General Sir John Kotelawala Defense University

Department of Electrical, Electronics & Telecommunication Engineering

Machine Learning

ET 4103

Assignment – 01

Linear Regression with a Single Variable

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

Name : M. A. E. Wijesuriya

Intake : 39

Submission Date : 20/06/2025

Q1. Utilize the given Jupyter notebook[1] 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
```

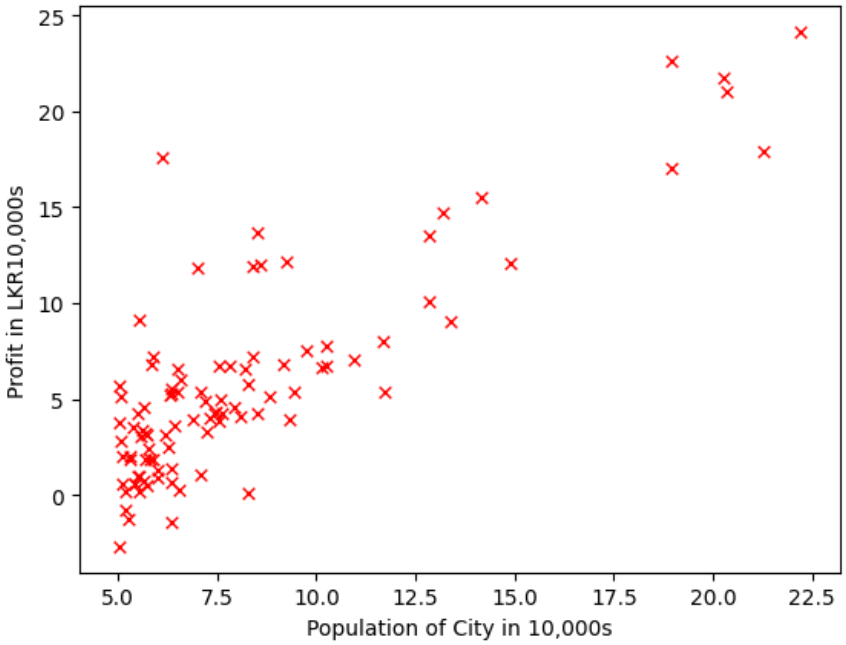
```
# Import the required Libraries

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
```

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()
```

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)

theta # Prints the calculated theta value
```

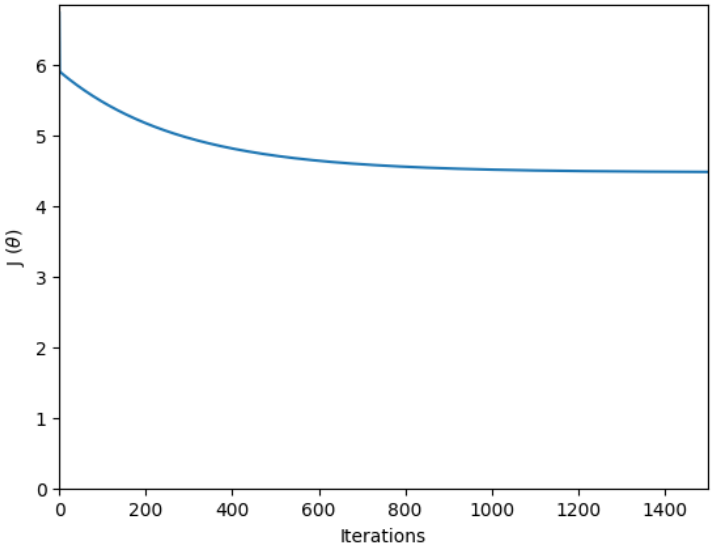
```
array([[-3.63029144],
       [ 1.16636235]])
```

```
cost_history # Prints the calculated cost history value
```

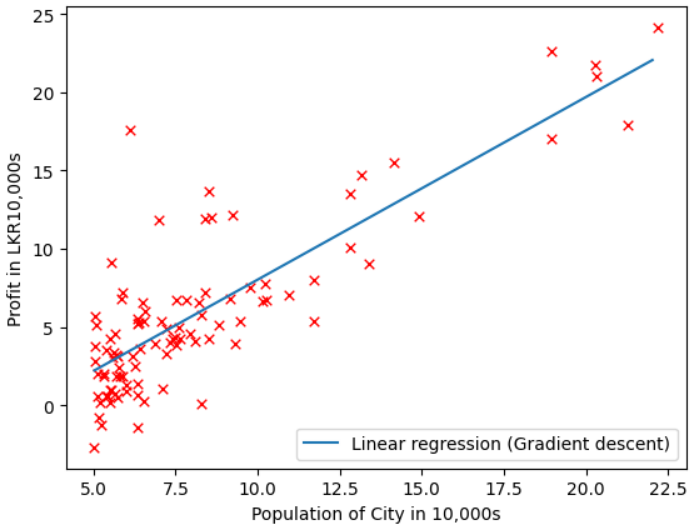
```
array([6.73719046, 5.93159357, 5.90115471, ..., 4.48343473, 4.48341145,
       4.48338826])
```

```
# Plot of cost_history vs. iterations
plt.plot(cost_history)
plt.ylabel('J' + ' (' + r'$\theta$' +')')
# or plt.ylabel('J' + ' (\u0398)' )
plt.xlabel('Iterations')
plt.ylim(ymin = 0)
plt.xlim(0,iterations)
```

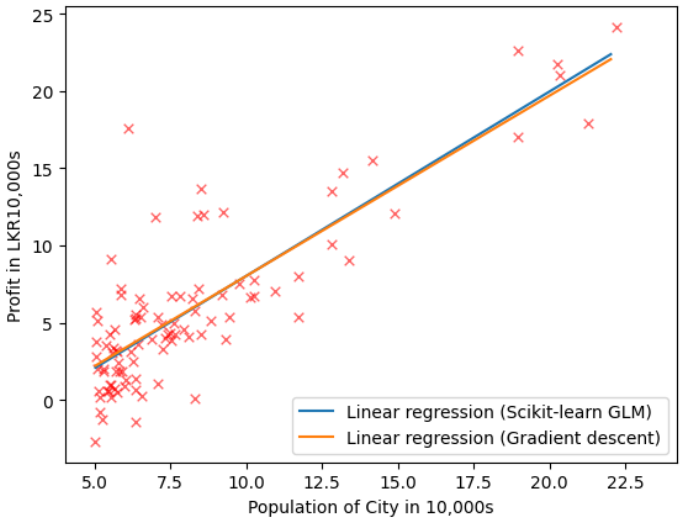
(0.0, 1500.0)




```
# Plots a scatter plot of the data along with the model predicted
through linear regression
x_range = np.arange(min(data['x1']),max(data['x1']))
y_range = theta[0] + theta[1]*x_range
plt.scatter(x1,y,s=30,c='r',marker='x',linewidths=1)
plt.plot(x_range,y_range, label='Linear regression (Gradient descent)')
plt.xlabel('Population of City in 10,000s')
plt.ylabel('Profit in LKR10,000s')
plt.legend(loc=4);
```

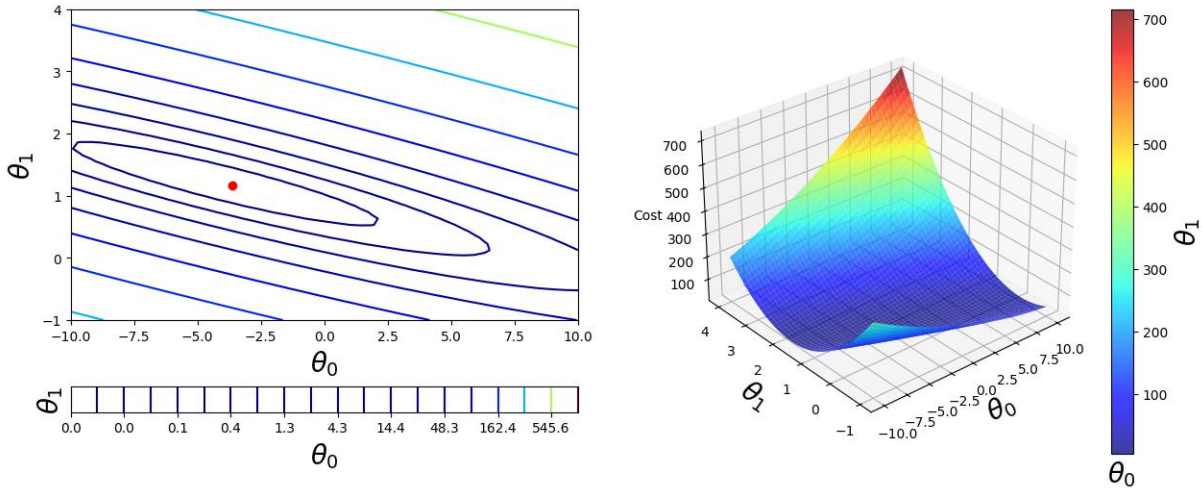


```
# Sklearn Implementation

from sklearn.linear_model import LinearRegression
linreg = LinearRegression()
linreg.fit(x[:,1].reshape(m,1),y)

# This plot compares directly the theoretical implementation of Linear
Regression with the Built-in Scikit version
plt.plot(x_range, (linreg.intercept_ + linreg.coef_*x_range).ravel(),
label='Linear regression (Scikit-learn GLM)')

plt.scatter(x1,y,s=30,c='r',marker='x',linewidths=1, alpha=0.6)
plt.plot(x_range,y_range, label='Linear regression (Gradient descent)')

plt.xlim(min(data['x1']-1),max(data['x1']+2))
plt.xlabel('Population of City in 10,000s')
plt.ylabel('Profit in LKR10,000s')
plt.legend(loc=4);
```




```
# Predict profit for a city with population of 35000 and 70000
print(theta.T.dot([1, 3.5])*10000)
print(theta.T.dot([1, 7])*10000)
[4519.7678677]
[45342.45012945]

# Create grid coordinates for plotting
B0 = np.linspace(-10, 10, 50)
B1 = np.linspace(-1, 4, 50)
xx, yy = np.meshgrid(B0, B1, indexing='xy')
Z = np.zeros((B0.size,B1.size))

# Calculate Z-values (Cost) based on grid of coefficients
for (i,j),v in np.ndenumerate(Z):
```

```
    Z[i,j] = Cost(x,y, theta=[[xx[i,j]], [yy[i,j]]])

from mpl_toolkits.mplot3d import axes3d

fig = plt.figure(figsize=(15,6))
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122, projection='3d')

# Left plot (Contour Plot of the Cost Function)
CS = ax1.contour(xx, yy, Z, np.logspace(-2, 3, 20), cmap=plt.cm.jet)
ax1.scatter(theta[0],theta[1], c='r')
plt.colorbar(CS,orientation='horizontal')

# Right plot (3D surface plot of Gradient Descent)
CS2 = ax2.plot_surface(xx, yy, Z, rstride=1, cstride=1, alpha=0.75,
cmap=plt.cm.jet)
ax2.set_zlabel('Cost')
ax2.set_zlim(Z.min(),Z.max())
ax2.view_init(elev=30, azim=230)
plt.colorbar(CS2, orientation='vertical')

# settings common to both plots
for ax in fig.axes:
    ax.set_xlabel(r'$\theta_0$', fontsize=20)
    ax.set_ylabel(r'$\theta_1$', fontsize=20)
```



The python program given above shows an implementation of Linear Regression with a single variable, in order to calculate the profit of a city, given its population.

First the cost function and gradient descent algorithm is calculated using mathematical first principles, and then the linear regression model is predicted using these functions. Next, this approach is compared to the LinearRegression function contained in the SciKit Learn python library. From the output of the comparison graph we can see that the regression models

obtained from first principles is almost identical to the on obtained in the SciKit Learn library.

Additionally, two values are predicted using the theta values calculated from first principles: The profit of a city with population 35,000, and the profit of a city with population 70,000. By looking at the predicted values for those two inputs, we can see that they are in line with the rest of the model.

The final code segment of the program generates two graphs that allow us to visualize the actual process behind calculation of a Linear Regression model. The graph on the left plots the contour plot for the cost function, using a colorbar to show values of the contours. The graph on the right shows a 3D surface plot, giving us a clear picture of how the gradient descent function iterates and descends to the minimum point.

Code Source:

[1]
https://colab.research.google.com/drive/1tckQFBK5_oQaivjAvZOkvk488jiEYrjY