**M4AI Assignment 2 Report**

*Dataset used: 5. ChoaChuKang Resale transactions 4\_room Jan June\_2023*

*Done By: Ryan Yeo (P2214452)*

**Question 1**

**1(a)**

A graph with blue dots and a red line

Description automatically generated

Equation: *Resale Price* = 6369 *Remaining Lease (years)*

**1(b)**

Since number of rows, n = 294 and yhat = bx, we have the following equations for E(b) and E’(b):

A black screen with white text

Description automatically generated

**1(c) (i)**

**Code:**

x = df['Remaining Lease (years)'].values # features for linear regression

y = df['Resale Price'].values # target variable for linear regression

n = 294 # number of rows

b = 100 # Starting value of b

rate = 0.00\_001 # Set learning rate

epsilon = 0.000\_001 # Stop algorithm when absolute difference between 2 consecutive x-values is less than epsilon

diff = 1 # difference between 2 consecutive iterates

max\_iter = 1000 # set maximum number of iterations

iter = 1 # iterations counter

e = lambda b: 1/n \* np.sum((y - np.dot(b,x))\*\*2) # error function

deriv = lambda b: -2\*(1/n) \* np.matmul((y - np.dot(b,x)),x)

# Now Gradient Descent

while diff > epsilon and iter < max\_iter:

    b\_new = b - rate \* deriv(b)

    print(f"Iteration: {iter}, b-value is: {b\_new:.2f}, e(b) is: {e(b\_new):.2f}, derivative is: {deriv(b\_new):.2f}")

    diff = abs(b\_new - b)

    iter = iter + 1

    b = b\_new

print('\n')

print(f"Number of iterations is {iter}\nThe local minimum occurs when b is {b:.2f}\nMinimum error is {e(b):.2f}")

**Sample Output:**

Number of iterations is 161

The local minimum occurs when b is 6369.42

Minimum error is 3,305,756,271.51

**1(c) (ii)**

Equation: *Resale Price* = 6369.42 *Remaining Lease (years)*

**Question 2**

**2(a)**

A graph with blue dots and red line

Description automatically generated

Equation: *Resale Price* = 316 901 + 2334 *Remaining Lease (years)*

**2(b)**

Since number of rows, n = 294 and yhat = bx + a, we would get the following for E(a,b):

A black background with white text

Description automatically generated

By using Partial Differentiation, we can obtain the following for the derivatives:

A black screen with white text

Description automatically generated

**2(c) (i)**

**Code:**

x = df['Remaining Lease (years)'].values # features for linear regression

y = df['Resale Price'].values # target variable for linear regression

n = 294 # number of rows

a = 1000 # Starting value of a

b = 1000 # Starting value of b

rate\_a = 0.1 # Set learning rate of a

rate\_b = 0.000\_01 # Set learning rate of b

epsilon = 0.000\_001 # Stop algorithm when absolute difference between 2 consecutive x-values is less than epsilon

change\_func = 1 # difference between 2 consecutive iterates

max\_iter = 5\_000\_000 # set maximum number of iterations

iter = 1 # iterations counter

e = lambda a, b: 1/n \* np.sum((y - (np.dot(b,x) + a))\*\*2) # error function

partiale\_a = lambda a, b: -2\*(1/n) \* np.sum(y - (np.dot(b,x) + a)) # derivative of error function with respect to a

partiale\_b = lambda a, b: -2\*(1/n) \* np.matmul((y - (np.dot(b,x) + a)),x) # derivative of error function with respect to b

cur\_e = e(a,b) # current error

# Now Gradient Descent

while change\_func > epsilon and iter < max\_iter:

    # Update a and b

    a\_new = a - rate \* partiale\_a(a,b)

    b\_new = b - rate \* partiale\_b(a,b)

    # Update error based on new values of a and b

    new\_e = e(a\_new,b\_new)

    change\_func = abs(new\_e-cur\_e) # stopping criterion: values of function converge

    iter += 1

    cur\_e = new\_e

    a = a\_new

    b = b\_new

print('\n')

print(f"Number of iterations is {iter}\nThe local minimum occurs when a is {a:.2f} and b is {b:.2f}\nMinimum error is {e(a, b):,.2f}")

**Sample Output:**

Number of iterations is 2660232

The local minimum occurs when a is 316897.33 and b is 2334.07

Minimum error is 1,157,255,207.48

**2 (c) (ii)**

Equation: *Resale Price* = 316 897.33 + 2334.07 *Remaining Lease (years)*

**2 (d)**

**Learning Rate:** Initially, I used a larger learning rate of 0.001. However, Gradient Descent does not converge and returns a minimum error of infinity which suggests that the learning rate was too large. However, I realised that it was just the variable ‘b’ which needed a smaller learning rate to converge. Hence, I used two different learning rates a larger one of 0.1 for ‘a’ and a smaller one of 0.00001 for ‘b’ so that the algorithm could converge.

**Epsilon:** Initially, I used a very small epsilon value of 0.000,000,001 but the algorithm took too long to converge. After increasing the epsilon value of 0.000,001, the algorithm managed to converge in a shorter time and the values of a and b were also not too far off.

**Max Iterations:** Initially, I used a smaller maximum number of iterations of 1,000,000. However, the algorithm took 1,000,000 to converge which means that it might not have found the best value of a and b yet before converging. Increasing the learning rate to reduce the number of iterations was not an option since a larger learning rate resulted in the algorithm not converging at all. As such, I increased the maximum number of iterations to 5,000,000 so that Gradient Descent can go through more iterations before converging.

**Question 3**

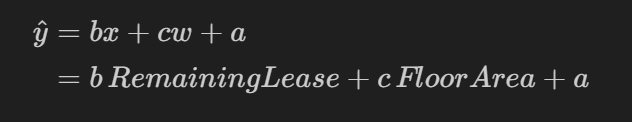
**3(a) Data Collection**

**Data Source:**

A screenshot of a computer screen

Description automatically generated

From the data source, I noticed that Floor Area was another column that was potentially useful for the Linear Regression Model. By adding Floor Area as a variable (w), our model would have the following equation:



The data collection procedure would just be inserting data from the existing data source using the column “Floor Area” into the record which will be used to create model 3.

This is how the records used to create the models look like (existing record was used to create model 2 and the edited record will be used to create model 3)

Existing Record: Edited Record:

A black and white table with numbers and a few white text

Description automatically generated A black rectangular table with white text

Description automatically generated

**3(b) Implementation: Error Function**

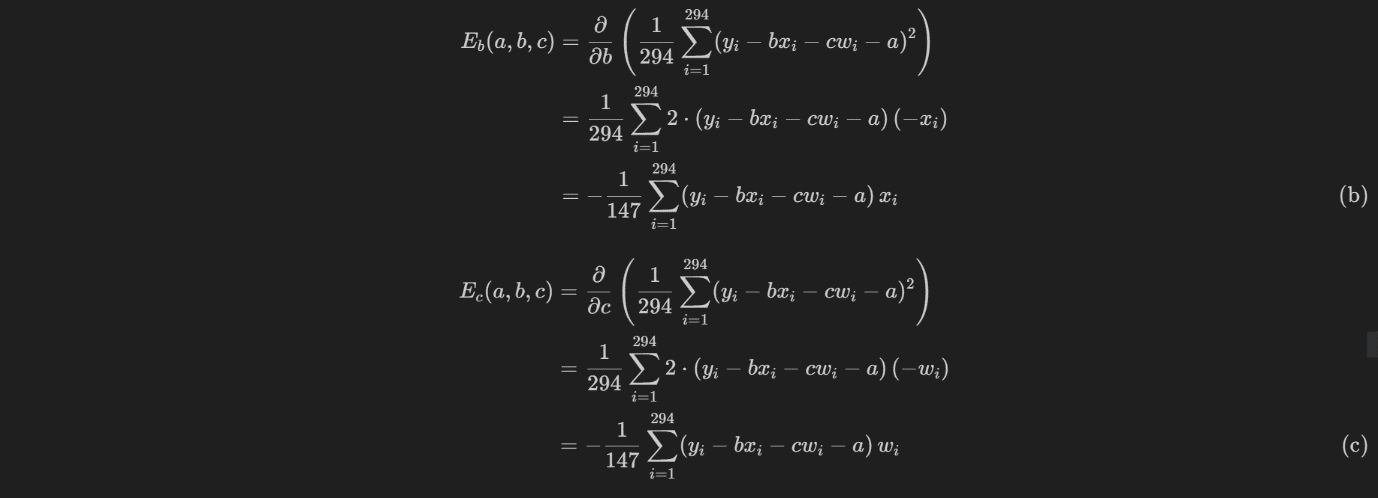
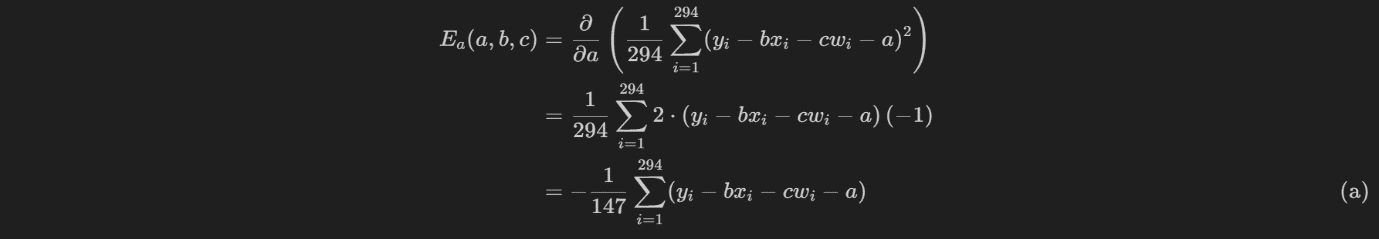
With the addition of w, our error function would also change:

A black screen with white text

Description automatically generated

We would also need to use partial differentiation to obtain the partial derivatives with respect to each of the variables (a, b and c). So that we can update each of the variable during gradient descent.

Differentiating the error function with respect to each of the variables would give the following equations:



We would then update the variables as follows during each iteration of gradient descent:

A black screen with white text

Description automatically generated

Where each of the variables would have their own learning rate since they would converge at different speeds.

**3(c)shut Implementation: Coding and Verification**

**Code:**

x = df['Remaining Lease (years)'].values # features for linear regression

w = df['Floor Area (sqm) /'].values # predictor for linear regression

y = df['Resale Price'].values # target variable for linear regression

n = 294 # number of rows

a = 1000 # Starting value of a

b = 1000 # Starting value of b

c = 1000 # Starting value of c

rate\_a = 0.01 # Set learning rate for a

rate\_b = 0.000\_01 # Set learning rate for b

rate\_c = 0.000\_01 # Set learning rate for c

epsilon = 0.000\_000\_000\_001 # Stop algorithm when absolute difference between 2 consecutive x-values is less than epsilon

change\_func = 1 # difference between 2 consecutive iterates

max\_iter = 1\_000\_000 # set maximum number of iterations

iter = 1 # iterations counter

e = lambda a, b, c: 1/n \* np.sum((y - (np.dot(b,x) + np.dot(c,w) + a))\*\*2) # error function

partiale\_a = lambda a, b, c: -2\*(1/n) \* np.sum(y - (np.dot(b,x) + np.dot(c,w) + a)) # derivative of error function with respect to a

partiale\_b = lambda a, b, c: -2\*(1/n) \* np.matmul((y - (np.dot(b,x) + np.dot(c,w) + a)),x) # derivative of error function with respect to b

partiale\_c = lambda a, b, c: -2\*(1/n) \* np.matmul((y - (np.dot(b,x) + np.dot(c,w) + a)),w) # derivative of error function with respect to c

cur\_e = e(a,b,c) # current error

# Now Gradient Descent

while change\_func > epsilon and iter < max\_iter:

    # Update a, b and c values

    a\_new = a - rate\_a \* partiale\_a(a,b,c)

    b\_new = b - rate\_b \* partiale\_b(a,b,c)

    c\_new = c - rate\_c \* partiale\_c(a,b,c)

    # Update error based on new values of a and b

    new\_e = e(a\_new,b\_new,c\_new)

    change\_func = abs(new\_e-cur\_e) # stopping criterion: values of function converge

    if (iter%100==0):

        print(f"Iter: {iter}, a: {a}, b: {b}, c: {c}")

    iter += 1

    cur\_e = new\_e

    a = a\_new

    b = b\_new

    c = c\_new

print('\n')

print(f"Number of iterations is {iter}\nThe local minimum occurs when a is {a:.2f}, b is {b:.2f} and c is {c:.2f}\nMinimum error is {e(a,b,c):,.2f}")

**Sample Output:**

Number of iterations is 309173

The local minimum occurs when a is -39258.28, b is 3400.28 and c is 2756.71

Minimum error is 925,690,854.70

Equation: *Resale Price* = -39 258.28 + 3400 *Remaining Lease (years)* + 2756.71 Floor Area *(sqm)*

To verify the model, I used Minitab’s regression to build a Linear Regression model with the predictors as Floor Area and Remaining Lease and a response variable of Resale Price.

A screenshot of a computer screen

Description automatically generated

From the output, we can see that the coefficients are all quite similar with the constant term being off by about 4 and floor area being off by about 0.3.