ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XIV, Issue XI, November 2025

"A Theoretical and Practical Study of Linear Regression"

Dr. Pranesh Kulkarni

Assistant Professor, T. John Institute of Technology (Affiliated to VTU, Belagavi), Gottigere near NICE Road Junction, Bannerghatta Road, Bangalore - 560083

DOI : https://doi.org/10.51583/IJLTEMAS.2025.1411000101

Received: 05 December 2025; Accepted: 12 December 2025; Published: 22 December 2025

ABSTRACT

This article provides a self-contained description of linear regression, covering both the necessary linear algebra concepts and their implementation in Python. Linear regression remains one of the most interpretable and widely used tools in the data scientist's toolbox. By mastering both its theoretical foundations and practical applications, one can build robust and explainable models.

In this paper, we explain the fundamentals of linear regression, outline how it works, and guide the reader through the implementation process step by step. We also discuss essential techniques such as feature scaling and gradient descent, which are crucial for improving model accuracy and efficiency. Whether applied to business trend analysis or broader data science applications, this paper serves as a comprehensive introduction to linear regression for beginners and practitioners alike.

keywords: Linear Regression, Regression Analysis, Statistical Modeling, Predictive Modeling, Machine Learning, Least Squares Method, Model Evaluation, Data Analysis, Regression Theory

INTRODUCTION

Linear regression is a supervised machine learning algorithm used to model the linear relationship between a dependent variable and one or more independent features by fitting a linear equation to observed data. When there is only one independent feature, the method is referred to as *Simple Linear Regression*. When multiple independent features are involved, it is known as *Multiple Linear Regression*. Similarly, if there is only one dependent variable, the model is called *Univariate Linear Regression*, whereas the presence of multiple dependent variables leads to *Multivariate Regression*.

To illustrate, consider the case of a used car dealership that sells only cars of the same model and year. In this setting, it is reasonable to assume that the selling price of a car depends primarily on the number of miles it has been driven. Suppose we acquire a car with 55,000 miles and wish to determine its selling price. If we had a function y=f(x)y = f(x)y=f(x), where yyy represents the selling price and xxx represents the mileage, we could simply substitute x=55,000x =55,000x=55,000 into the function to obtain the expected price. However, in practice, such an exact function is unknown and may not even exist.

What we do have, instead, is historical data: assume that five cars have previously been sold, with their respective mileages and selling prices summarized in **Table 1**. The problem now becomes: *Based on our past experience, at what price should we sell a car with 55,000 miles?* While multiple answers are possible since sellers are free to set asking prices linear regression [1], the focus of this paper, provides a systematic and data-driven approach to estimating such values.

Why Linear Regression is Important?

The interpretability of linear regression is one of its greatest strengths. The model's coefficients clearly show the impact of each independent variable on the dependent variable, providing valuable insights into the underlying dynamics of the data. Its simplicity is also a virtue linear regression is transparent, easy to implement, and forms the foundation for more advanced machine learning algorithms. Many techniques, such as regularization

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XIV, Issue XI, November 2025

methods and support vector machines, extend or are inspired by the principles of linear regression. Moreover, linear regression plays a critical role in statistical inference, helping researchers test assumptions and validate relationships within data.

At its core,[1] linear regression addresses the problem of predicting the value of a continuous dependent random variable Y from one or more continuous independent variables X. This is achieved using a regression function r(x), which takes a value of X and returns a predicted outcome y^.The objective is to find a function r(x) such that for any given x, the prediction y^ is as close as possible to the true value y.

In statistics, the optimal predictor of Y given X is the **conditional expectation**:

$$r(x)=E[Y|X=x]$$

This function provides the best possible prediction in terms of minimizing bias. It does not predict every observation of Y exactly randomness and noise make that impossible but it predicts the **average outcome** for a given X, which is the most reliable approach without additional information.

For example, consider a case where the predicted values should ideally be 5,7,9,11,13,15. Suppose our model predictions deviate slightly, producing values 1 unit higher or lower than the actual outcomes. This results in a **Residual Sum of Squares (RSS)** of 6.0, which quantifies the total error between predictions and true values. Even with such deviations, the function E[Y|X] ensures that the average prediction error is minimized, making it the most effective statistical predictor.

It is important to note that the true shape of E[Y|X] is unknown in practice it can take any form depending on the population data. Linear regression makes the simplifying assumption that this relationship is linear, which is both practical and powerful for many real-world problems.

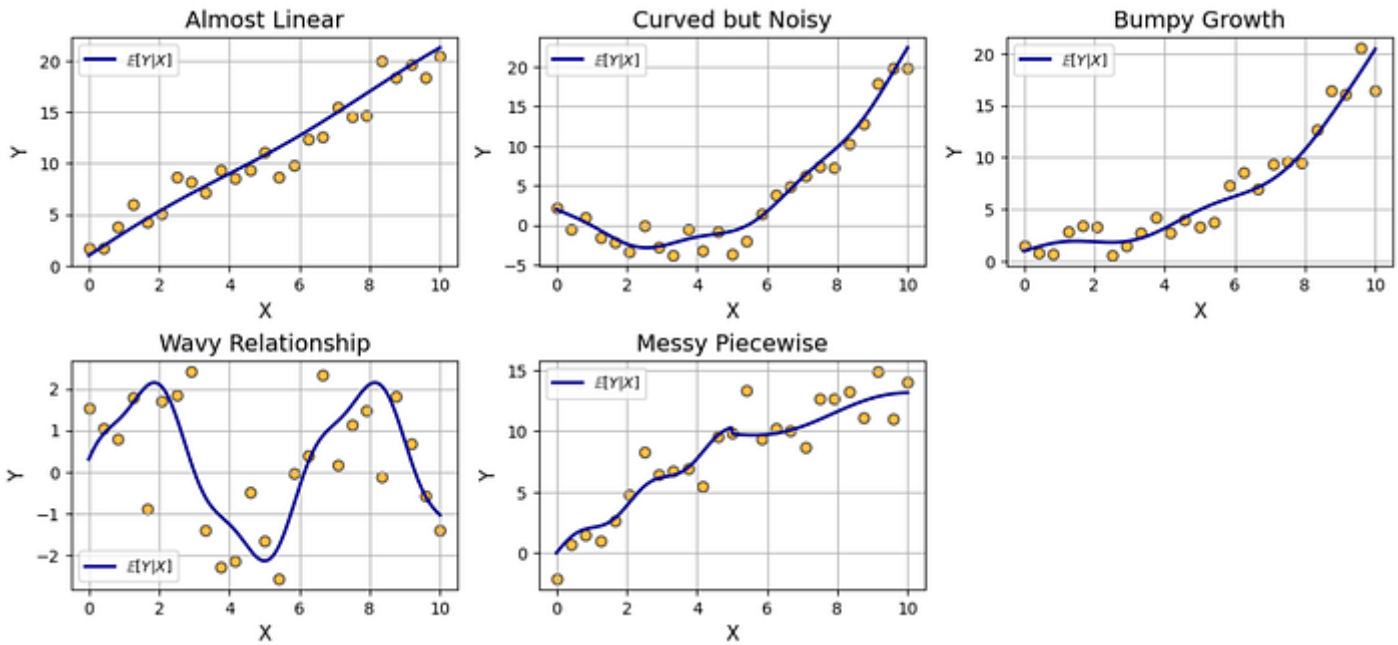

Fig:1"Different Types of Relationships Between X and Y"

Assumptions of Linear Regression

Linearity

The relationship between the independent variables (predictors) and the dependent variable (response) is assumed to be linear. This means that changes in the predictors are associated with proportional changes in the response. Mathematically, the model is expressed as:

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XIV, Issue XI, November 2025

$Y=\beta_0+\beta_1X_1+\beta_2X_2+\cdots+\beta_pX_p+\epsilon$

where the effect of each predictor on Y is additive and linear.

Independence

The error terms (ϵ) should be independent of each other. This assumption ensures that observations are not correlated across time or groups (no autocorrelation). Violation of independence often occurs in time series data or clustered data.

Normality of Errors

The error terms are assumed to be normally distributed with mean zero. This assumption is particularly important for statistical inference such as constructing confidence intervals and hypothesis testing. However, prediction accuracy does not strictly require normality.

Homoskedasticity (Constant Variance of Errors)

The variance of the errors should remain constant across all levels of the predictors. If the error variance changes (heteroskedasticity), standard errors may be biased, leading to invalid significance tests.

No Multicollinearity

The independent variables should not be highly correlated with each other. High multicollinearity inflates variance in the coefficient estimates, making it difficult to interpret the effect of individual predictors. Variance Inflation Factor (VIF) is often used to detect multicollinearity.

From assumptions to estimators a clean derivation

1. The linear model and parametric viewpoint⌊2⌋

We assume the conditional expectation $E[Y|X]$ is linear in the predictors. For a single predictor X the model is

$$Y_i = \beta_0+\beta_1X_i + \varepsilon_i, \quad i=1,\ldots,n,$$

where β_0 is the intercept, β_1 is the slope, and the error terms ε_i satisfy the usual assumptions (independence, mean zero, constant variance, and for inference normality).

This is a **parametric** model because we assume the conditional mean belongs to a family indexed by a finite set of parameters {β_0, β_1 just like assuming a normal distribution is parametrized by mean and variance.

2. Ordinary Least Squares (OLS) objective and normal equations

OLS estimates β_0,β_1 by minimizing the Residual Sum of Squares (RSS):

$$RSS(\beta_0, \beta_1) = f(x) = \sum_{i=1}^{n}(Y_i - \beta_0-\beta_1X_i)^2 .$$

Set partial derivatives to zero:

$$\frac{\partial RSS}{\partial \beta_0} = -2 \sum_{i=1}^{n}(Y_i - \beta_0-\beta_1X_i) = 0,$$

$$\frac{\partial RSS}{\partial \beta_1} = -2 \sum_{i=1}^{n} X_i (Y_i - \beta_0-\beta_1X_i) = 0,$$

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XIV, Issue XI, November 2025

These are the **normal equations**. Solving them gives the familiar closed form OLS solution (in scalar form):

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n} (Xi-\bar{X})(Yi-\bar{Y})}{\sum_{i=1}^{n} (Xi-\bar{X})2}, \quad \beta 0 = \bar{Y} - \beta 1 Xi,$$

Where $\bar{X}$ and $\bar{Y}$ are sample means.

In matrix form (for multiple predictors) the solution is

$$\hat{\beta} = (X^T X)^{-1} X^T y,$$

where X is the design matrix (with a column of ones for the intercept) and y is the response vector.

3. Maximum Likelihood (MLE) and equivalence with OLS under normal errors

Assume the errors ε_i are independent and normally distributed:

$$\varepsilon i \sim N(0, \sigma^2).$$

Then the density of Yi is

$$f(Y_i|X_i, \beta, \sigma 2) = \frac{1}{\sqrt{2\pi\sigma 2}} \exp\left(-\frac{((Yi-\beta 0-\beta 1Xi)2).}{2\sigma 2}\right)$$

The (log) likelihood for the sample is

$$\ell(\beta 0, \beta 1, \sigma^2) = -\frac{n}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma 2} \sum_{i=1}^{n} (Yi - \beta 0 - \beta 1 Xi)^2 .$$

For fixed σ^2, maximizing ℓ with respect to β is equivalent to minimizing the RSS (the second term). Thus, under the Gaussian error assumption, the MLE for β coincides with the OLS estimator. Intuitively: maximizing the likelihood is the same as finding the line that makes the observed residuals as small as possible in squared-error sense.

4. Interpretation: why the assumptions matter

- **Linearity**: if E[Y|X] is not (approximately) linear, the linear model is miss specified and estimates are biased for the true relationship.
- **Independence**: correlated errors (e.g., autocorrelation) break standard error formulas and invalidate usual tests.
- **Normality of errors**: required for exact small-sample inference (t, F tests); large samples often rely on asymptotic.
- **Homoskedasticity**: if error variance changes with X, OLS remains unbiased but standard errors are incorrect unless adjusted (robust SE).

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XIV, Issue XI, November 2025

- **No multicollinearity**: high predictor co linearity inflates variance of $\hat{\beta}$ making coefficient estimates unstable.

5. Numeric illustration residuals and RSS = 6.0

You mentioned an example where the "true" values (or intended/ideal predictions) for six observations are

$$y_{true}=[5, 7, 9, 11, 13, 15].$$

Suppose our model gave predictions that were exactly one unit above or below those true values, so the prediction errors (residuals) are:

$$r= [1, -1, -1, 1, -1, 1].$$

Compute the Residual Sum of Squares (RSS):

$$RSS= \sum r_i^2 = 1^2+(-1)^2 + (-1)^2+1^2+(-1)^2+1^2.$$

Calculate carefully, term by term

$$1+1+1+1+1+1=6.$$

So the RSS is 6.0 as you stated. This simple computation shows how residuals map directly to RSS: each unit-error contributes its square to the objective OLS minimizes.

computing the least squares estimates **manually for a simple linear regression model. code with explanations and final outputs:**

```
import numpy as np

x = np.array([1, 1, 1, 2, 2, 2, 3, 3, 3])
y = np.array([5, 6, 7, 9, 10, 11, 13, 14, 15])

# Least Squares Estimates
beta1 = np.sum((x - np.mean(x)) * (y - np.mean(y))) / np.sum((x - np.mean(x))**2)
beta0 = np.mean(y) - beta1 * np.mean(x)

# Fitting A Line
y_pred = beta0 + beta1 * x
RSS = np.sum((y - y_pred)**2)

Print('statistical optimal predictions: [6., 6., 6., 10., 10., 10., 14., 14., 14.]')

print('Model Prediction:', y_pred)

print('RSS: ', RSS)

statistical optimal predictions: [6., 6., 6., 10., 10., 10., 14., 14., 14.]')

Model Prediction: [ 6.  6.  6. 10. 10. 10. 14. 14. 14.]

RSS:  6.0

importmatplotlib.pyplotasplt
print('Statistical optimal predictions: [6., 6., 6., 10., 10., 10., 14., 14., 14.]')
```

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XIV, Issue XI, November 2025

```
print('Model Prediction:', y_pred)
print('RSS: ', RSS)
```

statistical optimal predictions : [6., 6., 6., 10., 10., 10., 14., 14., 14.]

RSS:6.0

```
importmatplotlib.pyplotasplt
importnumpyasnp
x = np.array([1, 1, 1, 2, 2, 2, 3, 3, 3])
y = np.array([5, 6, 7, 9, 10, 11, 13, 14, 15])
# Least Squares Estimates
beta1 = np.sum((x - np.mean(x)) * (y - np.mean(y))) / np.sum((x - np.mean(x))**2)
beta0 = np.mean(y) - beta1 * np.mean(x)
# Fitting A Line
y_pred = beta0 + beta1 * x
RSS = np.sum((y - y_pred)**2)

print('Statistical optimal predictions: [6., 6., 6., 10., 10., 10., 14., 14., 14.]')

print('Model Prediction:', y_pred)
print('RSS: ', RSS)
```

Statistical optimal predictions: [6., 6., 6., 10., 10., 10., 14., 14., 14.]

Model Prediction: [6. 6. 6. 10. 10. 10. 14. 14. 14.]
RSS: 6.0

```
importmatplotlib.pyplotasplt

print('Statistical optimal predictions: [6., 6., 6., 10., 10., 10., 14., 14., 14.]')
print('Model Prediction:', y_pred)
print('RSS: ', RSS)
```

Statistical optimal predictions: [6., 6., 6., 10., 10., 10., 14., 14., 14.]

Model Prediction: [6. 6. 6. 10. 10. 10. 14. 14. 14.]

RSS:6.0

```
importmatplotlib.pyplotasplt

plt.scatter(x, y, color='royalblue', edgecolor='black', s=50, alpha=0.8, label='Data Points')
plt.plot(x, y_pred, color='darkred', linewidth=2.5, label='Fitted Line')


plt.xlabel("X", fontsize=12)
plt.ylabel("Y", fontsize=12)
plt.title("Linear Regression Fit", fontsize=14)
plt.legend()
plt.grid(True, linestyle='-.', alpha=0.6)
plt.tight_layout()
plt.show()
```

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XIV, Issue XI, November 2025

OUTPUT:

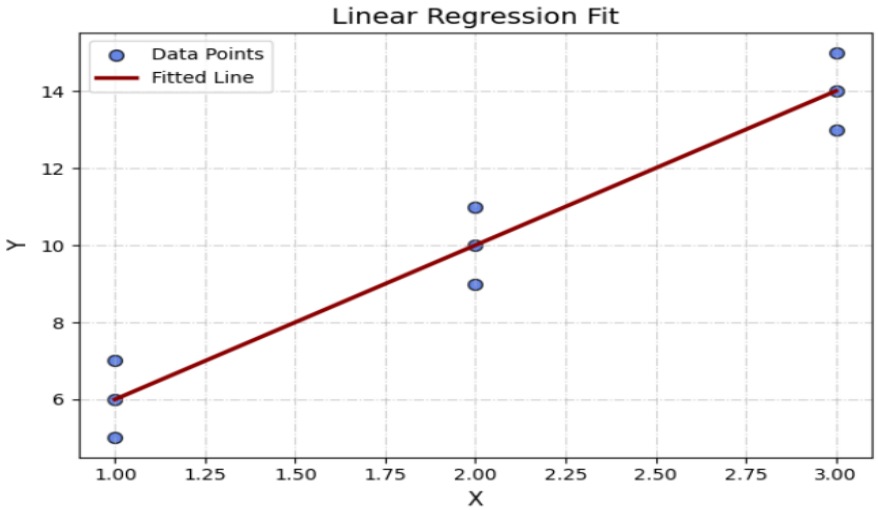

FIG:2"Best Fit Line for the Given Data Using Least Squares Method"

```
residuals = y - y_pred

plt.figure(figsize=(7, 5))

plt.scatter(y_pred, residuals, alpha=0.9, s=50, edgecolor='black', color='royalblue')
plt.axhline(0, color='darkred', linestyle='--', linewidth=1.5, label='Zero Residual Line')

plt.xlabel("Predicted Values", fontsize=12)
plt.ylabel("Residuals", fontsize=12)
plt.title("Residuals vs Predicted Values", fontsize=14)
plt.grid(True, linestyle='-.', linewidth=0.5, alpha=0.7)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.legend()

plt.tight_layout()
plt.show()
```

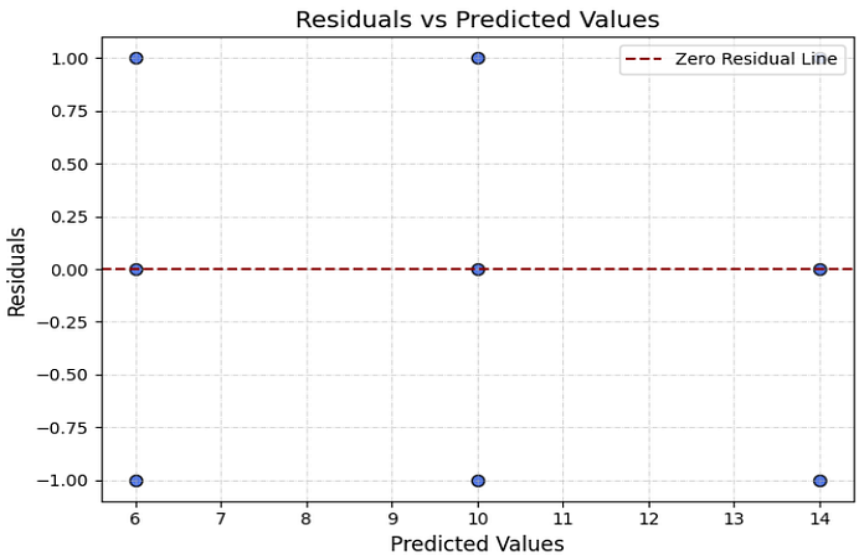

FIG:3""Residuals vs Predicted Values Plot"

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XIV, Issue XI, November 2025

Understanding Least Squares Estimation and R^2 Score⌊3⌋

The way least squares estimates predictions is directly related to the concept of **conditional expectation** E[Y|X] Least squares parameters are designed to **minimize the Residual Sum of Squares (RSS)**, i.e., the squared differences between the observed and predicted values.

Statistically, when multiple yyy values correspond to the same xxx, the best prediction (in the mean-squared error sense) is the **mean of those yyy values** that's why the least squares regression line essentially estimates E[Y|X].

The R^2 Statistic

The **R^2 (coefficient of determination)** measures how well our regression model explains the variability in the dependent variable Y.

We define:

$$R2= 1-\frac{RSS}{TSS}$$

where:

- RSS=∑(yi−y^i)^2 is the **Residual Sum of Squares** (unexplained variability).
- TSS=∑(yi−y¯)^2 is the **Total Sum of Squares**, representing the total variability in Y.

The **base model** (which always predicts the mean y¯\bar{y}y¯) explains no relationship between XXX and YYY. Thus, TSSTSSTSS can be interpreted as the variability explained by this base model compared to having no model at all.

Interpretation of R^2

- R^2-1: The model perfectly explains all variability in Y.
- R^2-0: The model performs no better than predicting the mean y¯
- R^2 <0: The model performs **worse** than the base model (i.e., RSS > TSS).
- R^2=0.5 R^2 = 0.5R2=0.5: The model explains **50% of the total variability** in YYY.

In practice, an R^2 close to 1 indicates a good fit, though perfect fits are rare, especially in real-world data where noise is present.

Finding *R square* of our straight line

```
RSS = np.sum((y - y_pred)**2)
TSS = np.sum((y - np.mean(y))**2)

R_square = 1 - (RSS/TSS)

Print(f'Rsquare: {R_square:.4f}")
```

R square: 0.9412

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XIV, Issue XI, November 2025

Model Interpretation and Statistical Inference⌊4⌋

From our results, the model explains approximately **94% of the variability** in Y compared to the base model, which always predicts the mean value (in this case, 10).

Although linear regression provides a good predictive performance in this instance, using **"accuracy"** as a metric for regression models is conceptually incorrect. Accuracy is suitable for classification tasks, whereas linear regression is a **parametric model** primarily designed for **inference and interpretability** rather than raw predictive power.

A major advantage of linear regression lies in its ability to perform **statistical inference** such as:

- Estimating **confidence intervals** for regression parameters and predictions,
- Conducting **hypothesis testing**,
- Evaluating **p-values** to assess parameter significance, and
- Interpreting model coefficients to understand relationships between variables.

Confidence Interval for the Slope β_1

For our example data, we can derive a **95% confidence interval** for the slope parameter β_1 using its variance formula, without the need for resampling or bootstrapping. The variance of β_1 is given by:

$$\mathrm{Var}(\beta_1) = \frac{\sigma 2}{\sum(xi - \bar{x})2}$$

where:

- $\sigma^2 = \frac{RSS}{n-2}$ is the estimated variance of residuals, and
- n is the number of observations.

Thus, the 95% confidence interval for β_1: is

$$\beta_1 \pm t_{\alpha/2,\, n-2} \sqrt{\mathrm{Var}(\beta 1)}$$

Where $t_{\alpha/2,n-2}$is the critical value from the **Student's t-distribution** with n−2-degrees of freedom.

```
var = (1/(len(y) - 2)) * np.sum((y - y_pred)**2) # error variance

var_beta1 = var/np.sum((x - np.mean(x))**2)
se_beta1 = var_beta1 ** 0.5
alpha = 1 - 0.95
beta1_interval = [beta1 - norm.ppf(1 - alpha/2) * se_beta1, beta1 + norm.ppf(1 - alpha/2) * se_beta1]
print(f"Standard Error of β1 : {se_beta1:.2f}")
print(f"Estimated slope : {beta1}")
print(f"With 95% confidence this interval contains the true slope : [{beta1_interval[0]:.2f},

{beta1_interval[1]:.2f}]")
```

```
Standard Error of β1 : 0.38
Estimated slope : 4.0

With 95% confidence this interval contains the true slope : [3.26, 4.74]
```

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XIV, Issue XI, November 2025

Output:

standard Error of $\beta 1$: 0.38

Estimated slope : 4.0

95% Confidence Interval: [3.26, 4.74]

1. Why the Normality of Errors matters

When we assume:

$\varepsilon_i \sim N(0,\sigma^2)$

it means every error term (the noise around the regression line) is **normally distributed**.

Because the slope β_1 is computed as a **linear combination** of those errors, mathematically:

$\beta_1{}^{\wedge} = \beta 1 + \frac{\sum(xi - x^{-})\varepsilon i}{\sum(xi - x^{-})2}$ and since linear combinations of normal random variables are also normal, it follows that:

$\beta_1{}^{\wedge} \sim N(\beta_1, Var(\beta 1^{\wedge}))$ That's what gives us the *green light* to use the **normal (or t-) distribution** to build confidence intervals and conduct hypothesis tests about β_1.

2. Why we care about the slope's sampling distribution

The slope we estimate from one dataset is just **one draw** from a larger "universe" of possible samples. The sampling distribution of $\beta 1^{\wedge}$ tells us how that slope would vary if we repeated our experiment many times.

So when we say:

text{95% CI for } \beta_1 = [3.26, 4.74]

we're not saying the slope "has a 95% chance" of being in that interval the *true* β_1 is fixed rather, **95% of such intervals built from repeated samples would contain the true β_1**.

That's the essence of **frequentist confidence** the "Confidence is the New Truth" idea.

3. From inference to prediction

Once we have:

$Y^{\wedge} = \beta 0^{\wedge} + \beta 1^{\wedge} X$ we can now use the fitted line to predict new Y values for unseen X values extending our model *beyond* the training data.

This is a major leap:

- The *inference* part (confidence intervals, hypothesis tests) tells us **how certain we are** about the parameters.
- The *prediction* part applies those estimated parameters to **new data** the real goal of modeling.

So yes we don't stop at E[Y|X] (theoretical expectation); we **estimate** it from data to **generalize** it.

In short

- Normal errors ⇒ normal slope ⇒ valid confidence intervals

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XIV, Issue XI, November 2025

- Confidence intervals quantify our uncertainty about parameters
- The model (β_0, β_1) lets us predict for new X values

So, the story flows naturally:

Normality of errors → normal slope estimate → valid confidence interval → reliable predictions.

```
x_new = 5
y_pred_new = beta0 + beta1 * x_new
print(f"For x:{x_new} y is {y_pred_new}")
```

Forx:5y is 22.0

The example we took is very convenient for our least squares estimates to predict near true values since the growth or slope of the means is constant 4. This is easy for a straight line to handle. What if i nudge the values a bit, breaking the constant slope? Like

```
x=np.array([1,1,1,2,2,2,3,3,3])

y=np.array([5,6,7,9,10,11,17,18,19])
```

Fitting a straight line through 6, 10, 18 is not possible with just 3 units distance in x-axis. The RSS increases, The Error Variance Increases, The Standard Error Increases, The Uncertainty Increases, The width of CI increases. Let's see if it happens:

```
x=np.array([1,1,1,2,2,2,3,3,3])
y=np.array([5,6,7,9,10,11,17,18,19])

beta1_m2 = np.sum((x - np.mean(x))*(y - np.mean(y))) / np.sum((x - np.mean(x))**2)
beta0_m2 = np.mean(y) - beta1_m2 * np.mean(x)

y_pred = beta0_m2 + beta1_m2 * x

RSS = np.sum((y - y_pred)**2)

print('Model Prediction:', y_pred)
print(f"RSS : {RSS:.2f}")
print(f'Slope: {beta1_m2}')
```

Model Prediction: [5.33333333 5.33333333 5.33333333 11.33333333 11.33333333 11.33333333

17.33333333 17.33333333 17.33333333]
RSS : 14.00
slope:6.0

```
plt.scatter(x, y, color='royalblue', edgecolor='black', s=50, alpha=0.8, label='Data Points')
plt.plot(x, y_pred, color='darkred', linewidth=2.5, label='Fitted Line')

plt.xlabel("X", fontsize=12)

plt.title("Linear Regression Fit", fontsize=14)

plt.legend()
plt.grid(True, linestyle='-.', alpha=0.6)
```

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XIV, Issue XI, November 2025

```
plt.tight_layout()
plt.show()
```

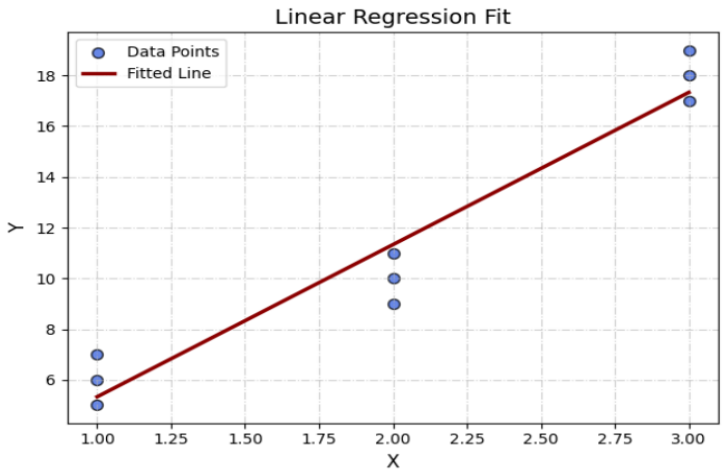

FIG:4"Confidence Is the New Truth: From Slope Estimation to Prediction"

The RSS is increased and we can see the line tried to fit the updated data points with getting as minimum error as possible

```
residuals = y - y_pred

plt.figure(figsize=(7, 5))
plt.scatter(y_pred, residuals, alpha=0.9, s=50, edgecolor='black', color='royalblue')

plt.axhline(0, color='darkred', linestyle='--', linewidth=1.5, label='Zero Residual Line')

plt.xlabel("Predicted Values", fontsize=12)

plt.ylabel("Residuals", fontsize=12)
plt.title("Residuals vs Predicted Values", fontsize=14)
plt.grid(True, linestyle='-.', linewidth=0.5, alpha=0.7)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.legend()
plt.tight_layout()

plt.show()
```

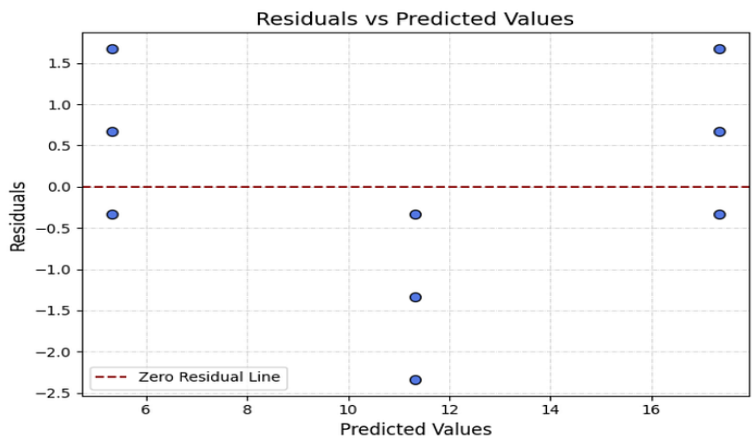

FIG:5"Residuals vs predicted values"

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XIV, Issue XI, November 2025

The straight line is not doing a bad job, it's doing great with its possible physical traits as we can see in the above plot, none of the predicted values have 0 error but it's the minimum error it could get, satisfying all the data points

```
RSS = np.sum((y - y_pred)**2)

TSS = np.sum((y - np.mean(y))**2)
R_square = 1 - (RSS/TSS)

print(f"R square : {R_square:.4f}")
```

R square:0.9391

The *R square* dropped only a little even though the RSS is higher than before, because it is still explaining a lot of variability in the Y values compared to just predicting ȳ.

```
var = (1/(len(y) - 2)) * np.sum((y - y_pred)**2)

var_beta1_m2 = var/np.sum((x - np.mean(x))**2)

se_beta1_m2 = var_beta1_m2 ** 0.5
alpha = 1 - 0.95
beta1_interval = [beta1_m2 - norm.ppf(1 - alpha/2) * se_beta1_m2, beta1_m2 + norm.ppf(1 - alpha/2) *
se_beta1_m2]
print(f"Standard Error of β1 : {se_beta1_m2:.2f}")

print(f"Estimated slope : {beta1_m2}")
print(f"With 95% confidence this interval contains the true slope : [{beta1_interval[0]:.2f}

, {beta1_interval[1]:.2f}]")
```

Standard Error of β1 : 0.58
Estimated slope : 6.0

With 95% confidence this interval contains the true slope : [4.87, 7.13]

Model B has extra residuals compared to model A, and if those residuals increase the spread of errors, they raise the variance. Higher variance means a higher standard Error (as it did for 0.58), which indicates that model B's β estimates leave more unexplained variation than model A. Higher SE can also increases the width of CI with same confidence.

Take aways until now: ⌊5⌋

- In linear regression, we assume normality of errors so that the slope's sampling distribution is normal, enabling valid normal-based confidence intervals.
- The fitted slope and intercept let us predict on new data, unlike simply stating E[Y|X] for our sample.
- The residual sum of squares (RSS) is what we minimize in least squares — and under normal errors, this is equivalent to maximizing the likelihood.
- Extra residuals can increase the spread of errors, raising variance and standard deviation, which means the model's β estimates are leaving more variation unexplained.

Let's see how a straight line is performing compared to E[Y|X] on real world data

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XIV, Issue XI, November 2025

```
#  LINEAR REGRESSION

import pandas aspd
df = pd.read_csv('weight-height.csv', usecols=['Height', 'Weight'])

train_samples = int(len(df) * 0.7)

samples_idx = np.random.choice(len(df), size=train_samples, replace=False)
df_train = df.iloc[samples_idx]
df_test = df.drop(df.index[samples_idx])

X_train = df_train['Height'].to_numpy()
Y_train = df_train['Weight'].to_numpy()
X_test = df_test['Height'].to_numpy()

Y_test = df_test['Weight'].to_numpy()
X_mean = np.mean(X_train)
Y_mean = np.mean(Y_train)

beta1 = np.sum((X_train - X_mean) * (Y_train - Y_mean)) / np.sum((X_train - X_mean)**2)
beta0 = Y_mean - beta1 * X_mean

Y_pred = beta0 + beta1 * X_train

# Model with E[Y|X] near performance
fromscipy.statsimportbinned_statistic

# Bin the heights and compute average weight in each bin -> E[Y|X]

bin_means, bin_edges, _ = binned_statistic(X_train, Y_train, statistic='mean', bins=50)

# Get the centers of the bin for plotting accuracy

bin_centers = (bin_edges[:-1] + bin_edges[1:]) / 2
```

Binned statistics group nearby x values into bins, since in regression data we rarely get identical x values repeating. This is about as close as we can get to replicating E[Y|X].

```
plt.figure(figsize=(8, 5) )
plt.scatter(X_train, Y_train, alpha=0.1, label='Training Data')

plt.plot(bin_centers, bin_means, color='black', linewidth=2, label='Estimated E[Y|X] (bin average)')
plt.plot(np.sort(X_train), np.sort(Y_pred), color='red', alpha=0.7,linewidth=2, label='Linear

Regression')
plt.xlabel('Height')
plt.ylabel('Weight')

plt.legend()
plt.grid(True)
plt.title('Estimated E[Y|X] vs Linear Regression')

plt.show()
```

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XIV, Issue XI, November 2025

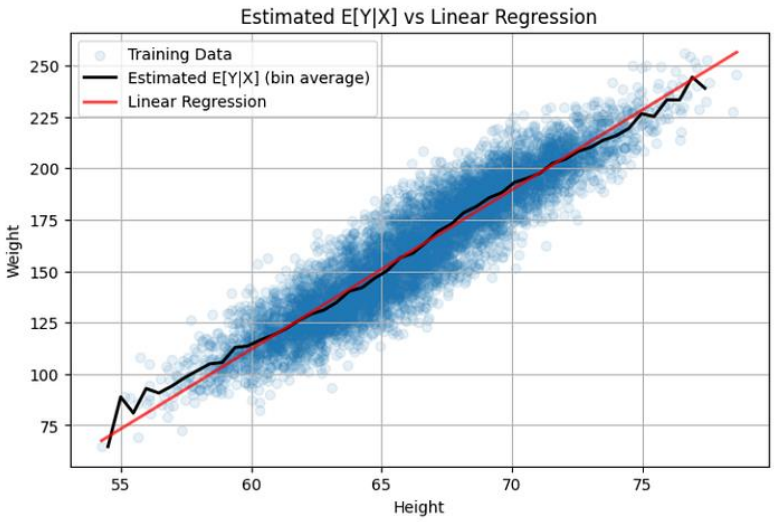

FIG:6“ Estimated E[Y|X] **vs Regression”**

The linear regression model (red line) closely follows the binned estimate of E[Y|X](black line) across the full range of heights, indicating that the linearity assumption holds strongly for this dataset. The small deviations of the binned curve from the straight line are minor and mostly due to local fluctuations in the data rather than systematic bias. The tight clustering of points around both lines suggests low variance and high predictive accuracy, with no visible pattern in residual spread. The model captures the underlying relationship between height and weight very effectively.

```
RSS = np.sum((Y_train - Y_pred)**2)

TSS = np.sum((Y_train - Y_mean)**2)

R_square = 1 - (RSS/TSS)

print(f"R square : {R_square:.4f}")
```

R square: 0.8562

The linear regression model capturing around 85% of the variability in the data compared to base model, that’s not bad at all. Let’s see if our assumption of normality in error holds for this data:

```
importseabornassns

residuals = Y_train - Y_pred
plt.figure(figsize=(8, 5))

sns.histplot(residuals, bins=35, kde=True, color="skyblue", edgecolor="black", alpha=0.7)

plt.title("Residuals Distribution", fontsize=16, fontweight='bold')

plt.xlabel("Residual", fontsize=14)
plt.ylabel("Frequency", fontsize=14)

plt.axvline(0, color='red', linestyle='--', linewidth=1.5, label="Zero Residual Line")

plt.legend()
plt.grid()

plt.show()
```

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XIV, Issue XI, November 2025

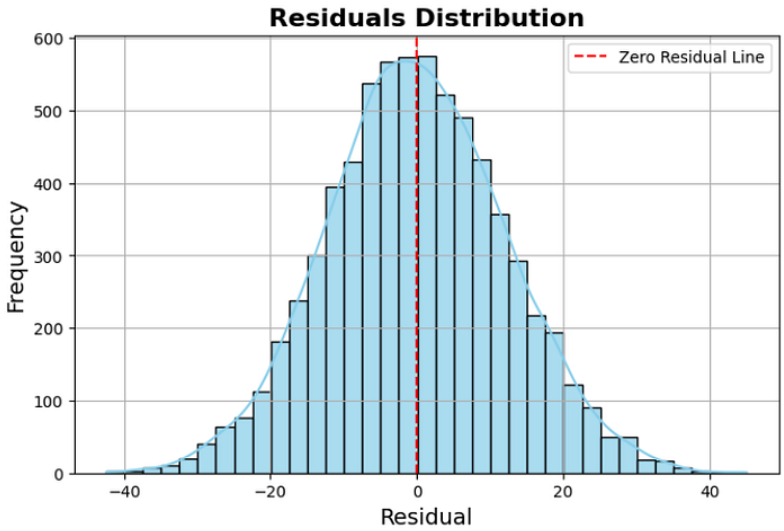

FIG:7“ Residuals Distribution”

The residuals appear approximately normally distributed, which supports our normality of errors assumption. This means we can confidently apply statistical inference such as confidence intervals and hypothesis tests on our model parameters. While normality of errors ensures our parameter estimates follow the distributions we rely on for inference, there’s another equally important assumption: **homoskedasticity**

Homoskedasticity: [6]

Homoskedasticity means the variance of the errors should be constant across all values of X. If the spread of residuals changes with X (a “fan” or “cone” shape in a residual plot), we have **heteroskedasticity**, which can lead to unreliable standard errors and misleading confidence intervals even if the errors are normally distributed. Which also means error variance (σ^2) is only valid for inference(we used error variance in past to get variance of $\beta 1$) if and only if the variance of the errors at each data point x is constant. Below plots show the text book definition of homoskedasticity and heteroskedasticity.

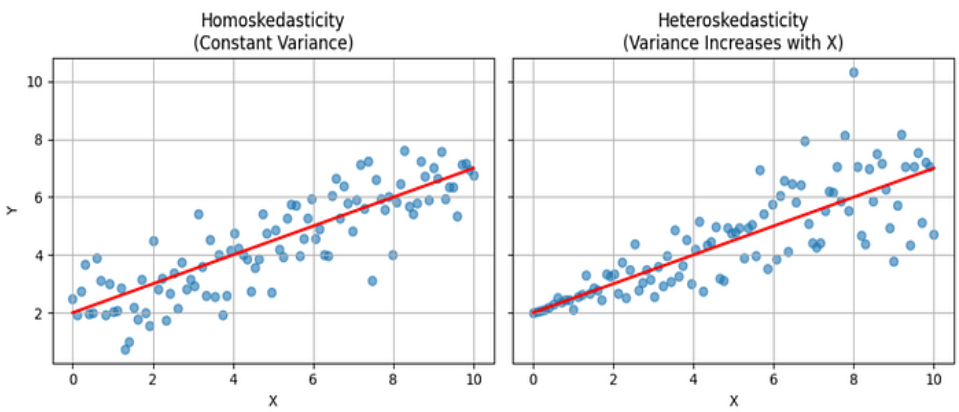

FIG:8“Homoskedasticity heteroskedasticity

Example[7]

For examle, take two x values”from the heteroskedasticity plot: x1 = 2 and x2 = 8. After prediction, suppose we get $\hat{y}1 = 3$ and $\hat{y}2 = 6$, while the actual values are y1 = 2.5 and y2 = 4.

If we construct confidence intervals for these predictions using the error variance from heteroskedastic data, the results may be misleading. Because the variance of errors changes with x, the CI for $\hat{y}1$ might be unnecessarily wide (overestimating uncertainty), while the CI for $\hat{y}2$ might be too narrow (underestimating uncertainty).

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XIV, Issue XI, November 2025

In contrast, with homoskedasticity, the error variance is constant and stable across all x values, so doing inference is more reliable. This is the reason why we **assume homoskedasticity** in linear regression

Note: there are statistical techniques to address heteroskedasticity, but some real world data — such as housing prices, where higher-priced homes have more variability — naturally exhibit it.

We can visualize doing a plot of either (Xvs Residuals) or (Y predictions vs residuals) since ŷ looks similar to x and in multiple regression(regression with more than 1 variable) ŷ can be our go to.

```
plt.figure(figsize=(8, 5))

sns.scatterplot(Y_pred, residuals, alpha=0.6, edgecolor=None)

plt.axhline(0, color='red', linestyle='--', linewidth=1.5)
plt.xlabel("Predicted Values", fontsize=12)

plt.ylabel("Residuals", fontsize=12)

plt.title("Residuals vs Predicted Values", fontsize=14, fontweight='bold')

plt.tight_layout()
plt.show()
```

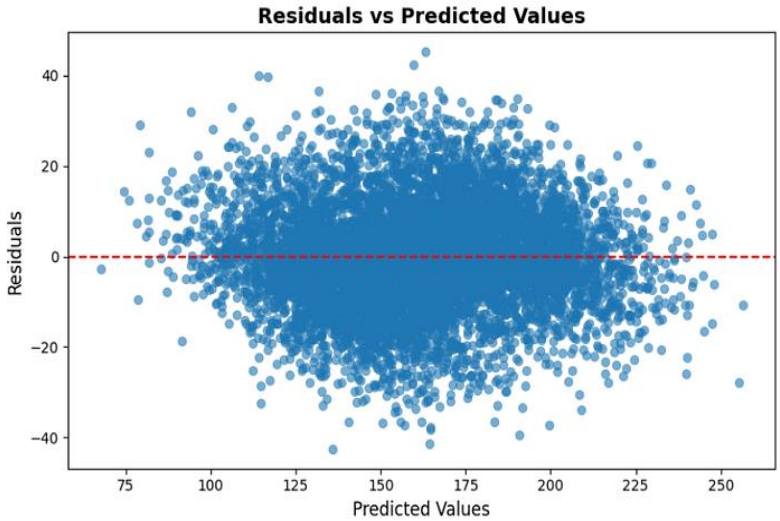

FIG:9 "Residuals vs predicted values"

I don't see any patterns or errors expanding with predicted values all the errors are scattered around the 0 erroredline, so we don't have heteroskedastic data

```
VAR = (1/(len(Y_train) - 2)) * np.sum((Y_train - Y_pred)**2)

VAR_BETA1 = VAR / np.sum((X_train - X_mean)**2)

SE_BETA1 = VAR_BETA1 ** 0.5

print(f'A valid variance due to homoskedasticity : {VAR:.4f}')
print(f"Standard Error of parameter β1: {SE_BETA1 :.4f}")
```

A valid variance due to homoskedasticity : 148.0742
Standard Error of parameter β1: 0.0376

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XIV, Issue XI, November 2025

Finding the true β1 (true slope) on true height and weights data would look like:

```
alpha = 1 - 0.95
beta1_interval = beta1 - norm.ppf(1 - alpha/2)* SE_BETA1, beta1 + norm.ppf(1 - alpha/2) * SE_BETA1

print(f"Our Estimate of true slope : {beta1:.2f}")

print(f"With 95% confidence this interval contains the true slope : [{beta1_interval[0]:.2f}

{beta1_interval[1]:.2f}]")
```

Our Estimate of true slope : 7.71

With 95% confidence this interval contains the true slope : [7.63, 7.78]

Now we got slope and intercept we can predict weight given height

```
X_new = 73.5
Y_new = beta0 + beta1 * X_new
SE_Y_new = np.sqrt(
   VAR * (1 + 1/len(X_train) + (X_new - X_mean)**2 / np.sum((X_train - X_mean)**2)) )
alpha = 1 - 0.95

z = norm.ppf(1 - alpha/2)

ci_lower = Y_new - z * SE_Y_new
ci_upper = Y_new + z * SE_Y_new

print(f'For height of {X_new} the predicted weight is: {Y_new:.2f}')

print(f"95% CI for prediction at X={X_new}: [{ci_lower:.2f}, {ci_upper:.2f}]")
```

For height of 73.5 the predicted weight is: 216.42

95% CI for prediction at X=73.5: [192.56, 240.28]

The standard error of a new prediction shows how uncertain that prediction is. It combines the natural spread of data points around the regression line, the fact that we estimated the model from limited data, and how far the new input is from the average of what the model has seen before. Predictions for values far from the average come with more uncertainty. We use this standard error to build confidence or prediction intervals ranges where the actual value is likely to fall

Now let's compare our parameters and *R square* score with Linear Regression() in scikit-learn library

```
fromsklearn.linear_modelimportLinearRegression

model = LinearRegression()

model.fit(X_train.reshape(-1, 1), Y_train)

 print(f"β1 (sklearn): {model.coef_[0]:.4f}")

 print(f"β0 (sklearn): {model.intercept_:.4f}")

 print(f"β1 (manual): {beta1:.4f}")
```

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XIV, Issue XI, November 2025

```
print(f"β0 (manual): {beta0:.4f}")
Y_test_pred = model.predict(X_test.reshape(-1, 1))

rss = np.sum((Y_test - Y_test_pred) ** 2)
tss = np.sum((Y_test - np.mean(Y_test)) ** 2)

model_r2_score = 1 - (rss / tss)

print(f"Model's R square score (sklearn): {model_r2_score:.4f}")

Y_test_pred = beta0 + beta1 * X_test

RSS_test = np.sum((Y_test - Y_test_pred) ** 2)
TSS_test = np.sum((Y_test - np.mean(Y_test)) ** 2)

R_square_test = 1 - (RSS_test / TSS_test)

print(f"Our R square score (manual): {R_square_test:.4f}")
```

β1 (sklearn): 7.7125

β0 (sklearn): -350.4167

β1 (manual): 7.7125

β0 (manual): -350.4167

Model's R square score (sklearn): 0.8611

Our R square score (manual): 0.8625

Both sklearn'sLinear Regression and our manual calculation gave almost the exact same slope (β1) and intercept (β0). That means our manual method is spot on. Looking at how the model does on new data (the test set), the R square scores are super close too sklearn's is 0.8611 and ours is 0.8625. The tiny difference is just from small rounding differences. This shows both methods predict almost equally well. The model explains about 86% of the variation in weight based on height, which is pretty solid. So, our manual math works great, but sklearn makes things easier and less error-prone when you're working with bigger data or more features

We just wrapped up simple linear regression where we modeled the relationship between one predictor and one response. But real-world problems rarely rely on just one factor. That's where multiple linear regressionscomes in. It lets us bring in many variables, tease apart their effects, and build more powerful models. But heads up: when you add more predictors, new challenges pop up. One biggie is **multicollinearity** when predictors are highly correlated with each other. This can confuse the model, inflate errors, and make your coefficient estimates unstable and unreliable. In the next blog, I'll walk you through how multiple linear regression works, why multicollinearity matters, how to detect it, and what you can do about it. We'll also dig into interpreting coefficients when predictors aren't independent and how to keep your model solid

CONCLUSION:

Linear Regression is a widely used technique used in many branches of science and technology. It is a core topic in Machine Learning and Data Science, two very popular fields that have found a wide range of applications. This article is a self-contained description of Linear Regression, the required Linear Algebra, and the required Python for its implementation.

Linear regression stands as a foundational pillar in statistical modeling and machine learning providing a powerful yet interpretable method for unraveling relationships between variables. Its widespread use across data

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XIV, Issue XI, November 2025

science, from predictive analytics to causal inference, stems from its ability to model linear dependence between a dependent variable and one or more independent variables. This comprehensive guide offers a practical, step-by-step journey through the core concepts of linear regression, its applications, and best practices, catering to both beginners and seasoned data

Linear regression remains a powerful tool for understanding and predicting relationships between variables. Whether analyzing economic trends or building predictive models, its simplicity and interpretability make it indispensable in the realm of data science and beyond. Understanding its principles equips analysts with essential skills to derive meaningful insights from data.

In summary, from theory to application, linear regression serves as a cornerstone in statistical modeling, bridging the gap between data and actionable insights.

To explore more about machine learning techniques and my Data Science Journey connect with me on *LinkedIn*. and check out my latest projects on *GitHub*. Let's stay connected to continue the journey of mastering data science and machine learning together!

Author Contributions: These authors contributed equally to all aspects of this work. All authors have read and agreed to the published version of the manuscript

Funding: This research received no external funding

Acknowledgments: The authors express their sincere gratitude to the unknown reviewers for their detailed reading and valuable advice.

Conflicts of Interest: The authors declare no conflict of interest

REFERENCES

1. Draper, N. R., & Smith, H. (1998). Applied Regression Analysis (3rd ed.). Wiley-Interscience. https://doi.org/10.1002/9781118625590
2. Freedman, D. A. (2009). Statistical Models: Theory and Practice (Revised ed.). Cambridge University Press. https://doi.org/10.1017/CBO9780511815867
3. Hastie, T., Tibshirani, R., & Friedman, J. (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction (2nd ed.). Springer. https://doi.org/10.1007/978-0-387-84858-7
4. Seber, G. A. F., & Lee, A. J. (2012). Linear Regression Analysis (2nd ed.). Wiley. https://doi.org/10.1002/9781118274422
5. Montgomery, D. C., Peck, E. A., & Vining, G. G. (2021). Introduction to Linear Regression Analysis (6th ed.). Wiley. https://doi.org/10.1002/9781119722106
6. Kutner, M. H., Nachtsheim, C. J., Neter, J., & Li, W. (2005). Applied Linear Statistical Models (5th ed.). McGraw-Hill/Irwin.
7. James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). An Introduction to Statistical Learning: With Applications in R and Python (2nd ed.). Springer. https://doi.org/10.1007/978-1-0716-1418-1