REGRESSION

What is Regression?

- statistical and machine learning technique
- used to model and analyze the relationship between a dependent variable (the thing you want to predict) and one or more independent variables (the factors that might influence it).
- Regression tries to find a mathematical equation that best describes how changes in one or more inputs affect an output.
- Dependent variable (Y) → The outcome you want to predict.
- Independent variable(s) (X) → The input(s) that might influence Y.
- Regression estimates how much Y changes when X changes.

Purpose of Regression

- Prediction → Predicting future or unknown values.
- Relationship Understanding → Measuring how variables are related.
- Trend Analysis → Identifying patterns over time.

Types of regression

1. Linear Regression – Models a straight-line relationship.
   Simple Linear Regression → One independent variable.
   Multiple Linear Regression → Multiple independent variables.
2. Polynomial Regression – Models curves (non-linear relationships).
3. Logistic Regression – Used for classification (output is categorical).
4. Ridge / Lasso Regression – Regularized linear regression to avoid overfitting.

- **Regularization in ML**

Regularization is a technique in machine learning used to prevent overfitting by adding a penalty term to the model's cost function.

Why Regularization is Needed

- In regression, if you have too many features or highly correlated features, the model can fit the noise in training data.

- This makes it perform poorly on unseen data (overfitting).
- Regularization shrinks the coefficients toward zero, which reduces complexity.

In a simple linear regression, the cost function is:

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

With regularization, we **add a penalty term**:

1. **L1 Regularization (Lasso)**

$$\text{Loss} = \text{MSE} + \lambda \sum |w_j|$$

   - Adds the absolute value of coefficients.
   - Can make some coefficients exactly **zero** → feature selection.

2. **L2 Regularization (Ridge)**

$$\text{Loss} = \text{MSE} + \lambda \sum w_j^2$$

   - Adds the square of coefficients.
   - Shrinks all coefficients but rarely makes them exactly zero.

Here, **λ (lambda)** controls the strength of the penalty:

- **λ = 0** → No regularization (plain regression).
- **Large λ** → More shrinkage, simpler model.

Intuition

- Without regularization → The model might create wild curves to perfectly match training data.
- With regularization → The model's curve becomes smoother and more generalizable.

Linear regression

One of the simplest and most commonly used techniques in statistics and machine learning to model the relationship between two (or more) variables.

Linear regression tries to find the best-fitting straight line through a set of data points, so that you can:

- Predict a dependent variable (Y) based on the value(s) of independent variable(s) (X).

- Understand the strength and direction of the relationship between variables.

If you believe there's a *straight-line relationship* between X and Y, you can model it as:

$$Y=mX+c$$

Where:

- Y → Dependent variable (what you want to predict)
- X → Independent variable (input)
- m → Slope of the line (change in Y for one unit change in X)
- c → Intercept (Y's value when X = 0)

Types

1. Simple Linear Regression

- One independent variable (X).
- Example: Predicting a student's score from study hours.

2. Multiple Linear Regression

- Two or more independent variables (X_1, X_2, X_3, ...).
- Example: Predicting house price based on size, location rating, and number of rooms.

Simple linear regression

Regression Equation

In simple linear regression, we model the relationship between:

- Y → Dependent variable (what we want to predict)
- X → Independent variable (the predictor)

The equation is:

$$\hat{Y}=mX+c$$

Where:

- $\hat{Y}$ → Predicted value of Y
- m → Slope (how much Y changes when X increases by 1)
- c → Intercept (value of Y when X = 0)

Example:

If $\hat{Y}=2.5X+5$ and X = 4:

$\hat{Y}=2.5(4)+5=15$

Assumptions of Simple Linear Regression

For the model to work well:

1. Linearity:

   The relationship between the dependent variable (Y) and the independent variable (X) should be linear. This means a straight line can reasonably approximate the relationship.
2. Independence of Errors:

   The errors (residuals) should be independent of each other. In other words, the error for one observation should not be related to the error for another observation. This is often violated in time series data.
3. Homoscedasticity (Equal Variance):

   The variance of the errors should be constant across all levels of the independent variable. In simpler terms, the spread of the data points around the regression line should be consistent for all values of X.
4. Normality of Errors:

   The errors should be normally distributed. This means that the distribution of the residuals should resemble a bell curve.
5. No Endogeneity:

   The error term should not be correlated with the independent variable. This means the independent variable should not be influenced by the error term.

Limitations of Linear Regression.

1. Assumes a Linear Relationship

- **Limitation:** It only captures straight-line relationships between the independent variables (X) and the dependent variable (Y).
- **Impact:** If the relationship is non-linear, predictions will be poor unless you transform features.

---

2. Sensitive to Outliers

- **Limitation:** Even a single extreme value can drastically affect the slope of the regression line.
- **Impact:** Can lead to misleading coefficients and predictions.

---

3. Multicollinearity

- **Limitation:** If predictors are highly correlated with each other, it becomes difficult to estimate their individual effect.
- **Impact:** Coefficients become unstable and interpretability is reduced.

---

4. Homoscedasticity Assumption

- **Limitation:** Linear regression assumes the variance of errors is constant across all values of X (homoscedasticity).
- **Impact:** If variance changes (heteroscedasticity), standard errors are biased.

---

5. Normality of Errors

- **Limitation:** For inference (p-values, confidence intervals), residuals should be normally distributed.
- **Impact:** Violations affect hypothesis testing reliability.

---

6. Independence of Errors

- **Limitation:** Assumes errors are independent from one another.
- **Impact:** Violations (like in time-series data with autocorrelation) lead to underestimated standard errors.

---

7. Cannot Handle Complex Relationships Automatically

- **Limitation:** Cannot naturally model interactions or polynomial relationships unless explicitly added.
- **Impact:** Requires feature engineering for complex patterns.

---

8. Overfitting in High-Dimensional Data

- **Limitation:** When the number of features is close to or exceeds the number of observations, the model may overfit.
- **Impact:** Poor generalization to new data.

---

9. Extrapolation Risk

- **Limitation:** Predictions outside the range of training data are unreliable.
- **Impact:** Can give absurd results if X is far beyond the observed range.

Violating these assumptions can lead to inaccurate predictions and unreliable statistical inferences.

Scenario:

We want to predict the monthly electricity bill (Y) of households based on average monthly temperature (X) in their city.

We collect data from 40 households across different months.

---

1. Linearity

Assumption: The relationship between temperature and electricity bill is linear.

Example:

If temperature increases, air conditioning usage increases, so the bill goes up at a steady rate.

✓ Good: Bills increase roughly in a straight-line pattern as temperature rises.

✗ Bad: If bills rise sharply only after a certain temperature (e.g., after 30°C), the relationship is curved, breaking this assumption.

---

2. Independence

Assumption: Each observation is independent of the others.

Example:

One household's bill should not directly affect another's.

✓ Good: Households are separate, pay their own bills.

✗ Bad: If two households share an electricity meter, their bills are linked, violating independence.

---

3. Homoscedasticity

Assumption: The spread of prediction errors is consistent across all temperatures.

Example:

For months at 20°C or 35°C, the difference between actual and predicted bills should

have similar variation.

❌ Bad: If at higher temperatures, bills vary widely (some people use AC heavily, others not at all), while at lower temperatures variation is small, the assumption fails.

---

4. Normality of Residuals

Assumption: Prediction errors are normally distributed.

Example:

If predicted bill is ₹3000 but actual is ₹3200, the error is +₹200.

If we plot all errors, they should form a bell-shaped curve around zero.

✅ This matters for hypothesis testing and confidence intervals.

---

5. No Major Outliers

Assumption: No extreme points dominate the model.

Example:

If one house's bill is ₹15,000 while most are between ₹2,000–₹5,000, it could be due to unusual events (maybe they hosted a large event).

Such an outlier can pull the regression line off track.

Applications of Linear Regression – Student Activity

"Regression in the Real World"

Explore how regression is used in different industries to predict, analyze, and make decisions, then present your findings about at least 2 real-world regression applications in any one sector.

"Linear Regression in Action"

Explore how linear regression is applied in real-life scenarios, find at least 2 real-life problems in a sector of your choice where *linear regression* could be applied. Identify both Dependent variable (Y) and Independent variable(s)

How Simple Linear Regression works?

1. The algorithm tries to find **best-fit line** through the data.
2. It uses the **Ordinary Least Squares (OLS)** method to minimize the **sum of squared errors (SSE)**:

$$SSE=\sum(y_i-\hat{y}_i)^2$$

3. The slope and intercept are chosen so that SSE is minimized.

Linear regression's job is to find **the slope (m)** and **intercept (c)** that make SSE as small as possible.

The exact best values can be calculated with formulas:

$$m = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}$$

$$c = \bar{y} - m\bar{x}$$

Gradient Descent

- Gradient Descent is a first-order iterative optimization algorithm used to find the minimum of a function.
- In linear regression, it helps us find the best parameter values (like m and c) that minimize the cost function.
- The gradient is a vector of partial derivatives of the cost function with respect to each parameter.
- The cost function J(m,c) depends on two variables (m and c) — or more if we have multiple features.
- A partial derivative tells us how J changes with respect to one parameter at a time, while keeping the others constant.

For example:

- $\frac{\partial J}{\partial m}$ → How cost changes if we nudge m
- $\frac{\partial J}{\partial c}$ → How cost changes if we nudge c

- The **gradient** is a vector:

$$\nabla J(m, c) = \begin{bmatrix} \frac{\partial J}{\partial m} \\ \frac{\partial J}{\partial c} \end{bmatrix}$$

- This vector points in the direction of **steepest increase** in the cost function.
- To minimize cost, gradient descent moves **in the opposite direction**.

Loss Function

Definition: Measures the error **for a single training example**.

Purpose: Tells how far the model's prediction is from the actual value for one sample.

Notation: $L(y_i, \hat{y}_i)$

Example (Mean Squared Error - per sample):

$$L(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$$

Cost Function

Definition: Aggregates the loss over **the entire dataset** (or batch).

Purpose: Provides a single value representing the model's total error — this is what we minimize.

Notation: $J(\theta)$

Example (MSE - averaged):

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Evaluation Metrics

Evaluation Metrics are measures used to assess how well a machine learning model performs on given data.

For **regression problems**, common evaluation metrics include:

Mean Squared Error (MSE)

- Formula:

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

- Measures the average squared difference between actual values (y_i) and predicted values ($\hat{y}_i$).
- Penalizes larger errors more heavily.

Root Mean Squared Error (RMSE)

- Formula:

$$RMSE = \sqrt{MSE}$$

- Same as MSE but in the original unit of the target variable.
- Easier to interpret compared to MSE.

Mean Absolute Error (MAE)

- Formula:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

- Measures the average absolute difference between predictions and actual values.
- Less sensitive to large errors compared to MSE.

R^2 Score (Coefficient of Determination)

- Formula:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

- Represents the proportion of variance in the dependent variable explained by the model.
- Value ranges from **0 to 1** (higher is better), but can be negative if the model is worse than a simple mean predicti

The **R^2-score** (coefficient of determination) tells you **how well your regression model explains the variation in the dependent variable**.

Formula

$$R^2 = 1 - \frac{\mathrm{SS}_{\mathrm{res}}}{\mathrm{SS}_{\mathrm{tot}}}$$

Where:

- $\mathrm{SS}_{\mathrm{res}}$ = Sum of squared residuals (errors)
- $\mathrm{SS}_{\mathrm{tot}}$ = Total sum of squares (variation in actual data)

Interpretation

R^2 Value	**Meaning**
1.0	Perfect fit — the model explains **100%** of the variance.
0.9	Very strong — explains **90%** of variance; only 10% left unexplained.
0.7 – 0.9	Strong fit — good predictive power.
0.5 – 0.7	Moderate fit — model explains some variance but may miss important factors.
0.3 – 0.5	Weak fit — model doesn't explain much variance.
0.0	Model explains **0%** — no predictive power.
Negative	Model is worse than simply predicting the mean of the data.

Example – Electricity Bill

If our model for **Bill = 50 + 7 × Units Consumed** gives an **R^2 = 0.95**,
that means **95% of the variation** in electricity bills is explained just by the number of units consumed.
The remaining 5% might be due to other factors like seasonal charges, penalties, or taxes.

Mean Squared Error (MSE)

- **Lower is better** — 0 means perfect prediction.
- **Limitation**: Large errors are punished heavily because of the squaring.

Example:
If MSE = **400**, it means your prediction is off by **~₹20** on average, since:

$$\sqrt{\text{MSE}} = RMSE = \sqrt{400} = ₹20$$

Root Mean Squared Error (RMSE)

- **Lower is better** — easier to interpret than MSE because it's in the same unit as actual values.

Example:
RMSE = ₹20 means your bill predictions are typically off by around ₹20.

Mean Absolute Error (MAE)

- **Lower is better** — less sensitive to outliers compared to MSE/RMSE.

Example:
MAE = ₹18 means, on average, your prediction misses the actual bill by ₹18.

Multiple Linear Regression

Multiple Linear Regression is a statistical technique used to model the relationship between one **dependent variable** (target) and **two or more independent variables** (predictors).

Equation:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n + \epsilon$$

Where:

- y = dependent variable (what we want to predict)
- $x_1, x_2, ..., x_n$ = independent variables
- β_0 = intercept (value of y when all x are 0)
- $\beta_1, \beta_2, ...$ = coefficients (impact of each predictor on y)
- ϵ = error term

Example: Predicting house price using size, number of rooms, and distance to city.

Assumptions of MLR

To ensure validity of the model, MLR assumes:

1. **Linearity**
   The relationship between dependent and independent variables is linear.
2. **Independence of Errors**
   Errors (residuals) are independent of each other (no autocorrelation).
3. **Homoscedasticity**
   The variance of residuals is constant across all levels of the independent variables.
4. **Normality of Errors**
   Residuals are normally distributed (important for inference, less for prediction).
5. **No Multicollinearity**
   Independent variables should not be highly correlated with each other.

Normal Equation

Instead of using iterative optimization like gradient descent, we can directly compute coefficients using the **Normal Equation**:

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

Where:

- X = matrix of input features (with column of 1's for intercept)
- y = vector of outputs
- β = vector of coefficients

⚠️ Works well for **small datasets** — for very large datasets, gradient descent is preferred.

Applications of MLR

Multiple Linear Regression is widely used for:

- **Economics:** Predicting GDP based on investment, consumption, exports, etc.
- **Agriculture:** Predicting crop yield from rainfall, fertilizer use, and soil quality.
- **Healthcare:** Estimating blood pressure from age, weight, and exercise levels.
- **Marketing:** Predicting sales from advertising spend across multiple channels.

- **Real Estate:** Predicting house price from size, location, and amenities.

Ridge and **Lasso Regression**

1. Why We Use Them

In plain regression:

- Too many features → overfitting
- Multicollinearity → unstable coefficients
- Model may perform well on training data but poorly on new data

Ridge and **Lasso** add a *penalty term* to the regression cost function to control the size of coefficients → making the model more generalizable.

Ridge Regression (L2 Regularization)

- Penalty term: $: \lambda \sum_{j=1}^{p} \beta_j^2$
- Shrinks coefficients toward zero **but never exactly zero**
- **When to use:**

  ✓ Many small/medium-effect features

  ✓ Multicollinearity (features are correlated)

  ✓ You don't want to completely drop any feature — just reduce their influence
- **Pros:** Keeps all variables in the model but with smaller weights
- **Cons:** Doesn't perform feature selection — model might still be complex

Lasso Regression (L1 Regularization)

- Penalty term: $\lambda \sum_{j=1}^{p} |\beta_j|$
- Shrinks some coefficients to **exactly zero** → does **automatic feature selection**
- **When to use:**

  ✓ You suspect only a few features matter (sparse solution)

  ✓ You want to reduce model complexity and remove irrelevant features
- **Pros:** Simple, interpretable model; automatic variable selection
- **Cons:** Can be unstable if predictors are highly correlated (might drop one arbitrarily)

Example — Electricity Bill Prediction

Say you have:

- Units consumed ✅ (important)
- Month ✅ (important)
- Day of week ❌ (irrelevant)
- Weather ✅ (important)
- Type of wire ❌ (irrelevant)
- **Ridge** → keeps all variables, just reduces effect of irrelevant ones.
- **Lasso** → drops "Day of week" and "Type of wire" entirely if they're useless.

For building a basic **linear regression model** and evaluating it (like in the electricity bill example), you'll generally need:

```
# 1️⃣ Import Required Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
```

Load the data into dataframe.

Preview data

Summarize data

#Assess data quality and fix issues - handle anomalies, data transformation – if required.

Winsorization – cap outliers

```python
from scipy.stats.mstats import winsorize
def winsorize_data(df, limits=(0.01, 0.01)):
    """
    Apply Winsorization to all numeric columns in a DataFrame.
    Cap values below the 1st percentile to the 1st percentile value.
    Cap values above the 99th percentile to the 99th percentile value.
    """
    for col in df.select_dtypes(include=[np.number]):
        df.loc[:,col] = winsorize(df[col], limits=limits)
    return df
```

standardization

```python
from sklearn.preprocessing import StandardScaler

def standardize_data(df, columns=None):
     if columns is None:
        columns = df.select_dtypes(include=['number']).columns

    scaler = StandardScaler()
    df.loc[:,columns] = scaler.fit_transform(df[columns])
    return df
```

#check the correlation

```python
# Compute correlation matrix
co_mtx = df.corr(numeric_only=True)

# Print correlation matrix
print(co_mtx)

# Plot correlation heatmap
sns.heatmap(co_mtx, cmap="YlGnBu", annot=True)

# Display heatmap
plt.show()
```

Split Data into Features (X) and Target (y).

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

# 5️⃣ Create and Train the Model
model = LinearRegression()
model.fit(X_train, y_train)

# 6️⃣ Make Predictions
y_pred = model.predict(X_test)
```

```python
# 7️⃣ Regression Equation
print(f"Intercept (b0): {model.intercept_}")
print(f"Coefficient (b1): {model.coef_[0]}")
```

```python
# 8️⃣ Evaluation Metrics
mse = metrics.mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = metrics.r2_score(y_test, y_pred)

print("\nModel Evaluation:")
print(f"MSE  : {mse:.2f}")
print(f"RMSE : {rmse:.2f}")
print(f"R²   : {r2:.2f}")
```