STOCK PREDICTION BY (LINEAR REGRESSION)

ENVIRONMENT SETUP

The project was implemented in **Google Colab**, which provided a ready-to-use Python environment with GPU acceleration. The required libraries **pandas** for data handling, **numpy** for numerical operations, **matplotlib** for visualizations, and **PyTorch** for building the linear regression model were pre-installed or easily imported. This setup ensured smooth data preprocessing, model training, and visualization without requiring local installation or configuration.

Data Manipulation

The dataset was loaded into the environment using **pandas**, enabling efficient tabular data handling. Any rows containing missing or null values were removed to ensure the quality and reliability of the model's training data. Since the dataset included records for multiple tickers, it was filtered to focus solely on a single stock (**AAPL**) to simplify the modeling process and maintain consistency in trends. No dummy data was added, as the dataset already contained sufficient entries for effective training and evaluation. This preprocessing ensured that the input to the model was clean, relevant, and consistent.

Feature Selection

In this project, **High**, **Low**, and **Volume** were selected as the independent variables (features), while **Close** was chosen as the dependent variable (target). These features were chosen because the **High** and **Low** prices represent the trading range for the day, providing valuable insight into market volatility, while **Volume** reflects the trading activity and investor sentiment. Together, these variables can significantly influence the closing price of a stock. The **Close** price is the most widely used metric in stock prediction and serves as a reliable indicator of a stock's daily value.

Data Splitting

For this project I used a chronological split to respect the time-series nature of the data (train on earlier dates, test on later dates to avoid look-ahead leakage). I experimented with multiple ratios—60/40, 70/30, 80/20, and 90/10—and compared performance on the held-out test set. The 80/20 split consistently produced the best generalization, giving the highest R^2 and the lowest MSE among the tested options. Intuitively, 80% provided enough history for the model to learn a stable relationship, while 20% left a meaningful future window for unbiased evaluation.

Model Training

I implemented linear regression from scratch using **NumPy matrix operations** (simple Python, no high-level ML training helpers). The model predicts

$$\mathbf{Y\hat{} = Xw+b}$$

and minimizes **Mean Squared Error (MSE**) via gradient descent. I standardized the features (High, Low, Volume) using training-set mean and std only, then ran a learning-rate sweep: 0.1, 0.2, … down to smaller values. While larger learning rates

trained faster, the most stable/lowest error on the test set was achieved with a smaller learning rate = 0.003, after sufficient epochs.
(Separately, I also fit an sklearn LinearRegression only as a sanity check; differences in scale reminded me to compare models mainly with R^2 or with RMSE in original units.)

Model Evaluation

After training, I evaluated on the 20% test window. I computed:

- **MSE**: average squared error between predictions and actual Close.

- R^2: goodness-of-fit on the test set.

  Across the tried splits, the **80/20** configuration yielded the **lowest MSE** and strong R^2, confirming it as the final choice. (Note: I compared learning rates using the same split to ensure a fair comparison.)

Visualization

I produced **two plots** to show performance clearly:

- **Time-ordered line plot** on the test window: Actual Close vs Predicted Close (with points/dots so you can see each day).

- **Scatter plot** of Actual vs Predicted with the **y = X** reference line; tight clustering around the diagonal indicates good fit.
  (Additional visuals like pair plots or full-series time plots could be added, but I intentionally kept the deliverables to these **two** concise figures as required.)

Interpretation and Reporting

I inspected the learned **coefficients** to understand feature influence. Because inputs were standardized, coefficient magnitudes are comparable "per 1-std" move:

- **High** and **Low** carried the **strongest positive effects** on Close, which matches domain intuition (Close usually lies within the daily price range).

- **Volume** had a **smaller effect** relative to price features—useful but less predictive in a simple linear relationship.

- Overall, price-based features dominated volume in this baseline.

Conclusion

Using a clean preprocessing pipeline, chronological **80/20 split**, and a **manual NumPy linear regression** trained with gradient descent, I achieved a solid baseline for predicting Close from High, Low, and Volume. A careful learning-rate search (ending at **0.003**) and standardized inputs helped stabilize training and minimize test error. The two plots demonstrate that predictions track actual values well over the test window.

Prepaired by : Muhammad umer hameed Date: 9 Augest 2025