implement BFS traversal.
- Declare a queue and insert the starting vertex.
- Initialize a visited array and mark the starting vertex as visited.
- Follow the below process till the queue becomes empty:
- Remove the first vertex of the queue.
- Mark that vertex as visited.
- Insert all the unvisited neighbors of the vertex into the queue.

Source Code:

```
from queue import Queue

def bfs(graph, start):
    visited = set()
    queue = Queue()
    queue.put(start)
    while not queue.empty():
        vertex = queue.get()
        if vertex not in visited:
            visited.add(vertex)
            print(vertex)
            for neighbor in graph[vertex]:
                if neighbor not in visited:
                    queue.put(neighbor)
# Example usage:
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}
bfs(graph, 'A')
```

Output:

```
A
B
C
D
E
F
```

Experiment Number : 03

Experiment Name : Write a Program to Implement Depth First Search algorithm.

Objectives:

1. To learn about simple python code.
2. Learn how to implement Depth First Search algorithm.

Theory:

Depth-first search is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.

So the basic idea is to start from the root or any arbitrary node and mark the node and move to the adjacent unmarked node and continue this loop until there is no unmarked adjacent node. Then backtrack and check for other unmarked nodes and traverse them. Finally, print the nodes in the path.

Follow the below steps to solve the problem:

- Create a recursive function that takes the index of the node and a visited array.
- Mark the current node as visited and print the node.
- Traverse all the adjacent and unmarked nodes and call the recursive function with the index of the adjacent node.

Source Code:

```
def DFS(graph, start):
    visited = set()      # Create a set to store visited nodes
    stack = [start]      # Create a stack for DFS traversal
    while stack:         # Loop until stack is empty
        node = stack.pop()      # Pop the last item from the stack
        if node not in visited:
            visited.add(node)  # Add the visited node to the visited set
            stack.extend(graph[node] - visited)  # Add adjacent nodes to the stack that have not been visited
    return visited       # Return the set of visited nodes
# Test the DFS function
graph = {'A': set(['B', 'C']),
         'B': set(['A', 'D', 'E']),
         'C': set(['A', 'F']),
         'D': set(['B']),
         'E': set(['B', 'F']),
         'F': set(['C', 'E'])}

print("DFS Traversal:")
print(DFS(graph, 'A')) # Output: {'E', 'D', 'F', 'A', 'C', 'B'}
```

Output:

```
DFS Traversal:
{'E', 'B', 'A', 'D', 'C', 'F'}
```

Experiment Number : 04

Experiment Name : Write a Program to Implement Travelling Salesman Problem.

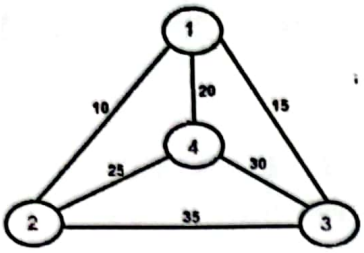
Objectives:

1. To learn about travelling salesman algorithm

Theory:

Travelling Salesman Problem (TSP):

Given a set of cities and the distance between every pair of cities, the problem is to find the shortest possible route that visits every city exactly once and returns to the starting point. Note the difference between Hamiltonian Cycle and TSP. The Hamiltonian cycle problem is to find if there exists a tour that visits every city exactly once. Here we know that Hamiltonian Tour exists (because the graph is complete) and in fact, many such tours exist, the problem is to find a minimum weight Hamiltonian Cycle.



For example, consider the graph shown in the figure on the right side. A TSP tour in the graph is 1-2-4-3-1. The cost of the tour is 10+25+30+15 which is 80. The problem is a famous NP-hard problem. There is no polynomial-time know solution for this problem. The following are different solutions for the traveling salesman problem.

Naive Solution:

- Consider city 1 as the starting and ending point.
- Generate all (n-1)! Permutations of cities.
- Calculate the cost of every permutation and keep track of the minimum cost permutation.
- Return the permutation with minimum cost.

Time Complexity: Θ(n!)

Source Code:

```
n = 4
dist = [[0, 0, 0, 0, 0], [0, 0, 10, 15, 20], [
    0, 10, 0, 25, 25], [0, 15, 25, 0, 30], [0, 20, 25, 30, 0]]

memo = [[-1] * (1 << (n + 1)) for _ in range(n + 1)]
def fun(i, mask):
    if mask == ((1 << 1) | 3):
        return dist[1][i]
    if memo[i][mask] != -1:
        return memo[i][mask]

    res = 10 ** 9  # result of this sub-problem
    for j in range(1, n + 1):
        if (mask & (1 << j)) != 0 and j != i and j != 1:
            res = min(res, fun(j, mask & (~(1 << i))) + dist[j][i])
    memo[i][mask] = res  # storing the minimum value
    return res
ans = 10 ** 9
for i in range(1, n + 1):
    ans = min(ans, fun(i, (1 << (n + 1)) - 1) + dist[i][1])
print("The cost of most efficient tour = " + str(ans))
```

Output:

```
The cost of most efficient tour = 80
```

Experiment Number : 05

Experiment Name :

I. Create and load different datasets in python.
II. Write a python program to compute Mean, Median, Mode, Variance and Standard Deviation using datasets.

Objectives:

1. Learn how to create and load datasets in python
2. Learn how to compute Mean, Median, Mode, Variance and Standard Deviation using datasets

Theory:

Mean: The mean is the average value of a dataset. It is calculated by adding up all the values in the dataset and dividing by the total number of values. For example, the mean of the numbers 3, 5, 7, and 9 is (3+5+7+9)/4 = 6.

Median: The median is the middle value in a dataset when the values are arranged in order. If there is an odd number of values, the median is the middle value. If there is an even number of values, the median is the average of the two middle values. For example, the median of the numbers 3, 5, 7, and 9 is 6.

Mode: The mode is the value that appears most frequently in a dataset. For example, the mode of the numbers 3, 5, 7, 7, and 9 is 7.

Variance: Variance is a measure of how spread out the data is. It is calculated by taking the average of the squared differences between each value and the mean. A high variance means that the data is spread out over a large range of values, while a low variance means that the data is clustered around the mean.

Standard Deviation: The standard deviation is the square root of the variance. It measures how spread out the data is from the mean in the same units as the data. A high standard deviation means that the data is spread out over a large range of values, while a low standard deviation means that the data is clustered around the mean.

Source Code:

```
import pandas as pd

# Creating and loading different datasets
data = {'Name': ['Jai', 'Princi', 'Gaurav', 'Anju', 'Ravi', 'Natasha', 'Riya'],
        'Age': [17, 17, 18, 17, 18, 17, 17],
        'Gender': ['M', 'F', 'M', 'M', 'M', 'F', 'F'],
        'Marks': [90, 76, 'NaN', 74, 65, 'NaN', 71]}
a = pd.DataFrame(data)
print(a, "\n")

# loading database from csv type dataset
b = pd.read_csv('1.car driving risk analysis.csv')
print("Read Dataset from csv file\n")
print(b)
```

```
import numpy as np
import statistics as stats
import pandas as pd

data = pd.read_csv('dd.csv')
print('Dataset:\n', data)
# Compute the mean
mean = np.mean(data['num'])
print('Mean:', mean)
# # Compute the median
median = np.median(data['num'])
print('Median:', median)
# # Compute the mode
mode = stats.mode(data['gender'])
# mode_val = mode.mode[0]
print('Mode:', mode)
# # Compute the variance
variance = np.var(data['num'])
print('Variance:', variance)
# # Compute the standard deviation
std_dev = np.std(data['num'])
std_dev = round(std_dev, 2);
print('Standard Deviation:', std_dev)
```

Output:

```
      Name  Age Gender Marks
0      Jai   17      M    90
1   Princi   17      F    76
2   Gaurav   18      M   NaN
3     Anju   17      M    74
4     Ravi   18      M    65
5  Natasha   17      F   NaN
6     Riya   17      F    71
```

```
Read Dataset from csv file

    speed  risk
0     200    95
1      90    20
2     300    98
3     110    60
4     240    72
5     115    10
6      50     7
7     230    85
8     190    45
9     260    91
10    290    82
11    185    59
12    310    93
13     95    18
14     30     2


    num  weight  gender
0     1     112    Male
1     2     190    Male
2     3     145  Female
3     4     205    Male
4     5     105  Female
Mean: 3.0
Median: 3.0
Mode: Male
Variance: 2.0
Standard Deviation: 1.41
```

Experiment Number : 06

Experiment Name : Write a python program to implement simple linear regression and plot the graph.

Objectives:

1. To know the relationship between variables
2. Predict future outcomes
3. Identify significant predictors

Theory:

Linear regression is a statistical method used to study the relationship between a dependent variable (also known as the response or outcome variable) and one or more independent variables (also known as predictor or explanatory variables). It is called "linear" regression because it assumes that the relationship between the variables can be represented by a straight line.The goal of linear regression is to find the best fitting line through the data, which can be used to make predictions or understand the relationship between the variables. The line is defined by an equation of the form:

$$y = a + bx$$

where y is the dependent variable, x is the independent variable, a is the y-intercept (the value of y when x=0), and b is the slope (the change in y for a one-unit change in x).The basic idea behind linear regression is to estimate the values of a and b that minimize the difference between the predicted values of y (based on the equation above) and the actual values of y in the data set. This is typically done using a method called least squares regression, which finds the values of a and b that minimize the sum of the squared differences between the predicted and actual values of y.

Source Code:

Implement simple linear Regression and plot the graph

In [26]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [27]:
```
df=pd.read_csv('1.car driving risk analysis.csv')
df
```

Out[27]:

	speed	risk
0	200	69
1	60	20
2	300	68
3	110	60
4	240	72
5	115	10
6	50	7
7	230	86
8	190	46
9	260	91
10	290	82
11	185	59
12	310	93
13	95	18
14	30	2

In [42]:
```
x=df[['speed']]  #only 3rd bracket,,two dimension for input means indepenaent variable
y=df['risk']     #only 3rd bracket,,one dimension for outpu means dependent variable
```

```
In [46]: from sklearn.model_selection import train_test_split
```

Risk should be analysed based on speed

#60% For training and 40% for test

```
In [49]: xtrain,xtest,ytrain,ytest = train_test_split(x,y,test_size=.40,random_state=1)
```

```
xtrain
```

```
In [23]: xtest
```

Out[23]:

	speed
3	110
7	230
6	60
2	300
10	290
4	240

```
In [24]: ytrain
```

```
Out[24]: 1     20
         13    18
         0     95
         14     2
         9     91
         8     45
         12    93
         11    59
         5     10
         Name: risk, dtype: int64
```

```
In [25]: ytest
```

```
In [35]: from sklearn.linear_model import LinearRegression
```

```
In [38]: reg=LinearRegression()
```

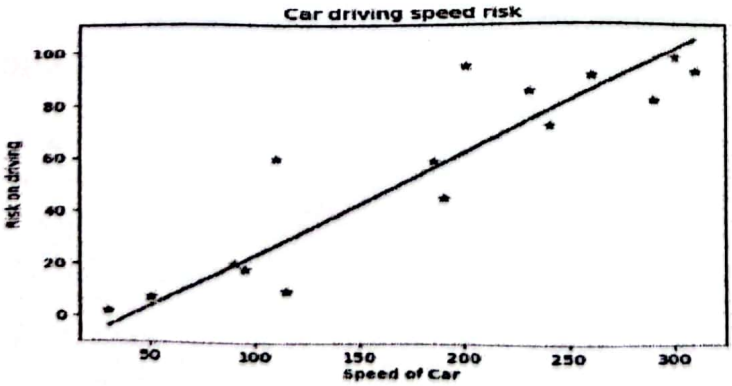
```
In [39]: reg.fit(xtrain,ytrain)
Out[39]: LinearRegression()
```

```
In [40]: reg.predict(xtest)  #compare with ytest
Out[40]: array([ 27.15301215,  73.82259334,   3.81822156, 101.04651569,
                 97.15738393,  77.7117251 ])
```

```
In [41]: ytest
```

```
Out[41]: 3     60
         7     85
         6      7
         2     98
         10    82
         4     72
         Name: risk, dtype: int64
```

```
In [54]: plt.scatter(df['speed'],df['risk'],marker='*',color='red')
         plt.xlabel('Speed of Car')
         plt.ylabel('Risk on driving')
         plt.title('Car driving speed risk')
         plt.plot(df.speed,reg.predict(df[['speed']]))
Out[54]: [<matplotlib.lines.Line2D at 0x2036c4adfd0>]
```




```
In [55]: reg.predict([[120]])
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names
  warnings.warn(
Out[55]: array([54.17693451])
```

```
In [56]: reg.coef_
Out[56]: array([0.38891318])
```

```
In [57]: reg.intercept_
Out[57]: -15.627437265817058
```

Experiment Number : 07

Experiment Name : Write a python program to implement Find S algorithm.

Objectives:

1. To know about find-s algorithm
2. To know how we use find-s algorithm to train a machine

Theory:

The Find-S algorithm is a basic and widely used algorithm in machine learning and artificial intelligence, specifically for supervised learning problems that involve classifying objects into discrete categories. The algorithm is used to find the most specific hypothesis that fits all the positive examples in the training data. Here's how it works:

➢ Start with the most specific hypothesis possible: This is typically represented by the empty set, which means that no attribute is common to all positive examples.

➢ For each positive example in the training data, update the hypothesis by adding any attribute that is consistent with that example. This means that if an attribute is present in the example, it should also be included in the hypothesis.

➢ After updating the hypothesis for all positive examples, return the final hypothesis.

Source Code:

```
import pandas as pd
import numpy as np
data=pd.read_csv('data.csv')
print(data)

# leave the last column
concept=np.array(data)[:,:-1]
# only access the last column
target=np.array(data)[:,-1]

def train(concept,target):
    for i,value in enumerate(target):
        if value.lower()=='yes':
            specific_h=concept[i].copy()
            break
    for i,value in enumerate(concept):
        if target[i].lower()=='yes':
            for x in range(len(specific_h)):
                if value[x]!=specific_h[x]:
                    specific_h[x]='?'
                else:
                    pass;
    return specific_h

result=train(concept,target)
print(result)
```

```
day=input("Enter 6 word to check:")
day=day.split()
check=True

for i in range(len(result)):
    if result[i]=='?'or result[i]==day[i]:
        check=True;
    else:
        check=False;
        break;
if check:
    print("Enjoy sport")
else:
    print("Not Possible")
```

Output:

```
      sky air temp humidity    wind water forecast enjoy sport
0  sunny     warm   normal  strong  warm     same         yes
1  sunny     warm     high  strong  warm     same         yes
2  rainy     cold     high  strong  warm   change          no
3  sunny     warm     high  strong  cool   change         yes
['sunny' 'warm' '?' 'strong' '?' '?']
Enter 6 word to check:sunny warm high strong cool change
Enjoy sport
```

Experiment Number : 08

Experiment Name : Write a python Program to implement Support Vector Machine (SVM) Algorithm

Objectives:

1. To know about SVM algorithm
2. To know Multi class classification
3. To Know how to implement the SVM algorithm in python

Theory:

Support Vector Machines (SVM) is a powerful supervised learning algorithm used for classification and regression analysis. The main objective of SVM is to find a hyperplane in a highdimensional space that maximally separates the different classes.In binary classification problems, SVM tries to find a decision boundary that separates the two classes with the largest possible margin. The margin is defined as the distance between the decision boundary and the closest points from each class. The points that define the margin are called support vectors, hence the name "support vector machine".

SVM is a powerful algorithm because it can handle non-linear decision boundaries using a technique called kernel trick. Kernel trick allows SVM to implicitly map the input data into a highdimensional space, where it is more likely that a linear decision boundary can separate the classes.

There are two main types of SVM: linear SVM and non-linear SVM. Linear SVM is used when the data can be separated by a linear decision boundary, while non-linear SVM is used when the data cannot be separated by a linear decision boundary. In non-linear SVM, a kernel function is used to transform the input data into a higher-dimensional space, where it is more likely that a linear decision boundary can separate the classes.

SVM has several advantages over other classification algorithms. Some of these advantages include:

1. SVM can handle high-dimensional data with ease.

2. SVM is effective in cases where the number of features is much greater than the number of samples.

3. SVM can handle non-linear decision boundaries.

4. SVM is less prone to overfitting than other classification algorithms.

Source Code and Output:

```
In [30]: import numpy as np
         import pandas as pd
         import matplotlib.pyplot as plt
         import seaborn as sb
```

```
In [31]: df=pd.read_csv('Social_Network_Ads.csv')
         df
```

Out[31]:

	Age	EstimatedSalary	Purchased
0	19	19000	0
1	35	20000	0
2	26	43000	0
3	27	57000	0
4	19	76000	0
...	...	...	...
395	46	41000	1
396	51	23000	1
397	50	20000	1
398	36	33000	0
399	49	36000	1

400 rows × 3 columns

```
In [32]: df.shape
Out[32]: (400, 3)
```

In [33]:
```
x=df.iloc[:,[0,1]]
x
```

Out[33]:

	Age	EstimatedSalary
0	19	19000
1	35	20000
2	26	43000
3	27	57000
4	19	76000
...	...	...
395	46	41000
396	51	23000
397	50	20000
398	36	33000
399	49	36000

400 rows × 2 columns

In [34]:
```
y=df.iloc[:,2]
y
```

Out[34]:
```
0      0
1      0
2      0
3      0
4      0
      ..
395    1
396    1
397    1
398    0
399    1
Name: Purchased, Length: 400, dtype: int64
```

In [35]:
```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
```

In [36]:
```
xtrain,xtest,ytrain,ytest = train_test_split(x,y,test_size=.40,random_state=1)
```

In [37]:
```
print('Training data : ',xtrain.shape)
```

```
Training data :  (240, 2)
```

In [38]:
```
print('Testing data : ',xtest.shape)
```

```
Testing data :  (160, 2)
```

In [40]:
```
xtrain
```

Out[40]:

	Age	EstimatedSalary
152	35	38000
247	57	122000
378	41	87000
145	24	89000
251	37	52000
...	...	...
255	52	90000
72	20	23000
396	51	23000
235	46	79000
37	30	49000

240 rows × 2 columns

In [41]:
```
xtest
```

Out[41]:

	Age	EstimatedSalary
398	36	33000
125	39	81000
328	36	118000
339	39	122000
172	28	118000
...	...	...
340	53	104000
70	25	80000
38	26	72000
150	26	15000
208	40	142000

160 rows × 2 columns

In [42]:
```
ytrain
```

Out[42]:
```
163    0
247    1
378    1
145    0
251    0
      ..
255    1
72     0
396    1
235    1
37     0
Name: Purchased, Length: 240, dtype: int64
```

In [43]:
```
model=SVC(gamma='auto')
```

In [ ]:

In [44]:
```
model.fit(xtrain,ytrain)
```

In [44]:
```
model.fit(xtrain,ytrain)
```

Out[44]:
```
SVC(gamma='auto')
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [45]:
```
model.score(xtest,ytest)
```

Out[45]:
```
0.66875
```