Experiment Number: 01

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

Objectives:

1. To learn about recursion, as the Tower of Hanoi puzzle is typically solved using a recursive algorithm.
2. To learn about tower of Hanoi for n disk problem and solution.

Theory:

The Tower of Hanoi is a classic mathematical puzzle that consists of three rods and a number of disks of different sizes, which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape.

The objective of the puzzle is to move the entire stack to another rod, 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 or on an empty rod.
- No disk may be placed on top of a smaller disk.

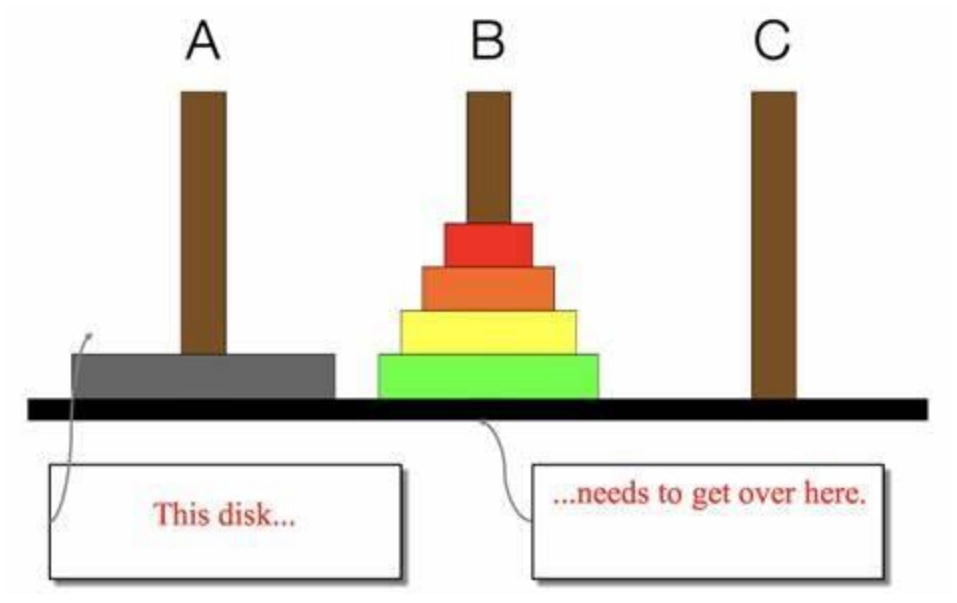


FIG:-1: Tower of Hanoi

Source Code:

```
#include <bits/stdc++.h>
using namespace std;

void towerOfHanoi(int n, char source, char auxilary, char destination)
{
    if (n == 1)
        cout << "move disk 1 from " << source << " to " << destination << endl;
    else
    {
        towerOfHanoi(n - 1, source, destination, auxilary);
        cout << "move disk " << n << " from " << source << " to " << destination << endl;
        towerOfHanoi(n - 1, auxilary, source, destination);
    }
}
int main()
{
    int n;
    cin >> n;
    towerOfHanoi(n, 'A', 'B', 'C');
    return 0;
}
```

Output:

```
3
move disk 1 from A to C
move disk 2 from A to B
move disk 1 from C to B
move disk 3 from A to C
move disk 1 from B to A
move disk 2 from B to C
move disk 1 from A to C
```

Experiment Number : 02

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

Objectives:

1. Visit all the vertices in the graph
2. Determine the shortest path between two vertices
3. Find all the connected components in a graph
4. Learn how to implement Breadth First Search algorithm.

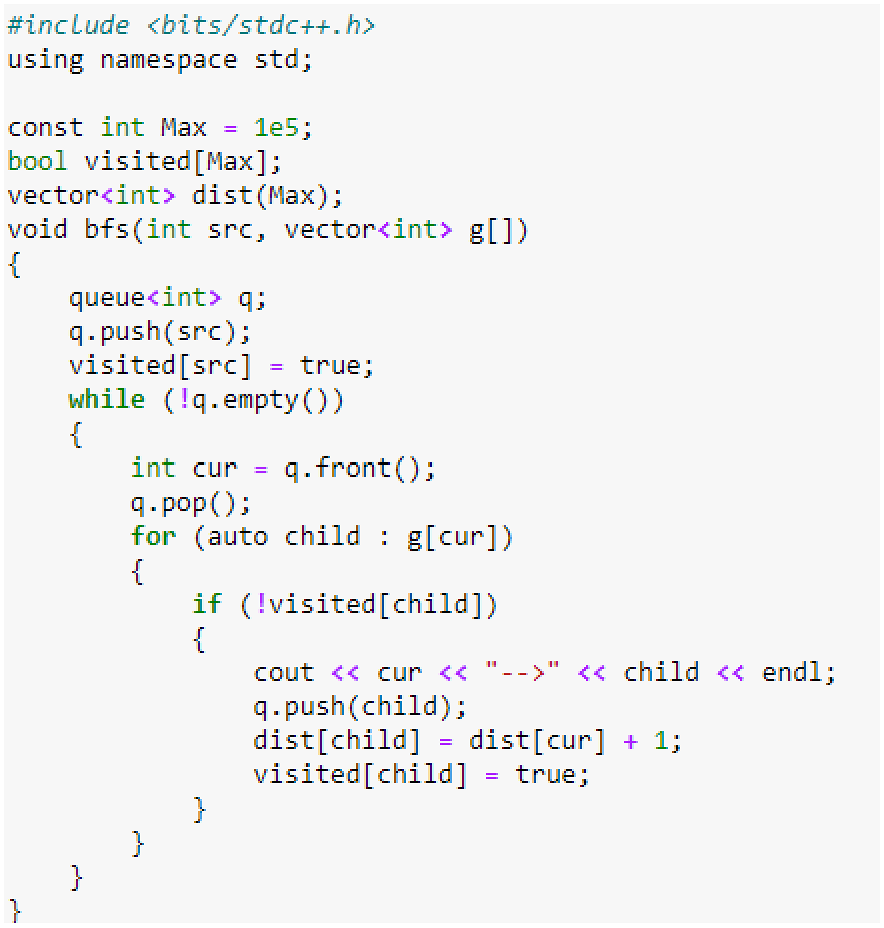
Theory:

Breadth First Search (BFS) is a graph traversal algorithm that visits all the vertices of a graph in breadth-first order. The algorithm starts at a particular vertex (or node) and explores all the vertices that are at the same level before moving on to the next level of vertices. BFS can be used to determine the shortest path between two vertices, check for connectivity, find all the connected components in a graph, and perform topological sorting on a directed acyclic graph.

Here is a step-by-step explanation of the BFS algorithm:

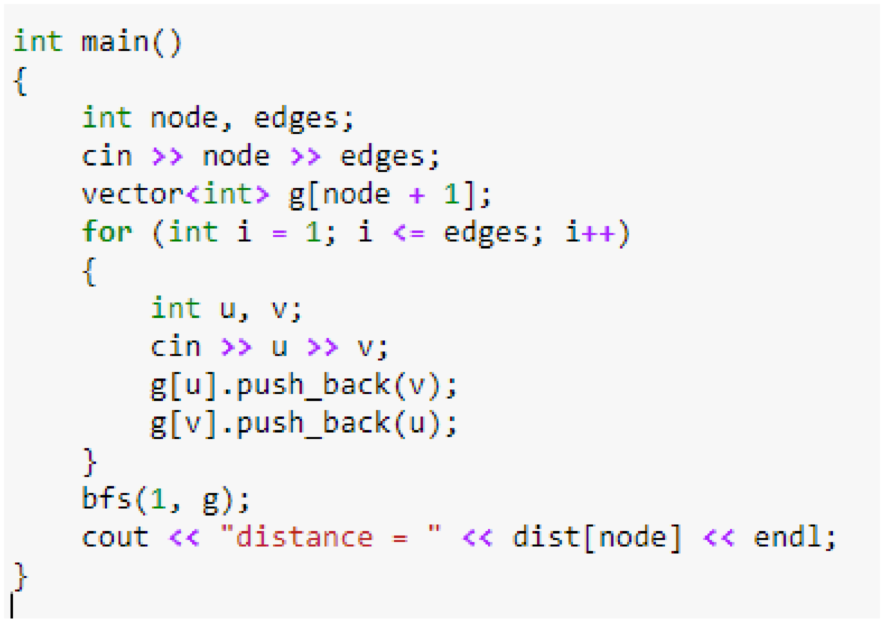
1. Choose a starting vertex
2. Visit adjacent vertices
3. Move to the next level
4. Repeat until all vertices have been visited
5. Steps 2 and 3 are repeated until all the vertices in the graph have been visited
6. Output the order of visited vertices

Source Code:



```
#include <bits/stdc++.h>
using namespace std;

const int Max = 1e5;
bool visited[Max];
vector<int> dist(Max);
void bfs(int src, vector<int> g[])
{
    queue<int> q;
    q.push(src);
    visited[src] = true;
    while (!q.empty())
    {
        int cur = q.front();
        q.pop();
        for (auto child : g[cur])
        {
            if (!visited[child])
            {
                cout << cur << "-->" << child << endl;
                q.push(child);
                dist[child] = dist[cur] + 1;
                visited[child] = true;
            }
        }
    }
}
```



```
int main()
{
    int node, edges;
    cin >> node >> edges;
    vector<int> g[node + 1];
    for (int i = 1; i <= edges; i++)
    {
        int u, v;
        cin >> u >> v;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    bfs(1, g);
    cout << "distance = " << dist[node] << endl;
}
```

Output: `1-->2 1-->3 3-->4 distance = 2`

Experiment Number : 03

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

Objectives:

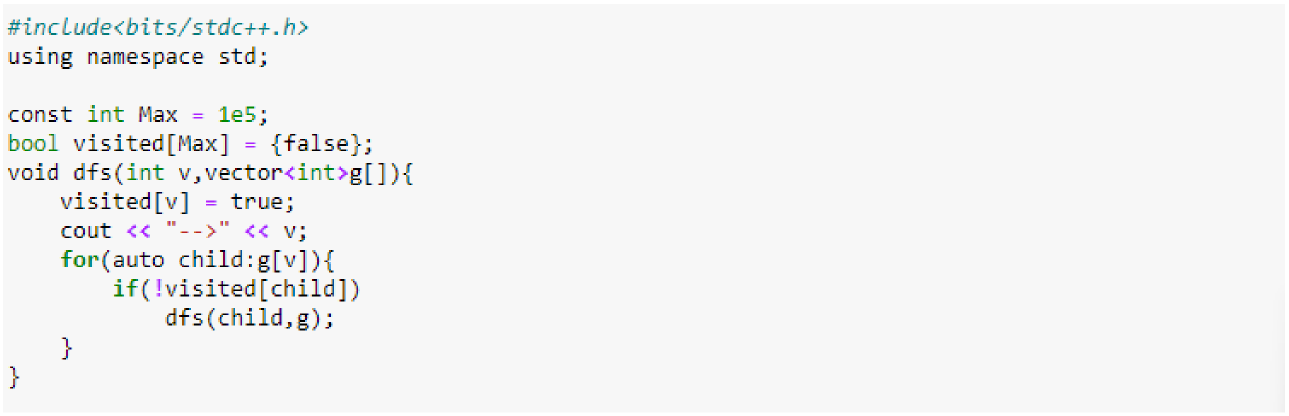
1. Visit all the vertices in the graph
2. Determine the shortest path between two vertices
3. Find all the connected components in a graph
4. Learn how to implement Depth First Search algorithm.

Theory:

Depth First Search (DFS) is a graph traversal algorithm that explores the graph by visiting its vertices and edges in a depth-first manner. It starts at a source vertex and visits all the vertices of the graph that are reachable from the source vertex. Here are the main steps of DFS:

1. Mark the source vertex as visited.
2. Visit the source vertex and process it.
3. Recursively visit all unvisited neighbors of the source vertex.
4. Backtrack to the previous vertex and visit its unvisited neighbors.
5. Repeat steps 3 and 4 until all vertices of the graph are visited.

Source Code:



```
#include<bits/stdc++.h>
using namespace std;

const int Max = 1e5;
bool visited[Max] = {false};
void dfs(int v,vector<int>g[]){
    visited[v] = true;
    cout << "-->" << v;
    for(auto child:g[v]){
        if(!visited[child])
            dfs(child,g);
    }
}
```

```cpp
int main(){
    int node, edges;
    cin >> node >> edges;
    vector<int> g[node];
    for (int i = 0; i < edges;i++){
        int u, v;
        cin >> u >> v;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    cout << "start";
    dfs(0,g);
}
```

Output:

```
4 4

0 1

0 2

1 2

2 3

start-->0-->1-->2-->3
```

Experiment Number : 04

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

Objectives:

- Minimize the distance traveled
- Optimize resource utilization
- Improve delivery or service times
- Minimize transportation costs
- Optimize routing in logistics

Theory:

Travelling Salesman Problem (TSP):

The Traveling Salesman Problem (TSP) is a classic optimization problem in computer science, operations research, and mathematics. The problem can be stated as follows: given a set of cities and the distances between them, find the shortest possible route that visits each city exactly once and returns to the starting city.

The TSP is an NP-hard problem, which means that it is not known how to solve the problem in polynomial time. However, there are several algorithms and heuristics that can be used to approximate the optimal solution or find a suboptimal solution that is close to the optimal one.

One of the most popular and effective algorithms for solving the TSP is dynamic programming. The dynamic programming approach breaks down the problem into smaller subproblems and solves them in a bottom-up fashion, using memorization to avoid redundant calculations. The time complexity of this approach is O(2^n * n^2), where n is the number of cities.

Source Code:

```
#include <bits/stdc++.h>
using namespace std;

const int N = 15; // maximum number of cities
const int INF = 1e9; // infinity
int n; // number of cities
int d[N][N]; // distance matrix
int dp[1 << N][N]; // dp table
bool vis[1 << N][N]; // visited table

int tsp(int S, int v) {
    if (S == (1 << n) - 1) { // all cities visited
        return d[v][0]; // return distance from last city to starting city
    }
    if (vis[S][v]) { // already visited this state
        return dp[S][v]; // return memoized value
    }
    vis[S][v] = true; // mark state as visited
    int ans = INF; // initialize answer to infinity
    for (int i = 0; i < n; i++) {
        if (!(S & (1 << i))) { // if city i has not been visited
            ans = min(ans, d[v][i] + tsp(S | (1 << i), i)); // update answer
        }
    }
    dp[S][v] = ans; // memoize answer
    return ans;
}

int main() {
    cin >> n; // input number of cities
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cin >> d[i][j]; // input distance matrix
        }
    }
    memset(vis, false, sizeof(vis)); // initialize visited table to false
    int ans = tsp(1, 0); // start from city 0 and visit all other cities
    cout << "Shortest distance: " << ans << endl; // output answer
    return 0;
}
```

Output:

```
4

0 10 15 20

10 0 35 25

15 35 0 30

20 25 30 0

Shortest distance: 80
```

Experiment Number : 05

Experiment Name :

1. Create and load different datasets in python.
2. 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.

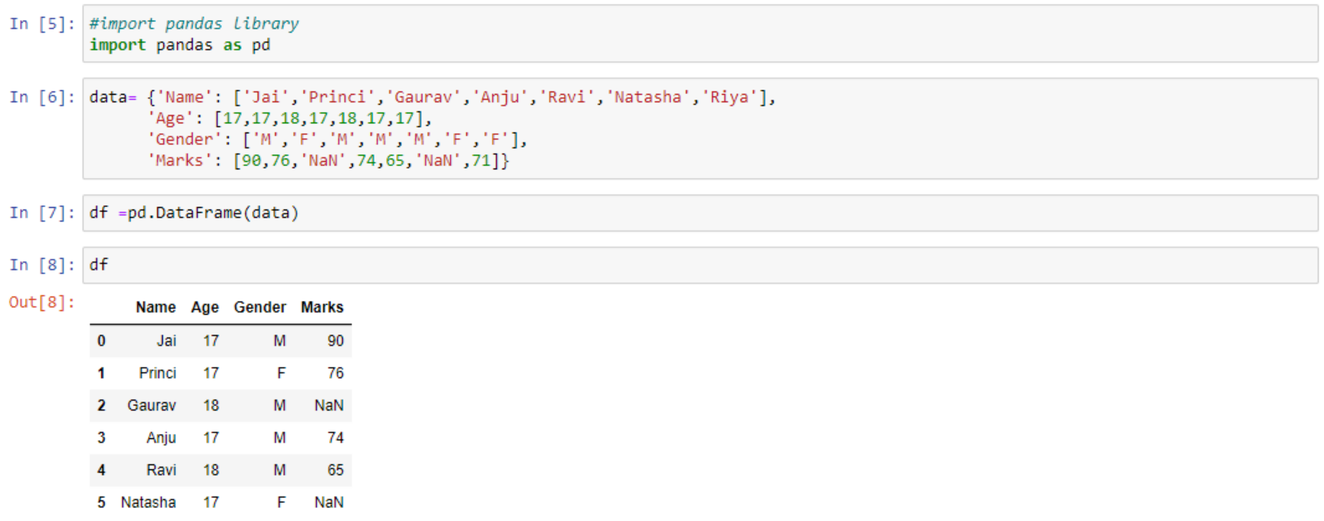
Mode: The mode is the value that appears most frequently in a dataset. For example, the mode of the numbers 3, 5, 7, 7, and 9 is 7.

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:

For Number 1



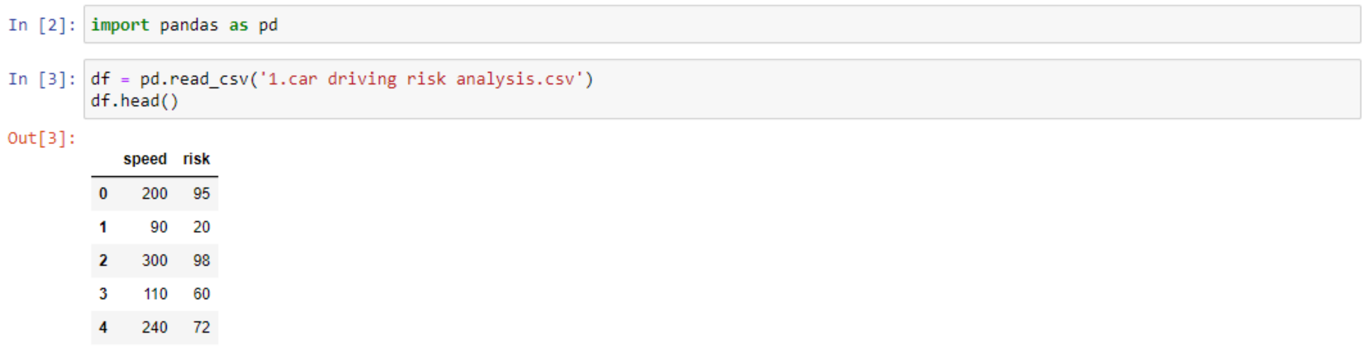
```
In [5]: #import pandas library
        import pandas as pd
```

```
In [6]: 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]}
```

```
In [7]: df =pd.DataFrame(data)
```

```
In [8]: df
```

```
Out[8]:
      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
```



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

```
In [3]: df = pd.read_csv('1.car driving risk analysis.csv')
        df.head()
```

```
Out[3]:
   speed  risk
0    200    95
1     90    20
2    300    98
3    110    60
4    240    72
```

For Number 2

In [1]:
```python
import numpy as np
from scipy import stats
import pandas as pd

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

Output:

```
Dataset:    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: 1
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.

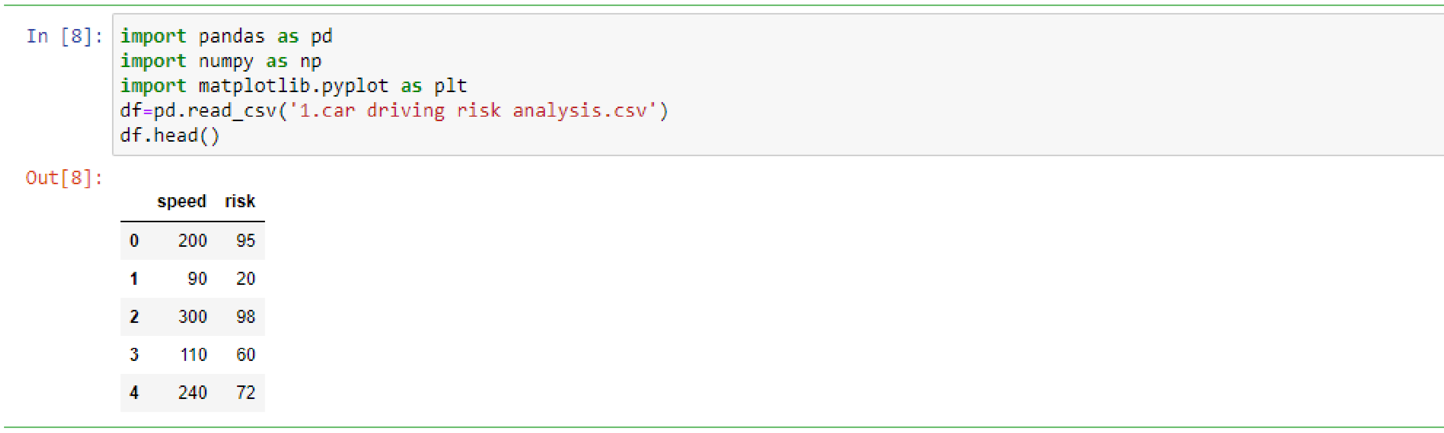
The goal of linear regression is to find the best fitting line through the data, which can be used to make predictions or understand the relationship between the variables. The line is defined by an equation of the form:

$$y = a + bx$$

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.

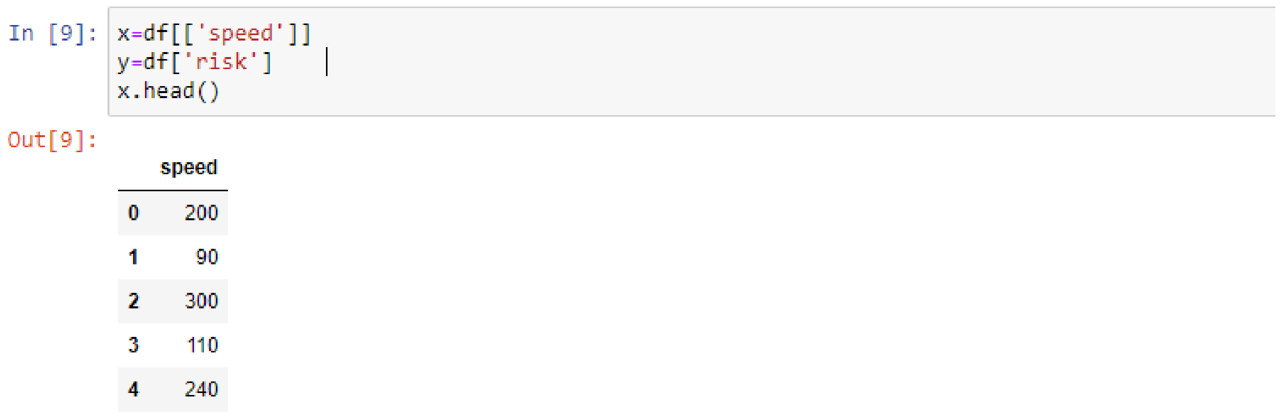
Source Code:



```
In [8]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        df=pd.read_csv('1.car driving risk analysis.csv')
        df.head()
```

Out[8]:

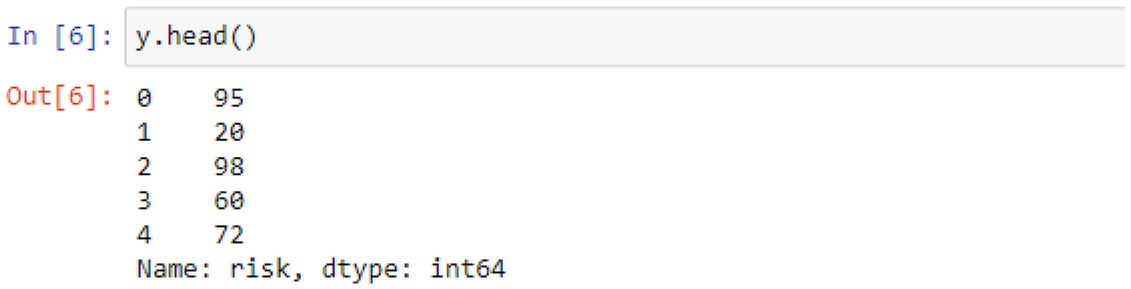
	speed	risk
0	200	95
1	90	20
2	300	98
3	110	60
4	240	72



```
In [9]: x=df[['speed']]
        y=df['risk']
        x.head()
```

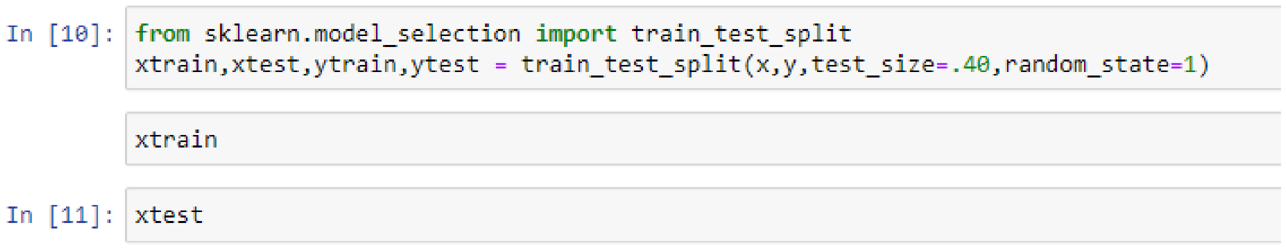
Out[9]:

	speed
0	200
1	90
2	300
3	110
4	240



```
In [6]: y.head()

Out[6]: 0    95
        1    20
        2    98
        3    60
        4    72
        Name: risk, dtype: int64
```



```
In [10]: from sklearn.model_selection import train_test_split
         xtrain,xtest,ytrain,ytest = train_test_split(x,y,test_size=.40,random_state=1)

         xtrain

In [11]: xtest
```

Out[11]:

	speed
3	110
7	230
6	50
2	300
10	290
4	240

In [12]:
```
ytrain
```

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

In [15]:
```
ytest
```

Out[15]:
```
3     60
7     85
6      7
2     98
10    82
4     72
Name: risk, dtype: int64
```

In [16]:
```
from sklearn.linear_model import LinearRegression
```

In [17]:
```
reg=LinearRegression()
```

In [18]:
```
reg.fit(xtrain,ytrain)
```

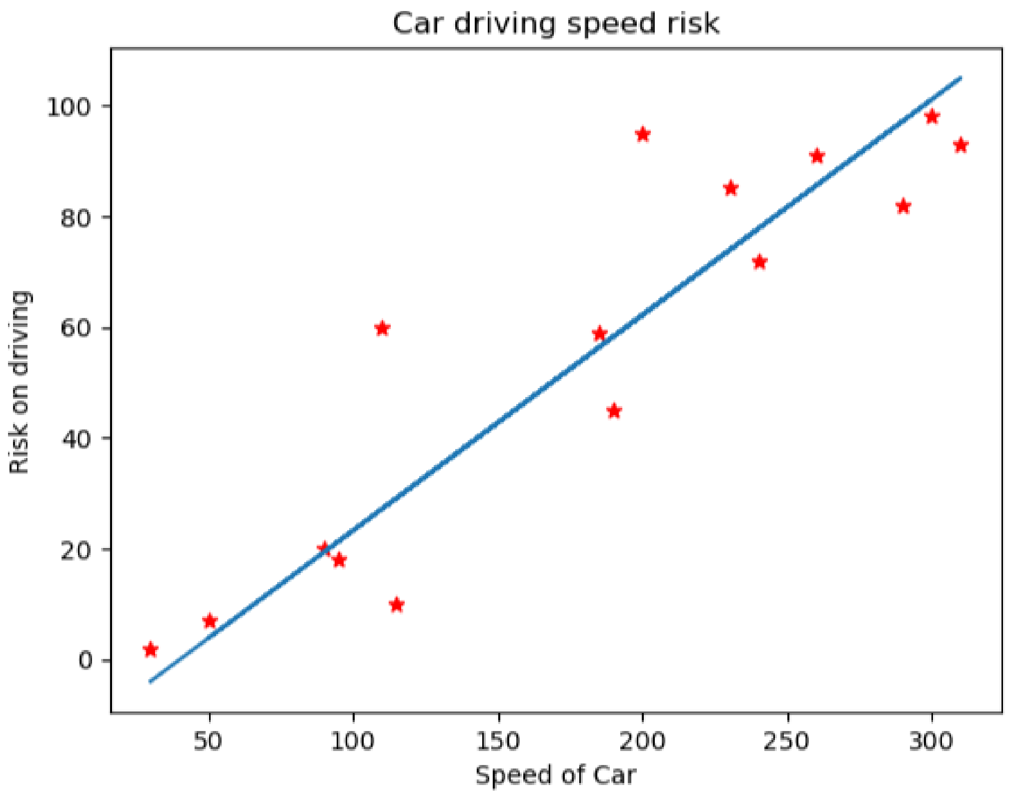
Out[18]:
```
▾ LinearRegression
LinearRegression()
```

```
In [21]: reg.predict(xtest)  #compare with ytest

Out[21]: array([ 27.15301215,  73.82259334,   3.81822156, 101.04651569,
                 97.15738393,  77.7117251 ])
```

```
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 [22]: reg.predict([[180]])

         C:\Users\MD_Sh\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\base.py:420: UserWarning: X does not have vali
         d feature names, but LinearRegression was fitted with feature names
           warnings.warn(

Out[22]: array([54.37693451])
```

Experiment Number : 07

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

Objectives:

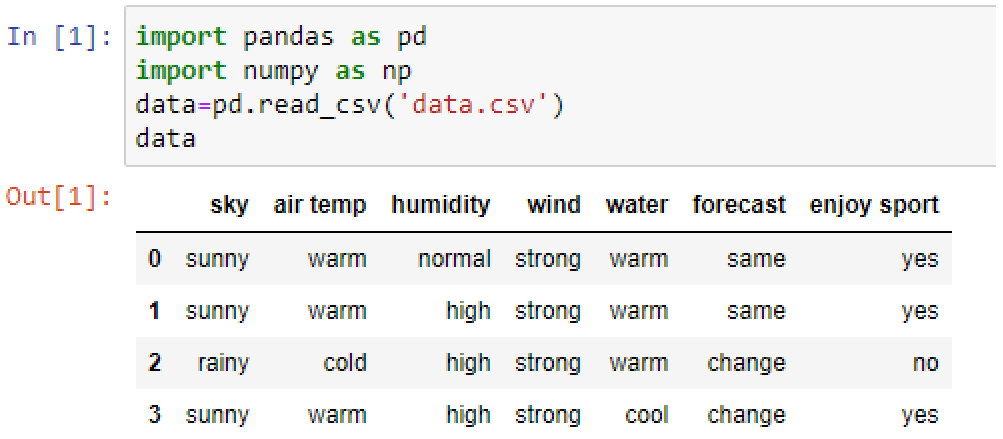
1. To know about find-s algorithm
2. To know how we use find-s algorithm to train a machine
3. To know how we implement find-s algorithm

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**.**

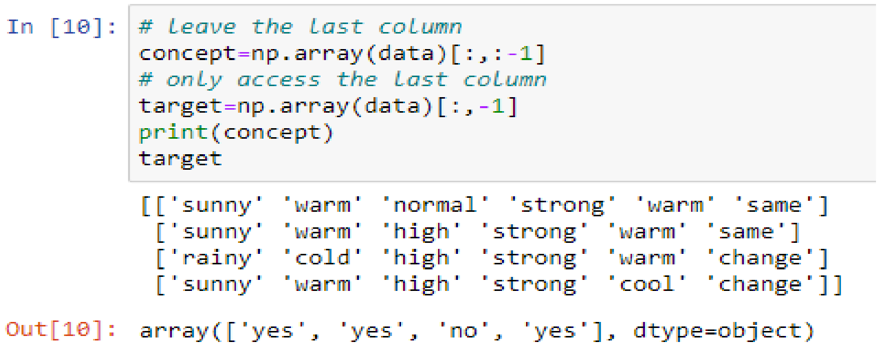
Source Code:



```
In [1]: import pandas as pd
        import numpy as np
        data=pd.read_csv('data.csv')
        data
```

Out[1]:

	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



```
In [10]: # Leave the last column
         concept=np.array(data)[:,:-1]
         # only access the last column
         target=np.array(data)[:,-1]
         print(concept)
         target

         [['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
          ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
          ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
          ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]

Out[10]: array(['yes', 'yes', 'no', 'yes'], dtype=object)
```

```
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
```

In [14]:
```
result=train(concept,target)
print(result)
```

```
['sunny' 'warm' '?' 'strong' '?' '?']
```

In [17]:
```
day=input("Enter 6 word to check:")
day=day.split()
check=True
```

```
Enter 6 word to check:sunny ok ? strong ? ?
```

In [18]:
```
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")
```

```
Not Possible
```

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 high-dimensional 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".

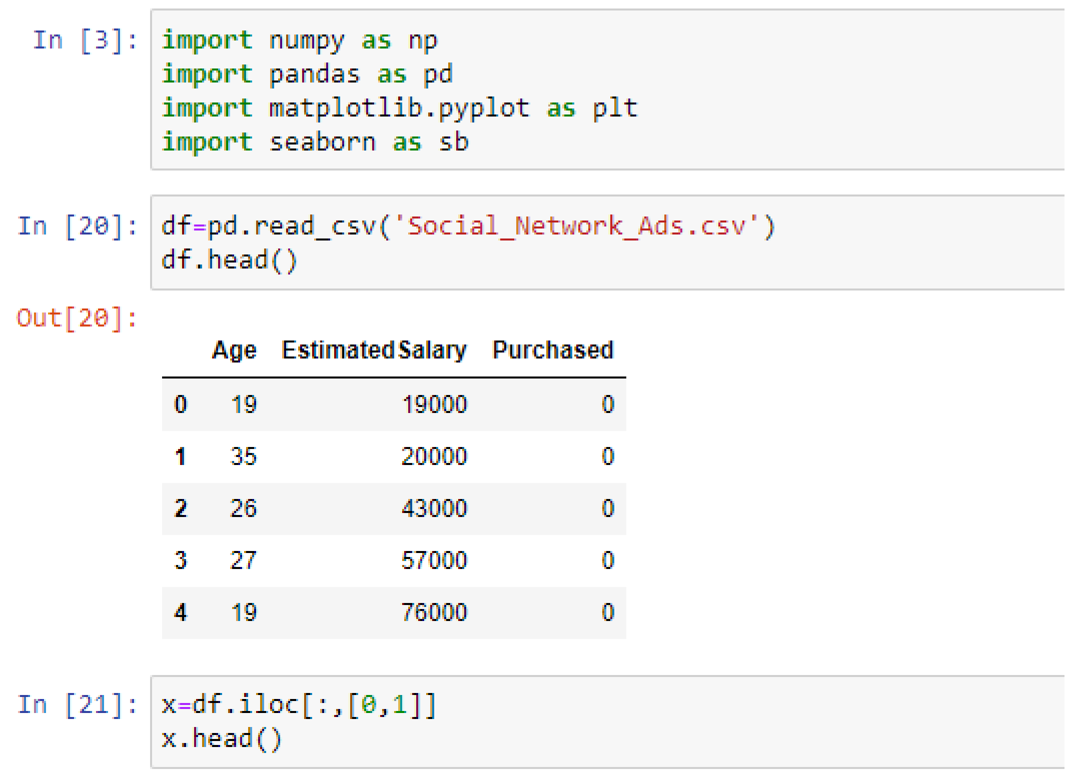
SVM is a powerful algorithm because it can handle non-linear decision boundaries using a technique called kernel trick. Kernel trick allows SVM to implicitly map the input data into a high-dimensional space, where it is more likely that a linear decision boundary can separate the classes.

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:



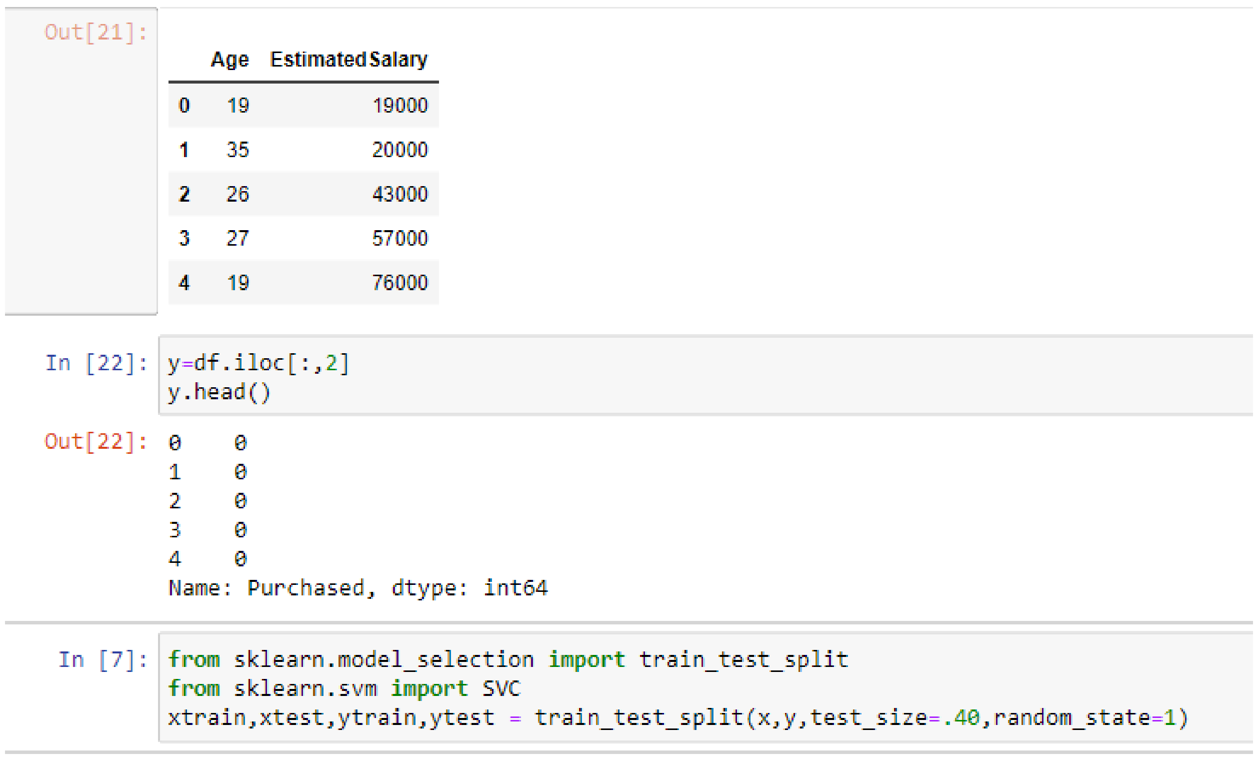
```
In [3]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sb
```

```
In [20]: df=pd.read_csv('Social_Network_Ads.csv')
         df.head()
```

Out[20]:

	Age	EstimatedSalary	Purchased
0	19	19000	0
1	35	20000	0
2	26	43000	0
3	27	57000	0
4	19	76000	0

```
In [21]: x=df.iloc[:,[0,1]]
         x.head()
```



Out[21]:

	Age	EstimatedSalary
0	19	19000
1	35	20000
2	26	43000
3	27	57000
4	19	76000

```
In [22]: y=df.iloc[:,2]
         y.head()
```

```
Out[22]: 0    0
         1    0
         2    0
         3    0
         4    0
         Name: Purchased, dtype: int64
```

```
In [7]: from sklearn.model_selection import train_test_split
        from sklearn.svm import SVC
        xtrain,xtest,ytrain,ytest = train_test_split(x,y,test_size=.40,random_state=1)
```

In [12]:
```python
print('Training data : ',xtrain.shape)
```
```
Training data :  (240, 2)
```

In [13]:
```python
print('Testing data : ',xtest.shape)
```
```
Testing data :  (160, 2)
```

In [23]:
```python
xtrain.head()
```
Out[23]:

	Age	EstimatedSalary
163	35	38000
247	57	122000
378	41	87000
145	24	89000
251	37	52000

In [24]:
```python
xtest.head()
```
Out[24]:

	Age	EstimatedSalary
398	36	33000
125	39	61000
328	36	118000
339	39	122000
172	26	118000

In [25]:
```python
ytrain.head()
```
Out[25]:
```
163    0
247    1
378    1
145    0
251    0
Name: Purchased, dtype: int64
```

In [17]:
```python
model=SVC(gamma='auto')
```

```
In [18]: model.fit(xtrain,ytrain)
Out[18]:
  ▾ SVC
  SVC(gamma='auto')

In [19]: model.score(xtest,ytest)
Out[19]: 0.66875
```