Name: Sai Ganesh S
USN: 1BM20CS137

MACHINE LEARNING LAB OBSERVATION

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

IRIS DATASET:

```
from sklearn.datasets import load_iris
iris = load_iris()
```

```
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], [5.1, 3.8, 1.5, 0.3], [5.4, 3.4, 1.7, 0.2], [5.1, 3.7, 1.5,
0.4], [4.6, 3.6, 1. , 0.2], [5.1, 3.3, 1.7, 0.5], [4.8, 3.4, 1.9, 0.2], [5. , 3. ,
1.6, 0.2], [5. , 3.4, 1.6, 0.4], [5.2, 3.5, 1.5, 0.2], [5.2, 3.4, 1.4, 0.2], [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], [4.6, 3.2, 1.4, 0.2],
[5.3, 3.7, 1.5, 0.2], [5. , 3.3, 1.4, 0.2], [7. , 3.2, 4.7, 1.4], [6.4, 3.2, 4.5,
1.5], [6.9, 3.1, 4.9, 1.5], [5.5, 2.3, 4. , 1.3], [6.5, 2.8, 4.6, 1.5], [5.7, 2.8,
4.5, 1.3], [6.3, 3.3, 4.7, 1.6], [4.9, 2.4, 3.3, 1. ], [6.6, 2.9, 4.6, 1.3], [5.2,
2.7, 3.9, 1.4], [5. , 2. , 3.5, 1. ], [5.9, 3. , 4.2, 1.5], [6. , 2.2, 4. , 1. ],
[6.1, 2.9, 4.7, 1.4], [5.6, 2.9, 3.6, 1.3], [6.7, 3.1, 4.4, 1.4], [5.6, 3. , 4.5,
1.5], [5.8, 2.7, 4.1, 1. ], [6.2, 2.2, 4.5, 1.5], [5.6, 2.5, 3.9, 1.1], [5.9, 3.2,
4.8, 1.8], [6.1, 2.8, 4. , 1.3], [6.3, 2.5, 4.9, 1.5], [6.1, 2.8, 4.7, 1.2], [6.4,
2.9, 4.3, 1.3], [6.6, 3. , 4.4, 1.4], [6.8, 2.8, 4.8, 1.4], [6.7, 3. , 5. , 1.7],
[6. , 2.9, 4.5, 1.5], [5.7, 2.6, 3.5, 1. ], [5.5, 2.4, 3.8, 1.1], [5.5, 2.4, 3.7,
1. ], [5.8, 2.7, 3.9, 1.2], [6. , 2.7, 5.1, 1.6], [5.4, 3. , 4.5, 1.5], [6. , 3.4,
4.5, 1.6], [6.7, 3.1, 4.7, 1.5], [6.3, 2.3, 4.4, 1.3], [5.6, 3. , 4.1, 1.3], [5.5,
2.5, 4. , 1.3], [5.5, 2.6, 4.4, 1.2], [6.1, 3. , 4.6, 1.4], [5.8, 2.6, 4. , 1.2],
[5. , 2.3, 3.3, 1. ], [5.6, 2.7, 4.2, 1.3], [5.7, 3. , 4.2, 1.2], [5.7, 2.9, 4.2,
1.3], [6.2, 2.9, 4.3, 1.3], [5.1, 2.5, 3. , 1.1], [5.7, 2.8, 4.1, 1.3], [6.3, 3.3,
6. , 2.5], [5.8, 2.7, 5.1, 1.9], [7.1, 3. , 5.9, 2.1], [6.3, 2.9, 5.6, 1.8], [6.5,
3. , 5.8, 2.2], [7.6, 3. , 6.6, 2.1], [4.9, 2.5, 4.5, 1.7], [7.3, 2.9, 6.3, 1.8],
[6.7, 2.5, 5.8, 1.8], [7.2, 3.6, 6.1, 2.5], [6.5, 3.2, 5.1, 2. ], [6.4, 2.7, 5.3,
1.9], [6.8, 3. , 5.5, 2.1], [5.7, 2.5, 5. , 2. ], [5.8, 2.8, 5.1, 2.4], [6.4, 3.2,
5.3, 2.3], [6.5, 3. , 5.5, 1.8], [7.7, 3.8, 6.7, 2.2], [7.7, 2.6, 6.9, 2.3], [6. ,
2.2, 5. , 1.5], [6.9, 3.2, 5.7, 2.3], [5.6, 2.8, 4.9, 2. ], [7.7, 2.8, 6.7, 2. ],
[6.3, 2.7, 4.9, 1.8], [6.7, 3.3, 5.7, 2.1], [7.2, 3.2, 6. , 1.8], [6.2, 2.8, 4.8,
1.8], [6.1, 3. , 4.9, 1.8], [6.4, 2.8, 5.6, 2.1], [7.2, 3. , 5.8, 1.6], [7.4, 2.8,
6.1, 1.9], [7.9, 3.8, 6.4, 2. ], [6.4, 2.8, 5.6, 2.2], [6.3, 2.8, 5.1, 1.5], [6.1,
2.6, 5.6, 1.4], [7.7, 3. , 6.1, 2.3], [6.3, 3.4, 5.6, 2.4], [6.4, 3.1, 5.5, 1.8],
[6. , 3. , 4.8, 1.8], [6.9, 3.1, 5.4, 2.1], [6.7, 3.1, 5.6, 2.4], [6.9, 3.1, 5.1,
2.3], [5.8, 2.7, 5.1, 1.9], [6.8, 3.2, 5.9, 2.3], [6.7, 3.3, 5.7, 2.5], [6.7, 3. ,
5.2, 2.3], [6.3, 2.5, 5. , 1.9], [6.5, 3. , 5.2, 2. ], [6.2, 3.4, 5.4, 2.3], [5.9,
3. , 5.1, 1.8]]), 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]), 'frame': None,
'target_names': array(['setosa', 'versicolor', 'virginica'], dtype='<U10'), 'DESCR':
'.. _iris_dataset:\n\nIris plants dataset\n--------------------\n\n**Data Set
Characteristics:**\n\n :Number of Instances: 150 (50 in each of three
classes)\n :Number of Attributes: 4 numeric, predictive attributes and the
class\n :Attribute Information:\n - sepal length in cm\n - sepal width in cm\n -
petal length in cm\n - petal width in cm\n - class:\n - Iris-Setosa\n - Iris-
Versicolour\n - Iris-Virginica\n \n :Summary Statistics:\n\n ============== ==== ====
======= ===== ====================\n Min Max Mean SD Class Correlation\n
============== ==== ==== ======= ===== ====================\n sepal length: 4.3 7.9
5.84 0.83 0.7826\n sepal width: 2.0 4.4 3.05 0.43 -0.4194\n petal length: 1.0 6.9
3.76 1.76 0.9490 (high!)\n petal width: 0.1 2.5 1.20 0.76 0.9565 (high!)\n
============== ==== ==== ======= ===== ====================\n\n :Missing Attribute
Values: None\n :Class Distribution: 33.3% for each of 3 classes.\n :Creator: R.A.
Fisher\n :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)\n :Date: July,
1988\n\nThe famous Iris database, first used by Sir R.A. Fisher. The dataset is
taken\nfrom Fisher\'s paper. Note that it\'s the same as in R, but not as in the
UCI\nMachine Learning Repository, which has two wrong data points.\n\nThis is perhaps
the best known database to be found in the\npattern recognition literature. Fisher\'s
paper is a classic in the field and\nis referenced frequently to this day. (See Duda
& Hart, for example.) The\ndata set contains 3 classes of 50 instances each, where
each class refers to a\ntype of iris plant. One class is linearly separable from the
other 2; the\nlatter are NOT linearly separable from each other.\n\n.. topic::
References\n\n - Fisher, R.A. "The use of multiple measurements in taxonomic
problems"\n Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to\n
Mathematical Statistics" (John Wiley, NY, 1950).\n - Duda, R.O., & Hart, P.E. (1973)
Pattern Classification and Scene Analysis.\n (Q327.D83) John Wiley & Sons. ISBN 0-
471-22361-1. See page 218.\n - Dasarathy, B.V. (1980) "Nosing Around the
Neighborhood: A New System\n Structure and Classification Rule for Recognition in
Partially Exposed\n Environments". IEEE Transactions on Pattern Analysis and
Machine\n Intelligence, Vol. PAMI-2, No. 1, 67-71.\n - Gates, G.W. (1972) "The
Reduced Nearest Neighbor Rule". IEEE Transactions\n on Information Theory, May 1972,
431-433.\n - See also: 1988 MLC Proceedings, 54-64. Cheeseman et al"s AUTOCLASS II\n
conceptual clustering system finds 3 classes in the data.\n - Many, many more ...',
'feature_names': ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',
'petal width (cm)'], 'filename': 'iris.csv', 'data_module': 'sklearn.datasets.data'}
```

```python
type(iris)
```
```
sklearn.utils.Bunch
```

```python
iris.keys()
```
```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names',
'filename', 'data_module'])
```

```python
n_samples, n_features = iris.data.shape
print("number of samples:",n_samples)
print("number of features:",n_features)
print(iris.data[0])
number of samples: 150
number of features: 4
[5.1 3.5 1.4 0.2]
```

```
iris.data[[12,26,89,114]]
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]])
```

```
print(iris.data.shape)
print(iris.target.shape)
(150, 4)
(150,)
```

```
print(iris.target)
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

```
import numpy as np
np.bincount(iris.target)
array([50, 50, 50])
```

```
print(iris.target_names)
['setosa' 'versicolor' 'virginica']
```

WINE DATASET:

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

```
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, 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]]), 'target': array([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, 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, 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]), 'frame': None, 'target_names': array(['class_0', 'class_1',
'class_2'], dtype='<U7'), 'DESCR': '.. _wine_dataset:\n\nWine recognition dataset\n--
----------------------\n\n**Data Set Characteristics:**\n\n :Number of Instances: 178
(50 in each of three classes)\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- Flavanoids\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 wines\n \t\t- Proline\n\n -
class:\n - class_0\n - class_1\n - class_2\n\t\t\n :Summary Statistics:\n \n
============================= ==== ===== ======= =====\n Min Max 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 Colour Intensity: 1.3 13.0 5.1 2.3\n Hue: 0.48
1.71 0.96 0.23\n OD280/OD315 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), class_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-
databases/wine/wine.data\n\nThe data is the results of a chemical analysis of wines
grown in the same\nregion in Italy by three different cultivators. There are thirteen
different\nmeasurements taken for different constituents found in the three types
of\nwine.\n\nOriginal Owners: \n\nForina, M. et al, PARVUS - \nAn Extendible Package
for Data Exploration, Classification and Correlation. \nInstitute of Pharmaceutical
and Food Analysis and Technologies,\nVia Brigata Salerno, 16147 Genoa,
Italy.\n\nCitation:\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 Science and Dept. of \n Mathematics and Statistics, James Cook University of
North Queensland. \n (Also submitted to Technometrics). \n\n The data was used with
many others for comparing various \n classifiers. The classes are separable, though
only 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 CLASSIFICATION
PERFORMANCE OF RDA" \n Tech. Rep. no. 92-01, (1992), Dept. of Computer Science and
Dept. of \n Mathematics and Statistics, James Cook University of North Queensland. \n
(Also submitted to Journal of Chemometrics).\n', 'feature_names': ['alcohol',
'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium', 'total_phenols', 'flavanoids',
'nonflavanoid_phenols', 'proanthocyanins', 'color_intensity', 'hue',
'od280/od315_of_diluted_wines', 'proline']}
```

```python
type(wine)
```

```
sklearn.utils.Bunch
```

```python
wine.keys()
```

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

```python
print(wine.target_names)
```

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

```python
n_samples,n_features = wine.data.shape
print("Number of samples:",n_samples)
print("Number of features:",n_features)
print(wine.data[1])
```

```
Number of samples: 178
Number of features: 13
[1.32e+01 1.78e+00 2.14e+00 1.12e+01 1.00e+02 2.65e+00 2.76e+00 2.60e-01
 1.28e+00 4.38e+00 1.05e+00 3.40e+00 1.05e+03]
```

```
wine.data[[15,177,13,45]]
```

```
array([[1.363e+01, 1.810e+00, 2.700e+00, 1.720e+01, 1.120e+02, 2.850e+00, 2.910e+00,
3.000e-01, 1.460e+00, 7.300e+00, 1.280e+00, 2.880e+00, 1.310e+03], [1.413e+01,
4.100e+00, 2.740e+00, 2.450e+01, 9.600e+01, 2.050e+00, 7.600e-01, 5.600e-01,
1.350e+00, 9.200e+00, 6.100e-01, 1.600e+00, 5.600e+02], [1.475e+01, 1.730e+00,
2.390e+00, 1.140e+01, 9.100e+01, 3.100e+00, 3.690e+00, 4.300e-01, 2.810e+00,
5.400e+00, 1.250e+00, 2.730e+00, 1.150e+03], [1.421e+01, 4.040e+00, 2.440e+00,
1.890e+01, 1.110e+02, 2.850e+00, 2.650e+00, 3.000e-01, 1.250e+00, 5.240e+00, 8.700e-
01, 3.330e+00, 1.080e+03]])
```

```
print(wine.data.shape)
print(wine.target.shape)
```

```
(178, 13)
(178,)
```

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

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

```
print(wine.target_names)
```

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

```
print(wine.target)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 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 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]
```

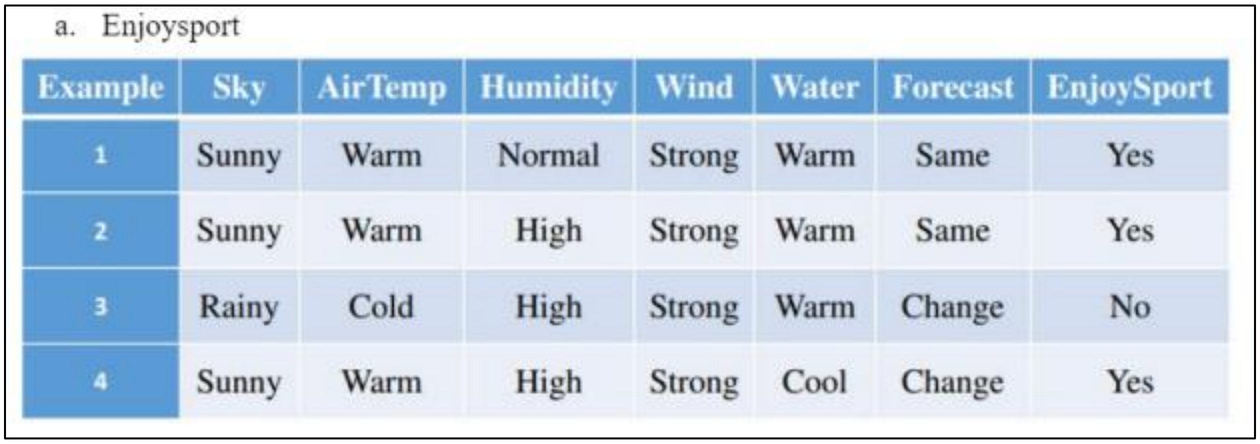
```
print(wine.feature_names)
```

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

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

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

Dataset:



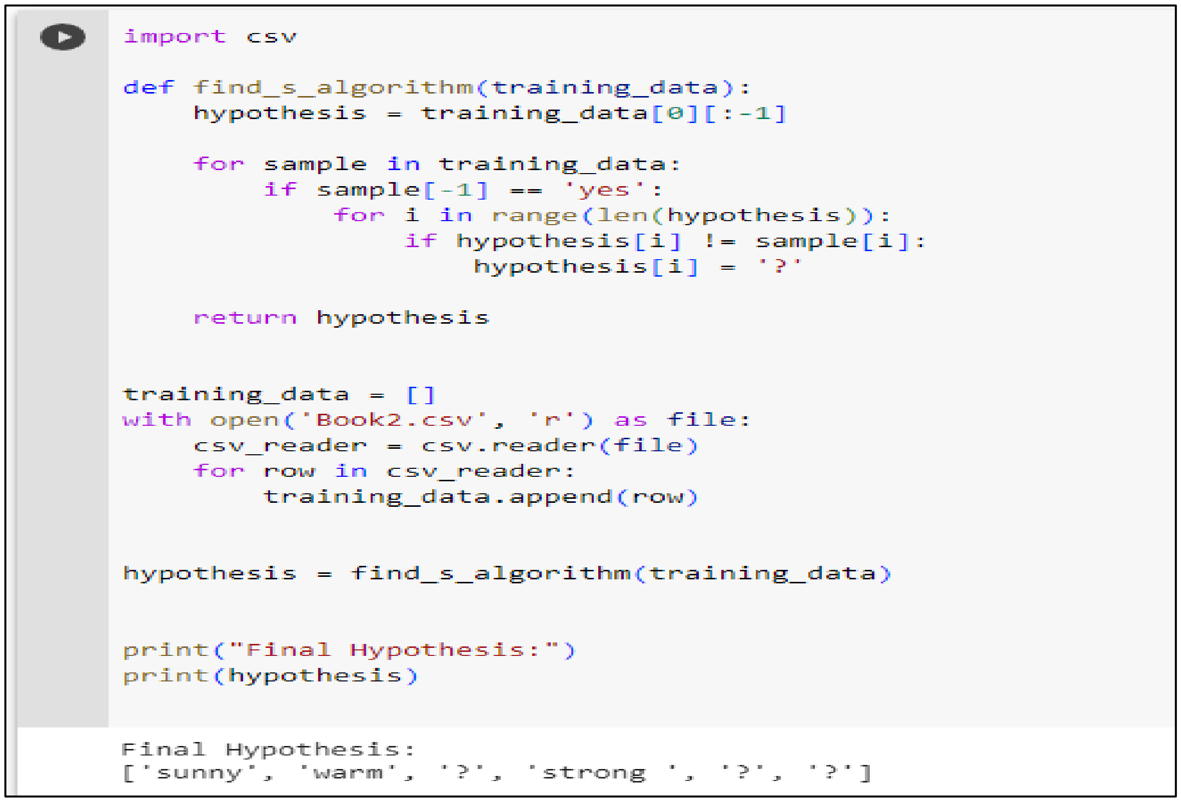
a. Enjoysport

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:

1. Initialize the hypothesis with the attribute values from the first positive training sample.
2. For each subsequent positive training sample:
    - Compare each attribute value in the hypothesis with the corresponding attribute value in the sample.
    - If the attribute values differ, update the hypothesis attribute value to **?**.
3. Return the final hypothesis.

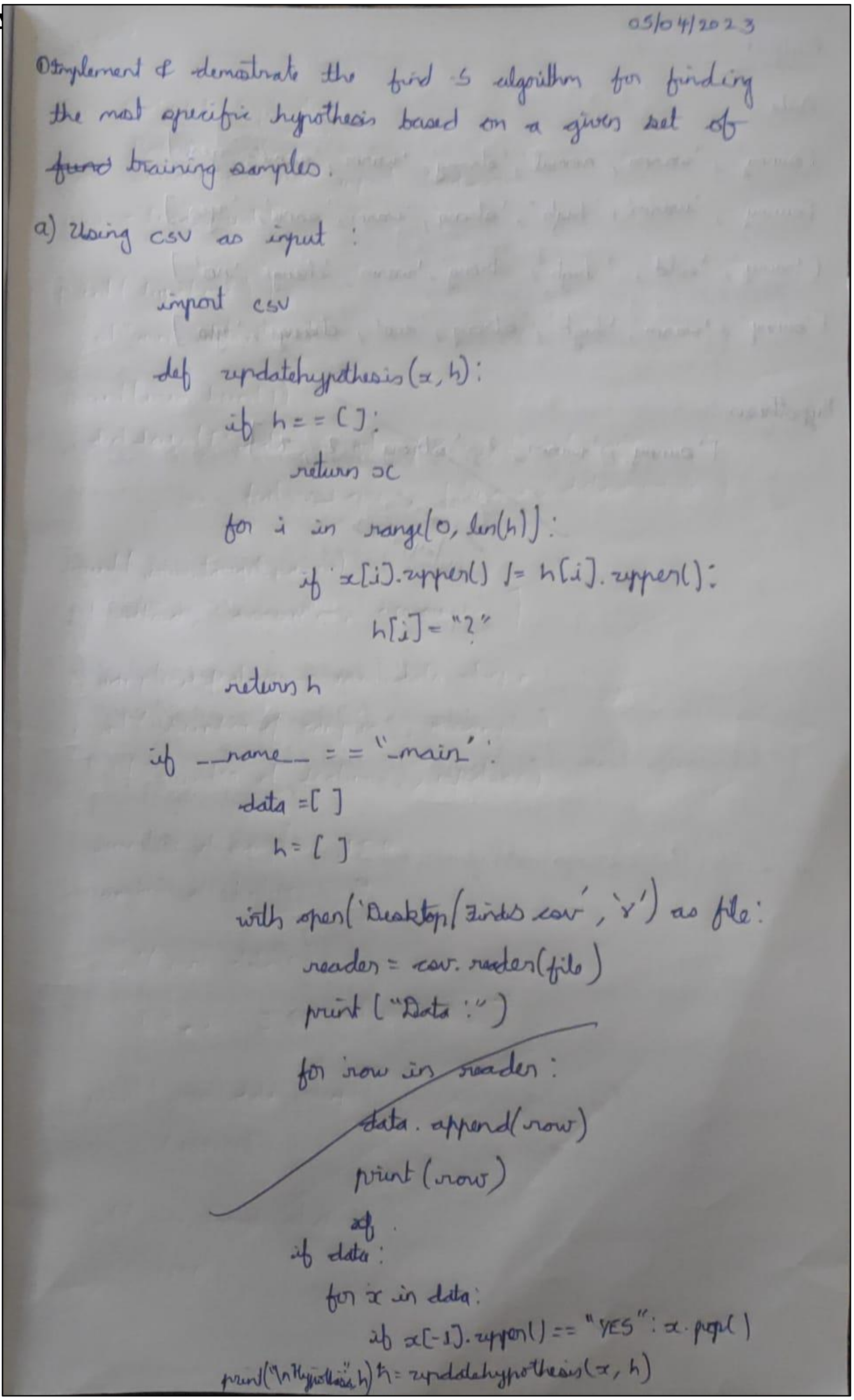
Code:



```python
import csv

def find_s_algorithm(training_data):
    hypothesis = training_data[0][:-1]

    for sample in training_data:
        if sample[-1] == 'yes':
            for i in range(len(hypothesis)):
                if hypothesis[i] != sample[i]:
                    hypothesis[i] = '?'

    return hypothesis


training_data = []
with open('Book2.csv', 'r') as file:
    csv_reader = csv.reader(file)
    for row in csv_reader:
        training_data.append(row)


hypothesis = find_s_algorithm(training_data)


print("Final Hypothesis:")
print(hypothesis)
```

```
Final Hypothesis:
['sunny', 'warm', '?', 'strong ', '?', '?']
```

Obser



05/04/2023

① Implement & demonstrate the find-S algorithm for finding the most specific hypothesis based on a given set of ~~fund~~ training samples.

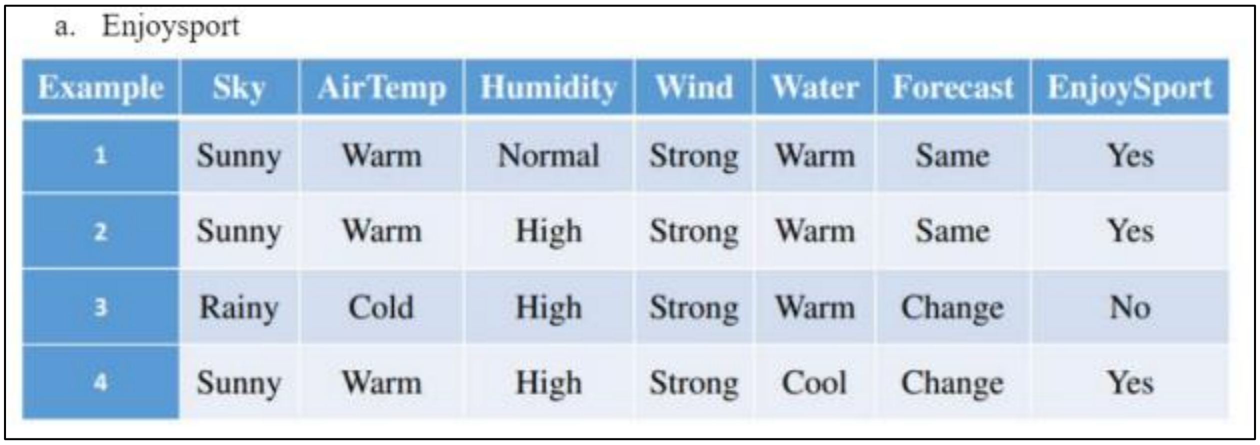
a) Using csv as input :

```
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 :")
        for row in reader:
            data.append(row)
            print (row)
    if data:
        for x in data:
            if x[-1].upper() == "YES": x.pop( )
print("\nHypothesis:", h) h = updatehypothesis(x, h)
```

DATE: 15/04/2023
LAB 3: CANDIDATE- ELIMINATION

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

Dataset:



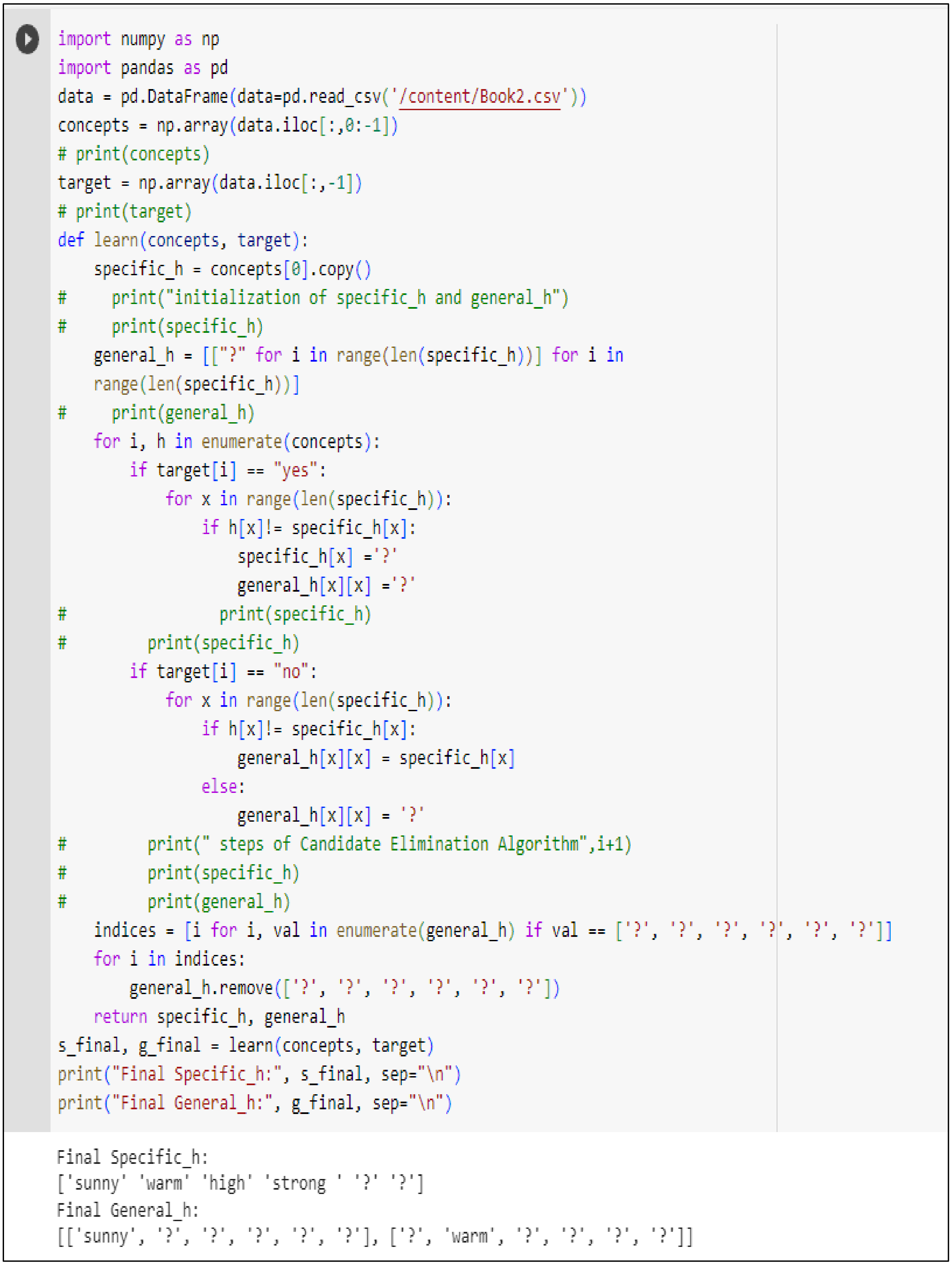
a. Enjoysport

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:

1. Initialize G to the set of maximally general hypotheses in H.
2. Initialize S to the set of maximally specific hypotheses in H.
3. For each training example d:
    - If d is a positive example:
        - Remove from G any hypothesis inconsistent with d.
        - For each hypothesis s in S that is not consistent with d:
            - Remove s from S.
            - Add to S all minimal generalizations h of s that are consistent with d and some member of G is more general than h.
            - Remove from S any hypothesis that is more general than another hypothesis in S.
    - If d is a negative example:
        - Remove from S any hypothesis inconsistent with d.
        - For each hypothesis g in G that is not consistent with d:
            - Remove g from G.
            - Add to G all minimal specializations h of g that are consistent with d and some member of S is more specific than h.
            - Remove from G any hypothesis that is less general than another hypothesis in G.

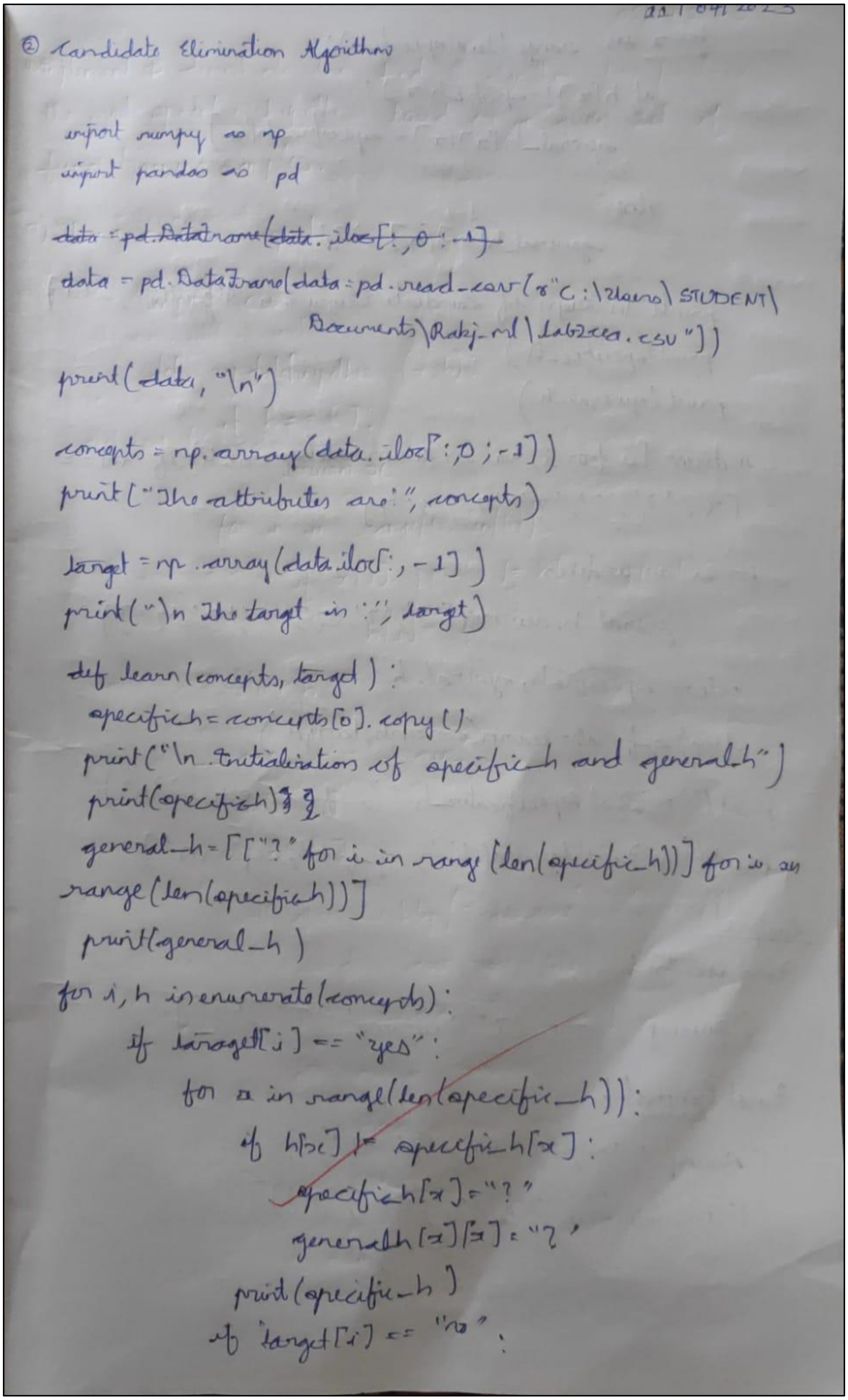
Code:



```
import numpy as np
import pandas as pd
data = pd.DataFrame(data=pd.read_csv('/content/Book2.csv'))
concepts = np.array(data.iloc[:,0:-1])
# print(concepts)
target = np.array(data.iloc[:,-1])
# print(target)
def learn(concepts, target):
    specific_h = concepts[0].copy()
#     print("initialization of specific_h and general_h")
#     print(specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in
    range(len(specific_h))]
#     print(general_h)
    for i, h in enumerate(concepts):
        if target[i] == "yes":
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    specific_h[x] ='?'
                    general_h[x][x] ='?'
#                 print(specific_h)
#         print(specific_h)
        if target[i] == "no":
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'
#         print(" steps of Candidate Elimination Algorithm",i+1)
#         print(specific_h)
#         print(general_h)
    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h
s_final, g_final = learn(concepts, target)
print("Final Specific_h:", s_final, sep="\n")
print("Final General_h:", g_final, sep="\n")
```

```
Final Specific_h:
['sunny' 'warm' 'high' 'strong ' '?' '?']
Final General_h:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

Observation:



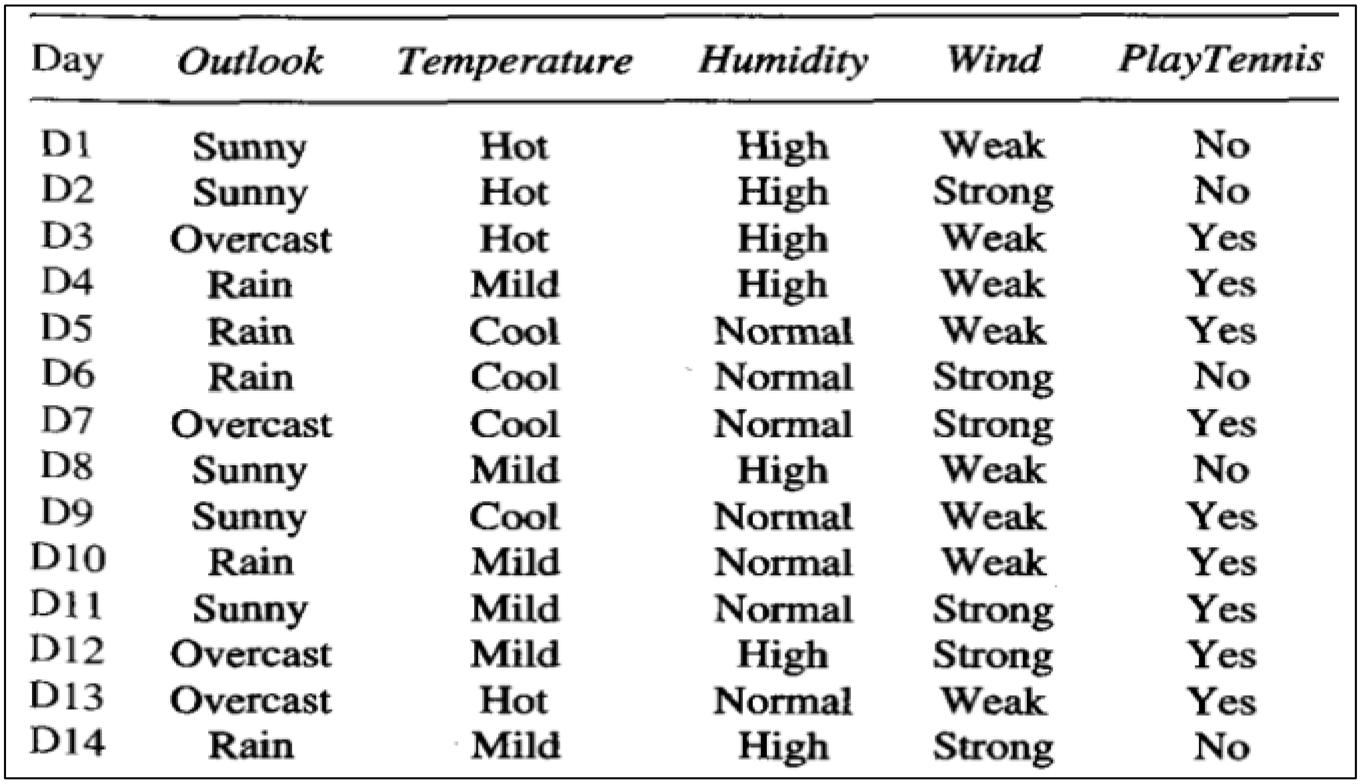
21/04/2023

② Candidate Elimination Algorithm

```
import numpy as np
import pandas as pd

~~data = pd.DataFrame(data.iloc[:, 0 : -1]~~
data = pd.DataFrame(data=pd.read_csv(r"C:\Users\STUDENT\
                    Documents\Rakj-ml\lab2cea.csv"))
print(data, "\n")

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

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

def learn(concepts, target ):
  specific_h = concepts[0].copy()
  print("\n Initialization of specific_h and general_h")
  print(specific_h)
  general_h = [["?" for i in range (len(specific_h))] for i in
range (len(specific_h))]
  print(general_h )
for i, h in enumerate(concepts):
     if target[i] == "yes":
          for x in range(len(specific_h)):
               if h[x] != specific_h[x]:
                    specific_h[x] = "?"
                    general_h[x][x] = "?"
          print(specific_h )
      if target[i] == "no":
```

DATE: 03/05/2023
LAB 4: ID3 ALGORITHM

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:



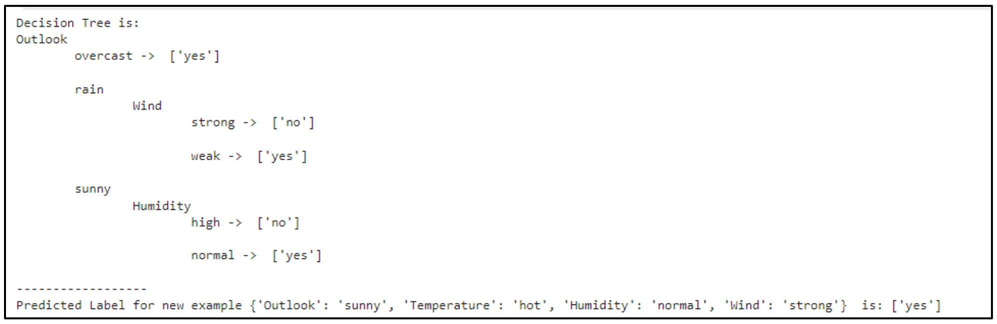
Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool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

1. Create a root node for the decision tree.
2. If all examples belong to the same class, return a leaf node with that class label.
3. If there are no more attributes to consider, return a leaf node with the majority class label of the examples.
4. Select the attribute that best classifies the examples using the information gain or another criterion.
5. Create a decision node for the selected attribute.
6. For each possible value of the selected attribute:
   - Create a new branch below the decision node.
   - Filter the examples that have the selected attribute value.
   - If the filtered examples are empty, add a leaf node with the majority class label of the examples.
   - Otherwise, recursively apply the ID3 algorithm to the filtered examples using the remaining attributes.
7. Return the root node of the decision tree.

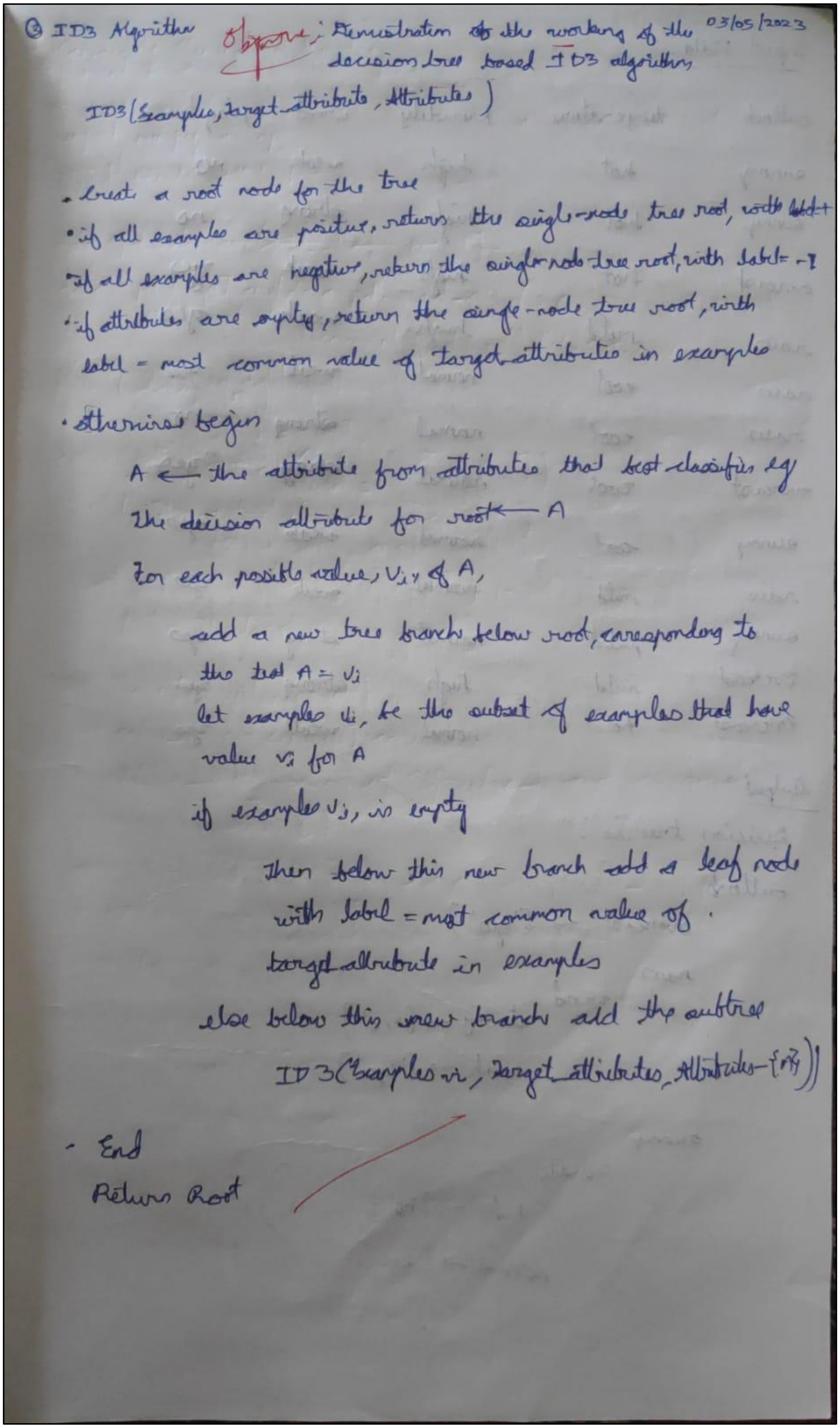
Code:

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

data = pd.read_csv("/kaggle/input/id3hhhh/id3.csv")
features = [feat for feat in data]
features.remove("Answer")

class Node:
    def __init__(self):
        self.children = []
        self.value = ""
        self.isLeaf = False
        self.pred = ""


def entropy(examples):
    pos = 0.0
    neg = 0.0
    for _, row in examples.iterrows():
        if row["Answer"] == "yes":
            pos += 1
        else:
            neg += 1
    if pos == 0.0 or neg == 0.0:
        return 0.0
    else:
        p = pos / (pos + neg)
        n = neg / (pos + neg)
        return -(p * math.log(p, 2) + n * math.log(n, 2))

def info_gain(examples, attr):
    uniq = np.unique(examples[attr])
    #print ("\n",uniq)
    gain = entropy(examples)
    #print ("\n",gain)
    for u in uniq:
        subdata = examples[examples[attr] == u]
        #print ("\n",subdata)
        sub_e = entropy(subdata)
        gain -= (float(len(subdata)) / float(len(examples))) * sub_e
        #print ("\n",gain)
    return gain

def ID3(examples, attrs):
    root = Node()

    max_gain = 0
    max_feat = ""
    for feature in attrs:
        #print ("\n",examples)
        gain = info_gain(examples, feature)
        if gain > max_gain:
            max_gain = gain
            max_feat = feature
    root.value = max_feat
    #print ("\nMax feature attr",max_feat)
    uniq = np.unique(examples[max_feat])
    #print ("\n",uniq)
    for u in uniq:
        #print ("\n",u)
        subdata = examples[examples[max_feat] == u]
        #print ("\n",subdata)
        if entropy(subdata) == 0.0:
            newNode = Node()
            newNode.isLeaf = True
            newNode.value = u
            newNode.pred = np.unique(subdata["Answer"])
            root.children.append(newNode)
        else:
            dummyNode = Node()
            dummyNode.value = u
            new_attrs = attrs.copy()
            new_attrs.remove(max_feat)
            child = ID3(subdata, new_attrs)
            dummyNode.children.append(child)
            root.children.append(dummyNode)

    return root

def printTree(root: Node, depth=0):
    for i in range(depth):
        print("\t", end="")
    print(root.value, end="")
    if root.isLeaf:
        print(" -> ", root.pred)
    print()
    for child in root.children:
        printTree(child, depth + 1)

def classify(root: Node, new):
    for child in root.children:
        if child.value == new[root.value]:
            if child.isLeaf:
                print ("Predicted Label for new example", new," is:", child.pred)
                exit
            else:
                classify (child.children[0], new)

root = ID3(data, features)
print("Decision Tree is:")
printTree(root)
print ("------------------")

new = {"Outlook":"sunny", "Temperature":"hot", "Humidity":"normal", "Wind":"strong"}
classify (root, new)
```



```
Decision Tree is:
Outlook
	overcast ->  ['yes']

	rain
		Wind
			strong ->  ['no']

			weak ->  ['yes']

	sunny
		Humidity
			high ->  ['no']

			normal ->  ['yes']

------------------
Predicted Label for new example {'Outlook': 'sunny', 'Temperature': 'hot', 'Humidity': 'normal', 'Wind': 'strong'}  is: ['yes']
```

Observation:



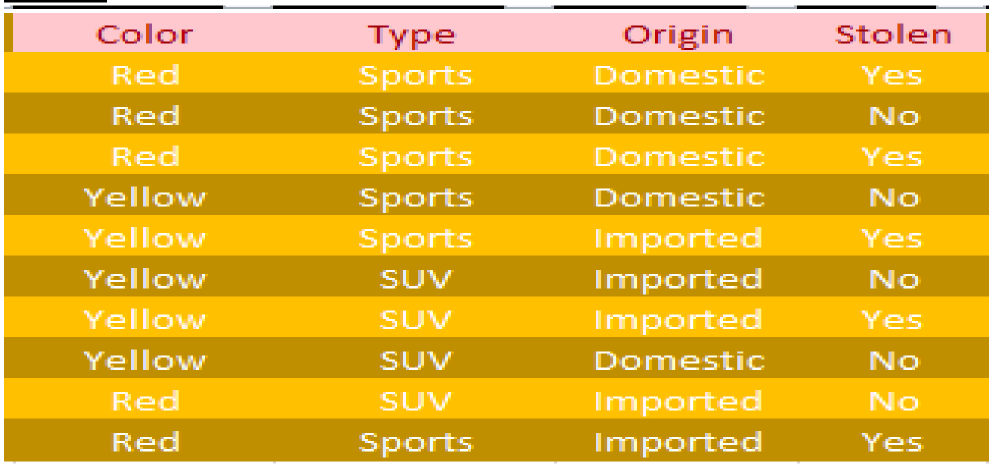
③ ID3 Algorithm Objective; Demonstration of the working of the decision tree based ID3 algorithm 03/05/2023

```
ID3(Examples, target_attribute, Attributes)

- 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 are empty, return the single-node tree root, with
  label = most common value of target_attributes in examples
• otherwise begin
    A ← the attribute from attributes that best classifies eg
    The decision attribute for root ← A
    For each possible value, vi, of A,
        add a new tree branch below root, corresponding to
        the test A = vi
        let examples vi, be the subset of examples that have
        value vi 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, Target_attributes, Attributes-{A})
- End
Return Root
```

DATE: 17/05/2023
LAB 5: BAYESIAN CLASSIFIER

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:



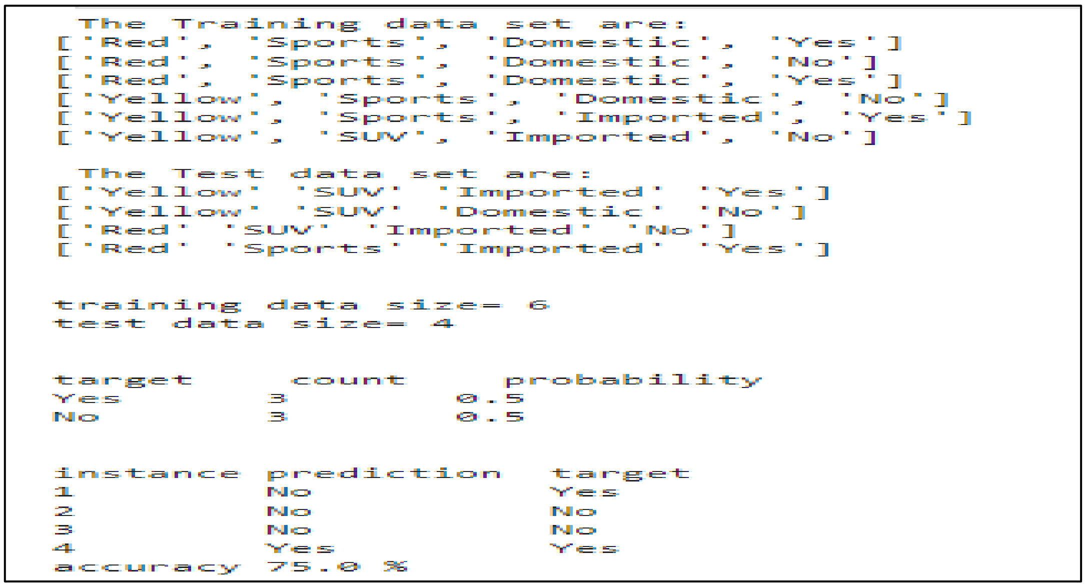
Color	Type	Origin	Stolen
Red	Sports	Domestic	Yes
Red	Sports	Domestic	No
Red	Sports	Domestic	Yes
Yellow	Sports	Domestic	No
Yellow	Sports	Imported	Yes
Yellow	SUV	Imported	No
Yellow	SUV	Imported	Yes
Yellow	SUV	Domestic	No
Red	SUV	Imported	No
Red	Sports	Imported	Yes

Algorithm:

1. Collect all words, punctuation, and other tokens that occur in the training examples. This forms the vocabulary, which is the set of all distinct words and tokens present in any document in the training examples.
2. Calculate the required probability terms:
   - For each target value vj in the set of target values V:
     - Select the subset of documents docs_j from the training examples for which the target value is vj.
     - Calculate the prior probability P(vj) as the number of documents in docs_j divided by the total number of training examples.
     - Create a text document Text_j by concatenating all the documents in docs_j.
     - Calculate the total number of distinct word positions n in Text_j.
     - For each word wk in the vocabulary:
       - Count the number of times word wk occurs in Text_j and store it as nk.
       - Calculate the conditional probability P(wk|vj) as (nk + 1) / (n + |Vocabulary|), where |Vocabulary| is the total number of distinct words in the vocabulary.
3. To classify a new document Doc:
   - Identify the positions in Doc that contain tokens found in the vocabulary. These are the relevant word positions.
   - For each target value vj in the set of target values:
     - Calculate the posterior probability P(vj|Doc) using the formula: P(vj|Doc) = P(vj) * ∏(P(ai|vj) for ai in relevant word positions)
   - Return the estimated target value for the document Doc as VNB, where VNB is the value of vj that maximizes P(vj|Doc).

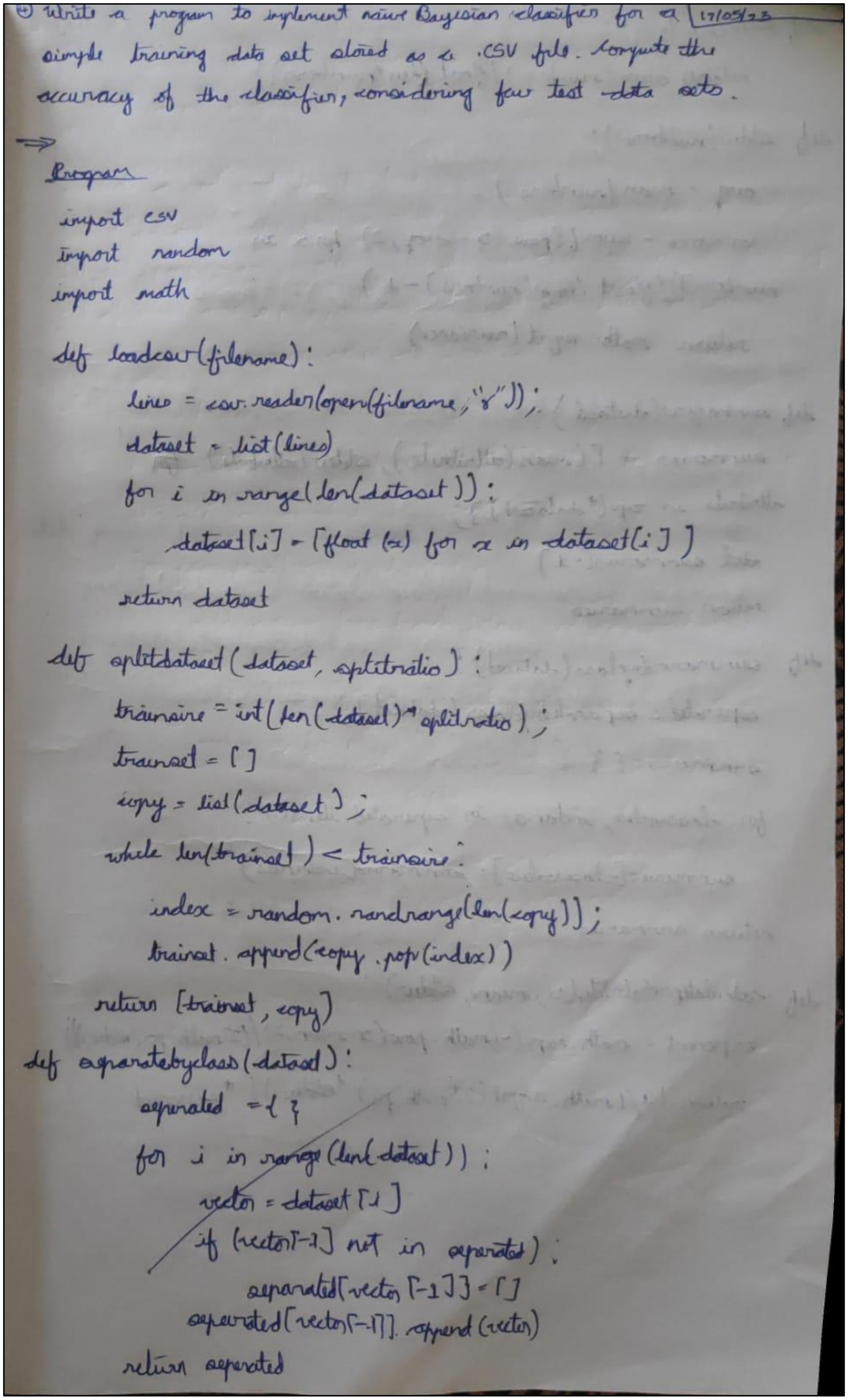
Code:

```
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):

    total_size = data.shape[0]
    print("\n")
    print("training data size=",total_size)
    print("test data size=",test.shape[0])

    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")
    for t in range(test.shape[0]):
        for k in range (test.shape[1]-1):
            count1=count0=0
            for j in range (data.shape[0]):
                #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("naive.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)
```



```
 The Training data set are:
['Red', 'Sports', 'Domestic', 'Yes']
['Red', 'Sports', 'Domestic', 'No']
['Red', 'Sports', 'Domestic', 'Yes']
['Yellow', 'Sports', 'Domestic', 'No']
['Yellow', 'Sports', 'Imported', 'Yes']
['Yellow', 'SUV', 'Imported', 'No']

 The Test data set are:
['Yellow' 'SUV' 'Imported' 'Yes']
['Yellow' 'SUV' 'Domestic' 'No']
['Red' 'SUV' 'Imported' 'No']
['Red' 'Sports' 'Imported' 'Yes']


training data size= 6
test data size= 4


target      count     probability
Yes         3        0.5
No          3        0.5


instance prediction   target
1        No           Yes
2        No           No
3        No           No
4        Yes          Yes
accuracy 75.0 %
```

Observation:



⊕ Write a program to implement naive Bayesian classifier for a simple training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets. 17/05/23

⟹

Program

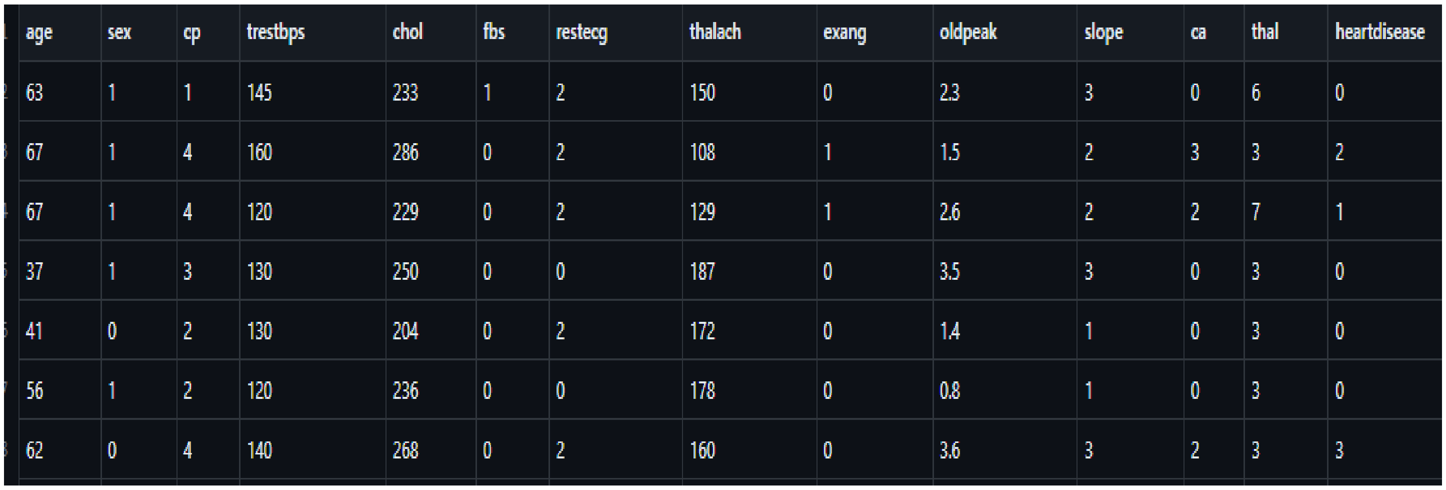
```
import csv
import random
import math

def loadcsv(filename):
        lines = csv.reader(open(filename, "r"));
        dataset = list(lines)
        for i in range(len(dataset)):
            dataset[i] = [float (x) for x in dataset[i] ]
        return dataset

def splitdataset (dataset, splitratio) :
        trainsize = int (len (dataset) * splitratio);
        trainset = [ ]
        copy = list (dataset );
        while len(trainset) < trainsize:
            index = random. randrange(len(copy));
            trainset . append(copy . pop (index) )
      return [trainset , copy]
def separatebyclass (dataset ):
        separated = { }
        for i in range (len(dataset)) :
            vector = dataset [i ]
            if (vector[-1] not in separated) :
                separated[vector [-1] ] = [ ]
            separated[vector[-1]]. append (vector)
        return separated
```

```
def mean(numbers):
        return sum(numbers)/float(len(numbers))

def stdev(numbers):
        avg = mean(numbers)
        variance - sum([pow(x-avg, 2) for x in
    numbers])/float(leng(numbers)-1)
        return math.sqrt(variance)

def summarize(dataset):
    summaries = [(mean(attribute), stdev(attribute) for
  attribute in zip(*dataset)];
    del summaries[-1]
    return summaries

def summarizebyclass(dataset):
    seperated = separatebyclass(dataset);
    summaries = { }
    for classvalue, instances in seperated.items():
        summaries[classvalue] = summarize(instances)
    return summaries

def calculateprobability(x, mean, stdev):
    exponent = math.exp(-(math.pow(x-mean, 2)/(2*math.pow(stdev,2))))
    return (1/(math.sqrt(2*math.pi)*stdev)) *exponent
def calculateclassprobabilities (summaries, inputvector):
    probabilities = { }
    for classvalue, classsummaries in summaries.items ():
        probabilities [classvalue] = 1
        for i in range (len (classsummaries)):
            mean, stdev = classsummaries[i]
            x = inputvector [i]
            probabilities [classvalue] *= calculateprobability (x, mean, stdev)
        return probabilities

def predict (summaries, inputvector):
    probabilities = calculateclassprobabilities(summaries, inputvector)
    bestlabel, bestprob = None, -1
    for classvalue, probability in probabilities.items ( ):
        if bestlabel is None or probability > bestprob :
            bestprob = probability
            bestlabel = classvalue
    return bestlabel

def getpredictions (summaries, testset) :
    predictions = [ ]
    for i in range(len(testset)):
        result = predict [summaries, testset [i]])
        prediction . append(result)
    return predictions.
```

DATE: 24/05/2023
LAB 6: BAYESIAN NETWORK

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

Dataset:



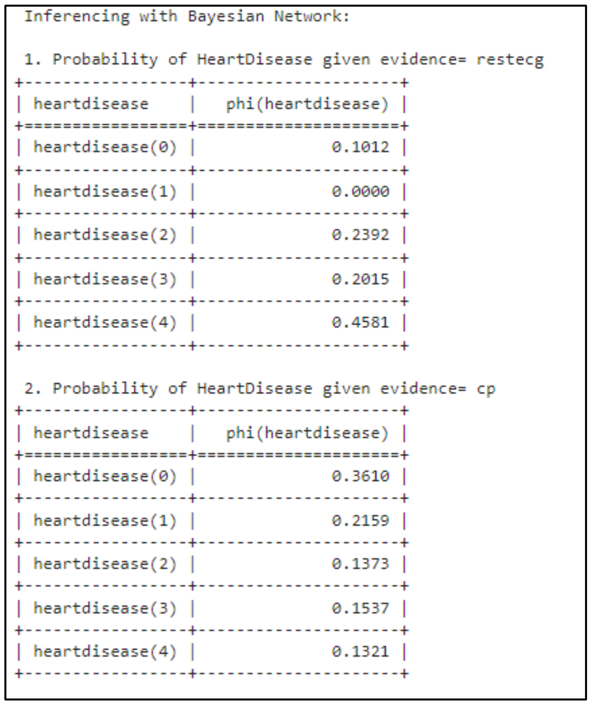
age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	heartdisease
63	1	1	145	233	1	2	150	0	2.3	3	0	6	0
67	1	4	160	286	0	2	108	1	1.5	2	3	3	2
67	1	4	120	229	0	2	129	1	2.6	2	2	7	1
37	1	3	130	250	0	0	187	0	3.5	3	0	3	0
41	0	2	130	204	0	2	172	0	1.4	1	0	3	0
56	1	2	120	236	0	0	178	0	0.8	1	0	3	0
62	0	4	140	268	0	2	160	0	3.6	3	2	3	3

Algorithm:

1. Define the Bayesian network structure: Specify the variables and their dependencies by defining the directed acyclic graph (DAG) structure of the Bayesian network.
2. Assign probability distributions: Assign probability distributions to each variable in the network based on prior knowledge or data. This involves specifying the conditional probability tables (CPTs) for each variable given its parents in the DAG.
3. Query and evidence variables: Identify the variables of interest for inference and set any observed evidence variables to their observed values.
4. Variable elimination:
   - Order the variables in a way that respects the network structure and ensures that parents are eliminated before their children.
   - For each variable in the elimination order, eliminate the variable by summing out or maximizing over its possible values.
   - Update the probability distributions of the remaining variables based on the eliminated variables and the evidence.
5. Perform inference: Calculate the desired probabilities or make predictions based on the updated probability distributions.

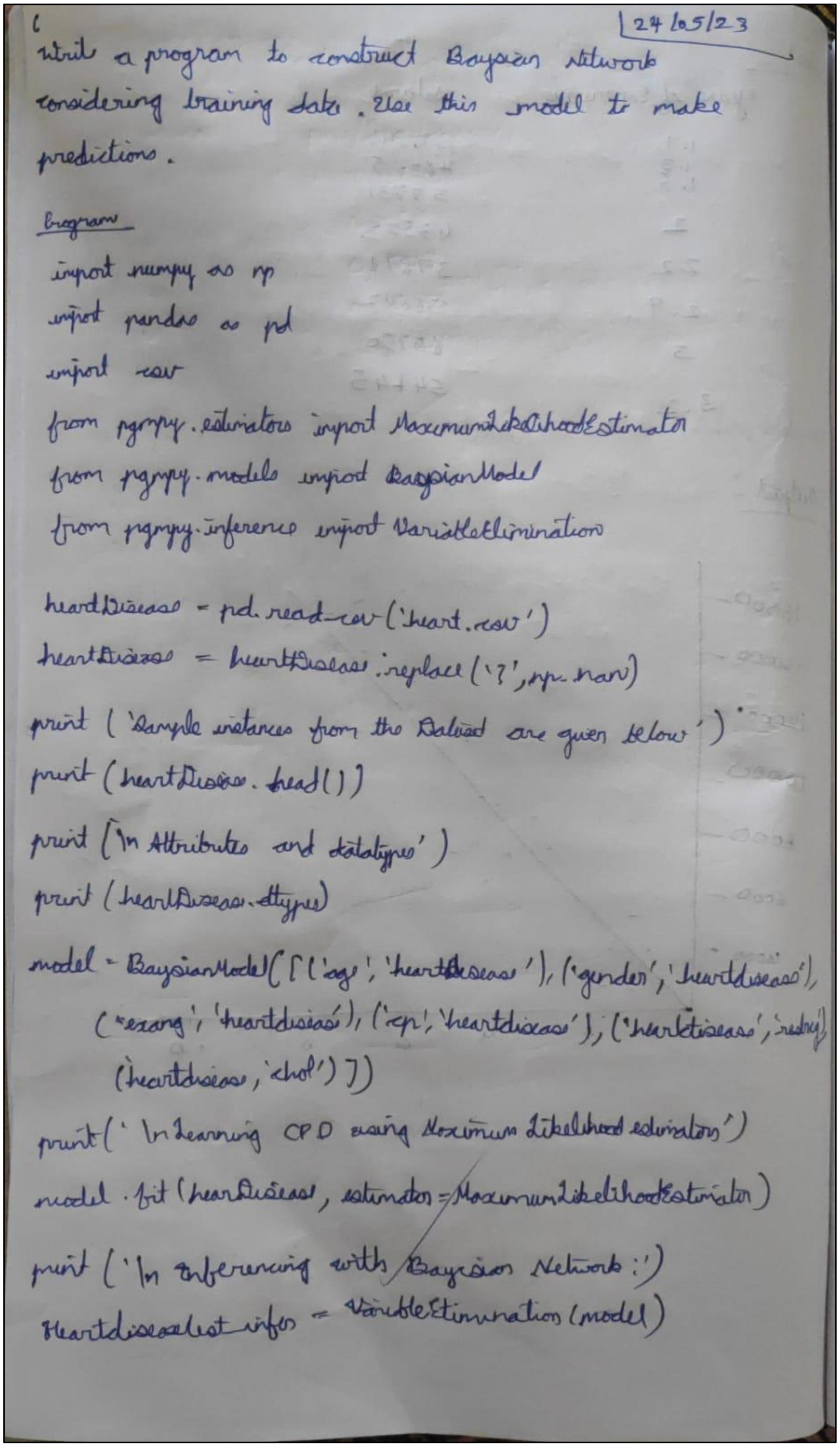
Code:

```
import numpy as np
import pandas as pd
import csv
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel
from pgmpy.inference import VariableElimination

heartDisease = pd.read_csv('/content/sample_data/heart.csv')
heartDisease = heartDisease.replace('?',np.nan)

print('Sample instances from the dataset are given below')
print(heartDisease.head())
print('\n Attributes and datatypes')
print(heartDisease.dtypes)

model=
BayesianModel([('age','heartdisease'),('sex','heartdisease'),('exang','heartdisease'),('cp','heartdisease'),('heartdisea
se','restecg'),('heartdisease','chol')])
print('\nLearning CPD using Maximum likelihood estimators')
model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)

print('\n Inferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)

print('\n 1. Probability of HeartDisease given evidence= restecg')
q1=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'restecg':1})
print(q1)

print('\n 2. Probability of HeartDisease given evidence= cp ')
q2=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'cp':2})
print(q2)
```



```
 Inferencing with Bayesian Network:

 1. Probability of HeartDisease given evidence= restecg
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.1012 |
+-----------------+---------------------+
| heartdisease(1) |              0.0000 |
+-----------------+---------------------+
| heartdisease(2) |              0.2392 |
+-----------------+---------------------+
| heartdisease(3) |              0.2015 |
+-----------------+---------------------+
| heartdisease(4) |              0.4581 |
+-----------------+---------------------+

 2. Probability of HeartDisease given evidence= cp
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.3610 |
+-----------------+---------------------+
| heartdisease(1) |              0.2159 |
+-----------------+---------------------+
| heartdisease(2) |              0.1373 |
+-----------------+---------------------+
| heartdisease(3) |              0.1537 |
+-----------------+---------------------+
| heartdisease(4) |              0.1321 |
+-----------------+---------------------+
```

Observation:



6 | 24/05/23

Write a program to construct Baysian Network considering training data. Use this model to make predictions.

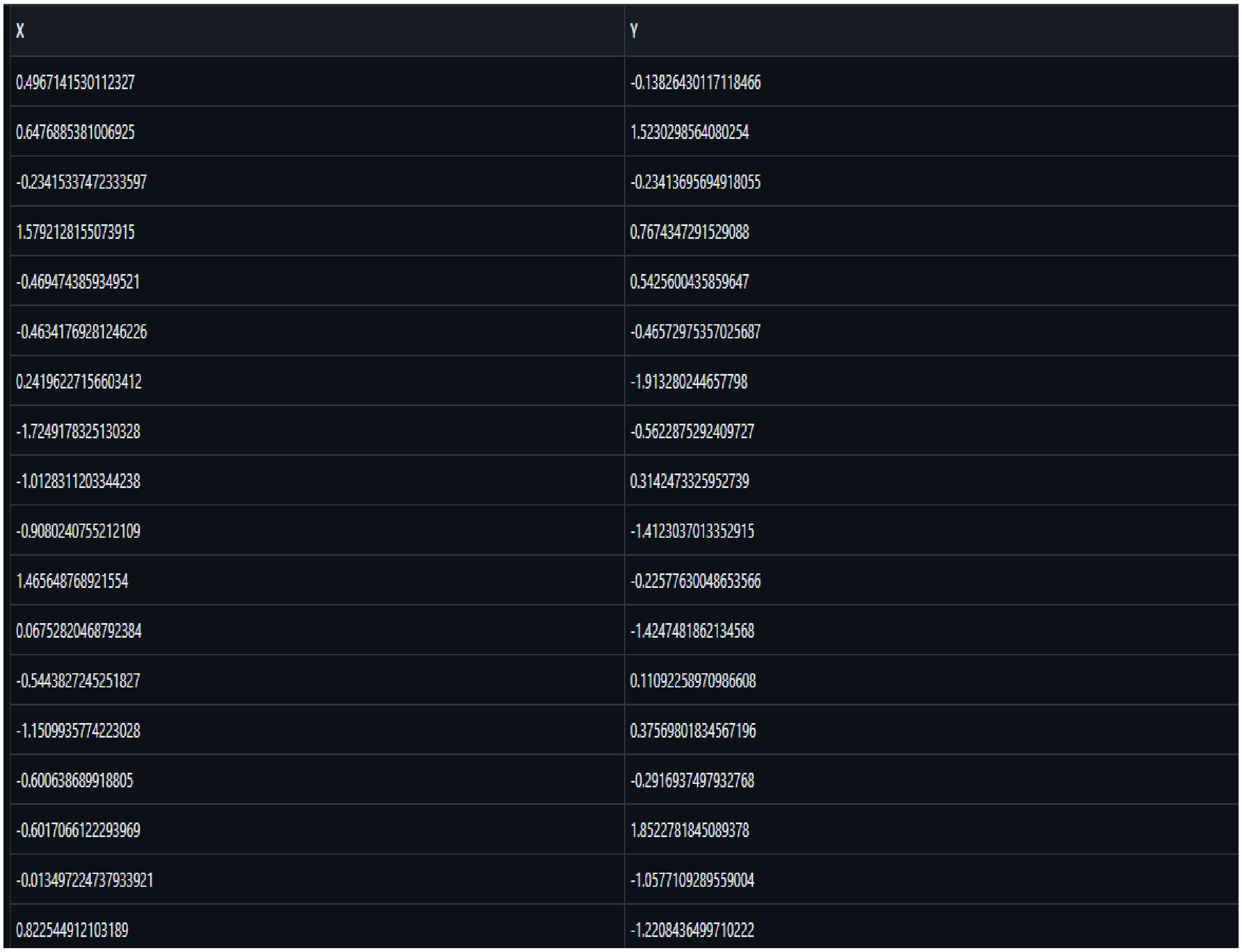
Program

```
import numpy as np
import pandas as pd
import csv
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BaysianModel
from pgmpy.inference import VariableElimination

heartDisease = pd.read_csv('heart.csv')
heartDisease = heartDisease.replace('?',np.nan)
print ('Sample instances from the Dataset are given below')
print (heartDisease.head())
print ('\n Attributes and datatypes')
print (heartDisease.dtypes)
model = BaysianModel([('age','heartdisease'),('gender','heartdisease'),
    ('exang','heartdisease'),('cp','heartdisease'),('heartdisease','restecg'),
    ('heartdisease','chol')])
print('\n Learning CPD using Maximum Likelihood estimators')
model.fit(heartDisease, estimator=MaximumLikelihoodEstimator)

print ('\n Inferencing with Baysian Network:')
Heartdiseasetest_infer = VariableElimination(model)
```

DATE: 07/06/2023
LAB 7: k-MEANS

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

Dataset:



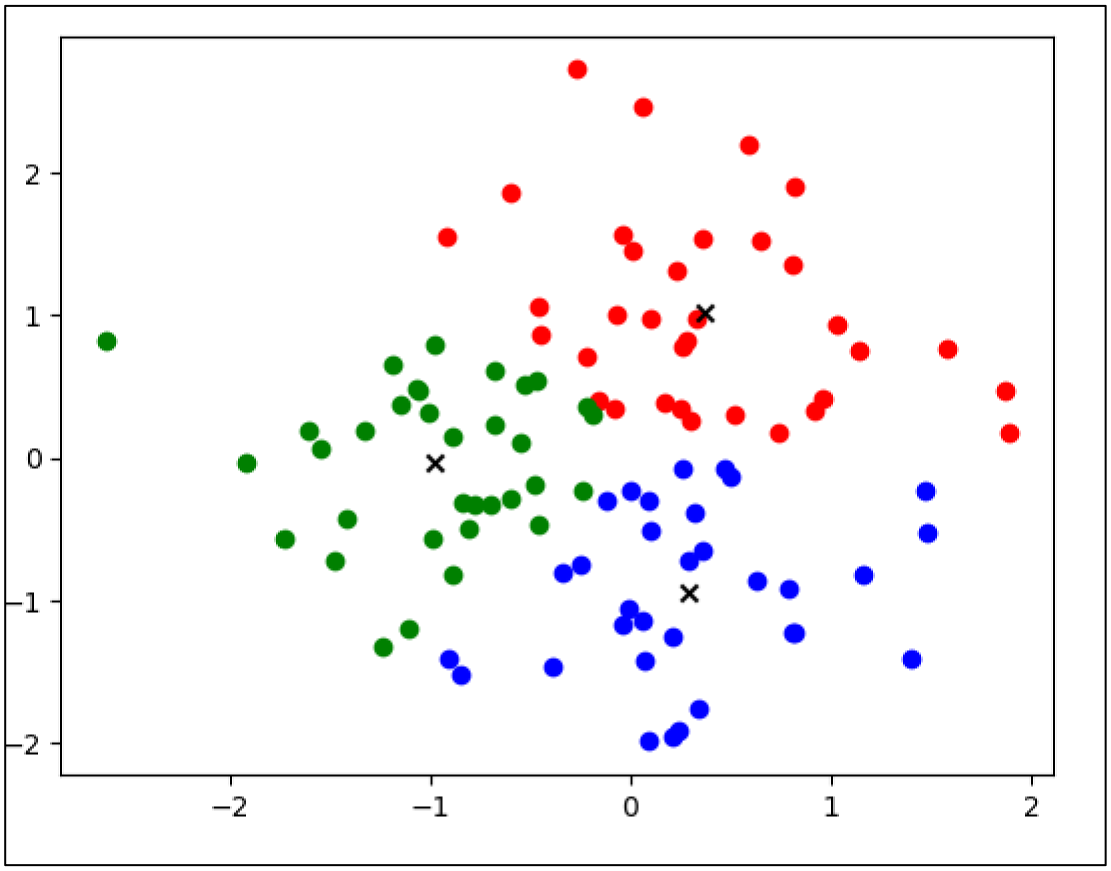
X	Y
0.4967141530112327	-0.13826430117118466
0.6476885381006925	1.5230298564080254
-0.23415337472333597	-0.23413695694918055
1.5792128155073915	0.7674347291529088
-0.4694743859349521	0.5425600435859647
-0.46341769281246226	-0.46572975357025687
0.24196227156603412	-1.913280244657798
-1.7249178325130328	-0.5622875292409727
-1.0128311203344238	0.3142473325952739
-0.9080240755212109	-1.4123037013352915
1.465648768921554	-0.22577630048653566
0.06752820468792384	-1.4247481862134568
-0.5443827245251827	0.11092258970986608
-1.1509935774223028	0.37569801834567196
-0.600638689918805	-0.2916937497932768
-0.6017066122293969	1.8522781845089378
-0.013497224737933921	-1.0577109289559004
0.822544912103189	-1.2208436499710222

Algorithm:

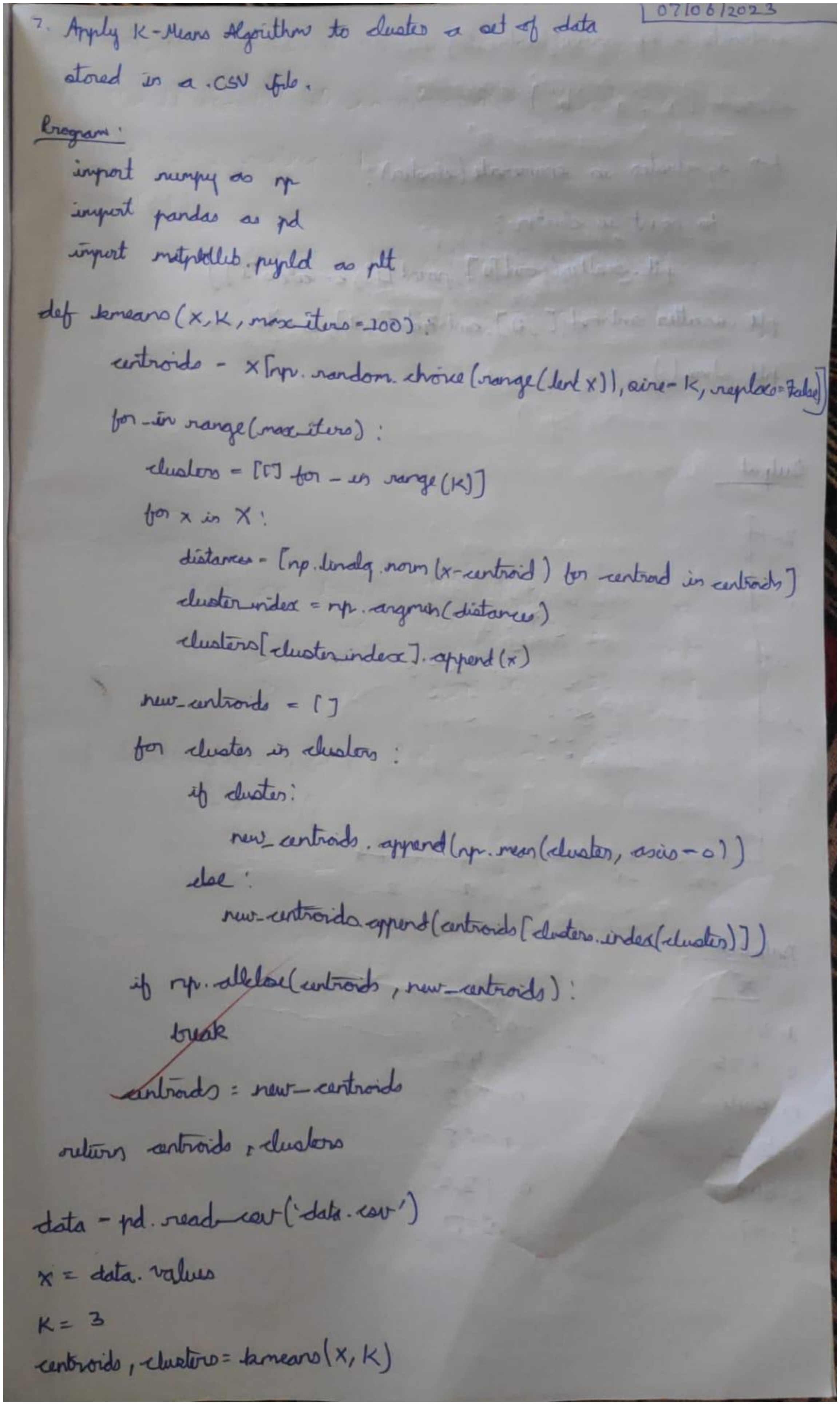
1. Initialize: Randomly select K data points from the dataset as initial cluster centroids.
2. Assign data points to clusters: For each data point, calculate its distance to each centroid and assign it to the cluster with the nearest centroid.
3. Update cluster centroids: Recalculate the centroids of each cluster by taking the mean of the data points assigned to that cluster.
4. Repeat steps 2 and 3 until convergence: Iterate steps 2 and 3 until the cluster assignments no longer change significantly or a maximum number of iterations is reached.
5. Output: Return the final cluster assignments and centroids.

Code:

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

def kmeans(X, K, max_iters=100):
    # Randomly initialize centroids
    centroids = X[np.random.choice(range(len(X)), size=K, replace=False)]

    for _ in range(max_iters):
        # Assign each data point to the nearest centroid
        clusters = [[] for _ in range(K)]
        for x in X:
            distances = [np.linalg.norm(x - centroid) for centroid in centroids]
            cluster_index = np.argmin(distances)
            clusters[cluster_index].append(x)

        # Update centroids
        new_centroids = []
        for cluster in clusters:
            if cluster:
                new_centroids.append(np.mean(cluster, axis=0))
            else:
                # If a centroid has no assigned points, keep the previous centroid value
                new_centroids.append(centroids[clusters.index(cluster)])

        # Check for convergence
        if np.allclose(centroids, new_centroids):
            break

        centroids = new_centroids

    return centroids, clusters

# Load data from CSV file
data = pd.read_csv('/kaggle/working/data.csv')

# Convert data to numpy array
X = data.values

# Perform k-means clustering
K = 3
centroids, clusters = kmeans(X, K)

# Convert centroids list to numpy array
centroids = np.array(centroids)

# Plot the clusters and centroids
colors = ['r', 'g', 'b']
for i, cluster in enumerate(clusters):
    for point in cluster:
        plt.scatter(point[0], point[1], c=colors[i])
plt.scatter(centroids[:, 0], centroids[:, 1], c='k', marker='x')
plt.show()
```


2
1
0
−1
−2
−2
−1
0
1
2

Observation:



07/06/2023

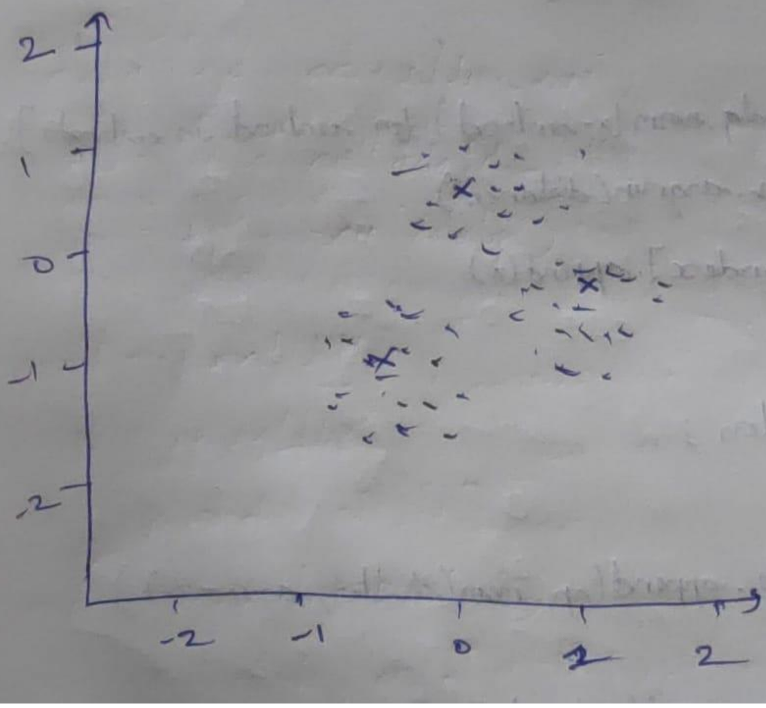
7. Apply K-Means Algorithm to clusters a set of data stored in a .CSV file.

Program:

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

def kmeans(X, K, max_iters=100):
    centroids = X[np.random.choice(range(len(X)), size=K, replace=False)]
    for _ in range(max_iters):
        clusters = [[] for _ in range(K)]
        for x in X:
            distances = [np.linalg.norm(x-centroid) for centroid in centroids]
            cluster_index = np.argmin(distances)
            clusters[cluster_index].append(x)
        new_centroids = []
        for cluster in clusters:
            if cluster:
                new_centroids.append(np.mean(cluster, axis=0))
            else:
                new_centroids.append(centroids[clusters.index(cluster)])
        if np.allclose(centroids, new_centroids):
            break
        centroids = new_centroids
    return centroids, clusters

data = pd.read_csv('data.csv')
X = data.values
K = 3
centroids, clusters = kmeans(X, K)
```

```
centroids = np.array(centroids)
colors = ['r', 'g', 'b']
for i, cluster in enumerate(cluster):
    for point in cluster:
        plt.scatter(point[0], point[1], c= colors[i])
plt.scatter(centroids[:,0], centroids[:,1], c='k', marker='x')
plt.show()
```

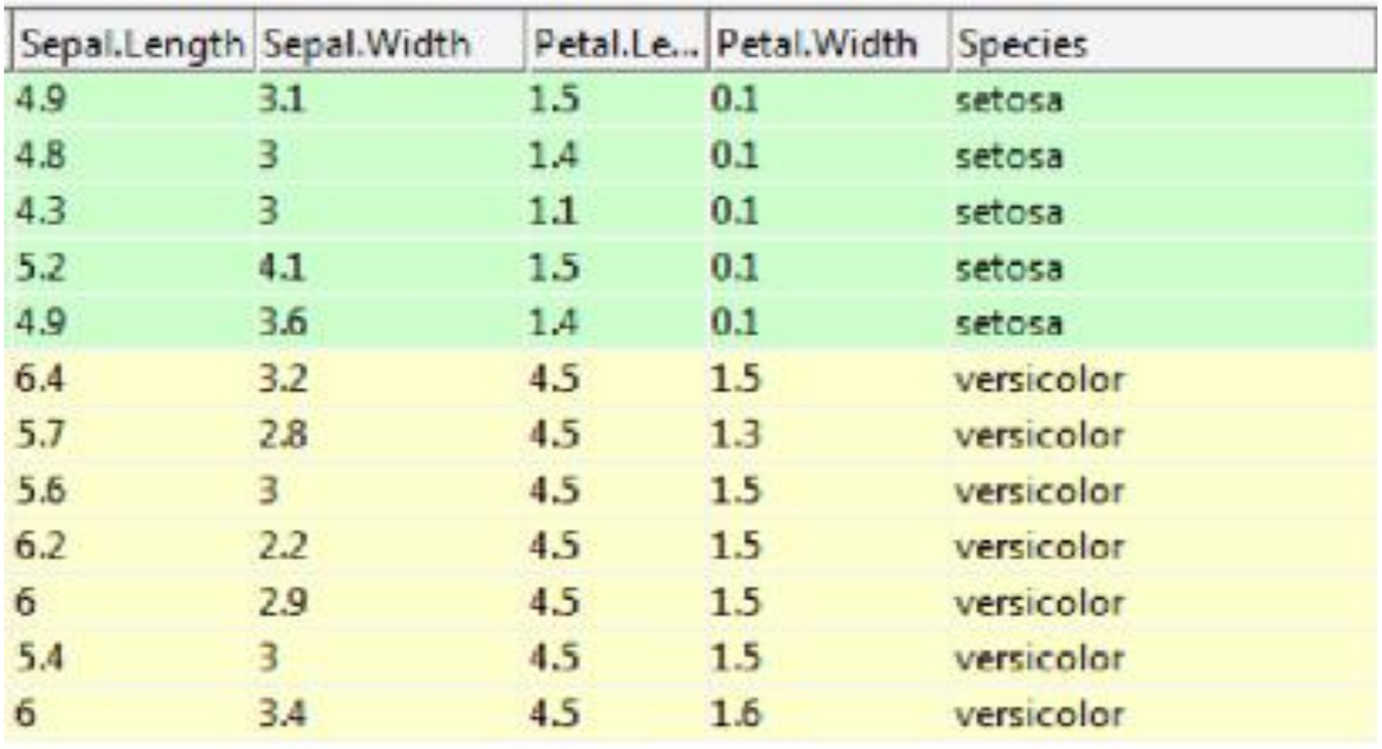
Output:



DATE: 14/06/2023
LAB 8: k-MEANS

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

Dataset:



Sepal.Length	Sepal.Width	Petal.Le...	Petal.Width	Species
4.9	3.1	1.5	0.1	setosa
4.8	3	1.4	0.1	setosa
4.3	3	1.1	0.1	setosa
5.2	4.1	1.5	0.1	setosa
4.9	3.6	1.4	0.1	setosa
6.4	3.2	4.5	1.5	versicolor
5.7	2.8	4.5	1.3	versicolor
5.6	3	4.5	1.5	versicolor
6.2	2.2	4.5	1.5	versicolor
6	2.9	4.5	1.5	versicolor
5.4	3	4.5	1.5	versicolor
6	3.4	4.5	1.6	versicolor

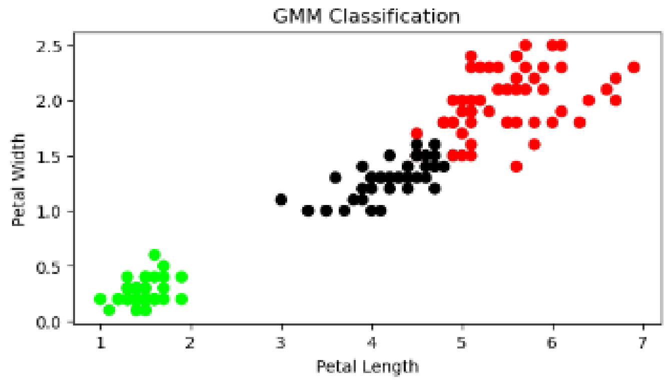
Algorithm:

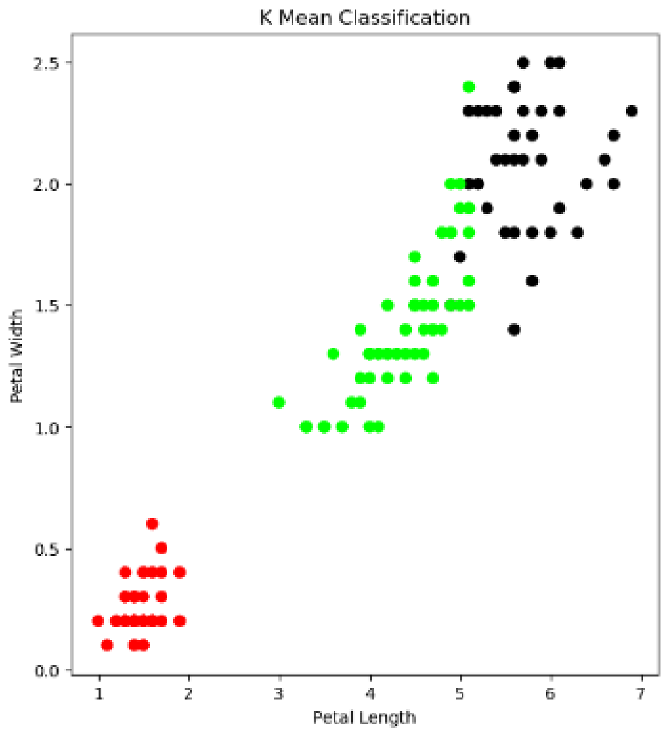
1. Initialize: Choose initial values for the model parameters.
2. Expectation step (E-step):
3. Compute the expected values of the missing or unobserved data given the current parameter estimates.
4. Calculate the posterior probabilities or responsibilities for each data point or latent variable.
5. Maximization step (M-step):
6. Update the model parameters by maximizing the expected log-likelihood (or another objective function) based on the completed data, incorporating the estimated values from the E-step.
7. Evaluate convergence: Check if the change in the model parameters or the log-likelihood is below a specified threshold. If not, go back to step 2.
8. Repeat steps 2-4 until convergence is achieved.
9. Output: Return the estimated model parameters as the final result.

Code:

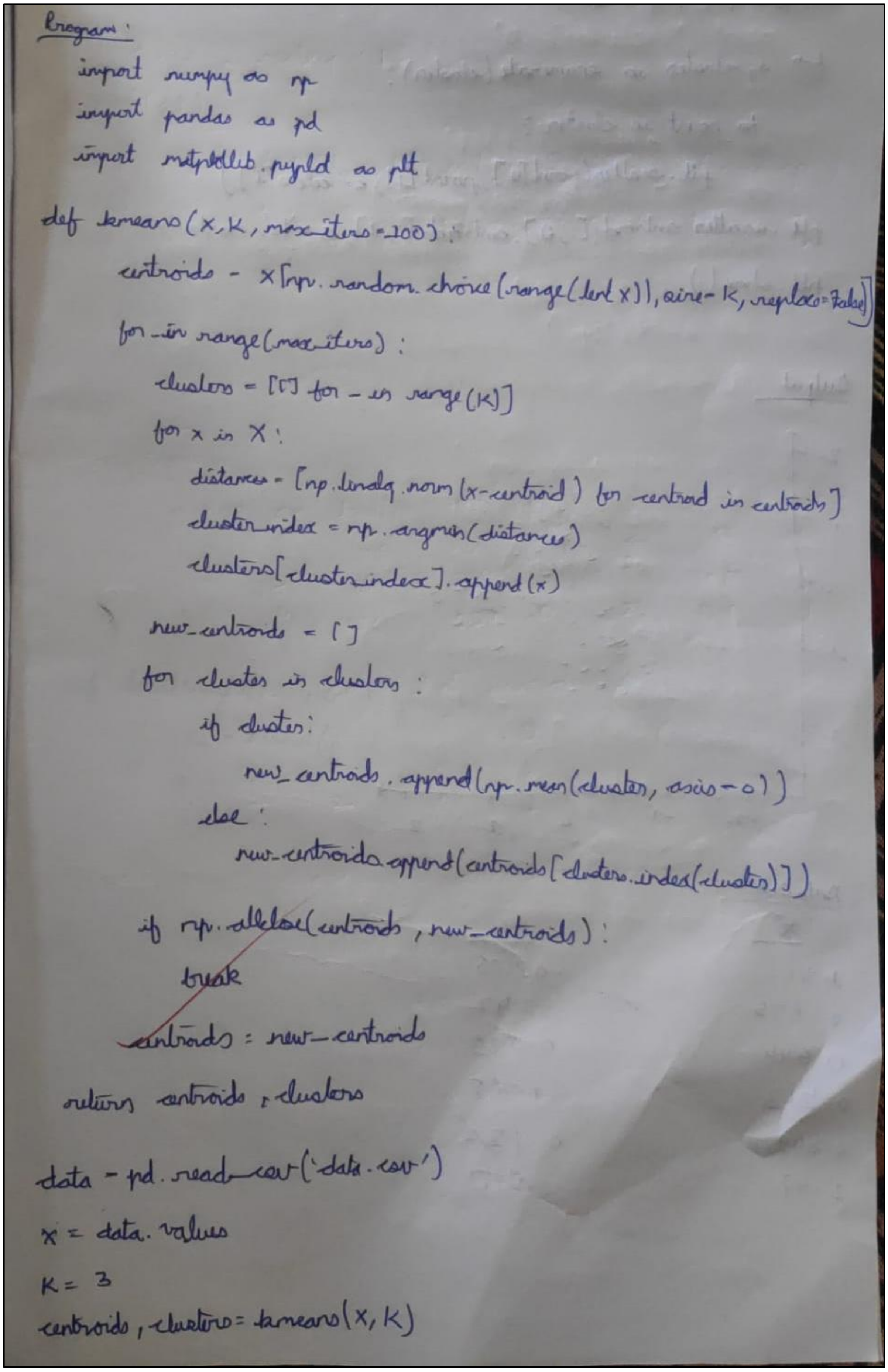
```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import sklearn.metrics as sm
import pandas as pd
import numpy as np

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'])

# Plot the Original Classifications
plt.subplot(1, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')


# Plot the Models Classifications
plt.subplot(1, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K Mean Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
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
scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)
#xs.sample(5)

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

y_gmm = gmm.predict(xs)
#y_cluster_gmm

plt.subplot(2, 2, 3)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y_gmm], s=40)
plt.title('GMM Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

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

```
The accuracy score of K-Mean:  0.8933333333333333
The Confusion matrixof K-Mean:  [[50  0  0]
 [ 0 48  2]
 [ 0 14 36]]
The accuracy score of EM:  0.0
The Confusion matrix of EM:  [[ 0 50  0]
 [ 5  0 45]
 [50  0  0]]
```





Observation:



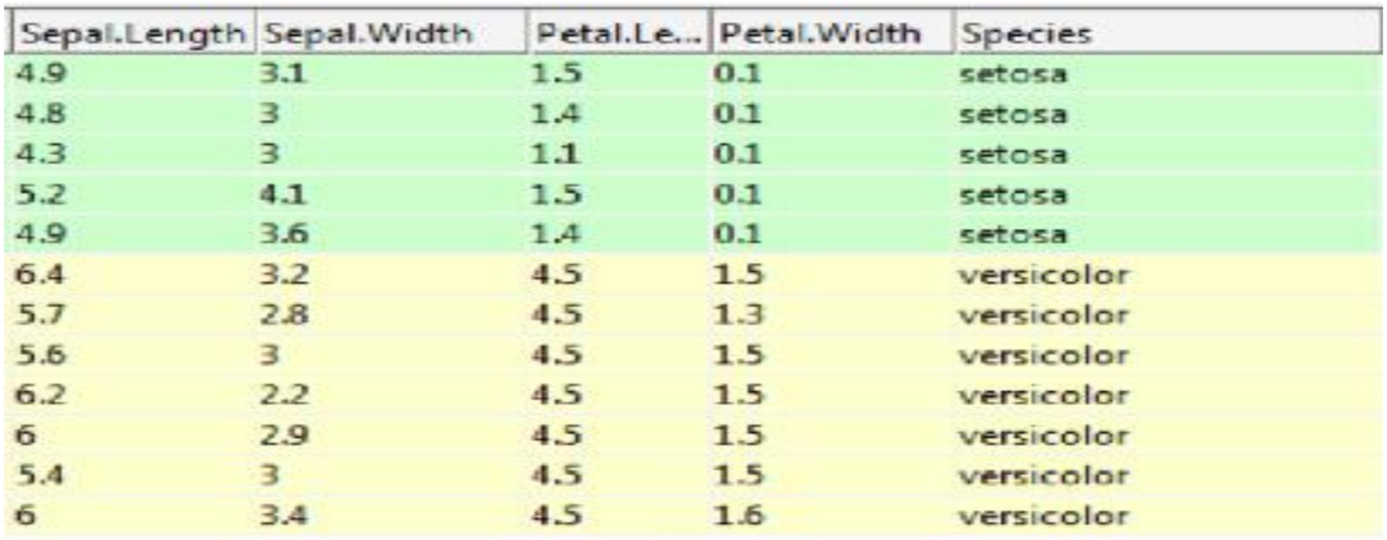
Program:

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

def kmeans(X, K, max_iters=100):
    centroids = X[np.random.choice(range(len(X)), size=K, replace=False)]
    for _ in range(max_iters):
        clusters = [[] for _ in range(K)]
        for x in X:
            distances = [np.linalg.norm(x-centroid) for centroid in centroids]
            cluster_index = np.argmin(distances)
            clusters[cluster_index].append(x)
        new_centroids = []
        for cluster in clusters:
            if cluster:
                new_centroids.append(np.mean(cluster, axis=0))
            else:
                new_centroids.append(centroids[clusters.index(cluster)])
        if np.allclose(centroids, new_centroids):
            break
        centroids = new_centroids
    return centroids, clusters

data = pd.read_csv('data.csv')
X = data.values
K = 3
centroids, clusters = kmeans(X, K)
```

DATE: 14/06/2023
LAB 9: k-NN

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

Dataset:



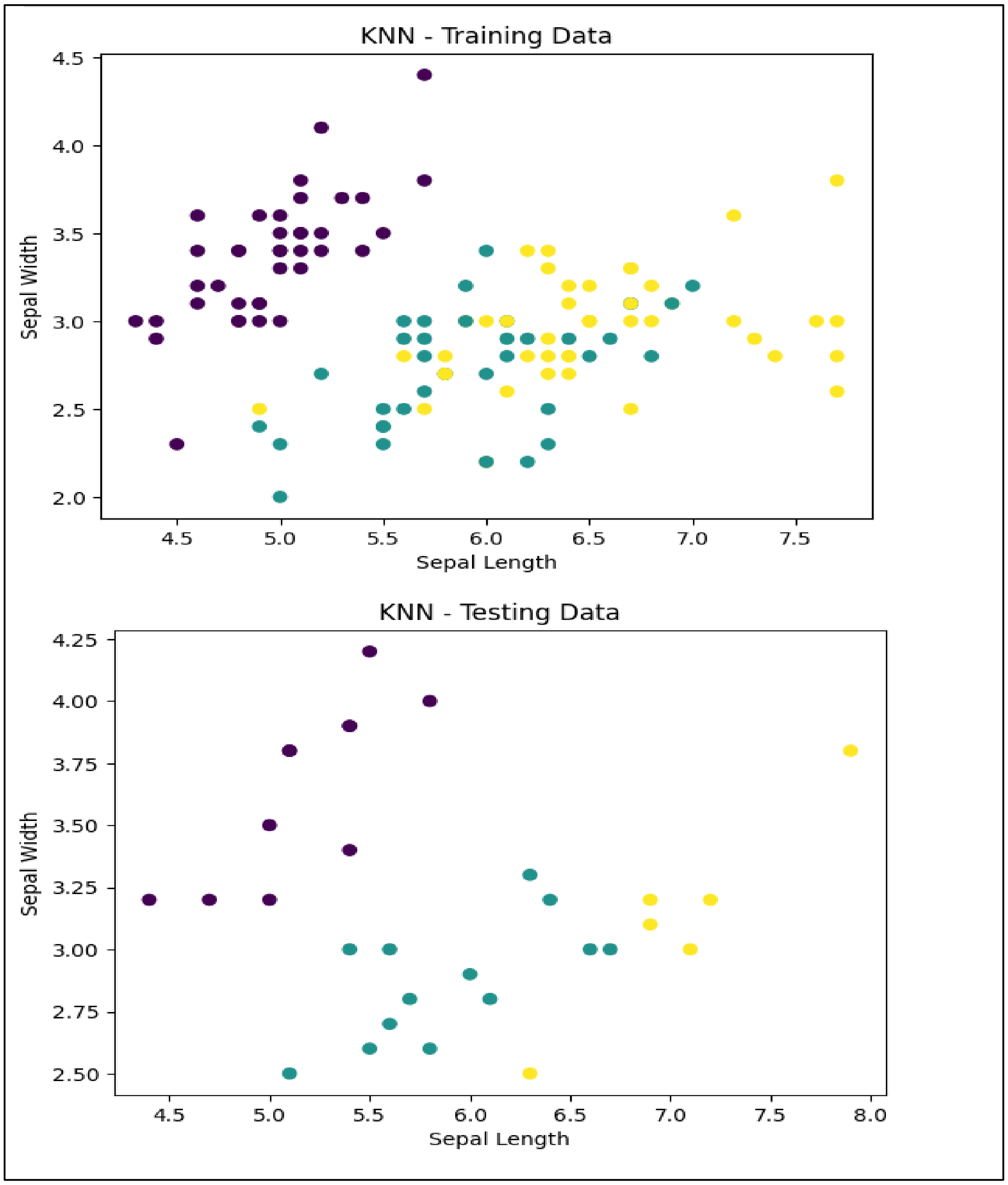
Sepal.Length	Sepal.Width	Petal.Le...	Petal.Width	Species
4.9	3.1	1.5	0.1	setosa
4.8	3	1.4	0.1	setosa
4.3	3	1.1	0.1	setosa
5.2	4.1	1.5	0.1	setosa
4.9	3.6	1.4	0.1	setosa
6.4	3.2	4.5	1.5	versicolor
5.7	2.8	4.5	1.3	versicolor
5.6	3	4.5	1.5	versicolor
6.2	2.2	4.5	1.5	versicolor
6	2.9	4.5	1.5	versicolor
5.4	3	4.5	1.5	versicolor
6	3.4	4.5	1.6	versicolor

Algorithm:

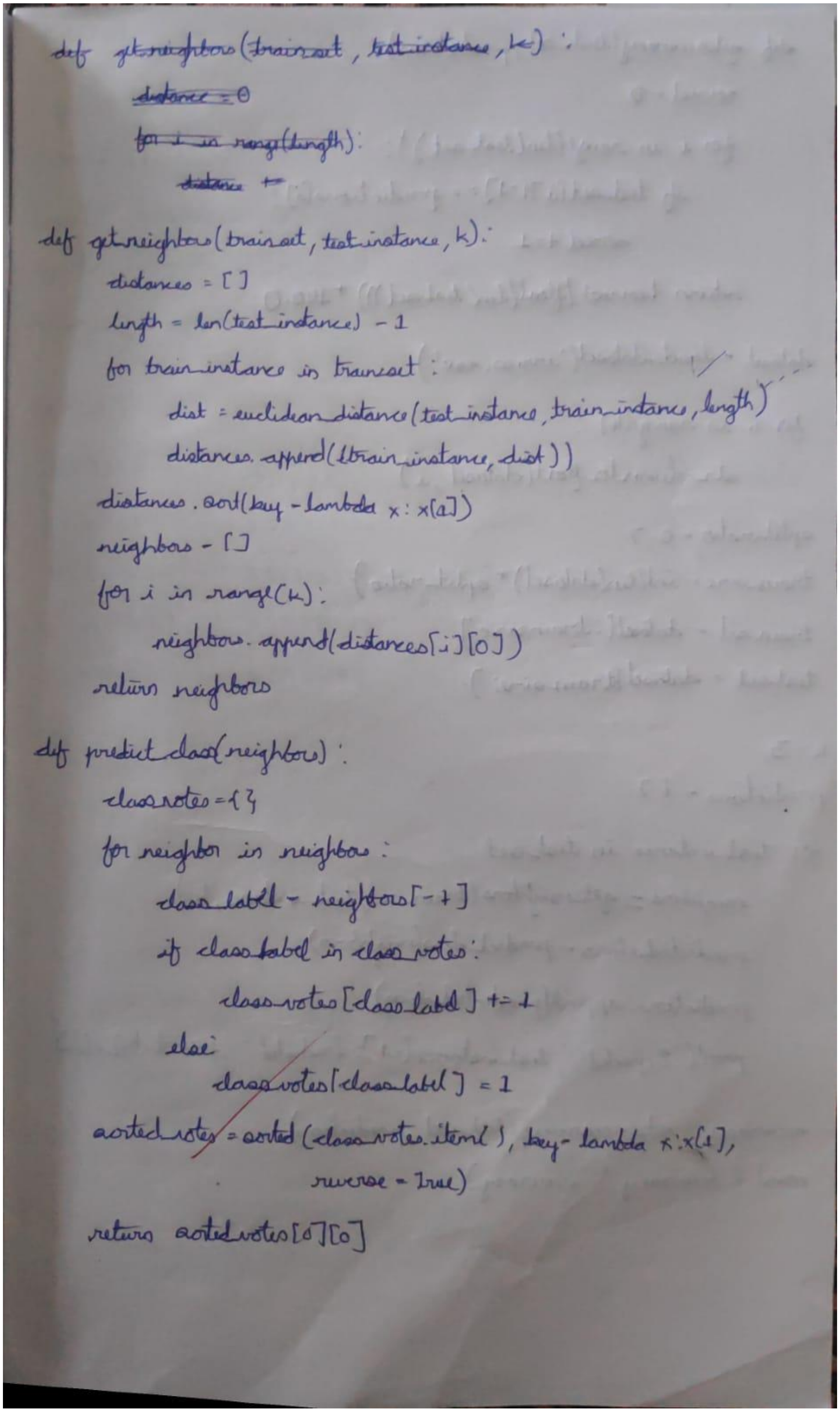
1. Load the training dataset: Prepare the dataset with labeled instances, where each instance consists of a set of features and a corresponding class label (for classification) or target value (for regression).
2. Select the value of K: Determine the number of nearest neighbors, K, that will be considered for making predictions.
3. Normalize the feature values (optional): If the features have different scales or units, it is often beneficial to normalize them to ensure they contribute equally to the distance calculations.
4. Prepare a test instance: Obtain the instance for which you want to make a prediction. This instance should contain the same set of features as the training instances.
5. Calculate distances: Compute the distance between the test instance and all the training instances using a distance metric such as Euclidean distance or Manhattan distance. The distance metric determines how similarity is measured in the feature space.
6. Find K nearest neighbors: Select the K training instances with the shortest distances to the test instance.
7. Make predictions:
    - For classification: Determine the majority class label among the K nearest neighbors and assign it as the predicted class label for the test instance.
    - For regression: Calculate the average or weighted average of the target values of the K nearest neighbors and assign it as the predicted target value for the test instance.
8. Output: Return the predicted class label (for classification) or target value (for regression) as the final result.

Code:

```
import numpy as np
from collections import Counter


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

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

    def euclidean_distance(self, x1, x2):
        return np.sqrt(np.sum((x1 - x2)**2))

    def predict(self, X):
        y_pred = [self._predict(x) for x in X]
        return np.array(y_pred)

    def _predict(self, x):
        distances = [self.euclidean_distance(x, x_train) for x_train in self.X_train]
        k_indices = np.argsort(distances)[:self.k]
        k_nearest_labels = [self.y_train[i] for i in k_indices]
        most_common = Counter(k_nearest_labels).most_common(1)
        return most_common[0][0]



knn = KNN(k=3)  # Specify the value of K (number of neighbors)



from sklearn.datasets import load_iris

data = load_iris()

X = data.data
y = data.target


# train test split
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=1)


knn.fit(X_train,y_train)


y_pred_test = knn.predict(X_test)
y_pred_train = knn.predict(X_train)


# Plotting scatter plot for the training data
import matplotlib.pyplot as plt
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap='viridis')
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.title('KNN - Training Data')
plt.show()

# Plotting scatter plot for the testing data
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap='viridis')
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.title('KNN - Testing Data')
plt.show()
```



Observation:



```
~~def getneighbors(trainset, test_instance, k):~~
    ~~distance = 0~~
    ~~for i in range(length):~~
        ~~distance +=~~

def getneighbors(trainset, test_instance, k):
    distances = []
    length = len(test_instance) - 1
    for train_instance in trainset:
        dist = euclidean_distance(test_instance, train_instance, length)
        distances.append((train_instance, dist))
    distances.sort(key=lambda x: x[1])
    neighbors = []
    for i in range(k):
        neighbors.append(distances[i][0])
    return neighbors

def predict_class(neighbors):
    class_votes = {}
    for neighbor in neighbors:
        class_label = neighbors[-1]
        if class_label in class_votes:
            class_votes[class_label] += 1
        else:
            class_votes[class_label] = 1
    sorted_votes = sorted(class_votes.items(), key=lambda x: x[1],
                          reverse=True)
    return sorted_votes[0][0]
```

Output :

```
Expected : setosa        Predicted : setosa
Expected : setosa        Predicted : setosa
Expected : virginica     Predicted : virginica
Expected : virginica     Predicted : virginica
Expected : versicolor    Predicted : versicolor

Accuracy = 100.0
```

Data set :

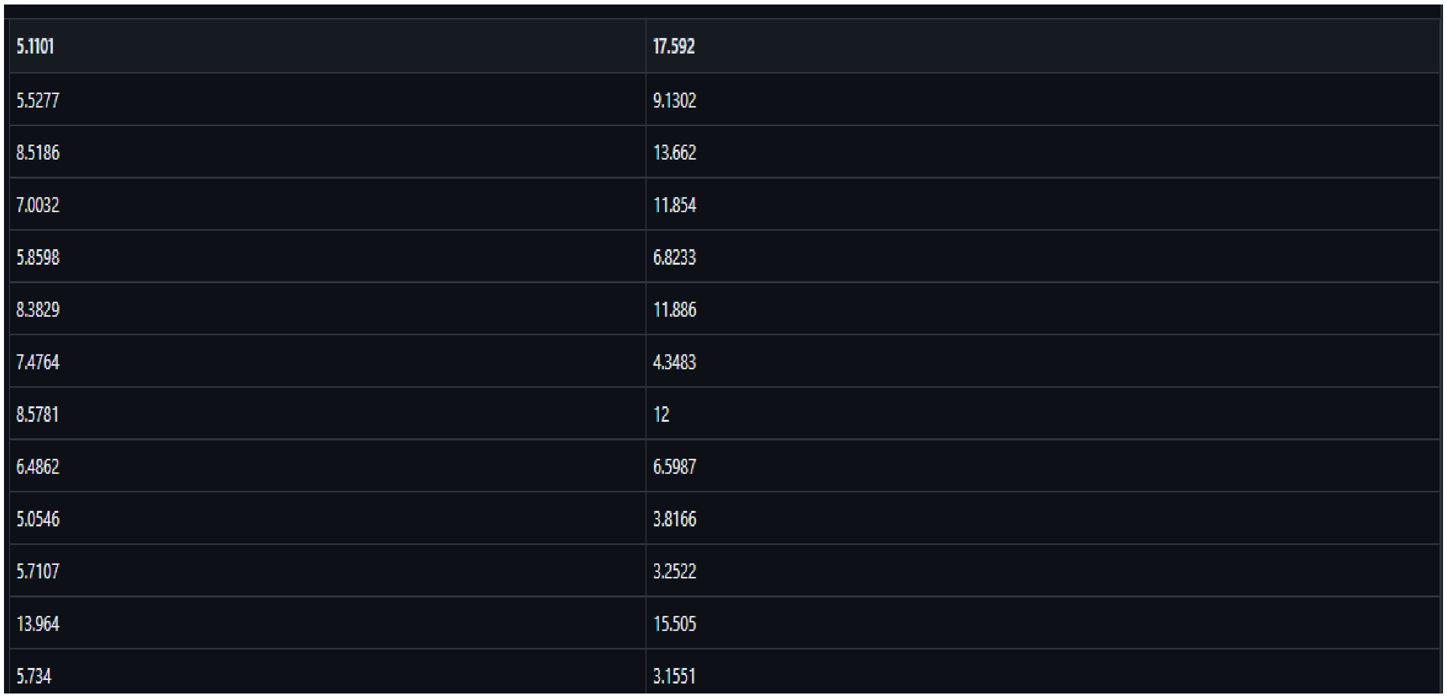
5.64385	2.6547069	3.946820	1109288	versicolor
6.3707907	3.090216828	5.8806962	1.6146228	virginica
5.0725133	3.021130348	1.5807435	0.3001107	setosa
4.766742368	3.970799277	1.1395141	0.5516651	setosa
2.1556401871	2.5935584	4.8932687	1.55768443	virginica

[illegible]
21/06/22

DATE: 14/06/2023
LAB 10: LINEAR REGRESSION

Implement the Linear Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

Dataset:



5.1101	17.592
5.5277	9.1302
8.5186	13.662
7.0032	11.854
5.8598	6.8233
8.3829	11.886
7.4764	4.3483
8.5781	12
6.4862	6.5987
5.0546	3.8166
5.7107	3.2522
13.964	15.505
5.734	3.1551

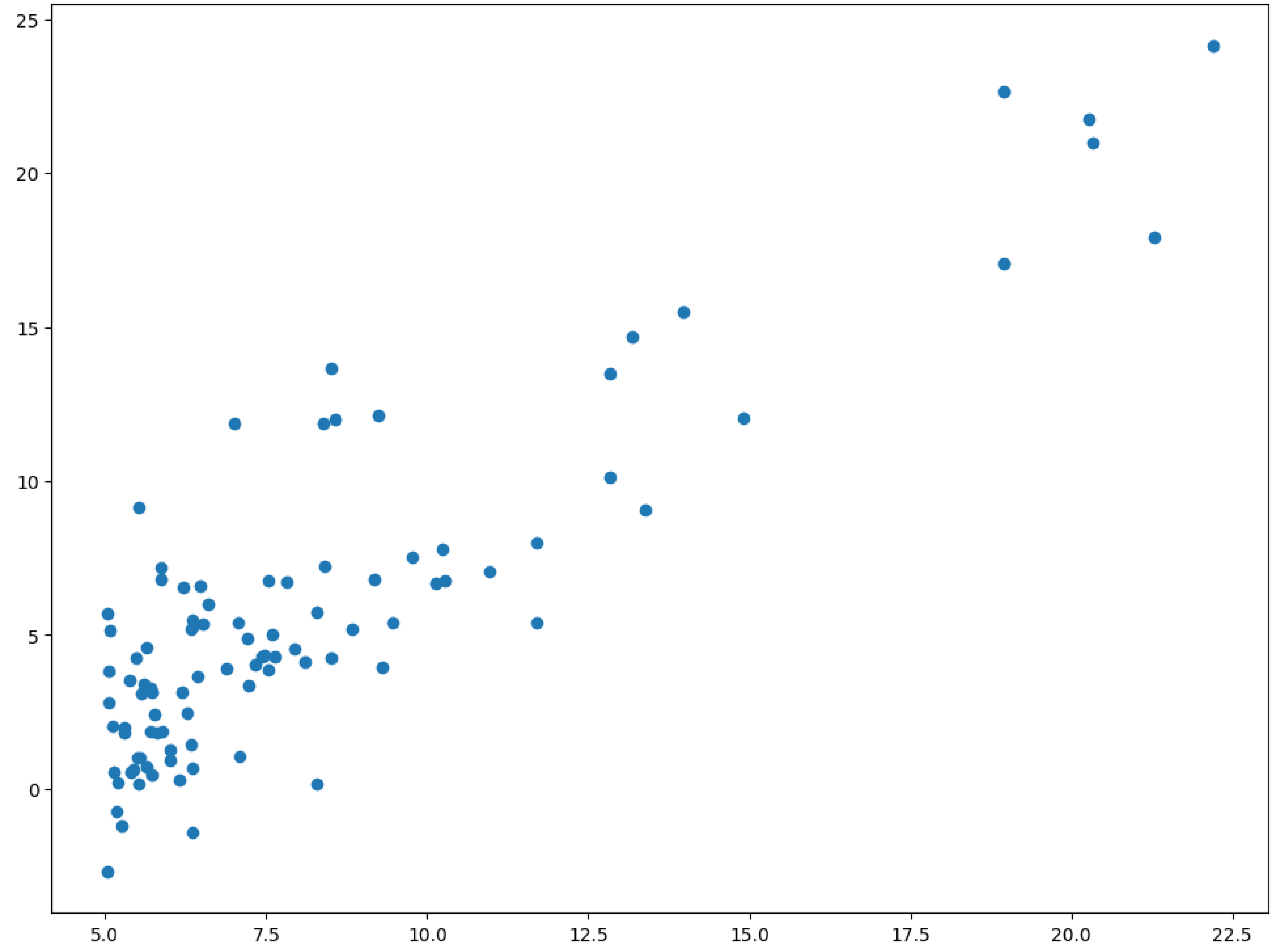
Algorithm:

1. Load the training dataset.
2. Normalize the feature values (optional).
3. Define the hypothesis function as a linear combination of the input features.
4. Initialize the weights.
5. Define the cost function (e.g., Mean Squared Error).
6. Optimize the weights using gradient descent:
    - Iterate through the training data.
    - Update the weights in the direction that minimizes the cost function.
    - Adjust the weights using the gradient and the learning rate.
7. Repeat the gradient descent process until convergence or a maximum number of iterations.

8. Return the learned weights as the coefficients of the linear regression equation.

Code:

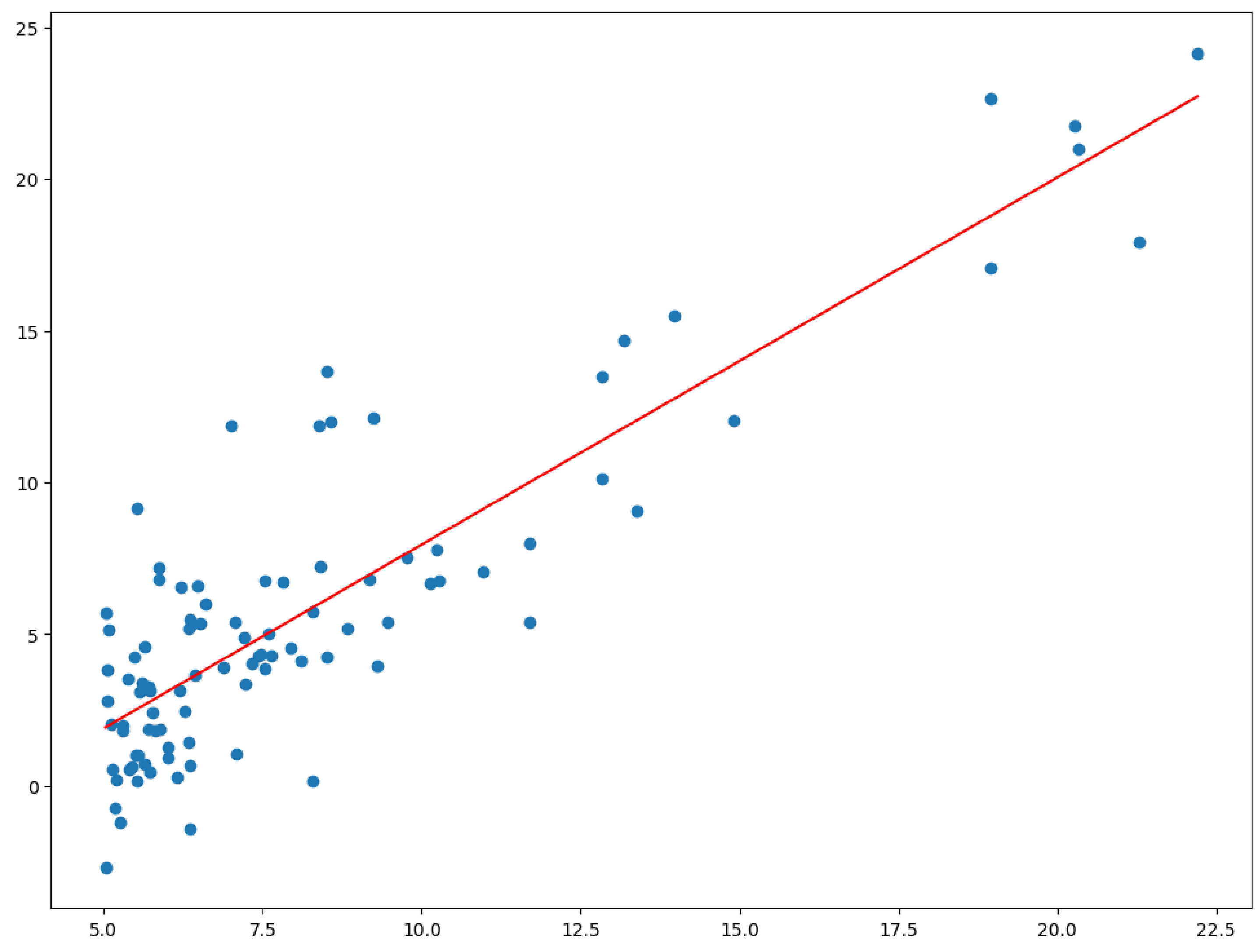
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (12.0, 9.0)

# Preprocessing Input data
data = pd.read_csv('example_data.csv')
X = data.iloc[:, 0]
Y = data.iloc[:, 1]
plt.scatter(X, Y)
plt.show()
```



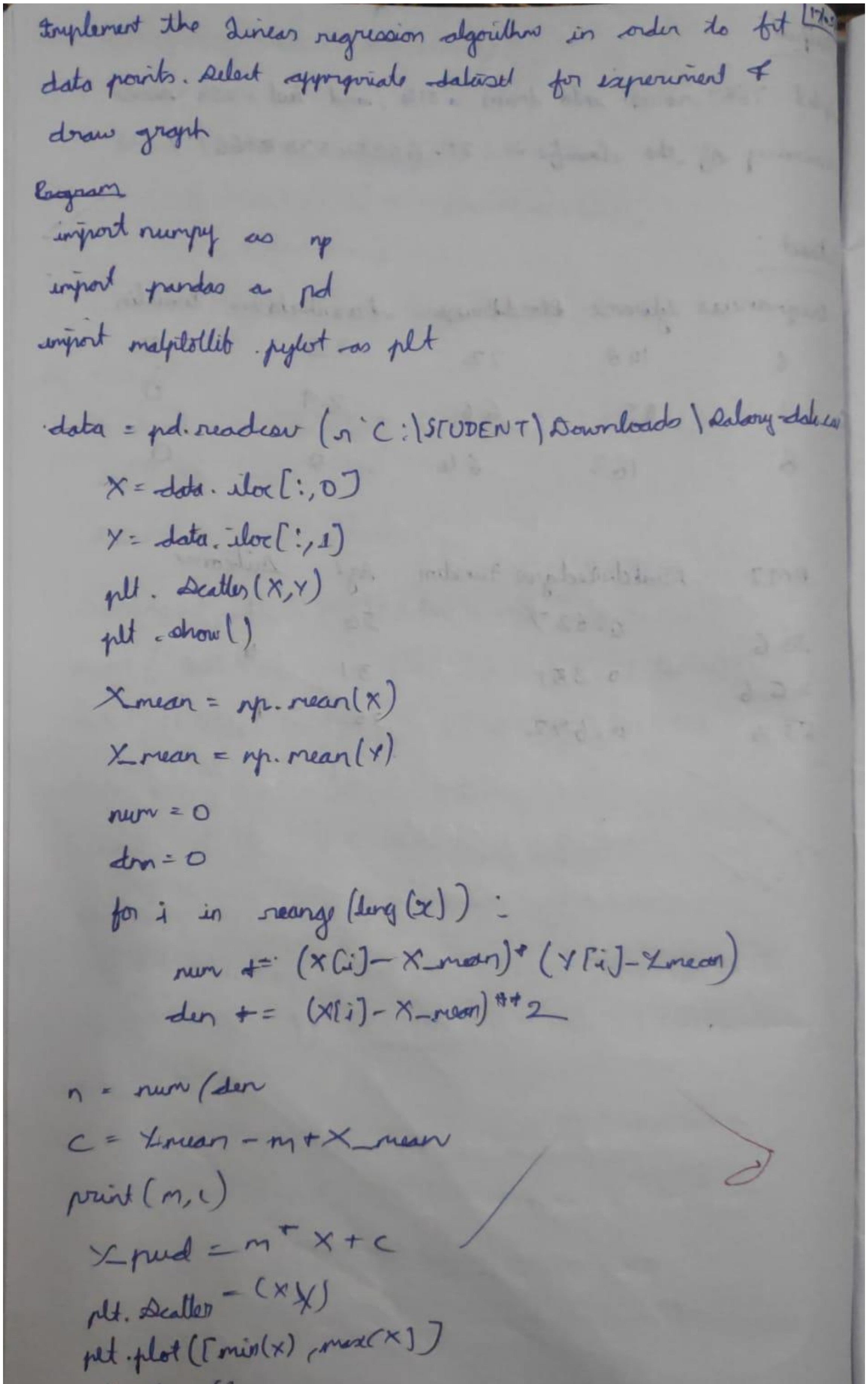
```
# Building the model
X_mean = np.mean(X)
Y_mean = np.mean(Y)

num = 0
den = 0
for i in range(len(X)):
    num += (X[i] - X_mean)*(Y[i] - Y_mean)
    den += (X[i] - X_mean)**2
m = num / den
c = Y_mean - m*X_mean

print (m, c)
```

```
1.210073946912064 -4.150315520211127
```

```
# Making predictions
Y_pred = m*X + c

plt.scatter(X, Y) # actual
# plt.scatter(X, Y_pred, color='red')
plt.plot([min(X), max(X)], [min(Y_pred), max(Y_pred)], color='red') # predicted
plt.show()
```



Observation:



Implement the linear regression algorithm in order to fit data points. Select appropriate dataset for experiment & draw graph

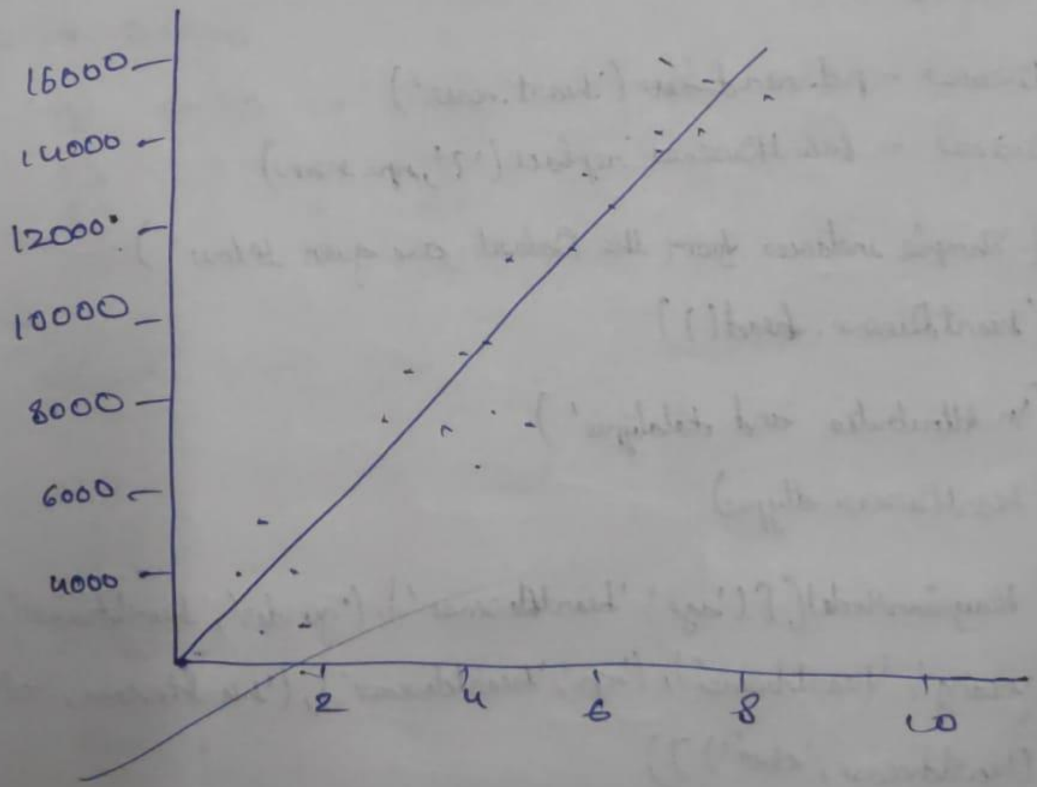
Program

```
import numpy as np
import pandas a pd
import matplotlib .pylot as plt

data = pd.readcsv (r`C:\STUDENT\Downloads\Salary-data.cs
    X = data.iloc[:,0]
    Y = data.iloc[:,1]
    plt. Scatter(X,Y)
    plt .show()
    Xmean = np.mean(X)
    Y_mean = np.mean(Y)
    num = 0
    dm = 0
    for i in range(leng(x)) :
        num += (X[i]-X_mean)* (Y[i]-Ymean)
        den += (X[i]-X_mean)**2

  n = num/den
  c = Ymean - m*X_mean
  print(m,c)
   Y_pred = m* X + c
   plt.Scatter - (X,Y)
   plt.plot([min(x) ,max(X]]
```

Dataset :

Years of Experience	Salary
1.1	39,343
1.3	46205
1.5	37731
2	43525
2.2	39891
2.9	56642
3	60150
3.2	54445

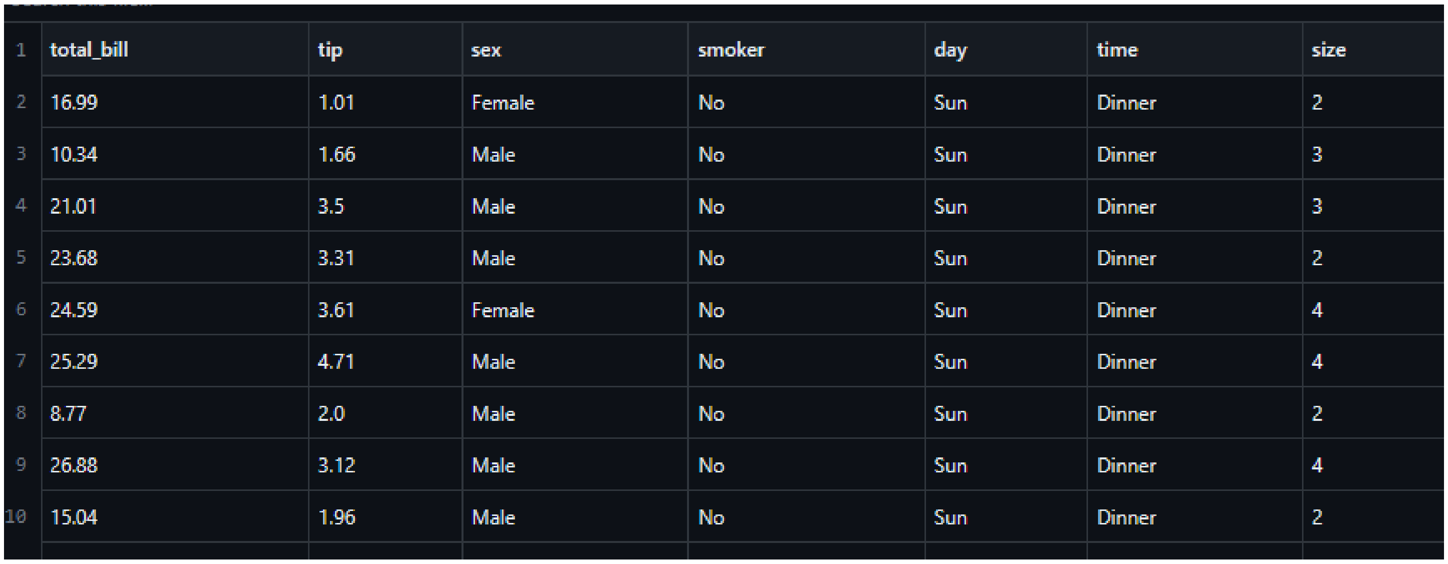
Output :



DATE: 14/06/2023
LAB 11: LOCALLY WEIGHTED REGRESSION

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:



total_bill	tip	sex	smoker	day	time	size
16.99	1.01	Female	No	Sun	Dinner	2
10.34	1.66	Male	No	Sun	Dinner	3
21.01	3.5	Male	No	Sun	Dinner	3
23.68	3.31	Male	No	Sun	Dinner	2
24.59	3.61	Female	No	Sun	Dinner	4
25.29	4.71	Male	No	Sun	Dinner	4
8.77	2.0	Male	No	Sun	Dinner	2
26.88	3.12	Male	No	Sun	Dinner	4
15.04	1.96	Male	No	Sun	Dinner	2

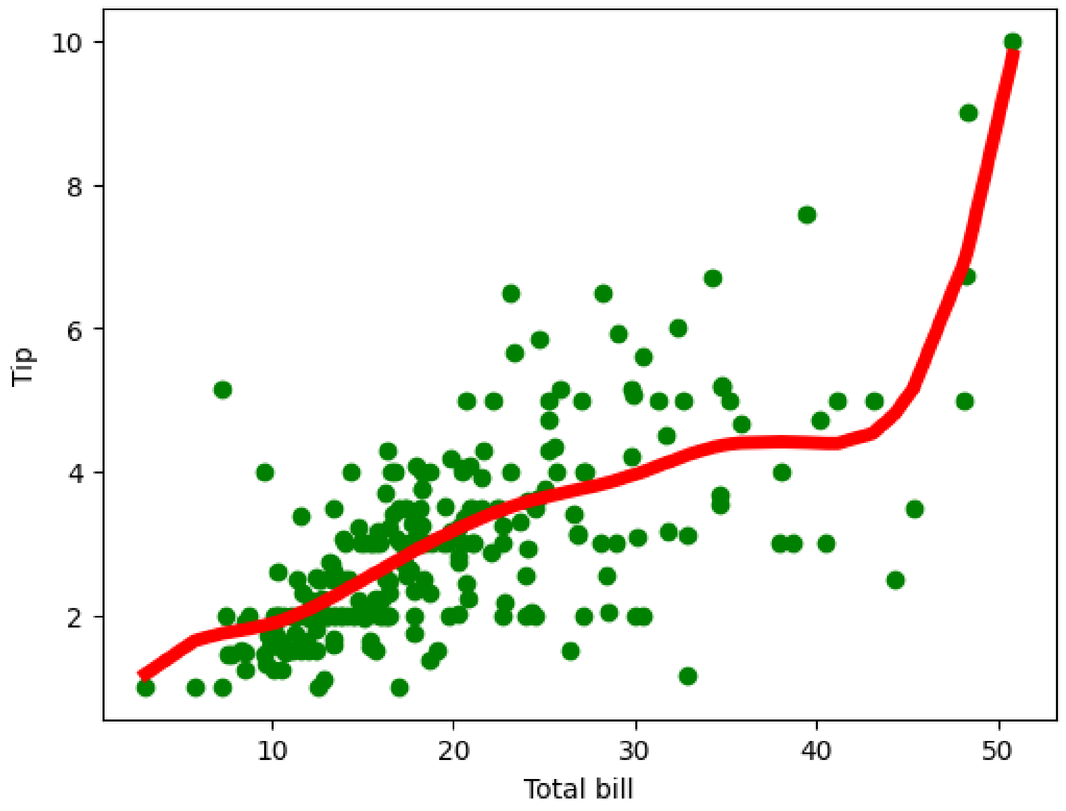
Algorithm:

1. Load the training dataset.
2. Normalize the feature values (optional).
3. Prepare a test instance for which you want to make a prediction.
4. Choose the bandwidth parameter (tau) that controls the weighting of training instances.
5. Calculate weights for each training instance based on its distance from the test instance and the chosen bandwidth.
6. Fit a regression model using the weighted training instances.
7. Make predictions by applying the fitted regression model to the test instance.
8. Return the predicted target value as the final result.

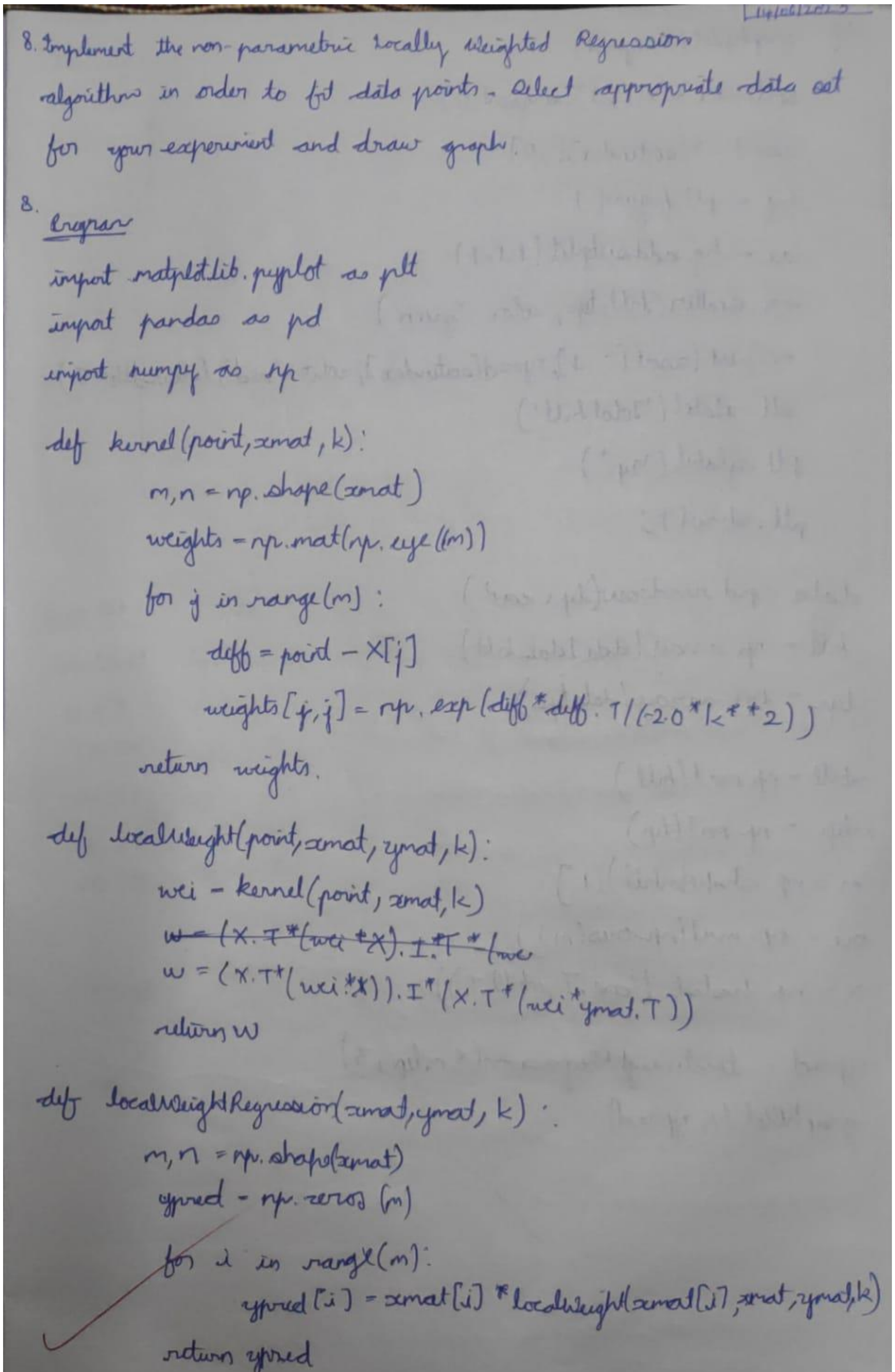
Code:

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

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

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

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

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

# load data points
```

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

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

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



Observation:

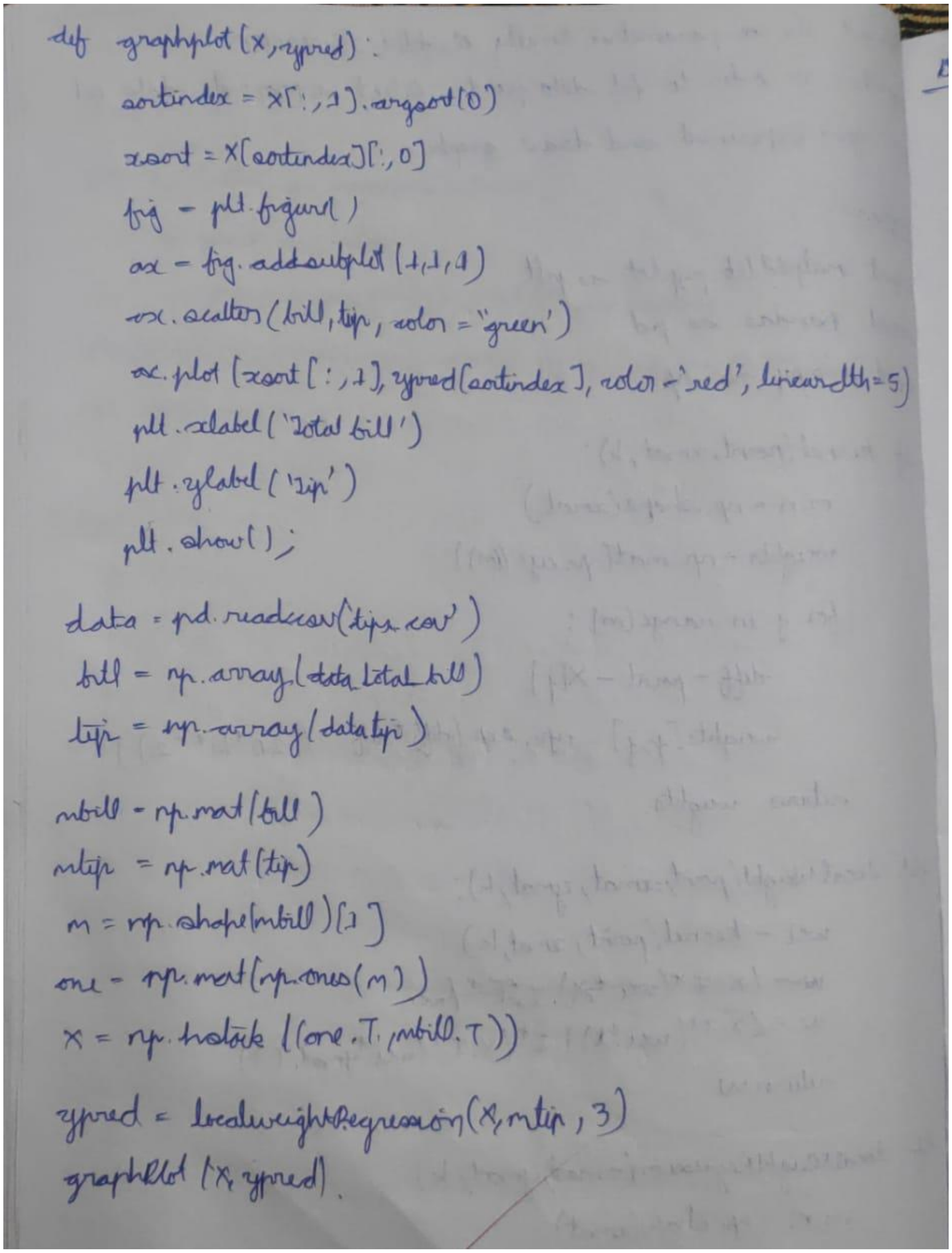


8. Implement the non-parametric locally weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

8.
Program

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

def kernel(point,xmat,k):
        m,n = np.shape(xmat)
        weights = np.mat(np.eye((m)))
        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.T*(we~~
        w = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
        return W

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

S



```
def graphplot(x,ypred):
    sortindex = X[:,1].argsort(0)
    xsort = X[sortindex][:,0]
    fig - plt.figure()
    ax - fig.addsubplot(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();

data = pd.readcsv('tips.csv')
bill = np.array(data.total_bill)
tip = np.array(data.tip)

mbill - np.mat(bill)
mtip = np.mat(tip)
m = np.shape(mbill)[1]
one - np.mat(np.ones(m))
X = np.hstack((one.T, mbill.T))

ypred = localweightRegression(X, mtip, 3)
graphPlot(X, ypred).
```