**MAI Assignment:**

**Build Multiple Linear Regression Model using**

**Gradient Descent Algorithm**

*Goh Pin Pin Isaac*

*P2317623*

*DAAA/FT/2B/23*

Table of Contents

[MODEL 1: SLR with intercept a fixed (𝒚̂𝒊 =𝒃𝒙𝒊) 2](#_Toc173375595)

[MODEL 2: SLR with intercept 𝒂 (𝒚̂𝒊 =𝒂+𝒃𝒙𝒊) 5](#_Toc173375596)

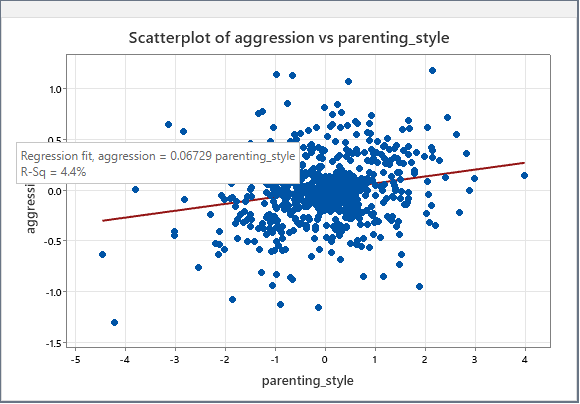
[MODEL 3: MLR (𝒚̂𝒊 =𝒂+𝒃𝒙𝒊 +𝒄𝒘𝒊) 9](#_Toc173375597)

# MODEL 1: SLR with intercept a fixed (𝒚̂𝒊 =𝒃𝒙𝒊)

Q1a(i)

|  |  |
| --- | --- |
| Variable | Description |
| Aggression (Predictor) | Score of the child’s aggression |
| Parenting Style (response variable) | Score of the parent’s parenting style |

Q1a(ii)



Q1b) Express Error function 𝐸(𝑏) in terms of b only since a=0, indicate the value of 𝑛 in your Error function. Hence, derive 𝐸′(𝑏).

|  |
| --- |
| A paper with math equations  Description automatically generated |

Q1c(i)

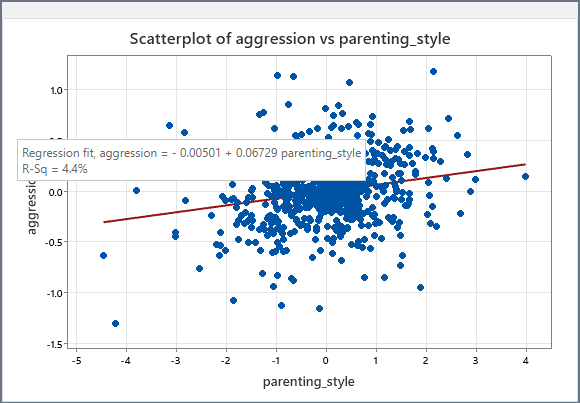
|  |  |
| --- | --- |
| # X and y data  x\_data = child\_aggression['parenting\_style'].values  y\_data = child\_aggression['aggression'].values  # Define the symbols  b = sp.symbols('b')  # Define the error function (Mean Squared Error)  error\_function = sum((y - b\*x)\*\*2 for x, y in zip(x\_data, y\_data))/len(x\_data)  error\_function.simplify()  # Initialize the variables for gradient descent  b\_value = 0.0  # Starting value of b  learning\_rate = 0.1  epsilon = 0.00001  max\_iter = 1000  diff = 1  iter\_count = 0  # differentiating the error function  error\_derivative = error\_function.diff(b)  error\_derivative\_function = sp.lambdify(b, error\_derivative, 'numpy') # Convert the symbolic derivative to a numerical function  # Gradient descent algorithm  while diff > epsilon and iter\_count < max\_iter:      b\_new = b\_value - learning\_rate \* error\_derivative\_function(b\_value)      diff = abs(b\_new - b\_value)      b\_value = b\_new      iter\_count += 1      print(f"Iteration {iter\_count}: b-value is {b\_value}") | Output  Model 1:  Aggression = 0.0673parenting\_style  Number of iterations is 34  The local minimum occurs when b: 0.0673  Minimum error: -0.0001 |

Q1c(ii)

Aggression = 0.0673parenting\_style

# MODEL 2: SLR with intercept 𝒂 (𝒚̂𝒊 =𝒂+𝒃𝒙𝒊)

2(a)



2(b) Express Error function 𝐸(𝑎,𝑏) in terms of a and b. Hence, derive 𝐸𝑎(𝑎,𝑏) and 𝐸𝑏(𝑎,𝑏).

|  |
| --- |
|  |

2(c)(i)

|  |  |
| --- | --- |
| next\_a = 1 # Initial value of a  next\_b = 1 # Initial value of b  alpha = 0.0005 # Learning rate  epsilon = 0.00001 # Stopping criterion constant  max\_iters = 1000 # Maximum number of iterations  # Define the symbols  a = sp.Symbol('a')  b = sp.Symbol('b')  func\_expr = sum((y\_data[i] - (a + b \* x\_data[i]))\*\*2 for i in range(len(x\_data)))  func\_expr.simplify()  # Calculate partial derivatives  partial\_a = sp.diff(func\_expr, a)  partial\_b = sp.diff(func\_expr, b)  # Convert symbolic expressions to numerical functions  partialf\_a = sp.lambdify((a, b), partial\_a)  partialf\_b = sp.lambdify((a, b), partial\_b)  func = sp.lambdify((a, b), func\_expr)  next\_func = func(next\_a, next\_b) # Initial value of function  for n in range(max\_iters):      current\_a = next\_a      current\_b = next\_b      current\_func = next\_func      next\_a = current\_a - alpha \* partialf\_a(current\_a, current\_b)  # update of a      next\_b = current\_b - alpha \* partialf\_b(current\_a, current\_b)  # update of b      next\_func = func(next\_a, next\_b)      change\_func = abs(next\_func - current\_func)  # stopping criterion: values of function converge      print("Iteration", n+1, ": a = ", next\_a, ", b = ", next\_b, ", f(a,b) = ", next\_func)      if change\_func < epsilon:          break | Output  Model 2:  Aggression = -0.0050 + 0.0673parenting\_style  Number of iterations is 10  The optimal value of intercept is: -0.0050  The local minimum occurs when b: 0.0673  Minimum error: -0.0001 |

2(c)(ii)

Aggression = 0.0673parenting\_style

2(d) Describe the changes and decisions you made on the parameters for your solution to reach convergence.

|  |  |
| --- | --- |
| Initial value (a & b) | After checking Minitab, I realized that both numbers were actually very small. Hence, I set both initial values (a & b) to a low number like 1 |
| Alpha / Learning Rate | After the initial setting of the alpha (0.5), the model couldn’t reach the minimum point because the alpha value was too big. Also, the actual gradient was very small (0.0673). Hence, I decided to have an alpha value of (0.0005) |
| Epsilon | Because the actual gradient was very small (0.0673). We need a small epsilon to ensure accuracy. Hence, I chose (0.00001) |

# MODEL 3: MLR (𝒚̂𝒊 =𝒂+𝒃𝒙𝒊 +𝒄𝒘𝒊)

3(a) Data Collection

|  |  |
| --- | --- |
| Variable | Description |
| Aggression (Predictor) | Score of the child’s aggression |
| Parenting Style (Response Variable) | Score of the parent’s parenting style |
| Sibling Aggression (Response Variable) | Score of the sibling’s aggression |

Children often learn behaviors by observing those around them, especially siblings who are close in age and spend considerable time together. If a child witnesses or is involved in aggressive interactions with siblings, they may learn that aggression is an acceptable way to handle conflicts or assert control. This observational learning can lead to the internalization of aggressive behaviors, making them more likely to act aggressively in other situations.

While other factors like television, computer games, diet, sibling aggression is uniquely impactful due to the intimate, frequent, and emotionally charged nature of sibling relationships.

**Data Collection Procedure:**

* In our case study, we were fortunate enough to collect the aggression in the older siblings
* Aggression in older sibling (high score = more aggression seen in their older sibling)

**Data used (First 10 Records):**

|  |  |
| --- | --- |
| Score | Aggression |
| -0.32821637 | Less |
| 0.57683667 | More |
| -0.21718405 | Less |
| 0.04622325 | More |
| -0.89104538 | Less |
| -0.15371262 | Less |
| 0.51178221 | More |
| 0.11305375 | More |
| 0.23494286 | More |
| -0.08567529 | Less |

3(b) Implementation: Error Function

|  |
| --- |
|  |

Gradient descent is an iterative optimization algorithm used to minimize the error function. The parameters a, b and c are updated iteratively using the following rules:

where α is the learning rate, a small positive number that determines the step size of each update.

The algorithm iteratively adjusts the parameters a, b, and c to minimize the error function . The gradients , , guide the adjustments by indicating the direction in which the error function decreases most rapidly.

This process continues until the changes in the parameters are smaller than a predefined threshold, or the maximum number of iterations is reached, at which point the parameters a, b and c are considered to have converged to their optimal values for the given dataset.

3(c) Implementation: Coding and Verification

|  |  |
| --- | --- |
| # X and y data  x\_data1 = child\_aggression['parenting\_style'].values  x\_data2 = child\_aggression['sibling\_aggression'].values  y\_data = child\_aggression['aggression'].values  # Combine input variables  X = np.column\_stack((x\_data1, x\_data2))  # Set up parameters  feature\_count = X.shape[1]  params = [0,0,0]  learning\_rate = 0.001  convergence\_threshold = 0.00001  max\_iterations = 1000  symbols = sp.symbols('x0:%d' % (feature\_count + 1))  # Define prediction function (Array with all the data points with symbols)  prediction = symbols[0] + sum(symbols[i+1] \* X[:, i] for i in range(feature\_count))  # Define loss function (sum of squared errors)  loss\_expr = sum((y\_data[i] - prediction[i])\*\*2 for i in range(len(y\_data)))  # Calculate gradients  gradients = [sp.diff(loss\_expr, symbol) for symbol in symbols]  # Convert symbolic expressions to numerical functions  grad\_funcs = [sp.lambdify(symbols, gradient) for gradient in gradients]  loss\_func = sp.lambdify(symbols, loss\_expr)  current\_loss = loss\_func(\*params)  # Initial loss value  for iteration in range(max\_iterations):      old\_params = params.copy()      old\_loss = current\_loss        # Update parameters      for i in range(len(params)):          params[i] = old\_params[i] - learning\_rate \* grad\_funcs[i](\*old\_params)        current\_loss = loss\_func(\*params)      loss\_change = abs(current\_loss - old\_loss)      print(f"Iteration {iteration+1}: params = {params}, loss = {current\_loss}")        if loss\_change < convergence\_threshold:          break  print("Final parameters:", params) | Output  Model3:  Aggression = -0.0058 + 0.0620parenting\_style + 0.0928sibling\_aggression  Number of iterations is 34  Intercept = -0.0058  Coefficient for parenting\_style = 0.0620  Coefficient for sibling\_aggression = 0.0928  Minimum error: 64.2300 |

***Verification:***

We can observe from the error that it has reached its minimum error of 64.3. Moreover, confirming with Minitab produced comparable outcomes with a difference of less than 0.001.

A screenshot of a calculator

Description automatically generated