Geetanjali group of Colleges, Rajkot

Department of Computer Science

Lab Manual

Subject: Machine Learning with Python

Index

Assignment – 1
To Perform Basic Operation of Python for Data Analysis in machine learning

1) List:-

```
#list is ordered & changable
x=['a','b','c']
print("List X[0] is:-",x[0])
x[1]='d' #change value of element at index 1 print("List X after updation",x)
for y in x: print(y)
if 'f' in x: print("available")
else:
print("not available")
x.append("Mihir Shukla") print("after append()",x)
x.insert(2,"160470107054") x.insert(4,'y')
x.insert(5,'z')
print("after insert()",x)
x.pop(2) #delete element at index 2 print("after pop()",x)
x.remove('a') #must be available in list otherwise conduct an error print("after remove()",x)
del x[0]
print("after del()",x)
z=x.copy() print("List Z is:-",z)
del x #delete whole list
#print(x) #now generates an error because x is not available now. print("Error")
```

```
List X[0] is:- a
List X after updation ['a', 'd', 'c']
a
d
c
not available
after append() ['a', 'd', 'c', 'Mihir Shukla']
after insert() ['a', 'd', '160470107054', 'c', 'y', 'z', 'Mihir Shukla']
after pop() ['a', 'd', 'c', 'y', 'z', 'Mihir Shukla']
after remove() ['d', 'c', 'y', 'z', 'Mihir Shukla']
after del() ['c', 'y', 'z', 'Mihir Shukla']
List Z is:- ['c', 'y', 'z', 'Mihir Shukla']
Error
```

Tuple:-

```
#Tuple is ordered and unchangable
x=('a','b','c','a')
print("Tuple x[0] is:-",x[0]) print("Whole Tuple x is:-",x)
#x[0]='d' not possible in tuple bcoz it is unordered
for y in x: print(y)
if 'f' in x: print("available")
else:
print("not available")
print("count of element a:-",x.count('a'))
print("index of element a:-",x.index('a')) ‘’’’ x.apppend()
x.remove() all this methods are not possible because tuple is unordered and unchangeable x.pop()
x.insert()"""
#print del(x) not possible because tuple is unchangeable
```

```
Tuple x[0] is:- a
Whole Tuple x is:- ('a', 'b', 'c', 'a')
a
b
c
a
not available
count of element a:- 2
index of element a:- 0
```

Set:-

#Set is unordered,unnindexed & unchangable

```
x={"a","b",'c'}
print("set x is:-",x)#print in random anner because set is unindexed
for y in x: print(y)
x.add('x') print("after add()",x)
x.update(['y','z']) print("after update()",x)
x.remove('y') print("after remove()",x)
x.clear()
print(x) #clear whole set returns --->set()
#x.discard()not possible because set is unchangable
#x.pop() not possible because set is unchangable
#del x not possible because set is unchangable
print("Eroor by pop(),del(),discard()")
```

```
set x is:- {'c', 'b', 'a'}
c
b
a
after add() {'c', 'b', 'a', 'x'}
after update() {'c', 'y', 'x', 'z', 'b', 'a'}
after remove() {'c', 'x', 'z', 'b', 'a'}
set()
Eroor by pop(),del(),discard()
```

In [13]:

Dictionary:-

#unordered,indexed & changable

```
x={"Name":"Mihir","Surname":"Shukla","enroll":160470107054} print("Dictionary x is:-",x)
print("Name element from dictionary:-",x["Name"])
x["enroll"]=54 #update value print("Updated value of x:-",x)
print(x.get("Surname"))
for y in x.values(): print("Values are:-",y)
for y in x.keys(): print("Keys are:-",y)
for y,z in x.items(): print("Key-value pair:-",y,z)
x["sem"]=7
print("add new key-value",x)
x.pop('sem') print("after pop()",x)
del x['Surname'] #remove particular element
print(x)
if "Name" in x: print("available")
```

```
Dictionary x is:- {'Name': 'Mihir', 'Surname': 'Shukla', 'enroll': 160470107054}
Name element from dictionary:- Mihir
Updated value of x:- {'Name': 'Mihir', 'Surname': 'Shukla', 'enroll': 54}
Shukla
Values are:- Mihir
Values are:- Shukla
Values are:- 54
Keys are:- Name
Keys are:- Surname
Keys are:- enroll
Key-value pair:- Name Mihir
Key-value pair:- Surname Shukla
Key-value pair:- enroll 54
add new key-value {'Name': 'Mihir', 'Surname': 'Shukla', 'enroll': 54, 'sem': 7}
after pop() {'Name': 'Mihir', 'Surname': 'Shukla', 'enroll': 54}
{'Name': 'Mihir', 'enroll': 54}
available
```

Assignment – 2
To Perform various usage Python Library for Data Analysis in machine learning

numpy

```
import numpy as np from scipy import stats
a=np.array([1,2,3]) print("Print whole array:-",a)
a[1]=7
print("updation:-",a)
print("size of array:-",a.shape) #size 3*1 print("type of array:-",type(a)) #type
print("matrix with zeros",np.zeros((2,2)))
print("matrix with ones",np.ones((2,2)))
print(np.eye(2)) #retuen matrix having 1 on diagonal elsewhere 0's.
print("matrix with all values 6:-",np.full((2,2),6))
print("matrix of random numbers:-",np.random.random((3,3))) #matrix of random digits generally in
double
a=np.array([[1,2,3,4], [5,6,7,8],
[9,10,11,12],
[13,14,15,16]]) b=np.array([0,3,1,2])
print(a[:3,1:5])
print(a[np.arange(4),b])
# return the element from given indexes i.e. 0 th element from 1 st row, 3 rd element from 2 nd row and
so on.....................
print(a[[1,2,3],[0,1,0]])
#element selection 1 row 0th element 2 row 1 st element 3 row 0 th elemen
a=np.array([[1,2,3,4],[0,0,1,2],[3,6,9,9],[0,0,0,0]]) b=np.array([[1,2,6,2],[1,0,1,0],[3,6,5,7],[1,1,1,1]])
print(a[a>=3])#condition

print("Sum:-",np.sum(a))
print("Add:-",np.add(a,b))
print("Subtract:-",np.subtract(a,b))
print("Divide:-",np.divide(a,b))
print("Multiply:-",np.multiply(a,b))
print("Square root:-",np.sqrt(a))
print("Dot product:-",np.dot(a,b))
```

```
Print whole array:- [1 2 3]
updation:- [1 7 3]
size of array:- (3,)
type of array:- <class 'numpy.ndarray'>
matrix with zeros [[ 0.  0.]
 [ 0.  0.]]
matrix with ones [[ 1.  1.]
 [ 1.  1.]]
[[ 1.  0.]
 [ 0.  1.]]
matrix with all values 6:- [[6 6]
 [6 6]]
matrix of random numbers:- [[ 0.16661818  0.32282688  0.34052091]
 [ 0.86835917  0.61992034  0.72282381]
 [ 0.19053607  0.50247281  0.47616272]]
[[ 2  3  4]
 [ 6  7  8]
 [10 11 12]]
[ 1  8 10 15]
[ 5 10 13]
[3 4 3 6 9 9]
Sum:- 40
Add:- [[ 2  4  9  6]
 [ 1  0  2  2]
 [ 6 12 14 16]
 [ 1  1  1  1]]
Subtract:- [[ 0  0 -3  2]
 [-1  0  0  2]
 [ 0  0  4  2]
 [-1 -1 -1 -1]]
Divide:- [[ 1.          1.          0.5         2.        ]
 [ 0.                nan  1.                inf]
 [ 1.          1.          1.8         1.28571429]
 [ 0.          0.          0.          0.        ]]
Multiply:- [[ 1  4 18  8]
 [ 0  0  1  0]
 [ 9 36 45 63]
 [ 0  0  0  0]]
```

```
x=np.array([[0, 1], [2, 3]]) print("Transpose:-",np.transpose(x)) x = np.array([[1], [2], [3]])
y = np.array([4, 5, 6])
b = np.broadcast(x, y)
print(b)
print(np.empty([2, 2]))
a = np.array([[1, 1], [2, 2], [3, 3]])
print(np.insert(a, 1, 5)) #here 1 is position and 5 is a value to be inserted further
a=np.append([1, 2, 3], [[4, 5, 6], [7, 8, 9]])
print(a)
print("Unique element:-",np.unique([1,2,3,4,3,3,3]))
a=np.array([[1,2],[3,4],[50,55],[34,17]]) print("Mode:-",stats.mode(a)) print(np.median(a))
b=np.array([[1,2],[3,4]])
print(np.mean(b)) print("Average:-",np.average(b)) print("Standard deviation:-",np.std(a))
print("Variance:-",np.var(a)) print("Copy:-",np.copy(b))
c=np.array([[3,1],[4,9],[12,1],[10,20],[5,2]]) #sort array in but from particular row not all array print("Sort:-
",np.sort(c))
```

```
Transpose:- [[0 2]
 [1 3]]
<numpy.broadcast object at 0x00000022CC9256B70>
[[ 1.  0.]
 [ 0.  1.]]
[1 5 1 2 2 3 3]
[1 2 3 4 5 6 7 8 9]
Unique element:- [1 2 3 4]
Mode:- ModeResult(mode=array([[1, 2]]), count=array([[1, 1]]))
10.5
2.5
Average:- 2.5
Standard deviation:- 21.0816863652
Variance:- 444.4375
Copy:- [[1 2]
 [3 4]]
Sort:- [[ 1  3]
 [ 4  9]
 [ 1 12]
 [10 20]
 [ 2  5]]
```

Assignment – 3

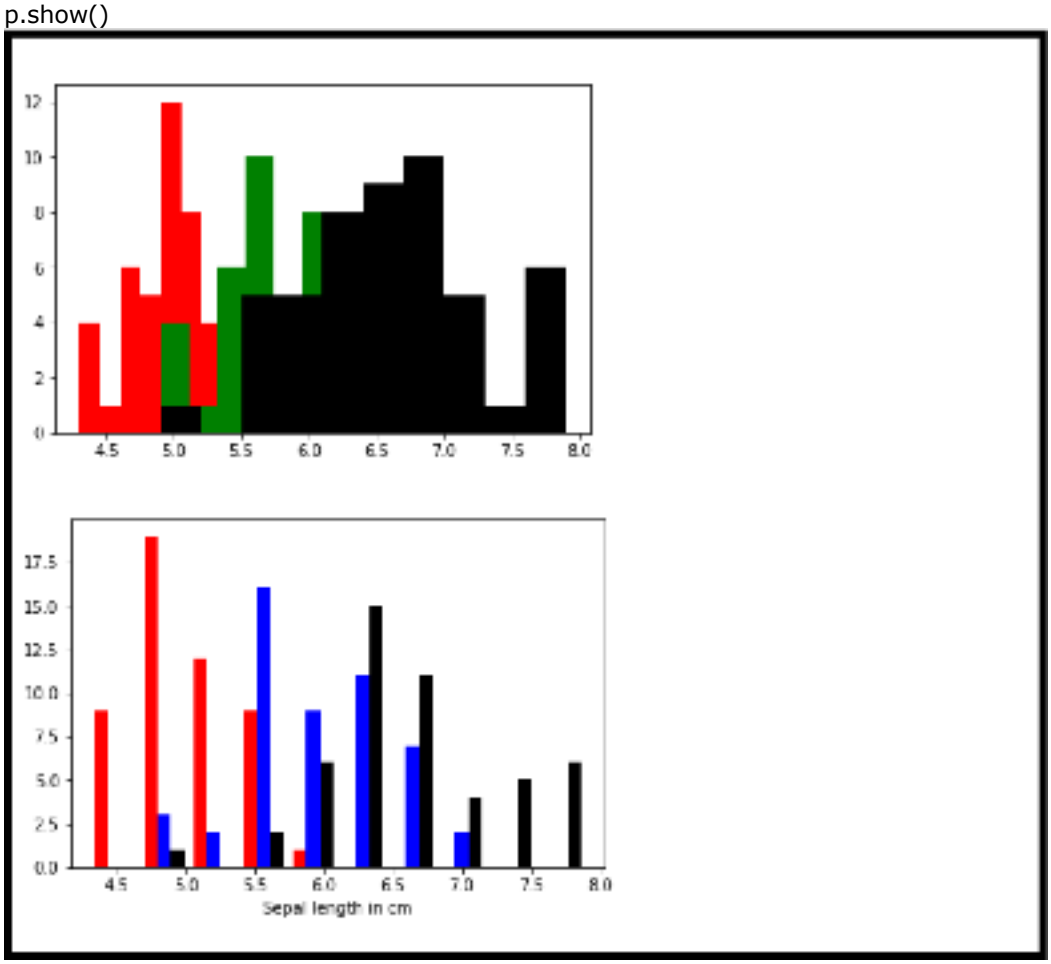
To Perform Data Importing and Visualization using Python in machine learning.

```
import numpy as np
#genfromtxt=load data from text files,while missing values handled as specified.
thedata = np.genfromtxt(
'iris.csv', skip_header=0, skip_footer=0, delimiter=',')
# file name
# lines to skip at the top
# lines to skip at the bottom
for row in thedata: print(row)
```

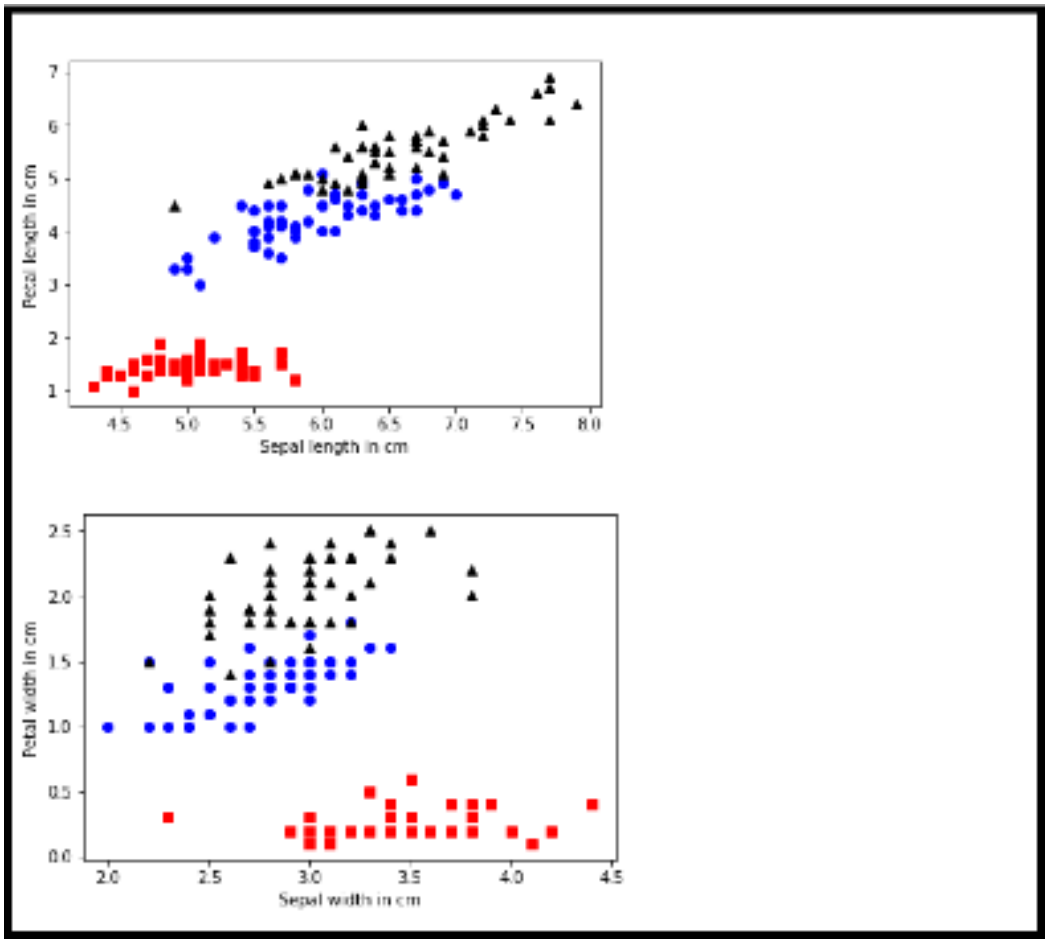
```
[ 5.1  3.5  1.4  0.2]
[ 4.9  3.   1.4  0.2]
[ 4.7  3.2  1.3  0.2]
[ 4.6  3.1  1.5  0.2]
[ 5.   3.6  1.4  0.2]
[ 5.4  3.9  1.7  0.4]
[ 4.6  3.4  1.4  0.3]
[ 5.   3.4  1.5  0.2]
[ 4.4  2.9  1.4  0.2]
[ 4.9  3.1  1.5  0.1]
[ 5.4  3.7  1.5  0.2]
[ 4.8  3.4  1.6  0.2]
[ 4.8  3.   1.4  0.1]
[ 4.3  3.   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]
[ 5.1  3.8  1.5  0.3]
[ 5.4  3.4  1.7  0.2]
[ 5.1  3.7  1.5  0.4]
[ 4.6  3.6  1.   0.2]
[ 5.1  3.3  1.7  0.5]
[ 4.8  3.4  1.9  0.2]
[ 5.   3.   1.6  0.2]
[ 5.   3.4  1.6  0.4]
[ 5.2  3.5  1.5  0.2]
[ 5.2  3.4  1.4  0.2]
[ 4.7  3.2  1.6  0.2]
[ 4.8  3.1  1.6  0.2]
[ 5.4  3.4  1.5  0.4]
[ 5.2  4.1  1.5  0.1]
[ 5.5  4.2  1.4  0.2]
[ 4.9  3.1  1.5  0.1]
[ 5.   3.2  1.2  0.2]
[ 5.5  3.5  1.3  0.2]
[ 4.9  3.1  1.5  0.1]
[ 4.4  3.   1.3  0.2]
[ 5.1  3.4  1.5  0.2]
```

```
Import numpy as np
Import matplotlib.pyplot as p
d=np.genfromtxt('iris.csv',delimiter=',',usecols=(0,1,2,3))
t=np.genfromtxt('iris.csv',delimiter=',',usecols=(4),dtype='str') p.hist(d[t=='setosa',0],color='red')
p.hist(d[t=='versicolor',0],color='green')
p.hist(d[t=='virginica',0],color='black')
p.show()
d1=d[t=='setosa',0]
d2=d[t=='versicolor',0]
d3=d[t=='virginica',0]
p.hist([d1,d2,d3],color=['red','blue','black'])
p.xlabel("Sepal length in cm")
```

p.show()



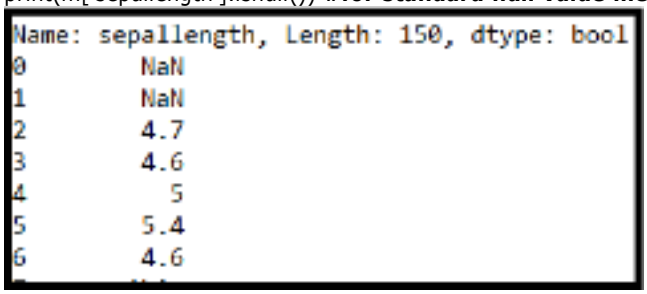
p.plot(d[t=='setosa',0],d[t=='setosa',2],'rs')
p.plot(d[t=='versicolor',0],d[t=='versicolor',2],'bo') p.plot(d[t=='virginica',0],d[t=='virginica',2],'k^')
p.xlabel("Sepal length in cm")
p.ylabel("Petal length in cm")
p.show() p.plot(d[t=='setosa',1],d[t=='setosa',3],'rs') p.plot(d[t=='versicolor',1],d[t=='versicolor',3],'bo')
p.plot(d[t=='virginica',1],d[t=='virginica',3],'k^')
p.xlabel("Sepal width in cm")
p.ylabel("Petal width in cm")
p.show()


Petal length in cm
Sepal length in cm
Petal width in cm
Sepal width in cm

Assignment – 4
To Perform Missing Data handling with Python in machine learning.

1) Standard missing values

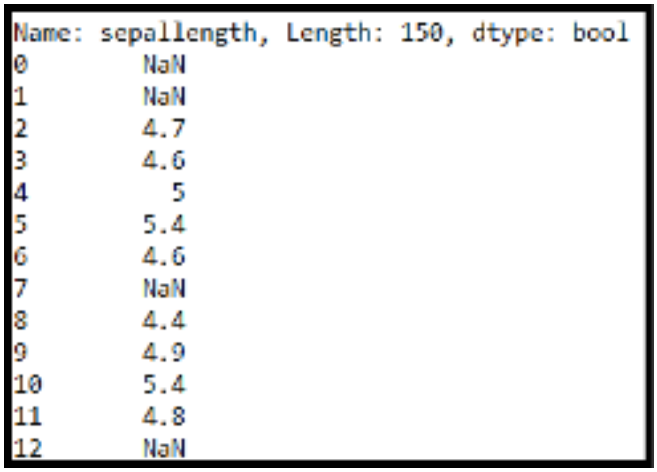
```
m=pd.read_csv("iris.csv")
print(m)
print(m['sepallength'].isnull()) #for standard null value means N/A or n/a print(m['sepallength'])
```



```
Name: sepallength, Length: 150, dtype: bool
0        NaN
1        NaN
2        4.7
3        4.6
4          5
5        5.4
6        4.6
```

Non-standard missing values

```
missing_values=['-','nA'] x=pd.read_csv("iris.csv",na_values=missing_values) print(x['sepallength'].isnull())
print(x['sepallength'])
#print(m.head())
```



```
Name: sepallength, Length: 150, dtype: bool
0        NaN
1        NaN
2        4.7
3        4.6
4          5
5        5.4
6        4.6
7        NaN
8        4.4
9        4.9
10       5.4
11       4.8
12       NaN
```

Unexpected missing values

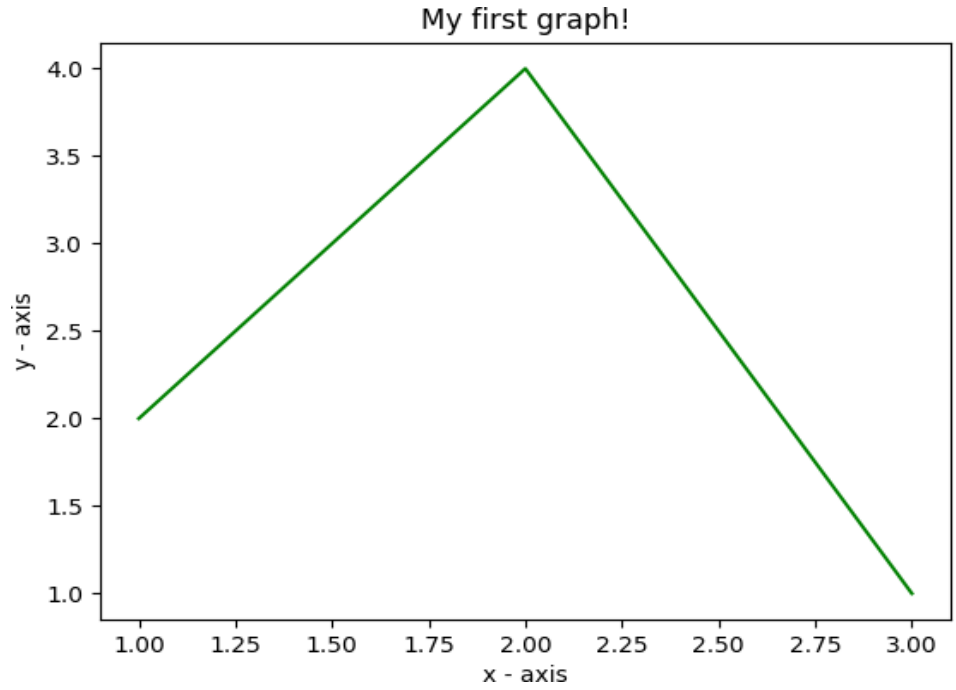
```
import numpy as np
x=pd.read_csv("iris.csv")
j=0
for i in x['fw']:
        if i.isdigit() :
        x.loc[j,'fw']=np.NaN
        j=j+1
print(x['fw'])
```

0	setosa
1	setosa
2	setosa
3	NaN
4	setosa
5	setosa
6	setosa
7	setosa
8	NaN
9	mihir
10	NaN
11	setosa
12	setosa
13	setosa
14	setosa
15	NaN

Assignment – 5
To Perform Data Plotting using Python in machine learning.

```
importing the required module
import matplotlib.pyplot as plt

# x axis values
x = [1,2,3]
# corresponding y axis values
y = [2,4,1]

# plotting the points
plt.plot(x, y)

# naming the x axis
plt.xlabel('x - axis')
# naming the y axis
plt.ylabel('y - axis')

# giving a title to my graph
plt.title('My first graph!')

# function to show the plot
plt.show()
```

Output:



The code seems self explanatory. Following steps were followed:

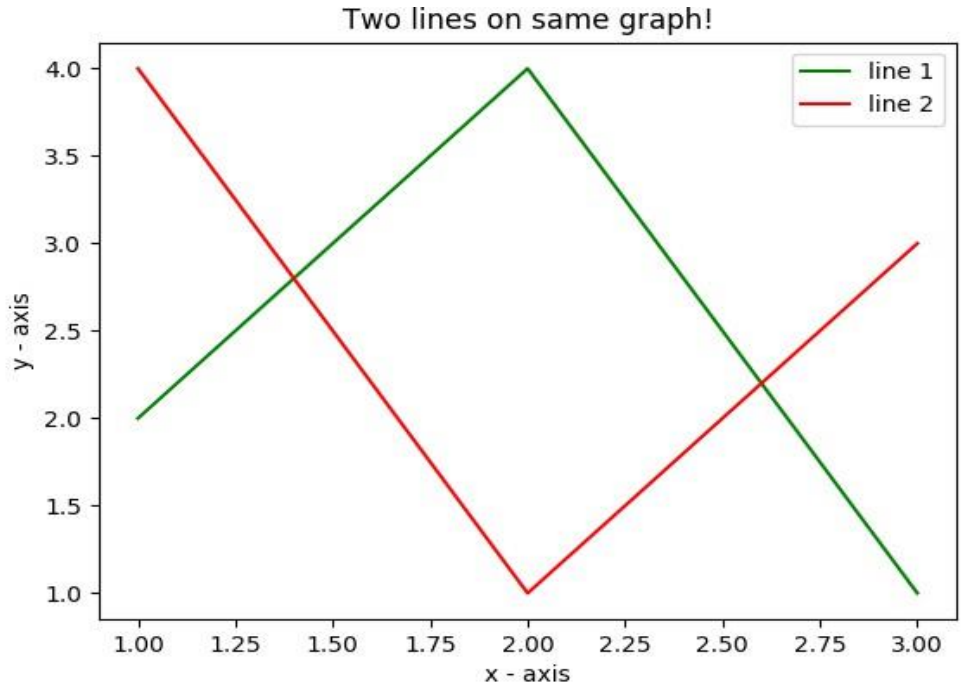
- Define the x-axis and corresponding y-axis values as lists.
- Plot them on canvas using **.plot()** function.
- Give a name to x-axis and y-axis using **.xlabel()** and **.ylabel()** functions.
- Give a title to your plot using **.title()** function.
- Finally, to view your plot, we use **.show()** function.

Plotting two or more lines on same plot

```
import matplotlib.pyplot as plt

# line 1 points
```

```
x1 = [1,2,3]
y1 = [2,4,1]
# plotting the line 1 points
plt.plot(x1, y1, label = "line 1")

# line 2 points
x2 = [1,2,3]
y2 = [4,1,3]
# plotting the line 2 points
plt.plot(x2, y2, label = "line 2")

# naming the x axis
plt.xlabel('x - axis')
# naming the y axis
plt.ylabel('y - axis')
# giving a title to my graph
plt.title('Two lines on same graph!')

# show a legend on the plot
plt.legend()

# function to show the plot
plt.show()
```

Output:



- Here, we plot two lines on same graph. We differentiate between them by giving them a name(**label**) which is passed as an argument of .plot() function.
- The small rectangular box giving information about type of line and its color is called legend. We can add a legend to our plot using **.legend()** function.

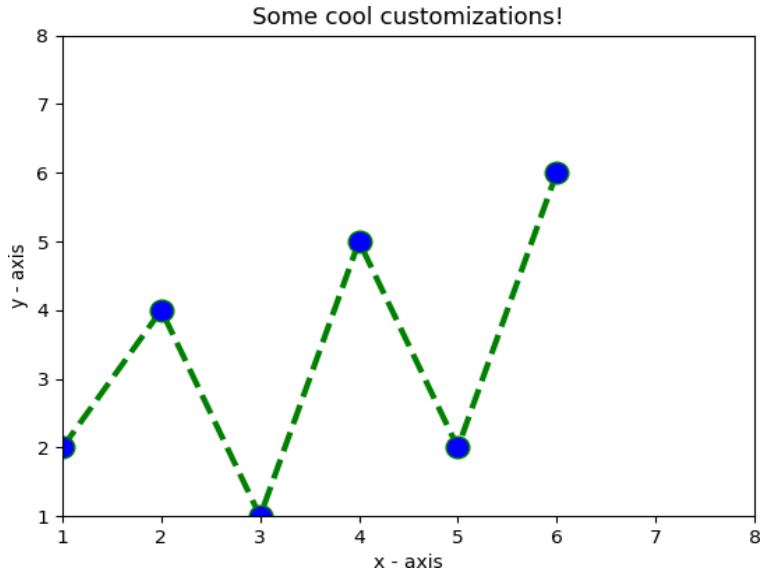
Customization of Plots

Here, we discuss some elementary customizations applicable on almost any plot.

```
import matplotlib.pyplot as plt

# x axis values
x = [1,2,3,4,5,6]
# corresponding y axis values
y = [2,4,1,5,2,6]
```

```
# plotting the points
plt.plot(x, y, color='green', linestyle='dashed', linewidth = 3,
        marker='o', markerfacecolor='blue', markersize=12)

# setting x and y axis range
plt.ylim(1,8)
plt.xlim(1,8)

# naming the x axis
plt.xlabel('x - axis')
# naming the y axis
plt.ylabel('y - axis')

# giving a title to my graph
plt.title('Some cool customizations!')

# function to show the plot
plt.show()
```

Output:



As you can see, we have done several customizations like

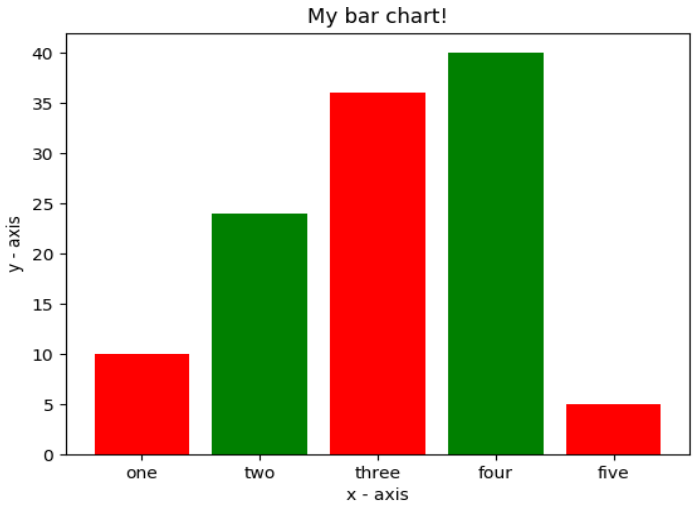
- setting the line-width, line-style, line-color.
- setting the marker, marker's face color, marker's size.
- overriding the x and y axis range. If overriding is not done, pyplot module uses auto-scale feature to set the axis range and scale.

Bar Chart

```
import matplotlib.pyplot as plt

# x-coordinates of left sides of bars
left = [1, 2, 3, 4, 5]

# heights of bars
height = [10, 24, 36, 40, 5]

# labels for bars
tick_label = ['one', 'two', 'three', 'four', 'five']

# plotting a bar chart
plt.bar(left, height, tick_label = tick_label,
```

```
    width = 0.8, color = ['red', 'green'])

# naming the x-axis
plt.xlabel('x - axis')
# naming the y-axis
plt.ylabel('y - axis')
# plot title
plt.title('My bar chart!')

# function to show the plot
plt.show()
```

Output :

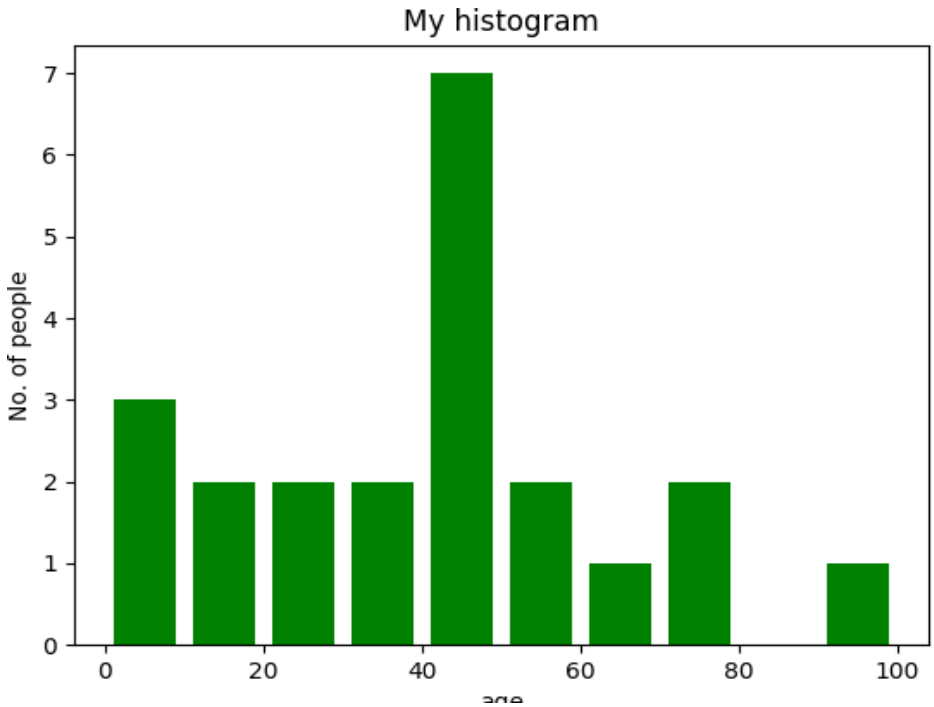


- Here, we use **plt.bar()** function to plot a bar chart.
- x-coordinates of left side of bars are passed along with heights of bars.
- you can also give some name to x-axis coordinates by defining **tick_labels**

Histogram

```
import matplotlib.pyplot as plt

# frequencies
ages = [2,5,70,40,30,45,50,45,43,40,44,
     60,7,13,57,18,90,77,32,21,20,40]

# setting the ranges and no. of intervals
range = (0, 100)
bins = 10

# plotting a histogram
plt.hist(ages, bins, range, color = 'green',
     histtype = 'bar', rwidth = 0.8)

# x-axis label
plt.xlabel('age')
# frequency label
plt.ylabel('No. of people')
# plot title
plt.title('My histogram')

# function to show the plot
plt.show()
```

Output:

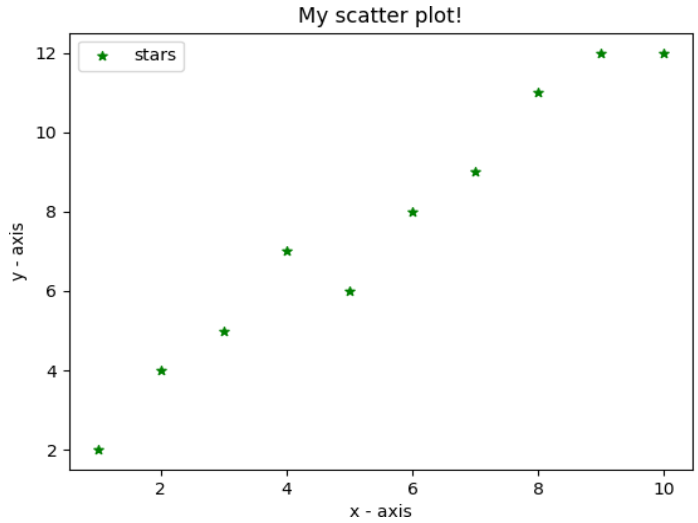


- Here, we use **plt.hist()** function to plot a histogram.
- frequencies are passed as the **ages** list.
- Range could be set by defining a tuple containing min and max value.
- Next step is to "**bin**" the range of values—that is, divide the entire range of values into a series of intervals—and then count how many values fall into each interval. Here we have defined **bins** = 10. So, there are a total of 100/10 = 10 intervals.

Scatter plot

```
import matplotlib.pyplot as plt

# x-axis values
x = [1,2,3,4,5,6,7,8,9,10]
# y-axis values
y = [2,4,5,7,6,8,9,11,12,12]

# plotting points as a scatter plot
plt.scatter(x, y, label= "stars", color= "green",
            marker= "*", s=30)

# x-axis label
plt.xlabel('x - axis')
# frequency label
plt.ylabel('y - axis')
# plot title
plt.title('My scatter plot!')
# showing legend
plt.legend()

# function to show the plot
plt.show()
```

Output:

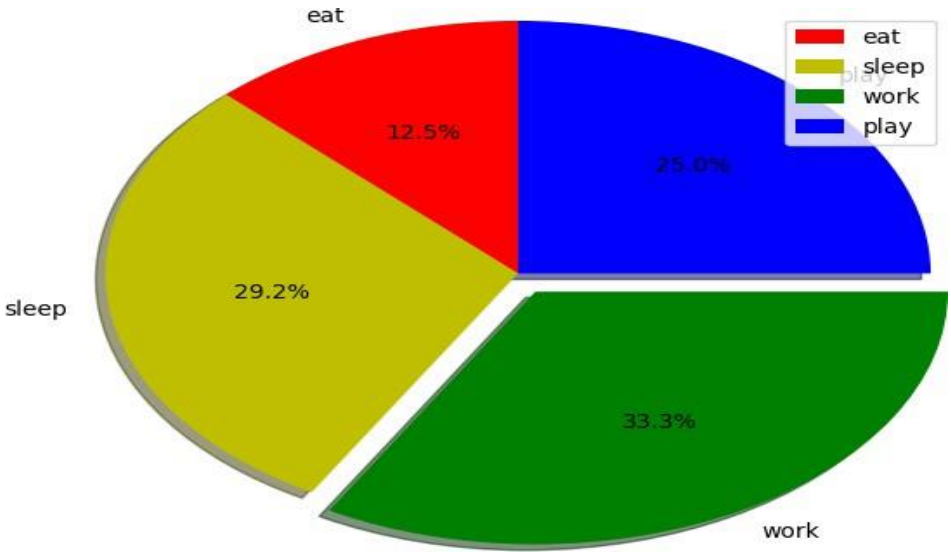


- Here, we use **plt.scatter()** function to plot a scatter plot.
- Like a line, we define x and corresponding y – axis values here as well.
- **marker** argument is used to set the character to use as marker. Its size can be defined using **s** parameter.

Pie-chart

```
import matplotlib.pyplot as plt

# defining labels
activities = ['eat', 'sleep', 'work', 'play']

# portion covered by each label
slices = [3, 7, 8, 6]

# color for each label
colors = ['r', 'y', 'g', 'b']

# plotting the pie chart
plt.pie(slices, labels = activities, colors=colors,
        startangle=90, shadow = True, explode = (0, 0, 0.1, 0),
        radius = 1.2, autopct = '%1.1f%%')

# plotting legend
plt.legend()

# showing the plot
plt.show()
```

Output of above program looks like this:

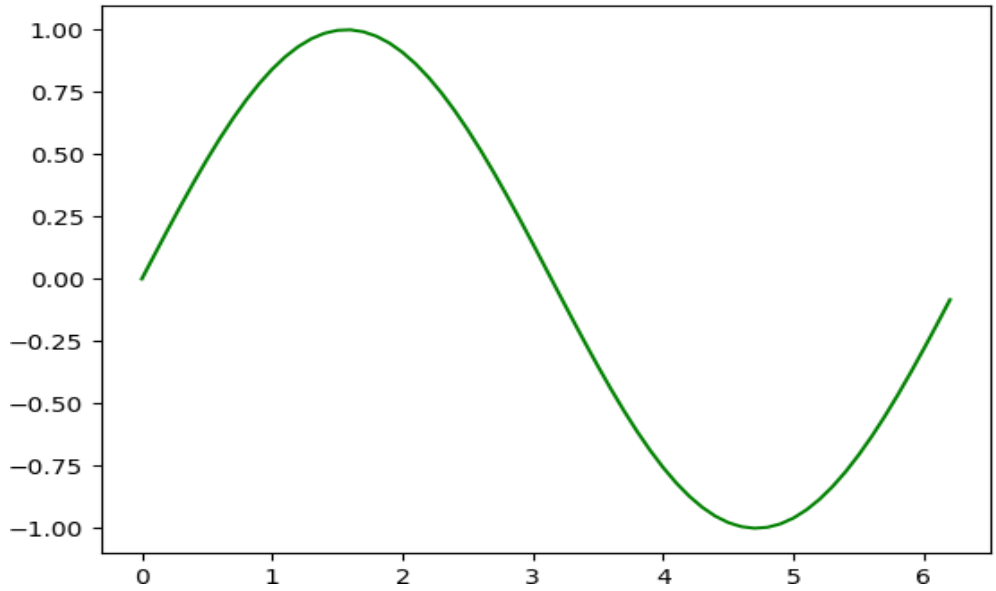


- Here, we plot a pie chart by using **plt.pie()** method.
- First of all, we define the **labels** using a list called **activities**.
- Then, portion of each label can be defined using another list called **slices**.
- Color for each label is defined using a list called **colors**.
- **shadow = True** will show a shadow beneath each label in pie-chart.
- **startangle** rotates the start of the pie chart by given degrees counterclockwise from the x-axis.
- **explode** is used to set the fraction of radius with which we offset each wedge.
- **autopct** is used to format the value of each label. Here, we have set it to show the percentage value only upto 1 decimal place.

Plotting curves of given equation

```
# importing the required modules
import matplotlib.pyplot as plt
import numpy as np

# setting the x - coordinates
x = np.arange(0, 2*(np.pi), 0.1)
# setting the corresponding y - coordinates
y = np.sin(x)

# potting the points
plt.plot(x, y)

# function to show the plot
plt.show()
```

Output of above program looks like this:

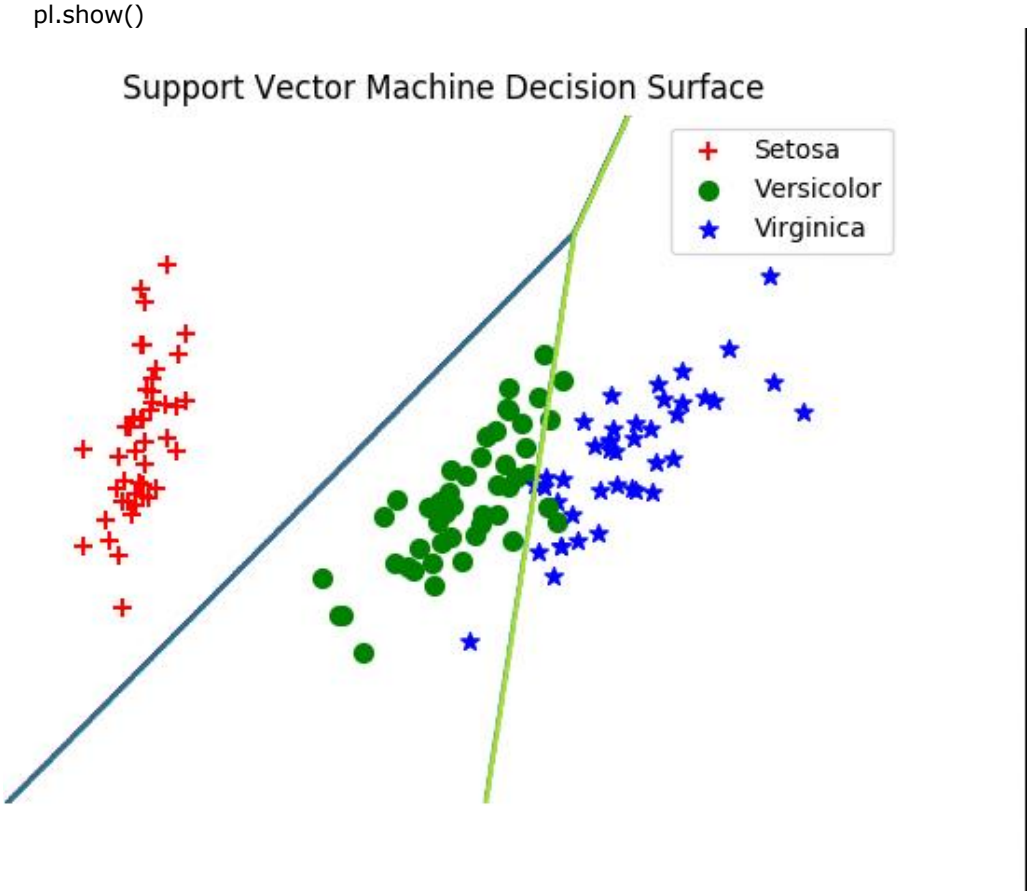


Here, we use **NumPy** which is a general-purpose array-processing package in python.

- To set the x – axis values, we use **np.arange()** method in which first two arguments are for range and third one for step-wise increment. The result is a numpy array.
- To get corresponding y-axis values, we simply use predefined **np.sin()** method on the numpy array.
- Finally, we plot the points by passing x and y arrays to the **plt.plot()** function.

Assignment – 6
Write python program for Support Vector Machine.

```
from sklearn.decomposition import PCA
from sklearn.datasets import load_iris
from sklearn import svm
from sklearn.model_selection import train_test_split
import pylab as pl
import numpy as np
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data,iris.target, test_size=0.10, random_state=111)
pca = PCA(n_components=2).fit(X_train)
pca_2d = pca.transform(X_train)
svmClassifier_2d = svm.LinearSVC(random_state=111).fit( pca_2d,y_train)
for i in range(0, pca_2d.shape[0]):
if y_train[i] == 0:
c1 =pl.scatter(pca_2d[i,0],pca_2d[i,1],c='r',s=50,marker='+')
elif y_train[i] == 1:
c2 = pl.scatter(pca_2d[i,0],pca_2d[i,1],c='g',s=50,marker='o')
elif y_train[i] == 2:
c3 = pl.scatter(pca_2d[i,0],pca_2d[i,1],c='b',s=50,marker='*')
pl.legend([c1, c2, c3], ['Setosa', 'Versicolor', 'Virginica'])
x_min, x_max = pca_2d[:, 0].min() - 1, pca_2d[:,0].max() + 1
y_min, y_max = pca_2d[:, 1].min() - 1, pca_2d[:, 1].max() + 1
xxyy= p.meshgrid(np.arange(x_min,x_max,.01),np.arange(y_min,y_max,.01))
Z = svmClassifier_2d.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
pl.contour(xx, yy, Z)
pl.title('Support Vector Machine Decision Surface')
pl.axis('off')
pl.show()
```



Assignment – 7
To Perform clustering of data using Python

we have to analyze the data in order to group them on the basis of a similarity criteria where groups (or clusters) are sets of similar samples. This kind of analysis is called unsupervised data analysis. One of the most famous clustering tools is the k-means algorithm, which we can run as follows:

```
from sklearn.cluster import KMeans
kmeans = KMeans(k=3, init='random') # initialization
kmeans.fit(data) # actual execution
```

The snippet above runs the algorithm and groups the data in 3 clusters (as specified by the parameter k). Now we can use the model to assign each sample to one of the clusters:

```
c = kmeans.predict(data)
```

And we can evaluate the results of clustering, comparing it with the labels that we already have using the completeness and the homogeneity score:

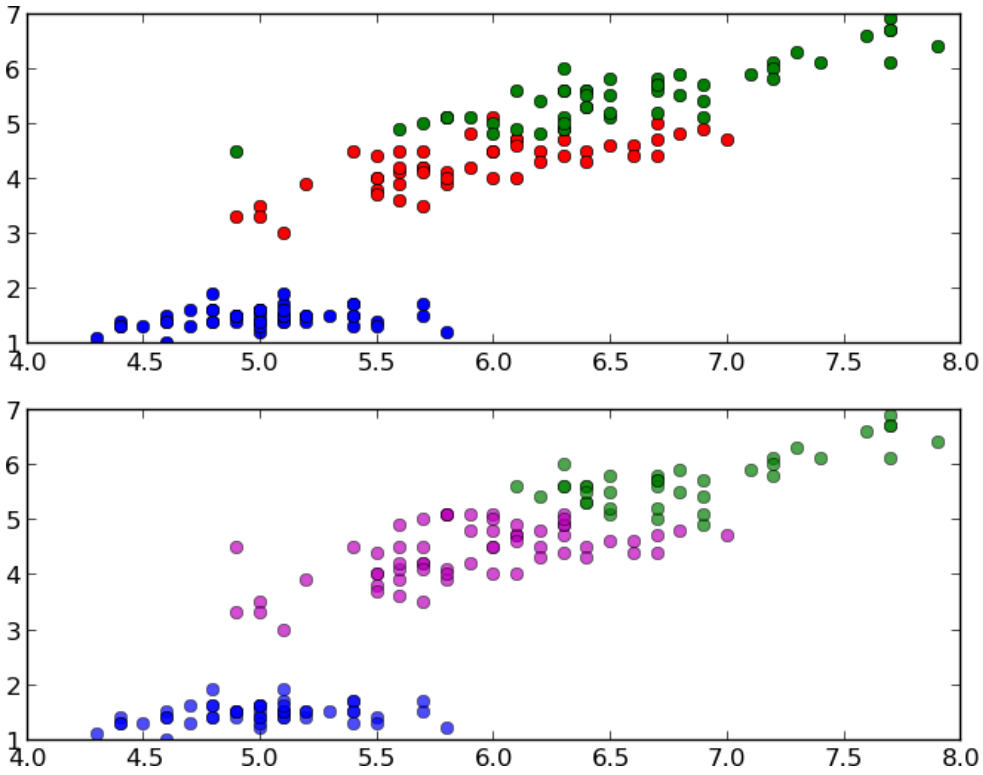
```
from sklearn.metrics import completeness_score, homogeneity_score
print completeness_score(t,c)
0.7649861514489815
print homogeneity_score(t,c)
0.7514854021988338
```

The completeness score approaches 1 when most of the data points that are members of a given class are elements of the same cluster while the homogeneity score approaches 1 when all the clusters contain almost only data points that are member of a single class.

We can also visualize the result of the clustering and compare the assignments with the real labels visually:

```
figure()
subplot(211) # top figure with the real classes
plot(data[t==1,0],data[t==1,2],'bo')
plot(data[t==2,0],data[t==2,2],'ro')
plot(data[t==3,0],data[t==3,2],'go')
subplot(212) # bottom figure with classes assigned automatically
plot(data[c==1,0],data[tt==1,2],'bo',alpha=.7)
plot(data[c==2,0],data[tt==2,2],'go',alpha=.7)
plot(data[c==0,0],data[tt==0,2],'mo',alpha=.7)
show()
```

The following graph shows the result:



Observing the graph we see that the cluster in the bottom left corner has been completely indentified by k-means while the two clusters on the top have been identified with some errors.

Assignment – 8
Write python program for Naïve Bayes.

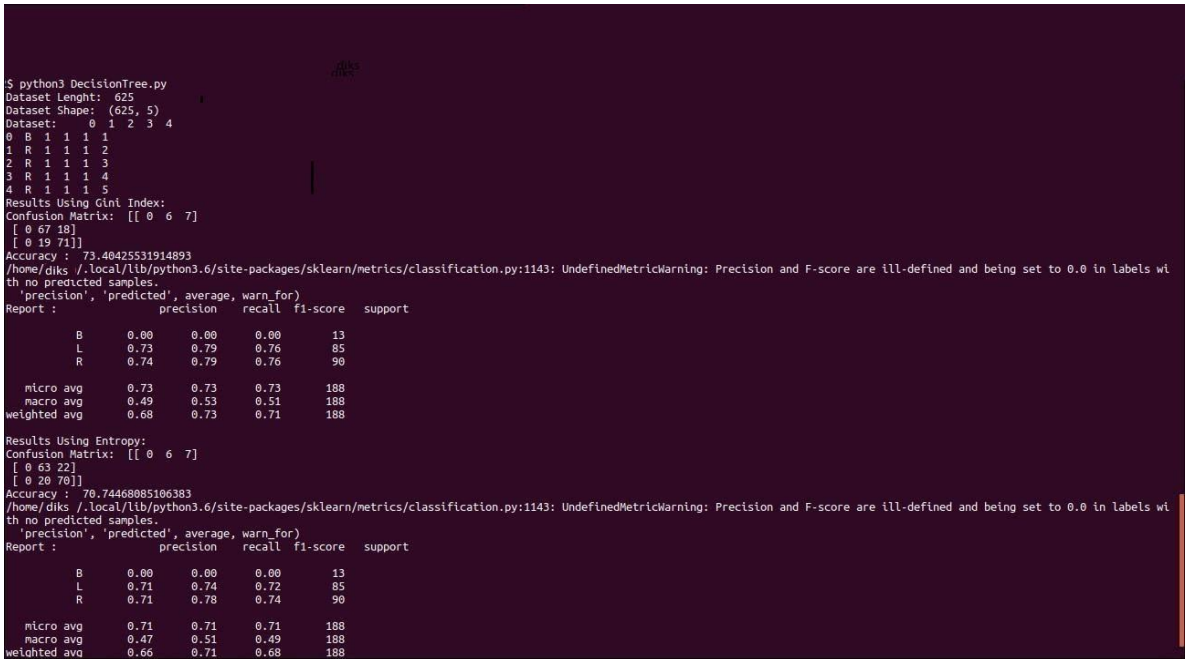
```
from sklearn import datasets
from sklearn import metrics
from sklearn.naive_bayes import GaussianNB
dataset = datasets.load_iris()
model = GaussianNB()
model.fit(dataset.data, dataset.target)
expected = dataset.target
predicted = model.predict(dataset.data)
print(metrics.classification_report(expected, predicted))
print(metrics.confusion_matrix(expected, predicted))
```

```
$ python3 naiveBayes.py
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        50
           1       0.94      0.94      0.94        50
           2       0.94      0.94      0.94        50

   micro avg       0.96      0.96      0.96       150
   macro avg       0.96      0.96      0.96       150
weighted avg       0.96      0.96      0.96       150

$ python3 naiveBayes.py
[[50  0  0]
 [ 0 47  3]
 [ 0  3 47]]
```

Assignment – 9
Write python program for Decision tree.

```
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
# Function importing Dataset
def importdata():
        balance_data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-'+
        'databases/balance-scale/balance-scale.data',sep= ',', header= None)
        # Printing the dataswet shape
        print ("Dataset Lenght: ", len(balance_data))
        print ("Dataset Shape: ", balance_data.shape)
        # Printing the dataset obseravtions
        print ("Dataset: ",balance_data.head())
        return balance_data
# Function to split the dataset
def splitdataset(balance_data):
        # Seperating the target variable
        X = balance_data.values[:, 1:5]
        Y = balance_data.values[:, 0]
        # Spliting the dataset into train and test
        X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.3, random_state = 100)
        return X, Y, X_train, X_test, y_train, y_test
# Function to perform training with giniIndex.
def train_using_gini(X_train, X_test, y_train):
        # Creating the classifier object
        clf_gini = DecisionTreeClassifier(criterion = "gini",random_state = 100,max_depth=3,
        min_samples_leaf=5)
        # Performing training
        clf_gini.fit(X_train, y_train)
        return clf_gini
# Function to perform training with entropy.
def tarin_using_entropy(X_train, X_test, y_train):
        # Decision tree with entropy
        clf_entropy = DecisionTreeClassifier(criterion = "entropy", random_state = 100, max_depth = 3,
        min_samples_leaf = 5)
        # Performing training
        clf_entropy.fit(X_train, y_train)
        return clf_entropy
# Function to make predictions
def prediction(X_test, clf_object):
        # Predicton on test with giniIndex
        y_pred = clf_object.predict(X_test)
        #print("Predicted values:")
        #print(y_pred)
        return y_pred
# Function to calculate accuracy
def cal_accuracy(y_test, y_pred):
        print("Confusion Matrix: ",confusion_matrix(y_test, y_pred))
        print ("Accuracy : ", accuracy_score(y_test,y_pred)*100)
        print("Report : ", classification_report(y_test, y_pred))
# Driver code
def main():
        # Building Phase
        data = importdata()
        X, Y, X_train, X_test, y_train, y_test = splitdataset(data)
        clf_gini = train_using_gini(X_train, X_test, y_train)
        clf_entropy = tarin_using_entropy(X_train, X_test, y_train)
```

```
        # Operational Phase
        print("Results Using Gini Index:")
        # Prediction using gini
        y_pred_gini = prediction(X_test, clf_gini)
        cal_accuracy(y_test, y_pred_gini)
        print("Results Using Entropy:")
        # Prediction using entropy
        y_pred_entropy = prediction(X_test, clf_entropy)
        cal_accuracy(y_test, y_pred_entropy)
# Calling main function
if __name__=="__main__":
main()
```



```
$ python3 DecisionTree.py
Dataset Lenght:  625
Dataset Shape:  (625, 5)
Dataset:     0  1  2  3  4
0  B  1  1  1  1
1  R  1  1  1  2
2  R  1  1  1  3
3  R  1  1  1  4
4  R  1  1  1  5
Results Using Gini Index:
Confusion Matrix:  [[ 0  6  7]
 [ 0 67 18]
 [ 0 19 71]]
Accuracy :  73.40425531914893
/home/diks /.local/lib/python3.6/site-packages/sklearn/metrics/classification.py:1143: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels wi
th no predicted samples.
  'precision', 'predicted', average, warn_for)
Report :               precision    recall  f1-score   support

          B       0.00      0.00      0.00        13
          L       0.73      0.79      0.76        85
          R       0.74      0.79      0.76        90

   micro avg       0.73      0.73      0.73       188
   macro avg       0.49      0.53      0.51       188
weighted avg       0.68      0.73      0.71       188

Results Using Entropy:
Confusion Matrix:  [[ 0  6  7]
 [ 0 63 22]
 [ 0 20 70]]
Accuracy :  70.74468085106383
/home/diks /.local/lib/python3.6/site-packages/sklearn/metrics/classification.py:1143: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels wi
th no predicted samples.
  'precision', 'predicted', average, warn_for)
Report :               precision    recall  f1-score   support

          B       0.00      0.00      0.00        13
          L       0.71      0.74      0.72        85
          R       0.71      0.78      0.74        90

   micro avg       0.71      0.71      0.71       188
   macro avg       0.47      0.51      0.49       188
weighted avg       0.66      0.71      0.68       188
```

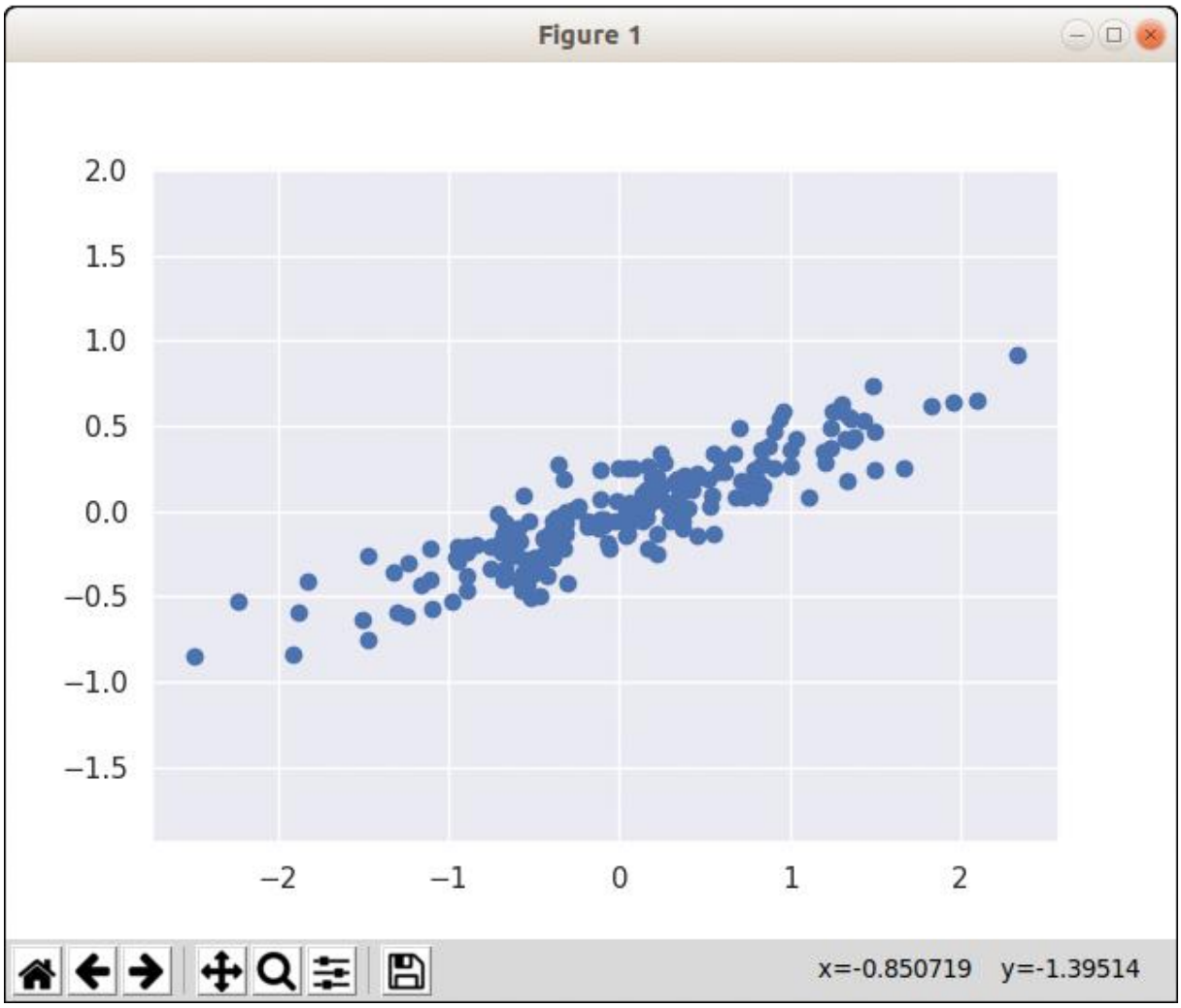
Assignment – 10
Implement Random forest using python.

```
from sklearn import datasets
#Load dataset
iris = datasets.load_iris()
import pandas as pd
data=pd.DataFrame({
'sepal length':iris.data[:,0],
'sepal width':iris.data[:,1],
'petal length':iris.data[:,2],
'petal width':iris.data[:,3],
'species':iris.target
})
data.head()
# Import train_test_split function
from sklearn.model_selection import train_test_split
X=data[['sepal length', 'sepal width', 'petal length', 'petal width']] # Features
y=data['species'] # Labels
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=0.3) # 70% training and 30% test
#Import Random Forest Model
from sklearn.ensemble import RandomForestClassifier
#Create a Gaussian Classifier
clf=RandomForestClassifier(n_estimators=100)
#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(X_train,y_train)
y_pred=clf.predict(X_test)
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
#print(metrics.confusion_matrix(y_test, y_pred))
```

```
$ python3 RandomForest.py
Accuracy: 0.9555555555555556
```

Assignment – 11
Perform dimensionality reduction using PCA.

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
rng = np.random.RandomState(1)
X = np.dot(rng.rand(2, 2), rng.randn(2, 200)).T
plt.scatter(X[:, 0], X[:, 1])
plt.axis('equal');
plt.show()
from sklearn.decomposition import PCA
# drawing vectors
pca = PCA(n_components=2)
pca.fit(X)
print(pca.components_)
print(pca.explained_variance_)
def draw_vector(v0, v1, ax=None):
ax = ax or plt.gca()
arrowprops=dict(arrowstyle='->',
linewidth=2,
shrinkA=0, shrinkB=0)
ax.annotate('', v1, v0, arrowprops=arrowprops)
# plot data
plt.scatter(X[:, 0], X[:, 1], alpha=0.2)
for length, vector in zip(pca.explained_variance_, pca.components_):
v = vector * 3 * np.sqrt(length)
draw_vector(pca.mean_, pca.mean_ + v)plt.axis('equal');
plt.show()
pca = PCA(n_components=1)
pca.fit(X)
X_pca = pca.transform(X)
print("original shape: ", X.shape)
print("transformed shape:", X_pca.shape)
X_new = pca.inverse_transform(X_pca)
plt.scatter(X[:, 0], X[:, 1], alpha=0.2)
plt.scatter(X_new[:, 0], X_new[:, 1], alpha=0.8)
plt.axis('equal');
plt.show()
```




```
$ python3 DR_PCA.py
original shape:    (200, 2)
transformed shape: (200, 1)
```

Assignment – 12
Perform Dimensionality reduction using LDA and NMF.

```
import numpy as np
X = np.array([[1,1], [2, 1], [3, 1.2], [4, 1], [5, 0.8], [6, 1]])
from sklearn.decomposition import NMF
model =NMF(n_components=2,init='random',random_state=0)
model.fit(X)
print('model components',model.components_)
print('model reconstruction error',model.reconstruction_err_)
```

```
$ python DR_NMF.py
model components [[2.09783018 0.30560234]
 [2.13443044 2.13171694]]
model reconstruction error 0.0011599349216014024
```