B. M.S. COLLEGE OF ENGINEERING

(Autonomous College under VTU)

Bull Temple Road, Basavangudi, Bangalore - 560019

MACHINE LEARNING LAB

(20CS6PCMAL)

Report

Submitted by

S Sanjith

(1BM20CS135)VI C

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

Lab In-charge

Dr Asha GR

Assistant Professor

2022-2023

LIST OF PROGRAMS

Date: 01.04.2023

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

Program

(i)First data set

```
In [16]: import pandas as pd
         from sklearn.datasets import load_iris
         iris = load_iris()

In [3]: iris

Out[3]: {'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 [4]: type(iris)

Out[4]: sklearn.utils._bunch.Bunch

In [5]: iris.keys()

Out[5]: dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module'])

In [5]: iris.keys()

Out[5]: dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module'])

In [6]: print(iris['target_names'])

        ['setosa' 'versicolor' 'virginica']

In [7]: n_samples,n_features=iris.data.shape

In [8]: n_samples,n_features

Out[8]: (150, 4)

In [9]: 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 [10]: iris.data[[12,26,90,114]]

Out[10]: array([[4.8, 3. , 1.4, 0.1],
                [5. , 3.4, 1.6, 0.4],
                [5.5, 2.6, 4.4, 1.2],
                [5.8, 2.8, 5.1, 2.4]])
```

In [11]:
```
print(iris.data.shape)
print(iris.target.shape)
print(iris.target)
```

```
(150, 4)
(150,)
[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 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]
```

In [12]:
```
import numpy as np
np.bincount(iris.target)
```

Out[12]:
```
array([50, 50, 50], dtype=int64)
```

In [13]:
```
import matplotlib.pyplot as plt

data = load_iris()


list(data.target_names)
```

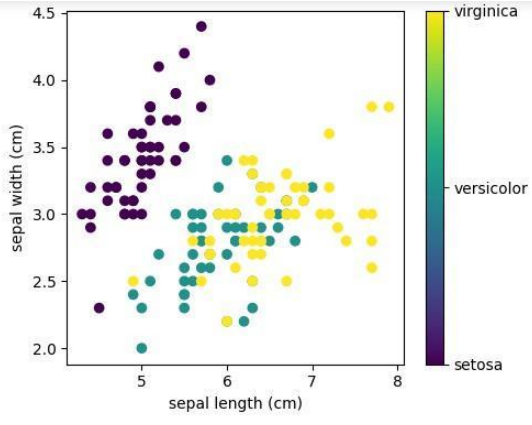
Out[13]:
```
['setosa', 'versicolor', 'virginica']
```

In [26]:
```
iris=pd.DataFrame(data=np.c_[iris['data'],iris['target']],columns=iris['feature_names']+['target'])
iris.head(5)
```

Out[26]:

	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

In [14]:
```
x_index = 0
y_index = 1

# this formatter will label the colorbar with the correct target names
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()
```



b. Practice Exercise - Dataset Exploration – wine data set

In [18]:
```
#iris=pd.DataFrame(data=np)
```

In [19]:
```
from sklearn.datasets import load_wine
wine = load_wine()

list(wine.target_names)
```

Out[19]:
```
['class_0', 'class_1', 'class_2']
```

In [20]:
```
print(wine.data.shape)
print(wine.target.shape)
print(wine.target)
```

```
(178, 13)
(178,)
[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 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 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]
```

In [21]:
```
np.bincount(wine.target)
```

Out[21]:
```
array([59, 71, 48], dtype=int64)
```

In [23]:
```
wine.DESCR
```

Out[23]:
```
'.. _wine_dataset:\n\nWine recognition dataset\n------------------------\n\n**Data Set Characteristics:**\n\n    :Number of
Instances: 178\n    :Number of Attributes: 13 numeric, predictive attributes and the class\n    :Attribute Information:\n \t
\t- Alcohol\n \t\t- Malic acid\n \t\t- Ash\n\t\t- Alcalinity of ash  \n \t\t- Magnesium\n\t\t- Total phenols\n \t\t- Flavano
ids\n \t\t- Nonflavanoid phenols\n \t\t- Proanthocyanins\n\t\t- Color intensity\n \t\t- Hue\n \t\t- OD280/OD315 of diluted w
ines\n \t\t- Proline\n\n    - class:\n            - class_0\n            - class_1\n            - class_2\n\t\t\n    :Summar
y Statistics:\n    \n    ============================= ==== ===== ======= =====\n                                Min   Ma
x   Mean     SD\n    ============================= ==== ===== ======= =====\n    Alcohol:                          11.0  14.8
13.0   0.8\n    Malic Acid:                    0.74  5.80    2.34  1.12\n    Ash:                          1.36  3.23    2.36
0.27\n    Alcalinity of Ash:            10.6  30.0    19.5   3.3\n    Magnesium:                    70.0 162.0    99.7  14.3
\n    Total Phenols:                0.98  3.88    2.29  0.63\n    Flavanoids:                   0.34  5.08    2.03  1.00\n
Nonflavanoid Phenols:         0.13  0.66    0.36  0.12\n    Proanthocyanins:              0.41  3.58    1.59  0.57\n    Colo
ur Intensity:                 1.3  13.0     5.1   2.3\n    Hue:                          0.48  1.71    0.96  0.23\n    OD280/OD
315 of diluted wines: 1.27  4.00    2.61  0.71\n    Proline:                       278  1680     746   315\n    ============
================= ==== ===== ======= =====\n\n    :Missing Attribute Values: None\n    :Class Distribution: class_0 (59), cl
ass_1 (71), class_2 (48)\n    :Creator: R.A. Fisher\n    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)\n    :Date:
July, 1988\n\nThis is a copy of UCI ML Wine recognition datasets.\nhttps://archive.ics.uci.edu/ml/machine-learning-database
s/wine/wine.data\n\nThe data is the results of a chemical analysis of wines grown in the same\nregion in Italy by three diff
erent cultivators. There are thirteen different\nmeasurements taken for different constituents found in the three types of\n
wine.\n\nOriginal Owners: \n\nForina, M. et al, PARVUS - \nAn Extendible Package for Data Exploration, Classification and Co
rrelation. \nInstitute of Pharmaceutical and Food Analysis and Technologies,\nVia Brigata Salerno, 16147 Genoa, Italy.\n\nCi
tation:\n\nLichman, M. (2013). UCI Machine Learning Repository\n[https://archive.ics.uci.edu/ml]. Irvine, CA: University of
California,\nSchool of Information and Computer Science. \n\n.. topic:: References\n\n  (1) S. Aeberhard, D. Coomans and O.
de Vel, \n  Comparison of Classifiers in High Dimensional Settings, \n  Tech. Rep. no. 92-02, (1992), Dept. of Computer Scie
nce and Dept. of  \n  Mathematics and Statistics, James Cook University of North Queensland. \n  (Also submitted to Technome
trics). \n\n  The data was used with many others for comparing various \n  classifiers. The classes are separable, though on
ly RDA \n  has achieved 100% correct classification. \n  (RDA : 100%, QDA 99.4%, LDA 98.9%, 1NN 96.1% (z-transformed data))
\n  (All results using the leave-one-out technique) \n\n  (2) S. Aeberhard, D. Coomans and O. de Vel, \n  "THE CLASSIFICATIO
N PERFORMANCE OF RDA" \n  Tech. Rep. no. 92-01, (1992), Dept. of Computer Science and Dept. of \n  Mathematics and Statistic
s, James Cook University of North Queensland. \n  (Also submitted to Journal of Chemometrics).\n'
```

Date: 08.04.2023

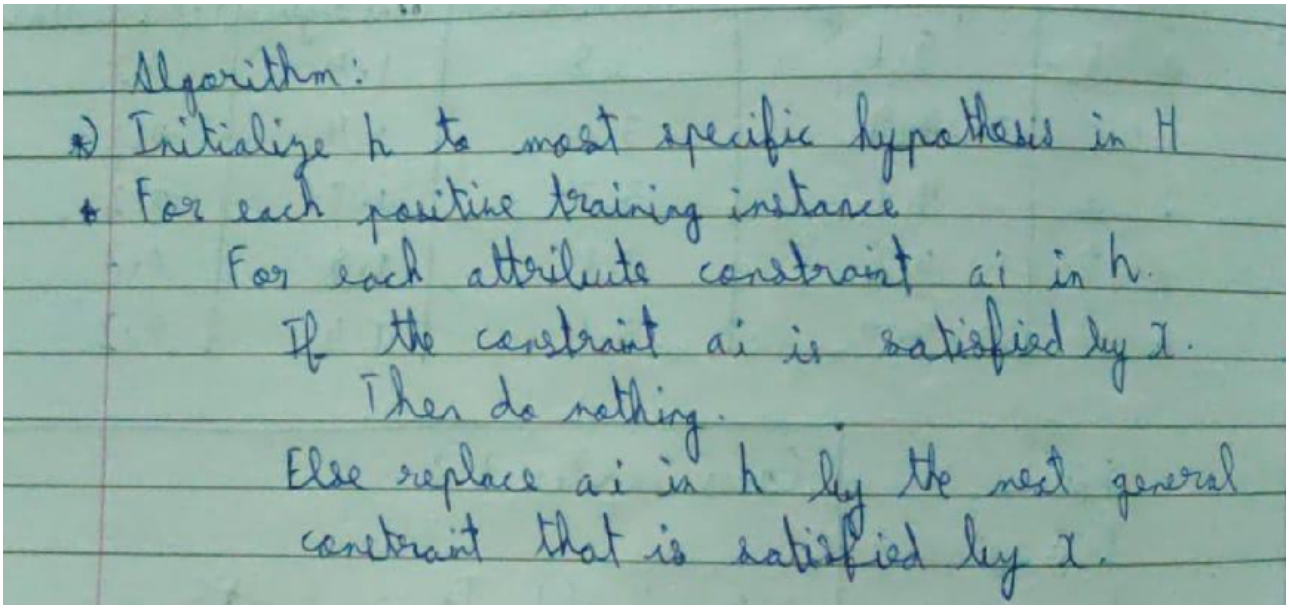
Program 2 – Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file

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:


Algorithm:
* Initialize h to most specific hypothesis in H
* For each positive training instance
  For each attribute constraint ai in h.
    If the constraint ai is satisfied by x.
      Then do nothing.
    Else replace ai in h by the next general constraint that is satisfied by x.

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, ?, ?>**

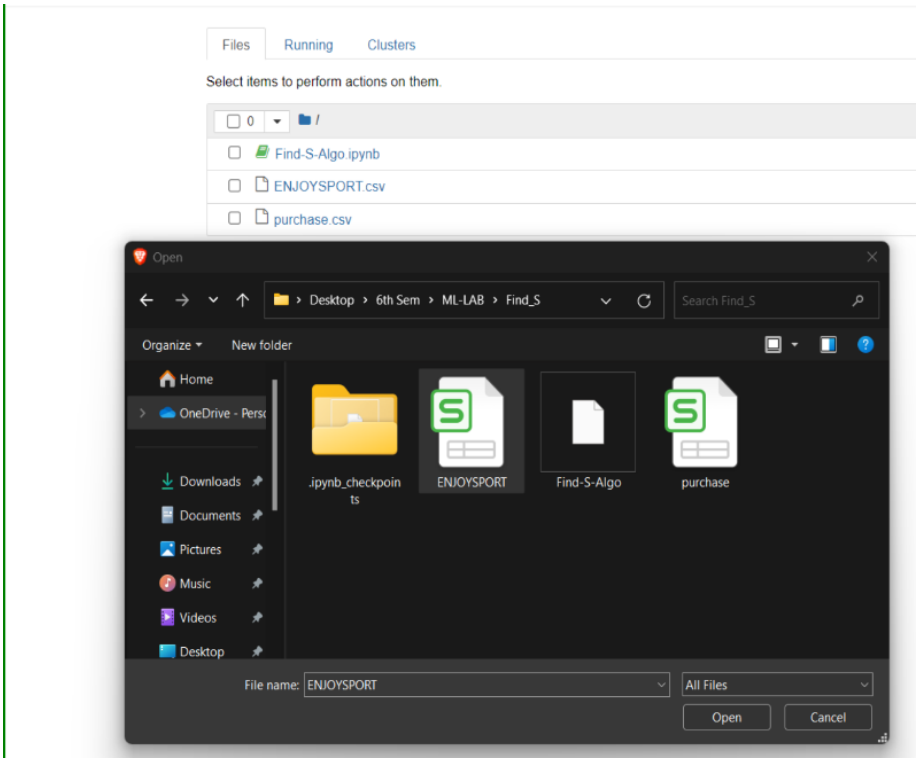
Uploading the csv file.

WPS Office | ENJOYSPORT.csv

Menu | Home | Insert | Page Layout | Formulas | Data | Review | View | Tools | Smart Toolbox

A1 | Sky

	A	B	C	D	E	F	G
1	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
2	Sunny	Warm	Normal	Strong	Warm	Same	1
3	Sunny	Warm	High	Strong	Warm	Same	1
4	Rainy	Cold	High	Strong	Warm	Change	0
5	Sunny	Warm	High	Strong	Cool	Change	1
6							
7							



Program:

Screenshot of the program executed

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

```
[ ] data=pd.read_csv('enjoysport.csv')
    data.head(5)
```

	sky	airtemp	humidity	wind	water	forcast	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

```
[ ] data1=np.array(data)[:,:6]
    print(data1)
```

```
[['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]
    target
```

```
array(['yes', 'yes', 'no', 'yes'], dtype=object)
```

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

  for i,var in enumerate(data):
    if target[i]=="yes":
      for x in range(len(hypothesis)):
        if var[x]!=hypothesis[x]:
          hypothesis[x]='?'
        else:
          pass

  return hypothesis

print("The Hypothesis is",find_s(data1,target))
```

```
The Hypothesis is ['sunny' 'warm' '?' 'strong' '?' '?']
```

Output:

Screenshot of the executed output

```
The Hypothesis is ['sunny' 'warm' '?' 'strong' '?' '?']
```

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

```
data=pd.read_csv('/content/seconddataset .csv')
data.head(5)
```

	example	citations	size	inLibrary	price	editions	buy
0	1	some	small	no	affordable	many	no
1	2	many	big	no	expensive	one	yes
2	3	some	big	always	expensive	few	no
3	4	many	medium	no	expensive	many	yes
4	5	many	small	no	affordable	many	yes

```
[ ] data1=np.array(data)[:,1:6]
    print(data1)
```

```
[['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']]
```

```
[ ] target=np.array(data)[:,-1]
    target
```

```
array(['no', 'yes', 'no', 'yes', 'yes'], dtype=object)
```

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

  for i,var in enumerate(data):
    if target[i]=="yes":
      for x in range(len(hypothesis)):
        if var[x]!=hypothesis[x]:
          hypothesis[x]='?'
        else:
          pass

  return hypothesis

print("The Hypothesis is",find_s(data1,target))
```

```
[ ]   for i,var in enumerate(data):
        if target[i]=="yes":
          for x in range(len(hypothesis)):
            if var[x]!=hypothesis[x]:
              hypothesis[x]='?'
            else:
              pass

      return hypothesis

    print("The Hypothesis is",find_s(data1,target))
```

```
The Hypothesis is ['many' '?' 'no' '?' '?']
```

Output:

Screenshot of the executed output

```
The Hypothesis is ['many' '?' 'no' '?' '?']
```

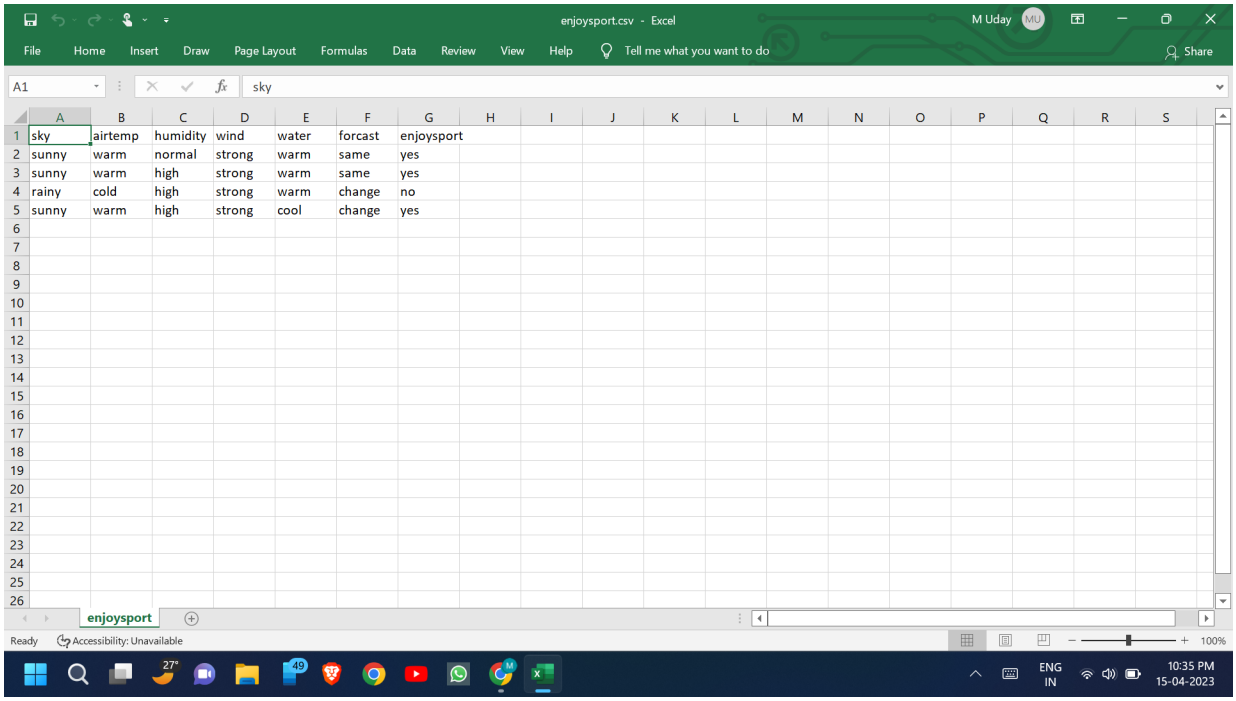
Program 3:

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

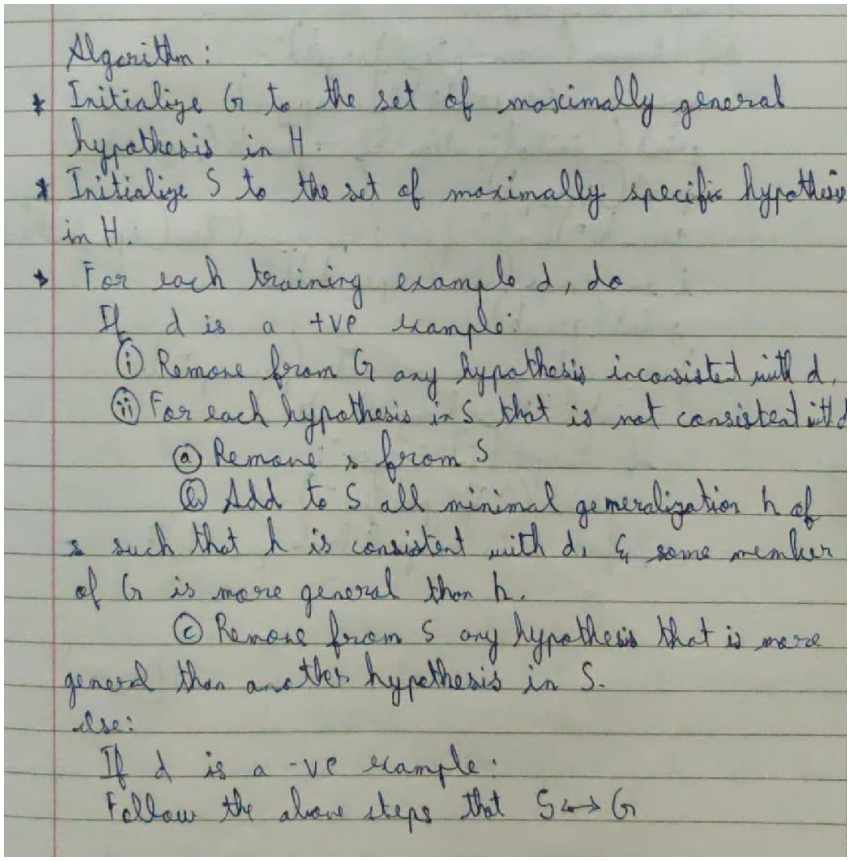
Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

Uploading the csv file:



Algorithm:

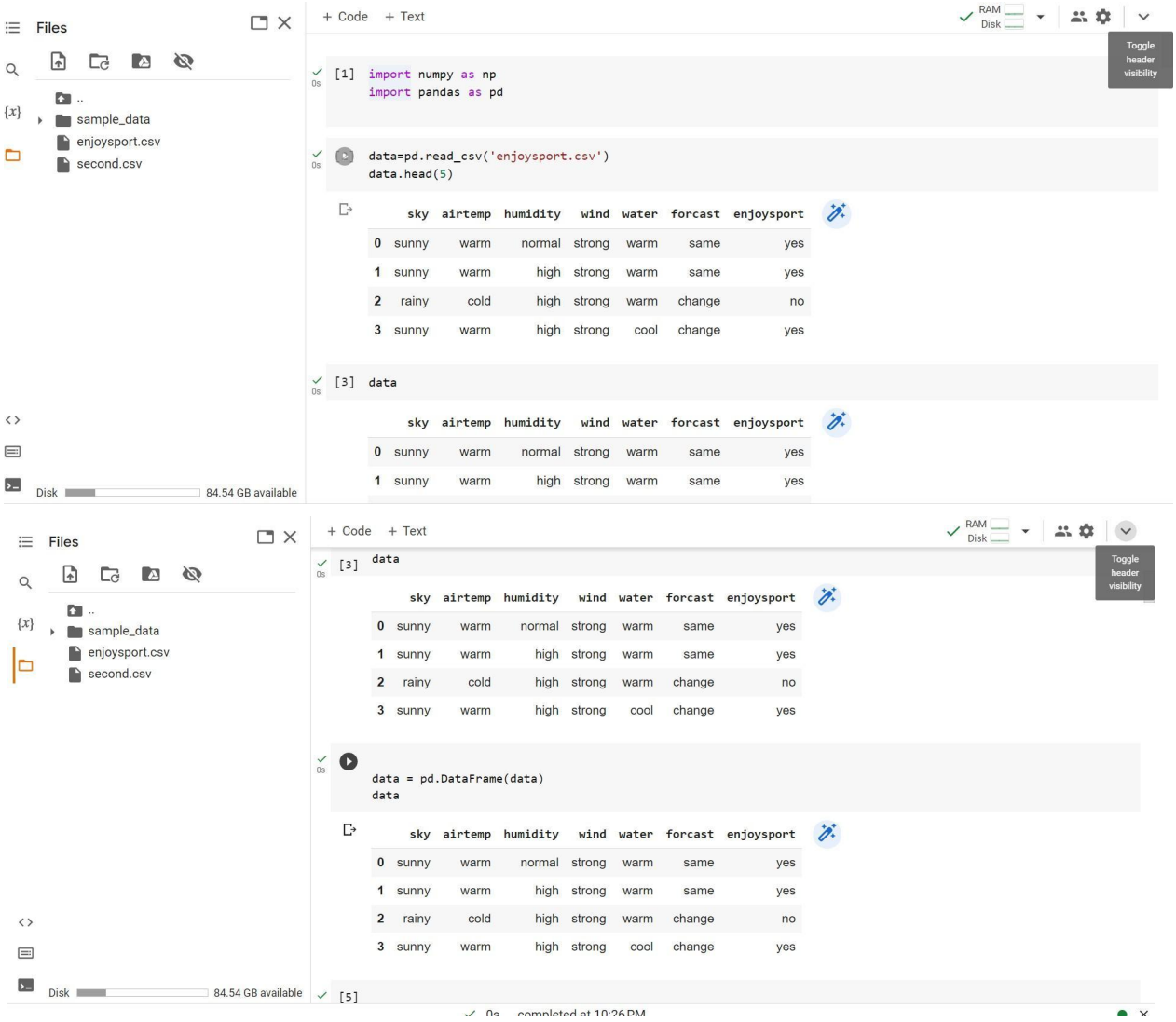
Handwritten observation:


Algorithm:

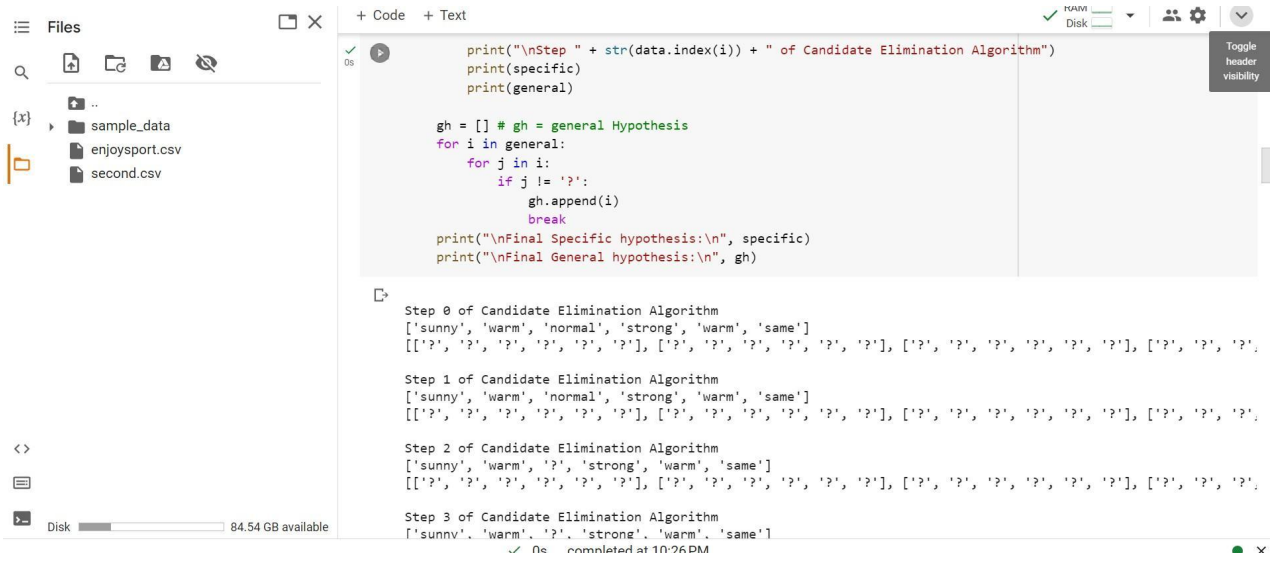
* Initialize G to the set of maximally general hypothesis in H.
* Initialize S to the set of maximally specific hypothesis in H.
* For each training example d, do
  If d is a +ve example:
    (i) Remove from G any hypothesis inconsistent with d.
    (ii) For each hypothesis in S that is not consistent with d
      (a) Remove s from S
      (b) Add to S all minimal generalization h of s such that h is consistent with d, & some member of G is more general than h.
      (c) Remove from S any hypothesis that is more general than another hypothesis in S.
  else:
  If d is a -ve example:
  Follow the above steps that S ↔ G

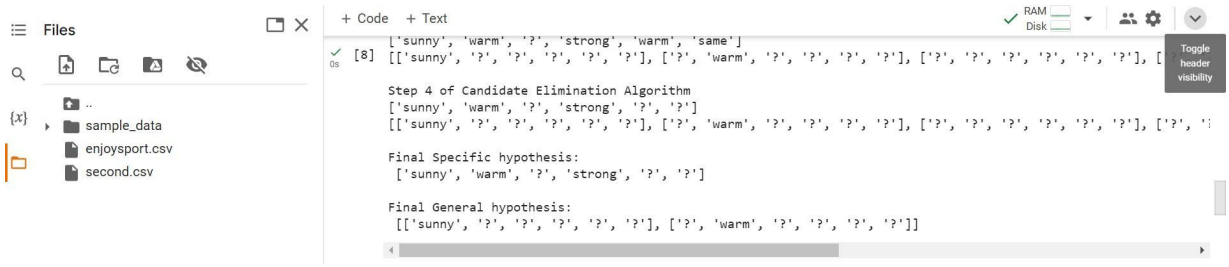
Program:

First data set




Files
sample_data
enjoysport.csv
second.csv
Disk 84.54 GB available
+ Code + Text
RAM Disk
Toggle header visibility
[5]
attribute = np.array(data.iloc[:,0:-1])
[6] attribute
array([['sunny', 'warm', 'normal', 'strong', 'warm', 'same'],
['sunny', 'warm', 'high', 'strong', 'warm', 'same'],
['rainy', 'cold', 'high', 'strong', 'warm', 'change'],
['sunny', 'warm', 'high', 'strong', 'cool', 'change']],
dtype=object)
[7] target = np.array(data.iloc[:,-1])
target
array(['yes', 'yes', 'no', 'yes'], dtype=object)
+ Code + Text
[8] import csv
with open("enjoysport.csv") as f:
csv_file = csv.reader(f)
data = list(csv_file)
specific = data[1][:-1]
general = [['?' for i in range(len(specific))] for j in range(len(specific))]
0s completed at 10:26 PM


Files
sample_data
enjoysport.csv
second.csv
Disk 84.54 GB available
+ Code + Text
Toggle header visibility
print("\nStep " + str(data.index(i)) + " of Candidate Elimination Algorithm")
print(specific)
print(general)
gh = [] # gh = general Hypothesis
for i in general:
for j in i:
if j != '?':
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']
Step 2 of Candidate Elimination Algorithm
['sunny', 'warm', '?', 'strong', 'warm', 'same']
Step 3 of Candidate Elimination Algorithm

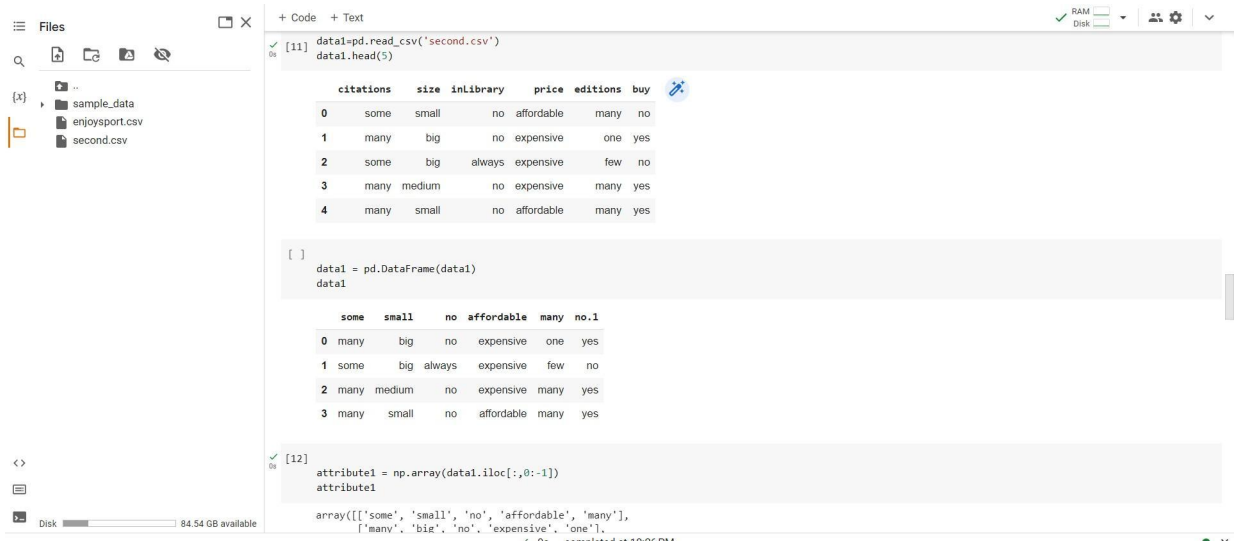

Files
sample_data
enjoysport.csv
second.csv
+ Code + Text
RAM Disk
Toggle header visibility
['sunny', 'warm', '?', 'strong', 'warm', 'same']
[8]
Step 4 of Candidate Elimination Algorithm
['sunny', 'warm', '?', 'strong', '?', '?']
Final Specific hypothesis:
['sunny', 'warm', '?', 'strong', '?', '?']
Final General hypothesis:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]

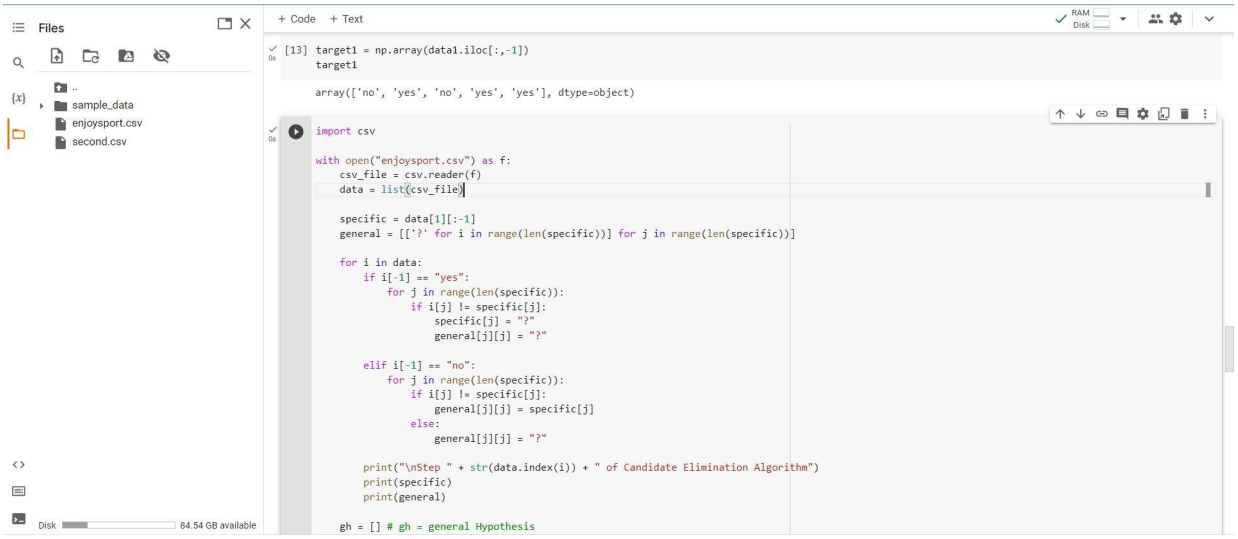
Output:

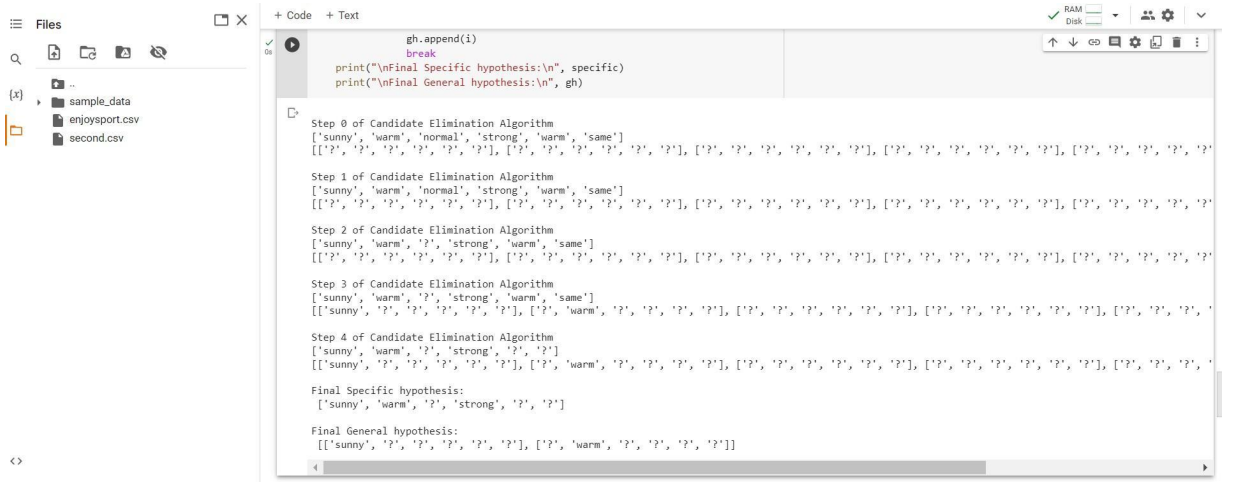
```
Step 0:
Specific Hypothesis:  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
General Hypothesis:  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
--------------------
Step 1 :
Specific Hypothesis:  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
General 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', '?', '?', '?', '?'], ['?', '?', '?', '?
--------------------
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


Files
sample_data
enjoysport.csv
second.csv
+ Code + Text
[11] data1=pd.read_csv('second.csv')
data1.head(5)
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
[ ]
data1 = pd.DataFrame(data1)
data1
some small no affordable many no.1
0 many big no expensive one yes
1 some big always expensive few no
2 many medium no expensive many yes
3 many small no affordable many yes
[12]
attribute1 = np.array(data1.iloc[:,0:-1])
attribute1
array([['some', 'small', 'no', 'affordable', 'many'],
['many', 'big', 'no', 'expensive', 'one'],
Disk 84.54 GB available


Files
sample_data
enjoysport.csv
second.csv
+ Code + Text
[13] target1 = np.array(data1.iloc[:,-1])
target1
array(['no', 'yes', 'no', 'yes', 'yes'], dtype=object)
import csv
with open("enjoysport.csv") as f:
csv_file = csv.reader(f)
data = list(csv_file)
specific = data[1][:-1]
general = [['?' for i in range(len(specific))] for j in range(len(specific))]
for i in data:
if i[-1] == "yes":
for j in range(len(specific)):
if i[j] != specific[j]:
specific[j] = "?"
general[j][j] = "?"
elif i[-1] == "no":
for j in range(len(specific)):
if i[j] != specific[j]:
general[j][j] = specific[j]
else:
general[j][j] = "?"
print("\nStep " + str(data.index(i)) + " of Candidate Elimination Algorithm")
print(specific)
print(general)
gh = [] # gh = general Hypothesis
Disk 84.54 GB available


Files
sample_data
enjoysport.csv
second.csv
+ Code + Text
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']
Step 2 of Candidate Elimination Algorithm
['sunny', 'warm', '?', 'strong', 'warm', 'same']
Step 3 of Candidate Elimination Algorithm
['sunny', 'warm', '?', 'strong', 'warm', 'same']
Step 4 of Candidate Elimination Algorithm
['sunny', 'warm', '?', 'strong', '?', '?']
Final Specific hypothesis:
['sunny', 'warm', '?', 'strong', '?', '?']
Final General hypothesis:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]

Output:

```
Step 0:
Specific Hypothesis:  ['some' 'small' 'no' 'affordable' 'many']
General Hypothesis:  [['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?
--------------------
Step 1 :
Specific Hypothesis:  ['some' 'small' 'no' 'affordable' 'many']
General Hypothesis:  [['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?
--------------------
Step 2 :
Specific Hypothesis:  ['?' '?' 'no' '?' '?']
General Hypothesis:  [['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?
--------------------
Step 3 :
Specific Hypothesis:  ['?' '?' 'no' '?' '?']
General Hypothesis:  [['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', 'no', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '
--------------------
Step 4 :
Specific Hypothesis:  ['?' '?' 'no' '?' '?']
General Hypothesis:  [['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', 'no', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '
--------------------
Step 5 :
Specific Hypothesis:  ['?' '?' 'no' '?' '?']
General Hypothesis:  [['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', 'no', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '
--------------------
Final S:
['?' '?' 'no' '?' '?']
Final G:
[['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', 'no', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?']]
```

Date:29-04-2023

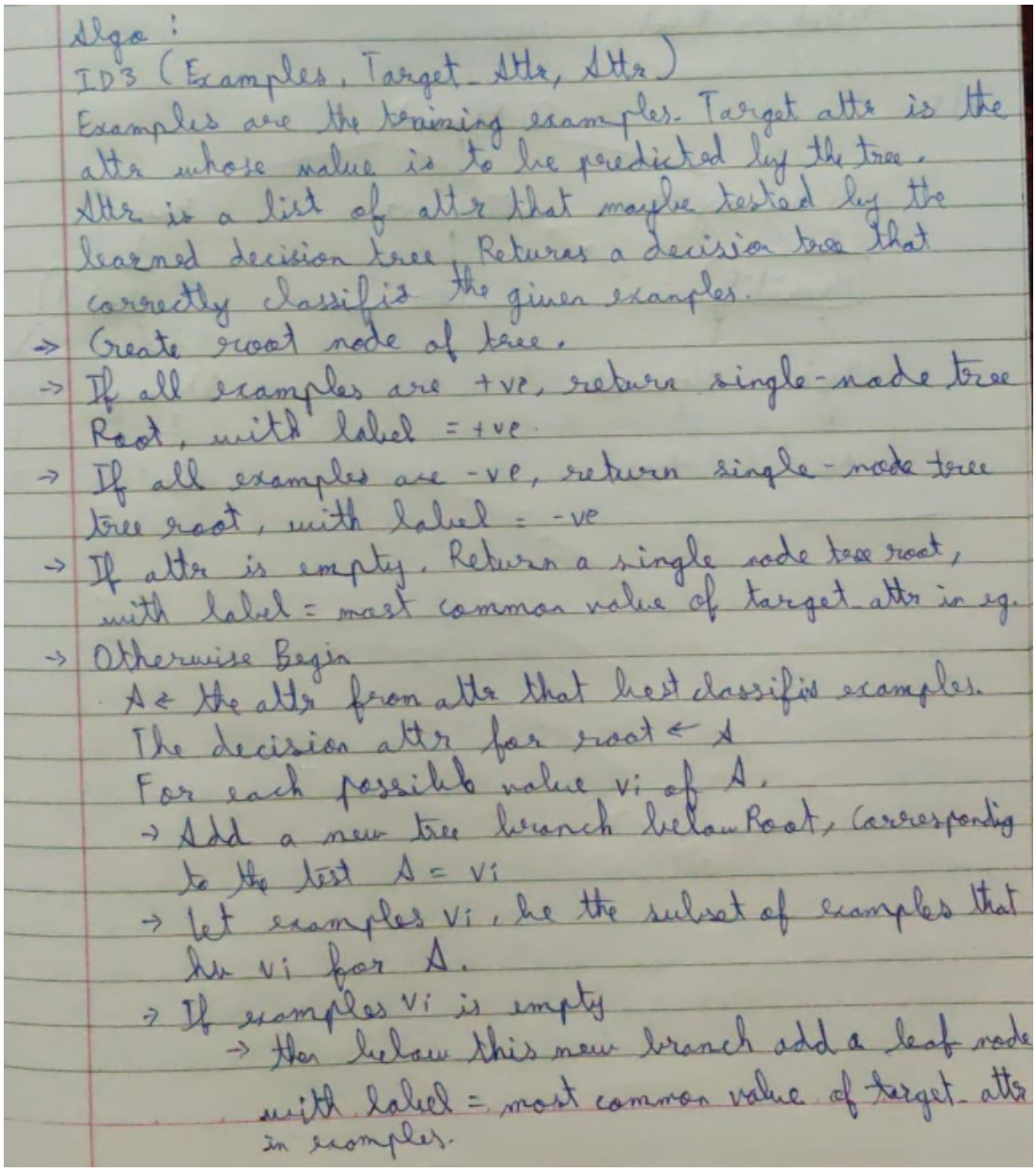
Program-4:

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: Playtennis

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

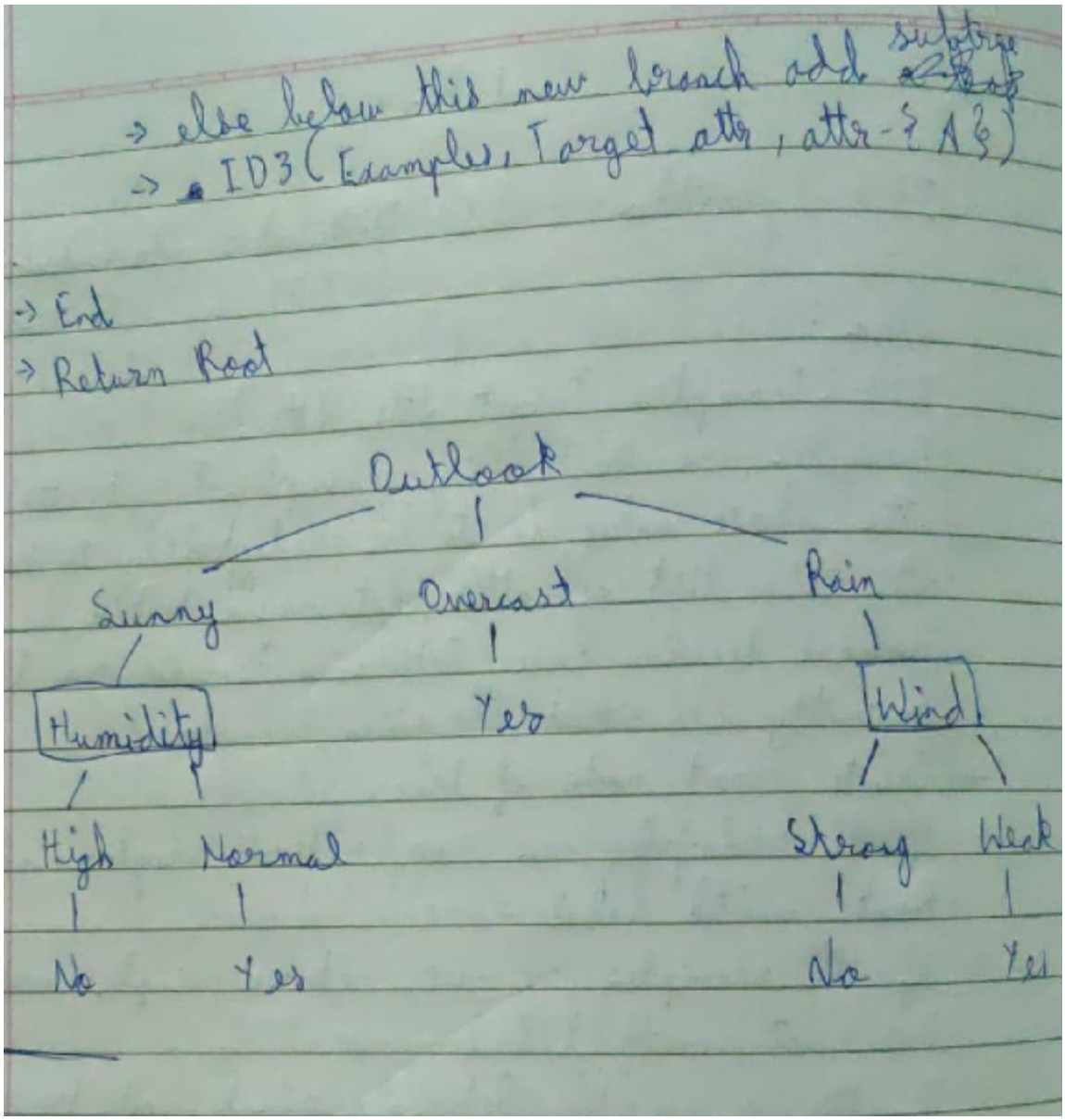
Observation:



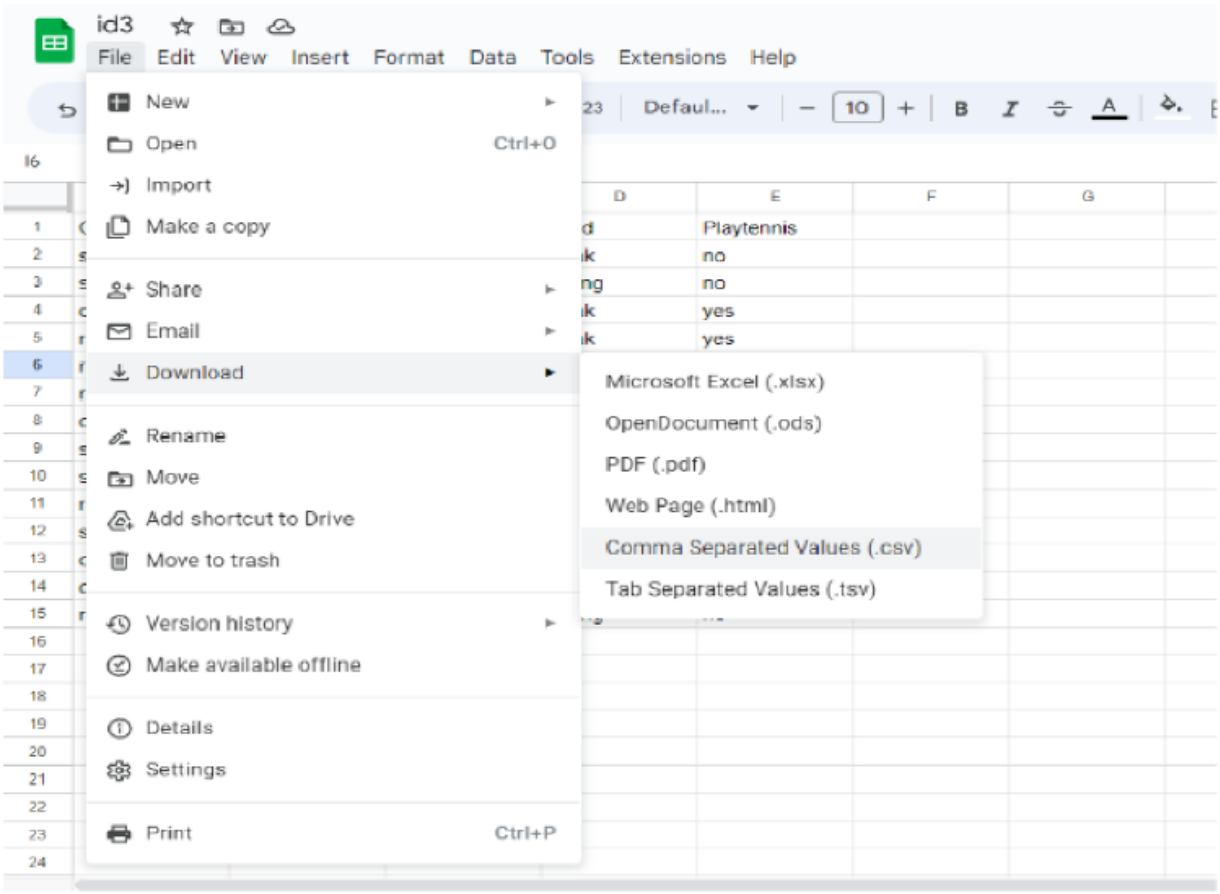
Algo :
ID3 (Examples, Target_Attr, Attr)
Examples are the training examples. Target attr is the attr whose value is to be predicted by the tree. Attr is a list of attr that maybe tested by the learned decision tree. Returns a decision tree that correctly classifies the given examples.
→ Create root node of tree.
→ If all examples are +ve, return single-node tree Root, with label = +ve
→ If all examples are -ve, return single-node tree tree root, with label = -ve
→ If attr is empty. Return a single node tree root, with label = most common value of target attr in eg.
→ Otherwise Begin
A ← the attr from attr that best classifies examples.
The decision attr 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 v_i, be the subset of examples that has v_i for A.
→ If examples v_i is empty
→ then below this new branch add a leaf node with label = most common value of target_attr in examples.

→ else below this new branch add subtree
→ ID3 (Examples, Target_attr, attr - {A})

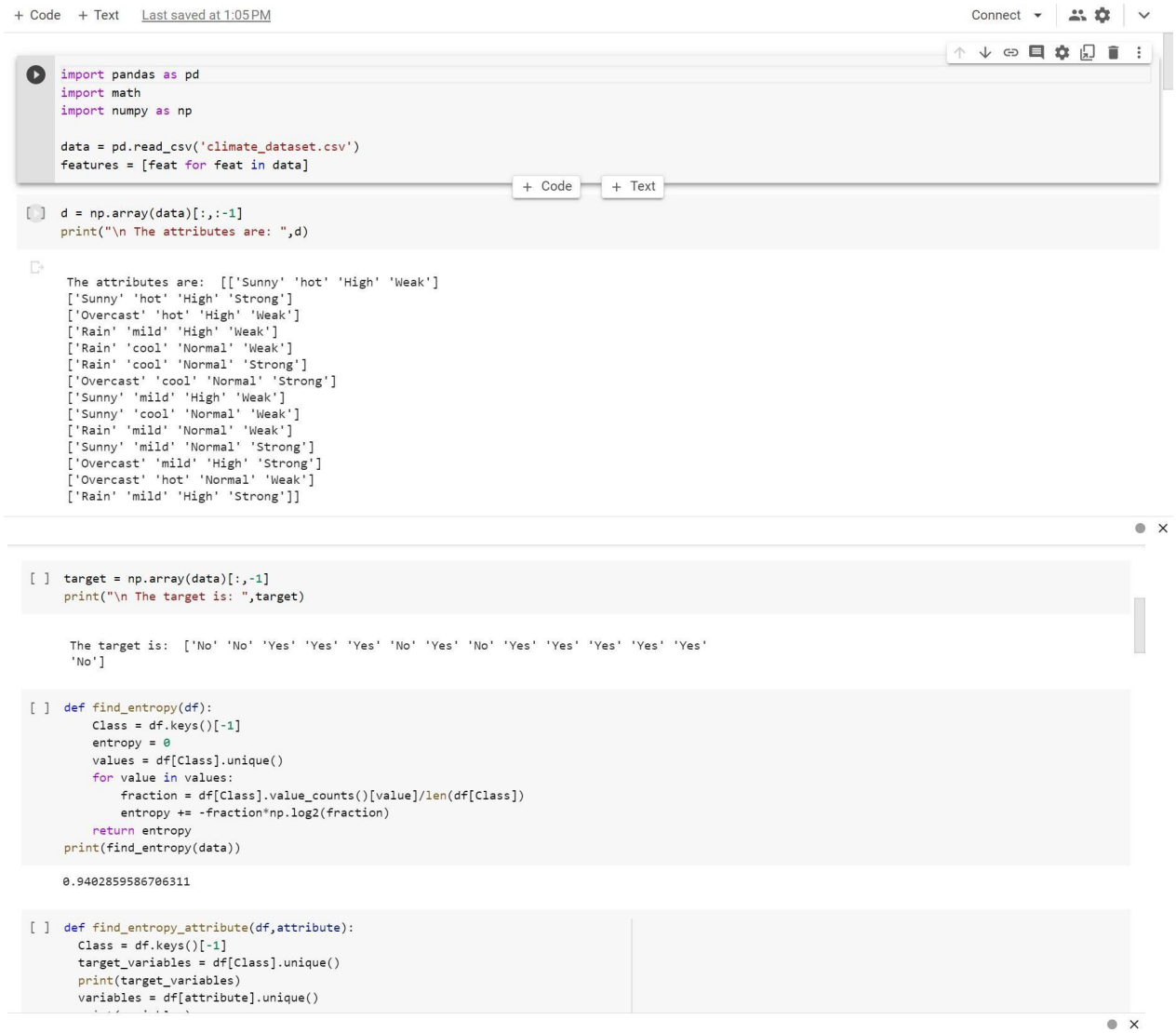
→ End
→ Return Root



CSV File:


id3
File
Edit
View
Insert
Format
Data
Tools
Extensions
Help
New
Open
Ctrl+O
Import
Make a copy
Share
Email
Download
Rename
Move
Add shortcut to Drive
Move to trash
Version history
Make available offline
Details
Settings
Print
Ctrl+P
Microsoft Excel (.xlsx)
OpenDocument (.ods)
PDF (.pdf)
Web Page (.html)
Comma Separated Values (.csv)
Tab Separated Values (.tsv)
Playtennis
no
no
yes
yes

Code:



```
+ Code + Text  Last saved at 1:05PM                                   Connect
```

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

data = pd.read_csv('climate_dataset.csv')
features = [feat for feat in data]
```

```
+ 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']
```

```
def find_entropy(df):
    Class = df.keys()[-1]
    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
print(find_entropy(data))
```

```
0.9402859586706311
```

```
def find_entropy_attribute(df,attribute):
  Class = df.keys()[-1]
  target_variables = df[Class].unique()
  print(target_variables)
  variables = df[attribute].unique()
```

```
    entropy2 = 0
    for variable in variables:
        entropy = 0
        for target_variable in target_variables:
            print(df[attribute][df[attribute]==variable][df[Class] ==target_variable])
            num = len(df[attribute][df[attribute]==variable][df[Class] ==target_variable])
            #print(num)
            #print(df[attribute][df[attribute]==variable])
            den = len(df[attribute][df[attribute]==variable])
            #print(num)
            fraction = num/(den+eps)
            entropy += -fraction*log(fraction+eps)
        fraction2 = den/len(df)
        entropy2 += -fraction2*entropy
    return abs(entropy2)
print(find_entropy_attribute(data,"outlook"))
```

```
['No' 'Yes']
['Sunny' 'Overcast' 'Rain']
0    Sunny
1    Sunny
7    Sunny
Name: outlook, dtype: object
8     Sunny
10    Sunny
Name: outlook, dtype: object
Series([], Name: outlook, dtype: object)
```

```
8     Sunny
10    Sunny
Name: outlook, dtype: object
Series([], Name: outlook, dtype: object)
2     Overcast
6     Overcast
11    Overcast
12    Overcast
Name: outlook, dtype: object
5     Rain
13    Rain
Name: outlook, dtype: object
3    Rain
4    Rain
9    Rain
Name: outlook, dtype: object
0.6935361388961914
```

+ 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]
    entropy = 0
    values = df[Class].unique()
```

```
def find_entropy_attribute(df,attribute):
  Class = df.keys()[-1]
  target_variables = df[Class].unique()
  variables = df[attribute].unique()

  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)


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))
         print(IG)
         print(df.keys()[:-1][np.argmax(IG)])
```

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


     def buildTree(df,tree=None):
         Class = df.keys()[-1]

         node = find_winner(df)

         attValue = np.unique(df[node])


         if tree is None:
             tree={}
             tree[node] = {}



         for value in attValue:

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

             if len(counts)==1:
                 tree[node][value] = clValue[0]
             else:
                 tree[node][value] = buildTree(subtable)
```

```
[0.24674981977443977]
outlook
[0.24674981977443977, 0.029222565658955535]
outlook
[0.24674981977443977, 0.029222565658955535, 0.15183550136234225]
outlook
[0.24674981977443977, 0.029222565658955535, 0.15183550136234225, 0.048127030408270155]
outlook
[5.551115123125783e-16]
outlook
[5.551115123125783e-16, 0.019973094021975557]
temperature
[5.551115123125783e-16, 0.019973094021975557, 0.019973094021975557]
temperature
[5.551115123125783e-16, 0.019973094021975557, 0.019973094021975557, 0.9709505944546682]
wind
[5.551115123125783e-16]
outlook
[5.551115123125783e-16, 0.5709505944546689]
temperature
[5.551115123125783e-16, 0.5709505944546689, 0.9709505944546682]
humidity
[5.551115123125783e-16, 0.5709505944546689, 0.9709505944546682, 0.019973094021975557]
humidity
```

```
import pprint

pprint.pprint(tree)
```

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

Output

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

Date:12-05-2023

Program-5:

Write a program to demonstrate the working simple Linear Regression algorithm. Use an appropriate data set to train and test

Dataset: Salary

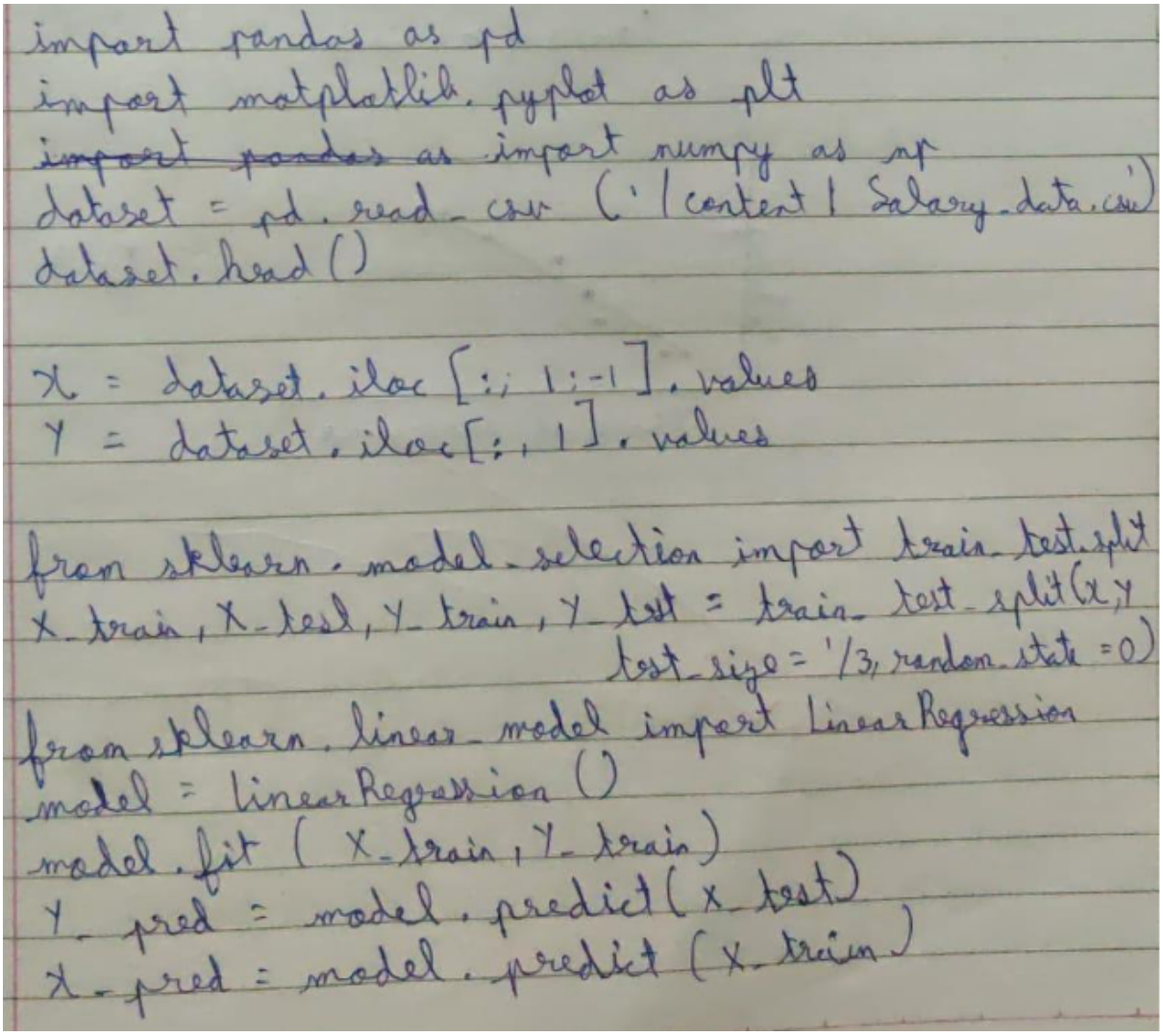
salary.csv × house_prices.csv

1 to 30 of 30 entries Filter

YearsExperience	Salary
1.1	39343
1.3	46205
1.5	37731
2	43525
2.2	39891
2.9	56642
3	60150
3.2	54445
3.2	64445
3.7	57189
3.9	63218
4	55794
4	56957
4.1	57081
4.5	61111
4.9	67938
5.1	66029
5.3	83088
5.9	81363
6	93940
6.8	91738
7.1	98273
7.9	101302
8.2	113812
8.7	109431
9	105582
9.5	116969
9.6	112635
10.3	122391
10.5	121872

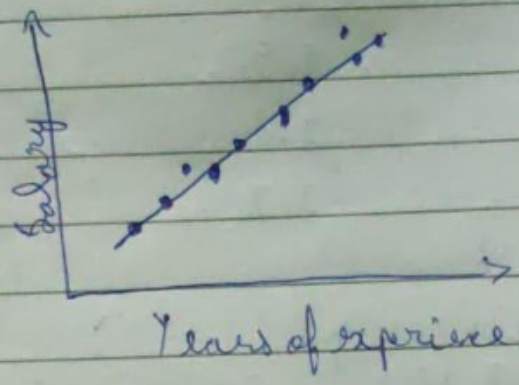
Show 100 per page

Observation:

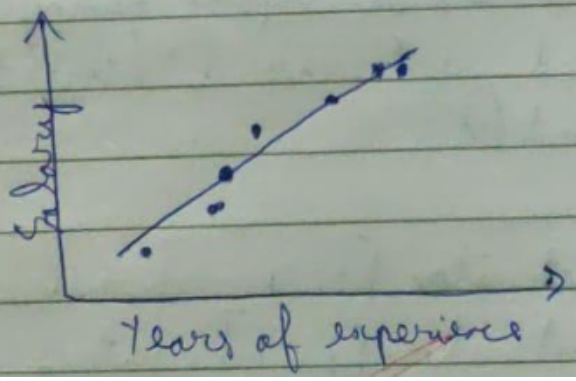


```
import pandas as pd
import matplotlib.pyplot as plt
import pandas as import numpy as np
dataset = pd.read_csv('/content/Salary_data.csv')
dataset.head()

X = dataset.iloc[:, 1:-1].values
Y = dataset.iloc[:, 1].values

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)
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit ( X_train, Y_train)
Y_pred = model.predict(X_test)
X_pred = model.predict (X_train)
```

```
plt.scatter (x_train, y_train, color = "Green")
plt.plot (x_train, x_pred, color = "Red")
plt.title ("Salary v/s Experience (Training Dataset)")
plt.xlabel ("Years of experience")
plt.ylabel ("Salary")
plt.show ()
```

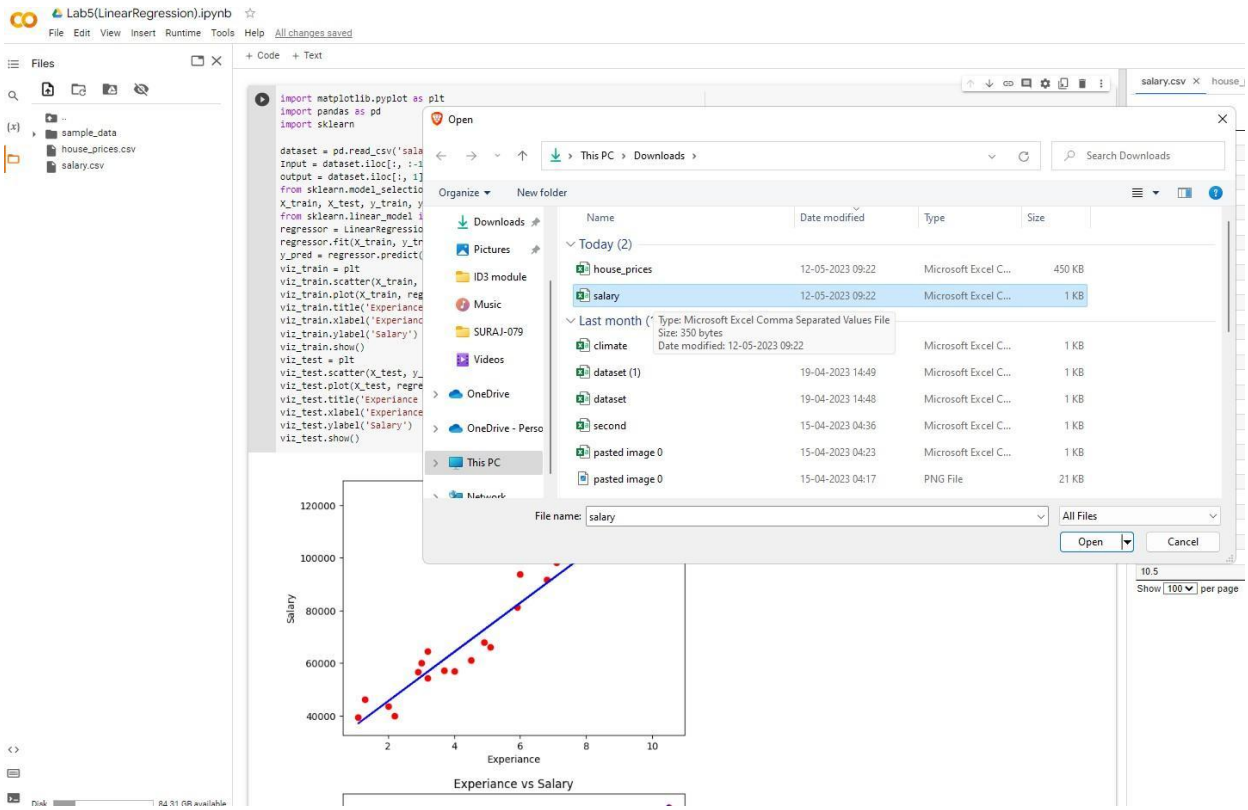


Training set



Test set

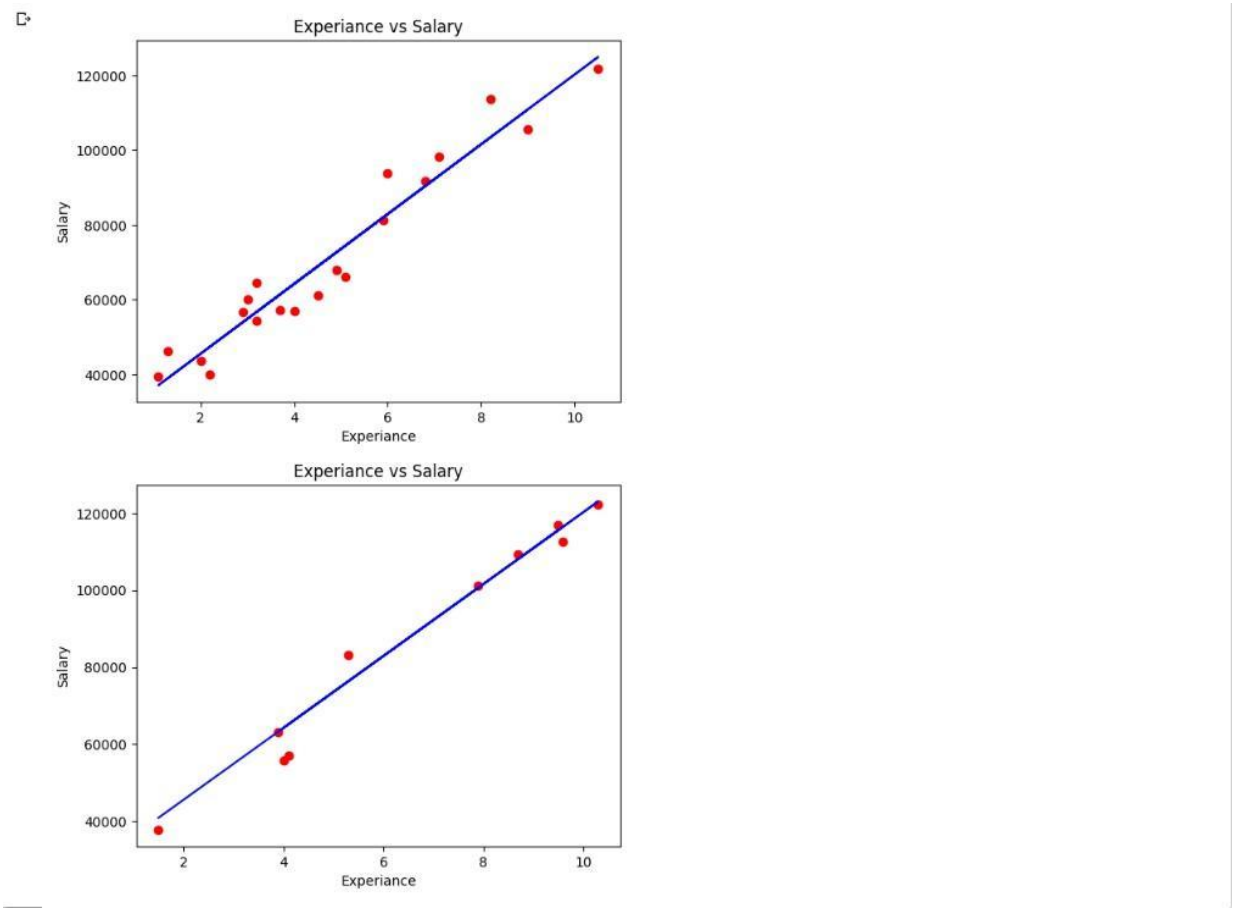
Uploading Dataset:



Program:

```
import matplotlib.pyplot as plt
import pandas as pd
import sklearn

dataset = pd.read_csv('salary.csv')
Input = dataset.iloc[:, :-1].values
output = dataset.iloc[:, 1].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(Input, output, test_size=1/3, random_state=0)
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
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('Experiance vs Salary')
viz_train.xlabel('Experiance')
viz_train.ylabel('Salary')
viz_train.show()
viz_test = plt
viz_test.scatter(X_test, y_test, color='red')
viz_test.plot(X_test, regressor.predict(X_test), color='blue')
viz_test.title('Experiance vs Salary')
viz_test.xlabel('Experiance')
viz_test.ylabel('Salary')
viz_test.show()
```

Output:


Experiance vs Salary
Salary
Experiance
120000
100000
80000
60000
40000
2
4
6
8
10
Experiance vs Salary
Salary
Experiance
120000
100000
80000
60000
40000
2
4
6
8
10

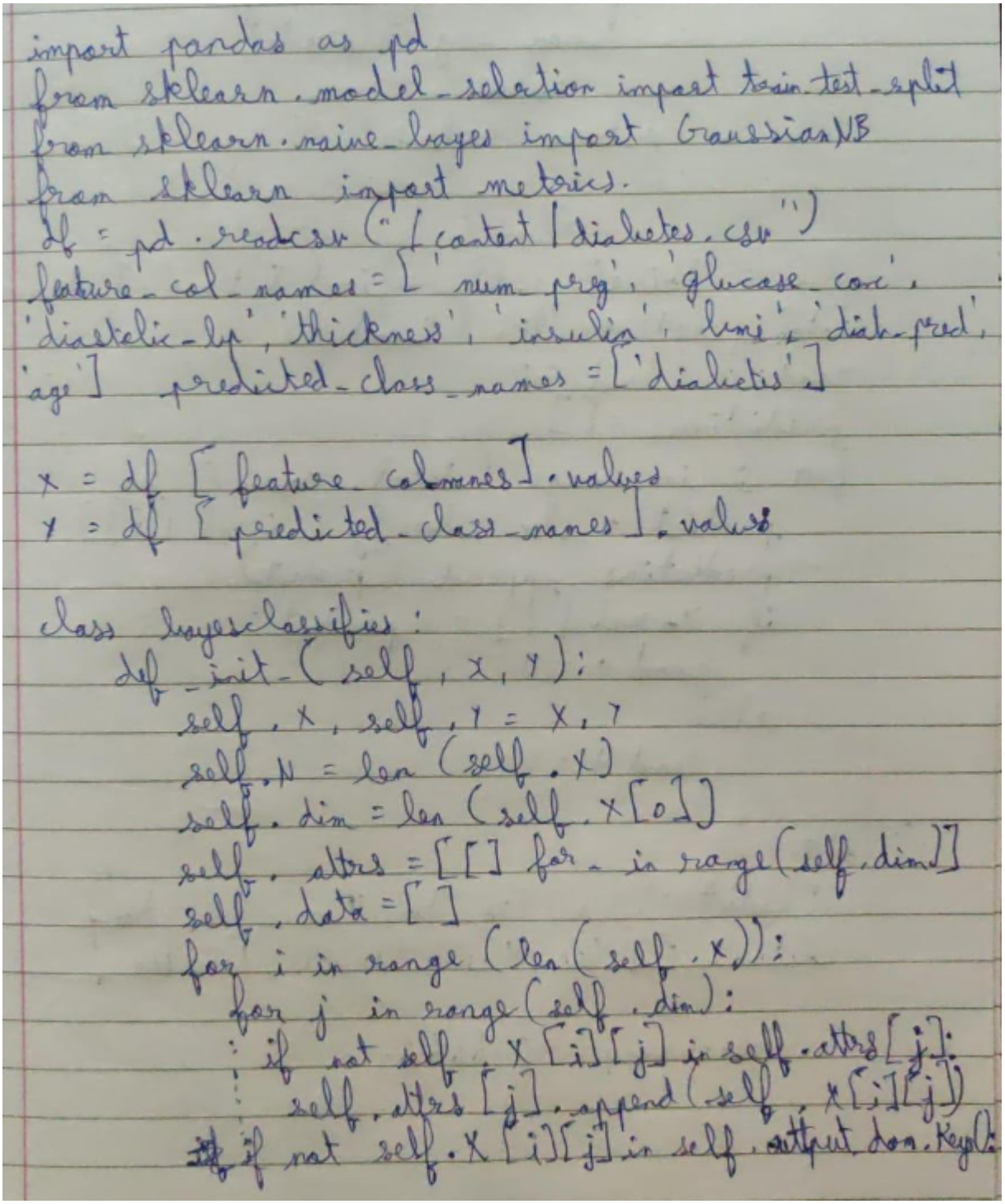
Date: 19.05.2023

Program 6 – Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file.

Data set: Climate Dataset

	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

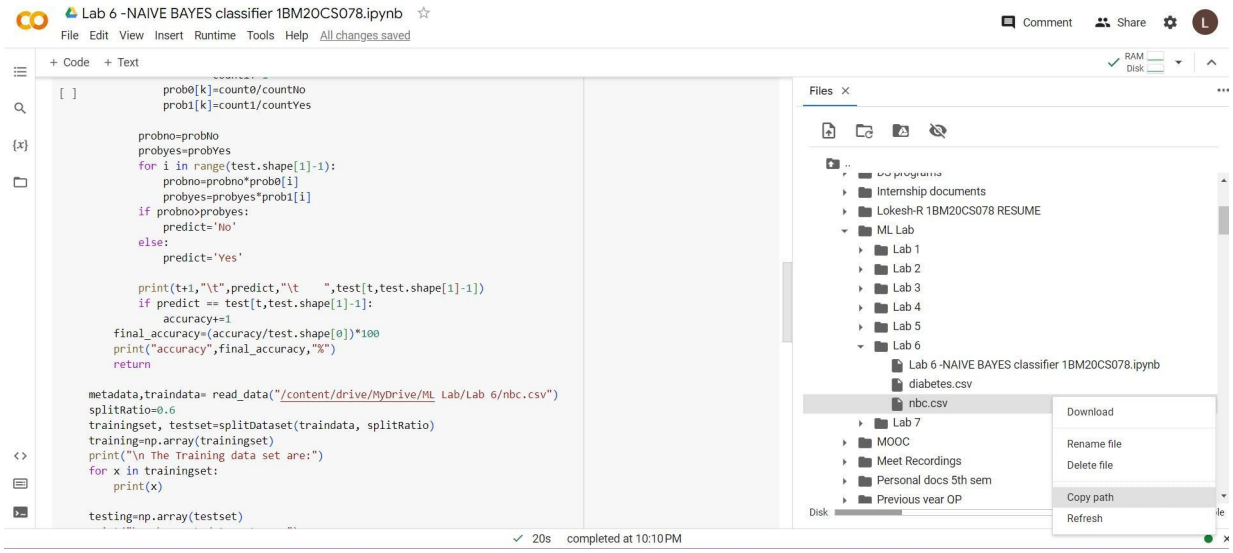
Observation:



```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics.
df = pd.read_csv("/content/diabetes.csv")
feature_col_names = ['num_preg', 'glucose_conc',
'diastolic_bp', 'thickness', 'insulin', 'bmi', 'diab_pred',
'age'] predicted_class_names = ['diabetes']

X = df[feature_columnes].values
y = df[predicted_class_names].values

class layerclassifier:
    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.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.x[i][j] in self.output_dom.keys():
```

```
        n = len(cases)
        prob = n/self.N
        if prob > max_arg:
            max_arg = prob
            solve = y
        return solve


nlc = NaiveBayesClassifier(X_train, Y_train)
total_cases = len(Y_val)
good = 0, bad = 0
predictions = []
for i in range(total_cases):
    predict = nlc.classify(X_val[i])
    predictions.append(predict)
    if Y_val[i] == predict:
        good += 1
    else
        bad += 1
```

Uploading csv file:



Program Execution:

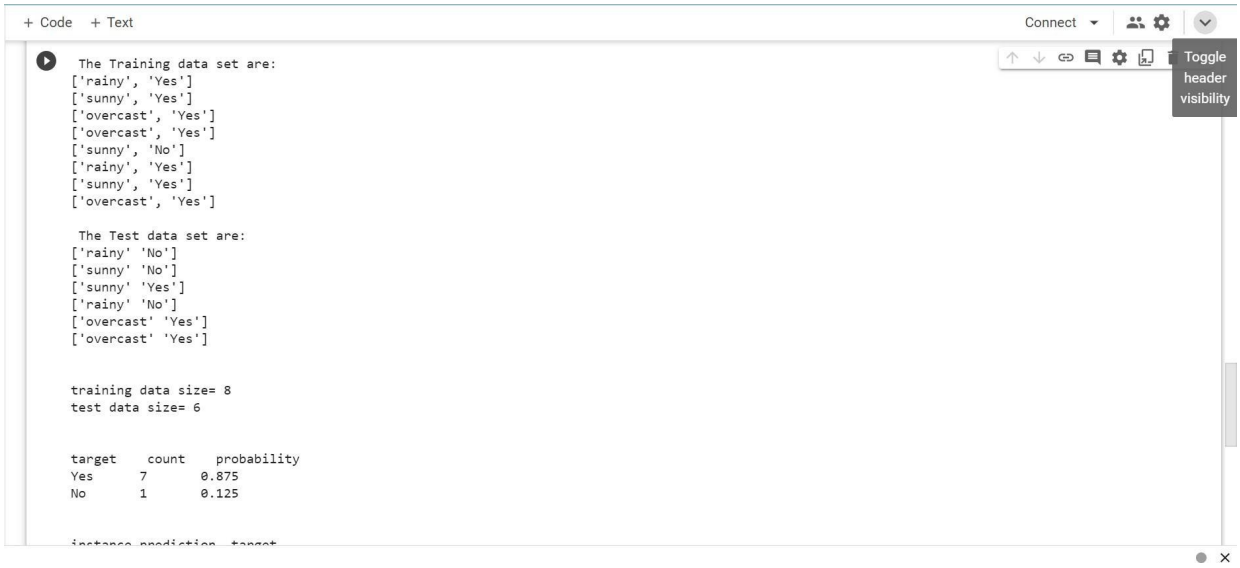
```
import numpy as np
import math
import csv
import pdb
def read_data(filename):

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

    return (metadata, traindata)

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]

def classify(data,test):
```

```
    countYes = 0
    countNo = 0
    probYes = 0
    probNo = 0
    print("\n")
    print("target    count    probability")

    for x in range(data.shape[0]):
        if data[x,data.shape[1]-1] == 'Yes':
            countYes +=1
        if data[x,data.shape[1]-1] == 'No':
            countNo +=1

    probYes=countYes/total_size
    probNo= countNo / total_size

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


    prob0 =np.zeros((test.shape[1]-1))
    prob1 =np.zeros((test.shape[1]-1))
    accuracy=0
    print("\n")
    print("instance prediction  target")
```

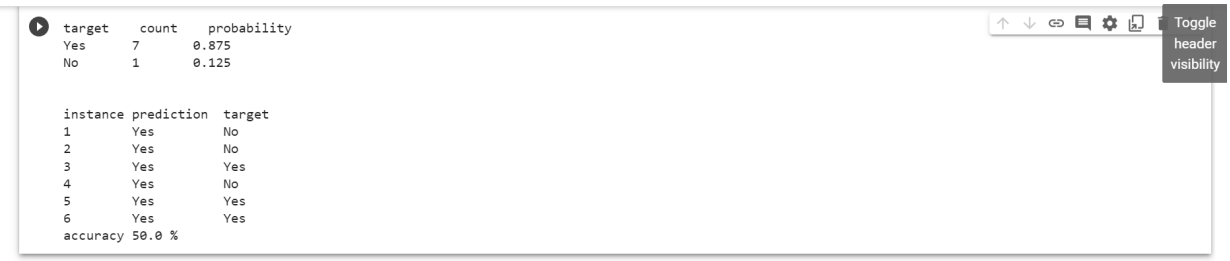
```
                #how many times appeared with no
                if test[t,k] == data[j,k] and data[j,data.shape[1]-1]=='No':
                    count0+=1
                #how many times appeared with yes
                if test[t,k]==data[j,k] and data[j,data.shape[1]-1]=='Yes':
                    count1+=1
            prob0[k]=count0/countNo
            prob1[k]=count1/countYes

        probno=probNo
        probyes=probYes
        for i in range(test.shape[1]-1):
            probno=probno*prob0[i]
            probyes=probyes*prob1[i]
        if probno>probyes:
            predict='No'
        else:
            predict='Yes'

        print(t+1,"\t",predict,"\t    ",test[t,test.shape[1]-1])
        if predict == test[t,test.shape[1]-1]:
            accuracy+=1
    final_accuracy=(accuracy/test.shape[0])*100
    print("accuracy",final_accuracy,"%")
    return

metadata,traindata= read_data("/content/sample_data/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
```


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 %
Toggle header visibility

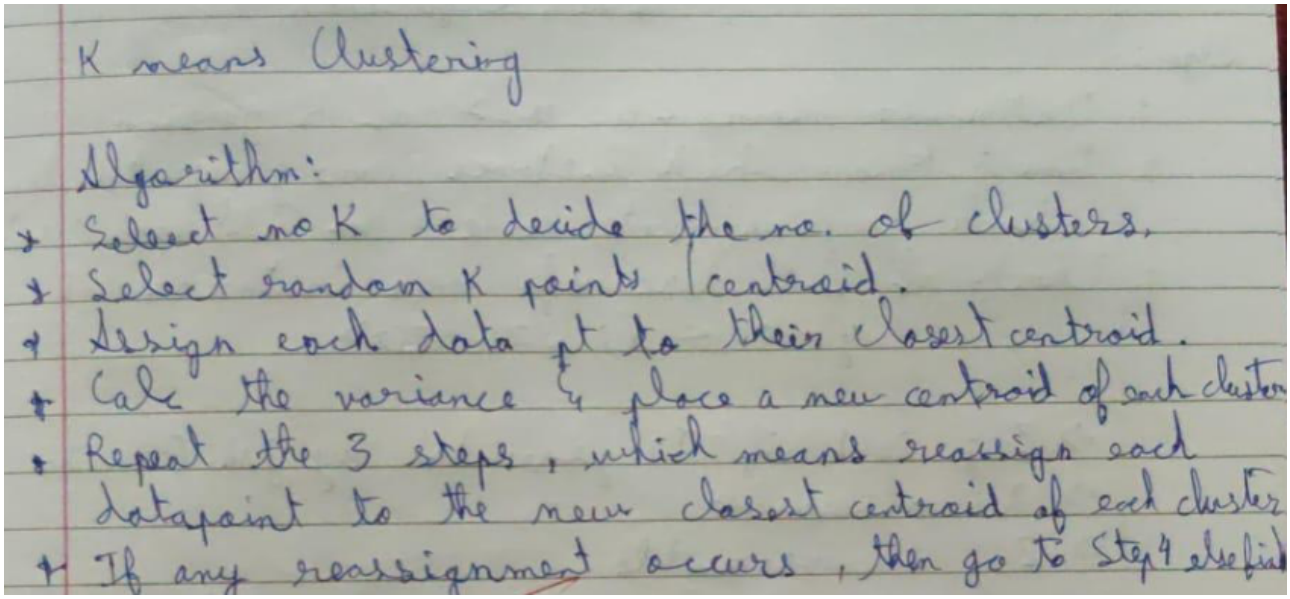
Date: 19.05.2023

Program 7 – Apply k-Means algorithm to cluster a set of data stored in a .CSV file.

Data set:

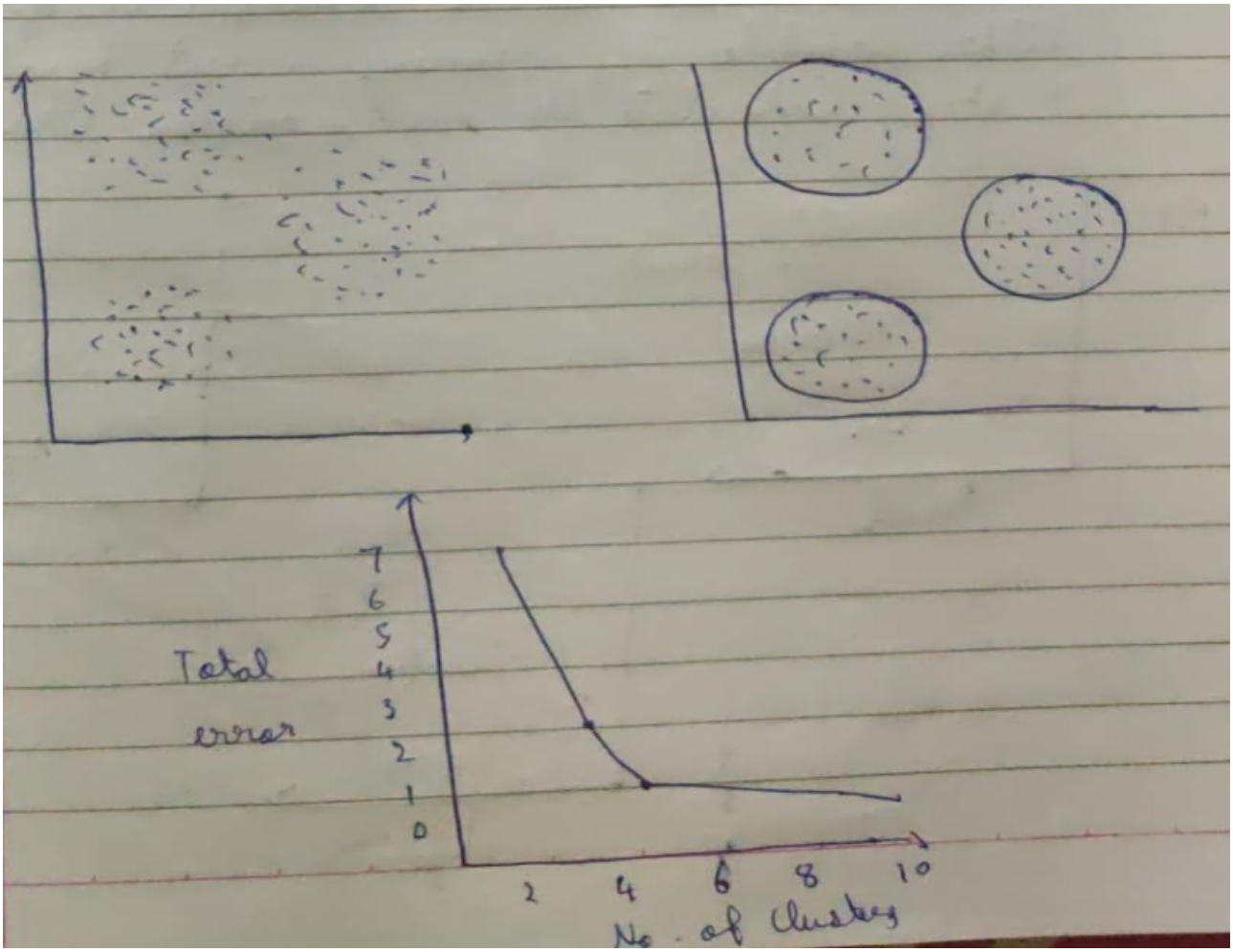
	A	B	C	D
1	ID	x	y	cluster
2	0	24.412	32.932	2
3	1	35.19	12.189	1
4	2	26.288	41.718	2
5	3	0.376	15.506	0
6	4	26.116	3.963	1
7	5	25.893	31.515	2
8	6	23.606	15.402	1
9	7	26.026	15.47	1
10	8	26.36	34.468	2
11	9	23.013	36.213	2
12	10	27.619	41.667	2
13	11	39.634	42.23	2
14	12	35.477	35.104	2
15	13	28.768	5.967	1
16	14	-0.684	21.105	0
17	15	3.387	17.61	0
18	16	32.988	3.412	1
19	17	34.258	9.031	1
20	18	6.313	29.426	0
21	19	33.699	37.535	2
22	20	4.718	12.125	0
23	21	21.054	8.067	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
28	26	3.884	28.616	0
29	27	29.091	34.539	2
30	28	14.843	23.263	0
31	29	32.189	40.421	2
32	30	32.572	16.944	1
33	31	33.673	13.163	1
34	32	29.189	13.228	1
35	33	25.994	34.444	2
36	34	16.513	23.396	0
37	35	23.492	11.142	1
38	36	26.878	36.009	2
39	37	31.604	36.911	2
40	38	34.078	33.627	2
41	39	11.286	18.092	0
42	40	30.15	9.642	1
43	41	36.569	45.827	2
44	42	3.983	15.639	0
45	43	12.891	23.832	0
46	44	21.314	13.264	1
47	45	29.101	44.781	2
48	46	30.671	9.294	1
49	47	35.139	0.803	1
50	48	35.563	42.759	2
51	49	36.028	16.779	1
52	50	9.776	18.988	0
53	51	24.268	5.693	1
54	52	-0.36	15.319	0
55	53	33.062	47.093	2
56	54	21.034	37.463	2
57	55	31.806	4.484	1
58	56	22.493	39.466	2
59	57	29.056	46.004	2
60	58	29.822	13.83	1
61	59	35.439	14.439	1

Observations:


K means Clustering

Algorithm:

* Select no K to decide the no. of clusters.
* Select random K points centroid.
* Assign each data pt to their closest centroid.
* Calc the variance & place a new centroid of each cluster
* Repeat the 3 steps, which means reassign each datapoint to the new closest centroid of each cluster
* If any reassignment occurs, then go to Step 4 else fin


Total error
7
6
5
4
3
2
1
0
2
4
6
8
10
No. of clusters

Program execution:

Open in Colab

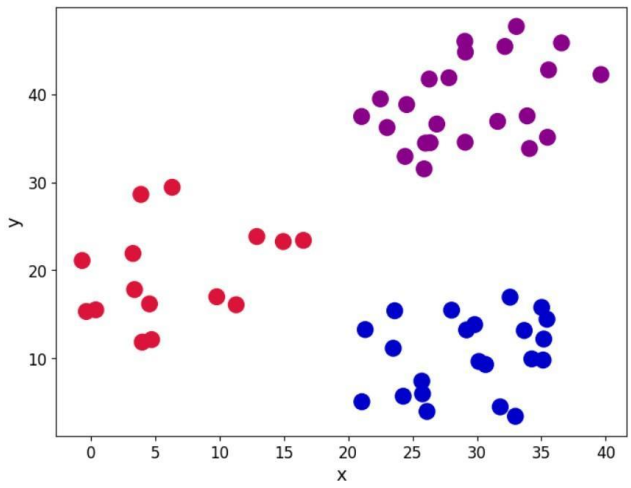
In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
%matplotlib inline

blobs = pd.read_csv('/content/drive/MyDrive/ML Lab/Lab 6/kmeans_blobs.csv')
colnames = list(blobs.columns[1:-1])
blobs.head()
```

Out[ ]:

	ID	x	y	cluster
0	0	24.412	32.932	2
1	1	35.190	12.189	1
2	2	26.288	41.718	2
3	3	0.376	15.506	0
4	4	26.116	3.963	1

In [ ]:
```
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.yticks(fontsize=12)
plt.show()
```



In [ ]:
```
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
```

Out[ ]:

	x	y
0	24.412	32.932
5	25.893	31.515
36	26.878	36.609

In [ ]:
```
def rsserr(a,b):
    '''
    Calculate the root of sum of squared errors.
    a and b are numpy arrays
    '''
    return np.square(np.sum((a-b)**2))
```

In [ ]:
```
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))
```

```
Error for centroid 0: 384.22
Error for centroid 1: 724.64
Error for centroid 2: 0.00
```

In [ ]:
```
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
```

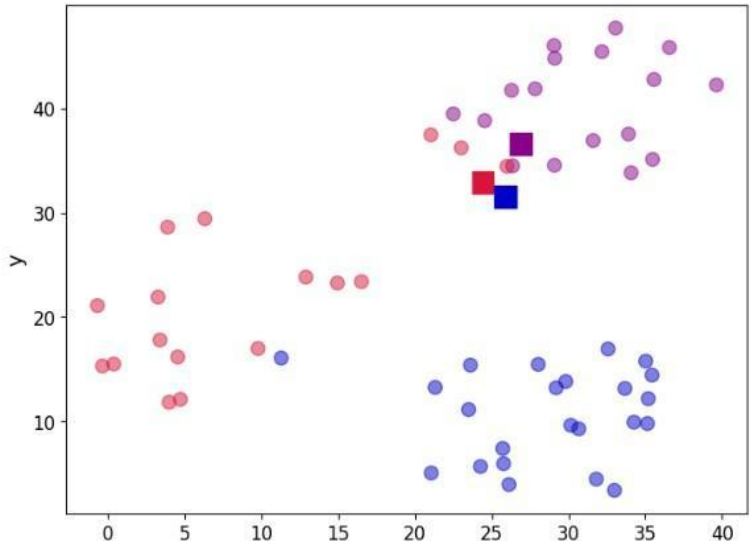
In [8]:
```
df['centroid'], df['error'] = centroid_assignation(df, centroids)
df.head()
```

Out[8]:

	x	y	centroid	error
0	24.412	32.932	0	0.000000
1	35.190	12.189	1	211534.211314
2	26.288	41.718	2	699.601495
3	0.376	15.506	0	776856.744109
4	26.116	3.963	1	576327.599678

In [9]:

```
fig, ax = plt.subplots(figsize=(8, 6))
plt.scatter(df.iloc[:,0], df.iloc[:,1],  marker = 'o',
            c=df['centroid'].astype('category'),
            cmap = customcmap, s=80, alpha=0.5)
plt.scatter(centroids.iloc[:,0], centroids.iloc[:,1],
            marker = 's', s=200, c=[0, 1, 2],
            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()
```



In [10]:
```
print("The total error is {0:.2f}".format(df['error'].sum()))
```

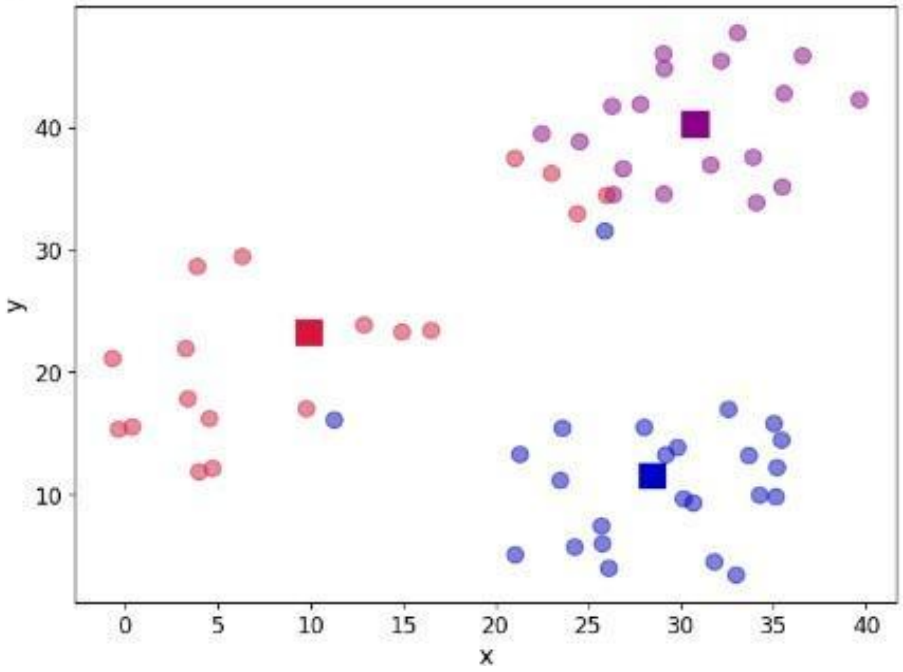
```
The total error is 11927659.01
```

In [11]:
```
centroids = df.groupby('centroid').agg('mean').loc[:, colnames].reset_index(drop = True)
centroids
```

Out[11]:

	x	y
0	9.889444	23.242611
1	28.435750	11.546250
2	30.759333	40.311167

In [12]:
```
fig, ax = plt.subplots(figsize=(8, 6))
plt.scatter(df.iloc[:,0], df.iloc[:,1],  marker = 'o',
            c=df['centroid'].astype('category'),
            cmap = customcmap, s=80, alpha=0.5)
plt.scatter(centroids.iloc[:,0], centroids.iloc[:,1],
            marker = 's', s=200,
            c=[0, 1, 2], 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()
```



In [13]:

```
def kmeans(dset, k=2, tol=1e-4):
    '''
    K-means implementationd for a
    `dset`:  DataFrame with observations
    `k`: number of clusters, default k=2
    `tol`: tolerance=1E-4
    '''
    # Let us work in a copy, so we don't mess the original
    working_dset = dset.copy()
    # We define some variables to hold the error, the
    # stopping signal and a counter for the iterations
    err = []
    goahead = True
    j = 0

    # Step 2: Initiate clusters by defining centroids
    centroids = initiate_centroids(k, dset)
    while(goahead):
        # Step 3 and 4 - Assign centroids and calculate error
        working_dset['centroid'], j_err = centroid_assignation(working_dset, centroids)
        err.append(sum(j_err))

        # Step 5 - Update centroid position
        centroids = working_dset.groupby('centroid').agg('mean').reset_index(drop = True)

        # Step 6 - Restart the iteration
        if j>0:
            # Is the error less than a tolerance (1E-4)
            if err[j-1]-err[j]<=tol:
                goahead = False
        j+=1

    working_dset['centroid'], j_err = centroid_assignation(working_dset, centroids)
    centroids = working_dset.groupby('centroid').agg('mean').reset_index(drop = True)
    return working_dset['centroid'], j_err, centroids
```

In [14]:

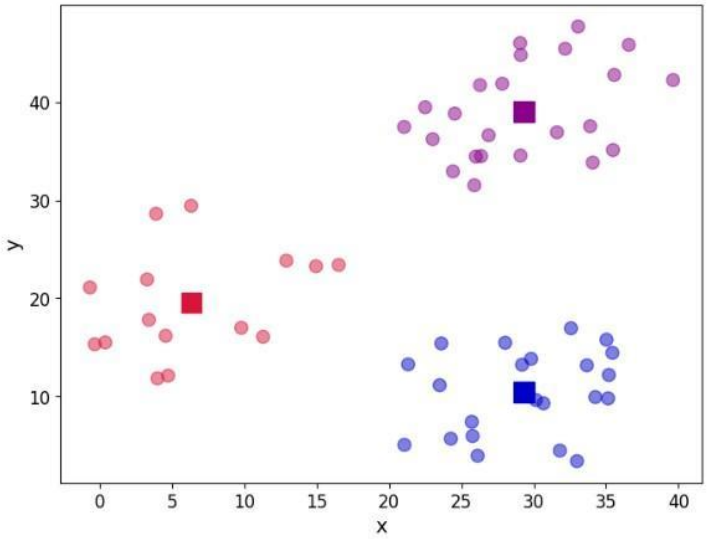
```
np.random.seed(42)
df['centroid'], df['error'], centroids =  kmeans(df[['x','y']], 3)
df.head()
```

Out[14]:

	x	y	centroid	error
0	24.412	32.932	2	3767.568743
1	35.190	12.189	1	1399.889001
2	26.288	41.718	2	262.961097
3	0.376	15.506	0	2683.086425
4	26.116	3.963	1	2723.650198

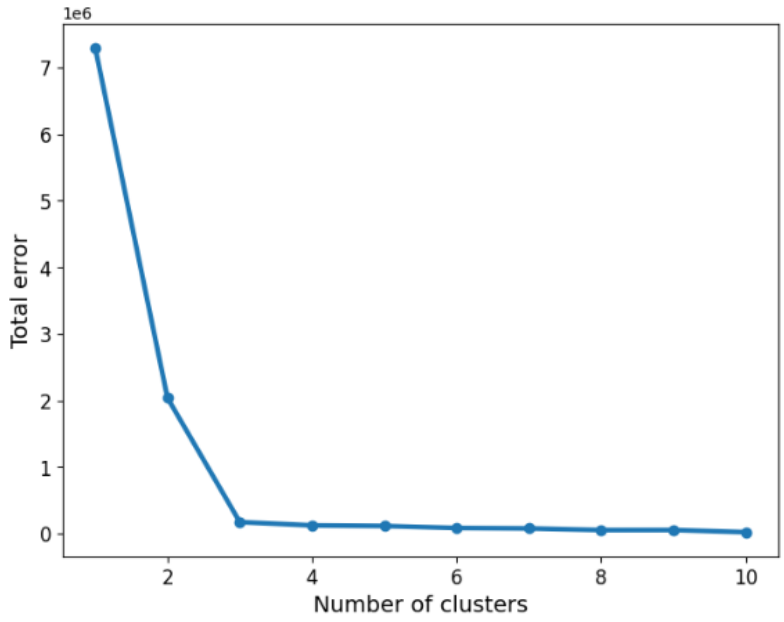
In [15]:

```
fig, ax = plt.subplots(figsize=(8, 6))
plt.scatter(df.iloc[:,0], df.iloc[:,1],  marker = 'o',
            c=df['centroid'].astype('category'),
            cmap = customcmap, s=80, alpha=0.5)
plt.scatter(centroids.iloc[:,0], centroids.iloc[:,1],
            marker = 's', s=200, c=[0, 1, 2],
            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()
```



In [16]:

```
err_total = []
n = 10

df_elbow = blobs[['x','y']]

for i in range(n):
    _, my_errs, _ = kmeans(df_elbow, i+1)
    err_total.append(sum(my_errs))
fig, ax = plt.subplots(figsize=(8, 6))
plt.plot(range(1,n+1), err_total, linewidth=3, marker='o')
ax.set_xlabel(r'Number of clusters', fontsize=14)
ax.set_ylabel(r'Total error', fontsize=14)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.show()
```

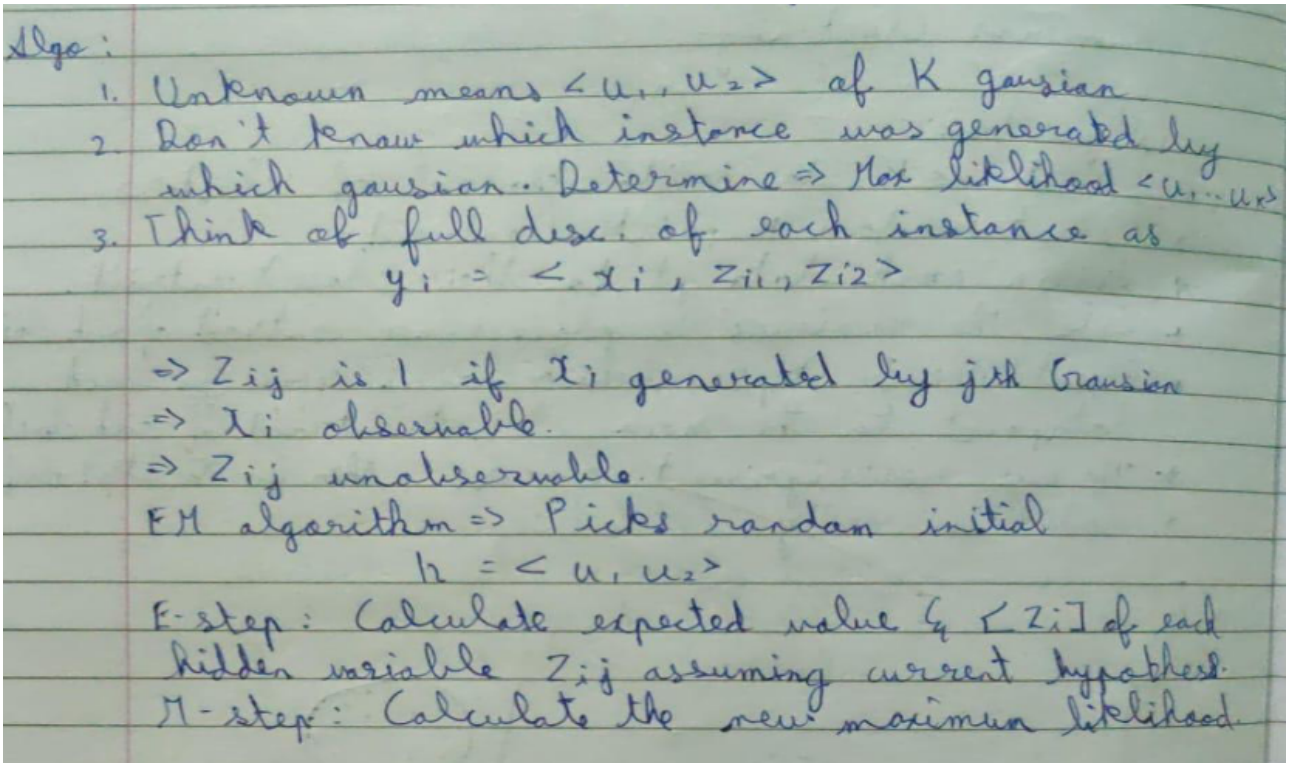


Date: 10.06.2023

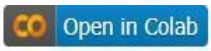
Program 8 – Apply EM algorithm to cluster a set of data stored in a .CSV file. Compare the results of k-Means algorithm and EM algorithm.

Data set: Iris Dataset

Observation:



Program execution :



In [1]:

```
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:
        break
    ll_old = ll_new

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
```

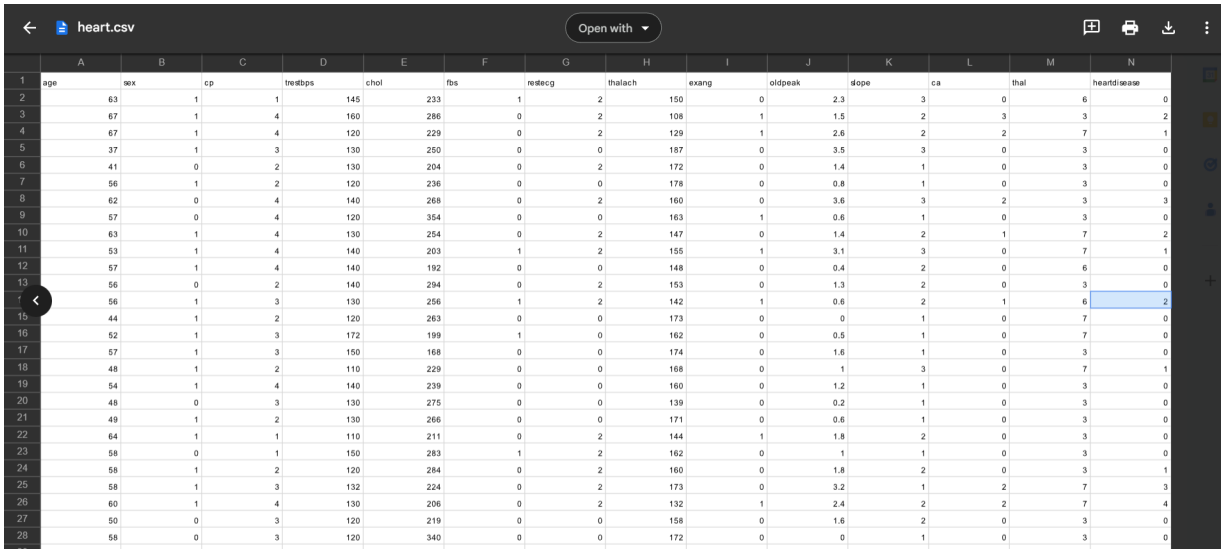
```
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y.Targets],s=40)
plt.title('Real')

# 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')
```

```
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
```

17.06.2023

Program 9: Write a program to construct a Bayesian Network considering training data. Use this model to make predictions.

Dataset-heart

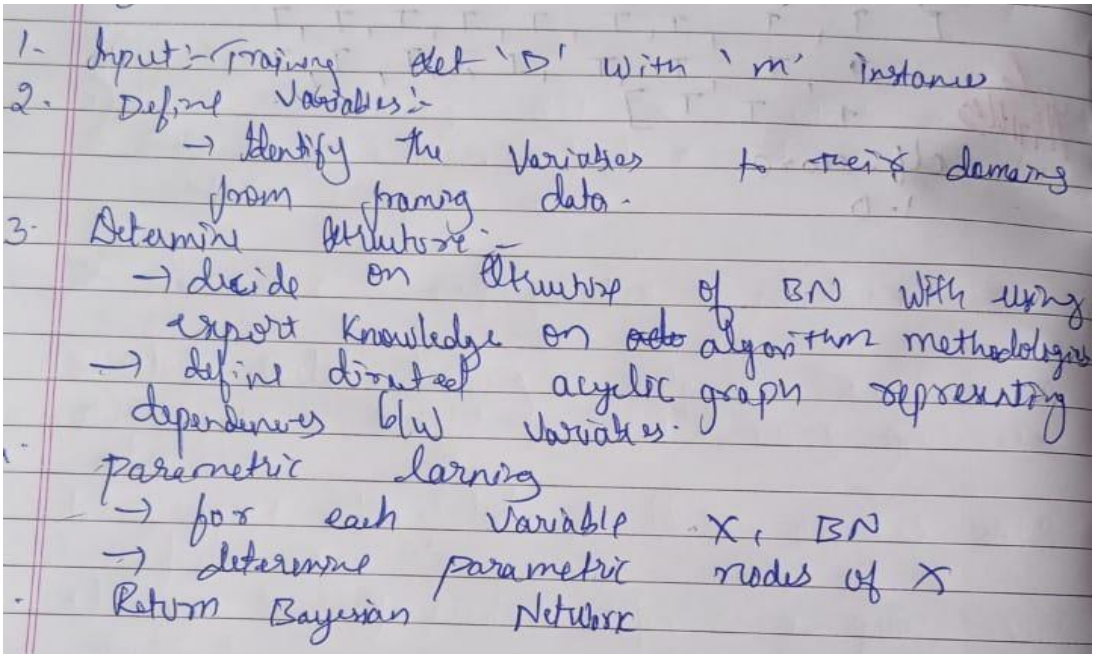


heart.csv

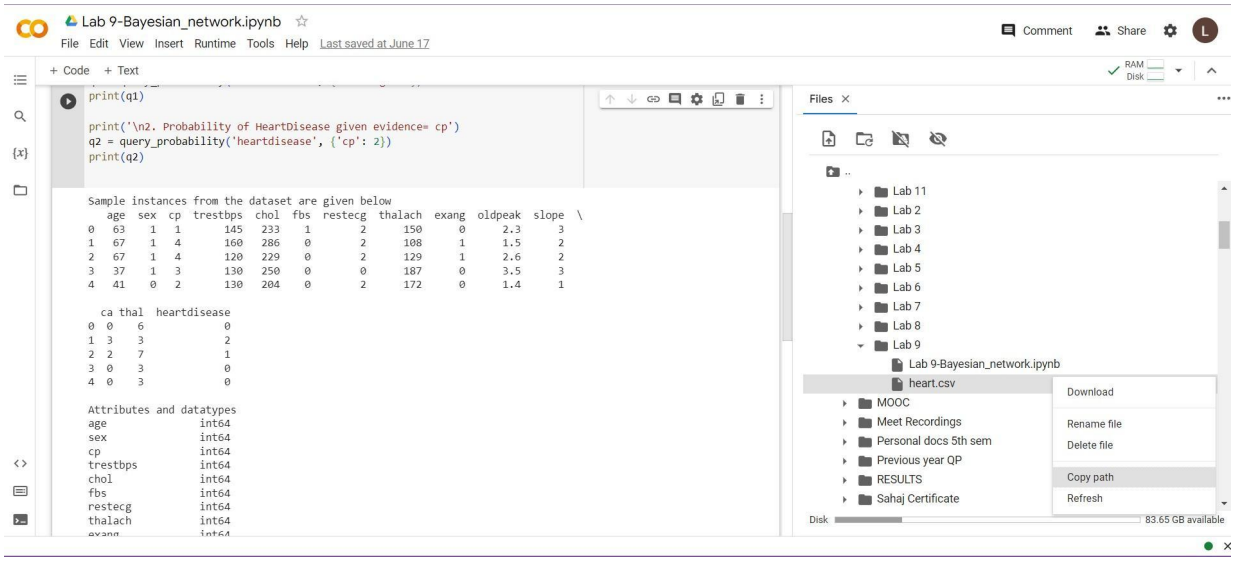
Open with

	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

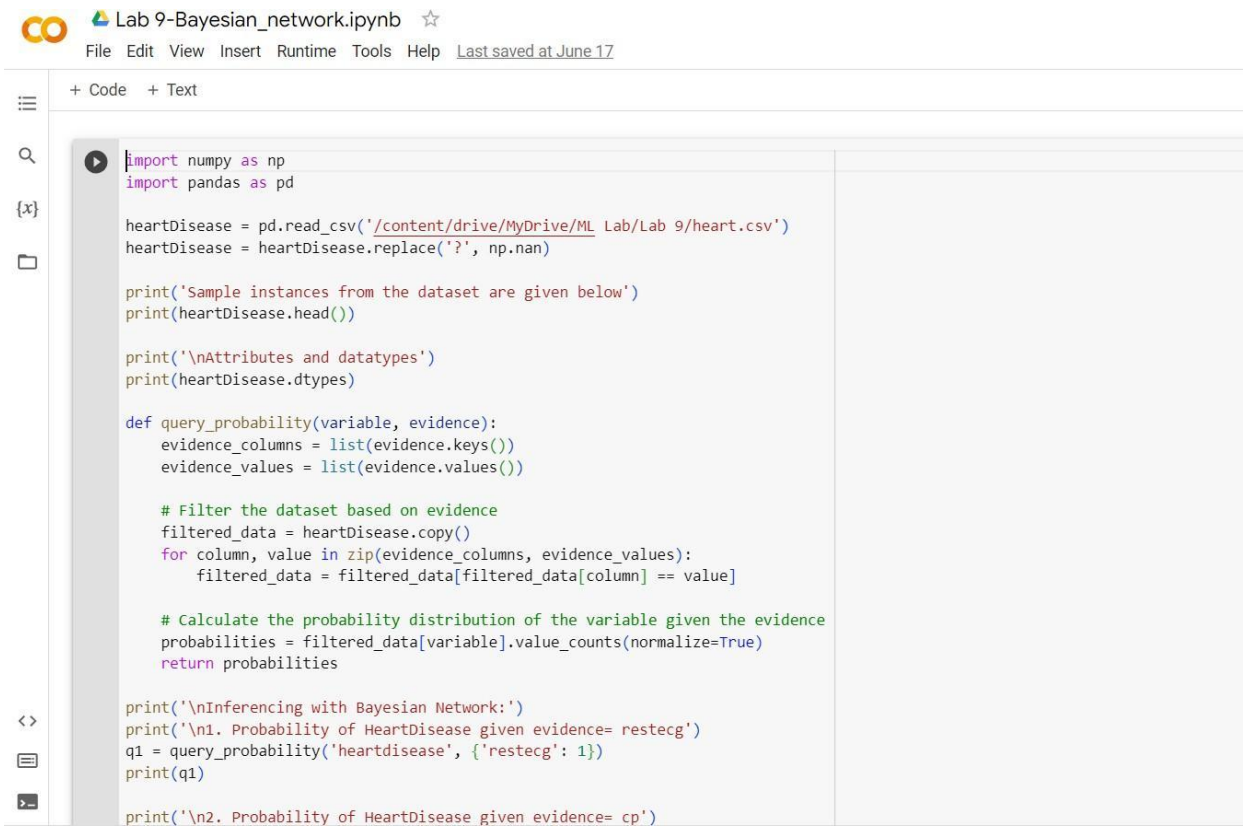
Algorithm:

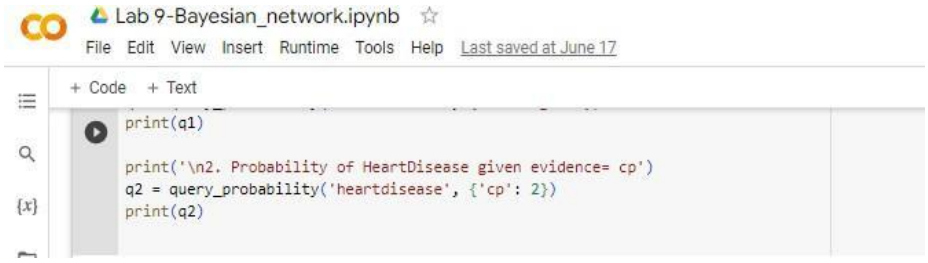

1. Input:- Training set 'D' with 'm' instances
2. Define Variables:-
   → Identify the variables to their domains from training data.
3. Determine structure:-
   → decide on structure of BN with using expert knowledge on algorithm methodologies
   → define directed acyclic graph representing dependencies b/w variables.
4. parametric learning
   → for each variable X, BN
   → determine parametric nodes of X
5. Return Bayesian Network

Uploading csv :



Program execution :





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
0    0.82
1    0.12
3    0.04
2    0.02
Name: heartdisease, dtype: float64
```

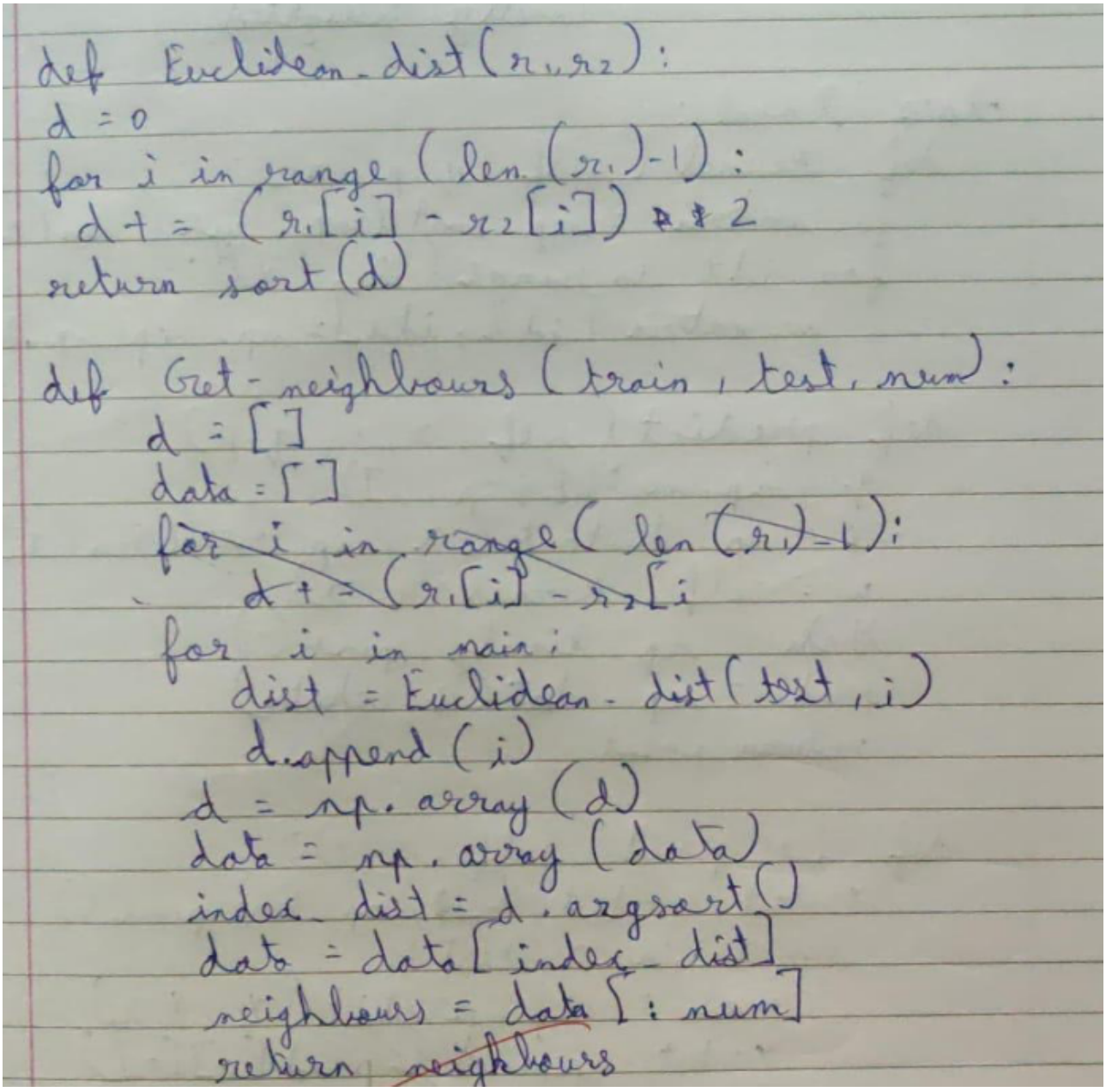
Date: 17.05.2023

Program 10:

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

Dataset :Iris Dataset

Algorithm:



```
def Euclidean_dist(r1,r2):
d = 0
for i in range(len(r1)-1):
  d += (r1[i] - r2[i]) ** 2
return sort(d)

def Get_neighbours(train, test, num):
    d = []
    data = []
    for i in range(len(r1)-1):
        d += (r1[i] - r2[i]
    for i in main:
        dist = Euclidean_dist(test, i)
        d.append(i)
    d = np.array(d)
    data = np.array(data)
    index_dist = d.argsort()
    data = data[index_dist]
    neighbours = data[:num]
    return neighbours
```

Program:

```
import scipy.spatial
from collections import Counter
```

```
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):
        predictions = self.predict(X_test)
        return (predictions == y_test).sum() / len(y_test)
```

```
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
```

Output:

```
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

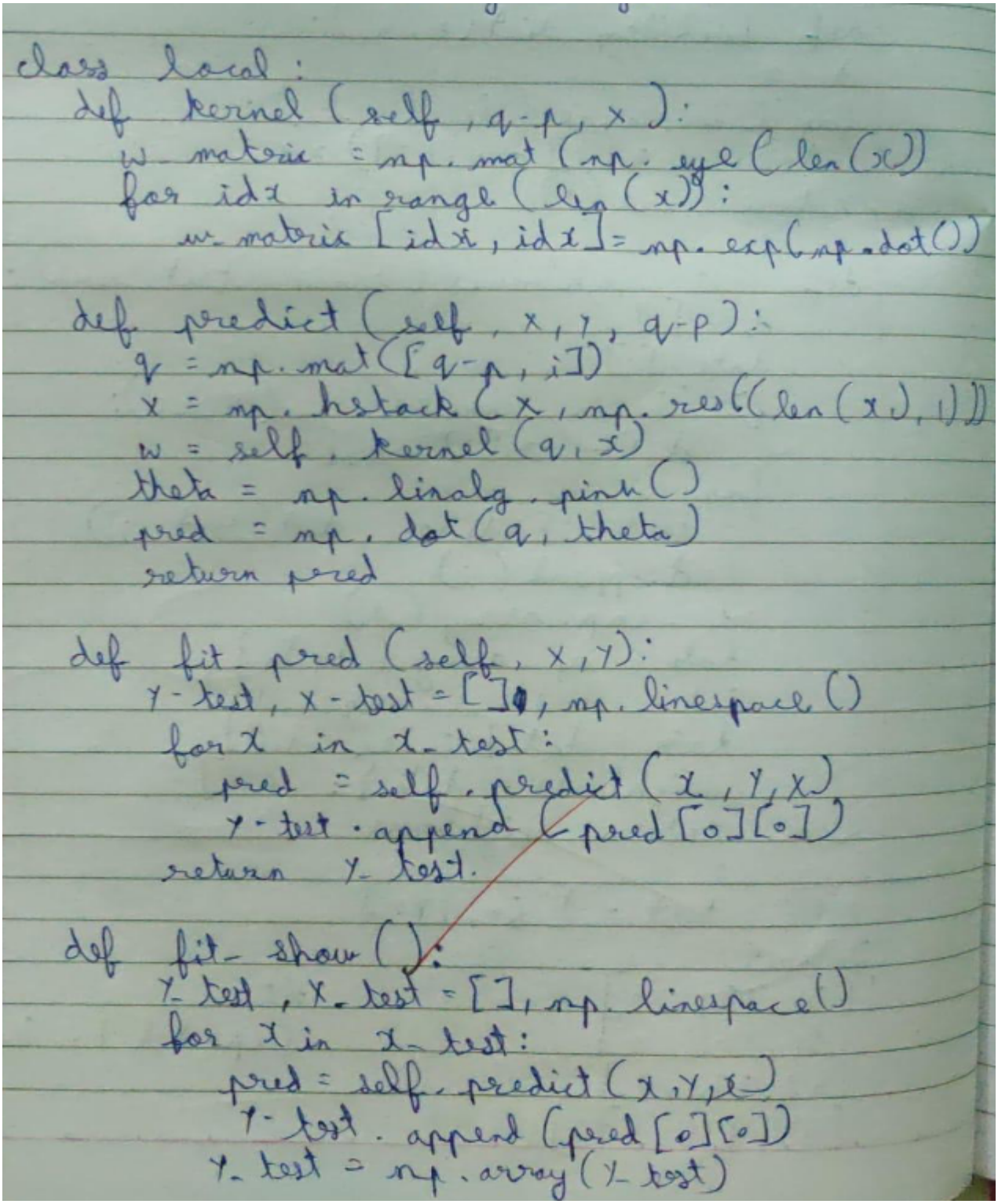
Program 11:

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

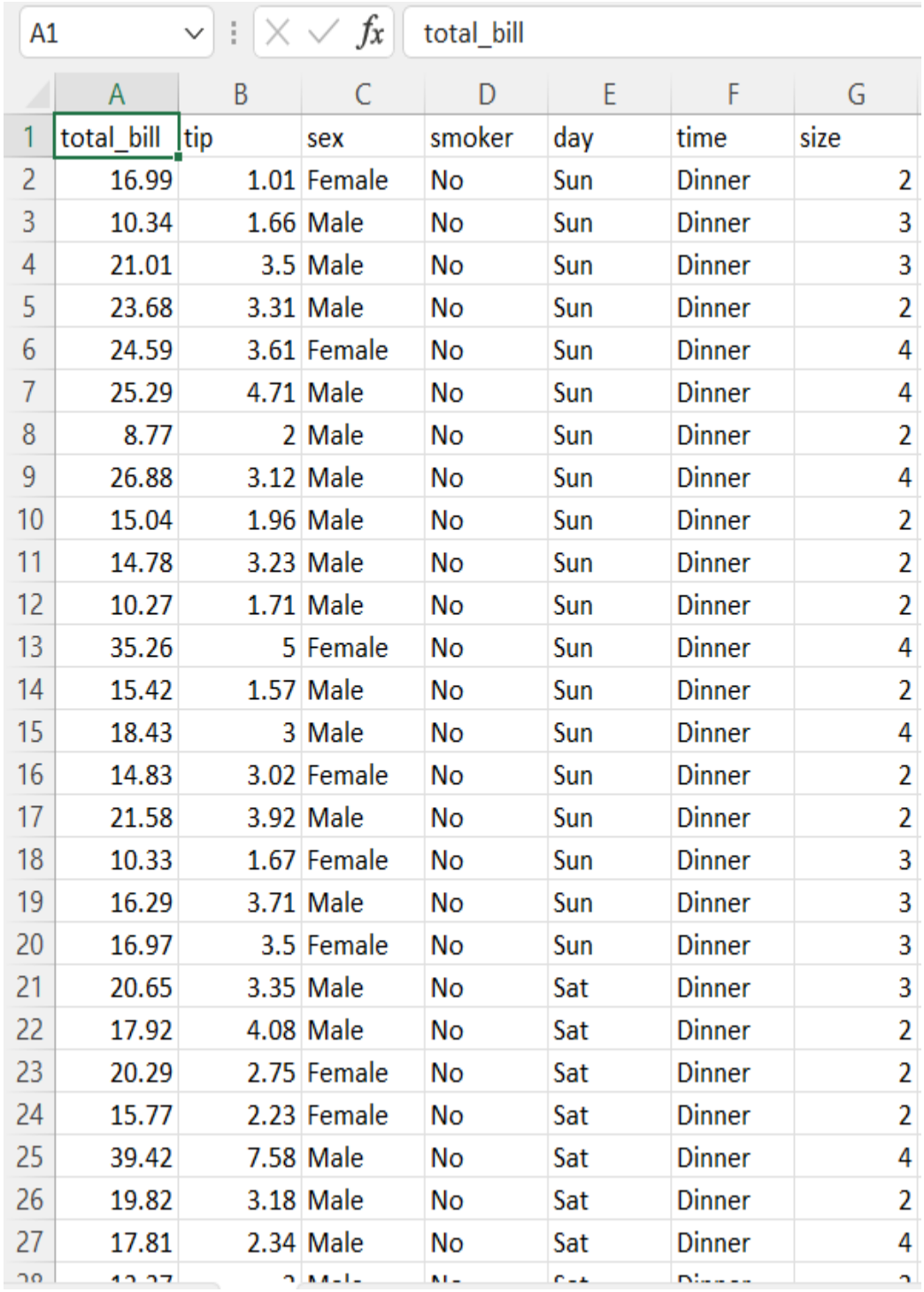
	A	B	C	D	E	F	G
1	total_bill	tip	sex	smoker	day	time	size
2	16.99	1.01	Female	No	Sun	Dinner	2
3	10.34	1.66	Male	No	Sun	Dinner	3
4	21.01	3.5	Male	No	Sun	Dinner	3
5	23.68	3.31	Male	No	Sun	Dinner	2
6	24.59	3.61	Female	No	Sun	Dinner	4
7	25.29	4.71	Male	No	Sun	Dinner	4
8	8.77	2	Male	No	Sun	Dinner	2
9	26.88	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:



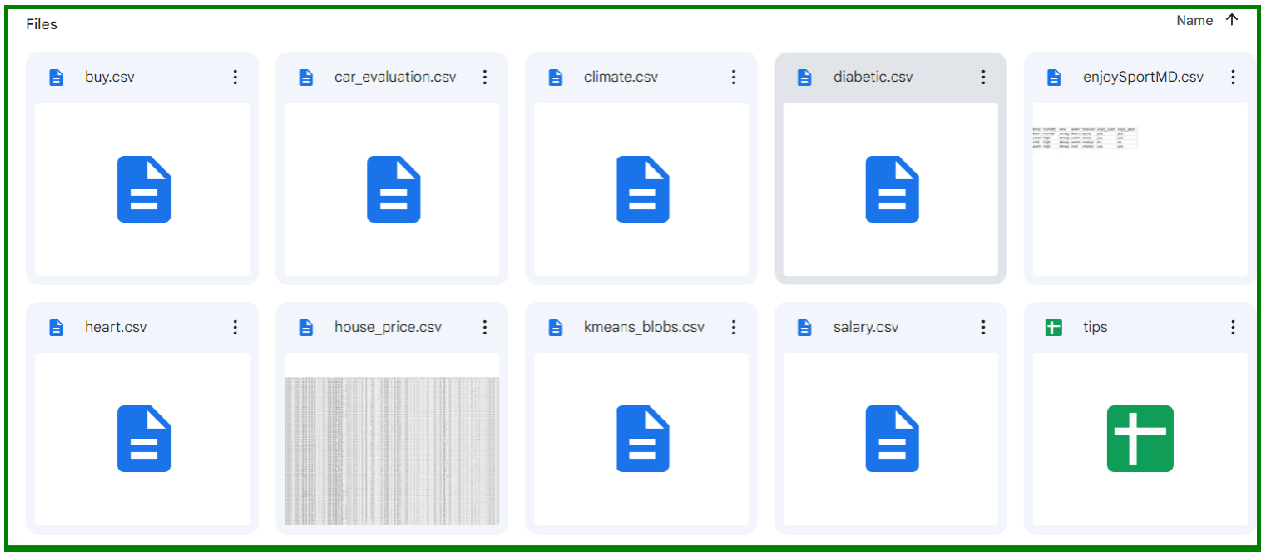
```
class local:
    def kernel(self, q-p, x):
        w_matrix = np.mat(np.eye(len(x)))
        for idx in range(len(x)):
            w_matrix[idx, idx] = np.exp(np.dot())

    def predict(self, x, y, q-p):
        q = np.mat([q-p, 1])
        x = np.hstack(x, np.ones((len(x), 1)))
        w = self.kernel(q, x)
        theta = np.linalg.pinv()
        pred = np.dot(q, theta)
        return pred

    def fit_pred(self, x, y):
        y-test, x-test = [], np.linspace()
        for x in x_test:
            pred = self.predict(x, y, x)
            y-test.append(pred[0][0])
        return y_test.

    def fit_show():
        y_test, x_test = [], np.linspace()
        for x in x_test:
            pred = self.predict(x, y, x)
            y-test.append(pred[0][0])
        y_test = np.array(y_test)
```

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


Files
Name
buy.csv
car_evaluation.csv
climate.csv
diabetic.csv
enjoySportMD.csv
heart.csv
house_price.csv
kmeans_blobs.csv
salary.csv
tips

Program:


Locally weighted.ipynb
File Edit View Insert Runtime Tools Help Last edited on June 17
Comment
Share
M
+ Code + Text
Connect
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):

```
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: