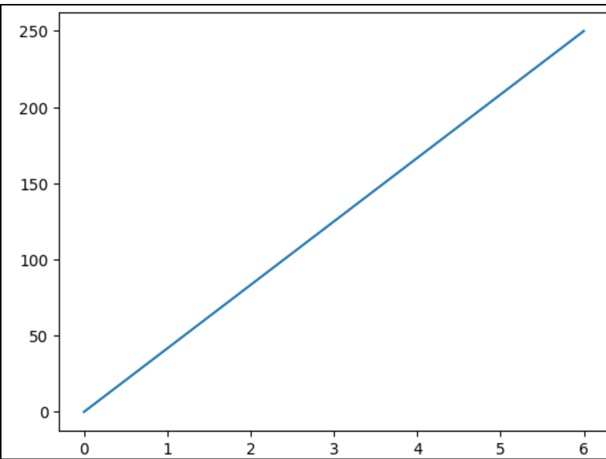
PROGRAMS

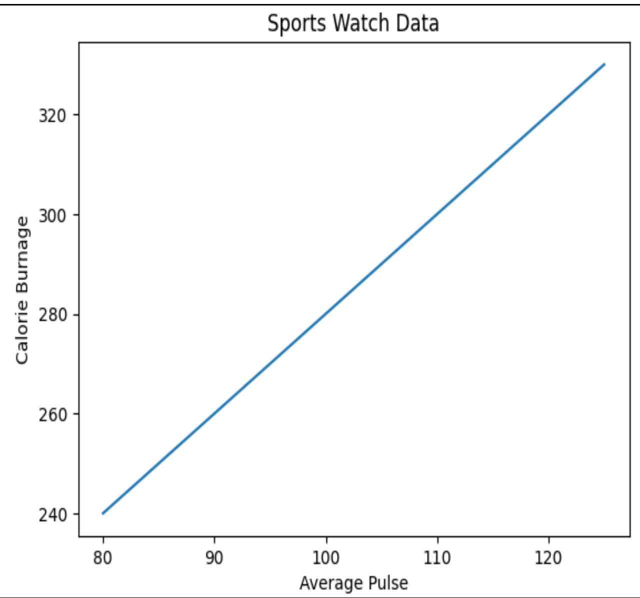
1. Demonstration of python Libraries for Machine Learning – pandas, Sklearn, numpy, matplotlib.

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

xpoints = np.array([0, 6])
ypoints = np.array([0, 250])

plt.plot(xpoints, ypoints)
plt.show()
```




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

xpoints = np.array([1, 2, 6, 8])
ypoints = np.array([3, 8, 1, 10])

plt.plot(xpoints, ypoints)
plt.show()
```

Machine Learning Laboratory

`import numpy as np`
`import matplotlib.pyplot as plt`
 	
`x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])`
`y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])`
 	
`plt.plot(x, y)`
 	
`plt.title("Sports Watch Data")`
`plt.xlabel("Average Pulse")`
`plt.ylabel("Calorie Burnage")`
 	
`plt.show()`
	

`import numpy as np`
 	
`a = np.array(42)`
`b = np.array([1, 2, 3, 4, 5])`
`c = np.array([[1, 2, 3], [4, 5, 6]])`
`d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])`
 	
`print(a.ndim)`
`print(b.ndim)`
`print(c.ndim)`
`print(d.ndim)`
	0
	1
	2
	3

2. Demonstration of Exploratory Data Analysis and Data Visualization.

<table>
<tr><td>

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.4, random_state=1)

print("X_train Shape:",  X_train.shape)
print("X_test Shape:", X_test.shape)
print("Y_train Shape:", y_train.shape)
print("Y_test Shape:", y_test.shape)
```

</td><td>X_train Shape: (90, 4)
X_test Shape: (60, 4)
Y_train Shape: (90,)
Y_test Shape: (60,)</td></tr>
<tr><td>

```
from sklearn.preprocessing import LabelEncoder

categorical_feature = ['cat', 'dog', 'dog', 'cat', 'bird']

encoder = LabelEncoder()

encoded_feature = encoder.fit_transform(categorical_feature)

print("Encoded feature:", encoded_feature)
```

</td><td>Encoded feature: [1 2 2 1 0]</td></tr>
</table>

3. Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesisbased 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:
                                        passj
                                = j + 1
print("Most specific hypothesis is")print(h)
```

OUTPUT:

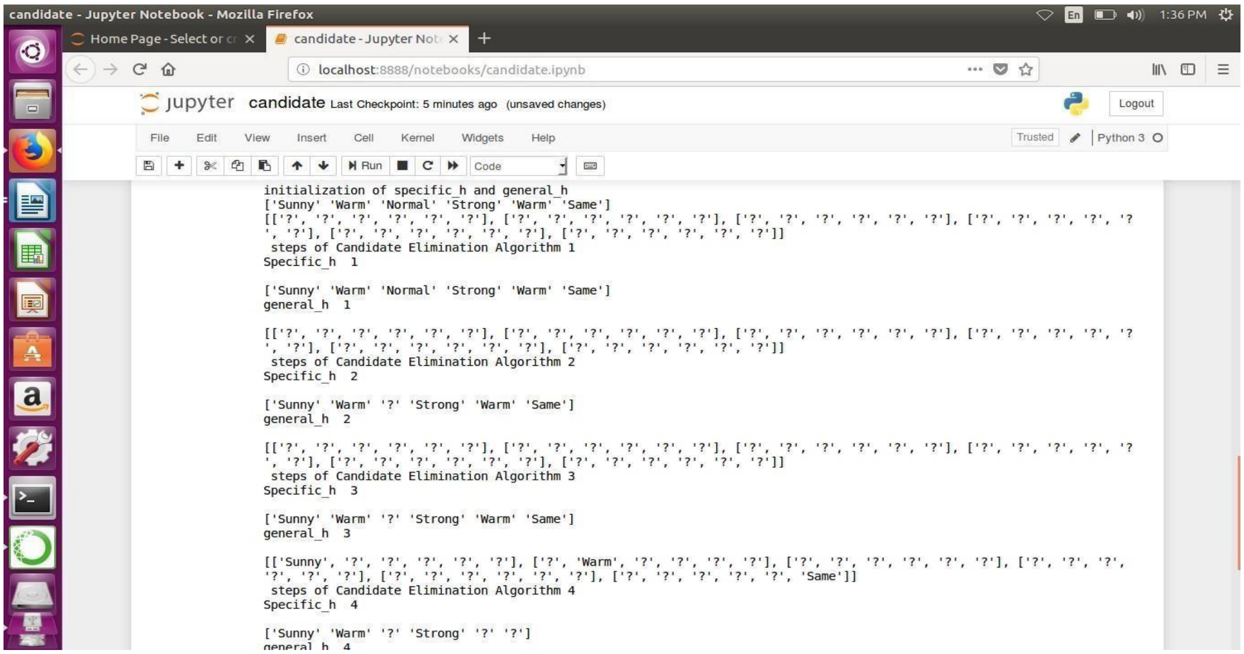
```
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'TRUE']
['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'TRUE']
['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'FALSE']
['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'TRUE']
Most specific hypothesis is
[['Sunny', 'Warm', '?', 'Strong', '?', '?']]
```

4. 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 Targetconcepts =
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 Candidateelimination 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 justpointing 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 inrange(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 changeif h[x] !=
                    specific_h[x]:
                        specific_h[x] = '?'
                        general_h[x][x] = '?'

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

                    # For negative hyposthesis change values only in Gif 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 areunchanged
        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")
```

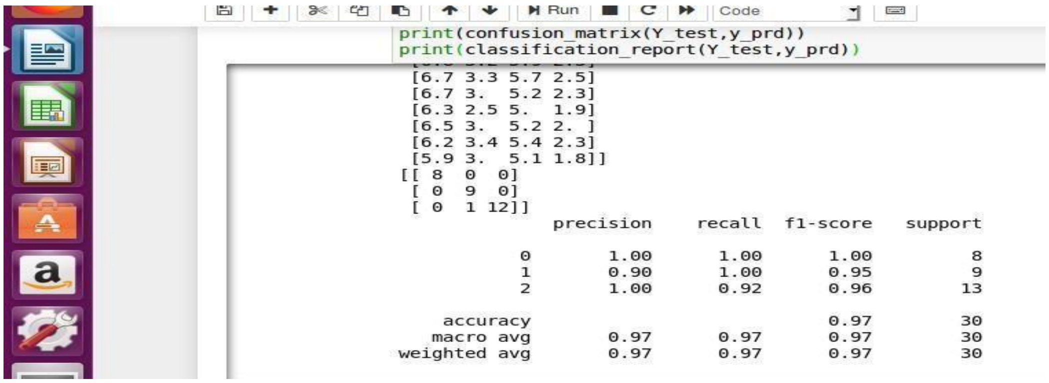
OUTPUT:


candidate - Jupyter Notebook - Mozilla Firefox
localhost:8888/notebooks/candidate.ipynb
jupyter candidate Last Checkpoint: 5 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted Python 3
Run Code
initialization of specific_h and general_h
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
steps of Candidate Elimination Algorithm 1
Specific_h 1
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
general_h 1
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
steps of Candidate Elimination Algorithm 2
Specific_h 2
['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
general_h 2
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
steps of Candidate Elimination Algorithm 3
Specific_h 3
['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
general_h 3
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]
steps of Candidate Elimination Algorithm 4
Specific_h 4
['Sunny' 'Warm' '?' 'Strong' '?' '?']

5. Write a program to implement *k*-Nearest Neighbour algorithm to classify the iris data set.Print both correct and wrong predictions.

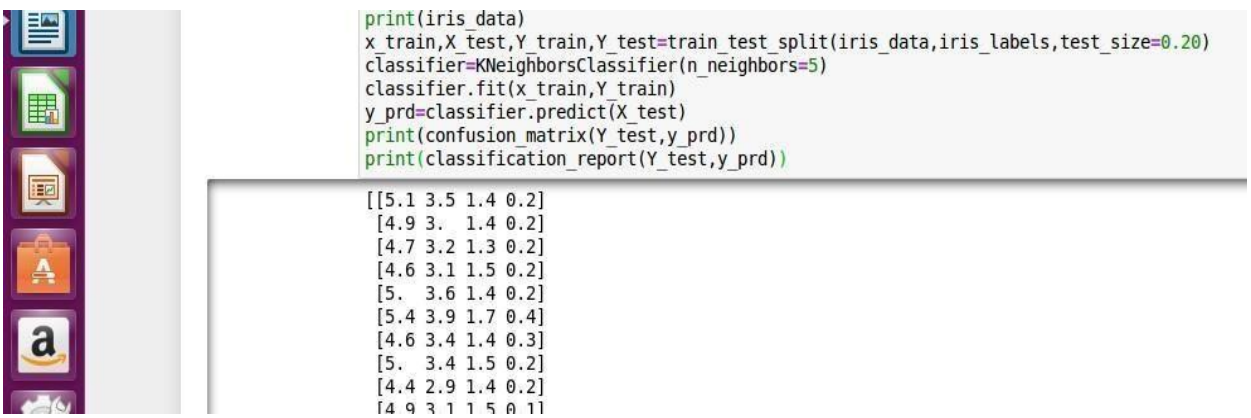
```
from sklearn.model_selection import train_test_splitfrom
sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix,classification_reportfrom sklearn import
datasets
iris=datasets.load_iris()
iris_data=iris.data
iris_labels=iris.target
print(iris_data)
x_train,X_test,Y_train,Y_test=train_test_split(iris_data,iris_labels,test_size=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:



```
print(confusion_matrix(Y_test,y_prd))
print(classification_report(Y_test,y_prd))
```

```
 [6.7 3.3 5.7 2.5]
 [6.7 3.  5.2 2.3]
 [6.3 2.5 5.  1.9]
 [6.5 3.  5.2 2. ]
 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]]
[[ 8  0  0]
 [ 0  9  0]
 [ 0  1 12]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         8
           1       0.90      1.00      0.95         9
           2       1.00      0.92      0.96        13

    accuracy                           0.97        30
   macro avg       0.97      0.97      0.97        30
weighted avg       0.97      0.97      0.97        30
```



```
print(iris_data)
x_train,X_test,Y_train,Y_test=train_test_split(iris_data,iris_labels,test_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))
```

```
[[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]
```

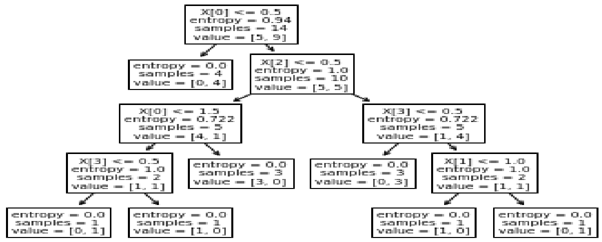
6. Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

```
#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 valuesusing LabelEncoder

from sklearn.preprocessing import LabelEncoderLe =
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_treetree.plot_tree(clf)

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

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

OUTPUT:

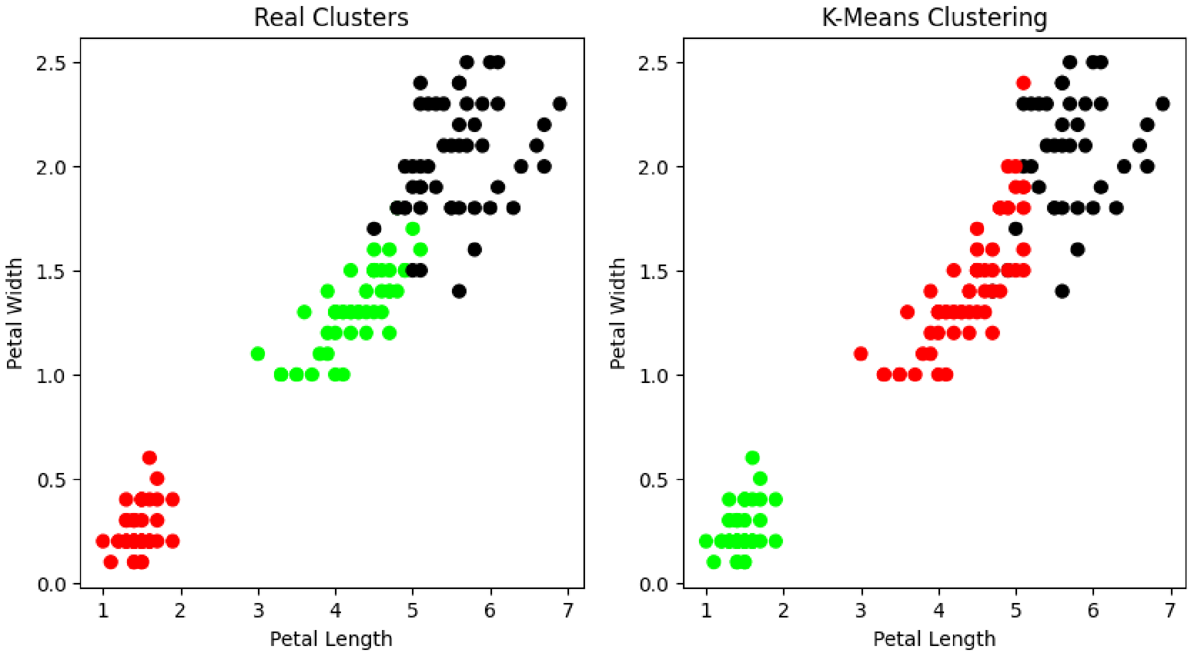
```
Out[1]: 0     True
        1     True
        2     True
        3     True
        4     True
        5     True
        6     True
        7     True
        8     True
        9     True
        10    True
        11    True
        12    True
        13    True
        Name: play, dtype: bool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



7. Implement a Clustering algorithm using K-means clustering for the dataset.

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

# import some data to play with
iris = datasets.load_iris()
X = pd.DataFrame(iris.data)
X.columns =  ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y = pd.DataFrame(iris.target)
y.columns = ['Targets']

# Build the K Means Model
model = KMeans(n_clusters=3)
model.fit(X) # model.labels_ : Gives cluster no for which samples belongs to

# # Visualise the clustering results
plt.figure(figsize=(10,5))
colormap = np.array(['red', 'lime', 'black'])
# Plot the Original Classifications using Petal features
plt.subplot(1, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Clusters')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
# Plot the Models Classifications
plt.subplot(1, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K-Means Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.show()
```

OUTPUT:

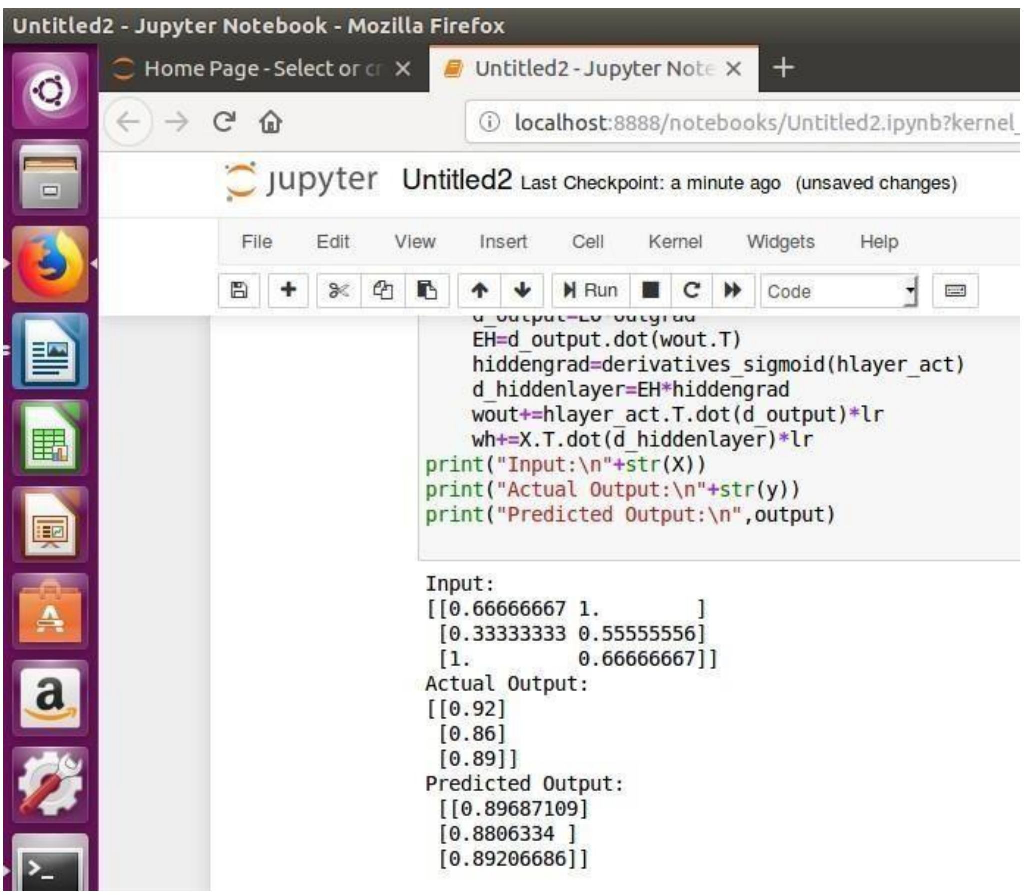

Real Clusters
K-Means Clustering
Petal Width
Petal Length
0.0
0.5
1.0
1.5
2.0
2.5
1
2
3
4
5
6
7

8. Build an Artificial Neural Network by implementing the Backpropagation algorithm and test thesame 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:



9. Write a program to implement the naïve Bayesian classifier for a sample training data setstored as a .CSV file. Compute the accuracy of the classifier, considering few test datasets.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics

df = pd.read_csv("pima_indian.csv")
feature_col_names = ['num_preg', 'glucose_conc', 'diastolic_bp', 'thickness', 'insulin', 'bmi', 'diab_pred', 'age']
predicted_class_names = ['diabetes']

X = df[feature_col_names].values # these are factors for the prediction
y = df[predicted_class_names].values # this is what we want to predict

#splitting the dataset into train and test data

xtrain,xtest,ytrain,ytest=train_test_split(X,y,test_size=0.33)

print ('\n the total number of Training Data :',ytrain.shape)
print ('\n the total number of Test Data :',ytest.shape)


# Training Naive Bayes (NB) classifier on training data.

clf = GaussianNB().fit(xtrain,ytrain.ravel())
predicted = clf.predict(xtest)
predictTestData= clf.predict([[6,148,72,35,0,33.6,0.627,50]])

#printing Confusion matrix, accuracy, Precision and Recall

print('\n Confusion matrix')
print(metrics.confusion_matrix(ytest,predicted))

print('\n Accuracy of the classifier is',metrics.accuracy_score(ytest,predicted))
```

```
print('\n The value of Precision', metrics.precision_score(ytest,predicted))

print('\n The value of Recall', metrics.recall_score(ytest,predicted))

print("Predicted Value for individual Test Data:", predictTestData)
```

OUTPUT:

```
 the total number of Training Data : (514, 1)

 the total number of Test Data : (254, 1)

 Confusion matrix
[[133  32]
 [ 32  57]]

 Accuracy of the classifier is 0.7480314960629921

 The value of Precision 0.6404494382022472

 The value of Recall 0.6404494382022472
Predicted Value for individual Test Data: [1]
```