Linear Regression in Data Science

Linear Regression is one of the most widely used and fundamental algorithms in ****Machine Learning**.** Despite its simplicity, it is a powerful statistical tool that forms the backbone of many predictive analytics models.

📌 What is Linear Regression?

Linear Regression is a **supervised learning algorithm** used to model the relationship between:

Dependent variable (target, Y) → the outcome we want to predict.

Independent variable(s) (features, X) → the inputs or predictors.

It assumes **a linear relationship between the variables**, expressed as:

Y = mX + c

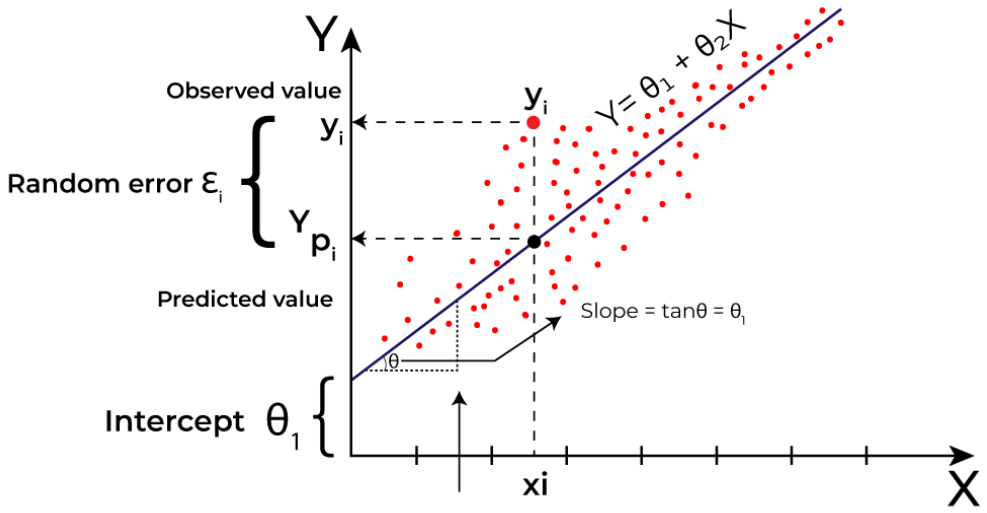
Where:

Y = Predicted value

X= Input feature(s)

m = Slope (effect of X on Y)

c= Intercept



📊 Example

Suppose we want to predict **student exam scores** based on the **hours studied.**

X = Hours studied

Y = Exam score

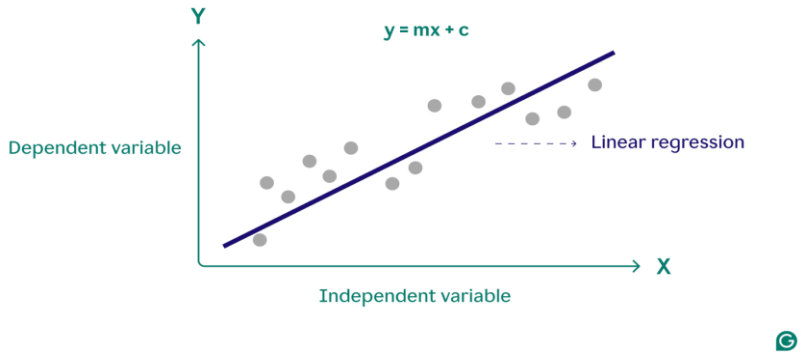
If the regression model finds the equation:

Score = 5 \times Hours + 30

👉 It means every extra hour of study increases the score by 5 points, starting with a base score of

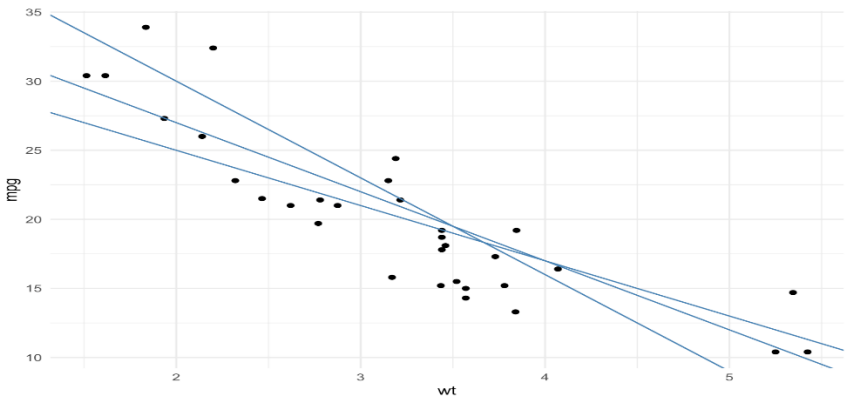
 Types of Linear Regression

1. Simple Linear Regression – One independent variable



Example: Predicting salary from years of experience.

2. Multiple Linear Regression – Multiple independent variables



Example: Predicting house prices from area, location, and number of bedrooms.

⚙️ **How Linear Regression Works**

1. Fits a **best-fit line** through the data points.

2. Uses the **Least Squares Method** to minimize the difference between actual and predicted values.

The error is measured as:

Error = (Actual - Predicted)

The model tries to minimize the **Mean Squared Error (MSE):**

$$MSE = \frac{1}{n}\sum(y_i - \hat{y}_i)^2$$

✅ Assumptions of Linear Regression

For the model to work well:

Relationship between X and Y is **linear**

Errors (residuals) are independent.

Variance of errors is constant (homoscedasticity).

Errors are normally distributed.

Features are not highly correlated (no multicollinearity).

🔍 **Evaluation Metrics**

To check performance, we use:

MSE (Mean Squared Error)

MAE (Mean Absolute Error)

RMSE (Root Mean Squared Error)

R^2 (Coefficient of Determination → Explains how much variance in Y is explained by X.

📓 Python Implementation

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
# Example dataset
data = {'Hours': [1,2,3,4,5], 'Score': [35, 40, 50, 60, 70]}
df = pd.DataFrame(data)

X = df[['Hours']]
y = df['Score']

# Train model
model = LinearRegression()
model.fit(X, y)

# Predictions
y_pred = model.predict(X)

print("Intercept:", model.intercept_)
print("Slope:", model.coef_)
print("R² Score:", r2_score(y, y_pred))
```

☑ Applications of Linear Regression

Business – Predicting sales from advertising spend.

Healthcare – Estimating patient recovery time based on lifestyle factors.

Finance– Predicting stock returns from market indicators.

Education – Forecasting student performance.

⚖️ **Advantages & Disadvantages**

✅ Advantages

* Easy to interpret

* Fast and computationally efficient

* Provides good baseline results

❌ Disadvantages

* Assumes linearity (not always true in real-world data)

* Sensitive to outliers

* Struggles with correlated features

Regularization techniques

Linear Regression may be simple, but it remains one of the most powerful starting points in Data Science. It's easy to understand, quick to implement, and forms the foundation for advanced techniques like **Ridge, Lasso, and Elastic Net**.

While **Linear Regression** is elegant and simple, it often struggles with real-world data. Here are the main challenges that motivated the introduction of these techniques:

Overfitting

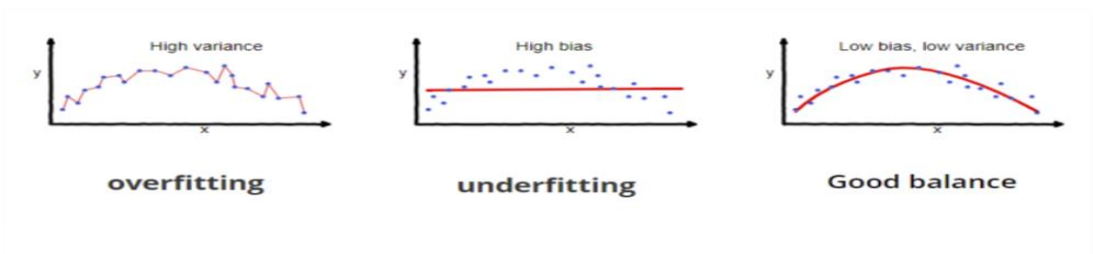
- The model learns **too much detail**, including noise from the training data.
- Performs **well on training data** but **poorly on unseen/test data**.
- Example: A student memorizes past exam questions but fails when new ones appear.

👉 **Happens when:** too many features, no regularization, very complex models.

Underfitting

- The model is **too simple** to capture the underlying trend in data.
- Performs **poorly on both training and test data**.
- Example: A student studies only definitions and cannot answer detailed questions.

👉 **Happens when:** too few features, oversimplified model, not enough training.



Multicollinearity

- Occurs when predictors are **highly correlated** with each other.
- In Linear Regression, this makes coefficient estimates **unstable** and sensitive to small data changes.

🔲 **Solution**: Ridge Regression stabilizes coefficients by shrinking them towards zero.

Too Many Irrelevant Features

- In modern datasets (finance, genomics, marketing), we may have **hundreds or thousands of predictors**, but only a few matter.
- Linear Regression includes all features, which adds noise and complexity.

🔲 **Solution**: Lasso Regression automatically performs **feature selection** by forcing some coefficients to zero.

✅ **Goal in Machine Learning**: Find the balance → a model that generalizes well without memorizing noise.

⚙️ **The Problem with Plain Linear Regression**

- Sensitive to **outliers**
- Struggles with **multicollinearity** (when features are highly correlated)
- May **overfit** when too many features are included

This is why **regularization techniques** were introduced.

🔲 Regularization Techniques

Regularization adds a **penalty term** to the cost function to prevent overfitting and handle multicollinearity.

The basic cost function of Linear Regression:

$$\text{MSE} = \frac{1}{n}\sum(y_i - \hat{y}_i)^2$$

1️⃣Ridge Regression (L2 Regularization)

- Adds penalty = **sum of squared coefficients**
- Shrinks coefficients but never makes them exactly zero.

$$\text{Cost} = MSE + \lambda \sum \beta_j^2$$

✅ **Advantages**

- Handles multicollinearity well
- Prevents overfitting
- Keeps all variables in the model

❌ **Limitations**

- Doesn't perform feature selection (all features stay)

2️⃣ Lasso Regression (L1 Regularization)

- Adds penalty = **sum of absolute values of coefficients**
- Shrinks some coefficients to **exactly zero**, performing **feature selection**.

$$\text{Cost} = MSE + \lambda \sum |\beta_j|$$

✅ **Advantages**

- Automatically selects important features
- Useful when you have many irrelevant predictors

❌ **Limitations**

- Can struggle when features are highly correlated
- Might randomly pick one variable and ignore others

3️⃣ Elastic Net (Combination of L1 & L2)

- Combines Ridge (L2) and Lasso (L1) penalties:

$$\text{Cost} = MSE + \lambda_1 \sum |\beta_j| + \lambda_2 \sum \beta_j^2$$

✅ **Advantages**

- Balances feature selection (Lasso) and coefficient shrinkage (Ridge)
- Works well with correlated features
- More flexible for real-world datasets

❌ **Limitations**

- Requires tuning of **two hyperparameters** (λ1,λ2\lambda_1, \lambda_2λ1,λ2)
- Computationally more expensive

📊 When to Use What?

- **Linear Regression** → When data is clean, no multicollinearity, and small number of predictors.
- **Ridge Regression** → When predictors are correlated, and we want to keep all features.
- **Lasso Regression** → When we want **automatic feature selection** (reduce irrelevant variables).
- **Elastic Net** → When features are correlated, and we want the **best of Ridge + Lasso**.

Python Example

```
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.metrics import mean_squared_error

X = [[1], [2], [3], [4], [5]]
y = [3, 5, 7, 9, 11]

# Models
lr = LinearRegression().fit(X, y)
ridge = Ridge(alpha=1.0).fit(X, y)
lasso = Lasso(alpha=0.1).fit(X, y)
elastic = ElasticNet(alpha=0.1, l1_ratio=0.5).fit(X, y)

print("Linear Regression:", lr.coef_)
print("Ridge:", ridge.coef_)
print("Lasso:", lasso.coef_)
print("Elastic Net:", elastic.coef_)
```

📈 Applications in Real Life

- **Business** → Predicting sales from advertising budgets
- **Finance** → Forecasting stock prices while handling correlated indicators
- **Healthcare** → Identifying most impactful health factors using Lasso
- **Marketing** → Handling large datasets with Elastic Net

Dataset Overview: Car Price Prediction

Your dataset contains car details and their prices:

Column	Meaning
Make	Car manufacturer (Honda, Ford, BMW, etc.)
Model	Specific model of the car
Year	Year the car was manufactured
Engine Size	Engine capacity in liters
Mileage	Distance driven (in km or miles, depending on dataset)
Fuel Type	Type of fuel (Petrol, Diesel, Electric, etc.)
Transmission	Transmission type (Manual / Automatic)
Price	Selling price of the car (target variable)

- **Independent variables (X):** All columns except Price.
- **Dependent variable (y):** Price (the value we want to predict).

What We Did With The Data

1. **Encoding Categorical Variables**:
   Columns like Make, Model, Fuel Type, and Transmission are text. Machine learning models can't work with text, so we converted them into numbers using **one-hot encoding**.
2. **Feature Scaling**:
   Features like Mileage and Engine Size have different ranges. Scaling them ensures that all features contribute equally to the model.
3. **Train-Test Split**:
   We separated the dataset into **training data** (80%) and **testing data** (20%) to evaluate the model fairly.
4. **Elastic Net Regression**:
   - This is a linear regression technique with **regularization**.
   - It combines **Lasso (L1)** and **Ridge (L2)** penalties.
   - Helps prevent **overfitting**, especially when some features are highly correlated.

What We Find

- **Prediction:**
  The model predicts the **car prices** based on features like make, model, year, engine size, mileage, fuel type, and transmission.

- **Performance Metrics:**
    - **MSE (Mean Squared Error):** Shows how far the predicted prices are from actual prices (lower is better).
    - **R^2 Score:** Indicates how much variation in car prices the model can explain (closer to 1 is better).
- **Insights:**
    - Cars with **newer years** or **bigger engine size** usually have **higher prices**.
    - **Electric cars** or specific makes/models may influence price significantly.
    - Elastic Net helps balance between simplicity and accuracy, preventing the model from relying too heavily on any one feature.
- **Visualization:**
  The scatter plot of **Actual vs Predicted Prices** shows how well the model predicts. Points close to the red diagonal line indicate **good predictions**.

Key Takeaway

By using **Elastic Net Regression**, we can predict car prices more accurately while avoiding overfitting. This is especially useful in real-world scenarios like:

- **Used car marketplaces** (like CarDekho, OLX, Cars24) for automated price prediction.
- **Dealerships** estimating resale prices.
- **Finance companies** predicting loan values against vehicles.

Integrating Linear Regression with Elastic Net

 # 📌 Car Price Prediction using Elastic Net Regression (User CSV Input)

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import ElasticNet
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Step 1: Read dataset
file_path = '/Car_Price_Prediction.csv'  # Your dataset path
df = pd.read_csv(file_path)
```

```
# Step 2: Encode categorical features
categorical_cols = ["Make", "Model", "Fuel Type", "Transmission"]
df = pd.get_dummies(df, columns=categorical_cols, drop_first=True)

# Step 3: Split features and target
X = df.drop("Price", axis=1)
y = df["Price"]

# Step 4: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 5: Feature Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Step 6: Elastic Net Regression
elastic_net = ElasticNet(alpha=0.5, l1_ratio=0.5, random_state=42)
elastic_net.fit(X_train_scaled, y_train)
y_pred = elastic_net.predict(X_test_scaled)

# Step 7: Evaluation
print("🔷 Elastic Net Regression Performance")
print("Mean Squared Error (MSE):", mean_squared_error(y_test, y_pred))
print("R² Score:", r2_score(y_test, y_pred))

# Step 8: Plot Actual vs Predicted Prices
plt.figure(figsize=(8,6))
plt.scatter(y_test, y_pred, color='blue', alpha=0.6)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.xlabel("Actual Prices")
```

```
plt.ylabel("Predicted Prices")
plt.title("Actual vs Predicted Car Prices (Elastic Net)")
plt.show()
```

◈ Elastic Net Regression Performance

Mean Squared Error (MSE): 5737909.2751348475

R^2 Score: 0.7903323238966551