MALNAD COLLEGE OF ENGINEERING

(AN AUTONOMOUS INSTITUTION UNDER VTU, BELAGAVI)

HASSAN, KARNATAKA 573202, INDIA

Department of Computer Science and Engineering

MACHINE LEARNING LAB MANUAL

19CS704

MACHINE LEARNING LABORATORY

Course code:CS704 **LTPC:** (0-0-3)1.5

Exam Hours : 3 **Hours / Week :** 3

SEE : 50 Marks **Total hours :** 40

Course Objective: Apply machine learning algorithms to solve real world problems.

Course Outcomes (COs):

Upon Completion of the course, students shall be able to:

COs	Statements	POs
1.	Design and Implement ML concepts and algorithms	PO2,PO3,PO4
2.	Documentation and implementation of ML programs.	PO10

SOFTWARE REQUIREMENTS

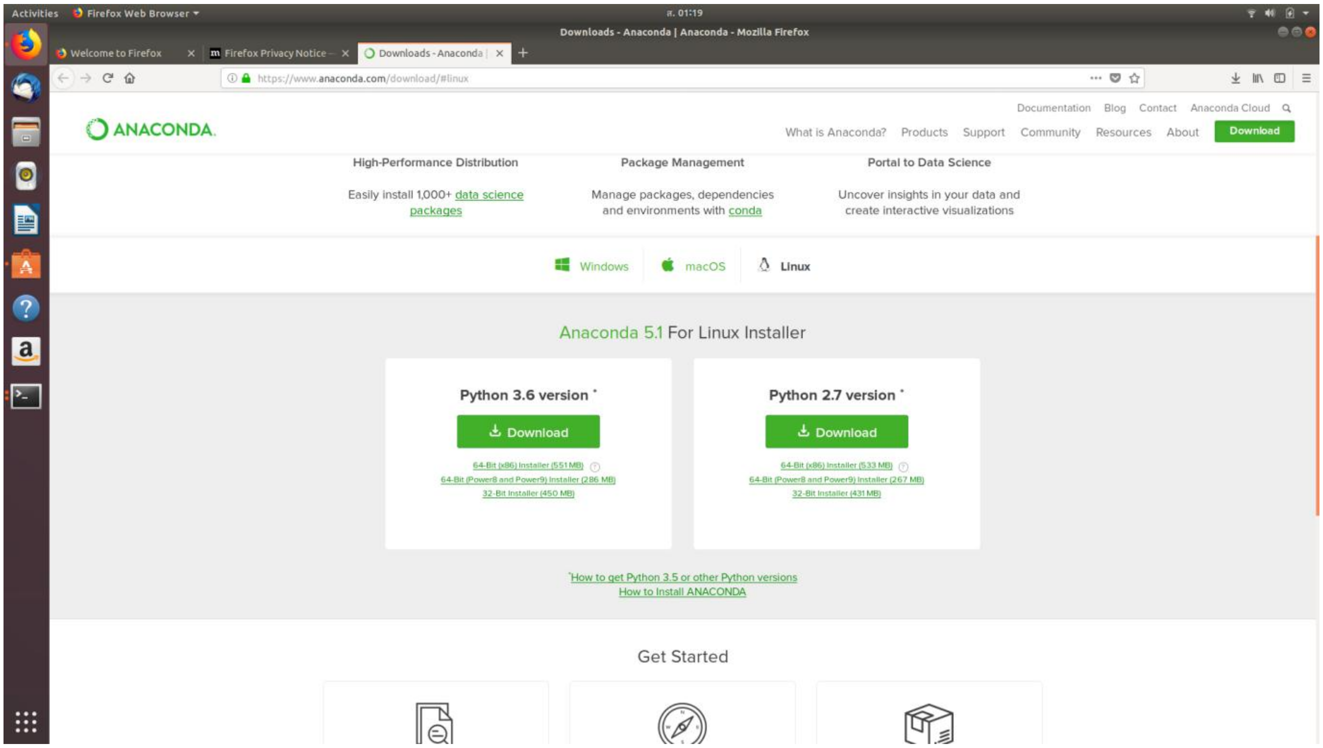
Operating System: Ubuntu

Open Source Software: Anaconda 3

Package Required for Anaconda: Jupyter

Steps to install Anaconda on Ubuntu

Step 1: Go to https://www.anaconda.com/download/ and pick your package distributions (Windows, Linux, MacOS)



jitsejan@jjsvps:~$ **cd Downloads/**

jitsejan@jjsvps:~/Downloads$ **wget https://repo.continuum.io/archive/Anaconda2-4.1.1-Linux-x86_64.sh**

Step 2: Run the installer.

jitsejan@jjsvps:~/Downloads$ **bash Anaconda2-4.1.1-Linux-x86_64.sh**

Step 3: Update the terminal to include the Anaconda references.

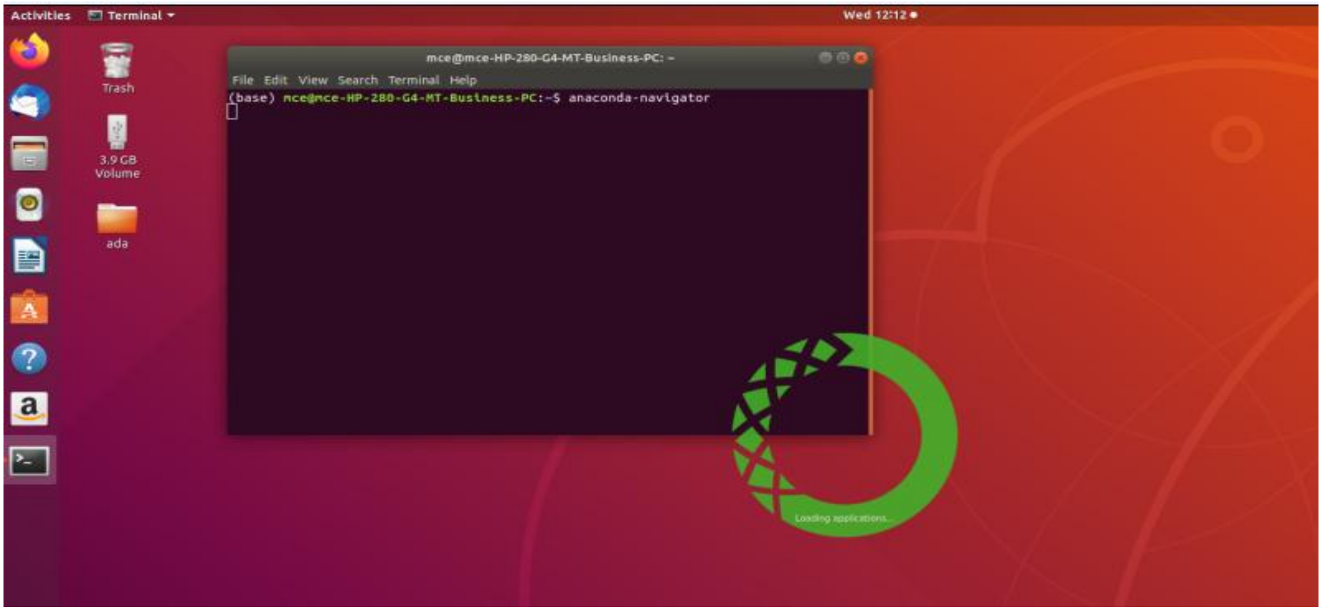
jitsejan@jjsvps:~/Downloads$ **source ~/.bashrc**

Step 4: Test if iPython is working now.

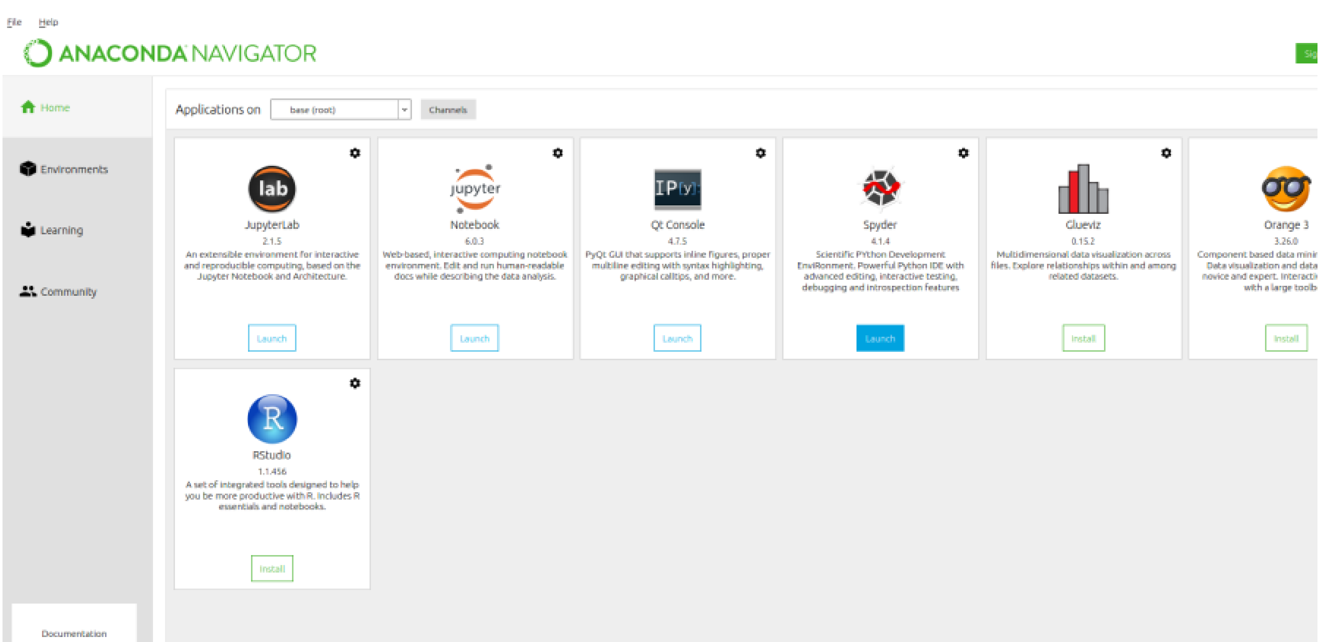
jitsejan@jjsvps:~$ **ipython -v**

All set.

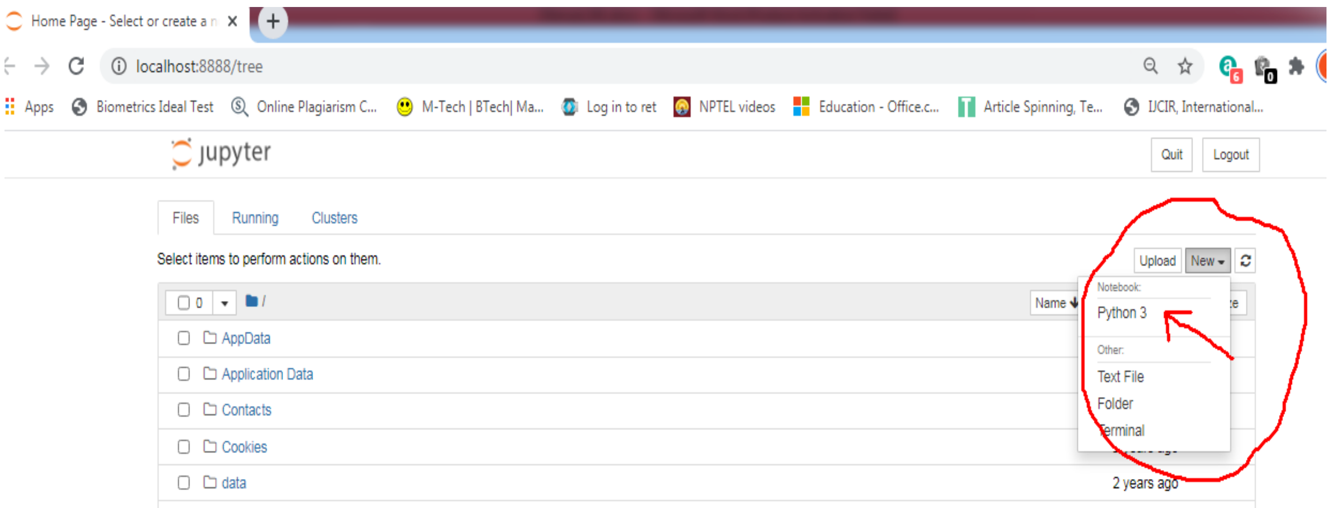
After anaconda installation open terminal and run anaconda-navigator



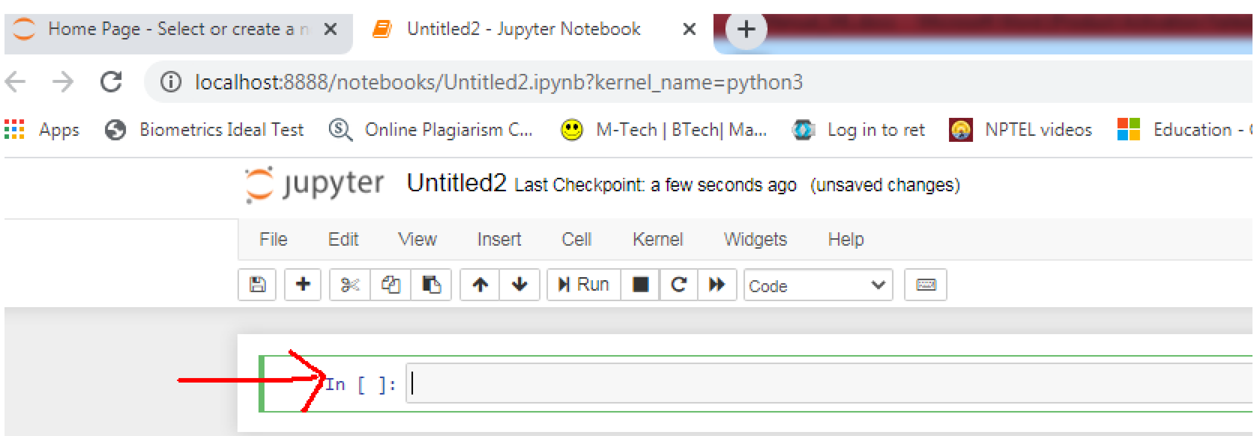
Launch jupyter to execute python program



After lanunching jupyter window choose new-Python3 option



New file will open , type the program inside In[]: terminal.



To run the program, Shif+ Enter

jupyter Untitled2 Last Checkpoint: 2 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Run Code

```
In [ ]: #Implementing Decision Tree
        #numpy and pandas initialization

        import numpy as np
        import pandas as pd

        #Loading the PlayTennis data

        PlayTennis = pd.read_csv("PlayTennis.csv")
        PlayTennis

        #It is easy to implement Decision Tree with numerical values.
        #We can convert all the non numerical values into numerical value

        from sklearn.preprocessing import LabelEncoder
        Le = LabelEncoder()
```

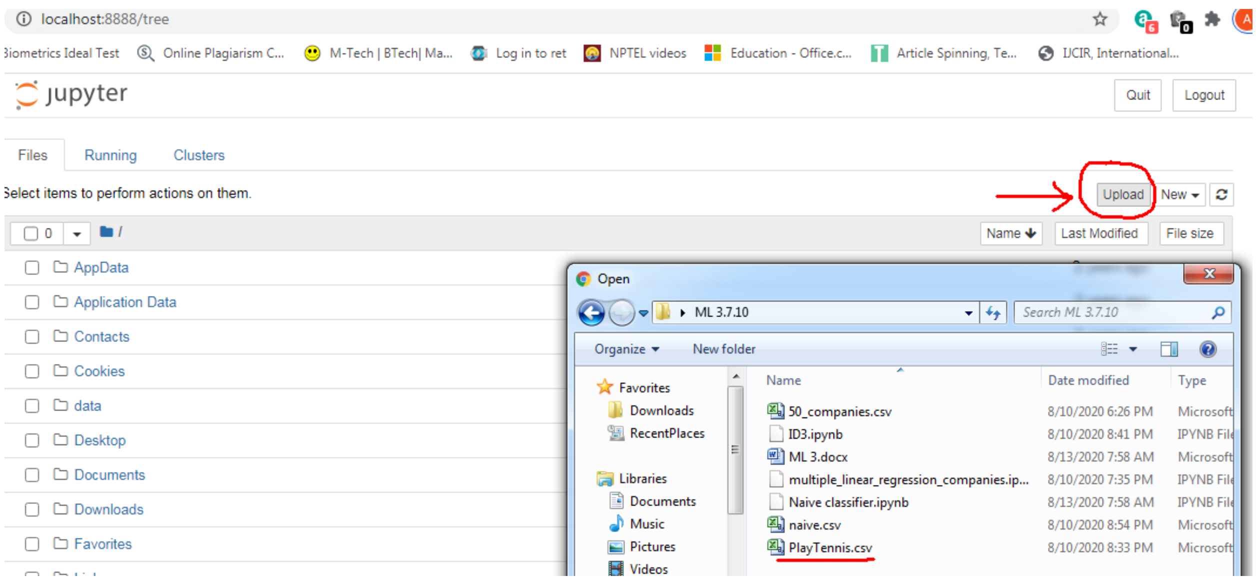
Output will be displayed in next line in[].

```
#X[0] -> Outlook
#X[1] -> Temperature
#X[2] -> Humidity
#X[3] -> Wind

# The predictions are stored in X_pred
X_pred = clf.predict(X)
# verifying if the model has predicted it all right.
X_pred == y
```

```
Out[1]: 0    True
        1    True
        2    True
        3    True
        4    True
        5    True
        6    True
        7    True
        8    True
```

O/P

For programs having .CSV as input, before running upload .CSV to jupyter source in Home page.

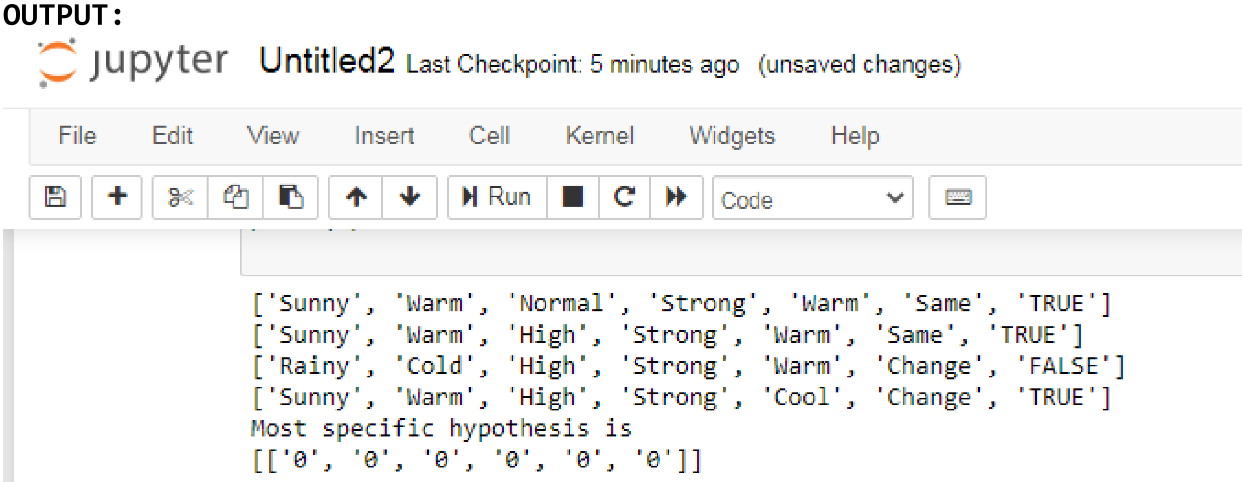


EXERCISE PROGRAMS

1. Implement and demonstrate the **FIND-S algorithm** for finding the most specific hypothesis based on a given set of training data samples. Read the training data from av.CSV file.

```
import csv

with open('lab1a.csv', 'r') as f:

    reader = csv.reader(f)
    your_list = list(reader)

h = [['0', '0', '0', '0', '0', '0']]

for i in your_list:
    print(i)
    if i[-1] == "True":
        j = 0
        for x in i:
            if x != "True":
                if x != h[0][j] and h[0][j] == '0':
                    h[0][j] = x
                elif x != h[0][j] and h[0][j] != '0':
                    h[0][j] = '?'
                else:
                    pass
                j = j + 1
print("Most specific hypothesis is")
print(h)
```

OUTPUT:

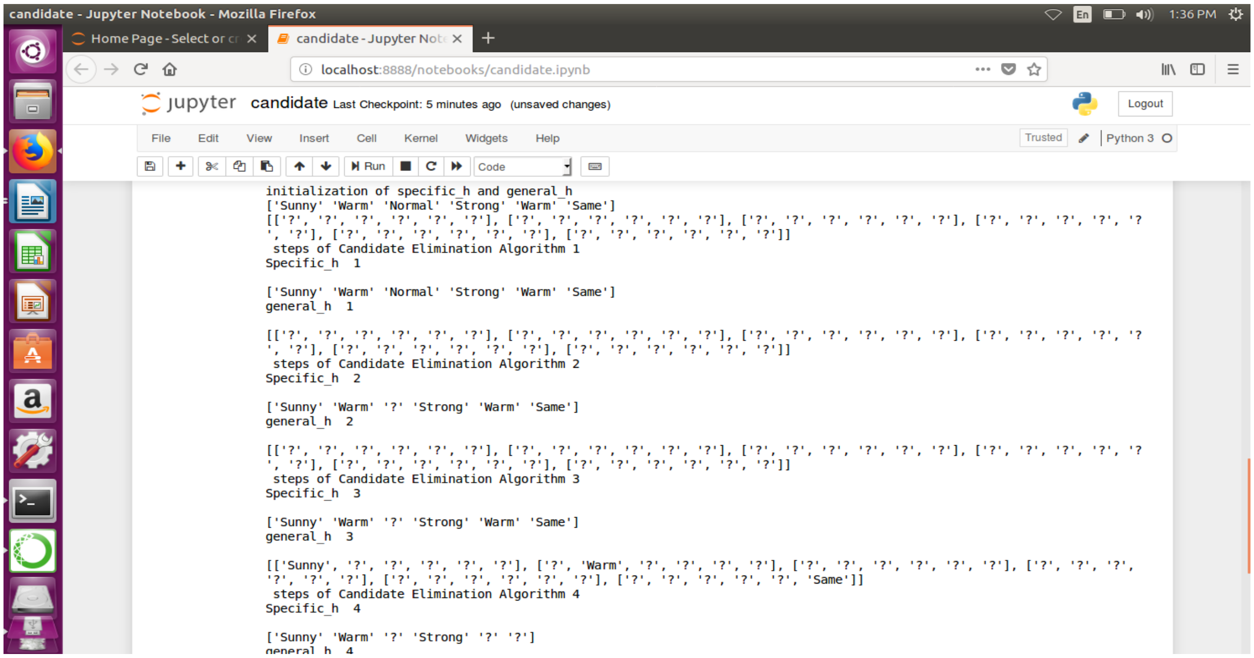


2. For a given set of training data examples stored in a .CSV file, implement and demonstrate the **Candidate-Elimination algorithm** to output a description of the set of all hypotheses consistent with the training examples.

```
import numpy as np
import pandas as pd

# Loading Data from a CSV File
data = pd.DataFrame(data=pd.read_csv('2b.csv'))


# Separating concept features from Target
concepts = np.array(data.iloc[:,0:-1])

# Isolating target into a separate DataFrame
#copying last column to target array
target = np.array(data.iloc[:,-1])


def learn(concepts, target):

    #learn() function implements the learning method of the Candidate
elimination algorithm.

    #Arguments:
    #concepts - a data frame with all the features
    #target - a data frame with corresponding output values


    # Initialise S0 with the first instance from concepts
    # .copy() makes sure a new list is created instead of just
pointing to the same memory location
    specific_h = concepts[0].copy()
    print("initialization of specific_h and general_h")
    print(specific_h)

    general_h = [["?" for i in range(len(specific_h))] for i in
range(len(specific_h))]
    print(general_h)
    # The learning iterations
    for i, h in enumerate(concepts):

        # Checking if the hypothesis has a positive target
        if target[i] == "Yes":
            for x in range(len(specific_h)):
```

```
                # Change values in S & G only if values change
                if h[x] != specific_h[x]:
                    specific_h[x] = '?'
                    general_h[x][x] = '?'

        # Checking if the hypothesis has a positive target
        if target[i] == "No":
            for x in range(len(specific_h)):

                # For negative hyposthesis change values only  in G
                if h[x] != specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'
        print(" steps of Candidate Elimination Algorithm",i+1)
        print("Specific_h ",i+1,"\n ")
        print(specific_h)
        print("general_h ", i+1, "\n ")
        print(general_h)
    # find indices where we have empty rows, meaning those that are
unchanged
    indices = [i for i, val in enumerate(general_h) if val == ['?',
'?', '?', '?', '?', '?']]
    for i in indices:
        # remove those rows from general_h
        general_h.remove(['?', '?', '?', '?', '?', '?'])

    # Return final values
    return specific_h, general_h
s_final, g_final = learn(concepts, target)
print("Final Specific_h:", s_final, sep="\n")
print("Final General_h:", g_final, sep="\n")
```

OUTPUT:



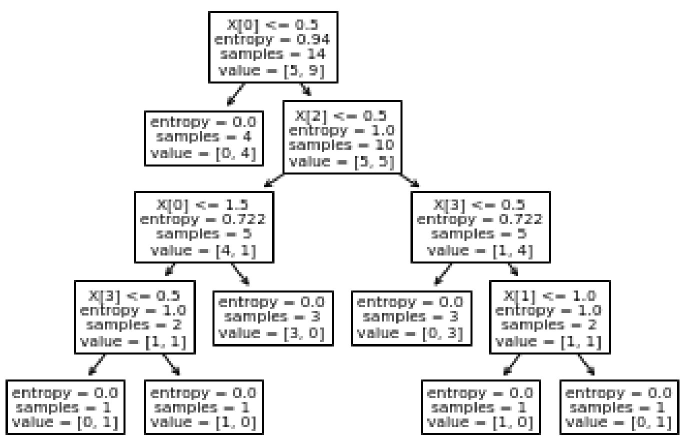
3. Write a program to demonstrate the working of the decision tree based **ID3 algorithm**. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

```
#Implementing Decision Tree
#numpy and pandas initialization

import numpy as np
import pandas as pd

#Loading the PlayTennis data

PlayTennis = pd.read_csv("PlayTennis.csv")
PlayTennis

#It is easy to implement Decision Tree with numerical values.
#We can convert all the non numerical values into numerical values
using LabelEncoder

from sklearn.preprocessing import LabelEncoder
Le = LabelEncoder()
```

```
PlayTennis['outlook'] = Le.fit_transform(PlayTennis['outlook'])
PlayTennis['temp'] = Le.fit_transform(PlayTennis['temp'])
PlayTennis['humidity'] = Le.fit_transform(PlayTennis['humidity'])
PlayTennis['windy'] = Le.fit_transform(PlayTennis['windy'])
PlayTennis['play'] = Le.fit_transform(PlayTennis['play'])

PlayTennis

#Lets split the training data and its coresponding prediction values.
#y - holds all the decisions.
#X - holds the training data.

y = PlayTennis['play']
X = PlayTennis.drop(['play'],axis=1)

# Fitting the model
from sklearn import tree
clf = tree.DecisionTreeClassifier(criterion = 'entropy')
clf = clf.fit(X, y)

# We can visualize the tree using tree.plot_tree
tree.plot_tree(clf)

#In Graph
#X[0] -> Outlook
#X[1] -> Temperature
#X[2] -> Humidity
#X[3] -> Wind

# The predictions are stored in X_pred
X_pred = clf.predict(X)
# verifying if the model has predicted it all right.
X_pred == y
```

OUTPUT:


```
Out[1]: 0     True
        1     True
        2     True
        3     True
        4     True
        5     True
        6     True
        7     True
        8     True
        9     True
        10    True
        11    True
        12    True
        13    True
        Name: play, dtype: bool
```

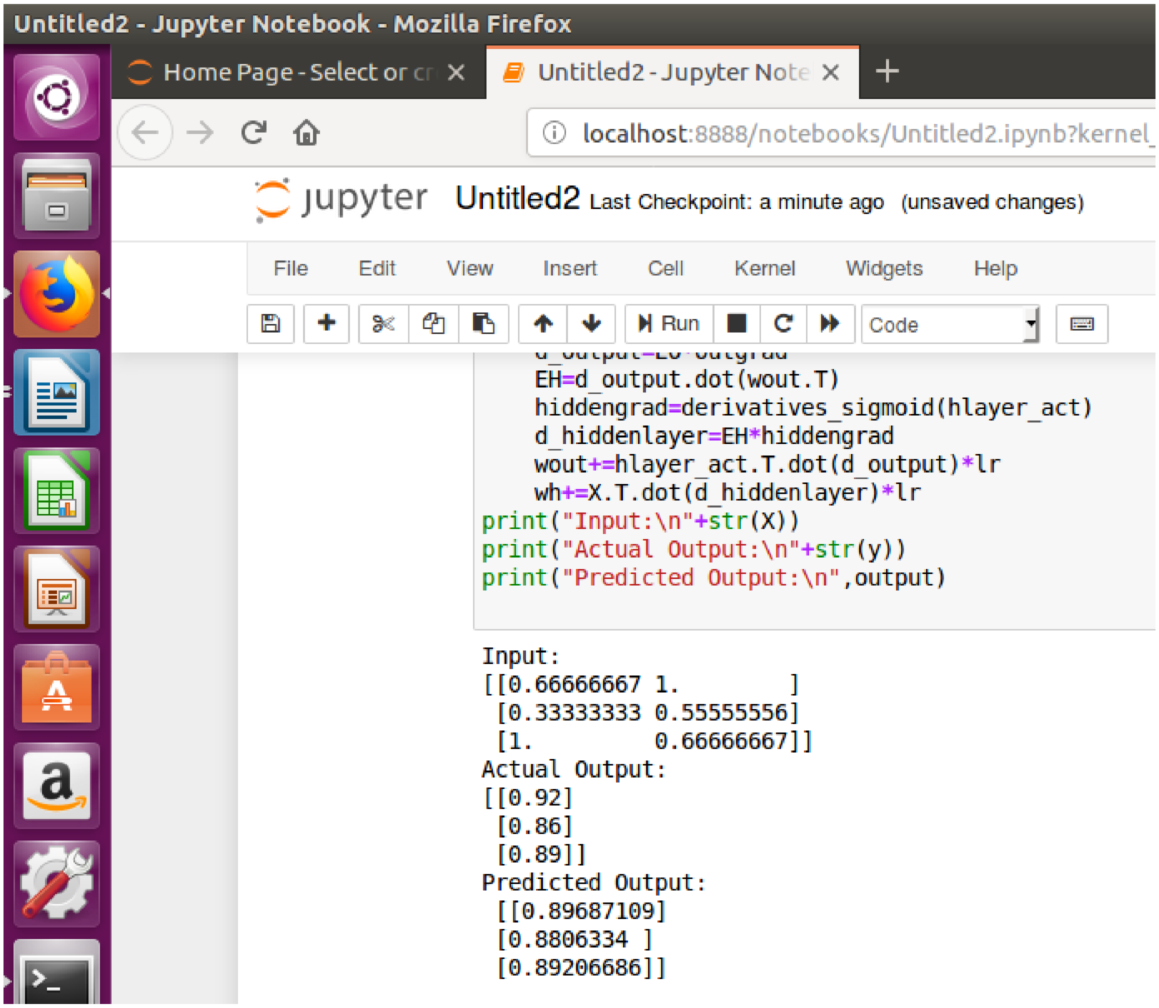


4. Build an Artificial Neural Network by implementing the **Backpropagation algorithm** and test the same using appropriate data sets.

```
import numpy as np
import matplotlib as m
X=np.array(([2,9],[1,5],[3,6]),dtype=float)
y=np.array(([92],[86],[89]),dtype=float)
X=X/np.amax(X,axis=0)
```

```
y=y/100

def sigmoid(x):
    return 1/(1+np.exp(-x))

def derivatives_sigmoid(x):
    return x*(1-x)

epoch=7000
lr=0.1
inputlayer_neurons=2
hiddenlayer_neurons=3
output_neurons=1

wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))

for i in range(epoch):
    hinp1=np.dot(X,wh)
    hinp=hinp1+bh
    hlayer_act=sigmoid(hinp)
    outinp1=np.dot(hlayer_act,wout)
    outinp=outinp1+bout
    output=sigmoid(outinp)

    EO=y-output
    outgrad=derivatives_sigmoid(output)
    d_output=EO*outgrad
    EH=d_output.dot(wout.T)
    hiddengrad=derivatives_sigmoid(hlayer_act)
    d_hiddenlayer=EH*hiddengrad
    wout+=hlayer_act.T.dot(d_output)*lr
    wh+=X.T.dot(d_hiddenlayer)*lr
print("Input:\n"+str(X))
print("Actual Output:\n"+str(y))
print("Predicted Output:\n",output)
```

OUTPUT :

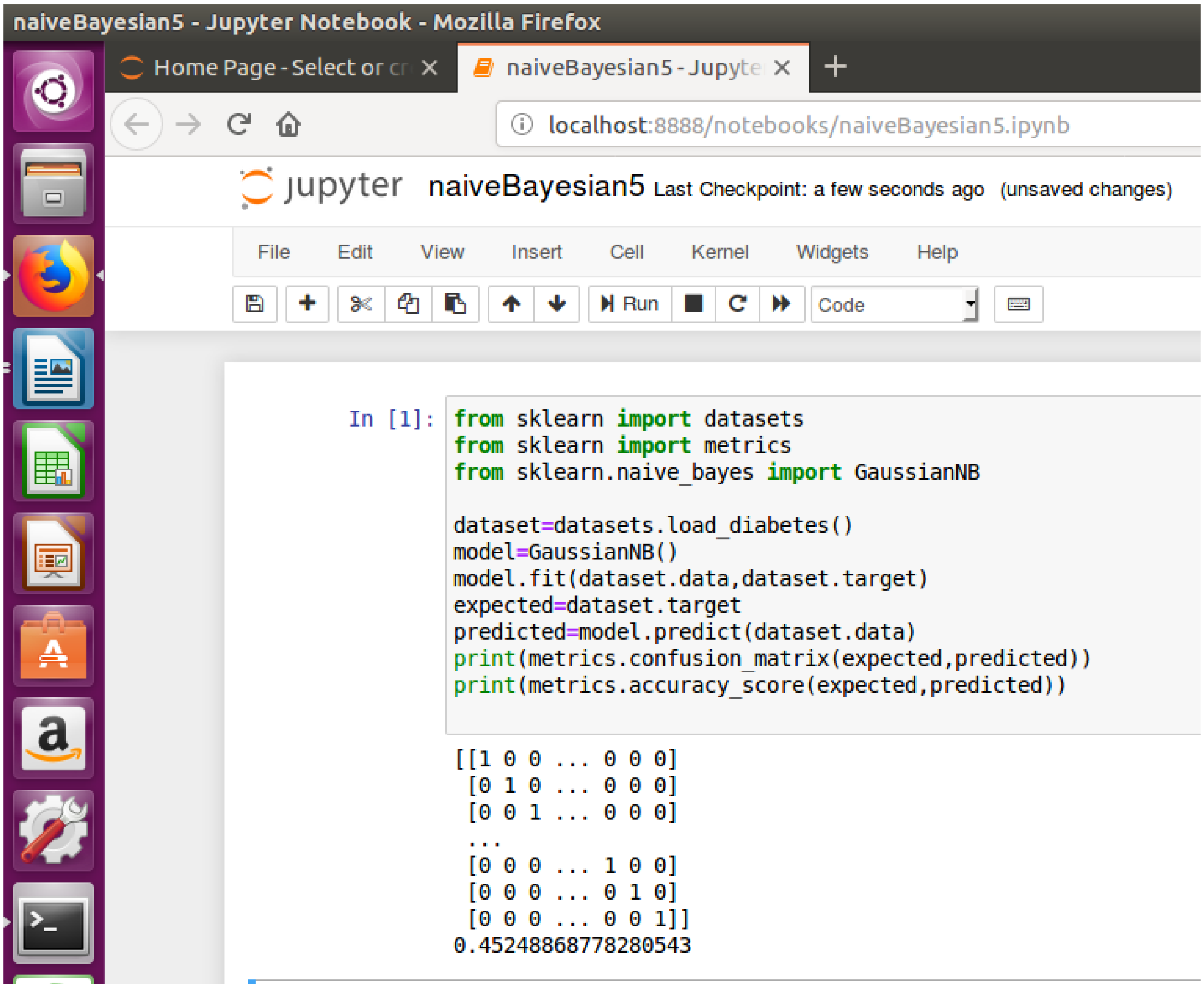


5. Write a program to implement the **naïve Bayesian classifier** for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

```
from sklearn import datasets
from sklearn import metrics
from sklearn.naive_bayes import GaussianNB

dataset=datasets.load_diabetes()
model=GaussianNB()
model.fit(dataset.data,dataset.target)
expected=dataset.target
predicted=model.predict(dataset.data)
```

```
print(metrics.confusion_matrix(expected,predicted))
print(metrics.accuracy_score(expected,predicted))
```

OUTPUT:

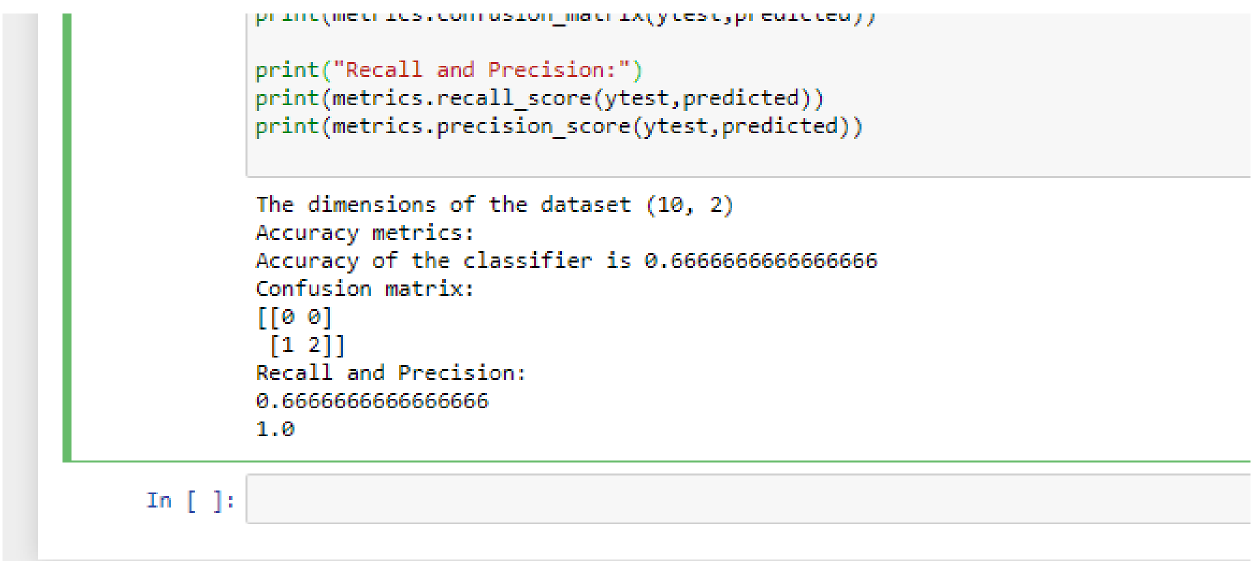


6. Assuming a set of documents that need to be classified, use the **naïve Bayesian Classifier** model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics
```

```
msg=pd.read_csv('naive.csv',header=None,names=['message','label'])
print("The dimensions of the dataset",msg.shape)
msg['labelnum']=msg.label.map({'pos':1,'neg':0})
x=msg.message
y=msg.labelnum

xtrain,xtest,ytrain,ytest=train_test_split(x,y,random_state=1)

count_vect=CountVectorizer()
xtrain_dtm=count_vect.fit_transform(xtrain)
xtest_dtm=count_vect.transform(xtest)

clf=MultinomialNB().fit(xtrain_dtm,ytrain)
predicted=clf.predict(xtest_dtm)

print("Accuracy metrics:")
print("Accuracy of the classifier
is",metrics.accuracy_score(ytest,predicted))

print("Confusion matrix:")
print(metrics.confusion_matrix(ytest,predicted))

print("Recall and Precision:")
print(metrics.recall_score(ytest,predicted))
print(metrics.precision_score(ytest,predicted))
```

OUTPUT:



```
print("Recall and Precision:")
print(metrics.recall_score(ytest,predicted))
print(metrics.precision_score(ytest,predicted))

The dimensions of the dataset (10, 2)
Accuracy metrics:
Accuracy of the classifier is 0.6666666666666666
Confusion matrix:
[[0 0]
 [1 2]]
Recall and Precision:
0.6666666666666666
1.0

In [ ]:
```

7. Write a program to construct a **Bayesian network** considering medical data. Use this model to demonstrate the diagnosis of lungs patients using standard lungs Disease Data Set. You can use Java/Python ML library classes/API.

```
from pomegranate import*
asia= DiscreteDistribution({'True':0.5,'False':0.5})
tuberculosis=ConditionalProbabilityTable(
        [['True','True',0.2],
         ['True','False',0.8],
         ['False','True',0.01],
         ['False','False',0.99]],[asia])
smoking=DiscreteDistribution({'True':0.5,'False':0.5})
lung=ConditionalProbabilityTable(
        [['True','True',0.75],
         ['True','False',0.25],
         ['False','True',0.02],
         ['False','False',0.98]],[smoking])
bronchitis=ConditionalProbabilityTable(
        [['True','True',0.92],
         ['True','False',0.08],
         ['False','True',0.03],
         ['False','False',0.97]],[smoking])
tuberculosis_or_cancer=ConditionalProbabilityTable(
        [['True','True','True',1.0],
         ['True','True','False',0.0],
         ['True','False','True',1.0],
         ['True','False','False',0.0],
         ['False','True','True',1.0],
```

```
         ['False','True','False',0.0],
         ['False','False','True',0.0],
         ['False','False','False',1.0]],[tuberculosis,lung])
xray=ConditionalProbabilityTable(
        [['True','True',0.885],
         ['True','False',0.115],
         ['False','True',0.04],
         ['False','False',0.96]],[tuberculosis_or_cancer])
dyspnea=ConditionalProbabilityTable(
        [['True','True','True',0.96],
         ['True','True','False',0.04],
         ['True','False','True',0.89],
         ['True','False','False',0.11],
         ['False','True','True',0.96],
         ['False','True','False',0.04],
         ['False','Flase','True',0.89],

['False','False','False',0.11]],[tuberculosis_or_cancer,bronchitis])
s0=State(asia,name='asia')
s1=State(tuberculosis,name="tuberculosis")
s2=State(smoking,name="smoker")
network=BayesianNetwork("asia")
network.add_nodes(s0,s1,s2)
network.add_edge(s0,s1)
network.add_edge(s1,s2)
network.bake()
print(network.predict_proba({'tuberculosis':'True'}))
```

OUTPUT:

```
[{
    "class" :"Distribution",
    "dtype" :"str",
    "name" :"DiscreteDistribution",
    "parameters" :[
        {
            "True" :0.9523809523809521,
            "False" :0.047619047619047825
        }
    ],
    "frozen" :false
}
 'True'
 {
    "class" :"Distribution",
    "dtype" :"str",
    "name" :"DiscreteDistribution",
    "parameters" :[
        {
            "True" :0.5,
            "False" :0.5
        }
    ],
    "frozen" :false
}]
```

8. Apply **EM algorithm** to cluster a set of data stored in a .CSV file. Use the same data set for clustering using ***k*-Means algorithm**. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML

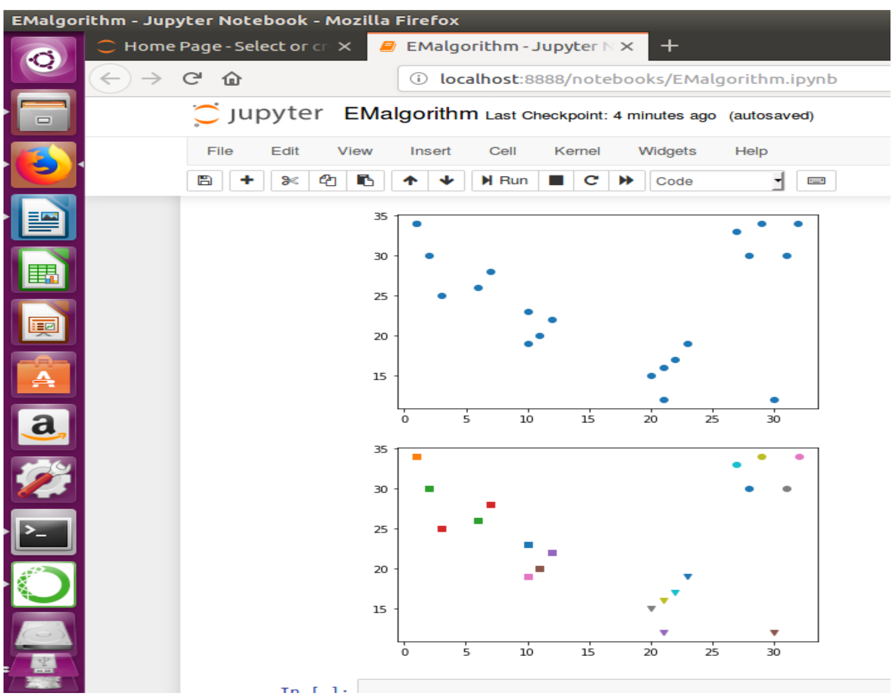
library classes/API in the program.

```
from sklearn.cluster import KMeans
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

data=pd.read_csv("a.csv")
x1=data['x'].values
x2=data['y'].values
print(data)
X=np.matrix(list(zip(x1,x2)))
plt.scatter(x1,x2)
```

```
plt.show()
markers=['s','o','v']
k=3
clusters=KMeans(n_clusters=k).fit(X)
for i,l in enumerate(clusters.labels_):
    plt.plot(x1[i],x2[i],marker=markers[l])
```

OUTPUT:

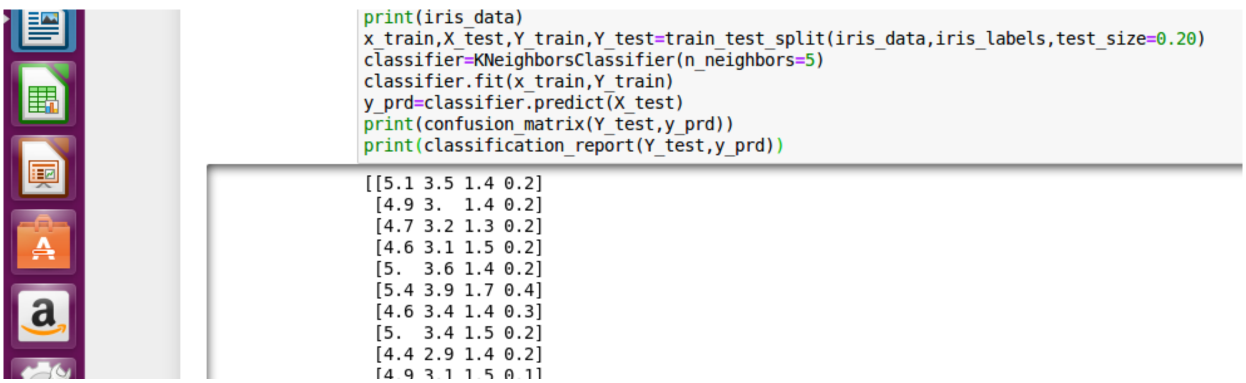
```
     x   y  Unnamed: 2  Unnamed: 3  Unnamed: 4  Unn
0   10  23         NaN         NaN         NaN
1    1  34         NaN         NaN         NaN
2    6  26         NaN         NaN         NaN
3    7  28         NaN         NaN         NaN
4   21  12         NaN         NaN         NaN
5   30  12         NaN         NaN         NaN
6   32  34         NaN         NaN         NaN
7   31  30         NaN         NaN         NaN
8   29  34         NaN         NaN         NaN
9   27  33         NaN         NaN         NaN
10  28  30         NaN         NaN         NaN
11   1  34         NaN         NaN         NaN
12   2  30         NaN         NaN         NaN
13   3  25         NaN         NaN         NaN
14  12  22         NaN         NaN         NaN
15  11  20         NaN         NaN         NaN
16  10  19         NaN         NaN         NaN
17  20  15         NaN         NaN         NaN
18  21  16         NaN         NaN         NaN
19  22  17         NaN         NaN         NaN
20  23  19         NaN         NaN         NaN
```

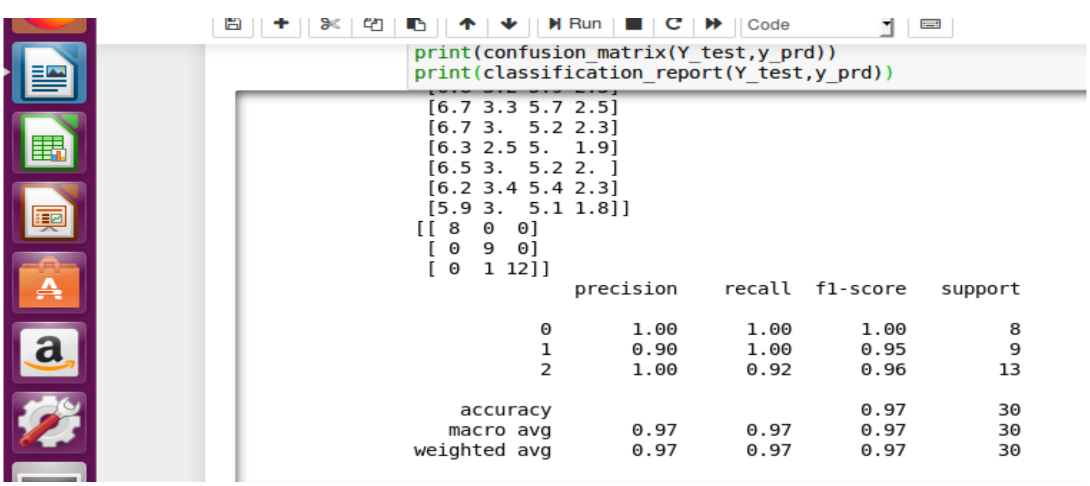


9. Write a program to implement ***k*-Nearest Neighbour algorithm** to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix,classification_report
from sklearn import datasets
iris=datasets.load_iris()
iris_data=iris.data
iris_labels=iris.target
print(iris_data)
x_train,X_test,Y_train,Y_test=train_test_split(iris_data,iris_labels,t
est_size=0.20)
classifier=KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train,Y_train)
y_prd=classifier.predict(X_test)
print(confusion_matrix(Y_test,y_prd))
print(classification_report(Y_test,y_prd))
```

OUTPUT:



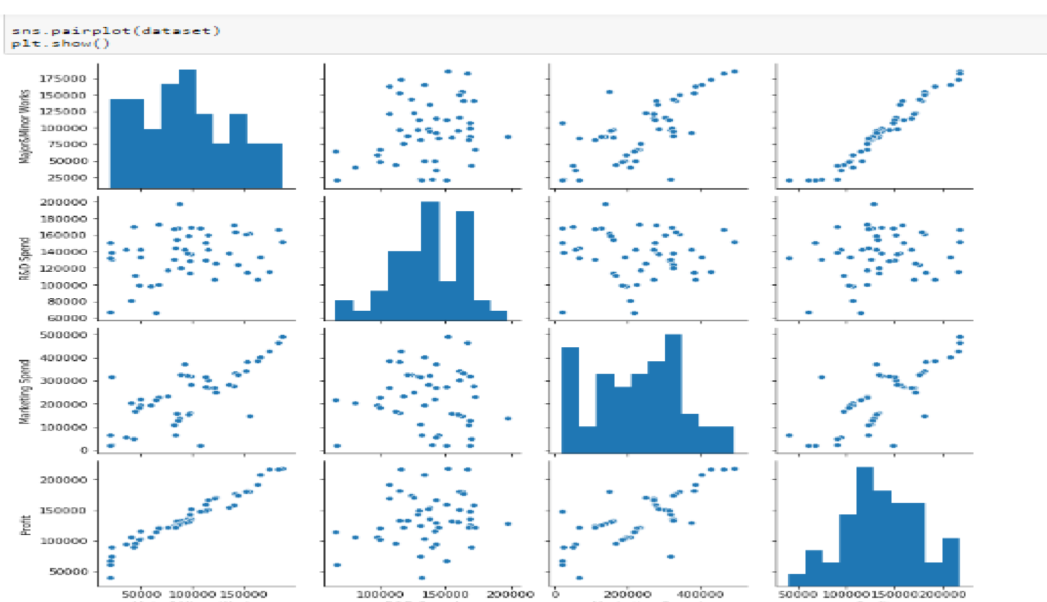


10. Implement **Multiple linear Regression algorithm** in order to fit data point Select the appropriate data set for your experiment and draw graph.

```
#multiple linear regression
#import the libraaries

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
#location of database
dataset = pd.read_csv('50_companies.csv')
dataset.head()
import matplotlib.pyplot as plt
import seaborn as sns
# Performing a correlation analysis on all the numerical variables

sns.pairplot(dataset)
plt.show()
#split the data into depenent and independent variable
X= dataset.iloc[:,:-1].values
Y=dataset.iloc[:,4].values
print(X)
# Encoding categorical data
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.compose import ColumnTransformer
labelencoder = LabelEncoder()
X[:, 3] = labelencoder.fit_transform(X[:, 3])
print(X)
#onehotencoder = OneHotEncoder(categories= 'auto' )
#onehotencoder = OneHotEncoder(categorical_features = [3])
#X = onehotencoder.fit_transform(X).toarray()

ct = ColumnTransformer([("Location", OneHotEncoder(), [3])], remainder
= 'passthrough')
X = ct.fit_transform(X)

print(X)
# Avoiding the Dummy Variable Trap
#X = X[:, 1:]
#print(X)
#splitting the dataset as training and testing dataset
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size =
0.2, random_state=0)
```

```
# Fitting Multiple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, Y_train)
# Predicting the Test set results
y_pred = regressor.predict(X_test)
#Actual Value
print(Y_test)
#Prediced Value
print(y_pred)

df = pd.DataFrame({'Actual': Y_test.flatten(), 'Predicted':
y_pred.flatten()})
df

df1 = df.head(10)
df1.plot(kind='bar',figsize=(16,10))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()

fig = plt.figure()
plt.scatter(Y_test, y_pred)
fig.suptitle('y_test vs y_pred', fontsize = 20)
plt.xlabel('y_test', fontsize = 18)
plt.ylabel('y_pred', fontsize = 16)
```

OUTPUT:



Out[29]:

	Actual	Predicted
0	128282.38	128015.201598
1	169259.40	157582.277608
2	171121.95	157447.738452
3	102798.83	96976.098513
4	216050.39	203537.482210
5	130008.31	141161.242302
6	106229.06	92851.692097
7	122483.56	123791.733747
8	135352.25	138969.435330
9	191187.94	192921.065695

Out[26]: Text(0, 0.5, 'y_pred')