National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Department of Electrical Engineering

Faculty Member: LE Munadi Sial **Date:**
Semester: **Group:**

CS471 Machine Learning

Lab 7: Linear Regression II – Train-Validation Split and Regularization

		PLO4 - CLO4	PLO4 - CLO4	PLO5 - CLO5	PLO8 - CLO6	PLO9 - CLO7
Name	**Reg. No**	**Viva /Quiz / Lab Performance** **5 Marks**	**Analysis of data in Lab Report** **5 Marks**	**Modern Tool Usage** **5 Marks**	**Ethics** **5 Marks**	**Individual and Team Work** **5 Marks**
Muhammad Anser Sohaib	**367628**					
Muhammad Taqi	**372071**					
Muhammad Talha	**370860**					
Zuha Fatima	**385647**					
Hashaam Zafar	**395508**					

Introduction

This laboratory exercise will extend the python implementation of linear regression performed in the previous lab. Linear regression is a basic supervised learning technique in which parameters are trained on a dataset to fit a model that best approximates that dataset. The problem with using simple linear regression is that the trained models can overfit the dataset at which point regularization must be used to prevent overfitting. This lab will focus on integrating the regularization concept into the gradient descent algorithm.

Objectives

The following are the main objectives of this lab:

- Extract and prepare the training and cross-validation datasets
- Use feature scaling to ensure uniformity among the feature columns
- Implement cost function on both training and cross-validation datasets
- Implement gradient descent algorithm
- Plot the training and cross-validation losses
- Use L2 regularization to counter overfitting

Lab Conduct

- Respect faculty and peers through speech and actions
- The lab faculty will be available to assist the students. In case some aspect of the lab experiment is not understood, the students are advised to seek help from the faculty.
- In the tasks, there are commented lines such as #YOUR CODE STARTS HERE# where you have to provide the code. You must put the code/screenshot/plot between the #START and #END parts of these commented lines. Do NOT remove the commented lines.

- Use the tab key to provide the indentation in python.
- When you provide the code in the report, keep the font size at 12

Theory

Linear Regression is a very basic supervised learning technique. To calculate the loss in each training example, the difference between a hypothesis and the label (y) is calculated. The hypothesis is a linear equation of the features (x) in the dataset with the coefficients acting as the weight parameters. These weight parameters are initialized to random values at the start but are then trained over time to learn the model. The cost function is used to calculated the error between the predicted y^ and the actual y.

A major problem in the training is that the weights that are trained may fit the model for only the data it is given. This means that the model will not generalize to examples outside the dataset and is referred to as "overfitting". Such overfitting makes the machine learning implementation very impractical for real-life applications where data has high variation. To prevent overfitting of the model, a modification in the cost function and gradient descent is implemented. This modification is called regularization and is itself controlled by a hyperparameter (lambda).

A brief summary of the relevant keywords and functions in python is provided below:

print()	output text on console
input()	get input from user on console
range()	create a sequence of numbers
len()	gives the number of characters in a string
if	contains code that executes depending on a logical condition
else	connects with **if** and **elif**, executes when conditions are not met
elif	equivalent to **else if**
while	loops code as long as a condition is true
for	loops code through a sequence of items in an iterable object
break	exit loop immediately
continue	jump to the next iteration of the loop

def	used to define a function

Lab Task 1 - Dataset Preparation, Feature Scaling

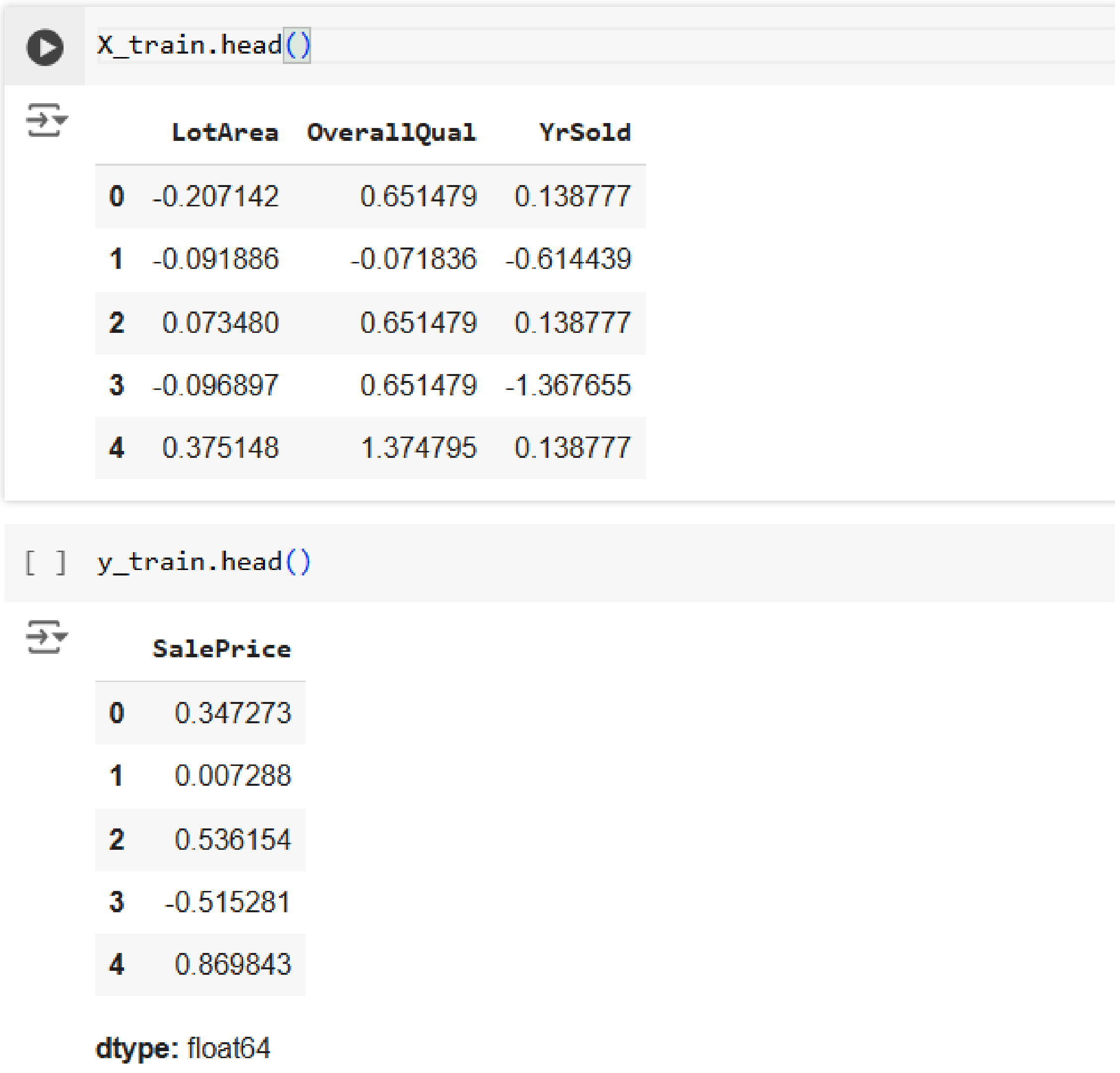
You have been provided with a dataset containing several feature columns. You will need to select any 3 of the feature columns to make your own dataset. The "Sale Price" is the label column that your model will predict. The dataset examples are to be divided into 2 separate portions: training and cross-validation datasets (choose from 80-20 to 70-30 ratios). Save the prepared datasets as CSV files. Next, load the datasets into your python program and store them as NumPy arrays (X_{train} , y_{train}, X_{val}, $y_{val,}$). Next, use feature scaling to rescale the feature columns of both datasets so that their values range from 0 to 1. Finally, print both of the datasets (you need to show any 5 rows of the datasets).

### TASK 1 CODE STARTS HERE ###

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

dataset = pd.read_csv('ML_Lab7_Dataset.csv')
refined_dataset = dataset[["LotArea", "OverallQual", "YrSold", "SalePrice"]]

def normalize(df):
  scaler = StandardScaler()
  df = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)
  return df

refined_dataset = normalize(refined_dataset)

def split_dataset(dataset, test_ratio):
  test_size = len(dataset) * test_ratio
  X_train = dataset.take(range(0, int(len(dataset) - test_size)))
  X_test = dataset.take(range(int(len(dataset) - test_size), len(dataset)))
  y_train = X_train.pop("SalePrice")
  y_test = X_test.pop("SalePrice")
  return X_train, X_test, y_train, y_test

X_train, X_test, y_train, y_test = split_dataset(refined_dataset, 0.2)
```

### TASK 1 CODE ENDS HERE ###

### TASK 1 SCREENSHOT STARTS HERE ###



```
X_train.head()
```

	LotArea	OverallQual	YrSold
0	-0.207142	0.651479	0.138777
1	-0.091886	-0.071836	-0.614439
2	0.073480	0.651479	0.138777
3	-0.096897	0.651479	-1.367655
4	0.375148	1.374795	0.138777

```
y_train.head()
```

	SalePrice
0	0.347273
1	0.007288
2	0.536154
3	-0.515281
4	0.869843

dtype: float64

### TASK 1 SCREENSHOT ENDS HERE ###

Lab Task 2 - Cost Function with Regularization

For linear regression, you will implement the following hypothesis:

$$h(x) = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + \ldots$$

The w_j and b represent the weights while the x_j represents the jth feature. The linear hypothesis h(x) is to be calculated for each training example and its difference with the label y of that training example will represent the loss. In this task, you will write a cost function that calculates the overall loss across a set of examples. This cost function will be useful to calculate the losses in both the training and cross-validation phases of the program.

```
cost_function(X, y, lambd)
```

The X and y are the features and labels of either the training or the cross-validation datasets. This is useful as it can be used for either the training examples or the cross-validation examples of the dataset. The *lambd* is the regularization parameter (Note that *lambda* is a keyword reserved in python). The function will calculate the losses to return the overall cost value. The cost function is given by:

$$J(w) = \frac{1}{2m}\sum_{i=1}^{m}(h(x^{(i)}) - y^{(i)})^2 + \frac{1}{n}\lambda\sum_{j=1}^{n}(w_j)^2$$

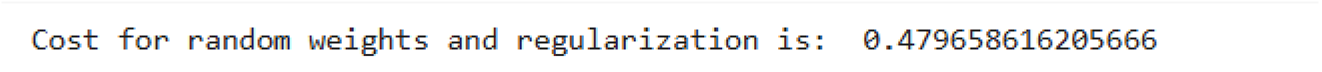
The m is the number of the examples in the dataset and n is the total number of features (or non-bias weights) in the hypothesis. Write the code for the cost function and implement it for your training and cross-validation datasets to print out the cost. Provide the code and all relevant screenshots of the final output.

### TASK 2 CODE STARTS HERE ###

```python
import numpy as np

def cost_with_regularization(X, y, lambd, w0, w1, w2, w3):

    m = len(X)
    predictions = w0 + X[:, 0] * w1 + X[:, 1] * w2 + X[:, 2] * w3


    # Compute the cost without regularization
    cost = (1 / (2 * m)) * np.sum((predictions - y) ** 2)

    # Add regularization term (note: w0 is excluded)
    reg_term = (lambd / (2 * m)) * (w1**2 + w2**2 + w3**2)
    total_cost = cost + reg_term

    return total_cost
print(cost_with_regularization(X_train, y_train, 0.01, 0.5, 0.01, 0.2, 0.001))
```

### TASK 2 CODE ENDS HERE ###

### TASK 2 SCREENSHOT STARTS HERE ###



```
Cost for random weights and regularization is:  0.479658616205666
```

### TASK 2 SCREENSHOT ENDS HERE ###

Lab Task 3 –Gradient Descent with Regularization

In this task, you will write a function that uses gradient descent to update the weight parameters:

```
gradient_descent(X, y, alpha, lambd)
```

The *alpha* is the learning rate (hyperparameter 1) and *lambd* is the regularization parameter (hyperparameter 2). The gradient descent algorithm is given as follows:

$$dw_j = \frac{\partial J}{\partial w_j} = \frac{1}{m}\sum_{i=1}^{m}(\mathrm{h}(\mathrm{x}^{(i)}) - \mathrm{y}^{(i)})x_j{}^{(i)} + \frac{\lambda}{m}w_j$$

$$db = \frac{\partial J}{\partial b} = \frac{1}{m}\sum_{i=1}^{m}(\mathrm{h}(\mathrm{x}^{(i)}) - \mathrm{y}^{(i)})$$

$$w_j := w_j - \alpha\frac{\partial J}{\partial w_j}$$

$$b := b - \alpha\frac{\partial J}{\partial w_j}$$

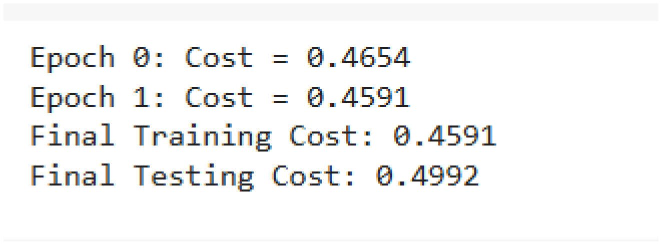
For the submission, you will need to run the gradient descent algorithm once to update the weights. You will need to print the weights, training cost and validation cost both before and after the weight update. Provide the code and all relevant screenshots of the final output.

### TASK 3 CODE STARTS HERE ###

```
def gradient_descent_with_regularization(X, y, alpha, lambd, num_epochs):
    w0 = np.random.randn() * 0.01
    w1 = np.random.randn() * 0.01
    w2 = np.random.randn() * 0.01
    w3 = np.random.randn() * 0.01
    for i in range(num_epochs):

        predictions = (w0 + X[:, 0] * w1 + X[:, 1] * w2 + X[:, 2] * w3)
        dw0 = 1 / len(X) * np.sum(predictions - y)
        dw1 = (1 / len(X) * np.sum((predictions - y) * X[:, 0])) + (lambd / len(X)) * w1
        dw2 = (1 / len(X) * np.sum((predictions - y) * X[:, 1])) + (lambd / len(X)) * w2
        dw3 = (1 / len(X) * np.sum((predictions - y) * X[:, 2])) + (lambd / len(X)) * w3

        w0 -= alpha * dw0
        w1 -= alpha * dw1
        w2 -= alpha * dw2
        w3 -= alpha * dw3

        # Log the cost every 50 epochs for better tracking
        if i % 2 == 0:
            cost = cost_with_regularization(X, y, lambd, w0, w1, w2, w3)
            print(f"Epoch {i}: Cost = {cost:.4f}")  # Ensure cost is a scalar

    return w0, w1, w2, w3  # Return the weights after training
```

### TASK 3 CODE ENDS HERE ###

### TASK 3 SCREENSHOT STARTS HERE ###



```
Epoch 0: Cost = 0.4654
Epoch 1: Cost = 0.4591
Final Training Cost: 0.4591
Final Testing Cost: 0.4992
```

### TASK 3 SCREENSHOT ENDS HERE ###

Lab Task 4 – Training and Validation Program

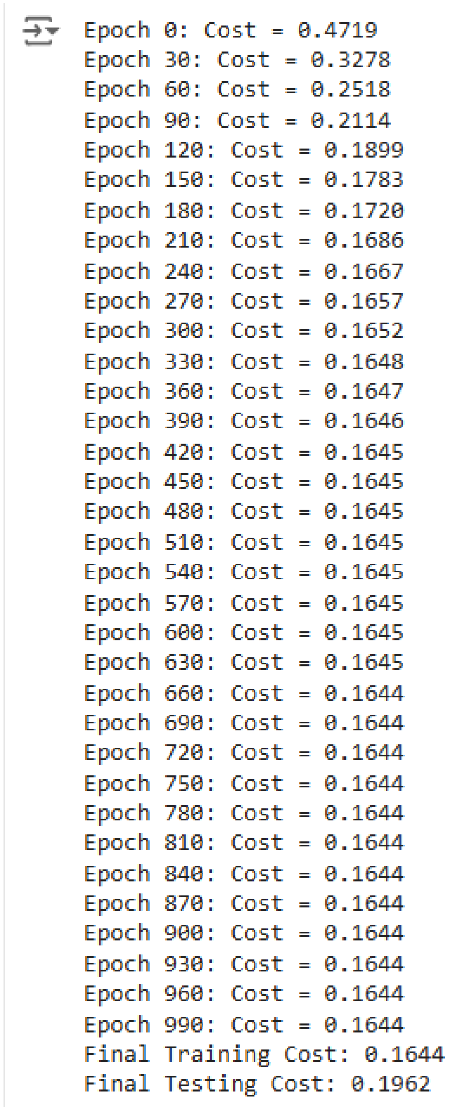
In this task, you will use the functions from the previous two tasks to write a "main" function that performs the actual training and validation. Use the cost function and gradient descent function on the training examples to determine the training loss and update the weights respectively. Then, use the cost function on the cross-validation examples to determine the cross-validation loss. This single iteration over the entire dataset (both training and cross-validation) marks the completion of one epoch. You will need to perform the training and cross-validation over several epochs (the epoch number is another hyperparameter that must be chosen). Ensure that at the end of each epoch, the training and cross-validation losses are stored for plotting purposes. When the final epoch is performed, note down the trained parameters (weights and bias) and make plot of the training and cross-validation losses (y-axis) over the epochs (x-axis). Ensure that both of the losses appear on the same graph. You only need to show a single plot for this task. Provide the code (excluding function definitions) and all relevant screenshots of the final output.

### TASK 4 CODE STARTS HERE ###

```
import numpy as np

def cost_with_regularization(X, y, lambd, w0, w1, w2, w3):
    m = len(X)
    predictions = w0 + X[:, 0] * w1 + X[:, 1] * w2 + X[:, 2] * w3

    # Compute the cost without regularization
    cost = (1 / (2 * m)) * np.sum((predictions - y) ** 2)

    # Add regularization term (note: w0 is excluded)
    reg_term = (lambd / (2 * m)) * (w1**2 + w2**2 + w3**2)
    total_cost = cost + reg_term

    return total_cost

def gradient_descent_with_regularization(X, y, alpha, lambd, num_epochs):
    w0 = np.random.randn() * 0.01
    w1 = np.random.randn() * 0.01
    w2 = np.random.randn() * 0.01
    w3 = np.random.randn() * 0.01
    for i in range(num_epochs):

        predictions = (w0 + X[:, 0] * w1 + X[:, 1] * w2 + X[:, 2] * w3)

        dw0 = 1 / len(X) * np.sum(predictions - y)
        dw1 = (1 / len(X) * np.sum((predictions - y) * X[:, 0])) + (lambd /
len(X)) * w1
        dw2 = (1 / len(X) * np.sum((predictions - y) * X[:, 1])) + (lambd /
len(X)) * w2
        dw3 = (1 / len(X) * np.sum((predictions - y) * X[:, 2])) + (lambd /
len(X)) * w3

        w0 -= alpha * dw0
        w1 -= alpha * dw1
        w2 -= alpha * dw2
        w3 -= alpha * dw3

        # Log the cost every 50 epochs for better tracking
        if i % 30 == 0:
            cost = cost_with_regularization(X, y, lambd, w0, w1, w2, w3)
            print(f"Epoch {i}: Cost = {cost:.4f}")  # Ensure cost is a scalar

    return w0, w1, w2, w3  # Return the weights after training

# Assuming X_train, y_train, X_test, y_test are defined before this
w0, w1, w2, w3 = gradient_descent_with_regularization(X_train, y_train, 0.01,
0.01, 1000)
train_cost = cost_with_regularization(X_train, y_train, 0.01, w0, w1, w2, w3)
test_cost = cost_with_regularization(X_test, y_test, 0.01, w0, w1, w2, w3)
print(f"Final Training Cost: {train_cost:.4f}")
print(f"Final Testing Cost: {test_cost:.4f}")
```

### TASK 4 CODE ENDS HERE ###

### TASK 4 SCREENSHOT STARTS HERE ###


```
Epoch 0: Cost = 0.4719
Epoch 30: Cost = 0.3278
Epoch 60: Cost = 0.2518
Epoch 90: Cost = 0.2114
Epoch 120: Cost = 0.1899
Epoch 150: Cost = 0.1783
Epoch 180: Cost = 0.1720
Epoch 210: Cost = 0.1686
Epoch 240: Cost = 0.1667
Epoch 270: Cost = 0.1657
Epoch 300: Cost = 0.1652
Epoch 330: Cost = 0.1648
Epoch 360: Cost = 0.1647
Epoch 390: Cost = 0.1646
Epoch 420: Cost = 0.1645
Epoch 450: Cost = 0.1645
Epoch 480: Cost = 0.1645
Epoch 510: Cost = 0.1645
Epoch 540: Cost = 0.1645
Epoch 570: Cost = 0.1645
Epoch 600: Cost = 0.1645
Epoch 630: Cost = 0.1645
Epoch 660: Cost = 0.1644
Epoch 690: Cost = 0.1644
Epoch 720: Cost = 0.1644
Epoch 750: Cost = 0.1644
Epoch 780: Cost = 0.1644
Epoch 810: Cost = 0.1644
Epoch 840: Cost = 0.1644
Epoch 870: Cost = 0.1644
Epoch 900: Cost = 0.1644
Epoch 930: Cost = 0.1644
Epoch 960: Cost = 0.1644
Epoch 990: Cost = 0.1644
Final Training Cost: 0.1644
Final Testing Cost: 0.1962
```

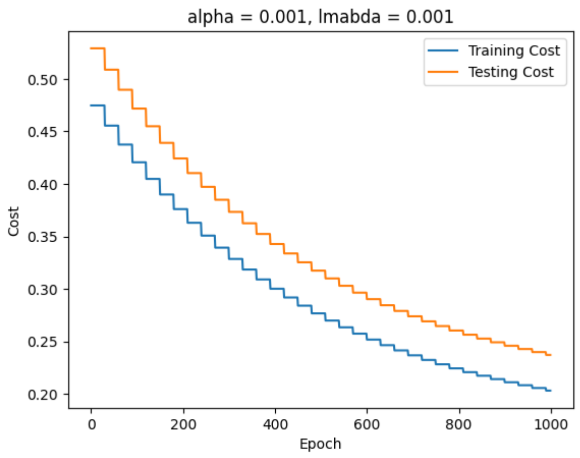
### TASK 4 SCREENSHOT ENDS HERE ###

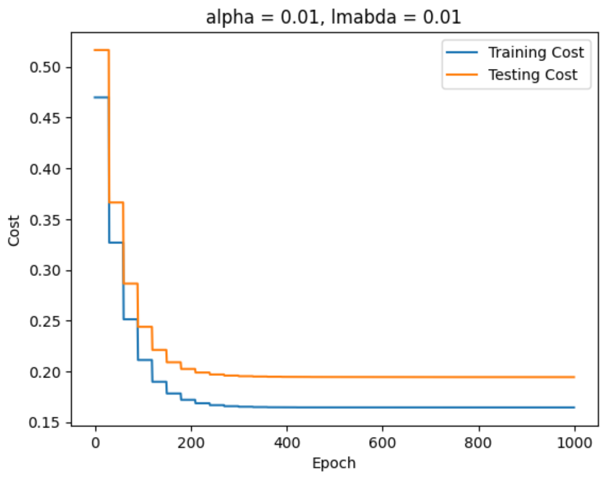
Lab Task 5 – Tuning Alpha and Lambda

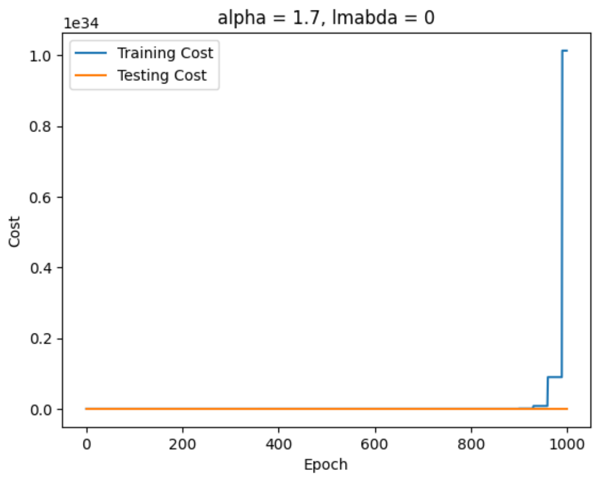
In this task, you will use your linear regression code from the previous task. Tune the alpha and lambda hyperparameters at different values to get several plots.

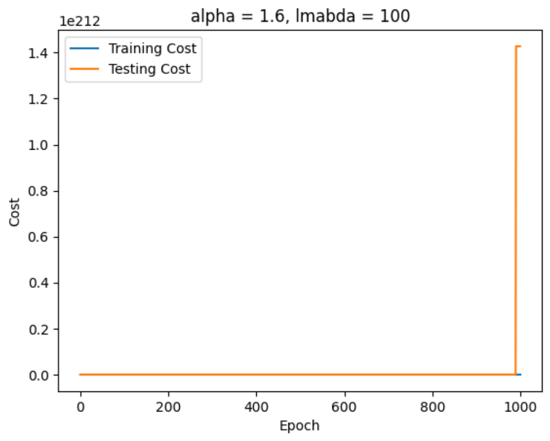
You need to get at least 6 plots. Mention the alpha and lambda values in the plot titles. Ensure all axes are labeled appropriately.

### TASK 5 PLOTS START HERE ###









### TASK 5 PLOTS END HERE ###