AIR UNIVERSITY

Department of Electrical and Computer Engineering

AI for Engineers Lab

Lab #10: Vectorized Linear Regression

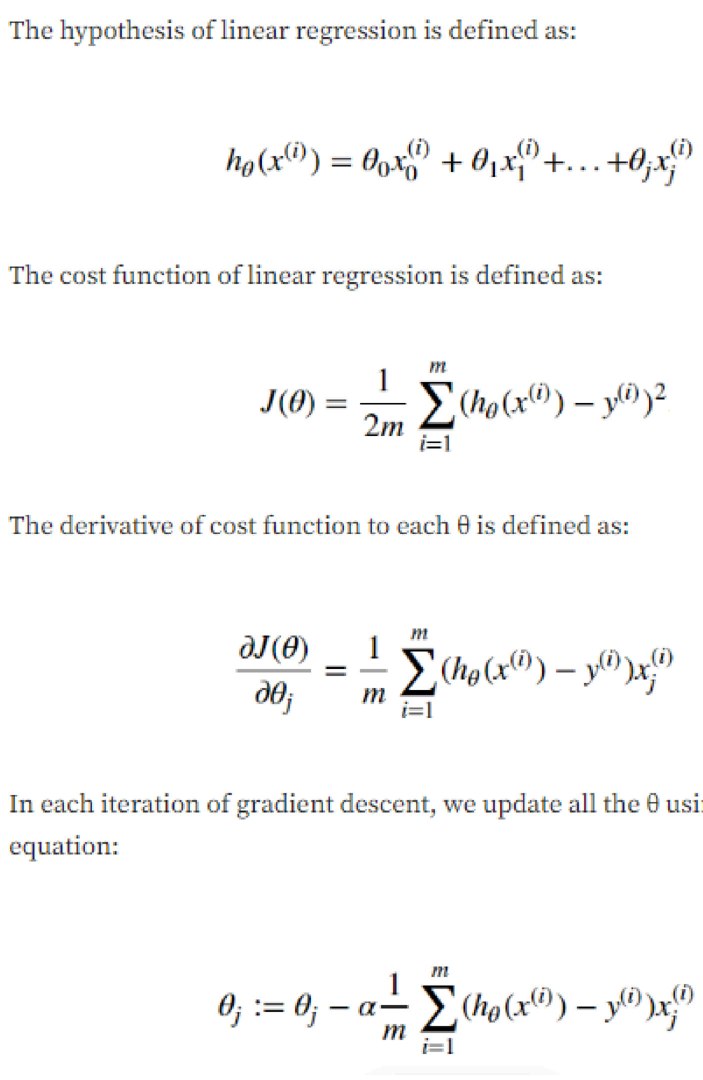
Student Name: Ahmed Khalid

Roll No: 200148

Instructor: M Awais Manzoor

Vectorized Linear Regression

```
from IPython import display
display.Image("VLR.PNG")
```



The hypothesis of linear regression is defined as:

$$h_\theta(x^{(i)}) = \theta_0 x_0^{(i)} + \theta_1 x_1^{(i)} + \ldots + \theta_j x_j^{(i)}$$

The cost function of linear regression is defined as:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

The derivative of cost function to each θ is defined as:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

In each iteration of gradient descent, we update all the θ using the following equation:

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Univariate

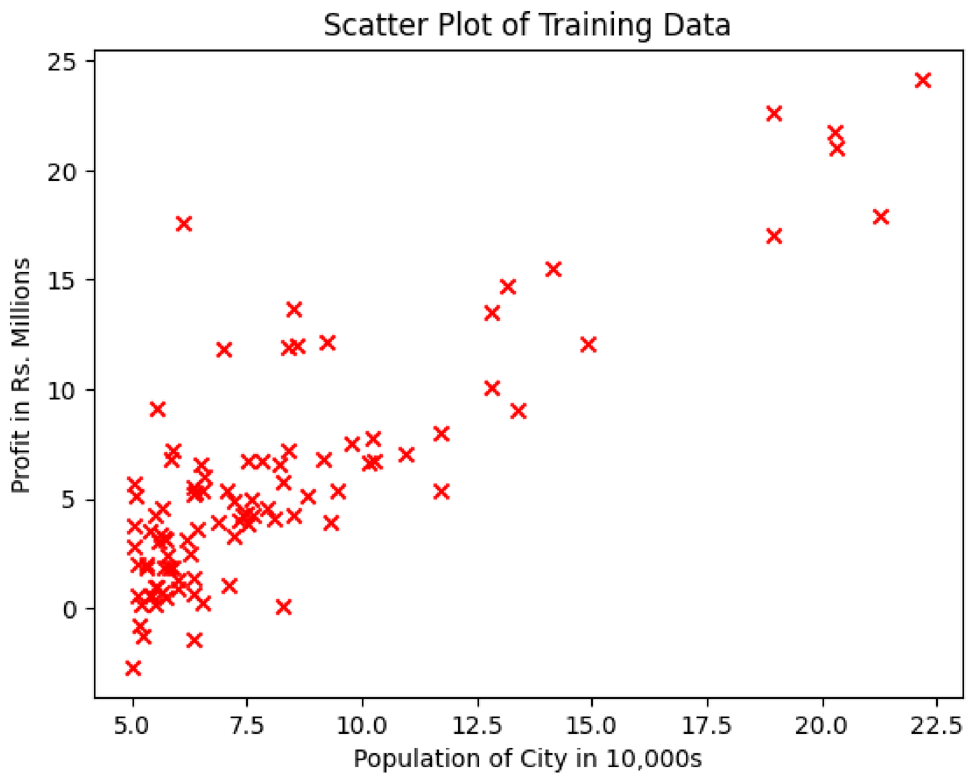
```
data = pd.read_csv('ex1data1.txt', sep=",", header=None)       #store the given data in pandas data fr
data.columns = ["population", "profit"]         #data loaded in "population" and "profit" variable
data.head(5)            #checking if the data is loaded correctly
```

	population	profit
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

Next steps: View recommended plots

```
plt.title("Scatter Plot of Training Data")
plt.xlabel("Population of City in 10,000s")
plt.ylabel("Profit in Rs. Millions")
plt.scatter(data.population,data.profit, color ='red', marker='x')  #Plotting the data
```

```
<matplotlib.collections.PathCollection at 0x7c990a77f040>
```



```
# Extract X and y
X = data[['population']].values  # Feature matrix
y = data['profit'].values.reshape(-1, 1)  # Target variable

print(f'X shape: {X.shape}')
print(f'y shape: {y.shape}')
```

```
X shape: (97, 1)
y shape: (97, 1)
```

```
# Add a column of ones to X to account for the bias term
X_with_bias = np.hstack((np.ones((X.shape[0], 1)), X))

print(X_with_bias[:, :])
```

```
[ 1.      6.3534]
[ 1.      5.4069]
[ 1.      6.8825]
[ 1.     11.708 ]
[ 1.      5.7737]
[ 1.      7.8247]
[ 1.      7.0931]
[ 1.      5.0702]
[ 1.      5.8014]
[ 1.     11.7   ]
[ 1.      5.5416]
[ 1.      7.5402]
[ 1.      5.3077]
```

```
 [ 1.      7.6366]
 [ 1.      5.8707]
 [ 1.      5.3054]
 [ 1.      8.2934]
 [ 1.     13.394 ]
 [ 1.      5.4369]]
```

```
# Initialize Theta (parameters)
theta = np.random.rand(2, 1)
print(f'Theta shape: {theta.shape}')
print('Initial Theta:', theta)
```

```
Theta shape: (2, 1)
Initial Theta: [[0.15848366]
 [0.42076001]]
```

Hypothesis

```
from IPython import display
display.Image("hypothesis.png")
```

$$h_\theta(x) = X\theta = \begin{bmatrix} x_0^{(1)} & x_1^{(1)} & \dots & x_n^{(1)} \\ x_0^{(2)} & x_1^{(2)} & & x_n^{(2)} \\ \vdots & \vdots & \dots & \vdots \\ x_0^{(m)} & x_1^{(m)} & \dots & x_n^{(m)} \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

Hypothesis

```
# Compute the hypothesis
hypothesis = np.dot(X_with_bias, theta)
print(f'Hypothesis shape: {hypothesis.shape}')
print('Hypothesis:')
print(hypothesis[:, :])
```

```
       [2.33772596]
       [9.11183583]
       [6.43117383]
       [8.13567262]
       [3.19561354]
       [3.64872999]
       [4.46538309]
       [2.47241124]
       [8.71716295]
       [4.42330708]
       [3.24454793]
       [2.68565241]
       [3.19885339]
       [2.27360214]
       [2.91357811]
       [3.33042504]
       [2.27764143]
       [4.48137197]
       [2.30759955]
       [2.56910189]
       [2.34155488]
       [2.83270803]
       [4.26876193]
       [2.90011379]
       [3.74218079]
       [4.02114467]
       [2.68388522]
       [2.4812472 ]
       [2.28727684]
       [2.56005555]
       [3.37165952]
       [2.62863943]
       [2.3907838 ]
       [3.6480147 ]
       [5.79414318]
       [2.44611374]]
```

Cost Function

```python
from IPython import display
display.Image("cost.PNG")
```

$$J(\theta) = \frac{1}{2m}(X\theta - y)^T(X\theta - y)$$

```python
# Implement Cost Function
def compute_cost(X, y, theta):
    m = len(y)  # Number of training examples
    hypothesis = np.dot(X, theta)
    cost = (1 / (2 * m)) * np.sum((hypothesis - y) ** 2)
    return cost

# Compute initial cost
initial_cost = compute_cost(X_with_bias, y, theta)
print(f'Initial Cost: {initial_cost}')
```

```
Initial Cost: 11.422078030974793
```

Cost Function Derivative and Update Theta

```
from IPython import display
display.Image("update theta.PNG")
```

The derivation of cost function regards to each θ can be vectorized as:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}(x_j)^T(X\theta - y)$$

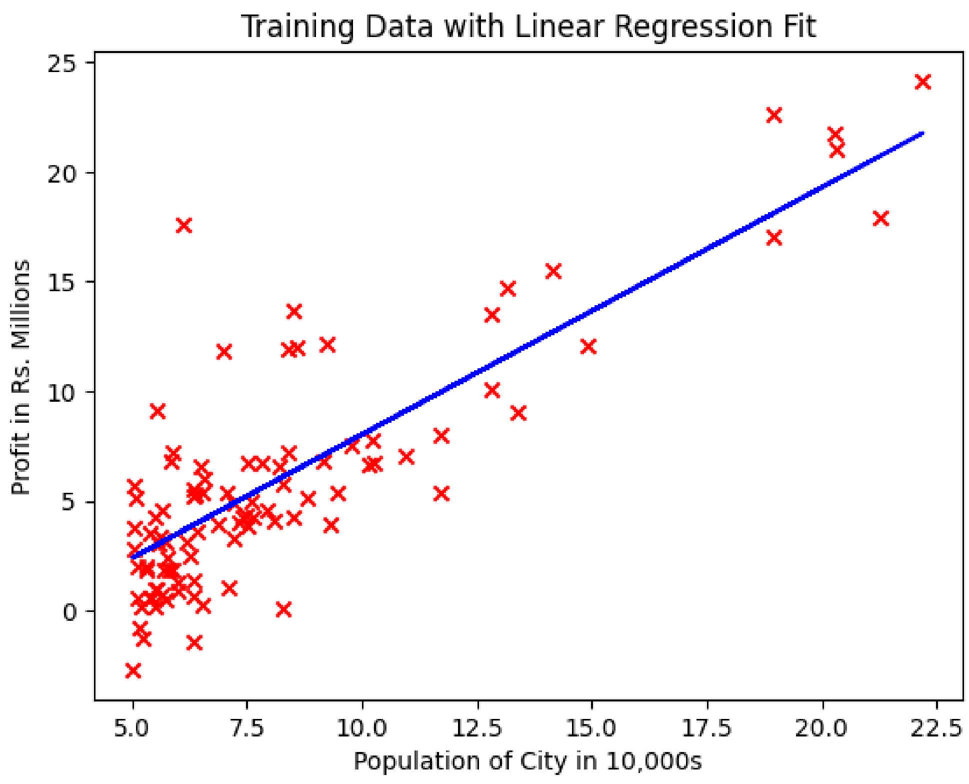
The derivation of cost function to all θ can be vectorized as:

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{m}X^T(X\theta - y)$$

```
# Gradient Descent Function
def gradient_descent(X, y, theta, alpha, iterations):
    m = len(y)
    for i in range(iterations):
        hypothesis = np.dot(X, theta)
        loss = hypothesis - y
        gradient = np.dot(X.T, loss) / m
        theta -= alpha * gradient
        cost = compute_cost(X, y, theta)
        print(f'theta1: {theta[1]}, theta0: {theta[0]}, cost: {cost}, iteration: {i}')
    return theta


# Set learning rate and number of iterations
alpha = 0.01
iterations = 1000

# Perform Gradient Descent
theta = gradient_descent(X_with_bias, y, theta, alpha, iterations)
print('Theta after gradient descent:', theta)

# Compute final cost
final_cost = compute_cost(X_with_bias, y, theta)
print(f'Final Cost: {final_cost}')
```

```
theta1: [1.12124202], theta0: [-3.18115791], cost: 4.523463930027846, iteration: 966
theta1: [1.12137143], theta0: [-3.18244608], cost: 4.523296466531192, iteration: 967
theta1: [1.1215006], theta0: [-3.18373194], cost: 4.523129606228387, iteration: 968
theta1: [1.12162955], theta0: [-3.18501547], cost: 4.5229633469467645, iteration: 969
theta1: [1.12175826], theta0: [-3.1862967], cost: 4.5227976865214785, iteration: 970
theta1: [1.12188674], theta0: [-3.18757561], cost: 4.522632622795486, iteration: 971
theta1: [1.12201499], theta0: [-3.18885222], cost: 4.522468153619508, iteration: 972
theta1: [1.12214301], theta0: [-3.19012652], cost: 4.522304276852011, iteration: 973
theta1: [1.1222708], theta0: [-3.19139853], cost: 4.522140990359173, iteration: 974
theta1: [1.12239835], theta0: [-3.19266825], cost: 4.521978292014858, iteration: 975
theta1: [1.12252568], theta0: [-3.19393568], cost: 4.521816179700592, iteration: 976
theta1: [1.12265278], theta0: [-3.19520082], cost: 4.521654651305526, iteration: 977
theta1: [1.12277965], theta0: [-3.19646368], cost: 4.521493704726417, iteration: 978
theta1: [1.12290629], theta0: [-3.19772427], cost: 4.521333337867597, iteration: 979
theta1: [1.1230327], theta0: [-3.19898258], cost: 4.52117354864095, iteration: 980
theta1: [1.12315888], theta0: [-3.20023863], cost: 4.521014334965873, iteration: 981
theta1: [1.12328484], theta0: [-3.20149241], cost: 4.520855694769267, iteration: 982
theta1: [1.12341057], theta0: [-3.20274393], cost: 4.520697625985496, iteration: 983
theta1: [1.12353607], theta0: [-3.2039932], cost: 4.5205401265563605, iteration: 984
theta1: [1.12366134], theta0: [-3.20524021], cost: 4.52038319443108, iteration: 985
theta1: [1.12378639], theta0: [-3.20648498], cost: 4.520226827566257, iteration: 986
theta1: [1.12391122], theta0: [-3.2077275], cost: 4.520071023925859, iteration: 987
theta1: [1.12403582], theta0: [-3.20896778], cost: 4.519915781481181, iteration: 988
theta1: [1.12416019], theta0: [-3.21020583], cost: 4.519761098210831, iteration: 989
theta1: [1.12428435], theta0: [-3.21144164], cost: 4.519606972100694, iteration: 990
theta1: [1.12440827], theta0: [-3.21267523], cost: 4.519453401143911, iteration: 991
theta1: [1.12453198], theta0: [-3.21390659], cost: 4.519300383340852, iteration: 992
theta1: [1.12465546], theta0: [-3.21513574], cost: 4.51914791669909, iteration: 993
theta1: [1.12477872], theta0: [-3.21636267], cost: 4.518995999233373, iteration: 994
theta1: [1.12490175], theta0: [-3.21758738], cost: 4.518844628965598, iteration: 995
theta1: [1.12502457], theta0: [-3.21880989], cost: 4.518693803924793, iteration: 996
theta1: [1.12514716], theta0: [-3.2200302], cost: 4.518543522147083, iteration: 997
theta1: [1.12526953], theta0: [-3.2212483], cost: 4.518393781675662, iteration: 998
theta1: [1.12539168], theta0: [-3.22246421], cost: 4.518244580560777, iteration: 999
Theta after gradient descent: [[-3.22246421]
 [ 1.12539168]]
Final Cost: 4.518244580560777
```

```python
# Plot the scatter plot with the fitted regression line
plt.scatter(data.population, data.profit, color='red', marker='x')
plt.title("Training Data with Linear Regression Fit")
plt.xlabel("Population of City in 10,000s")
plt.ylabel("Profit in Rs. Millions")
plt.plot(data.population, X_with_bias.dot(theta), color='blue')
plt.show()
```



Multivariate

```
'''
ToDo: Repeat the same steps for multivariate case, add same headings as above
      use the providedex1data2.txt dataset
'''

data = pd.read_csv('ex1data2.txt', sep=",", header=None)       #store the given data in pandas data fra
data.columns = ["X1", "X2", "Y"]         #data loaded in "population" and "profit" variable
data.head(5)            #checking if the data is loaded correctly
```

	X1	X2	Y
0	2104	3	399900
1	1600	3	329900
2	2400	3	369000
3	1416	2	232000
4	3000	4	539900

Next steps: View recommended plots

```
# Data Extraction
X1 = data['X1'].values.reshape(-1, 1)
X2 = data['X2'].values.reshape(-1, 1)
Y = data['Y'].values.reshape(-1, 1)

print(X1.shape)
print(X2.shape)
print(Y.shape)
```

```
(47, 1)
(47, 1)
(47, 1)
```

```
# Stack Features
X_with_bias = np.hstack((np.ones((X1.shape[0], 1)), X1, X2))
print(X_with_bias.shape)
print(X_with_bias[:, :])
```

```
(47, 3)
[[1.000e+00 2.104e+03 3.000e+00]
 [1.000e+00 1.600e+03 3.000e+00]
 [1.000e+00 2.400e+03 3.000e+00]
 [1.000e+00 1.416e+03 2.000e+00]
 [1.000e+00 3.000e+03 4.000e+00]
 [1.000e+00 1.985e+03 4.000e+00]
 [1.000e+00 1.534e+03 3.000e+00]
 [1.000e+00 1.427e+03 3.000e+00]
 [1.000e+00 1.380e+03 3.000e+00]
 [1.000e+00 1.494e+03 3.000e+00]
 [1.000e+00 1.940e+03 4.000e+00]
 [1.000e+00 2.000e+03 3.000e+00]
 [1.000e+00 1.890e+03 3.000e+00]
 [1.000e+00 4.478e+03 5.000e+00]
 [1.000e+00 1.268e+03 3.000e+00]
 [1.000e+00 2.300e+03 4.000e+00]
 [1.000e+00 1.320e+03 2.000e+00]
 [1.000e+00 1.236e+03 3.000e+00]
 [1.000e+00 2.609e+03 4.000e+00]
 [1.000e+00 3.031e+03 4.000e+00]
 [1.000e+00 1.767e+03 3.000e+00]
 [1.000e+00 1.888e+03 2.000e+00]
 [1.000e+00 1.604e+03 3.000e+00]
 [1.000e+00 1.962e+03 4.000e+00]
 [1.000e+00 3.890e+03 3.000e+00]
 [1.000e+00 1.100e+03 3.000e+00]
 [1.000e+00 1.458e+03 3.000e+00]
 [1.000e+00 2.526e+03 3.000e+00]
 [1.000e+00 2.200e+03 3.000e+00]
 [1.000e+00 2.637e+03 3.000e+00]
 [1.000e+00 1.839e+03 2.000e+00]
 [1.000e+00 1.000e+03 1.000e+00]
 [1.000e+00 2.040e+03 4.000e+00]
 [1.000e+00 3.137e+03 3.000e+00]
 [1.000e+00 1.811e+03 4.000e+00]
 [1.000e+00 1.437e+03 3.000e+00]
 [1.000e+00 1.239e+03 3.000e+00]
 [1.000e+00 2.132e+03 4.000e+00]
 [1.000e+00 4.215e+03 4.000e+00]
 [1.000e+00 2.162e+03 4.000e+00]
 [1.000e+00 1.664e+03 2.000e+00]
 [1.000e+00 2.238e+03 3.000e+00]
 [1.000e+00 2.567e+03 4.000e+00]
 [1.000e+00 1.200e+03 3.000e+00]
 [1.000e+00 8.520e+02 2.000e+00]
 [1.000e+00 1.852e+03 4.000e+00]
 [1.000e+00 1.203e+03 3.000e+00]]
```

```
# Initialize Theta (parameters)
theta = np.random.rand(X_with_bias.shape[1], 1)
print('Initial Theta:', theta)
```

```
Initial Theta: [[0.64931045]
 [0.48080006]
 [0.2658947 ]]
```

```
# Compute hypothesis
hypothesis = np.dot(X_with_bias, theta)
print('Hypothesis shape:', hypothesis.shape)
print('Hypothesis:', hypothesis)
```

```
Hypothesis shape: (47, 1)
Hypothesis: [[1013.05032245]
 [ 770.72709181]
 [1155.36714045]
 [ 681.99398593]
 [1444.11307163]
 [ 956.10100992]
 [ 738.9942878 ]
 [ 687.54868129]
 [ 664.95107844]
 [ 719.76228537]
 [ 934.46500718]
 [ 963.04711613]
 [ 910.15910944]
 [2155.00145618]
 [ 611.10147163]
 [1107.55302907]
 [ 635.83718009]
 [ 595.71586968]
 [1256.12024785]
 [1459.01787351]
 [ 851.02070197]
 [ 908.93161463]
 [ 772.65029206]
 [ 945.04260852]
 [1871.75923104]
 [ 530.32706141]
 [ 702.45348318]
 [1215.94794811]
 [1059.20712829]
 [1269.31675486]
 [ 885.37241165]
 [ 481.71526594]
 [ 982.54501326]
 [1509.71678526]
 [ 872.44179934]
 [ 692.3566819 ]
 [ 597.15826986]
 [1026.77861885]
 [2028.28514549]
 [1041.20262068]
 [ 801.23240101]
 [1077.4775306 ]
 [1235.9266453 ]
 [ 578.40706749]
 [ 410.82275164]
 [ 892.15460183]
 [ 579.84946768]]
```

```
# Implement Cost Function
def compute_cost(X, y, theta):
    m = len(y)  # Number of training examples
    hypothesis = np.dot(X, theta)
    cost = (1 / (2 * m)) * np.sum((hypothesis - y) ** 2)
    return cost
```

```
# Gradient Descent Function
def gradient_descent(X, y, theta, alpha, iterations):
    m = len(y)
    for i in range(iterations):
        hypothesis = np.dot(X, theta)
        loss = hypothesis - y
        gradient = np.dot(X.T, loss) / m
        theta -= alpha * gradient
        cost = compute_cost(X, y, theta)
        print(f'Iteration {i}: Theta {theta.flatten()}, Cost {cost}')
    return theta


# Set learning rate and number of iterations
alpha = 0.01
iterations = 30

# Perform Gradient Descent
theta = gradient_descent(X_with_bias, Y, theta, alpha, iterations)
```

```
Iteration 953: Theta [nan nan nan], Cost nan
Iteration 954: Theta [nan nan nan], Cost nan
Iteration 955: Theta [nan nan nan], Cost nan
Iteration 956: Theta [nan nan nan], Cost nan
Iteration 957: Theta [nan nan nan], Cost nan
Iteration 958: Theta [nan nan nan], Cost nan
Iteration 959: Theta [nan nan nan], Cost nan
Iteration 960: Theta [nan nan nan], Cost nan
Iteration 961: Theta [nan nan nan], Cost nan
Iteration 962: Theta [nan nan nan], Cost nan
```

Conclusion:

In this lab, we explored linear regression, a fundamental technique in machine learning used for predicting continuous target variables. We implemented both single-variable and multivariable linear regression using gradient descent optimization. Here are the key takeaways from the lab:

1. **Linear Regression:**
    - Linear regression is a supervised learning algorithm used for modeling the relationship between a dependent variable (target) and one or more independent variables (features).
    - It assumes a linear relationship between the input features and the target variable.
2. **Gradient Descent:**
    - Gradient descent is an optimization algorithm used to minimize the cost function by iteratively adjusting the parameters (theta) of the model.
    - It works by taking steps in the direction of the steepest decrease of the cost function.
3. **Implementation:**
    - We implemented linear regression from scratch using Python and NumPy.
    - For both single-variable and multivariable cases, we initialized random parameters and updated them using gradient descent to minimize the cost function.
4. **Prediction:**
    - After training the model, we can make predictions on new data using the learned parameters.
    - For single-variable regression, we predict the target variable based on a single input feature.
    - For multivariable regression, we predict the target variable based on multiple input features.
5. **Next Steps:**