B. M.S. COLLEGE OF ENGINEERING

(Autonomous College under VTU)

Bull Temple Road, Basavangudi, Bangalore - 560019

MACHINE LEARNING LAB

(20CS6PCMAL)

Report

Submitted by

MD SURAJ KUMAR (1BM20CS079)

VI B

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

Lab Incharge

ANTARA ROY CHOUDHURY

Assistant Professor

M. Tech, CSE

2023-2024

LIST OF PROGRAMS

Date: 01.04.2023

USN: 1BM20CS079

Program 1 – a) Dataset Exploration – Iris data set

- Features in the Iris dataset:
    1. sepal length in cm
    2. sepal width in cm
    3. petal length in cm
    4. petal width in cm
- Target classes to predict:
    1. Iris Setosa
    2. Iris Versicolour
    3. Iris Virginica

In [ ]:
```
from sklearn.datasets import load_iris
iris=load_iris()
```

In [ ]:
```
iris
```

Out[ ]:
```
{'data': array([[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],
```

In [ ]:
```
type(iris)
```

Out[ ]:
```
sklearn.utils._bunch.Bunch
```

In [ ]:
```
iris.keys()
```

Out[ ]:
```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module'])
```

In [ ]:
```
print(iris["target_names"])
```

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

In [ ]:
```
n_samples,n_features=iris.data.shape
print("no of samples",n_samples)
print("no of features",n_features)
print(iris.data[0])
```

```
no of samples 150
no of features 4
[5.1 3.5 1.4 0.2]
```

In [ ]:
```
iris.data[[12,26,89,114]]
```

Out[ ]:
```
array([[4.8, 3. , 1.4, 0.1],
       [5. , 3.4, 1.6, 0.4],
       [5.5, 2.5, 4. , 1.3],
       [5.8, 2.8, 5.1, 2.4]])
```

In [ ]:
```
print(iris.data.shape)
print(iris.target.shape)
```

```
(150, 4)
(150,)
```

In [ ]:

```
print(iris.target)
```

```
[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 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 2
 2 2]
```

In [ ]:

```
import numpy as np
np.bincount(iris.target)
```

Out[ ]:

```
array([50, 50, 50])
```

In [ ]:

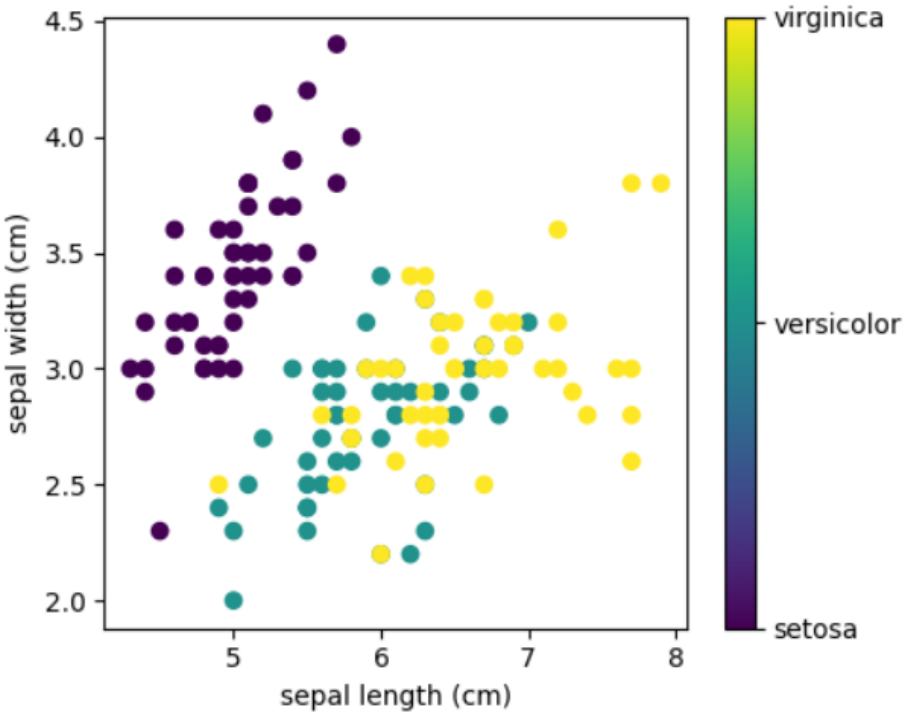
```
import pandas as pd
iris=pd.DataFrame(data=np.c_[iris['data'],iris['target']],columns=iris['feature_names']+['target'])
iris.head(10)
```

Out[ ]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0.0
1	4.9	3.0	1.4	0.2	0.0
2	4.7	3.2	1.3	0.2	0.0
3	4.6	3.1	1.5	0.2	0.0
4	5.0	3.6	1.4	0.2	0.0
5	5.4	3.9	1.7	0.4	0.0
6	4.6	3.4	1.4	0.3	0.0
7	5.0	3.4	1.5	0.2	0.0
8	4.4	2.9	1.4	0.2	0.0
9	4.9	3.1	1.5	0.1	0.0

In [ ]:

```
from matplotlib import pyplot as plt


x_index = 0
y_index = 1


formatter = plt.FuncFormatter(lambda i, *args: iris.target_names[int(i)])

plt.figure(figsize=(5, 4))
plt.scatter(iris.data[:, x_index], iris.data[:, y_index], c=iris.target)
plt.colorbar(ticks=[0, 1, 2], format=formatter)
plt.xlabel(iris.feature_names[x_index])
plt.ylabel(iris.feature_names[y_index])

plt.tight_layout()
plt.show()
```



DESCRIPTION:

Graph is the illustration of comparision between sepal length and sepal width of the three classes of iris flower i.e, Setosa, virginica, versicolor, depicted by 3 different colours for each category.

b. Practice Exercise - Dataset Exploration – Wine data set

In [ ]:
```
from sklearn.datasets import load_wine
wine=load_wine()
```

In [ ]:
```
wine.keys()
```

Out[ ]:
```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names'])
```

In [ ]:
```
print(wine.DESCR)
```

```
.. _wine_dataset:

Wine recognition dataset
------------------------

**Data Set Characteristics:**

    :Number of Instances: 178
    :Number of Attributes: 13 numeric, predictive attributes and the class
    :Attribute Information:
                - Alcohol
                - Malic acid
                - Ash
                - Alcalinity of ash
                - Magnesium
                - Total phenols
                - Flavanoids
                - Nonflavanoid phenols
                - Proanthocyanins
                - Color intensity
                - Hue
                - OD280/OD315 of diluted wines
                - Proline

    - class:
            - class_0
            - class_1
            - class_2
```

In [ ]:
```
print(wine.target_names)
```

```
['class_0' 'class_1' 'class_2']
```

In [ ]:
```
print(wine.feature_names)
```

```
['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium', 'total_phenols', 'flavanoids', 'nonflavanoid_phenols', 'proanthocyanins', 'color_int
ensity', 'hue', 'od280/od315_of_diluted_wines', 'proline']
```

In [ ]:
```
import numpy as np
np.bincount(wine.target)
```

Out[ ]:
```
array([59, 71, 48])
```

1-4-23

Wine-data set

1. Find and load wine data set

```
from sklearn.datasets import load_wine
wine = load_wine()
```

2. Find description of data set

```
wine.DESCR
```

3. Print name of classes

```
wine.target_names
```

4. Print the features

```
wine.feature_names
['alcohol', 'malic_acid', 'ash', hue'
 ...]
```

5. Count the samples in each class

```
import numpy as np
np.bincount(wine.target)
array([59, 71, 48])
```

Date: 08.04.2023

Program 2 – 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 a .CSV file

Data set:

a. Enjoysport

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

Algorithm:

PROGRAM-2
Find-S Algorithm

→ Implement and demonstrate FIND-S algorithm for finding the most specific hypothesis based on a given set of training data sample. Read the training data from ".csv" file

Data set :- enjoy-sport

example	sky	temp	humidity	wind	water	forecast	enjoy sport
1	sunny	warm	normal	strong	warm	same	yes
2	sunny	warm	high	strong	warm	same	yes
3	rainy	cold	high	strong	warm	change	no
4	sunny	warm	high	strong	warm	change	yes

Algorithm :-

1. Initialize h to most specific hypothesis in H.
2. for each positive training instance :
   for each attribute constraint a_i in h :

   if constraint a_i is satisfied by x :
   then do nothing
   else :
   replace a_i in h by the next most general constraint that is satisfied by x

	sky	temp	humidity	wind	water	forecast	enjoySport
0	sunny	warm	normal	strong	warm	same	Yes
1	sunny	warm	high	strong	warm	same	Yes
2	sunny	cold	high	strong	warm	change	no
3	rainy	warm	high	strong	cool warm	change	Yes

```
* d = np.array(data)[:, :-1]
  print("\n Attributes are:", d)

* target = np.array(data)[:, -1]
  print("\n target is ", target)
```

target is : ['Yes', 'Yes', 'no', 'Yes']

```
* def findS(c,t):
      for i, val in enumerate(t):
          if val == "Yes":
              s_hy = c[i].copy()
              break

      for i, val in enumerate(c)
          if t[i] == "Yes":
              for x in range(len(s_hy)):
                  if val[x] != s_hy[x]:
                      s_hy[x] = '?'
                  else:
                      pass
      return s_hy
print("final hypothesis", findS(d, target))
```

final hypothesis is ['sunny', 'warm', '?', 'strong', '?', '?']

3. output hypothesis:-

* $h = \langle \phi, \phi, \phi, \phi, \phi, \phi \rangle$

$x1 = \langle$ sunny, warm, normal, strong, same$\rangle$
$h1 = \langle$ sunny, warm, normal, strong, same$\rangle$

$x2 = \langle$ sunny, warm, high, strong, warm, same$\rangle$
$h2 = \langle$ sunny, warm, ?, strong, warm, same$\rangle$

$x3 = \langle$ rainy, warm, high, strng, warm, change$\rangle$
$h3 = \langle$ sunny, warm, ?, strong, warm, same$\rangle$

$x4 = \langle$ sunny, warm, high, strong, cool, change$\rangle$
$h4 = \langle$ sunny, warm, ?, strong, ?, ?$\rangle$

CODE:-

* import numpy as np
import pandas as pd

* from google.colab import drive
drive.mount("/content/drive")

mounted at /content/drive

* path= "/content/drive/MyDrive/ML-LAB-MD/enjoySport.csv"

* data = pd.read_csv(path)

* print(data, "\n")

Dataset :- 2nd

* data = pd.read_csv("/content/drive/MyDrive/ML-LAB-MD/enjoySport.csv")

* print("\n Target is ", target)

example	citation	size	inlibrary	price	edition	buy
1	some	small	no	affordable	many	no
2	many	big	no	expensive	one	yes
3	some	big	always	expespive	few	no
4	many	medium	no	expespive	many	yes
5	many	small	no	affordable	many	yes

* d = np.array(data)[:, :-1]

* target = np.array(data)[:, -1]
print("Target is", target)

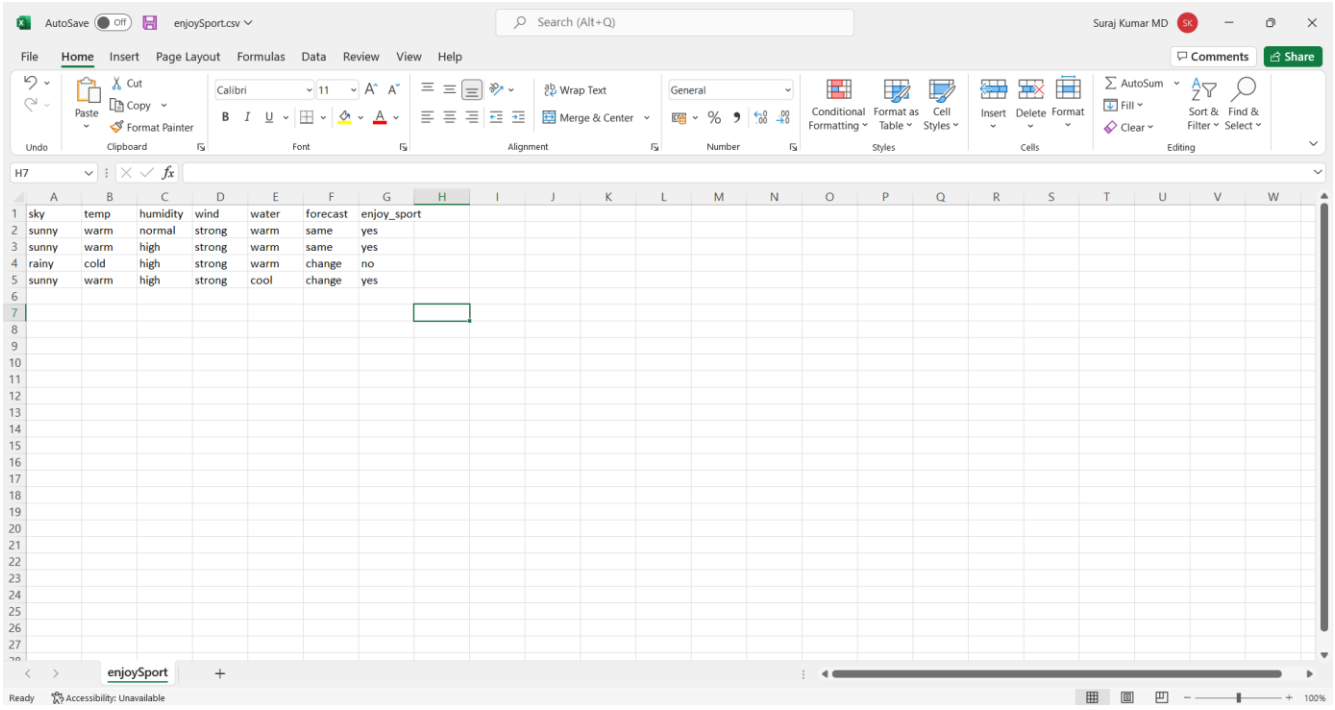
The target is: ['no' 'yes' 'no' 'yes' 'yes']

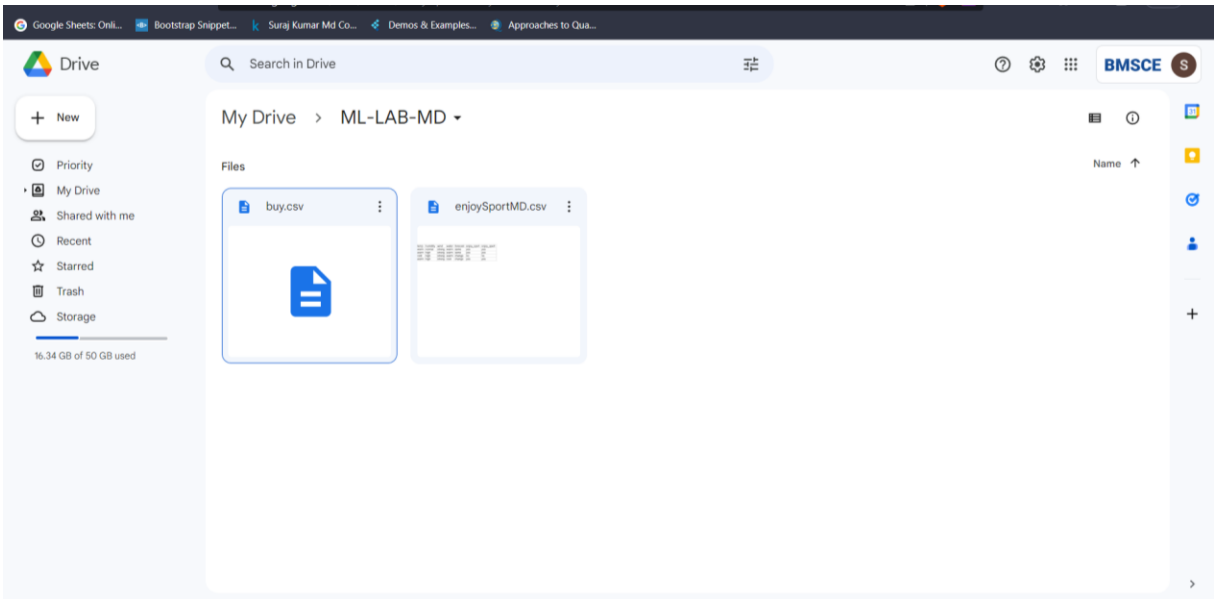
Final output:

The final hypothesis is: ['many' '?' 'no' '?' '?']

1. The first step of FIND-S is to initialize h to the most specific hypothesis in H **h — (Ø, Ø, Ø, Ø, Ø, Ø)**
2. First training example x1 = < Sunny, Warm, Normal, Strong ,Warm ,Same>, EnjoySport = +ve. Observing the first training example, it is clear that hypothesis h is too specific. None of the "Ø" constraints in h are satisfied by this example, so each is replaced by the next more general constraint that fits the example **h1 = < Sunny, Warm, Normal, Strong ,Warm, Same>**.
3. Consider the second training example x2 = < Sunny, Warm, High, Strong, Warm, Same>, EnjoySport = +ve. The second training example forces the algorithm to further generalize h, this time substituting a "?" in place of any attribute value in h that is not satisfied by the new example. Now **h2 =< Sunny, Warm, ?, Strong, Warm, Same>**
4. Consider the third training example x3 =< Rainy, Cold, High, Strong, Warm, Change>,EnjoySport = — ve. The FIND-S algorithm simply ignores every negative example. So the hypothesis remain as before, so **h3 =< Sunny, Warm, ?, Strong, Warm, Same>**
5. Consider the fourth training example x4 =<Sunny,Warm,High,Strong, Cool,Change>, EnjoySport =+ve. The fourth example leads to a further generalization of h as **h4 =< Sunny, Warm, ?, Strong, ?, ?>**
6. So the final hypothesis is **< Sunny, Warm, ?, Strong, ?, ?>**

Uploading the csv file:





a. enjoySport.csv

Program:

SURAJ-LAB2.ipynb

File Edit View Insert Runtime Tools Help Last saved at 9:22 AM

+ Code + Text

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

```
[ ]  from google.colab import drive
     drive.mount("/content/drive")
```

```
Mounted at /content/drive
```

```
[ ]  path ="/content/drive/MyDrive/ML-LAB-MD/enjoySport.csv"
```

```
[ ]  data = pd.read_csv(path)
```

```
[ ]  print(data,"\n")
```

```
      sky  temp humidity    wind water forecast enjoy_sport
0   sunny  warm   normal  strong  warm     same          yes
1   sunny  warm     high  strong  warm     same          yes
2   rainy  cold     high  strong  warm   change           no
3   sunny  warm     high  strong  cool   change          yes
```

SURAJ-LAB2.ipynb

File Edit View Insert Runtime Tools Help Last saved at 9:22 AM

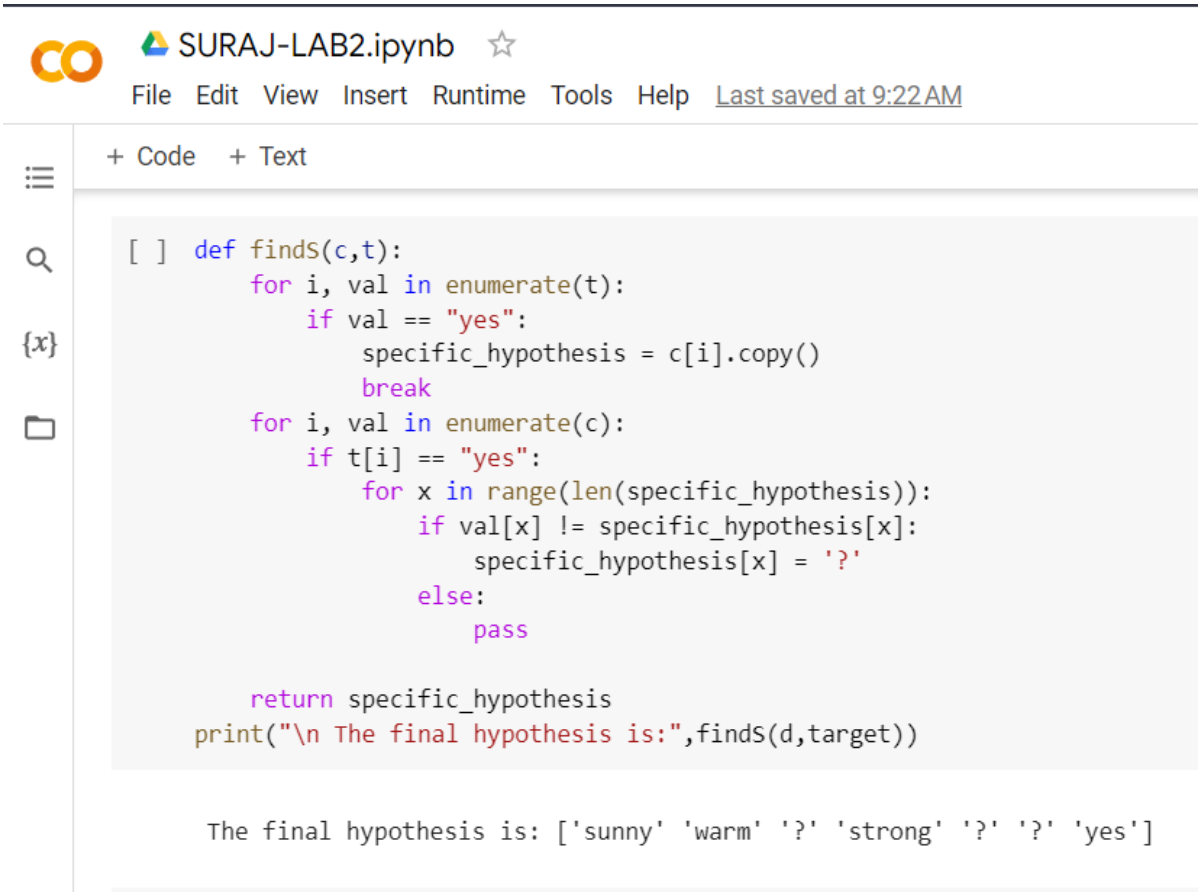
+ Code + Text

```
[ ]  d = np.array(data)[:,:-1]
     print("\n The attributes are: ",d)
```

```
The attributes are:  [['sunny' 'warm' 'normal' 'strong' 'warm' 'same' 'yes']
['sunny' 'warm' 'high' 'strong' 'warm' 'same' 'yes']
['rainy' 'cold' 'high' 'strong' 'warm' 'change' 'no']
['sunny' 'warm' 'high' 'strong' 'cool' 'change' 'yes']]
```

```
[ ]  target = np.array(data)[:,-1]
     print("\n The target is: ",target)
```

```
The target is:  ['yes' 'yes' 'no' 'yes']
```



Output:

```
The final hypothesis is: ['sunny' 'warm' '?' 'strong' '?' '?' 'yes']
```

b. Purchase.csv

example	*citations*	*size*	*inLibrary*	*price*	*editions*	*buy*
1	some	small	no	affordable	many	no
2	many	big	no	expensive	one	yes
3	some	big	always	expensive	few	no
4	many	medium	no	expensive	many	yes
5	many	small	no	affordable	many	yes

Uploading the csv file:

buy.csv

	A	B	C	D	E	F
1	citations	size	inLibrary	price	editions	buy
2	some	small	no	affordable	many	no
3	many	big	no	expensive	one	yes
4	some	big	always	expensive	few	no
5	many	medium	no	expensive	many	yes
6	many	small	no	affordable	many	yes

SURAJ-LAB2.ipynb

File Edit View Insert Runtime Tools Help Last saved at 9:22 AM

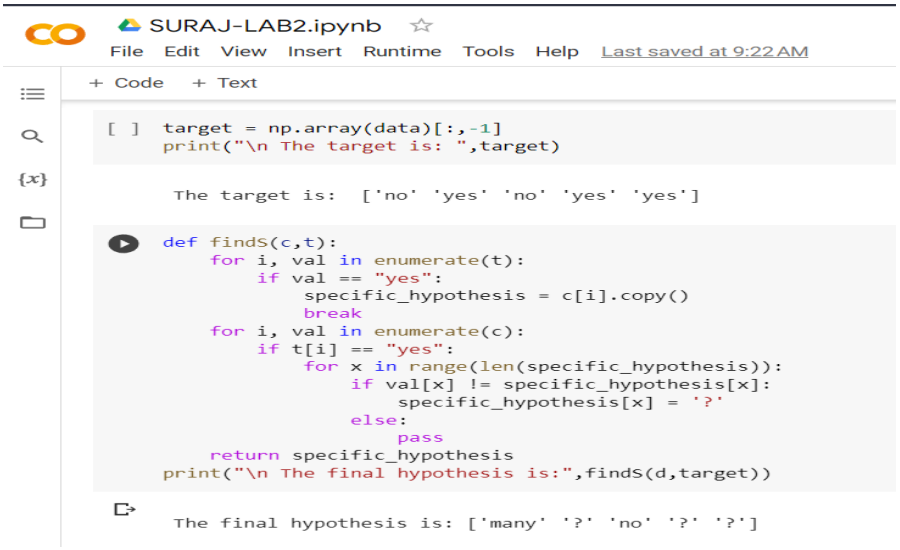
+ Code + Text

```
[ ]  #2ND QUESTION

[ ]  path ="/content/drive/MyDrive/ML-LAB-MD/buy.csv"

     data = pd.read_csv(path)

[ ]  print(data,"\n")
```

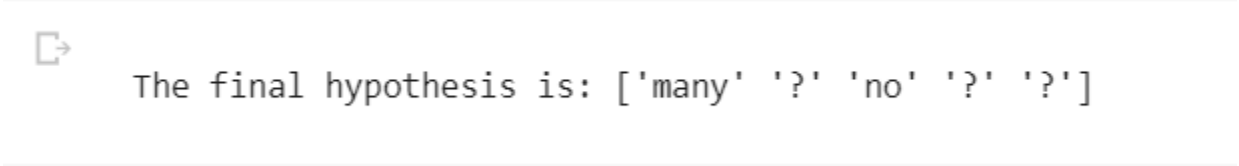
```
  citations    size inLibrary       price editions  buy
0      some   small        no  affordable      many   no
1      many     big        no   expensive       one  yes
2      some     big    always   expensive       few   no
3      many  medium        no   expensive      many  yes
4      many   small        no  affordable      many  yes
```

```
[ ]  d = np.array(data)[:,:5]
     print("\n The attributes are: ",d)
```

```
 The attributes are:  [['some' 'small' 'no' 'affordable' 'many']
['many' 'big' 'no' 'expensive' 'one']
['some' 'big' 'always' 'expensive' 'few']
['many' 'medium' 'no' 'expensive' 'many']
['many' 'small' 'no' 'affordable' 'many']]
```



Output



Date: 15.04.2023

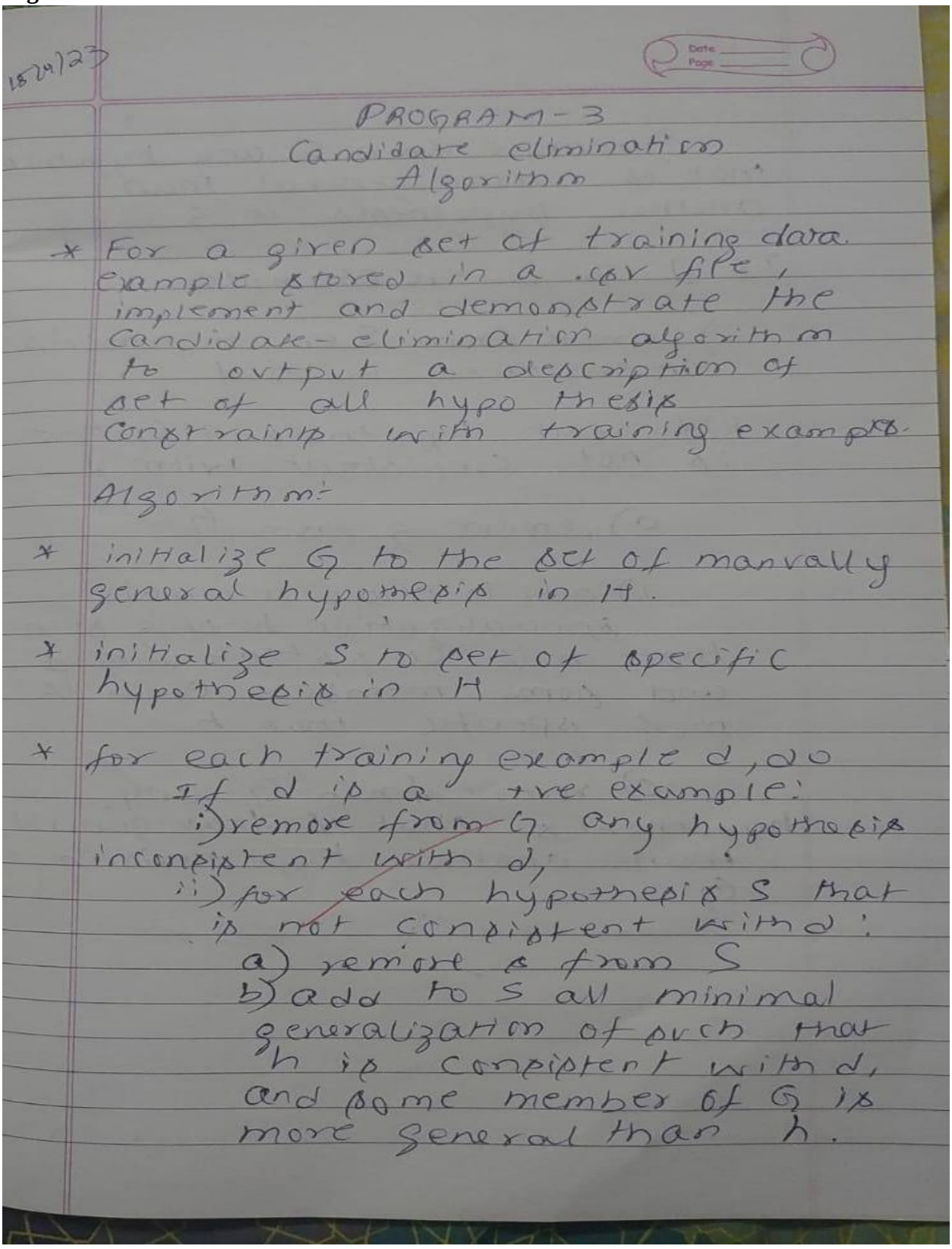
Program 3:

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.

Dataset : Enjoysport

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

Algorithm:



18/4/23

PROGRAM-3

Candidate elimination Algorithm

* For a given set of training data example stored in a .csv file, implement and demonstrate the Candidate-elimination algorithm to output a description of set of all hypothesis constraints with training examples.

Algorithm:

* initialize G to the set of manually general hypothesis in H.
* initialize S to set of specific hypothesis in H
* for each training example d, do
  If d is a +ve example:
    i) remove from G any hypothesis inconsistent with d,
    ii) for each hypothesis S that is not consistent with d:
      a) remove s from S
      b) add to S all minimal generalization of such that h is consistent with d, and some member of G is more general than h.

c) remove from S any hypothesis that is more general than another hypothesis in S

If d is a -ve example:

i) remove from S any hypothesis inconsistent with d

ii) for each hypothesis in G that is not consistent with d

a) remove g from G

b) add to G all minimal specialization h of g such that h is consistent with d, and some members of S is more specific than h.

c) remove from G any hypothesis that is less general than another hypothesis in G

Iterations for enjoysport.csv

I_0 $S_0 = \langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$
$G_0 = \langle ? \; ? \; ? \; ? \; ? \; ? \rangle$

I_1 $S_1 = \langle$sunny, warm, normal, strong, warm, same$\rangle$
$G_1 = \langle ? \; ? \; ? \; ? \; ? \; ? \rangle$

I_2 $S_2 = \langle$sunny, warm, ?, strong, warm, same$\rangle$
$G_2 = \langle ? \; ? \; ? \; ? \; ? \; ? \rangle$

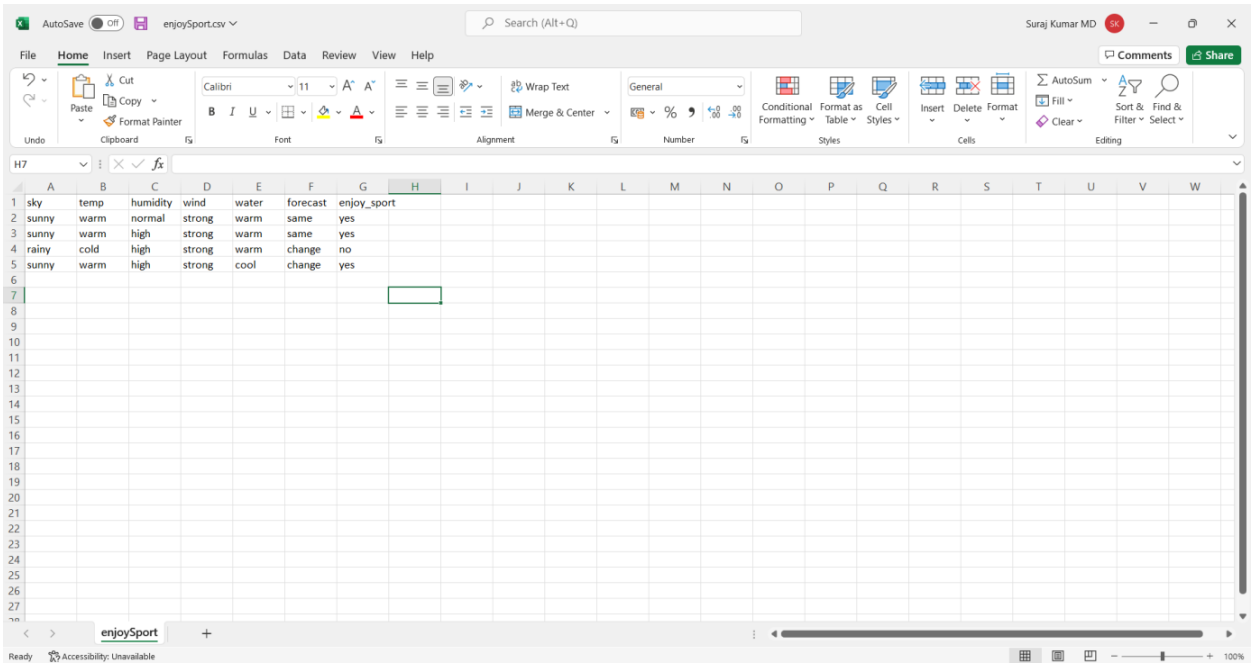
I_3 $S_3 = \langle$sunny, warm, ?, strong, warm, same$\rangle$

$G_3 = \langle$sunny ? ? ? ? ?$\rangle$
$\langle ?$ warm ? ? ? ?$\rangle$
$\langle ? \; ? \; ? \; ? \; ? \;$ same$\rangle$

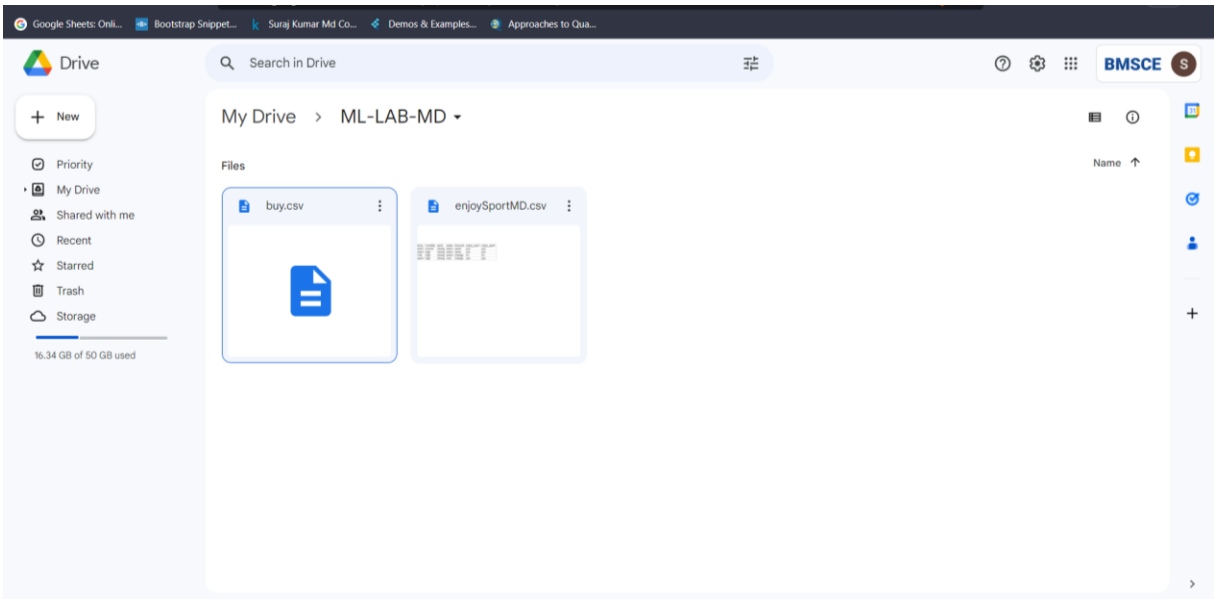
I_4 $S_4 = \langle$sunny, warm, ?, strong, ?, ?$\rangle$
$G_4 = \langle$sunny, ? ? ? ? ?$\rangle$
$\langle ?$ warm ? ? ? ?$\rangle$

13/11/23

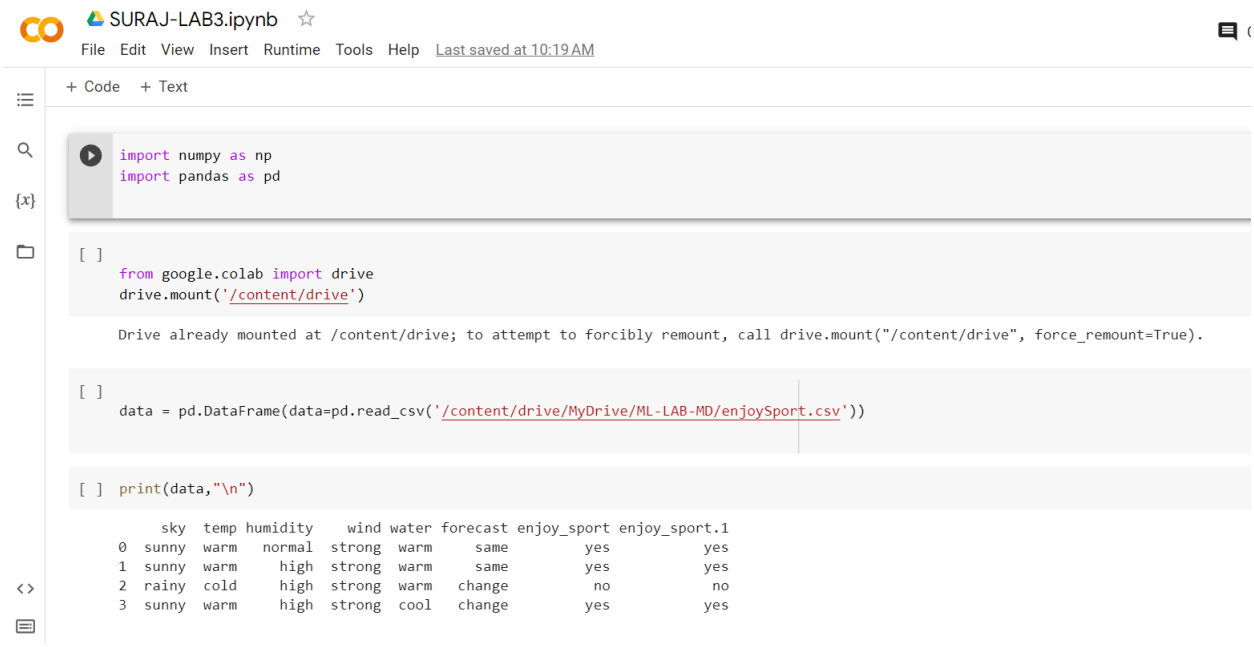
Uploading the csv file:

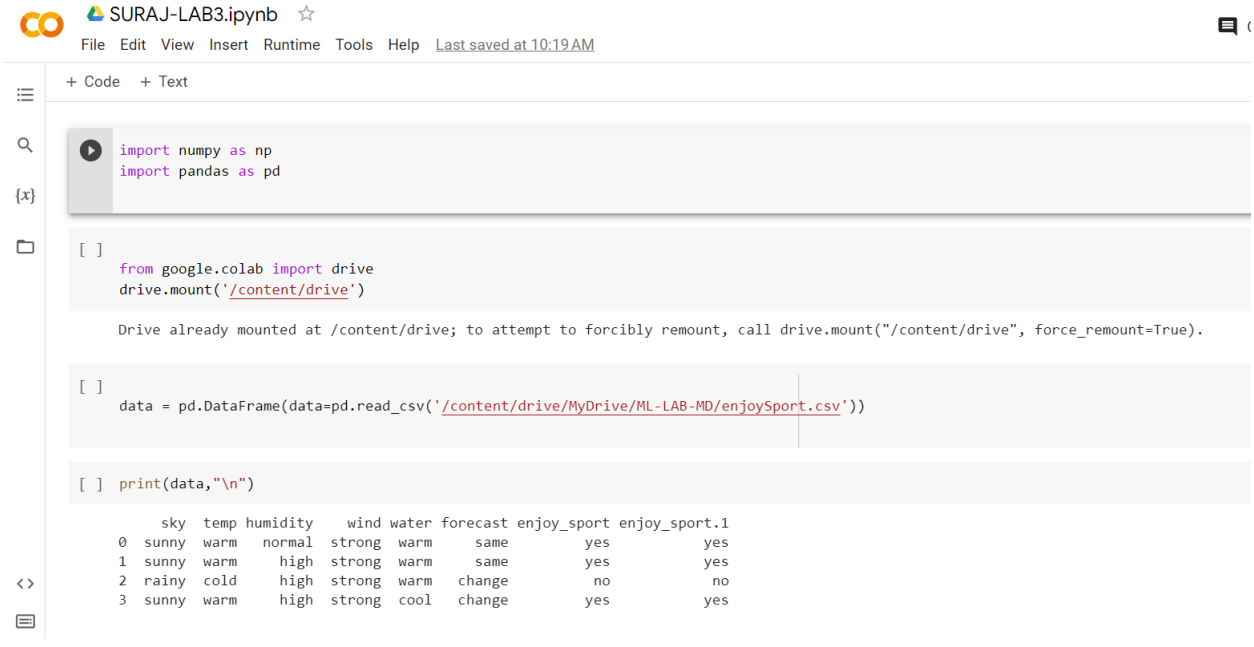


sky	temp	humidity	wind	water	forecast	enjoy_sport
sunny	warm	normal	strong	warm	same	yes
sunny	warm	high	strong	warm	same	yes
rainy	cold	high	strong	warm	change	no
sunny	warm	high	strong	cool	change	yes



Program:


SURAJ-LAB3.ipynb
File Edit View Insert Runtime Tools Help Last saved at 10:19 AM
+ Code + Text
import numpy as np
import pandas as pd
[ ]
from google.colab import drive
drive.mount('/content/drive')
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
[ ]
data = pd.DataFrame(data=pd.read_csv('/content/drive/MyDrive/ML-LAB-MD/enjoySport.csv'))
[ ] print(data,"\n")
sky temp humidity wind water forecast enjoy_sport enjoy_sport.1
0 sunny warm normal strong warm same yes yes
1 sunny warm high strong warm same yes yes
2 rainy cold high strong warm change no no
3 sunny warm high strong cool change yes yes


SURAJ-LAB3.ipynb
File Edit View Insert Runtime Tools Help Last saved at 10:19 AM
+ Code + Text
import numpy as np
import pandas as pd
[ ]
from google.colab import drive
drive.mount('/content/drive')
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
[ ]
data = pd.DataFrame(data=pd.read_csv('/content/drive/MyDrive/ML-LAB-MD/enjoySport.csv'))
[ ] print(data,"\n")
sky temp humidity wind water forecast enjoy_sport enjoy_sport.1
0 sunny warm normal strong warm same yes yes
1 sunny warm high strong warm same yes yes
2 rainy cold high strong warm change no no
3 sunny warm high strong cool change yes yes

```python
def learn(concepts, target):
    specific_h = concepts[0].copy()
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print("Step 0:")
    print("Specific Hypothesis: ", specific_h)
    print("General Hypothesis: ", general_h)
    print("--------------------")
    for i, h in enumerate(concepts):
        if target[i] == "yes":
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    specific_h[x] = '?'
                    general_h[x][x] = '?'
        if target[i] == "no":
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'
        print("Step", i+1, ":")
        print("Specific Hypothesis: ", specific_h)
        print("General Hypothesis: ", general_h)
        print("--------------------")
    indices = [i for i,val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h


s_final, g_final = learn(concepts, target)
print("Final S:", s_final, sep="\n")
print("Final G:", g_final, sep="\n")
```

Output:

```
Step 0:
Specific Hypothesis:  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
General Hypothesis:  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
--------------------
Step 1 :
Specific Hypothesis:  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
General Hypothesis:  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
--------------------
Step 2 :
Specific Hypothesis:  ['sunny' 'warm' '?' 'strong' 'warm' 'same']
General Hypothesis:  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
--------------------
Step 3 :
Specific Hypothesis:  ['sunny' 'warm' '?' 'strong' 'warm' 'same']
General Hypothesis:  [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]
--------------------
Step 4 :
Specific Hypothesis:  ['sunny' 'warm' '?' 'strong' '?' '?']
General Hypothesis:  [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
--------------------
Final S:
['sunny' 'warm' '?' 'strong' '?' '?']
Final G:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

b. Purchase.csv

example	*citations*	*size*	*inLibrary*	*price*	*editions*	*buy*
1	some	small	no	affordable	many	no
2	many	big	no	expensive	one	yes
3	some	big	always	expensive	few	no
4	many	medium	no	expensive	many	yes
5	many	small	no	affordable	many	yes

Uploading the csv file:

buy.csv

	A	B	C	D	E	F
1	citations	size	inLibrary	price	editions	buy
2	some	small	no	affordable	many	no
3	many	big	no	expensive	one	yes
4	some	big	always	expensive	few	no
5	many	medium	no	expensive	many	yes
6	many	small	no	affordable	many	yes

Program:

SURAJ-LAB3.ipynb

File Edit View Insert Runtime Tools Help Last saved at 10:19 AM

+ Code + Text

```
[ ]  data = pd.DataFrame(data=pd.read_csv('/content/drive/MyDrive/ML-LAB-MD/buy.csv'))
     print(data,"\n")
     concepts = np.array(data.iloc[:,0:-1])
     print("The attributes are: ",concepts)
     target = np.array(data.iloc[:,-1])
     print("\n The target is: ",target)
```

```
  citations    size inLibrary       price editions  buy
0      some   small        no  affordable     many   no
1      many     big        no   expensive      one  yes
2      some     big    always   expensive      few   no
3      many  medium        no   expensive     many  yes
4      many   small        no  affordable     many  yes

The attributes are:  [['some' 'small' 'no' 'affordable' 'many']
 ['many' 'big' 'no' 'expensive' 'one']
 ['some' 'big' 'always' 'expensive' 'few']
 ['many' 'medium' 'no' 'expensive' 'many']
 ['many' 'small' 'no' 'affordable' 'many']]

 The target is:  ['no' 'yes' 'no' 'yes' 'yes']
```

```python
def learn(concepts, target):
    specific_h = concepts[1].copy()
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print("Step 0:")
    print("Specific Hypothesis: ", specific_h)
    print("General Hypothesis: ", general_h)
    print("--------------------")
    for i, h in enumerate(concepts):
        print(target[0])
        if target[i] == "yes":
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    specific_h[x] = '?'
                    general_h[x][x] = '?'
        if target[i] == "no":
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'
        print("Step", i+1, ":")
        print("Specific Hypothesis: ", specific_h)
        print("General Hypothesis: ", general_h)
        print("--------------------")
    indices = [i for i,val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h


s_final, g_final = learn(concepts, target)
print("Final S:", s_final, sep="\n")
print("Final G:", g_final, sep="\n")
```

Output:

```
Step 0:
Specific Hypothesis:  ['many' 'big' 'no' 'expensive' 'one']
General Hypothesis:  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
--------------------
no
Step 1 :
Specific Hypothesis:  ['many' 'big' 'no' 'expensive' 'one']
General Hypothesis:  [['many', '?', '?', '?', '?', '?'], ['?', 'big', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', 'expensive', '?', '?'], ['?', '?', '?', '?', 'one']]
--------------------
no
Step 2 :
Specific Hypothesis:  ['many' 'big' 'no' 'expensive' 'one']
General Hypothesis:  [['many', '?', '?', '?', '?', '?'], ['?', 'big', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', 'expensive', '?', '?'], ['?', '?', '?', '?', 'one']]
--------------------
no
Step 3 :
Specific Hypothesis:  ['many' 'big' 'no' 'expensive' 'one']
General Hypothesis:  [['many', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', 'no', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', 'one']]
--------------------
no
Step 4 :
Specific Hypothesis:  ['many' '?' 'no' 'expensive' '?']
General Hypothesis:  [['many', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', 'no', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
--------------------
no
Step 5 :
Specific Hypothesis:  ['many' '?' 'no' '?' '?']
General Hypothesis:  [['many', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', 'no', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
--------------------
Final S:
['many' '?' 'no' '?' '?']
Final G:
[['many', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', 'no', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
```

Date: 29.04.2023

Program 4:

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.

Dataset : Climate(Forecast)

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Algorithm:

ID-3 DECISION-TREE

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

Algorithm:-

ID3 (examples, target_attribute, attributes)

* examples:- training examples
* Target_attribute:- value to be predicted by tree.
* Attributes :- list of attributes that may be tested by learned decision tree.

→ Create a root node for tree.

→ If all samples are +ve, return the single node tree with label=+

→ If all examples are negative, return single node root with label = -

single node tree root, with label = most common value of target attributes in examples.

→ otherwise Begin:

i) A ← attribute from attributes that best classifies examples

ii) The decision attribute for root ← A

iii) for each possible value, v_i, of A

Add a new tree branch below root, corresponding to the A = v_i;

iv) let examples v_i, be the subset of examples, that have a value v_i for A

v) If examples v_i is empty.

Then below their new branch add a leaf node with label = most common attribute of Target attributes in examples.

else

below their new branch add the subtree ID3(example,

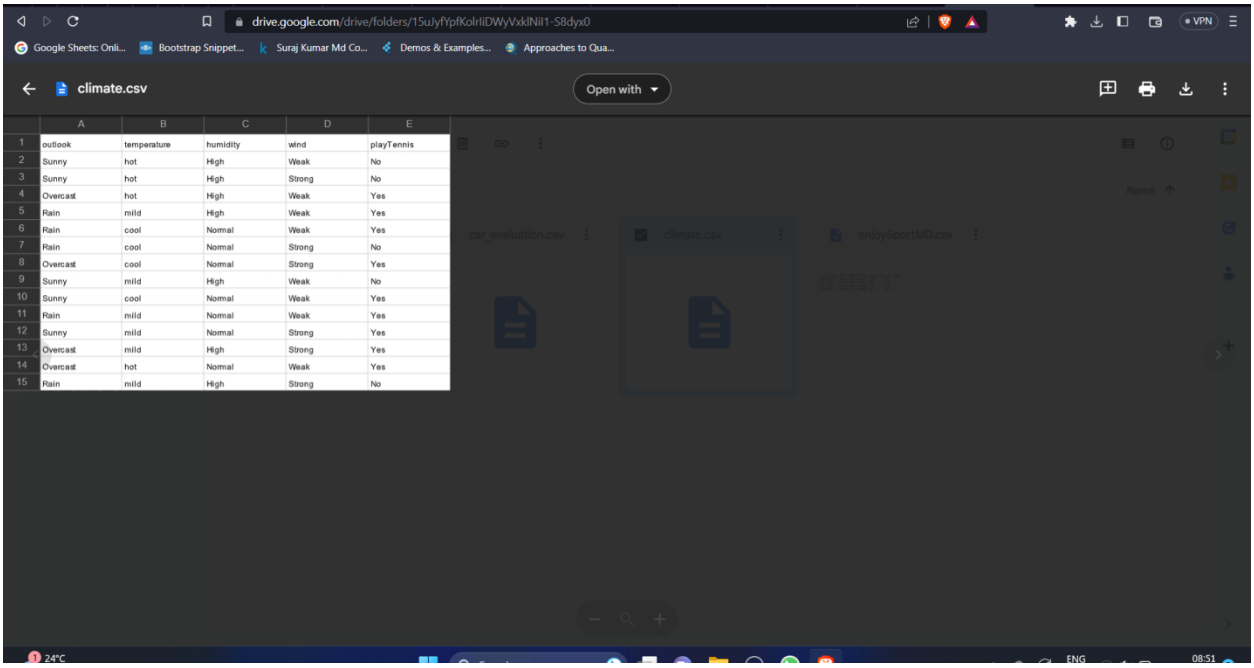
Entropy(S) = $-P_{\oplus} \log_2 P_{\oplus} - P_{\ominus} \log_2 P_{\ominus}$

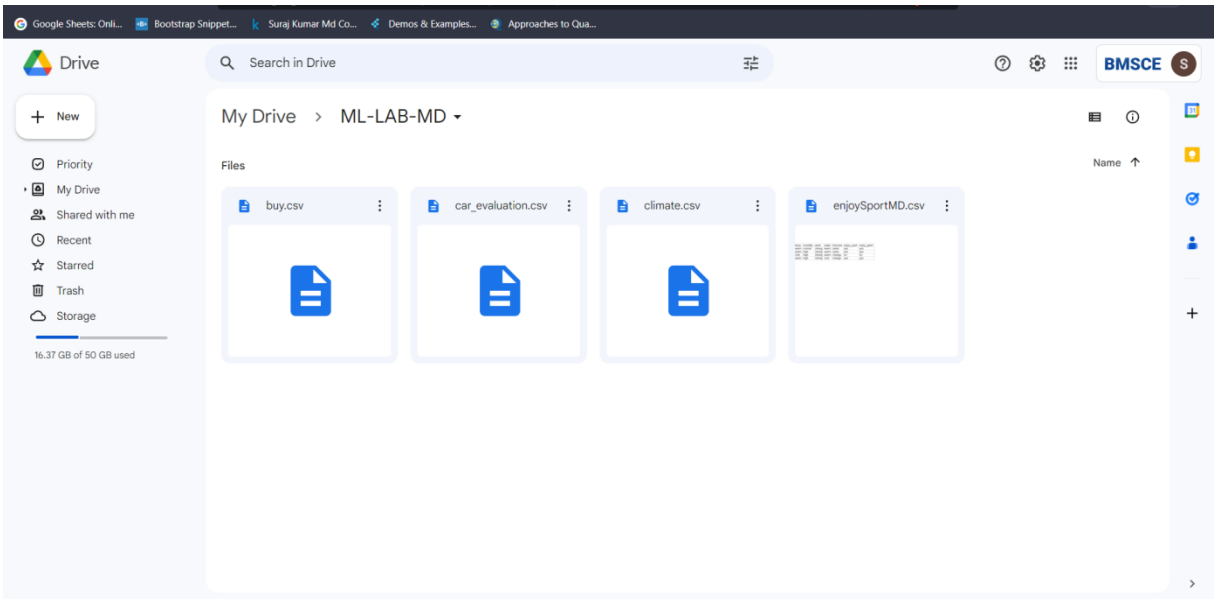
$IG = S - \sum_{v \in V} \frac{|S_v|}{|S|} \text{entropy}(v)$

OUT PUT

{'outlook': {'overcast': 'yes',
'rain': {'wind': {'strong': 'no',
'weak': 'yes'}},
'sunny': {'humidity': {'high': 'no', 'normal': 'yes'}}}}

Uploading the csv file:





Program:

SURAJ-LAB3.ipynb

File Edit View Insert Runtime Tools Help Last saved at April 29

+ Code + Text

```
https://medium.com/@lope.ai/decision-trees-from-scratch-using-id3-python-coding-it-up-6b79e3458de4
```

```
[ ] from google.colab import drive
    drive.mount("/content/drive")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
[ ] path ="/content/drive/MyDrive/ML-LAB-MD/climate.csv"
```

```
import pandas as pd
import math
import numpy as np

data = pd.read_csv(path)
features = [feat for feat in data]
```

```
[ ] print(data,"\n")
```

```
     outlook temperature humidity    wind playTennis
0      Sunny         hot     High    Weak         No
1      Sunny         hot     High  Strong         No
2   Overcast         hot     High    Weak        Yes
3       Rain        mild     High    Weak        Yes
4       Rain        cool   Normal    Weak        Yes
5       Rain        cool   Normal  Strong         No
6   Overcast        cool   Normal  Strong        Yes
7      Sunny        mild     High    Weak         No
8      Sunny        cool   Normal    Weak        Yes
9       Rain        mild   Normal    Weak        Yes
10     Sunny        mild   Normal  Strong        Yes
11  Overcast        mild     High  Strong        Yes
12  Overcast         hot   Normal    Weak        Yes
13      Rain        mild     High  Strong         No
```

```
[ ] d = np.array(data)[:,:-1]
    print("\n The attributes are: ",d)
```

SURAJ-LAB3.ipynb

File Edit View Insert Runtime Tools Help Last saved at April 29

+ Code + Text

```
d = np.array(data)[:,:-1]
print("\n The attributes are: ",d)
```

```
 The attributes are:  [['Sunny' 'hot' 'High' 'Weak']
 ['Sunny' 'hot' 'High' 'Strong']
 ['Overcast' 'hot' 'High' 'Weak']
 ['Rain' 'mild' 'High' 'Weak']
 ['Rain' 'cool' 'Normal' 'Weak']
 ['Rain' 'cool' 'Normal' 'Strong']
 ['Overcast' 'cool' 'Normal' 'Strong']
 ['Sunny' 'mild' 'High' 'Weak']
 ['Sunny' 'cool' 'Normal' 'Weak']
 ['Rain' 'mild' 'Normal' 'Weak']
 ['Sunny' 'mild' 'Normal' 'Strong']
 ['Overcast' 'mild' 'High' 'Strong']
 ['Overcast' 'hot' 'Normal' 'Weak']
 ['Rain' 'mild' 'High' 'Strong']]
```

```
[ ] target = np.array(data)[:,-1]
    print("\n The target is: ",target)
```

```
 The target is:  ['No' 'No' 'Yes' 'Yes' 'Yes' 'No' 'Yes' 'No' 'Yes' 'Yes' 'Yes' 'Yes' 'Yes'
 'No']
```

SURAJ-LAB3.ipynb

File Edit View Insert Runtime Tools Help Last saved at April 29

+ Code + Text

```
import numpy as np
import pandas as pd
eps = np.finfo(float).eps
from numpy import log2 as log

def find_entropy(df):
    Class = df.keys()[-1]   #To make the code generic, changing target variable class name
    entropy = 0
    values = df[Class].unique()
    for value in values:
        fraction = df[Class].value_counts()[value]/len(df[Class])
        entropy += -fraction*np.log2(fraction)
    return entropy


def find_entropy_attribute(df,attribute):
  Class = df.keys()[-1]
  target_variables = df[Class].unique()  #This gives all 'Yes' and 'No'
  variables = df[attribute].unique()
    #This gives different features in that attribute (like 'Hot','Cold' in Temperature)
  entropy2 = 0
  for variable in variables:
      entropy = 0
      for target_variable in target_variables:
          num = len(df[attribute][df[attribute]==variable][df[Class] ==target_variable])
          den = len(df[attribute][df[attribute]==variable])
          fraction = num/(den+eps)
          entropy += -fraction*log(fraction+eps)
      fraction2 = den/len(df)
      entropy2 += -fraction2*entropy
  return abs(entropy2)
```

SURAJ-LAB3.ipynb

File Edit View Insert Runtime Tools Help Last saved at April 29

+ Code + Text

```
def find_winner(df):
    Entropy_att = []
    IG = []
    for key in df.keys()[:-1]:
#         Entropy_att.append(find_entropy_attribute(df,key))
        IG.append(find_entropy(df)-find_entropy_attribute(df,key))
    #rint(IG)
    #print(df.keys()[:-1][np.argmax(IG)])
    return df.keys()[:-1][np.argmax(IG)]


def get_subtable(df, node,value):
  return df[df[node] == value].reset_index(drop=True)
```

SURAJ-LAB3.ipynb

File Edit View Insert Runtime Tools Help Last saved at April 29

+ Code + Text

```
def buildTree(df,tree=None):
    Class = df.keys()[-1]
    #Here we build our decision tree

    #Get attribute with maximum information gain
    node = find_winner(df)

    #Get distinct value of that attribute e.g Salary is node and Low,Med and High are val
    attValue = np.unique(df[node])

    #Create an empty dictionary to create tree
    if tree is None:
        tree={}
        tree[node] = {}

   #We make loop to construct a tree by calling this function recursively.
    #In this we check if the subset is pure and stops if it is pure.

    for value in attValue:

        subtable = get_subtable(df,node,value)
        clValue,counts = np.unique(subtable['playTennis'],return_counts=True)

        if len(counts)==1:#Checking purity of subset
            tree[node][value] = clValue[0]
        else:
            tree[node][value] = buildTree(subtable) #Calling the function recursively

    return tree
tree = buildTree(data)
```

Output:

SURAJ-LAB3.ipynb

File Edit View Insert Runtime Tools Help Last saved at April 29

+ Code + Text

```
import pprint
pprint.pprint(tree)
```

```
{'outlook': {'Overcast': 'Yes',
             'Rain': {'wind': {'Strong': 'No', 'Weak': 'Yes'}},
             'Sunny': {'humidity': {'High': 'No', 'Normal': 'Yes'}}}}
```

Date: 29.04.2023

Program 5:

Write a program to demonstrate the working of Linear Regression

Dataset : Salary.csv

	A	B	C	D	E	F	G	H	I	J
1	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities
2	1461	20	RH	80	11622	Pave	NA	Reg	Lvl	AllPub
3	1462	20	RL	81	14267	Pave	NA	IR1	Lvl	AllPub
4	1463	60	RL	74	13830	Pave	NA	IR1	Lvl	AllPub
5	1464	60	RL	78	9978	Pave	NA	IR1	Lvl	AllPub
6	1465	120	RL	43	5005	Pave	NA	IR1	HLS	AllPub
7	1466	60	RL	75	10000	Pave	NA	IR1	Lvl	AllPub
8	1467	20	RL	NA	7980	Pave	NA	IR1	Lvl	AllPub
9	1468	60	RL	63	8402	Pave	NA	IR1	Lvl	AllPub
10	1469	20	RL	85	10176	Pave	NA	Reg	Lvl	AllPub
11	1470	20	RL	70	8400	Pave	NA	Reg	Lvl	AllPub
12	1471	120	RH	26	5858	Pave	NA	IR1	Lvl	AllPub
13	1472	160	RM	21	1680	Pave	NA	Reg	Lvl	AllPub
14	1473	160	RM	21	1680	Pave	NA	Reg	Lvl	AllPub
15	1474	160	RL	24	2280	Pave	NA	Reg	Lvl	AllPub
16	1475	120	RL	24	2280	Pave	NA	Reg	Lvl	AllPub
17	1476	60	RL	102	12858	Pave	NA	IR1	Lvl	AllPub
18	1477	20	RL	94	12883	Pave	NA	IR1	Lvl	AllPub
19	1478	20	RL	90	11520	Pave	NA	Reg	Lvl	AllPub
20	1479	20	RL	79	14122	Pave	NA	IR1	Lvl	AllPub
21	1480	20	RL	110	14300	Pave	NA	Reg	HLS	AllPub
22	1481	60	RL	105	13650	Pave	NA	Reg	Lvl	AllPub
23	1482	120	RL	41	7132	Pave	NA	IR1	Lvl	AllPub
24	1483	20	RL	100	18494	Pave	NA	IR1	Lvl	AllPub

12/5/23

PROGRAM-5

Linear Regression :- computes linear relationship between a dependent variable and one or more independent feature.

ex:- independent feature = experience
= X

dependend = salary
= Y

equation ⇒ $Y = b + aX + \varepsilon$
(ε ↓ error term)

Algorithm:-

dataset :-

experience	salary
1.1	39343
1.3	46205
1.5	37731
2	43525

* Assing independent column to X
* Assign dependent column to y
* split testing and training data set accordingly
  ex:- 70% train , 30% test
* teach the model using linear regression.

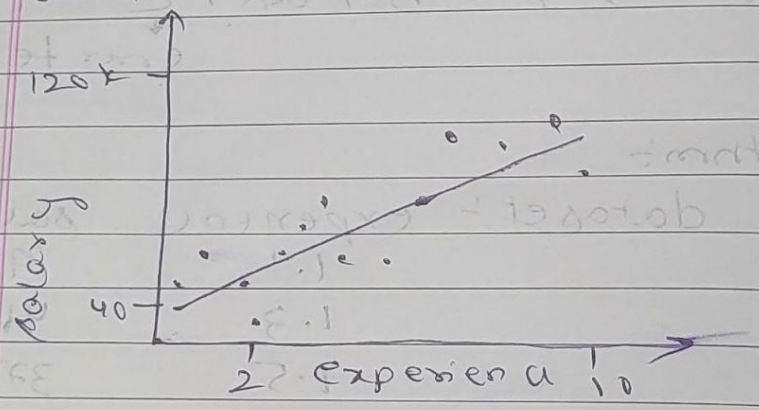
* predict the values of test set

* plot graph for both test and training dataset

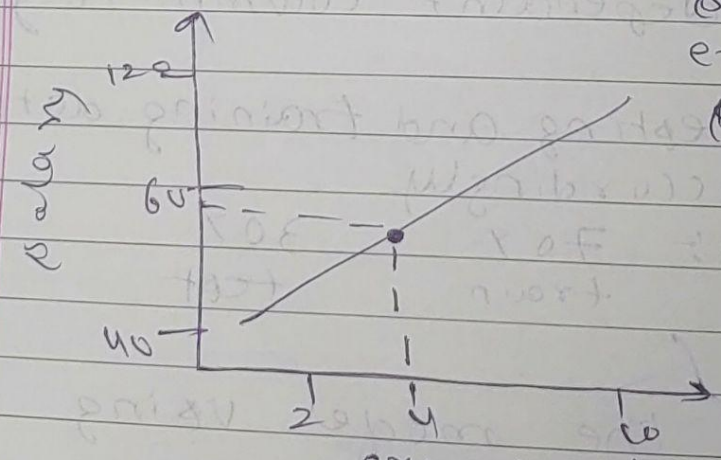
OUTPUT :-

ex:-

~~test~~ set :-
train

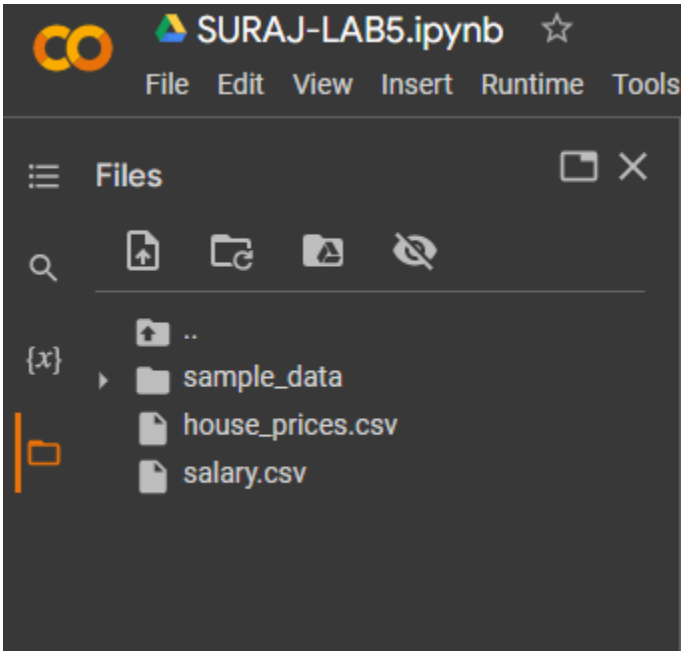


test set :-



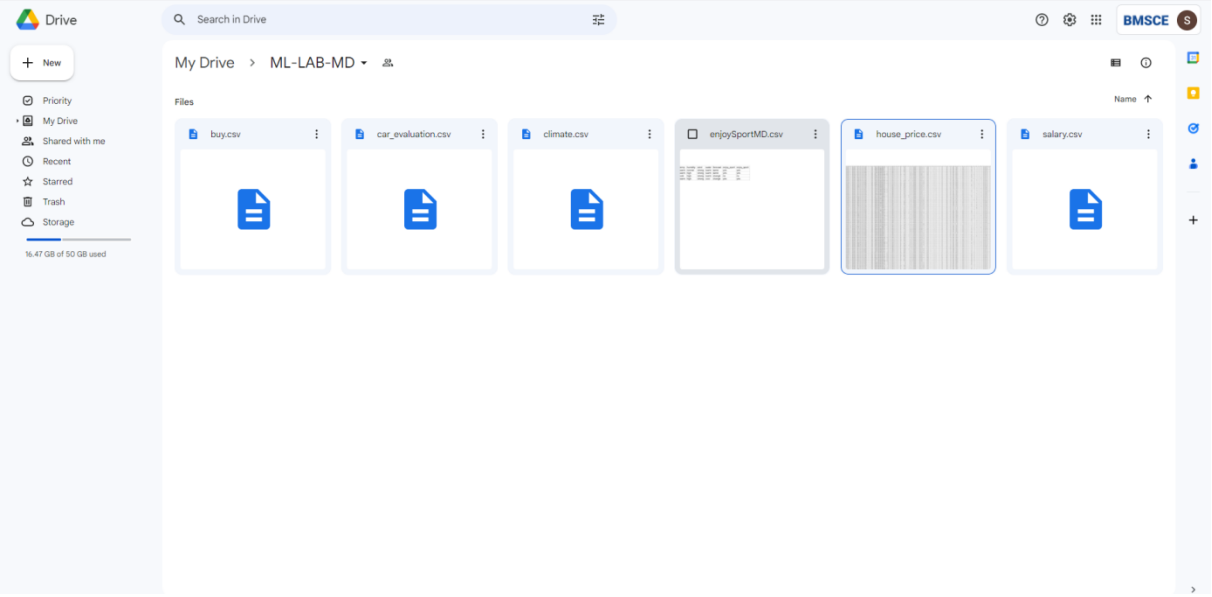
says when experience = 4yrs salary is around 59000

Uploading the csv file:

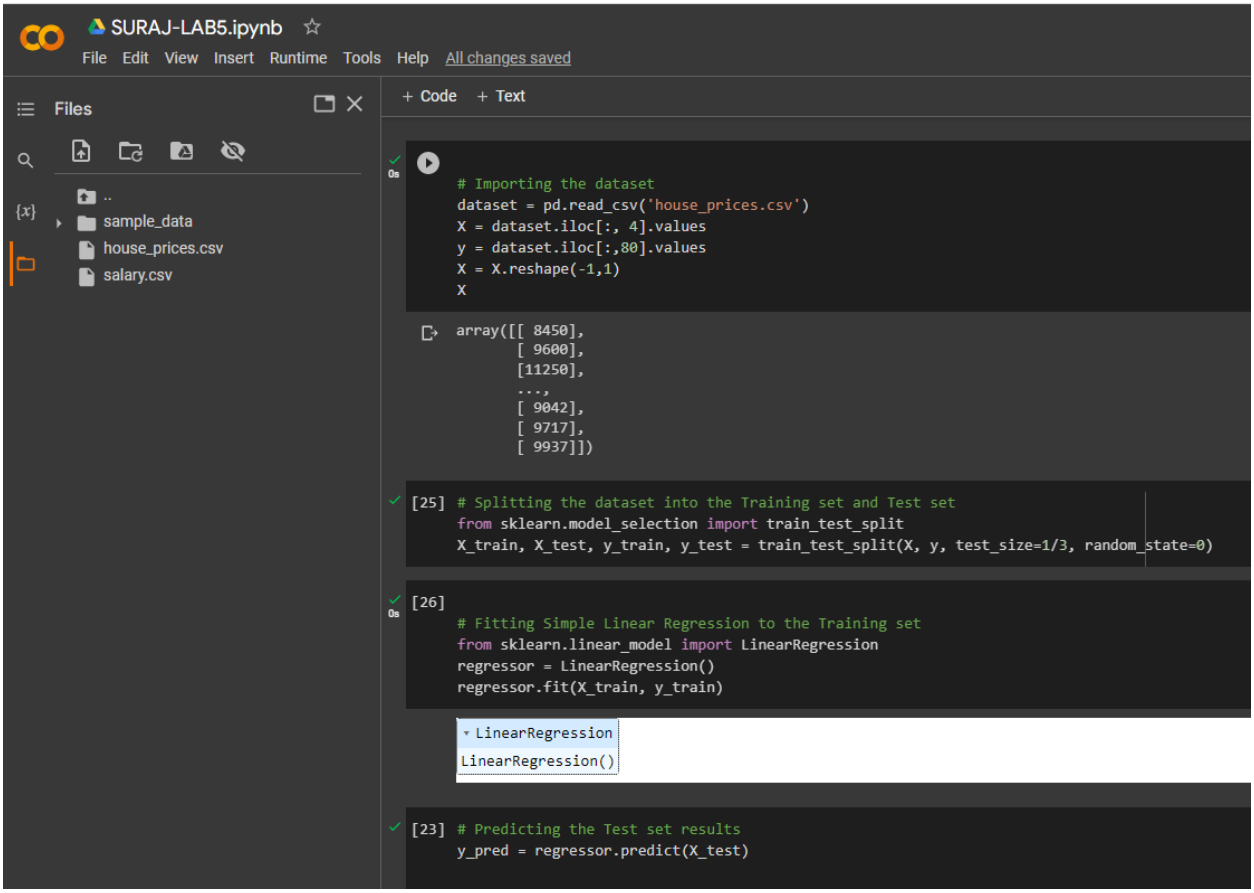


house_price.csv

	A	B	C	D	E	F	G	H
1	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape
2	1461	20	RH	80	11622	Pave	NA	Reg
3	1462	20	RL	81	14267	Pave	NA	IR1
4	1463	60	RL	74	13830	Pave	NA	IR1
5	1464	60	RL	78	9978	Pave	NA	IR1
6	1465	120	RL	43	5005	Pave	NA	IR1
7	1466	60	RL	75	10000	Pave	NA	IR1
8	1467	20	RL	NA	7980	Pave	NA	IR1
9	1468	60	RL	63	8402	Pave	NA	IR1
10	1469	20	RL	85	10176	Pave	NA	Reg
11	1470	20	RL	70	8400	Pave	NA	Reg
12	1471	120	RH	26	5858	Pave	NA	IR1
13	1472	160	RM	21	1680	Pave	NA	Reg
14	1473	160	RM	21	1680	Pave	NA	Reg
15	1474	160	RL	24	2280	Pave	NA	Reg
16	1475	120	RL	24	2280	Pave	NA	Reg
17	1476	60	RL	102	12858	Pave	NA	IR1
18	1477	20	RL	94	12883	Pave	NA	IR1
19	1478	20	RL	90	11520	Pave	NA	Reg
20	1479	20	RL	79	14122	Pave	NA	IR1
21	1480	20	RL	110	14300	Pave	NA	Reg
22	1481	60	RL	105	13650	Pave	NA	Reg
23	1482	120	RL	41	7132	Pave	NA	IR1


Drive
Search in Drive
BMSCE
New
My Drive
ML-LAB-MD
Priority
My Drive
Shared with me
Recent
Starred
Trash
Storage
16.47 GB of 50 GB used
Files
Name
buy.csv
car_evaluation.csv
climate.csv
enjoySportMD.csv
house_price.csv
salary.csv

Program:


SURAJ-LAB5.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Files
+ Code + Text
sample_data
house_prices.csv
salary.csv
# Importing the dataset
dataset = pd.read_csv('house_prices.csv')
X = dataset.iloc[:, 4].values
y = dataset.iloc[:,80].values
X = X.reshape(-1,1)
X
array([[8450],
[9600],
[11250],
...,
[9042],
[9717],
[9937]])
[25] # Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/3, random_state=0)
[26]
# Fitting Simple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
LinearRegression
LinearRegression()
[23] # Predicting the Test set results
y_pred = regressor.predict(X_test)

Output:

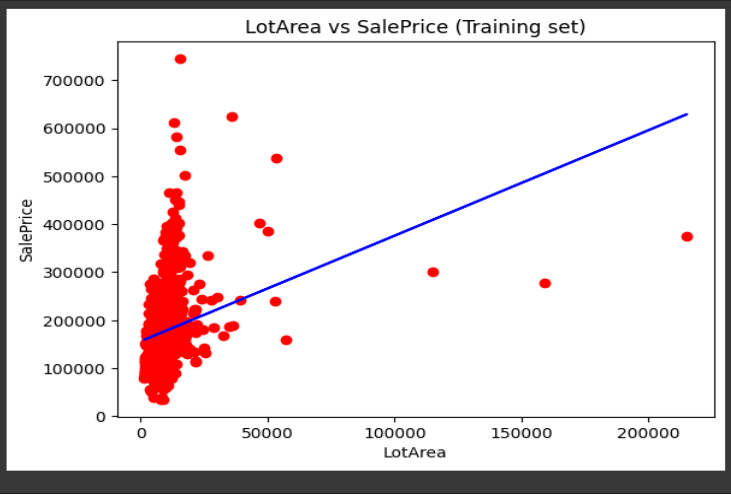
SURAJ-LAB5.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Files

+ Code + Text

```
4
# Visualizing the Training set results
viz_train = plt
viz_train.scatter(X_train, y_train, color='red')
viz_train.plot(X_train, regressor.predict(X_train), color='blue')
viz_train.title('LotArea vs SalePrice (Training set)')
viz_train.xlabel('LotArea')
viz_train.ylabel('SalePrice')
viz_train.show()
```



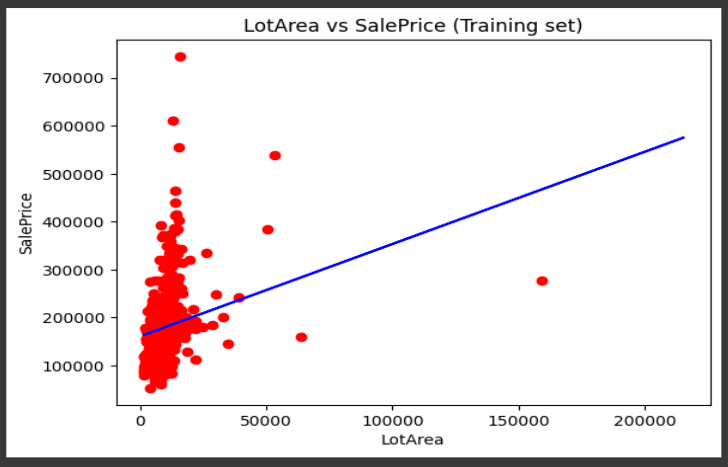
SURAJ-LAB5.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Files

+ Code + Text

```
# Visualizing the Test set results
viz_test = plt
viz_test.scatter(X_test, y_test, color='red')
viz_train.plot(X_train, regressor.predict(X_train), color='blue')
viz_train.title('LotArea vs SalePrice (Training set)')
viz_train.xlabel('LotArea')
viz_train.ylabel('SalePrice')
viz_train.show()
```



Date: 29/5/23

Program 6:

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

Dataset:Climate.csv

Algorithm:

The formula for Bayes' theorem is given as:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

We need to follow the below steps:

1. Convert the given dataset into frequency tables.
2. Generate Likelihood table by finding the probabilities of given features.
3. Now, use Bayes theorem to calculate the posterior probability.

- Data Pre-processing step
- Fitting Naive Bayes to the Training set
- Predicting the test result
- Test accuracy of the result(Creation of Confusion matrix)
- Visualizing the test set result.

8/5/23

PROGRAM-6

Implement naive-bayes classifier

→ Supervised learning algorithm used for classification problem, where attributes are independent in predicting final output.

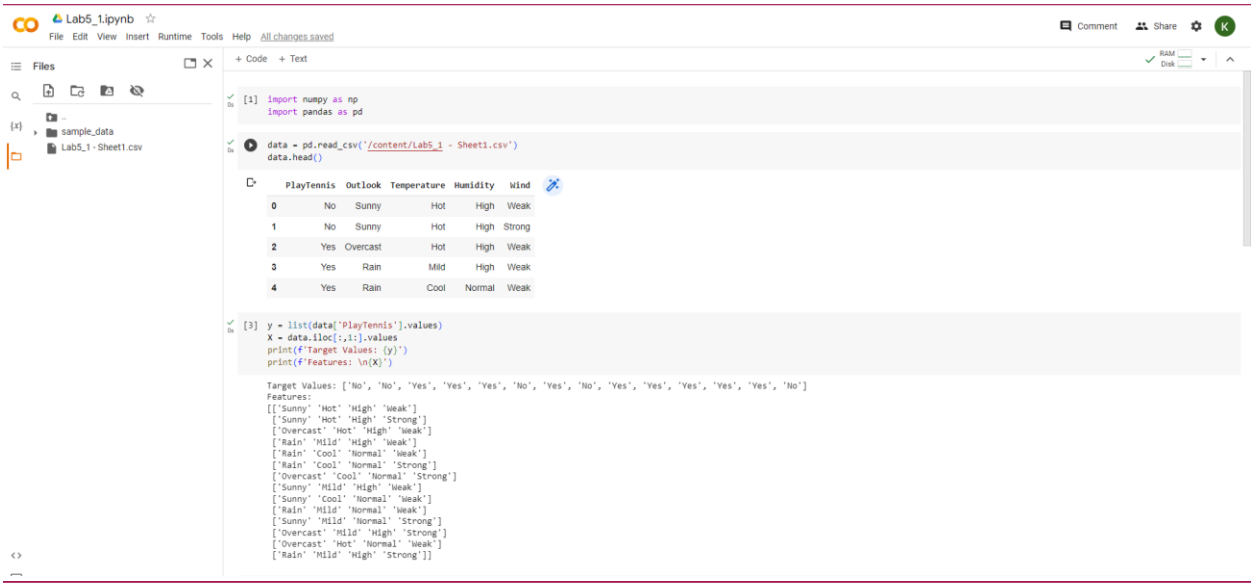
Algorithm :-

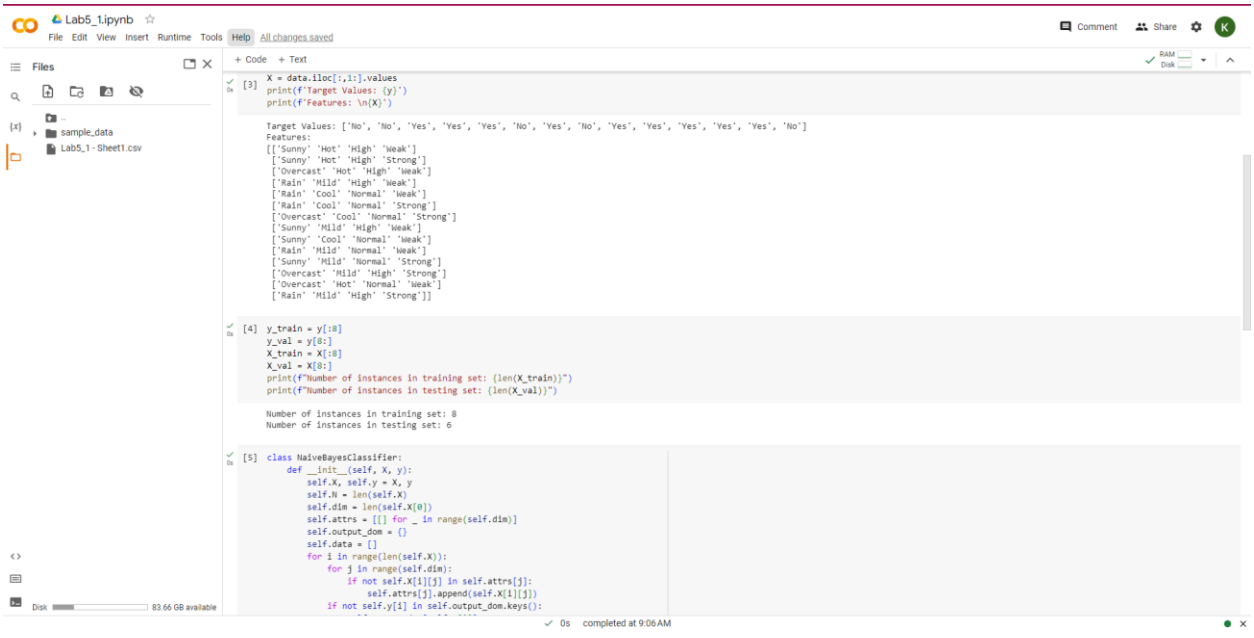
$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

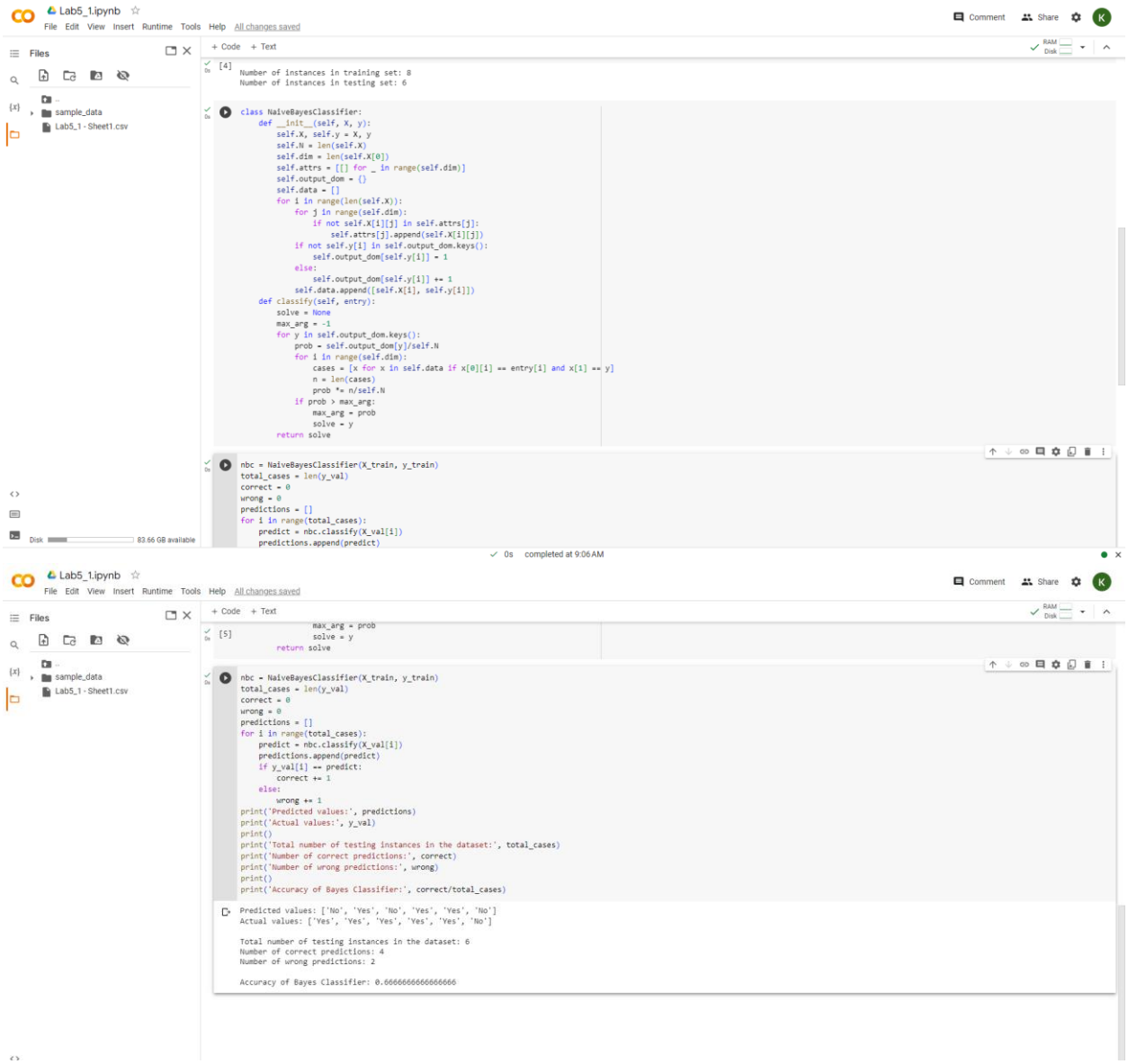
* convert given dataset into frequency table

* generate likelihood table by finding probabilities of given features.

* use bayes theorem to calculate posterior probability.
  * data processing step
  * fitting naive bayes to training set
  * predicting test result.
  * test accuracy of result
  * visualizing test result.


Lab5_1.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
Files
sample_data
Lab5_1 - Sheet1.csv
+ Code + Text
[1] import numpy as np
import pandas as pd
data = pd.read_csv('/content/Lab5_1 - Sheet1.csv')
data.head()
PlayTennis Outlook Temperature Humidity Wind
0 No Sunny Hot High Weak
1 No Sunny Hot High Strong
2 Yes Overcast Hot High Weak
3 Yes Rain Mild High Weak
4 Yes Rain Cool Normal Weak
[3] y = list(data['PlayTennis'].values)
X = data.iloc[:,1:].values
print(f'Target Values: {y}')
print(f'Features: \n{X}')
Target Values: ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']
Features:
[['Sunny' 'Hot' 'High' 'Weak']
['Sunny' 'Hot' 'High' 'Strong']
['Overcast' 'Hot' 'High' 'Weak']
['Rain' 'Mild' 'High' 'Weak']
['Rain' 'Cool' 'Normal' 'Weak']
['Rain' 'Cool' 'Normal' 'Strong']
['Overcast' 'Cool' 'Normal' 'Strong']
['Sunny' 'Mild' 'High' 'Weak']
['Sunny' 'Cool' 'Normal' 'Weak']
['Rain' 'Mild' 'Normal' 'Weak']
['Sunny' 'Mild' 'Normal' 'Strong']
['Overcast' 'Mild' 'High' 'Strong']
['Overcast' 'Hot' 'Normal' 'Weak']
['Rain' 'Mild' 'High' 'Strong']]


Lab5_1.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
Files
sample_data
Lab5_1 - Sheet1.csv
+ Code + Text
[3] X = data.iloc[:,1:].values
print(f'Target Values: {y}')
print(f'Features: \n{X}')
Target Values: ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']
Features:
[['Sunny' 'Hot' 'High' 'Weak']
['Sunny' 'Hot' 'High' 'Strong']
['Overcast' 'Hot' 'High' 'Weak']
['Rain' 'Mild' 'High' 'Weak']
['Rain' 'Cool' 'Normal' 'Weak']
['Rain' 'Cool' 'Normal' 'Strong']
['Overcast' 'Cool' 'Normal' 'Strong']
['Sunny' 'Mild' 'High' 'Weak']
['Sunny' 'Cool' 'Normal' 'Weak']
['Rain' 'Mild' 'Normal' 'Weak']
['Sunny' 'Mild' 'Normal' 'Strong']
['Overcast' 'Mild' 'High' 'Strong']
['Overcast' 'Hot' 'Normal' 'Weak']
['Rain' 'Mild' 'High' 'Strong']]
[4] y_train = y[:8]
y_val = y[8:]
X_train = X[:8]
X_val = X[8:]
print(f"Number of instances in training set: {len(X_train)}")
print(f"Number of instances in testing set: {len(X_val)}")
Number of instances in training set: 8
Number of instances in testing set: 6
[5] class NaiveBayesClassifier:
def __init__(self, X, y):
self.X, self.y = X, y
self.N = len(self.X)
self.dim = len(self.X[0])
self.attrs = [[] for _ in range(self.dim)]
self.output_dom = {}
self.data = []
for i in range(len(self.X)):
for j in range(self.dim):
if not self.X[i][j] in self.attrs[j]:
self.attrs[j].append(self.X[i][j])
if not self.y[i] in self.output_dom.keys():
Disk 83.66 GB available
0s completed at 9:06 AM


Lab5_1.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
Files
+ Code + Text
sample_data
Lab5_1 - Sheet1.csv
Disk 83.66 GB available
[4]
Number of instances in training set: 8
Number of instances in testing set: 6
class NaiveBayesClassifier:
def __init__(self, X, y):
self.X, self.y = X, y
self.N = len(self.X)
self.dim = len(self.X[0])
self.attrs = [[] for _ in range(self.dim)]
self.output_dom = {}
self.data = []
for i in range(len(self.X)):
for j in range(self.dim):
if not self.X[i][j] in self.attrs[j]:
self.attrs[j].append(self.X[i][j])
if not self.y[i] in self.output_dom.keys():
self.output_dom[self.y[i]] = 1
else:
self.output_dom[self.y[i]] += 1
self.data.append([self.X[i], self.y[i]])
def classify(self, entry):
solve = None
max_arg = -1
for y in self.output_dom.keys():
prob = self.output_dom[y]/self.N
for i in range(self.dim):
cases = [x for x in self.data if x[0][i] == entry[i] and x[1] == y]
n = len(cases)
prob *= n/self.N
if prob > max_arg:
max_arg = prob
solve = y
return solve
nbc = NaiveBayesClassifier(X_train, y_train)
total_cases = len(y_val)
correct = 0
wrong = 0
predictions = []
for i in range(total_cases):
predict = nbc.classify(X_val[i])
predictions.append(predict)
0s completed at 9:06 AM

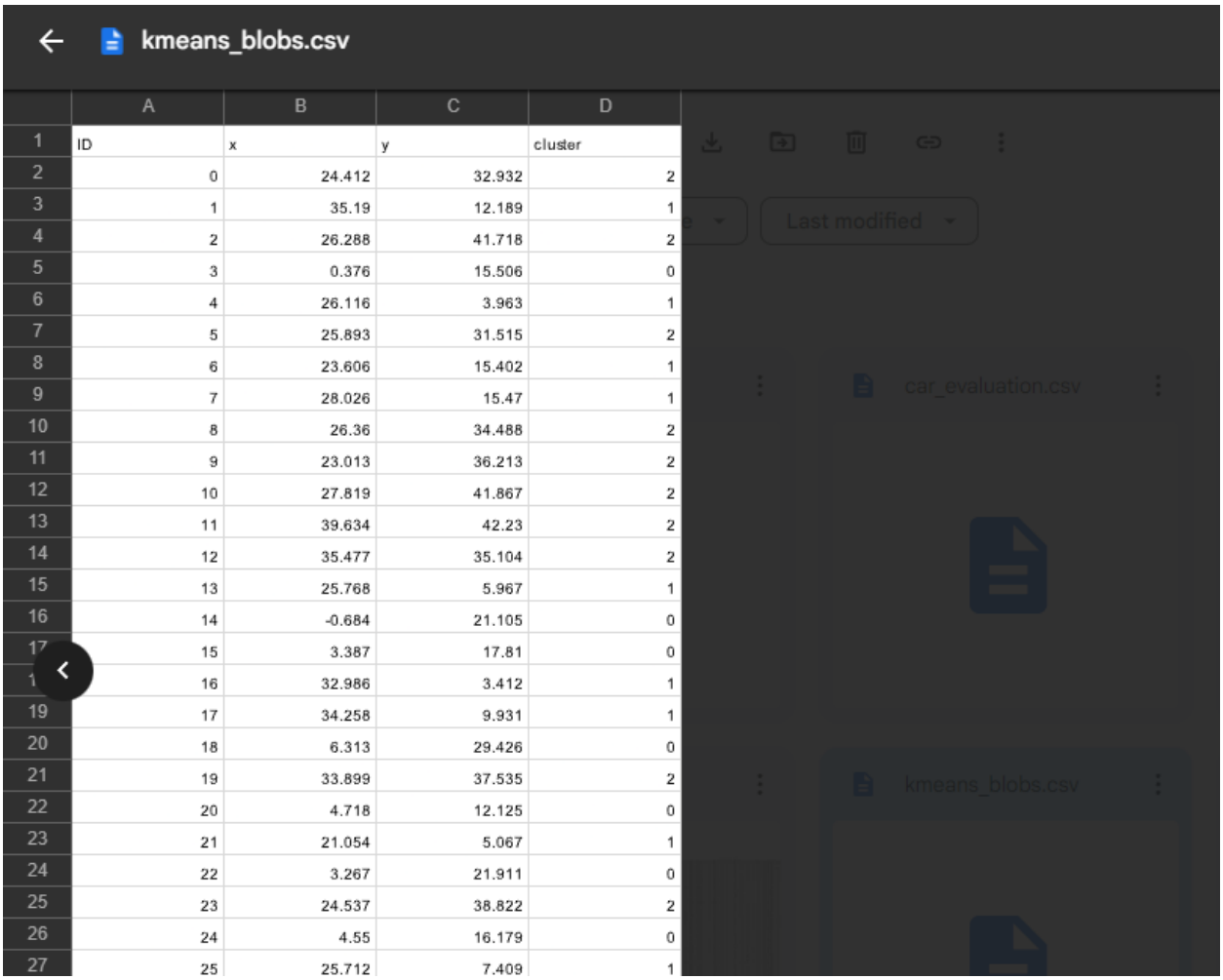
Lab5_1.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
Files
+ Code + Text
sample_data
Lab5_1 - Sheet1.csv
[5]
max_arg = prob
solve = y
return solve
nbc = NaiveBayesClassifier(X_train, y_train)
total_cases = len(y_val)
correct = 0
wrong = 0
predictions = []
for i in range(total_cases):
predict = nbc.classify(X_val[i])
predictions.append(predict)
if y_val[i] == predict:
correct += 1
else:
wrong += 1
print('Predicted values:', predictions)
print('Actual values:', y_val)
print()
print('Total number of testing instances in the dataset:', total_cases)
print('Number of correct predictions:', correct)
print('Number of wrong predictions:', wrong)
print()
print('Accuracy of Bayes Classifier:', correct/total_cases)
Predicted values: ['No', 'Yes', 'No', 'Yes', 'Yes', 'No']
Actual values: ['Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']
Total number of testing instances in the dataset: 6
Number of correct predictions: 4
Number of wrong predictions: 2
Accuracy of Bayes Classifier: 0.6666666666666666

Date: 6.05.2023

Program 7:

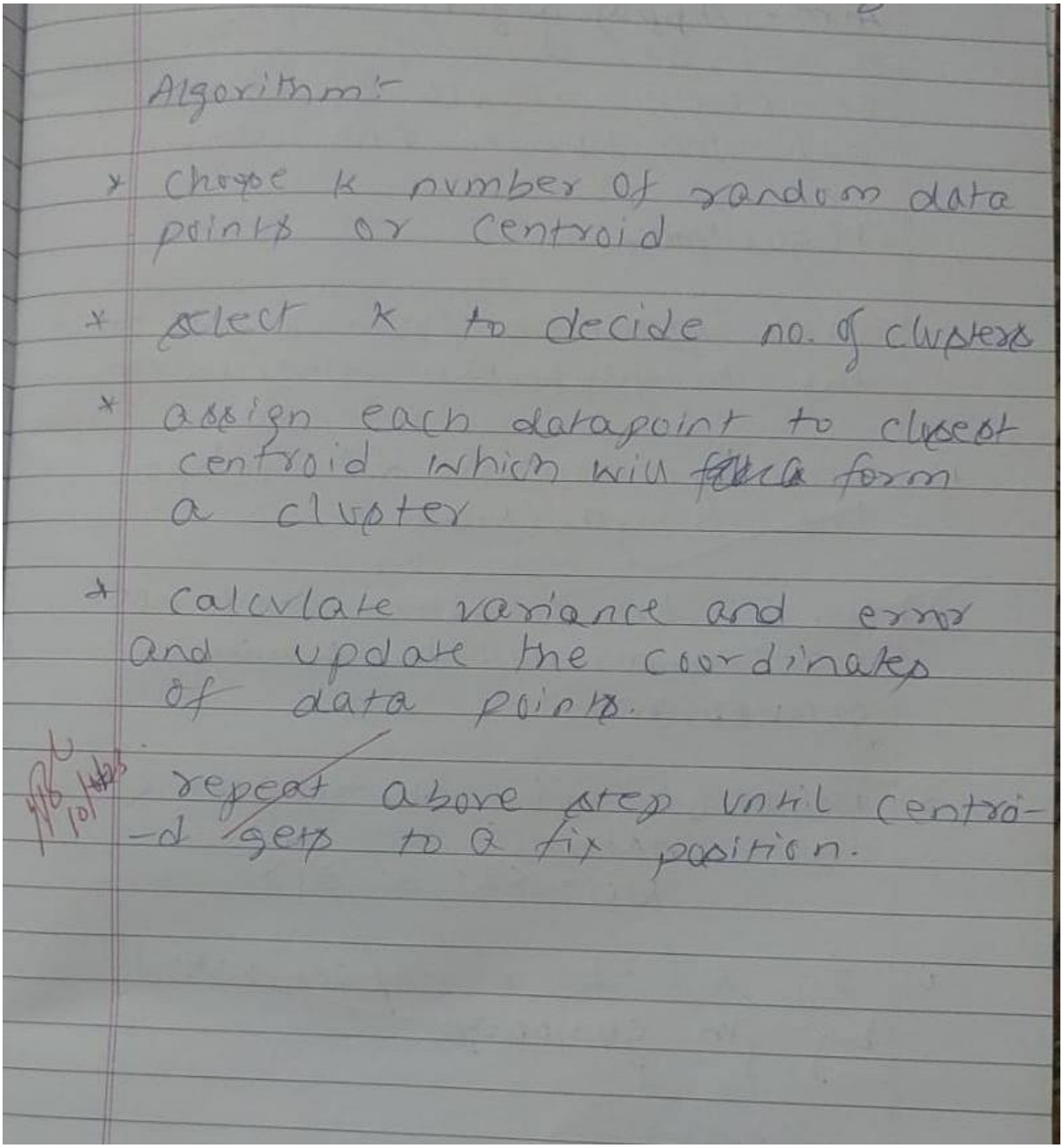
Apply k-Means algorithm to cluster a set of data stored in a .CSV file.

Dataset : Blobs.csv



	A	B	C	D
1	ID	x	y	cluster
2	0	24.412	32.932	2
3	1	35.19	12.189	1
4	2	26.288	41.718	2
5	3	0.376	15.506	0
6	4	26.116	3.963	1
7	5	25.893	31.515	2
8	6	23.606	15.402	1
9	7	28.026	15.47	1
10	8	26.36	34.488	2
11	9	23.013	36.213	2
12	10	27.819	41.867	2
13	11	39.634	42.23	2
14	12	35.477	35.104	2
15	13	25.768	5.967	1
16	14	-0.684	21.105	0
17	15	3.387	17.81	0
18	16	32.986	3.412	1
19	17	34.258	9.931	1
20	18	6.313	29.426	0
21	19	33.899	37.535	2
22	20	4.718	12.125	0
23	21	21.054	5.067	1
24	22	3.267	21.911	0
25	23	24.537	38.822	2
26	24	4.55	16.179	0
27	25	25.712	7.409	1

Algorithm:



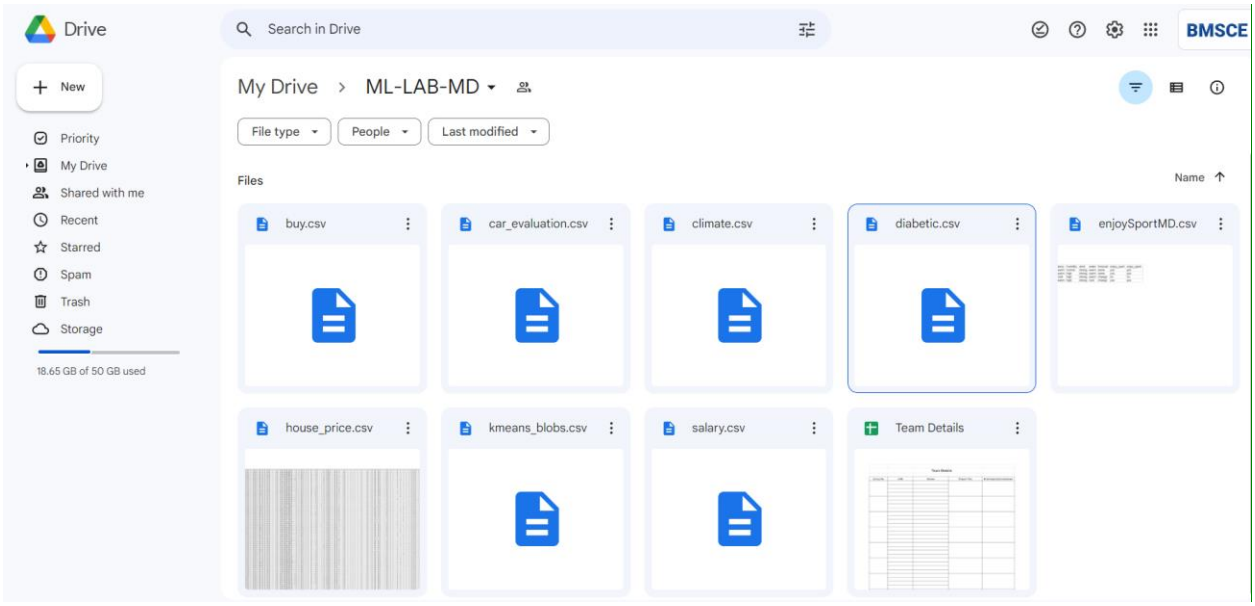
Algorithm:

* Choose k number of random data points or centroid
* Select k to decide no. of clusters
* assign each datapoint to closest centroid which will form a cluster
* Calculate variance and error and update the coordinates of data points.
* repeat above step until centroid gets to a fix position.

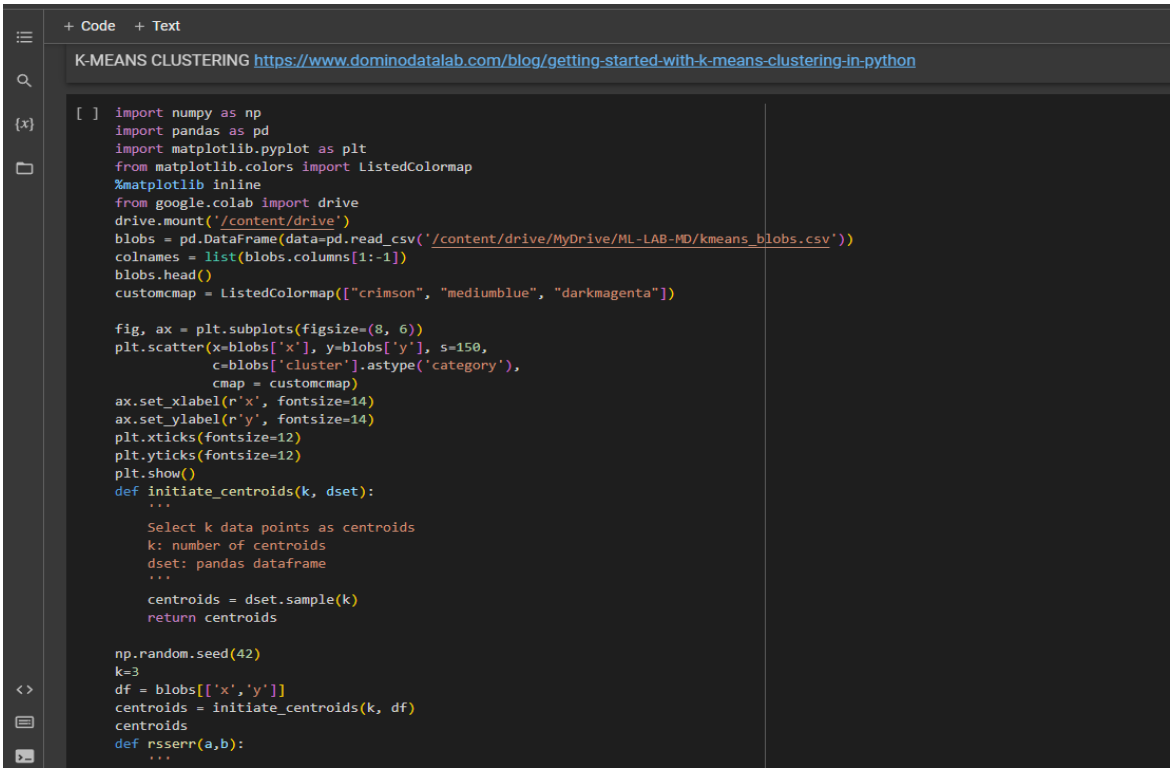
Uploading the csv file:

kmeans_blobs.csv

	A	B	C	D
1	ID	x	y	cluster
2	0	24.412	32.932	2
3	1	35.19	12.189	1
4	2	26.288	41.718	2
5	3	0.376	15.506	0
6	4	26.116	3.963	1
7	5	25.893	31.515	2
8	6	23.606	15.402	1
9	7	28.026	15.47	1
10	8	26.36	34.488	2
11	9	23.013	36.213	2
12	10	27.819	41.867	2
13	11	39.634	42.23	2
14	12	35.477	35.104	2
15	13	25.768	5.967	1
16	14	-0.684	21.105	0
17	15	3.387	17.81	0
18	16	32.986	3.412	1
19	17	34.258	9.931	1
20	18	6.313	29.426	0
21	19	33.899	37.535	2
22	20	4.718	12.125	0
23	21	21.054	5.067	1
24	22	3.267	21.911	0
25	23	24.537	38.822	2
26	24	4.55	16.179	0
27	25	25.712	7.409	1



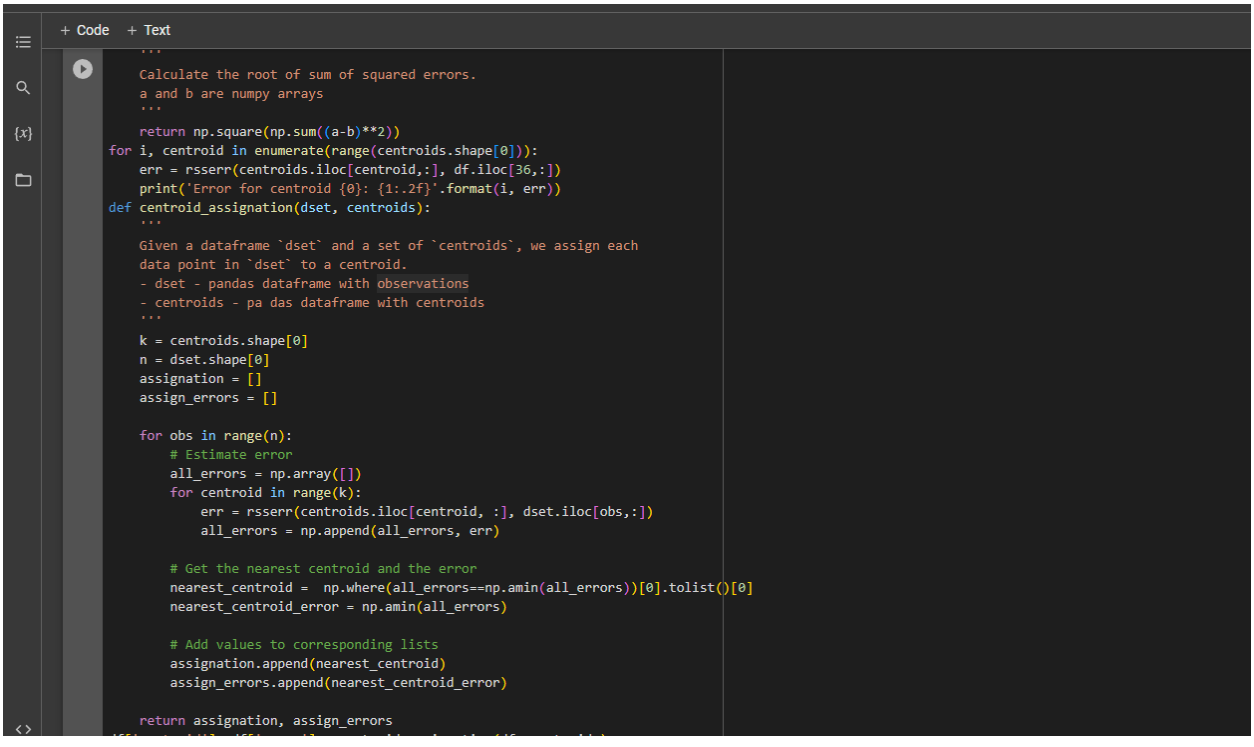
Program:



+ Code + Text

K-MEANS CLUSTERING https://www.dominodatalab.com/blog/getting-started-with-k-means-clustering-in-python

```
[ ] import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    from matplotlib.colors import ListedColormap
    %matplotlib inline
    from google.colab import drive
    drive.mount('/content/drive')
    blobs = pd.DataFrame(data=pd.read_csv('/content/drive/MyDrive/ML-LAB-MD/kmeans_blobs.csv'))
    colnames = list(blobs.columns[1:-1])
    blobs.head()
    customcmap = ListedColormap(["crimson", "mediumblue", "darkmagenta"])

    fig, ax = plt.subplots(figsize=(8, 6))
    plt.scatter(x=blobs['x'], y=blobs['y'], s=150,
                c=blobs['cluster'].astype('category'),
                cmap = customcmap)
    ax.set_xlabel(r'x', fontsize=14)
    ax.set_ylabel(r'y', fontsize=14)
    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)
    plt.show()
    def initiate_centroids(k, dset):
        '''
        Select k data points as centroids
        k: number of centroids
        dset: pandas dataframe
        '''
        centroids = dset.sample(k)
        return centroids

    np.random.seed(42)
    k=3
    df = blobs[['x','y']]
    centroids = initiate_centroids(k, df)
    centroids
    def rsserr(a,b):
        '''
```

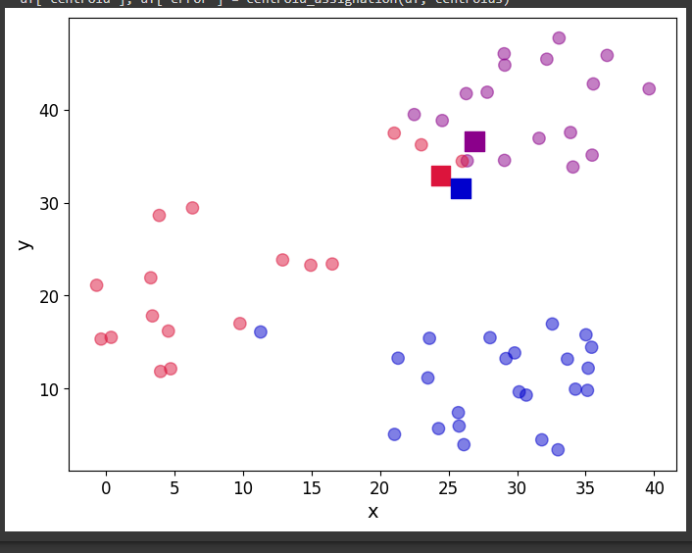


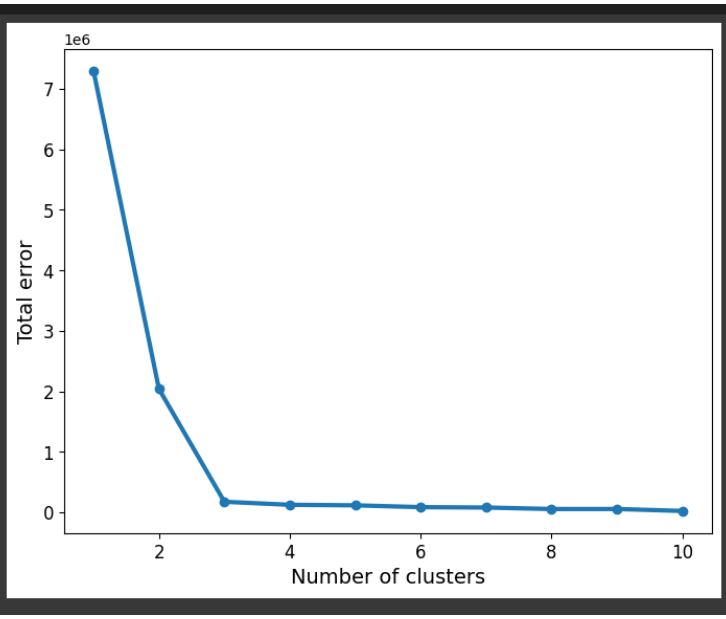
+ Code + Text

```
        '''
        Calculate the root of sum of squared errors.
        a and b are numpy arrays
        '''
        return np.square(np.sum((a-b)**2))
    for i, centroid in enumerate(range(centroids.shape[0])):
        err = rsserr(centroids.iloc[centroid,:], df.iloc[36,:])
        print('Error for centroid {0}: {1:.2f}'.format(i, err))
    def centroid_assignation(dset, centroids):
        '''
        Given a dataframe `dset` and a set of `centroids`, we assign each
        data point in `dset` to a centroid.
        - dset - pandas dataframe with observations
        - centroids - pa das dataframe with centroids
        '''
        k = centroids.shape[0]
        n = dset.shape[0]
        assignation = []
        assign_errors = []

        for obs in range(n):
            # Estimate error
            all_errors = np.array([])
            for centroid in range(k):
                err = rsserr(centroids.iloc[centroid, :], dset.iloc[obs,:])
                all_errors = np.append(all_errors, err)

            # Get the nearest centroid and the error
            nearest_centroid =  np.where(all_errors==np.amin(all_errors))[0].tolist()[0]
            nearest_centroid_error = np.amin(all_errors)

            # Add values to corresponding lists
            assignation.append(nearest_centroid)
            assign_errors.append(nearest_centroid_error)

        return assignation, assign_errors
```

Output:

```
<ipython-input-1-77b43ffa398d>:73: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['centroid'], df['error'] = centroid_assignation(df, centroids)
```





Date: 10.05.2023

Program 8:

Apply EM algorithm to cluster a set of data stored in a .CSV file. Compare the results of k-Means algorithm and EM algorithm..

Dataset : Diabetes.csv

	A	B	C	D	E	F	G	H	I
1	num_preg	glucose_conc	diastolic_bp	thickness	insulin	bmi	diab_pred	age	diabetes
2	6	148	72	35	0	33.6	0.627	50	1
3	1	85	66	29	0	26.6	0.351	31	0
4	8	183	64	0	0	23.3	0.672	32	1
5	1	89	66	23	94	28.1	0.167	21	0
6	0	137	40	35	168	43.1	2.288	33	1
7	5	116	74	0	0	25.6	0.201	30	0
8	3	78	50	32	88	31	0.248	26	1
9	10	115	0	0	0	35.3	0.134	29	0
10	2	197	70	45	543	30.5	0.158	53	1
11	8	125	96	0	0	0	0.232	54	1
12	4	110	92	0	0	37.6	0.191	30	0
13	10	168	74	0	0	38	0.537	34	1
14	10	139	80	0	0	27.1	1.441	57	0
15	1	189	60	23	846	30.1	0.398	59	1
16	5	166	72	19	175	25.8	0.587	51	1
17	7	100	0	0	0	30	0.484	32	1
18	0	118	84	47	230	45.8	0.551	31	1
19	7	107	74	0	0	29.6	0.254	31	1
20	1	103	30	38	83	43.3	0.183	33	0
21	1	115	70	30	96	34.6	0.529	32	1
22	3	126	88	41	235	39.3	0.704	27	0
23	8	99	84	0	0	35.4	0.388	50	0
24	7	196	90	0	0	39.8	0.451	41	1
25	9	119	80	35	0	29	0.263	29	1
26	11	143	94	33	146	36.6	0.254	51	1
27	10	125	70	26	115	31.1	0.205	41	1

Algorithm:

a set of data in a csv file. Compare results of k-means algorithm with EM algorithm

Algorithm:-

Given instances from X generated by k gaussian distribution.

1) unknown means $\langle u_1, u_2 \rangle$ of the k gaussian.

2) Dont know which instance was generated by which gaussian

3) Think of full description of each instance

$$y_{ij} = \langle x_i, z_i, z_j \rangle$$

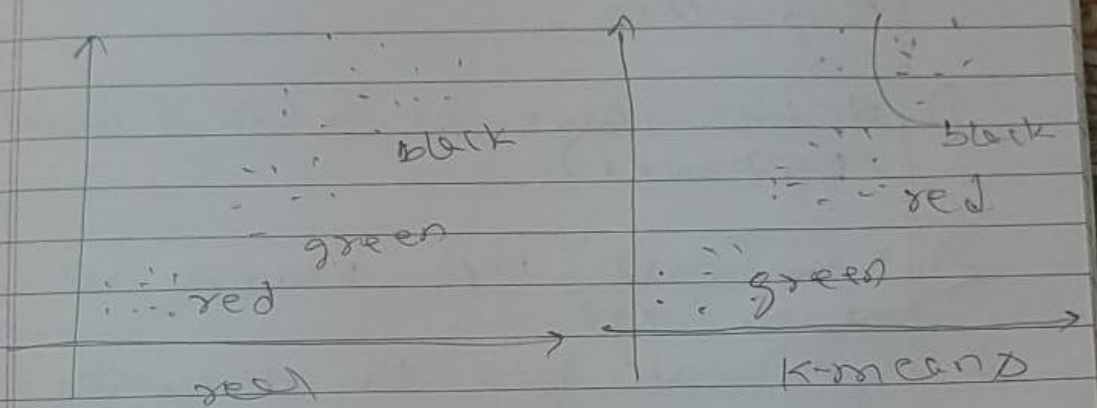
4) z_{ij} is 1 if x_i is generated by jth gaussian

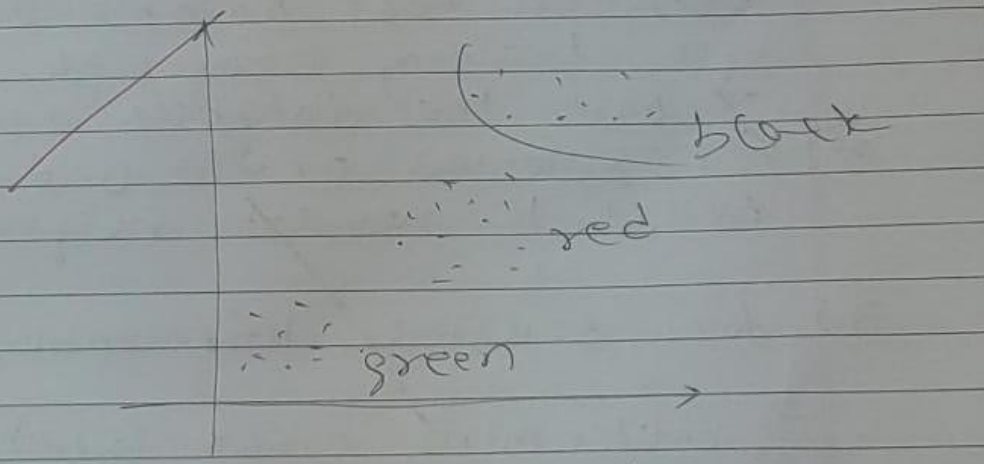
→ x_i is observable
→ z_{ij} is unobservable

E step:- calculate expected values and [z i] of each hidden variable zij assuming the current hypothesis.

M step:- calculate the new maximum likelihood.

OUTPUT:-

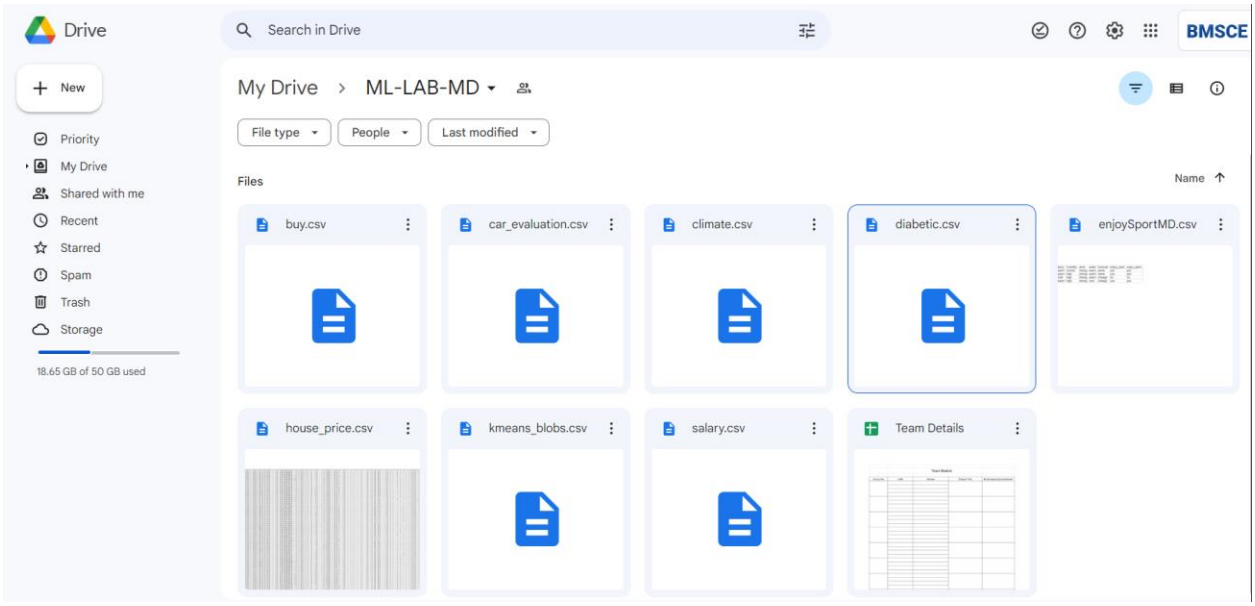




GMM classifier

Uploading the csv file:

	A	B	C	D	E	F	G	H	I
1	num_preg	glucose_conc	diastolic_bp	thickness	insulin	bmi	diab_pred	age	diabetes
2	6	148	72	35	0	33.6	0.627	50	1
3	1	85	66	29	0	26.6	0.351	31	0
4	8	183	64	0	0	23.3	0.672	32	1
5	1	89	66	23	94	28.1	0.167	21	0
6	0	137	40	35	168	43.1	2.288	33	1
7	5	116	74	0	0	25.6	0.201	30	0
8	3	78	50	32	88	31	0.248	26	1
9	10	115	0	0	0	35.3	0.134	29	0
10	2	197	70	45	543	30.5	0.158	53	1
11	8	125	96	0	0	0	0.232	54	1
12	4	110	92	0	0	37.6	0.191	30	0
13	10	168	74	0	0	38	0.537	34	1
14	10	139	80	0	0	27.1	1.441	57	0
15	1	189	60	23	846	30.1	0.398	59	1
16	5	166	72	19	175	25.8	0.587	51	1
17	7	100	0	0	0	30	0.484	32	1
18	0	118	84	47	230	45.8	0.551	31	1
19	7	107	74	0	0	29.6	0.254	31	1
20	1	103	30	38	83	43.3	0.183	33	0
21	1	115	70	30	96	34.6	0.529	32	1
22	3	126	88	41	235	39.3	0.704	27	0
23	8	99	84	0	0	35.4	0.388	50	0
24	7	196	90	0	0	39.8	0.451	41	1
25	9	119	80	35	0	29	0.263	29	1
26	11	143	94	33	146	36.6	0.254	51	1
27	10	125	70	26	115	31.1	0.205	41	1



Program:

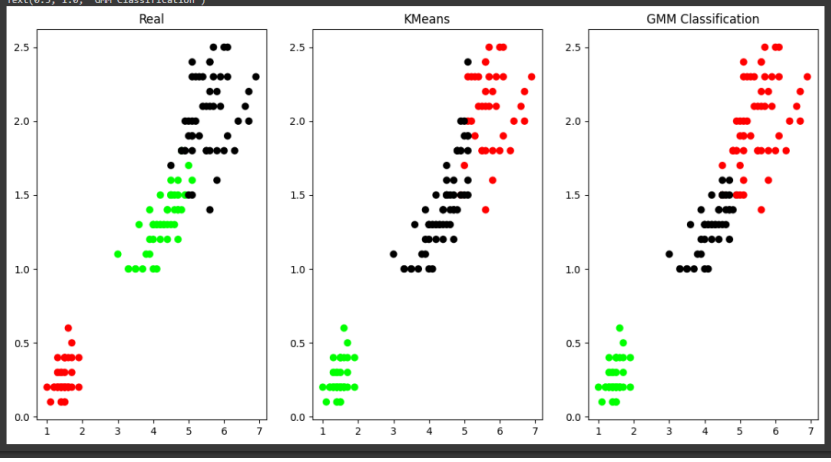
```
import numpy as np
xs = np.array([(5,5), (9,1), (8,2), (4,6), (7,3)])
thetas = np.array([[0.6, 0.4], [0.5, 0.5]])

tol = 0.01
max_iter = 100

ll_old = 0
for i in range(max_iter):
    ws_A = []
    ws_B = []

    vs_A = []
    vs_B = []

    ll_new = 0

    # E-step: calculate probability distributions over possible completions
    for x in xs:

        # multinomial (binomial) log likelihood
        ll_A = np.sum([x*np.log(thetas[0])])
        ll_B = np.sum([x*np.log(thetas[1])])

        # [EQN 1]
        denom = np.exp(ll_A) + np.exp(ll_B)
        w_A = np.exp(ll_A)/denom
        w_B = np.exp(ll_B)/denom

        ws_A.append(w_A)
        ws_B.append(w_B)

        # used for calculating theta
        vs_A.append(np.dot(w_A, x))
        vs_B.append(np.dot(w_B, x))

        # update complete log likelihood
        ll_new += w_A * ll_A + w_B * ll_B

    # M-step: update values for parameters given current distribution
    # [EQN 2]
    thetas[0] = np.sum(vs_A, 0)/np.sum(vs_A)
    thetas[1] = np.sum(vs_B, 0)/np.sum(vs_B)
    # print distribution of z for each x and current parameter estimate

    print ("Iteration: %d" % (i+1))
    print ("theta_A = %.2f, theta_B = %.2f, ll = %.2f" % (thetas[0,0], thetas[1,0], ll_new))

    if np.abs(ll_new - ll_old) < tol:
```

```
from sklearn.cluster import KMeans
from sklearn import preprocessing
from sklearn.mixture import GaussianMixture
from sklearn.datasets import load_iris
import sklearn.metrics as sm
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
dataset=load_iris()
# print(dataset)
X=pd.DataFrame(dataset.data)
X.columns=['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y=pd.DataFrame(dataset.target)
y.columns=['Targets']
# print(X)
plt.figure(figsize=(14,7))
colormap=np.array(['red','lime','black'])

# REAL PLOT
plt.subplot(1,3,1)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y.Targets],s=40)
plt.title('Real')

# K-PLOT
plt.subplot(1,3,2)
model=KMeans(n_clusters=3)
model.fit(X)
predY=np.choose(model.labels_,[0,1,2]).astype(np.int64)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[predY],s=40)
plt.title('KMeans')

# GMM PLOT
scaler=preprocessing.StandardScaler()
scaler.fit(X)
xsa=scaler.transform(X)
xs=pd.DataFrame(xsa,columns=X.columns)
gmm=GaussianMixture(n_components=3)
gmm.fit(xs)
y_cluster_gmm=gmm.predict(xs)
plt.subplot(1,3,3)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y_cluster_gmm],s=40)
plt.title('GMM Classification')
```

Output:

```
Iteration: 1
theta_A = 0.71, theta_B = 0.58, ll = -32.69
Iteration: 2
theta_A = 0.75, theta_B = 0.57, ll = -31.26
Iteration: 3
theta_A = 0.77, theta_B = 0.55, ll = -30.76
Iteration: 4
theta_A = 0.78, theta_B = 0.53, ll = -30.33
Iteration: 5
theta_A = 0.79, theta_B = 0.53, ll = -30.07
Iteration: 6
theta_A = 0.79, theta_B = 0.52, ll = -29.95
Iteration: 7
theta_A = 0.80, theta_B = 0.52, ll = -29.90
Iteration: 8
theta_A = 0.80, theta_B = 0.52, ll = -29.88
Iteration: 9
theta_A = 0.80, theta_B = 0.52, ll = -29.87
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
Text(0.5, 1.0, 'GMM Classification')
```

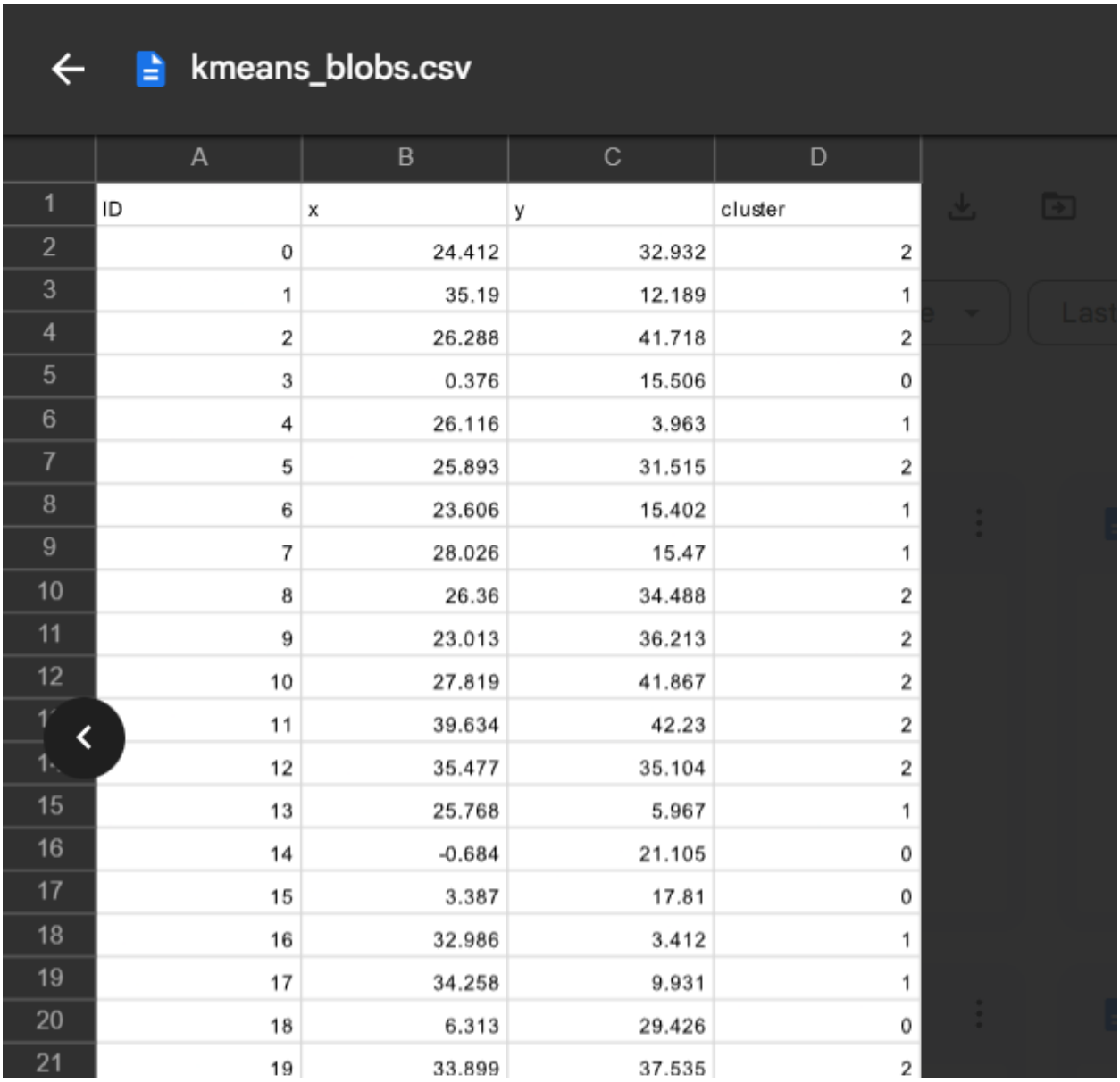


Date: 17.05.2023

Program 9:

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

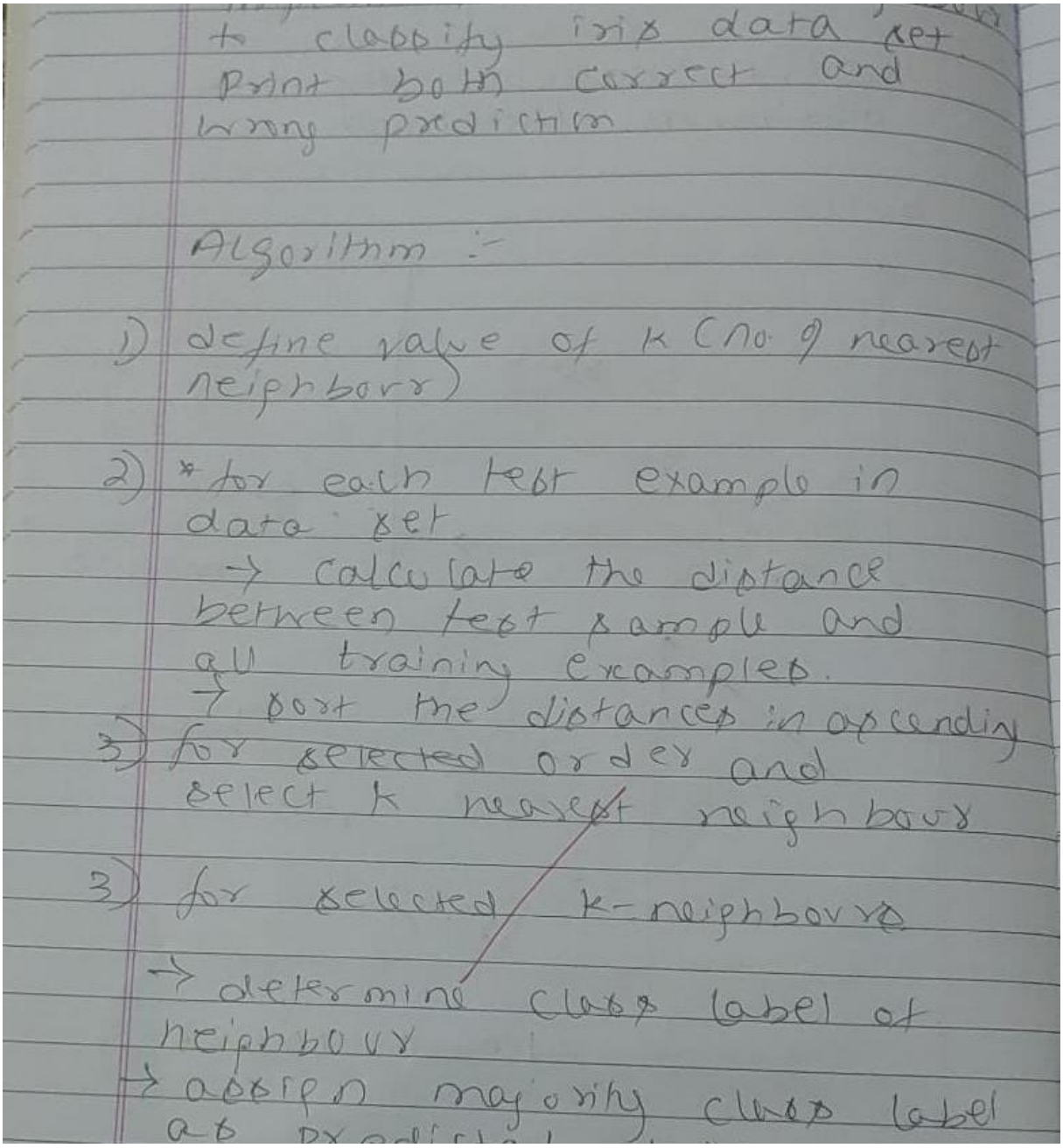
Dataset :kmeans_blobs.csv



kmeans_blobs.csv

	A	B	C	D
1	ID	x	y	cluster
2	0	24.412	32.932	2
3	1	35.19	12.189	1
4	2	26.288	41.718	2
5	3	0.376	15.506	0
6	4	26.116	3.963	1
7	5	25.893	31.515	2
8	6	23.606	15.402	1
9	7	28.026	15.47	1
10	8	26.36	34.488	2
11	9	23.013	36.213	2
12	10	27.819	41.867	2
13	11	39.634	42.23	2
14	12	35.477	35.104	2
15	13	25.768	5.967	1
16	14	-0.684	21.105	0
17	15	3.387	17.81	0
18	16	32.986	3.412	1
19	17	34.258	9.931	1
20	18	6.313	29.426	0
21	19	33.899	37.535	2

Algorithm:



to classify iris data set
Print both correct and
wrong prediction

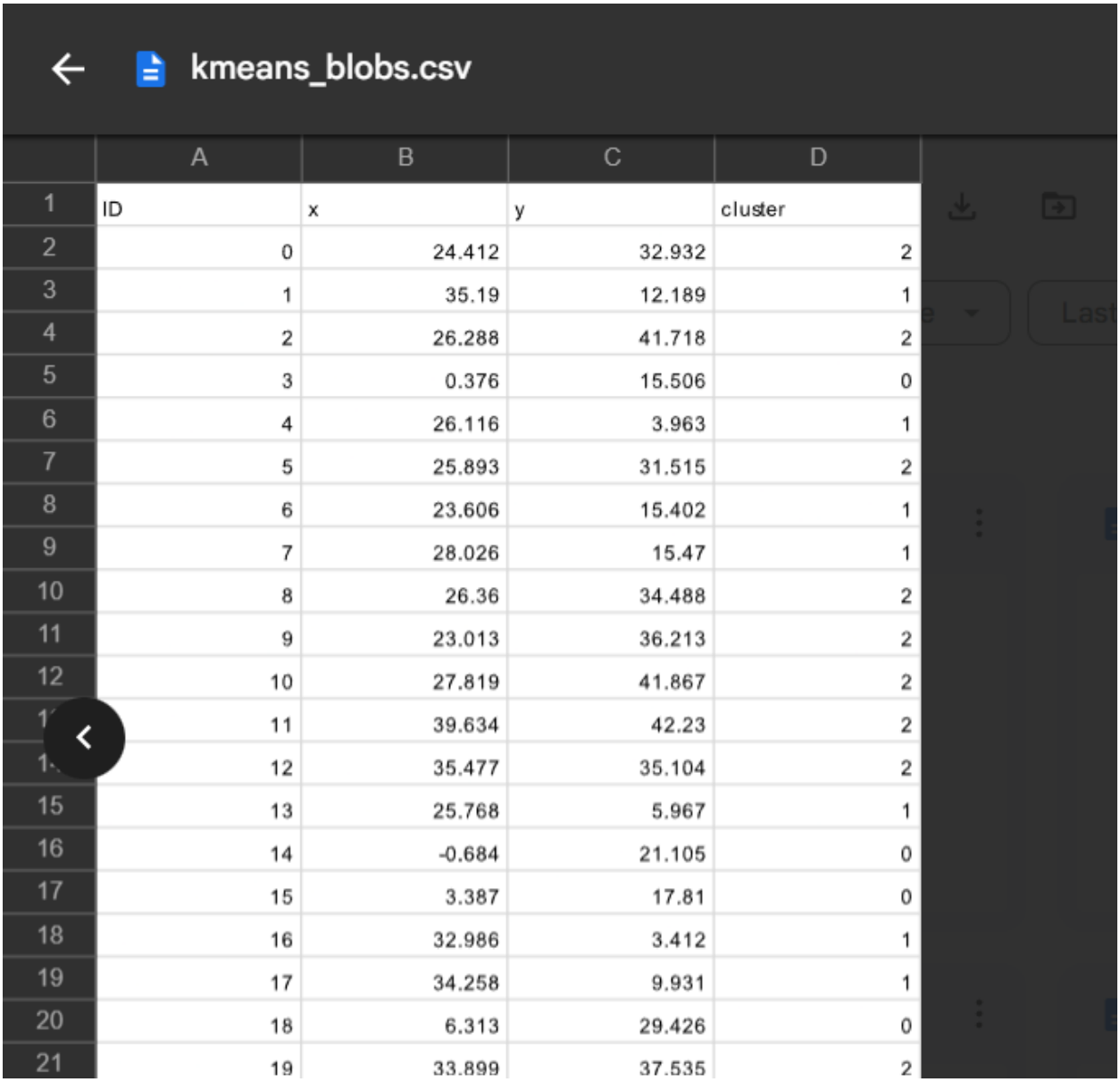
Algorithm :-

1) define value of k (no. of nearest neighbors)

2) * for each test example in data set
   → calculate the distance between test sample and all training examples.
   → sort the distances in ascending
3) for selected order and select k nearest neighbors

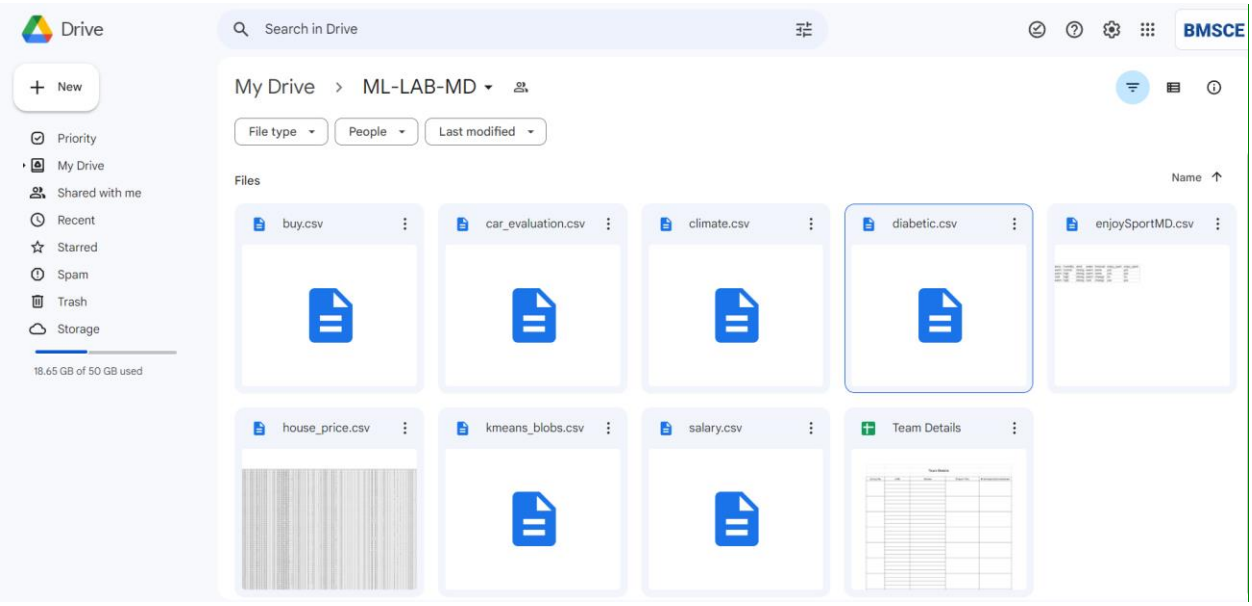
3) for selected k-neighbors
   → determine class label of neighbour
   → assign majority class label as

Uploading the csv file:



kmeans_blobs.csv

	A	B	C	D
1	ID	x	y	cluster
2	0	24.412	32.932	2
3	1	35.19	12.189	1
4	2	26.288	41.718	2
5	3	0.376	15.506	0
6	4	26.116	3.963	1
7	5	25.893	31.515	2
8	6	23.606	15.402	1
9	7	28.026	15.47	1
10	8	26.36	34.488	2
11	9	23.013	36.213	2
12	10	27.819	41.867	2
13	11	39.634	42.23	2
14	12	35.477	35.104	2
15	13	25.768	5.967	1
16	14	-0.684	21.105	0
17	15	3.387	17.81	0
18	16	32.986	3.412	1
19	17	34.258	9.931	1
20	18	6.313	29.426	0
21	19	33.899	37.535	2



Program:

```
[ ] from sklearn import datasets
    from sklearn.model_selection import train_test_split
    iris = datasets.load_iris()
    X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, random_state = 42, test_size = 0.2)
```

```
class KNN:
    def __init__(self, k):
        self.k = k

    def fit(self, X, y):
        self.X_train = X
        self.y_train = y

    def distance(self, X1, X2):
        distance = scipy.spatial.distance.euclidean(X1, X2)

    def predict(self, X_test):
        final_output = []
        for i in range(len(X_test)):
            d = []
            votes = []
            for j in range(len(X_train)):
                dist = scipy.spatial.distance.euclidean(X_train[j] , X_test[i])
                d.append([dist, j])
            d.sort()
            d = d[0:self.k]
            for d, j in d:
                votes.append(y_train[j])
            ans = Counter(votes).most_common(1)[0][0]
            final_output.append(ans)

        return final_output

    def score(self, X test, y test):
```

Output:

```
clf = KNN(3)
clf.fit(X_train, y_train)
prediction = clf.predict(X_test)
for i in prediction:
    print(i, end= ' ')
```

```
1 0 2 1 1 0 1 2 1 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0
```

```
prediction == y_test
```

```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True])
```

```
clf.score(X_test, y_test)
```

```
1.0
```

Date: 17.05.2023

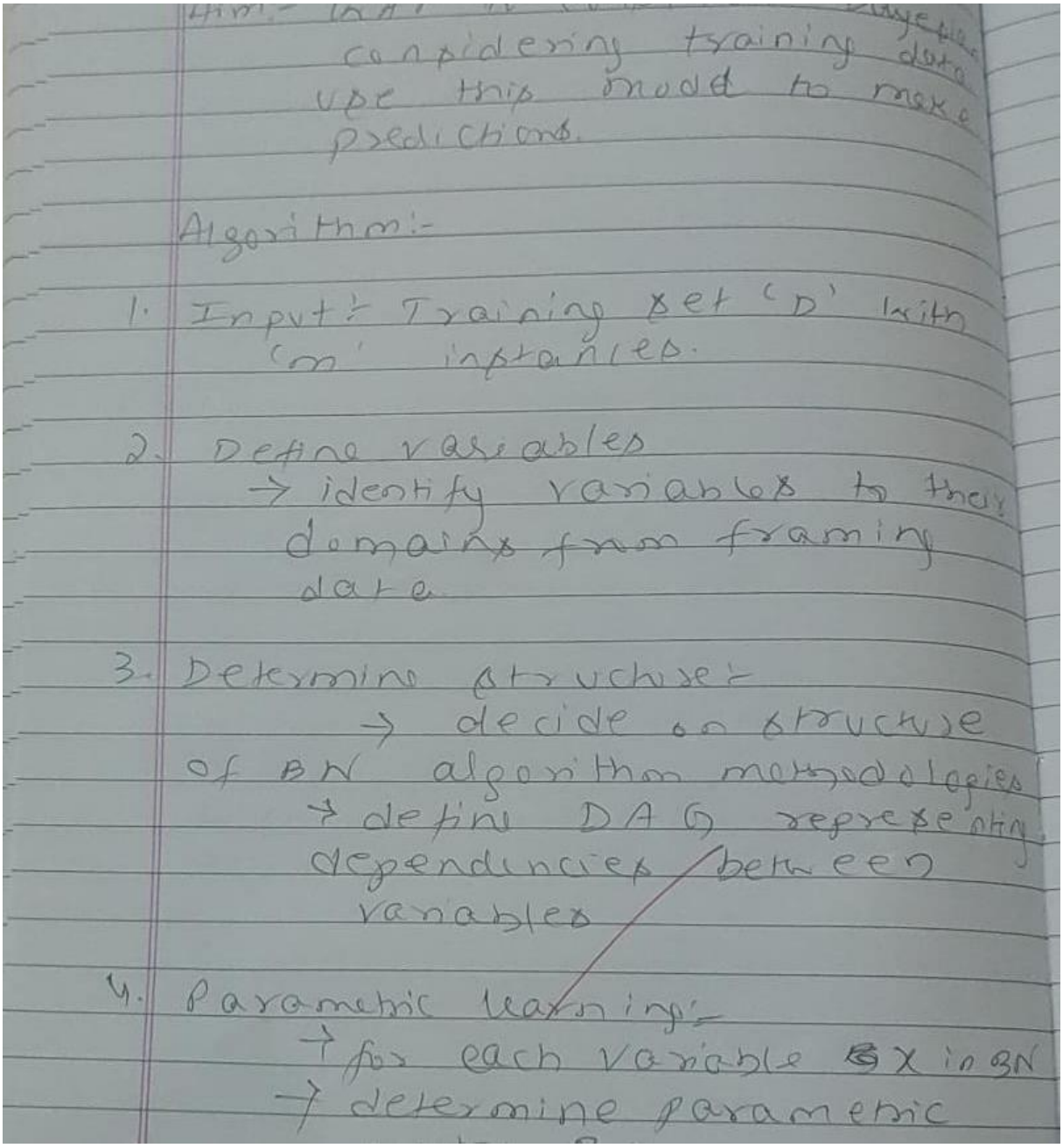
Program 10:

Write a program to construct a Bayesian network considering training data. Use this model to make predictions..

Dataset :heart.csv

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	heartdisease
2	63	1	1	145	233	1	2	150	0	2.3	3	0	6	0
3	67	1	4	160	286	0	2	108	1	1.5	2	3	3	2
4	67	1	4	120	229	0	2	129	1	2.6	2	2	7	1
5	37	1	3	130	250	0	0	187	0	3.5	3	0	3	0
6	41	0	2	130	204	0	2	172	0	1.4	1	0	3	0
7	56	1	2	120	236	0	0	178	0	0.8	1	0	3	0
8	62	0	4	140	268	0	2	160	0	3.6	3	2	3	3
9	57	0	4	120	354	0	0	163	1	0.6	1	0	3	0
10	63	1	4	130	254	0	2	147	0	1.4	2	1	7	2
11	53	1	4	140	203	1	2	155	1	3.1	3	0	7	1
12	57	1	4	140	192	0	0	148	0	0.4	2	0	6	0
13	56	0	2	140	294	0	2	153	0	1.3	2	0	3	0
14	56	1	3	130	256	1	2	142	1	0.6	2	1	6	2
15	44	1	2	120	263	0	0	173	0	0	1	0	7	0
16	52	1	3	172	199	1	0	162	0	0.5	1	0	7	0
17	57	1	3	150	168	0	0	174	0	1.6	1	0	3	0
18	48	1	2	110	229	0	0	168	0	1	3	0	7	1
19	54	1	4	140	239	0	0	160	0	1.2	1	0	3	0
20	48	0	3	130	275	0	0	139	0	0.2	1	0	3	0
21	49	1	2	130	266	0	0	171	0	0.6	1	0	3	0
22	64	1	1	110	211	0	2	144	1	1.8	2	0	3	0
23	58	0	1	150	283	1	2	162	0	1	1	0	3	0
24	58	1	2	120	284	0	2	160	0	1.8	2	0	3	1
25	58	1	3	132	224	0	2	173	0	3.2	1	2	7	3
26	60	1	4	130	206	0	2	132	1	2.4	2	2	7	4
27	50	0	3	120	219	0	0	158	0	1.6	2	0	3	0
28	58	0	3	120	340	0	0	172	0	0	1	0	3	0
29	66	0	1	150	226	0	0	114	0	2.6	3	0	3	0
30	43	1	4	150	247	0	0	171	0	1.5	1	0	3	0
31	40	1	4	110	167	0	2	114	1	2	2	0	7	3
32	69	0	1	140	239	0	0	151	0	1.8	1	2	3	0
33	60	1	4	117	230	1	0	160	1	1.4	1	2	7	2

Algorithm:



considering training data
use this model to make
predictions.

Algorithm:-

1. Input :- Training set 'D' with 'm' instances.

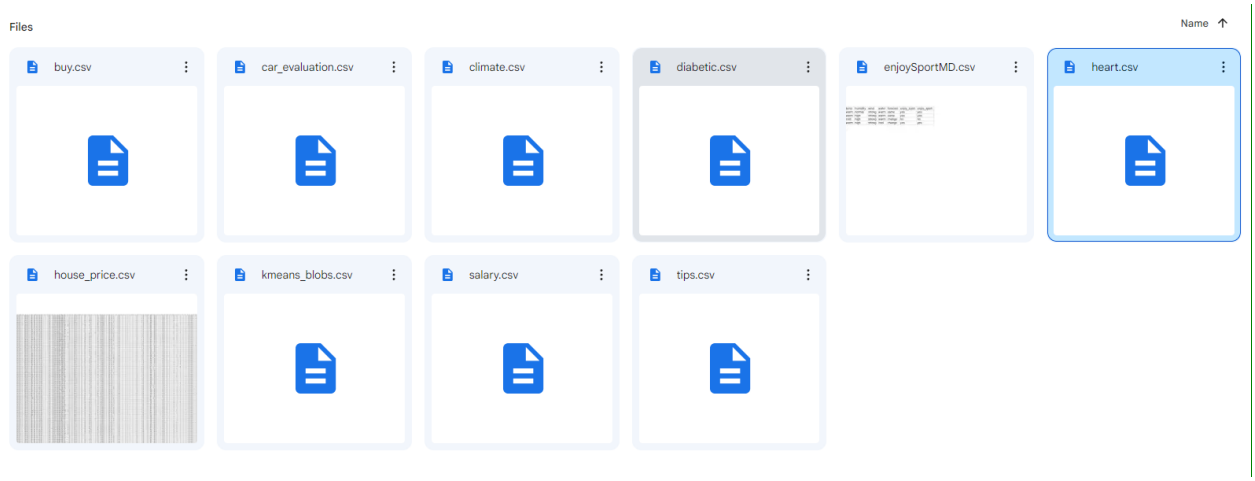
2. Define variables
   → identify variables to their domains from framing data.

3. Determine structure :-
   → decide on structure of BN algorithm methodologies
   → define DAG representing dependencies between variables

4. Parametric learning :-
   → for each variable X in BN
   → determine parametric

Uploading the csv file:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	heartdisease
2	63	1	1	145	233	1	2	150	0	2.3	3	0	6	0
3	67	1	4	160	286	0	2	108	1	1.5	2	3	3	2
4	67	1	4	120	229	0	2	129	1	2.6	2	2	7	1
5	37	1	3	130	250	0	0	187	0	3.5	3	0	3	0
6	41	0	2	130	204	0	2	172	0	1.4	1	0	3	0
7	56	1	2	120	236	0	0	178	0	0.8	1	0	3	0
8	62	0	4	140	268	0	2	160	0	3.6	3	2	3	3
9	57	0	4	120	354	0	0	163	1	0.6	1	0	3	0
10	63	1	4	130	254	0	2	147	0	1.4	2	1	7	2
11	53	1	4	140	203	1	2	155	1	3.1	3	0	7	1
12	57	1	4	140	192	0	0	148	0	0.4	2	0	6	0
13	56	0	2	140	294	0	2	153	0	1.3	2	0	3	0
14	56	1	3	130	256	1	2	142	1	0.6	2	1	6	2
15	44	1	2	120	263	0	0	173	0	0	1	0	7	0
16	52	1	3	172	199	1	0	162	0	0.5	1	0	7	0
17	57	1	3	150	168	0	0	174	0	1.6	1	0	3	0
18	48	1	2	110	229	0	0	168	0	1	3	0	7	1
19	54	1	4	140	239	0	0	160	0	1.2	1	0	3	0
20	48	0	3	130	275	0	0	139	0	0.2	1	0	3	0
21	49	1	2	130	266	0	0	171	0	0.6	1	0	3	0
22	64	1	1	110	211	0	2	144	1	1.8	2	0	3	0
23	58	0	1	150	283	1	2	162	0	1	1	0	3	0
24	58	1	2	120	284	0	2	160	0	1.8	2	0	3	1
25	58	1	3	132	224	0	2	173	0	3.2	1	2	7	3
26	60	1	4	130	206	0	2	132	1	2.4	2	2	7	4
27	50	0	3	120	219	0	0	158	0	1.6	2	0	3	0
28	58	0	3	120	340	0	0	172	0	0	1	0	3	0
29	66	0	1	150	226	0	0	114	0	2.6	3	0	3	0
30	43	1	4	150	247	0	0	171	0	1.5	1	0	3	0
31	40	1	4	110	167	0	2	114	1	2	2	0	7	3
32	69	0	1	140	239	0	0	151	0	1.8	1	2	3	0
33	60	1	4	117	230	1	0	160	1	1.4	1	2	7	2



Program:

```
import numpy as np
import pandas as pd

heartDisease = pd.read_csv('/content/drive/MyDrive/ML-LAB-MD/heart.csv')
heartDisease = heartDisease.replace('?', np.nan)

print('Sample instances from the dataset are given below')
print(heartDisease.head())

print('\nAttributes and datatypes')
print(heartDisease.dtypes)

def query_probability(variable, evidence):
    evidence_columns = list(evidence.keys())
    evidence_values = list(evidence.values())

    # Filter the dataset based on evidence
    filtered_data = heartDisease.copy()
    for column, value in zip(evidence_columns, evidence_values):
        filtered_data = filtered_data[filtered_data[column] == value]

    # Calculate the probability distribution of the variable given the evidence
    probabilities = filtered_data[variable].value_counts(normalize=True)
    return probabilities

print('\nInferencing with Bayesian Network:')
print('\n1. Probability of HeartDisease given evidence= restecg')
q1 = query_probability('heartdisease', {'restecg': 1})
print(q1)

print('\n2. Probability of HeartDisease given evidence= cp')
q2 = query_probability('heartdisease', {'cp': 2})
```

Output:

```
Sample instances from the dataset are given below
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   63    1   1       145   233    1        2      150      0      2.3      3
1   67    1   4       160   286    0        2      108      1      1.5      2
2   67    1   4       120   229    0        2      129      1      2.6      2
3   37    1   3       130   250    0        0      187      0      3.5      3
4   41    0   2       130   204    0        2      172      0      1.4      1

  ca thal  heartdisease
0  0    6             0
1  3    3             2
2  2    7             1
3  0    3             0
4  0    3             0

Attributes and datatypes
age               int64
sex               int64
cp                int64
trestbps          int64
chol              int64
fbs               int64
restecg           int64
thalach           int64
exang             int64
oldpeak         float64
slope             int64
ca               object
thal             object
heartdisease      int64
dtype: object

Inferencing with Bayesian Network:

1. Probability of HeartDisease given evidence= restecg
2    0.25
0    0.25
3    0.25
4    0.25
Name: heartdisease, dtype: float64

2. Probability of HeartDisease given evidence= cp
```

Date: 17.05.2023

Program 11:

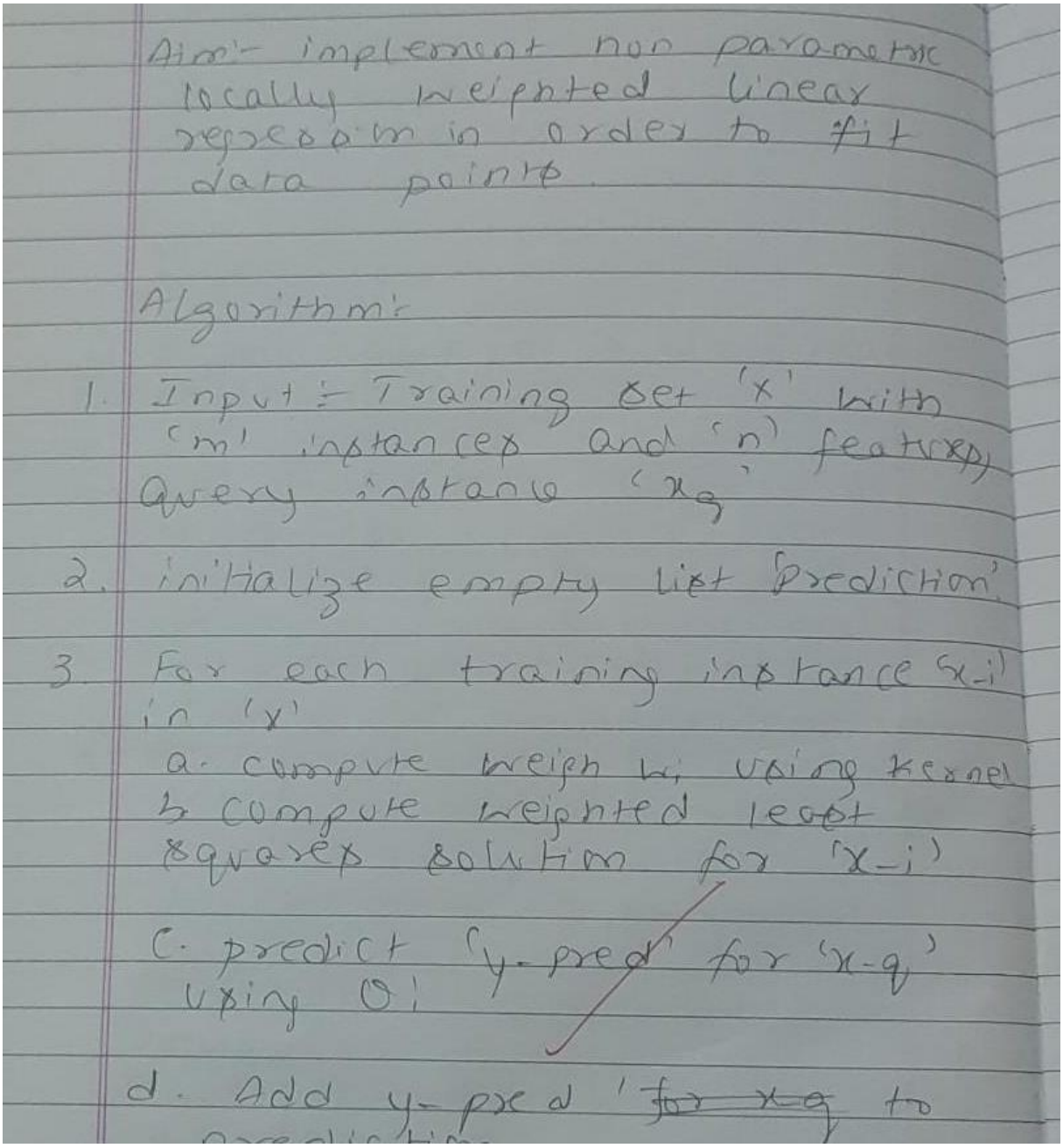
Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

Dataset :tips.csv

A1 | total_bill

	A	B	C	D	E	F	G
1	total_bill	tip	sex	smoker	day	time	size
2	16.99	1.01	Female	No	Sun	Dinner	2
3	10.34	1.66	Male	No	Sun	Dinner	3
4	21.01	3.5	Male	No	Sun	Dinner	3
5	23.68	3.31	Male	No	Sun	Dinner	2
6	24.59	3.61	Female	No	Sun	Dinner	4
7	25.29	4.71	Male	No	Sun	Dinner	4
8	8.77	2	Male	No	Sun	Dinner	2
9	26.88	3.12	Male	No	Sun	Dinner	4
10	15.04	1.96	Male	No	Sun	Dinner	2
11	14.78	3.23	Male	No	Sun	Dinner	2
12	10.27	1.71	Male	No	Sun	Dinner	2
13	35.26	5	Female	No	Sun	Dinner	4
14	15.42	1.57	Male	No	Sun	Dinner	2
15	18.43	3	Male	No	Sun	Dinner	4
16	14.83	3.02	Female	No	Sun	Dinner	2
17	21.58	3.92	Male	No	Sun	Dinner	2
18	10.33	1.67	Female	No	Sun	Dinner	3
19	16.29	3.71	Male	No	Sun	Dinner	3
20	16.97	3.5	Female	No	Sun	Dinner	3
21	20.65	3.35	Male	No	Sat	Dinner	3
22	17.92	4.08	Male	No	Sat	Dinner	2
23	20.29	2.75	Female	No	Sat	Dinner	2
24	15.77	2.23	Female	No	Sat	Dinner	2
25	39.42	7.58	Male	No	Sat	Dinner	4
26	19.82	3.18	Male	No	Sat	Dinner	2
27	17.81	2.34	Male	No	Sat	Dinner	4

Algorithm:

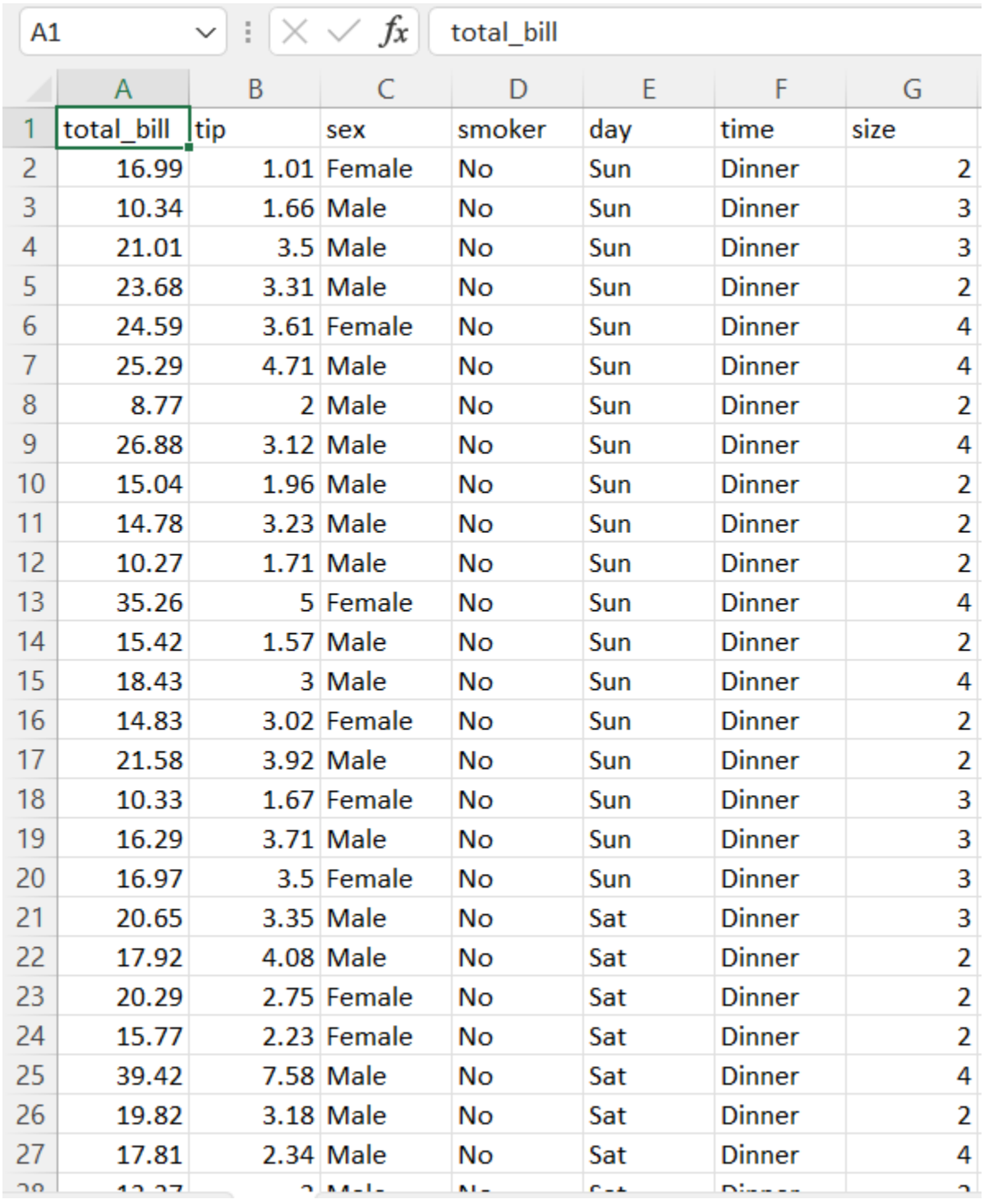


Aim:- implement non parametric locally weighted linear regression in order to fit data points.

Algorithm:

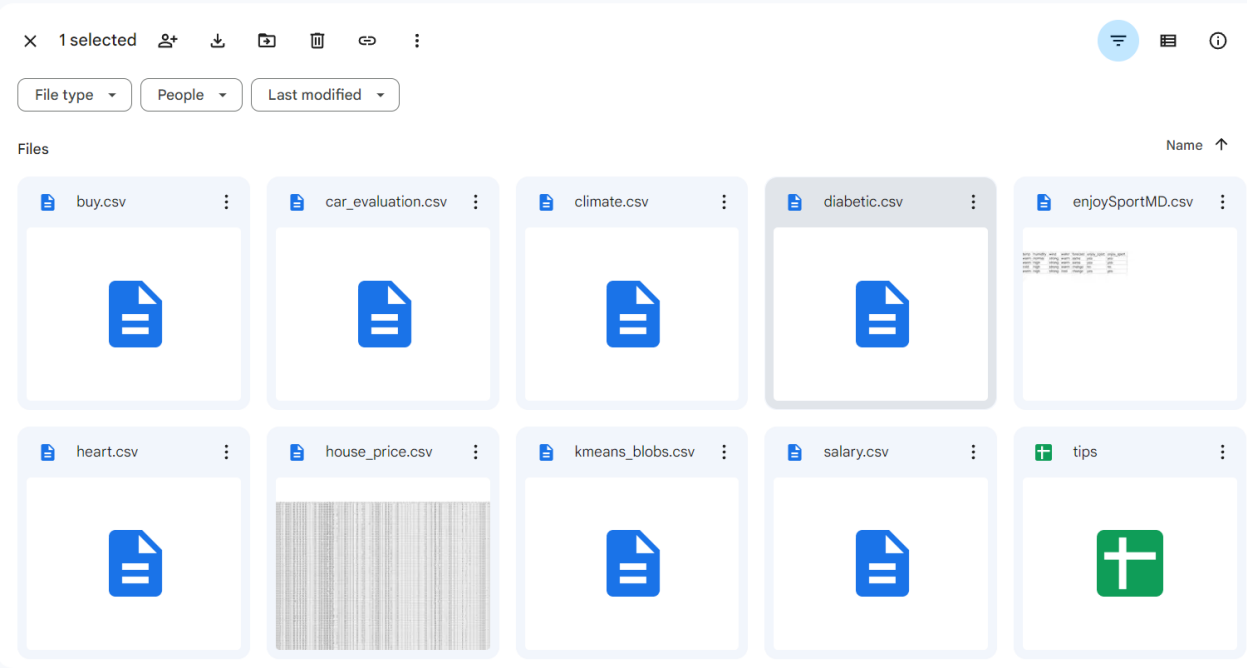
1. Input :- Training set 'X' with 'm' instances and 'n' features, Query instance 'x_q'
2. initialize empty list 'prediction'
3. For each training instance 'x_i' in 'X'
   a. compute weight w_i using kernel
   b. compute weighted least squares solution for 'x_i'
   c. predict 'y_pred' for 'x_q' using θ_i
   d. Add y_pred ~~for x_q~~ to

Uploading the csv file:



A1 | total_bill

	A	B	C	D	E	F	G
1	total_bill	tip	sex	smoker	day	time	size
2	16.99	1.01	Female	No	Sun	Dinner	2
3	10.34	1.66	Male	No	Sun	Dinner	3
4	21.01	3.5	Male	No	Sun	Dinner	3
5	23.68	3.31	Male	No	Sun	Dinner	2
6	24.59	3.61	Female	No	Sun	Dinner	4
7	25.29	4.71	Male	No	Sun	Dinner	4
8	8.77	2	Male	No	Sun	Dinner	2
9	26.88	3.12	Male	No	Sun	Dinner	4
10	15.04	1.96	Male	No	Sun	Dinner	2
11	14.78	3.23	Male	No	Sun	Dinner	2
12	10.27	1.71	Male	No	Sun	Dinner	2
13	35.26	5	Female	No	Sun	Dinner	4
14	15.42	1.57	Male	No	Sun	Dinner	2
15	18.43	3	Male	No	Sun	Dinner	4
16	14.83	3.02	Female	No	Sun	Dinner	2
17	21.58	3.92	Male	No	Sun	Dinner	2
18	10.33	1.67	Female	No	Sun	Dinner	3
19	16.29	3.71	Male	No	Sun	Dinner	3
20	16.97	3.5	Female	No	Sun	Dinner	3
21	20.65	3.35	Male	No	Sat	Dinner	3
22	17.92	4.08	Male	No	Sat	Dinner	2
23	20.29	2.75	Female	No	Sat	Dinner	2
24	15.77	2.23	Female	No	Sat	Dinner	2
25	39.42	7.58	Male	No	Sat	Dinner	4
26	19.82	3.18	Male	No	Sat	Dinner	2
27	17.81	2.34	Male	No	Sat	Dinner	4



Program:

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

def kernel(point,xmat, k):
    m,n = np.shape(xmat)
    weights = np.mat(np.eye((m))) # eye - identity matrix
    for j in range(m):
        diff = point - X[j]
        weights[j,j] = np.exp(diff*diff.T/(-2.0*k**2))
    return weights

def localWeight(point,xmat,ymat,k):
    wei = kernel(point,xmat,k)
    W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
    return W

def localWeightRegression(xmat,ymat,k):
    m,n = np.shape(xmat)
    ypred = np.zeros(m)
    for i in range(m):
        ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
    return ypred

def graphPlot(X,ypred):
    sortindex = X[:,1].argsort(0) #argsort - index of the smallest
    xsort = X[sortindex][:,0]
    fig = plt.figure()
    ax = fig.add_subplot(1,1,1)
    ax.scatter(bill,tip, color='green')
    ax.plot(xsort[:,1],ypred[sortindex], color = 'red', linewidth=5)
    plt.xlabel('Total bill')
    plt.ylabel('Tip')
    plt.show();
```

```
# load data points
data = pd.read_csv('tips.csv')
bill = np.array(data.total_bill) # We use only Bill amount and Tips data
tip = np.array(data.tip)

mbill = np.mat(bill) # .mat will convert nd array is converted in 2D array
mtip = np.mat(tip)
m= np.shape(mbill)[1]
one = np.mat(np.ones(m))
X = np.hstack((one.T,mbill.T)) # 244 rows, 2 cols

# increase k to get smooth curves
ypred = localWeightRegression(X,mtip,3)
graphPlot(X,ypred)
```

Output: