MACHINE LEARNING – 2CS501

PRACTICAL 3

Name: Bhanderi Mihir

Roll No.: 19BCE023

Batch No.: A-1

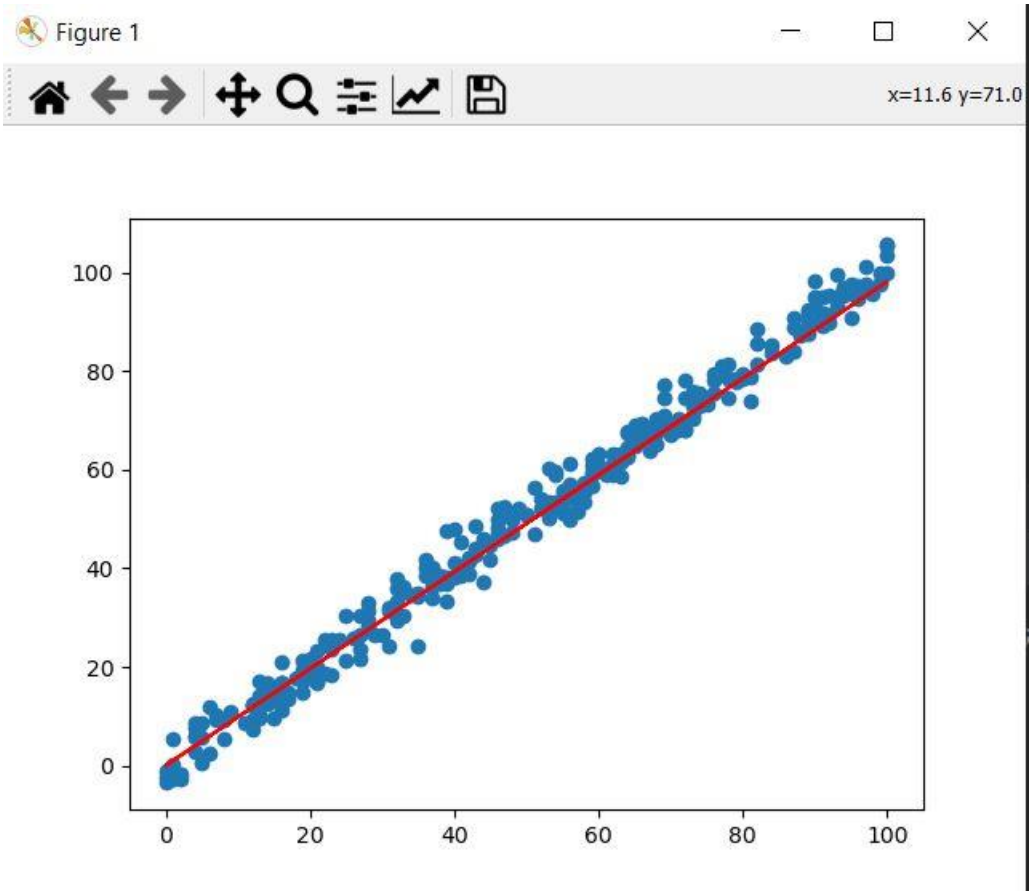
1) SIMPLE LINEAR REGRESSION

Code:

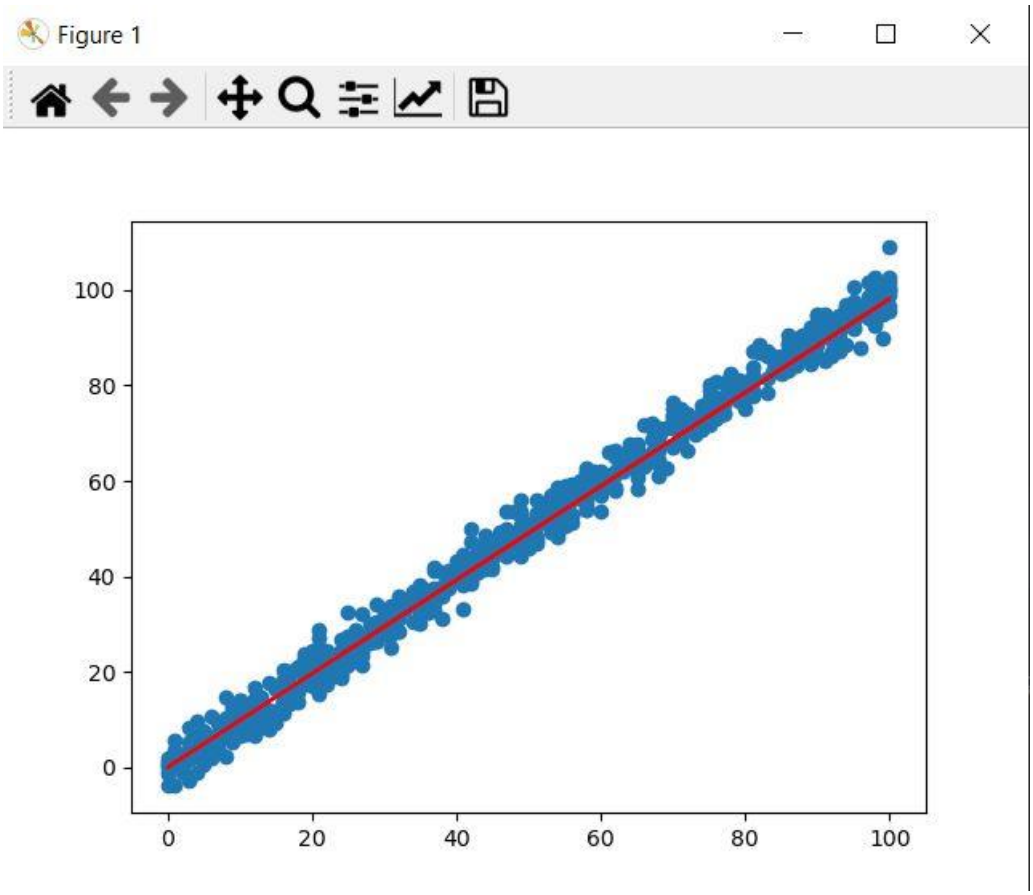
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import  metrics


def Plot_CF(theta1_Val, Cf):
    plt.plot(theta1_Val, Cf)
    plt.show()


def Plot_TrainSet(x_train, y_train, predytrain):
    plt.scatter(x_train[:, 1], y_train)
    plt.plot(x_train[:, 1], predytrain, color='red')
    plt.show()


def Plot_TestSet(x_test, y_test, predytest):
    plt.scatter(x_test[:, 1], y_test)
    plt.plot(x_test[:, 1], predytest, color='red')
    plt.show()


if __name__ == '__main__':
    df1 = pd.read_csv("Train.csv")
    df2 = pd.read_csv("Test.csv")
    x_train = np.ones((699, 2))
    x_train[:, 1] = df1['x']
    y_train = np.array(df1['y'])

    x_test = np.ones((300, 2))
    x_test[:, 1] = df2['x']
    y_test = np.array(df2['y'])

    theta = (0, 0)
    alpha = 0.0001
    nitterations = 10
    m = x_train.shape[0]
    n = x_train.shape[1]
    theta1_Val = []
    Cf = list()
```

```
    for i in range(nitterations):
        update = np.zeros(x_train.shape[1])
        ypred = np.dot(x_train, theta)
        error = ypred - y_train
        for j in range(n):
            update[j] = np.sum(np.dot(error, x_train[:, j]))

        theta = theta - (1 / m) * alpha * update
        theta1_Val.append(theta[1])
        # Cost-Function
        sum1 = 0
        for k in range(m):
            sum1 += (theta[0] - y_train[k] + theta[1] * x_train[k, 1]) ** 2

        Cf.append(sum1 * (1 / (2 * m)))

    print("Theta : ", theta)

    print("CF : ", Cf)

    prediction = np.dot(theta, x_test.T)
    print("MAE: ", metrics.mean_absolute_error(y_true=y_test, y_pred=prediction))
    print("MSE: ", metrics.mean_squared_error(y_true=y_test, y_pred=prediction))

    predytrain = theta[0] + theta[1] * x_train[:, 1]
    predytest = theta[0] + theta[1] * x_test[:, 1]
    print("\nPlotting of Graphs : ")
    print("""    1. Cost Function
    2. Training Set Model
    3. Test Set Model
    4. EXIT""")
    while True:
        n = int(input("Enter What to Plot : "))
        if n == 1:
            Plot_CF(theta1_Val, Cf)
        elif n == 2:
            Plot_TrainSet(x_train, y_train, predytrain)
        elif n == 3:
            Plot_TestSet(x_test, y_test, predytest)
        else:
            exit("Thanks for Using")


"""
OUTPUT:

Theta :  [0.01467635 0.98166009]
CF :  [743.1010875141254, 331.8612360146683, 149.4175775553528, 68.47773575109728,
32.56934599437108, 16.63884219635945, 9.57138653986051, 6.435959653629742,
5.044949586914297, 4.427837733564734]
MAE:  2.6724795357316324
MSE:  11.43993794509001

Plotting of Graphs :
    1. Cost Function
    2. Training Set Model
    3. Test Set Model
    4. EXIT
Enter What to Plot : 1
Enter What to Plot : 2
Enter What to Plot : 3
Enter What to Plot : 4
```

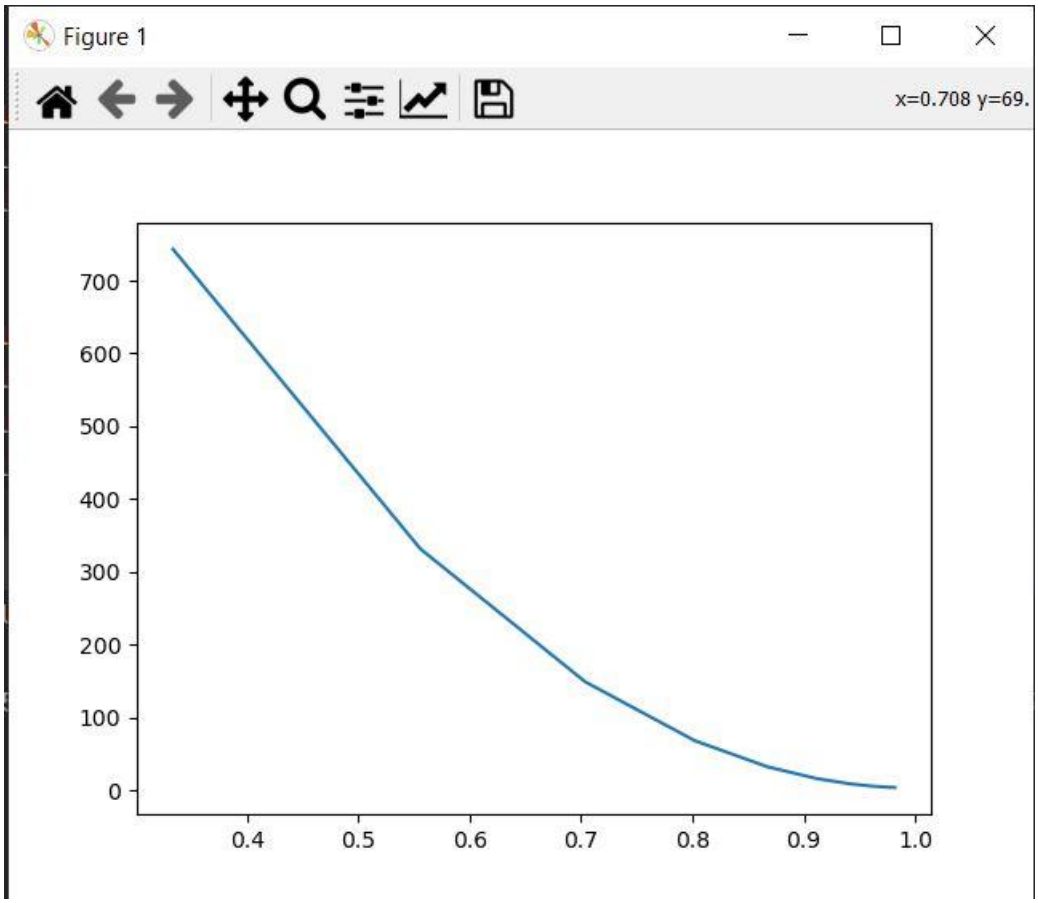
```
Thanks for Using

"""
```



Graph of Test Case and Predicted value



Graph of Training Case and Predicted value



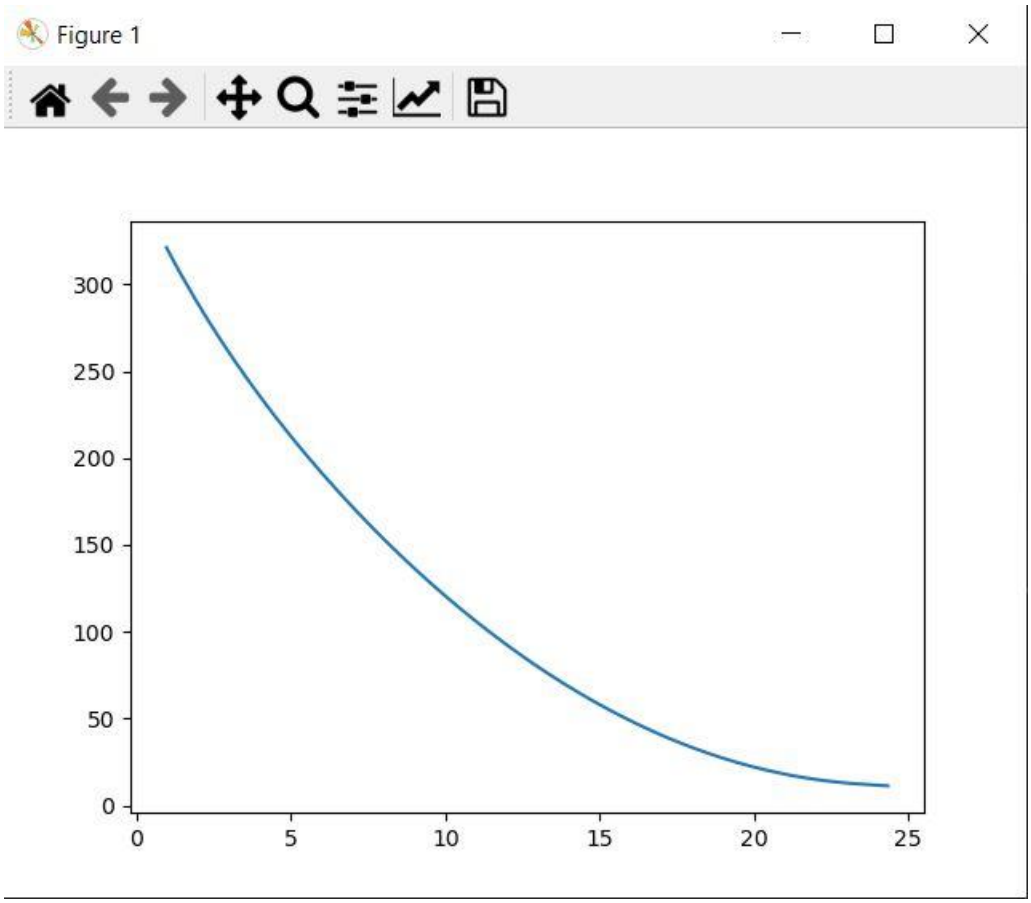
Graph of Cost Function

2) Multiple Linear Regression (Gradient Descent)

Code:

```
# Multiple Linear Regression Gradient Descent

import numpy as np
import matplotlib.pyplot as plt
from sklearn import metrics, datasets
from sklearn.preprocessing import StandardScaler


def Plot_CF(theta1_Val, Cf):
    plt.plot(theta1_Val, Cf)
    plt.show()


if __name__ == '__main__':
    x, y = datasets.load_boston(return_X_y=True)

    x_train_temp1 = x[0:400, :]
    x_train = np.ones((x_train_temp1.shape[0], x_train_temp1.shape[1] + 1))
    x_train[:, 1:] = x_train_temp1
    y_train = y[:400]
    x_test_temp1 = x[400:506, :]
    x_test = np.ones((x_test_temp1.shape[0], x_test_temp1.shape[1] + 1))
    x_test[:, 1:] = x_test_temp1
    y_test = y[400:506]

    m = x_train.shape[0]
    n = x_train.shape[1]

    scaler = StandardScaler()
    scaler.fit(x_train[:,1:])
    x_train[:,1:] = scaler.transform(x_train[:,1:])
    x_test[:,1:] = scaler.transform(x_test[:, 1:])

    theta = np.random.uniform(0,1,n)
    alpha = 0.01
    nitterations = 1000

    theta1_Val = []
    Cf = list()

    for i in range(nitterations):
        update = np.zeros(x_train.shape[1])
        ypred = np.dot(x_train, theta)
        error = ypred - y_train
        for j in range(n):
            update[j] = np.sum(np.dot(error, x_train[:, j]))

        theta = theta - (1 / m) * alpha * update
        theta1_Val.append(theta[0])

        # Cost-Function
        sum1 = 0
        for k in range(m):
            sum2 = 0
            for i in range(n):
                sum2 += theta[i] * x_train[k, i]
            sum1 += (sum2 - y_train[k]) ** 2
```

```
        Cf.append(sum1 * (1 / (2 * m)))

    print("Theta : ", theta)
    print(("Theta[1] : ", len(theta1_Val)))

    print("CF : ", len(Cf))
    prediction = np.dot(theta, x_test.T)
    print("MAE: ", metrics.mean_absolute_error(y_true=y_test, y_pred=prediction))
    print("MSE: ", metrics.mean_squared_error(y_true=y_test, y_pred=prediction))


    print("\nPlotting of Graphs : ")
    print("""    1. Cost Function
    2. EXIT""")
    while True:
        n = int(input("Enter What to Plot : "))
        if n == 1:
            Plot_CF(theta1_Val, Cf)
        else:
            exit("Thanks for Using")

"""
OUTPUT:

Theta :  [ 2.43334807e+01 -1.00515791e+00  9.15022551e-01  1.13357521e-02
  5.61123609e-01 -1.20836710e+00  3.73442039e+00 -2.73681723e-02
 -2.54183088e+00  2.06095317e+00 -1.08449809e+00 -1.65437094e+00
  5.40425479e-02 -3.47470775e+00]
('Theta[1] : ', 1000)
CF :  1000
MAE:  4.771415037813911
MSE:  32.86957950538916

Plotting of Graphs :
    1. Cost Function
    2. EXIT
Enter What to Plot : 1
Enter What to Plot : 2
Thanks for Using

"""
```



Graph of Cost Function

3) Multiple Linear Regression (Normal Equation)

Code:

```
import numpy as np
from sklearn import datasets,metrics
from numpy.linalg import inv, pinv, LinAlgError
from sklearn.preprocessing import StandardScaler

if __name__ == '__main__':
    X, y = datasets.load_boston(return_X_y=True)

    x_train_temp1 = X[0:400, :]
    x_train = np.ones((x_train_temp1.shape[0], x_train_temp1.shape[1] + 1))
    x_train[:, 1:] = x_train_temp1

    y_train = y[:400]
    x_test_temp1 = X[400:506, :]
    x_test = np.ones((x_test_temp1.shape[0], x_test_temp1.shape[1] + 1))
    x_test[:, 1:] = x_test_temp1

    y_test = y[400:506]

    scaler = StandardScaler()
    scaler.fit(x_train[:, 1:])
    x_train[:, 1:] = scaler.transform(x_train[:, 1:])
    x_test[:, 1:] = scaler.transform(x_test[:, 1:])

    theta = np.zeros(x_train.shape[1])
    try:
        XTXi = np.linalg.inv(np.dot(x_train.T, x_train))
    except np.linalg.LinAlgError:
        XTXi = np.linalg.pinv(np.dot(x_train.T, x_train))

    XTy = np.dot(x_train.T, y_train)
    theta = np.dot(XTXi, XTy)

    print("Theta : ",theta)

    prediction = np.dot(theta, x_test.T)

    print("MAE: ", metrics.mean_absolute_error(y_true=y_test, y_pred=prediction))
    print("MSE: ", metrics.mean_squared_error(y_true=y_test, y_pred=prediction))

"""
OUTPUT:

Theta :  [24.3345     -1.14370921  1.12191092  0.35913222  0.48497247 -1.7061696
  3.58169796  0.07554815 -2.8156326   3.05189603 -1.97502535 -1.7937352
 -0.05252128 -3.50239563]
MAE:  5.142232214465314
MSE:  37.893778599602236

"""
```