Name: Issa Tijani | Class Name: DATA 527| Deadline: 3/01/2025 | Term: Spring 2025

Multiple Linear Regression using Gradient Descent

OVERVIEW

Multiple Linear Regression (MLR) is a widely used statistical technique for modeling the relationship between a dependent variable and multiple independent variables. It plays a crucial role in various fields, including finance, engineering, healthcare, and environmental science, where understanding data patterns and making accurate predictions are essential. **This report details the implementation of MLR using Gradient Descent, an optimization algorithm that iteratively adjusts regression coefficients to minimize error and improve predictive performance**.

The primary goal of this project is to develop an MLR model that minimizes the Mean Squared Error (MSE) through Gradient Descent. The dataset used consists of three variables (x_1, x_2, and y) with 47 observations. Since variables often exist on different scales, feature normalization is applied to ensure that all inputs contribute proportionally to the learning process. A Python script was developed to automate data handling, feature scaling, parameter optimization, and model evaluation.

Gradient Descent updates the regression coefficients iteratively by calculating the cost function's gradient and adjusting parameters accordingly. One critical aspect explored in this study is the effect of different learning rates on model performance. **A learning rate that is too high may prevent convergence or lead to instability, while a rate that is too low can slow down the optimization process.**

To assess model effectiveness, key performance metrics such as Mean Squared Error (MSE), R-squared value, intercept, and regression coefficients are analyzed. The R-squared metric quantifies how well the independent variables explain the variance in the dependent variable. A high R^2 value suggests a strong model fit, though overfitting is also considered.

Additionally, this project includes logging the training process, visualizing optimization progress, and discussing the challenges encountered during implementation. **The results highlight the importance of selecting an appropriate learning rate, the role of feature scaling in improving convergence, and the benefits of iterative optimization in predictive modeling**. Overall, this study provides practical insights into applying Multiple Linear Regression with Gradient Descent and demonstrates its effectiveness in solving real-world analytical problems.

METHODOLOGY

Multiple Linear Regression (MLR) is an extension of simple linear regression that models the relationship between a dependent variable (y) and multiple independent variables

$(x_1, x_2 \ldots\ldots\ldots\ldots, x_n)$.

The general equation of MLR

$h(x) = \text{intercept} + \text{slope}_1 \cdot x_1 + \text{slope}_2 \cdot x_2 + \cdots + \text{slope}_n \cdot x_n$

The goal is to find the optimal regression coefficients that minimize the difference between predicted and actual values. This is achieved by minimizing the **Mean Squared Error (MSE)** using the **Gradient Descent** optimization algorithm. The implementation follows a structured process:

Data Preprocessing

The first step involves loading myDataMLR.xlsx dataset into Python environment and this is achieved by importing openpyxl library. There are no missing values, but the variables are not on the same scale thus normalization is required. **Normalization** is performed to scale all variables to a similar range. Normalization prevents features with larger magnitudes from dominating weight updates during optimization, thereby improving convergence speed.

Model Initialization

To start the training process, all model parameters such as intercept and slopes are initialized to zero. This process provides a neutral starting point for the gradient descent optimization process.

Gradient Descent Optimization

Gradient Descent is an iterative algorithm used to optimize the model's parameters. It follows these steps:

1) Calculate the predicted output using the current parameters values.
2) Compute the MSE to measure error.
3) Calculate partial derivatives of MSE with respect to each parameter (slope1, slope2, intercept).
4) Update the parameters based on the learning rate and derivatives.

Performance Evaluation

The trained model is evaluated using key performance metrics:

- **R-squared (R^2):** Measures how well independent variables explain the variance in the dependent variable.
- **Root Mean Squared Error (RMSE)**: Indicates the average deviation of predictions from actual values.

Logging and Visualization

The training process is logged to track parameter updates and convergence progress. Graphs are generated to visualize MSE reduction over iterations, showing how well the model learns.

To enhance flexibility, **learning rate** and **iteration count** are passed as **command-line arguments**, allowing easy tuning for improved optimization.

IMPLEMENTATION

Data Loading and Normalization:

The myDataMLR.xlsx dataset was loaded into Python software and processed using three different libraries: Openpyxl (to work with excel file), math (for mathematical computation) and matplotlib for data visualization. The data manipulation relies heavily on using list and other Python in-built functions. The dataset was normalized to achieve even variable weight distribution. The Z-score normalization (standardization) because Max-min method was not producing good results.

Normalized each column (independent and dependent variables) using the formula: **x_normalized = (x - mean(x)) / std(x)**

Gradient Descent Algorithm:

1. **Initialize** the slope (coefficients) and intercept (bias) to zero.
2. **For each iteration** n in total number of iterations N:
    1. Compute the **predicted output** y^ for each observation O using the current coefficients and intercept.
    2. Compute the **Mean Squared Error (MSE)** as the cost function.
    3. Calculate the **partial derivative of MSE** with respect to the slope (coefficients).
    4. Calculate the **partial derivative of MSE** with respect to the intercept (bias).
    5. **Update the coefficients** using the computed gradient and the learning rate.
    6. **Update the intercept** using the computed gradient and the learning rate.

This process iteratively minimizes the MSE by adjusting the parameters until convergence or the maximum number of iterations is reached.

```
slope1 = slope1 - learning_rate * slope1_derivative
slope2 = slope2 - learning_rate * slope2_derivative
intercept = intercept - learning_rate * intercept_derivative
```

Performance Metrics and Logging

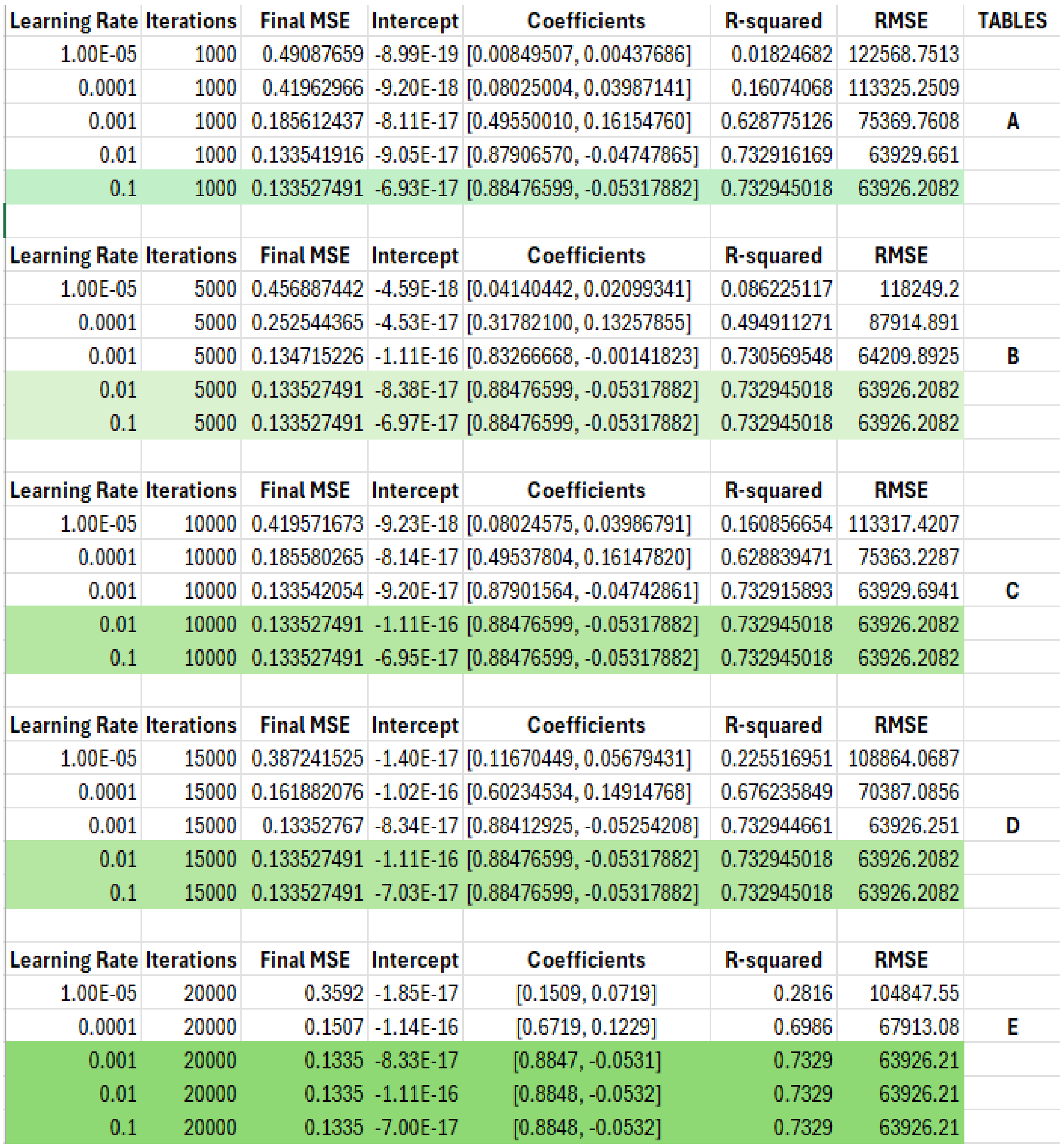
- The script computes **R-squared**, **RMSE**, and logs training details in a file.
- Results are saved and displayed in table format.
- Two plots are generated:
    1. **Actual vs. Predicted Values**
    2. **MSE Trend Over Iterations**

RESULTS

The table below shows the performance of a regression model across various learning rates and iterations. Key metrics such as Final MSE, Intercept, Coefficients, R-squared, and RMSE are recorded across five different cases (A to E**). The model parameters are stored in MLRModelParameters.log.**

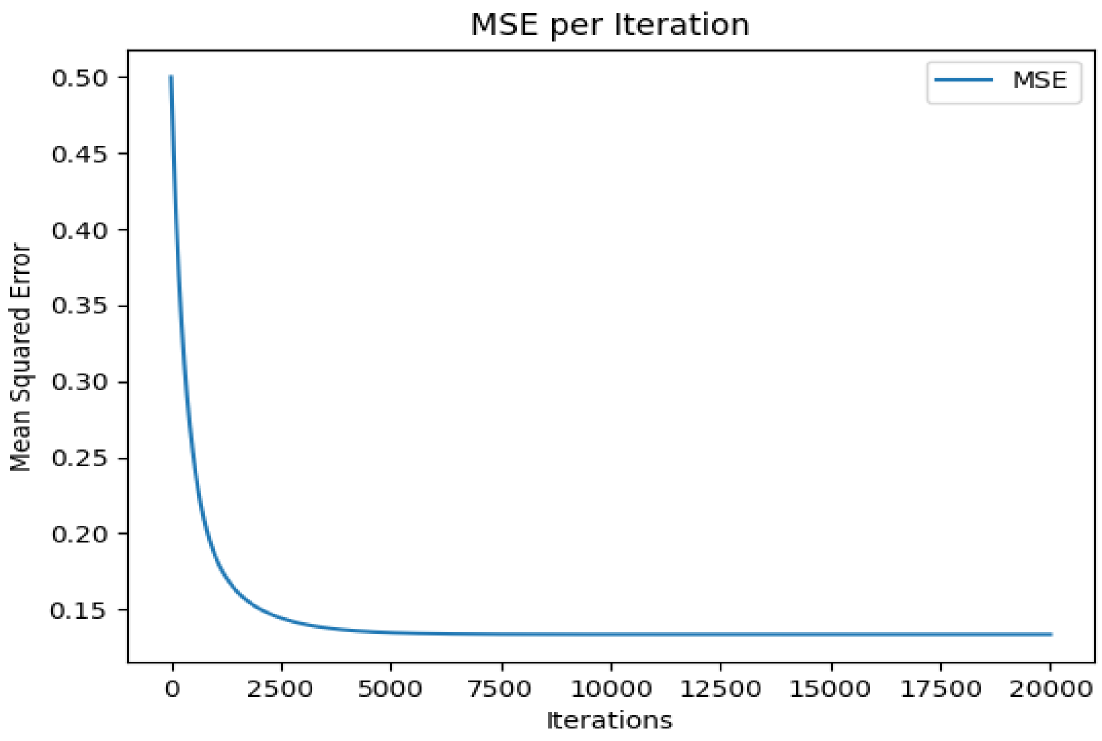
While a learning rate of 0.1 and 1000 iterations achieves fast convergence, the best way to train the model is with a learning rate of 0.001 and 20000+ iterations, as it ensures a stable and well-trained model.

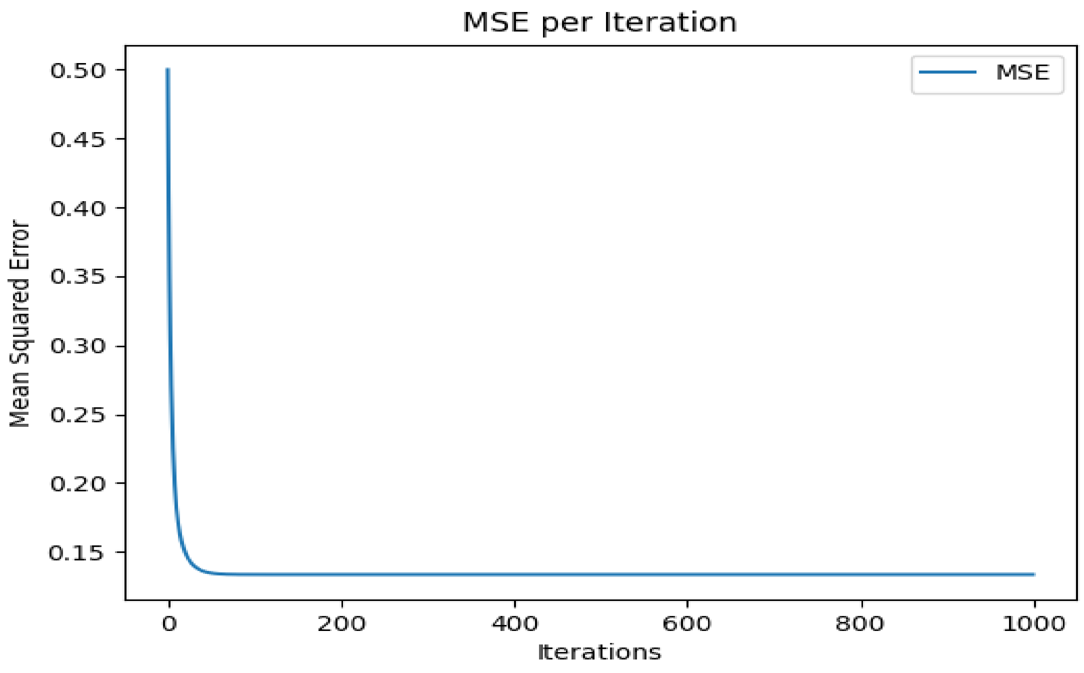
Name: Issa Tijani | Class Name: DATA 527| Deadline: 3/01/2025 | Term: Spring 2025

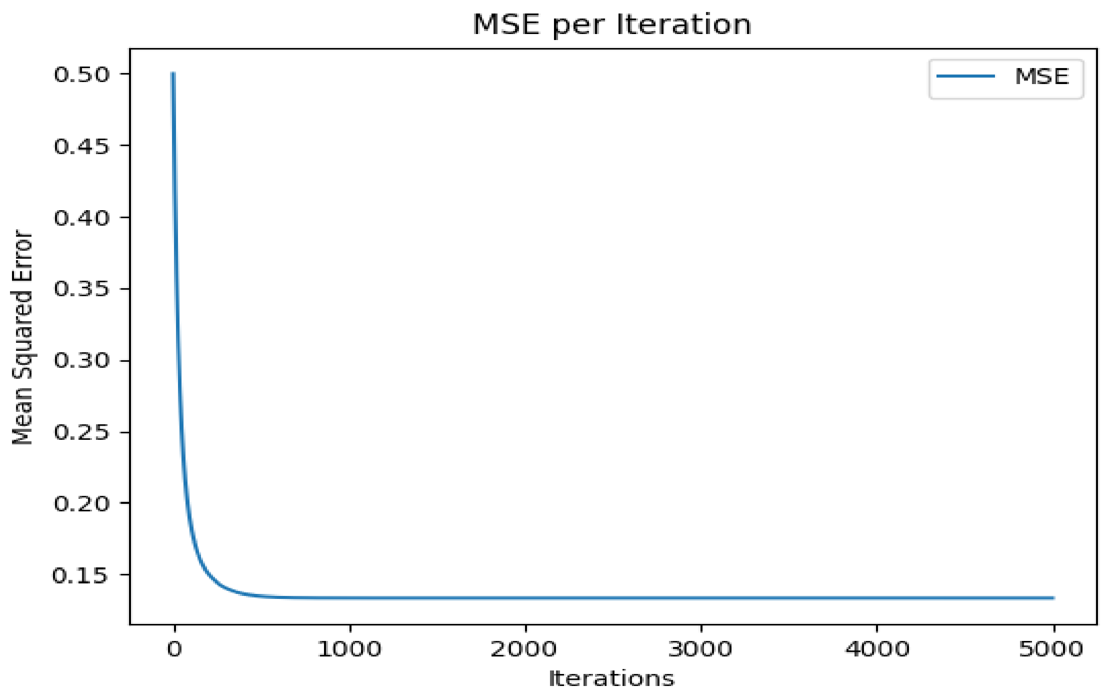


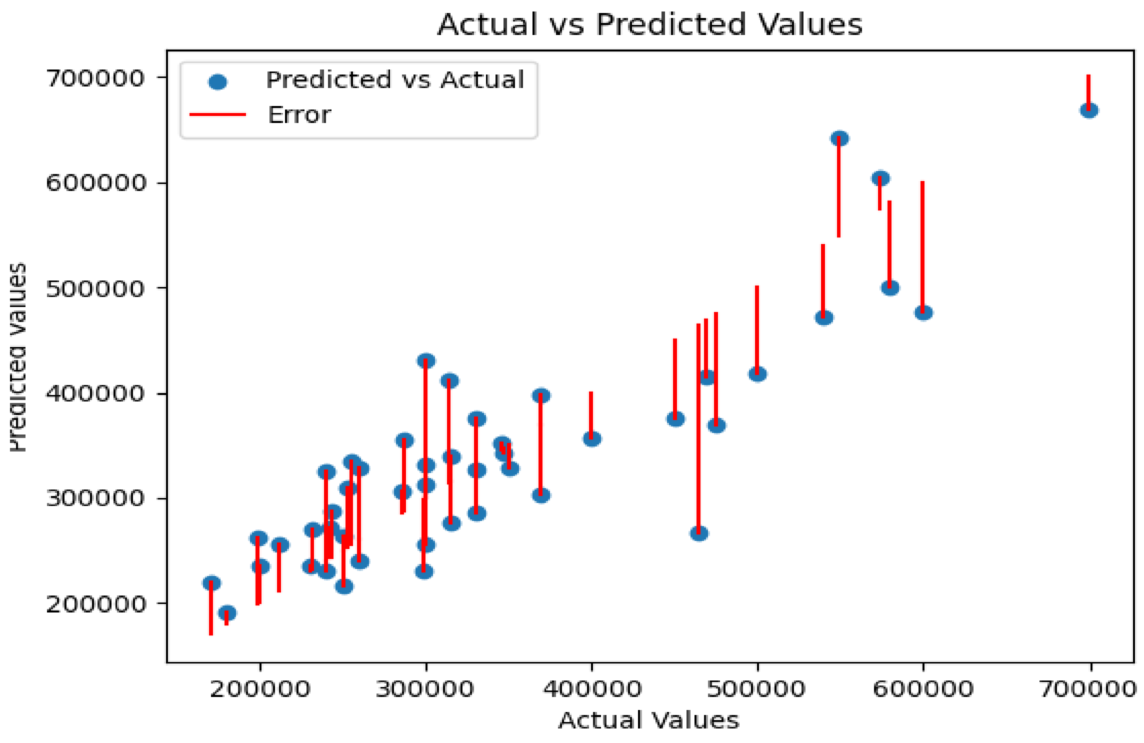
Learning Rate	Iterations	Final MSE	Intercept	Coefficients	R-squared	RMSE	TABLES
1.00E-05	1000	0.49087659	-8.99E-19	[0.00849507, 0.00437686]	0.01824682	122568.7513	
0.0001	1000	0.41962966	-9.20E-18	[0.08025004, 0.03987141]	0.16074068	113325.2509	
0.001	1000	0.185612437	-8.11E-17	[0.49550010, 0.16154760]	0.628775126	75369.7608	**A**
0.01	1000	0.133541916	-9.05E-17	[0.87906570, -0.04747865]	0.732916169	63929.661	
0.1	1000	0.133527491	-6.93E-17	[0.88476599, -0.05317882]	0.732945018	63926.2082	
Learning Rate	**Iterations**	**Final MSE**	**Intercept**	**Coefficients**	**R-squared**	**RMSE**	
1.00E-05	5000	0.456887442	-4.59E-18	[0.04140442, 0.02099341]	0.086225117	118249.2	
0.0001	5000	0.252544365	-4.53E-17	[0.31782100, 0.13257855]	0.494911271	87914.891	
0.001	5000	0.134715226	-1.11E-16	[0.83266668, -0.00141823]	0.730569548	64209.8925	**B**
0.01	5000	0.133527491	-8.38E-17	[0.88476599, -0.05317882]	0.732945018	63926.2082	
0.1	5000	0.133527491	-6.97E-17	[0.88476599, -0.05317882]	0.732945018	63926.2082	
Learning Rate	**Iterations**	**Final MSE**	**Intercept**	**Coefficients**	**R-squared**	**RMSE**	
1.00E-05	10000	0.419571673	-9.23E-18	[0.08024575, 0.03986791]	0.160856654	113317.4207	
0.0001	10000	0.185580265	-8.14E-17	[0.49537804, 0.16147820]	0.628839471	75363.2287	
0.001	10000	0.133542054	-9.20E-17	[0.87901564, -0.04742861]	0.732915893	63929.6941	**C**
0.01	10000	0.133527491	-1.11E-16	[0.88476599, -0.05317882]	0.732945018	63926.2082	
0.1	10000	0.133527491	-6.95E-17	[0.88476599, -0.05317882]	0.732945018	63926.2082	
Learning Rate	**Iterations**	**Final MSE**	**Intercept**	**Coefficients**	**R-squared**	**RMSE**	
1.00E-05	15000	0.387241525	-1.40E-17	[0.11670449, 0.05679431]	0.225516951	108864.0687	
0.0001	15000	0.161882076	-1.02E-16	[0.60234534, 0.14914768]	0.676235849	70387.0856	
0.001	15000	0.13352767	-8.34E-17	[0.88412925, -0.05254208]	0.732944661	63926.251	**D**
0.01	15000	0.133527491	-1.11E-16	[0.88476599, -0.05317882]	0.732945018	63926.2082	
0.1	15000	0.133527491	-7.03E-17	[0.88476599, -0.05317882]	0.732945018	63926.2082	
Learning Rate	**Iterations**	**Final MSE**	**Intercept**	**Coefficients**	**R-squared**	**RMSE**	
1.00E-05	20000	0.3592	-1.85E-17	[0.1509, 0.0719]	0.2816	104847.55	
0.0001	20000	0.1507	-1.14E-16	[0.6719, 0.1229]	0.6986	67913.08	**E**
0.001	20000	0.1335	-8.33E-17	[0.8847, -0.0531]	0.7329	63926.21	
0.01	20000	0.1335	-1.11E-16	[0.8848, -0.0532]	0.7329	63926.21	
0.1	20000	0.1335	-7.00E-17	[0.8848, -0.0532]	0.7329	63926.21	

PLOTS


MSE per Iteration
MSE
Mean Squared Error
0.50
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0
2500
5000
7500
10000
12500
15000
17500
20000
Iterations


MSE per Iteration
MSE
Mean Squared Error
0.50
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0
200
400
600
800
1000
Iterations


MSE per Iteration
MSE
Mean Squared Error
Iterations
0.50
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0
1000
2000
3000
4000
5000

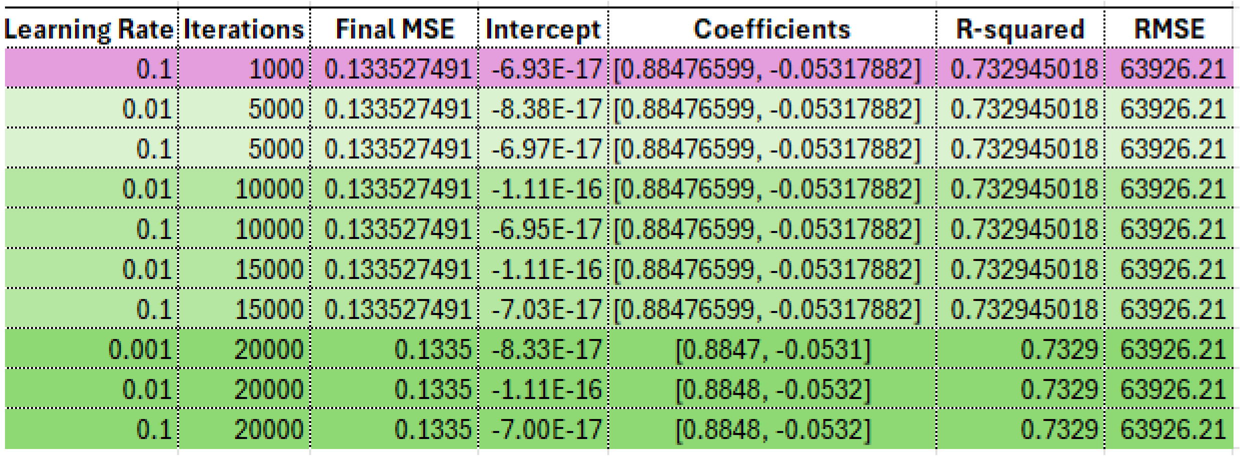

Actual vs Predicted Values
Predicted vs Actual
Error
Predicted Values
Actual Values
700000
600000
500000
400000
300000
200000
200000
300000
400000
500000
600000
700000

DISCUSSION

The results table is presented in an image format to ensure it fits within a single page. The highlighted sections for each iteration cycle emphasize the best-performing result at that stage. Although the summary table suggests similar optimal values across iterations, visualization plays a crucial role in accurately identifying the best results. Below, a summary table provides an overview of the optimal results obtained across different cycles.

At first glance, the table indicates that a learning rate of **0.1** with **1,000 iterations** yields the best results. However, a closer examination of the accompanying graphs (**see plots above**) reveals that the optimal performance is achieved with a **learning rate of 0.001** and **20,000 iterations**.

The following section analyzes and interprets the effects of varying learning rates and iteration counts on **Multiple Linear Regression (MLR)** using **Gradient Descent**.



Learning Rate	Iterations	Final MSE	Intercept	Coefficients	R-squared	RMSE
0.1	1000	0.133527491	-6.93E-17	[0.88476599, -0.05317882]	0.732945018	63926.21
0.01	5000	0.133527491	-8.38E-17	[0.88476599, -0.05317882]	0.732945018	63926.21
0.1	5000	0.133527491	-6.97E-17	[0.88476599, -0.05317882]	0.732945018	63926.21
0.01	10000	0.133527491	-1.11E-16	[0.88476599, -0.05317882]	0.732945018	63926.21
0.1	10000	0.133527491	-6.95E-17	[0.88476599, -0.05317882]	0.732945018	63926.21
0.01	15000	0.133527491	-1.11E-16	[0.88476599, -0.05317882]	0.732945018	63926.21
0.1	15000	0.133527491	-7.03E-17	[0.88476599, -0.05317882]	0.732945018	63926.21
0.001	20000	0.1335	-8.33E-17	[0.8847, -0.0531]	0.7329	63926.21
0.01	20000	0.1335	-1.11E-16	[0.8848, -0.0532]	0.7329	63926.21
0.1	20000	0.1335	-7.00E-17	[0.8848, -0.0532]	0.7329	63926.21

1. Learning Rate and Convergence

- The table shows different learning rates (1.00E-05, 0.0001, 0.001, 0.01, 0.1) with increasing iterations (1000, 5000, 10000, 15000, 20000).
- Small learning rates (1.00E-05, 0.0001): Slow convergence, higher MSE.
- Moderate learning rates (0.001): Balanced learning, leading to a significant reduction in MSE.
- Higher learning rates (0.01, 0.1): Fastest convergence, reaching the lowest Final MSE.

2. Final MSE and Model Performance

- The Final MSE (Mean Squared Error) decreases with the number of iterations.
- A lower Final MSE indicates a better-fitting model.

3. Coefficients and Intercept

- The Coefficients represent the learned weights for independent variables.
- The values converge to (0.88476599, -0.05317882), suggesting a stable solution.

4. R-Squared and RMSE (Root Mean Squared Error)

- R-Squared measures the goodness-of-fit. Higher values (~0.73) indicate a good model fit.
- RMSE shows prediction errors. A lower RMSE is preferred.

- The best R-Squared (0.7329) and lowest RMSE (63926.21) are observed at higher learning rates with sufficient iterations.

Challenges Faced:

- ➢ Standardizing both X and y was essential for effective gradient descent convergence.
- ➢ Finding the right learning rate required multiple tests; too high caused divergence, too low slowed training.
- ➢ Implementing gradient descent manually without libraries was challenging but improved understanding of optimization.
- ➢ Handling file outputs for logs and results required structured organization.

Straightforward Steps:

- ➢ Loading and parsing the dataset manually from Excel using OpenPyXL was simple.
- ➢ Computing the cost function and updating weights using gradient descent was systematic and structured.
- ➢ Saving logs and visualizing results using Matplotlib was easy and effective.

CONCLUSION

The implementation of multiple linear regression using gradient descent effectively predicted the target variable, demonstrating stable model performance with an optimal learning rate. The R-squared value indicates a reasonable fit, though further improvements—such as feature selection, regularization, or advanced optimization techniques—could enhance accuracy and robustness.

References

- Course materials from DATA 527.
- Online resources on linear regression and gradient descent.
- Python documentation for mathematical computations and plotting.
- James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An Introduction to Statistical Learning*.