SRI VASAVI ENGINEERINGCOLLEGE (AUTONOMOUS)

PEDATADEPALLI, TADEPALLIGUDEM.

Certificate

This is to certify that this is a bonafide record of Practical Work done in ***Machine Learning Lab using Python*** *by Mr/Mrs.* _____ *bearing Roll No.*_____*of* ***CSE*** *branch of* ***VI Semester*** *during the academic year 2023 to 2024.*

No. of Experiments Done: 12

Signature of Faculty in charge of the Laboratory

Head of the Department

Signature of the external examiner

Index

S. No	Name of the experiment	Date		Signature
		Performed	Submitted	
1	Introduction to required python library Numpy and working with it.			
2	Introduction to python library Pandas for data handling a. Series data structure in Pandas. b. Creation of DataFrame data structure in Pandas. c. Indexing of DataFrame data structure in Pandas. d. Operations on rows and columns of DataFrame data structure in Pandas.			
3	Introduction to python library Matplotlib for visualization.			
4	Import, preprocess, and split the datasets using sklearn library in python.			
5	Construct a classification model using the Bayes classifier using python Programming.			
6	Implement a Logistic Regression algorithm for binary classification using python programming.			
7	Implement the KNN algorithm a. For classification (building the model) b. Demonstrate the process of finding out optimal K value using python programming.			
8	Construct an SVM classifier using python programming.			
9	Demonstrate the process of the Decision Tree construction for classification problems using python programming.			
10	Implement an Ensemble Learner using Random Forest Algorithm and Adaboost algorithm using python programming.			
11	Demonstrate the working of Multi-layer perceptron with MLPClassifier() using python programming.			
12	Demonstrate the K-Means algorithm for the given data set using python programming.			

Date: 1.Numpy

Aim:

Working with NumPy

Outcomes

At the end of this chapter, you will be able to

- Define NumPy data types
- Create and work with NumPy Arrays
- Understand the applications of NumpPy

What is Numpy?

- A Python package for scientific computations
- NumPy = ***Numerical Python***
- Numpy Library = Multi-dimensional array objects & Functions for processing it.
- Multi-dimensional object --> ***ndarray*** object
- Variaous operations supported: Arithmetic, Logical, Shape manipulation, Sort, DFT, Statistics, Random simulation etc.
- Importing NumPy: ***import numpy as np***

In [2]:
```
# import numpy
import numpy as np
```

Creating and Accessing ndarray object

- ndarray: Collection of items of the same type
- Items can be accessed using zero-based index.
- array() Function: ***numpy.array(object)***

```
--> Creates an ndarray from an object.
--> object: Any array or sequence.
```

One Dimensional Array (Vector):

In [3]:
```
# Create an one dimensional ndarray object from a list, display its values and type
li=[1,2,3,5]
a1=np.array(li)
```

```
print(type(a1))
print(a1)
```

```
<class 'numpy.ndarray'>
[1 2 3 5]
```

In [5]:
```
# Create an one dimensional ndarray object from a tuple, display its values and type
li=(1,2,3,5)
a1=np.array(li)
print(type(a1))
print(a1)
```

```
<class 'numpy.ndarray'>
[1 2 3 5]
```

Indexing & Slicing:

Consider arr = numpy.array([10, 20, 30, 40])

Item	10	20	30	40	arr
Index	0	1	2	3	

In [23]:
```
# Create above one dimensional ndarray object and access its elements individually. An
a=np.array([1,2,3,4])
print(a[0])
a[0]+=1
print(a[0])
print(a[1:3])
```

```
1
2
[2 3]
```

Two-Dimensional Arrays(Matrices)

- Each element is itself 1D array
- arr2 = numpy.array([[row1], [row2], ..., [rowM]])
- Accessing an element: arr2[row_index, column_index]

In [24]:
```
# Create a two dimensional array and access their rows and columns individually.
# Access the individual elements.
l2=[[1,2,3],[4,5,6],[7,8,9]]
a1=np.array(l2)
print(a1)
#first row
print("first row is",a1[0,:])
#second column
print("second colum is",a1[:,2])
#sub array
print(a1[0:2,1:3])
print(1,1)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
first row is [1 2 3]
second colum is [3 6 9]
[[2 3]
 [5 6]]
1 1
```

ndarray Attribtes

- ndarray.ndim: Number of dimensions
- ndarry.shape: Array dimensions as a tuple.
- ndarry.size: Total number of elements
- ndarry.dtype: Type of the array elements.
- ndarray.itemsize: The size of each element in bytes

In [3]:
```
# Create an one dimensional ndarray object and diplay all its attribute values.
a=np.array([1,3,6,9,5])
print(a)
#ndarray attributes
print("ndarray attributes for integer datatype")
print(a.ndim)
print(a.shape)
print(a.size)
print(a.dtype)
print(a.itemsize)
#float datatype
a1=np.array([1.5,2.6,7.9])
print(a1)
#ndarray attributes
print("ndarray  attributes for float datatype")
print(a1.ndim)
print(a1.shape)
print(a1.size)
print(a1.dtype)
print(a1.itemsize)
```

```
[1 3 6 9 5]
ndarray attributes for integer datatype
1
(5,)
5
int32
4
[1.5 2.6 7.9]
ndarray  attributes for float datatype
1
(3,)
3
float64
8
```

Reshaping using reshape(shape) method

In [56]:
```
# Reshape any of the above ndarray object using reshape method
a1=np.array([1,2,3,4,5,6]).reshape(2,3)
print(a1)
```

```
[[1 2 3]
 [4 5 6]]
```

Conclusion:

Date: 2a. Pandas_Series

Aim:

Data Handling using Pandas

At the end of this chapter, you will be able to

- Create and use various data structures in Pandas
- Perform various operations on Pandas data structures
- Import/Export data from/to different sources

What is Pandas?

- PANDAS - PANel DAta
- Built on top of NumPy
- Used for data analysis and pre-processing
- Two Data Structures
  1. Series
  2. DataFrame
- Pandas Data structures can have different data types --> object, int64, float64, bool, datetime64, timedelta[ns], cetegory
- Preferred when the data is in tabular format
- Supports integration with different types of data sorces such as excel files, csv files, sql tables, json files etc.

Installation

- Installing Pandas fron PyPI via pip on commannd line

  pip install pandas
- Installing Pandas with Anaconda

  conda install pandas

Loading Pandas

- Need to import the pandas to load and use it

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

Pandas Data Structures

1. Series

2. DataFrame

Pandas Series

- One-dimensional array-like object contains a sequence of data values of same type.
- Index - Label associated with a value in Series. (Index starts with 0)
- Example series: Employees

```
Index     Value

  0        Ram
  1        Sam
  2        Raj
  3        Rani
```

Creation of a Series

- pandas.Series(data): Returns a pandas Series object with values specified by given iterable or scalar value.

Pandas Series from a list

In [5]:
```
l=[1,2,10,15,6.9]
s=pd.Series(l);
print(type(s))
```

```
<class 'pandas.core.series.Series'>
```

In [6]:
```
print(s)
```

```
0     1.0
1     2.0
2    10.0
3    15.0
4     6.9
dtype: float64
```

Pandas Series from a Tuple

In [7]:
```
t=("harshi","bhanu","venky","padhu")
s1=pd.Series(t)
```

```
print(type(s))
print(s1)
```

```
<class 'pandas.core.series.Series'>
0    harshi
1     bhanu
2     venky
3     padhu
dtype: object
```

Pandas Series from a dictionary

In [5]:
```
d={"Name":"Harshitha","Dept":"CSE","Age":19,"RNo":73}
s3=pd.Series(d)
print(type(s3))
print(s3)
```

```
<class 'pandas.core.series.Series'>
Name    Harshitha
Dept          CSE
Age            19
RNo            73
dtype: object
```

Pandas Series from a ndarray

In [9]:
```
import numpy as np
a=np.array(l)
print(type(a))
s4=pd.Series(a)
print(s4)
```

```
<class 'numpy.ndarray'>
0     1.0
1     2.0
2    10.0
3    15.0
4     6.9
dtype: float64
```

Accessing Series Elements

1. Indexing: To access individial elements

2. Slicing: To access a part of the Series

1. Indexing

Two Types of Indexing

1. Positional Indexing: Values can be accessed by their position. (Starts from 0)

2. Labelled Indexing: Values can be accessed using any user-defined labels.

In [10]:
```
# Positional Indexing
print(s3)
print()
print(s3[0])
print()
print(s3[2])
```

```
Name    Harshitha
Dept          CSE
Age            19
RNo            73
dtype: object

Harshitha

19
```

In [11]:
```
# Labelled Indexing (We can assign labels as a list with index attribute)
print(s3["Name"])
print()
print(s3["RNo"])
```

```
Harshitha

73
```

2. Slicing

- Usage: pandas.Series[start_index:end_index:step]
- -> Default start_index is 0
- -> Default step is 1
- -> end_index is excluded

In [12]:
```
s31=s3[0:1]
print(s31)
```

```
Name    Harshitha
dtype: object
```

In [13]:
```
s32=s3[: : 2]
print(s32)
```

```
Name    Harshitha
Age            19
dtype: object
```

In [14]:
```
#revesing a string
s33=s3[: : -1]
print(s33)
```

```
RNo            73
Age            19
Dept          CSE
Name    Harshitha
dtype: object
```

Series Attributes

1. values: Array of all values in the series

2. index: Object with all indexes as its values

3. dtype: Data type of the Series's elements

4. name: Name of the Series object

5. size: Number of values in the Series object

6. empty: True if Series is empty

In [15]:
```
print(s3)
print()
print(s3.values)
```

```
Name    Harshitha
Dept          CSE
Age            19
RNo            73
dtype: object

['Harshitha' 'CSE' 19 73]
```

In [16]:
```
print(s3.index)
```

```
Index(['Name', 'Dept', 'Age', 'RNo'], dtype='object')
```

In [17]:
```
s3.index=["n","d","a","r"]
print(s3)
```

```
n    Harshitha
d          CSE
a           19
r           73
dtype: object
```

In [24]:
```
print(s3.dtype)
```

```
object
```

In [25]:
```
print(s3.name)
```

```
None
```

In [16]:
```
#assingning name to a series
s3.name="Student Details"
print(s3.name)
```

```
Student Details
```

In [17]:
```
print(s3.size)
```

```
4
```

In [6]:
```
print(s3.empty)
print()
s6=pd.Series()
print(s6.empty)
```

```
False

True
```

print

Series Methods

head() and tail() Methods

Series.head(number_of_elements): Displays specified number of Series elements from the first.

Series.tail(number_of_elements): Displays specified number of Series elements from the last.

Note: The default number of elements to be displayed is 5

In [19]:
```
print(s3.head())
```

```
n    Harshitha
d          CSE
a           19
r           73
Name: Student Details, dtype: object
```

In [20]:
```
l2=[1,2,3,4,5,6,7,8,9,0]
s5=pd.Series(l2)
print(s5.tail())
```

```
5    6
6    7
7    8
8    9
9    0
dtype: int64
```

In [21]:
```
print(s5.head())
```

```
0    1
1    2
2    3
3    4
4    5
dtype: int64
```

In [22]:
```
print(s5.head(6))
```

```
0    1
1    2
2    3
3    4
4    5
5    6
dtype: int64
```

In [23]:
```
print(s5.tail(4))
```

```
6    7
7    8
8    9
9    0
dtype: int64
```

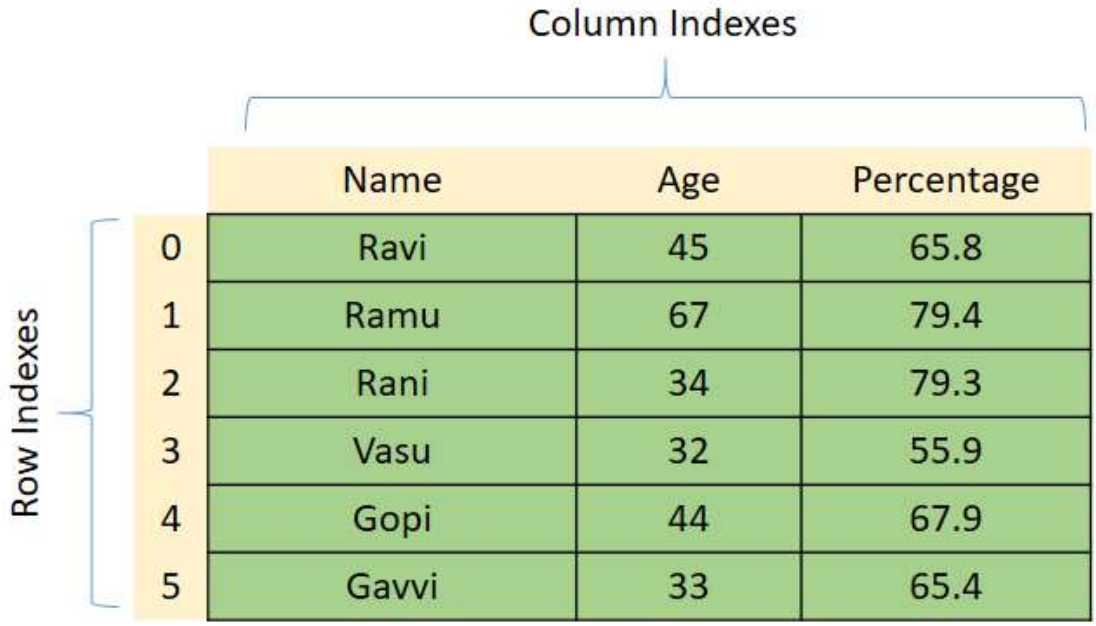
Conclusion:

Date: 2b. Pandas DataFrame

Aim:

pandas.DataFrame

* Two dimensional data structure * Used to store tabular data --> rows and column with row index and column index * Every column can have its own type of data. * We can think of it like a spreadsheet or database table.



In [1]:
```
# Import pandas library
import pandas as pd
```

DataFrame Creation

Creation of Empty DataFrame

- Dataframe object canbe created by using a calss **pandas.DataFrame().**

In [2]:
```
df=pd.DataFrame()
print(df)
print()
print(type(df))
```

```
Empty DataFrame
Columns: []
Index: []

<class 'pandas.core.frame.DataFrame'>
```

Creation of DataFrame from dictionary of lists

Here each item(key:value pair) wiil be a column in the dataframe

In [3]:
```
dl={"Name":["harshi","bhanu","venky","padhu","suma","mahi"],
    "Age":[19,18,40,38,23,27]
    ,"percentage":[86.9,89,88,87,87.9,85.5]}
print(dl)
print()
df1=pd.DataFrame(dl)
print(df1)
```

```
{'Name': ['harshi', 'bhanu', 'venky', 'padhu', 'suma', 'mahi'], 'Age': [19, 18, 40, 3
8, 23, 27], 'percentage': [86.9, 89, 88, 87, 87.9, 85.5]}

     Name  Age  percentage
0  harshi   19        86.9
1   bhanu   18        89.0
2   venky   40        88.0
3   padhu   38        87.0
4    suma   23        87.9
5    mahi   27        85.5
```

Creation of DataFrame from list of dictionaries

Each dictionary in the list will become a row in the dataframe

In [4]:
```
ld=[{"Name":"harshi","Age":19,"Percentage":86.9},
    {"Name":"bhanu","Age":18,"Percentage":89},
    {"Name":"venky","Age":40,"Percentage":88},
    {"Name":"padhu","Age":38,"Percentage":87},
    {"Name":"suma","Age":23,"Percentage":87.9},
    {"Name":"mahi","Age":27,"Percentage":85.5}]
print(ld)
print()
df2=pd.DataFrame(ld);
print(df2)
```

```
[{'Name': 'harshi', 'Age': 19, 'Percentage': 86.9}, {'Name': 'bhanu', 'Age': 18, 'Per
centage': 89}, {'Name': 'venky', 'Age': 40, 'Percentage': 88}, {'Name': 'padhu', 'Ag
e': 38, 'Percentage': 87}, {'Name': 'suma', 'Age': 23, 'Percentage': 87.9}, {'Name':
'mahi', 'Age': 27, 'Percentage': 85.5}]

     Name  Age  Percentage
0  harshi   19        86.9
1   bhanu   18        89.0
2   venky   40        88.0
3   padhu   38        87.0
4    suma   23        87.9
5    mahi   27        85.5
```

Creation of a DataFrame from list of lists

Each inner list will be a row in the dataframe

We can assign row indices and column names while creating a dataframe with index and columns attributes

In [5]:
```
ll=[["harshi","bhanu","venky", "padhu"],[19,18,40,38]]
df3=pd.DataFrame(ll)
print(df3)
```

```
        0      1      2      3
0  harshi  bhanu  venky  padhu
1      19     18     40     38
```

In [6]:
```
df3.index=["R1","R2"]
df3.columns=["C1","C2","C3","C4"]
print(df3)
```

```
        C1     C2     C3     C4
R1  harshi  bhanu  venky  padhu
R2      19     18     40     38
```

Working with CSV/Excel files

- CSV --> Comma Separated Values

Importing CSV file to a DataFrame

- **pandas.read_csv(path)**: Function to create a DataFrame from a csv file.
- Note: In windows system, represent path as a raw string to supress its escape functionality

In [17]:
```
df4=pd.read_csv("student.csv")
print(df4)
```

```
     Name  IQ  TimeSpent  Performance
0    Ramu   7          5           65
1  Ganesh   6          7           60
2  Kamesh   5          7           55
3  Rakesh   8          8           75
4   Subbu   8          4           70
5  Chinni   6          5           72
6     Nag   8          6           80
7     uma   6          6           65
8   srinu   6          7           65
9  Laxman   6          9           60
```

- **pandas.read_csv(path, names = [list_of_column_names])** : names keyword argument can be used when file is not having header.

In [21]:
```
df4["Performance"]=df4["Performance"]/10
print(df4)
```

```
     Name  IQ  TimeSpent  Performance
0    Ramu   7          5          6.5
1  Ganesh   6          7          6.0
2  Kamesh   5          7          5.5
3  Rakesh   8          8          7.5
4   Subbu   8          4          7.0
5  Chinni   6          5          7.2
6     Nag   8          6          8.0
7     uma   6          6          6.5
8   srinu   6          7          6.5
9  Laxman   6          9          6.0
```

Exporting DataFrame to a CSV file

- **pandas.DataFrame.to_csv(path)**: Copies the contents with row index to a csv file specified by path.

In [28]:
```
df4.to_csv("student1.csv")
```

- **pandas.DataFrame.to_csv(path, index = False)**: Copies the contents without row index to a csv file specified by path.

Working with Excel

- **pandas.read_excel(path)**: Function to create a DataFrame from an excel file.
- **pandas.DataFrame.to_excel(path)**: Function to save a DataFrame to a excel file.

In [38]:
```
df6=pd.read_excel("sample.xlsx")
print(df6)
```

	Timestamp	Name	Height in Feet	\
0	2023-11-15 15:25:36.653	Sunny	5.11	
1	2023-11-15 15:26:10.979	Abc	5.40	
2	2023-11-15 15:26:24.689	Gvs	5.40	
3	2023-11-15 15:26:25.053	J Narasimha	6.10	
4	2023-11-15 15:26:26.174	Ram babu	5.70	
5	2023-11-15 15:26:35.244	madhu	5.20	
6	2023-11-15 15:26:35.563	Tejaswi	5.10	
7	2023-11-15 15:26:38.084	Satya Pavan	5.50	
8	2023-11-15 15:33:21.269	Dhoni	5.70	
9	2023-11-15 15:26:47.488	Naidu	5.50	
10	2023-11-15 15:26:47.648	Anuu	5.50	
11	2023-11-15 15:26:48.043	Likhith	5.90	
12	2023-11-15 15:26:50.136	Sherlock Holmes	5.60	
13	2023-11-15 15:26:50.910	Virat	5.90	
14	2023-11-15 15:26:56.314	Kakashi	6.00	
15	2023-11-15 15:26:58.226	Ammu	5.00	
16	2023-11-15 15:26:59.532	Sita	5.00	
17	2023-11-15 15:27:09.117	Abhi	5.70	
18	2023-11-15 15:27:18.746	Nazriya	5.00	
19	2023-11-15 15:27:19.439	Paddu	5.30	
20	2023-11-15 15:27:19.919	Tarak	5.70	
21	2023-11-15 15:27:32.222	Nandhu	5.90	
22	2023-11-15 15:27:32.893	Rohith	5.80	
23	2023-11-15 15:27:54.511	bhavya	5.10	
24	2023-11-15 15:28:05.685	Dhanush	5.70	
25	2023-11-15 15:28:11.849	Hasini	5.30	
26	2023-11-15 15:28:12.015	Varshini	5.20	
27	2023-11-15 15:28:15.787	Erisa	5.00	
28	2023-11-15 15:28:23.332	Deepika	5.80	
29	2023-11-15 15:28:23.690	Harshitha	5.20	
30	2023-11-15 15:28:24.943	Durga Akash	6.20	
31	2023-11-15 15:28:26.522	Swetha	5.40	
32	2023-11-15 15:28:28.179	Yunshu	5.00	
33	2023-11-15 15:28:36.821	Sri	4.00	
34	2023-11-15 15:28:43.321	Yeswanth	5.10	
35	2023-11-15 15:28:46.213	Siri	5.20	
36	2023-11-15 15:28:46.957	Naruto	6.00	
37	2023-11-15 15:28:47.908	Seetha	5.50	
38	2023-11-15 15:28:56.094	Kathyayani	5.00	
39	2023-11-15 15:28:57.632	Nikhil	5.10	
40	2023-11-15 15:29:02.083	Radhika	5.60	
41	2023-11-15 15:29:05.323	Sravya	5.10	
42	2023-11-15 15:29:08.756	Arjun	6.00	
43	2023-11-15 15:29:14.865	sri	5.00	
44	2023-11-15 15:29:32.386	Geddam Prasanth Kumar	5.10	
45	2023-11-15 15:29:32.580	Kavya	4.11	
46	2023-11-15 15:29:47.325	Harshitha	5.00	
47	2023-11-15 15:29:51.550	Mahima	5.00	
48	2023-11-15 15:30:02.478	Ramya	5.30	
49	2023-11-15 15:30:07.572	cseb	5.20	
50	2023-11-15 15:30:39.363	Rupa	5.40	
51	2023-11-15 15:30:42.254	Prem	5.00	
52	2023-11-15 15:31:55.891	Hari	5.60	
53	2023-11-15 15:31:57.390	csebg	6.10	
54	2023-11-15 15:32:02.662	Swami	5.75	
55	2023-11-15 15:32:30.334	Karthikeya	5.60	
56	2023-11-15 15:32:51.624	Divya	5.20	
57	2023-11-15 15:33:12.571	srija	5.10	

	Weight in KGs	Gender
0	76	M
1	53	F
2	54	F
3	93	M
4	92	M
5	54	F
6	55	F
7	90	M
8	83	M
9	80	M
10	58	F
11	65	M
12	62	M
13	69	M
14	80	M
15	38	F
16	65	F
17	55	M
18	45	F
19	55	F
20	56	M
21	68	M
22	85	M
23	41	F
24	71	M
25	52	F
26	50	F
27	34	F
28	56	F
29	53	F
30	48	M
31	47	F
32	42	F
33	40	F
34	64	M
35	65	F
36	75	M
37	51	F
38	35	F
39	45	M
40	58	F
41	40	F
42	75	M
43	55	F
44	83	M
45	45	F
46	45	F
47	55	F
48	68	F
49	45	F
50	54	F
51	50	M
52	68	M
53	80	M
54	63	M
55	60	M
56	68	F
57	65	F

Conclusion:

Date: 2c. Pandas DataFrame_Indexing

Aim:

Indexing

- To access a row or column of the DataFrame
- Types

    1. Label Based Indexing
    2. Location(Position) Based Indexing

    Note that all Indexing operations returns selected row or column as a Series.

1. Label Based Indexing

- Differenet Label Based Indexing Operations

    1. Select a row by label: **Series = DataFrame.loc[row_label]**
    2. Select a column by label

        a) Select a column by label: **Series = DataFrame[col_label]**
        b) Select a column by label with loc method: **Series = DataFrame.loc[: , col_label]**

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

In [42]:
```
# Create a dataFrame from a dictionary of lists with some row labels (We can u
d={"Name":["harshitha","bhavana","surya","goushi"],
    "Dept":["CSE","CSE","ECE","EEE"],
   "Percentage":[80.0,85.4,83.1,75.5]}
df=pd.DataFrame(d, index=["R1","R2","R3","R4"])
print(df)
```

```
        Name Dept  Percentage
R1  harshitha  CSE        80.0
R2    bhavana  CSE        85.4
R3      surya  ECE        83.1
R4     goushi  EEE        75.5
```

In [43]:
```
# Select a row by label
r1=df.loc["R1"]
print(r1)
print()
r2=df.loc["R2"]
print(r2)
print()
r3=df.loc["R3"]
print(r3)
print()
r4=df.loc["R4"]
print(r4)
```

```
Name          harshitha
Dept                CSE
Percentage         80.0
Name: R1, dtype: object

Name          bhavana
Dept              CSE
Percentage       85.4
Name: R2, dtype: object

Name          surya
Dept            ECE
Percentage     83.1
Name: R3, dtype: object

Name          goushi
Dept             EEE
Percentage      75.5
Name: R4, dtype: object
```

In [44]:
```
# Select a column by a label
c1=df["Name"]
print(c1)
print()
c2=df.Dept
print(c2)
print()
```

```
R1    harshitha
R2      bhavana
R3        surya
R4       goushi
Name: Name, dtype: object

R1    CSE
R2    CSE
R3    ECE
R4    EEE
Name: Dept, dtype: object
```

In [45]:
```
# Selecting a column by label with loc mehod
c3=df.loc[:,"Percentage"]
print(c3)
```

```
R1    80.0
R2    85.4
R3    83.1
R4    75.5
Name: Percentage, dtype: float64
```

In [46]:
```
# Select multiple columns as a new dataframe
df1=df[["Name","Dept"]]
print(df1)
```

```
         Name Dept
R1  harshitha  CSE
R2    bhavana  CSE
R3      surya  ECE
R4     goushi  EEE
```

In [47]:
```
# Select the rows where percentage is greater then 80
df2=df[df["Percentage"]>80]
print(df2)
```

```
       Name Dept  Percentage
R2  bhavana  CSE        85.4
R3    surya  ECE        83.1
```

2. Location(Position) Based Indexing ¶

- Location Based Indexing Operations

    1. Select a row by integer location: **Series = DataFrame.iloc[row_location]**
    2. Select a column by integer location **Series = DataFrame.iloc[: , col_location]**

In [48]:
```
# Select a row by position(integer location)
row1=df.iloc[3]
print(row1)
```

```
Name          goushi
Dept             EEE
Percentage      75.5
Name: R4, dtype: object
```

In [49]:
```
# Select a column by position(integer location)
col=df.iloc[:,1]
print(col)
```

```
R1    CSE
R2    CSE
R3    ECE
R4    EEE
Name: Dept, dtype: object
```

In [50]:
```
# Select multiple columns
col1=df.iloc[:,0:2]
print(col1)
```

```
         Name Dept
R1  harshitha  CSE
R2    bhavana  CSE
R3      surya  ECE
R4     goushi  EEE
```

Conclusion:

Date:

3.Visualization using matplotlib

Aim:

1. Import numpy library

In [1]:
```
import numpy as np
```

2. Create three ndarray objects: x with 100 values between 0 and 20, y with corresponding sin values of x and z with corresponding cos values of x.

Hint: You can use numpy.linspace(start, end, #values), numpy.sin(ndarray) and numpy.cos(ndarray) functions respectively for above task.

In [2]:
```
x=np.linspace(0,20,100)
y=np.sin(x)
z=np.cos(x)
```

In [3]:
```
### 3. Import matplotlib.pyplot module
```

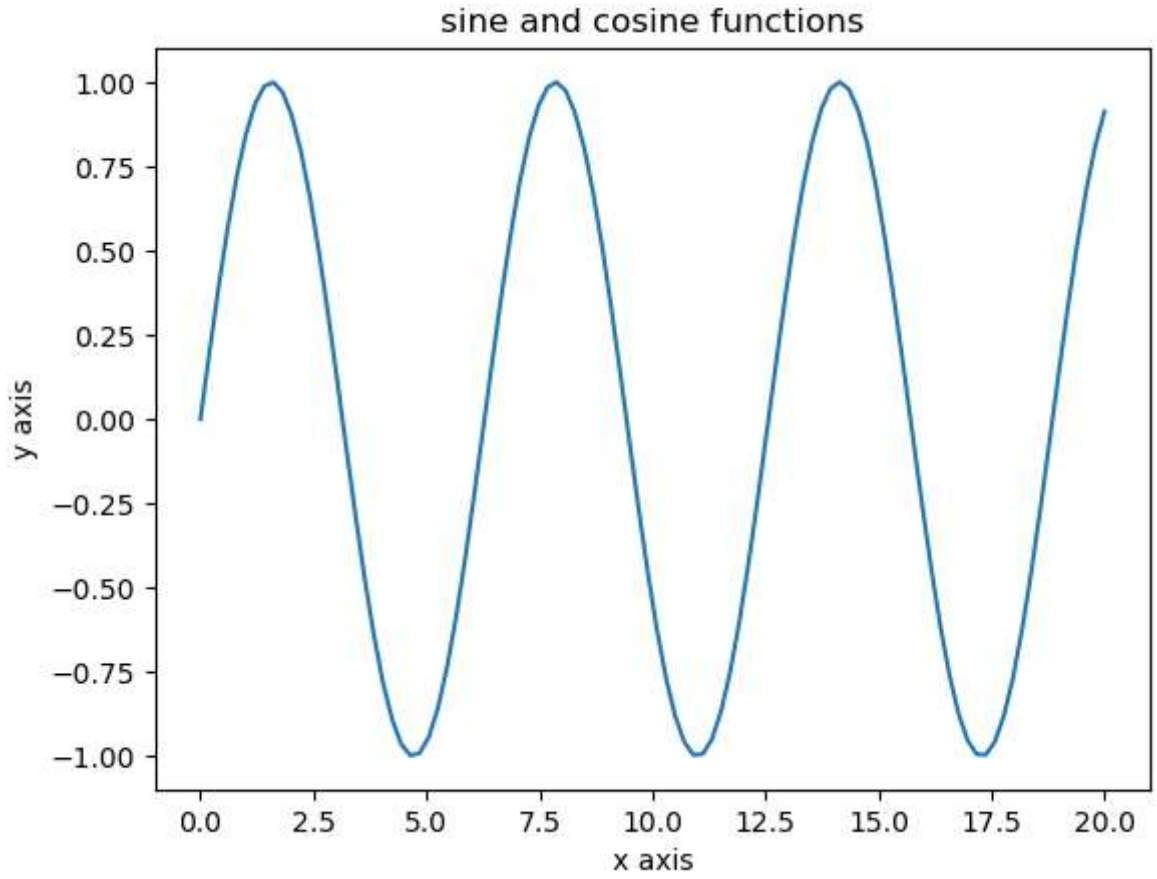
In [4]:
```
from matplotlib import pyplot as plt
```

4. Plot the values of y against x.

Hint: plot(x, y) and then show() functions from pyplot module can be used.

Hint: Additional functions title(text), xlabel(text) and ylabel(text) fcan also be used from pyplot.

In [5]:
```
plt.plot(x,y)
plt.xlabel("x axis")
plt.ylabel("y axis")
plt.title("sine and cosine functions")
plt.show()
```

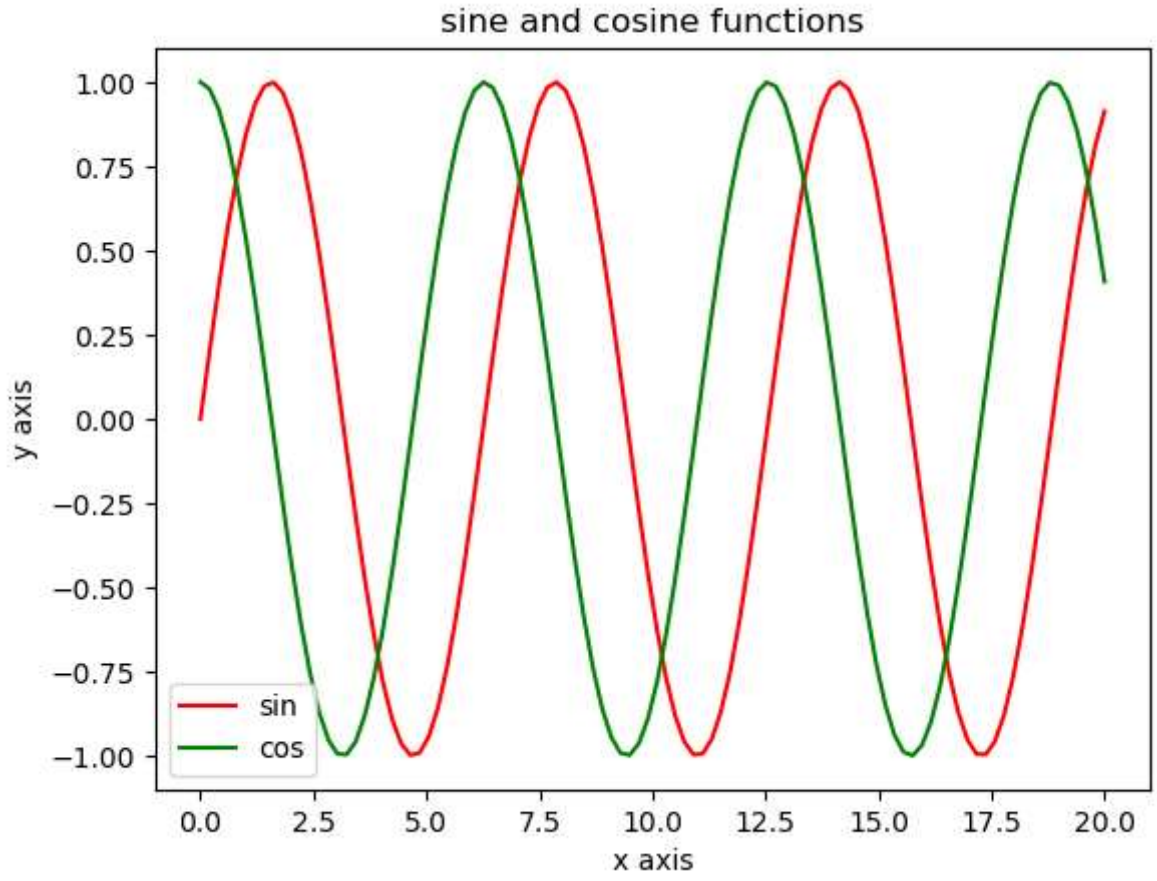


5. Plot the values of y and z against x with different line colors on the same window.

Hint 1: We can use third argument in plot function to represent color. "r" or "red" for red, "g" or "green" for green....

Hint 2: legend(list_of_strings) function can be used to describe the elements of the plot.

In [6]:
```
plt.plot(x,y,"r")
plt.plot(x,z,"g")
plt.xlabel("x axis")
plt.ylabel("y axis")
plt.title("sine and cosine functions")
plt.legend(["sin","cos"])
plt.show()
```

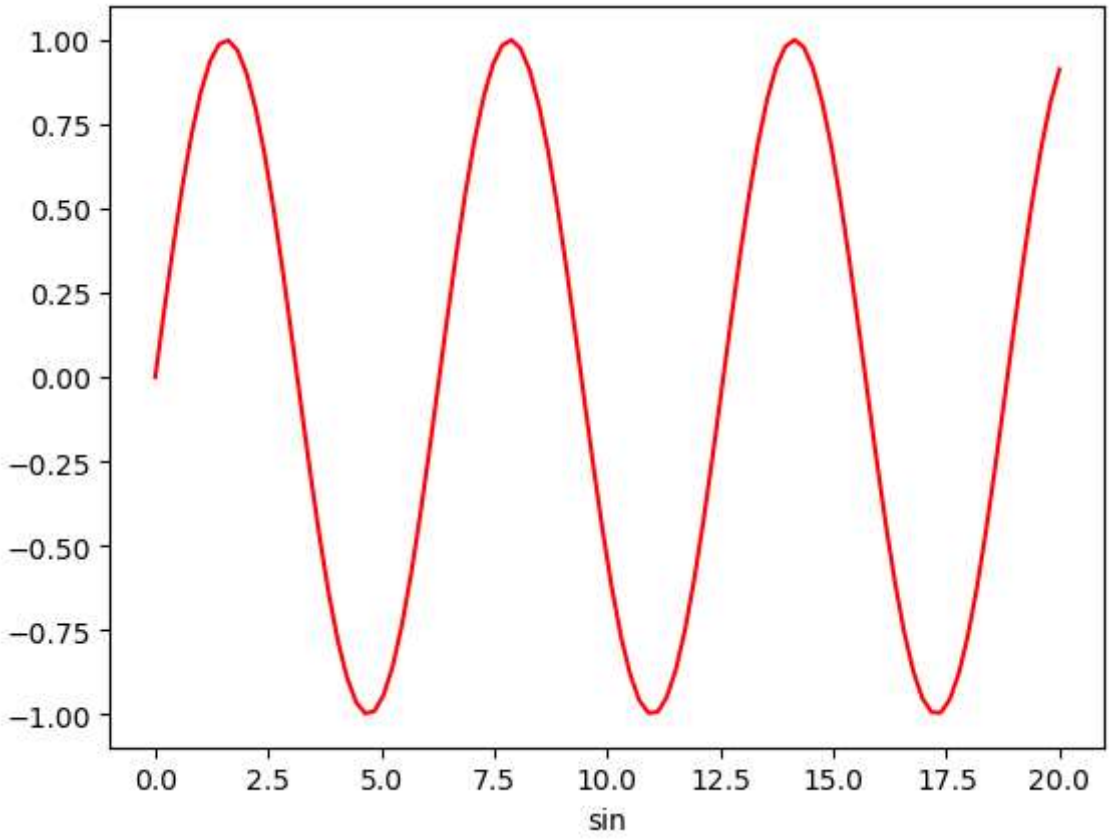


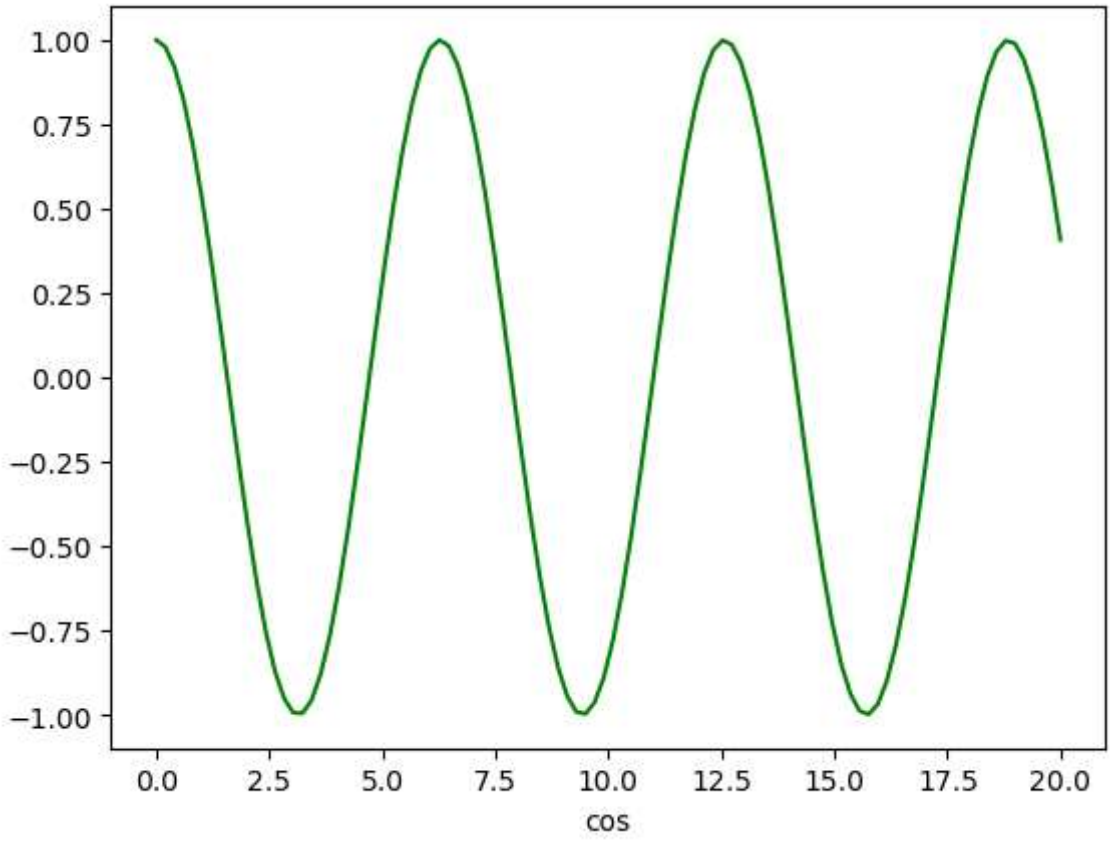
6. Plot the values of y and z against x with different line colors on seperate windows.

Hint 1: We can use third argument in plot function to represent color. "r" for red, "g" for green.... or we can also use keyword argument *color*.

Hint 2: We can use keyword argument *marker* to use requred marker in the position of data points.

In [12]:
```
plt.plot(x,y,"r")
plt.xlabel("sin")
plt.show()
plt.plot(x,z,"g")
plt.xlabel("cos")
plt.show()
```





7. Create two ndarray objects each with 25 random integers.

Hint: ndarray = numpy.random.randint(start, stop, size) ¶

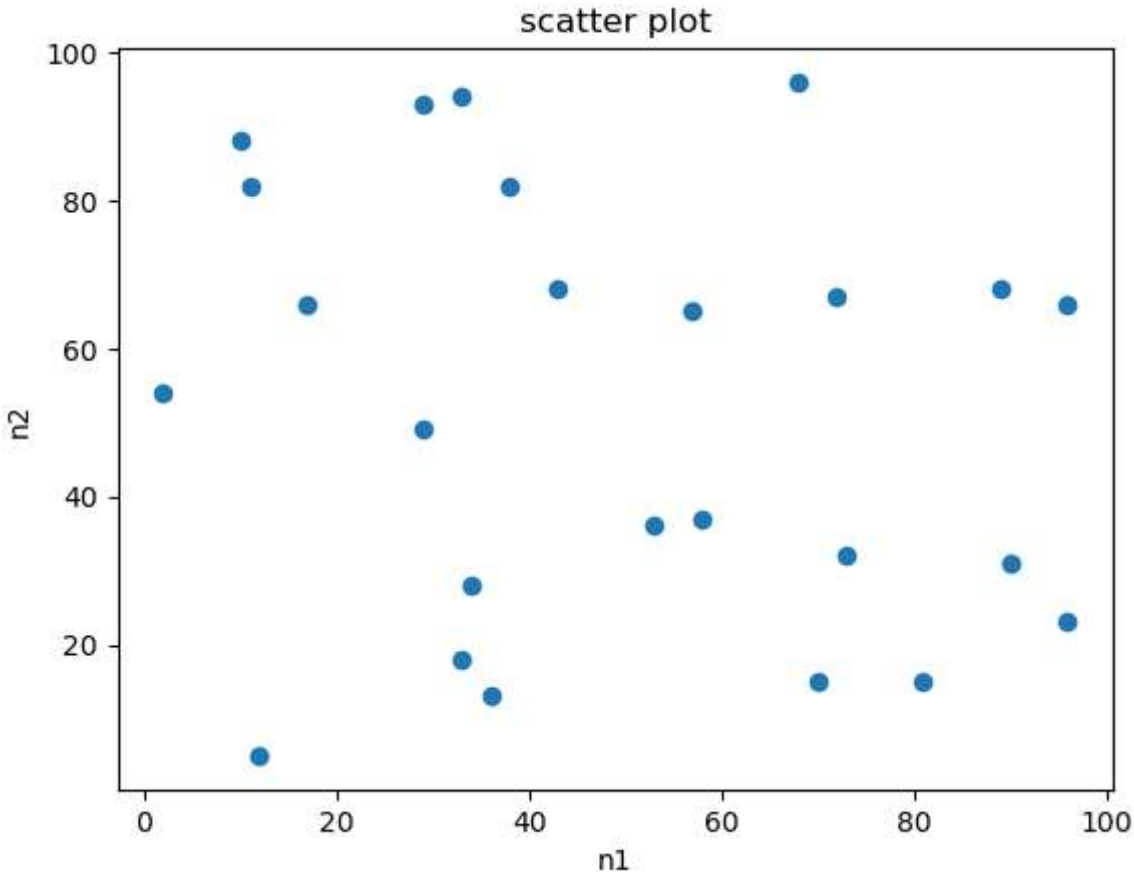
In [8]:
```
n1=np.random.randint(0,100,25)
print(n1)
n2=np.random.randint(0,100,25)
print(n2)
```

```
[10 38 70 53 33 68 34 72 90 89 57 12 43 36 29 96 58 17 11  2 29 81 33 73
 96]
[88 82 15 36 94 96 28 67 31 68 65  5 68 13 49 23 37 66 82 54 93 15 18 32
 66]
```

8. Display scatter plot between above two variables.

Hint: Use scatter(x, y) function from pyplot module.

In [9]:
```
plt.scatter(n1,n2)
plt.xlabel("n1")
plt.ylabel("n2")
plt.title("scatter plot")
plt.show()
```



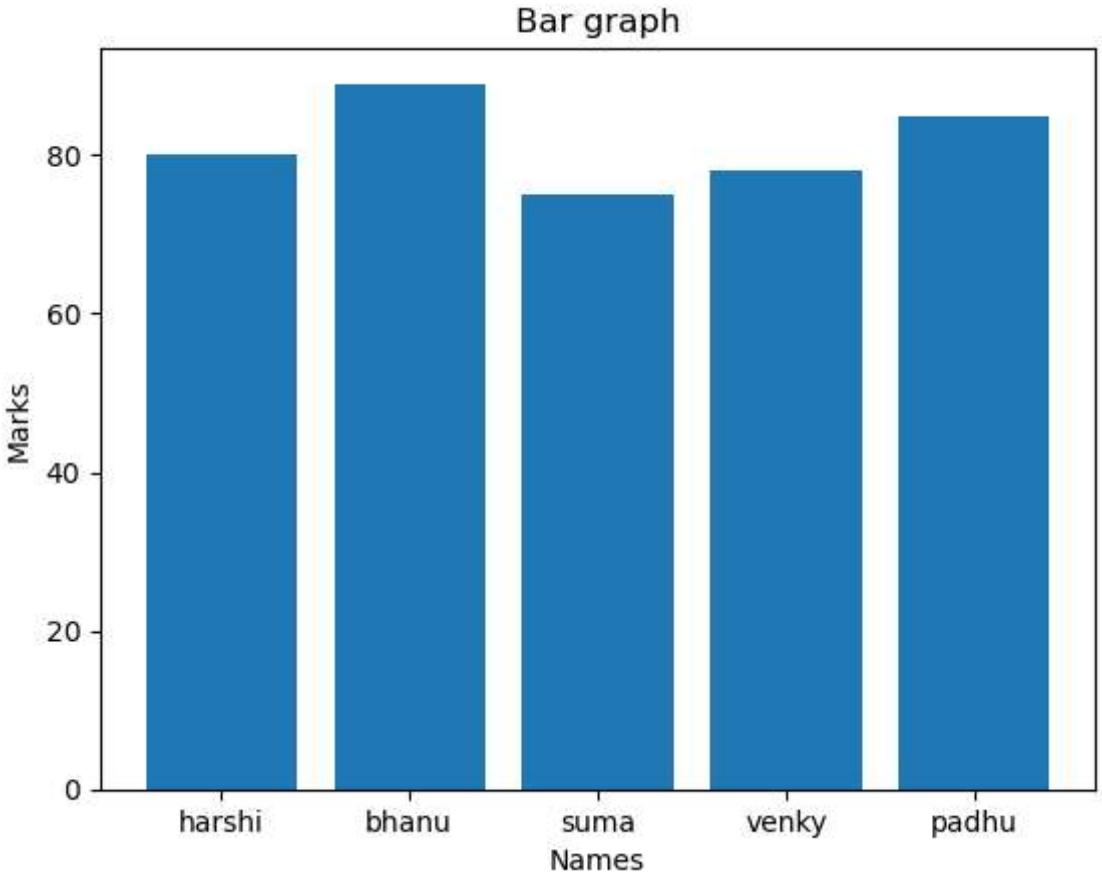
9. Create two lists: one with names of five students and other with their marks.

In [10]:
```
l1=["harshi","bhanu","suma","venky","padhu"]
l2=[80,89,75,78,85]
```

10. Display bar plot between above two variables.

Hint: Use bar(x, y) function from pyplot module.

In [11]:
```
plt.bar(l1,l2)
plt.xlabel("Names")
plt.ylabel("Marks")
plt.title("Bar graph")
plt.show()
```



Conclusion:

Date: 4.Preprocessing

Aim:

Objectives

1. Loading data.

2. Checking for missing values.

3. Splitting the data in to train set and test set.

Loading Data

In [5]:
```
# Load the data from external flat file - use pandas.read_csv() or pandas.read_excel()
import pandas as pd
df=pd.read_csv("573Churn-Modelling.csv")
print(type(df))
print(df.shape)
print()
print(df.columns)
print()
print(df.head())
```

```
<class 'pandas.core.frame.DataFrame'>
(10000, 7)

Index(['CreditScore', 'Age', 'Tenure', 'Balance', 'HasCrCard', 'Salary',
       'Exited'],
      dtype='object')

   CreditScore  Age  Tenure    Balance  HasCrCard     Salary  Exited
0          619   42       2       0.00          1  101348.88       1
1          608   41       1   83807.86          0  112542.58       0
2          502   42       8  159660.80          1  113931.57       1
3          699   39       1       0.00          0   93826.63       0
4          850   43       2  125510.82          1   79084.10       0
```

In [ ]:
```
# Load the data from sklearn.datasets - load_iris(), load_diabetes(), load_digits(), l
# Parameter return_X_y=True returns the data set as separate input and output nupy arr
# Parameter as_frame=True returns the data as a dataframe.
```

In [18]:
```
#Loading dataset as Bunch object
from sklearn.datasets import load_iris
Dataset=load_iris()
print(type(Dataset))
x=Dataset.data
print(x.shape)
print(type(x))
print()
print(x[0:5])
print()
print(Dataset.feature_names)
y=Dataset.target
print(type(y))
print()
```

```
print(y[145:150])
print()
print(Dataset.target_names)
```

```
<class 'sklearn.utils._bunch.Bunch'>
(150, 4)
<class 'numpy.ndarray'>

[[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]]

['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
<class 'numpy.ndarray'>

[2 2 2 2 2]

['setosa' 'versicolor' 'virginica']
```

In [23]:
```
#loading data seperately as input and output numpy arrays
x,y=load_iris(return_X_y=True)
print(type(x))
print()
print(x[0:5])
print()
print(type(y))
print()
print(y[0:5])
```

```
<class 'numpy.ndarray'>

[[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]]

<class 'numpy.ndarray'>

[0 0 0 0 0]
```

In [24]:
```
#loading data seperately as input and output dataframe
x,y=load_iris(return_X_y=True,as_frame=True)
print(type(x))
print()
print(x[0:5])
print()
print(type(y))
print()
print(y[0:5])
```

```
<class 'pandas.core.frame.DataFrame'>

   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

<class 'pandas.core.series.Series'>

0    0
1    0
2    0
3    0
4    0
Name: target, dtype: int32
```

pandas.isna(object)

Detects missing values.

Returns a boolen object of same size.

None and np.NaN are mapped True values. Everything else gets mapped to False values.

In [26]:
```
df.isna().sum()
```

Out[26]:
```
CreditScore    0
Age            0
Tenure         0
Balance        0
HasCrCard      0
Salary         0
Exited         0
dtype: int64
```

In [31]:
```
x=df.drop("Exited",axis=1)
y=df["Exited"]
print(x.head())
print()
print(y.head())
```

```
   CreditScore  Age  Tenure    Balance  HasCrCard     Salary
0          619   42       2       0.00          1  101348.88
1          608   41       1   83807.86          0  112542.58
2          502   42       8  159660.80          1  113931.57
3          699   39       1       0.00          0   93826.63
4          850   43       2  125510.82          1   79084.10

0    1
1    0
2    1
3    0
4    0
Name: Exited, dtype: int64
```

x_train, x_test, y_tran, y_test = train_test_split(x, y, test_size)

Defined in sklearn.model_selection

test_size attribute decides the proportion of split

In [36]:
```
# Split the data
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test =train_test_split(x, y, test_size=0.3)
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(7000, 6)
(7000,)
(3000, 6)
(3000,)
```

We can use random_state keyword argument to get same set of training and testing samples.

In [41]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test =train_test_split(x, y, test_size=0.3,random_state=19
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(7000, 6)
(7000,)
(3000, 6)
(3000,)
```

Conclusion:

Date:

5. Naive Bayes Classifier

Aim:

Step 1: Load Dataset and split it

sklearn.datasets.load_iris()

Loads and returns iris dataset as a Bunch object with the following attributes.

--> data: ndarray object of input samples of size (150, 4)

--> target: ndarray object of taget values of size (150,)

This data sets consists of 3 different types of irises' -> Setosa, Versicolour, and Virginica.

The rows being the samples and the columns being: Sepal Length, Sepal Width, Petal Length and Petal Width.

Iris Versicolor **Iris Setosa** **Iris Virginica**

Other attributes:

--> feature_names: List of all input feature names

--> target_names: List of all target classes

- We can use **as_frame = True** to get data as pandas DataFrame and target as pandas Series/DataFrame.

In [1]:
```
from sklearn.datasets import load_iris
Dataset=load_iris()
print(f"Feature names:{Dataset.feature_names}")
print()
print(f"target names:{Dataset.target_names}")
```

```
Feature names:['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal w
idth (cm)']

target names:['setosa' 'versicolor' 'virginica']
```

In [2]:
```
x=Dataset.data
y=Dataset.target
```

```
print("Input Shape:",x.shape)
print("Output Shape:",y.shape)
```

```
Input Shape: (150, 4)
Output Shape: (150,)
```

sklearn.model_selection.train_test_split(x, y, test_size)

Returns the train and test sets from given data set

In [3]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test =train_test_split(x, y, test_size=0.3)
print("training set shapes:")
print(x_train.shape)
print(y_train.shape)
print("testing set shapes:")
print(x_test.shape)
print(y_test.shape)
```

```
training set shapes:
(105, 4)
(105,)
testing set shapes:
(45, 4)
(45,)
```

Step 2: Build the model

sklearn.naive_bayes.GaussianNB()

--> Creates a Gaussian Naive Bayes classifier object.

--> fit(x_train, y_train) method train the model with train data.

--> predict(x_test) method predicts the output for given test input.

In [4]:
```
from sklearn.naive_bayes import GaussianNB
modelGNB=GaussianNB()
modelGNB.fit(x_train,y_train)
```

Out[4]:
```
GaussianNB
GaussianNB()
```

Step 3: Test the model

In [5]:
```
y_predict=modelGNB.predict(x_test)
```

sklearn.metrics.accuracy_score(y_pred, y_test)

--> Returns the accuracy from predicted and actual test label values

In [6]:
```
from sklearn.metrics import confusion_matrix,accuracy_score
con_mat=confusion_matrix(y_test,y_predict)
print("confusion matrix:",con_mat)
print()
accuracy=accuracy_score(y_test,y_predict)
print("Accuracy:",accuracy)
```

```
confusion matrix: [[16  0  0]
 [ 0 17  1]
 [ 0  0 11]]

Accuracy: 0.9777777777777777
```

Perform the classification using Multinomial Naive Bayes Classifier

Classifier object can be created by importing *MultinomialNB* class from sklearn.naive_bayes module

In [7]:
```
#training
from sklearn.naive_bayes import MultinomialNB
modelNB=MultinomialNB()
modelNB.fit(x_train,y_train)
```

Out[7]:
```
MultinomialNB()
```

In [8]:
```
#test the model
y_predict=modelNB.predict(x_test)
```

In [9]:
```
con_mat=confusion_matrix(y_test,y_predict)
print("confusion matrix:",con_mat)
print()
accuracy=accuracy_score(y_test,y_predict)
print("Accuracy:",accuracy)
```

```
confusion matrix: [[16  0  0]
 [ 0  0 18]
 [ 0  0 11]]

Accuracy: 0.6
```

Conclusion:

Date:

6.Logistic Regression

Aim:

Step 1: Load the data set, Check if any missing vaues and split it into train & test sets.

In [5]:
```
from sklearn.datasets import load_breast_cancer
dataset=load_breast_cancer()
print("Feature Names:",dataset.feature_names)
print("target Names:",dataset.target_names)
```

```
Feature Names: ['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
target Names: ['malignant' 'benign']
```

In [6]:
```
import pandas as pd
x=dataset.data
y=dataset.target
print("Input shape:",x.shape)
print("output shape:",y.shape)
```

```
Input shape: (569, 30)
output shape: (569,)
```

In [7]:
```
import pandas as pd
pd.isna(pd.DataFrame(x)).sum()
```

Out[7]:
```
0     0
1     0
2     0
3     0
4     0
5     0
6     0
7     0
8     0
9     0
10    0
11    0
12    0
13    0
14    0
15    0
16    0
17    0
18    0
19    0
20    0
21    0
22    0
23    0
24    0
25    0
26    0
27    0
28    0
29    0
dtype: int64
```

In [8]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=7)
print("Training set shapes:")
print(x_train.shape)
print(y_train.shape)
print("Testing set shapes:")
print(x_test.shape)
print(y_test.shape)
```

```
Training set shapes:
(398, 30)
(398,)
Testing set shapes:
(171, 30)
(171,)
```

Step 2: Build the model

sklearn.linear_model.LogisticRegression class can be used to cleate model object

fit() method can be used to train the model

predict() method can be used to predict the outputs for unseen data

In [10]:
```
from sklearn.linear_model import LogisticRegression
modelLR=LogisticRegression()
modelLR.fit(x_train,y_train)
```

```
C:\Users\HP\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:460: Conver
genceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

Out[10]:
```
LogisticRegression()
```

Step 3: Test the model performance

we can import various metrics(confusion_matrix, accuracy_score, recall_score, precision_score, f1_score) from sklearn.metrics

In [11]:
```
y_pred=modelLR.predict(x_test)
```

In [12]:
```
from sklearn.metrics import confusion_matrix, accuracy_score, recall_score, precision_
print("Confusion Matrics",confusion_matrix(y_test,y_pred))
print("Accuracy",accuracy_score(y_test,y_pred))
print("Recall",recall_score(y_test,y_pred))
print("precision",precision_score(y_test,y_pred))
print("f1_score",f1_score(y_test,y_pred))
```

```
Confusion Matrics [[ 48   7]
 [  1 115]]
Accuracy 0.9532163742690059
Recall 0.9913793103448276
precision 0.9426229508196722
f1_score 0.9663865546218487
```

Conclusion:

Date:

7a.KNN-For Classification

Aim:

Step 1: Load breast cancer dataset from sklean and split it as train and test sets.

sklearn.datasets.load_breast_cancer()

--> Number of classes: 2 ['malignant', 'benign']

--> Total samples: 569 ['malignant' 212, 'benign' 357]

--> Dimensionality: 30

Returns dictionary-like Bunch object with following attributes

--> data: The ndarray of input data samples with a shape (569, 30)

--> target: The ndarray of classification target with a shape (569,)

--> feature_names: List of feature names

--> target_names: List of target class names

In [5]:
```
from sklearn.datasets import load_breast_cancer
d=load_breast_cancer()
print("Feature names",d.feature_names)
print("target classes",d.target_names)
print()
x=d.data
y=d.target
print("Input shape:",x.shape)
print("output shape:",y.shape)
```

```
Feature names ['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
target classes ['malignant' 'benign']

Input shape: (569, 30)
output shape: (569,)
```

In [10]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=7)
print("Training data input shape:",x_train.shape)
print("Training data input shape:",y_train.shape)
print("Testing data output shape:",x_test.shape)
print("Testing data output shape:",y_test.shape)
```

```
Training data input shape: (398, 30)
Training data input shape: (398,)
Testing data output shape: (171, 30)
Testing data output shape: (171,)
```

Step 2: Build the model

Import KNearestClassifier from sklearn.neighbors

--> Object creation: model = KNeighborsClassifier(n_neighbors)

In [12]:
```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=4)
knn.fit(x_train,y_train)
```

Out[12]:
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=4)
```

Step 3: Test the model

We can import confusion_matrix, accuracy_score, recall_score, precision_score, f1_score functions from sklearn.metrics

In [17]:
```
y_pred=knn.predict(x_test)
```

In [22]:
```
from sklearn.metrics import confusion_matrix, accuracy_score, recall_score, precision_
print("Confusion Matrix:",confusion_matrix(y_test,y_pred))
print("accuracy:",accuracy_score(y_test,y_pred))
print("Recall:",recall_score(y_test,y_pred))
print("precision:",precision_score(y_test,y_pred))
print("f1_score:",f1_score(y_test,y_pred))
```

```
Confusion Matrix: [[ 50   5]
 [ 10 106]]
accuracy: 0.9122807017543859
Recall: 0.9137931034482759
precision: 0.954954954954955
f1_score: 0.9339207048458151
```

Conclusion:

Date:

7b.KNN-Finding optimal k value

Aim:

Step 1: Load the dataset and split it

In [27]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
d=load_breast_cancer()
x=d.data
y=d.target
print("Input shape:",x.shape)
print("output shape:",y.shape)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=7)
print("Training data input shape:",x_train.shape)
print("Training data input shape:",y_train.shape)
print("Testing data output shape:",x_test.shape)
print("Testing data output shape:",y_test.shape)
```

```
Input shape: (569, 30)
output shape: (569,)
Training data input shape: (398, 30)
Training data input shape: (398,)
Testing data output shape: (171, 30)
Testing data output shape: (171,)
```

Step 2: Find training accuracy and testing accuracy scores for a range of K values

Step 1: Start with empty lists for both training and testing accuracies.

Step 2: For each K value in the selected range, do the following

```
    --> Build the model.

    --> Find the accuracy with training data and append it to list of
training accuracies.

    --> Find the accuracy with testing data and append it to list of
testing accuracies.
```

In [28]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
k_values=range(1,26,2)
train_accuracies=[]
test_accuracies=[]
for k in k_values:
    knn=KNeighborsClassifier(n_neighbors=k)
    knn.fit(x_train,y_train)
    train_accuracy=accuracy_score(y_train,knn.predict(x_train))
    train_accuracies.append(train_accuracy)
    test_accuracy=accuracy_score(y_test,knn.predict(x_test))
    test_accuracies.append(test_accuracy)
print(train_accuracies)
print(test_accuracies)
```

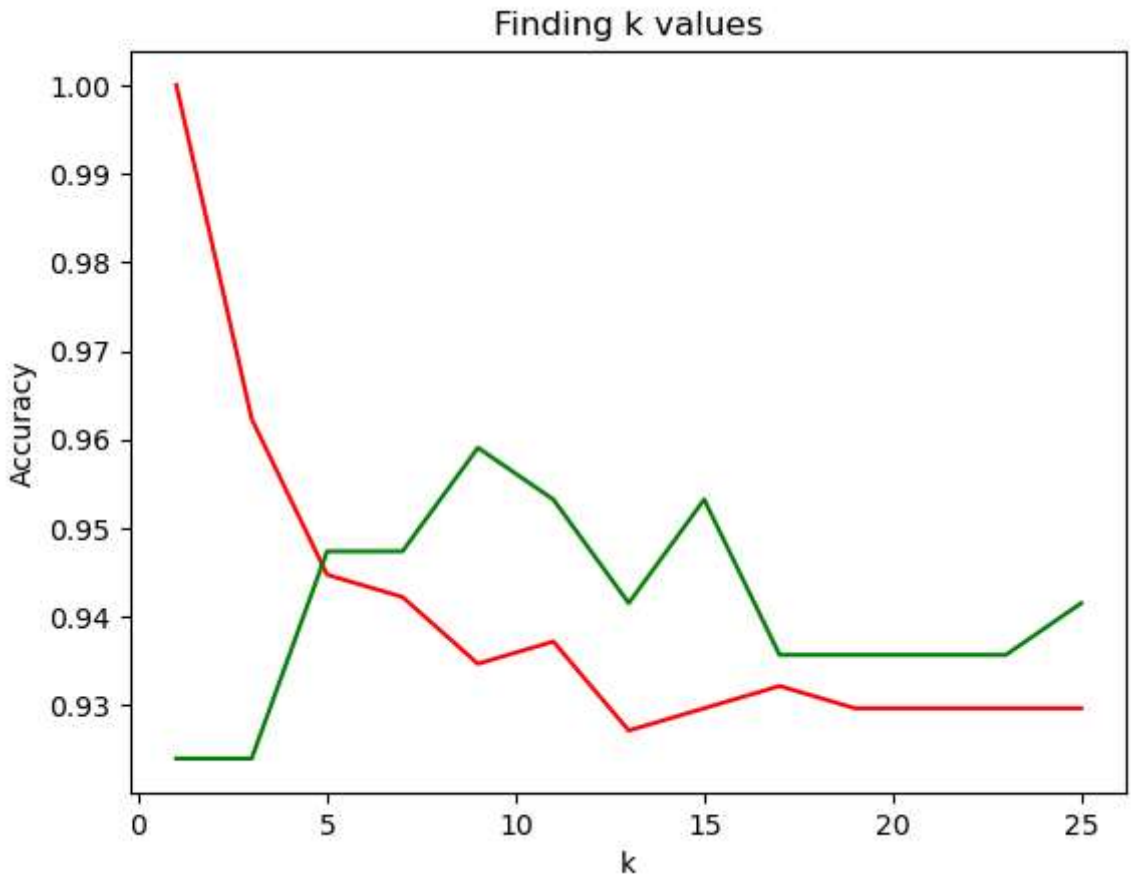
```
[1.0, 0.9623115577889447, 0.9447236180904522, 0.9422110552763819, 0.9346733668341709,
0.9371859296482412, 0.9271356783919598, 0.9296482412060302, 0.9321608040201005, 0.929
6482412060302, 0.9296482412060302, 0.9296482412060302, 0.9296482412060302]
[0.9239766081871345, 0.9239766081871345, 0.9473684210526315, 0.9473684210526315, 0.95
90643274853801, 0.9532163742690059, 0.9415204678362573, 0.9532163742690059, 0.9356725
14619883, 0.935672514619883, 0.935672514619883, 0.935672514619883, 0.941520467836257
3]
```

Step 3: Plot training & testing accuracies against the range of K values and find the optimal value for K from it.

Hint: Pick a K value where you dont find any overfitting or underfitting.

In [34]:
```
import matplotlib.pyplot as plt
plt.plot(k_values,train_accuracies,color="red")
plt.plot(k_values,test_accuracies,color="green")
plt.xlabel("k")
plt.ylabel("Accuracy")
plt.title("Finding k values")
```

Out[34]:
```
Text(0.5, 1.0, 'Finding k values')
```



In [ ]:
```
from this plot the best choice for k is 5
```

Conclusion:

Date: 8.SVM

Aim:

Step 1: Import the digits dataset from sklearn and split it into train and test sets.

Digits dataset in sklearn.datasets:

This dataset is made up of 1797 8x8 images.

- Number of classes: 10 --> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- Number of samples: 1797
- Number of features: 64
- Each sample represents the 64 pixel values of a 8x8 image.
- Range of each pixel value is 0 - 16.

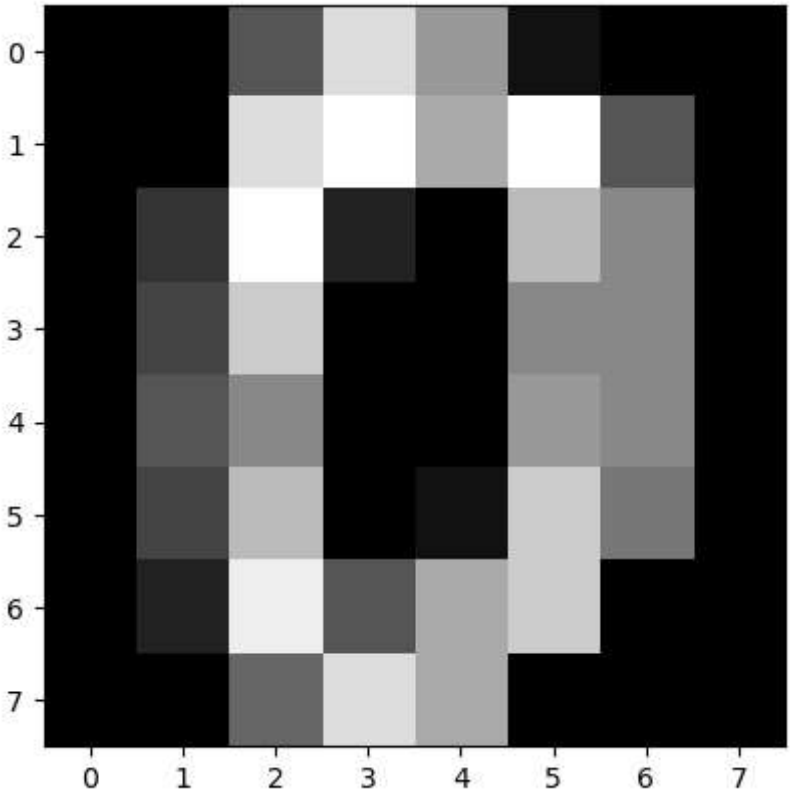
In [47]:
```
from sklearn.datasets import load_digits
d=load_digits()
print("feature names\n",d.feature_names)
print("target names\n",d.target_names)
```

```
feature names
 ['pixel_0_0', 'pixel_0_1', 'pixel_0_2', 'pixel_0_3', 'pixel_0_4', 'pixel_0_
5', 'pixel_0_6', 'pixel_0_7', 'pixel_1_0', 'pixel_1_1', 'pixel_1_2', 'pixel_1
_3', 'pixel_1_4', 'pixel_1_5', 'pixel_1_6', 'pixel_1_7', 'pixel_2_0', 'pixel_
2_1', 'pixel_2_2', 'pixel_2_3', 'pixel_2_4', 'pixel_2_5', 'pixel_2_6', 'pixel
_2_7', 'pixel_3_0', 'pixel_3_1', 'pixel_3_2', 'pixel_3_3', 'pixel_3_4', 'pixe
l_3_5', 'pixel_3_6', 'pixel_3_7', 'pixel_4_0', 'pixel_4_1', 'pixel_4_2', 'pix
el_4_3', 'pixel_4_4', 'pixel_4_5', 'pixel_4_6', 'pixel_4_7', 'pixel_5_0', 'pi
xel_5_1', 'pixel_5_2', 'pixel_5_3', 'pixel_5_4', 'pixel_5_5', 'pixel_5_6', 'p
ixel_5_7', 'pixel_6_0', 'pixel_6_1', 'pixel_6_2', 'pixel_6_3', 'pixel_6_4',
'pixel_6_5', 'pixel_6_6', 'pixel_6_7', 'pixel_7_0', 'pixel_7_1', 'pixel_7_2',
'pixel_7_3', 'pixel_7_4', 'pixel_7_5', 'pixel_7_6', 'pixel_7_7']
target names
 [0 1 2 3 4 5 6 7 8 9]
```

In [48]:
```
x=d.data
y=d.target
print(x.shape)
print(y.shape)
```

```
(1797, 64)
(1797,)
```

In [49]:
```
import matplotlib.pyplot as plt
img=x[0].reshape((8,8))
plt.imshow(img,cmap="gray")
plt.show()
print("class",y[1])
```




```
class 1
```

In [50]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=
print("Training set shape")
print(x_train.shape)
print(y_train.shape)
print("Testing set shape")
print(x_test.shape)
print(y_test.shape)
```

```
Training set shape
(1257, 64)
(1257,)
Testing set shape
(540, 64)
(540,)
```

Step 2: Build the model

- Import the class SVC from sklearn.svm module
- Create a model objet using SVC class

- Use fit method to train the model with training dataset.

In [51]:
```
from sklearn.svm import SVC
svm1=SVC()
svm1.fit(x_train,y_train)
```

Out[51]:
```
SVC()
```

Step 3: Test and Evaluate the model.

- Use predict method to predict the class for unseen data samples.
- Import accuracy_score function from sklearn.metrics module for evaluation

In [52]:
```
y_pred=svm1.predict(x_test)
```

In [61]:
```
from sklearn.metrics import confusion_matrix,accuracy_score
print("Confusion matrix:\n",confusion_matrix(y_test,y_pred))
print("Accuracy is:",accuracy_score(y_test,y_pred))
```

```
Confusion matrix:
 [[60  0  0  0  1  0  0  0  0  0]
 [ 0 55  0  0  0  0  0  0  0  0]
 [ 0  0 53  0  0  0  0  0  0  0]
 [ 0  0  0 52  0  0  0  0  0  0]
 [ 0  0  0  0 58  0  0  0  0  0]
 [ 0  0  0  0  0 48  0  0  0  0]
 [ 0  0  0  0  0  0 48  0  0  0]
 [ 0  0  0  0  0  0  0 49  0  0]
 [ 0  2  0  0  0  0  0  1 59  0]
 [ 0  0  0  0  0  1  0  0  0 53]]
Accuracy is: 0.9907407407407407
```

Step 4: Evaluate the model with different kernels

- SVC class takes a parameter kernel which can take one of linear / poly / rbf / sigmoid.
- rbf (Radial Basis Function) is the default value.

In [58]:
```
from sklearn.metrics import accuracy_score
svm2=SVC(kernel="linear")
svm2.fit(x_train,y_train)
y_pred1=svm2.predict(x_test)
print("Accuracy is:",accuracy_score(y_test,y_pred1))
```

```
Accuracy is: 0.9592592592592593
```

In [59]:
```
from sklearn.metrics import accuracy_score
svm3=SVC(kernel="poly")
svm3.fit(x_train,y_train)
y_pred2=svm3.predict(x_test)
print("Accuracy is:",accuracy_score(y_test,y_pred2))
```

```
Accuracy is: 0.987037037037037
```

In [60]:
```
from sklearn.metrics import accuracy_score
svm4=SVC(kernel="sigmoid")
svm4.fit(x_train,y_train)
y_pred3=svm4.predict(x_test)
print("Accuracy is:",accuracy_score(y_test,y_pred3))
```

```
Accuracy is: 0.8981481481481481
```

Conclusion:

Date:

9.Decision Tree Classification

Aim:

Step 1:Import the wine dataset and split it into training and testing parts

In [74]:
```
from sklearn.datasets import load_wine
d=load_wine()
print("feature names:",d.feature_names)
print()
print("target classes:",d.target_names)
x=d.data
y=d.target
print()
print("shape of input data:",x.shape)
print("shape of output data:",y.shape)
```

```
feature names: ['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium', 'to
tal_phenols', 'flavanoids', 'nonflavanoid_phenols', 'proanthocyanins', 'color_intensi
ty', 'hue', 'od280/od315_of_diluted_wines', 'proline']

target classes: ['class_0' 'class_1' 'class_2']

shape of input data: (178, 13)
shape of output data: (178,)
```

In [75]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.15,random_state=7)
print("Training data input shape:",x_train.shape)
print("Training data output shape:",y_train.shape)
print("Testing data input shape:",x_test.shape)
print("Testing data output shape:",y_test.shape)
```

```
Training data input shape: (151, 13)
Training data output shape: (151,)
Testing data input shape: (27, 13)
Testing data output shape: (27,)
```

Step 2: Build the tree

Import the class *DecisionTreeClassifier* from sklearn.tree module.

We can use parameter *croterion* to set the metric for the split. Possible metrics are 'gini' (default), 'entropy' or 'log_loss'.

In [76]:
```
from sklearn.tree import DecisionTreeClassifier
t=DecisionTreeClassifier()
t.fit(x_train,y_train)
```

Out[76]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [77]:
```
from sklearn.tree import DecisionTreeClassifier
t1=DecisionTreeClassifier(criterion="entropy")
t1.fit(x_train,y_train)
```

Out[77]:

```
DecisionTreeClassifier(criterion='entropy')
```

Step 3: Evaluate the performance of the tree

--> Compute both Training Accuracy and Test Accuracy to check overfitting

In [78]:

```
from sklearn.metrics import accuracy_score
print("Training accuracy:",accuracy_score(y_train,t.predict(x_train)))
print("Testing accuracy:",accuracy_score(y_test,t.predict(x_test)))
```

```
Training accuracy: 1.0
Testing accuracy: 0.9259259259259259
```

In [79]:

```
from sklearn.metrics import accuracy_score
print("Training accuracy:",accuracy_score(y_train,t1.predict(x_train)))
print("Testing accuracy:",accuracy_score(y_test,t1.predict(x_test)))
```

```
Training accuracy: 1.0
Testing accuracy: 0.9259259259259259
```

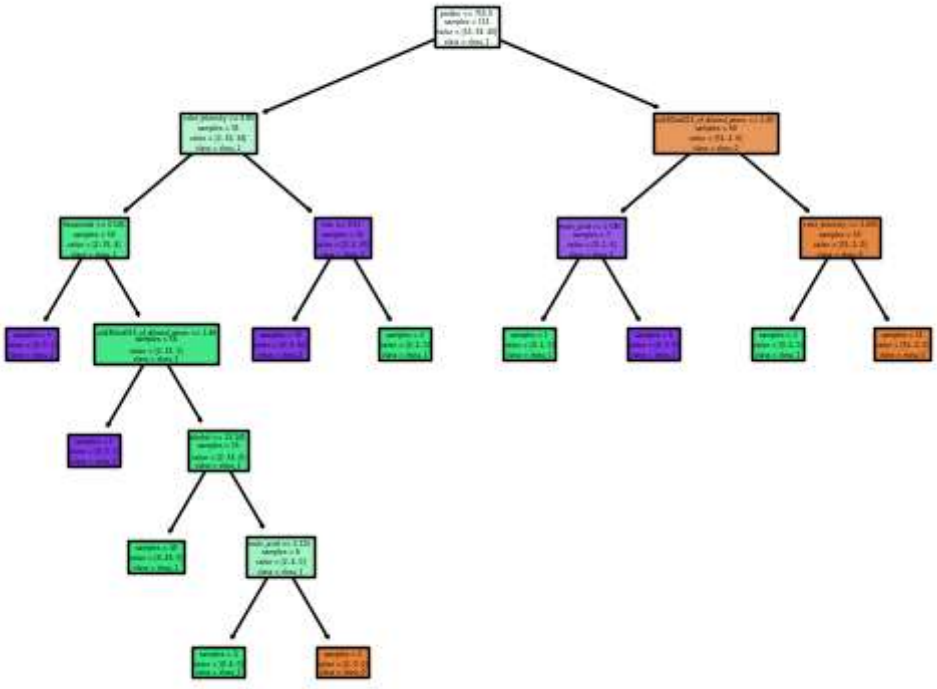
Step 4: Visualize the tree

Visual tree can be generated using *sklearn.tree.export_graphviz(tree, out_file, class_names, feature_names, impurity, filled)* function. Which return a .dot file with the required graph.

In [80]:

```
from sklearn.tree import plot_tree
plot_tree(t,class_names=['class_0', 'class_1', 'class_2'],feature_names=d.feature_name
```

Out[80]:
```
[Text(0.5, 0.9285714285714286, 'proline <= 755.0\nsamples = 151\nvalue = [53, 58, 40]
\nclass = class_1'),
 Text(0.25, 0.7857142857142857, 'color_intensity <= 4.85\nsamples = 91\nvalue = [2, 5
5, 34]\nclass = class_1'),
 Text(0.125, 0.6428571428571429, 'flavanoids <= 0.545\nsamples = 59\nvalue = [2, 53,
4]\nclass = class_1'),
 Text(0.0625, 0.5, 'samples = 3\nvalue = [0, 0, 3]\nclass = class_2'),
 Text(0.1875, 0.5, 'od280/od315_of_diluted_wines <= 1.48\nsamples = 56\nvalue = [2, 5
3, 1]\nclass = class_1'),
 Text(0.125, 0.35714285714285715, 'samples = 1\nvalue = [0, 0, 1]\nclass = class_2'),
 Text(0.25, 0.35714285714285715, 'alcohol <= 13.145\nsamples = 55\nvalue = [2, 53, 0]
\nclass = class_1'),
 Text(0.1875, 0.21428571428571427, 'samples = 49\nvalue = [0, 49, 0]\nclass = class_
1'),
 Text(0.3125, 0.21428571428571427, 'malic_acid <= 2.125\nsamples = 6\nvalue = [2, 4,
0]\nclass = class_1'),
 Text(0.25, 0.07142857142857142, 'samples = 4\nvalue = [0, 4, 0]\nclass = class_1'),
 Text(0.375, 0.07142857142857142, 'samples = 2\nvalue = [2, 0, 0]\nclass = class_0'),
 Text(0.375, 0.6428571428571429, 'hue <= 0.91\nsamples = 32\nvalue = [0, 2, 30]\nclas
s = class_2'),
 Text(0.3125, 0.5, 'samples = 30\nvalue = [0, 0, 30]\nclass = class_2'),
 Text(0.4375, 0.5, 'samples = 2\nvalue = [0, 2, 0]\nclass = class_1'),
 Text(0.75, 0.7857142857142857, 'od280/od315_of_diluted_wines <= 2.49\nsamples = 60\n
value = [51, 3, 6]\nclass = class_0'),
 Text(0.625, 0.6428571428571429, 'malic_acid <= 2.085\nsamples = 7\nvalue = [0, 1, 6]
\nclass = class_2'),
 Text(0.5625, 0.5, 'samples = 1\nvalue = [0, 1, 0]\nclass = class_1'),
 Text(0.6875, 0.5, 'samples = 6\nvalue = [0, 0, 6]\nclass = class_2'),
 Text(0.875, 0.6428571428571429, 'color_intensity <= 3.435\nsamples = 53\nvalue = [5
1, 2, 0]\nclass = class_0'),
 Text(0.8125, 0.5, 'samples = 2\nvalue = [0, 2, 0]\nclass = class_1'),
 Text(0.9375, 0.5, 'samples = 51\nvalue = [51, 0, 0]\nclass = class_0')]
```



Conclusion:

Date:

10.Ensemble Learning

Aim:

Step 1: Load the breast_cancer dataset from sklearn.datasets using load_breast_cancer class and split it into traing and testing subsets.

In [23]:
```
from sklearn.datasets import load_breast_cancer
d=load_breast_cancer()
print("feature names:",d.feature_names)
print()
print("target classes:",d.target_names)
x=d.data
y=d.target
print()
print("shape of input data:",x.shape)
print("shape of output data:",y.shape)
```

```
feature names: ['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']

target classes: ['malignant' 'benign']

shape of input data: (569, 30)
shape of output data: (569,)
```

In [24]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=7)
print("Training data input shape:",x_train.shape)
print("Training data output shape:",y_train.shape)
print("Testing data input shape:",x_test.shape)
print("Testing data output shape:",y_test.shape)
```

```
Training data input shape: (398, 30)
Training data output shape: (398,)
Testing data input shape: (171, 30)
Testing data output shape: (171,)
```

Step 2: Build the Decision Tree, Random Forest and AdaBoost models

--> Import DecisionTreeClassifier from sklearn.tree module

--> Import RandomForestClassifier and AdaBoostClassifier classes from sklearn.ensemble module.

--> We can use n_estimators parameter while creating models to change the default number of 10 weak models.

In [25]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
tree= DecisionTreeClassifier (criterion="entropy")
tree.fit(x_train,y_train)
```

Out[25]:
```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy')
```

In [26]:
```
RFmodel=RandomForestClassifier(n_estimators=5)
RFmodel.fit(x_train,y_train)
```

Out[26]:
```
RandomForestClassifier
RandomForestClassifier(n_estimators=5)
```

In [27]:
```
Adamodel=AdaBoostClassifier(n_estimators=5)
Adamodel.fit(x_train,y_train)
```

Out[27]:
```
AdaBoostClassifier
AdaBoostClassifier(n_estimators=5)
```

Step 3: Compare the performance of all three models

--> Import accuracy_score from sklearn.metrics module.

--> Compute both Traing and Test accuracies to observe any overfitting.

In [28]:
```
#Training accuracies
from sklearn.metrics import accuracy_score
print("Decision tree training accuracy:",accuracy_score(y_train,tree.predict(x_train)))
print("Random forest training accuracy:",accuracy_score(y_train,RFmodel.predict(x_trai
print("Adaboost training accuracy:",accuracy_score(y_train,Adamodel.predict(x_train)))
```

```
Decision tree training accuracy: 1.0
Random forest training accuracy: 0.9899497487437185
Adaboost training accuracy: 0.9698492462311558
```

In [29]:
```
#Testing accuracies
from sklearn.metrics import accuracy_score
print("Decision tree testing accuracy:",accuracy_score(y_test,tree.predict(x_test)))
print("Random forest testing accuracy:",accuracy_score(y_test,RFmodel.predict(x_test)))
print("Adaboost testing accuracy:",accuracy_score(y_test,Adamodel.predict(x_test)))
```

```
Decision tree testing accuracy: 0.9473684210526315
Random forest testing accuracy: 0.9766081871345029
Adaboost testing accuracy: 0.9590643274853801
```

Conclusion:

Date:

Aim:

Step 1: Import the iris dataset using load_iris class from sklearn.datasets and split it into training and testing subsets

In [21]:
```
from sklearn.datasets import load_iris
d=load_iris()
x=d.data
y=d.tacluster_labels=model.labels_
print("Clusterlabels are:",cluster_labels)
centroids=model.cluster_centers_
print("centroids are:",centroids)rget
```

In [22]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.15,random_state=7)
```

Step 2: Build the MLP Model

--> Import MLPClassifier from sklearn.neural_network module

--> Multilayer Perceptron Classifier

--> Some Hyper Parameters:

```
    * max_iter - Maximum number of iterations. Default is 200.

    * hidden_layer_sizes - Determines the number of hidden layers and
    number of neurons in each layer. Default is (100, ) i.e., 1 hidden
    layer with 100 neurons.

    * activation - Activation function for the hidden layers {'identity',
    'logistic', 'tanh', 'relu'}. Default is 'relu'.

    * solver - The solver for weight optimization {'lbfgs', 'sgd',
    'adam'}. Default is 'adam'.

    * batch_size - Size of minibatches for stochostic optimizers.

    * learning_rate - Learning rate schedule for weight update
    {'constant', 'invscaling', 'adaptive}. Default is 'constant'.

    * learning_rate_init - The initial learning rate. Default is 0.001.

    * momentum - Momentum for gradient decent update {0 - 1}. Defult is
    0.9.
```

--> fit(x_train, y_train) method can be used to build the model.

In [8]:
```
from sklearn.neural_network import MLPClassifier
```

In [23]:
```
# MLP with 100 iterations and one hidden layer with 100 default nodes.
model=MLPClassifier(max_iter=100,hidden_layer_sizes=(100,))
model.fit(x_train,y_train)
```

```
C:\Users\HP\anaconda3\Lib\site-packages\sklearn\neural_network\_multilayer_perceptro
n.py:691: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100) reached
and the optimization hasn't converged yet.
  warnings.warn(
```

Out[23]:
```
MLPClassifier(max_iter=100)
```

Step 3: Evaluate the model performance

--> Import accuracy_score function from sklearn.metrics

--> Check for any overfitting

In [24]:
```
from sklearn.metrics import accuracy_score
print("Training accuracy:",accuracy_score(y_train,model.predict(x_train)))
print("Testing accuracy:",accuracy_score(y_test,model.predict(x_test)))
```

```
Training accuracy: 0.889763779527559
Testing accuracy: 0.7391304347826086
```

Step 4: Fine tune the model performance by channging hyper parammeter values

--> Check for any overfitting

In [25]:
```
# MLP with 100 iterations and two hidden layers with 100 and 20 nodes respectively.
from sklearn.metrics import accuracy_score
model1=MLPClassifier(max_iter=100,hidden_layer_sizes=(100,20))
model1.fit(x_train,y_train)
print("Training accuracy:",accuracy_score(y_train,model1.predict(x_train)))
print("Testing accuracy:",accuracy_score(y_test,model1.predict(x_test)))
```

```
Training accuracy: 0.984251968503937
Testing accuracy: 0.8695652173913043
C:\Users\HP\anaconda3\Lib\site-packages\sklearn\neural_network\_multilayer_perceptro
n.py:691: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100) reached
and the optimization hasn't converged yet.
  warnings.warn(
```

In [27]:
```
# MLP with 100 iterations, two hidden layers with 100 and 20 nodes respectively and ta
from sklearn.metrics import accuracy_score
model2=MLPClassifier(max_iter=100,hidden_layer_sizes=(100,20),activation="tanh")
model2.fit(x_train,y_train)
print("Training accuracy:",accuracy_score(y_train,model2.predict(x_train)))
print("Testing accuracy:",accuracy_score(y_test,model2.predict(x_test)))
```

```
Training accuracy: 0.984251968503937
Testing accuracy: 0.9130434782608695
```

```
C:\Users\HP\anaconda3\Lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100) reached and the optimization hasn't converged yet.
  warnings.warn(
```

Conclusion:

Date: 12.KMeans

Aim:

Clustering:

--> Example of Unsupervised Learning.

--> Clustering divides the data points into groups based on siilarity or distance measure.

--> Goal: Smaples with in the cluster are vary similar and samples in different clusters are different.

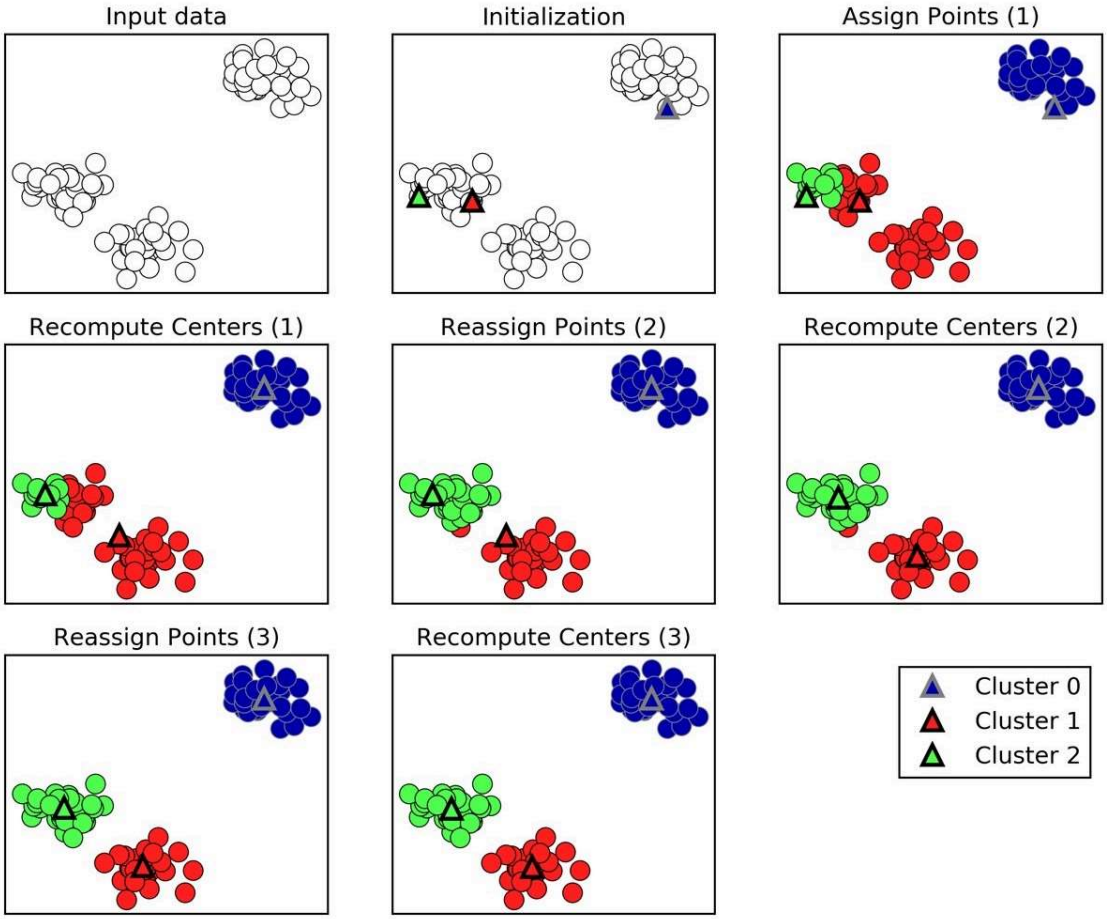
K-Means Algorithm

--> Inputs: X - Input samples and K - Number of clusters need to be created.

--> Outputs: Group Labels assigned to each input and Cluster centers.

--> Pocedure:

```
i.   Randomly picks centroids for K clusters
ii.  Each data point is assigned to one of the K clusters based on th
e distance from centroids.
iii. Recalculate each centroid using the mean of all the data points
assigned to that cluster.
iV.  Repeat the steps ii and iii until there is no further re-arrange
ment of cluster centers.
```



Step 1: Load iris dataset and trim it with only two features, petal length and petal width.

--> Import load_iris dataset from sklearn.datasets module.

In [1]:
```
from sklearn.datasets import load_iris
d=load_iris()
x=d.data[:,2:4]
print(x[0:5])
```

```
[[1.4 0.2]
 [1.4 0.2]
 [1.3 0.2]
 [1.5 0.2]
 [1.4 0.2]]
```

Step 2: Fit the KMeans clustering model

--> Import the KMeans class from sklearn.cluster module.

--> Create a model object using n_clusters attribute.

--> Fit the model using fit(x) method.

--> Use labels_ attribute to get labels assigned to all samples.

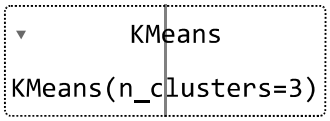
--> Use cluster_centers_ attribute to get cluster centers.

--> Use predict method to know the cluster label for new data

In [4]:
```
from sklearn.cluster import KMeans
model=KMeans(n_clusters=3)
model.fit(x)
```

```
C:\Users\HP\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: Futu
reWarning: The default value of `n_init` will change from 10 to 'auto' in 1.
4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\HP\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: User
Warning: KMeans is known to have a memory leak on Windows with MKL, when ther
e are less chunks than available threads. You can avoid it by setting the env
ironment variable OMP_NUM_THREADS=1.
  warnings.warn(
```

Out[4]:



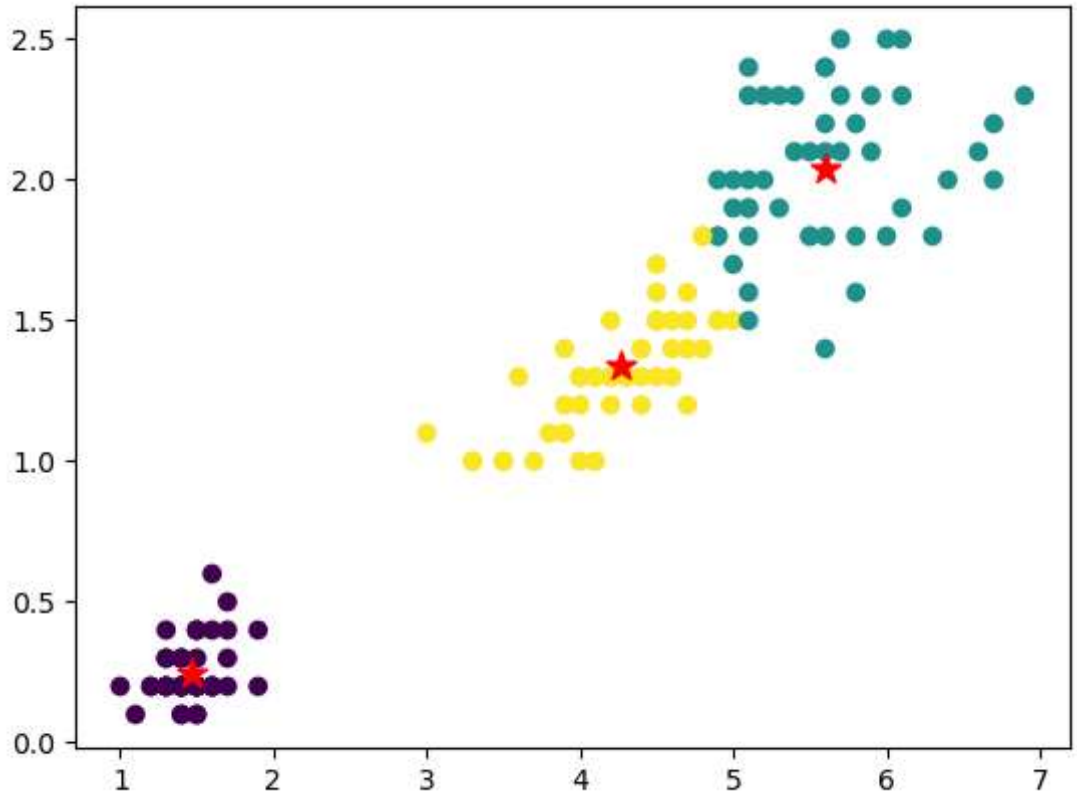
In [5]:
```
cluster_labels=model.labels_
print("Clusterlabels are:",cluster_labels)
centroids=model.cluster_centers_
print("centroids are:",centroids)
```

```
Clusterlabels are: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 1 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 2 1 1 1 1
 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1
 1 1]
centroids are: [[1.462      0.246     ]
 [5.59583333 2.0375    ]
 [4.26923077 1.34230769]]
```

Step 3: Visulaize the clusters with centers

* Use matplotlib.pyplot.scatter(feature1, feature2, c=categorical_variable, marker = "*", s = size)

In [7]:
```
import matplotlib.pyplot as plt
plt.scatter(x[:,0],x[:,1],c=cluster_labels)
plt.scatter(centroids[:,0],centroids[:,1],c="red",marker="*",s=120)
plt.show()
```



Step 4: Create a data frame with input samples and assigned cluster labels.

In [8]:
```
import pandas as pd
df=pd.DataFrame(x,columns=["petallength","petalwidth"])
df["CLusterLabels"]=cluster_labels
df
```

Out[8]:

	petallength	petalwidth	CLusterLabels
0	1.4	0.2	0
1	1.4	0.2	0
2	1.3	0.2	0
3	1.5	0.2	0
4	1.4	0.2	0
...	...	...	...
145	5.2	2.3	1
146	5.0	1.9	1
147	5.2	2.0	1
148	5.4	2.3	1
149	5.1	1.8	1

150 rows × 3 columns

Conclusion: