Practical No. 01

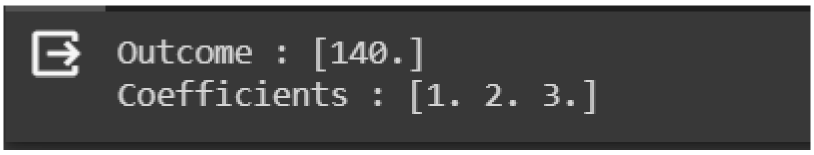
Aim: Design a simple machine learning model to train the training instances and test the same.

Code:

```
# python library to generate random numbers
from random import randint

# the limit within which random numbers are generated
TRAIN_SET_LIMIT = 1000

# to create exactly 100 data items
TRAIN_SET_COUNT = 100

# list that contains input and corresponding output
TRAIN_INPUT = list()
TRAIN_OUTPUT = list()

# loop to create 100 data items with three columns each
for i in range(TRAIN_SET_COUNT):
        a = randint(0, TRAIN_SET_LIMIT)
        b = randint(0, TRAIN_SET_LIMIT)
        c = randint(0, TRAIN_SET_LIMIT)

# creating the output for each data item
        op = a + (2 * b) + (3 * c)
        TRAIN_INPUT.append([a, b, c])

# adding each output to output list
        TRAIN_OUTPUT.append(op)

# Sk-Learn contains the linear regression model
from sklearn.linear_model import LinearRegression
```

```
# Initialize the linear regression model
predictor = LinearRegression(n_jobs =-1)

# Fill the Model with the Data
predictor.fit(X = TRAIN_INPUT, y = TRAIN_OUTPUT)



# Random Test data
X_TEST = [[ 10, 20, 30 ]]

# Predict the result of X_TEST which holds testing data
outcome = predictor.predict(X = X_TEST)

# Predict the coefficients
coefficients = predictor.coef_

# Print the result obtained for the test data
print('Outcome : {}\nCoefficients : {}'.format(outcome, coefficients))
```

Output:



```
Outcome : [140.]
Coefficients : [1. 2. 3.]
```

Practical No. 02

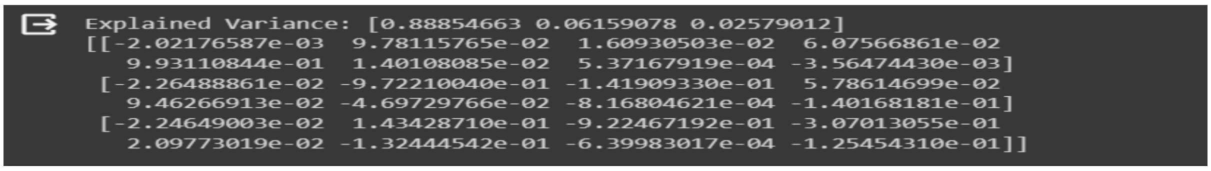
Aim: Perform Data Loading, Feature selection (Principal Component analysis).

Principal component analysis, or PCA, is a dimensionality reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set. Reducing the number of variables of a data set naturally comes at the expense of accuracy, but the trick in dimensionality reduction is to trade a little accuracy for simplicity. Because smaller data sets are easier to explore and visualize and make analyzing data points much easier and faster for machine learning algorithms without extraneous variables to process.

Code :

```
import numpy

from pandas import read_csv

from sklearn.decomposition import PCA

from sklearn.feature_selection import RFE

from sklearn.linear_model import LogisticRegression

# load data

url = "pima-indians-diabetes.csv"

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']

dataframe = read_csv(url, names=names)

array = dataframe.values

X = array[:,0:8]

Y = array[:,8]


# feature extraction

pca = PCA(n_components=3)

fit = pca.fit(X)

# summarize components

print("Explained Variance: %s" % fit.explained_variance_ratio_)

print(fit.components_)
```

Output :



```
Explained Variance: [0.88854663 0.06159078 0.02579012]
[[-2.02176587e-03  9.78115765e-02  1.60930503e-02  6.07566861e-02
   9.93110844e-01  1.40108085e-02  5.37167919e-04 -3.56474430e-03]
 [-2.26488861e-02 -9.72210040e-01 -1.41909330e-01  5.78614699e-02
   9.46266913e-02 -4.69729766e-02 -8.16804621e-04 -1.40168181e-01]
 [-2.24649003e-02  1.43428710e-01 -9.22467192e-01 -3.07013055e-01
   2.09773019e-02 -1.32444542e-01 -6.39983017e-04 -1.25454310e-01]]
```

Practical No. 03

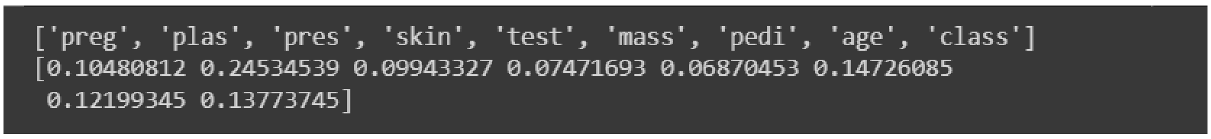
Aim: Perform Data Loading, Feature selection Feature Scoring and Ranking.

An extra tree classifier is a supervised machine learning method that uses decision trees to aggregate the results of multiple de-correlated decision trees. It's also known as an Extremely Randomized Trees Classifier. Extra tree classifiers use averaging to improve predictive accuracy and control over-fitting. They're less sensitive to noise and irrelevant features, and can effectively handle datasets with a large number of features and noisy data.

Code:

```
from pandas import read_csv

from sklearn.ensemble import ExtraTreesClassifier

# load data

url = "pima-indians-diabetes.csv"

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']

dataframe = read_csv(url, names=names)

array = dataframe.values

X = array[:,0:8]

Y = array[:,8]

# feature extraction

model = ExtraTreesClassifier(n_estimators=10)

model.fit(X, Y)

print(names)

print(model.feature_importances_)
```

Output :



```
['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
[0.10480812 0.24534539 0.09943327 0.07471693 0.06870453 0.14726085
 0.12199345 0.13773745]
```

Practical No. 04

Aim:Write a program to implement Decision Tree.

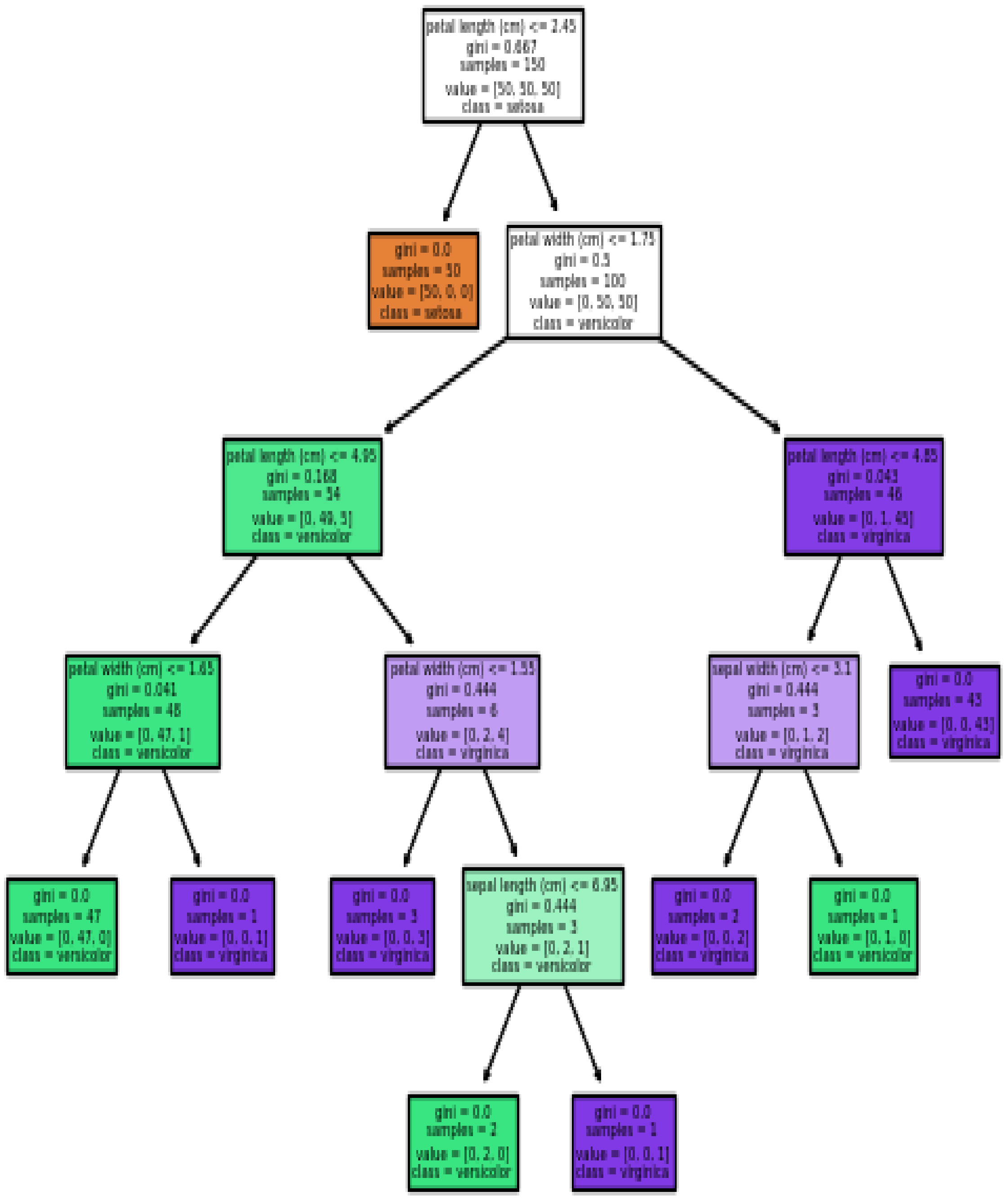
A decision tree is a non-parametric supervised learning algorithm, which is utilized for both classification and regression tasks. It has a hierarchical, tree structure, which consists of a root node, branches, internal nodes and leaf nodes. It builds a flowchart-like tree structure where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label. It is constructed by recursively splitting the training data into subsets based on the values of the attributes until a stopping criterion is met, such as the maximum depth of the tree or the minimum number of samples required to split a node.

During training, the Decision Tree algorithm selects the best attribute to split the data based on a metric such as entropy or Gini impurity, which measures the level of impurity or randomness in the subsets. The goal is to find the attribute that maximizes the information gain or the reduction in impurity after the split.

Code :

```
from matplotlib import pyplot as plt

from sklearn import datasets

from sklearn.tree import DecisionTreeClassifier

from sklearn import tree

# Prepare the data data

iris = datasets.load_iris()

X = iris.data

y = iris.target

# Fit the classifier with default hyper-parameters

clf = DecisionTreeClassifier(random_state=1234)

model = clf.fit(X, y)

fig = plt.figure()

_ = tree.plot_tree(clf,

                feature_names=iris.feature_names,

                class_names=iris.target_names,

                filled=True)

plt.show()
```

Output :


petal length (cm) <= 2.45 gini = 0.667 samples = 150 value = [50, 50, 50] class = setosa
gini = 0.0 samples = 50 value = [50, 0, 0] class = setosa
petal width (cm) <= 1.75 gini = 0.5 samples = 100 value = [0, 50, 50] class = versicolor
petal length (cm) <= 4.95 gini = 0.168 samples = 54 value = [0, 49, 5] class = versicolor
petal length (cm) <= 4.85 gini = 0.043 samples = 46 value = [0, 1, 45] class = virginica
petal width (cm) <= 1.65 gini = 0.041 samples = 48 value = [0, 47, 1] class = versicolor
petal width (cm) <= 1.55 gini = 0.444 samples = 6 value = [0, 2, 4] class = virginica
sepal width (cm) <= 3.1 gini = 0.444 samples = 3 value = [0, 1, 2] class = virginica
gini = 0.0 samples = 43 value = [0, 0, 43] class = virginica
gini = 0.0 samples = 47 value = [0, 47, 0] class = versicolor
gini = 0.0 samples = 1 value = [0, 0, 1] class = virginica
gini = 0.0 samples = 3 value = [0, 0, 3] class = virginica
sepal length (cm) <= 6.95 gini = 0.444 samples = 3 value = [0, 2, 1] class = versicolor
gini = 0.0 samples = 2 value = [0, 0, 2] class = virginica
gini = 0.0 samples = 1 value = [0, 1, 0] class = versicolor
gini = 0.0 samples = 2 value = [0, 2, 0] class = versicolor
gini = 0.0 samples = 1 value = [0, 0, 1] class = virginica

Practical No. 05

Aim: For a given set of training data examples stored in a .CSV file implement Least Square Regression algorithm.

Least squares regression is a method of linear regression that establishes the relationship between dependent and independent variables along a linear line. It's also known as a line of best fit or a trend line. The least squares regression line represents the relationship between variables in a scatterplot. It fits the line to the data points in a way that minimizes the sum of the squared vertical distances between the line and the points. The least squares regression line is a mathematical model used to predict the value of y for a given x. It helps predict results based on an existing set of data as well as clear anomalies in the data. Anomalies are values that are too good, or bad, to be true or that represent rare cases. The term least squares is used because it is the smallest sum of squares of errors, which is also called the variance.

Code :

```
# regression line

"""

To find regression line, we need to find a and b.

Calculate a, which is given by a = (\sum yi)/n - b * (\sum xi)/n

Calculate b, which is given by

b = (n*\sum(xi*yi) - \sum (xi)* \sum (yi))/(n*\sum (xi)^{2}-(\sum xi)^{2})

Put value of a and b in the equation of regression line.

"""

# Function to calculate b

def calculateB(x, y, n):


        # sum of array x

        sx = sum(x)


        # sum of array y

        sy = sum(y)


        # for sum of product of x and y

        sxsy = 0


        # sum of square of x

        sx2 = 0
```

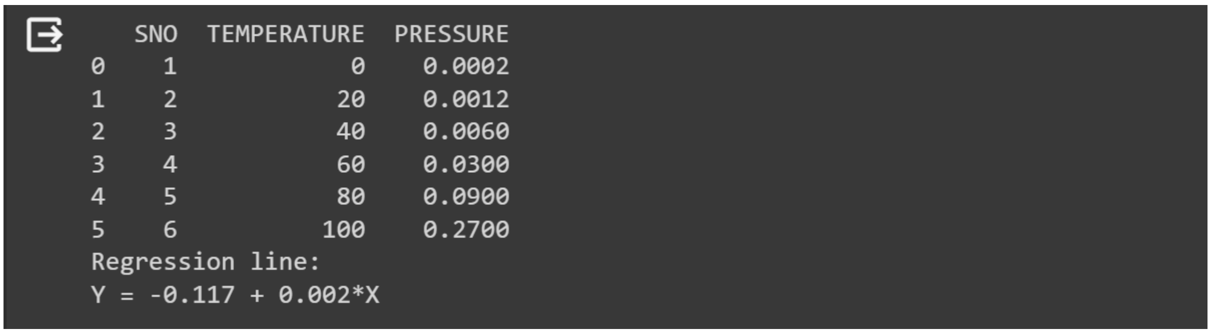
```
    for i in range(n):
        sxsy += x[i] * y[i]
        sx2 += x[i] * x[i]
    b = (n * sxsy - sx * sy)/(n * sx2 - sx * sx)
    return b

# Function to find the
# least regression line
def leastRegLine(X,Y,n):

    # Finding b
    b = calculateB(X, Y, n)
    meanX = int(sum(X)/n)
    meanY = int(sum(Y)/n)

    # Calculating a
    a = meanY - b * meanX

    # Printing regression line
    print("Regression line:")
    print("Y = ", '%.3f'%a, " + ", '%.3f'%b, "*X", sep="")

# Driver code

# Statistical data
import pandas as pd
# Step 1 :Import libraries and dataset
datas = pd.read_csv('data.csv')
print(datas )
X = datas.TEMPERATURE
```

Y = datas.PRESSURE

n = len(X)

leastRegLine(X, Y, n)

Output:



```
   SNO  TEMPERATURE  PRESSURE
0    1            0    0.0002
1    2           20    0.0012
2    3           40    0.0060
3    4           60    0.0300
4    5           80    0.0900
5    6          100    0.2700
Regression line:
Y = -0.117 + 0.002*X
```

Practical No. 06

Aim: For a given set of training data examples stored in a .CSV file implement Linear Regression algorithm.

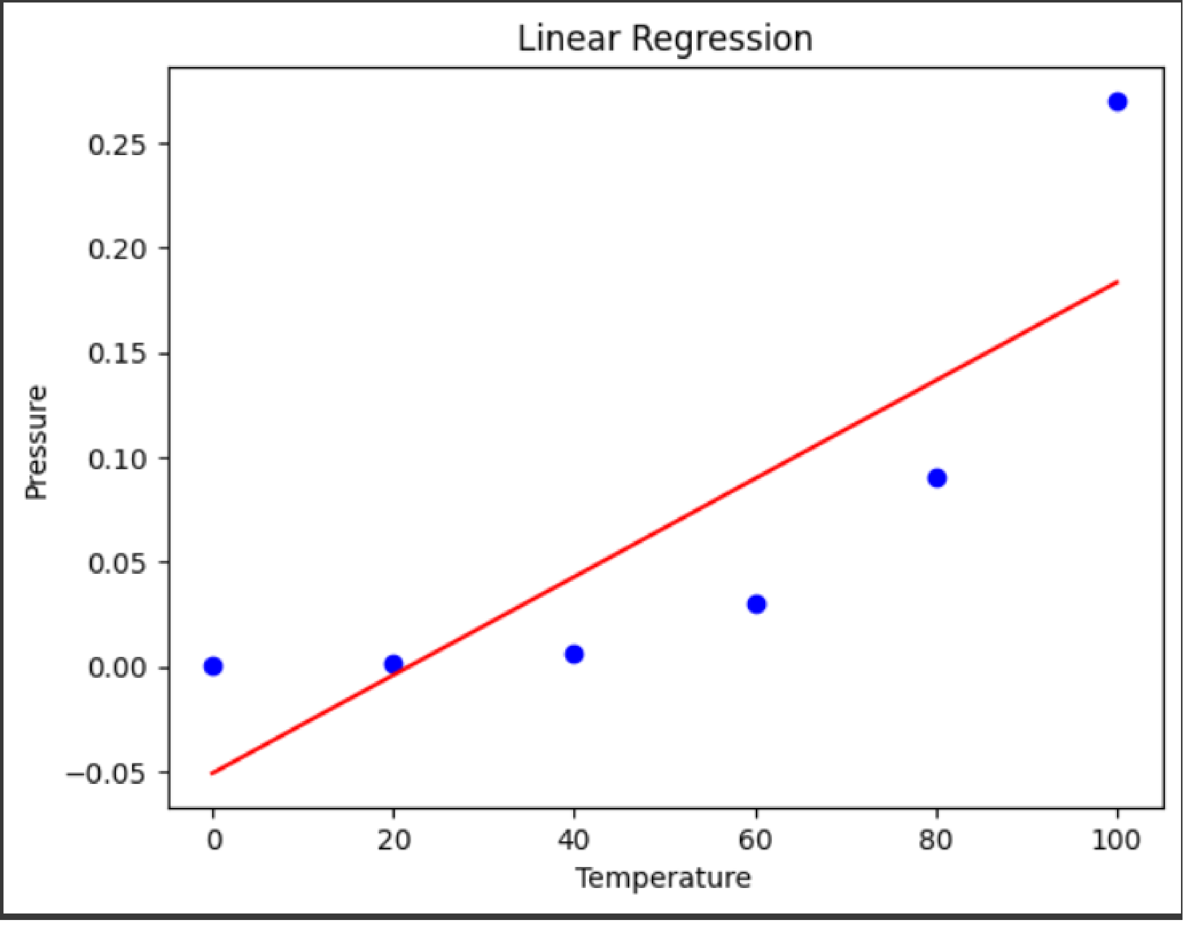
Linear regression is a statistical method that uses a linear equation to model the relationship between two variables. It's used in data science and machine learning to predict the outcome of future events. Linear regression assumes a linear relationship between the independent variable and the dependent variable. It aims to find the best-fitting line that describes the relationship.

Code :

```
import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Step 1 :Import libraries and dataset

datas = pd.read_csv('data.csv')

print(datas )

#Step 2: Dividing the dataset into 2 components

X = datas.iloc[:, 1:2].values

y = datas.iloc[:, 2].values

#Step 3: Fitting Linear Regression to the dataset

from sklearn.linear_model import LinearRegression

lin = LinearRegression()

lin.fit(X, y)

plt.scatter(X, y, color = 'blue')

plt.plot(X, lin.predict(X), color = 'red')

plt.title('Linear Regression')

plt.xlabel('Temperature')

plt.ylabel('Pressure')

plt.show()
```

Output :

```
   SNO  TEMPERATURE  PRESSURE
0    1            0    0.0002
1    2           20    0.0012
2    3           40    0.0060
3    4           60    0.0300
4    5           80    0.0900
5    6          100    0.2700
```



Practical No. 07

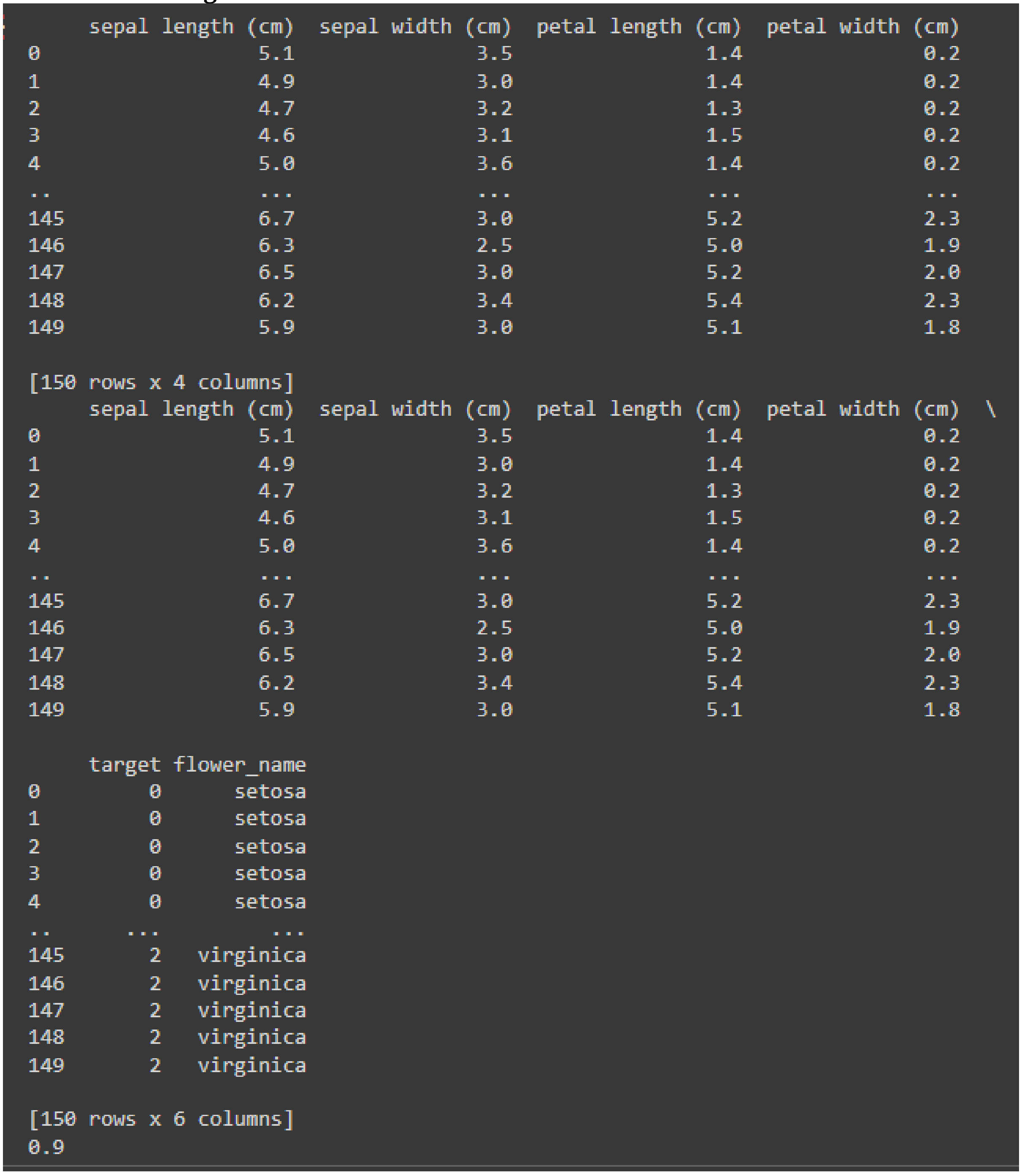
Aim: Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set.

K-nearest neighbors (KNN) is a supervised learning algorithm used for classification and regression. It's a simple and effective machine learning technique. KNN is based on the idea that similar data points tend to have similar labels or values. It uses proximity to make classifications or predictions about the grouping of an individual data point. You can calculate the distance between a new entry and other existing values using the Euclidean distance formula. You can also calculate the distance using the Manhattan and Minkowski distance formulas.

Code :

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier


iris = load_iris()
df = pd.DataFrame(iris.data,columns=iris.feature_names)
print(df)
df['target'] = iris.target
df['flower_name'] =df.target.apply(lambda x: iris.target_names[x])
print(df)
df0 = df[:50]          # setosa
df1 = df[50:100]        # versicolor
df2 = df[100:]          # virginica
X = df.drop(['target','flower_name'], axis='columns')
y = df.target
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2)


knn = KNeighborsClassifier(n_neighbors=10)
knn.fit(X_test, y_test)
print(knn.score(X_test, y_test))
```

Output :



```
     sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                  5.1               3.5                1.4               0.2
1                  4.9               3.0                1.4               0.2
2                  4.7               3.2                1.3               0.2
3                  4.6               3.1                1.5               0.2
4                  5.0               3.6                1.4               0.2
..                 ...               ...                ...               ...
145                6.7               3.0                5.2               2.3
146                6.3               2.5                5.0               1.9
147                6.5               3.0                5.2               2.0
148                6.2               3.4                5.4               2.3
149                5.9               3.0                5.1               1.8

[150 rows x 4 columns]
     sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                  5.1               3.5                1.4               0.2
1                  4.9               3.0                1.4               0.2
2                  4.7               3.2                1.3               0.2
3                  4.6               3.1                1.5               0.2
4                  5.0               3.6                1.4               0.2
..                 ...               ...                ...               ...
145                6.7               3.0                5.2               2.3
146                6.3               2.5                5.0               1.9
147                6.5               3.0                5.2               2.0
148                6.2               3.4                5.4               2.3
149                5.9               3.0                5.1               1.8

     target flower_name
0         0      setosa
1         0      setosa
2         0      setosa
3         0      setosa
4         0      setosa
..      ...         ...
145       2   virginica
146       2   virginica
147       2   virginica
148       2   virginica
149       2   virginica

[150 rows x 6 columns]
0.9
```

Practical No. 08

Aim: Implement the classification model using clustering for the following techniques with K means clustering

K-means clustering is an unsupervised machine learning algorithm that groups unlabeled data into different clusters. The algorithm partitions n observations into k clusters, where each observation belongs to the cluster with the nearest mean.

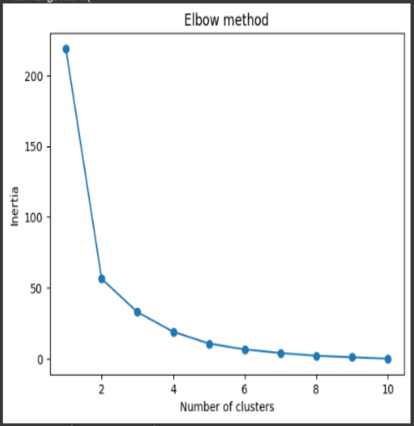
Code :

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
x = [4, 5, 10, 4, 3, 11, 14 , 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]
######
from sklearn.cluster import KMeans
data = list(zip(x, y))
inertias = []
for i in range(1,11):
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(data)
    inertias.append(kmeans.inertia_)
plt.plot(range(1,11), inertias, marker='o')
plt.title('Elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()
#####
data = list(zip(x, y))
kmeans = KMeans(n_clusters=2)
kmeans.fit(data)
plt.scatter(x, y, c=kmeans.labels_)
```

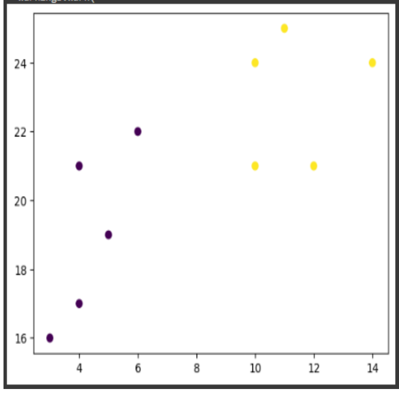
plt.show()

Output :

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



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



Practical No. 09

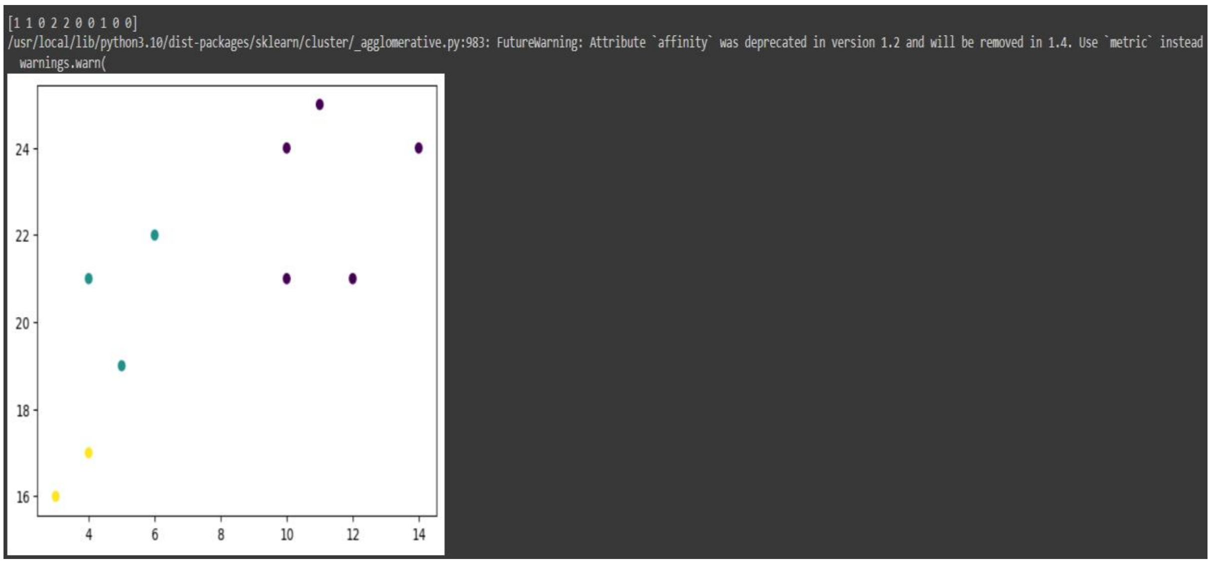
Aim: Implement the classification model using clustering for the following techniques with hierarchical clustering with Prediction

Hierarchical clustering is a popular method for grouping objects. It creates groups so that objects within a group are similar to each other and different from objects in other groups. Clusters are visually represented in a hierarchical tree called a dendrogram.

Code :

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import AgglomerativeClustering

from scipy.cluster.hierarchy import dendrogram, linkage

x = [4, 5, 10, 4, 3, 11, 14 , 6, 10, 12]

y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]

data = list(zip(x, y))

hierarchical_cluster = AgglomerativeClustering(n_clusters=3,
affinity='euclidean', linkage='ward')

labels = hierarchical_cluster.fit_predict(data)

print(labels)

plt.scatter(x, y, c=labels)

plt.show()
```

Output :



```
[1 1 0 2 2 0 0 1 0 0]
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_agglomerative.py:983: FutureWarning: Attribute `affinity` was deprecated in version 1.2 and will be removed in 1.4. Use `metric` instead
  warnings.warn(
```