Linear Regression in ML

**Linear Regression in Machine learning**

Linear regression is a type of [**supervised machine-learning algorithm**](https://www.geeksforgeeks.org/supervised-machine-learning/) that learns from the labelled datasets and maps the data points with most optimized linear functions which can be used for prediction on new datasets. It assumes that there is a linear relationship between the input and output, meaning the output changes at a constant rate as the input changes. This relationship is represented by a straight line.

**For example** we want to predict a student's exam score based on how many hours they studied. We observe that as students study more hours, their scores go up. In the example of predicting exam scores based on hours studied. Here

* **Independent variable (input):** Hours studied because it's the factor we control or observe.
* **Dependent variable (output):**Exam score because it depends on how many hours were studied.

We use the independent variable to predict the dependent variable.

**Best Fit Line in Linear Regression**

In linear regression, the best-fit line is the straight line that most accurately represents the relationship between the independent variable (input) and the dependent variable (output). It is the line that minimizes the difference between the actual data points and the predicted values from the model.

**1. Goal of the Best-Fit Line**

The goal of linear regression is to find a straight line that minimizes the error (the difference) between the observed data points and the predicted values. This line helps us predict the dependent variable for new, unseen data.



Linear Regression

Here Y is called a dependent or target variable and X is called an independent variable also known as the predictor of Y. There are many types of functions or modules that can be used for regression. A linear function is the simplest type of function. Here, X may be a single feature or multiple features representing the problem.

**2. Equation of the Best-Fit Line**

For simple linear regression (with one independent variable), the best-fit line is represented by the equation

*y=mx+b*

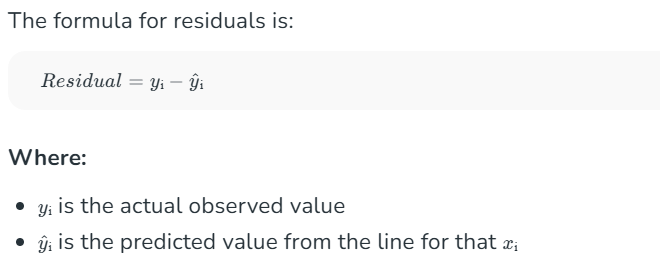
**Where:**

* **y** is the predicted value (dependent variable)
* **x** is the input (independent variable)
* **m** is the slope of the line (how much y changes when x changes)
* **b** is the intercept (the value of y when x = 0)

The best-fit line will be the one that optimizes the values of m (slope) and b (intercept) so that the predicted y values are as close as possible to the actual data points.

**3. Minimizing the Error: The Least Squares Method**

To find the best-fit line, we use a method called [**Least Squares**](https://www.geeksforgeeks.org/least-square-method/). The idea behind this method is to minimize the sum of squared differences between the actual values (data points) and the predicted values from the line. These differences are called residuals.

**

The least squares method minimizes the sum of the squared residuals:

**

This method ensures that the line best represents the data where the sum of the squared differences between the predicted values and actual values is as small as possible.

**4. Interpretation of the Best-Fit Line**

* **Slope (m):** The slope of the best-fit line indicates how much the dependent variable (y) changes with each unit change in the independent variable (x). For example if the slope is 5, it means that for every 1-unit increase in x, the value of y increases by 5 units.
* **Intercept (b):** The intercept represents the predicted value of y when x = 0. It’s the point where the line crosses the y-axis.

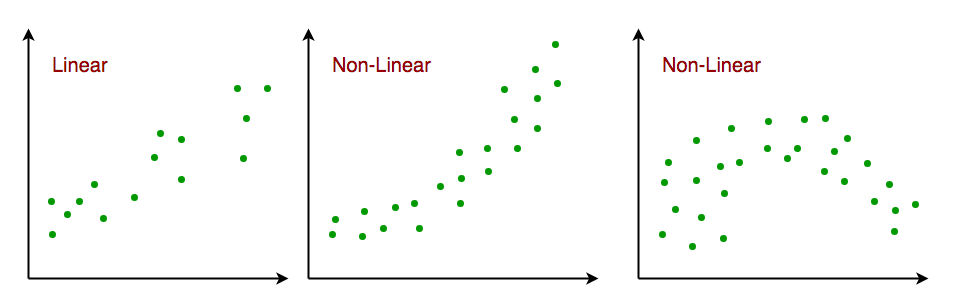
In linear regression some hypothesis are made to ensure reliability of the model's results.

***Limitations***

* ***Assumes Linearity:*** *The method assumes the relationship between the variables is linear. If the relationship is non-linear, linear regression might not work well.*
* ***Sensitivity to Outliers:*** *Outliers can significantly affect the slope and intercept, skewing the best-fit line.*

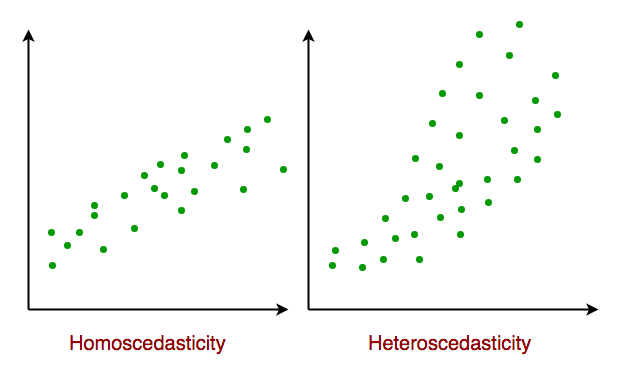
**Assumptions of the Linear Regression**

**1. Linearity**: The relationship between inputs (X) and the output (Y) is a straight line.

Linearity

**2. Independence of Errors**: The errors in predictions should not affect each other.

**3. Constant Variance (Homoscedasticity):**The errors should have equal spread across all values of the input. If the spread changes (like fans out or shrinks), it's called heteroscedasticity and it's a problem for the model.

Homoscedasticity

**4. Normality of Errors**: The errors should follow a normal (bell-shaped) distribution.

**5. No Multicollinearity*(*for multiple regression)**: Input variables shouldn’t be too closely related to each other.

**6. No Autocorrelation**: Errors shouldn't show repeating patterns, especially in time-based data.

**7. Additivity**: The total effect on Y is just the sum of effects from each X, no mixing or interaction between them.'

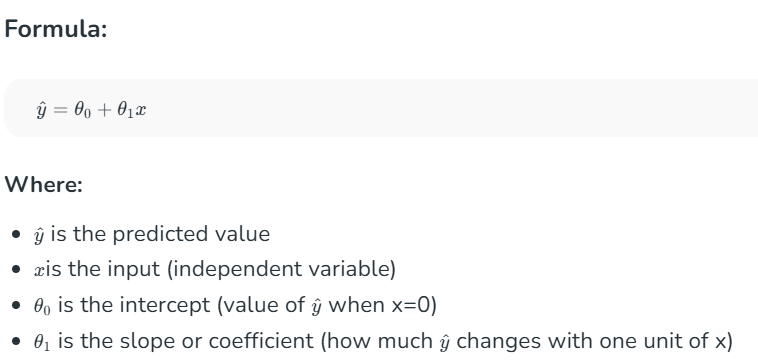
*To understand Multicollinearityin detail refer to article:* [***Multicollinearity***](https://www.geeksforgeeks.org/multicollinearity-in-regression-analysis/)*.*

**Types of Linear Regression**

When there is only one independent feature it is known as Simple Linear Regression or [Univariate Linear Regression](https://www.geeksforgeeks.org/univariate-linear-regression-in-python/) and when there are more than one feature it is known as Multiple Linear Regression or [Multivariate Regression](https://www.geeksforgeeks.org/multivariate-regression/).

**1. Simple Linear Regression**

[Simple linear regression](https://www.geeksforgeeks.org/simple-linear-regression-in-python/) is used when we want to predict a target value (dependent variable) using only one input feature (independent variable). It assumes a straight-line relationship between the two.

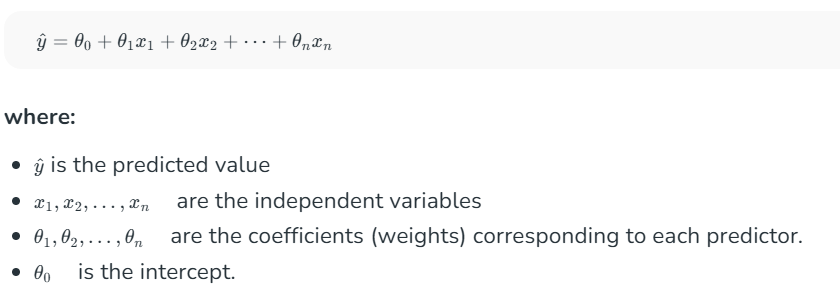
****

**Example:**

Predicting a person’s salary (y) based on their years of experience (x).

**2. Multiple Linear Regression**

[Multiple linear regression](https://www.geeksforgeeks.org/ml-multiple-linear-regression-using-python/) involves more than one independent variable and one dependent variable. The equation for multiple linear regression is:

**

The goal of the algorithm is to find the best Fit Line equation that can predict the values based on the independent variables.

In regression set of records are present with X and Y values and these values are used to learn a function so if you want to predict Y from an unknown X this learned function can be used. In regression we have to find the value of Y, So, a function is required that predicts continuous Y in the case of regression given X as independent features.

**Use Case of Multiple Linear Regression**

Multiple linear regression allows us to analyze relationship between multiple independent variables and a single dependent variable. Here are some use cases:

* **Real Estate Pricing:** In real estate MLR is used to predict property prices based on multiple factors such as location, size, number of bedrooms, etc. This helps buyers and sellers understand market trends and set competitive prices.
* **Financial Forecasting:** Financial analysts use MLR to predict stock prices or economic indicators based on multiple influencing factors such as interest rates, inflation rates and market trends. This enables better investment strategies and risk management24.
* **Agricultural Yield Prediction:** Farmers can use MLR to estimate crop yields based on several variables like rainfall, temperature, soil quality and fertilizer usage. This information helps in planning agricultural practices for optimal productivity
* **E-commerce Sales Analysis:** An e-commerce company can utilize MLR to assess how various factors such as product price, marketing promotions and seasonal trends impact sales.

Now that we have understood about linear regression, its assumption and its type now we will learn how to make a linear regression model.

**Cost function for Linear Regression**

As we have discussed earlier about best fit line in linear regression, its not easy to get it easily in real life cases so we need to calculate errors that affects it. These errors need to be calculated to mitigate them. The difference between the predicted value Y hat (y^)     and the true value Y and it is called [cost function](https://www.geeksforgeeks.org/what-is-cost-function/) or the[loss function](https://www.geeksforgeeks.org/ml-common-loss-functions/).

In Linear Regression, the Mean Squared Error (MSE) cost function is employed, which calculates the average of the squared errors between the predicted values y*i hat*​ and the actual values yi. The purpose is to determine the optimal values for the intercept θ1*θ*1​ and the coefficient of the input feature θ2*θ*2​ providing the best-fit line for the given data points. The linear equation expressing this relationship is .

MSE function can be calculated as:

**

Utilizing the MSE function, the iterative process of gradient descent is applied to update the values of \θ1&θ2*θ*1​&*θ*2​. This ensures that the MSE value converges to the global minima, signifying the most accurate fit of the linear regression line to the dataset.

This process involves continuously adjusting the parameters \(\theta\_1\) and \(\theta\_2\) based on the gradients calculated from the MSE. The final result is a linear regression line that minimizes the overall squared differences between the predicted and actual values, providing an optimal representation of the underlying relationship in the data.

Now we have calculated loss function we need to optimize model to mtigate this error and it is done through gradient descent.

**Evaluation Metrics for Linear Regression**

A variety of [evaluation measures](https://www.geeksforgeeks.org/metrics-for-machine-learning-model/) can be used to determine the strength of any linear regression model. These assessment metrics often give an indication of how well the model is producing the observed outputs.

The most common measurements are:

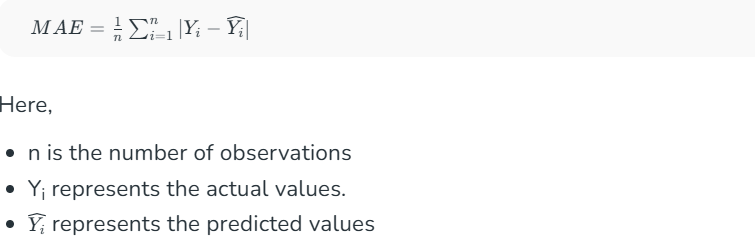
**Regression Evaluation Metrics**

In the regression task, we are supposed to predict the target variable which is in the form of continuous values. To evaluate the performance of such a model below mentioned evaluation metrics are used:

* [**Mean Absolute Error**](https://www.geeksforgeeks.org/how-to-calculate-mean-absolute-error-in-python)
* [**Mean Squared Error**](https://www.geeksforgeeks.org/python-mean-squared-error)
* [**Root Mean Square Error**](https://www.geeksforgeeks.org/root-mean-square-error-in-r-programming)
* **Root Mean Square Logarithmic Error**
* [**R2 - Score**](https://www.geeksforgeeks.org/python-coefficient-of-determination-r2-score)

**Mean Absolute Error (MAE)**

Mean Absolute Error(MAE) is the average distance between predicted and original values. Basically, it gives how we have predicted from the actual output. However, there is one limitation i.e. it doesn't give any idea about the direction of the error which is whether we are under-predicting or over-predicting our data. It can be represented mathematically in this way:



Lower MAE value indicates better model performance. It is not sensitive to the outliers as we consider absolute differences.

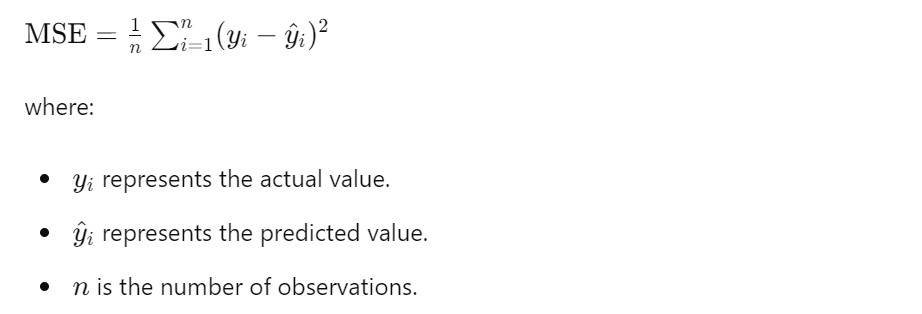
**Mean Squared Error (MSE)**

MSE is similar to mean absolute error but the difference is it takes the square of the average of between predicted and original values. The main advantage to take this metric is here, it is easier to calculate the gradient whereas, in the case of mean absolute error, it takes complicated programming tools to calculate the gradient. By taking the square of errors it pronounces larger errors more than smaller errors, we can focus more on larger errors. It can be expressed mathematically in this way.



What is Mean Squared Error?

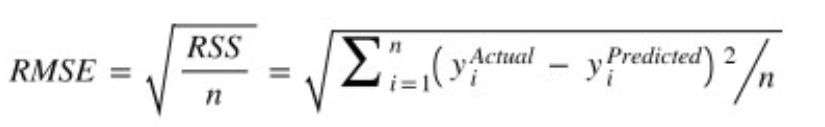
The mean squared error is the average of the squared differences between the expected and actual values. The mathematical notation for it is as follows:



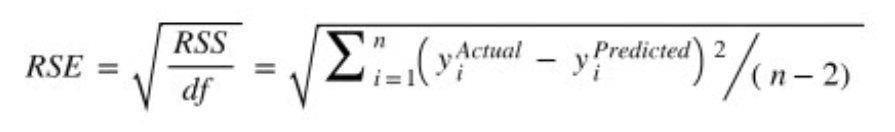
The squaring of errors ensures that positive and negative differences do not cancel each other out. Additionally, squaring emphasizes larger errors, making MSE sensitive to outliers.

**Root Mean Square Error (RMSE)**

The Root Mean Squared Error is the square root of the variance of the residuals. It specifies the absolute fit of the model to the data i.e. how close the observed data points are to the predicted values. Mathematically it can be represented as,



To make this estimate unbiased, one has to divide the sum of the squared residuals by the **degrees of freedom** rather than the total number of data points in the model. This term is then called the **Residual Standard Error(RSE)**. Mathematically it can be represented as,



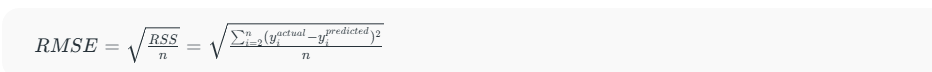
R-squared is a better measure than RSME. Because the value of Root Mean Squared Error depends on the units of the variables (i.e. it is not a normalized measure), it can change with the change in the unit of the variables.

**Root Mean Squared Logarithmic Error (RMSLE)**

There are times when the target variable varies in a wide range of values. And hence we do not want to penalize the overestimation of the target values but penalize the underestimation of the target values. For such cases, RMSLE is used as an evaluation metric which helps us to achieve the above objective.

Some changes in the original formula of the RMSE code will give us the RMSLE formula that is as shown below:

The square root of the residuals' variance is the [Root Mean Squared Error](https://www.geeksforgeeks.org/root-mean-square-error-in-r-programming/). It describes how well the observed data points match the expected values or the model's absolute fit to the data.  
In mathematical notation, it can be expressed as:

**

Rather than dividing the entire number of data points in the model by the number of degrees of freedom, one must divide the sum of the squared residuals to obtain an unbiased estimate. Then, this figure is referred to as the Residual Standard Error (RSE).

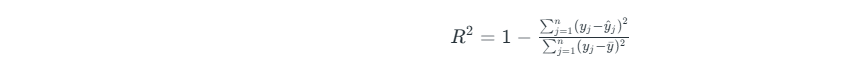
In mathematical notation, it can be expressed as:

**

RSME is not as good of a metric as R-squared. Root Mean Squared Error can fluctuate when the units of the variables vary since its value is dependent on the variables' units (it is not a normalized measure).

**R2 - Score**

The coefficient of determination also called the R2 score is used to evaluate the performance of a linear regression model. It is the amount of variation in the output-dependent attribute which is predictable from the input independent variable(s). It is used to check how well-observed results are reproduced by the model, depending on the ratio of total deviation of results described by the model.



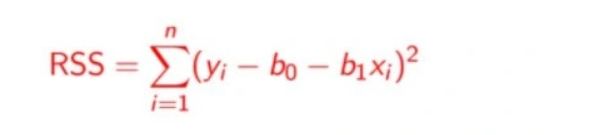
**Coefficient of Determination or R-Squared (R2)**

R-squared is a number that explains the amount of variation that is explained/captured by the developed model. It always ranges between 0 & 1 . Overall, the higher the value of R-squared, the better the model fits the data.

Mathematically it can be represented as,

**R2 = 1 – ( RSS/TSS )**

* **Residual sum of Squares (RSS)** is defined as the sum of squares of the residual for each data point in the plot/data. It is the measure of the difference between the expected and the actual observed output.

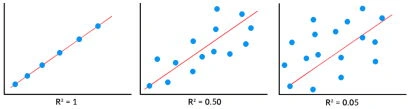


* **Total Sum of Squares (TSS)** is defined as the sum of errors of the data points from the mean of the response variable. Mathematically TSS is,

Total Sum of Squares

where y hat is the mean of the sample data points.

The significance of R-squared is shown by the following figures,



**Understanding About the Linear Regression**

Linear Regression is a statistical method used to model the relationship between a dependent variable and one or more independent variables by fitting a straight line (linear equation) to the observed data.

The goal is to find the best-fit line that minimizes the difference between the predicted and actual values.

**Simple Formula (for one variable):**

*y*=*mx*+*b*

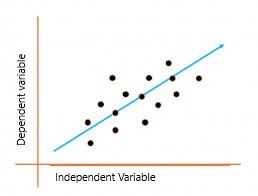
* *y* = Dependent variable (output)
* x*x* = Independent variable (input)
* m*m* = Slope of the line
* b*b* = Y-intercept

It’s used for **predicting numerical values**, like house prices, sales forecasts, etc.

**Simple Linear Regression**

In a simple linear regression, there is one independent variable and one dependent variable. The model estimates the slope and intercept of the line of best fit, which represents the relationship between the variables. The slope represents the change in the dependent variable for each unit change in the independent variable, while the intercept represents the predicted value of the dependent variable when the independent variable is zero.

Linear regression is a quiet and the simplest statistical regression technique used for predictive analysis in [**machine learning**](https://www.analyticsvidhya.com/blog/2022/01/machine-learning-algorithms/). It shows the linear relationship between the independent(predictor) variable i.e. X-axis and the dependent (output) variable i.e. Y-axis, called linear regression. If there is a single input variable X (independent variable), such linear regression is ***simple linear regression***.



The graph above presents the linear relationship between the output(y) and predictor(X) variables. The blue line is referred to as the best-fit straight line. Based on the given data points, we attempt to plot a line that fits the points the best.

Simple Regression Calculation

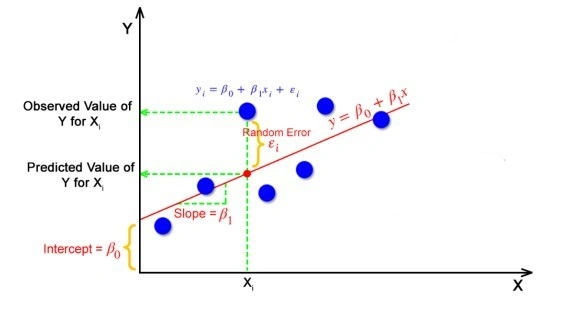
To calculate best-fit line linear regression uses a traditional slope-intercept form which is given below,

**Y i =  β 0 + β 1 X i**

* where Y i  = Dependent variable,  **β 0** = constant/Intercept, **β 1** = Slope/Intercept, **X i** = Independent variable.
* This algorithm explains the linear relationship between the dependent(output) variable y and the independent(predictor) variable X using a straight line  Y= B 0 + B 1 X.

But how does the regression find out which is the best-fit line?

The goal of the linear regression algorithm is to get the **best values for B 0 and B 1** to find the best-fit line. The best-fit line is a line that has the least error which means the error between predicted values and actual values should be minimum.



But how the linear regression finds out which is the best fit line?

The goal of the linear regression algorithm is to get the **best values for B0 and B1** to find the best fit line. The best fit line is a line that has the least error which means the error between predicted values and actual values should be minimum.

Random Error(Residuals)

In regression, the difference between the observed value of the dependent variable(**y i**) and the predicted value(**predicted**) is called the residuals.

* **ε i =** **y predicted** –   **y i**
* **where y predicted  =   B 0 + B 1 X i**

**Multiple Linear Regression**

[**Multiple linear regression**](https://www.analyticsvidhya.com/blog/2021/05/multiple-linear-regression-using-python-and-scikit-learn/) is a technique to understand the relationship between a *single*dependent variable and *multiple*independent variables.

The formulation for multiple linear regression is also similar to simple linear regression with

the small change that instead of having one beta variable, you will now have betas for all the variables used. The formula is given as:

 Y = B0 + B1X1 + B2X2 + … + BpXp + **ε**

**Considerations of Multiple Linear Regression**

All four assumptions made for Simple Linear Regression still hold for Multiple Linear Regression along with a few new additional assumptions.

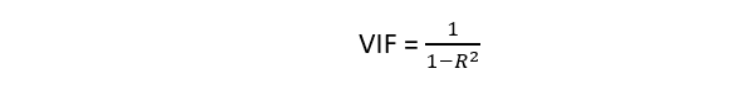
* **Overfitting**: When more and more variables are added to a model, the model may become far too complex and usually ends up memorizing all the data points in the training set. This phenomenon is known as the overfitting of a model. This usually leads to high training accuracy and very low test accuracy.
* **Multicollinearity**: It is the phenomenon where a model with several independent variables, may have some variables interrelated.
* **Feature Selection:** With more variables present, selecting the optimal set of predictors from the pool of given features (many of which might be redundant) becomes an important task for building a relevant and better model.

**Multicollinearity**

As [**multicollinearity**](https://www.analyticsvidhya.com/blog/2020/03/what-is-multicollinearity/) makes it difficult to find out which variable is contributing towards the prediction of the response variable, it leads one to conclude incorrectly, the effects of a variable on the target variable.  Though it does not affect the precision of the model predictions, it is essential to properly detect and deal with the multicollinearity present in the model, as random removal of any of these correlated variables from the model causes the coefficient values to swing wildly and even change signs.

Multicollinearity can be detected using the following methods.

* **Pairwise Correlations:** Checking the pairwise correlations between different pairs of independent variables can throw useful insights into detecting multicollinearity.
* **Variance Inflation Factor (VIF):** Pairwise correlations may not always be useful as it is possible that just one variable might not be able to completely explain some other variable but some of the variables combined could be ready to do this. Thus, to check these sorts of relations between variables, one can use VIF. VIF explains the relationship of one independent variable with all the other independent variables. VIF is given by,



*where i* refers to the*ith* variable which is being represented as a linear combination of the rest of the independent variables.

The common heuristic followed for the VIF values is if VIF > 10 then the value is high and it should be dropped. And if the VIF=5 then it may be valid but should be inspected first. If VIF < 5, then it is considered a good VIF value.

**Overfitting and Underfitting in Linear Regression**

There have always been situations where a model performs well on training data but not on the test data. While training models on a dataset, overfitting, and underfitting are the most common problems faced by people.

Before understanding overfitting and underfitting one must know about bias and variance.

**Bias**

Bias is a measure to determine how accurate a model’s predictions are likely to be on future unseen data. Complex models, assuming there is enough training data available, can make accurate model predictions. Whereas the models that are too naive, are very likely to perform badly concerning model predictions. Simply, Bias is errors made by training data.

Generally, linear algorithms have a high bias which makes them fast to learn and easier to understand but in general, are less flexible. Implying lower predictive performance on complex problems that fail to meet the expected outcomes.

**Variance**

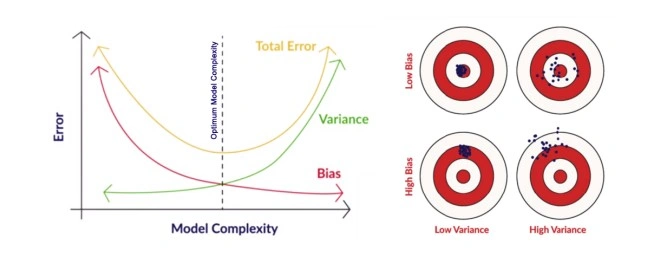
Variance is the sensitivity of the model towards training data, that is it quantifies how much the model will react when input data is changed.

Ideally, the model shouldn’t change too much from one training dataset to the next training data, which means that the algorithm is good at picking out the hidden underlying patterns between the inputs and the output variables.

Ideally, a model should have lower variance which means that the model doesn’t change drastically after changing the training data(it is generalizable). Having higher variance will make a model change drastically even on a small change in the training dataset.

Let’s understand what a bias-variance tradeoff is.

**Bias Variance Tradeoff**



In the pursuit of optimal performance, a supervised machine learning algorithm seeks to strike a balance between low bias and low variance for increased robustness.

In the realm of machine learning, there exists an inherent relationship between bias and variance, characterized by an inverse correlation.

* Increased bias leads to reduced variance.
* Conversely, heightened variance results in diminished bias.

Finding an equilibrium between bias and variance is crucial, and algorithms must navigate this trade-off for optimal outcomes.

In practice, calculating precise bias and variance error terms is challenging due to the unknown nature of the underlying target function.

Now, let’s delve into the nuances of [**overfitting and underfitting**](https://www.analyticsvidhya.com/blog/2020/02/underfitting-overfitting-best-fitting-machine-learning/).

**Overfitting**

When a model learns every pattern and noise in the data to such an extent that it affects the performance of the model on the unseen future dataset, it is referred to as ***overfitting***. The model fits the data so well that it interprets noise as patterns in the data.

When a model has low bias and higher variance it ends up memorizing the data and causing overfitting.  Overfitting causes the model to become specific rather than generic. This usually leads to high training accuracy and very low test accuracy.

Detecting overfitting is useful, but it doesn’t solve the actual problem. There are several ways to prevent overfitting, which are stated below:

* Cross-validation
* If the training data is too small to train add more relevant and clean data.
* If the training data is too large, do some feature selection and remove unnecessary features.
* Regularization

**Underfitting**

Underfitting is not often discussed as often as overfitting is discussed. When the model fails to learn from the training dataset and is also not able to generalize the test dataset, is referred to as ***underfitting***. This type of problem can be very easily detected by the performance metrics.

When a model has high bias and low variance it ends up not generalizing the data and causing underfitting. It is unable to find the hidden underlying patterns in the data. This usually leads to low training accuracy and very low test accuracy. The ways to prevent underfitting are stated below,

* Increase the model complexity
* Increase the number of features in the training data
* Remove noise from the data.

**Assumptions of Linear Regression:**

What is **The equal variance of residuals (Homoscedasticity, Heteroscedasticity)**

**Normal distribution of residuals**

**Independence of residuals:**

**Linearity of residuals**:

**What is the Best Fit Line?**

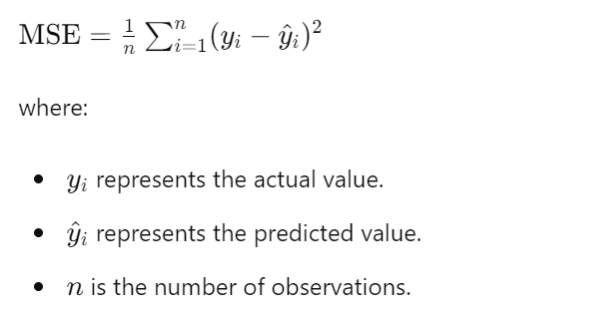
In simple terms, the best-fit line is a line that best fits the given scatter plot. Mathematically, you obtain the best-fit line by minimizing the Residual Sum of Squares (RSS).

**Cost Function for Linear Regression**

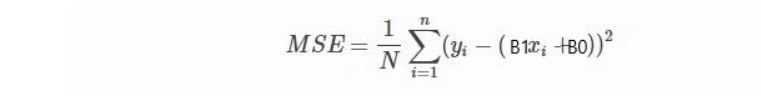
The [**cost function**](https://www.analyticsvidhya.com/blog/2021/03/data-science-101-introduction-to-cost-function/) helps to work out the optimal values for B 0  and B 1 , which provides the best-fit line for the data points.

In Linear Regression, generally [**Mean Squared Erro**](https://www.analyticsvidhya.com/blog/2024/07/mean-squared-error/)[**r (MSE)**](https://www.analyticsvidhya.com/blog/2024/07/mean-squared-error/) cost function is used, which is the average squared error that occurred between the **y predicted**and **y i**.

We calculate MSE using the simple linear equation y=mx+b:



Where y^ = B1xi + B0;

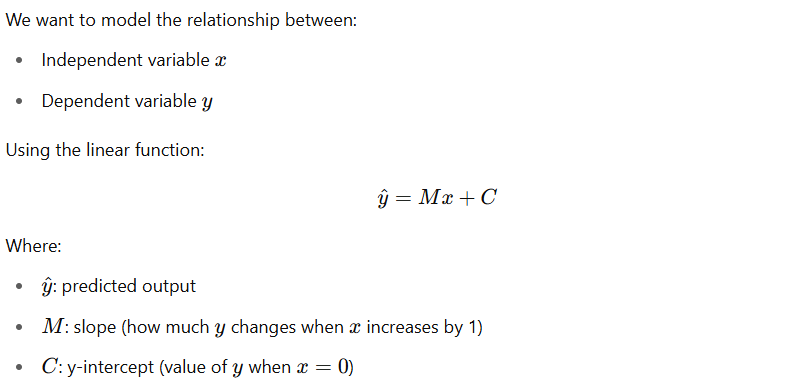


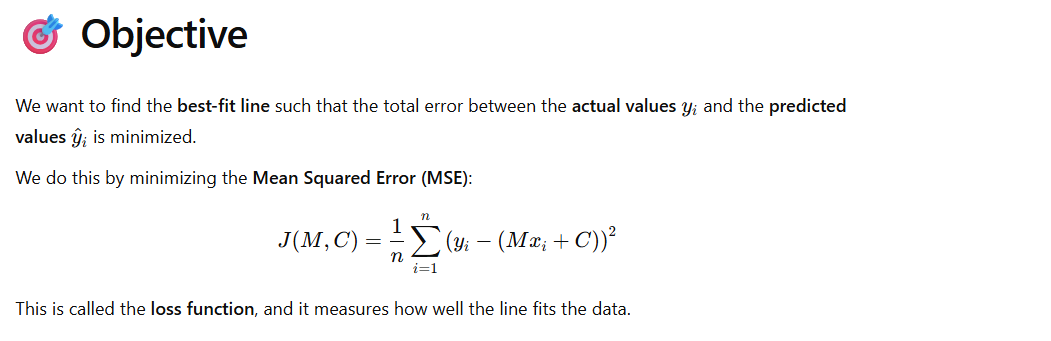
Using the MSE function, we’ll update the values of B 0  and B 1  such that the MSE value settles at the minima.  These parameters can be determined using the gradient descent method such that the value for the cost function is minimum.

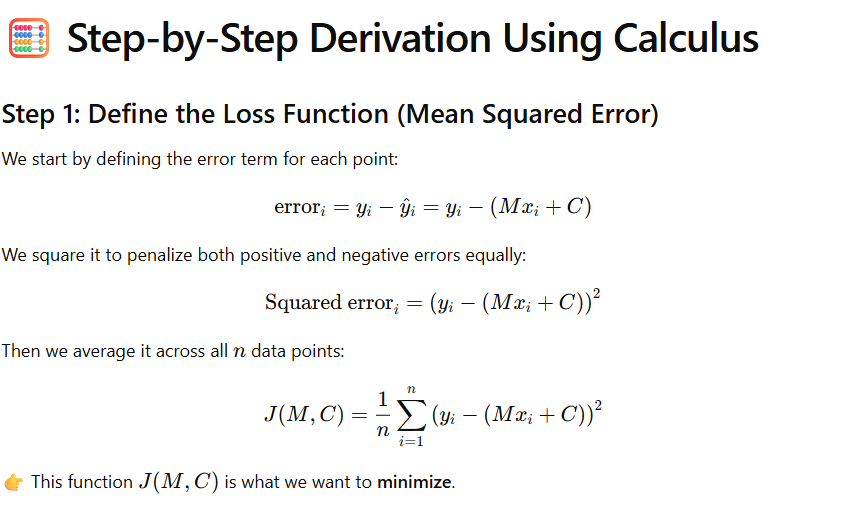
**Regularisation in linear regression**

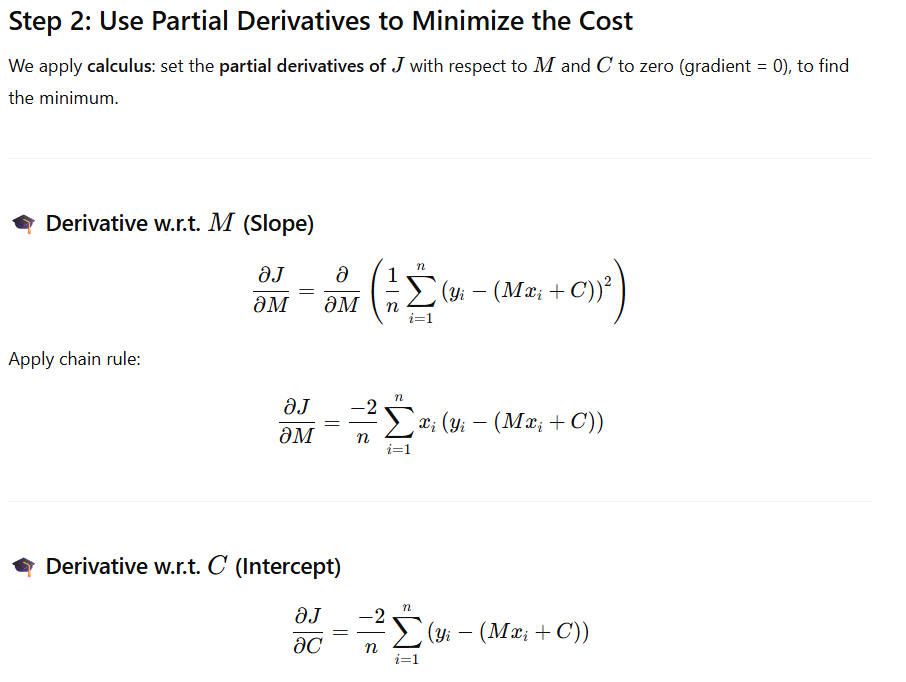
<https://www.analyticsvidhya.com/blog/2021/11/study-of-regularization-techniques-of-linear-model-and-its-roles/>

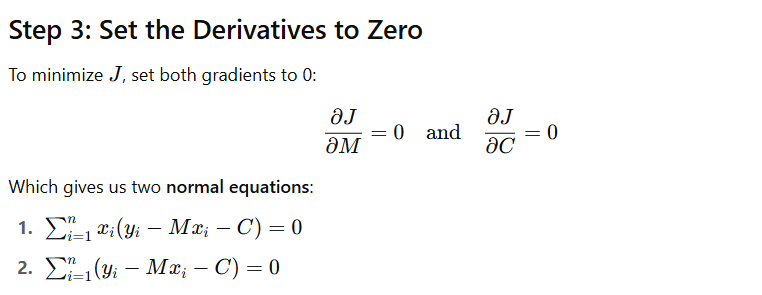
**Goal of Simple Linear Regression (slop intuition):**

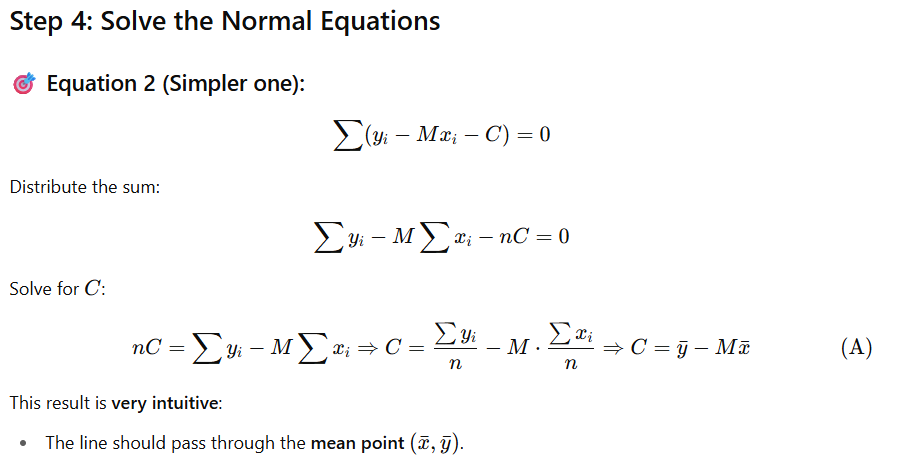


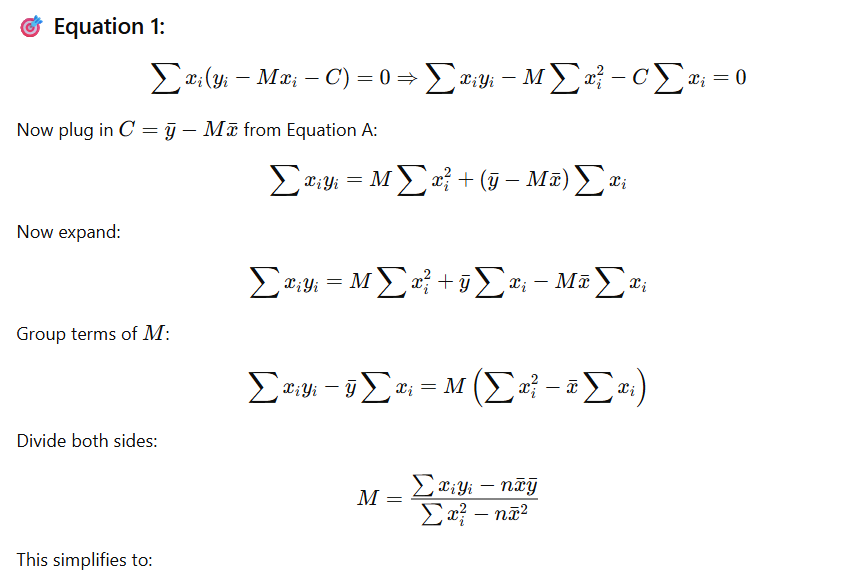




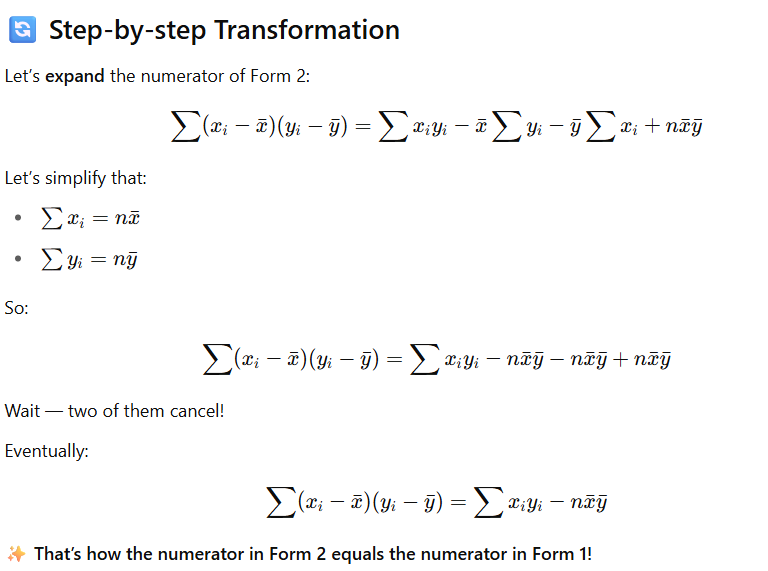


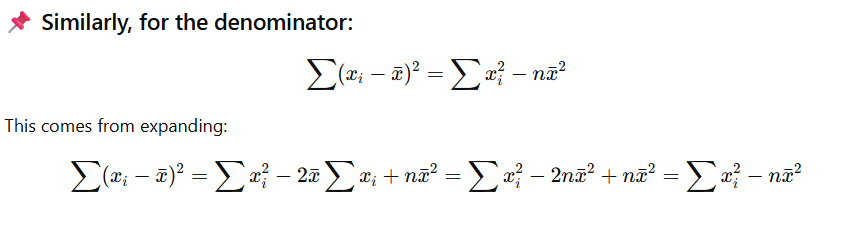


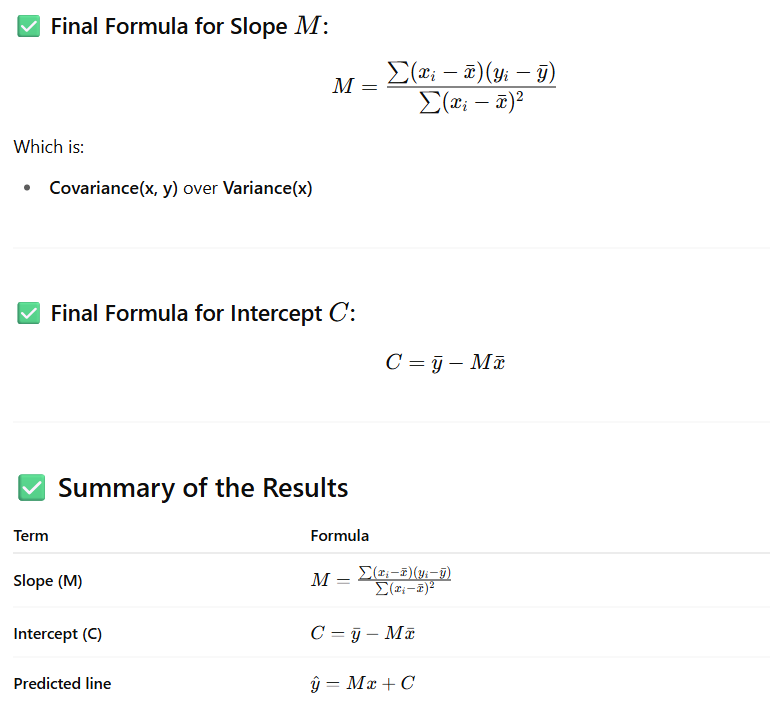


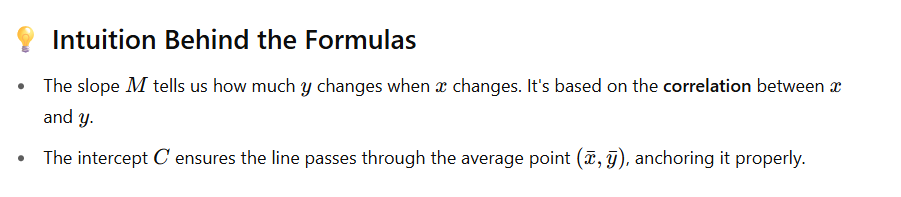


Numerator conversion –

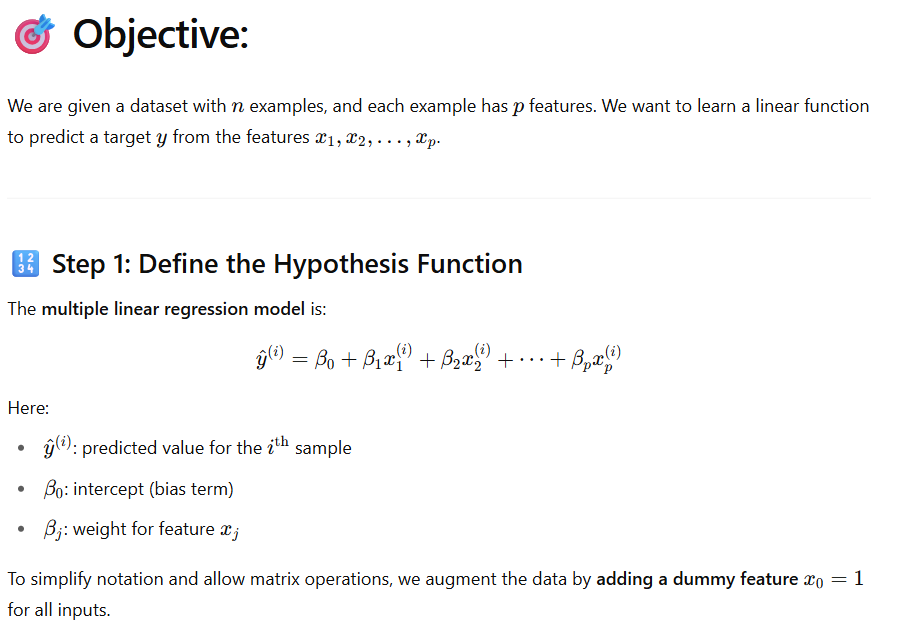


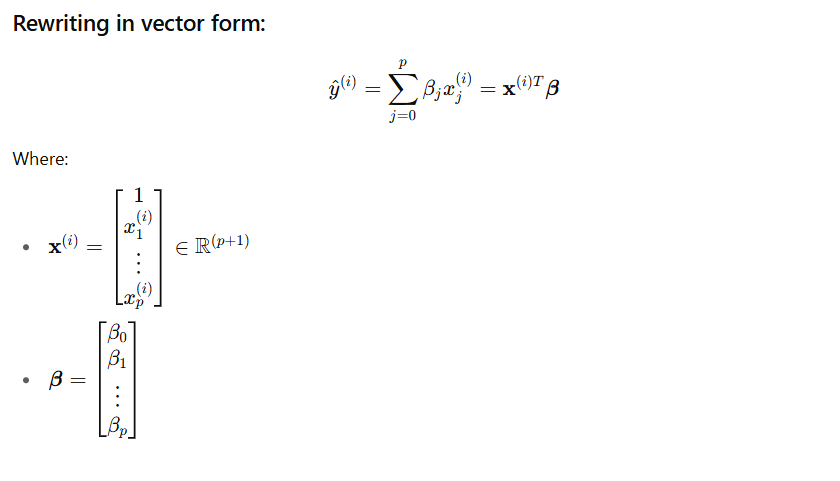


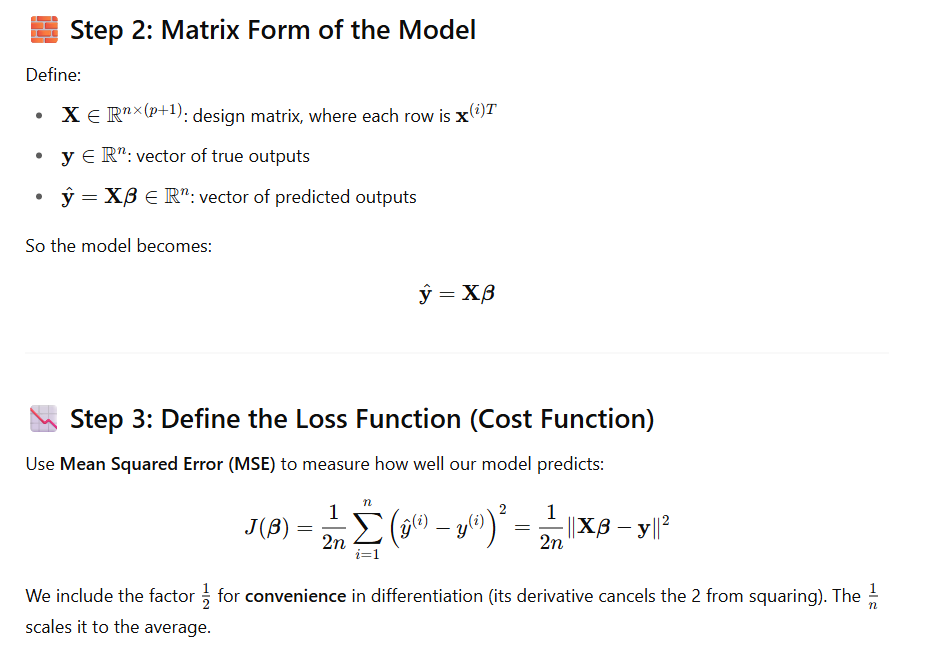




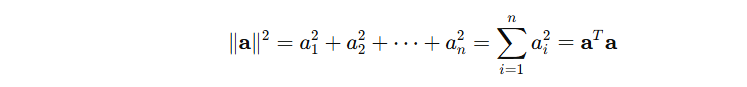
**Full derivation of multiple linear regression**

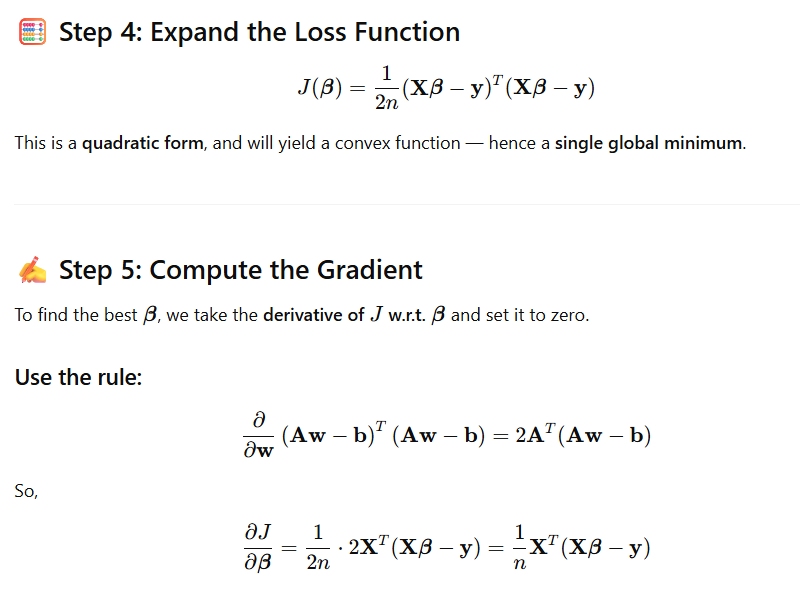


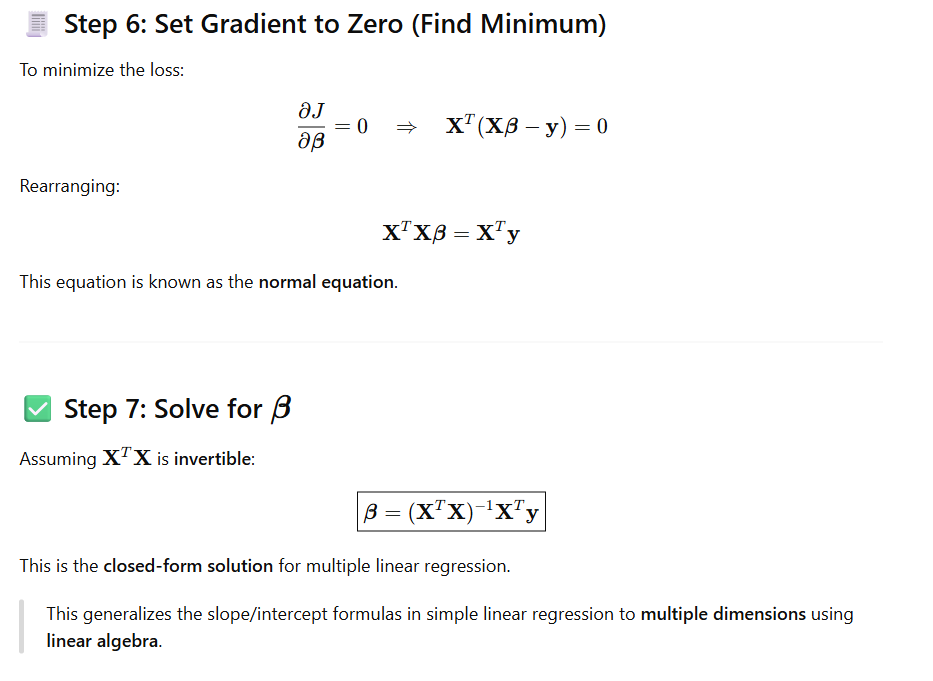


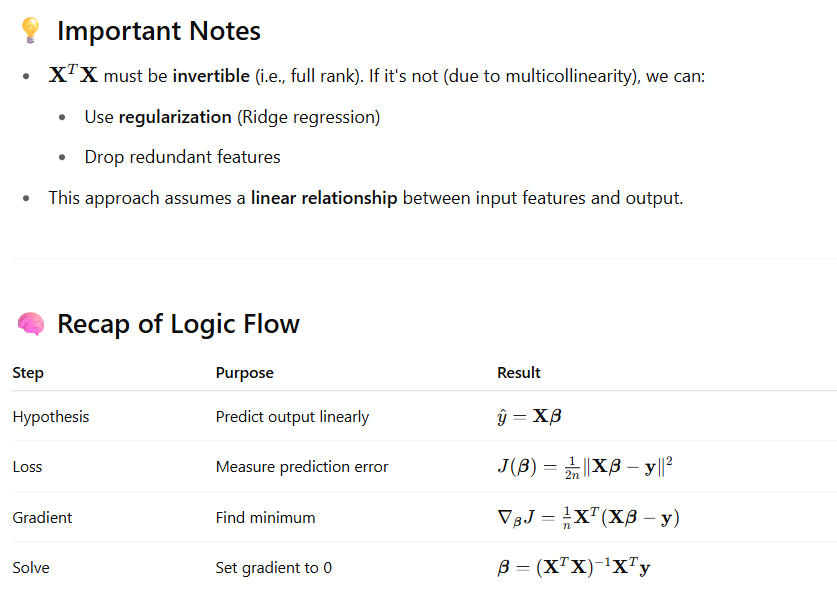


Use this below formula and convert above formula

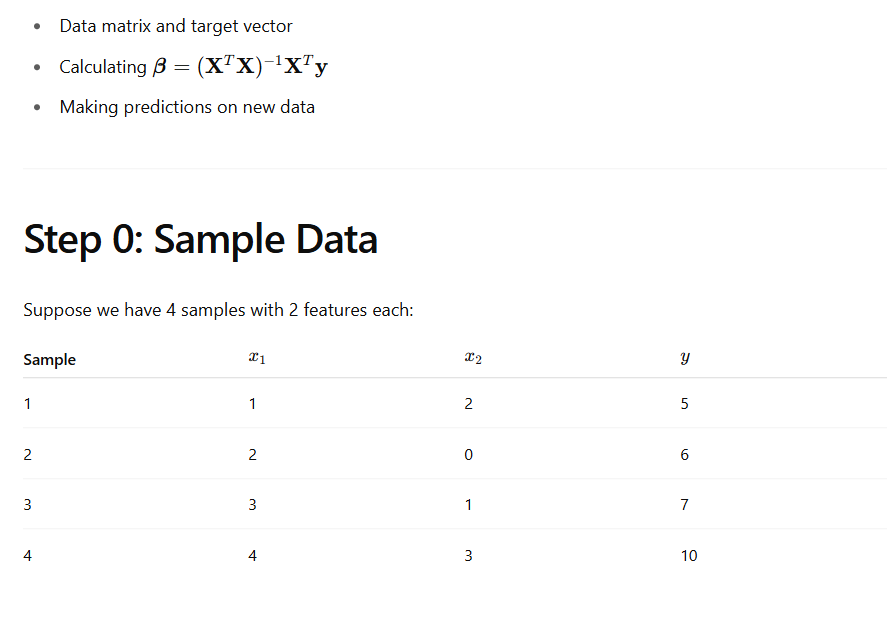


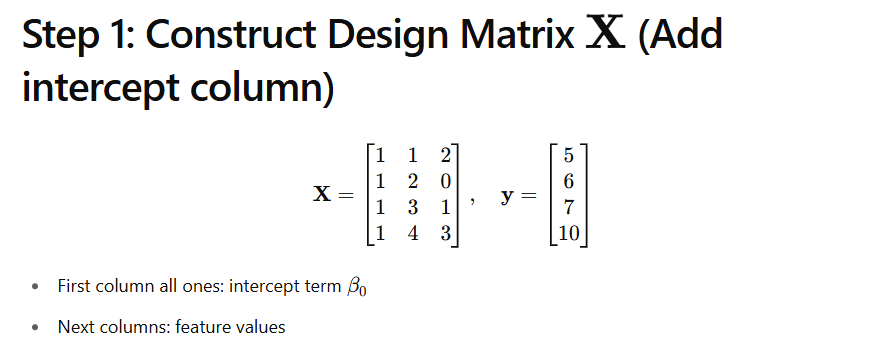


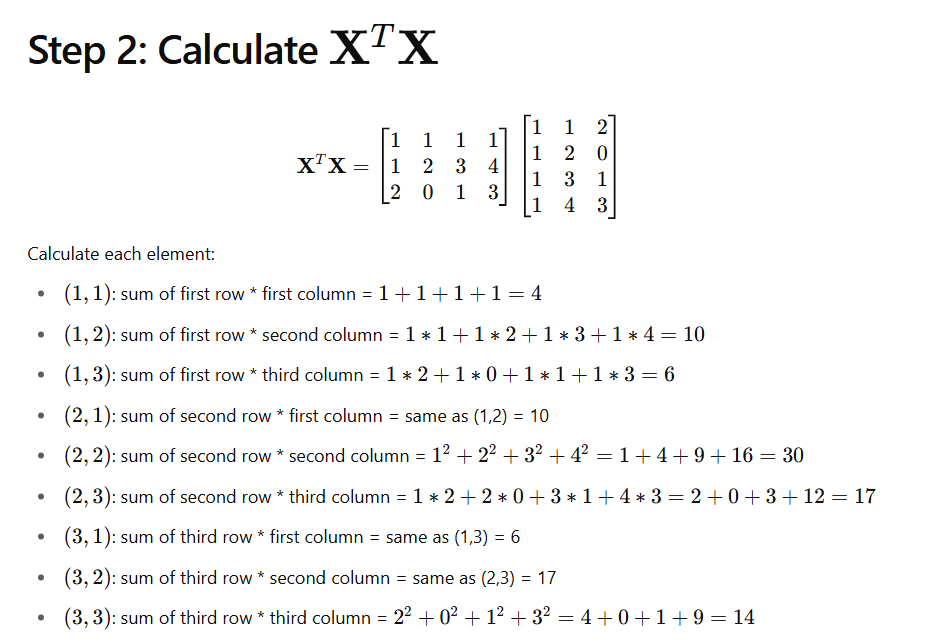


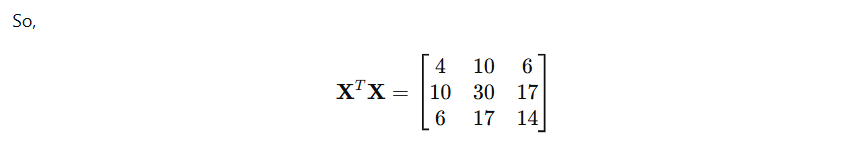


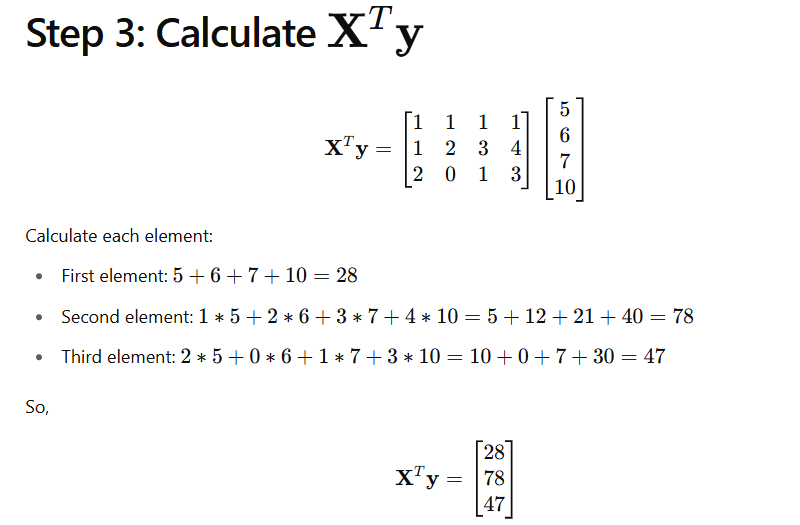
**Let’s do a full, step-by-step example of multiple linear regression with 2 features and 1 target, showing:**

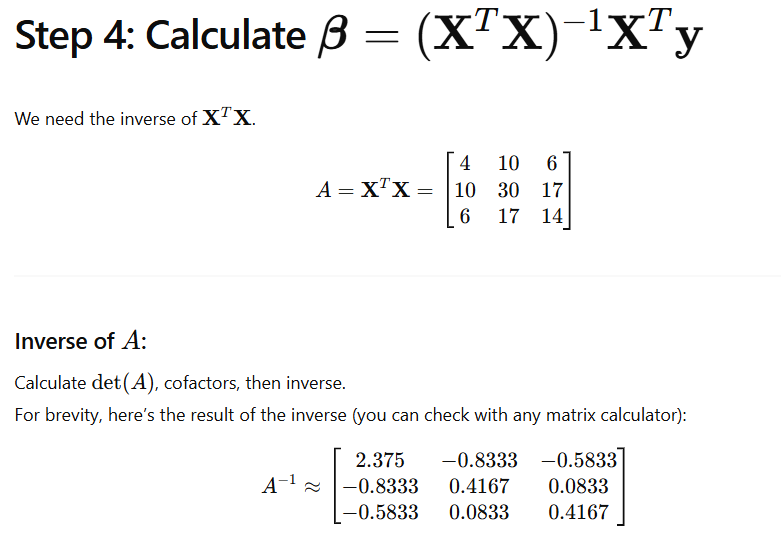


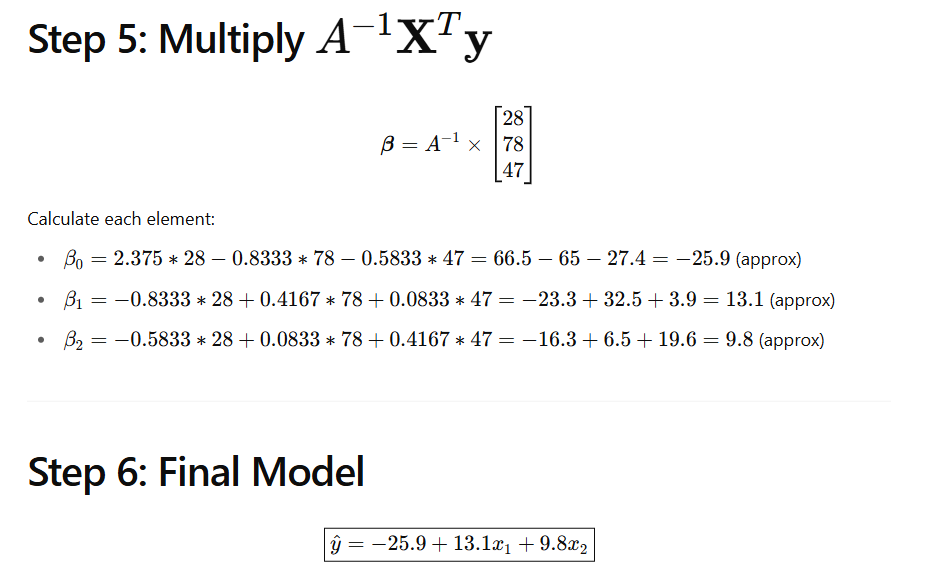


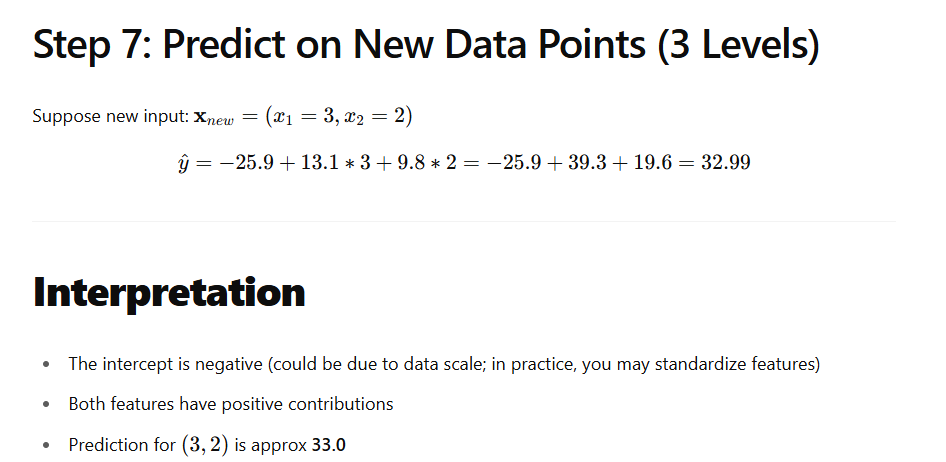




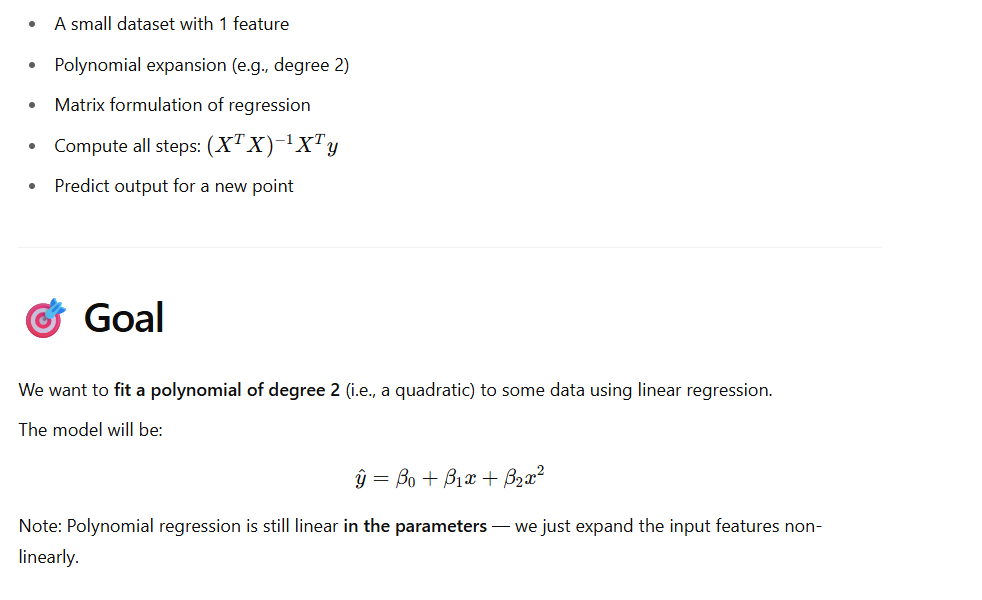


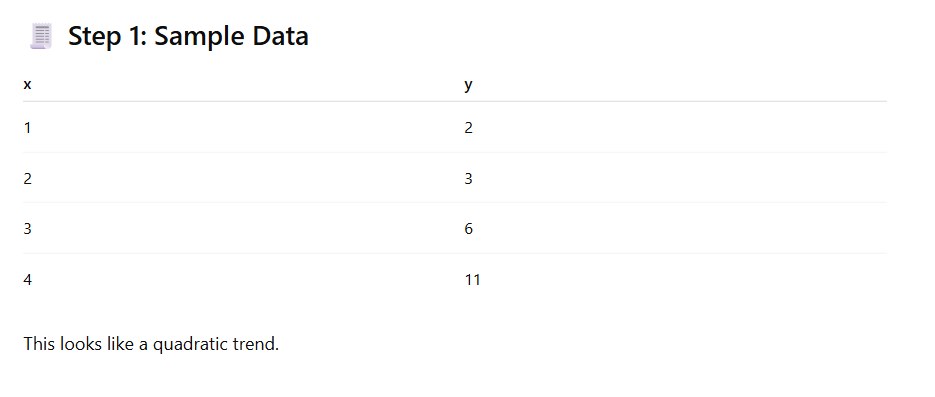


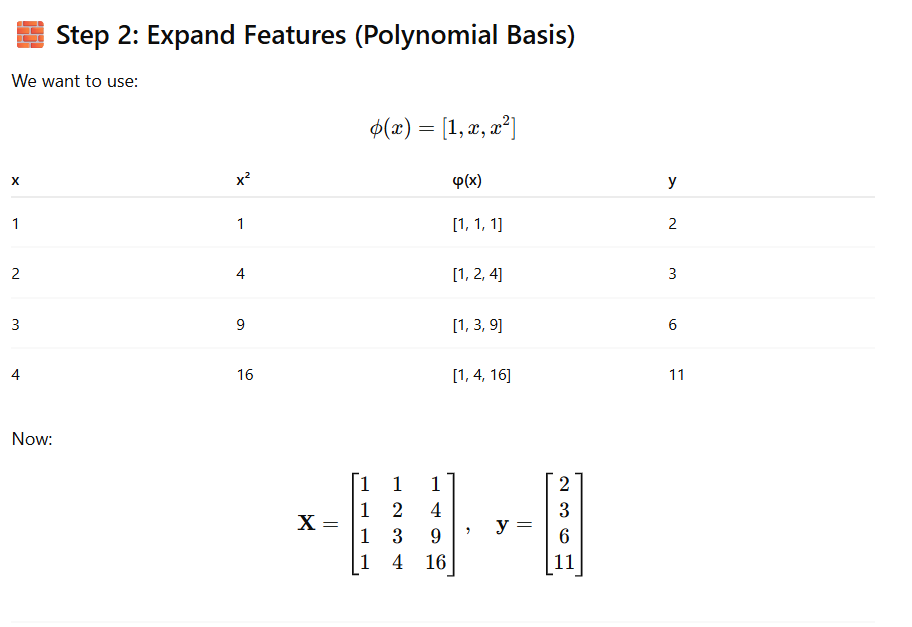


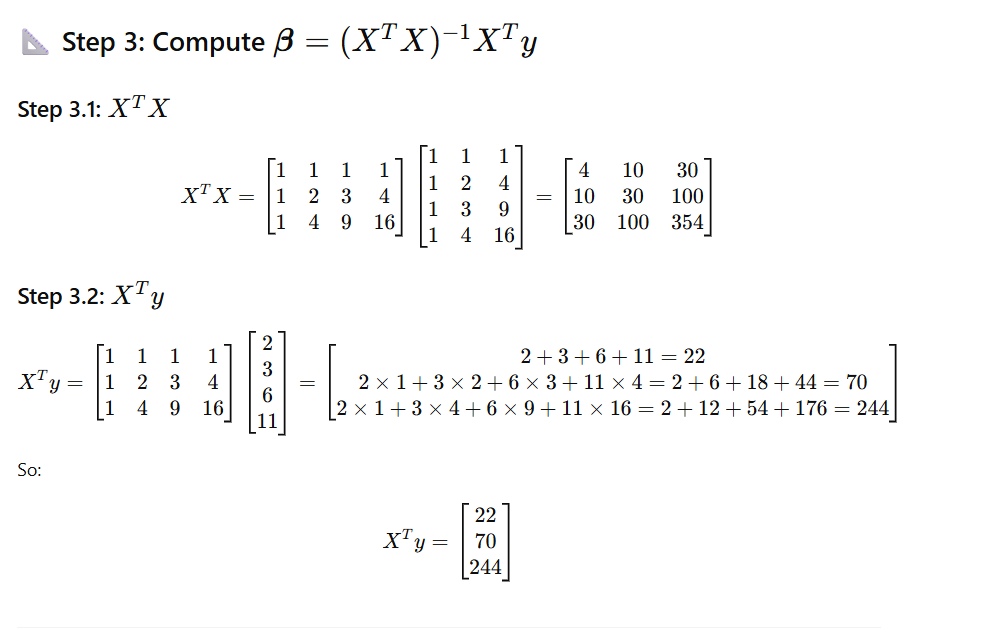


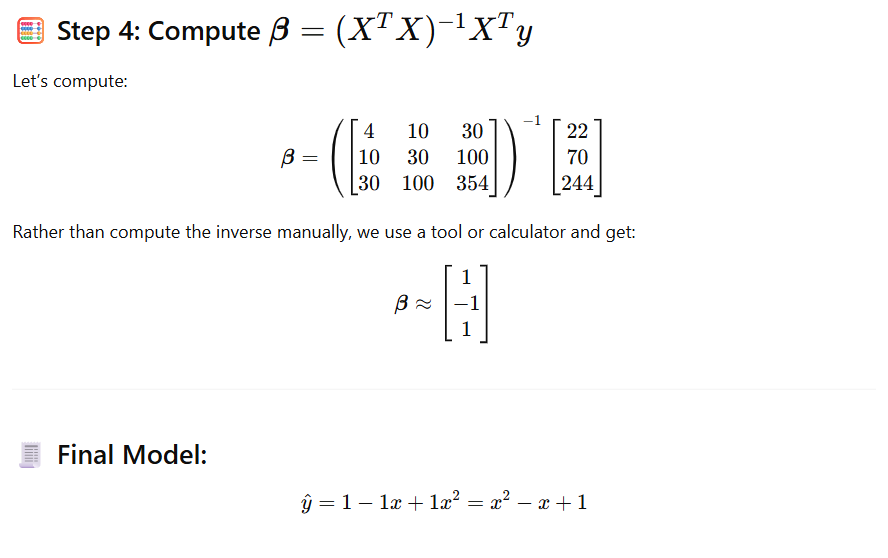
**Let’s now do a detailed worked-out numerical example for polynomial regression, including:**

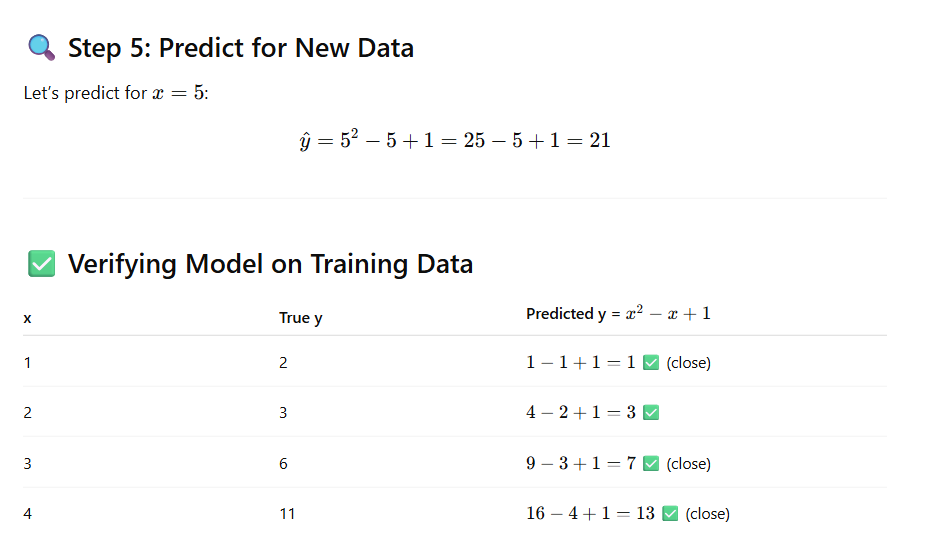










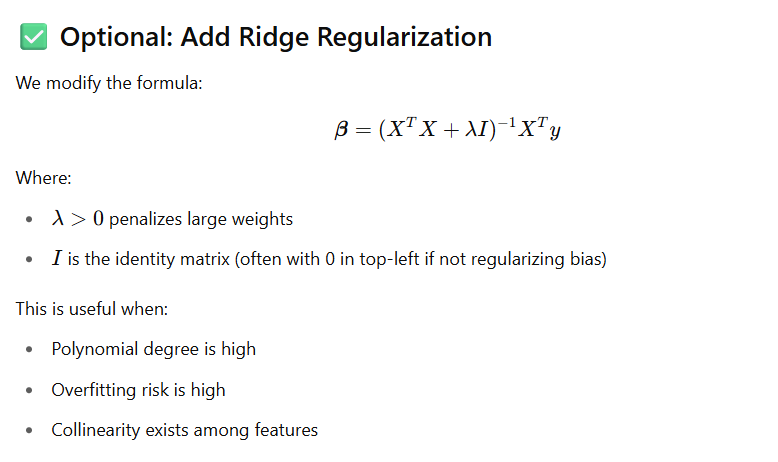


**📌 Summary**

We:

* Expanded the input features using a degree-2 polynomial
* Used standard linear regression steps on the transformed data
* Derived the coefficients
* Predicted on unseen input

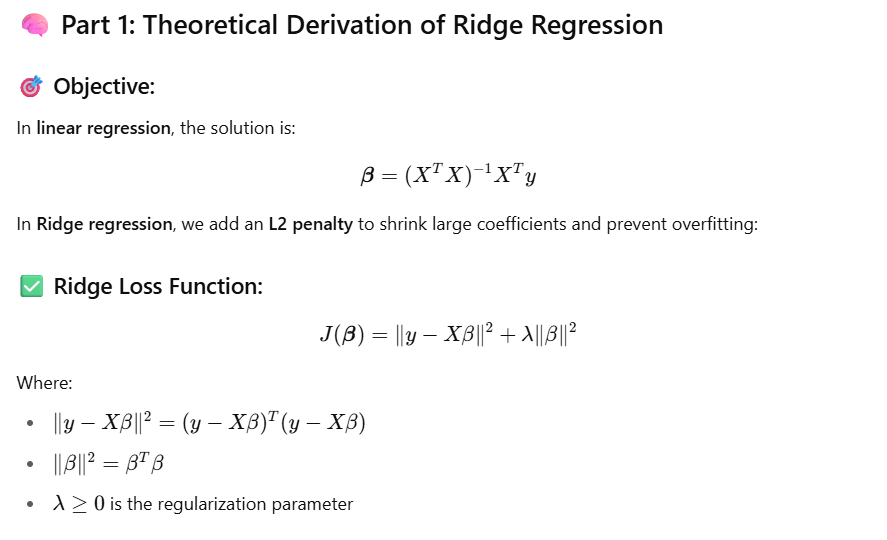
Polynomial regression is **linear regression on a non-linear transformation** of features.

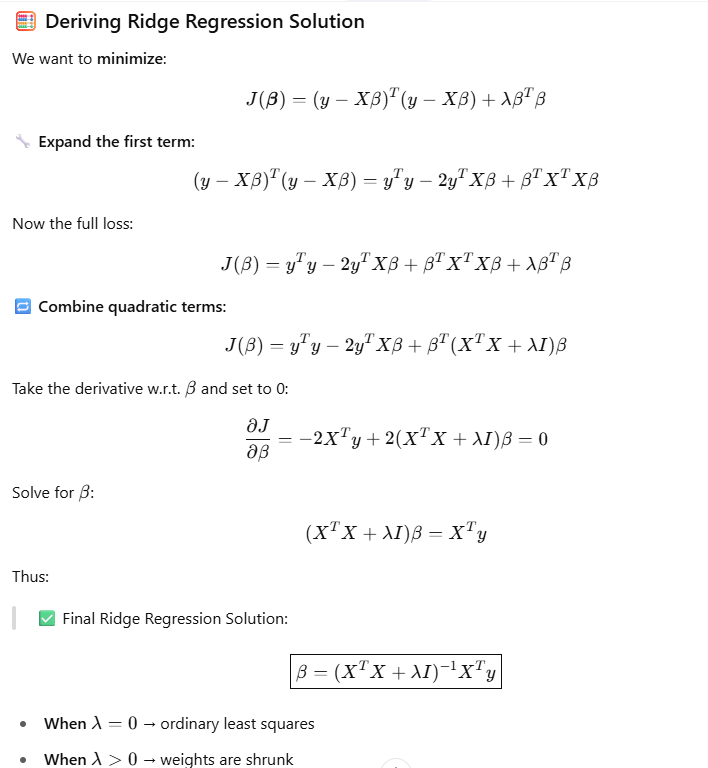


Try a range of λ values (e.g., 0.001, 0.01, 0.1, 1, 10, 100)

**Balance: Bias vs Variance**

* **Too small λ** → overfitting (high variance)
* **Too large λ** → underfitting (high bias)





import numpy as np

# Feature matrix (with bias term)

X = np.array([

[1, 1],

[1, 2],

[1, 3],

[1, 4]

]) # Shape: (4, 2) -> 1 bias + 1 feature

# Target values

y = np.array([2, 3, 6, 11]).reshape(-1, 1) # Shape: (4, 1)

# Regularization parameter

lmbda = 1.0

# Identity matrix (don’t regularize bias term)

I = np.eye(X.shape[1])

I[0, 0] = 0 # Don’t penalize bias term

# Ridge Regression formula

XtX = X.T @ X

ridge\_term = XtX + lmbda \* I

Xty = X.T @ y

beta = np.linalg.inv(ridge\_term) @ Xty

print("Ridge coefficients:", beta.ravel())

what all hyperparameters used in linear regression

**ChatGPT said:**

In **Linear Regression**, especially as implemented in libraries like **scikit-learn**, there are a few key **hyperparameters** — although basic linear regression has none in the mathematical sense, various **variants** like **Ridge**, **Lasso**, and **ElasticNet** do include hyperparameters.

Here's a breakdown by model type:

**✅ 1. Ordinary Linear Regression (OLS)**

In **pure OLS** (e.g., LinearRegression() in scikit-learn), there are **no core hyperparameters** for model complexity control, but a few **optional parameters** exist:

| **Hyperparameter** | **Purpose** |
| --- | --- |
| fit\_intercept | Whether to calculate the intercept bbb or assume it is zero |
| normalize *(deprecated)* | Previously normalized the input data (now handled in pipeline) |
| copy\_X | Whether to copy input X or overwrite |
| n\_jobs | Number of cores used for computation |
| positive | If True, forces coefficients to be positive |

**✅ 2. Ridge Regression (L2 Regularization)**

python

CopyEdit

from sklearn.linear\_model import Ridge

| **Hyperparameter** | **Purpose** |
| --- | --- |
| alpha | Controls the regularization strength (λ\lambdaλ); higher → more penalty |
| fit\_intercept | Whether to estimate the intercept |
| solver | Algorithm to use: "auto", "svd", "cholesky", "saga", etc. |
| normalize | Deprecated (use StandardScaler instead) |
| max\_iter | Maximum number of iterations (for solvers like saga) |
| tol | Tolerance for convergence |
| random\_state | For stochastic solvers |

**✅ 3. Lasso Regression (L1 Regularization)**

python

CopyEdit

from sklearn.linear\_model import Lasso

| **Hyperparameter** | **Purpose** |
| --- | --- |
| alpha | Regularization parameter; larger → more sparsity (zero coefficients) |
| max\_iter | Max number of iterations (important for convergence) |
| tol | Tolerance for stopping criterion |
| selection | "cyclic" or "random" (coordinate descent update rule) |

**✅ 4. ElasticNet (Combination of L1 and L2)**

python

CopyEdit

from sklearn.linear\_model import ElasticNet

| **Hyperparameter** | **Purpose** |
| --- | --- |
| alpha | Overall regularization strength |
| l1\_ratio | Mix between Lasso and Ridge: |

markdown

CopyEdit

- 1 = Lasso

- 0 = Ridge

- 0.5 = Equal mix |

| max\_iter | Max iterations for convergence |  
| tol | Stopping tolerance |

**🧪 Hyperparameter Tuning Tips**

* Use **cross-validation** to choose alpha, l1\_ratio, etc.
* Use RidgeCV, LassoCV, or ElasticNetCV for automated tuning.

python

CopyEdit

from sklearn.linear\_model import RidgeCV

model = RidgeCV(alphas=[0.01, 0.1, 1.0, 10.0], cv=5)

**📌 Summary Table**

| **Model** | **Key Hyperparameters** |
| --- | --- |
| OLS | fit\_intercept, copy\_X, positive |
| Ridge | alpha, solver, tol, max\_iter |
| Lasso | alpha, selection, tol, max\_iter |
| ElasticNet | alpha, l1\_ratio, tol, max\_iter |

Would you like a table showing how these hyperparameters affect **bias/variance tradeoff** or **example plots**?

**You said:**

how is this also works with different types of data like outliers, nonhomogeneous, polynomial, linear , non linear and others

**ChatGPT said:**

Great question! Linear regression — and its regularized variants — has different behavior when dealing with **varied types of data**, such as **outliers**, **non-linearity**, **non-homogeneous variance**, and more.

Let’s explore how linear regression behaves in each scenario and what you can do to handle it effectively.

**📘 How Linear Regression Handles Different Data Characteristics**

| **Scenario Type** | **Behavior of Linear Regression** | **Problem** | **Recommended Solution** |
| --- | --- | --- | --- |

**1. 🔺 Outliers in Data**

|  |  |  |  |
| --- | --- | --- | --- |
| Linear regression minimizes **squared error** | Outliers cause large errors → model shifts to reduce them | High sensitivity to outliers | Use **Robust Regression**, **Huber loss**, or **RANSAC** |

💡 Tip: Squared errors give more weight to large deviations → outliers distort the fit.

python

CopyEdit

from sklearn.linear\_model import HuberRegressor

**2. 📈 Non-Linear Relationships**

|  |  |  |  |
| --- | --- | --- | --- |
| Linear model can't fit curves | Model underfits and has high bias | Poor predictive performance | Use **Polynomial Regression** or **add interaction terms** |

💡 Transforming features using powers, logs, etc. can linearize a nonlinear relationship.

python

CopyEdit

from sklearn.preprocessing import PolynomialFeatures

**3. ⚖️ Non-Homogeneous Variance (Heteroscedasticity)**

|  |  |  |  |
| --- | --- | --- | --- |
| Assumes constant variance (homoscedasticity) | If variance changes with input, model's assumptions break | Leads to inefficient and biased inference | Use **Weighted Least Squares**, or apply **log-transform** |

💡 Check with a residual vs. fitted values plot — fan-shaped residuals indicate heteroscedasticity.

**4. 📉 Collinear or Highly Correlated Features**

|  |  |  |  |
| --- | --- | --- | --- |
| Makes XTXX^T XXTX nearly singular (non-invertible) | Coefficients become unstable (high variance) | Model is unreliable | Use **Ridge Regression** or **PCA** to reduce multicollinearity |

💡 Lasso can eliminate redundant features; Ridge shrinks them without zeroing.

**5. 🧱 Non-Normal Distribution of Errors**

|  |  |  |  |
| --- | --- | --- | --- |
| Assumes normally distributed residuals for inference | Impacts **confidence intervals and p-values** | Predictions still work, but uncertainty is less valid | Apply **bootstrapping** or use **non-parametric models** |

**6. 🔀 Categorical Variables**

|  |  |  |  |
| --- | --- | --- | --- |
| Linear regression works only with numerical data | Can't handle strings or categories directly | Will crash or treat them as numbers incorrectly | Use **One-Hot Encoding** or **Ordinal Encoding** before fitting |

python

CopyEdit

from sklearn.preprocessing import OneHotEncoder

**7. 🔁 Non-Independent Observations**

|  |  |  |  |
| --- | --- | --- | --- |
| Assumes samples are independent | Time series or panel data violates this | Biased coefficients | Use **time series models** like ARIMA, or **Generalized Least Squares (GLS)** |

**8. 🌈 Polynomial Features**

|  |  |  |  |
| --- | --- | --- | --- |
| Extend linear regression by transforming features to powers | Captures curves and trends | Higher degrees → risk of overfitting | Use **regularization (Ridge/Lasso)** to control complexity |

**🧠 Summary Table: Linear Regression vs. Data Characteristics**

| **Data Characteristic** | **Linear Regression Behavior** | **Recommended Fix** |
| --- | --- | --- |
| Outliers | Highly sensitive | Huber, RANSAC, log-transform |
| Non-linear relationship | Underfits | Polynomial regression, feature engineering |
| Heteroscedasticity | Biased errors | Weighted regression, transform target |
| Multicollinearity | Unstable coefficients | Ridge, Lasso, PCA |
| Categorical data | Not supported directly | One-hot encoding |
| Non-normal errors | Misleading inference | Bootstrap or robust regression |
| Time-series correlation | Assumption violated | GLS, ARIMA, time-aware models |

**Advantages of Linear Regression**

| **Advantage** | **Explanation** |
| --- | --- |
| 💡 **Simplicity & Interpretability** | The model is easy to understand and interpret. Coefficients directly show how features influence the output. |
| 🧮 **Fast Training & Prediction** | It’s computationally efficient even with large datasets — can be solved analytically using normal equations or optimized with gradient descent. |
| 📈 **Works Well with Linearly Separable Data** | If a true linear relationship exists, linear regression can model it accurately. |
| 📊 **Statistical Foundations** | Comes with strong statistical tools: confidence intervals, hypothesis testing, R², etc. |
| 🔍 **Feature Importance** | Coefficients can indicate which features are most impactful, making it useful for feature selection. |
| 🧪 **Basis for Many Models** | Linear regression is the foundation for more advanced models (logistic regression, GLMs, regularized models). |

**❌ Disadvantages of Linear Regression**

| **Disadvantage** | **Explanation** |
| --- | --- |
| 🔄 **Assumes Linearity** | Can't capture complex or non-linear relationships without feature engineering or transformation (e.g., polynomial terms). |
| 🔺 **Sensitive to Outliers** | Since it minimizes squared error, extreme values can disproportionately affect the model. |
| 🔀 **Multicollinearity Issues** | When features are highly correlated, coefficient estimates become unstable and unreliable. |
| 📈 **Assumes Homoscedasticity** | Assumes equal variance of errors — heteroscedasticity can lead to inefficiency and misleading results. |
| 🔄 **Assumes Independence of Errors** | Not suitable for time-series or sequential data without modifications. |
| ⛔ **Does Not Handle Missing Data Automatically** | Requires imputation or data cleaning before modeling. |
| 🎯 **Prone to Underfitting** | Tends to have high bias and may not capture complex patterns in the data. |

**📌 Summary Table**

| **✅ Pros** | **❌ Cons** |
| --- | --- |
| Easy to interpret | Assumes linear relationships only |
| Fast to train and predict | Sensitive to outliers |
| Good with small to medium data | Poor performance with complex data |
| Helps understand feature impact | Suffers from multicollinearity |
| Useful as a baseline model | Requires preprocessed and clean data |

**🧠 When to Use Linear Regression**

Use it when:

* The relationship between features and target is mostly linear
* You want a fast and interpretable model
* Data is not too noisy or full of outliers
* You need a strong baseline to compare with more complex models

Avoid it when:

* The relationship is complex or non-linear
* There are many categorical or highly correlated features
* You're working with high-dimensional, sparse, or noisy data without regularization

**✅ Category:**

* Supervised Learning
* Regression Algorithm (predicts continuous values)

**🔍 What is Linear Regression?**

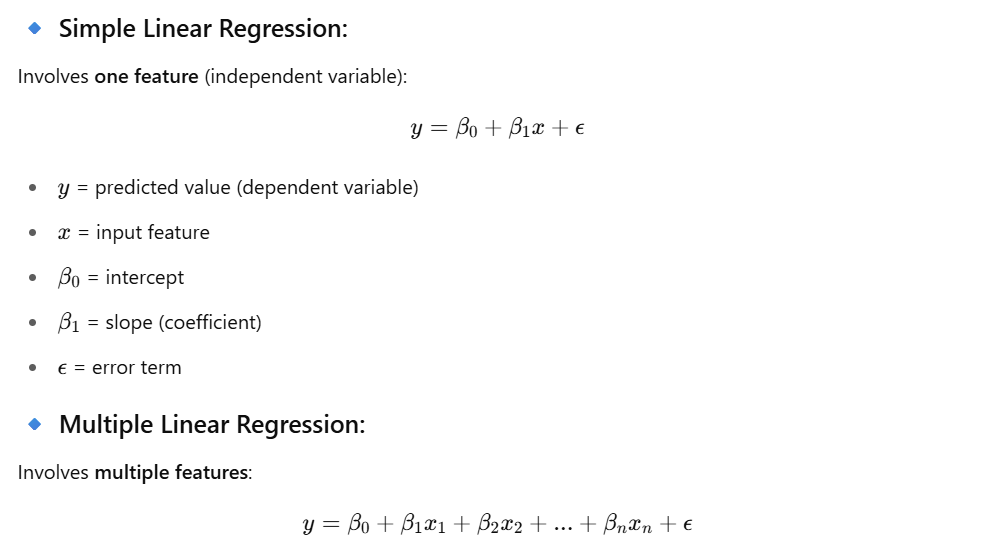
[**https://www.geeksforgeeks.org/linear-regression-python-implementation/**](https://www.geeksforgeeks.org/linear-regression-python-implementation/)

[**https://www.geeksforgeeks.org/ml-linear-regression/**](https://www.geeksforgeeks.org/ml-linear-regression/)

Linear Regression is a statistical method that models the **relationship between a dependent variable (target)** and one or more **independent variables (features)** using a **linear equation**.

**🔹 Simple Linear Regression:**

Bias variance trade off

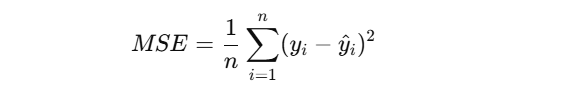


**🎯 Use Cases of Linear Regression**

| **Domain** | **Application** |
| --- | --- |
| Finance | Predicting stock prices, housing prices |
| Marketing | Sales forecasting, campaign ROI |
| Healthcare | Predicting disease progression (e.g., blood pressure) |
| HR/Analytics | Predicting employee salary or attrition risk |
| Sports | Player performance prediction |

**🛠️ How It Works (Training Process)**

1. **Input**: Historical data with features and labels
2. **Model**: Fit a line that minimizes the distance between the line and the actual data points
3. **Objective**: Minimize the **Mean Squared Error (MSE)**:



1. **Optimization**: Use **Gradient Descent** or **Normal Equation** to find optimal β\betaβ values

**📈 Assumptions of Linear Regression**

1. **Linearity**: The relationship between X and y is linear
2. **Independence**: Observations are independent
3. **Homoscedasticity**: Constant variance of errors
4. **Normality**: Errors are normally distributed
5. **No multicollinearity** (for multiple linear regression)

**Linear Regression Real Use Case**

**📘 Goal: Predict median house value based on average number of rooms per household.**

# 📌 Step 1: Import Libraries

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import fetch\_california\_housing

from sklearn.linear\_model import LinearRegression

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error, r2\_score

from sklearn.preprocessing import StandardScaler

from sklearn.pipeline import make\_pipeline

# 📌 Step 2: Load California Housing Dataset

data = fetch\_california\_housing()

X = data.data # Features

y = data.target # Target: Median house value

# Let's use only 1 feature for visualization: 'AveRooms' (index 3)

X = X[:, [3]] # Average number of rooms per household

# 📌 Step 3: Train/Test Split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# 📌 Step 4: Define and Train Linear Regression Model

model = make\_pipeline(

StandardScaler(), # Normalize input features

LinearRegression()

)

model.fit(X\_train, y\_train)

# 📌 Step 5: Predict on Test Data

y\_pred = model.predict(X\_test)

# 📌 Step 6: Evaluate Model

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print("✅ Mean Squared Error (MSE):", mse)

print("✅ R^2 Score:", r2)

# 📌 Step 7: Visualize Results

plt.figure(figsize=(10,6))

plt.scatter(X\_test, y\_test, color='blue', label='Actual', alpha=0.5)

plt.plot(X\_test, y\_pred, color='red', label='Predicted Line', linewidth=2)

plt.xlabel('Average Rooms per Household')

plt.ylabel('Median House Value')

plt.title('Linear Regression - California Housing')

plt.legend()

plt.grid(True)

plt.show()

📊 Model Evaluation

| Metric | Meaning |
| --- | --- |
| MSE | Lower MSE means better prediction performance |
| R² Score | Closer to 1 is better (1 = perfect fit) |

🔁 Comparison with SVR

| Feature | Linear Regression | SVR (RBF Kernel) |
| --- | --- | --- |
| Model Type | Global linear fit | Localized, flexible fit |
| Works well on | Linear data | Nonlinear, noisy data |
| Easy to interpret | ✅ Yes | ❌ No |
| Handles outliers | ❌ Poorly | ✅ Better (with ε margin) |
| Performance | Fast, simple | Slower, more powerful |

✅ Next Steps

**✅ Pros and Cons**

**✅ Pros:**

* Simple and easy to understand
* Interpretable coefficients
* Fast to train
* Works well when assumptions are met

**❌ Cons:**

* Assumes linearity
* Sensitive to outliers
* Can underperform on complex datasets
* Requires assumptions to be met

**🔎 When to Use Linear Regression?**

* When the relationship between features and target is roughly linear
* When you need a quick, interpretable model
* When the dataset is small to medium size and clean