

General Sir John Kotelawala Defense University

Department of Electrical, Electronics & Telecommunication Engineering

Machine Learning

ET 4103

Assignment - 01

Index No : D/ENG/22/0120/ET

Name : M. A. E. Wijesuriya

Intake : 39

Submission Date : 13/06/2025

Q1. Utilize the given Jupyter notebook for Linear Regression with a single variable. Comment on the code and the output of the program, explaining utilized Machine Learning concepts where necessary

Code with Explanation:

(text in *italics,* along with any graphs or tables,are the output of the preceding code segment)

# File Location: The file we want to access is currently placed in the current working directory of Python.

import os

from google.colab import drive

drive.mount('/content/drive') # Grants Colab access to Google Drive in order to retrieve the data files

%cd "/content/drive/MyDrive/ML\_files"

*Mounted at /content/drive*

*/content/drive/MyDrive/ML\_files*

# Import the required Libraries

import pandas as pd

import numpy as np

import sklearn

import matplotlib.pyplot as plt

path = 'ex1data1.txt'

data\_path = path

data = pd.read\_csv(path, header=None, names = ["x1", "y"])

data.head() # Prints the first five rows of the data

|  |  |  |
| --- | --- | --- |
|  | *x1* | *y* |
| *0* | *6.1101* | *17.5920* |
| *1* | *5.5277* | *9.1302* |
| *2* | *8.5186* | *13.6620* |
| *3* | *7.0032* | *11.8540* |
| *4* | *5.8598* | *6.8233* |

data.shape # Returns the shape of the data in the form (rows, columns)

*(97, 3)*

x1 = data['x1'] # Extracts x1 values into list

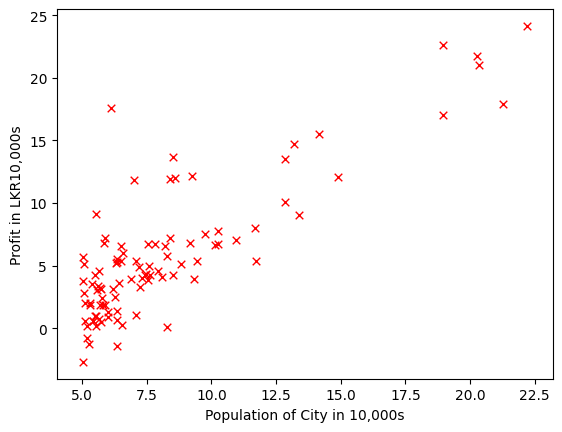
y = data['y'] # Extracts y values into list

plt.scatter(x1,y,s=30,c='r',marker='x',linewidths=1) # Prepares a scatter plot of the data

plt.xlim(min(data['x1']-1),max(data['x1']+1)) # Sets limits for the extent of the graph

plt.xlabel('Population of City in 10,000s') # Labels X axis

plt.ylabel('Profit in LKR10,000s'); # Labels Y axis



# Cost Function

m = data.shape[0]

def Cost(x,y,theta):

    J = 0

    #Hypothesis

    h = x.dot(theta)

    #Cost Function

    J = 1/(2\*m)\*np.sum(np.square(h-y))

    return J

data.insert(loc=0,column='x0',value=np.ones(m))

data.head()

|  |  |  |  |
| --- | --- | --- | --- |
|  | x0 | x1 | y |
| 0 | 1.0 | 6.1101 | 17.5920 |
| 1 | 1.0 | 5.5277 | 9.1302 |
| 2 | 1.0 | 8.5186 | 13.6620 |
| 3 | 1.0 | 7.0032 | 11.8540 |
| 4 | 1.0 | 5.8598 | 6.8233 |

x = data[data.columns[0:data.shape[1]-1]]

n = data.shape[1]-1

y = data[data.columns[n:n+1]]

# conversion to an np.array

x = x.values

y = y.values

m = y.shape[0]

theta\_initial = np.array([[0],[0]])

Cost(x,y,theta\_initial) # calculates the cost function for x, y using theta\_initial

np.float64(32.072733877455676)

# Gradient Descent implementation

def gradientDescent(x, y, theta, alpha, num\_iters):

    J\_history = np.zeros(num\_iters)

    for iter in np.arange(num\_iters):

        h = x.dot(theta)

        theta = theta - alpha\*(1/m)\*(x.T.dot(h-y))

        J\_history[iter] = Cost(x, y, theta)

    return(theta, J\_history)

theta\_initial = np.array([[0],[0]])

alpha = 0.01 # Sets the learning Rate

iterations = 1500 # Sets the number of iterations

theta, cost\_history = gradientDescent(x,y,theta\_initial,alpha,iterations)