**Project: Linear Regression from Scratch using SGD**

**Objective**

In this project, you will implement a simple linear regression model from scratch using Stochastic Gradient Descent (SGD) in Python with NumPy, and train it to predict house prices using a single, strong numerical feature from the Kaggle House Prices dataset. You will then tune key hyperparameters, experiment with different loss functions, and finally compare your model’s performance to scikit-learn’s built-in LinearRegression in a separate notebook.

**Deliverables**

You must submit two Jupyter Notebooks:

1. LinearRegression\_Scratch\_YourLastName.ipynb

• Your complete implementation of linear regression using:

o SGD-based optimization

o Your own loss functions

• Experiments with:

o Learning rate

o Batch size

o Number of epochs

o Loss functions: MSE, MAE (optional: Huber)

• Visualization of:

o Raw data

o Training loss over epochs

o Final prediction line

• Final results summary

2. LinearRegression\_Sklearn\_YourLastName.ipynb

• Use scikit-learn’s LinearRegression

• Train and evaluate using the same feature

• Plot prediction line and report metrics

• Compare performance (e.g., RMSE, R-squared, MAE) with your custom model **Dataset**

Use the dataset from the Kaggle competition:

House Prices: Advanced Regression Techniques

https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques/data Use only these columns:

• Feature (x): GrLivArea (Above ground living area)

• Target (y): SalePrice

These columns have a strong linear relationship and are ideal for a single-feature regression task.

**Instructions**

**Tasks for LinearRegression\_Scratch.ipynb:**

1. **Data Loading & Preprocessing**

o Load train.csv

o Extract and clean GrLivArea and SalePrice

o Normalize or standardize features (optional but recommended)

o Visualize the raw data (scatter plot)

2. **Model Implementation**

o Build a simple linear regression model

o Implement Stochastic Gradient Descent to optimize w and b

3. **Loss Functions**

Implement at least:

o Mean Squared Error (MSE)

o Mean Absolute Error (MAE)

o *(Optional)* Huber Loss

4. **Training**

o Implement a full training loop

o Train your model with:

▪ Batch processing (mini-batches)

▪ Loss computation

▪ Gradient updates

▪ Logging of training loss per epoch

5. **Hyperparameter Tuning**

o Experiment with and document:

▪ Learning rate (e.g., 1e-6 to 1e-3)

▪ Batch size (e.g., 16, 32, 64)

▪ Number of epochs

▪ Loss function choice

o Use plots and notes to explain your tuning process and observations 6. **Visualization**

o Plot training loss over epochs

o Plot model predictions vs actual values (with regression line) 7. **Final Summary**

o Best hyperparameters

o Final performance (e.g., final MSE or RMSE)

o Observations and insights

**Tasks for LinearRegression\_Sklearn.ipynb:**

1. Load the same GrLivArea and SalePrice data

2. Train a LinearRegression model using scikit-learn

3. Plot the regression line alongside the actual data

4. Compute performance metrics (MSE or RMSE)

5. Compare your results to your custom implementation

6. Reflect: How close is your model? What differences do you observe?

**Bonus (Optional Challenges. Ensure you understand what these do before implementing)** • Implement Huber Loss and compare its performance

• Add a learning rate schedule (e.g., decay over time)

• Log loss and gradients at every step for visualization

• Compare convergence speeds of different batch sizes

**Deadline: *8/8/2025 9:00am***