MANIPAL
ACADEMY of HIGHER EDUCATION
(Deemed to be University under Section 3 of the UGC Act, 1956)

APPLIED MACHINE LEARNING LAB ASSIGNMENT

Submitted to

Manipal School of Information Sciences, MAHE, Manipal

Registration No	Name	Branch
211057020	**Aditi**	**Artificial Intelligence and Machine Learning**

MANIPAL SCHOOL OF INFORMATION SCIENCES
MANIPAL
(A constituent unit of MAHE, Manipal)

TABLE OF CONTENTS

LAB ASSIGNMENT 1

1) Write a program to prompt the user for hours and rate per hour to compute gross pay. Hour = 35 & Rate = 2.75

Answer:

We enter the hours and the rate as float value and then multiply hours and rate to get gross pay.

```
[ ]  hours=float(input("Enter the hours"))#enter the hours
     rate=float(input("Enter the rate")) #enter the rate
     pay=hours*rate
     print(pay)
```

```
Enter the hours35
Enter the rate2.75
96.25
```

2) Which will never be printed out of the following two code sets:

```
if x < 2 :
    print('Below 2')
elif x >= 2 :
    print('Two or more')
else :
    print('Something else')
```

```
if x < 2 :
    print('Below 2')
elif x < 20 :
    print('Below 20')
elif x < 10 :
    print('Below 10')
else :
    print('Something else')
```

Answer:

In the first code block else will never execute as all the values above or equal to 2 are satisfying the “elif” condition and all the values below two are satisfying the “if” condition.

```
x=4
if x<2:
  print('Below 2')
elif x>=2:
  print('two or more')
else:
  print('Something else') # else will never be executed because all values above 2 will satisfy the elif condition
```

```
two or more
```

In the second block elif < 10 will never execute because all values below 10 will satisfy the first elif condition.

```
[ ] x=9
    if x<2:
      print('Below 2')
    elif x<20:
      print('Below 20')
    elif x<10:
      print('Below 10')# elif x<10 will never be executed because all values below 10 will satisfy the first elif condition
    else:
      print('Something else')
```

```
Below 20
```

3) Rewrite the program-1 using try and except to prompt the user for hours and rate per hour to compute gross pay. (If non-numeric inputs entered, it should except). Hour = 35 & Rate = 2.75.

Answer:

When we enter a non numeric value it will execute the except block.

```
[ ]  hours=input("Enter the hours")
     rate=input("Enter the rate")
     try:
       hours=float(hours)
       rate=float(rate)
       pay=hours*rate
       print(pay)
     except:
       print("Not a numeric")
```

```
Enter the hoursharshi
Enter the rate2
Not a numeric
```

4) Rewrite the program-1 with time-and-a half for overtime and create a function called paycomp which takes two parameters (hours and rate per hour). Hours = 45 and rate = 10.

Answer:

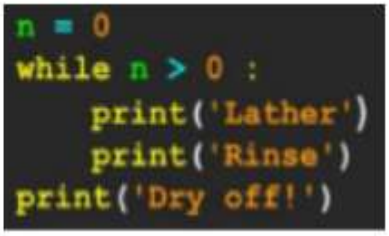
When hours is greater than 45 it will be multiplied with 1.5 times the rate.

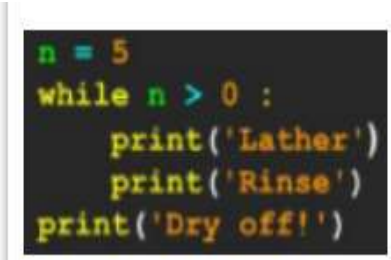
```
[ ]  def paycomp(hours,rate):
       pay=hours*rate
       if(hours>45):
         pay1=(hours-45)*(1.5*rate)+pay #multiplies by 1.5 when hours greater than 45
         print("the pay is",pay1)
       else:
         print("The pay is",pay)

     paycomp(55,10)
```

```
the pay is 700.0
```

5) What is the code doing?





Answer:

In the first case as the value of n=0 the while loop fails and only the last print statement executes.

In the second case as the n value is not updated so as it enters the while loop it executes infinitely and the last print statement never executes.

```
[ ]  n=5
     while n>0:
       print('Lather')
       print('Rinse')
     print('Dry off')#infinite loop
```

```
[ ]  n=0
     while n>0:
       print('Lather')
       print('Rinse')
     print('Dry off') # does not enter the while loop as it does not satisfy the while condition
```

```
Dry off
```

6) Consider the list of elements [9, 41, 23, 54, 33, 21, 8] use for loop to find

1. Largest number
2. Smallest number

3. Number of numbers
4. Number of odd numbers
5. Number of even numbers
6. Number of prime numbers
7. Sum and average of numbers
8. Filter the numbers greater than 20
9. Filter the numbers less than 15
10. Search for number 3

Answer:

```
[ ]  list1=[9, 41, 23, 54, 33, 21, 8]
     #maximum Number
     max=list1[0]
     for i in list1:
       if i>max:
         max=i
     print("Max is",max)

     #smallest Number
     min=list1[0]
     for i in list1:
       if i<min:
         min=i
     print("min is",min)

     #Number of Numbers
     count=0
     for i in list1:
       count=count+1
     print("Number of numbers is",count)

     #number of odd numbers
     odd=0
     for i in list1:
       if i%2!=0:
         odd=odd+1
     print("Number of odd no",odd)
```

```
[ ] #number of even numbers
    even=0
    for i in list1:
      if i%2==0:
        even=even+1
    print("Number of even no",even)


    #number of prime numbers
    counter=0;
    for i in list1:
      flag=0
      for j in range(2,i):
        if(i%j==0):
          flag=0
          break
        else:
          flag=1
      if flag==1:
          counter=counter+1;
    print("the count of prime numbers are",counter)
```

```
#sum and average of number
sum=cnt=0
for i in list1:
  sum=sum+i;
  cnt=cnt+1;
avg=sum/cnt
print("The sum is",sum)
print("The average is",avg)


#filter numbers greater than 20
list2=[]
for i in list1:
  if i>20:
    list2.append(i)
print("List after removing elements",list2)

#filter numbers less than 15
list3=[]
for i in list1:
  if i<15:
    list3.append(i)
print("List after removing elements",list3)
```

```
#search for element 3
list4=[]
list4.append(5)
a=[6,7,9,0,10,11,3]
list4.extend(a)
print("The list is",list4)
key=3
if key in list4:
    print("Key is found")
else:
  print("key is not found")
```

```
Max is 54
min is 8
Number of numbers is 7
Number of odd no 5
Number of even no 2
the count of prime numbers are 2
The sum is 189
The average is 27.0
List after removing elements [41, 23, 54, 33, 21]
List after removing elements [9, 8]
The list is [5, 6, 7, 9, 0, 10, 11, 3]
Key is found
```

7) Illustrate the use of type operator and type conversion

Answer:

```
[ ] int1=9
    print(type(int1)) # type operator is used to find which data type the variable belongs to
    int1=str(int1) #type conversion is used to convert one type to another
    print(type(int1))
```

```
<class 'int'>
<class 'str'>
```

8) Illustrate the use of break and continue

Answer:

```
for i in list1:
  if i==54:
    continue; #skips one iteration
  print(i)

for i in list1:
  if i==54:
    break; #comes out of the loop
  print(i)
```

```
9
41
23
33
21
8
9
41
23
```

LAB ASSIGNMENT 2

1) Find the shape and type of (a, a*2)

Answer:

Use shape to find the shape and the type of the variable.

```
import numpy as np
a=np.arange(10).reshape(2,5)
z=np.vstack((a,a*2))
print(z)
print(z.shape)
print(z.dtype)

b=np.arange(10)*2
np.array(b)
print(b)
print(b.shape)
print(b.dtype)
```

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [ 0  2  4  6  8]
 [10 12 14 16 18]]
(4, 5)
int64
[ 0  2  4  6  8 10 12 14 16 18]
(10,)
int64
```

---

2) An array a= (4, -3, -2, 1, 3, 4).

a) Find sum of the elements in the array.

b) Find the small and big elements of the array.

c) Find the mean and standard deviation of the array elements.

d) Find the index of the biggest and smallest elements of the array.

Answer:

We can use sum to find the sum () of the elements in the array.

min () can be used to find the minimum elements and max () can be used to find the maximum elements in the array.

mean () and sd () can be used to find the mean and the standard deviation of the elements in array respectively.

argmax () and argmin () can be used to find the index of the largest element and the index of the smallest element in the array respectively.

```
[ ]  arr=np.array([4, -3, -2, 1, 3, 4])
     print(arr)
```

```
[ 4 -3 -2  1  3  4]
```

find sum of the elements in the array

```
[ ] #sum of the elements in the array
    print(sum(arr))
```

```
7
```

find the small and big elements of the array

```
[ ] print(min(arr)) #prints the smallest element in the array
```

```
-3
```

```
[ ] print(max(arr))#prints the largest element in the array
```

```
4
```

find the mean and standard deviation of the array elements

```
[ ] print(arr.mean())#mean of the array elements
```

```
1.1666666666666667
```

```
print(arr.std())#standard deviation of elements
```

```
2.793842435706702
```

find the index of the biggest and smallest elements of the array

```
[ ] print(arr.argmax())#index of maximum element
```

```
0
```

```
print(arr.argmin())#index of minimum element
```

```
1
```

3) Find the dot product of (2j, 3j), (3j, 2j)

Answer:

Use dot function to perform dot product.

```
[ ]  arr1=np.array([2j,3j])
     arr2=np.array([3j,2j])
     np.dot(arr1,arr2) #performs the dot product
```

```
(-12+0j)
```

4) Create an array which contain square of all integer numbers between 0 and 13

Answer:

```
[ ]  arr3=np.arange(13)**2 #returns the square of the elements uptil 12
     arr3
```

```
array([  0,   1,   4,   9,  16,  25,  36,  49,  64,  81, 100, 121, 144])
```

5) Use bracket notation to get value particular index.

Answer:

```
[ ]  arr5=np.random.randint(-5,9,size=10)
     print(arr5)
```

```
[-4  8  8  3  1 -3 -1 -4  6 -3]
```

```
[ ]  arr5[4] #returns the 4th element in the array
```

```
1
```

```
[ ]  arr5[:5] #returns elements of the array till the 4th index excluding 5th element
```

```
array([-4,  8,  8,  3,  1])
```

```
[ ]  arr5[2:6] #returns the elements corresponding to index 2 to index 5
```

```
array([ 8,  3,  1, -3])
```

```
     arr5[-4]#returns the 4th element from the back of the array
```

```
-1
```

```
[ ]  arr5[-4:-1] # returns the  4th elemnt from the back of the array till the last but one element
```

```
array([-1, -4,  6])
```

```
[ ]  arr5[-4: ] #returns all the elements from the index -4 to the end of the array
```

```
array([-1, -4,  6, -3])
```

```
[ ]  arr5[-5::-2] #returns the elements from the 5th index from behind of the array with steps of -2
```

```
array([-3,  3,  8])
```

```
[ ]  arr5[-5::2] #returns the elements from the 5th index from behind of the array with steps of 2
```

```
array([-3, -4, -3])
```

LAB ASSIGNMENT 3

Implementing Decision Tree, KNN, Naïve Bayes on Tele-Customer Dataset

Tele-Customer Dataset

In this dataset the data of the telecommunication customers is stored like their age, income, marital status and on the basis of that they are classified into 4 categories A, B, C, and D.

	A	B	C	D	E	F	G	H	I	J	K	L
1	region	tenure	age	income	marital	address	ed	employ	retire	gender	reside	custcat
2	2	13	44	64	1	9	4	5	0	0	2	A
3	3	11	33	136	1	7	5	5	0	0	6	D
4	3	68	52	116	1	24	1	29	0	1	2	C
5	2	33	33	33	0	12	2	0	0	1	1	A
6	2	23	30	30	1	9	1	2	0	0	4	C
7	2	41	39	78	0	17	2	16	0	1	1	C
8	3	45	22	19	1	2	2	4	0	1	5	B
9	2	38	35	76	0	5	2	10	0	0	3	D
10	3	45	59	166	1	7	4	31	0	0	5	C
11	1	68	41	72	1	21	1	22	0	0	3	B
12	2	5	33	125	0	10	4	5	0	1	1	A
13	3	7	35	80	0	14	2	15	0	1	1	C
14	1	41	38	37	1	8	2	9	0	1	3	A
15	2	57	54	115	1	30	4	23	0	1	3	D
16	2	9	46	25	0	3	1	8	0	1	2	A
17	1	29	38	75	1	12	5	1	0	0	4	B
18	3	60	57	162	0	38	2	30	0	0	1	C
19	3	34	48	49	0	3	2	6	0	1	3	C
20	2	1	24	20	0	3	1	3	0	0	1	A
21	1	26	29	77	1	3	4	2	0	0	4	D
22	3	6	30	16	0	7	3	1	0	1	1	B
23	1	68	52	120	1	17	1	24	0	0	2	A

Telecust1

Decision Tree

First step is to import the necessary libraries

```
[1]  #importing the necessary libraries
     import numpy as np
     import matplotlib.pyplot as plt
     import pandas as pd
```

Second step is to import the dataset

```
[2]  #importing the dataset
     dataset=pd.read_csv('Telecust1.csv')
     dataset
```

	region	tenure	age	income	marital	address	ed	employ	retire	gender	reside	custcat
0	2	13	44	64	1	9	4	5	0	0	2	A
1	3	11	33	136	1	7	5	5	0	0	6	D
2	3	68	52	116	1	24	1	29	0	1	2	C
3	2	33	33	33	0	12	2	0	0	1	1	A
4	2	23	30	30	1	9	1	2	0	0	4	C
...	...	...	...	...	...	...	...	...	...	...	...	...
995	3	10	39	27	0	0	3	0	0	1	3	A
996	1	7	34	22	0	2	5	5	0	1	1	A
997	3	67	59	944	0	40	5	33	0	1	1	D

Third step is to define the dependent and the independent variables

```
[3]  #Defining the independent variables
     X=dataset.iloc[:,0:11].values
     X
```

```
array([[ 2, 13, 44, ...,  0,  0,  2],
       [ 3, 11, 33, ...,  0,  0,  6],
       [ 3, 68, 52, ...,  0,  1,  2],
       ...,
       [ 3, 67, 59, ...,  0,  1,  1],
       [ 3, 70, 49, ...,  0,  1,  1],
       [ 3, 50, 36, ...,  0,  1,  3]])
```

```
[4]  #Defining the independent variables
     y=dataset.iloc[:,11].values
     y
```

```
array(['A', 'D', 'C', 'A', 'C', 'C', 'B', 'D', 'C', 'B', 'A', 'C', 'A',
       'D', 'A', 'B', 'C', 'C', 'A', 'D', 'B', 'A', 'D', 'A', 'C', 'C',
       'D', 'C', 'A', 'B', 'B', 'D', 'C', 'B', 'D', 'B', 'D', 'C', 'C',
       'C', 'C', 'C', 'A', 'D', 'C', 'B', 'B', 'D', 'A', 'C', 'A', 'B',
       'D', 'A', 'A', 'A', 'B', 'A', 'C', 'B', 'C', 'D', 'A', 'D', 'D',
       'A', 'C', 'C', 'A', 'A', 'A', 'B', 'A', 'B', 'C', 'A', 'C', 'A',
       'C', 'D', 'D', 'A', 'D', 'C', 'B', 'A', 'B', 'D', 'D', 'C', 'B',
       'A', 'B', 'D', 'A', 'D', 'C', 'B', 'C', 'B', 'D', 'C', 'C', 'B',
       'B', 'B', 'A', 'D', 'A', 'B', 'B', 'B', 'A', 'B', 'C', 'A', 'C',
       'C', 'C', 'A', 'A', 'A', 'A', 'B', 'A', 'B', 'D', 'B', 'B', 'D',
       'C', 'A', 'D', 'B', 'C', 'A', 'A', 'C', 'C', 'D', 'D', 'C', 'D',
       'B', 'C', 'A', 'D', 'B', 'B', 'B', 'A', 'C', 'D', 'D', 'D', 'C',
       'A', 'A', 'B', 'C', 'C', 'C', 'A', 'C', 'B', 'B', 'D', 'A', 'A',
       'B', 'D', 'A', 'D', 'A', 'B', 'A', 'C', 'D', 'D', 'A', 'D', 'A',
```

Fourth step is to split dataset into training and testing and as we want all the features in the data on a same scale we perform feature scaling.

```
[5]  #Splitting the dataset into the Training set and Test set
     from sklearn.model_selection import train_test_split
     Xtrain,Xtest,ytrain,ytest=train_test_split(X,y,test_size=0.25,random_state=0)
```

```
[6]  #Feature Scaling
     from sklearn.preprocessing import StandardScaler
     sc=StandardScaler()
     Xtrain=sc.fit_transform(Xtrain)
     Xtest=sc.transform(Xtest)
     Xtrain
```

```
array([[-1.24415351,  0.4587481 ,  1.94630851, ..., -0.20764257,
         0.96333824, -0.93866715],
       [-1.24415351, -1.21642078, -1.07508659, ..., -0.20764257,
        -1.038057  ,  0.44443769],
       [-0.01798382, -0.09964153, -1.15459698, ..., -0.20764257,
        -1.038057  ,  1.13599011],
       ...,
       [-0.01798382, -0.37883634, -1.39312818, ..., -0.20764257,
         0.96333824,  1.13599011],
       [-0.01798382,  0.73794292, -0.43900341, ..., -0.20764257,
        -1.038057  , -0.24711473],
       [ 1.20818587,  1.6685923 ,  2.02581891, ..., -0.20764257,
         0.96333824, -0.24711473]])
```

In the fifth step we fit the decision tree classification to the training set.

```
[7]  #fitting decision tree classification to the training set
     from sklearn.tree import DecisionTreeClassifier
     classifier=DecisionTreeClassifier(criterion='entropy',random_state=0)
     classifier.fit(Xtrain,ytrain)
```

```
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

In the sixth step we perform the prediction.

```
[8]  #Predicting the Test set results
     y_pred=classifier.predict(Xtest)
     y_pred
```

```
array(['B', 'A', 'D', 'A', 'B', 'D', 'C', 'B', 'D', 'A', 'A', 'A', 'C',
       'C', 'C', 'B', 'C', 'D', 'A', 'A', 'C', 'A', 'A', 'B', 'B', 'B',
       'D', 'D', 'D', 'B', 'C', 'A', 'C', 'D', 'B', 'C', 'C', 'C', 'B',
       'B', 'D', 'C', 'D', 'D', 'C', 'D', 'D', 'D', 'B', 'A', 'C', 'C',
       'B', 'C', 'C', 'C', 'D', 'C', 'A', 'B', 'C', 'A', 'D', 'C', 'C',
       'A', 'D', 'A', 'D', 'D', 'B', 'A', 'B', 'D', 'B', 'B', 'C', 'C',
       'C', 'A', 'A', 'B', 'B', 'A', 'C', 'A', 'B', 'A', 'B', 'C', 'C',
       'C', 'C', 'C', 'C', 'C', 'A', 'D', 'B', 'B', 'A', 'A', 'C', 'B',
       'D', 'B', 'D', 'D', 'A', 'C', 'C', 'D', 'D', 'A', 'A', 'D', 'D',
       'D', 'D', 'B', 'B', 'C', 'A', 'D', 'C', 'C', 'B', 'B', 'A', 'A',
       'B', 'C', 'A', 'B', 'D', 'B', 'C', 'B', 'A', 'D', 'D', 'C', 'D',
       'D', 'C', 'C', 'D', 'D', 'D', 'D', 'D', 'C', 'C', 'C', 'A', 'B',
       'A', 'B', 'D', 'A', 'A', 'D', 'B', 'A', 'C', 'B', 'A', 'C', 'C',
       'D', 'B', 'D', 'D', 'D', 'C', 'C', 'C', 'C', 'A', 'A', 'C', 'D',
       'D', 'B', 'A', 'D', 'D', 'C', 'B', 'D', 'B', 'C', 'D', 'B', 'C',
       'D', 'A', 'B', 'A', 'B', 'C', 'A', 'A', 'C', 'B', 'A', 'B', 'A',
       'B', 'A', 'B', 'C', 'C', 'A', 'C', 'A', 'C', 'B', 'C', 'C', 'C',
       'D', 'A', 'A', 'B', 'C', 'A', 'C', 'A', 'C', 'B', 'A', 'C', 'A'.
```

In the next step we print the confusion matrix and the classification report.

```
[10] #Making the confusion matrix
     from sklearn.metrics import confusion_matrix,classification_report
     cm=confusion_matrix(ytest,y_pred)
     cm
```

```
array([[26, 11, 24, 11],
       [14, 12, 16, 14],
       [12, 15, 26, 16],
       [ 7, 17, 13, 16]])
```

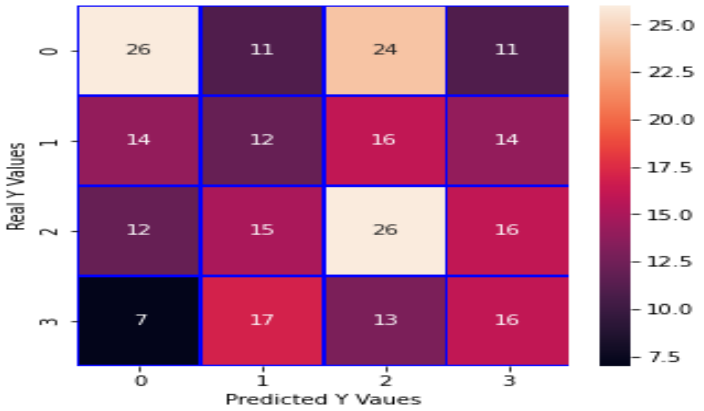
```
[11] print(classification_report(ytest,y_pred))
```

```
              precision    recall  f1-score   support

           A       0.44      0.36      0.40        72
           B       0.22      0.21      0.22        56
           C       0.33      0.38      0.35        69
           D       0.28      0.30      0.29        53

    accuracy                           0.32       250
   macro avg       0.32      0.31      0.31       250
weighted avg       0.33      0.32      0.32       250
```

```
[13] import seaborn as sns
     f,ax=plt.subplots(figsize=(5,5))
     sns.heatmap(cm, linecolor="blue",annot=True,linewidths=0.5)
     plt.xlabel("Predicted Y Vaues")
     plt.ylabel("Real Y Values")
     plt.show()
```



Finally we find the accuracy of the model.

```
[12] from sklearn import metrics #import scikit learn metrics module for accuracy calculation
     #Model Accuracy, how often is the classifier correct?
     prediction=metrics.accuracy_score(ytest,y_pred)
     print('Accuracy:',prediction*100,'%')
```

```
Accuracy: 32.0 %
```

We can see that the decision tree gives 32% accuracy so the model needs to be improvised.

KNN

The same first four steps are followed for KNN as well. In the 5th step we fit the KNN model to the training data.

```
[14] #Fitting KNN to the training set
     from sklearn.neighbors import KNeighborsClassifier
     classifier=KNeighborsClassifier(n_neighbors=5,metric='minkowski',p=2)
     classifier.fit(Xtrain,ytrain)
```

```
KNeighborsClassifier()
```

Again we follow the same steps performed for the decision tree and then finally we print the accuracy of the model.

```
[19] from sklearn import metrics #import scikit learn metrics module for accuracy calculation
     #Model Accuracy, how often is the classifier correct?
     prediction=metrics.accuracy_score(ytest,y_pred)
     print('Accuracy:',prediction*100,'%')
```

```
Accuracy: 33.6 %
```

We can see a 1% difference in the accuracy of the model when compared to DT.

NAÏVE BAYES

Perform the same steps as for the DT and KNN in the fifth step fit the Naïve Bayes classifier on the training data.

```
[21] #fitting naive bayes to the training set
     from sklearn.naive_bayes import GaussianNB
     classifier=GaussianNB()
     classifier.fit(Xtrain,ytrain)

     GaussianNB()
```

After prediction print the accuracy of the model.

```
[26] from sklearn import metrics #import scikit learn metrics module for accuracy calculation
     #Model Accuracy, how often is the classifier correct?
     prediction=metrics.accuracy_score(ytest,y_pred)
     print('Accuracy:',prediction*100,'%')

     Accuracy: 38.4 %
```

The accuracy has improved when compared to that of the DT and KNN.

LAB ASSIGNMENT 4

Implementing hierarchical clustering on Customer Purchase Dataset

Hierarchical Clustering

Hierarchical clustering (also called hierarchical cluster analysis or HCA) is a method of cluster analysis which seeks to build a hierarchy of clusters. Strategies for hierarchical clustering generally fall into two types:

- **Agglomerative**: This is a "bottom-up" approach: each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy.
- **Divisive**: This is a "top-down" approach: all observations start in one cluster, and splits are performed recursively as one moves down the hierarchy.

In general, the merges and splits are determined in a greedy manner. The results of hierarchical clustering are usually presented in a dendrogram.

Customer Purchase Dataset

In this dataset the information on the products purchased by a customer, the sales, shipping cost, profit and based on these divides the customers into 3 categories

	A	B	C	D	E	F	G	H	I
1	Products	Prod_id	Cust_id	Sales	Order_Quan	Profit	Shipping_Cost	Product_Base_Margin	Customer_Segme
2	COPIERS AND FAX	Prod_16	Cust_1088	10909.64	40	2616.46	22	0.57	HOME OFFICE
3	COPIERS AND FAX	Prod_16	Cust_1820	10607.64	27	2977.36	23.69	0.56	CORPORATE
4	PAPER	Prod_6	Cust_1659	754.641	13	29.8	8.9	0.37	SMALL BUSINESS
5	PAPER	Prod_6	Cust_622	572.641	22	92.78	10.99	0.39	SMALL BUSINESS
6	OFFICE MACHINES	Prod_17	Cust_564	10128.64	50	5133.68	20.5	0.36	SMALL BUSINESS
7	COPIERS AND FAX	Prod_16	Cust_1388	10378.64	54	5504.1	27.99	0.56	SMALL BUSINESS
8	COPIERS AND FAX	Prod_16	Cust_1052	11135.64	27	3317.67	17.5	0.59	HOME OFFICE
9	COPIERS AND FAX	Prod_16	Cust_346	11337.64	52	2800.43	20	0.6	CORPORATE
10	COPIERS AND FAX	Prod_16	Cust_1440	10992.64	36	3553.31	16.2	0.55	HOME OFFICE
11	COPIERS AND FAX	Prod_16	Cust_444	10314.64	24	2954.96	21.69	0.57	CORPORATE
12	BINDERS AND BINDER ACCESSORIES	Prod_3	Cust_513	9553.641	48	2736.56	20.03	0.39	CORPORATE
13	TELEPHONES AND COMMUNICATION	Prod_4	Cust_1457	8763.641	48	3671.13	24.8	0.56	HOME OFFICE
14	COPIERS AND FAX	Prod_16	Cust_113	10882.64	50	3624.16	19.08	0.55	SMALL BUSINESS
15	TELEPHONES AND COMMUNICATION	Prod_4	Cust_710	11259.64	25	2786.66	23	0.8	CORPORATE
16	ENVELOPES	Prod_9	Cust_604	891.641	39	31.41	25.08	0.39	CORPORATE
17	COPIERS AND FAX	Prod_16	Cust_62	11214.64	55	4608.87	19.5	0.83	SMALL BUSINESS
18	COPIERS AND FAX	Prod_16	Cust_1757	10410.64	18	2596.88	19.99	0.56	HOME OFFICE
19	COPIERS AND FAX	Prod_16	Cust_377	10549.64	23	3387.63	26.99	0.58	CORPORATE
20	COPIERS AND FAX	Prod_16	Cust_97	11167.64	24	2900.46	23.1	0.57	CORPORATE
21	COPIERS AND FAX	Prod_16	Cust_1597	10124.64	49	3953.32	20.9	0.83	CORPORATE
22	TELEPHONES AND COMMUNICATION	Prod_4	Cust_1597	10416.64	33	3841.49	18.26	0.4	CORPORATE

data Sheet1

First step is importing the necessary libraries.

```
[1] import pandas as pd
    import  matplotlib.pyplot as plt
    import numpy as np
```

Second step we import the dataset.

There are multiple products and each product is purchased for a different purpose. Our aim is to make clusters from this data that can segment similar clients together.

```
[2] #importing the dataset with pandas
    dataset=pd.read_excel('purchase.xlsx')
    dataset
```

	Products	Prod_id	Cust_id	Sales	Order_Quan	Profit	Shipping_Cost	Product_Base_Margin	Customer_Segment
0	COPIERS AND FAX	Prod_16	Cust_1068	10909.641	40	2616.46	22.00	0.57	HOME OFFICE
1	COPIERS AND FAX	Prod_16	Cust_1820	10607.641	27	2977.36	23.69	0.56	CORPORATE
2	PAPER	Prod_6	Cust_1659	754.641	13	29.80	8.90	0.37	SMALL BUSINESS
3	PAPER	Prod_6	Cust_622	572.641	22	92.78	10.99	0.39	SMALL BUSINESS
4	OFFICE MACHINES	Prod_17	Cust_564	10128.641	50	5133.68	20.50	0.36	SMALL BUSINESS
...	...	...	...	...	...	...	...	...	...
5972	COPIERS AND FAX	Prod_16	Cust_1397	20265.220	47	6168.64	14.70	0.37	HOME OFFICE
5973	COPIERS AND FAX	Prod_16	Cust_247	24559.910	47	7358.66	16.63	0.37	SMALL BUSINESS
5974	COPIERS AND FAX	Prod_16	Cust_1795	28359.400	49	14440.39	24.49	0.52	SMALL BUSINESS
5975	COPIERS AND FAX	Prod_16	Cust_595	41343.210	8	3852.19	24.49	0.37	HOME OFFICE
5976	COPIERS AND FAX	Prod_16	Cust_1266	25312.000	48	8788.81	16.63	0.50	CORPORATE

5977 rows × 9 columns

Third step we perform feature scaling and then normalize the data so that the scaling is same for each feature.

```
X=dataset.iloc[:,[1,3]].values
X
```

```
array([[1.0909641e+04, 2.6164600e+03],
       [1.0607641e+04, 2.9773600e+03],
       [7.5464100e+02, 2.9800000e+01],
       ...,
       [2.8359400e+04, 1.4440390e+04],
       [4.1343210e+04, 3.8521900e+03],
       [2.5312000e+04, 8.7888100e+03]])
```

```
[6] #Feature Scaling
    from sklearn.preprocessing import StandardScaler
    sc=StandardScaler()
    X=sc.fit_transform(X)
```

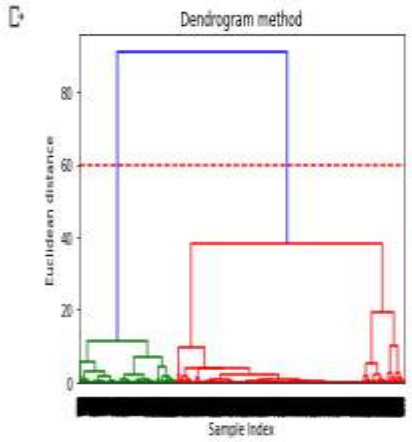
```
[7] from sklearn.preprocessing import normalize
    data_scaled=normalize(X)
```

Fourth step we generate a dendrogram to help us decide with the number of clusters.

```
#using dendrogram method to find the optimal number of clusters
import scipy.cluster.hierarchy as sch
dendrogram=sch.dendrogram(sch.linkage(data_scaled,method='ward')) #ward is a within cluster variance methods finds dist btwn num and minimizes the variance
plt.title('Dendrogram method')
plt.xlabel('Sample Index')
plt.ylabel('Euclidean distance')
plt.show()
```

The X-axis contains the samples and the y axis contains the Euclidean distance. The vertical line with the maximum distance is the blue line so we can decide on the threshold value to be 60 and cut the dendrogram.

```
dendrogram=sch.dendrogram(sch.linkage(data_scaled,method='ward')) #ward is a within cluster variance methods finds dist btwn num and minimizes the variance
plt.axhline(y=60,color='r',linestyle='--')
plt.title('Dendrogram method')
plt.xlabel('Sample Index')
plt.ylabel('Euclidean distance')
plt.show()
```



We have two clusters at this point and now we apply the agglomerative clustering.

```
[10] #fitting HC to the dataset
     #agglomerative HC

     from sklearn.cluster import AgglomerativeClustering
     hc=AgglomerativeClustering(n_clusters=2, affinity="euclidean",linkage='ward')

     y_hc=hc.fit_predict(X)
     print(y_hc)
```

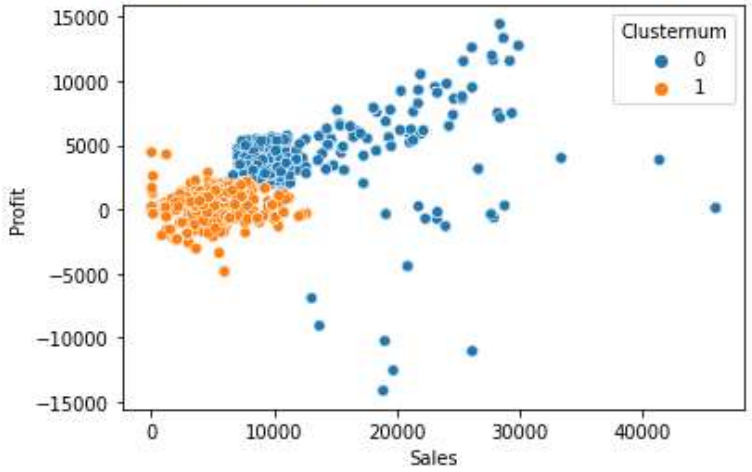
```
     [0 0 1 ... 0 0 0]
```

```
[11] dataset['Clusternum']=hc.labels_
```

We can see the values of 0s and 1s in the output since we defined 2 clusters. 0 represents the points that belong to the first cluster and 1 represents points in the second cluster. We will now visualize the two clusters.

```
[14] import seaborn as sns
     sns.scatterplot(x='Sales',y='Profit',data=dataset,hue='Clusternum')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f80d85fb450>
```



LAB ASSIGNMENT 5

Problem Set:-

Use Numeric, Pandas, and other Python libraries

a. Write the Python code to compute entropy and information gain
b. Write the Python code to demonstrate a conditional probability
c. Write the Python code to compute Euclidean Distance between data points
d. Write the Python code to calculate covariance matrix, Eigenvalues, and Eigenvectors
e. Write the Python code to calculate the following
    a. Accuracy
    b. Misclassification
    c. Type-1 and Type-2 error rates
    d. Sensitivity
    e. Specificity

Answers:

a) We import the necessary libraries and load the dataset.

```
[1]  import numpy
     import pandas as pd
     import math
```

```
[2]  midwest = pd.read_csv('midwest.csv')
```

2) To compute the counts of each unique value in the column

Counts = np.bincount (column)

3) We divide by the total column length to get the probability of each counts

```
probabilities = counts / len(column)
```

4) We can initialize a variable name entropy and set it to 0

Entropy=0

5) Loop through the probabilities and add each one to the total entropy.

We use log from math module and set base 2 to compute the entropy.

6) In the final step we return the negative entropy

Return –entropy

```
[3]  def calc_entropy(column):
         """
         Calculate entropy given a pandas series, list, or numpy array.
         """
         # Compute the counts of each unique value in the column
         counts = np.bincount(column)
         # Divide by the total column length to get a probability
         probabilities = counts / len(column)

         # Initialize the entropy to 0
         entropy = 0
         # Loop through the probabilities, and add each one to the total entropy
         for prob in probabilities:
             if prob > 0:
                 # use log from math and set base to 2
                 entropy += prob * math.log(prob, 2)

         return -entropy
```

Entropy result:

```
[25] entropy=calc_entropy(midwest['midwest?'])
     entropy

     0.9182958340544896
```

Information gain

```
def calc_information_gain(data, split_name, target_name):
    """
    Calculate information gain given a data set, column to split on, and target
    """
    # Calculate the original entropy
    original_entropy = calc_entropy(data[target_name])

    #Find the unique values in the column
    values = data[split_name].unique()


    # Make two subsets of the data, based on the unique values
    left_split = data[data[split_name] == values[0]]
    right_split = data[data[split_name] == values[1]]

    # Loop through the splits and calculate the subset entropies
    to_subtract = 0
    for subset in [left_split, right_split]:
        prob = (subset.shape[0] / data.shape[0])
        to_subtract += prob * calc_entropy(subset[target_name])

    # Return information gain
    return original_entropy - to_subtract
```

Information Gain result:

```
columns = ['apple_pie?', 'potato_salad?', 'sushi?']

for col in columns:
  information_gain=calc_information_gain(midwest,col,'midwest?')
  print(information_gain)

0.03170514719803608
0.059773130149317 4
0.24902249956730627
```

b)

```
[44] import pandas as pd
     import numpy as np

     #create pandas DataFrame with raw data
     df = pd.DataFrame({'gender': np.repeat(np.array(['Male', 'Female']), 150),
                        'sport': np.repeat(np.array(['Baseball', 'Basketball', 'Football',
                                                     'Soccer', 'Baseball', 'Basketball',
                                                     'Football', 'Soccer']),
                                           (34, 40, 58, 18, 34, 52, 20, 44))})

     #produce contingency table to summarize raw data
     survey_data = pd.crosstab(index=df['gender'], columns=df['sport'], margins=True)

     #view contingency table
     survey_data
```

sport	Baseball	Basketball	Football	Soccer	All
gender					
Female	34	52	20	44	150
Male	34	40	58	18	150
All	68	92	78	62	300

```
[45] #calculate probability of being male, given that individual prefers baseball
     survey_data.iloc[1, 0] / survey_data.iloc[2, 0]
```

```
0.5
```

```
[46] #calculate probability of preferring basketball, given that individual is female
     survey_data.iloc[0, 1] / survey_data.iloc[0, 4]
```

```
0.3466666666666667
```

c) We can import the distance module from the Scipy.spatial library to calculate the Euclidean distance.

```
[38] from scipy.spatial import distance
     a = [1, 3, 27]
     b=  [3, 6, 8]
     print(distance.euclidean(a,b))
```

```
19.339079605813716
```

d) We can calculate the covariance matrix using numpy library and Eigenvalues and Eigenvectors from “linalg” library and importing “eig” module.

```
[43] import numpy as np
     from sklearn.preprocessing import StandardScaler
     from numpy.linalg import eig
     #
     # Percentage of marks and no. of hours studied
     #
     students = np.array([[85.4, 5],
                 [82.3, 6],
                 [97, 7],
                 [96.5, 6.5]])
     #
     # Scale the features
     #
     sc = StandardScaler()
     students_scaled = sc.fit_transform(students)
     #
     # Calculate covariance matrix; One can also use the following
     # code: np.cov(students_scaled, rowvar=False)
     #
     cov_matrix = np.cov(students_scaled.T)
     print("Cov-matrix:\n",cov_matrix)
     #
     # Calculate Eigenvalues and Eigenmatrix
     #
     eigenvalues, eigenvectors = eig(cov_matrix)
     print("Eigen values:",eigenvalues)
     print("Eigen vector:\n",eigenvectors)
```

```
Cov-matrix:
 [[1.33333333 1.01240122]
 [1.01240122 1.33333333]]
Eigen values: [2.34573455 0.32093211]
Eigen vector:
 [[ 0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]
```

e) In the below code we need to pass the confusion matrix to the defined function.

```python
def confusion_metrics (conf_matrix):
# save confusion matrix and slice into four pieces
    TP = conf_matrix[1][1]
    TN = conf_matrix[0][0]
    FP = conf_matrix[0][1]
    FN = conf_matrix[1][0]
    print('True Positives:', TP)
    print('True Negatives:', TN)
    print('False Positives:', FP)
    print('False Negatives:', FN)

    # calculate accuracy
    conf_accuracy = (float (TP+TN) / float(TP + TN + FP + FN))

    # calculate mis-classification
    conf_misclassification = 1- conf_accuracy

    # calculate the sensitivity
    conf_sensitivity = (TP / float(TP + FN))
    # calculate the specificity
    conf_specificity = (TN / float(TN + FP))

    # calculate precision
    conf_precision = (TN / float(TN + FP))
    # calculate f_1 score
    conf_f1 = 2 * ((conf_precision * conf_sensitivity) / (conf_precision + conf_sensitivity))
    print('-'*50)
    print(f'Accuracy: {round(conf_accuracy,2)}')
    print(f'Mis-Classification: {round(conf_misclassification,2)}')
    print(f'Sensitivity: {round(conf_sensitivity,2)}')
    print(f'Specificity: {round(conf_specificity,2)}')
    print(f'Precision: {round(conf_precision,2)}')
    print(f'f_1 Score: {round(conf_f1,2)}')
```

LAB ASSIGNMENT 6

Download iris dataset from UCI machine learning repository and use the dataset to develop the following classifiers and find the accuracy of the model. Compare and comment on the results:

1. Decision Tree Classifier
2. Naïve Bayes Classifier
3. Logistic Regression classifier
4. K-NN classifier
5. Logistic Regression model and apply PAC
6. SVM – linear and non-linear classifiers
7. Random Forest Model
8. Ada Boost Model

Answer:

1. Import the libraries and the dataset.

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

```
#importing the iris dataset with pandas
dataset=pd.read_csv('IRIS.csv')
dataset
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica

2. Split the dataset into train and test and perform feature scaling, and label encoding.

```
[ ]  X=dataset.iloc[:,[0,1]].values
     y=dataset.iloc[:,4].values
```

```
#Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
Xtrain,Xtest,ytrain,ytest=train_test_split(X,y,test_size=0.25,random_state=0)
```

```
[ ]  #Feature Scaling
     from sklearn.preprocessing import StandardScaler
     sc=StandardScaler()
     Xtrain=sc.fit_transform(Xtrain)
     Xtest=sc.transform(Xtest)
     #Xtrain
```

```
[ ]  from sklearn.preprocessing import LabelEncoder
     labelencoder_y=LabelEncoder()
     y=labelencoder_y.fit_transform(y)
     print(y)
```

```
[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 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 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 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 2]
```

3. Fit the models and find their accuracy

Decision Tree

```
[ ] #fitting decision tree classification to the training set
    from sklearn.tree import DecisionTreeClassifier
    classifier=DecisionTreeClassifier(criterion='entropy',random_state=0)
    classifier.fit(Xtrain,ytrain)
```

```
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

```
[ ] #prdeicting the test set results
    y_pred=classifier.predict(Xtest)
    y_pred
```

```
array(['Iris-versicolor', 'Iris-virginica', 'Iris-setosa',
       'Iris-virginica', 'Iris-setosa', 'Iris-versicolor', 'Iris-setosa',
       'Iris-versicolor', 'Iris-virginica', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-virginica', 'Iris-versicolor',
       'Iris-virginica', 'Iris-versicolor', 'Iris-setosa',
       'Iris-virginica', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',
       'Iris-virginica', 'Iris-setosa', 'Iris-setosa', 'Iris-virginica',
       'Iris-versicolor', 'Iris-setosa', 'Iris-versicolor', 'Iris-setosa',
       'Iris-setosa', 'Iris-versicolor', 'Iris-virginica',
       'Iris-versicolor', 'Iris-setosa', 'Iris-virginica'], dtype=object)
```

```
[ ] ytest
```

```
array(['Iris-virginica', 'Iris-versicolor', 'Iris-setosa',
       'Iris-virginica', 'Iris-setosa', 'Iris-virginica', 'Iris-setosa',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-virginica', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',
       'Iris-virginica', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',
       'Iris-virginica', 'Iris-setosa', 'Iris-setosa', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-setosa', 'Iris-virginica',
       'Iris-versicolor', 'Iris-setosa', 'Iris-virginica',
       'Iris-virginica', 'Iris-versicolor', 'Iris-setosa',
       'Iris-versicolor'], dtype=object)
```

```
[ ] #making confusion matrix
    from sklearn.metrics import confusion_matrix
    cm=confusion_matrix(ytest,y_pred)
    cm
```

```
array([[13,  0,  0],
       [ 1,  8,  7],
       [ 0,  6,  3]])
```

```
[ ] from sklearn import metrics #import scikit learn metrics module for accuracy calculation
    #Model Accuracy, how often is the classifier correct?
    prediction=metrics.accuracy_score(ytest,y_pred)
    print('Accuracy:',prediction*100,'%')
```

```
Accuracy: 63.1578947368421 %
```

Accuracy of DT is 63.15%.

KNN

```
[ ]  #Fitting KNN to the training set
     from sklearn.neighbors import KNeighborsClassifier
     classifier=KNeighborsClassifier(n_neighbors=5,metric='minkowski',p=2)
     classifier.fit(Xtrain,ytrain)
```

```
     KNeighborsClassifier()
```

```
[ ]  #prdeicting the test set results
     y_pred=classifier.predict(Xtest)
     y_pred
```

```
     array(['Iris-versicolor', 'Iris-versicolor', 'Iris-setosa',
            'Iris-virginica', 'Iris-setosa', 'Iris-virginica', 'Iris-setosa',
            'Iris-versicolor', 'Iris-virginica', 'Iris-virginica',
            'Iris-versicolor', 'Iris-virginica', 'Iris-virginica',
            'Iris-virginica', 'Iris-virginica', 'Iris-setosa',
            'Iris-virginica', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',
            'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',
            'Iris-virginica', 'Iris-setosa', 'Iris-setosa', 'Iris-virginica',
            'Iris-versicolor', 'Iris-setosa', 'Iris-virginica',
            'Iris-versicolor', 'Iris-setosa', 'Iris-virginica',
            'Iris-virginica', 'Iris-versicolor', 'Iris-setosa',
            'Iris-versicolor'], dtype=object)
```

```
[ ]  #making confusion matrix
     from sklearn.metrics import confusion_matrix
     cm=confusion_matrix(ytest,y_pred)
     cm
```

```
     array([[13,  0,  0],
            [ 0,  8,  8],
            [ 0,  3,  6]])
```

```
[ ]  from sklearn import metrics #import scikit learn metrics module for accuracy calculation
     #Model Accuracy, how often is the classifier correct?
     prediction=metrics.accuracy_score(ytest,y_pred)
     print('Accuracy:',prediction*100,'%')
```

```
     Accuracy: 71.05263157894737 %
```

Accuracy of KNN is 71.05%.

Naïve Bayes

```
[ ]  #fitting naive bayes to the training set
     from sklearn.naive_bayes import GaussianNB
     classifier=GaussianNB()
     classifier.fit(Xtrain,ytrain)
```

```
GaussianNB()
```

```
[ ]  #prdeicting the test set results
     y_pred=classifier.predict(Xtest)
     y_pred
```

```
array(['Iris-versicolor', 'Iris-versicolor', 'Iris-setosa',
       'Iris-virginica', 'Iris-setosa', 'Iris-virginica', 'Iris-setosa',
       'Iris-virginica', 'Iris-virginica', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-virginica', 'Iris-versicolor',
       'Iris-virginica', 'Iris-versicolor', 'Iris-setosa',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',
       'Iris-versicolor', 'Iris-setosa', 'Iris-setosa', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-setosa', 'Iris-virginica',
       'Iris-versicolor', 'Iris-setosa', 'Iris-versicolor',
       'Iris-virginica', 'Iris-versicolor', 'Iris-setosa',
       'Iris-versicolor'], dtype='<U15')
```

```
[ ]  #making confusion matrix
     from sklearn.metrics import confusion_matrix
     cm=confusion_matrix(ytest,y_pred)
     cm
```

```
array([[13,  0,  0],
       [ 0, 12,  4],
       [ 0,  5,  4]])
```

```
[ ]  from sklearn import metrics #import scikit learn metrics module for accuracy calculation
     #Model Accuracy, how often is the classifier correct?
     prediction=metrics.accuracy_score(ytest,y_pred)
     print('Accuracy:',prediction*100,'%')
```

```
Accuracy: 76.31578947368422 %
```

Accuracy of Naïve Bayes is 76.31%.

Logistic Regression

```
[ ]  #Fitting logistic regression
     from sklearn.linear_model import LogisticRegression
     classifier=LogisticRegression(random_state=0)
     classifier.fit(Xtrain,ytrain)
```

```
LogisticRegression(random_state=0)
```

```
[ ]  #prdeicting the test set results
     y_pred=classifier.predict(Xtest)
     y_pred
```

```
array(['Iris-versicolor', 'Iris-versicolor', 'Iris-setosa',
       'Iris-virginica', 'Iris-setosa', 'Iris-virginica', 'Iris-setosa',
       'Iris-virginica', 'Iris-virginica', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-virginica', 'Iris-versicolor',
       'Iris-virginica', 'Iris-versicolor', 'Iris-setosa',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',
       'Iris-virginica', 'Iris-setosa', 'Iris-setosa', 'Iris-virginica',
       'Iris-versicolor', 'Iris-setosa', 'Iris-virginica',
       'Iris-versicolor', 'Iris-setosa', 'Iris-virginica',
       'Iris-virginica', 'Iris-versicolor', 'Iris-setosa',
       'Iris-versicolor'], dtype=object)
```

```
[ ]  #making confusion matrix
     from sklearn.metrics import confusion_matrix
     cm=confusion_matrix(ytest,y_pred)
     cm
```

```
array([[13,  0,  0],
       [ 0, 11,  5],
       [ 0,  3,  6]])
```

```
[ ]  from sklearn import metrics #import scikit learn metrics module for accuracy calculation
     #Model Accuracy, how often is the classifier correct?
     prediction=metrics.accuracy_score(ytest,y_pred)
     print('Accuracy:',prediction*100,'%')
```

```
Accuracy: 78.94736842105263 %
```

Accuracy of Logistic Regression (without PCA) is 78.94%.

Logistic Regression without PCA

With PCA

```
[ ]  X=dataset.iloc[:,[0,1,2,3]].values
     y=dataset.iloc[:,4].values
```

```
[ ]  #Splitting the dataset into the Training set and Test set
     from sklearn.model_selection import train_test_split
     Xtrain,Xtest,ytrain,ytest=train_test_split(X,y,test_size=0.25,random_state=0)
```

```
[ ]  #Feature Scaling
     from sklearn.preprocessing import StandardScaler
     sc=StandardScaler()
     Xtrain=sc.fit_transform(Xtrain)
     Xtest=sc.transform(Xtest)
     #Xtrain
```

```
[ ]  from sklearn.preprocessing import LabelEncoder
     labelencoder_y=LabelEncoder()
     y=labelencoder_y.fit_transform(y)
     print(y)
```

```
[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 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 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 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 2]
```

```
[ ]  # apply the pca
     from sklearn.decomposition import PCA
     pca=PCA(n_components=2)
     Xtrain=pca.fit_transform(Xtrain)
     Xtest=pca.transform(Xtest)
     explained_variance=pca.explained_variance_ratio_
     explained_variance
```

```
array([0.72212663, 0.24133062])
```

```
[ ]  #Fitting logistic regression
     from sklearn.linear_model import LogisticRegression
     classifier=LogisticRegression(random_state=0)
     classifier.fit(Xtrain,ytrain)
```

```
LogisticRegression(random_state=0)
```

```
[ ]  #making confusion matrix
     from sklearn.metrics import confusion_matrix
     cm=confusion_matrix(ytest,y_pred)
     cm
```

```
array([[13,  0,  0],
       [ 0, 12,  4],
       [ 0,  2,  7]])
```

```
[ ]  from sklearn import metrics #import scikit learn metrics module for accuracy calculation
     #Model Accuracy, how often is the classifier correct?
     prediction=metrics.accuracy_score(ytest,y_pred)
     print('Accuracy:',prediction*100,'%')
```

```
Accuracy: 84.21052631578947 %
```

Accuracy of Logistic Regression (with PCA) is 84.21%.

Random Forest

```
# Import the model we are using
from sklearn.ensemble import RandomForestClassifier
# Instantiate model with 1000 decision trees
rf = RandomForestClassifier(n_estimators = 100,max_depth=2,random_state = 0)
# Train the model on training data
rf.fit(Xtrain,ytrain)
```

```
RandomForestClassifier(max_depth=2, random_state=0)
```

```
[ ]  #prdeicting the test set results
     y_pred=classifier.predict(Xtest)
     y_pred
```

```
array(['Iris-virginica', 'Iris-versicolor', 'Iris-setosa',
       'Iris-virginica', 'Iris-setosa', 'Iris-virginica', 'Iris-setosa',
       'Iris-virginica', 'Iris-virginica', 'Iris-versicolor',
       'Iris-virginica', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-virginica', 'Iris-versicolor', 'Iris-setosa',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',
       'Iris-virginica', 'Iris-setosa', 'Iris-setosa', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-setosa', 'Iris-virginica',
       'Iris-versicolor', 'Iris-setosa', 'Iris-versicolor',
       'Iris-virginica', 'Iris-versicolor', 'Iris-setosa',
       'Iris-virginica'], dtype=object)
```

```
[ ]  #making confusion matrix
     from sklearn.metrics import confusion_matrix
     cm=confusion_matrix(ytest,y_pred)
     cm
```

```
array([[13,  0,  0],
       [ 0, 12,  4],
       [ 0,  2,  7]])
```

```
[ ]  from sklearn import metrics #import scikit learn metrics module for accuracy calculation
     #Model Accuracy, how often is the classifier correct?
     prediction=metrics.accuracy_score(ytest,y_pred)
     print('Accuracy:',prediction*100,'%')
```

```
Accuracy: 84.21052631578947 %
```

Accuracy of Random Forest is 84.21%.

AdaBoost

AdaBoost

```
[ ]  from sklearn.ensemble import AdaBoostClassifier
     clf = AdaBoostClassifier(n_estimators=100, random_state=0)
     clf.fit(X, y)
```

```
AdaBoostClassifier(n_estimators=100, random_state=0)
```

```
[ ]  #prdeicting the test set results
     y_pred=classifier.predict(Xtest)
     y_pred
```

```
array(['Iris-virginica', 'Iris-versicolor', 'Iris-setosa',
       'Iris-virginica', 'Iris-setosa', 'Iris-virginica', 'Iris-setosa',
       'Iris-virginica', 'Iris-virginica', 'Iris-versicolor',
       'Iris-virginica', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-virginica', 'Iris-versicolor', 'Iris-setosa',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',
       'Iris-virginica', 'Iris-setosa', 'Iris-setosa', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-setosa', 'Iris-virginica',
       'Iris-versicolor', 'Iris-setosa', 'Iris-versicolor',
       'Iris-virginica', 'Iris-versicolor', 'Iris-setosa',
       'Iris-virginica'], dtype=object)
```

```
[ ]  #making confusion matrix
     from sklearn.metrics import confusion_matrix
     cm=confusion_matrix(ytest,y_pred)
     cm
```

```
array([[13,  0,  0],
       [ 0, 12,  4],
       [ 0,  2,  7]])
```

```
[ ]  from sklearn import metrics #import scikit learn metrics module for accuracy calculation
     #Model Accuracy, how often is the classifier correct?
     prediction=metrics.accuracy_score(ytest,y_pred)
     print('Accuracy:',prediction*100,'%')
```

```
Accuracy: 84.21052631578947 %
```

Accuracy of AdaBoost is 84.21%.

SVM

SVM

```
[ ]  from sklearn.svm import LinearSVC
     from sklearn.pipeline import make_pipeline
     clf = make_pipeline(StandardScaler(),LinearSVC(random_state=0, tol=1e-5))
     clf.fit(X, y)
```

```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('linearsvc', LinearSVC(random_state=0, tol=1e-05))])
```

```
[ ]  #prdeicting the test set results
     y_pred=classifier.predict(Xtest)
     y_pred
```

```
array(['Iris-virginica', 'Iris-versicolor', 'Iris-setosa',
       'Iris-virginica', 'Iris-setosa', 'Iris-virginica', 'Iris-setosa',
       'Iris-virginica', 'Iris-virginica', 'Iris-versicolor',
       'Iris-virginica', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-virginica', 'Iris-versicolor', 'Iris-setosa',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',
       'Iris-virginica', 'Iris-setosa', 'Iris-setosa', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-setosa', 'Iris-virginica',
       'Iris-versicolor', 'Iris-setosa', 'Iris-versicolor',
       'Iris-virginica', 'Iris-versicolor', 'Iris-setosa',
       'Iris-virginica'], dtype=object)
```

```
[ ]  #making confusion matrix
     from sklearn.metrics import confusion_matrix
     cm=confusion_matrix(ytest,y_pred)
     cm
```

```
array([[13,  0,  0],
       [ 0, 12,  4],
       [ 0,  2,  7]])
```

```
[ ]  from sklearn import metrics #import scikit learn metrics module for accuracy calculation
     #Model Accuracy, how often is the classifier correct?
     prediction=metrics.accuracy_score(ytest,y_pred)
     print('Accuracy:',prediction*100,'%')
```

```
Accuracy: 84.21052631578947 %
```

Accuracy of SVM is 84.21%.