Experiment Name: Implement Linear Regression

a) from scratch [without tools/library] and
b) using scikit-learn

Visit the link and study:

https://www.geeksforgeeks.org/machine-learning/ml-linear-regression/

For your program you may use the following dataset:

1. **https://www.kaggle.com/datasets/ashydv/advertising-dataset**
2. **https://www.kaggle.com/datasets/shree1992/housedata**

1. Objective

To predict **employee salary** based on **years of experience** using Linear Regression — implemented both **manually (from scratch)** and **using Scikit-learn**.

---

2. Dataset

We'll use a simple dataset (`Salary_Data.csv`) containing:

YearsExperience	Salary
1.1	39343
1.3	46205
1.5	37731
2.0	43525
2.2	39891
2.9	56642
3.0	60150
3.2	54445
3.2	64445
3.7	57189
3.9	63218
4.0	55794
4.5	56957
4.9	57081
5.3	61111

YearsExperience	Salary
5.9	67938
6.0	66029
6.8	83088
7.1	81363
7.9	93940
8.2	91738
8.7	98273
9.0	101302
9.5	113812
9.6	109431
10.3	122391
10.5	121872

---

3. Implementation (From Scratch)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Load dataset
data = pd.read_csv("Salary_Data.csv")
X = data["YearsExperience"].values
y = data["Salary"].values

# Mean
mean_x = np.mean(X)
mean_y = np.mean(y)

# Calculate slope (m) and intercept (c)
num = np.sum((X - mean_x) * (y - mean_y))
den = np.sum((X - mean_x)**2)
m = num / den
c = mean_y - m * mean_x

print("Slope (m):", m)
print("Intercept (c):", c)

# Predict
y_pred = m * X + c

# Visualization
plt.scatter(X, y, color='blue', label='Actual data')
```

```
plt.plot(X, y_pred, color='red', label='Regression Line')
plt.xlabel("Years of Experience")
plt.ylabel("Salary")
plt.title("Linear Regression from Scratch")
plt.legend()
plt.show()
```

✅ **Output:** Displays a regression line fitting the salary vs experience data.

Here,

Mathematical Explanation

1. Formula Behind It

We are using the **Least Squares Method** to fit a straight line of the form:

y=mX+c

The goal is to find `m` (slope) and `c` (intercept) such that the line minimizes the error between the predicted and actual values.

---

2. Slope Calculation (`m`)

Mathematically:

$$m = \frac{\sum(X_i - \bar{X})(Y_i - \bar{Y})}{\sum(X_i - \bar{X})^2}$$

```
num = np.sum((X - mean_x) * (y - mean_y))  # Numerator
den = np.sum((X - mean_x)**2)              # Denominator
m = num / den
```

🔹 **Meaning:**

- The numerator measures how **X and Y vary together** (covariance).
- The denominator measures how **X varies** by itself (variance of X).
- Their ratio gives the slope — **how much Y changes for a unit change in X**.

Example:
If `m = 9360`, it means **for every extra year of experience, salary increases by 9360** (on average).

---

3. Intercept Calculation (`c`)

$$c = \bar{Y} - m\bar{X}$$

In Python:

```
c = mean_y - m * mean_x
```

🔷 **Meaning:**
When `X = 0`, the expected value of `Y` is `c`.
It's the point where the regression line crosses the Y-axis.

Example:
If `c = 26780`, it means **even with 0 years of experience, the model predicts a salary of 26,780** (base pay).

---

4. Summary of What's Happening

Step	Formula	Description
Compute Mean	`mean_x`, `mean_y`	Average of X and Y
Compute Slope	$m = \frac{\sum(X_i - \bar{X})(Y_i - \bar{Y})}{\sum(X_i - \bar{X})^2}$	Change in Y for each change in X
Compute Intercept	$c = \bar{Y} - m\bar{X}$	Predicted Y when X = 0

---

5. Visual Intuition

The line `y = mX + c` is drawn in such a way that:

- The **sum of squared errors** between actual and predicted Y values is **minimum**.
- That's why it's called the **"least squares regression line."**

4. Implementation (Using Scikit-learn)

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

# Load dataset
data = pd.read_csv("Salary_Data.csv")
X = data[['YearsExperience']]
y = data['Salary']

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Create and train model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

# Evaluation
print("Intercept (c):", model.intercept_)
print("Slope (m):", model.coef_[0])
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R² Score:", r2_score(y_test, y_pred))

# Visualization
plt.scatter(X_test, y_test, color='blue', label='Actual data')
plt.plot(X_test, y_pred, color='red', linewidth=2,
label='Predicted line')
plt.xlabel("Years of Experience")
plt.ylabel("Salary")
plt.title("Linear Regression using Scikit-learn")
plt.legend()
plt.show()
```

✅ **Output Includes:**

- Regression equation (e.g., `Salary = 9360.26 * YearsExperience + 26780.09`)
- Model performance:
  - **MSE (Mean Squared Error)** — how far predictions are from actual
  - **R^2 Score** — goodness of fit (closer to 1 means better fit)

---

5. Results and Analysis

Metric	Value (Example)
Slope (m)	9360.26
Intercept (c)	26780.09
MSE	2.12×10^7
R^2 Score	0.98

Interpretation:
The high R^2 score (close to 1) indicates a strong linear relationship between experience and salary — meaning the model can predict salary accurately based on experience.

---

6. Learning Outcomes

- Understood the mathematical foundation of linear regression.
- Implemented regression manually and using a machine learning library.
- Learned how to evaluate model performance using error metrics.
- Gained practical skills in visualizing regression models.

Explanation:

Split data into training and test sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

◆ Purpose

In Machine Learning, we **don't train and test a model on the same data** — because that would make it memorize rather than generalize.

So, we **split the dataset** into:

- **Training Set (X_train, y_train):** Used to train the model (learn patterns).
- **Test Set (X_test, y_test):** Used to evaluate how well the model performs on unseen data.

This ensures we measure the model's **real-world predictive accuracy**.

◆ Parameters Explained

Parameter	Description
`X`	Input features (independent variable, e.g., YearsExperience)
`y`	Target variable (dependent variable, e.g., Salary)
`test_size=0.2`	20% of the data is used for testing, 80% for training
`random_state=42`	Ensures reproducibility — same random split every time

Example

Suppose your dataset has **30 rows**:

- `test_size=0.2` → 20% of 30 = 6 rows used for testing
- Remaining 24 rows → used for training

After Splitting

Variable	Contains	Example Rows
`X_train`	YearsExperience for training	`[1.1, 2.2, 4.5, 5.3, 7.9, 9.6, ...]`
`y_train`	Salary for training	`[39343, 39891, 56957, 61111, 93940, 109431, ...]`
`X_test`	YearsExperience for testing	`[3.2, 6.0, 9.0, 10.3, ...]`
`y_test`	Actual salary for testing	`[54445, 66029, 101302, 122391, ...]`

◆ Why We Use `train_test_split()`

✅ To **evaluate performance** on unseen data
✅ To **prevent overfitting** (memorizing training data)
✅ To **measure generalization** ability
✅ To **reproduce** results using `random_state`

Optional Tip for Students

You can also use different split ratios:

```
train_test_split(X, y, test_size=0.3)    # 70% train, 30% test
train_test_split(X, y, test_size=0.1)    # 90% train, 10% test
```