Linear Regression from Scratch - Learning Guide

DSMLAI Content Intern Assignment

Dataset Used	**:**	Kaggle Housing Prices Dataset
Prepared By	**:**	Pooja M S
Role Applied	**:**	Content Strategy Intern – DSMLAI
Tools Used	**:**	Python, Jupyter Notebook, NumPy, Pandas
Date of Submission	**:**	11-02-2026

Learning Path Overview

This learning guide presents a structured walkthrough of how a Linear Regression model works internally by implementing it completely from scratch using Python. Instead of relying on machine learning libraries such as sklearn or statsmodels, the model is built step-by-step using core Python, NumPy, and Pandas. This approach helps learners clearly understand the mathematical logic and algorithmic flow behind linear regression.

The notebook associated with this guide is organized into four logical parts: data loading and preprocessing, model definition, training using gradient descent, and performance testing. Each part can be treated as an individual learning module, and this document explains the purpose, concepts, and outcomes of each module in a learner-friendly sequence.

This learning experience is designed for a mixed audience of technical and non-technical learners who already have basic familiarity with Python and machine learning concepts. The goal is not only to show how to run a regression model, but to build conceptual clarity on how predictions are generated, how error is measured, and how model parameters are optimized through iterative training.

By the end of this learning sequence, learners will understand how linear regression calculates predictions, how loss functions guide improvement, and how gradient descent updates model coefficients, all without depending on pre-built ML frameworks.

Video Sequence

Video 1 — Data Loading and Preprocessing

Introduction

In this module, the housing dataset is loaded and systematically prepared for building a linear regression model from scratch. Since regression models operate only on numerical inputs and are sensitive to feature scale and data quality, preprocessing plays a critical role in ensuring stable and meaningful model training.

This section implements custom preprocessing functions using only pandas and NumPy, without relying on machine learning libraries. The workflow includes missing value handling, categorical encoding, dataset splitting, feature standardization, and bias term addition.

Data Loading

The dataset is loaded using pandas from a CSV file containing housing features and prices. Initial inspection is performed to understand column structure and data types before applying transformations.

Missing Value Treatment

A custom function is used to handle missing values column-wise:

Numerical columns are filled using the column mean, Categorical columns are filled using the most frequent value (mode). This ensures that no rows are dropped unnecessarily and that statistical consistency is maintained.

Categorical Encoding

Categorical features are converted into numerical format using one-hot encoding through pd.get_dummies. The first dummy column is dropped to avoid redundant feature representation and multicollinearity.

Train–Test Split (Custom Implementation)

Instead of using external libraries, a custom train–test split function is implemented using NumPy permutation. The dataset is randomly shuffled and divided into training (80%) and testing (20%) subsets. Feature matrices and target vectors are separated at this stage.

Feature Standardization Without Leakage

Feature scaling is applied using standardization:

Xscaled = (X – mean) / std

The mean and standard deviation are computed only from the training data, and the same values are used to scale the test data. This prevents information leakage and preserves the integrity of model evaluation. Zero standard deviation values are safely handled to avoid division errors.

Bias Term Addition

A column of ones is appended to the feature matrix to represent the intercept (bias) term in the linear regression equation. This allows the model to learn a constant offset during training.

Key Learning Takeaways

- How to preprocess mixed-type datasets
- Why categorical encoding is necessary
- How to implement train–test split manually
- Why scaling must use only training statistics
- How bias terms are added in linear models

Video 2 — Linear Model Function

Introduction

In this module, the linear regression prediction model is implemented from scratch using matrix multiplication. Instead of using pre-built regression libraries, a custom prediction function is defined using NumPy operations. This helps learners understand how regression models generate predictions mathematically.

A linear regression model predicts the target value as a weighted sum of input features along with an intercept term. The intercept is handled by adding a bias column to the feature matrix during preprocessing.

The prediction rule for linear regression can be written as:

$$y=w0+w1x1+w2x2+...+wnxn$$

Where:

Xi	=	**feature values**
Wi	=	**model coefficients (weights)**
w0	=	**bias (intercept)**
y	=	**predicted output**

In the implementation, this equation is computed using matrix multiplication for efficiency.

Prediction Function Implementation

A custom prediction function is defined that takes:

Feature matrix X

Weight vector weights and returns predicted values using the dot product.
Conceptually:

Each row of X	=	one house
Each column	=	one feature
Weights	=	learned importance of each feature
Dot product	=	weighted sum across features

Because a bias column of ones was added earlier, the intercept term is automatically included in the same matrix multiplication step.

Why Matrix Multiplication Is Used

Matrix multiplication allows predictions for all samples to be computed at once instead of using loops. This makes the implementation both cleaner and faster while still remaining fully transparent and library-free (except NumPy basics).
This approach also mirrors how regression models are implemented internally in professional ML libraries.

Key Learning Takeaways

- Linear regression predictions are weighted sums of features
- The bias term can be handled by adding a column of ones
- Matrix multiplication replaces manual summation loops
- A prediction function can be implemented in just one line using NumPy dot product
- Understanding this step clarifies how regression models produce outputs

Video 3 — Training the Model Using Gradient Descent

Introduction

In this module, the linear regression model is trained using a custom implementation of gradient descent and Mean Squared Error (MSE) loss. Instead of using built-in optimization tools, the training process is built step-by-step using NumPy operations. This allows learners to understand exactly how model coefficients are adjusted iteratively to reduce prediction error.

Training is the stage where the model “learns” the correct weights by repeatedly comparing predictions with actual values and updating parameters in the direction that reduces error.

Loss Function — Mean Squared Error (MSE)

The model uses Mean Squared Error as the loss function. MSE measures the average squared difference between actual and predicted values.

Conceptually:

- Compute prediction error for each sample
- Square the error to remove negative signs
- Take the average across all samples

This produces a single number representing how far predictions are from true values. Lower MSE indicates better model performance.

A custom function is implemented to compute MSE using NumPy arithmetic without ML libraries.

Gradient Descent — Core Training Logic

Gradient descent is used to minimize the loss function by iteratively updating the weights. The algorithm follows these steps:

- Initialize all weights to zero
- Generate predictions using current weights
- Compute error (prediction − actual)
- Compute gradient using matrix operations
- Update weights by moving in the negative gradient direction
- Repeat for multiple epochs

The update rule used is:

Weights = weights − learning_rate × gradient

The learning rate controls how large each update step is. A smaller learning rate leads to more stable but slower convergence.

Vectorized Gradient Computation

Instead of computing gradients feature-by-feature using loops, the implementation uses matrix multiplication:

- Transpose of feature matrix
- Dot product with error vector
- Scaling by number of samples

This keeps the training efficient and mathematically aligned with standard regression derivations.

Training Progress Tracking

During training, the loss value is recorded at every epoch and stored in a loss history list. The loss is also printed periodically to monitor convergence. After training completes, the loss history is plotted to produce a training curve. This curve visually shows whether the model is learning and whether the error is decreasing steadily across epochs.

Key Learning Takeaways

- Training adjusts weights to reduce prediction error
- MSE is a smooth and widely used regression loss function
- Gradient descent updates weights iteratively
- Learning rate controls update step size
- Vectorized math replaces slow loops
- Loss curves help verify convergence behavior

Video 4 — Model Testing and Performance Evaluation

Introduction

In this module, the trained linear regression model is evaluated on unseen test data to measure how well it generalizes beyond the training set. Model evaluation is a critical step because good performance on training data alone does not guarantee real-world predictive ability.

A separate test dataset created earlier during the train–test split is used to ensure unbiased performance measurement.

Test Prediction Process

A custom evaluation function is implemented that:

- Uses the trained weight vector
- Generates predictions on the test feature matrix
- Computes prediction error using the same MSE loss function

Because the same prediction and loss functions are reused, evaluation remains consistent with the training objective.

Mean Squared Error on Test Data

The primary evaluation metric used is Mean Squared Error (MSE). Test MSE measures the average squared difference between actual house prices and predicted prices on unseen data.

Interpretation:

- Lower test MSE → better generalization
- Large gap between training and test error → possible overfitting
- Similar training and test error → stable model behavior

Using MSE for both training and testing keeps evaluation aligned with the optimization target.

Visual Evaluation — Actual vs Predicted Plot

A scatter plot is generated comparing:

- Actual house prices (true values)
- Predicted house prices (model outputs)

This visualization helps learners intuitively assess model quality:

- Points close to diagonal trend → accurate predictions
- Wide scatter → higher prediction error
- Systematic patterns → possible model bias

Visual diagnostics complement numerical metrics and make model behavior easier to interpret.

Key Learning Takeaways

- Evaluation must be done on unseen test data
- Test metrics measure model generalization
- MSE provides quantitative performance measurement
- Reusing prediction and loss functions ensures consistency
- Scatter plots help visually validate prediction quality
- Evaluation confirms whether training was effective

Learning Outcomes

- After completing this learning series, learning the implementation of the notebook, the following can be achieved:
- Understand what linear regression is and how it performs its predictions based on feature values.
- Discuss the need to pre-process the data before the model is trained.
- Handling missing values in the dataset is of essence, and the next step is transforming the data from a categorical form to a numerical state.
- Split the data properly without the help of any machine learning library.
- Correctly apply feature scaling, and be able to explain how it helps to make the model training more stable.

Assessment Questions

Below are practice questions to test understanding of how linear regression works internally and how it was implemented in this project.

Q1 why do we handle missing values before training the model?

A. Because Python cannot read missing values

B. Because models cannot train properly on incomplete data

C. To make the dataset smaller

D. Only to improve visualization

Correct Answer: B

Feedback: Models require complete numerical inputs to perform calculations and update weights correctly, so handling missing values is necessary. Python itself can read missing values, but training algorithms cannot work reliably with them. The purpose is to improve data quality, not reduce dataset size, and it is not done just for visualization.

Q2 In you're preprocessing, how were numeric missing values filled?

A. With zero

B. With median

C. With mean

D. Rows were deleted

Correct Answer: C

Feedback: Numeric missing values were filled using the column mean to preserve the overall distribution. Filling with zero could distort the data, and while median is another valid method, it was not used here. Rows were not deleted because the approach aimed to retain as much data as possible.

Q3 Why was one-hot encoding used?

A. To make the dataset larger

B. To convert categories into numeric form

C. To remove outliers

D. To speed up plotting

Correct Answer: B

Feedback: One-hot encoding converts categorical variables into numeric columns so that the regression model can process them. Increasing dataset size is only a side effect, not the goal. Encoding does not remove outliers and is not related to plotting speed.

Q4 Why was " drop_first = True " used in dummy encoding?

A. To reduce file size

B. To avoid duplicate columns

C. To avoid multicollinearity

D. To improve visualization

Correct Answer: C

Feedback: Dropping the first dummy column prevents redundant feature relationships and reduces multicollinearity in regression models. The purpose is not file size reduction or visualization improvement, and the dummy columns are related categories rather than exact duplicates.

Q5 Why is feature scaling important for gradient descent?

A. It makes features look nicer

B. It helps weights update more smoothly

C. It removes noise

D. It increases accuracy automatically

Correct Answer: B

Feedback: Feature scaling keeps values in similar ranges, which helps gradient descent update weights more smoothly and stably. It is not done for appearance, does not directly remove noise, and does not automatically guarantee higher accuracy, although it often improves training behavior.

Q6 Why was scaling done using only training data statistics?

A. To make code shorter

B. To prevent data leakage

C. Because test data is not numeric

D. To reduce memory use

Correct Answer: B

Feedback: Scaling used only training data mean and standard deviation to prevent data leakage from the test set into training. This keeps evaluation fair. The reason is not shorter code or memory savings, and the test data is numeric after preprocessing.

Q7 What does adding a bias column of ones allow the model to learn?

A. Feature importance

B. Intercept term

C. Error rate

D. Learning rate

Correct Answer: B

Feedback: Adding a column of ones allows the model to learn the intercept (bias) term as part of the weight vector. Feature importance comes from the learned weights, while error rate and learning rate are separate concepts not created by the bias column.

Q8 What does Mean Squared Error (MSE) measure?

A. Number of wrong predictions

B. Average squared difference between actual and predicted values

C. Percentage accuracy

D. Training speed

Correct Answer: B

Feedback: Mean Squared Error measures the average squared difference between actual and predicted values, making it a standard regression loss metric. Counting wrong predictions and percentage accuracy are more common in classification, and MSE does not measure training speed.

Q9 In gradient descent, what happens to weights each epoch?

A. They are randomized

B. They are deleted

C. They are updated to reduce error

D. They stay constant

Correct Answer: C

Feedback: During each epoch of gradient descent, weights are updated in the direction that reduces prediction error. They are not randomized repeatedly or deleted, and if they stayed constant, the model would not learn anything.

Q10 Why do we evaluate on test data instead of training data? (Multiple Correct)

A. To check generalization ability

B. To avoid overfitting bias

C. Because training data is unavailable

D. Only for plotting

Correct Answers: A and B

Feedback: Test data evaluation shows how well the model performs on unseen examples and helps measure generalization ability. It also prevents overly optimistic performance estimates that can happen if we evaluate only on training data. Training data is available and used for learning, and plotting is optional rather than the main purpose.

Final summary

In this learning walkthrough, we built a complete linear regression model entirely from scratch using Python, NumPy, and Pandas, without relying on machine learning libraries. Each stage of the pipeline from data preprocessing to model prediction, training with gradient descent, and final evaluation was implemented step by step to make the internal working of regression fully transparent.

With manual handling of missing values, encoding of features, a train-test split, scaling, as well as inclusion of bias, it was guaranteed that preparation was adequate for training. The prediction function explained how predictions are made from the regression output, while matrix multiplication was illustrated in the gradient descent function, showing how model weights get optimized through Minimizing Mean Squared Error.

Finally, the evaluation stage verified the effectiveness of the model through the use of various numeric measures as well as graphical plots. This highlights the need to test a model using unseen data.

The overall effect of this exercise is to provide learners with not only practical but also conceptual clarity regarding how linear regression truly functions behind the scenes. This means learners are not forced to think of machine learning models as a "black box" but have moreconfidence with regards to understanding and applying the underlying workings of these models themselves.