Lab 10

```
import numpy as np
import matplotlib.pyplot as plt
import copy
import math

# Function to load data (You need to define your own load_data function or manually provide
data)
def load_data():
  # Example dataset
    x_train = np.array([6.1101, 5.5277, 8.5186, 7.0032, 5.8598])
      y_train = np.array([17.592, 9.1302, 13.662, 11.854, 6.8233])
        return x_train, y_train
# Compute cost function
def compute_cost(x, y, w, b):
  m = x.shape[0]
    yp = np.dot(x, w) + b
      total_cost = (1 / (2 * m)) * np.sum((yp - y) ** 2)
        return total_cost
# Compute gradients
def compute_gradient(x, y, w, b):
  m = x.shape[0]
    yp = np.dot(x, w) + b
      dj_dw = (1 / m) * np.sum((yp - y) * x)
        dj_db = (1 / m) * np.sum(yp - y)
          return dj_dw, dj_db
# Gradient descent algorithm
def gradient_descent(x, y, w_in, b_in, cost_function, gradient_function, alpha, num_iters):
  m = len(x)
    J_history = []
      w = copy.deepcopy(w_in)
        b = b_in
          for i in range(num_iters):
              dj_dw, dj_db = gradient_function(x, y, w, b)
                  w = w - alpha * dj_dw
                      b = b - alpha * dj_db
                          if i < 100000:
                                cost = cost_function(x, y, w, b)
                                      J_history.append(cost)
                                          if i % math.ceil(num_iters / 10) == 0:
                                                print(f"Iteration {i}: Cost {cost:.2f}")
                                                  return w, b, J_history
# Load data
x_train, y_train = load_data()

# Gradient descent settings
initial_w = 0.0
```

```
initial_b = 0.0
iterations = 1500
alpha = 0.01

# Run gradient descent
w, b, _ = gradient_descent(x_train, y_train, initial_w, initial_b, compute_cost, compute_gradient,
alpha, iterations)
print(f"Optimal parameters: w = {w}, b = {b}")

# Plot the linear fit
m = x_train.shape[0]
predicted = np.zeros(m)
for i in range(m):
    predicted[i] = w * x_train[i] + b
plt.plot(x_train, predicted, c="b")
plt.scatter(x_train, y_train, marker='x', c='r')
plt.title("Profits vs. Population per city")
plt.ylabel('Profit in $10,000')
plt.xlabel('Population of City in 10,000s')
plt.show()

# Predict profit for cities with population of 35,000 and 70,000
predict1 = 3.5 * w + b
predict2 = 7.0 * w + b
print(f'For population = 35,000, we predict a profit of ${predict1 * 10000:.2f}')
print(f'For population = 70,000, we predict a profit of ${predict2 * 10000:.2f}')
[9/9, 4:04 PM] Iman Rafiq PU: import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import copy
import math

# Function to load data with error handling
def load_data(file_path):
    try:
            # Load the data without headers
                    data = pd.read_csv(file_path, header=None)
        # Assign column names manually
                data.columns = ['Population', 'Profit']
        # Print column names for debugging purposes
                print("Columns in the dataset:", data.columns)
        x_train = np.array(data['Population'])  # Population as x
                y_train = np.array(data['Profit'])      # Profit as y
                        return x_train, y_train
    except FileNotFoundError:
            print(f"Error: The file {file_path} was not found.")
                    raise
                        except KeyError as e:
```

```
                print(f"Error: {e}")
                    raise
                        except pd.errors.EmptyDataError:
                            print("Error: The file is empty.")
                                raise
                                    except Exception as e:
                                        print(f"An unexpected error occurred: {e}")
                                            raise
# Example usage
file_path = 'ex1data1.txt'  # Replace with the path to your actual data file
x_train, y_train = load_data(file_path)

# Compute cost function
def compute_cost(x, y, w, b):
  m = x.shape[0]
    yp = np.dot(x, w) + b
      total_cost = (1 / (2 * m)) * np.sum((yp - y) ** 2)
        return total_cost
# Compute gradients
def compute_gradient(x, y, w, b):
  m = x.shape[0]
    yp = np.dot(x, w) + b
      dj_dw = (1 / m) * np.sum((yp - y) * x)
        dj_db = (1 / m) * np.sum(yp - y)
          return dj_dw, dj_db
# Gradient descent algorithm
def gradient_descent(x, y, w_in, b_in, cost_function, gradient_function, alpha, num_iters):
  m = len(x)
    J_history = []
      w = copy.deepcopy(w_in)
        b = b_in
          for i in range(num_iters):
              dj_dw, dj_db = gradient_function(x, y, w, b)
                  w = w - alpha * dj_dw
                      b = b - alpha * dj_db
                          if i < 100000:
                                cost = cost_function(x, y, w, b)
                                      J_history.append(cost)
                                          if i % math.ceil(num_iters / 10) == 0:
                                                print(f"Iteration {i}: Cost {cost:.2f}")
                                                  return w, b, J_history
# Gradient descent settings
initial_w = 0.0
initial_b = 0.0
iterations = 1500
alpha = 0.01

# Run gradient descent
```

```
w, b, _ = gradient_descent(x_train, y_train, initial_w, initial_b, compute_cost, compute_gradient,
alpha, iterations)
print(f"Optimal parameters: w = {w:.2f}, b = {b:.2f}")

# Plot the linear fit
m = x_train.shape[0]
predicted = np.zeros(m)
for i in range(m):
    predicted[i] = w * x_train[i] + b
plt.plot(x_train, predicted, c="b")
plt.scatter(x_train, y_train, marker='x', c='r')
plt.title("Profits vs. Population per city")
plt.ylabel('Profit in $10,000')
plt.xlabel('Population of City in 10,000s')
plt.show()

# Predict profit for cities with population of 35,000 and 70,000
predict1 = 3.5 * w + b
predict2 = 7.0 * w + b
print(f'For population = 35,000, we predict a profit of ${predict1 * 10000:.2f}')
print(f'For population = 70,000, we predict a profit of ${predict2 * 10000:.2f}')
```