Multiple Linear Regression in Python

Multiple Linear Regression is a fundamental statistical technique used to model the relationship between one **dependent variable and multiple independent variables.**

It is an extension of simple linear regression, which models the relationship between a dependent variable and one or more independent variable.

Mathematically, it is represented as :-

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_n X_n + \varepsilon$$

Where:

- Y = Dependent variable (Target variable)
- $X_1, X_2, ...X_n$ = Independent variables (Predictors)
- β_0 = Intercept (Constant term)
- $\beta_1, \beta_2, ...\beta_n$ = Regression coefficients
- ε = Error term

Key Terminology

- **Dependent Variable (Y):** The outcome we aim to predict.
- **Independent Variables (X):** The factors influencing the dependent variable.
- **Regression Coefficients (β\beta):** They measure the impact of each independent variable on Y.
- **Intercept (β0\beta_0):** Baseline value of Y when all X = 0
- **Error Term (ε):** The difference between actual and predicted values.

Project : Predicting Profit For Startup

Dataset : https://raw.githubusercontent.com/yash240990/Python/master/Startups_Data.csv

	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	New York	192261.83
1	162597.70	151377.59	443898.53	California	191792.06
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94

🔹 What is One-Hot Encoding?

One-Hot Encoding is a **data preprocessing technique** used to convert **categorical (text) data** into **numerical form** so that **machine learning models** can understand it.

🔹 Why we need it?

Most ML models (like Linear Regression, Random Forest, Neural Networks) **cannot work directly with text** — they need **numbers**.
So we must convert categories like "Red", "Blue", "Green" into numbers **without implying any order or ranking**.

	A	B	C
1	**Player City**	**Player Score**	**Player Status**
2	Kuala Lumpur	450	1
3	Klang	390	1
4	Malacca	670	1
5	Johor Bahru	427	1
6	Shah Alam	389	1
7	Klang	250	0
8	Malacca	120	0
9	Kuala Lumpur	109	0
10	Ipoh	423	1

Step 1 : Find The Number Of Unique Values

✅ **Number of unique values = 6**

Kuala Lumpur

Klang

Malacca

Johor Bahru

Shah Alam

Ipoh

Step 2 : Sort them in alphabetical order

Ipoh

Johor Bahru

Klang

Kuala Lumpur

Malacca

Shah Alam

Step 3 : After **One-Hot Encoding**, the model actually *sees this* 👇

Ipoh	Johor Bahru	Klang	Kuala Lumpur	Malacca	Shah Alam	Player Score	Player Status
0	0	0	1	0	0	450	1
0	0	1	0	0	0	390	1
0	0	0	0	1	0	670	1
0	1	0	0	0	0	427	1
0	0	0	0	1	0	389	1
0	0	1	0	0	0	250	0
0	0	0	0	1	0	120	0
0	0	0	1	0	0	109	0
1	0	0	0	0	0	423	1

✅ So for the model,

Kuala Lumpur	→	**0**	**0**	**0**	**1**	**0**	**0**
Klang	→	**0**	**0**	**1**	**0**	**0**	**0**
Malacca	→	**0**	**0**	**0**	**0**	**1**	**0**

🧠 What is "Train-Test Split"?

In **Machine Learning (ML)**, we teach computers to make predictions or decisions based on **data**.
To make sure our model learns properly **and doesn't just memorize**, we divide our data into two parts:

1. **Training set** – used to **teach** the model
2. **Testing set** – used to **check** if the model really learned or just memorized

🎓 Think of it like studying for an exam:

- You have **a book of 100 questions**.
- You **practice** with **80 questions** → this is your **training data**.
- Then, you **test yourself** with the remaining **20 questions** you haven't seen → this is your **testing data**.

If you do well on the **new 20 questions**, it means you actually **understood** the topic, not just memorized answers.

That's exactly how ML works!

🧩 Why do we need Train-Test Split?

If we train and test the model on the **same data**, it will look like the model performs perfectly —
but in reality, it has **just memorized the answers**, not **learned patterns**.

To test the model's **real-world performance**, we keep some data hidden (test set).

🌱 What is random_state (and why 42?)

When we use train_test_split, the computer **randomly** decides which data goes into the **training set** and which goes into the **testing set**.

So if you run the code **multiple times**,
you might get **different splits each time**, like this 👇

🧮 Example (without random_state):

Run 1:

```less
Training Data: [1, 2, 4, 5, 6, 9, 10, 3]
Testing Data: [7, 8]
```

Run 2:

```less
Training Data: [1, 3, 5, 6, 7, 8, 10, 2]
Testing Data: [4, 9]
```

Notice? The split changes every time because it's random.

🧩 So what does random_state do?

random_state acts like a **seed** — it makes sure the random process happens **the same way every time** you run the code.

That means if you (or someone else) run your code again,
you'll get **exactly the same split**.

🧮 **Example (with random_state=42):**

Run 1:

```less
Training Data: [1, 3, 5, 6, 7, 10, 8, 2]
Testing Data: [4, 9]
```

Run 2 (same code again):

```less
Training Data: [1, 3, 5, 6, 7, 10, 8, 2]
Testing Data: [4, 9]
```

✅ Same every time — because we "locked" the randomness.

🧠 Why number 42?

There's **nothing special** about 42 mathematically —
it's just a popular "inside joke" among programmers,
from a famous sci-fi book *"The Hitchhiker's Guide to the Galaxy"*,
where **42 is "the answer to life, the universe, and everything."**

So, people just use it as a default value —
you can use **any number** (like random_state=7 or 123) —
as long as you use the **same one**, you'll get consistent results.