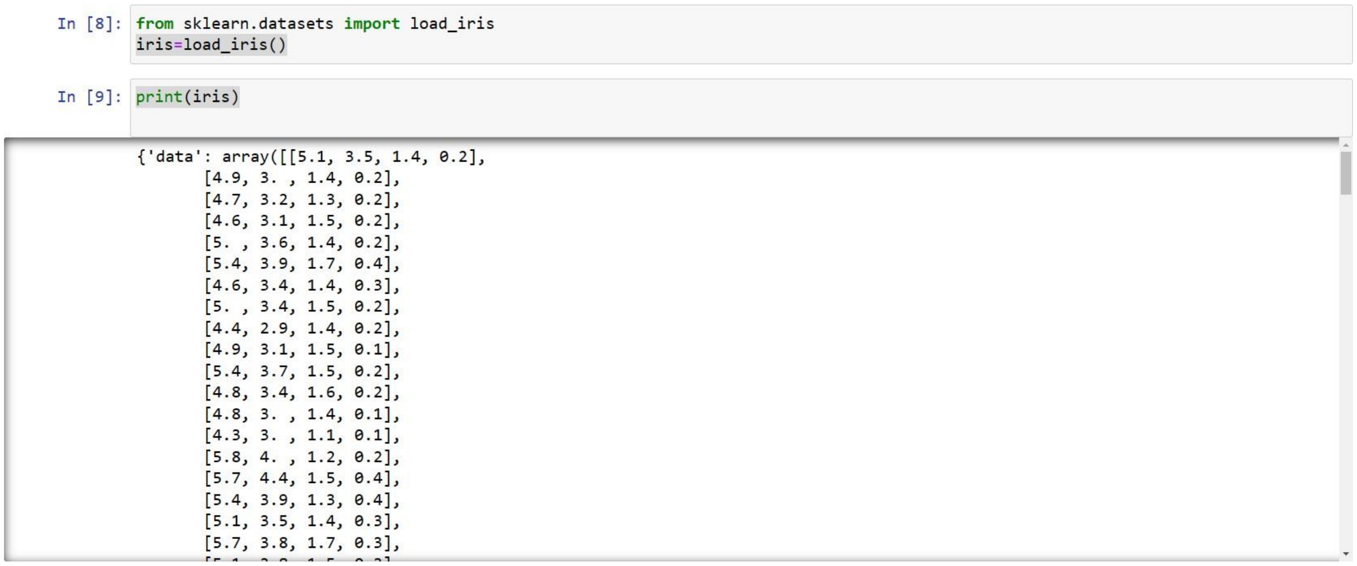
NAME: SHREYAS CHOLKE
USN: 1BM20CS154

MACHINE LEARNING LAB OBSERVATION

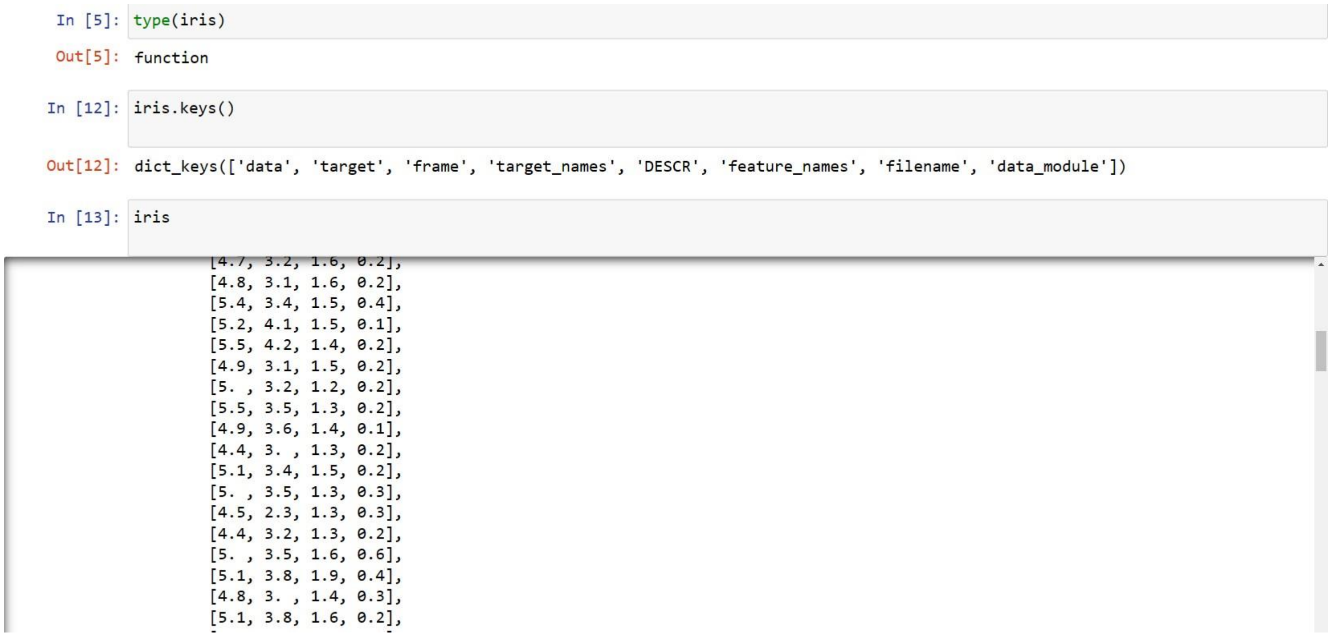
Date: 1-04-2023
Lab 1: Exploring Datasets

IRIS DATASET:

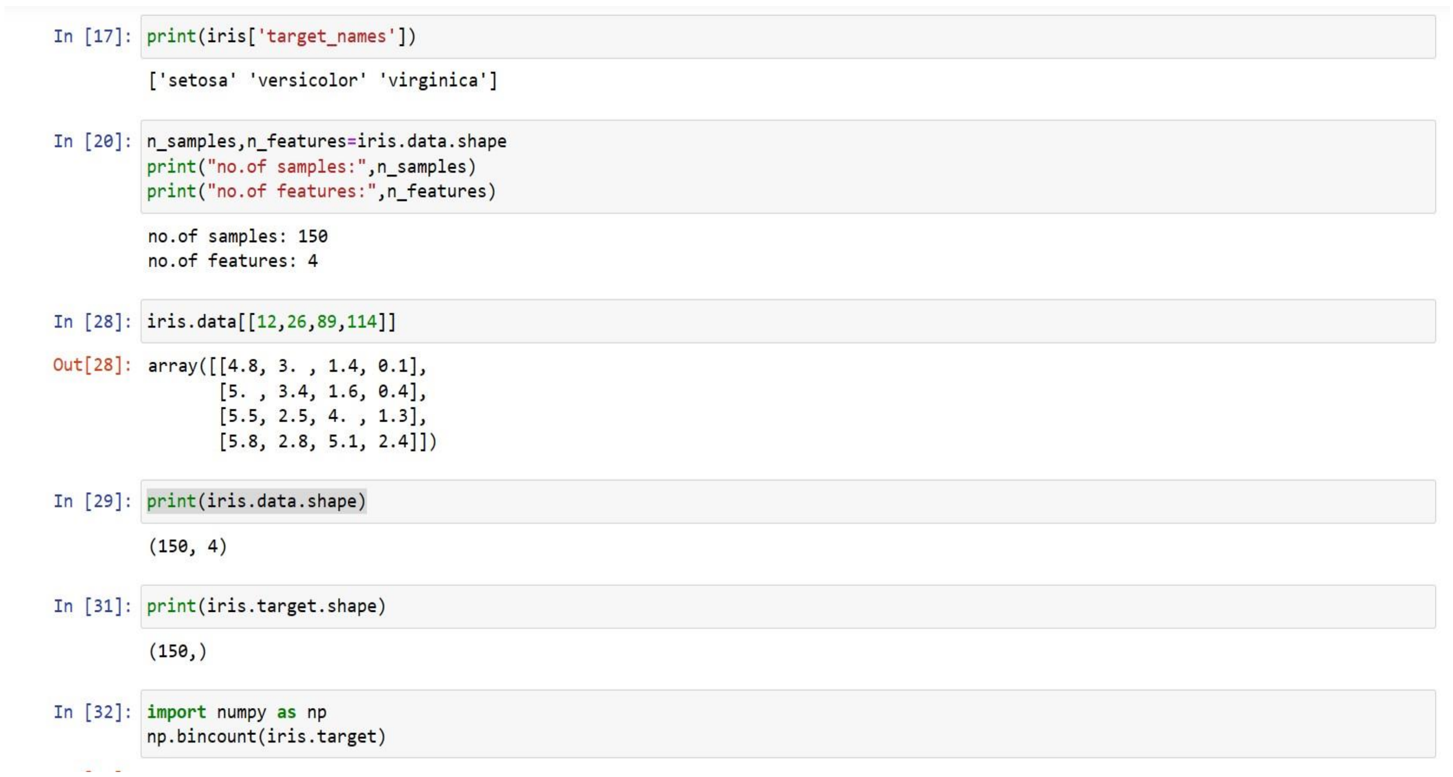
- Features in the Iris dataset:
  1. sepal length in cm
  2. sepal width in cm
  3. petal length in cm
  4. petal width in cm
- Target classes to predict:
  1. Iris Setosa
  2. Iris Versicolour
  3. Iris Virginica



```
In [8]: from sklearn.datasets import load_iris
        iris=load_iris()

In [9]: print(iris)

{'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],
```

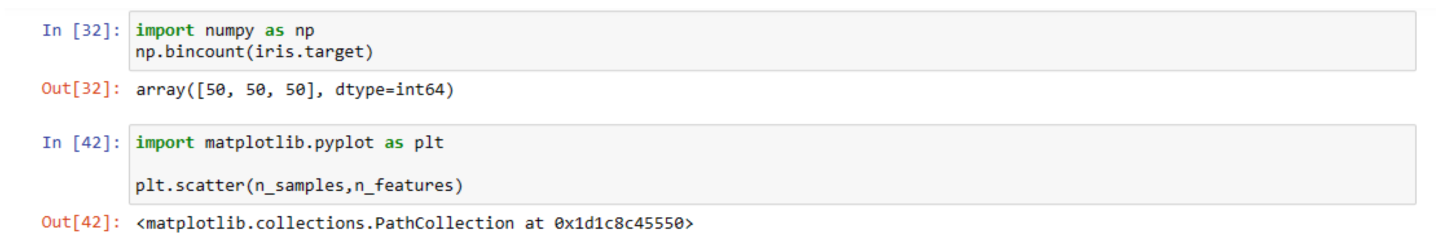


```
In [5]: type(iris)

Out[5]: function

In [12]: iris.keys()

Out[12]: dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module'])

In [13]: iris

       [4.7, 3.2, 1.6, 0.2],
       [4.8, 3.1, 1.6, 0.2],
       [5.4, 3.4, 1.5, 0.4],
       [5.2, 4.1, 1.5, 0.1],
       [5.5, 4.2, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.2],
       [5. , 3.2, 1.2, 0.2],
       [5.5, 3.5, 1.3, 0.2],
       [4.9, 3.6, 1.4, 0.1],
       [4.4, 3. , 1.3, 0.2],
       [5.1, 3.4, 1.5, 0.2],
       [5. , 3.5, 1.3, 0.3],
       [4.5, 2.3, 1.3, 0.3],
       [4.4, 3.2, 1.3, 0.2],
       [5. , 3.5, 1.6, 0.6],
       [5.1, 3.8, 1.9, 0.4],
       [4.8, 3. , 1.4, 0.3],
       [5.1, 3.8, 1.6, 0.2],
```

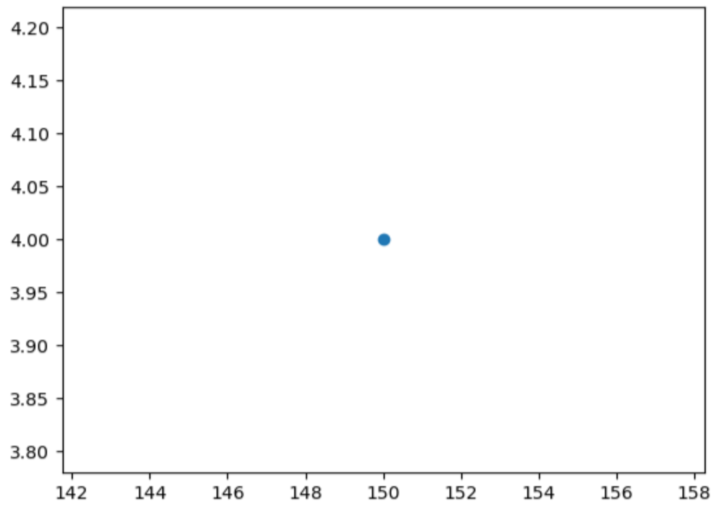


```
In [17]: print(iris['target_names'])

         ['setosa' 'versicolor' 'virginica']

In [20]: n_samples,n_features=iris.data.shape
         print("no.of samples:",n_samples)
         print("no.of features:",n_features)

         no.of samples: 150
         no.of features: 4

In [28]: iris.data[[12,26,89,114]]

Out[28]: 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 [29]: print(iris.data.shape)

         (150, 4)

In [31]: print(iris.target.shape)

         (150,)

In [32]: import numpy as np
         np.bincount(iris.target)
```

Scattered graph for samples vs features.

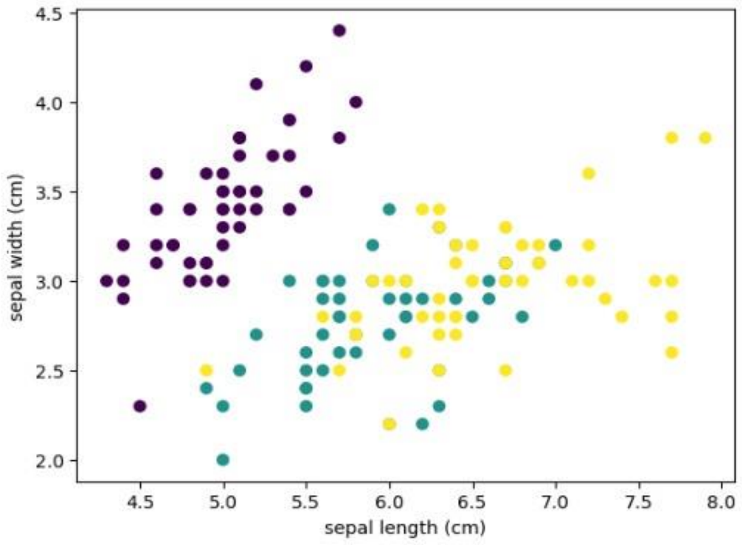


```
In [32]: import numpy as np
         np.bincount(iris.target)

Out[32]: array([50, 50, 50], dtype=int64)

In [42]: import matplotlib.pyplot as plt

         plt.scatter(n_samples,n_features)

Out[42]: <matplotlib.collections.PathCollection at 0x1d1c8c45550>
```



Scattered graph: with first two features(septal width vs septal length)
The three colors represents three different classes respectively.

In [47]:
```
features = iris.data.T

plt.scatter(features[0], features[1],
            c=iris.target)
plt.xlabel(iris.feature_names[0])
plt.ylabel(iris.feature_names[1]);
```



In [49]:
```
iris.data[[1,2,3,4,5]]
```

Out[49]:
```
array([[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]])
```

WINE DATASET:

In [51]:
```
from sklearn.datasets import load_wine
wine=load_wine()
```

In [52]:
```
print(wine)
```

```
{'data': array([[1.423e+01, 1.710e+00, 2.430e+00, ..., 1.040e+00, 3.920e+00,
        1.065e+03],
       [1.320e+01, 1.780e+00, 2.140e+00, ..., 1.050e+00, 3.400e+00,
        1.050e+03],
       [1.316e+01, 2.360e+00, 2.670e+00, ..., 1.030e+00, 3.170e+00,
```

In [57]:
```
wine.data
```

Out[57]:
```
array([[1.423e+01, 1.710e+00, 2.430e+00, ..., 1.040e+00, 3.920e+00,
        1.065e+03],
       [1.320e+01, 1.780e+00, 2.140e+00, ..., 1.050e+00, 3.400e+00,
        1.050e+03],
       [1.316e+01, 2.360e+00, 2.670e+00, ..., 1.030e+00, 3.170e+00,
        1.185e+03],
       ...,
       [1.327e+01, 4.280e+00, 2.260e+00, ..., 5.900e-01, 1.560e+00,
        8.350e+02],
       [1.317e+01, 2.590e+00, 2.370e+00, ..., 6.000e-01, 1.620e+00,
        8.400e+02],
       [1.413e+01, 4.100e+00, 2.740e+00, ..., 6.100e-01, 1.600e+00,
        5.600e+02]])
```

In [58]:
```
wine.keys()
```

Out[58]:
```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names'])
```

In [60]:
```
print(wine['target_names'])
```

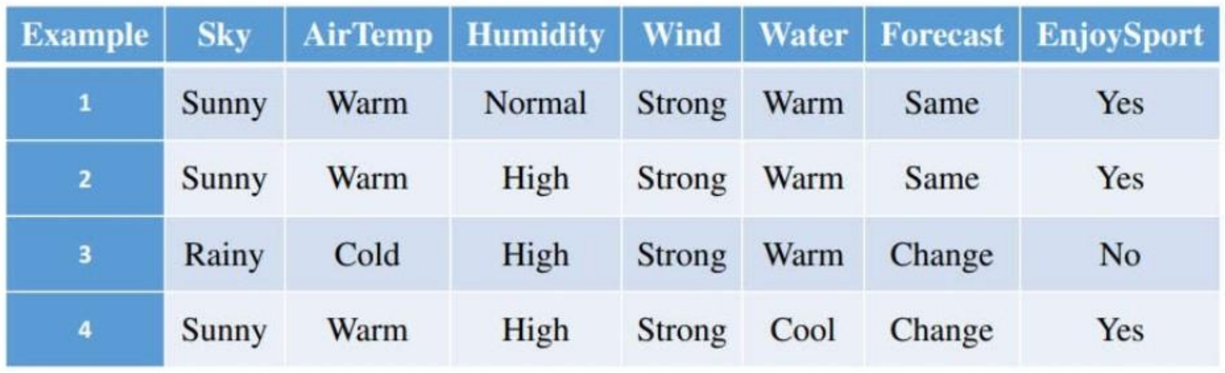
```
['class_0' 'class_1' 'class_2']
```

```
In {9] : pri nt (nine[ ' feature_nanes ' ])

   alcohol ', ' uiit_itiâ ' , ' asfi ', 's 1rsii r ity_i›f_s iâ ', 'nsgnesiun' , ' totil _pfii nols ', ’flsvinoidi ', ' nonf1svanoid_ ptcnols ', ' pros
  nthocyanins', 'color_intensity', 'hue', 'od280/od315_of_diluted_wines', 'proline']

ir {ii) Isport nonpJ is np
        up. âintevat(nine.target)

Out[11]  array((!i9, 71, GB], dtype•intâ4)
```

Date: 15/04/2023
Lab 2: FIND-S ALGORITHM FOR ENJOY SPORT:

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:Enjoysport

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:
initialize h to the most specific hypothesis in H h-(Ø, Ø, Ø, Ø, Ø, Ø)

1. First training example X1=< Sunny, Warm. Normal, Strong Warm Same>. EnjoySport=+ve Observing.The first trainin 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>.

2.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>

3. 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 13=< Sunny, Warm, ?, Strong,Warm, Same>

4. 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, ?, ?>

5. So the final hypothesis is < Sunny, Warm, ?, Strong, ?, ?>

5/4/23

Lab Program 1

Find S algorithm

Dataset : enjoysports. csv file

Sample	Sky	Air temp	Humidity	Wind	Water	Forcast	enjoy sports?
1)	Sunny	warm	normal	Strong	warm	Same	Yes +
2)	Sunny	warm	high	strong	warm	Same	Yes +
3)	Sunny	cold	high	Strong	warm	Same	No -
4)	Sunny	warm	high	Strong	warm	same	Yes. +

* Find S algorithm: Is a basic-concept-learning algo in ML.

* It finds what is most-specific hypothesis that fits all the "Positive" examples.

* This algo starts with the most specific hypothesis and moves to the most general hypothesis.

? → accepts any value General.
Ø → accepts No value. specific (value)
MGD → (?? ? ?) accepts everything.
MSD → (Ø Ø Ø Ø) accepts None → Null.

initial hypo : {Ø, Ø, Ø, Ø}

iteration 1 h1 = < 'Sunny', 'warm', 'normal', 'strong', 'warm' 'Same' > +ve

iteration 2 h2 = < 'Sunny', 'warm', 'high', 'strong', 'warm', 'Same'

iteration 3 h3 = < 'Raing', 'cold', 'high', 'strong', 'warm', 'Change' >

iteration 4 h4 = < 'Sunny', 'warm', 'high', 'Strong', 'cool', 'Change' >.

(Not considered)

1) Initialize 'h' to the most specific hypo in H.

2) For each positive training instance 'x' each attribute constraint ai in ~~h~~ if the constraint ai is set satisfied by 'x' then do nothing. else replace ai in h by the next more general constraint that is required by 'x' hypothese h.

3) Output hypothesis h.

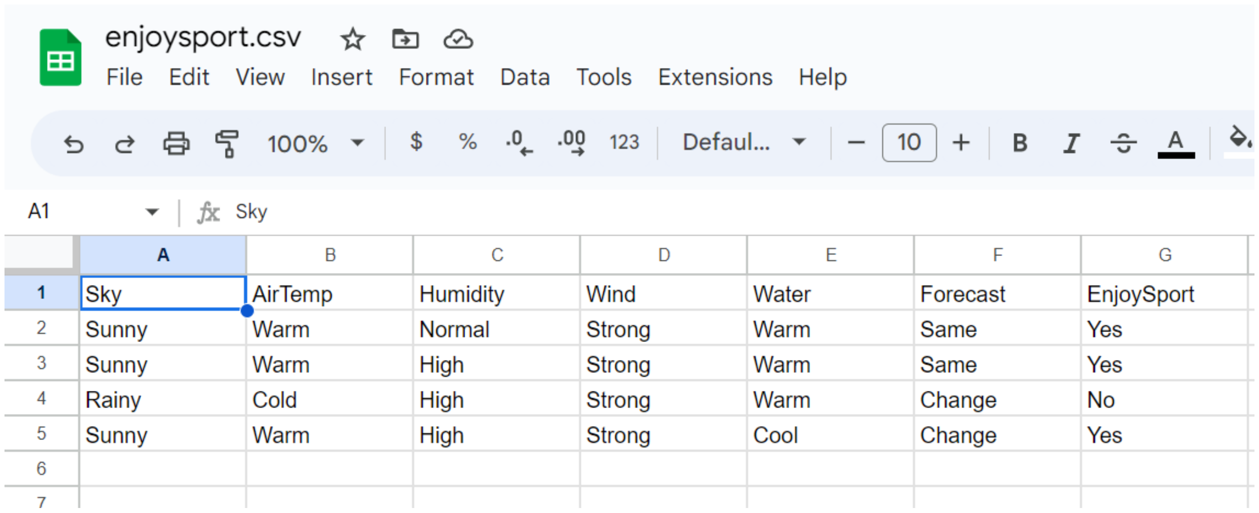
Program

```
import csv
def updateHypothesis(x,h):
  if h == []
   return x
 for i in range(0, len(h)):
  if x[i].upper() != h[i].upper():
      h[i] = '?'
  return h
 if __name__ == '__main__'
     data = []
     h = []
 with open("Desktop FindS.csv", 'r') as file:
     reader = csv.reader(file)
     print("Data:")
     data.append(row)
     print(row)
  if data:
     for x in data
    if x[-1].upper() == "Yes": x.pop()
    updateHypothesis(x,h)
 print("\n Hypothesis:", h).
```

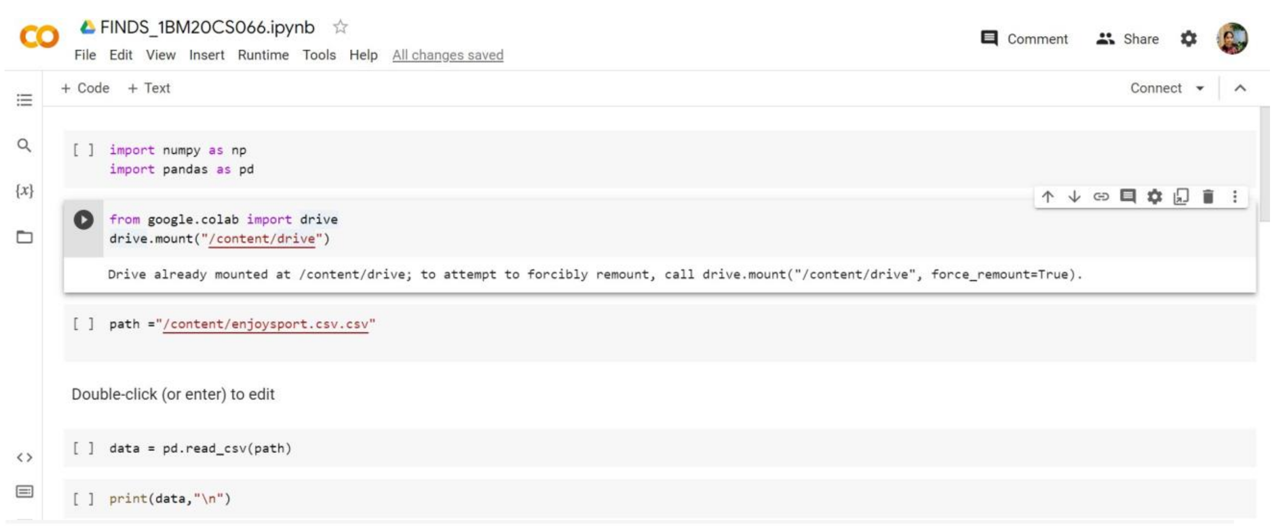
8/10

5/4/23

CREATING CSV FILE:


enjoysport.csv
File Edit View Insert Format Data Tools Extensions Help
100% $ % .0 .00 123 Defaul... 10 B I A
A1 fx Sky

	A	B	C	D	E	F	G
1	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
2	Sunny	Warm	Normal	Strong	Warm	Same	Yes
3	Sunny	Warm	High	Strong	Warm	Same	Yes
4	Rainy	Cold	High	Strong	Warm	Change	No
5	Sunny	Warm	High	Strong	Cool	Change	Yes
6							
7							



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

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

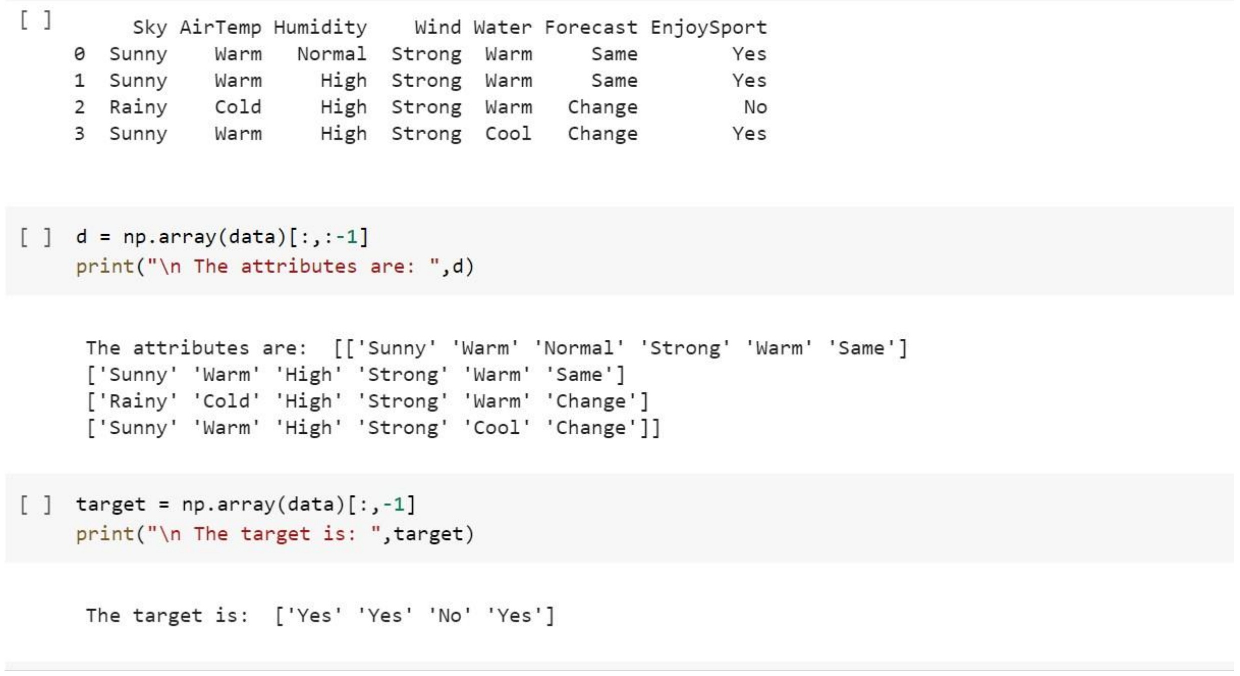
```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
path ="/content/enjoysport.csv.csv"
```

Double-click (or enter) to edit

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

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



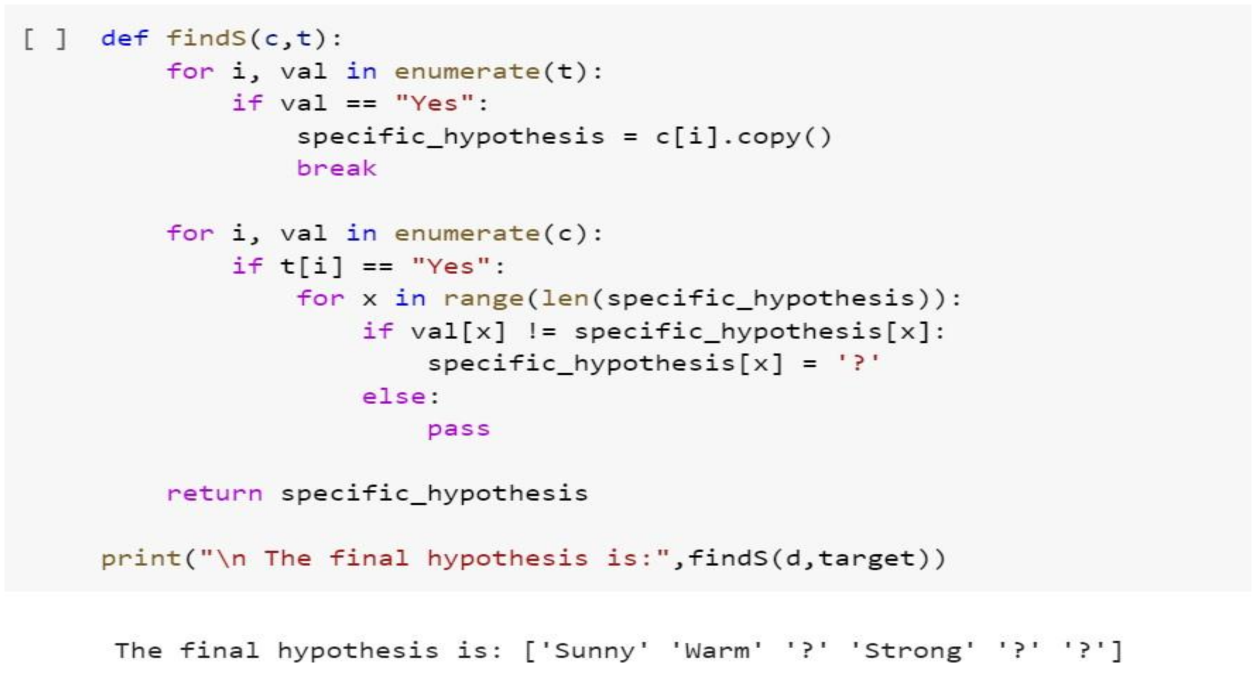
```
     Sky AirTemp Humidity    Wind Water Forecast EnjoySport
0  Sunny    Warm   Normal  Strong  Warm     Same        Yes
1  Sunny    Warm     High  Strong  Warm     Same        Yes
2  Rainy    Cold     High  Strong  Warm   Change         No
3  Sunny    Warm     High  Strong  Cool   Change        Yes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

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

```
 The attributes are:  [['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
 ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
 ['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
 ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']]
```

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

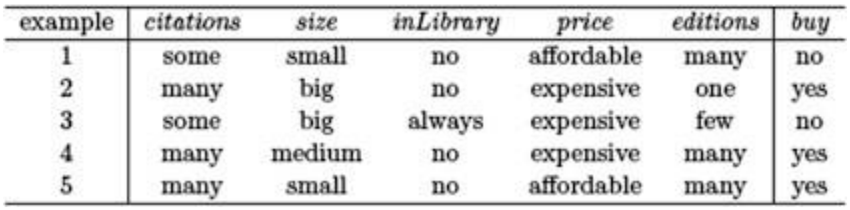
```
 The target is:  ['Yes' 'Yes' 'No' 'Yes']
```



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

         for i, val in enumerate(c):
             if t[i] == "Yes":
                 for x in range(len(specific_hypothesis)):
                     if val[x] != specific_hypothesis[x]:
                         specific_hypothesis[x] = '?'
                     else:
                         pass

         return specific_hypothesis

     print("\n The final hypothesis is:",findS(d,target))
```

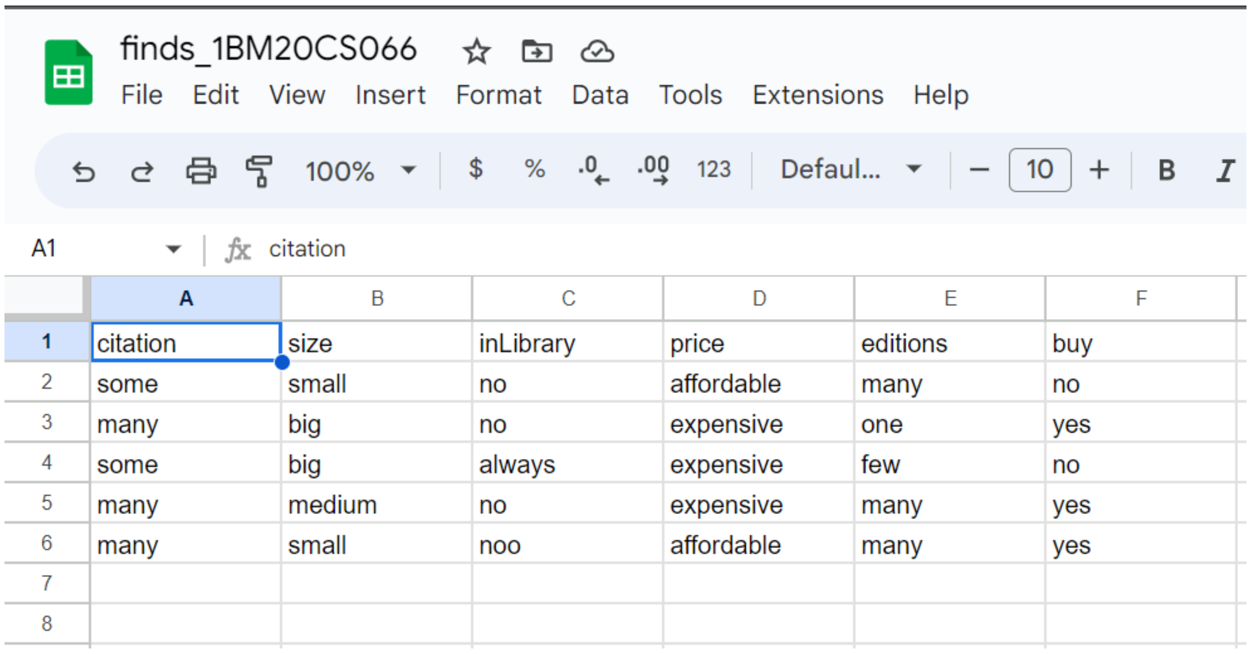
```
 The final hypothesis is: ['Sunny' 'Warm' '?' 'Strong' '?' '?']
```

SECOND DATASET: FIND-S ALGORITHM



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

CREATING CSV FILE



	A	B	C	D	E	F
1	citation	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	noo	affordable	many	yes
7						
8						

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

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

    Mounted at /content/drive
```

```
[ ] path ="/content/finds_1BPl20CS066 - Sheetl.csv"
```

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

```
[ ] p oi nt ( dat a , "\ n" )

      citation     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        noo   affordable      many   yes
```

```
[ ] d = np . a or a y ( dat a ) [ : , : - 1]
    print("\n The attributes are: ",d)

     The attributes are:  [['some' 'small' 'no' 'affordable' 'many']
     ['many' 'big' 'no' 'expensive' 'ore 'j
     ['some' 'big' 'always' 'expensive' '*e ›v']
     ['many' 'medium' 'no' 'expensive' 'many']
     ['many' 'small' ' noo' 'affordable' 'mary']]

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

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

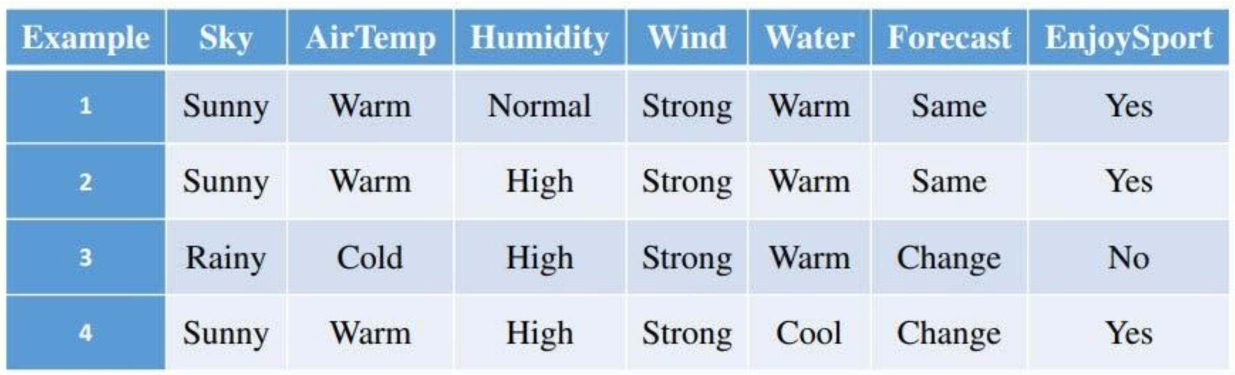
+ Code   + Text

```
[ ]  def find_s(d,target):
       for i,val in enumerate(target):
         if val=='yes' :
           hypothesis=d[i].copy()
           break

       for i,var in enumerate(d):
         if target[i]=="yes":
           for x in  range(leo(hypothesis)):
             i-F ma r' [ x ] ! =h ypo4 hes1s [ x ] :
               hypolz hes 1s x ] = '*'
             else:
               pass

       return hypothesis

     print("The Hypothesis  is",find_s(d,target))

     The Hypothesis is ['many' '?' '7' '7' '?']
```

DATE: 15/04/2023

LAB 3: CANDIDATE- ELIMINATION- ENJOY SPORT

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.
Data set: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:

```
Step1: Load Data set
Step2: Initialize General Hypothesis and Specific Hypothesis.
Step3: For each training example
Step4: If example is positive example
        if attribute_value == hypothesis_value:
          Do nothing
        else:
          replace attribute value with '?' (Basically generalizing it)
Step5: If example is Negative example
        Make generalize hypothesis more specific.
```

12/4/23

Lab Program 2
Candidate Elimination Algorithm

Example	Sky	Air temp	Humidity	Wind	Water
1	Sunny	warm	Normal	Strong	warm
2	Sunny	warm	high	strong	warm
3	Raing	cold	high	strong	warm
4	Sunny	warm	high	strong.	cool.

forcast	Enjoy sport
Same	Yes +ve
Same	Yes +ve
Change	No -ve
Change	Yes. +ve

Target variable.
6 attributes | candidate
Concept learning

gives out binary results.
Considers both negative and Positive values.

To find consistent hypothesis for a given solution of training example
most General $G_0 = <?, ?, ?, ?, ?, ?>$
Most Specific $S_0 = <\phi, \phi, \phi, \phi, \phi, \phi>$
Start from Generic Boundary,
first takes generic attribute values.
Whenever matches retain generic values if hypothesis matches expected is +ve and outcome +ve.
$G_1 = G_0$
Null value in S_0 is replaced by S_1.
$S_0 = < \phi \phi \phi \phi \phi \phi >$
No match → negative classification

* All question marks match with example, hence +ve classification. Target variable = +ve ⇒ GB

* All null values doesn't match, hence -ve Classifier. But expected = +ve classifier, therefore it is inconsistent hypothesis. When inconsist exist write next general hypothesis that is:

→ Replace Null values by 1st examples.

I) $G_1 = < ?, ?, ?, ?, ?, ? >$
sa
$S_1 = <$ Sunny, warm, Normal, Strong, warm, cha

II) $G_1 = < ? ? ? ? ? ? ? >$
Consider prev generic hypothesis.
$S_2 = G_1$
if generic +ve → retain
if match retain
if Target value -ve → Start from S.
if Target value +ve → Start from G

$S_2 = <$ Sunny, warm, ?, Strong, warm, same $>$

* GB, all ? matches with S_1, hence +ve Classification and exped.

* SB, when inconsistency make it General (?)

Target value
III) $S_3 = <$ Sunny, warm, ?, Strong, warm, same $>$ -ve
$G_3 = \{ <$ Sunny, ?, ?, ?, ?, ? $> \}$.
* Since all values are generic in Previous hypothesis, only Posible when example is +ve. and if there exists inconsistency, then all hypothesis which are Consistent with all the training examples seen now.

* ? match with all the attribute but expected -ve, hence inconsistency.

* All hypothesis which are consistent till now.
→ To do that, consider 1 ? at a time.

Program

```
        import numpy as np
        import Pandas as pd.
data = Pd. DataFrame (data=pd.read_csv (enjoysport.csv))
Concepts = np.array (data.iloc [:, 0, -1])
print (Concepts)
target = np.array (data.iloc[: 0, -1]) .
print (general_h)
print (specific_h) .
```

—— x ——

X_1 (+)

S_4 = < ? large, light, ?, thick >
G_4 = < ? ? light, ? ? > , < ? ? ? ? thick >

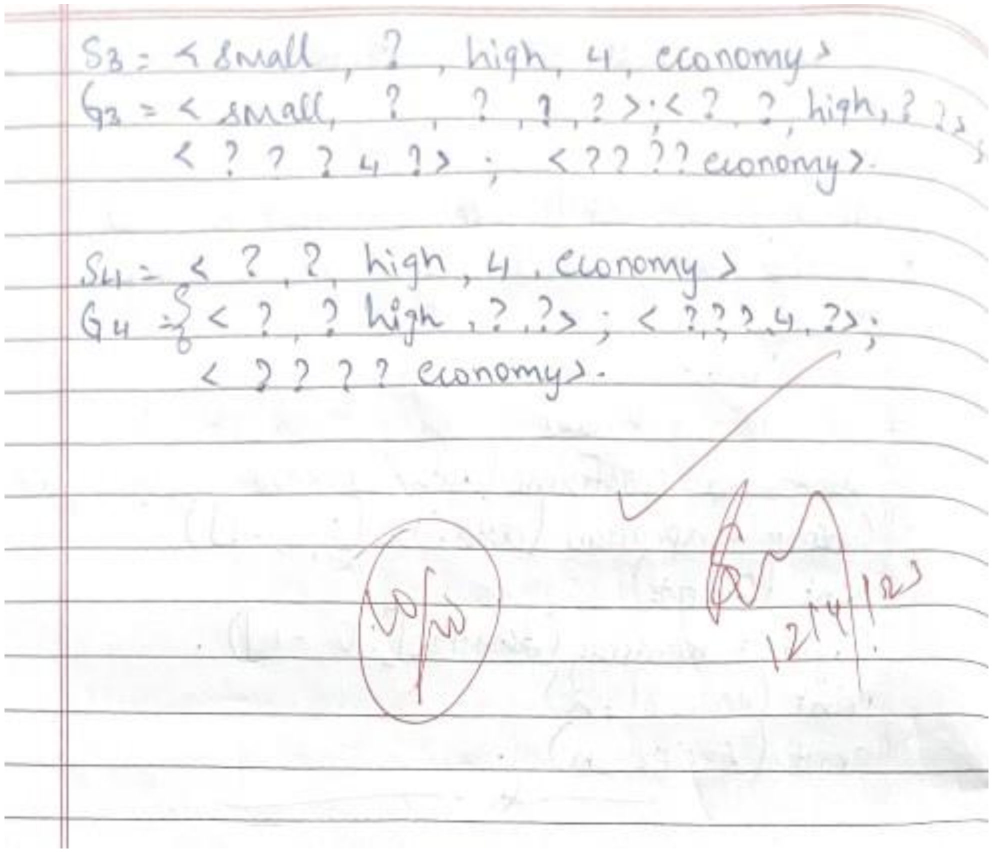
Dataset

Size	Trunk	Fuel economy	No of Passengers	Type	Target value.
Small	Available	High	4	economy	Y
Big	Available	low	2	sports	N
Small	Available	high	4	economy	Y
Small	Not Available	low	2	sports	N

G_0 = < ?, ?, ?, ?, ? >
S_0 = < ϕ, ϕ, ϕ, ϕ, ϕ >

S_1 = < small, available, high, 4, economy >
G_1 = < ? , ? , ? , ? , ? >

S_2 = < Small, ? high, 4, economy >
G_2 = { < small, ? , ? , ?, ? > ; < ?, ?, high, ?, ? > ;
{ < ?, ?, ? 4, ? > ; < ? ? ? ? economy > . }



S_3 = < small , ? , high, 4, economy >

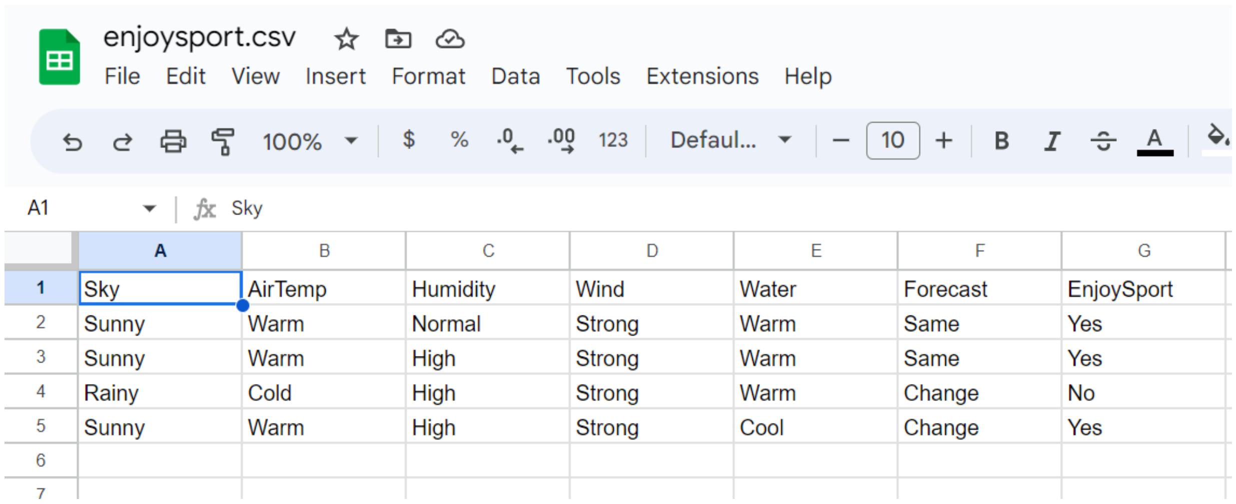
G_3 = < small, ? , ? , ? , ? >; < ? , ? , high, ? ?>,
< ? ? ? 4 ?> ; < ? ? ? ? economy>.

S_4 = < ? , ? , high , 4 , economy >

G_4 = { < ? , ? high , ? , ?> ; < ? , ? , ? , 4 , ?> ;
< ? ? ? ? economy>.

12/4/23

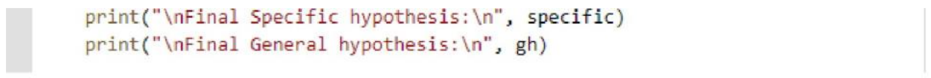
CREATING CSV FILE:



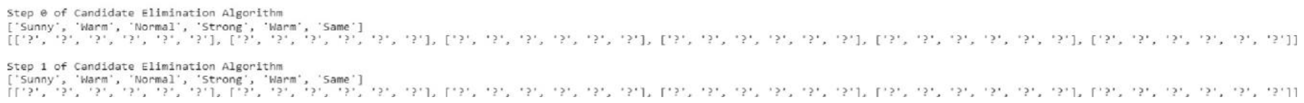
	A	B	C	D	E	F	G
1	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
2	Sunny	Warm	Normal	Strong	Warm	Same	Yes
3	Sunny	Warm	High	Strong	Warm	Same	Yes
4	Rainy	Cold	High	Strong	Warm	Change	No
5	Sunny	Warm	High	Strong	Cool	Change	Yes
6							
7							

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



     from google.colab import drive
     drive.mount('/content/drive')



     data    pd.DataFrame(data=pd.read_csv('/content/enjoysport.csv.csv'))



[ ]  print(data,"\n")

          Sky AirTemp Humidity     Wind Water Forecast  EnjoySport
     0  Su n n y    \Ja rm    No r'ma 1   St r ong  Nar'm     Same        Ye s
     1  Sunny       harm        High    Strong    Warm       Same         Yes
     2  Ra iny      Co ld       High    St rang   Nairn      Change       No
     3  Sunny       harm        High    Strong    Cool       Change       Yes



§ ]  concepts
              -1]),0::[ilocarray(data.np.


§ ]  print(conceots)

     §[ 'Su n n y'  'la r  m'
      [ 'Su n n y'  'la r  m'
      ['R a 1 n y'  'C a 1 d'
      [ 'Su n n y'  'la r  m'
                                   '
                                  Wa
b )  I: a oget   = n p . a o o a y ( ci a I:a . i 1 o c [ ,:  - 1 ] )
     p m i n I: ( I:a a get )      Wa
                                  Co


[ ] imoo t csv
```

```
with open("'/content/enjoyspont.csv.csv'") as f:
    csv file =csv.reader(f)
    data = list(csv file)

    specific = data[1][:-1]
    general = [['2' fon i in range(len(specific))] for j in range(len(specific))]

    I-or i 1n data:
        if i[-1j == "Yes™:
            for j in range(len(specific)):
                lf i[j] != specific[j]:
                    specific[j] = "2"
                    general[j][j] = "2"

        elif i[-1] == "No":
            for j in range(len(specific)):
                if i[j] != specific[j]:
                    genera1[j] [j] = spec1flc [j]
                e lse :
                    general[ j ][j] = " ° "

        print("\nStep " a str(data.index(i)) + " of Candidate Elimination Algorithm"}
        print(specific)
        print(general)

    gh = [] # gh = general Hypothesis
    hon  i  in  general:


                gh.append(i)
                break
```

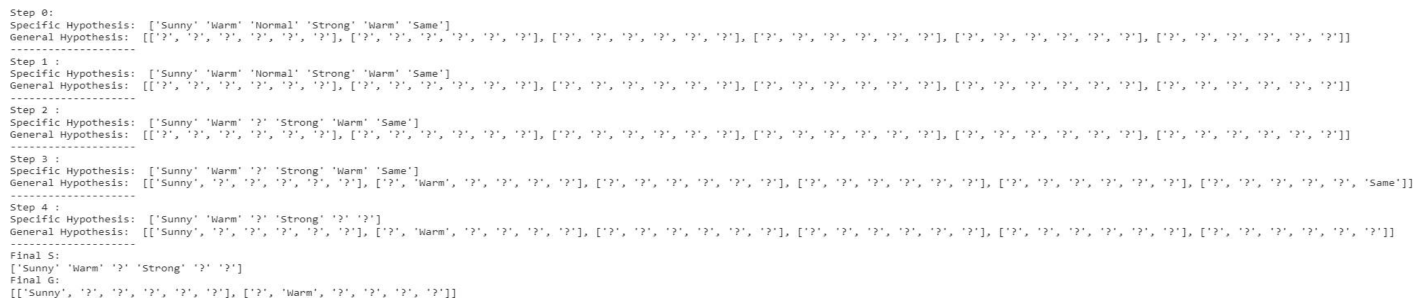


```
    print("\nFinal Specific hypothesis:\n", specific)
    print("\nFinal General hypothesis:\n", gh)
```



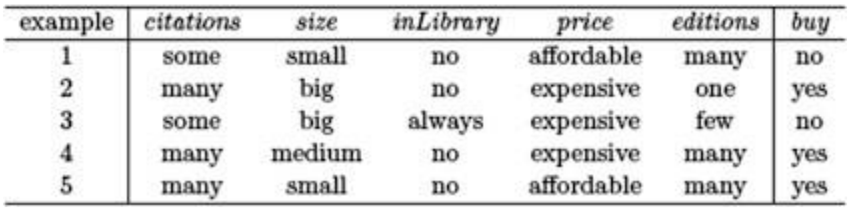
```
Step 0 of Candidate Elimination Algorithm
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Step 1 of Candidate Elimination Algorithm
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
```

```
    pr1nt("lenerai Hypotlaes*s: ", generaI_In)



                    general   h[xj[xj :     *'


                    general h[xj[xj  : specific h xJ
                 else:


    indices = [i for  i  va1  in                    if val     ['?',               '?'JJ

        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h

s fzraal.   final    Iearn(concepts. ta:rget)
```



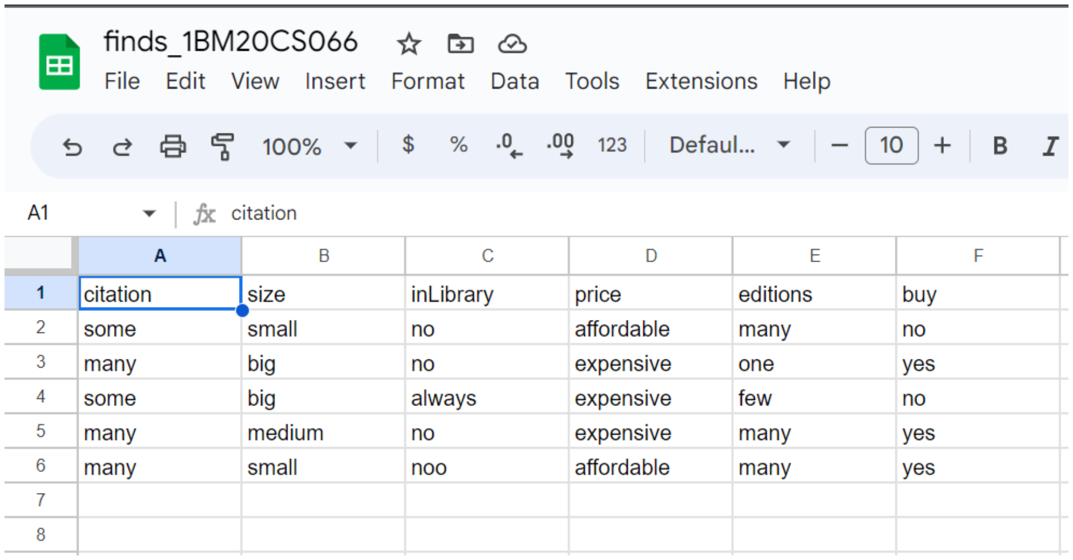
```
Step 0:
Specific Hypothesis:  ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
General Hypothesis:  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
--------------------
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', '?', '?', '?', '?']]
```

SECOND DATASET:

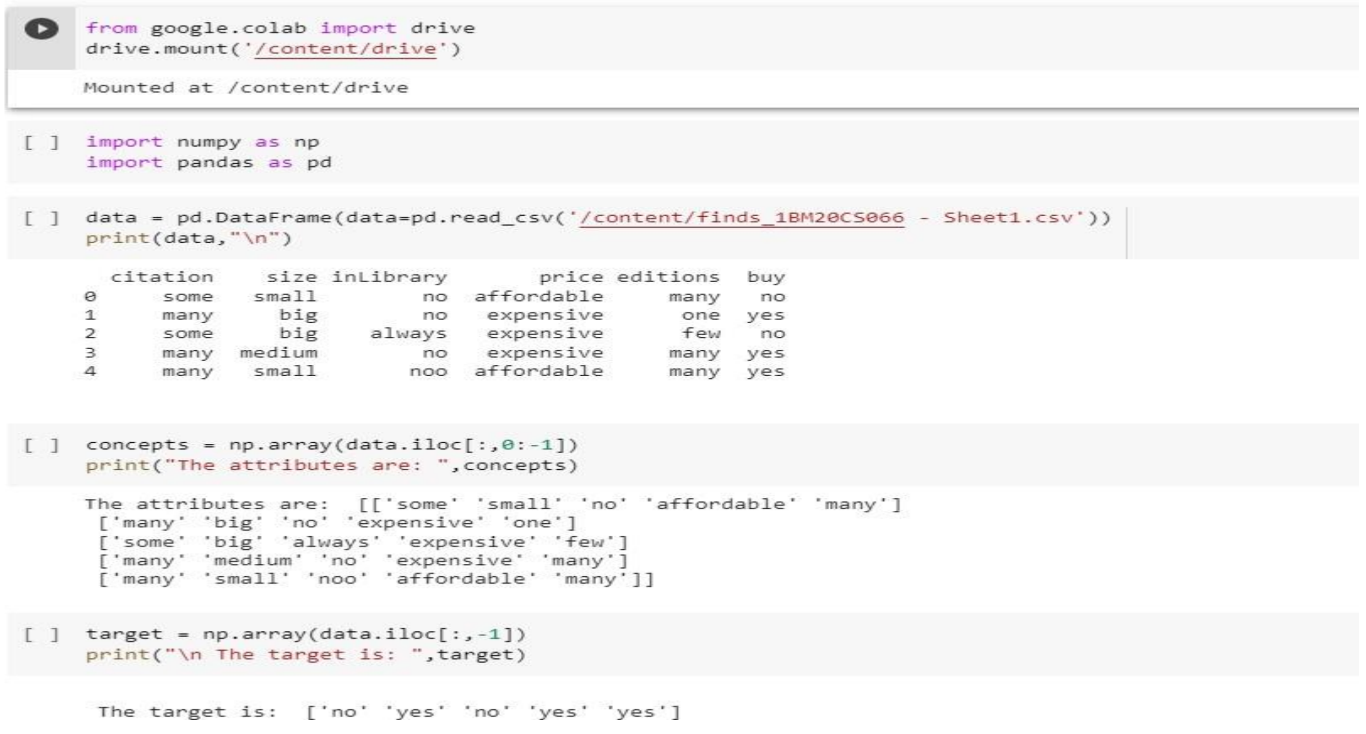


example	*citations*	*size*	*inLibrary*	*price*	*editions*	*buy*
1	some	small	no	affordable	many	no
2	many	big	no	expensive	one	yes
3	some	big	always	expensive	few	no
4	many	medium	no	expensive	many	yes
5	many	small	no	affordable	many	yes

CREATING CSV FILE:



	A	B	C	D	E	F
1	citation	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	noo	affordable	many	yes
7						
8						



```
from google.colab import drive
drive.mount('/content/drive')

Mounted at /content/drive
```

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

```
[ ]  data = pd.DataFrame(data=pd.read_csv('/content/finds_1BM20CS066 - Sheet1.csv'))
     print(data,"\n")

       citation    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       noo   affordable     many  yes
```

```
[ ]  concepts = np.array(data.iloc[:,0:-1])
     print("The attributes are: ",concepts)

     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' 'noo' 'affordable' 'many']]
```

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

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

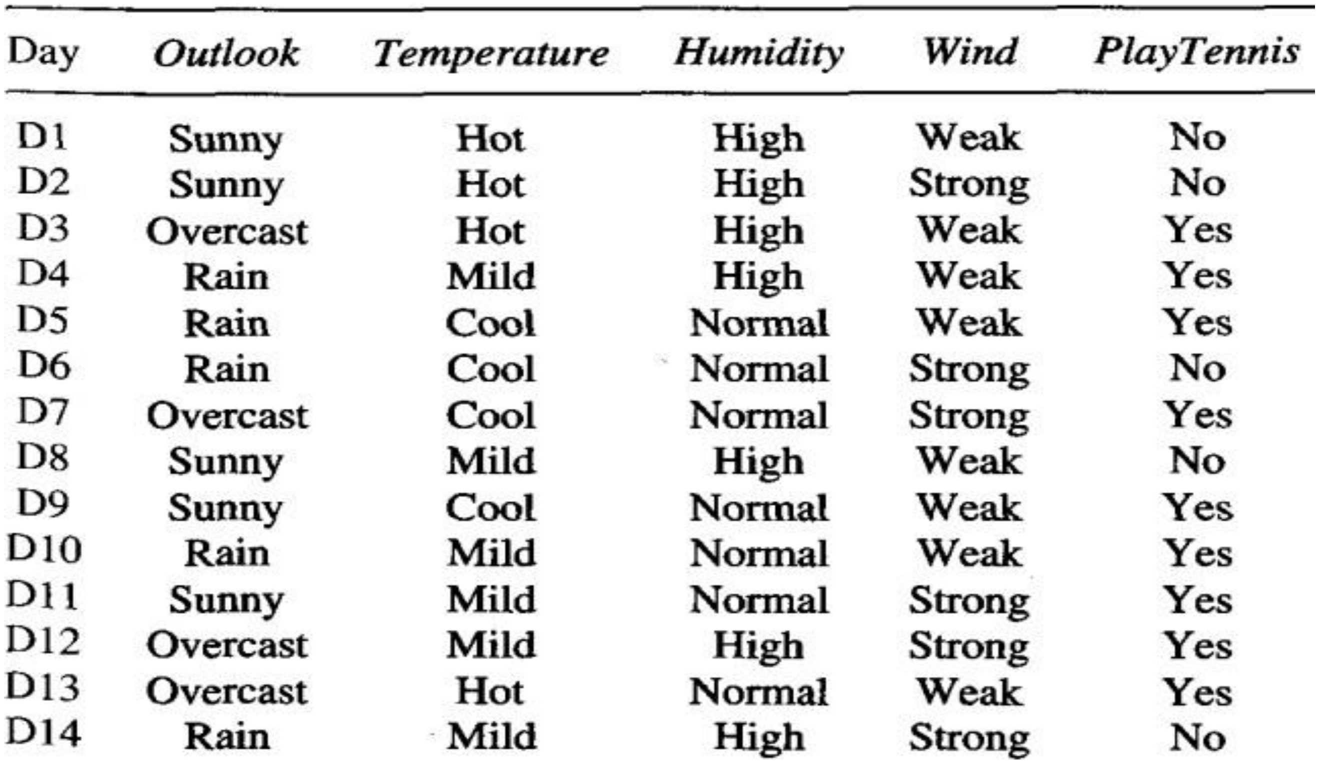
```
[ ] def learn(concepts, target):
      specific_h = concepts[B].copy()
      pn1nt("\n Initialization of specific h and ganenal h")
      print(specific_h)
      general_h = [["?" for i in range(len(specific_h))] for i in
     range(len(specific_h}}]
      pnint(general_h)
      for i, h in enumerate(concepts):
          if target[i] == "yes":
              for x innange(lan(specific_h)):
                  if h[x]!= specific_h[xj:
                      specific_h x] = ' ?'
                      general_h[xj[x] =' 2'
                  print(specific_h)
          print(specific_h}
          if target[i] == "no":
              for x in nange(lan(specific_h)):
                  lf h[x]!= specific_h[xj:
                      general_h[xj[xj = specific_h[x]
                  else:
                      general_h[xj[x] = '2'
          print("\n Steps of candidate Elimination Algorithm",i+1)
          print(specific_h}
          print(general_h}
      indices = i fon i, val in enumerate(genenal_h) if val ==

      for i in indices:
          genenal_h.remove(['2', '?', '?', ' 7', '?', ' 7'])
      ne £unn spec 1-F1c_h, genena1_h
     s_final, g_final = learn(concepts, target)
```

```
 Initi alization of spec ific_h and genera1_h
['some' 'small' 'no' 'df{Ordable' 'many']

['some' 'small' 'no' 'affordable' 'many']

 Steps of Candidate Elimination Algorithm 1
['some' 'small' 'no' 'affordable' 'many']
['?' 'small' 'no' 'affordable' 'many']
['*' '?' 'no' 'affordable' 'nanny'§
['*' '?' 'no' 'affordable' 'nag"]



 steps of candidate Elimination Vgorithm 2



 Steps of candidate Elimination Ggorithm 3



 steps of candidate Elimination Vgorithm 4



 3teps of candidate Ellninat ion Mgoritfzr S
```

```
pr1nt ("'.riFIn<l specIf1c_h : ", s_finaJ, sep="'-,n")
pr1nt ("'..riFIncl General h:",      final, sep="'..ri")
```

```
Fina] specific_h:


Final General h:
```

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.



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:

- Create a Root node for the tree
- If all Examples are positive, Return the single-node tree Root, with label = +
- If all Examples are negative, Return the single-node tree Root, with label = -
- If Attributes is empty, Return the single-node tree Root, with label = most common value of Target_attribute in Examples

  Otherwise Begin
- A ← the attribute from Attributes that best* classifies Examples
- The decision attribute for Root ← A
- For each possible value, v_i, of A,
- Add a new tree branch below *Root*, corresponding to the test A = v_i
- Let *Examples* $_{vi}$, be the subset of Examples that have value v_i for*A*
- If *Examples* $_{vi}$, is empty
- Then below this new branch add a leaf node with label = most common value of Target_attribute in Examples
- Else below this new branch add the subtree ID3(*Examples* $_{vi}$, Targe_tattribute, Attributes – {A}))
- End
- Return Root

19/4/23

Decision Tree - Play Glof

1)

Day	Outlook	Temp	Humidity	Wind	Play
1	Sunny	Hot	high	weak	No
2	Sunny	Hot	high	Strong	No
3	Overcast	Hot	high	weak	Yes
4	Rain	Mild	high	weak	Yes
5	Rain	Cool	Normal	weak	Yes
6	Rain	Cool	Normal.	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	high	weak	No
9	Sunny	Cool	Normal	weak	Yes
10	Rain	Mild	Normal	weak	Yes
11	Sunny	Mild.	Normal	Strong	Yes
12	Overcast	Mild	high	Strong	Yes
13	Overcast	hot	Normal	weak	Yes.
14	Rain	mild.	High.	Strong.	No

$= \left(-9/14(-0.6374)\right) + \left(-5/14(-1.9854)\right)$

$= (0.941)$

Information Gain: $G(S, A, T)$

$$= E(S) - \sum_{V \in Attri(A)} \frac{|S_V|}{S} E(S_V)$$

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

$$G(s, \text{temp}) = E(s) - \left[\frac{4}{14} * E(\text{temp}=\text{hot}) + \frac{6}{14} * E(\text{temp}=\text{Mild}) + \frac{9}{14} * E(\text{temp}=\text{Cool}) \right]$$

$$E(s) = \left[\frac{4}{14} * \left(\frac{-2}{4} \log_2 \frac{2}{4} \right) + \left(-\frac{2}{4} \log_2 \frac{2}{4} \right) + \right.$$

$$\left[\frac{6}{14} * \left(-\frac{4}{6} \log_2 \frac{4}{6} \right) + \left(-\frac{2}{6} \log_2 \frac{2}{6} \right) + \right.$$

$$\left[\frac{9}{14} * \left(\frac{-3}{4} \log_2 \frac{3}{4} \right) + \left(\frac{-1}{4} \log_2 \frac{1}{4} \right) \right]$$

$$= 0.94 - 0.969$$
$$= \boxed{0.029}$$

$$G(S, \text{Humidity}) = 0.151 \quad \text{Entropy}(s)$$
$$= -(7/14) \text{entropy}(S_{high}) - (7/14) \text{entropy}(S_{normal})$$
$$= 9.40 - (7/14).985 - (7/14).592$$
$$= \boxed{0.0489}$$

[illegible]
19/4/23

$$G(S, \text{Wind}) = \text{Entropy}(S) - (8/14) \text{entropy}(S_{weak}) - (6/14) \text{entropy}(S_{strong})$$
$$= 0.940 - (8/14) 0.811 - (6/14) 1$$
$$= \boxed{0.48}$$

Algorithm

ID3 (Example, Target-attribute, attribute)

* Create a root node for the tree.

* If all examples are +ve,
return the singlenode tree
Root with label = +ve.

* If all examples are -ve,
return the singlenode tree
Root with label = -ve.

* Otherwise Begin,

→ A ← the attribute from attributes that best*
Classifies examples.

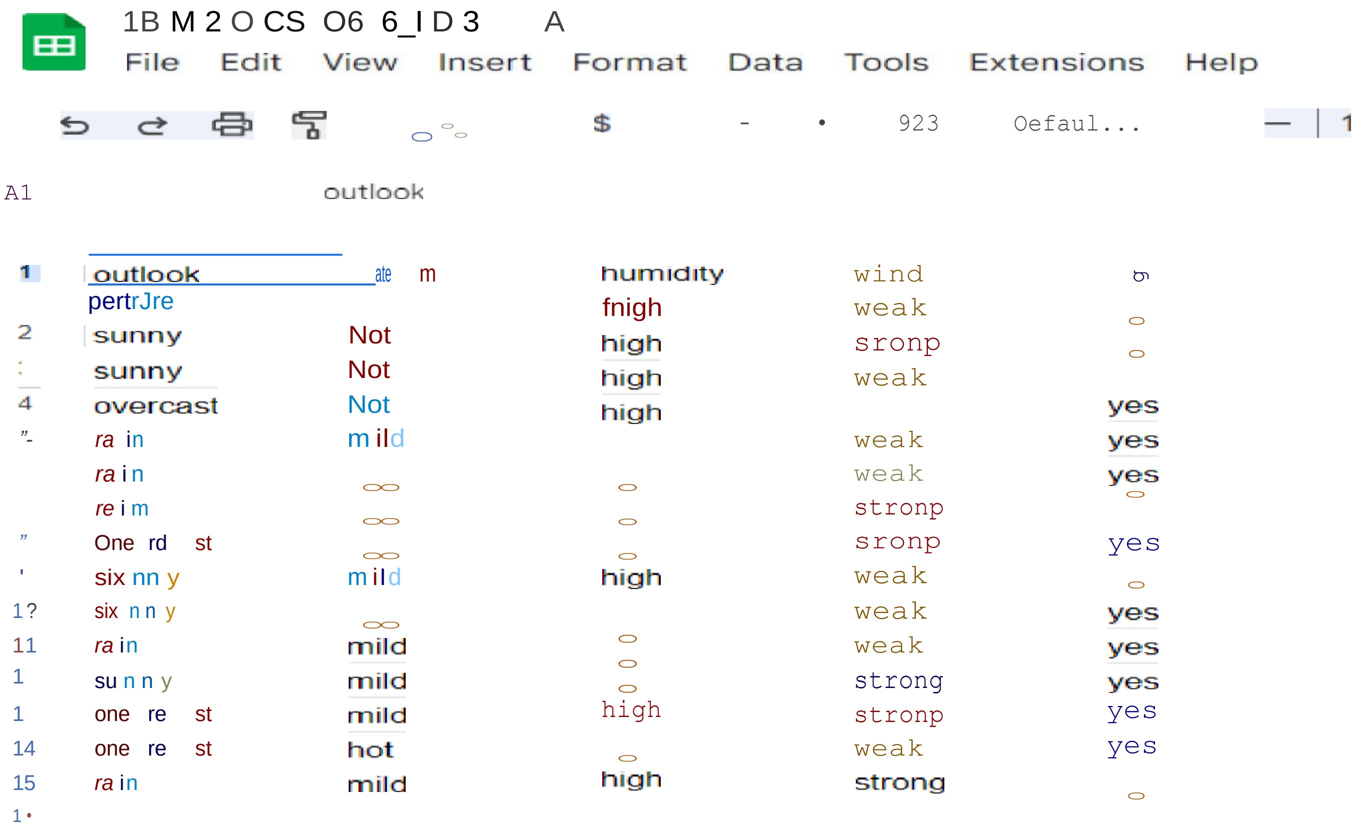
→ The decision attribute for root ← A

→ Add a new tree-branch below root, corresponding to the test A = Vi

→ Let example Vi, be the subset of example that have values Vi from A.

→ If example Vi, that is empty.

• Then a below this new branch at a leaf node with label S' most Common value of Target attribute in example.



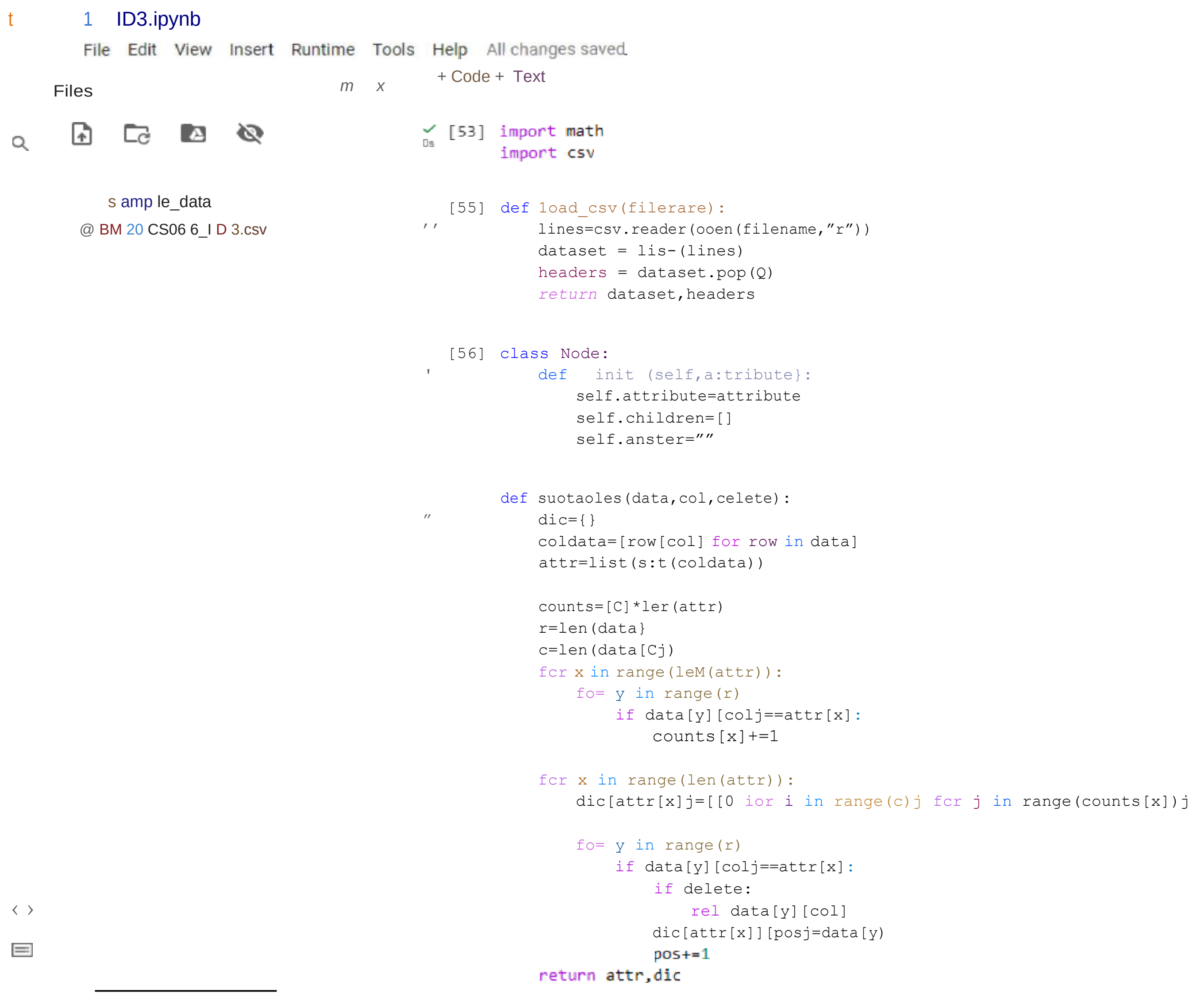
1B M 2 O CS O6 6_I D 3 A

File Edit View Insert Format Data Tools Extensions Help

$ - • 923 Oefaul... — | 1

A1 outlook

1	outlook	ate m	humidity	wind	ɞ
	pertrJre		fnigh	weak	⬭
2	sunny	Not	high	sronp	⬭
:	sunny	Not	high	weak	
4	overcast	Not	high		yes
”-	ra in	m ild		weak	yes
	ra i n	⬭⬭	⬭	weak	yes
	re i m	⬭⬭	⬭	stronp	⬭
”	One rd st	⬭⬭	⬭	stronp	yes
'	six nn y	m il d	high	weak	⬭
1?	six n n y	⬭⬭		weak	yes
11	ra in	mild	⬭	weak	yes
1	su n n y	mild	⬭	strong	yes
1	one re st	mild	high	stronp	yes
14	one re st	hot	⬭	weak	yes
15	ra in	mild	high	strong	⬭
1•					



```
[53] import math
     import csv

[55] def load_csv(filerare):
         lines=csv.reader(ooen(filename,"r"))
         dataset = lis-(lines)
         headers = dataset.pop(Q)
         return dataset,headers

[56] class Node:
         def   init (self,a:tribute):
             self.attribute=attribute
             self.children=[]
             self.anster=""

     def suotaoles(data,col,celete):
         dic={}
         coldata=[row[col] for row in data]
         attr=list(s:t(coldata))

         counts=[C]*ler(attr)
         r=len(data}
         c=len(data[Cj)
         fcr x in range(leM(attr)):
             fo= y in range(r)
                 if data[y][colj==attr[x]:
                     counts[x]+=1

         fcr x in range(len(attr)):
             dic[attr[x]j=[[0 ior i in range(c)j fcr j in range(counts[x])j

             fo= y in range(r)
                 if data[y][colj==attr[x]:
                     if delete:
                         rel data[y][col]
                     dic[attr[x]][posj=data[y)
                     pos+=1
         return attr,dic
```

```
  {5B] defentropv(S):
"         attr=list(set(5))
          i-f len(attr)==1:
              return 0

          counts=[8,0]
          fcr i in range(2):
              counts[i]sum[1 for x in S lf attr[l]=x)7(ens)*v0)

          sums=0
          fcr cnt in counts:
              sumse=-l*cnt°math.log(cnt,2)
          return sums


   59] del conpute aln( data, ccl) :
“         attn,d1c = subtabl es{data,co1,deIete=FaJse )

          total_si   e=1en{data)
          ent nop1es=[e] ”ten(attr)
          ratio=[ej ”1en{ attr)

          tot a1_ent ropy=en tropy ( [root [ -1 ] -L-c i root 1 n dat a ] }
          Tc i- x 1n r ange (Jen( att r) ) :
              ratio[x§=Len (d1c [ attr[x§ ] )/(tota1_si e•z . e}
              endrop1es [x§ =entropy([ row[ -1]   for   row 1n dbe[attr[x§§])
              tot  al_entrapy-=ratto[x § ” entropYes[x§
          r .°t urn tot a1 entropy


  [die] de-L- bu11d_tr e e (dat a, -L-eat ures) :
“         1a st co1= [rohi[ -1§ 'or  rohi 1n d ata]
          If  (1en{set(1asts o1)))==1:
              node= Node { "" )
              node.ans\‹er=lastcot[0]
              retui- n node

          n= be n(dat a [e] ) -1
          gains=[e]’n
          fc+- col in range(n):
              gains[col]=compute_gain(data,col)
          spllt=gains.index(max(gains)}
          node=Node(features[split])
          fea = features[:spl*t] features[split+1:]


          att r, die= s ubt a b I es (d at a, sp lit, de lete=T rue )

          dci- x In range (ten( attr) ) :
              ch11d=bu11d_tree{d1c [attn[x] ],-Eea)
              node.ch11dren.append((attr[x],ch11d))
          return node
```

```
                                ”*level,node.attc*bute)
                         value,n in node.children:
```

```
[62] def c lassify (node, x  test,features):
         if node . answer ! - "" :
             pr1rt(node. answer)

         pos•featrres. index(rode. attribute)
         for value, n ir node.ch11drer :
             if   X tg5t t §0S]-- Vd) Ug I
                 classify(n,x test,features)


     dataset, features=load_csv("1Bh2@S866 ID3. csv")
     node1=bu11d tree(dataset,features)

     gr1nt("The dec1s1or tree for the dataset using IDA algorithm is")
     pr1nt  tree(rode1,e)
     testdata,features=load_csv("1Bh28CS066_ID3.csv")

     for x/est i testdata:
         print("The test Instance: ",xtest)
         print("The label for test instance: ")
         classify(node1,xtest,features)
```

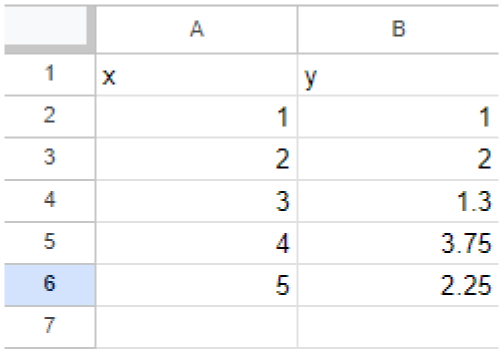
```
ltt itciiion trtt f0F ttt iatastt vsint iDi élj0ritlz is
  Ootloot




        ii?00$
          80
     sunny
        humidity

          no
        normal


     0VtFC8St
```

```
The te5t instance: ['sunny', 'hot', 'high'  'weak', 'no']
The label for test instance:

The test instance: ['sunny', 'hot', 'high', 'strong', 'no'j
The label for test instance:

The test instance: ['overcast', 'hot', 'high', 'neak', 'yes']
The label for test instance:
yes
The te5t instance: ['rain', 'mild', 'high'  ' eak', 'yes'j
The label for test instance:
yes
The te5t instance: ['rain', 'cool', 'normal', 'weak', 'yes'}
rhe label for test instance:
yes
The te5t instance: ['rain', 'cool', 'normal', 'strong'  'no']
The label Cor test instance:

The te5t instance: ['overcast , 'cool', 'normal'  'strong', 'yes'}
The label for test instance:
yes
The test instance:  t sunny,'   '•i  Ld'    high',    iiea k', 'no' J
The label for test instance:

The test instance: ['sunny',  cool', 'normal', 'weak', 'yes']
The label for test instance:
yes
The te5t instance: ['rain', 'mild', 'normal', 'weak', 'yes'}
rhe label for test instance:
yes
```

```
ThR tRst instance: ['sunny', 'mlld', 'normal', 'strong', 'yes']
ThR ldbel for tRst instance:
yes
ThR tRst instance: ['overcast', 'mild', 'high', 'strong', 'yes']
ThR ldbel for tRst instance:
yes
ThR tRst instance: ['overcast', 'hot', 'normal', 'weak', 'yes']
ThR ldbel for tRst instance:
yes
ThR tRst instance: ['rain', 'mild', 'high', 'strong', 'no']
The label for test instance:
no
```

PROGRAM 5: Simple linear regression program

Dataset used:



	A	B
1	x	y
2	1	1
3	2	2
4	3	1.3
5	4	3.75
6	5	2.25
7		

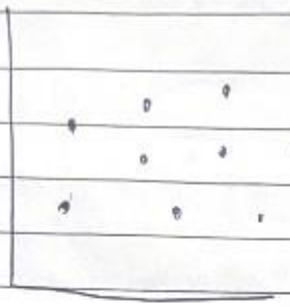
ALGORITHM:

- The main function to calculate values of coefficients
- Initialize the parameters.
- Predict the value of a dependent variable by giving an independent variable.
- Calculate the error in prediction for all data points.
- Calculate partial derivatives w.r.t a0 and a1.
- Calculate the cost for each number and add them.
- Update the values of a0 and a1.

Linear Regression

$$b_0 = \frac{(\sum_i y)(\sum_i x^2) - (\sum_i x)(\sum_i xy)}{n(\sum_i x^2) - (\sum_i x)^2}$$

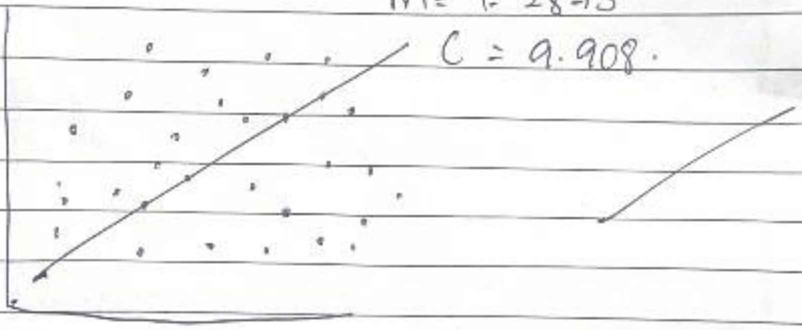
$$b_1 = \frac{n(\sum_i xy) - (\sum_i x)(\sum_i y)}{n(\sum_i x^2) - (\sum_i x)^2}.$$



$Y = Mx + C.$

$M = 1.2873$

$C = 9.908.$



A straight-line equation involving slope (dy/dx) and Y-intercept

$Y = Mx + C.$

y = dependent value of x.

$Y = b_1 x_i + b_0.$

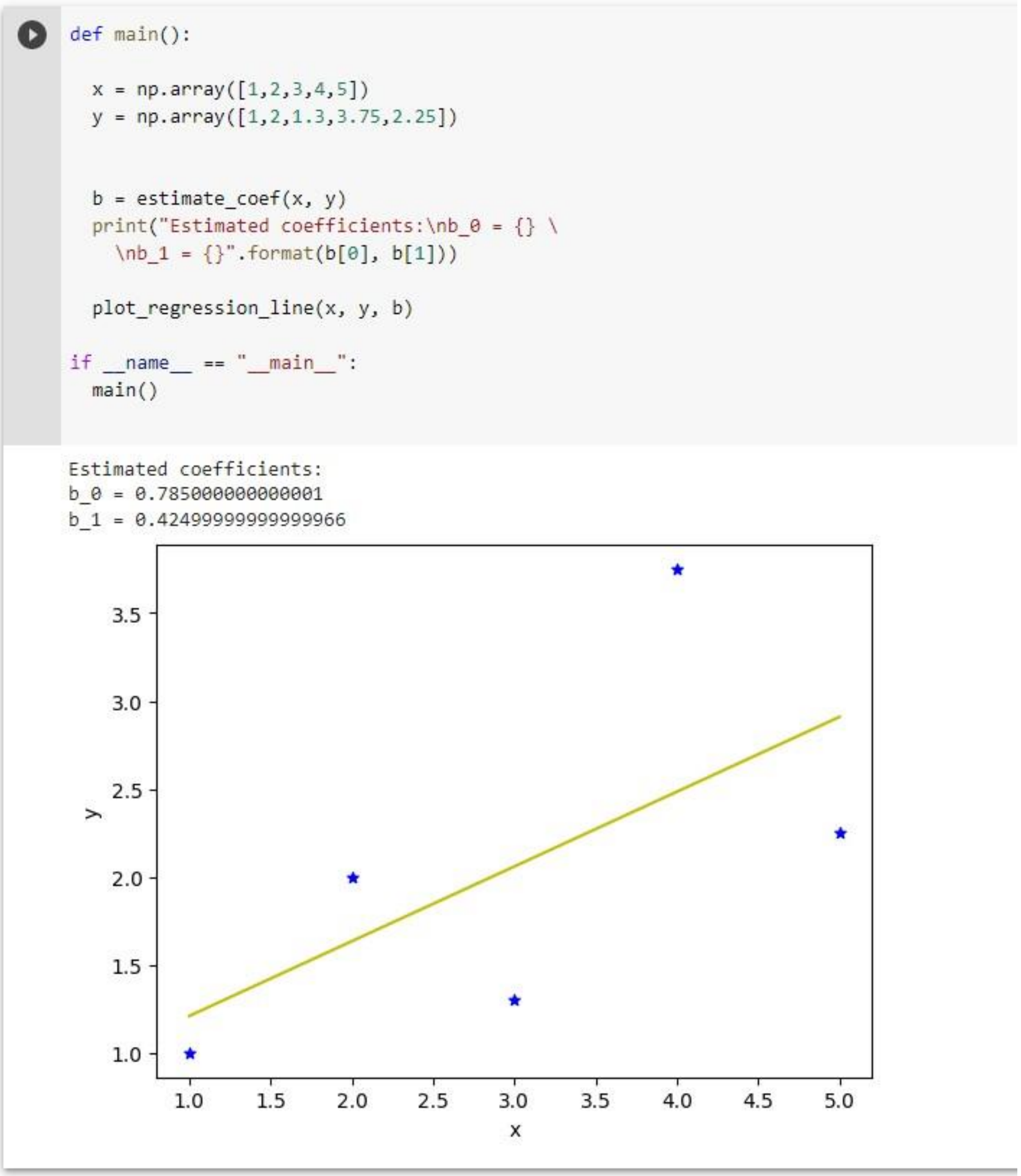
Y_i = Predicted Y value for observation

b_0 = Estimate of Regression intercept.

b_i = Estimate of regression slope.

x_i = Input.

```
[ j  import          as np
     import matplotllb.pyplot as pLt

[ ]  def plot_regression_line(x, y, b):

       plt.scatter(x, y,
                                       ,"m"

                    b[0] + b[1]*x


       p La . p 1 a 1 (,x    y_ p r' e d              "g")


       pLt. xlabel ( 'x  CA-EFF ' )
       pLt.ylabel ( 'y• CA-EFF' )


       p La  .s  h a » ( )




       n = np.size(x)


       m_x   = n p . me a n ( x )
       my'   = n p . me a n ( y )

       S    X\'  =  II  . '5 LI IT  y "x )   —  n " my   "m_x
       S    X X  =  II  . '5 LI IT  x "x )   — n "m_x" m_x

       b_1 = S S_xy / SS_xx
       b_B  =  my'   – b_  1* m_x

       return (b_0, b_1)


 ▌    beef plot_cegcession_lzne(^   y, b)
       plt.scatter(x, y, color = "b",
           marker = "*", s = 30)


                b$C] + b[ij*x


                               ㅁ


       pLt.xlabel('x')
       pLt.ylabel('y')
```



```
def main():

  x = np.array([1,2,3,4,5])
  y = np.array([1,2,1.3,3.75,2.25])


  b = estimate_coef(x, y)
  print("Estimated coefficients:\nb_0 = {} \
    \nb_1 = {}".format(b[0], b[1]))

  plot_regression_line(x, y, b)

if __name__ == "__main__":
  main()
```

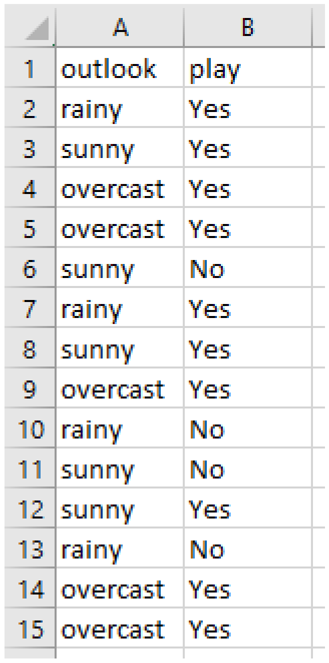
```
Estimated coefficients:
b_0 = 0.785000000000001
b_1 = 0.42499999999999966
```

Conclusion:

This model is not appropriate for this model. All the points of this dataset are away from the prediction line.

Program 6:Write a program to implement the naïve Bayesian classifier for a sampl training data set stored as a .CSV file. Compute the accuracy of the classifie considering few tes data sets.

Data set used:

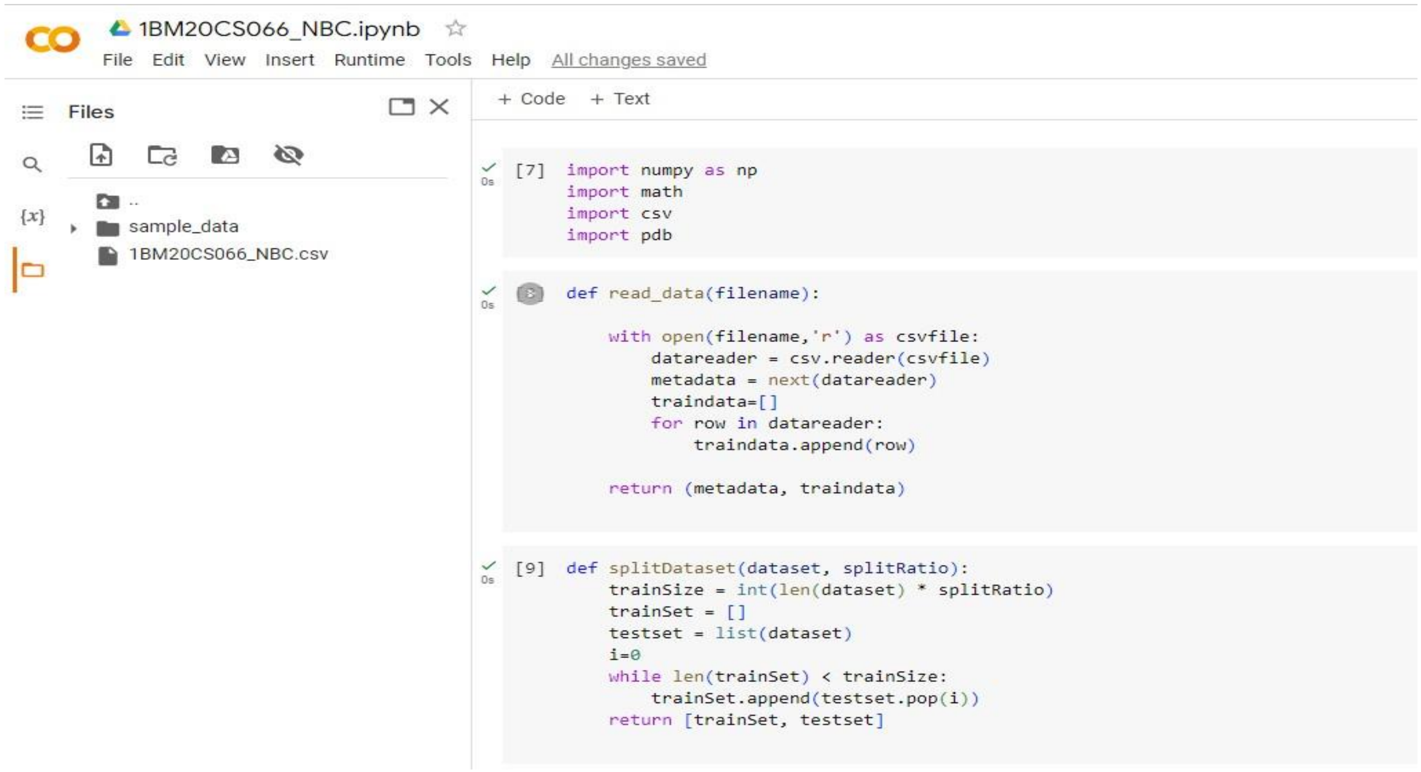


	A	B
1	outlook	play
2	rainy	Yes
3	sunny	Yes
4	overcast	Yes
5	overcast	Yes
6	sunny	No
7	rainy	Yes
8	sunny	Yes
9	overcast	Yes
10	rainy	No
11	sunny	No
12	sunny	Yes
13	rainy	No
14	overcast	Yes
15	overcast	Yes

Algorithm:

Formula for naive bayes classifier is as follows→ $P(A|B)=\frac{P(B|A)P(A)}{P(B)}$

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.
4. Test accuracy of the result and visualizing the test set result.

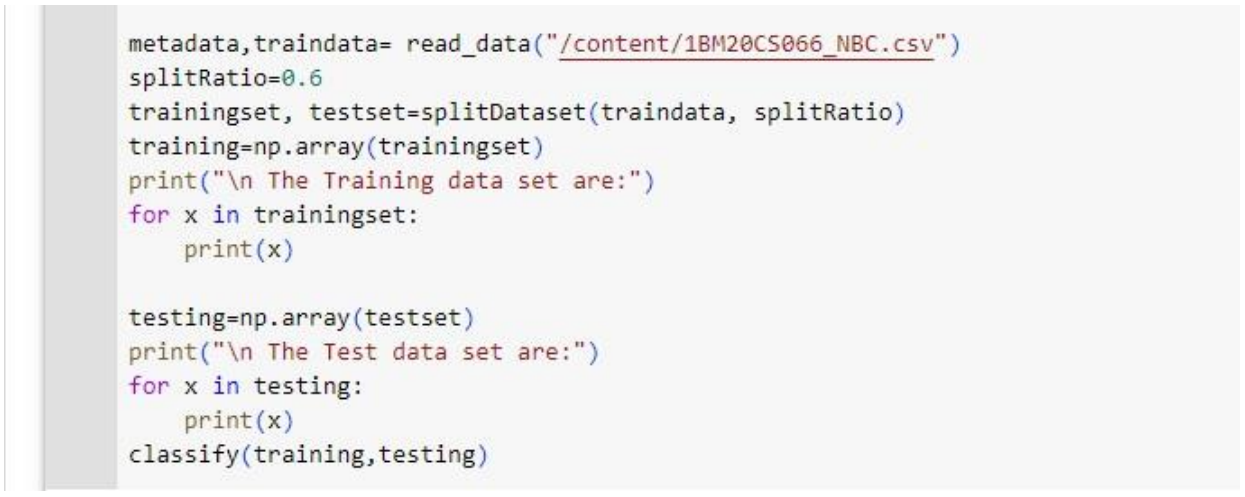


```
[7] import numpy as np
    import math
    import csv
    import pdb
```

```
def read_data(filename):

    with open(filename,'r') as csvfile:
        datareader = csv.reader(csvfile)
        metadata = next(datareader)
        traindata=[]
        for row in datareader:
            traindata.append(row)

    return (metadata, traindata)
```

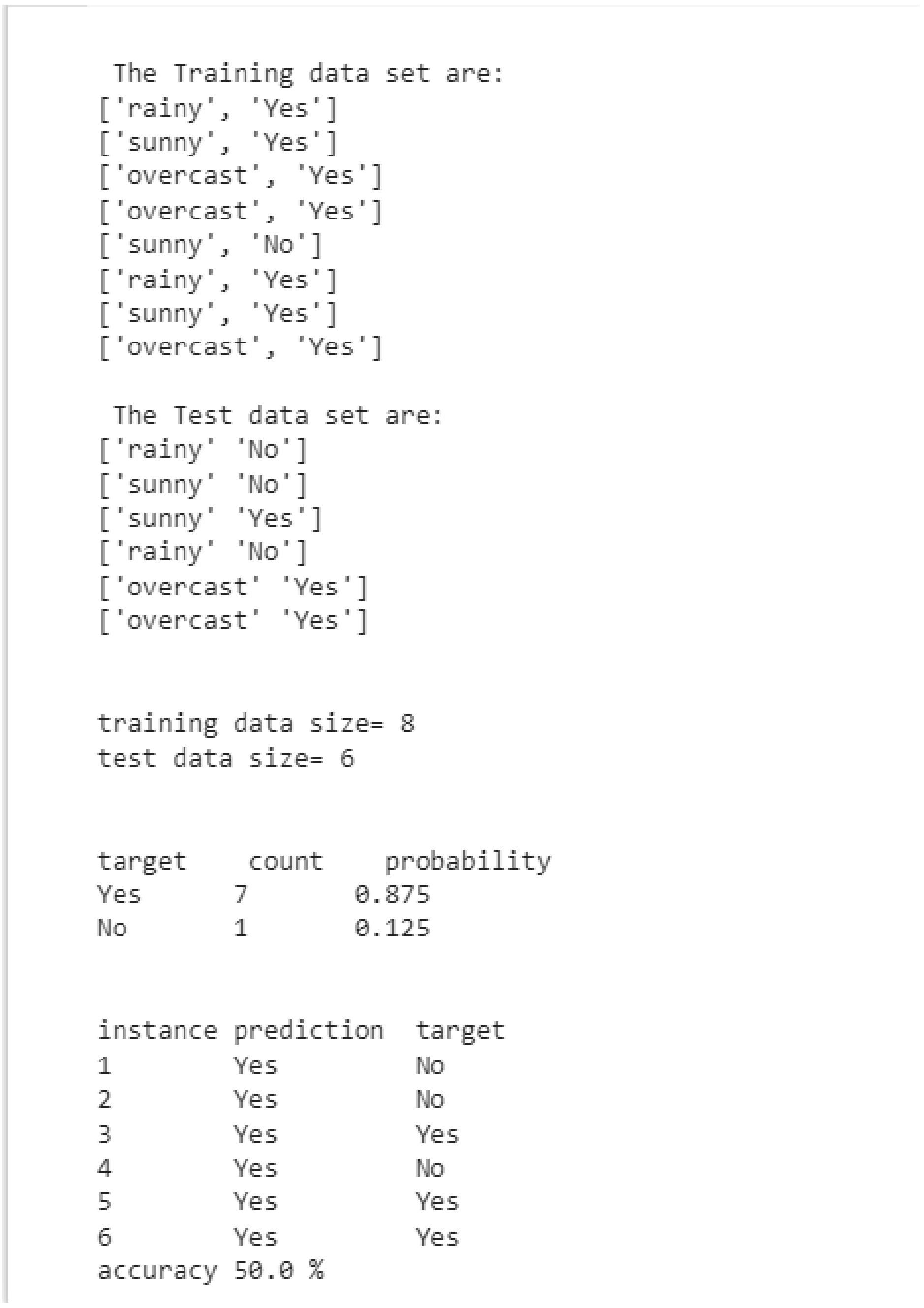
```
[9] def splitDataset(dataset, splitRatio):
        trainSize = int(len(dataset) * splitRatio)
        trainSet = []
        testset = list(dataset)
        i=0
        while len(trainSet) < trainSize:
            trainSet.append(testset.pop(i))
        return [trainSet, testset]
```

```
0       def classify(data,test):

            total_size = data.shape 0]
            print("\n")
            pnint("tnaining data size=",total size)
            pnint("test datasize=",test.shape[0]}

            countYes = 0
            countNo  = B
            probYes  = B
            probNo    0
            pnint("\n")
            pnint("target     count    probability")

            {or x in range(data.shape[B]):
                if data[x,data.shape[1]-1] == 'Yes' :
                    COuntYeS +=1
                if data[x,data.shape[1]-1] == 'No':
                    countNo +=1

            probYes=countYes/total_size
            probEo= countNo / tOtal_size

            print('Yes',"\t",countYes,"\t",probYes)
            print('No','\t',countEo,'\t',probNo)


            poob8 -np . ze nos ( (test .shape [1 ] -l) )
            poobl -np . ze ros ( (test .shape [1 ] - l) )
            accuracy=0
            print("\n")
            print("instance prediction  target")

            {on t in range(test.shape[0]):
                for k in range (test.shape lj-1):
                    count1=count0=0
                    for j in range (data.shape[0]}:
                        Show many times appeared with no
                        if test[t,k] == data[j,k] and data[j,data.shape[1]-1]=='No':
                            count0+=l
                        Show many times appeared with yes
                        if test[t,k]==data j,k] and data[j,data.shape[1]-lj=='Yes':
                            count1+=l



                pr O bn O = p ra bN O
                pr O by es = p mobves
                for i in range(test.shape§ij-&):
                       _bno    o  o
                       _bye    ≥  V
                if probno>probyes:


                    predict='Yes'

                print(t+i."\t",predict."\t     ",test§t,test.sñapej1j-lj)
                if predict == test[t,test.shape[l]-lj:
                    accuracy+=i
            final accuracy=(accuracy/test.shape[Bj)*lGB
            print("accuracy".final_accuracy,"n")
```



```
metadata,traindata= read_data("/content/1BM20CS066_NBC.csv")
splitRatio=0.6
trainingset, testset=splitDataset(traindata, splitRatio)
training=np.array(trainingset)
print("\n The Training data set are:")
for x in trainingset:
    print(x)

testing=np.array(testset)
print("\n The Test data set are:")
for x in testing:
    print(x)
classify(training,testing)
```

output:



```
 The Training data set are:
['rainy', 'Yes']
['sunny', 'Yes']
['overcast', 'Yes']
['overcast', 'Yes']
['sunny', 'No']
['rainy', 'Yes']
['sunny', 'Yes']
['overcast', 'Yes']

 The Test data set are:
['rainy' 'No']
['sunny' 'No']
['sunny' 'Yes']
['rainy' 'No']
['overcast' 'Yes']
['overcast' 'Yes']


training data size= 8
test data size= 6


target    count    probability
Yes      7       0.875
No       1       0.125


instance prediction  target
1        Yes         No
2        Yes         No
3        Yes         Yes
4        Yes         No
5        Yes         Yes
6        Yes         Yes
accuracy 50.0 %
```

Naive Baye's

Training Dataset

Color Type Origin Stolen.

[Red, Sports, domestic, Yes]
[Red, Sports, Domestic, No]
[Red, Sports, Domestic, Yes]
[Yellow, Sports, Domestic, No]
[Yellow, Sports, Imported, Yes]
[Yellow, SUV, Imported, No]

Size = 6

Test Data Set:

Color	Type	Origin	Stolen.
Yellow	SUV	imported	Yes
Yellow	SUV	Domestic	No
Red	SUV	Imported	No
Red	Sports	Imported	Yes.

Size = 4

Target	Count	Probability.
Yes	3	1/2
No	3	1/2

Instance	Prediction	Target.
1	No	Yes
2	No	No
3	No	No
4	Yes	Yes.

Accuracy : 75.0%

$$P(h|D) = \frac{P(D|h) \cdot P(h)}{P(D)}$$

$P(h|D)$ = Posterior Probability

$P(h)$ = Prior Probability

$P(D)$ = Probability over data set

$P(D|h)$ = Current Probability.

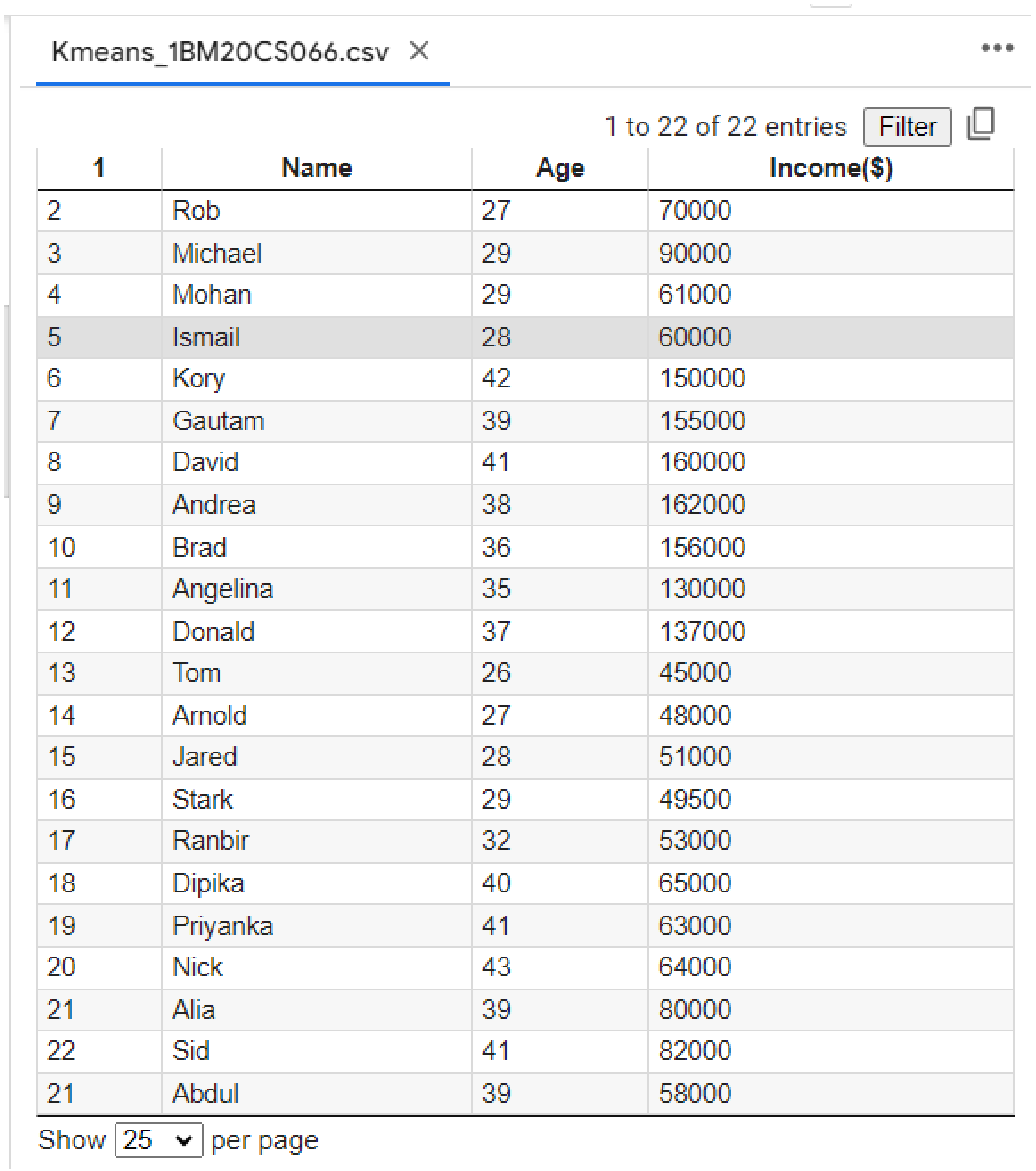
O/p [illegible]
12/5/23

Program 7:K- means clustering

Algorithm:

```
Initialize k means with random values
For a given number of iterations:
 Iterate through items:
 Find the mean closest to the item by calculating the euclidean distance of the item with each of the means
 Assign item to mean
 Update mean by shifting it to the average of the items in that cluster
```

Dataset:



Kmeans_1BM20CS066.csv

1 to 22 of 22 entries Filter

1	Name	Age	Income($)
2	Rob	27	70000
3	Michael	29	90000
4	Mohan	29	61000
5	Ismail	28	60000
6	Kory	42	150000
7	Gautam	39	155000
8	David	41	160000
9	Andrea	38	162000
10	Brad	36	156000
11	Angelina	35	130000
12	Donald	37	137000
13	Tom	26	45000
14	Arnold	27	48000
15	Jared	28	51000
16	Stark	29	49500
17	Ranbir	32	53000
18	Dipika	40	65000
19	Priyanka	41	63000
20	Nick	43	64000
21	Alia	39	80000
22	Sid	41	82000
21	Abdul	39	58000

Show 25 per page

K-means Algorithm

① Select the number K to decide the number of clusters.

② select random K points or centroids.

③ Assign each data point to their closest centroid which will form the Predefined K cluster.

④ Calculate the variance and new place centroid of each cluster.

⑤ Repeat the third steps, which means re-assign each datapoint to new closest centroid

⑥ If any re-assignment occurs, go to step 4 else FINISH

⑦ Model is ready.

GMM - Gausian Mixture model.

[illegible]
9/6/23

```
[1 ]  impo-t  pandas   s pcI
      frcm   sklearn.cluster          KMeans
             sklearn.preprocessing i+rpcrt {linhlaxscaler
      rom    matplotlib iirpcr-: pyplot as pit
      rom
      %matplotlib inline
```

```
[Z]   df =  pd.read_csv('./content,'Kweans 1B›23CSB6o. cs ' ' }
      df. head(1G}
```

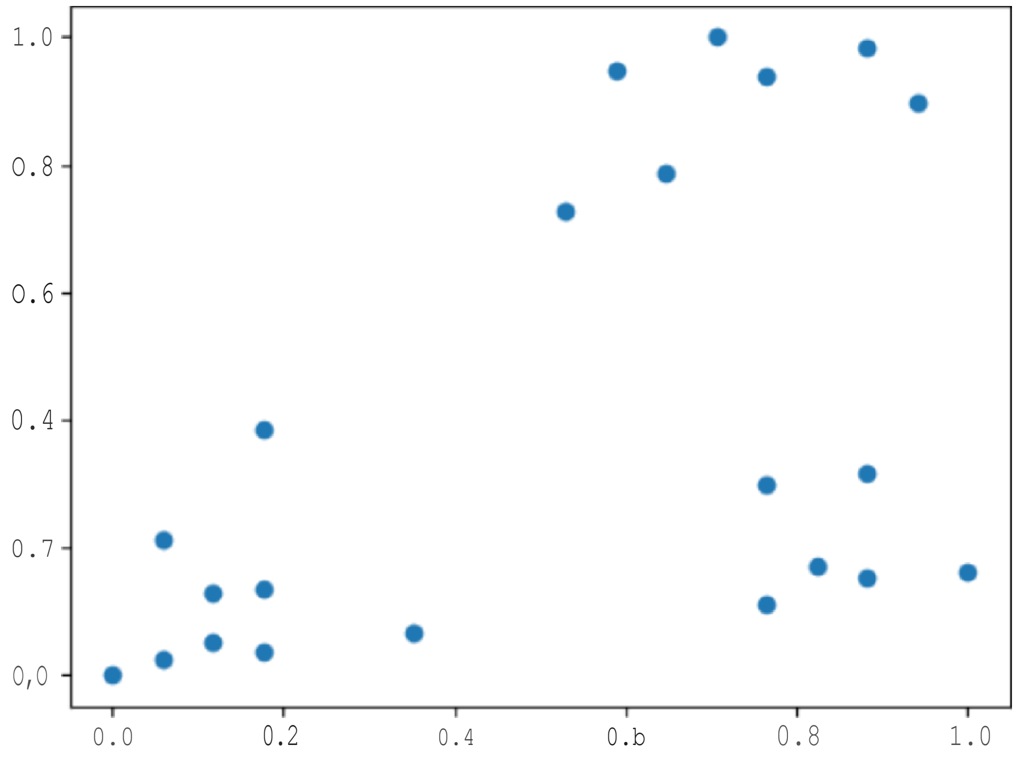
```
        1         Name    Age   I n c ome $ )

  O     2          Rob     27          YOOOO
  1     3     I'dichael    2g
  2     4        f'7ohan   29          61OOO
  3     5       Ism ail   28          60000
  4     G         Kory     42        1 SOOOO
  5     7      Gautam     39        1 SSOOO
  6     8        Darid     41        16OOOO
  7     g      Andre a    3B        162000
  B    1Q        B ra d    3G        1SGOOO
  9    11   Ang e li n a   35        13OOOO
```

```
[4]   scaler  = hlinMaxScaler()
      scaler.fit(df[['Age']j)
      dv[['age' ] j  = scaler.transform(df[['Age'  j])

      scaler.fit(df[[ ' I n c a me ( â ) "]])
      d-F[  'I n c ome ( g } '  ]     s cafes . 1z  an  s la  mm( d I [ [ 'Zn c ame ( $ ) " ] ] )
      d -F . h ea d ( 1U }
```

```
       1          Name           Ag e   Z n c cm e ( $ )

  0    2           Rob      D. 05 B824      O .2 13B75
  1    3      I'dichael     D. 17 64T 1    O .3B4B 15
  2    4       f'7o h an     D. 17 64T 1     O . 136T52
  3    5         Ismail     O. 117647     O . 1 282 05
  4    6          Kory    O. 94 11 VG       0 .897436
  5    7       Gautam     D.76 47DG      0.94017 1
  6    8         David     0.882353       0  .982906
  7    9      Andre a    D.7O 58B2       1.ODOD0O
  B   1 0        Bra d     D. 58 B235    0.948 T 1 8
  9   11    An g eli n a   D. 52 94 1 2      O .726496
```

```
plt.scatter(df['Age'], df['IncDme($)'])
```

```
[• « matplotlib.collectiDns.PathCollection at 0x7fA3820d1a50>
```

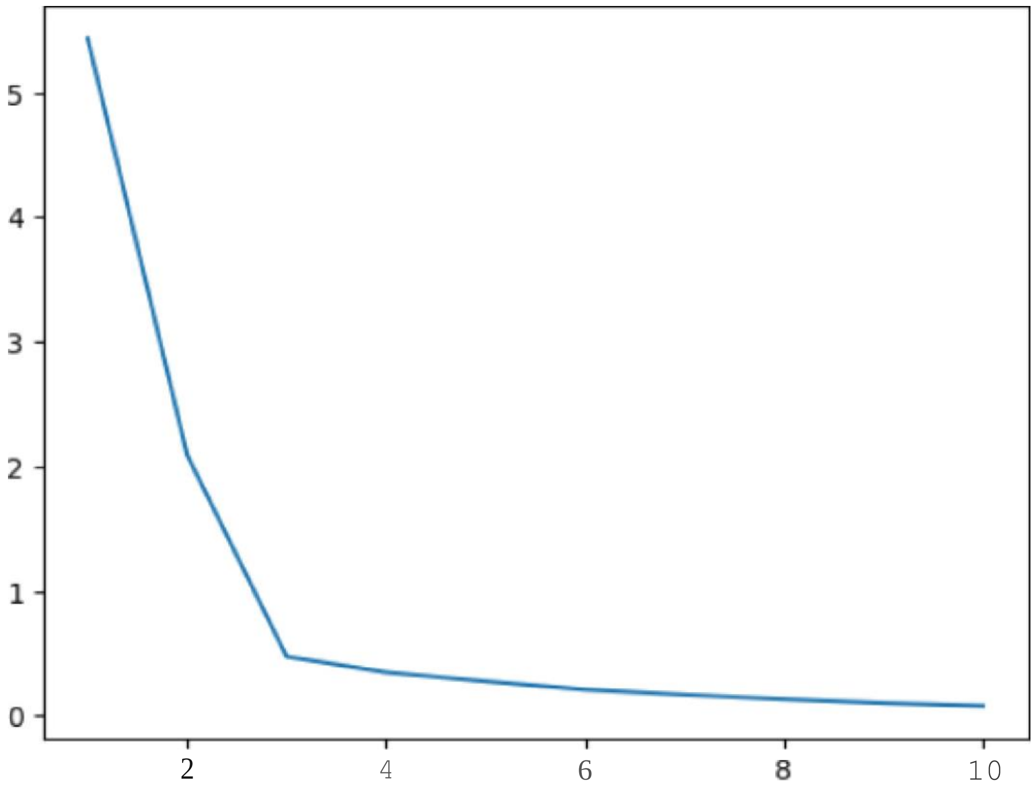


```
k_range = range(l, 11)
sse = []
for k in k range:
    kmc = KMeans(n_clusters=k}
    kmc.fit(df[['Age , 'Income(6)']])
    s s e . a ppend (kmc . 1 ne nt1a_)
sse
```

```
 [5.434011511988178,
  2.091136388699078,
  B47507834985538B96,
  B.3491047094419566,
  B.2798062931046179,
  B.22037641690770B67,
  B16858512236B2976,
  B.13265419827245162,
  B.1038375258660356,
   .&g5 &915216361345]
```

```
H  plt.xlabel   'Mumber of Clusters'
   plt.yLabeL   'Sum of Squared Errors '
   plt.pLot(k_range, sse)
```

```
[<matplotlib.lines.Line2D at Ox7f438G0TaSeO>]
```



```
[8]  km    KNeans(n_clusters=3)
     km
```

```
▾ KMeans
KMeans(n_clusters=3)
```

```
y_predict - km.fit_predict(df[['Age', 'Income($)']])
y_predict
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870  Futunel•.'arn•tng:Thede-FauTtva°lue   of `n init` will change
  warnings.warn(
arrav([1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 2. 2. 2. 2. 2. *J
```

0	2	'o O	0. 058 B2 4	0. 2 36 7	0
		Michael			
Z	4	Mohan	0.176471	0.136752	
3	5		0.117647	0.128205	
4	6	N o ry	0 9 4 Z O	0.B9Z436	0
4	6	K o ry	0 9 41 4 Z€	0.897H 26	0
5	7	Gautam	0.764706	0.940171	0
6	8	Oavid	O.882308	0.982906	0
8	10	Brad	0 5882 35	0 94 8718	0

```
[12] df1    df[df.cluster == 1]
     df1
```

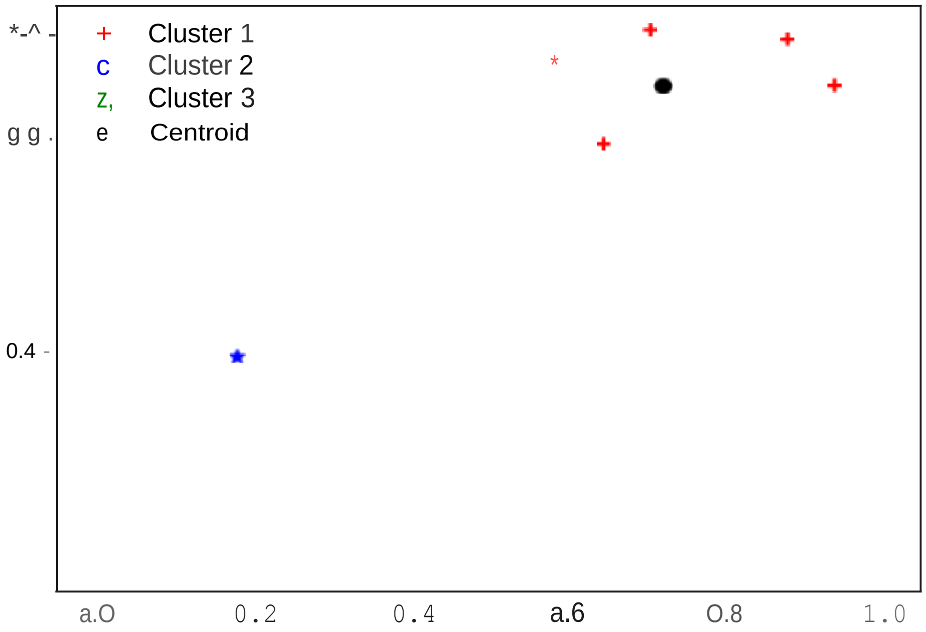
	[	Name	Age	Income(#)	cluster
0	2	Rob	0.058824	0.213G75	1
1	3	I.1ichael	0.17G471	0.394G15	1
2	4	hlohan	0.17G471	0.136752	1
3	5	Ismail	0.117647	0.128205	1
11	13	Tom	0.0D0000	0.DOOOOD	1
12	14	Anne Id	0.059824	0. C'25G41	1
13	15	Jared	0.117647	0.D51282	1
14	16	Stark	0 17G471	0 C'38462	1
15	17	Ranbir	0.352941	0.DG8376	1

```
[13] df2    df[df.cluster    2]
     df2
```

	I	Name	Age	Income($)	cluster
16	18	D ipika	0.823529	D. 17D940	2
17	19	Priyanka	0.882353	C. 15394G	2
18	20	Nick	1.0DOD00	D. 162393	2
19	21	Alia	0 764706	C 290145	2
20	22	Sid	0.882353	D. 316239	2
21	21	Abdul	0.764706	O. 1111a 1	2

```
[14] km. cluster_centers_

     array([[0.722689B8, 0.8974359 ],
            [0.1372549 , 0.11633428],
            [0.g52 4118, 0.2022752 ]])
```

```
[17] p1 = p11z.scatt er‘(d-£0 'Age'    df6 ['Income ($)'], oar‘ke r‘='+'   motor='r‘ed')
     p2 = p1t:.scat:ter‘(d-f1 'Age'    d-FI['Income ($)'], oar‘ke r‘='"'   color='blue')
     p3 = pit.scatter(dfZ['Age'j, df2['Income($)'], marker="', color='green')
     c   plt.scatter(km.cluster_centers_[:,Bj, km.cluster_centers_[:,1],color='black')
     pit.legend((p1, p2, p3, c),
               ( Cluster 1', 'Cluster 2', 'Cluster 3', 'Centroid '))

<matp1at 11b.legend.Legend at Bx7-F437d4c73aB>
```

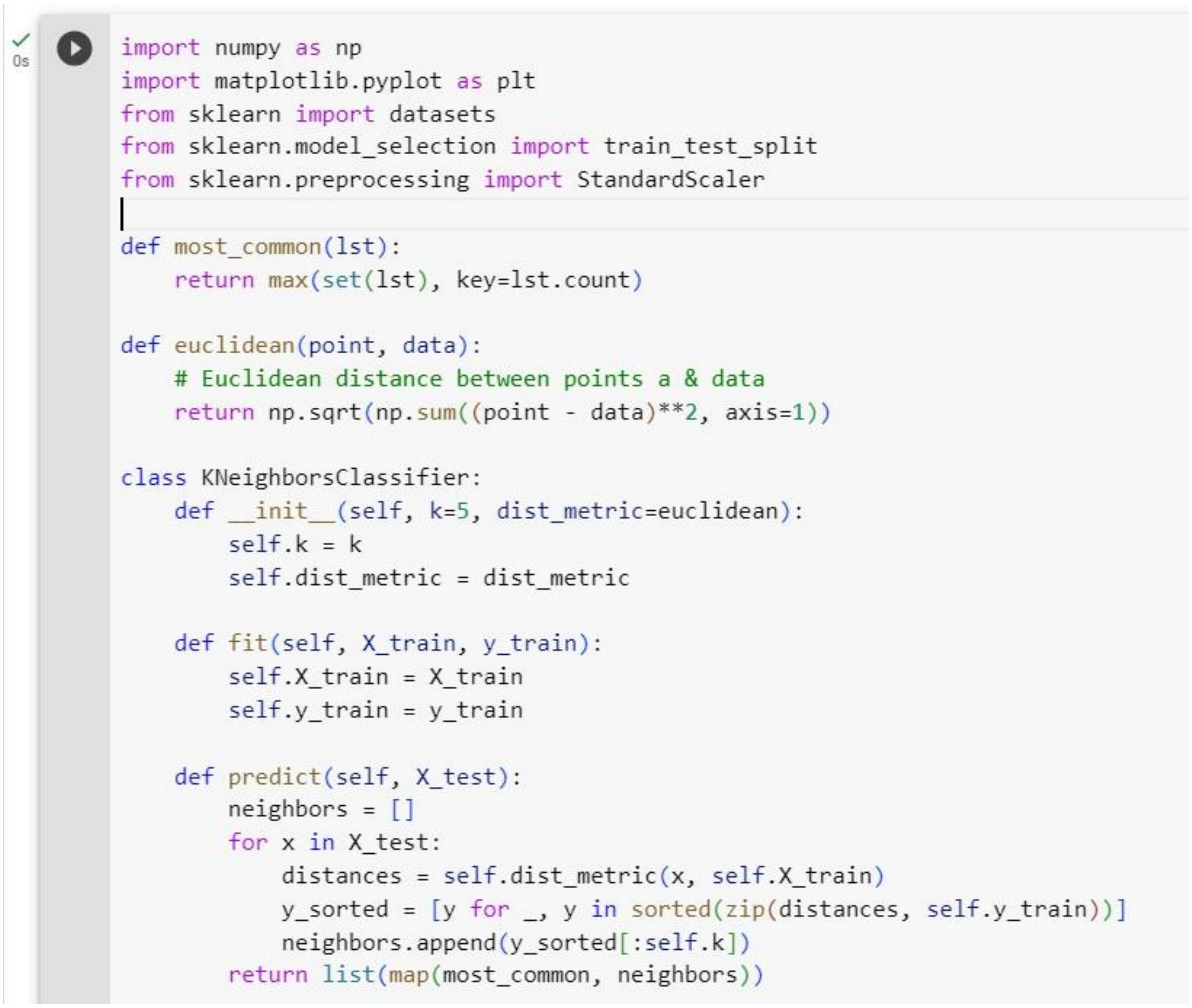


Program 8: KNN ALGORITHM

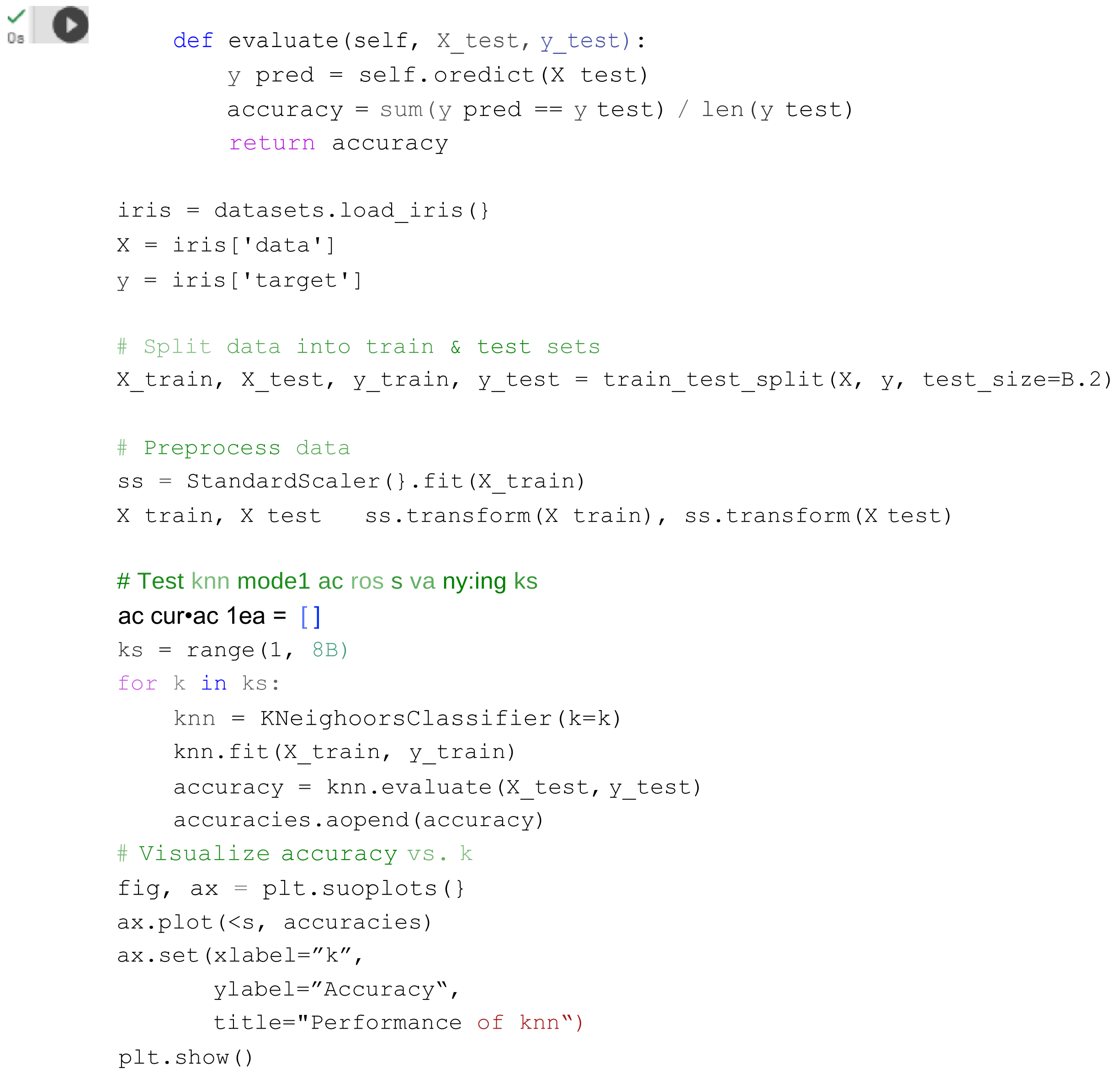
Dataset used: Iris dataset

Algorithm:

- Select the number K of the neighbor
- Calculate the Euclidean distance of K number of neighbors
- Take the K nearest neighbors as per the calculated Euclidean distance.
- Among these k neighbors, count the number of the data points in each category.
- Assign the new data points to that category for which the number of the neighbor is maximum.



```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

def most_common(lst):
    return max(set(lst), key=lst.count)

def euclidean(point, data):
    # Euclidean distance between points a & data
    return np.sqrt(np.sum((point - data)**2, axis=1))

class KNeighborsClassifier:
    def __init__(self, k=5, dist_metric=euclidean):
        self.k = k
        self.dist_metric = dist_metric

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

    def predict(self, X_test):
        neighbors = []
        for x in X_test:
            distances = self.dist_metric(x, self.X_train)
            y_sorted = [y for _, y in sorted(zip(distances, self.y_train))]
            neighbors.append(y_sorted[:self.k])
        return list(map(most_common, neighbors))
```



```
    def evaluate(self, X_test, y_test):
        y pred = self.oredict(X test)
        accuracy = sum(y pred == y test) / len(y test)
        return accuracy

iris = datasets.load_iris()
X = iris['data']
y = iris['target']

# Split data into train & test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=B.2)

# Preprocess data
ss = StandardScaler().fit(X_train)
X train, X test   ss.transform(X train), ss.transform(X test)

# Test knn mode1 ac ros s va ny:ing ks
ac cur•ac 1ea = [ ]
ks = range(1, 8B)
for k in ks:
    knn = KNeighoorsClassifier(k=k)
    knn.fit(X_train, y_train)
    accuracy = knn.evaluate(X_test, y_test)
    accuracies.aopend(accuracy)
# Visualize accuracy vs. k
fig, ax = plt.suoplots()
ax.plot(<s, accuracies)
ax.set(xlabel=”k”,
       ylabel=”Accuracy�much",
       title="Performance of knn“)
plt.show()
```

K-nearest Neighbor Algorithm

* For each given training example $(x, f(x))$, add the example to the list training examples to the list training examples classification algorithm.

* Given a query instance x_q to be classified,
Let $x_1 \cdots x_k$ denote the K instances from training examples that are nearest to x_q.

* Return

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{K}.$$

~~Sepal~~

Output.

sepal-length sepal-width petal-length petal-width

```
[[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]
 [5.0  3.6 1.4 0.2]
  .     .    .   .   .
  .     .    .   .   .
 [6.2 3.4  5.4  2.3]
 [5.9.3   5.1 1.8]]
```

Class : 0-Iris-sentosa, 1-Iris Versicolor, 2-Iris-virginica

```
[000 ---- 00111 ---11222 --- 22]
```

Confusion Matrix

```
[[20  0  0]
 [0  10  0]
 [0  1  14]]
```

Accuracy Metrics

```
            Precision  Recall  f1-score  support

   0         1.00      1.00     1.00      20
   1         0.91      1.00     0.95      10
   2         1.00      0.93     0.97      15

avg/total    0.98      0.98     0.98      45
```

O/p seen
7/6/23

Program 9: Apply EM algorithm to cluster a set of data stored in a .CSV file. Compare the results of k-Means algorithm and EM algorithm.

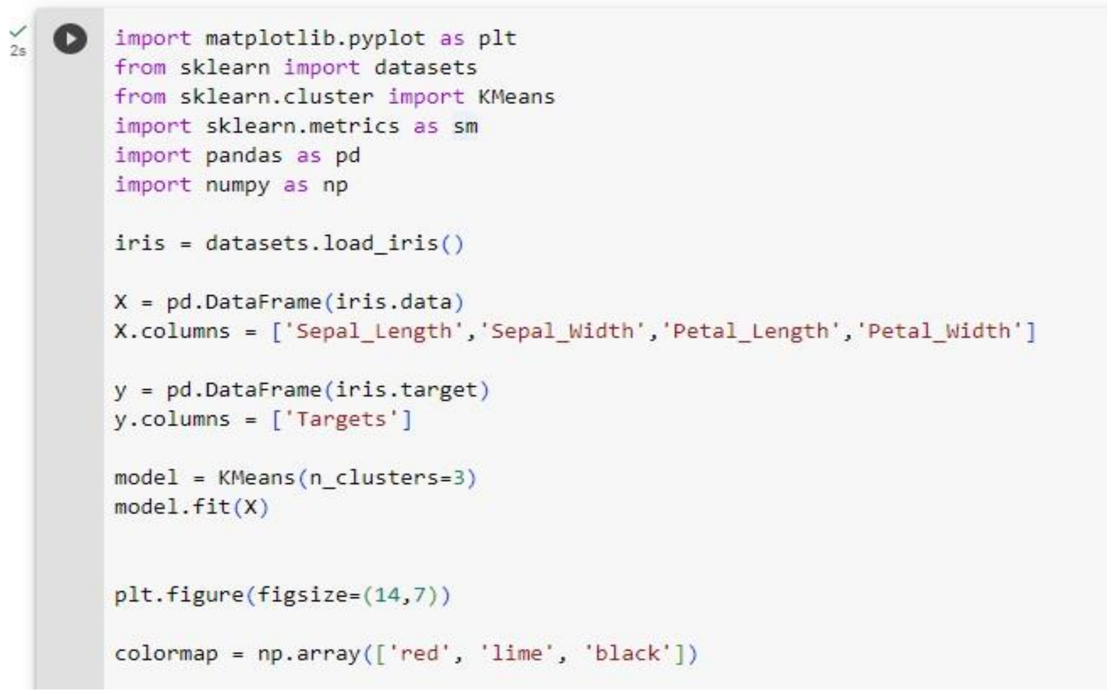
Algorithm for k means clustering:

- Initialize k means with random values
- For a given number of iterations:
- Iterate through items:
- Find the mean closest to the item by calculating the euclidean distance of the item with each of the means
- Assign item to mean
- Update mean by shifting it to the average of the items in that clusters

Algorithm for EM algorithm:

- The very first step is to initialize the parameter values. Further, the system is provided with incomplete observed data with the assumption that data is obtained from a specific model.
  - This step is known as Expectation or E-Step, which is used to estimate or guess the values of the missing or incomplete data using the observed data. Further, E-step primarily updates the variables.
  - This step is known as Maximization or M-step, where we use complete data obtained from the 2nd step to update the parameter values. Further, M-step primarily updates the hypothesis.
  - The last step is to check if the values of latent variables are converging or not.

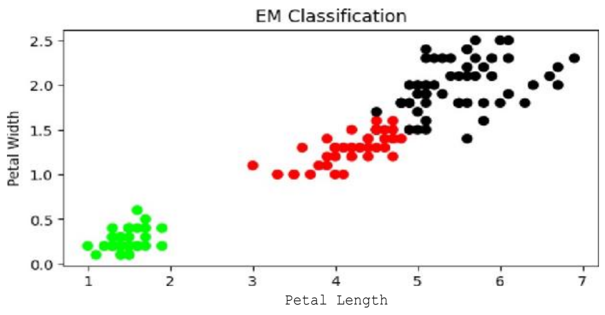
Dataset: Iris dataset

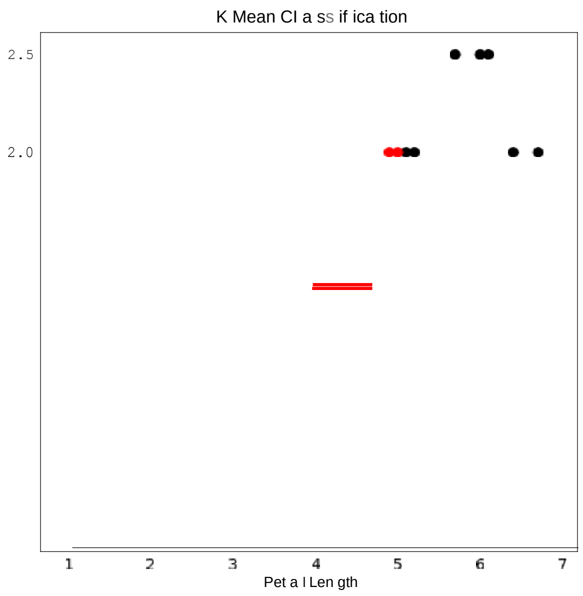


```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import sklearn.metrics as sm
import pandas as pd
import numpy as np

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']

model = KMeans(n_clusters=3)
model.fit(X)


plt.figure(figsize=(14,7))

colormap = np.array(['red', 'lime', 'black'])
```

```
S Plot the Original Classifications
pit.subplot(1, 2, 1)
pit.scatter(X.Petal_Length, X.Petal_Width, c=colonmap[y.Targets], s=40)
pit.title('Real Classification')
pit.xlabel('Metal Length')
pit.ylabel('Metal Eidth')


S Plot the Models Classifications
pit.subplot(l, 2, 2)
pit.scatter(X.Petal_Length, X.Peta1_Width, c=colormap[model.labels_], s=40)
pit.title('K !lean Classification')
pit.xlabel('Metal Length')
pit.ylabel('Metal Lidth')
print('The accuracy score of K-Mean: ',sm.accuracy_score(y, model.labels_})
print('The Confusion matrixof K-Mean: ',sm.confusion_matrix(y, model.labels_))


from sklearn import preprocessing
sca1er = preproces sing. SJzanda rds c a1er ( )
sca1er.III (X)
xsa = sca1er . transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)
Sxs.sample(5)

from sklearn.mixture impont GaussianMixture
gmm = GaussianMixture(n_components=3)
gmm.fit(xs)

y_gmm      gmm.predict(xs)
#y_cluster  mm



pit.subplot(2, 2, 3)
pit.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y_gmm], s=40)
pit.title('EM Classification')
pit.xlabel('Petal Length')
pit.ylabel('Petal L%dth'}

print('The accuracy score of E': ',sm.accuracy_score(y, y  mm))
print('The Confusion matrix of EM: ',sm.confusion_matrix(y, y_gmm))
```

```
The accuracy score of K-jean:  0.24
The Confusion matrixof K-Mean:  [[ 0 50  0]
 [48  0  2]
 [14  0 36]]
The accuracy score of EM:  0.3333333333333333
The Confusion matrix of EM:  [[ 0 50  0]
 [4S  0  5]
 [ 0  0 50]]
```


EM Classification
Petal Width
Petal Length
2.5
2.0
1.5
1.0
0.5
0.0
1
2
3
4
5
6
7


K Mean Cl a ss if ica tion
2.5
2.0
1
2
3
4
5
6
7
Pet a l Len gth

EM- Algorithm

* Expectation Step (E step) : It involves the estimation of all missing values in dataset so that after campeleting this step, there should not be any missing value.

* Maximize step (M-step): This step involves the Use of estimated data in E-step and updating the parameter.

* Repeat E step and M step until the Convergence of value occur.

① Initialize Parameter Values. Further, the system is provided with incomplete observerd data with assumption that data is obtained from specific model.

② E-step, which is used to estimate or guess the value of the missing data Using the observerd data.

③ Maximization step, where we use the complete data obtained from 2nd step to Update Parameter values.

* The last step is to check if value of variables are covering or not.

* If yes, stop Process else repeat Until convergence occurs.

Euclidian distance Formula:

$$\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$$

x_1 = x co-ordinate of point 1

y_1 = y co-ordinate of point 1

x_2 = x co-ordinate of pt 2

y_2 = y co-ordinate of pt 2

[illegible]

Program 10: Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select the appropriate data set for your experiment and draw graphs.

Algorithm:

1. F is approximated near Xq using a linear function:

$$\hat{f}(x) = w_0 + \sum_{u=1}^{k} w_u K_u(d(x_u, x))$$

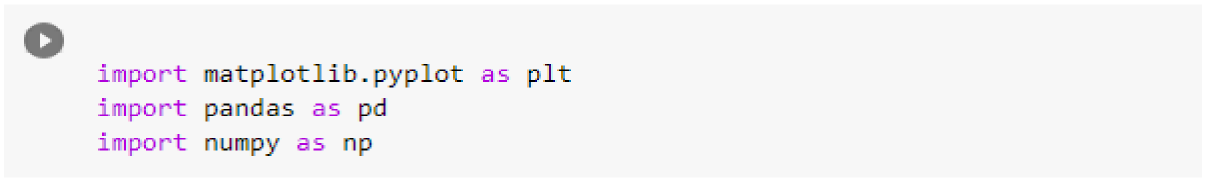
2. **Minimize the squared error:**

$$E_3(x_q) \equiv \frac{1}{2} \sum_{x \in \; k \; nearest \; nbrs \; of \; x_q} (f(x) - \hat{f}(x))^2 \; K(d(x_q, x))$$

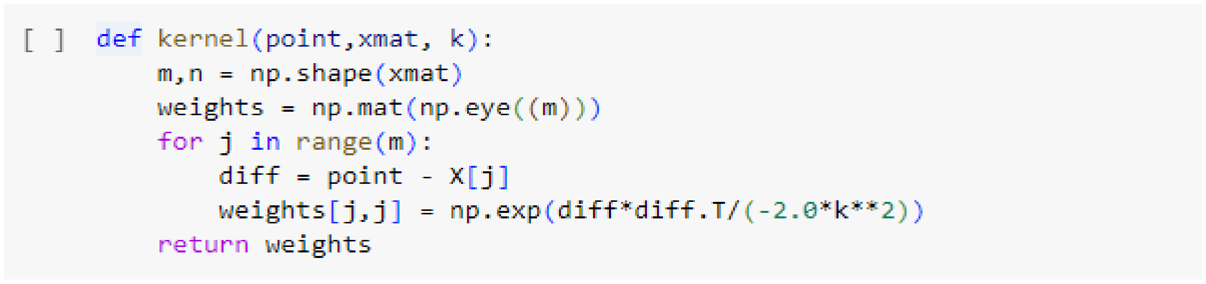
$$\Delta w_j = \eta \sum_{x \in \; k \; nearest \; nbrs \; of \; x_q} K(d(x_q, x)) \; (f(x) - \hat{f}(x)) \; a_j(x)$$

3. It is weighted because the contribution of each training example is weighted by its distance from the query point.

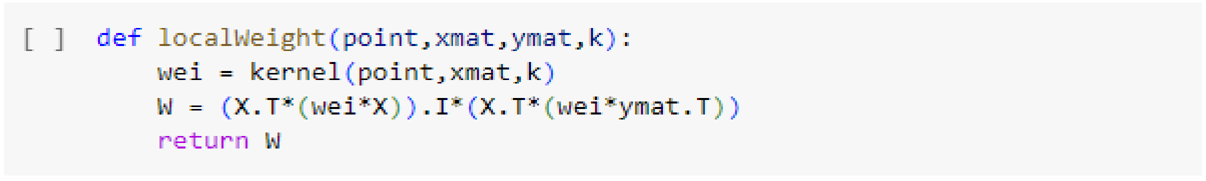
Dataset: tip.csv



```
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)))
    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]*localHeight(xmat[i],xmat,ymat,k)
         return ypred
```

```
[ ]  def gr aphP lot (X, ypred) :
         sortindex = X[:,1].argsort(0)
         xsort = X[sontindex] :,0]
         fig = pit.figure()
         ax = fig.add_subplot(1,1,t)
         ax.scatter(bill,tip, colon='green' )
         ax.plot(xsort[:,1],ypred[sortindex],color='red', linexidth=5)
         plt.xlabel('7otal bill')
         plt.ylabel('7ip ')
         plt.show();
```

```
      data  = pd . read_csv ( ' /' c onte nt/tips .c sv ' )
     bill = np.array(data.total_bill)
     tip = np.array(data.tip)

     nb111 = np . mat (bill)
     nJz1p = np . nat (t1p )
     n= np.shape(nb111)[1]
     one = np . mat (np . ones (n) )
     X = np. hs tack ( (one . T,nb111. T) )

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

Lab-7 Locally weighted Regression Algorithm

1) Read the given data sample to X and the curve (linear or non linear) to Y.

2) Set the value of Smoothening Parameter of Free Parameter say τ.

3) Set the bias / Point of Interest set X_0 which is subset of X.

4) Determine the weight matrix using:

$$w(w, w_0) = e^{\frac{-(x - x_0)^2}{2\tau^2}}$$

5) Determine the value of model term parameter β using:

$$\hat{\beta}(x_0) = (X^T W X)^{-1} X^T W Y$$

6) Prediction = $x_0 * \beta$;

O/p Seen 7/6/23