**Name:** Edward Tan Yuan Chong

**INTRODUCTION**

The dataset includes information about condominium projects near Tampines and Pasir Ris, including information such as Transacted Price, Area (SQFT), Number of Units, Street Name, Type of Sale etc.

Our target variables are Transacted Price ($) as and Area (SQFT) as . A screenshot of a computer

Description automatically generated

From looking at the unique values of some of the columns:

Distinct values for Market Segment Column: ['Outside Central Region']

Distinct values for Number of Units Column: [1]

Distinct values for Property Type Column: ['Condominium']

Distinct values for Type of Sale Column: ['Resale']

Distinct values for Type of Area Column: ['Strata']

Distinct values for Postal District Column: [18]

We can see that these columns only have 1 distinct value, and hence would not be useful as predictors for question 3 later on.

A graph with different colored lines

Description automatically generated A graph of different colored bars

Description automatically generated

These charts also show the different project names there are for these condominiums and the number of them, and mean transacted price of each of them.

**MODEL 1: SLR WITH INTERCEPT FIXED 0**

1a) Equation: , where and

**Regression fit**:

A close up of a white background

Description automatically generated

A graph with blue and red dots

Description automatically generated

1b)

= = , and since in our case is as = 0, and where n = number of records which was found to be 307 from the shape of the dataset/number of records, our error function would then be = .

Thus, taking the derivative of E(b), we get = .

Hence, = .

1c) i)

With our derivative of the error function, we then implemented the Gradient Descent Algorithm to identify a local minimum in the error function.

**Gradient Descent Algorithm Code:**

X, Y = price['Area (SQFT)'],price['Transacted Price ($)']

b = 0 # Starting value of b (gradient)

rate = 0.0000001 # Set learning rate [It has to be extremely small as the values from X and Y are very large]

epsilon = 0.001 # Stop algorithm when absolute difference between 2 consecutive b-values is less than epsilon

diff = 1 # Difference between 2 consecutive iterations

max\_iter = 1000 # Maximum number of iterations

iter = 1 # Iterations counter

n = float(len(price)) # Value of n (Number of data points)

f = lambda b: 1/n \* (sum((Y - (b \* X))\*\*2)) # Function E(b)

deriv = lambda b: -2/n \* (sum(X\*(Y - (b \* X))))  # Derivative of function E'(b)

# Gradient Descent Algorithm

while diff > epsilon and iter < max\_iter:

    b\_new = b - rate \* deriv(b)

    diff = abs(f(b\_new) - f(b))

    iter = iter + 1

    b = b\_new

if not iter < max\_iter:

    print("Maximum Iterations Reached.")

print("\nSummary Output:\n" + "Number of iterations is " + str(iter-1) + "\nThe local minimum occurs when b is " + str(b) + "\nMinimum Error is " + str(diff))

**Gradient Descent Algorithm Output:**

Summary Output:

Number of iterations is 59

The local minimum occurs when b is 1146.5100190343794

Minimum Error is 0.00058746337890625

1c) ii)

Equation Obtained from 1c) i) in the format of :

Equation:

The equation does match the regression equation from Minitab, verifying that the gradient descent algorithm worked well.

**MODEL 2: SLR |**

2a) Equation: , where and

**Regression fit:**

A white background with black text

Description automatically generated

A graph with blue and red dots

Description automatically generated

Using a suitable predictor value for the Area (SQFT) of the condominium, such as a value of 1528.49 SQFT, the response variable, Transacted Price ($) would be estimated to be around $1660000 from the Linear Regression Equation in Minitab.

A screen shot of numbers and dots

Description automatically generated

2b)

Error function, where n=307, 𝐸(𝑎, 𝑏) = =

Through differentiation using chain rule, we can derive:

Partial Derivative of 𝐸(𝑎, 𝑏) with respect to a, 𝐸𝑎(𝑎, 𝑏) =

Partial Derivative of 𝐸(𝑎, 𝑏) with respect to b, 𝐸𝑏(𝑎, 𝑏) =

2c) i)

**Gradient Descent Algorithm Code:**

a = 1 # Starting value of a (y-intercept)

b = 1 # Starting value of b (gradient)

rate = 0.1 # Set learning rate

epsilon = 0.000000000001 # Stop algorithm when absolute difference between 2 consecutive values is less than epsilon

diff = 1 # Difference between 2 consecutive iterations

max\_iter = 1000 # Maximum number of iterations

iter = 1 # Iterations counter

n = float(len(price)) # Value of n (Number of data points)

e = lambda a,b: 1/n \* (sum((Y - a - (b \* X))\*\*2)) # Function E(a,b)

ea = lambda a,b: -2/n \* (sum((Y - a - (b \* X))))  # Derivative of function E(a,b) with respect to a

eb = lambda a,b: -2/n \* (sum(X\*(Y - a - (b \* X))))  # Derivative of function E(a,b) with respect to b

# Gradient Descent Algorithm

while diff > epsilon and iter < max\_iter:

    ea\_new = a - rate \* ea(a,b)

    eb\_new = b - rate \* eb(a,b)

    diff = abs(e(ea\_new,eb\_new) - e(a,b))

    iter = iter + 1

    a,b = ea\_new, eb\_new

if not iter < max\_iter:

    print("Maximum Iterations Reached.")

print("Summary Output:\n" + "Number of iterations is " + str(iter-1) + "\nThe local minimum occurs when a is " + str(a) + " and b is " + str(b) + "\nMinimum Error is " + str(diff))

**Gradient Descent Algorithm Output\*:**

Summary Output:

Number of iterations is 61

The local minimum occurs when a is 1.2259964331540898e-06 and b is 0.9004111761253796

Minimum Error is 8.54011306117286e-13

\*Do take note that value of a and b here is scaled, and hence we would need to unscale it before using it to form the regression equation

2c) ii) We then proceed to unscale the values of a and b we obtained.

**UNSCALING**

Unscaling is done with this formula to obtain unscaled values of :

A math equation on a white board

Description automatically generated

where are the standard deviations of each variable respectively, and are the means of each variable respectively.

**Unscaling Output**:

Value of a: 332009.30222487776 | Value of b: 876.5253815063592

With that, we can form our equations with values of a and b obtained from the gradient descent algorithm.

Equation Obtained from 2c) i) in the format of :

Equation:

2d)

**Parameters**:

: 1

: 1

Learning rate (rate): 0.1

Epsilon (epsilon): 0.000000000001/1e-12

**Describe changes to parameters**:

The starting value of and were initially tested with zero, however it did not converge to reasonable values, hence I set the starting values of and to 1. The learning rate was increased from a value close to zero to 0.1, as a higher learning rate would allow for faster change in parameters for faster convergence, requiring less iterations. Meanwhile, the epsilon is lowered significantly closer to 0 in order to minimize the error function as much as possible for convergence to allow the parameters to be as precise as possible, as after scaling the values are more sensitive to change as a small change in scaled values would cause a large difference in unscaled values for and due to the large magnitudes in and , and thus the epsilon was lowered significantly to 0.000000000001/1e-12, from initial values I tested such as 0.001 or 0.0001.

**MODEL 3: MLR |**

3a) **Data Collection**

**Records:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Transacted Price ($) | Area (SQFT) | Lease Left |
| 55 | 1115000 | 882.65 | 88 |
| 270 | 1290000 | 1528.49 | 67 |
| 127 | 875000 | 968.76 | 68 |
| 139 | 2080000 | 1797.59 | 86 |
| 131 | 1600000 | 1453.14 | 84 |
| 150 | 2380000 | 2637.18 | 84 |
| 38 | 1680000 | 1270.15 | 84 |
| 231 | 2820000 | 2691.00 | 88 |
| 156 | 1470000 | 1119.46 | 88 |
| 239 | 1088888 | 775.01 | 87 |
| 204 | 965000 | 775.01 | 88 |
| 202 | 757000 | 527.44 | 89 |
| 61 | 1888000 | 1410.08 | 86 |
| 135 | 885000 | 936.47 | 68 |
| 115 | 1442000 | 914.94 | 84 |

These are the columns from each row that will be utilized for Gradient Descent. The full dataframe would be in the excel sheet “3AData.xlsx” included in the submission.

**Data Collection Explanation:**

For selecting the 15 records, I performed random sampling where I utilized the random library from python to generate 15 random integers in the range of 1 to 307 (total number of records) from the same seed in order to pick 15 random records, which helps to reduce biasness and provide a diverse representation of the dataset.

For the predictor , I decided to make use of an existing column, “Tenure”, in order to calculate the years of lease left for each resident as column “Lease Left”, using 2023 as the benchmark, with the formula of (Number of years of lease - ( 2023 - Commencement Year of Lease)), eg 99 – ( 2023 – 2008 ) = 84. I chose to do this as the years of lease left would be a vital and influential factor for the price of the condominium due to its duration of ownership and potential future cost and thus would also affect the condominium current pricing and be a prominent predictor for transacted price. Hence, I created “Lease Left” as my predictor .

3b) **Implementation: Error Function**

Functions needed:

Error Function = , where , and

Through differentiation using chain rule, we can derive the partial derivatives with respect to each variable:

* =
* =
* =

The error function was derived with the base error function given as , and substituting , to give us the equation of = . From there, we differentiate the equation with respect to each unknown variables , which would be used later on in gradient descent.

How will the above expressions be used in the gradient descent algorithm to find the values of and , in the regression line, 𝑦̂ = 𝑎 + 𝑏𝑥 + 𝑐𝑤:

* The error function will be used to determine how well each iterations parameter’s fit the data through the residuals and will be used as a guideline for the errors made to find parameters that provide minimum error indicating better fit for the data.
* The partial derivatives of , with respect to each variable, , and , will be used in the gradient descent algorithm to iteratively update the parameters and to minimize the error function and find the optimal values for the regression line to fit the data, with the formula of eg. – α
* This helps to guide the adjustment of parameters to reduce the error.
* The parameters would be updated iteratively until convergence based on the stopping criterions set, which would indicate a local minimum in error.

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

**CODING**

**Gradient Descent Algorithm Code:**

a = 1 # Starting value of a (y-intercept)

b = 1 # Starting value of b (gradient)

c = 1 # Starting value of c

rate = 0.1 # Set learning rate

epsilon = 0.00000000000000001 # Stop algorithm when absolute difference between 2 consecutive values is less than epsilon

diff = 1 # Difference between 2 consecutive iterations

max\_iter = 1000 # Maximum number of iterations

iter = 1 # Iterations counter

n = float(len(df\_price)) # Value of n (Number of data points)

e = lambda a,b,c: 1/n \* (sum((Y - a - (b \* X) - (c \* W))\*\*2)) # Function E(a,b,c)

ea = lambda a,b,c: -2/n \* (sum((Y - a - (b \* X) - (c \* W))))  # Derivative of function E(a,b,c) with respect to a

eb = lambda a,b,c: -2/n \* (sum(X\*(Y - a - (b \* X) - (c \* W))))  # Derivative of function E(a,b,c) with respect to b

ec = lambda a,b,c: -2/n \* (sum(W\*(Y - a - (b \* X) - (c \* W))))  # Derivative of function E(a,b,c) with respect to c

# Gradient Descent Algorithm

while diff > epsilon and iter < max\_iter:

    ea\_new = a - rate \* ea(a,b,c)

    eb\_new = b - rate \* eb(a,b,c)

    ec\_new = c - rate \* ec(a,b,c)

    diff = abs(e(ea\_new,eb\_new,ec\_new) - e(a,b,c))[0]

    iter = iter + 1

    a,b,c = ea\_new[0], eb\_new[0], ec\_new[0]

if not iter < max\_iter:

    print("Maximum Iterations Reached.")

print("\nSummary Output:\n" + "Number of iterations is " + str(iter-1) + "\nThe local minimum occurs when a is " + str(a) + ", b is " + str(b) + " and c is " + str(c) + "\nMinimum Error is " + str(diff))

**Gradient Descent Algorithm Output**\***:**

Summary Output:

Number of iterations is 89

The local minimum occurs when a is 2.3714216417633633e-09, b is 0.9098674424645822 and c is 0.27084866467564367

Minimum Error is 0.0

\*Do take note that value of a, b and c here is scaled, and hence we would need to unscale it before using it to form the regression equation.

**UNSCALING**

Unscaling is done with this formula to obtain unscaled values of and :

A math equations on a white paper

Description automatically generated

where are the standard deviations of each variable respectively, and are the means of each variable respectively

From unscaling, these are the values of and I obtained:

Value of a: -1312981.364946904 | Value of b: 846.0966860011729 | Value of c: 20478.76557149456

With that, we can form our equations with values of and obtained from the gradient descent algorithm.

Equation Obtained from 2c) i) in the format of :

Equation:

**VERIFICATION**

Methods Used:

* Comparison to Minitab output
* Calculate Error
* Evaluation Metrics
  + MAE
  + **MAPE [FOCUS]**
  + MSE
  + RMSE
  + **R-squared [FOCUS]**
  + Explained Variance
* Residual Plot

**Comparison to Minitab output**

I compared the linear regression equation I obtained from gradient descent to the Minitab’s regression equation formed by the data in order to verify the regression equation.

A screenshot of a computer

Description automatically generated

We can see that the linear regression equations are similar, and hence this verifies that the linear regression equation obtained from the gradient descent is correct.

**Calculate Error**

I created a DataFrame showing each row’s original transacted price and transacted price calculated from the regression equation formed by the gradient descent algorithm, along with the error calculated with (Original Transacted Price ($) – Gradient Descent Calculated Transacted Price ($)).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Index | Original Transacted Price ($) | Gradient Descent Calculated Transacted Price ($) | Gradient Descent Error |
| 0 | 55 | 1115000 | 1235957 | -120957 |
| 1 | 270 | 1290000 | 1352346 | -62346 |
| 2 | 127 | 875000 | 899239 | -24239 |
| 3 | 139 | 2080000 | 1969127 | 110873 |
| 4 | 131 | 1600000 | 1636732 | -36732 |
| 5 | 150 | 2380000 | 2638544 | -258544 |
| 6 | 38 | 1680000 | 1481905 | 198095 |
| 7 | 231 | 2820000 | 2765996 | 54004 |
| 8 | 156 | 1470000 | 1436321 | 33679 |
| 9 | 239 | 1088888 | 1124405 | -35517 |
| 10 | 204 | 965000 | 1144883 | -179883 |
| 11 | 202 | 757000 | 955894 | -198894 |
| 12 | 61 | 1888000 | 1641256 | 246744 |
| 13 | 135 | 885000 | 871919 | 13081 |
| 14 | 115 | 1442000 | 1181363 | 260637 |

We can see that most errors made were not significant and that the linear regression equation was a rather good fit.

**Evaluation Metrics**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | MAE | MAPE | MSE | RMSE | R-SQUARED | EXPLAINED VARIANCE |
| 0 | 122281.666667 | 0.089158 | 2.300321e+10 | 151668.076346 | 0.931225 | 0.931225 |

From the evaluation metrics, focusing on MAPE and R-squared as they are easier to interpret, we can see that:

* The R-squared shows that 93% of the variance of the data is explained by the variance of the independent variables and predictors and .
* The MAPE shows that there was around 8% deviation in values from gradient descent compared to the actual values.
* Overall, this shows us that the gradient descent algorithm performed very well as the linear regression equation formed fits very well to the data.

**Residual Plot**

I plotted a residual plot to visualise how far the errors made deviate from the zero line to visualise how well the linear regression equation fit the data.

A graph with blue dots

Description automatically generated

The residual plot shows that even though there are quite a few residuals that deviate far from the zero line, there are also quite a few that are relatively close to the zero line. However, even though some residuals deviate far from the zero line, relative to how large the magnitude the transacted price of the condominiums are, the errors are considered relatively small still, showing that the linear regression equation formed through the gradient descent algorithm provided a decent fit to the data.

All in all, this verifies that the linear regression equation obtained from the gradient descent algorithm is correct and a good fit for the data shown by minimal errors made.

**END**