OVERVIEW

This program performs **Linear Regression** from scratch (without using scikit-learn's model).
It predicts **Salary based on Years of Experience** using a dataset salary_dataset.csv.

We'll break it into sections:

📦 1. Importing Libraries

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
```

Explanation:

- **NumPy (np)** → For numerical calculations (arrays, math operations).
- **Pandas (pd)** → For reading and handling datasets (like Excel/CSV files).
- **Matplotlib (plt)** → For drawing graphs (visualizing data).
- **train_test_split** → A helper from scikit-learn that splits your data into:
    - **Training set:** used to train the model.
    - **Testing set:** used to test how well the model predicts.

2. Creating a Custom Linear Regression Class

This is the heart of the code.

a. Define the class

```
class Linear_Regression:

    def __init__(self, learning_rate=0.01, no_of_iterations=1000):

        self.learning_rate = learning_rate

        self.no_of_iterations = no_of_iterations
```

Explanation:

- The __init__ function runs automatically when you create an object.

- **learning_rate:** How big each step of learning is.
  (Too high = overshoot, too low = too slow.)
- **no_of_iterations:** How many times the model should learn (loop through data).

b. Fit function (Training the model)

```
def fit(self, X, Y):
    self.m, self.n = X.shape
    self.w = np.zeros(self.n)
    self.b = 0
    self.X = X
    self.Y = Y
    for i in range(self.no_of_iterations):
        self.update_weights()
```

Explanation:

- **X** = input features (Years of Experience).
- **Y** = output labels (Salary).
- self.m → number of samples (rows).
- self.n → number of features (columns).

We initialize:

- self.w → weight(s) (starts as 0).
- self.b → bias (starts as 0).

Then we run update_weights() repeatedly to improve w and b.

c. Update weights (Gradient Descent)

```
def update_weights(self):
    Y_pred = self.predict(self.X)
    dw = -(2 * (self.X.T).dot(self.Y - Y_pred)) / self.m
```

```
    db = -2 * np.sum(self.Y - Y_pred) / self.m

    self.w -= self.learning_rate * dw

    self.b -= self.learning_rate * db
```

Explanation:

Here we apply **Gradient Descent**, a learning algorithm.

Let's break it simply:

1. **Y_pred** → model's current predictions = (w × X + b)
2. **Error** = (Y - Y_pred) → how far the predictions are from the real values.
3. **dw** → derivative (slope) showing how w should change.
4. **db** → derivative showing how b should change.
5. Update w and b by moving opposite to the error:
6. w = w - learning_rate * dw
7. b = b - learning_rate * db

This process repeats many times, slowly improving accuracy.

d. Predict function

```
def predict(self, X):

    return X.dot(self.w) + self.b
```

Explanation:

Formula for a straight line:
[
y = w × x + b
]
This function calculates predicted salaries given input years of experience.

3. Data Handling

```
salary_data = pd.read_csv("salary_dataset.csv")
```

- Reads the dataset from a CSV file into a **DataFrame**.

```
X = salary_data[['YearsExperience']].values
```

Y = salary_data['Salary'].values

- Extracts only the relevant columns:
  - **X** → YearsExperience (input)
  - **Y** → Salary (output)
- .values converts them to NumPy arrays for calculation.

4. Splitting the Dataset

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33, random_state=2)

Explanation:

- **33%** of data → testing
- **67%** → training
- random_state=2 ensures the same split every time you run it (for reproducibility).

5. Model Training

model = Linear_Regression(learning_rate=0.02, no_of_iterations=1000)

model.fit(X_train, Y_train)

Explanation:

- Creates a model object with a learning rate of 0.02.
- Calls fit() to train the model — internally runs gradient descent 1000 times.

6. Display Learned Parameters

print("Weight:", model.w[0])

print("Bias:", model.b)

- These are the final values of w and b after training.
- They define the **best-fit line**:
  [
  Salary = w × YearsExperience + b
  ]

7. Model Evaluation

Y_pred = model.predict(X_test)

```
print("Predicted Values:", Y_pred)
```

- Uses the trained model to predict salaries for **unseen test data**.
- Prints out the predictions.

8. Visualization

```
plt.scatter(X_test.flatten(), Y_test, color='red', label='Actual Data')

plt.plot(X_test.flatten(), Y_pred, color='blue', label='Predicted Line')

plt.xlabel('Years of Experience')

plt.ylabel('Salary')

plt.title('Experience vs Salary Prediction')

plt.legend()

plt.show()
```

Explanation:

- **Red points:** Actual data from the test set.
- **Blue line:** The line predicted by your model.
- The plot visually shows how well your model fits the data.

9. Summary of How It Works

Step	What Happens	Code Part
1️⃣	Load data	pd.read_csv()
2️⃣	Split into train/test	train_test_split()
3️⃣	Initialize model	Linear_Regression()
4️⃣	Train model	fit() → update_weights()
5️⃣	Predict results	predict()
6️⃣	Visualize predictions	matplotlib plot

10. Real Concept Behind It

You're training a simple straight line that best fits the data:
[
Salary = (Weight × YearsExperience) + Bias
]

The model **learns** the best Weight (w) and Bias (b) by minimizing the **error** (difference between real and predicted salaries) using **gradient descent**.