(Assignment-01)

Numpy

```
from genericpath import getsize
import sys as sys
a="Hello, man how are you "
sys. getsizeof(a)
fact = 1
for i in range(1,11):
 fact = fact*i
 print(sys,a,fact)
```

```
<module 'sys' (built-in)> Hello, man how are you  1
<module 'sys' (built-in)> Hello, man how are you  2
<module 'sys' (built-in)> Hello, man how are you  6
<module 'sys' (built-in)> Hello, man how are you  24
<module 'sys' (built-in)> Hello, man how are you  120
<module 'sys' (built-in)> Hello, man how are you  720
<module 'sys' (built-in)> Hello, man how are you  5040
<module 'sys' (built-in)> Hello, man how are you  40320
<module 'sys' (built-in)> Hello, man how are you  362880
<module 'sys' (built-in)> Hello, man how are you  3628800
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
x = []
print(type(x))

x = ["hello", "kaho", "chalo", "suno", 67]
# print(x[1:4])
# print(len(x))
x [4] = 'hello habibi'
# print(x)
x.append('hey mister')
# print(x)
x[4] = ["hey", 5, "say hi"]
```

```
x[4][0] = [2,6,7,[4,5]]
print(x[:5:1])
for t in x:
  print(t,end=" ")
```

```
<class 'list'>
['hello', 'kaho', 'chalo', 'suno', [[2, 6, 7, [4, 5]], 5, 'say hi']]
hello kaho chalo suno [[2, 6, 7, [4, 5]], 5, 'say hi'] hey mister
```

```
import numpy as np
list1 = [1,2,3,4,5,6,7,78,8,8,6,5,4,53,[1,5]]
list2 = []
print(dir(list1))
print(len(dir(list1)))
print(list1)
x1 = np.array(list1)
y = x
print(y)
print(x1)
for i in list1:
  print(i)
print(list2)
list2.append(list1)
print(list2)
```

```
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__'
46
[1, 2, 3, 4, 5, 6, 7, 78, 8, 8, 6, 5, 4, 53, [1, 5]]
[[54, 5], 6, 7]
[1 2 3 4 5 6 7 78 8 8 6 5 4 53 list([1, 5])]
```

```
import numpy as np

list1 = [1,2,3,25,4,53,[1,5]]

list2 = []

print(dir(list1))

print(len(dir(list1)))

print(list1)

x1 = np.array(list1)

y = x

print(y)

print(x1)
```

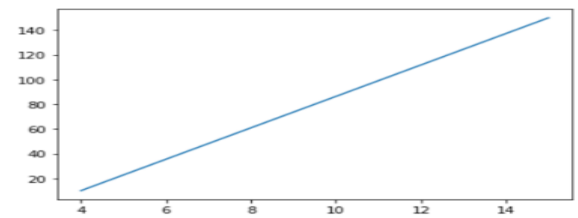
```
for i in list1:
  print(i)
print(list2)
list2.append(list1)
print(list2)
 import numpy as np
list1 = [1,2,3,25,4,53,[1,5]]
list2 = []
print(dir(list1))
print(len(dir(list1)))
print(list1)
x1 = np.array(list1)
y = x
print(y)
print(x1)
for i in list1:
  print(i)
print(list2)
list2.append(list1)
print(list2)
```

```
['__add__', '__class__', '__class_getitem__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__'
47
[1, 2, 3, 25, 4, 53, [1, 5]]
['hello', 'kaho', 'chalo', 'suno', [[2, 6, 7, [4, 5]], 5, 'say hi'], 'hey mister']
[1 2 3 25 4 53 list([1, 5])]
1
2
3
25
4
53
[1, 5]
[]
[[1, 2, 3, 25, 4, 53, [1, 5]]]
<ipython-input-9-381b22bfe96a>:7: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-c
  x1 = np.array(list1)
```

(Assignment-02) Matplotlib

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array([4,15])
y = np.array([10,150])
plt.plot(x,y)
plt.show()
```

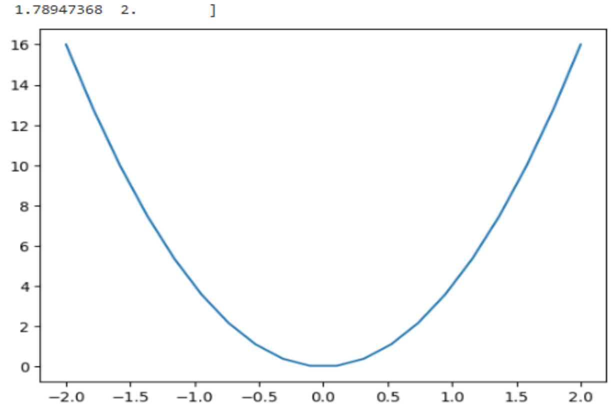
Output :-




```
import matplotlib.pyplot as plt
import numpy as np
x  = np.linspace(-2,2,20)
print(x)
y = 4*x**2
plt.plot(x,y)
plt.show()
```

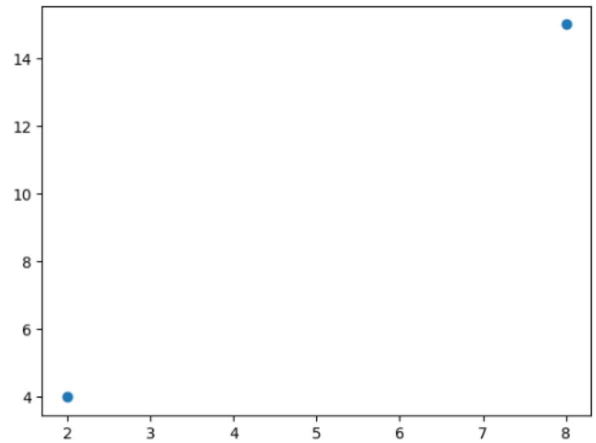
output:-

```
[-2.         -1.78947368 -1.57894737 -1.36842105 -1.15789474 -0.94736842
 -0.73684211 -0.52631579 -0.31578947 -0.10526316  0.10526316  0.31578947
  0.52631579  0.73684211  0.94736842  1.15789474  1.36842105  1.57894737
  1.78947368  2.        ]
```



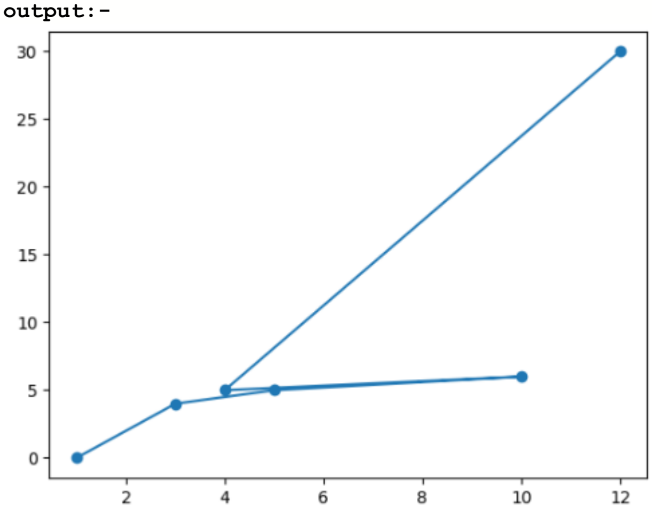

```
x = np.array([2,8])
y = np.array([4,15])
plt.plot(x,y,'o')
plt.show()
```

Output :-



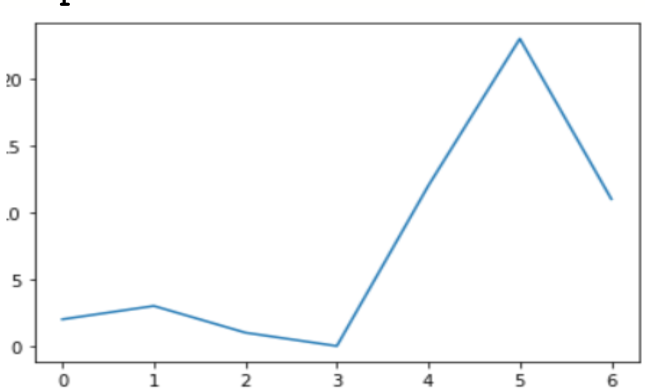

```
x = np.array([1,3,5,10,4,12])
y = np.array([0,4,5,6,5,30])
plt.plot(x,y,marker='o')
plt.show()
```

output:-




```
y = np.array([2,3,1,0,12,23,11])
plt.plot(y)
plt.show()
```

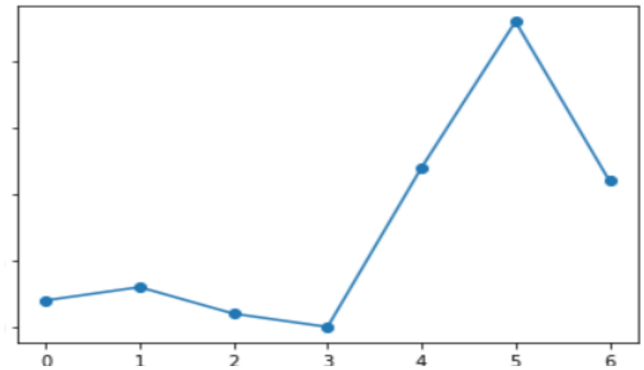
output:-




```
y = np.array([2,3,1,0,12,23,11])
```

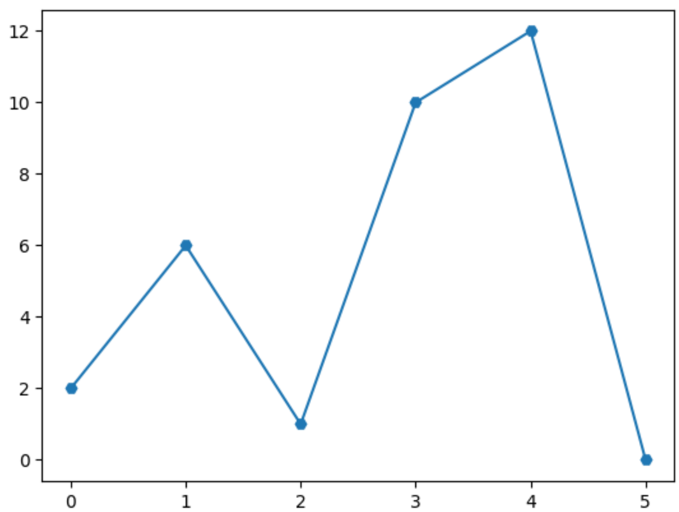
```
plt.plot(y,marker='o')
plt.show()
```

output:-

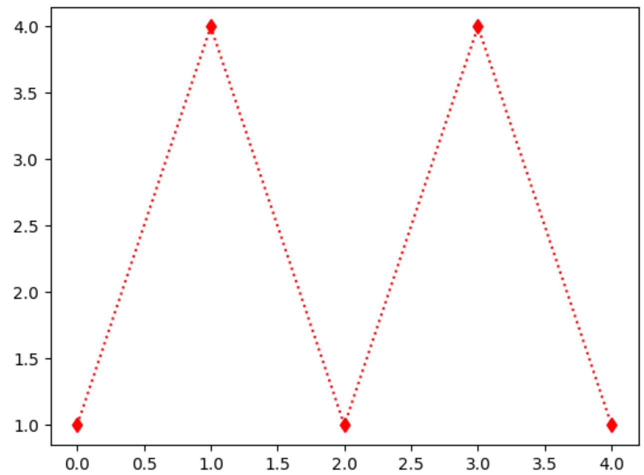



```
y = np.array([2,6,1,10,12,0])
plt.plot(y,marker='H')
plt.show()
```

Output:-

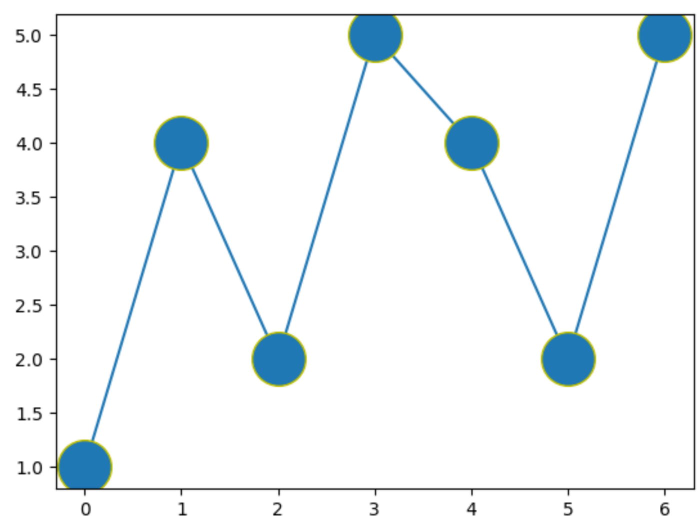



```
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([1,4,1,4,1])
plt.plot(ypoints, 'd:r')
plt.show()
```

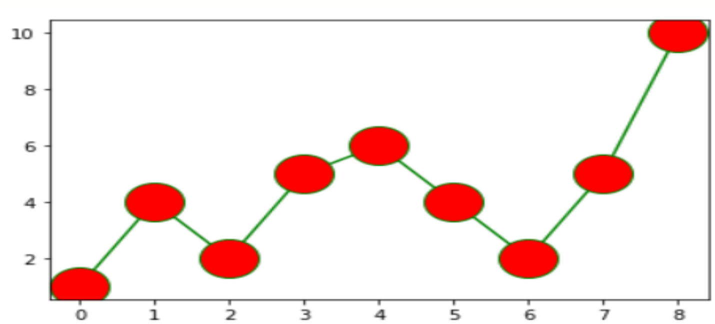



```
y = np.array([1,4,2,5,4,2,5])
plt.plot(y, marker='o', ms=30, mec = 'y')
plt.show()
```

output:-




```
y = np.array([1,4,2,5,6,4,2,5,10])
line, = plt.plot(y, marker='o', ms=30, mfc = 'r')
line.set_color("green")
```



(Assignment-03) Confusion Matrix

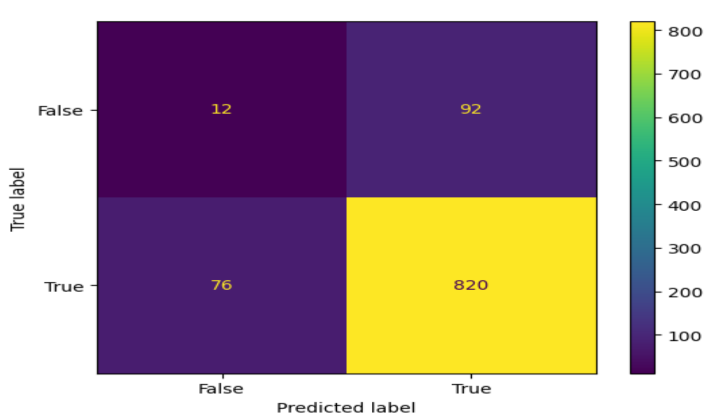
```
from sklearn.datasets import load_iris
import sklearn;
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
iris = load_iris();
x = iris.data
y = iris.target
x_train, x_test, y_train, y_test = train_test_split(x,y)
dtc = LinearRegression()
dtc.fit(x_train, y_train)
print(y_train)
print(y_test)
```

output:-

```
[1 0 2 1 1 1 2 0 1 0 2 0 1 1 1 1 0 0 1 1 1 2 2 0 0 0 2 2 1 2 2 1 0 0 0 1 1
 1 2 2 0 2 0 1 2 0 1 0 1 0 1 2 0 1 1 2 1 2 0 2 1 1 0 2 0 2 2 2 0 1 2 0 0 0
 0 1 1 0 0 0 0 2 0 2 2 1 2 0 0 1 2 1 2 1 2 0 1 1 0 0 0 2 0 0 2 1 2 2 2 0 2
 2]
[2 1 2 1 0 0 0 0 2 0 1 2 1 1 1 2 1 1 2 0 1 0 2 2 0 2 1 1 1 2 0 2 1 2 2 1 2
 0]
```

```
import matplotlib.pyplot as plt
import numpy
from sklearn import metrics
actual = numpy.random.binomial(1,.9,size = 1000)
predicted = numpy.random.binomial(1,.9,size = 1000)
confusion_matrix = metrics.confusion_matrix(actual, predicted)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix =
confusion_matrix, display_labels = [False, True])
cm_display.plot()
plt.show()
```

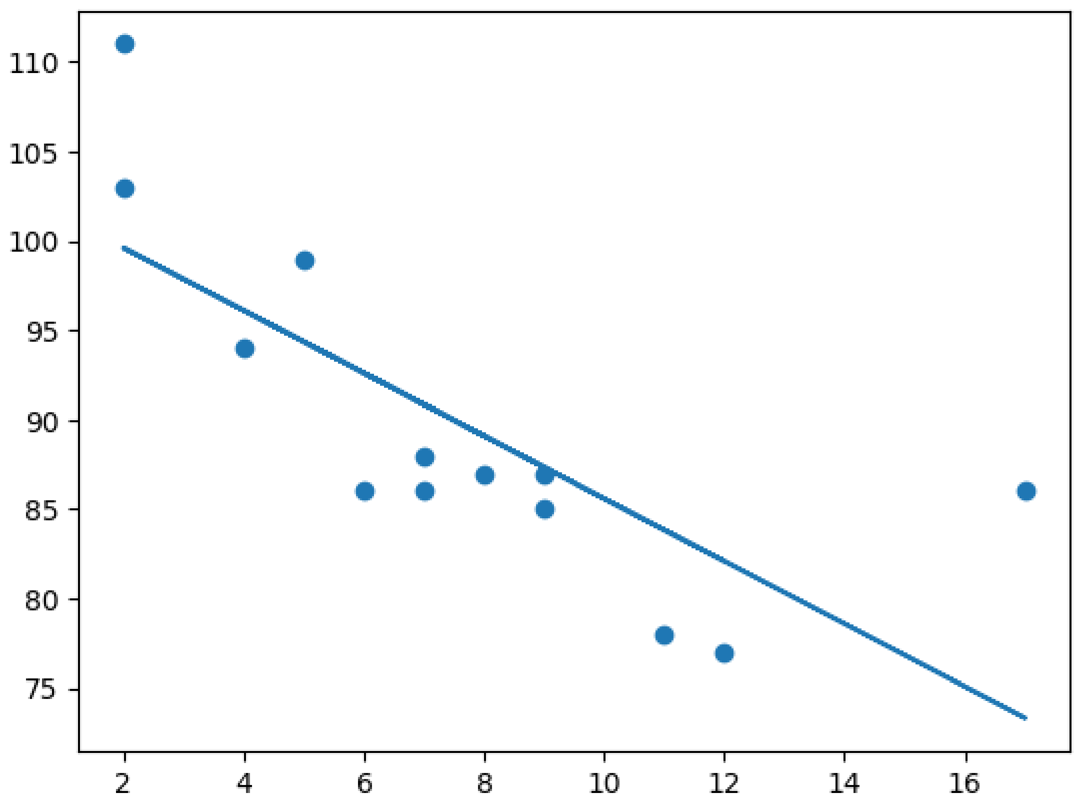
output:-



(Assignment-04)

Simple Linear Regression

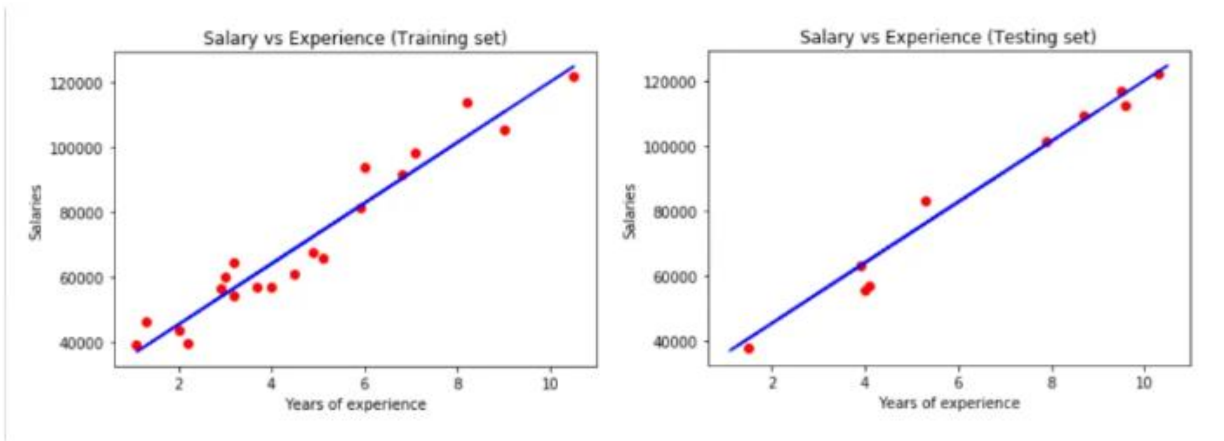
```
import matplotlib.pyplot as plt
from scipy import stats

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

slope, intercept, r, p, std_err = stats.linregress(x, y)

def myfunc(x):
  return slope * x + intercept

mymodel = list(map(myfunc, x))

plt.scatter(x, y)
plt.plot(x, mymodel)
plt.show()
```



```
# importing the dataset
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

dataset = pd.read_csv('Salary_Data.csv')
dataset.head()

# data preprocessing
X = dataset.iloc[:, :-1].values  #independent variable array
y = dataset.iloc[:,1].values  #dependent variable vector

# splitting the dataset
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)

# fitting the regression model
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train,y_train) #actually produces the linear eqn for the da

# predicting the test set results
y_pred = regressor.predict(X_test)
y_pred

y_test

# visualizing the results
#plot for the TRAIN

plt.scatter(X_train, y_train, color='red') # plotting the observation line
plt.plot(X_train, regressor.predict(X_train), color='blue') # plotting the
regression line
plt.title("Salary vs Experience (Training set)") # stating the title of the
graph

plt.xlabel("Years of experience") # adding the name of x-axis
plt.ylabel("Salaries") # adding the name of y-axis
plt.show() # specifies end of graph

#plot for the TEST

plt.scatter(X_test, y_test, color='red')
plt.plot(X_train, regressor.predict(X_train), color='blue') # plotting the
```

```
regression line
plt.title("Salary vs Experience (Testing set)")

plt.xlabel("Years of experience")
plt.ylabel("Salaries")
plt.show()
```

Output:-



Multiple Linear Regression

```
import pandas as pd
from sklearn import linear_model
import statsmodels.api as sm

data = {'year': [2017,2017,2017,2017,2017,2017,2017,2017,2017,20
17,2017,2017,2016,2016,2016,2016,2016,2016,2016,2016,2016,2016,2
016,2016],
        'month': [12,11,10,9,8,7,6,5,4,3,2,1,12,11,10,9,8,7,6,5,
4,3,2,1],
        'interest_rate': [2.75,2.5,2.5,2.5,2.5,2.5,2.5,2.25,2.25
,2.25,2,2,2,1.75,1.75,1.75,1.75,1.75,1.75,1.75,1.75,1.75,1.75,1.
75],
        'unemployment_rate': [5.3,5.3,5.3,5.3,5.4,5.6,5.5,5.5,5.
5,5.6,5.7,5.9,6,5.9,5.8,6.1,6.2,6.1,6.1,6.1,5.9,6.2,6.2,6.1],
        'index_price': [1464,1394,1357,1293,1256,1254,1234,1195,
1159,1167,1130,1075,1047,965,943,958,971,949,884,866,876,822,704
,719]
        }

df = pd.DataFrame(data)

x = df[['interest_rate','unemployment_rate']]
y = df['index_price']
```

```python
# with sklearn
regr = linear_model.LinearRegression()
regr.fit(x, y)

print('Intercept: \n', regr.intercept_)
print('Coefficients: \n', regr.coef_)

# with statsmodels
x = sm.add_constant(x) # adding a constant

model = sm.OLS(y, x).fit()
predictions = model.predict(x)

print_model = model.summary()
print(print_model)
```

output :-

```
Intercept: 
 1798.4039776258544
Coefficients: 
 [ 345.54008701 -250.14657137]
                            OLS Regression Results                            
==============================================================================
Dep. Variable:            index_price   R-squared:                       0.898
Model:                            OLS   Adj. R-squared:                  0.888
Method:                 Least Squares   F-statistic:                     92.07
Date:                Mon, 10 Apr 2023   Prob (F-statistic):           4.04e-11
Time:                        16:52:30   Log-Likelihood:                -134.61
No. Observations:                  24   AIC:                             275.2
Df Residuals:                      21   BIC:                             278.8
Df Model:                           2                                         
Covariance Type:            nonrobust                                         
=====================================================================================
                        coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------
const              1798.4040    899.248      2.000      0.059     -71.685    3668.493
interest_rate       345.5401    111.367      3.103      0.005     113.940     577.140
unemployment_rate  -250.1466    117.950     -2.121      0.046    -495.437      -4.856
==============================================================================
Omnibus:                        2.691   Durbin-Watson:                   0.530
Prob(Omnibus):                  0.260   Jarque-Bera (JB):                1.551
Skew:                          -0.612   Prob(JB):                        0.461
Kurtosis:                       3.226   Cond. No.                         394.
==============================================================================
```

(Assignment-05) Logistic Regression

```
import numpy
from sklearn import linear_model

X = numpy.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37,
 4.96, 4.52, 3.69, 5.88]).reshape(-1,1)
y = numpy.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])

logr = linear_model.LogisticRegression()
logr.fit(X,y)

def logit2prob(logr, X):
  log_odds = logr.coef_ * X + logr.intercept_
  odds = numpy.exp(log_odds)
  probability = odds / (1 + odds)
  return(probability)

print(logit2prob(logr, X))
```

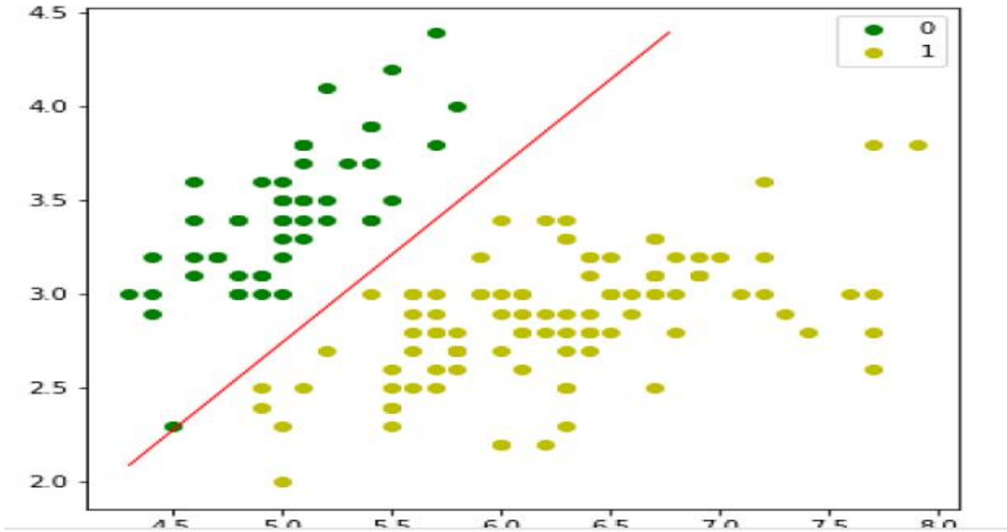
output:-

```
[[0.60749955]
 [0.19268876]
 [0.12775886]
 [0.00955221]
 [0.08038616]
 [0.07345637]
 [0.88362743]
 [0.77901378]
 [0.88924409]
 [0.81293497]
 [0.57719129]
 [0.96664243]]
```

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets
iris = datasets.load_iris()
X = iris.data[:, :2]
y = (iris.target != 0) * 1
plt.figure(figsize=(6, 6))
plt.scatter(X[y == 0][:, 0], X[y == 0][:, 1], color='g', label='0')
plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1], color='y', label='1')
plt.legend();
class LogisticRegression:
   def __init__(self, lr=0.01, num_iter=100000, fit_intercept=True, verbos
e=False):
```

```
        self.lr = lr
        self.num_iter = num_iter
        self.fit_intercept = fit_intercept
        self.verbose = verbose
    def __add_intercept(self, X):
        intercept = np.ones((X.shape[0], 1))
        return np.concatenate((intercept, X), axis=1)
    def __sigmoid(self, z):
        return 1 / (1 + np.exp(-z))
    def __loss(self, h, y):
        return (-y * np.log(h) - (1 - y) * np.log(1 - h)).mean()
    def fit(self, X, y):
        if self.fit_intercept:
            X = self.__add_intercept(X)

self.theta = np.zeros(X.shape[1])
    for i in range(self.num_iter):
        z = np.dot(X, self.theta)
        h = self.__sigmoid(z)
        gradient = np.dot(X.T, (h - y)) / y.size
        self.theta -= self.lr * gradient
        z = np.dot(X, self.theta)
        h = self.__sigmoid(z)
        loss = self.__loss(h, y)
        if(self.verbose ==True and i % 10000 == 0):
            print(f'loss: {loss} \t')


            def predict_prob(self, X)
if self.fit_intercept:
        X = self.__add_intercept(X)
    return self.__sigmoid(np.dot(X, self.theta))
def predict(self, X):
    return self.predict_prob(X).round()

    model = LogisticRegression(lr=0.1, num_iter=300000)
preds = model.predict(X)
(preds == y).mean()

plt.figure(figsize=(10, 6))
plt.scatter(X[y == 0][:, 0], X[y == 0][:, 1], color='g', label='0')
plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1], color='y', label='1')
plt.legend()
x1_min, x1_max = X[:,0].min(), X[:,0].max(),
x2_min, x2_max = X[:,1].min(), X[:,1].max(),
```

```
xx1, xx2 = np.meshgrid(np.linspace(x1_min, x1_max), np.linspace(x2_min, x2_max))
grid = np.c_[xx1.ravel(), xx2.ravel()]
probs = model.predict_prob(grid).reshape(xx1.shape)
plt.contour(xx1, xx2, probs, [0.5], linewidths=1, colors='red');
```




```
import sklearn
from sklearn import datasets
from sklearn import linear_model
from sklearn import metrics
from sklearn.model_selection import train_test_split
digits = datasets.load_digits()
X = digits.data
y = digits.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=1)
digreg = linear_model.LogisticRegression()
digreg.fit(X_train, y_train)
y_pred = digreg.predict(X_test)
print("Accuracy of Logistic Regression model is:",
metrics.accuracy_score(y_test, y_pred)*100)
```

Output

```
Accuracy of Logistic Regression model is: 95.6884561891516
```

(Assignment-06) Desition Tree

```
# Importing the required packages
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

# Function importing Dataset
def importdata():
  balance_data = pd.read_csv(
'https://archive.ics.uci.edu/ml/machine-learning-'+
'databases/balance-scale/balance-scale.data',
  sep= ',', header = None)

  # Printing the dataswet shape
  print ("Dataset Length: ", len(balance_data))
  print ("Dataset Shape: ", balance_data.shape)

  # Printing the dataset obseravtions
  print ("Dataset: ",balance_data.head())
  return balance_data

# Function to split the dataset
def splitdataset(balance_data):

  # Separating the target variable
  X = balance_data.values[:, 1:5]
  Y = balance_data.values[:, 0]

  # Splitting the dataset into train and test
  X_train, X_test, y_train, y_test = train_test_split(
  X, Y, test_size = 0.3, random_state = 100)

  return X, Y, X_train, X_test, y_train, y_test

# Function to perform training with giniIndex.
def train_using_gini(X_train, X_test, y_train):

  # Creating the classifier object
  clf_gini = DecisionTreeClassifier(criterion = "gini",
      random_state = 100,max_depth=3, min_samples_leaf=5)

  # Performing training
```

```
  clf_gini.fit(X_train, y_train)
  return clf_gini

# Function to perform training with entropy.
def tarin_using_entropy(X_train, X_test, y_train):

  # Decision tree with entropy
  clf_entropy = DecisionTreeClassifier(
      criterion = "entropy", random_state = 100,
      max_depth = 3, min_samples_leaf = 5)

  # Performing training
  clf_entropy.fit(X_train, y_train)
  return clf_entropy


# Function to make predictions
def prediction(X_test, clf_object):

  # Predicton on test with giniIndex
  y_pred = clf_object.predict(X_test)
  print("Predicted values:")
  print(y_pred)
  return y_pred

# Function to calculate accuracy
def cal_accuracy(y_test, y_pred):

  print("Confusion Matrix: ",
    confusion_matrix(y_test, y_pred))

  print ("Accuracy : ",
  accuracy_score(y_test,y_pred)*100)

  print("Report : ",
  classification_report(y_test, y_pred))

# Driver code
def main():

  # Building Phase
  data = importdata()
  X, Y, X_train, X_test, y_train, y_test = splitdataset(data)
  clf_gini = train_using_gini(X_train, X_test, y_train)
  clf_entropy = tarin_using_entropy(X_train, X_test, y_train)

  #  Operational Phase
  # print("Results Using Gini Index:")
```

```
  #  Prediction using gini
  # y_pred_gini = prediction(X_test, clf_gini)
  # cal_accuracy(y_test, y_pred_gini)

  print("Results Using Entropy:")
  # Prediction using entropy
  y_pred_entropy = prediction(X_test, clf_entropy)
  cal_accuracy(y_test, y_pred_entropy)


# Calling main function
if __name__=="__main__":
  main()
```

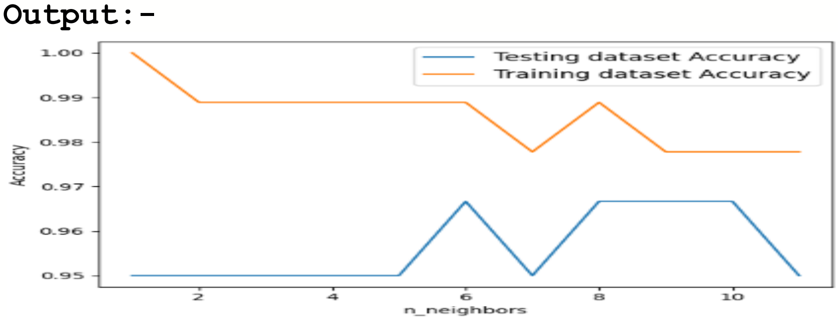
Output :-

```
Dataset Length:  625
Dataset Shape:  (625, 5)
Dataset:     0  1  2  3  4
0  B  1  1  1  1
1  R  1  1  1  2
2  R  1  1  1  3
3  R  1  1  1  4
4  R  1  1  1  5
Results Using Entropy:
Predicted values:
['R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L'
 'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L'
 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L'
 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'R' 'L'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'R' 'R' 'R' 'R' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L'
 'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R'
 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'L' 'R'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'R']
Confusion Matrix:  [[ 0  6  7]
 [ 0 63 22]
 [ 0 20 70]]
Accuracy :  70.74468085106383
Report :               precision    recall  f1-score   support

           B       0.00      0.00      0.00        13
           L       0.71      0.74      0.72        85
           R       0.71      0.78      0.74        90

    accuracy                           0.71       188
   macro avg       0.47      0.51      0.49       188
weighted avg       0.66      0.71      0.68       188
```

(Assignment-07) KNN Implementation

```
import numpy as nm;
import matplotlib.pyplot as plt;
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
import numpy as np
import matplotlib.pyplot as plt
irisData = load_iris()
# Create feature and target arrays
X = irisData.data
y = irisData.target
# Split into training and test set
X_train, X_test, y_train, y_test = train_test_split(
             X, y, test_size = 0.4, random_state=52)
neighbors = np.arange(1, 12)
train_accuracy = np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))
# Loop over K values
for i, k in enumerate(neighbors):
   # print(i + " " + k)
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    # Compute training and test data accuracy
    train_accuracy[i] = knn.score(X_train, y_train)
    test_accuracy[i] = knn.score(X_test, y_test)
# Generate plot
plt.plot(neighbors, test_accuracy, label = 'Testing dataset Accu
racy')
plt.plot(neighbors, train_accuracy, label = 'Training dataset Ac
curacy')

plt.legend()
plt.xlabel('n_neighbors')
plt.ylabel('Accuracy')
plt.show()
```

Output:-



(Assignment-08) Naïve Base Classifier

```
import numpy as np
import pandas as pd

# load iris dataset

from sklearn.datasets import load_iris
iris = load_iris()

# store the feature matrix (X) and response vector (y)
X = iris.data
y = iris.target

# splitting X and y into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_s
ize=0.4, random_state=1)

# training the model on training set
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X_train, y_train)

# making predictions on the testing set
y_pred = gnb.predict(X_test)

# comparing actual response values (y_test) with predicted respo
nse values (y_pred)
from sklearn import metrics

print("Gaussian Naive Bayes model accuracy(in %):", metrics.accu
racy_score(y_test, y_pred)*100)
```

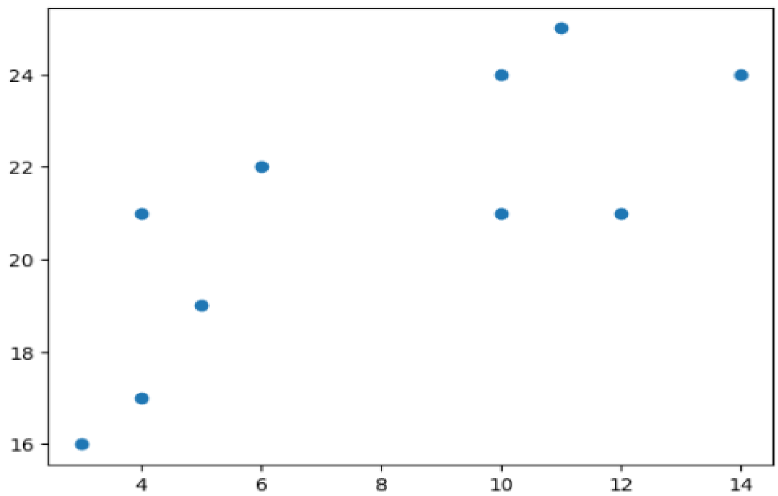
output :-

```
Gaussian Naive Bayes model accuracy(in %): 95.0
```

(Assignment-09) K-MeansClustering

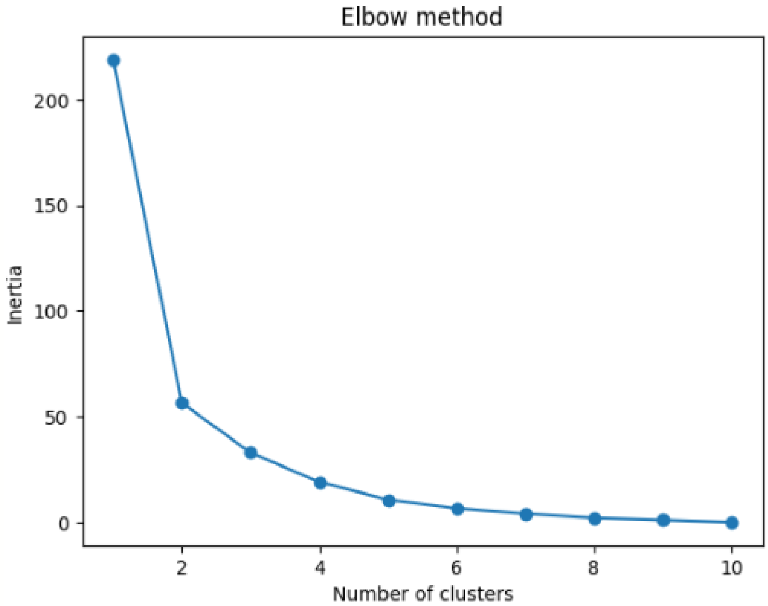
```
import matplotlib.pyplot as plt

x = [4, 5, 10, 4, 3, 11, 14 , 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]

plt.scatter(x, y)
plt.show()
```

Output:-



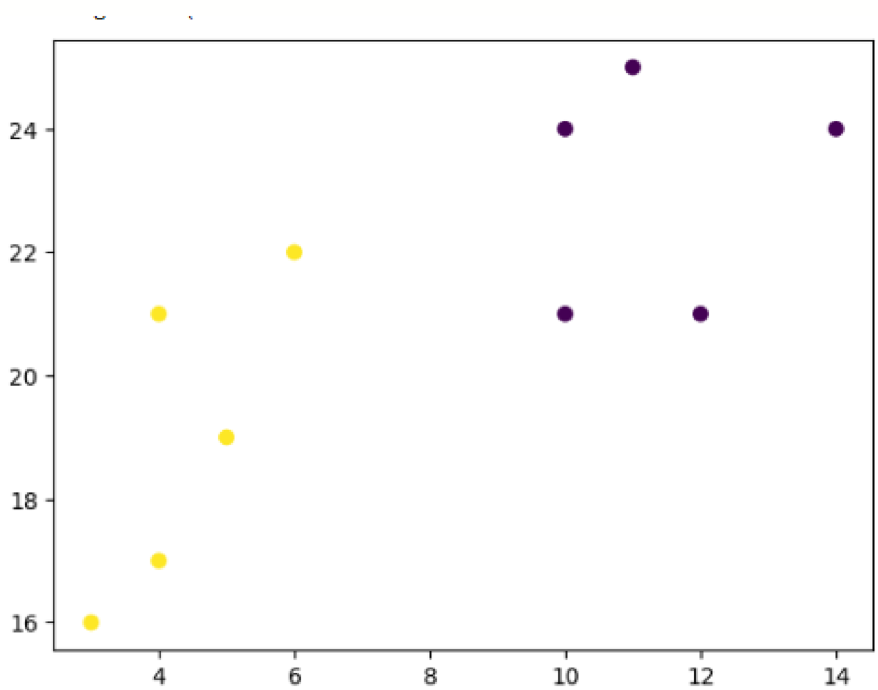

```
from sklearn.cluster import KMeans

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()
```

Output:-




```
kmeans = KMeans(n_clusters=2)
kmeans.fit(data)

plt.scatter(x, y, c=kmeans.labels_)
plt.show()
```

Output:-



(Assignment-10)
Support Vector Machine(SVM)

```
#Import scikit-learn dataset library
from sklearn import datasets
#Load dataset
cancer = datasets.load_breast_cancer()
# print the names of the 13 features
print("Features: ", cancer.feature_names)
# print the label type of cancer('malignant' 'benign')
print("Labels: ", cancer.target_names)
```

output:-

```
Features:  ['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
Labels:  ['malignant' 'benign']
```

```
# print data(feature)shape
cancer.data.shape
# print the cancer labels (0:malignant, 1:benign)
print(cancer.target)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 0 1 1 1 1 0 1 0 0 1 1 1 1 0 1 0 0
 1 0 1 0 0 1 1 1 0 0 1 0 0 0 1 1 1 0 1 1 0 0 1 1 1 0 0 1 1 1 1 0 1 1 0 1 1
 1 1 1 1 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 1 1 0 1
 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 1 1 1 1 0 1 1 0 0 0 1 0
 1 0 1 1 1 0 1 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 1 1 0 0 1 1
 1 0 1 1 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 1 1 1 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 1 0 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1
 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 0 0 0 1 1
 1 1 0 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0
 0 1 0 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1
 1 0 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 1 1 1 1 0 1 1
 0 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1
 1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1 0 0 1 0 1 0 1 1 1 1 1 0 1 1 0 1 0 1 0 0
 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 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 0 0 0 0 0 0 1]
```

```
# Import train_test_split function
from sklearn.model_selection import train_test_split

# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(cancer.data,
 cancer.target, test_size=0.3,random_state=109) # 70% training a
nd 30% test
#Import svm model
from sklearn import svm

#Create a svm Classifier
clf = svm.SVC(kernel='linear') # Linear Kernel

#Train the model using the training sets
clf.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics

# Model Accuracy: how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
# Model Precision: what percentage of positive tuples are labele
d as such?
print("Precision:",metrics.precision_score(y_test, y_pred))

# Model Recall: what percentage of positive tuples are labelled
as such?
print("Recall:",metrics.recall_score(y_test, y_pred))
```

Output:-

```
Accuracy: 0.9649122807017544
Precision: 0.9811320754716981
Recall: 0.9629629629629629
```