# 📘 DAY 10: OPTIMIZATION, GRADIENT DESCENT, MSE, $R^2$ & MULTIPLE LINEAR REGRESSION

# 🎯 GOAL OF THE DAY

Understand how **optimization** works using **Gradient Descent**, how it's powered by **Calculus**, evaluated using **Mean Squared Error (MSE)** and **$R^2$ Score**, and extended to **Multiple Linear Regression** for more complex predictions.

## 🔍 WHAT IS GRADIENT DESCENT?

Gradient Descent is an **optimization algorithm** used to find the **minimum of a function** — in ML, it minimizes the **cost function**.

## ✅ WHY USE IT?

In linear regression, we want to minimize the cost:

```
Cost (MSE) = (1/n) * Σ (y - ŷ)^2
```

But we don't solve it manually when data is large — instead, **Gradient Descent** updates weights iteratively:

## 🧮 UPDATE RULE:

```
w = w - α * ∂L/∂w
b = b - α * ∂L/∂b
```

Where:

α = learning rate

∂L/∂w = partial derivative of loss w.r.t. weight

## 🔄 STEPS OF GRADIENT DESCENT:

Initialize weights `w` and `b` randomly

Calculate predictions: `ŷ = wx + b`

Compute cost (MSE)

Compute gradients

Update `w` and `b`

Repeat until convergence

# 📌 MSE AND R² IN DEPTH

## 🧮 MEAN SQUARED ERROR (MSE)

Measures **average squared difference** between actual and predicted values:

```
MSE = (1/n) * Σ(y - ŷ)^2
```

**Lower MSE** → better model

Used as a cost function in regression

## 🧠 R-SQUARED (R² SCORE)

Represents how much variance in target `Y` is explained by input `X`:

```
R² = 1 - (SS_res / SS_tot)
```

Where:

SS_res = Σ(y - ŷ)^2

SS_tot = Σ(y - mean(y))^2

✅ $R^2$ close to 1: Good fit
✅ $R^2$ close to 0: Bad fit

# ✍️ SIMPLE LINEAR REGRESSION EXAMPLE

```
from sklearn.linear_model import LinearRegression
import numpy as np
import pandas as pd

# Dataset
data = pd.read_csv("Salary_dataset.csv")
X = np.array(data['YearsExperience']).reshape(-1, 1)
y = data['Salary']

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

# Predict
print("Predicted salary for 5 years experience:",
model.predict([[5]]))
```

## 🎯 OUTPUT

Coefficient (w): model.coef_

Intercept (b): model.intercept_

R-squared: model.score(X, y)

# ✍️ MULTIPLE LINEAR REGRESSION EXAMPLE

```
# Dataset: Housing.csv with ['area', 'bedrooms',
'stories']
X = np.array(data[['area', 'bedrooms', 'stories']])
y = data['price']

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)
model = LinearRegression()
model.fit(X_train, y_train)

print("R^2 score:", r2_score(y_test,
model.predict(X_test)))
```

## 🔍 INTERPRETATION

Predict house price using multiple features

Visualize actual vs predicted prices

# ✅ SUMMARY TABLE

| Concept | Description |
| --- | --- |
| **Gradient Descent** | Optimizes parameters to minimize cost |
| **MSE** | Average squared error between actual and predicted |
| **Learning Rate (α)** | Controls step size during updates |
| **Derivatives** | Calculus tool to find slope for parameter updates |
| **$R^2$ Score** | Measures model accuracy (0 to 1 scale) |
| **Multiple Regression** | Extends linear regression to multiple input variables |

# 🚀 FINAL TAKEAWAY

"Gradient Descent is the heart of machine learning optimization — powered by calculus, evaluated by MSE, and validated by $R^2$ — it helps us build accurate and reliable prediction models."