Data Types

Integer- 0-9, Float- 25.26, String- A-Z, Boolean- True/False, Complex -Real and Imaginary Variable is a memory container use for store value into memory location

In [11]:

```
RollNo=101
print(RollNo)
print(type(RollNo))
```

```
101
<class 'int'>
```

In [12]:

```
StdName="James"
print(StdName)
print(type(StdName))
```

```
James
<class 'str'>
```

In [15]:

```
Marks=95.25
print(Marks)
print(type(Marks))
```

```
95.25
<class 'float'>
```

In [18]:

```
Result=True
print(Result)
print(type(Result))
```

```
True
<class 'bool'>
```

In [22]:

```
cmpnum=6+5j
print(cmpnum)
print(type(cmpnum))
```

```
(6+5j)
<class 'complex'>
```

Input Information from user

In [53]:

```
EmpCode=int(input("Enter Employee Code "))
print(EmpCode)
EmpName=input("Enter Employee Name ")
print(EmpName)
Salary=float(input("Enter Employee Salary "))
print(Salary+1000)
```

```
Enter Employee Code 1001
1001
Enter Employee Name james
james
Enter Employee Salary 8000
9000.0
```

Data Structure

List is Mutable, it can be changed, it is ordered

list=[] and the index number started from 0

In [34]:

```
StdInfo=[101,'James','Delhi',250001]
print(StdInfo)
print(type(StdInfo))
```

```
[101, 'James', 'Delhi', 250001]
<class 'list'>
```

In [37]:

```
print(StdInfo[1])
```

```
James
```

In [39]:

```
StdInfo[0]=501
print(StdInfo)
```

```
[501, 'James', 'Delhi', 250001]
```

In [42]:

```
print(StdInfo[0:3])
```

```
[501, 'James', 'Delhi']
```

In [67]:

```
Marks=[40,90,60,45,35,67,89,15,68]
print(Marks)
```

```
[40, 90, 60, 45, 35, 67, 89, 15, 68]
```

In [68]:

```
Marks.sort()
print(Marks)
Marks.reverse()
print(Marks)
```

```
[15, 35, 40, 45, 60, 67, 68, 89, 90]
[90, 89, 68, 67, 60, 45, 40, 35, 15]
```

In [54]:

```
Marks=[40,90,60,45,35,67,89,15,68]
print(Marks[::-1])
Marks.append(66)
print(Marks)
print(Marks)
Marks.extend([11,22,33])
print(Marks)
Marks.remove(66)  #delete according to value
print(Marks)
Marks.pop()  #delete from last
print(Marks)
```

```
[68, 15, 89, 67, 35, 45, 60, 90, 40]
[40, 90, 60, 45, 35, 67, 89, 15, 68, 66]
[40, 90, 60, 45, 35, 67, 89, 15, 68, 66]
[40, 90, 60, 45, 35, 67, 89, 15, 68, 66, 11, 22, 33]
[40, 90, 60, 45, 35, 67, 89, 15, 68, 11, 22, 33]
[40, 90, 60, 45, 35, 67, 89, 15, 68, 11, 22]
```

Tuple is immutable It can not be changed. it is ordered. it is fast as compare to list

tuple=()

In [76]:

```
College=('C001','RG College',6500.25,'Delhi')
print(College)
print(type(College))
```

```
('C001', 'RG College', 6500.25, 'Delhi')
<class 'tuple'>
```

In [80]:

```
#College[0]='C002' we can not change into the tuple
```

In [85]:

```
print(len(College))
print(College[0])
```

```
4
C001
```

Set show only unique values

In [89]:

```
ids={101,102,102,103,104,104,105,106,106,107}
print(ids)
print(type(ids))
```

```
{101, 102, 103, 104, 105, 106, 107}
<class 'set'>
```

Dictionery store values into key,pair and key should be unique

In [1]:

```
color={1:"Red",2:"Green",3:"Blue"}
print(color)
print(type(color))
```

```
{1: 'Red', 2: 'Green', 3: 'Blue'}
<class 'dict'>
```

In [5]:

```
print(color.keys())
print(color.values())
print(color[2])
```

```
dict_keys([1, 2, 3])
dict_values(['Red', 'Green', 'Blue'])
Green
```

Condition with if and elif and else

In [8]:

```
marks=int(input("Enter your marks "))
if marks>=33:
    print("Pass")
else:
    print("Fail")
```

```
Enter your marks 60
Pass
```

In [13]:

```
marks=int(input("Enter your marks "))
if marks>=60 and marks<=100:
    print("1st Division")
elif marks>=45 and marks<60:
    print("2nd Division")
elif marks>=33 and marks<45:
    print("3rd Division")
elif marks>=0 and marks<33:
    print("Fail")
else:
    print("Invalid Marks ")
```

```
Enter your marks 150
Invalid Marks
```

Loop use for iteration set of instruction or statement

In [2]:

```
count=1                 #initilizaion
while count<6:          #condition
    print("While Loop ",count)
    count=count+1       #increment
```

```
While Loop  1
While Loop  2
While Loop  3
While Loop  4
While Loop  5
```

In [5]:

```
count=6               #initilizaion
while count>=1:        #condition
    print("While Reverse Loop ",count)
    count=count-1
```

```
While Reverse Loop  6
While Reverse Loop  5
While Reverse Loop  4
While Reverse Loop  3
While Reverse Loop  2
While Reverse Loop  1
```

For Loop

In [13]:

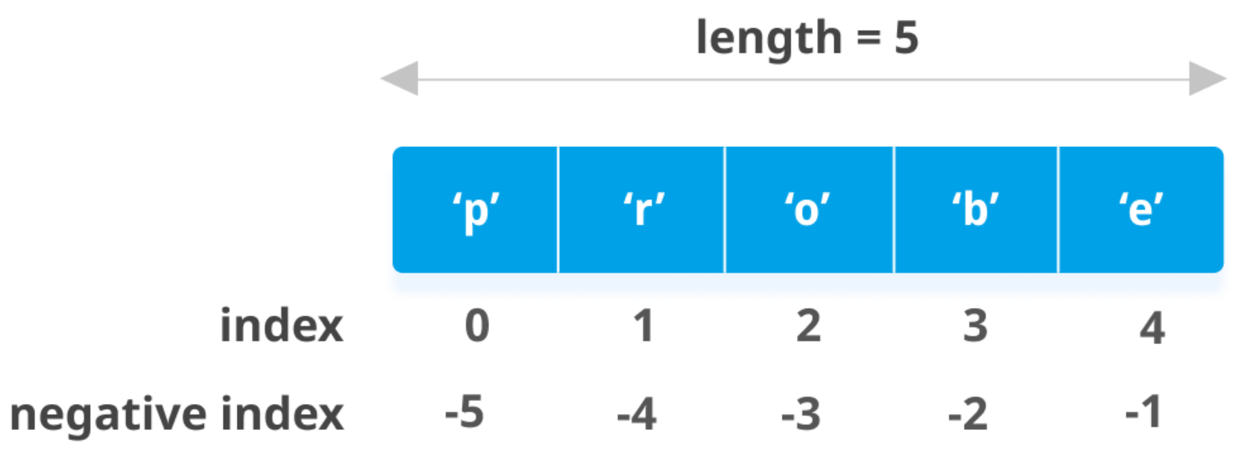
```
for i in range(1,6):
    print("For Loop ",i)
```

```
For Loop  1
For Loop  2
For Loop  3
For Loop  4
For Loop  5
```

In [14]:

```
for i in range(0,10,2):
    print("For Loop ",i)
```

```
For Loop  0
For Loop  2
For Loop  4
For Loop  6
For Loop  8
```



In [12]:

```
for i in range(6,0,-1):
    print("Reverse For Loop ",i)
```

```
Reverse For Loop  6
Reverse For Loop  5
Reverse For Loop  4
Reverse For Loop  3
Reverse For Loop  2
Reverse For Loop  1
```

In [23]:

```
marks=[40,60,80,90,82]
print(marks[0:5])
print(marks[-1])
```

```
[40, 60, 80, 90, 82]
82
```

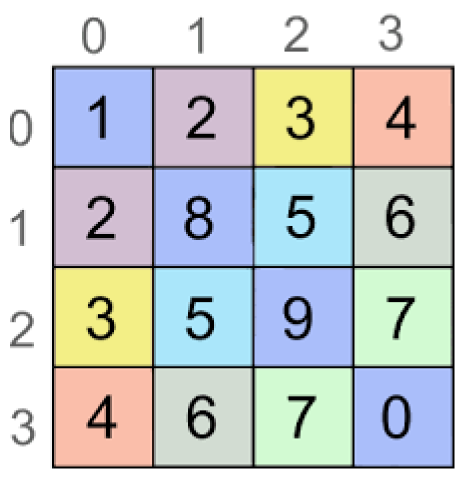
Nested Condition

In [26]:

```
age=int(input("How old are you "))
if age>16:
    print("Your age is enough for drive a car")
else:
    if age==16:
        print("your age is ok ")
    else:
        print("you age is below for drive a car")
```

```
How old are you 12
you age is below for drive a car
```

Matrix with list



$$\begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 1 & 2 & 3 & 4 \\ 1 & 2 & 8 & 5 & 6 \\ 2 & 3 & 5 & 9 & 7 \\ 3 & 4 & 6 & 7 & 0 \end{array}$$

In [32]:

```
matrix=[[1,2,3,4],
       [2,8,5,6],
       [3,5,9,7],
       [4,6,7,0]]
print(matrix[1][2])
for i in matrix:
    print(i)
```

```
5
[1, 2, 3, 4]
[2, 8, 5, 6]
[3, 5, 9, 7]
[4, 6, 7, 0]
```

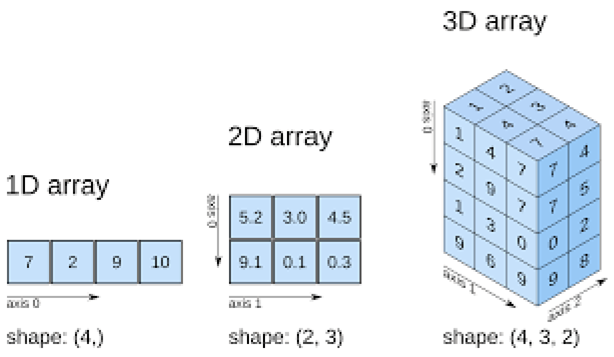
Nested Loop

In [35]:

```
for i in range(1,6):
    for j in range(1,6):
        print(i,end=" ")
    print()
```

```
1 1 1 1 1
2 2 2 2 2
3 3 3 3 3
4 4 4 4 4
5 5 5 5 5
```

Numpy mostly use for matrix or Array



In [40]:

```
import numpy as np
arr1=np.array([7,2,9,10])
print(arr1)
print(arr1.shape)
```

```
[ 7  2  9 10]
(4,)
```

In [42]:

```
import numpy as np
arr2=np.array([[5.2,3.1,4.5],[9.1,0.1,0.3]])
print(arr2)
print(arr2.shape)
print(arr2[0][0])
```

```
[[5.2 3.1 4.5]
 [9.1 0.1 0.3]]
(2, 3)
5.2
```

Pandas

In [48]:

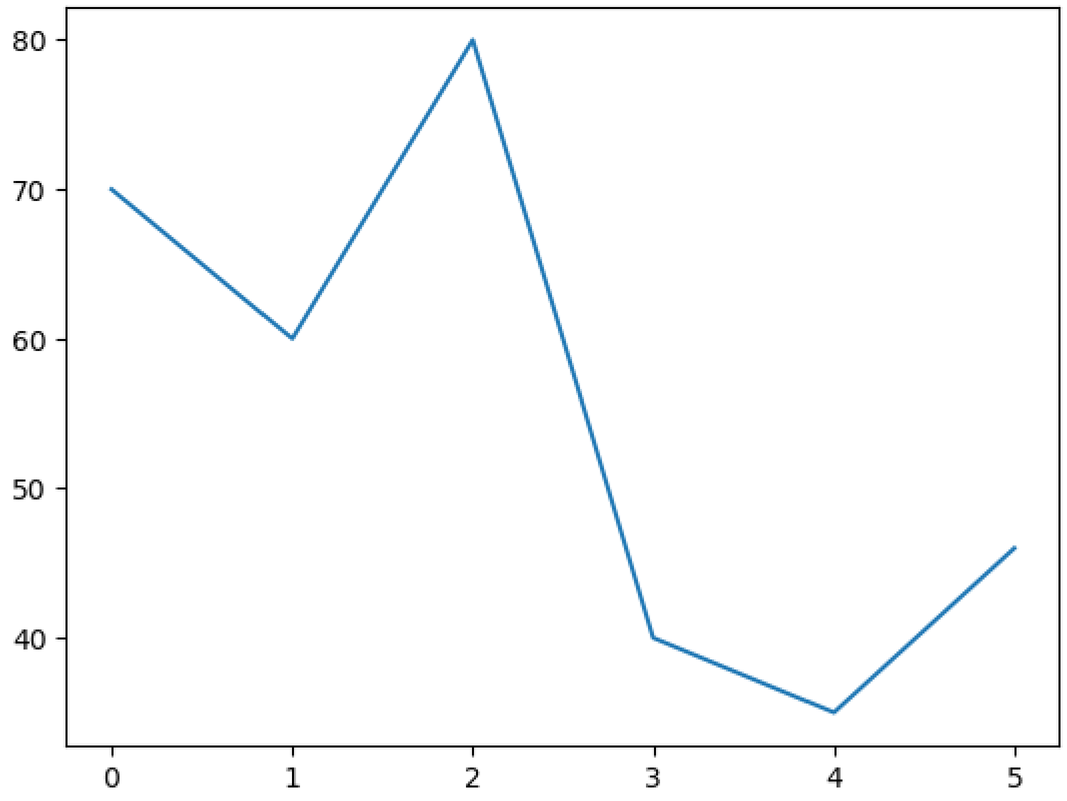
```
# import pandas as pd
import numpy as np

arr2=np.array([[5.2,3.1,4.5],[9.1,0.1,0.3]])
data=pd.DataFrame(arr2)
print(data)
```

```
     0    1    2
0  5.2  3.1  4.5
1  9.1  0.1  0.3
```

Matplotlib use for data visualization

In [49]:

```
import matplotlib.pyplot as plt
import numpy as np

marks=np.array([70,60,80,40,35,46])
plt.plot(marks)
plt.show()
```



Pandas Library use for load files and Data Manipulation

In [1]:

```
import pandas as pd

df=pd.read_csv('CSVFileForPanda.csv')
df
```

Out[1]:

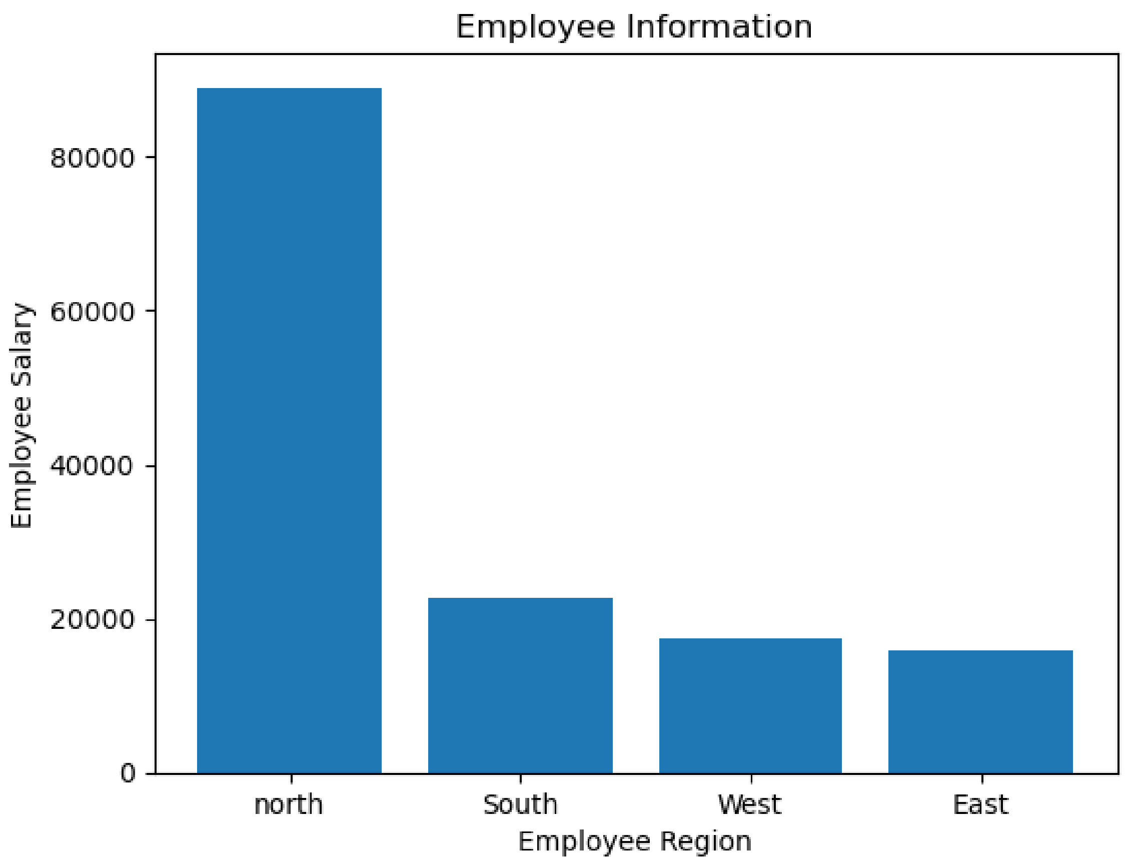
	empcode	First Name	Last Name	Dept	Region	Branch	Hiredate	Salary
0	5	Seema	Ranganathan	R&D	north	Kanpur	NaN	89000
1	84	Raja	Raymondekar	Sales	north	Ferozepur	1/1/1977	30000
2	39	Sheetal	Dodhia	Finance	north	Delhi	1/1/1977	24500
3	38	Chitra	Pednekar	Finance	north	Aligarh	10/1/1982	24500
4	90	Priyanka	Mehta	R&D	north	Jaipur	1/6/1980	22750
5	450	Tejal	Patel	Sales	South	Aligarh	2/21/1979	22750
6	67	Uday	Naik	Personnel	South	Lucknow	10/28/1988	20125
7	12	Shilpa	Lele	Admin	South	Jammu	3/1/1983	20000
8	62	Asha	Trivedi	Sales	South	Kanpur	11/26/1987	19250
9	61	Anuradha	Zha	Admin	South	Agra	11/25/1987	19250
10	52	Heena	Godbole	CCD	South	Lucknow	10/21/1988	19250
11	86	Sonia	Sasan	CCD	South	Jammu	1/15/1998	17500
12	80	Dayanand	Gandhi	Mktg	South	Ferozepur	10/30/1988	17500
13	79	Sagar	Bidkar	Mktg	South	Delhi	10/29/1988	17500
14	77	Drishti	Shah	R&D	West	Delhi	10/27/1988	17500
15	73	Laveena	Shenoy	CCD	West	Jaipur	10/23/1988	17500
16	66	Parul	Shah	Personnel	West	Agra	10/27/1988	17500
17	63	Waheda	Sheikh	R&D	West	Jammu	10/24/1988	17500
18	58	Maya	Panchal	Mktg	West	Agra	11/5/1990	17500
19	98	Chetan	Dalvi	CCD	West	Delhi	4/11/1989	17325
20	83	Kunal	Shah	CCD	West	Aligarh	3/2/1999	3500
21	33	Tapan	Ghoshal	CCD	West	Ambala	7/7/1997	4000
22	85	Ruby	Joseph	R&D	West	Agra	1/14/1998	7000
23	34	Zarina	Vora	CCD	West	Lucknow	9/29/1991	8750
24	29	Hajra	Hoonjan	Admin	West	Jaipur	5/4/1996	9625
25	19	Parvati	Khanna	Mktg	West	Mathura	8/13/1986	10500
26	45	Kinnari	Mehta	R&D	East	Ferozepur	7/7/1997	11375
27	1	Raja	Sadiq	Mktg	East	PUNE	6/5/1999	12000
28	4	Beena	Mavadia	Admin	East	Delhi	9/23/2014	12250
29	6	Julie	D'Souza	R&D	East	Mathura	9/4/1988	12425
30	8	Neena	Mukherjee	R&D	East	Agra	9/4/1989	12425
31	68	Mandakini	Desai	Sales	East	Delhi	12/1/1995	14000
32	69	Pravin	Joshi	Personnel	East	Delhi	3/3/1995	14000
33	9	Pankaj	Sutradhar	Sales	East	Ambala	12/12/1999	14875
34	15	K.Sita	Narayanan	Personnel	East	Jammu	12/13/1984	14875
35	16	Priya	Shirodkar	Personnel	East	Jaipur	12/14/1984	14875
36	31	Giriraj	Gupta	R&D	East	Agra	10/1/1982	15750

	empcode	First Name	Last Name	Dept	Region	Branch	Hiredate	Salary
37	46	Jeena	Baig	Sales	East	Lucknow	9/29/1991	15750
38	47	Vicky	Joshi	Admin	East	Kanpur	2/7/1988	15750
39	51	Mario	Fernandes	Sales	East	Jammu	10/20/1988	15750

Group By for Arrange Data according to given column

In [13]:

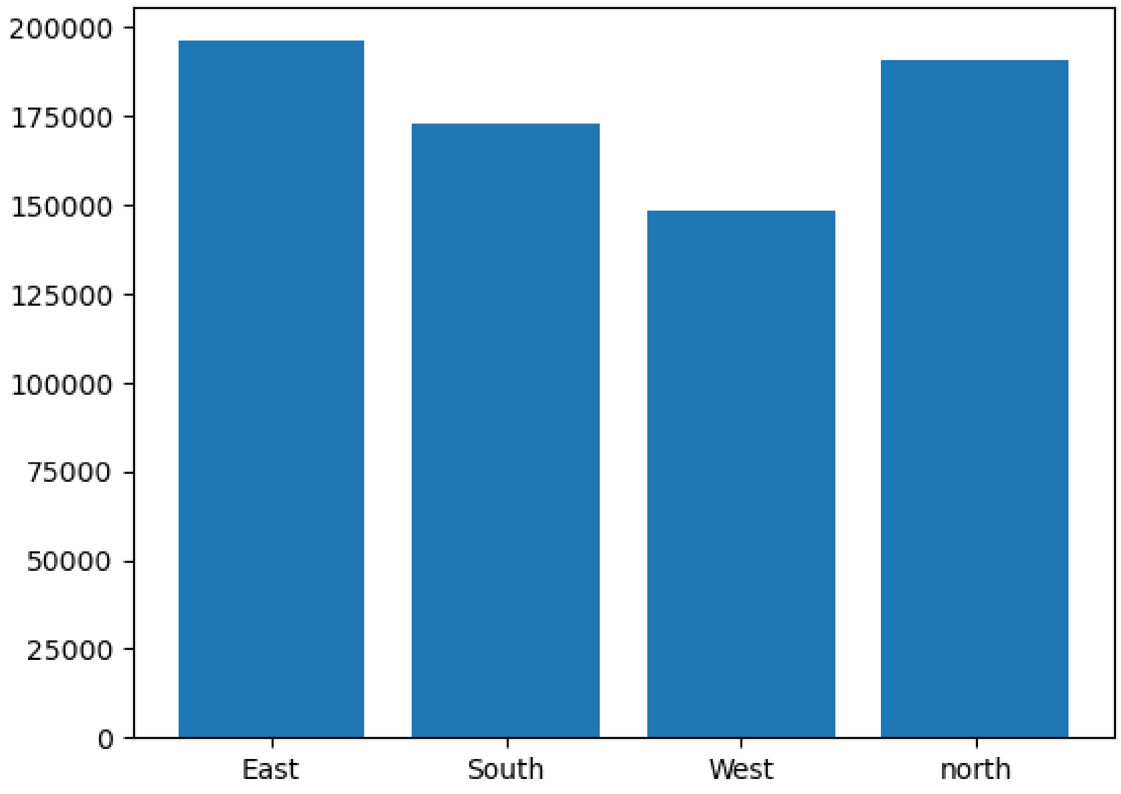
```
import matplotlib.pyplot as plt
EmpRegion=df['Region']
EmpSalary=df['Salary']

plt.bar(EmpRegion,EmpSalary)
plt.title("Employee Information")
plt.xlabel('Employee Region')
plt.ylabel('Employee Salary')
plt.show()
```



In [30]:

```
dfd=df[['Region','Salary']]
# print(dfd)
pie_df=dfd.groupby('Region').sum()
pie_df=pd.DataFrame(pie_df)

pie_df.reset_index(inplace=True)
print(pie_df)
plt.bar(pie_df['Region'],pie_df['Salary'])
plt.show()
```

```
  Region  Salary
0   East  196100
1  South  173125
2   West  148200
3  north  190750
```



In [52]:

```
import pandas as pd
dff=pd.read_csv('DataForStat.csv')
dff
```

Out[52]:

	Gender	Height	Weight	bmi	Age
0	Male	174.0	80	26.4	25.0
1	Male	189.0	87	24.4	27.0
2	Female	185.0	80	23.4	30.0
3	Female	165.0	70	25.7	26.0
4	Male	149.0	61	27.5	28.0
5	Male	177.0	70	22.3	29.0
6	Female	NaN	65	30.1	31.0
7	Male	154.0	62	26.1	32.0
8	Male	174.0	90	29.7	NaN

In [46]:

```
dff.describe()
```

Out[46]:

	Height	Weight	bmi	Age
count	9.000000	9.000000	9.000000	8.00000
mean	168.444444	73.888889	26.177778	28.50000
std	15.034220	10.740629	2.639497	2.44949
min	149.000000	61.000000	22.300000	25.00000
25%	154.000000	65.000000	24.400000	26.75000
50%	174.000000	70.000000	26.100000	28.50000
75%	177.000000	80.000000	27.500000	30.25000
max	189.000000	90.000000	30.100000	32.00000

```
In [57]:
```

```
Emp_age=dff['Age'].max()
print("Maximum ",Emp_age)
Emp_age=dff['Age'].min()
print("Minimum ",Emp_age)
Emp_age=dff['Age'].mean()
print("Mean ",Emp_age)
Emp_age=dff['Age'].median()
print("Median ",Emp_age)
Emp_age=dff['Age'].var()
print("Variance ",Emp_age)
Emp_age=dff['Age'].std()
print("Standard Deviation ",Emp_age)
dff.shape
# dff.duplicated()
```

```
Maximum  32.0
Minimum  25.0
Mean  28.5
Median  28.5
Variance  6.0
Standard Deviation  2.449489742783178
```

```
Out[57]:
```

```
(9, 5)
```

Load Excel File

In [60]:

```python
import pandas as pd

df=pd.read_excel('SampleExcelFile.xlsx')
df
```

Out[60]:

	First Name	Last Name	Dept	Branch	Years	Total Salary
0	Parvati	Khanna	Mktg	Mathura	1986	10500
1	Hajra	Hoonjan	Admin	Jaipur	1996	9625
2	Tapan	Ghoshal	CCD	Ambala	1997	4000
3	Zarina	Vora	CCD	Lucknow	1991	8750
4	Maya	Panchal	Mktg	Agra	1990	17500
5	Waheda	Sheikh	R&D	Jammu	1988	17500
6	Parul	Shah	Personnel	Agra	1988	17500
7	Laveena	Shenoy	CCD	Jaipur	1988	17500
8	Drishti	Shah	R&D	Delhi	1988	17500
9	Kunal	Shah	CCD	Aligarh	1999	3500
10	Ruby	Joseph	R&D	Agra	1998	7000
11	Chetan	Dalvi	CCD	Delhi	1989	17325

Load Json File

In [62]:

```python
import pandas as pd
df=pd.read_json('employeedata.json')
df
```

Out[62]:

	empid	title	salary
0	101	IT Company Employee	41000
1	102	Sales Company Employee	20000

In [63]:

```
df=pd.read_json('https://dummyjson.com/products/1')
df
```

Out[63]:

	id	title	description	price	discountPercentage	rating	stock	brand	category	
0	1	iPhone 9	An apple mobile which is nothing like apple	549	12.96	4.69	94	Apple	smartphones	http
1	1	iPhone 9	An apple mobile which is nothing like apple	549	12.96	4.69	94	Apple	smartphones	http
2	1	iPhone 9	An apple mobile which is nothing like apple	549	12.96	4.69	94	Apple	smartphones	http
3	1	iPhone 9	An apple mobile which is nothing like apple	549	12.96	4.69	94	Apple	smartphones	http
4	1	iPhone 9	An apple mobile which is nothing like apple	549	12.96	4.69	94	Apple	smartphones	http

Cross Tab

In [64]:

```
import numpy as np
import pandas as pd

a = np.array(["foo", "foo", "foo", "foo", "bar", "bar",
              "bar", "bar", "foo", "foo", "foo"], dtype=object)
b = np.array(["one", "one", "one", "two", "one", "one",
              "one", "two", "two", "two", "one"], dtype=object)
c = np.array(["dull", "dull", "shiny", "dull", "dull", "shiny",
              "shiny", "dull", "shiny", "shiny", "shiny"],
             dtype=object)
dff=pd.crosstab(a, [b, c], rownames=['a'], colnames=['b', 'c'])
dff
```

Out[64]:

b	**one**		**two**	
c	**dull**	**shiny**	**dull**	**shiny**
a				
bar	1	2	1	0
foo	2	2	1	2

Performing Some Data Operation on File

In [76]:

```
import pandas as pd
import numpy as np

data=np.array([['Gita',"James",'Ravi','James'],
               [50000,60000,40000,60000],
              [50000,60000,40000,60000]])
data=pd.DataFrame(data)
data
data.duplicated()
data.drop_duplicates()
```

Out[76]:

	0	**1**	**2**	**3**
0	Gita	James	Ravi	James
1	50000	60000	40000	60000

Pivot Return reshaped DataFrame organized by given index / column values.

Reshape data (produce a “pivot” table) based on column values. Uses unique values from specified index / columns to form axes of the resulting DataFrame. This function does not support data aggregation, multiple values will result in a MultiIndex in the columns. See the User Guide for more on reshaping.

DataFrame.pivot(*, columns, index=typing.Literal[], values=typing.Literal[])[source]

In [78]:

```
df = pd.DataFrame({'foo': ['one', 'one', 'one', 'two', 'two','two'],
                   'bar': ['A', 'B', 'C', 'A', 'B', 'C'],
                   'baz': [1, 2, 3, 4, 5, 6],
                   'zoo': ['x', 'y', 'z', 'q', 'w', 't']})
df1=df.pivot(index='foo', columns='bar', values='baz')
df1
```

Out[78]:

bar	A	B	C
foo			
one	1	2	3
two	4	5	6

MultiIndex, Stack and Unstack

Multiindex arranging your matrix data accoding to x,y cordinate stack transpose data from column to row unstack transpose data from row to column

In [79]:

```
dfmulti=pd.DataFrame({'city':['Noida','Pune',"Delhi"],
                      'institue':['ABC','DEF','PQR']})
dfmulti
```

Out[79]:

	city	institue
0	Noida	ABC
1	Pune	DEF
2	Delhi	PQR

In [83]:

```
mdx=pd.MultiIndex(levels=[['Noida','Pune','Delhi'],
                          ['ABC','DEF','PQR']],
                          codes=[[0,1,0],[1,0,1]])
mdx
```

Out[83]:

```
MultiIndex([('Noida', 'DEF'),
            ( 'Pune', 'ABC'),
            ('Noida', 'DEF')],
           )
```

Remove Missing and Null Value

In [6]:

```
import pandas as pd
import numpy as np

df=pd.DataFrame(np.random.randn(5,3),index=['a','c','e','f','h'],columns=['one','two','t
df=df.reindex(['a','b','c','d','e','f','g','h'])
print(df)
print(df['one'].isnull())
```

```
        one       two     three
a  0.580882  0.498150 -1.814720
b       NaN       NaN       NaN
c  1.290225 -0.668934 -0.148518
d       NaN       NaN       NaN
e  0.598051  1.756825 -0.181667
f  1.812799  0.439962  0.520269
g       NaN       NaN       NaN
h  0.019002 -0.674815  1.387486
a    False
b     True
c    False
d     True
e    False
f    False
g     True
h    False
Name: one, dtype: bool
```

In [3]:

```
df['one'].drop_duplicates()
```

Out[3]:

```
a   -1.550300
b         NaN
c    0.020409
e   -0.977804
f   -0.825009
h    1.340749
Name: one, dtype: float64
```

In [4]:

```
df['one'].dropna()
```

Out[4]:

```
a   -1.550300
c    0.020409
e   -0.977804
f   -0.825009
h    1.340749
Name: one, dtype: float64
```

Merge and Join using with Pandas

The merge() method updates the content of two DataFrame by merging them together, using the specified method(s).¶ Use the parameters to control which values to keep and which to replace Syntax dataframe.merge(right, how, on, left_on, right_on, left_index, right_index, sort, suffixes, copy, indicator,

In [46]:

```
import numpy as np
import pandas as pd
df1 = pd.DataFrame({'lkey': ['foo', 'bar', 'baz','foo'], 'value': [4, 2, 3,4]})
df2 = pd.DataFrame({'rkey': ['foo', 'bar', 'baz','foo'],'value': [5, 6, 7,4]})
print(df1)
print(df2)
```

```
  lkey  value
0  foo      4
1  bar      2
2  baz      3
3  foo      4
  rkey  value
0  foo      5
1  bar      6
2  baz      7
3  foo      4
```

In [47]:

```
#Full outer join
df1=df1.merge(df2, left_on='lkey', right_on='rkey')
df1
```

Out[47]:

	lkey	value_x	rkey	value_y
0	foo	4	foo	5
1	foo	4	foo	4
2	foo	4	foo	5
3	foo	4	foo	4
4	bar	2	bar	6
5	baz	3	baz	7

Merge or Join Second Exercise

In [41]:

```
import numpy as np
import pandas as pd
data1 = {
  "name": ["Sally", "Mary", "John","Ram"],
  "age": [50, 40, 30,60]
}

data2 = {
  "name": ["Sally", "Peter", "Micky","Ram"],
  "age": [77, 44, 22,60]
}
d1=pd.DataFrame(data1)
d2=pd.DataFrame(data2)
print(d1)
print(d2)
```

```
    name  age
0  Sally   50
1   Mary   40
2   John   30
3    Ram   60
    name  age
0  Sally   77
1  Peter   44
2  Micky   22
3    Ram   60
```

In [38]:

```
#show Left table
leftable=d1.merge(d2,how="left")
leftable
```

Out[38]:

	name	age
0	Sally	50
1	Mary	40
2	John	30
3	Ram	60

In [35]:

```
#Left outer join
leftjoin=d1.merge(d2,left_on="name",right_on="name",how="left")
leftjoin
```

Out[35]:

	name	age_x	age_y
0	Sally	50	77.0
1	Mary	40	NaN
2	John	30	NaN
3	Ram	60	60.0

In [39]:

```
#Right outer join
rightjoin=d1.merge(d2,left_on="name",right_on="name",how="right")
rightjoin
```

Out[39]:

	name	age_x	age_y
0	Sally	50.0	77
1	Peter	NaN	44
2	Micky	NaN	22
3	Ram	60.0	60

In [48]:

```
#innerjoin
innerjoin=d1.merge(d2)
innerjoin
```

Out[48]:

	name	age
0	Ram	60

Pandas Indexing Iloc, Loc

In [75]:

```
import pandas as pd
df=pd.read_csv('DataForStat.csv')
df
```

Out[75]:

	Gender	Height	Weight	bmi	Age
0	Male	174.0	80	26.4	25.0
1	Male	189.0	87	24.4	27.0
2	Female	185.0	80	23.4	30.0
3	Female	165.0	70	25.7	26.0
4	Male	149.0	61	27.5	28.0
5	Male	177.0	70	22.3	29.0
6	Female	NaN	65	30.1	31.0
7	Male	154.0	62	26.1	32.0
8	Male	174.0	90	29.7	NaN

In [53]:

```
ldf=df.iloc[0:3]
ldf
```

Out[53]:

	Gender	Height	Weight	bmi	Age
0	Male	174.0	80	26.4	25.0
1	Male	189.0	87	24.4	27.0
2	Female	185.0	80	23.4	30.0

In [58]:

```
#index line of code
ldf=df.iloc[0:3,[0,2]]
ldf
```

Out[58]:

	Gender	Weight
0	Male	80
1	Male	87
2	Female	80

In [60]:

```
#line of code or selected row
ldf=df.loc[2]
ldf
```

Out[60]:

```
Gender    Female
Height     185.0
Weight        80
bmi         23.4
Age         30.0
Name: 2, dtype: object
```

In [77]:

```
df2=df[["Gender","Age"]]
df2
```

Out[77]:

	Gender	Age
0	Male	25.0
1	Male	27.0
2	Female	30.0
3	Female	26.0
4	Male	28.0
5	Male	29.0
6	Female	31.0
7	Male	32.0
8	Male	NaN

In [78]:

```
df2=df2.iloc[[1,3,5]]
df2
```

Out[78]:

	Gender	Age
1	Male	27.0
3	Female	26.0
5	Male	29.0

Sk-Learn Library

In [94]:

```
import pandas as pda
from sklearn.preprocessing import StandardScaler
df=pd.read_csv('DataForStat.csv')
# df.head()
df=df[['Height','Age']]
df

#initlise the Scalar
scaler=StandardScaler()
# #to scale data
scaler.fit(df)
```

Out[94]:

```
StandardScaler()
```

Satistics using Statistics Library

In [101]:

```
n_num=[1,2,3,4,5]
n=len(n_num)
total=sum(n_num)
mean=total/n
print("Mean/ Average is ",mean)
```

```
Mean/ Average is  3.0
```

In [17]:

```
import statistics as st
num=[1,2,3,4,5,2]
mean=st.mean(num)
print("Mean is %0.02f"%mean)

#[1,2,2,3,4,5]  (2+3)=5/2=2.5
median=st.median(num)
print("Median is ",median)

mod=st.mode(num)
print("Mode is ",mod)
```

```
Mean is 2.83
Median is  2.5
Mode is  2
```

```
In [33]:
```

```python
import statistics as st

data=[75,72,68,65,67,73]
mean=st.mean(data)
print("Mean is ",mean)

pvar=st.pvariance(data)
print("Population Variance is %0.02f"%pvar)

svar=st.variance(data)
print("Sample variance is ",svar)

psdev=st.pstdev(data)
print("Population Standard Deviation ",psdev)

ssdev=st.stdev(data)
print("Sample Standard Deviation %0.02f"%ssdev)
```

```
Mean is  70
Population Variance is 12.67
Sample variance is  15.2
Population Standard Deviation  3.559026084010437
Sample Standard Deviation 3.90
```

Statistics using Pandas

In [46]:

```
import pandas as pd
import numpy as np
data2=np.array([75,72,68,65,67,73])
data2=pd.DataFrame(data2)
data2
print("Mean is ",data2.mean())
print("Variance is ",data2.var())
print("Standard Deviation is ",data2.std())
data2.describe()
```

```
Mean is  0    70.0
dtype: float64
Variance is  0    15.2
dtype: float64
Standard Deviation is  0    3.898718
dtype: float64
```

Out[46]:

	0
count	6.000000
mean	70.000000
std	3.898718
min	65.000000
25%	67.250000
50%	70.000000
75%	72.750000
max	75.000000

InterQuartile using numpy percentile

In [75]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

data3=np.array([14,20,22,25,40,80,90,70,60,80,60])
q3,q1=np.percentile(data3,[75,25])
print("Qtr1 ",q1)
print("Qtr3 ",q3)
iqr=q3-q1
iqr
```

```
Qtr1  23.5
Qtr3  75.0
```

Out[75]:

```
51.5
```

Interquartile range (IQR)

Quartiles describe the spread of data by breaking into quarters. The median exactly divides the data into two parts. Q1(Lower quartile) is the middle value in the first half of the sorted dataset. Q2– is the median value Q3 (Upper quartile) is the middle value in the second half of the sorted dataset The interquartile range is the difference between the 75th percentile(Q3) and the 25th percentile(Q1). 50% of data fall within this range

In [76]:

```python
import pandas as pd
df=pd.read_csv("DataForStatistics.csv")
df
```

Out[76]:

	Gender	Age
0	Male	25
1	Male	27
2	Female	30
3	Female	26
4	Male	28
5	Male	29
6	Female	31
7	Male	32
8	Male	27

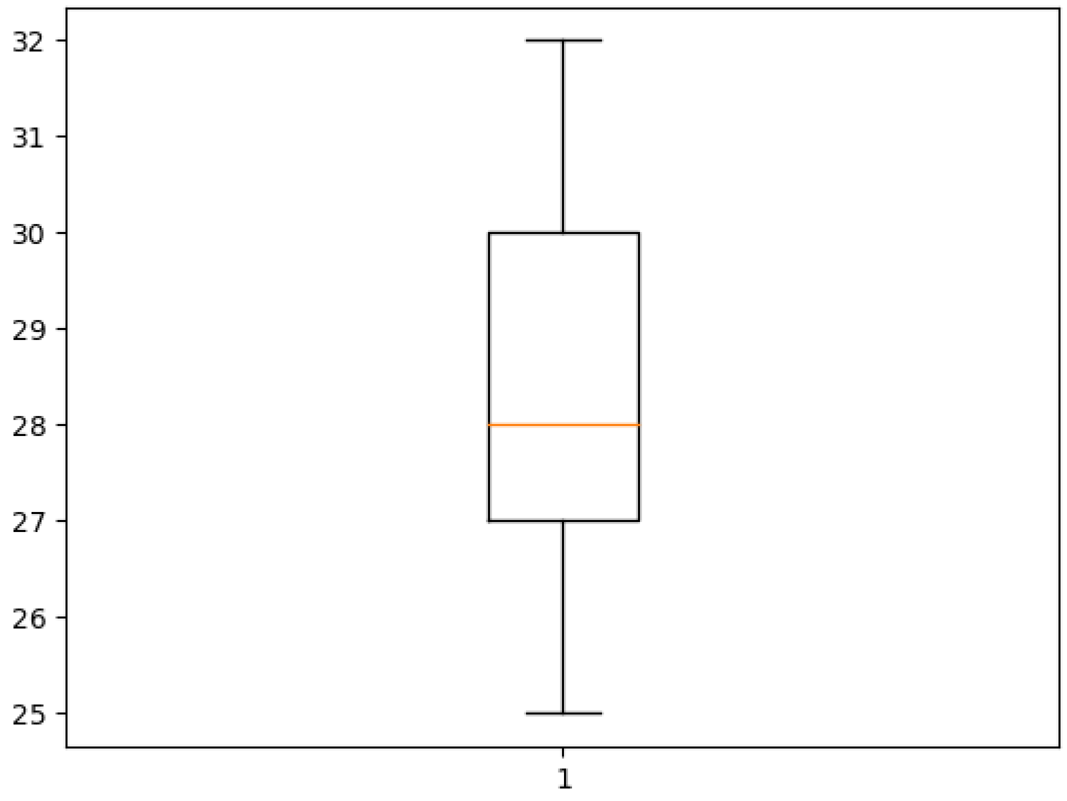
In [78]:

```python
Q1=df["Age"].quantile(0.25)
print("Quater One ",Q1)
Q2=df["Age"].quantile(0.50)
print("Quater Two ",Q2)
Q3=df["Age"].quantile(0.75)
print("Quater Three ",Q3)
IQR=Q3-Q1
print("IQR ",IQR)
```

```
Quater One  27.0
Quater Two  28.0
Quater Three  30.0
IQR  3.0
```

In [79]:

```
import matplotlib.pyplot as plt
plt.boxplot(df["Age"])
plt.show()
```



Skewness show Negative and Positive 0=normal, >0 Left Tail, <0 Right Tail

In [64]:

```
data3=pd.DataFrame(data3)
data3.skew()
```

Out[64]:

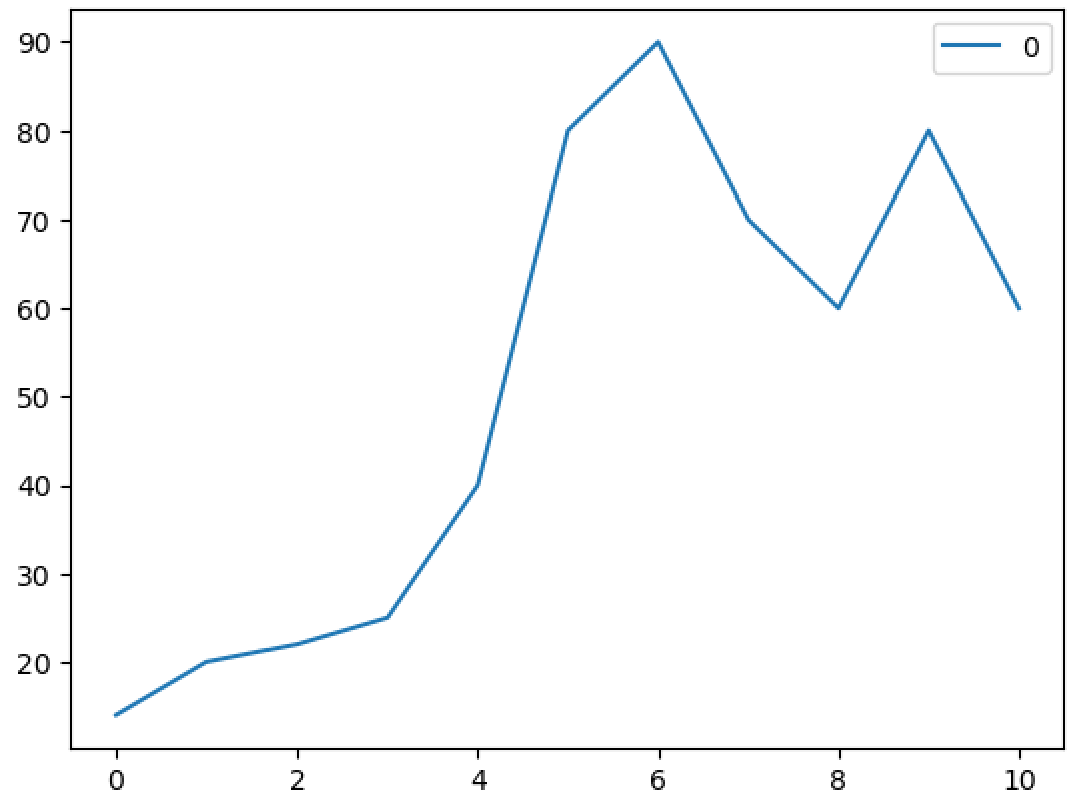
```
0   -0.041955
dtype: float64
```

In [65]:

```
data3.plot()
```

Out[65]:

```
<AxesSubplot:>
```



Kurtosis Normal Distribution= 3, <3 Play Kurtic, >3= lepto kurtic

In [69]:

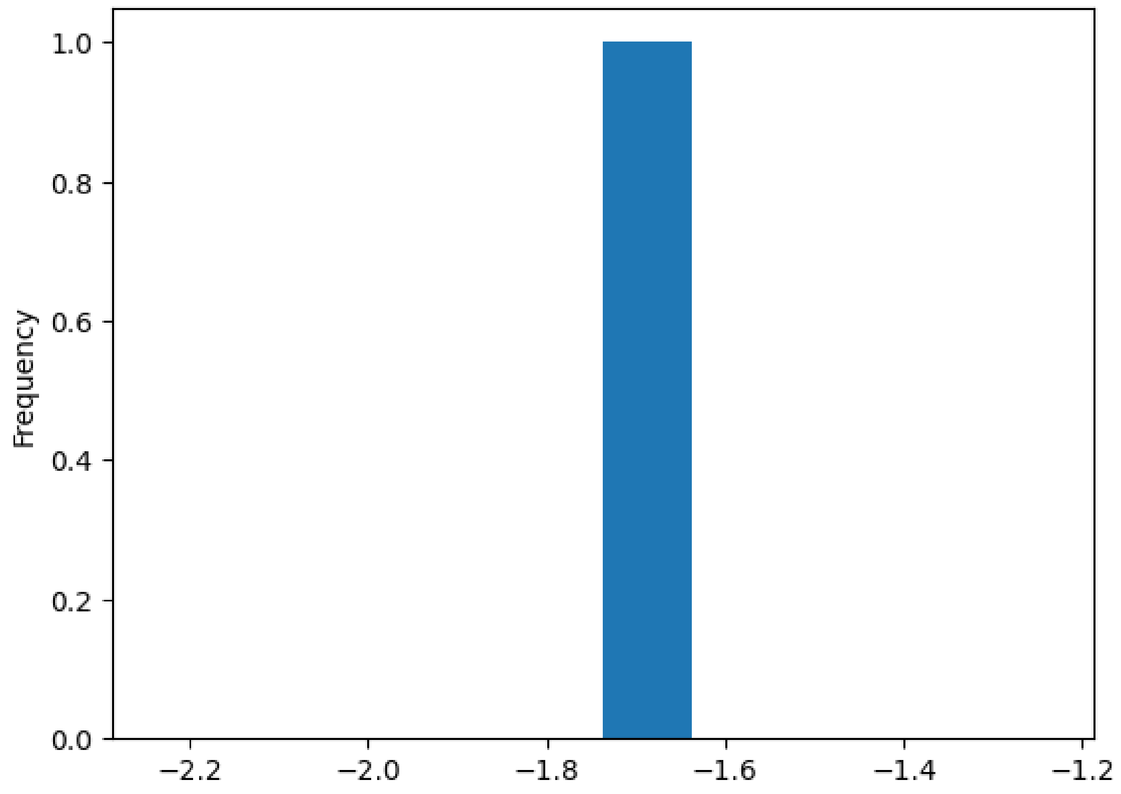
```
kdata=data3.kurtosis()
kdata
```

Out[69]:

```
0   -1.736156
dtype: float64
```

In [72]:

```
kdata.plot(kind='hist')
plt.show()
```



Hypothesis Testing

Null hypothesis and alternative hypothesis are the two different methods of hypothesis testing. The premise for a null hypothesis is an occurrence (also called the ground truth). An alternative hypothesis is a presumption that disputes the primary hypothesis. Imagine a woman in her seventies who has a noticeable tummy bump. Medical professionals could presume the bulge is a fibroid.

One Sample T- Test When comparing the mean values of two samples that specific characteristics may connect, a t-test is performed to see if there exists a substantial difference. It is typically employed when data sets, such as those obtained from tossing a coin 100 times and stored as results, would exhibit a normal distribution. It could have unknown variances. The t-test is a method for evaluating hypotheses that allows you to assess a population-applicable assumption.

```
In [81]:
```

```
from scipy.stats import ttest_1samp
import numpy as np

ages = [45, 89, 23, 46, 12, 69, 45, 24, 34, 67]
print(ages)

mean = np.mean(ages)
print(mean)

# Performing the T-Test
t_test, p_val = ttest_1samp(ages, 30)
print("P-value is: ", p_val)

# taking the threshold value as 0.05 or 5%
if p_val < 0.05:
    print(" We can reject the null hypothesis")
else:
    print("We can accept the null hypothesis")


print("T-Test ",t_test)
```

```
[45, 89, 23, 46, 12, 69, 45, 24, 34, 67]
45.4
P-value is:  0.07179988272763561
We can accept the null hypothesis
T-Test  2.0397003109502543
```

Two sampled t-test:-

To ascertain if there exists any statistical confirmation that the related population means are statistically substantially distinct, the Independent Samples T-Test, also known as the 2-sample T-test, analyses the mean values of two independent samples. The Independent Samples T-Test is also a parametric test. The Independent t Test is another name for this test.

```
In [82]:
```

```python
# Python program to implement Independent T-Test on the two independent samples

from scipy.stats import ttest_ind
import numpy as np

data_group1 = np.array([12, 18, 12, 13, 15, 1, 7, 20,
                        21, 25, 19, 31, 21, 17,
                        17, 15, 19, 15, 12, 15])

data_group2 = np.array([23, 22, 24, 25, 21, 26, 21,
                        21, 25, 30, 24, 21, 23, 19,
                        14, 18, 14, 12, 19, 15])

# Calculating the mean of the two data groups
mean1 = np.mean(data_group1)
mean2 = np.mean(data_group2)

# Print mean values
print("Data group 1 mean value:", mean1)
print("Data group 2 mean value:", mean2)

# Calculating standard deviation
std1 = np.std(data_group1)
std2 = np.std(data_group2)

# Printing standard deviation values
print("Data group 1 std value:", std1)
print("Data group 2 std value:", std2)

# Implementing the t-test
t_test,p_val = ttest_ind(data_group1, data_group2)
print("The P-value is: ", p_val)

# taking the threshold value as 0.05 or 5%
if p_val < 0.05:
    print("We can reject the null hypothesis")
else:
    print("We can accept the null hypothesis")
```

```
Data group 1 mean value: 16.25
Data group 2 mean value: 20.85
Data group 1 std value: 6.171507109288622
Data group 2 std value: 4.452808102759426
The P-value is:  0.012117171124028792
We can reject the null hypothesis
```

In [84]:

```
import pandas as pd
df=pd.read_csv('normalcsvfile.csv')
print(df)
```

```
     name  age
0   james   60
1     ram   80
2    ravi   70
3    ritu   45
4   anita   90
5     bob   62
```

Z-Test

In the z-test, which also functions as a hypothesis test, the z-statistic has a normal distribution. As the central limit theorem states, observations are assumed to be roughly normally distributed as sample size increases. Hence the z-test is most effective for samples bigger than 30.

One-Sample Z-Test:- Assume a trader wants to determine if a stock's daily mean gain is more than 3%. The average is defined for a straightforward arbitrary sample of 50 results, which is 2%. Assume that the profits' standard deviation is 1.5 percent. Therefore, the null hypothesis in this situation is whenever the mean is equal to 3%.

In [1]:

```
# Python program to implement One Sample Z-Test

# Importing the required libraries
import pandas as pd
from scipy import stats
from statsmodels.stats import weightstats as stests

# Creating a dataset
data = [89, 93, 95, 93, 97, 98, 96, 99, 93, 97,
        110, 104, 119, 105, 104, 110, 110, 112, 115, 114]

# Performing the z-test
z_test ,p_val = stests.ztest(data, x2 = None, value = 160)
print(p_val)

# taking the threshold value as 0.05 or 5%
if p_val < 0.05:
    print("We can reject the null hypothesis")
else:
    print("We can accept the null hypothesis")
```

```
2.417334226169332e-186
We can reject the null hypothesis
```

Chi-Square Test

This test is used when two categorized variables are from the same population. Its purpose is to decide if the two elements are significantly associated.

For example, we may group people in an election campaign survey based on their preferred method of voting and gender (male or female) (Democratic, Republican, or Independent). To determine if gender affects voting choice, we may apply a chi-square test evaluating independence.

In [3]:

```
# Python program to perform a chi-square test

# Importing the required modules
from scipy.stats import chi2_contingency

# defining our data
data = [[231, 256, 321],
        [245, 312, 213]]

# Performing chi-square test
test, p_val, dof, expected_val = chi2_contingency(data)

# interpreting the p-value
alpha = 0.05
print("The p-value of our test is " + str(p_val))

# Checking the hypothesis
if p_val <= alpha:
    print('We can reject the null hypothesis')
else:
    print('We can accept the null hypothesis')
```

```
The p-value of our test is 1.4585823594475804e-06
We can reject the null hypothesis
```

Data Distribution¶

In the real world, the data sets are much bigger, but it can be difficult to gather real world data, at least at an early stage of a project.

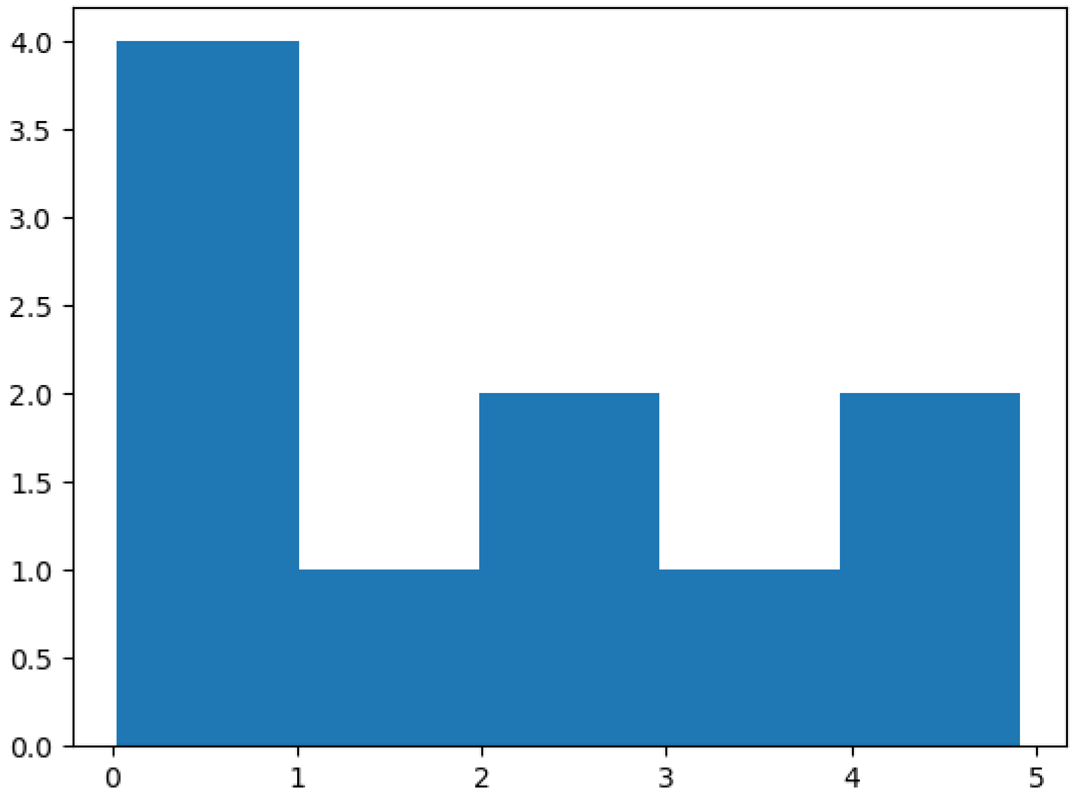
In [12]:

```
#Create an array containing 250 random floats between 0 and 5:

import numpy

x = numpy.random.uniform(0.0, 5.0, 10)

print(x)
```

```
[2.20053229 4.41315    1.8377668  4.9542851  4.33122068 4.71939017
 0.83557306 3.33115735 4.00803386 2.58628556]
```

In [15]:

```
import numpy
import matplotlib.pyplot as plt

x = numpy.random.uniform(0.0, 5.0, 10)

plt.hist(x, 5)
plt.show()
```

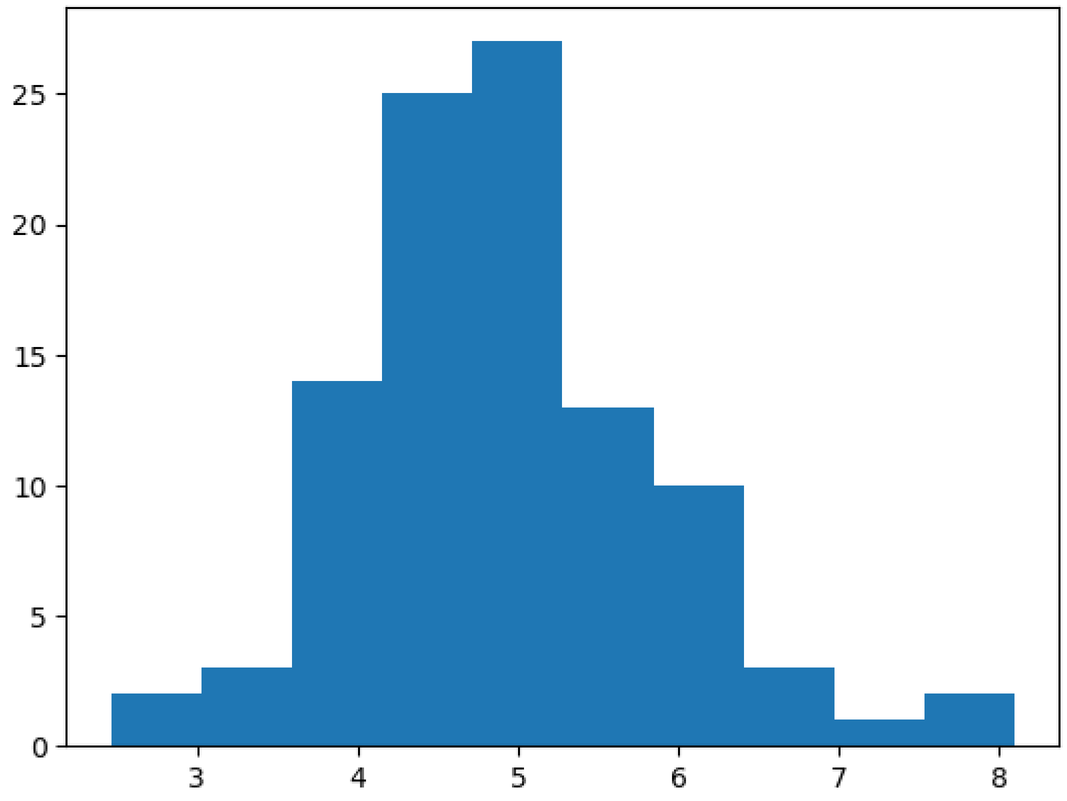


Normal Data Distribution

In probability theory this kind of data distribution is known as the normal data distribution, or the Gaussian data distribution, after the mathematician Carl Friedrich Gauss who came up with the formula of this data distribution.

In [19]:

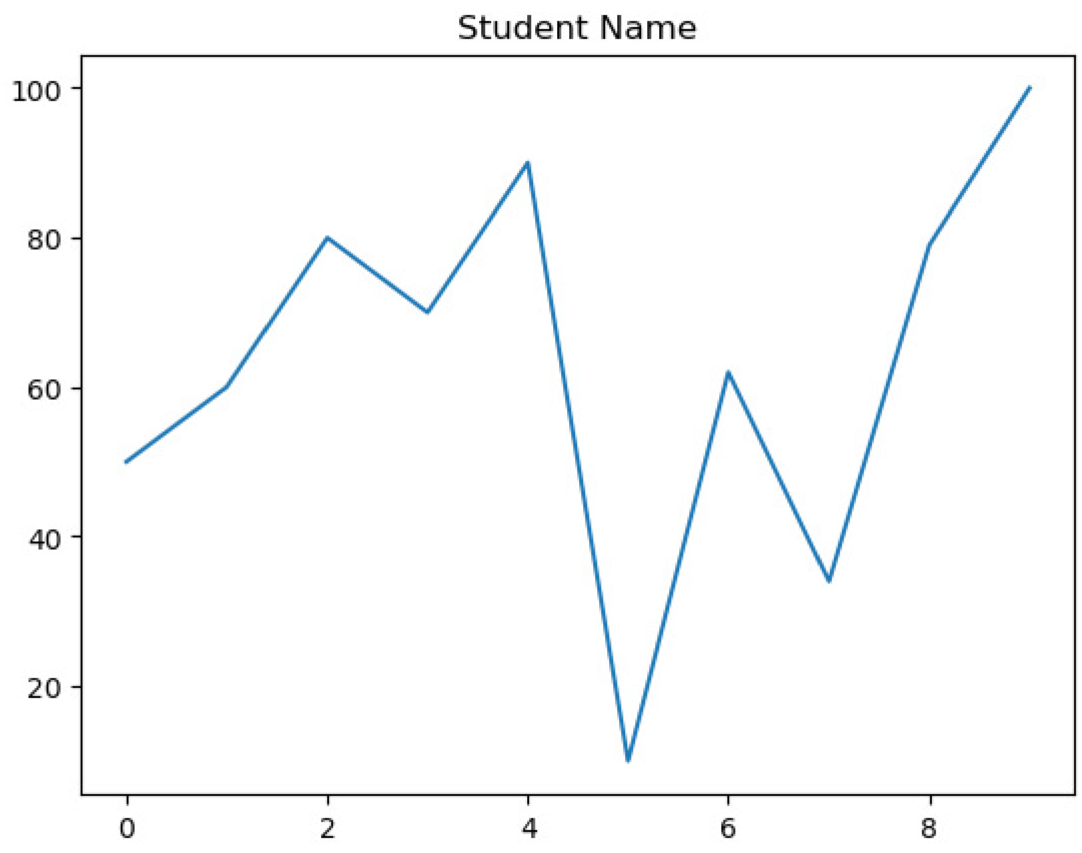
```
import numpy
import matplotlib.pyplot as plt

x = numpy.random.normal(5.0, 1.0, 100)

plt.hist(x, 10)
plt.show()
```



Data Visualization using matplotlib in python

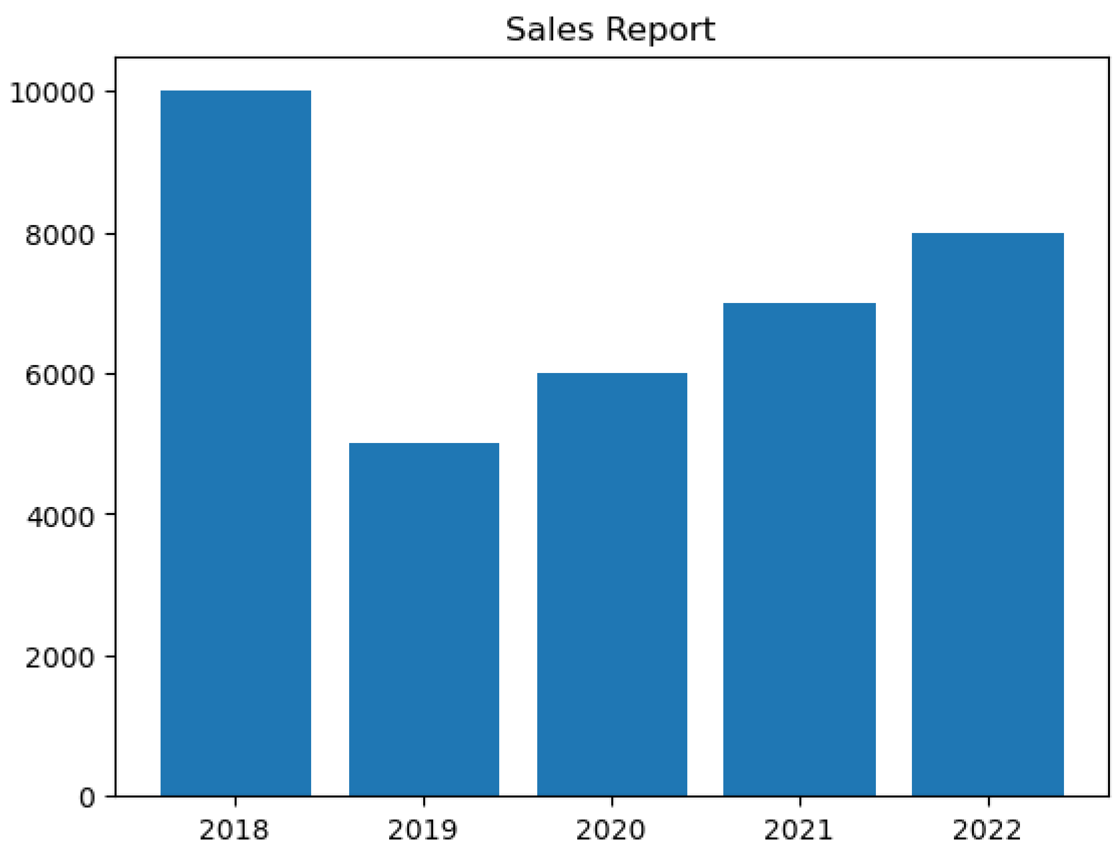
In [25]:

```
import numpy as np
import matplotlib.pyplot as plt

#line chart
data=np.array([50,60,80,70,90,10,62,34,79,100])
plt.plot(data)
plt.title("Student Name")
plt.show()
```



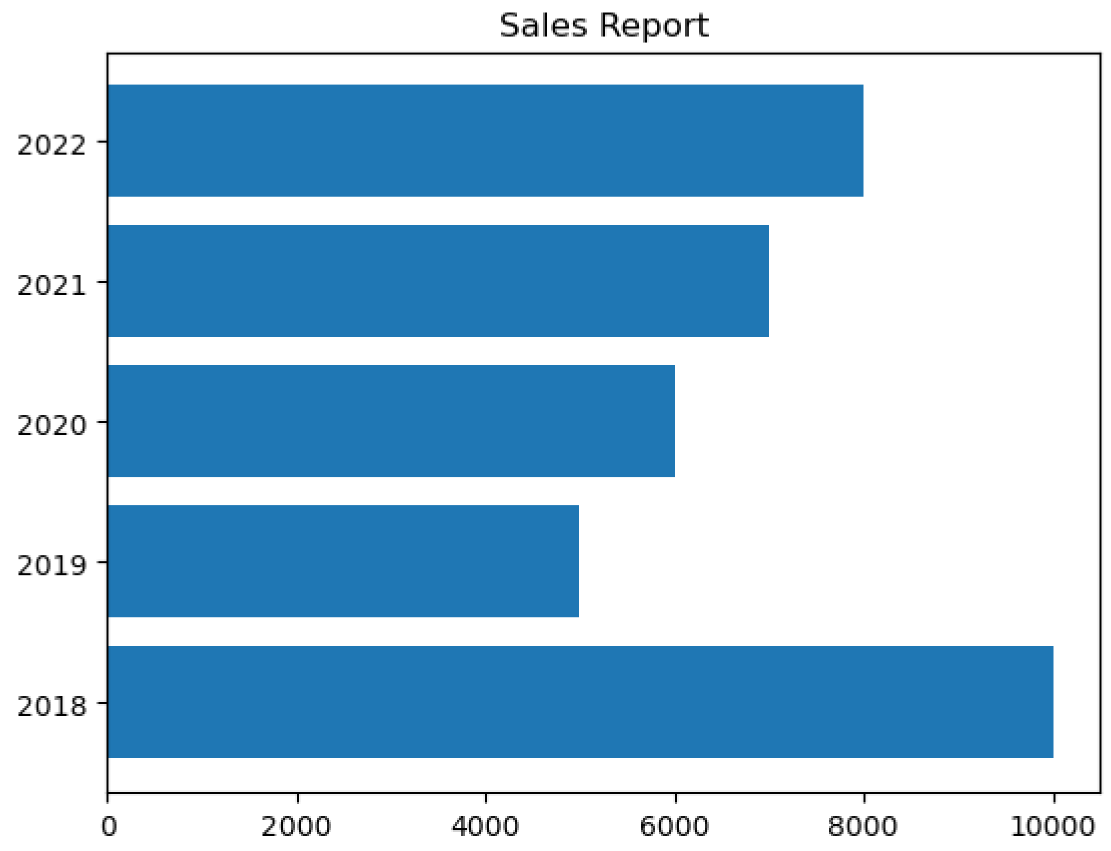
In [29]:

```
import numpy as np
import matplotlib.pyplot as plt

year=np.array([2018,2019,2020,2021,2022])
sales=np.array([10000,5000,6000,7000,8000])
#column chart
plt.title("Sales Report")
plt.bar(year,sales)
plt.show()
```



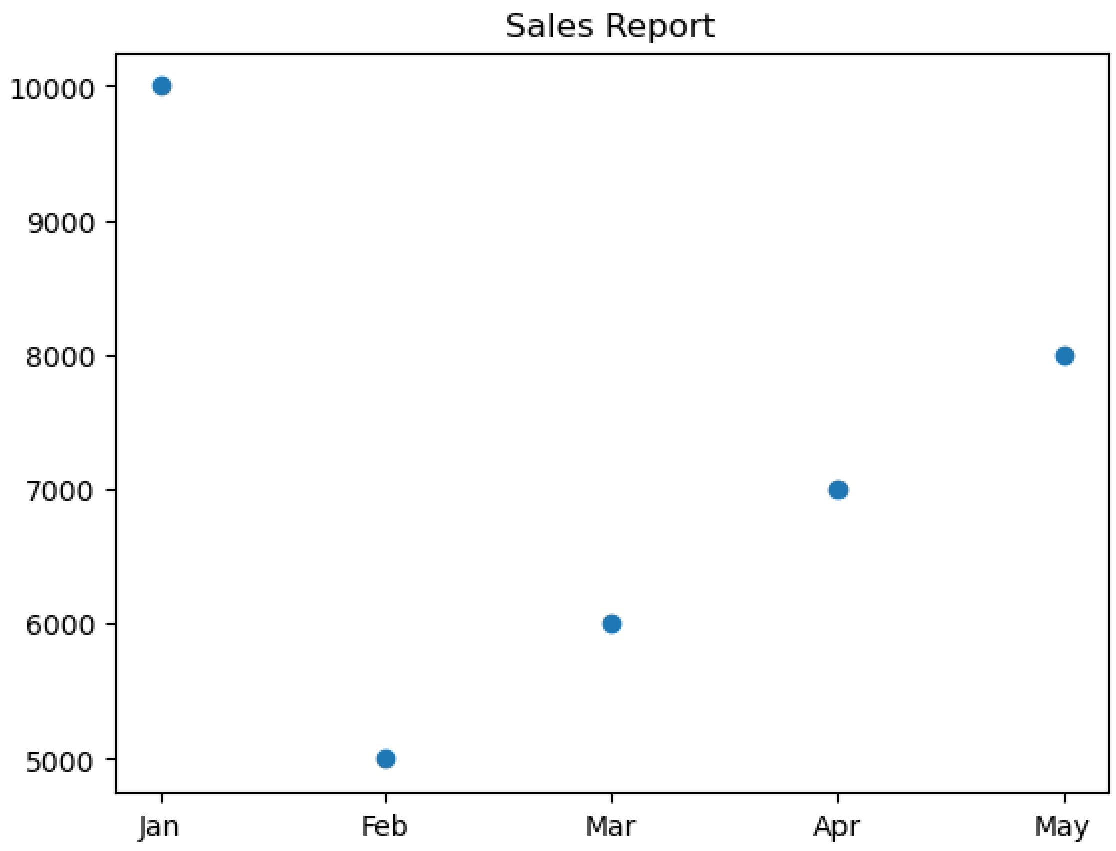
In [34]:

```
#bar chart
plt.title("Sales Report")
plt.barh(year,sales)
plt.show()
```



In [40]:

```
#scatter Chart
plt.title("Sales Report")
plt.scatter(['Jan','Feb','Mar','Apr','May'],sales)
plt.show()
```

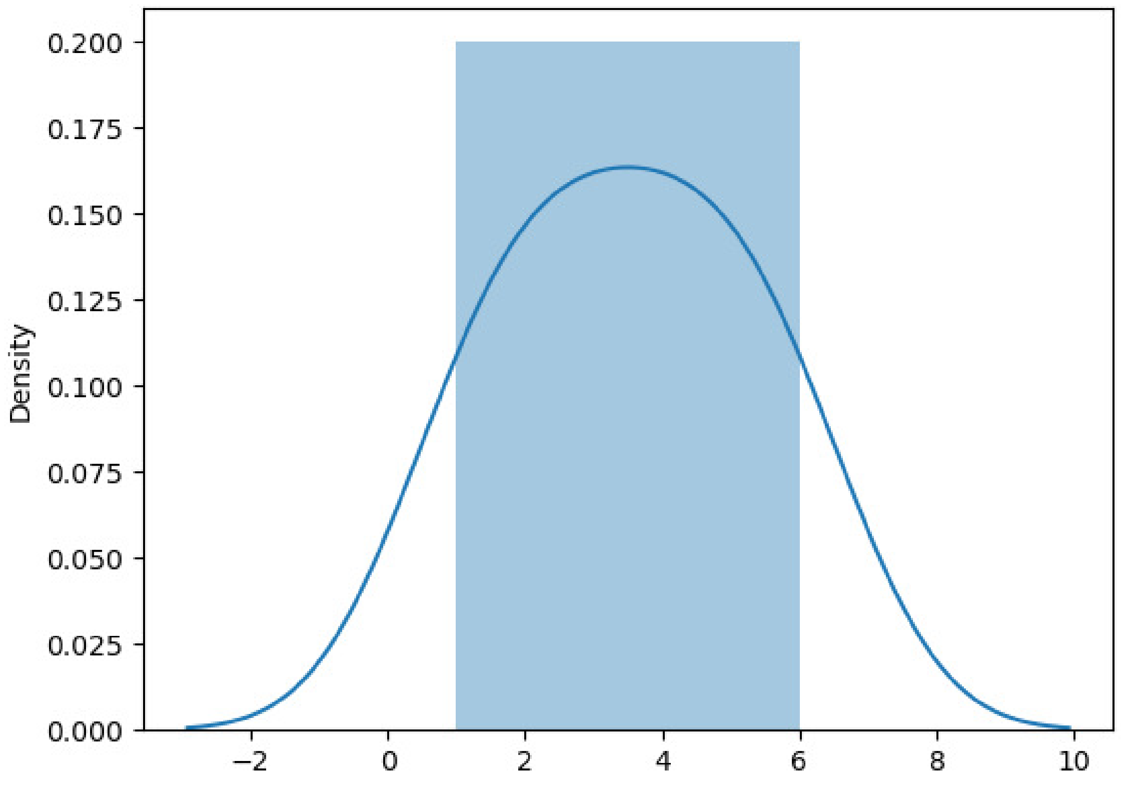


Data Visualization using seaborn in python

In [2]:

```
import matplotlib.pyplot as plt
import seaborn as sns

#dist chart with histogram
sns.distplot([1,2,3,4,5,6])
plt.show()
```

```
C:\Users\Sachin sirohi\AppData\Local\Temp\ipykernel_24340\1836585849.py:5:
UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.
0.

Please adapt your code to use either `displot` (a figure-level function wi
th
similar flexibility) or `histplot` (an axes-level function for histogram
s).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751 (https://
gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751)

  sns.distplot([1,2,3,4,5,6])
```

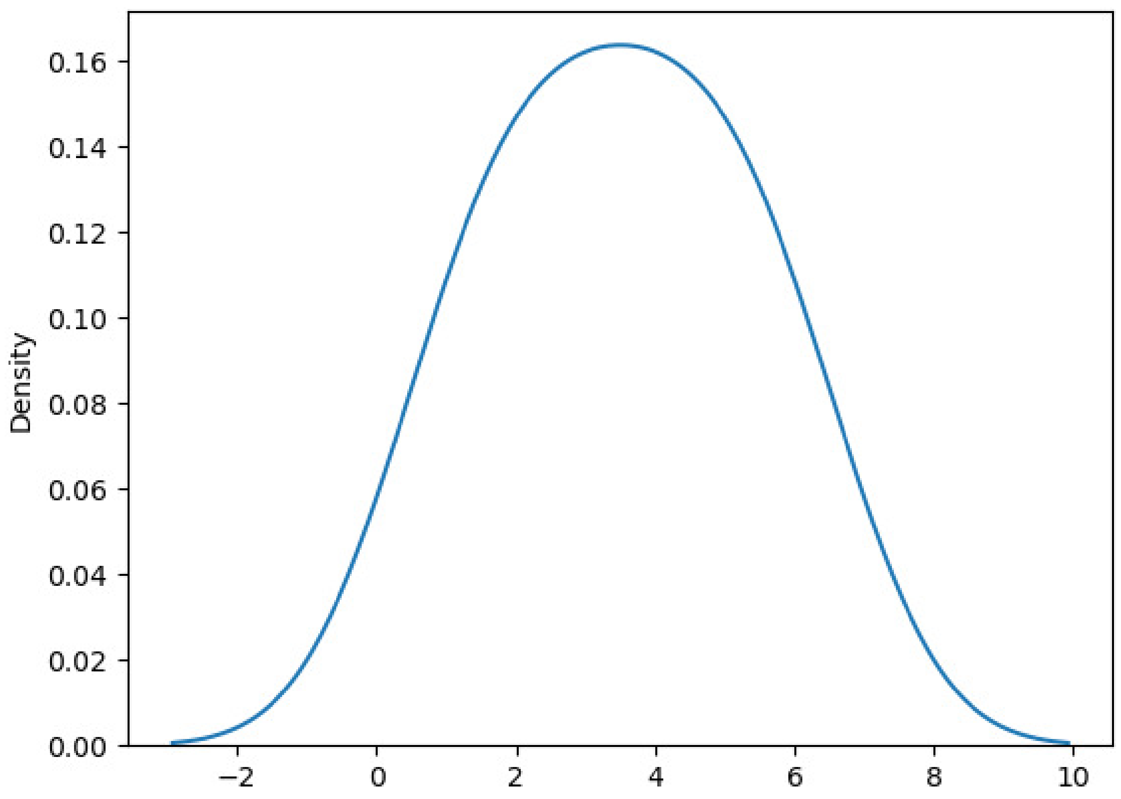


In [ ]:

In [3]:

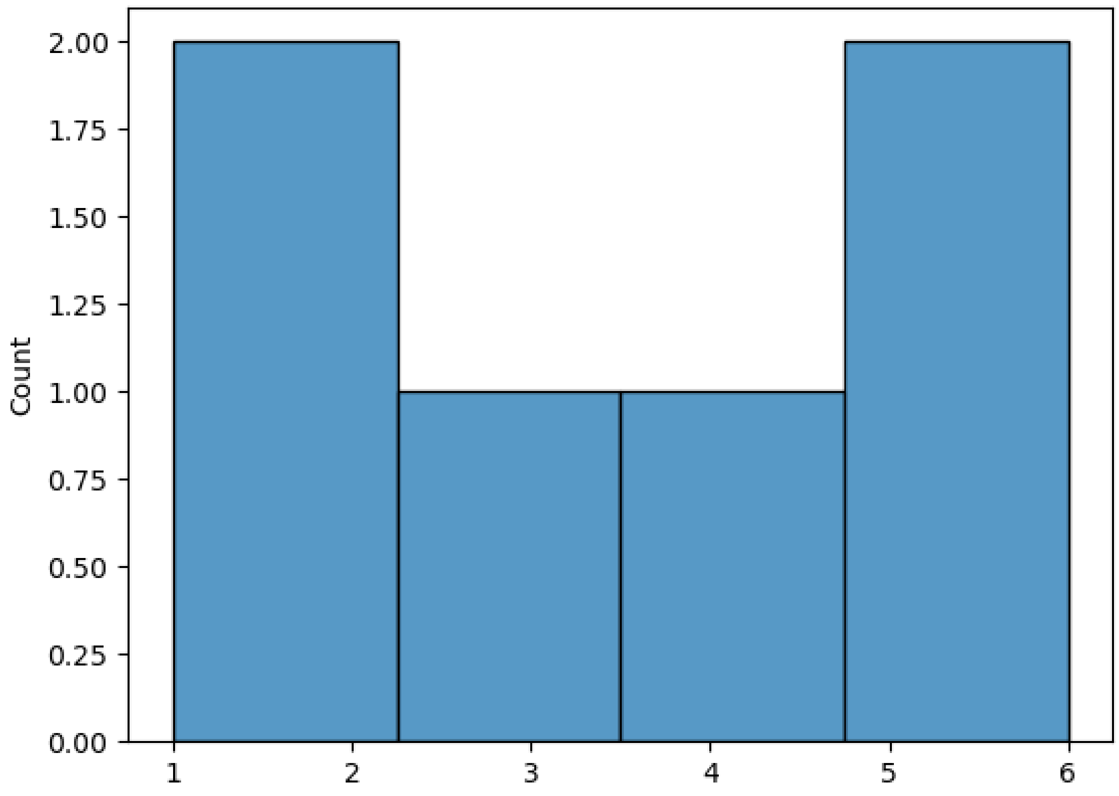
```
#distplot chart with histogram
sns.distplot([1,2,3,4,5,6],hist=False)
plt.show()
```

```
C:\Users\Sachin sirohi\AppData\Local\Temp\ipykernel_24340\3988205668.py:2:
UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.
0.

Please adapt your code to use either `displot` (a figure-level function wi
th
similar flexibility) or `kdeplot` (an axes-level function for kernel densi
ty plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751 (https://
gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751)

  sns.distplot([1,2,3,4,5,6],hist=False)
```



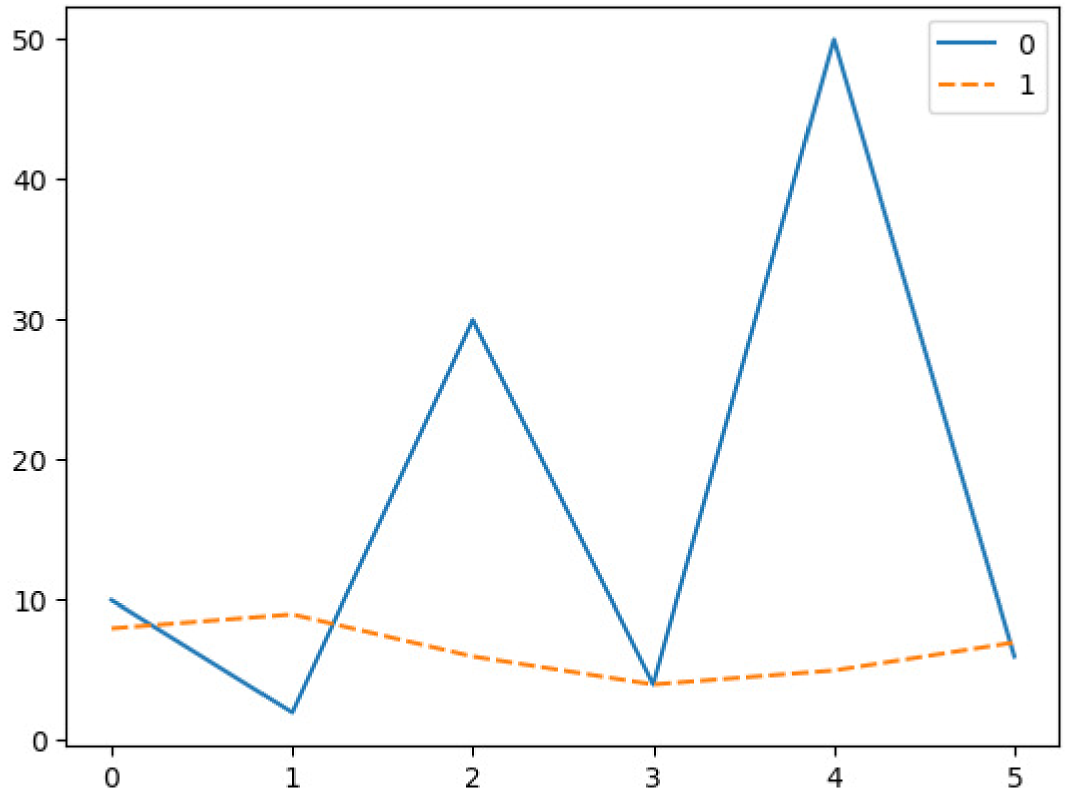
In [49]:

```
#histogram chart
sns.histplot([1,2,3,4,5,6])
plt.show()
```



In [4]:

```
#lineplot chart of seaborn library
sns.lineplot([[10,2,30,4,50,6],[8,9,6,4,5,7]])
plt.show()
```



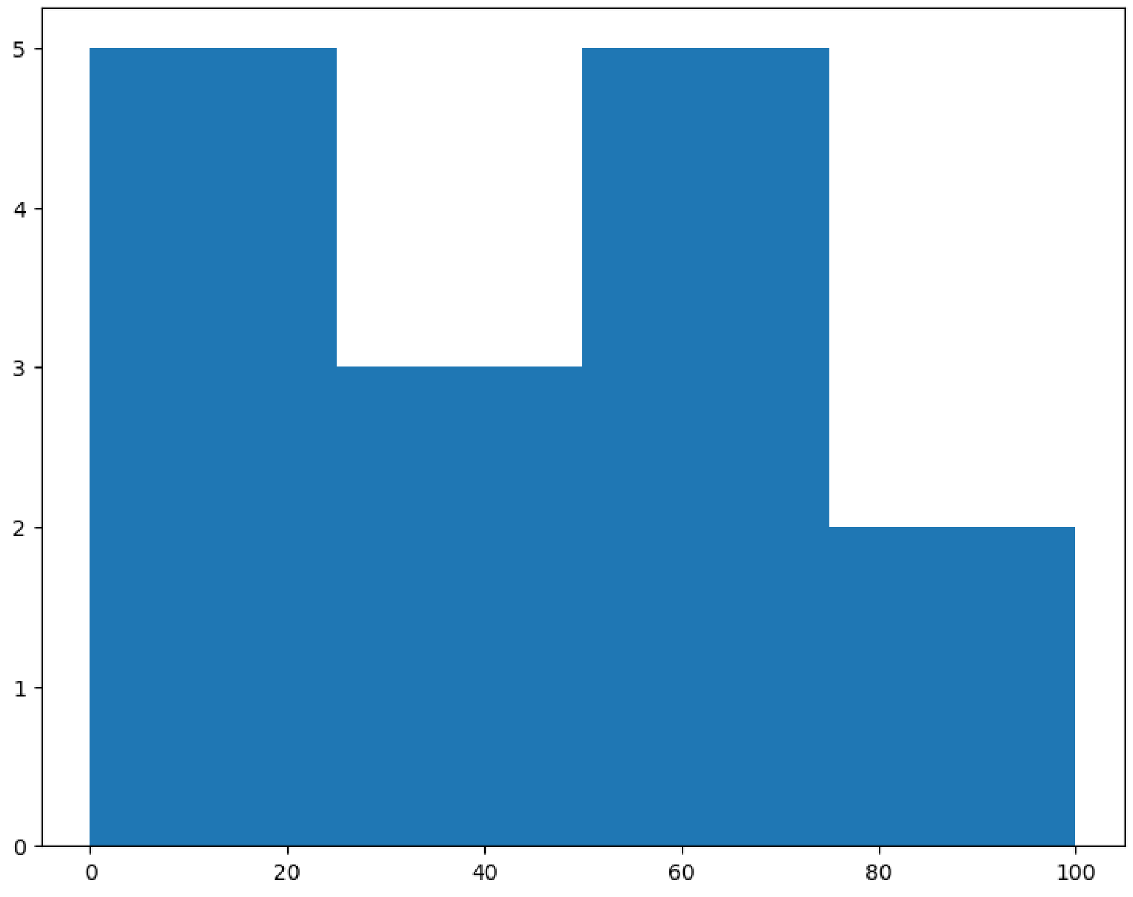
Histogram Chart

In [8]:

```
from matplotlib import pyplot as plt
import numpy as np
example=np.array([22,87,5,43,56,73,55,
                  54,11,20,51,5,79,31,27])
figure,ax=plt.subplots(figsize=(9,7))  #width, height
ax.hist(example,bins=[0,25,50,75,100])
```

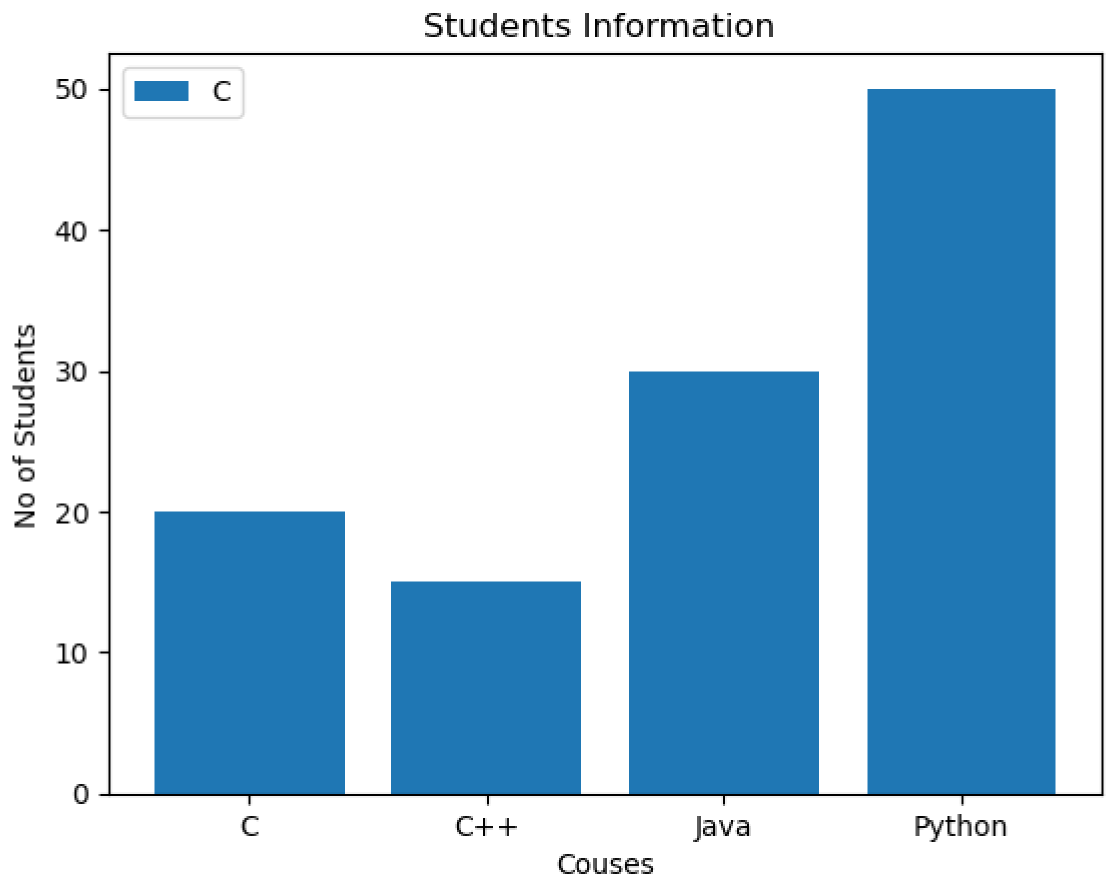
Out[8]:

```
(array([5., 3., 5., 2.]),
 array([  0,  25,  50,  75, 100]),
 <BarContainer object of 4 artists>)
```



In [32]:

```
from matplotlib import pyplot as plt
import numpy as np

course=np.array(['C','C++','Java','Python'])
nostd=np.array([20,15,30,50])
plt.bar(course,nostd)
plt.title("Students Information ")
plt.xlabel("Couses")
plt.ylabel("No of Students")
plt.legend(['C','C++','Java','Python'],loc="upper left")
plt.show()
```

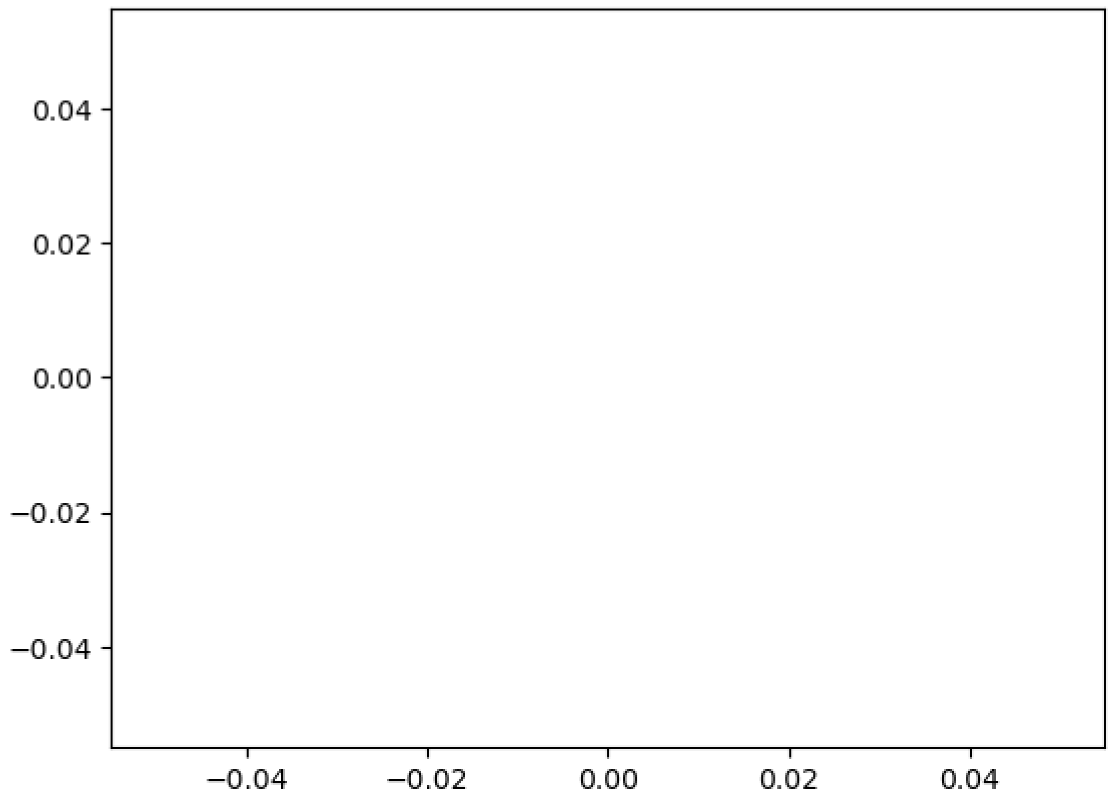


In [37]:

```
from matplotlib import pyplot as plt
import numpy as np

plt.plot()
```

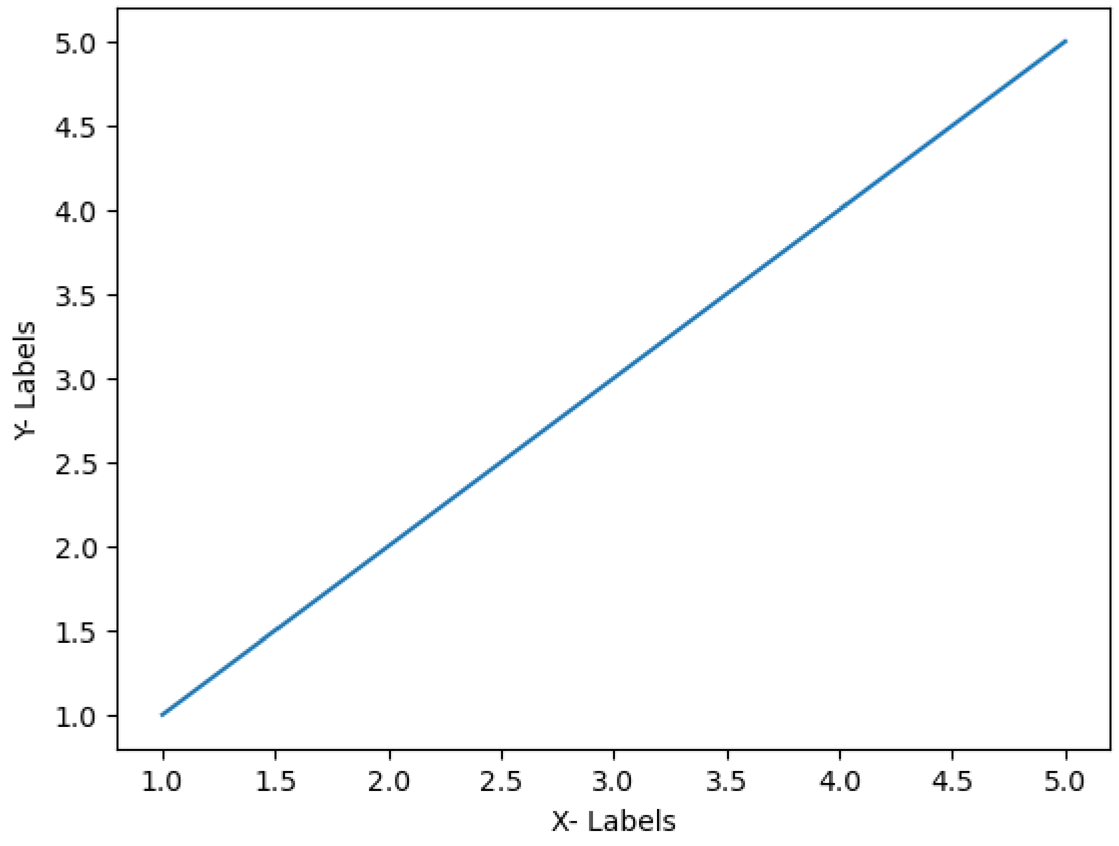
Out[37]:

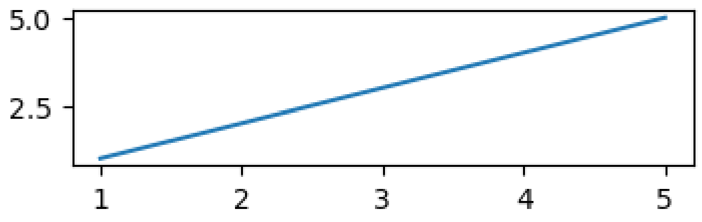
```
[]
```



In [45]:

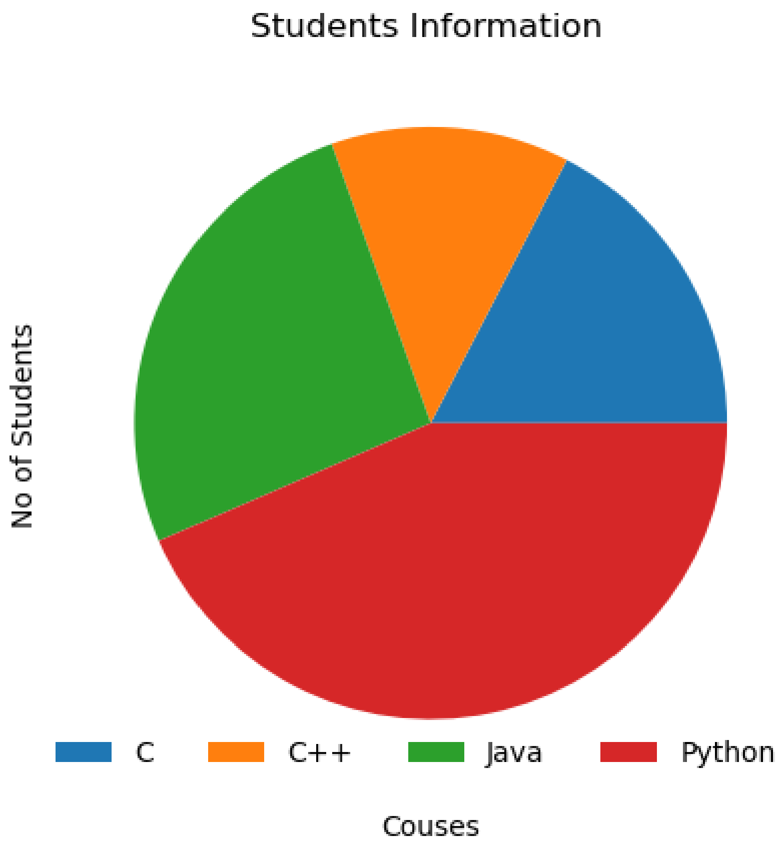
```
from matplotlib import pyplot as plt
import numpy as np

x=[1,2,3,4,5]
Y=[1,2,3,4,5]

plt.xlabel('X- Labels')
plt.ylabel('Y- Labels')

plt.plot(x,y)
plt.show()

f=plt.figure()
f.set_figwidth(4)
f.set_figheight(1)

plt.plot(x,y)
plt.show()
```





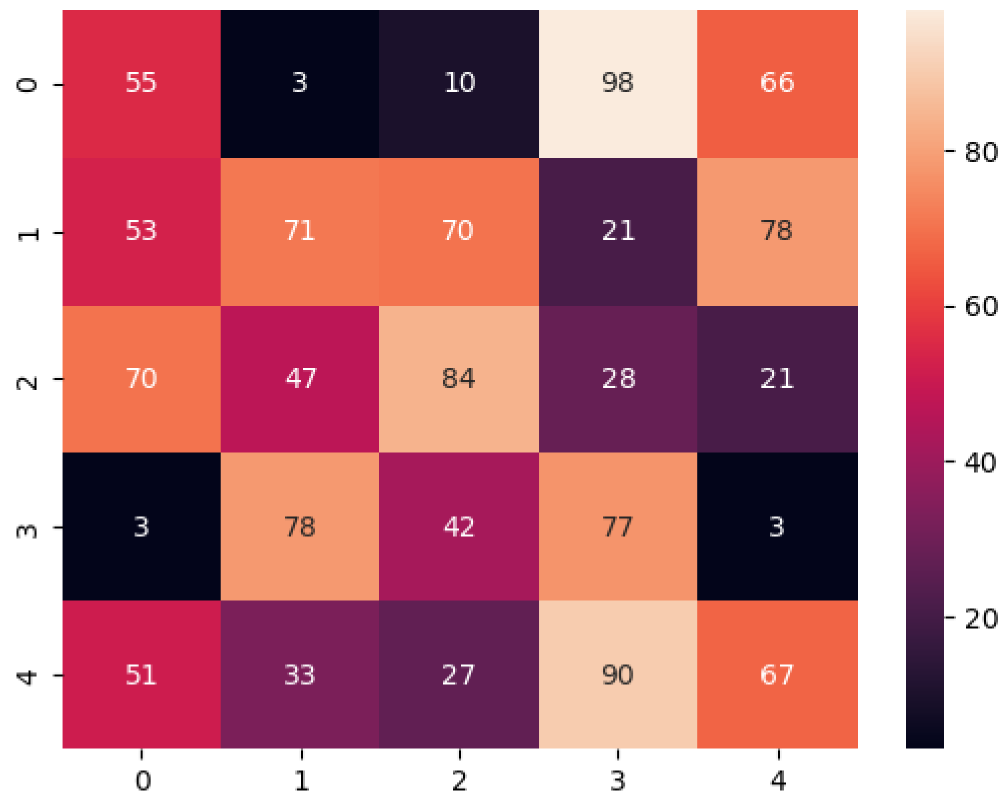
In [55]:

```
from matplotlib import pyplot as plt
import numpy as np

course=np.array(['C','C++','Java','Python'])
nostd=np.array([20,15,30,50])
plt.pie(nostd)
plt.title("Students Information ")
plt.xlabel("Couses")
plt.ylabel("No of Students")
plt.legend(['C','C++','Java','Python'],frameon=False,loc="lower right",ncol=4)
plt.show()
```



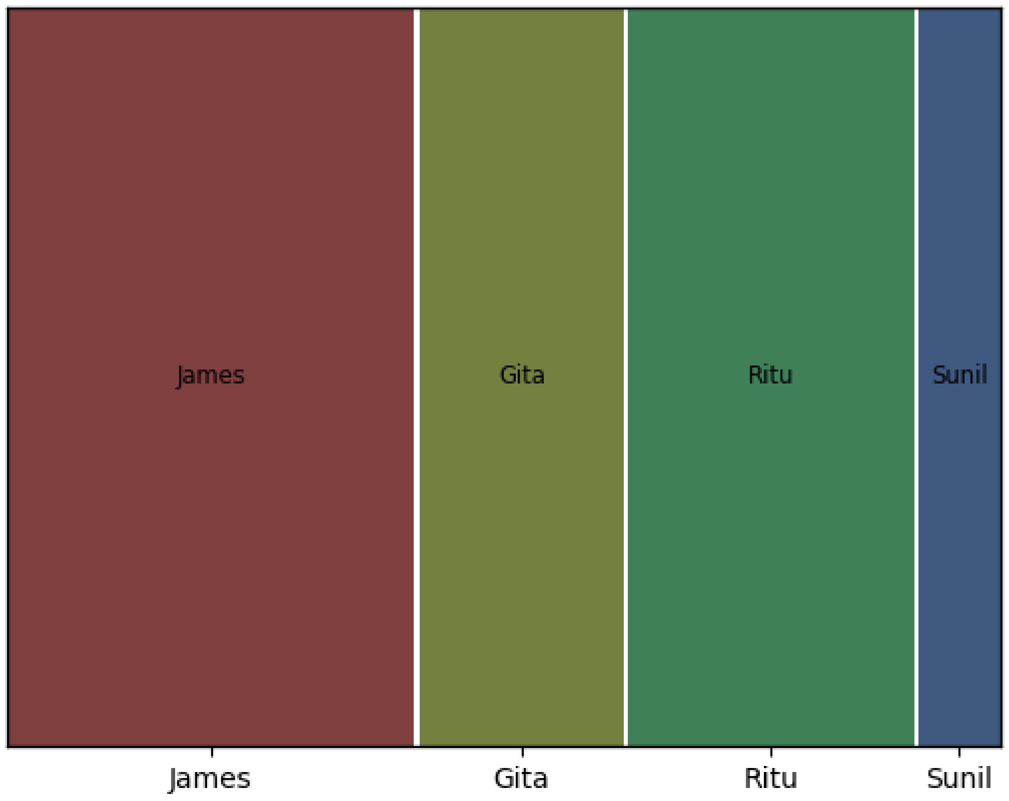
In [62]:

```
from matplotlib import pyplot as plt
import numpy as np
import seaborn as sn

R_data=np.random.randint(low=1,high=100,size=(5,5))
hm=sn.heatmap(data=R_data,annot=True) #annot use for pass/show parameter values into cha
plt.show()
```



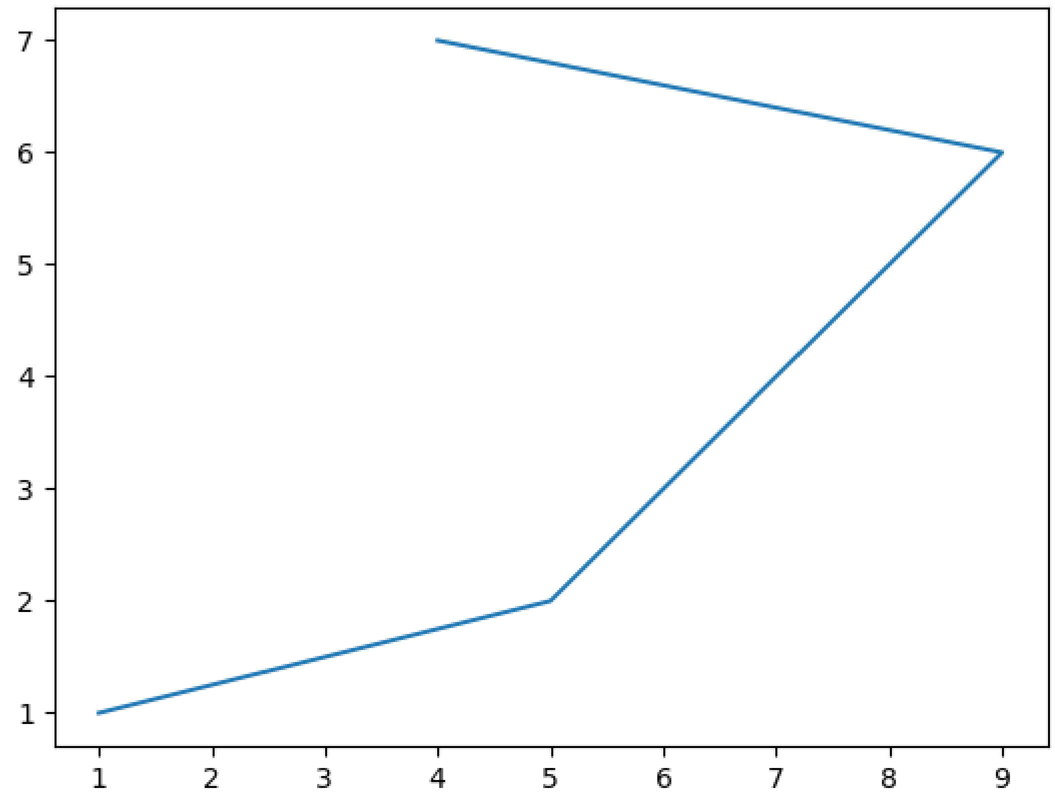
In [68]:

```
from matplotlib import pyplot as plt
import numpy as np
from statsmodels.graphics.mosaicplot import mosaic

data={'James':10,"Gita":5,"Ritu":7,"Sunil":2}
mosaic(data)
plt.show()
```



In [71]:

```
from matplotlib import pyplot as plt
x=[1,5,6,7,8,9,4]
y=[1,2,3,4,5,6,7]
plt.plot(x,y)
plt.savefig("LineChart.png") #use for save the chart
plt.show()
```



Regression Analysis

The term regression is used when you try to find the relationship between variables.

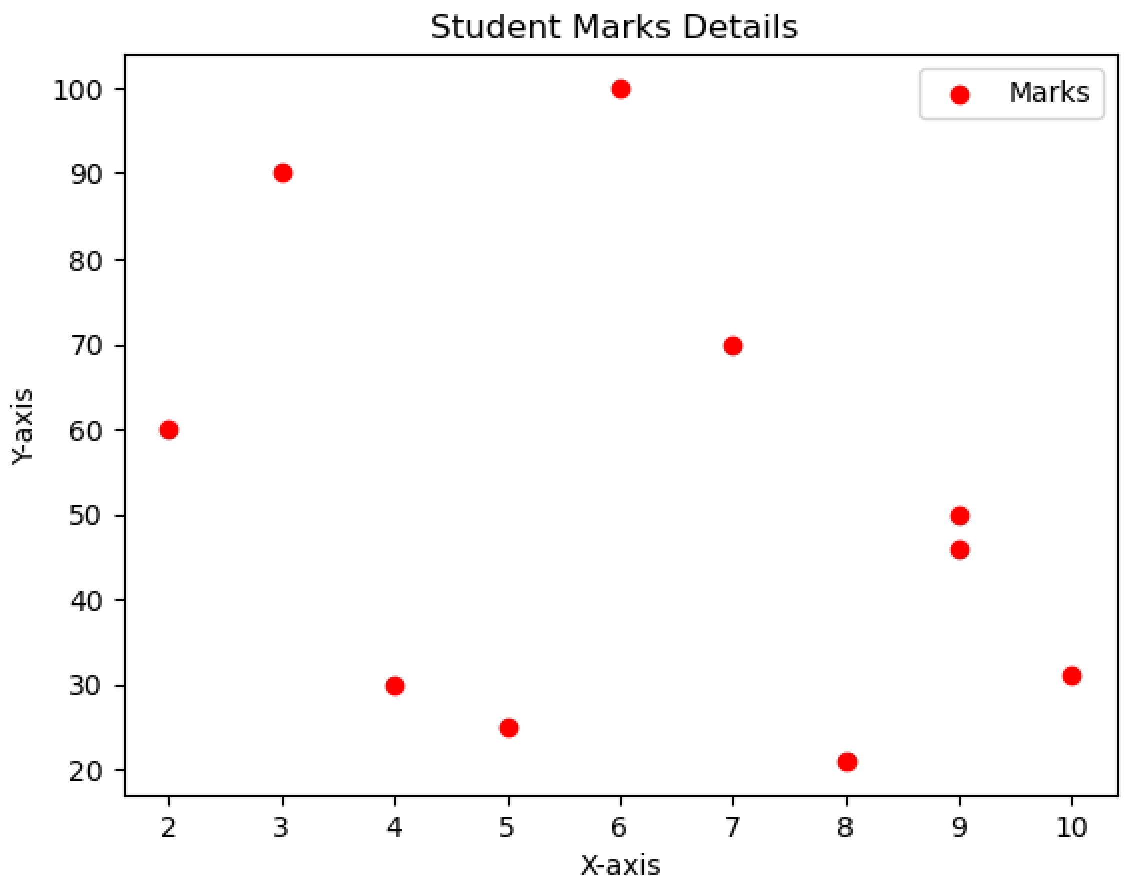
In Machine Learning, and in statistical modeling, that relationship is used to predict the outcome of future events.

Linear Regression Linear regression uses the relationship between the data-points to draw a straight line through all them.

This line can be used to predict future values.

In [73]:

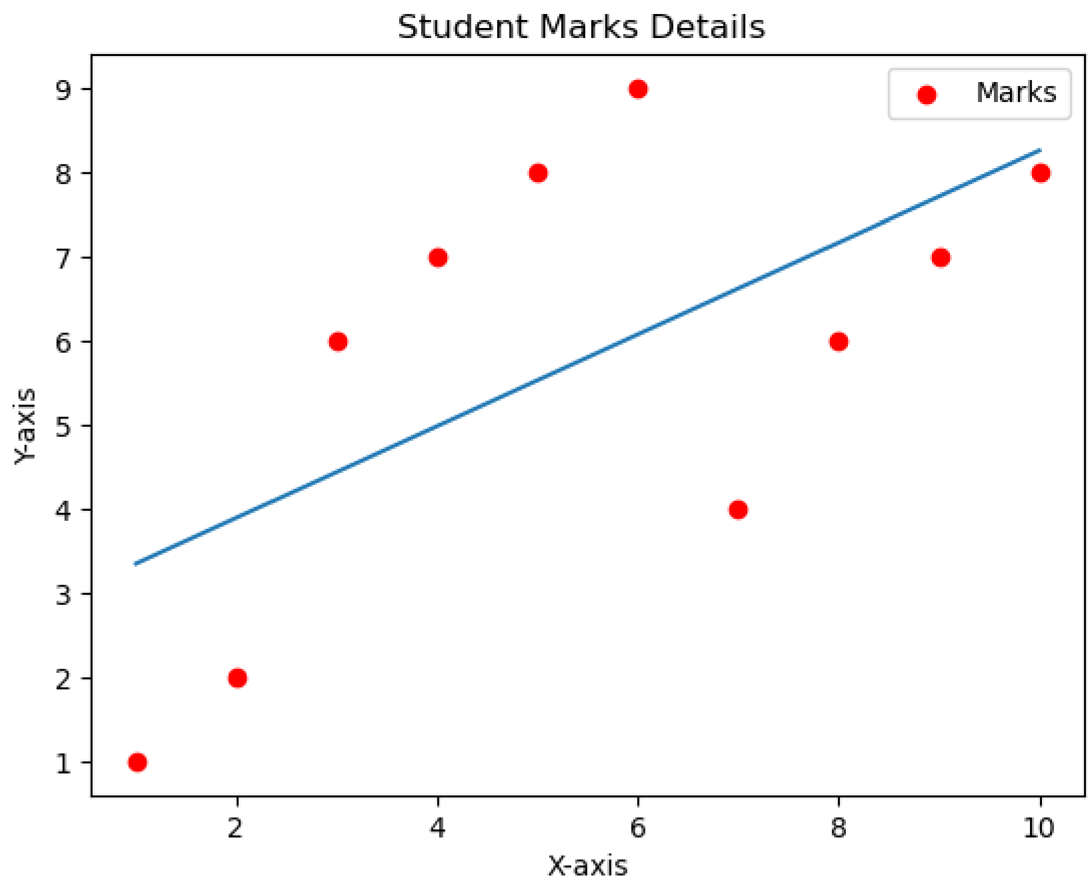
```
import numpy as np
import matplotlib.pyplot as plt

x=np.array([9,2,3,4,5,6,7,8,9,10])
y=np.array([50,60,90,30,25,100,70,21,46,31])

plt.scatter(x,y,c="red")
plt.title("Student Marks Details ")
plt.xlabel("X-axis")
plt.ylabel('Y-axis')
plt.legend(["Marks"])
plt.show()
```



Linear Regression using with Trend Line

In [82]:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

x=np.array([1,2,3,4,5,6,7,8,9,10])
y=np.array([1,2,6,7,8,9,4,6,7,8])

slope, intercept, r, p, std_err = stats.linregress(x, y)

def myfunc(x):
  return slope * x + intercept

mymodel = list(map(myfunc, x))

plt.scatter(x,y,c="red")
plt.plot(x,mymodel)
plt.title("Student Marks Details ")
plt.xlabel("X-axis")
plt.ylabel('Y-axis')
plt.legend(["Marks"])
plt.show()
```



In [76]:

```
#Simple function
def addition(x):
    print("this is addition ",x)
addition(50)
```

```
this is addition  50
```

Random Forest

is an ensemble learning technique used for both classification and regression problems. In this technique, multiple decision trees are created and their output is averaged to give the final result. Random Forest Regression is known to produce very robust results by avoiding overfitting.

In [18]:

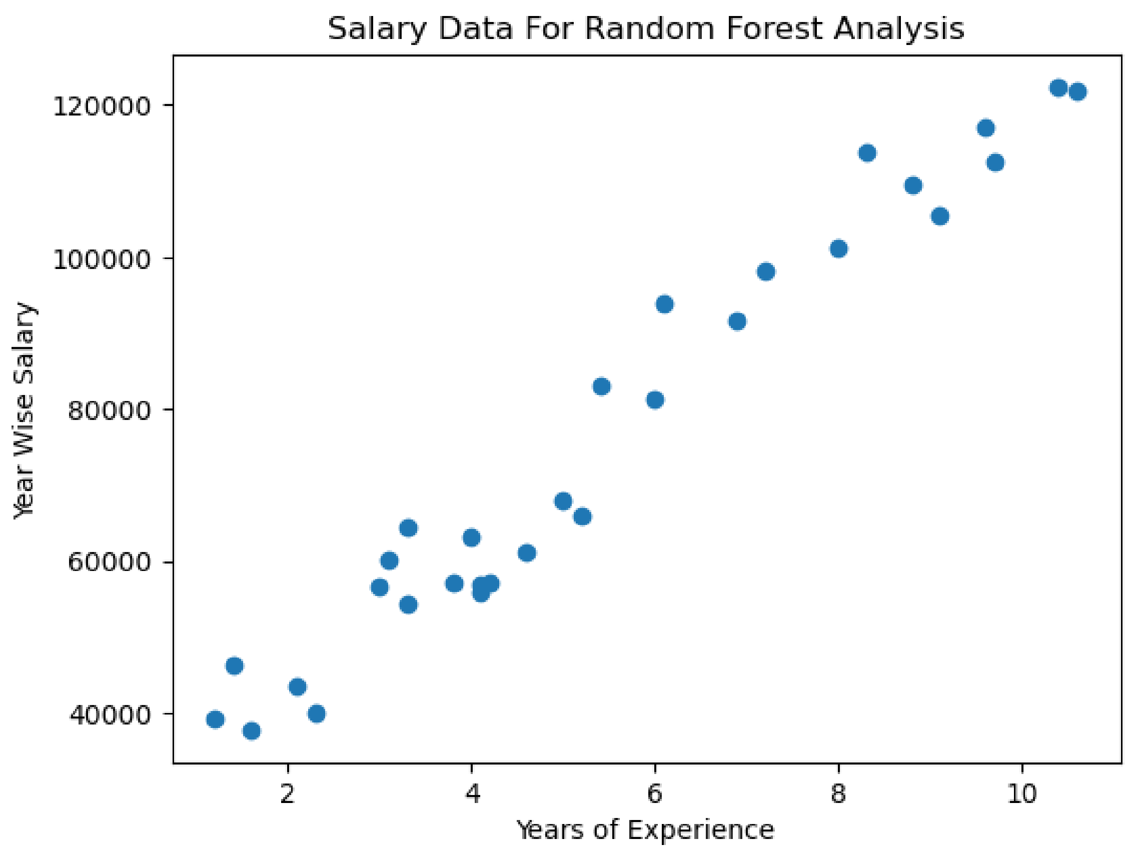
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from sklearn.model_selection import GridSearchCV

from sklearn.ensemble import RandomForestRegressor

df = pd.read_csv('salarydataset.csv')
df
```

Out[18]:

	index	YearsExperience	Salary
0	0	1.2	39344
1	1	1.4	46206
2	2	1.6	37732
3	3	2.1	43526
4	4	2.3	39892
5	5	3.0	56643
6	6	3.1	60151
7	7	3.3	54446
8	8	3.3	64446
9	9	3.8	57190
10	10	4.0	63219
11	11	4.1	55795
12	12	4.1	56958
13	13	4.2	57082
14	14	4.6	61112
15	15	5.0	67939
16	16	5.2	66030
17	17	5.4	83089
18	18	6.0	81364
19	19	6.1	93941
20	20	6.9	91739
21	21	7.2	98274
22	22	8.0	101303
23	23	8.3	113813
24	24	8.8	109432
25	25	9.1	105583
26	26	9.6	116970
27	27	9.7	112636
28	28	10.4	122392
29	29	10.6	121873

In [22]:

```
plt.scatter(x = df['YearsExperience'],y = df['Salary'])
plt.xlabel("Years of Experience")
plt.ylabel("Year Wise Salary")
plt.title("Salary Data For Random Forest Analysis")
plt.show()
```



Splitting the Dataset into Train & Test Dataset

In this section, we are first creating a dataframe of independent variable X and dependent variable y from the original dataframe df. Then we use train_test_split module to randomly create train and test datasets with an 80-20% split.

In [25]:

```
X = df.iloc[:, :-1]
y = df.iloc[:, -1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state
```

Training the RandomForestRegressor

Now we are creating an object of RandomForestRegressor with n_estimators = 10 i.e. with 10 decision trees. And then we fit this object over X_train and y_train for training the model.

In [26]:

```
rf_regressor = RandomForestRegressor(n_estimators = 10, random_state = 0)
rf_regressor.fit(X_train, y_train)
```

Out[26]:

```
RandomForestRegressor(n_estimators=10, random_state=0)
```

Training Accuracy

Here we use the R2 score to calculate the training accuracy which turns out to be 98.1% which is quite impressive.

In [27]:

```
y_pred_train = rf_regressor.predict(X_train)
r2_score(y_train, y_pred_train)
```

Out[27]:

```
0.9858127841768993
```

Training Accuracy

In [28]:

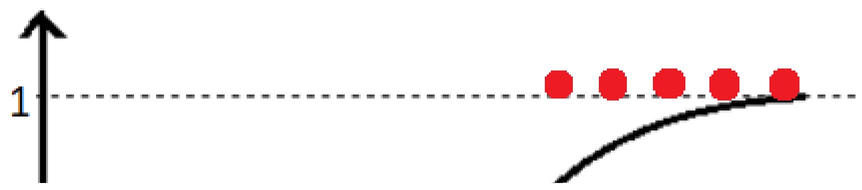
```
y_pred = rf_regressor.predict(X_test)
r2_score(y_test, y_pred)
```

Out[28]:

```
0.9553714788145329
```

Logistic Regression

Logistic regression aims to solve classification problems. It does this by predicting categorical outcomes, unlike linear regression that predicts a continuous



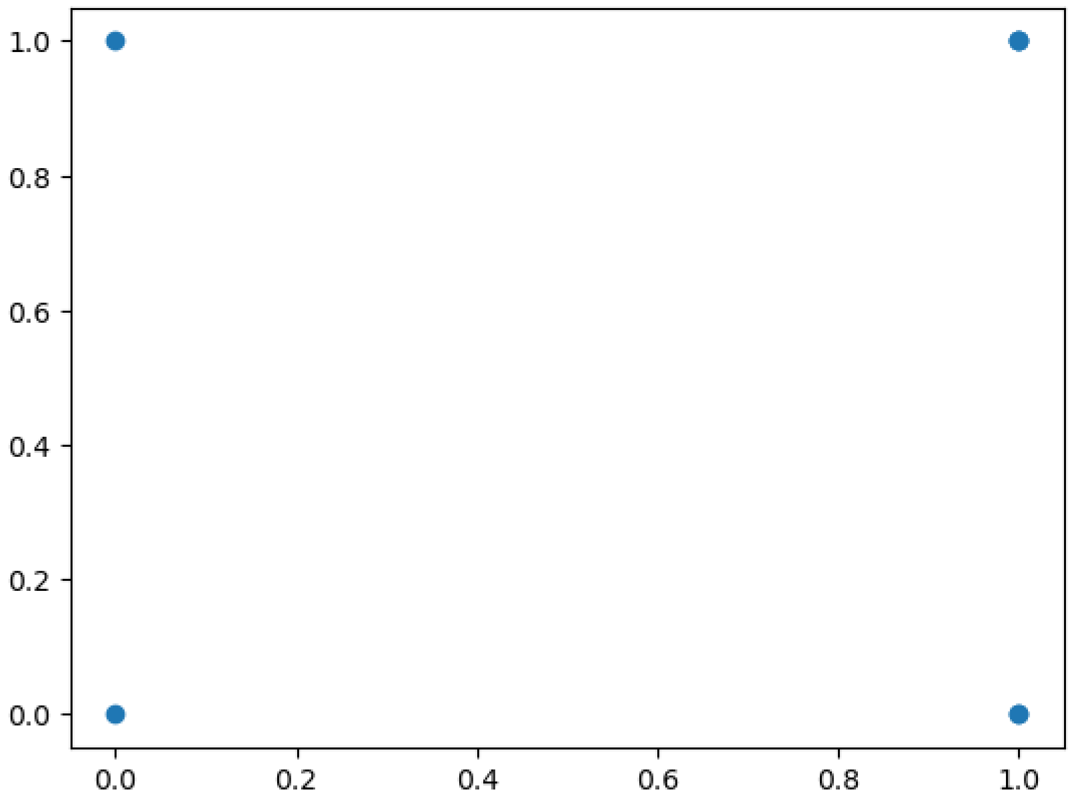
In [61]:

```
import numpy as np
from sklearn.datasets import load_iris
# from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# data={"data":[.5,1,.3,.5, 1,.4, 0,.2],"target":[0, 0, 0, 0,1,1,0,1 ]}
# data
# # Splitting the independent and dependent variables
# X = data["data"]
# Y = data["target"]
# Loading our dataset
data = load_iris()
data
# Splitting the independent and dependent variables
X = data.data
Y = data.target
print(X)
print(Y)
print("The size of the complete X dataset is: ", len(X))
print("The size of the complete Y dataset is: ", len(Y))
# Creating an instance of the LogisticRegression class for implementing logistic regress
log_reg = LogisticRegression()
# Segregating the training and testing dataset
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state =
# Performing the logistic regression on train dataset
log_reg.fit(X_train, Y_train)
# Printing the accuracy score
print("Accuracy score of the predictions made by the model: ", accuracy_score(log_reg.pr
```

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.1 3.5 1.4 0.3]
 [5.7 3.8 1.7 0.3]
```

In [49]:

```
plt.scatter(x=[0,1,1,0,1,1,1],y=[1,0,1,0,1,1,0])
plt.show()
```



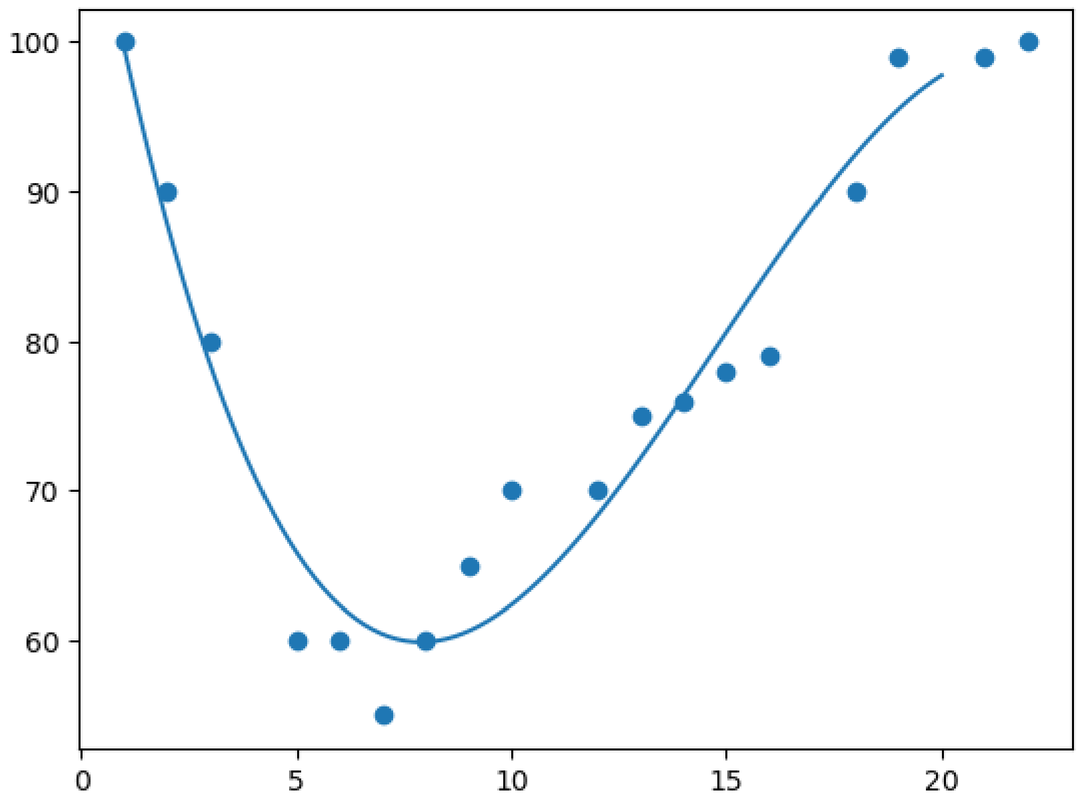
Polynomial Regression

If your data points clearly will not fit a linear regression (a straight line through all data points), it might be ideal for polynomial regression.

Polynomial regression, like linear regression, uses the relationship between the variables x and y to find the best way to draw a line through the data points.

In [72]:

```
import numpy
import matplotlib.pyplot as plt

x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]

mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))

myline = numpy.linspace(1, 20, 100)

plt.scatter(x, y)
plt.plot(myline, mymodel(myline))
plt.show()
```



Load Data World Cookies Data set

In [76]:

```python
import pandas as pd
df = pd.read_csv('https://query.data.world/s/ffmzyttox3oovahpf35iytn3cyfl55?dws=00000')
df
```

Out[76]:

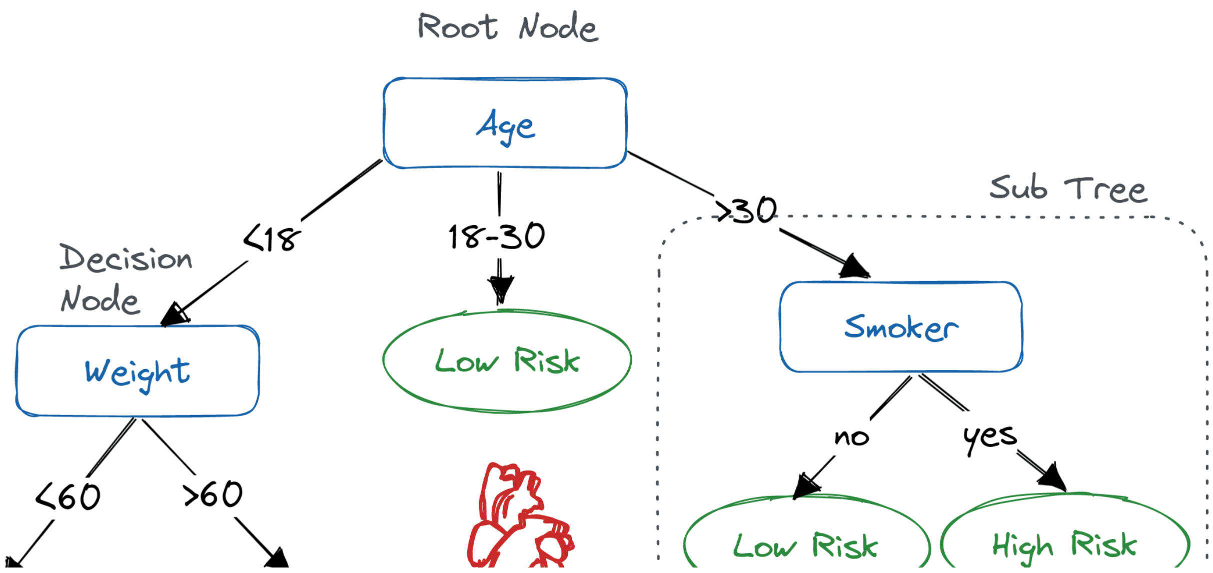
	Country	Country Name	Performance Oriented	Autocratic	Modesty	Charismatic 3: Self-sacrifice	Team 1: Collaborative Team Orientation	Decisiv
0	2	England	6.38	2.55	4.91	4.90	5.33	6.0
1	3	Costa Rica	6.15	2.46	5.48	5.67	5.74	5.6
2	5	Italy	6.18	2.64	4.67	5.20	5.53	6.0
3	6	India	5.96	3.10	5.33	5.45	5.51	5.8
4	7	Namibia	6.16	2.58	5.10	4.79	5.46	6.1
...	...	...	...	...	...	...	...	
57	94	Germany (WEST)	6.11	1.95	4.61	4.87	5.05	5.7
58	96	Denmark	6.05	2.10	4.32	5.05	5.28	6.0
59	97	Georgia	5.94	2.64	5.56	4.91	5.63	6.0
60	98	Thailand	5.98	2.58	5.30	4.96	5.32	5.8
61	99	USA	6.46	2.03	5.24	5.16	5.38	5.9

62 rows × 30 columns

Machine Learning- Deep Learning Classification

Decision Tree Classifier

is one of the most powerful and popular algorithm. Decision-tree algorithm falls under the category of supervised learning algorithms. It works for both continuous as well as categorical output variables.



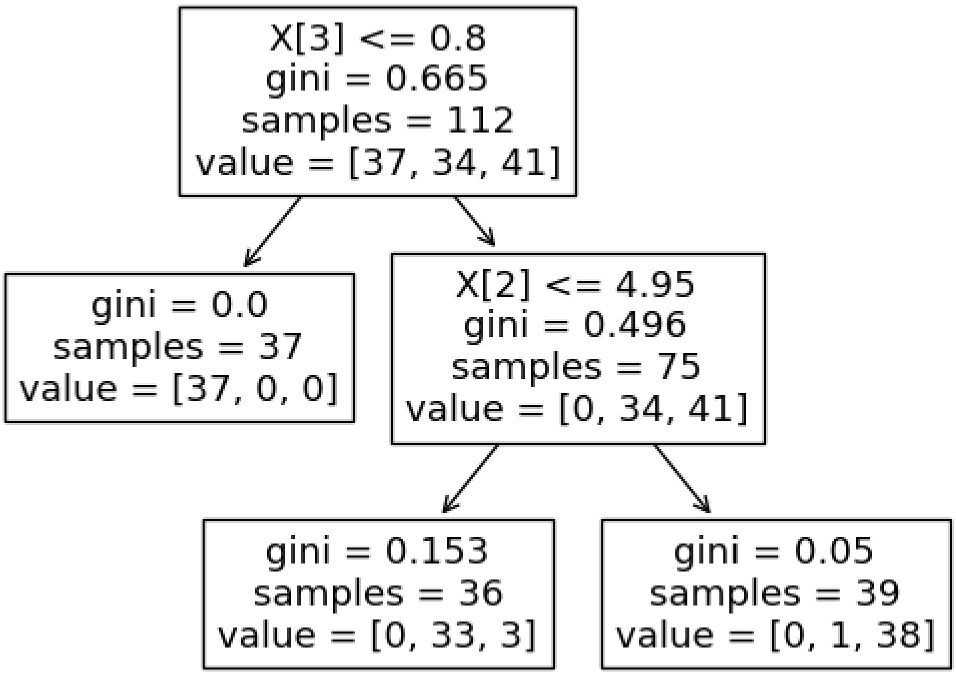
In [9]:

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
from sklearn import tree
from sklearn import metrics

iris=load_iris()
X=iris.data
Y=iris.target
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,random_state = 0)
clf=DecisionTreeClassifier(max_leaf_nodes=3,random_state=0)
# Train Decision Tree Classifer
clf.fit(X_train,Y_train)
#Predict the response for test dataset
y_pred = clf.predict(X_test)
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(Y_test, y_pred))
```

```
Accuracy: 0.8947368421052632
```

In [12]:

```
tree.plot_tree(clf)
plt.show()
```



Understanding the decision tree structure¶

The decision tree structure can be analysed to gain further insight on the relation between the features and the target to predict. In this example, we show how to retrieve:

the binary tree structure;

the depth of each node and whether or not it’s a leaf;

the nodes that were reached by a sample using the decision_path method;

the leaf that was reached by a sample using the apply method;

the rules that were used to predict a sample;

the decision path shared by a group of samples.

In [ ]:

```
import numpy as np
from matplotlib import pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
```

Train tree classifier

First, we fit a DecisionTreeClassifier using the load_iris dataset.

In [13]:

```
iris = load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

clf = DecisionTreeClassifier(max_leaf_nodes=3, random_state=0)
clf.fit(X_train, y_train)
```

Out[13]:

```
DecisionTreeClassifier(max_leaf_nodes=3, random_state=0)
```

Tree structure

The decision classifier has an attribute called tree_ which allows access to low level attributes such as node_count, the total number of nodes, and max_depth, the maximal depth of the tree. It also stores the entire binary tree structure, represented as a number of parallel arrays. The i-th element of each array holds information about the node i. Node 0 is the tree's root. Some of the arrays only apply to either leaves or split nodes. In this case the values of the nodes of the other type is arbitrary. For example, the arrays feature and threshold only apply to split nodes. The values for leaf nodes in these arrays are therefore arbitrary.

Among these arrays, we have:

children_left[i]: id of the left child of node i or -1 if leaf node

children_right[i]: id of the right child of node i or -1 if leaf node

feature[i]: feature used for splitting node i

threshold[i]: threshold value at node i

n_node_samples[i]: the number of training samples reaching node i

impurity[i]: the impurity at node i

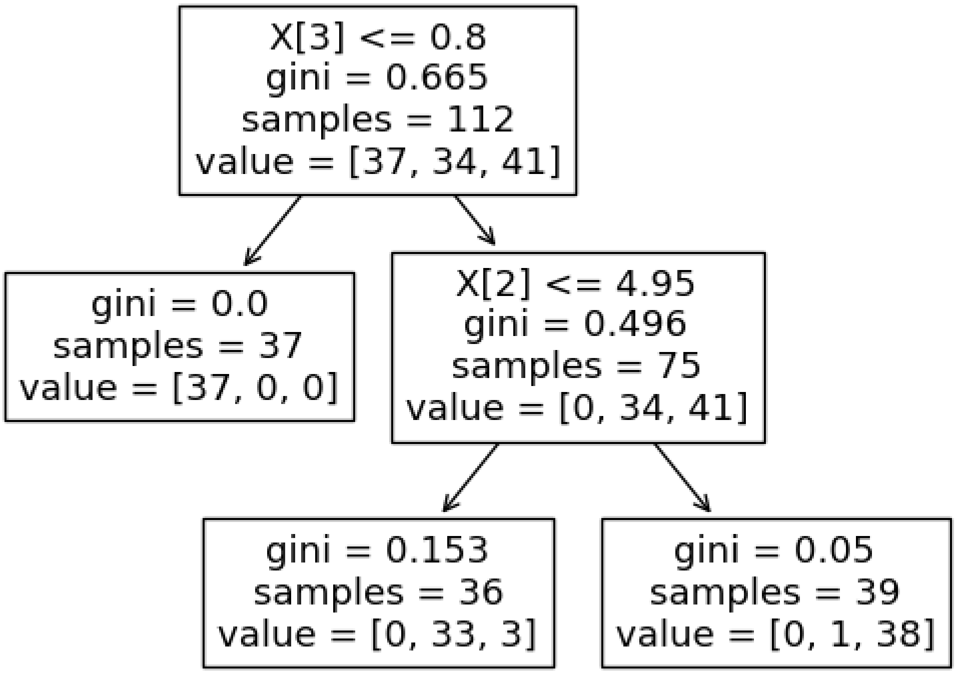
In [14]:

```python
n_nodes = clf.tree_.node_count
children_left = clf.tree_.children_left
children_right = clf.tree_.children_right
feature = clf.tree_.feature
threshold = clf.tree_.threshold

node_depth = np.zeros(shape=n_nodes, dtype=np.int64)
is_leaves = np.zeros(shape=n_nodes, dtype=bool)
stack = [(0, 0)]  # start with the root node id (0) and its depth (0)
while len(stack) > 0:
    # `pop` ensures each node is only visited once
    node_id, depth = stack.pop()
    node_depth[node_id] = depth

    # If the left and right child of a node is not the same we have a split
    # node
    is_split_node = children_left[node_id] != children_right[node_id]
    # If a split node, append left and right children and depth to `stack`
    # so we can loop through them
    if is_split_node:
        stack.append((children_left[node_id], depth + 1))
        stack.append((children_right[node_id], depth + 1))
    else:
        is_leaves[node_id] = True

print(
    "The binary tree structure has {n} nodes and has "
    "the following tree structure:\n".format(n=n_nodes)
)
for i in range(n_nodes):
    if is_leaves[i]:
        print(
            "{space}node={node} is a leaf node.".format(
                space=node_depth[i] * "\t", node=i
            )
        )
    else:
        print(
            "{space}node={node} is a split node: "
            "go to node {left} if X[:, {feature}] <= {threshold} "
            "else to node {right}.".format(
                space=node_depth[i] * "\t",
                node=i,
                left=children_left[i],
                feature=feature[i],
                threshold=threshold[i],
                right=children_right[i],
            )
        )
```

```
The binary tree structure has 5 nodes and has the following tree structur
e:

node=0 is a split node: go to node 1 if X[:, 3] <= 0.800000011920929 else
to node 2.
        node=1 is a leaf node.
        node=2 is a split node: go to node 3 if X[:, 2] <= 4.9500000476837
16 else to node 4.
                node=3 is a leaf node.
                node=4 is a leaf node.
```

In [15]:

```
tree.plot_tree(clf)
plt.show()
```



Decision path

We can also retrieve the decision path of samples of interest. The decision_path method outputs an indicator matrix that allows us to retrieve the nodes the samples of interest traverse through. A non zero element in the indicator matrix at position (i, j) indicates that the sample i goes through the node j. Or, for one sample i, the positions of the non zero elements in row i of the indicator matrix designate the ids of the nodes that sample goes through.

The leaf ids reached by samples of interest can be obtained with the apply method. This returns an array of the node ids of the leaves reached by each sample of interest. Using the leaf ids and the decision_path we can obtain the splitting conditions that were used to predict a sample or a group of samples. First, let's do it for one sample. Note that node_index is a sparse matrix.

In [16]:

```
node_indicator = clf.decision_path(X_test)
leaf_id = clf.apply(X_test)

sample_id = 0
# obtain ids of the nodes `sample_id` goes through, i.e., row `sample_id`
node_index = node_indicator.indices[
    node_indicator.indptr[sample_id] : node_indicator.indptr[sample_id + 1]
]

print("Rules used to predict sample {id}:\n".format(id=sample_id))
for node_id in node_index:
    # continue to the next node if it is a leaf node
    if leaf_id[sample_id] == node_id:
        continue

    # check if value of the split feature for sample 0 is below threshold
    if X_test[sample_id, feature[node_id]] <= threshold[node_id]:
        threshold_sign = "<="
    else:
        threshold_sign = ">"

    print(
        "decision node {node} : (X_test[{sample}, {feature}] = {value}) "
        "{inequality} {threshold})".format(
            node=node_id,
            sample=sample_id,
            feature=feature[node_id],
            value=X_test[sample_id, feature[node_id]],
            inequality=threshold_sign,
            threshold=threshold[node_id],
        )
    )
```

```
Rules used to predict sample 0:

decision node 0 : (X_test[0, 3] = 2.4) > 0.800000011920929)
decision node 2 : (X_test[0, 2] = 5.1) > 4.950000047683716)
```

For a group of samples, we can determine the common nodes the samples go through.

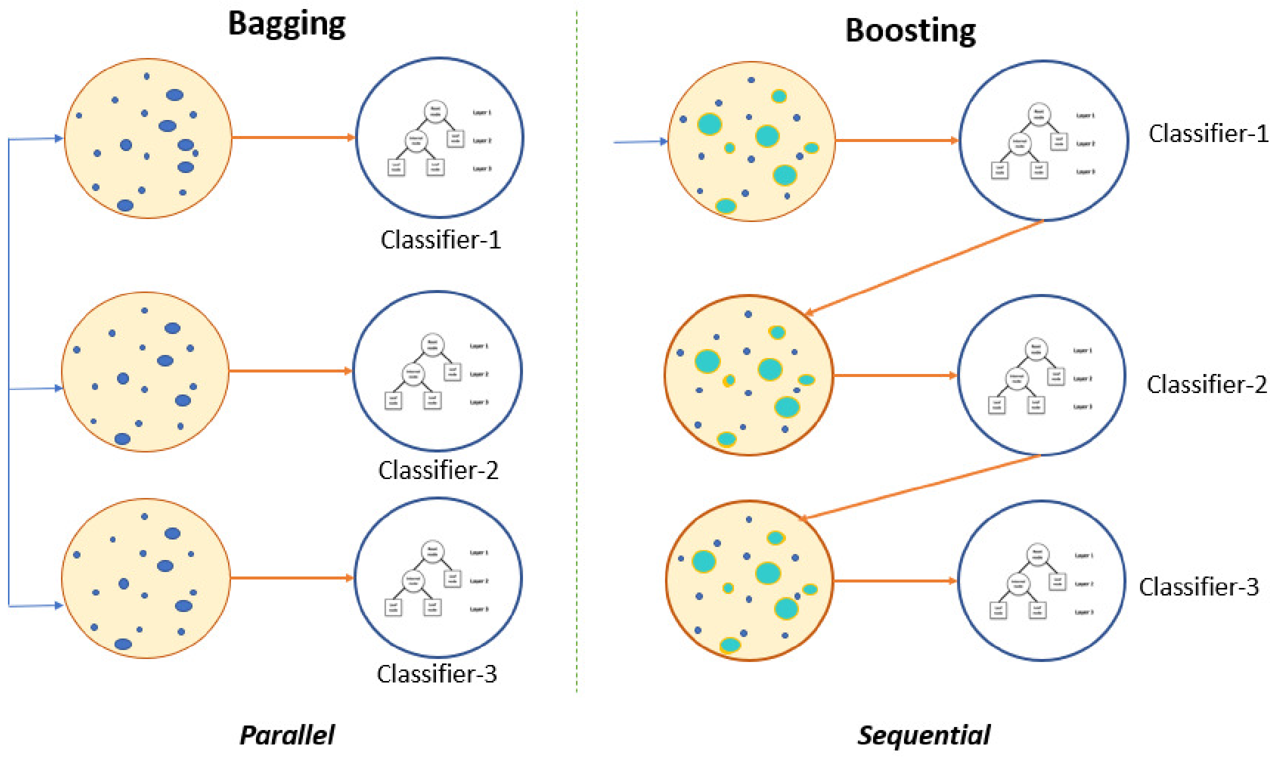
In [18]:

```
sample_ids = [0, 1]
# boolean array indicating the nodes both samples go through
common_nodes = node_indicator.toarray()[sample_ids].sum(axis=0) == len(sample_ids)
# obtain node ids using position in array
common_node_id = np.arange(n_nodes)[common_nodes]

print(
    "\nThe following samples {samples} share the node(s) {nodes} in the tree.".format(
        samples=sample_ids, nodes=common_node_id
    )
)
print("This is {prop}% of all nodes.".format(prop=100 * len(common_node_id) / n_nodes))
```

```
The following samples [0, 1] share the node(s) [0 2] in the tree.
This is 40.0% of all nodes.
```

Bagging vs Boosting

As mentioned above, in Bagging, multiple homogenous algorithms are trained independently in parallel, while in Boosting, multiple homogenous algorithms are trained sequentially. The image below shows the difference between Bagging and Boosting.



In [1]:

```
import pandas as pd
df = pd.read_csv("diabetes.csv")
df.head()
```

Out[1]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunc
0	6	148	72	35	0	33.6	0.
1	1	85	66	29	0	26.6	0.
2	8	183	64	0	0	23.3	0.
3	1	89	66	23	94	28.1	0.
4	0	137	40	35	168	43.1	2.

In [2]:

```
df.isnull().sum()
```

Out[2]:

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [25]:

```
X = df.drop("Outcome",axis="columns")
y = df.Outcome
```

Dataset scaling

Dataset scaling is transforming a dataset to fit within a specific range. For example, you can scale a dataset to fit within a range of 0-1, -1-1, or 0-100.

In [26]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled[:3]
```

Out[26]:

```
array([[ 0.63994726,  0.84832379,  0.14964075,  0.90726993, -0.69289057,
         0.20401277,  0.46849198,  1.4259954 ],
       [-0.84488505, -1.12339636, -0.16054575,  0.53090156, -0.69289057,
        -0.68442195, -0.36506078, -0.19067191],
       [ 1.23388019,  1.94372388, -0.26394125, -1.28821221, -0.69289057,
        -1.10325546,  0.60439732, -0.10558415]])
```

Splitting the Dataset

We will split the scaled dataset into training and testing. To split the dataset, we will use the train_test_split method.

In [29]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, stratify=y, random_stat
X_train.shape
X_test.shape
```

Out[29]:

```
(192, 8)
```

Model building using Decision Tree Classifier

The decision tree classifier is the Scikit-learn algorithm used for classification. To import this algorithm, use this code:

In [30]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
scores = cross_val_score(DecisionTreeClassifier(), X, y, cv=5)
scores
```

Out[30]:

```
array([0.68831169, 0.64935065, 0.67532468, 0.81045752, 0.7124183 ])
```

Getting the Mean Accuracy Score

In [31]:

```
scores.mean()
```

Out[31]:

```
0.7071725659960955
```

Building the model using Bagging Classifier

In [34]:

```
from sklearn.ensemble import BaggingClassifier
bag_model = BaggingClassifier(
base_estimator=DecisionTreeClassifier(),
n_estimators=100,
max_samples=0.8,
bootstrap=True,
oob_score=True,
random_state=0
)
bag_model.fit(X_train, y_train)
# Accuracy Score
bag_model.oob_score_
```

Out[34]:

```
0.7534722222222222
```

In [35]:

```
bag_model.score(X_test, y_test)
```

Out[35]:

```
0.7760416666666666
```

Random Forest Classifier

Random Forest Classifier has several decision trees trained on the various subsets. This algorithm is a typical example of a bagging algorithm.

In [40]:

```
from sklearn.ensemble import RandomForestClassifier
scores = cross_val_score(RandomForestClassifier(n_estimators=50), X, y, cv=5)
scores.mean()
```

Out[40]:

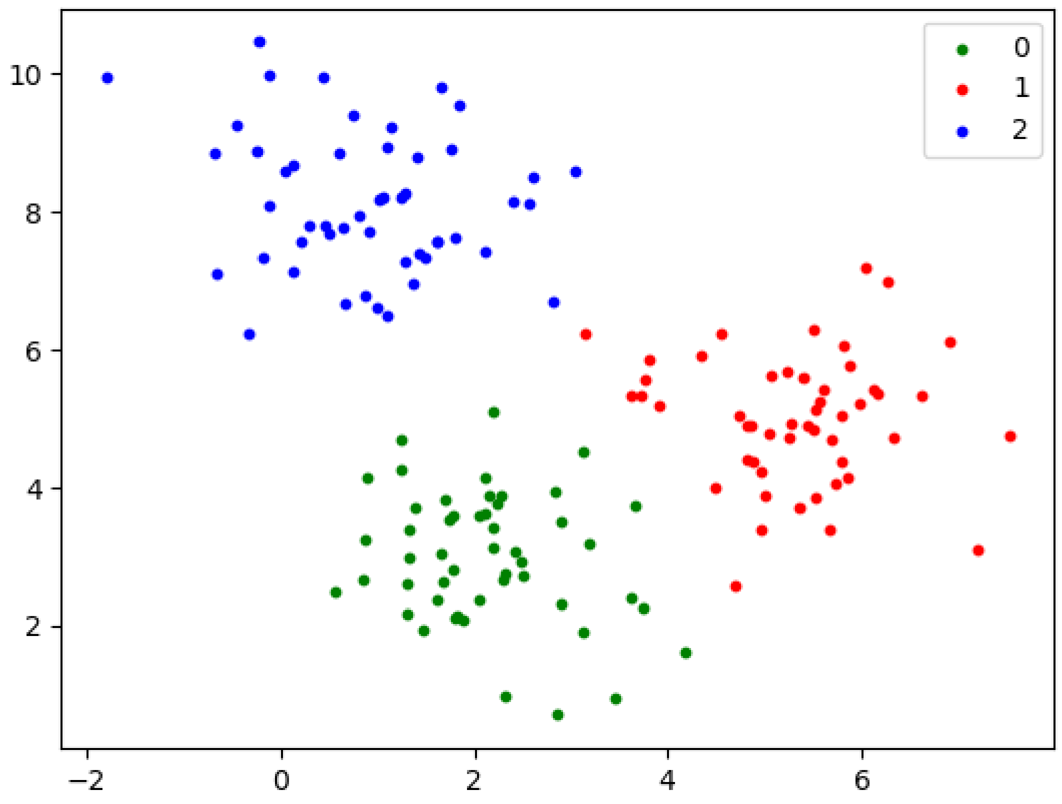
```
0.7643748408454292
```

(KNN) K Neighbors Classifier

is based on the k nearest neighbors of a sample, which has to be classified. The number 'k' is an integer

In [43]:

```
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
import numpy as np

centers = [[2, 3], [5, 5], [1, 8]]
n_classes = len(centers)
data, labels = make_blobs(n_samples=150,
                          centers=np.array(centers),
                          random_state=1)
import matplotlib.pyplot as plt

colours = ('green', 'red', 'blue')
n_classes = 3

fig, ax = plt.subplots()
for n_class in range(0, n_classes):
    ax.scatter(data[labels==n_class, 0], data[labels==n_class, 1],
               c=colours[n_class], s=10, label=str(n_class))



ax.legend(loc='upper right');
```



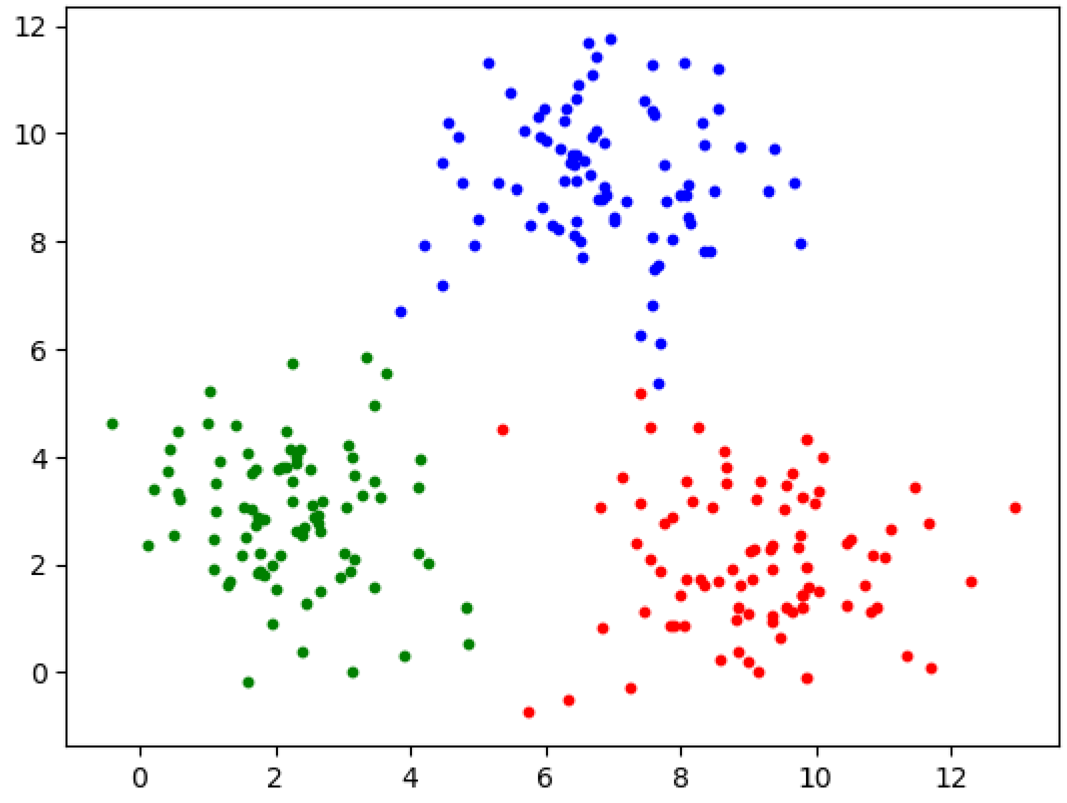
In [45]:

```
from sklearn import datasets
from sklearn.model_selection import train_test_split

iris = datasets.load_iris()
data, labels = iris.data, iris.target

res = train_test_split(data, labels,
                       train_size=0.8,
                       test_size=0.2,
                       random_state=12)
train_data, test_data, train_labels, test_labels = res
# Create and fit a nearest-neighbor classifier
from sklearn.neighbors import KNeighborsClassifier
# classifier "out of the box", no parameters
knn = KNeighborsClassifier()
knn.fit(train_data, train_labels)


print("Predictions from the classifier:")
test_data_predicted = knn.predict(test_data)
print(test_data_predicted)
print("Target values:")
print(test_labels)
# Accuracy
print(accuracy_score(test_data_predicted, test_labels))
```

```
Predictions from the classifier:
[0 2 0 1 2 2 2 0 2 0 1 0 0 0 1 2 2 1 0 2 0 1 2 1 0 2 1 1 0 0]
Target values:
[0 2 0 1 2 2 2 0 2 0 1 0 0 0 1 2 2 1 0 1 0 1 2 1 0 2 1 1 0 0]
0.9666666666666667

F:\Software\Data Science\AnacondInstallFile\lib\site-packages\sklearn\neig
hbors\_classification.py:228: FutureWarning: Unlike other reduction functi
ons (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically pr
eserves the axis it acts along. In SciPy 1.11.0, this behavior will chang
e: the default value of `keepdims` will become False, the `axis` over whic
h the statistic is taken will be eliminated, and the value None will no lo
nger be accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

In [48]:

```
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
import numpy as np

centers = [[2, 3], [9, 2], [7, 9]]
n_classes = len(centers)
data, labels = make_blobs(n_samples=255,
                          centers=np.array(centers),
                          cluster_std = 1.3,
                          random_state=1)
import matplotlib.pyplot as plt

colours = ('green', 'red', 'blue')
n_classes = 3     # not using the outlier 'class'

fig, ax = plt.subplots()
for n_class in range(0, n_classes):
    ax.scatter(data[labels==n_class, 0], data[labels==n_class, 1],
               c=colours[n_class], s=10, label=str(n_class))
```



Neural Network Using Keras

In [59]:

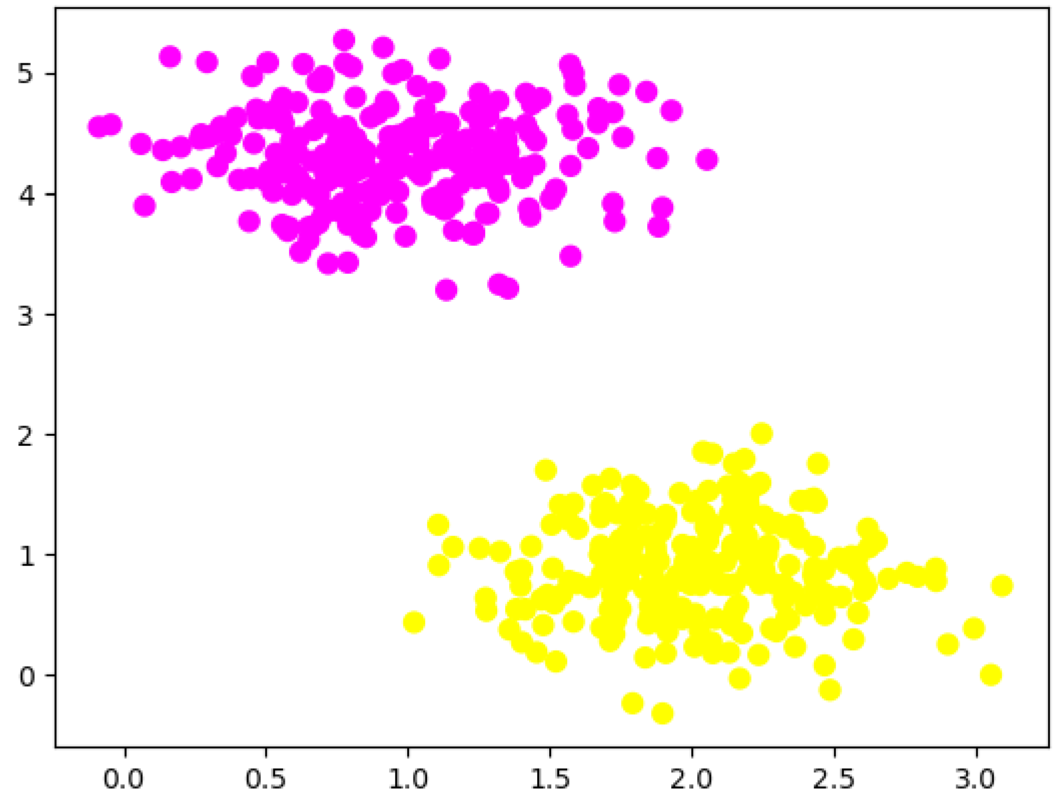
```
from keras.models import Sequential
from keras.layers import Dense, Activation
import numpy as np

# Use numpy arrays to store inputs (x) and outputs (y):
x = np.array([[0,0], [0,1], [1,0], [1,1]])
y = np.array([[0], [1], [1], [0]])

# Define the network model and its arguments.
# Set the number of neurons/nodes for each layer:
model = Sequential()
model.add(Dense(2, input_shape=(2,)))
model.add(Activation('sigmoid'))
model.add(Dense(1))
model.add(Activation('sigmoid'))

# Compile the model and calculate its accuracy:
model.compile(loss='mean_squared_error', optimizer='sgd', metrics=['accuracy'])

# Print a summary of the Keras model:
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_2 (Dense)             (None, 2)                 6

 activation_2 (Activation)   (None, 2)                 0

 dense_3 (Dense)             (None, 1)                 3

 activation_3 (Activation)   (None, 1)                 0

=================================================================
Total params: 9
Trainable params: 9
Non-trainable params: 0
_________________________________________________________________
```

SVM (Support Vector Machine)

In [55]:

```
from sklearn.datasets import make_blobs
X, Y = make_blobs(n_samples=500,
                          centers=2,random_state=0,
                          cluster_std = 0.40
                          )

import matplotlib.pyplot as plt
plt.scatter(X[:, 0], X[:, 1],c=Y, s=50, cmap='spring')
plt.show()
```



Data Visualization for K-Means

In [53]:

```
import pandas as pd
df=pd.read_csv('iris.csv')
df=pd.DataFrame(df)
df.head()
```

Out[53]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [ ]:

```

```

In [41]:

```
label_0=df['sepal_length']
label_1=df['sepal_width']
label_2=df['petal_length']
```

In [44]:

```
import matplotlib.pyplot as plt
cols=df.columns
cols
```

Out[44]:

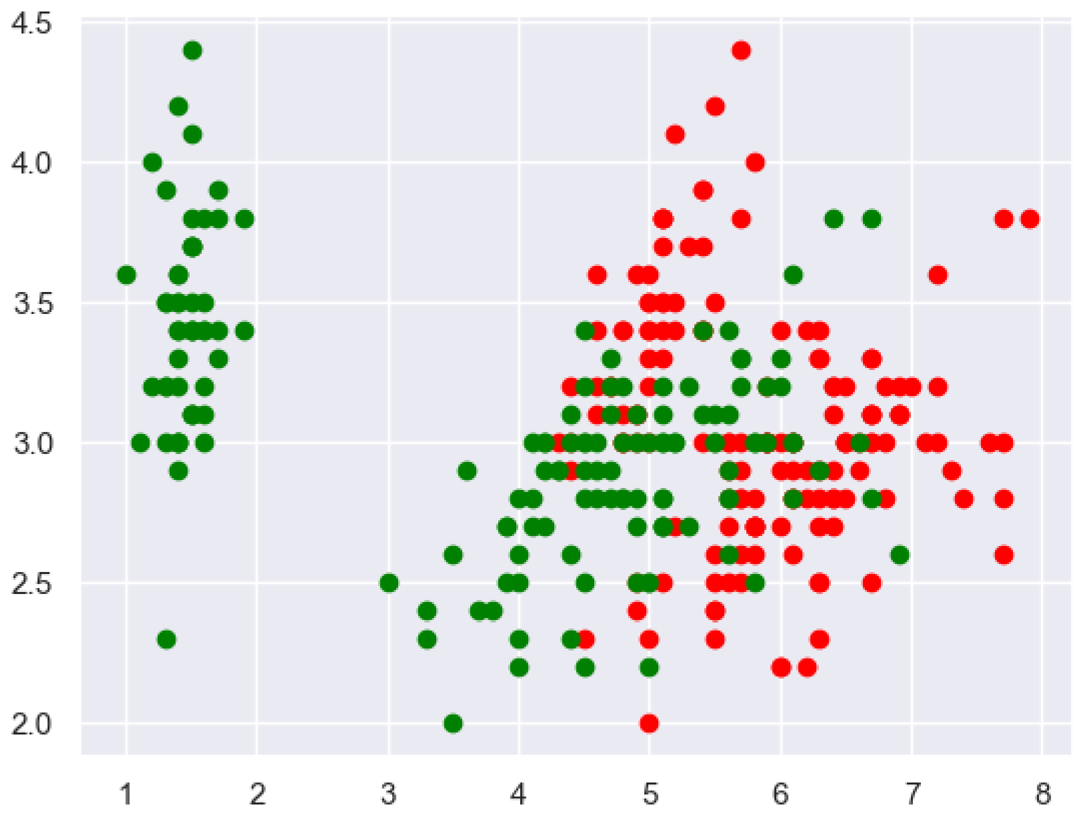
```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
       'species'],
      dtype='object')
```

In [50]:

```
plt.scatter(df['sepal_length'],df['sepal_width'],color='red')
plt.scatter(df['petal_length'],df['sepal_width'],color='green')
```

Out[50]:

```
<matplotlib.collections.PathCollection at 0x1b18489ac40>
```



In [ ]:

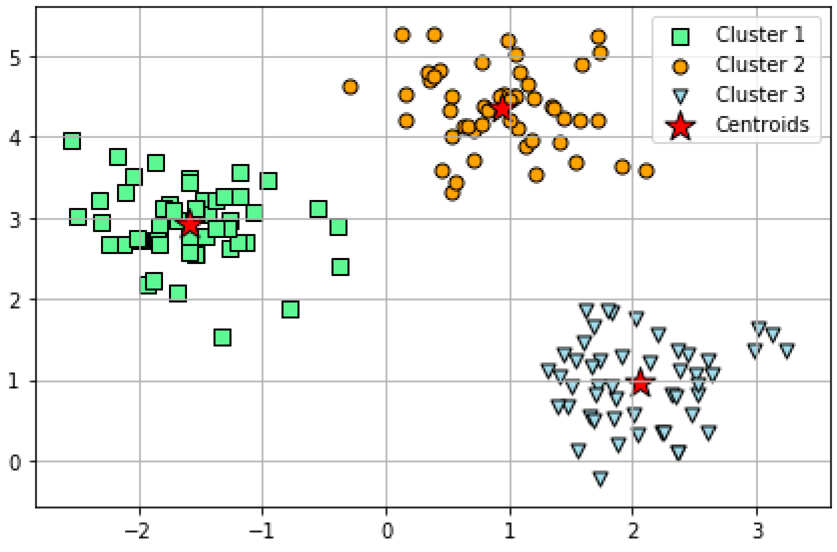
```

```

In [ ]:

```

```

K-Means Clustering?

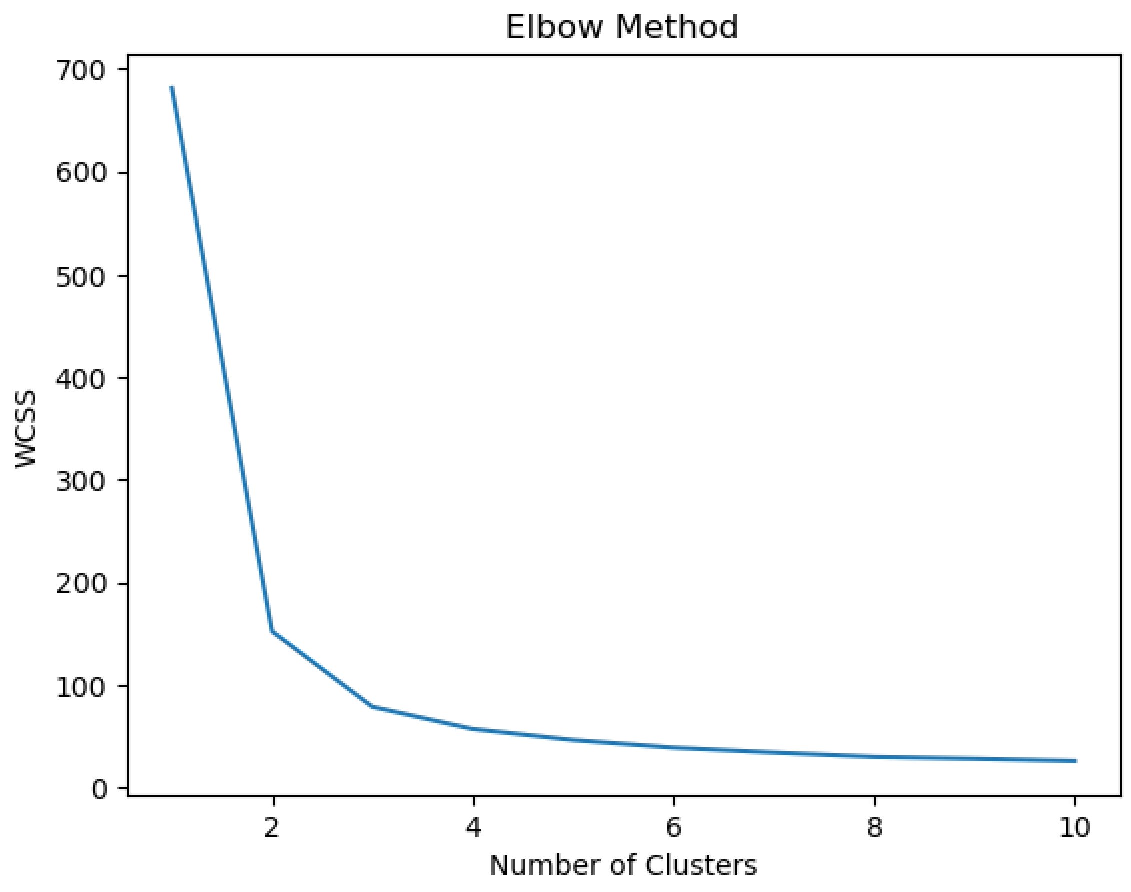
Clustering is a popular unsupervised machine learning technique used in data analysis to group similar data points together. The K-Means clustering algorithm is one of the most commonly used clustering algorithms due to its simplicity, efficiency, and effectiveness on a wide range of datasets.



In [2]:

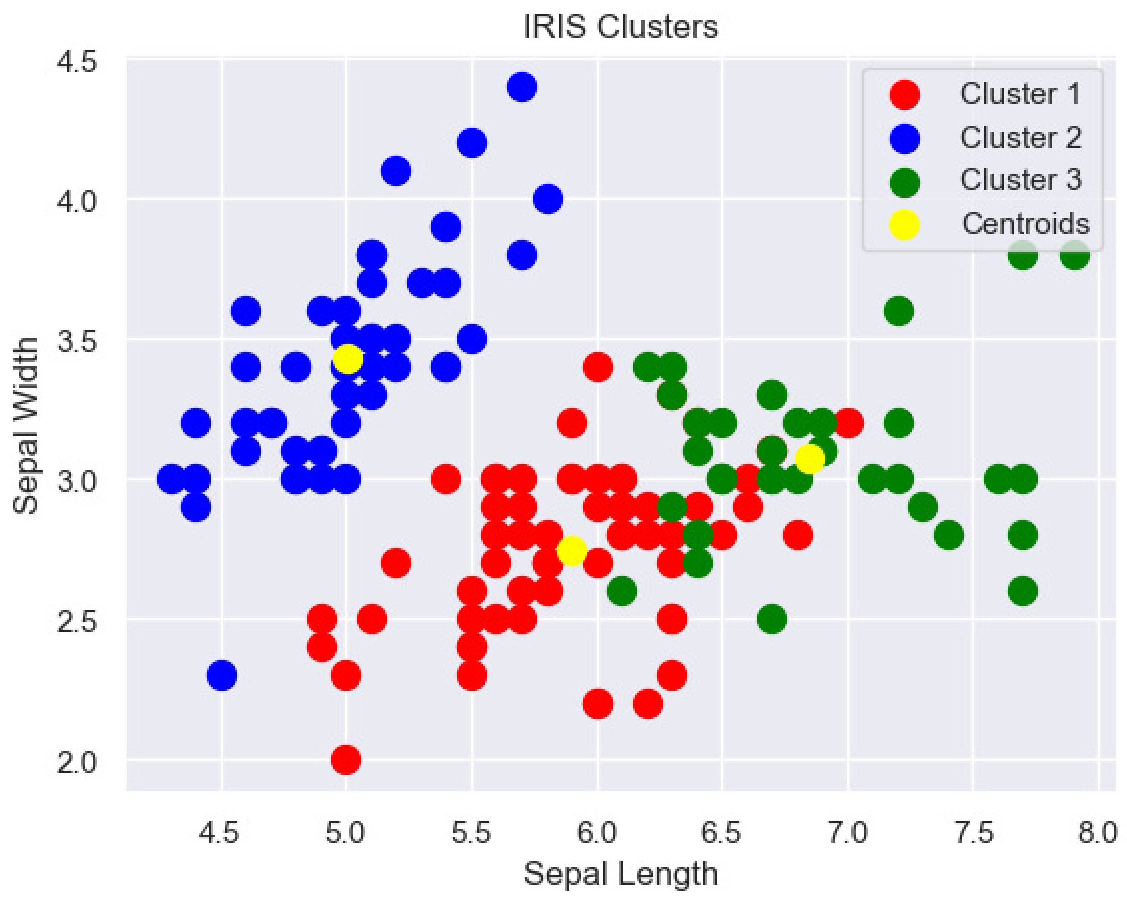
```
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Load IRIS Dataset
iris = load_iris()
X = iris.data

# Create the WCSS Plot against no. of clusters
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_stat
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()
```

```
F:\Software\Data Science\AnacondInstallFile\lib\site-packages\sklearn\clus
ter\_kmeans.py:1036: UserWarning: KMeans is known to have a memory leak on
Windows with MKL, when there are less chunks than available threads. You c
an avoid it by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```



In [54]:

```
# Train K-Means Clusters
kmeans = KMeans(n_clusters=3, init='k-means++', max_iter=300, n_init=10, random_state=0)
y_kmeans = kmeans.fit_predict(X)

# Create the visualization plot of the clusters
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'red', label = 'Clust
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'blue', label = 'Clus
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Clu
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 100, c = '
plt.title('IRIS Clusters')
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.legend()
plt.show()
```



In [ ]:

```
```

In [ ]:

```
```

Principal Component Analysis (PCA)

Principal Component Analysis is an unsupervised learning algorithm that is used for the dimensionality reduction in machine learning. It is a statistical process that converts the observations of correlated features into a set of linearly uncorrelated features with the help of orthogonal transformation. These new transformed features are called the Principal Components. It is one of the popular tools that is used for exploratory data analysis and predictive modeling. It is a technique to draw strong patterns from the given dataset by reducing the variances.

PCA generally tries to find the lower-dimensional surface to project the high-dimensional data.

PCA works by considering the variance of each attribute because the high attribute shows the good split between the classes, and hence it reduces the dimensionality. Some real-world applications of PCA are image processing, movie recommendation system, optimizing the power allocation in various

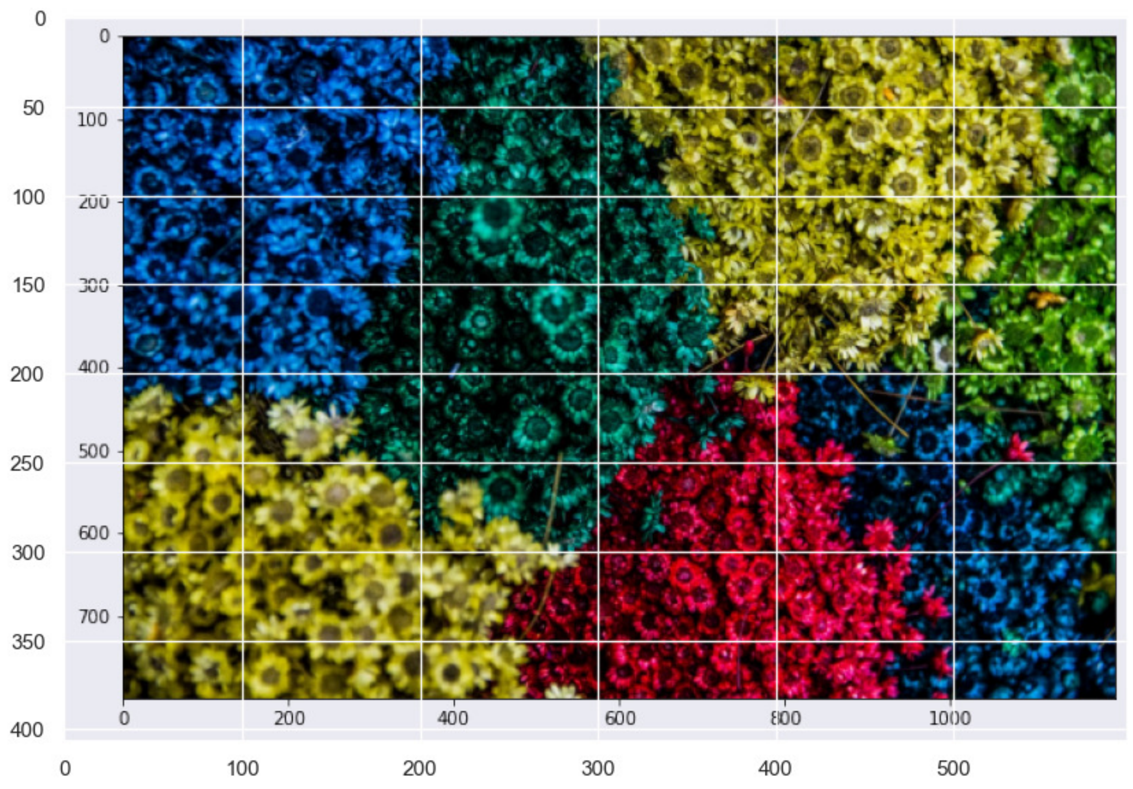
[illegible]

In [4]:

```
import numpy as np
import pandas as pd
import scipy

import matplotlib.pyplot as plt
import seaborn as sns
sns.set()

from sklearn.preprocessing import StandardScaler
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.cluster import KMeans

from sklearn.decomposition import PCA

import pickle
```

In [6]:

```
import matplotlib.pyplot as plt
from matplotlib import image
img = image.imread("clustering.jpg")
plt.figure(figsize=(10,10))
plt.imshow(img)
plt.show()
```



Importing Data

In [57]:

```
df_segmentation= pd.read_csv("segmentationdata2.csv", index_col=0)
df_segmentation.head()
```

Out[57]:

	Sex	Marital status	Age	Education	Income	Occupation	Settlement size
ID							
100000001	0	0	67	2	124670	1	2
100000002	1	1	22	1	150773	1	2
100000003	0	0	49	1	89210	0	0
100000004	0	0	45	1	171565	1	1
100000005	0	0	53	1	149031	1	1

In [58]:

```
df_segmentation.describe()
```

Out[58]:

	Sex	Marital status	Age	Education	Income	Occupation	Set
count	2000.000000	2000.000000	2000.000000	2000.00000	2000.000000	2000.000000	2000
mean	0.457000	0.496500	35.909000	1.03800	120954.419000	0.810500	0
std	0.498272	0.500113	11.719402	0.59978	38108.824679	0.638587	0
min	0.000000	0.000000	18.000000	0.00000	35832.000000	0.000000	0
25%	0.000000	0.000000	27.000000	1.00000	97663.250000	0.000000	0
50%	0.000000	0.000000	33.000000	1.00000	115548.500000	1.000000	1
75%	1.000000	1.000000	42.000000	1.00000	138072.250000	1.000000	1
max	1.000000	1.000000	76.000000	3.00000	309364.000000	2.000000	2

Corellation Estimation

In [59]:

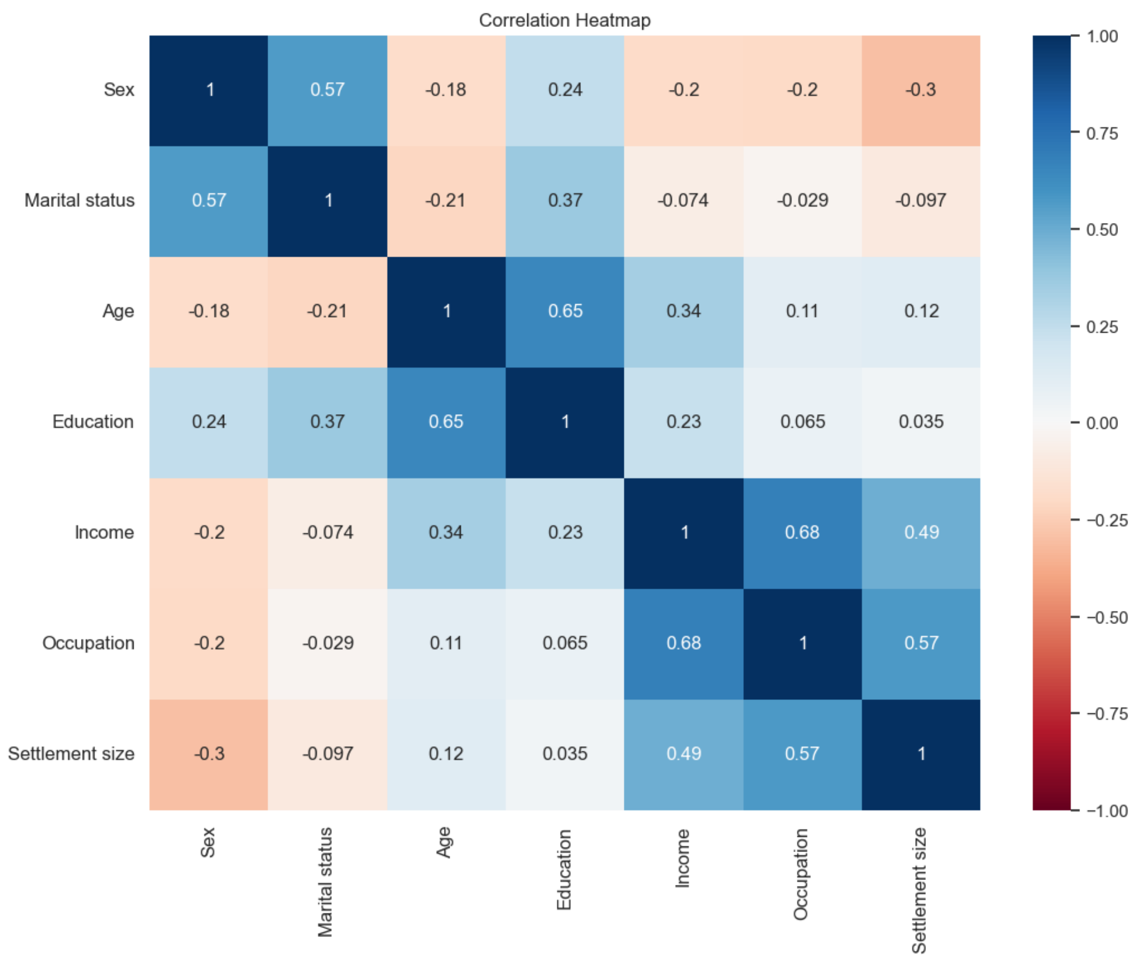
```
df_segmentation.corr()
```

Out[59]:

	Sex	Marital status	Age	Education	Income	Occupation	Settlement size
Sex	1.000000	0.566511	-0.182885	0.244838	-0.195146	-0.202491	-0.300803
Marital status	0.566511	1.000000	-0.213178	0.374017	-0.073528	-0.029490	-0.097041
Age	-0.182885	-0.213178	1.000000	0.654605	0.340610	0.108388	0.119751
Education	0.244838	0.374017	0.654605	1.000000	0.233459	0.064524	0.034732
Income	-0.195146	-0.073528	0.340610	0.233459	1.000000	0.680357	0.490881
Occupation	-0.202491	-0.029490	0.108388	0.064524	0.680357	1.000000	0.571795
Settlement size	-0.300803	-0.097041	0.119751	0.034732	0.490881	0.571795	1.000000

In [61]:

```
plt.figure(figsize=(12,9))
s=sns.heatmap(df_segmentation.corr(), annot= True, cmap='RdBu', vmin=-1, vmax=1)
s.set_yticklabels(s.get_yticklabels(), rotation=0, fontsize=12)
s.set_xticklabels(s.get_xticklabels(), rotation=90, fontsize=12)
plt.title("Correlation Heatmap")
plt.show()
```

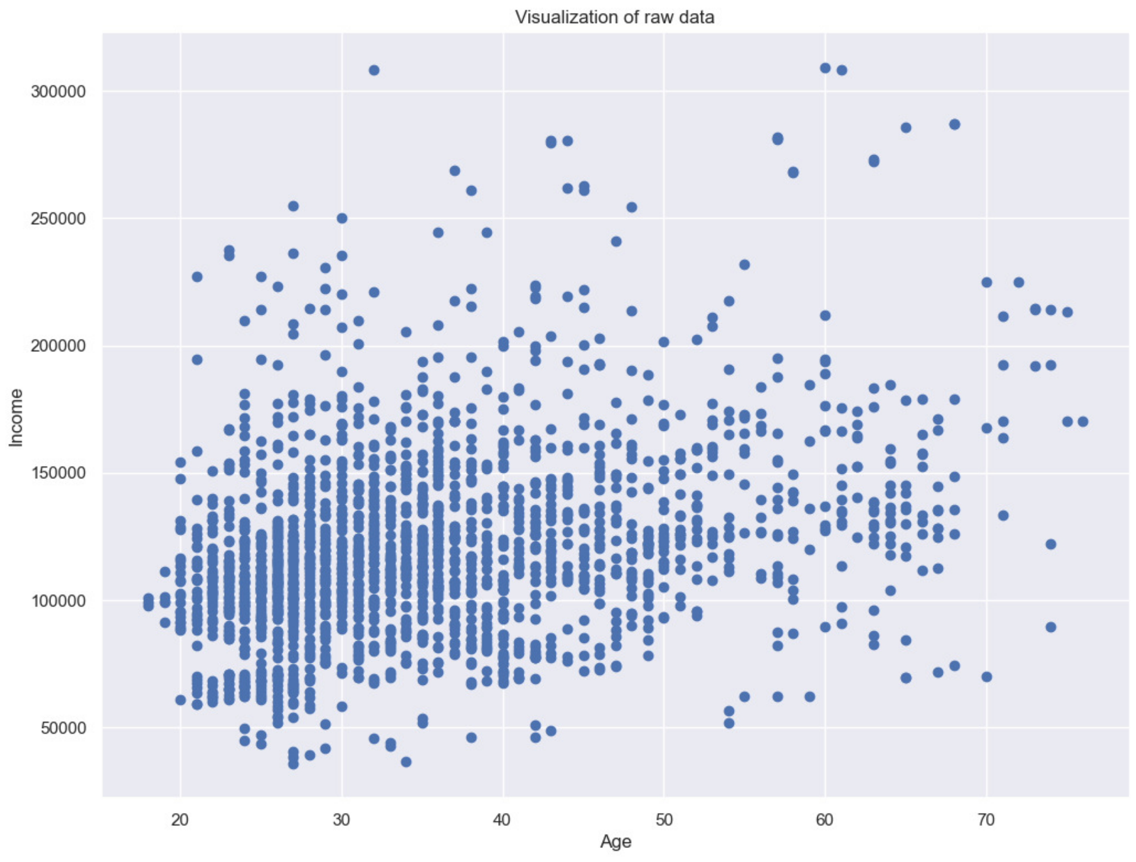


	Sex	Marital status	Age	Education	Income	Occupation	Settlement size
Sex	1	0.57	-0.18	0.24	-0.2	-0.2	-0.3
Marital status	0.57	1	-0.21	0.37	-0.074	-0.029	-0.097
Age	-0.18	-0.21	1	0.65	0.34	0.11	0.12
Education	0.24	0.37	0.65	1	0.23	0.065	0.035
Income	-0.2	-0.074	0.34	0.23	1	0.68	0.49
Occupation	-0.2	-0.029	0.11	0.065	0.68	1	0.57
Settlement size	-0.3	-0.097	0.12	0.035	0.49	0.57	1

Visualizing Raw data

In [14]:

```
plt.figure(figsize=(12,9))
plt.scatter(df_segmentation.iloc[:,2], df_segmentation.iloc[:,4])
plt.xlabel("Age")
plt.ylabel("Income")
plt.title("Visualization of raw data")
plt.show()
```



Standardization

In [65]:

```
scaler= StandardScaler()
segmentation_std= scaler.fit_transform(df_segmentation)
```

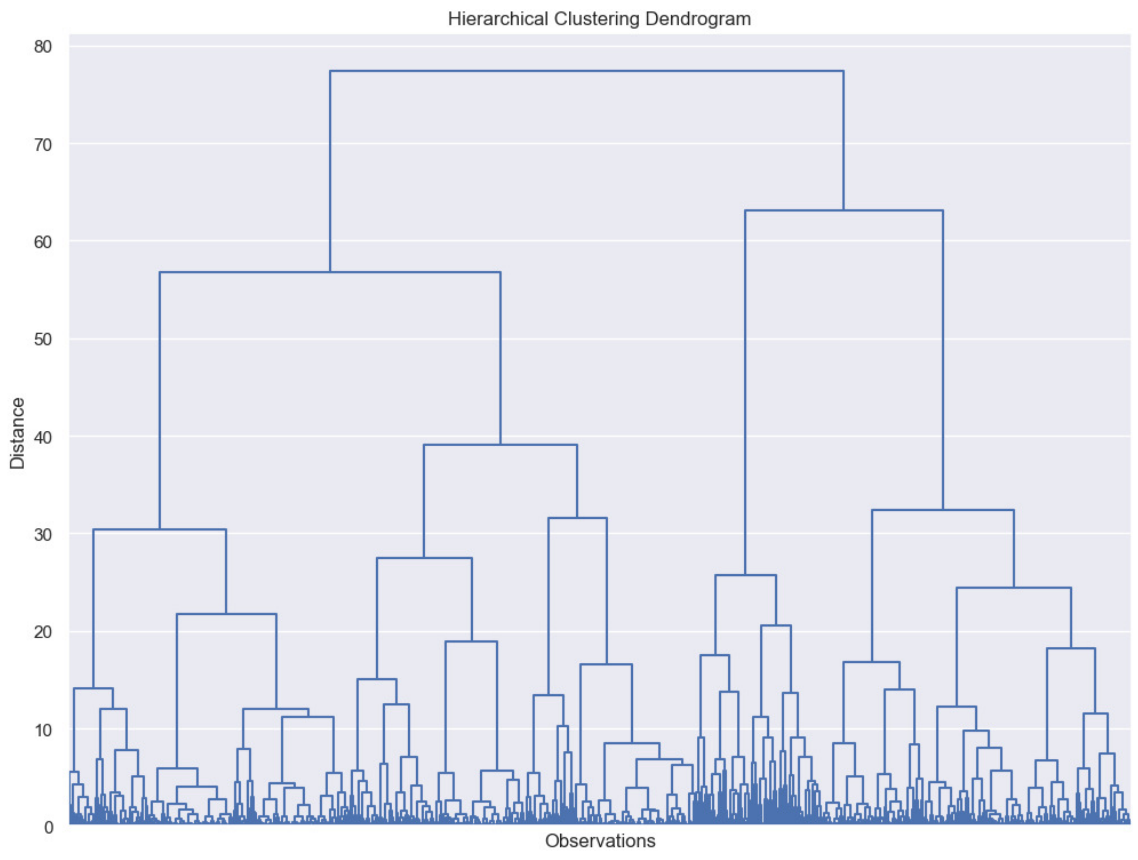
Hierarchical Clustering

In [66]:

```
hier_clust= linkage(segmentation_std, method='ward')
```

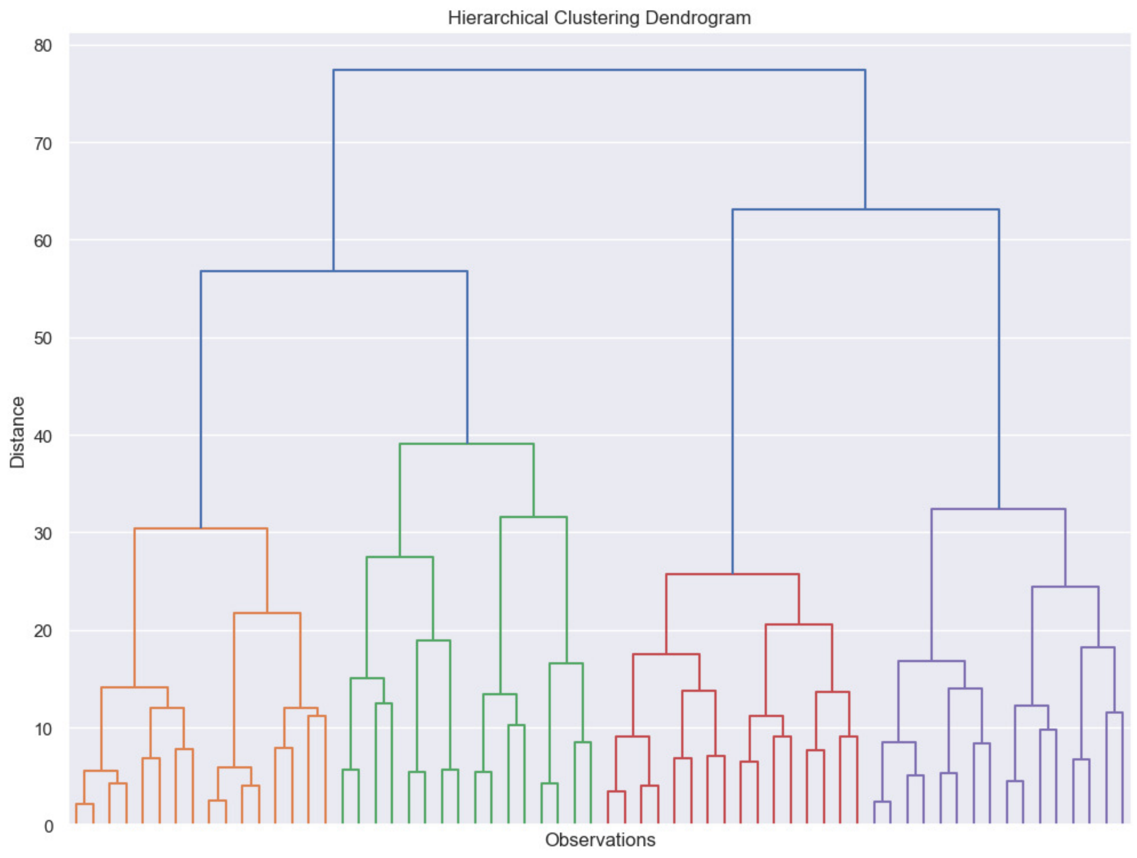
In [67]:

```
plt.figure(figsize=(12,9))
plt.title("Hierarchical Clustering Dendrogram")
plt.xlabel("Observations")
plt.ylabel("Distance")
dendrogram(hier_clust, show_leaf_counts=False, no_labels=True, color_threshold=0)
plt.show()
```



In [68]:

```
plt.figure(figsize=(12,9))
plt.title("Hierarchical Clustering Dendrogram")
plt.xlabel("Observations")
plt.ylabel("Distance")
dendrogram(hier_clust, truncate_mode='level', p=5, show_leaf_counts=False, no_labels=Tru
plt.show()
```



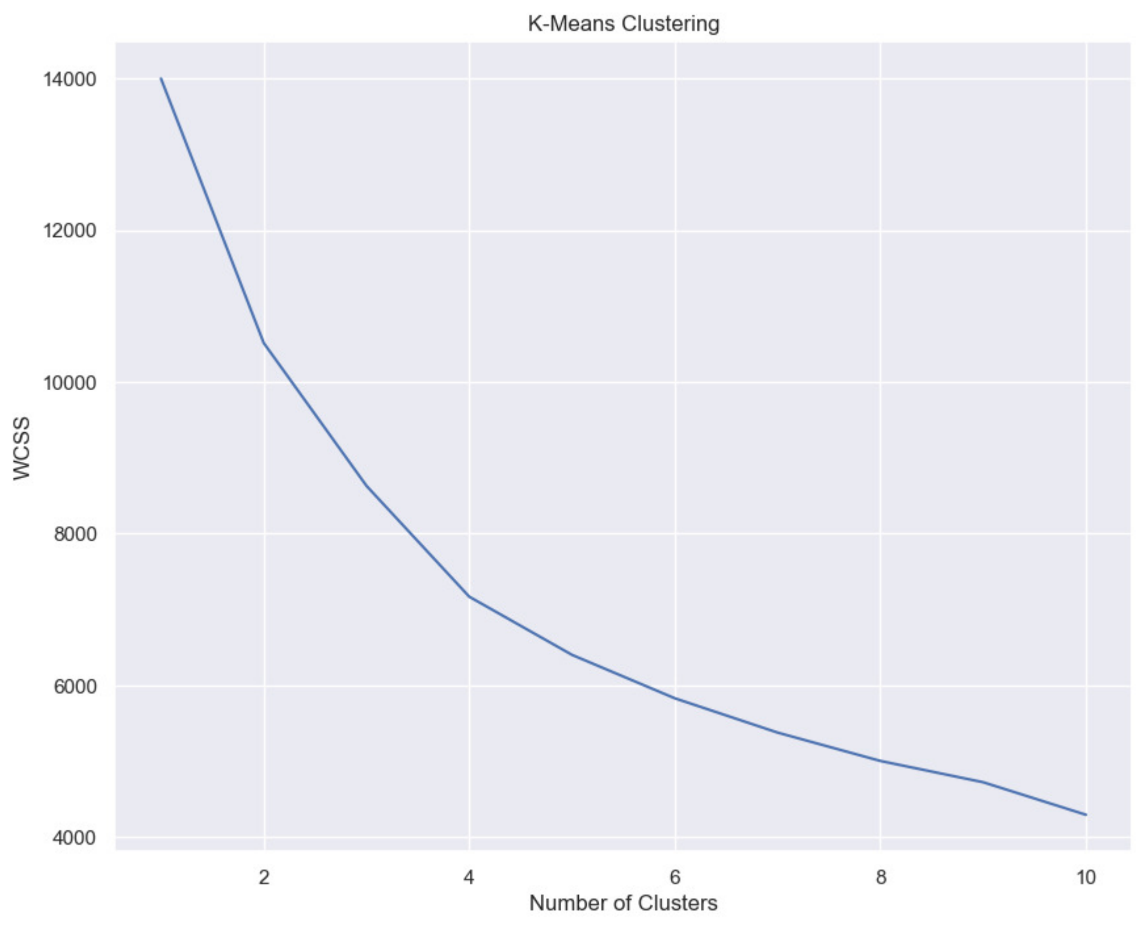
K-Means Clustering

In [69]:

```
wcss=[]
for i in range(1,11):
    kmeans=KMeans(n_clusters=i, init='k-means++', random_state=42)
    kmeans.fit(segmentation_std)
    wcss.append(kmeans.inertia_)
```

In [70]:

```
plt.figure(figsize=(10,8))
plt.plot(range(1,11), wcss)
plt.xlabel("Number of Clusters")
plt.ylabel("WCSS")
plt.title("K-Means Clustering")
plt.show()
```



In [74]:

```
kmean=KMeans(n_clusters=4, init='k-means++', random_state=42)
kmean.fit(segmentation_std)
```

Out[74]:

```
KMeans(n_clusters=4, random_state=42)
```

Results

In [75]:

```
df_segm_kmeans=df_segmentation.copy()
df_segm_kmeans['Segment k-Means']=kmean.labels_
```

In [76]:

```
df_segm_analysis=df_segm_kmeans.groupby(['Segment k-Means']).mean()
df_segm_analysis
```

Out[76]:

	Sex	Marital status	Age	Education	Income	Occupation	Settlement size
Segment k-Means							
0	0.501901	0.692015	55.703422	2.129278	158338.422053	1.129278	1.110266
1	0.352814	0.019481	35.577922	0.746753	97859.852814	0.329004	0.043290
2	0.853901	0.997163	28.963121	1.068085	105759.119149	0.634043	0.422695
3	0.029825	0.173684	35.635088	0.733333	141218.249123	1.271930	1.522807

In [77]:

```
df_segm_analysis['N-obs']=df_segm_kmeans[['Sex','Segment k-Means']].groupby(['Segment k-
```

In [79]:

```
df_segm_analysis['Prop-obs']=df_segm_analysis['N-obs']/df_segm_analysis['N-obs'].sum()
df_segm_analysis
```

Out[79]:

	Sex	Marital status	Age	Education	Income	Occupation	Settlement size
Segment k-Means							
0	0.501901	0.692015	55.703422	2.129278	158338.422053	1.129278	1.110266
1	0.352814	0.019481	35.577922	0.746753	97859.852814	0.329004	0.043290
2	0.853901	0.997163	28.963121	1.068085	105759.119149	0.634043	0.422695
3	0.029825	0.173684	35.635088	0.733333	141218.249123	1.271930	1.522807

In [80]:

```
df_segm_analysis.rename({0:'Well off',1:'Standard',2: 'Fewer Opportunities',3:'Career Fo
```

Out[80]:

	Sex	Marital status	Age	Education	Income	Occupation	Settlem s
Segment k-Means							
Well off	0.501901	0.692015	55.703422	2.129278	158338.422053	1.129278	1.110
Standard	0.352814	0.019481	35.577922	0.746753	97859.852814	0.329004	0.043
Fewer Opportunities	0.853901	0.997163	28.963121	1.068085	105759.119149	0.634043	0.422
Career Focussed	0.029825	0.173684	35.635088	0.733333	141218.249123	1.271930	1.522

In [83]:

```
df_segm_kmeans['Labels']= df_segm_kmeans['Segment k-Means'].map({0:'Well off',1:'Standar
```

In [98]:

```
# x_axis=df_segm_kmeans['Age']
# y_axis=df_segm_kmeans['Income']
# plt.figure(figsize=(12,9))
# sns.scatterplot(x_axis,y_axis,hue= df_segm_kmeans['Labels'], palette={'g','r','c','m'}
# plt.title("Segmented K-Means")
# plt.show()
```

Principal Component Analysis(PSA)

In [99]:

```
pca=PCA()
pca.fit(segmentation_std)
```

Out[99]:

```
PCA()
```

In [100]:

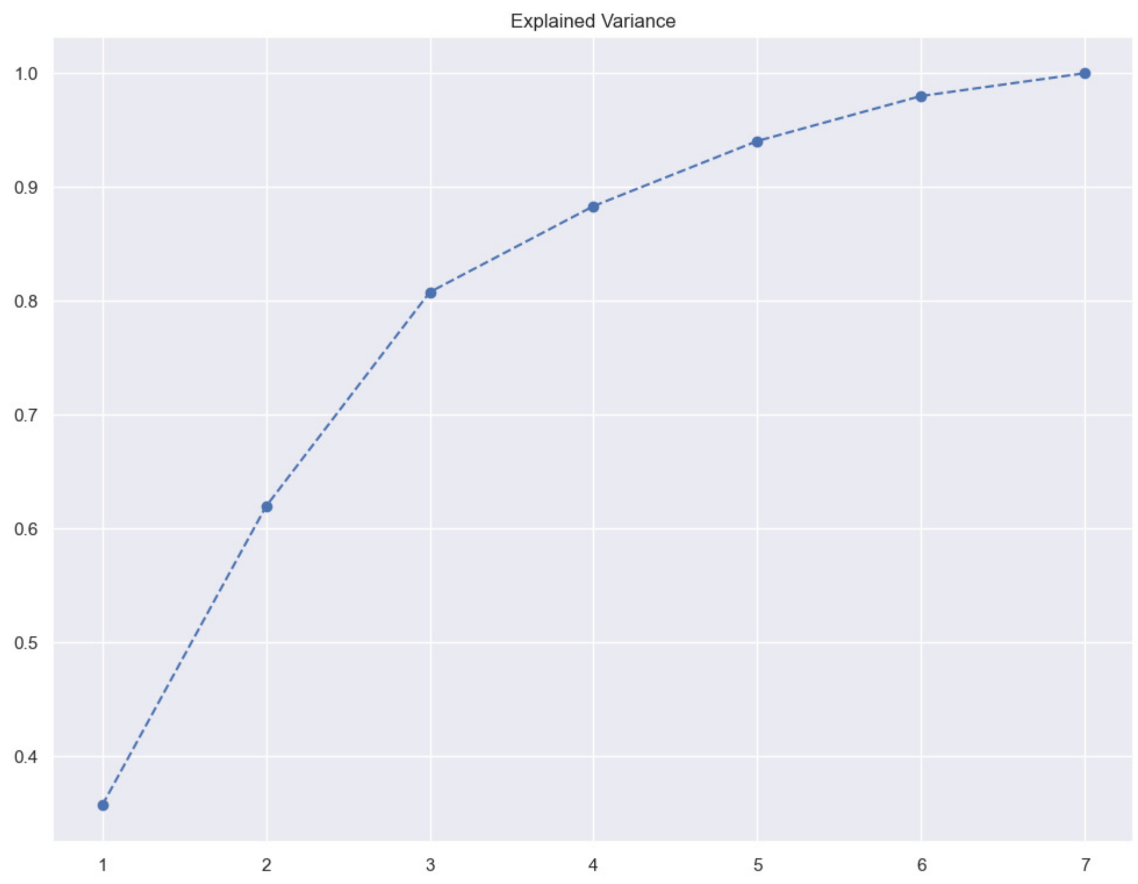
```
pca.explained_variance_ratio_
```

Out[100]:

```
array([0.35696328, 0.26250923, 0.18821114, 0.0755775 , 0.05716512,
       0.03954794, 0.02002579])
```

In [101]:

```
plt.figure(figsize=(12,9))
plt.plot(range(1,8), pca.explained_variance_ratio_.cumsum(), marker='o',linestyle='--' )
plt.title("Explained Variance")
plt.show()
```



In [102]:

```
pca=PCA(n_components=3)
pca.fit(segmentation_std)
```

Out[102]:

```
PCA(n_components=3)
```

PCA Results

In [103]:

```
pca.components_
```

Out[103]:

```
array([[-0.31469524, -0.19170439,  0.32609979,  0.15684089,  0.52452463,
         0.49205868,  0.46478852],
       [ 0.45800608,  0.51263492,  0.31220793,  0.63980683,  0.12468314,
         0.01465779, -0.06963165],
       [-0.29301261, -0.44197739,  0.60954372,  0.27560461, -0.16566231,
        -0.39550539, -0.29568503]])
```

In [105]:

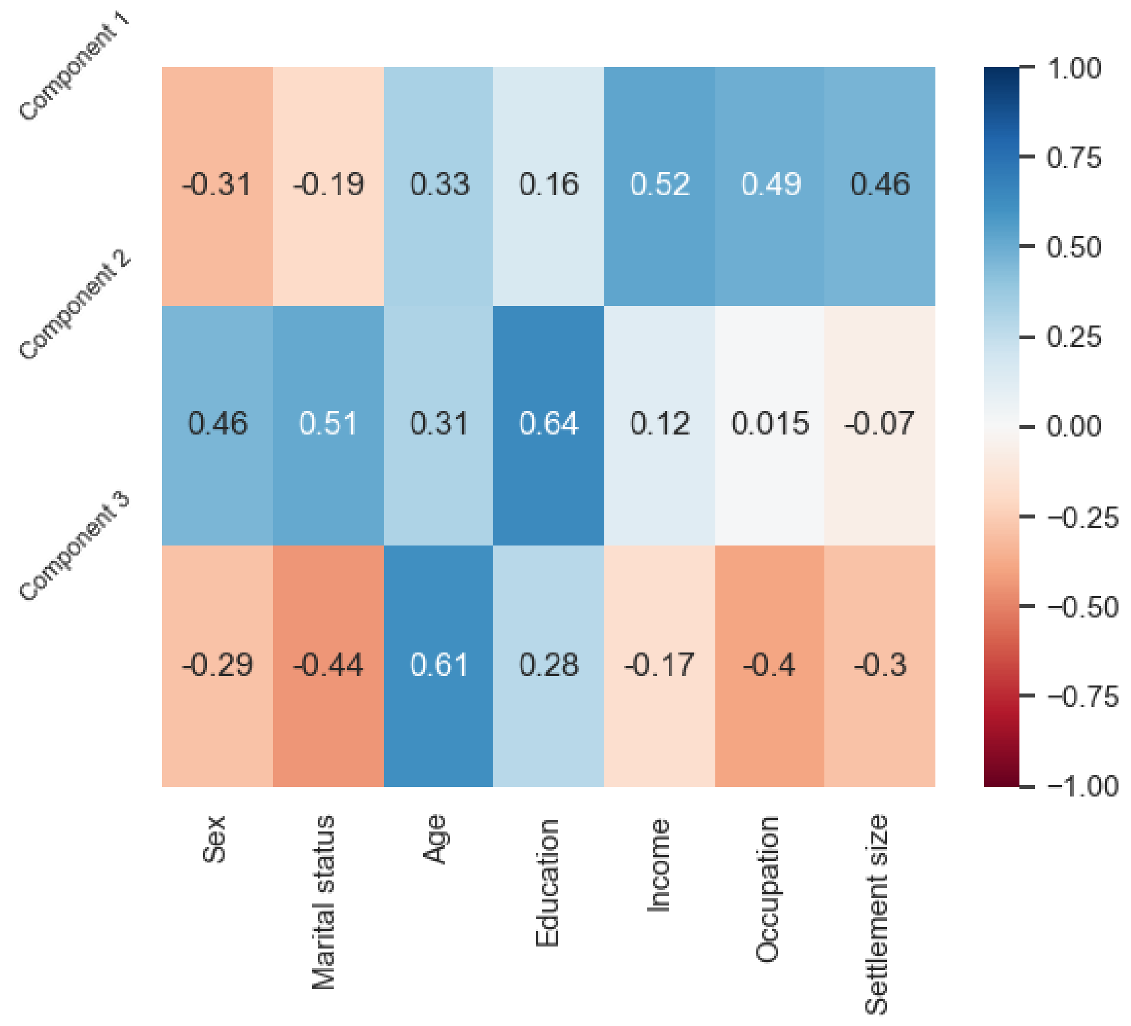
```
df_pca_comp= pd.DataFrame(data= pca.components_, columns= df_segmentation.columns.values
df_pca_comp
```

Out[105]:

	Sex	Marital status	Age	Education	Income	Occupation	Settlement size
Component 1	-0.314695	-0.191704	0.326100	0.156841	0.524525	0.492059	0.464789
Component 2	0.458006	0.512635	0.312208	0.639807	0.124683	0.014658	-0.069632
Component 3	-0.293013	-0.441977	0.609544	0.275605	-0.165662	-0.395505	-0.295685

In [106]:

```
sns.heatmap(df_pca_comp, vmin=-1, vmax=1, annot=True, cmap='RdBu')
plt.yticks([0,1,2], ['Component 1','Component 2', 'Component 3'], rotation=45, fontsize=
plt.show()
```



In [108]:

```
scores_pca= pca.transform(segmentation_std)
scores_pca
```

Out[108]:

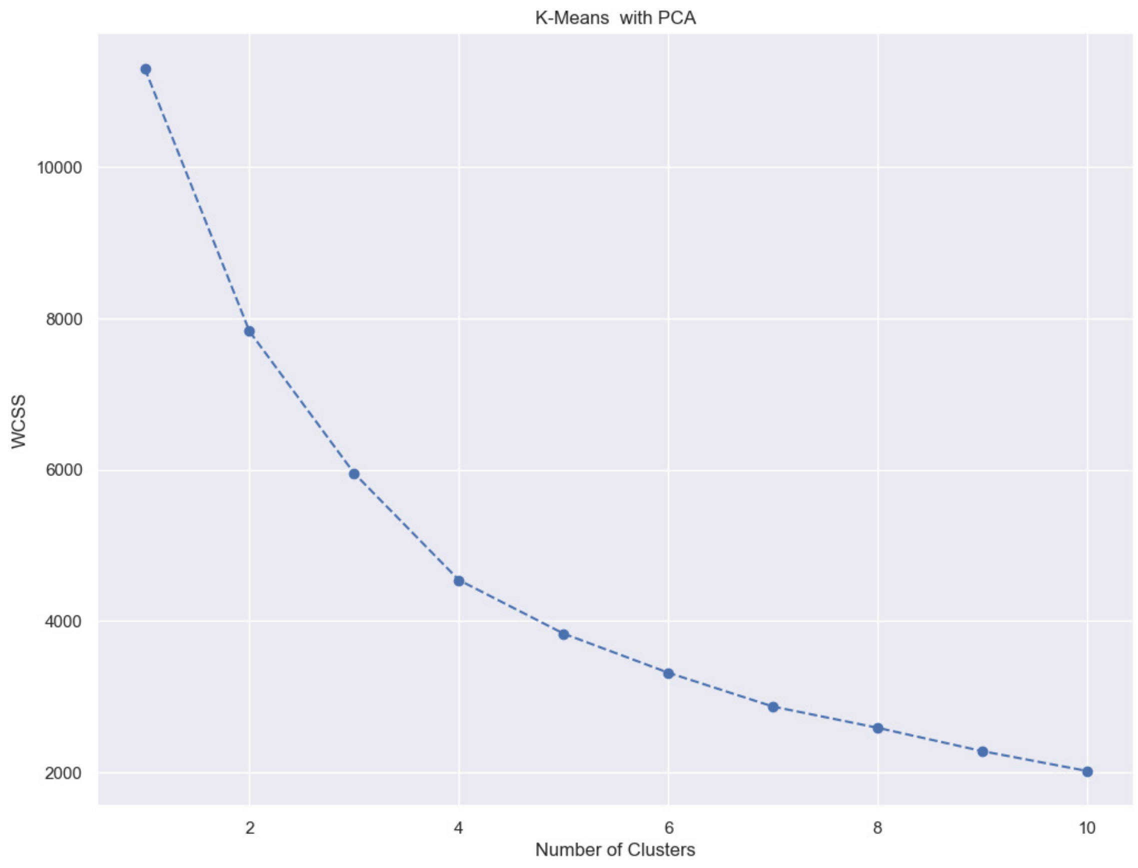
```
array([[ 2.51474593,  0.83412239,  2.1748059 ],
       [ 0.34493528,  0.59814564, -2.21160279],
       [-0.65106267, -0.68009318,  2.2804186 ],
       ...,
       [-1.45229829, -2.23593665,  0.89657125],
       [-2.24145254,  0.62710847, -0.53045631],
       [-1.86688505, -2.45467234,  0.66262172]])
```

In [109]:

```
wcss_pca=[]
for i in range(1,11):
    kmeans_pca=KMeans(n_clusters= i, init='k-means++', random_state=42)
    kmeans_pca.fit(scores_pca)
    wcss_pca.append(kmeans_pca.inertia_)
```

In [110]:

```
plt.figure(figsize=(12,9))
plt.plot(range(1,11),wcss_pca, marker='o', linestyle='--')
plt.xlabel("Number of Clusters")
plt.ylabel("WCSS")
plt.title("K-Means  with PCA")
plt.show()
```



In [111]:

```
kmeans_pca=KMeans(n_clusters= 4, init='k-means++', random_state=42)
kmeans_pca.fit(scores_pca)
```

Out[111]:

```
KMeans(n_clusters=4, random_state=42)
```

K-Means clustering with PCA results

In [112]:

```
df_segm_pca_kmeans= pd.concat([df_segmentation.reset_index(drop=True), pd.DataFrame(scor
df_segm_pca_kmeans.columns.values[-3: ]= ['Component 1','Component 2','Component 3']
df_segm_pca_kmeans['Segment k-means PCA']= kmeans_pca.labels_
```

In [113]:

```
df_segm_pca_kmeans
```

Out[113]:

	Sex	Marital status	Age	Education	Income	Occupation	Settlement size	Component 1	Componer
0	0	0	67	2	124670	1	2	2.514746	0.83412
1	1	1	22	1	150773	1	2	0.344935	0.59814
2	0	0	49	1	89210	0	0	-0.651063	-0.68009
3	0	0	45	1	171565	1	1	1.714316	-0.57992
4	0	0	53	1	149031	1	1	1.626745	-0.44049
...	...	...	...	...	...	...	...	...	.
1995	1	0	47	1	123525	0	0	-0.866034	0.29833
1996	1	1	27	1	117744	1	0	-1.114957	0.79472
1997	0	0	31	0	86400	0	0	-1.452298	-2.23593
1998	1	1	24	1	97968	0	0	-2.241453	0.62710
1999	0	0	25	0	68416	0	0	-1.866885	-2.45467

2000 rows × 11 columns

In [115]:

```
df_segm_pca_kmeans_freq= df_segm_pca_kmeans.groupby(['Segment k-means PCA']).mean()
df_segm_pca_kmeans_freq
```

Out[115]:

Segment k-means PCA	Sex	Marital status	Age	Education	Income	Occupation	Settlement size
0	0.900289	0.965318	28.878613	1.060694	107551.500000	0.677746	0.440751
1	0.027444	0.168096	35.737564	0.734134	141525.826758	1.267581	1.480274
2	0.306522	0.095652	35.313043	0.760870	93692.567391	0.252174	0.039130
3	0.505660	0.690566	55.679245	2.128302	158019.101887	1.120755	1.101887

In [116]:

```
df_segm_pca_kmeans_freq['N-obs']=df_segm_pca_kmeans[['Sex','Segment k-means PCA']].group
df_segm_pca_kmeans_freq['Prop-obs']=df_segm_pca_kmeans_freq['N-obs']/df_segm_pca_kmeans_
df_segm_pca_kmeans_freq=df_segm_pca_kmeans_freq.rename({0:'Standard',1:'Career Focussed'
df_segm_pca_kmeans_freq
```

Out[116]:

Segment k-means PCA	Sex	Marital status	Age	Education	Income	Occupation	Settlem s
Standard	0.900289	0.965318	28.878613	1.060694	107551.500000	0.677746	0.440
Career Focussed	0.027444	0.168096	35.737564	0.734134	141525.826758	1.267581	1.480
Fewer Opportunities	0.306522	0.095652	35.313043	0.760870	93692.567391	0.252174	0.039
Well off	0.505660	0.690566	55.679245	2.128302	158019.101887	1.120755	1.101

In [ ]:

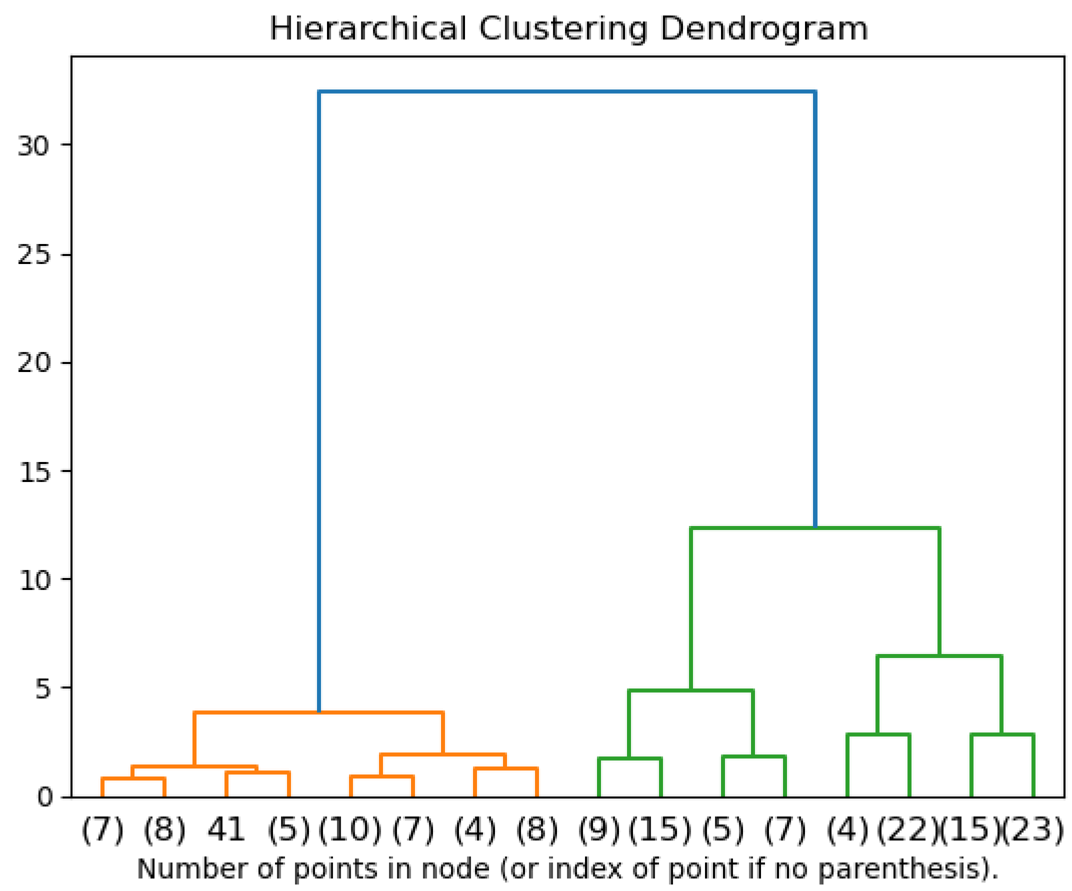
In [ ]:

Plot Hierarchical Clustering Dendrogram

This example plots the corresponding dendrogram of a hierarchical clustering using AgglomerativeClustering and the dendrogram method available in scipy.

In [1]:

```
import numpy as np

from matplotlib import pyplot as plt
from scipy.cluster.hierarchy import dendrogram
from sklearn.datasets import load_iris
from sklearn.cluster import AgglomerativeClustering


def plot_dendrogram(model, **kwargs):
    # Create linkage matrix and then plot the dendrogram

    # create the counts of samples under each node
    counts = np.zeros(model.children_.shape[0])
    n_samples = len(model.labels_)
    for i, merge in enumerate(model.children_):
        current_count = 0
        for child_idx in merge:
            if child_idx < n_samples:
                current_count += 1  # leaf node
            else:
                current_count += counts[child_idx - n_samples]
        counts[i] = current_count

    linkage_matrix = np.column_stack(
        [model.children_, model.distances_, counts]
    ).astype(float)

    # Plot the corresponding dendrogram
    dendrogram(linkage_matrix, **kwargs)


iris = load_iris()
X = iris.data

# setting distance_threshold=0 ensures we compute the full tree.
model = AgglomerativeClustering(distance_threshold=0, n_clusters=None)

model = model.fit(X)
plt.title("Hierarchical Clustering Dendrogram")
# plot the top three levels of the dendrogram
plot_dendrogram(model, truncate_mode="level", p=3)
plt.xlabel("Number of points in node (or index of point if no parenthesis).")
plt.show()
```



✨Cluster Analysis: Visualize customer segmentation

for more information click on below link

https://www.kaggle.com/code/patelris/cluster-analysis-visualize-customer-segmentation

In [ ]:

```

```

In [ ]:

```

```