Name – Yash Mav | **Div** - TY09 | **Roll no** - 33 | **Subject** - DWM | **Experiment No** - 6

Aim: To implement a simple linear regression algorithm using Python to predict output based on input data.

Introduction (Theory):

Linear Regression is a **supervised learning algorithm** used for **predictive modeling**. It models the relationship between a **dependent variable** (target) and one or more **independent variables** (features) using a **linear equation**. The equation of a simple linear regression is: `y = mx + c` Where:

- `y` is the predicted value
- `m` is the slope (coefficient)
- `x` is the input feature
- `c` is the intercept

Scikit-learn's `LinearRegression` model simplifies the process of **training** and **predicting**.

Procedure:

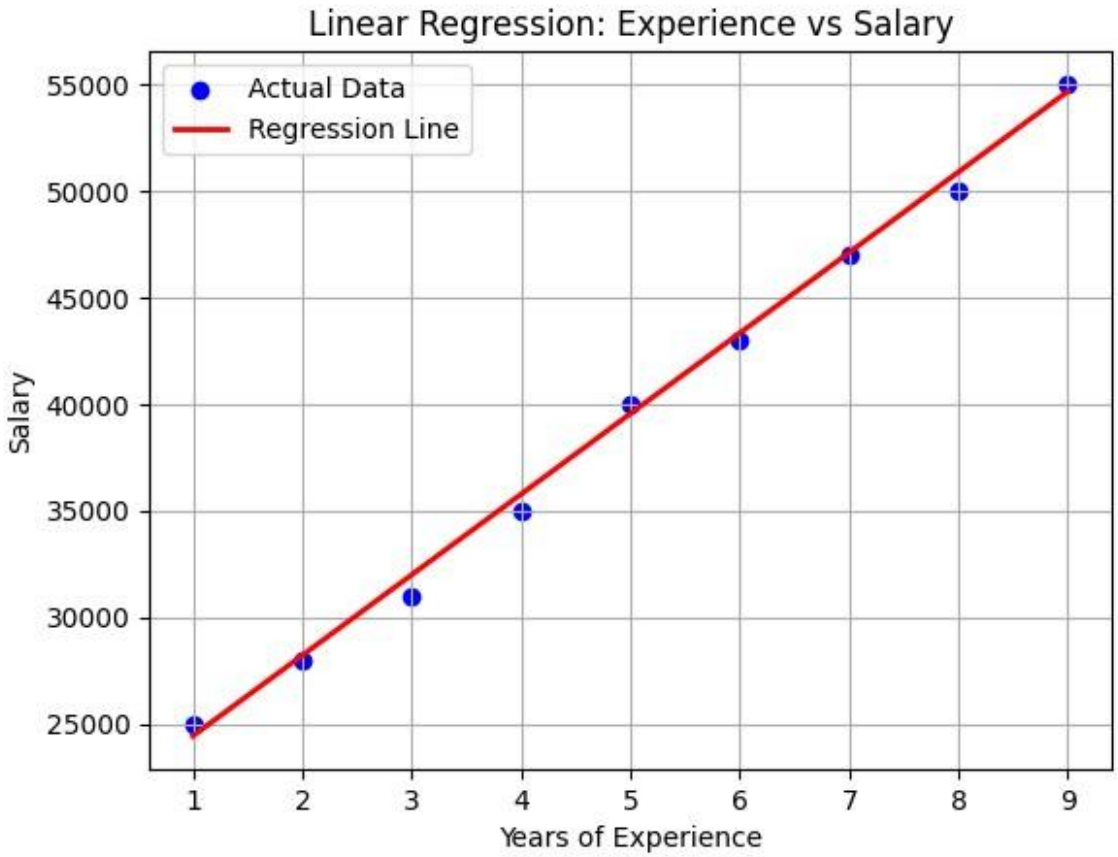
1. **Import Libraries:**
   - Use `numpy`, `matplotlib.pyplot`, and `sklearn.linear_model`.
   - Import `train_test_split` from `sklearn.model_selection`.
2. **Load and Prepare Data:**
   - Create or load input (e.g., experience) and output (e.g., salary) data.
   - Format the data as arrays or DataFrames.
3. **Split the Dataset:**
   - Use `train_test_split()` to create training and testing sets.
4. **Train the Model:**
   - Create `LinearRegression()` object and use `.fit()` with training data.
5. **Predict and Evaluate:**
   - Predict using `.predict(X_test)`.
   - Evaluate using `mean_squared_error()` and `r2_score()`.
6. **Visualize Results:**
   - Plot data points and regression line using `matplotlib`.

Program Code:

```
# Importing necessary libraries
import numpy as np import
matplotlib.pyplot as plt
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression from
sklearn.metrics import mean_squared_error, r2_score

# Sample data (Years of Experience vs Salary)
X = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9]).reshape(-1, 1)  # Years of
Experience
y = np.array([25000, 28000, 31000, 35000, 40000, 43000, 47000, 50000,
55000])  # Salary

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=0)

# Creating and training the Linear Regression model
model = LinearRegression() model.fit(X_train,
y_train)

# Predicting the values y_pred
= model.predict(X_test)

# Display the title before output
from IPython.display import display, Markdown
display(Markdown("**Implementation/Output snap shot:**"))

# Displaying the actual and predicted values
print("Actual values:", y_test)
print("Predicted values:", y_pred.astype(int))

# Evaluating the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error (MSE):", round(mse, 2))
print("R² Score:", round(r2, 2))

# Plotting the regression line
plt.scatter(X, y, color='blue', label='Actual Data')
plt.plot(X, model.predict(X), color='red', linewidth=2,
label='Regression Line')
plt.title("Linear Regression: Experience vs Salary")
plt.xlabel("Years of Experience")
plt.ylabel("Salary") plt.legend() plt.grid(True)
plt.show()
<IPython.core.display.Markdown object>
Actual values: [50000 31000]

Predicted values: [50871 31984]
```

```
Mean Squared Error (MSE): 865021.75
R² Score: 0.99
```



Conclusion: In this experiment, we successfully implemented a simple Linear Regression model using Python and the scikit-learn library. We trained the model using sample data, made predictions, and evaluated the model using MSE and R^2 Score. The visualization confirmed the linear relationship between the input and output variables. This experiment helped us understand how prediction algorithms work and how to evaluate their performance. **Review Questions:**

1. **What are the key steps involved in implementing a simple linear regression model using Python and scikit-learn? Ans.** The following are the key steps to implement a **simple linear regression** using **Python** and **scikit-learn**:
   - **Import libraries**: Required modules like `pandas`, `numpy`, `matplotlib`, `sklearn`.
   - **Load/prepare dataset**: Read and preprocess the data.
   - **Split the data** using `train_test_split()`: Separate into training and testing datasets.
   - **Create and train the model**: Use `LinearRegression()` and `.fit()` to train.
   - **Predict outcomes**: Use `.predict()` to make predictions on test data.
   - **Evaluate**: Use metrics like Mean Squared Error (MSE) and R^2 Score.

- **Visualize results**: Plot regression line and residuals for better understanding.

2. **How can you evaluate the performance of a linear regression model in Python? List and explain at least two metrics. Ans.** Two commonly used metrics to evaluate a linear regression model are:

- **Mean Squared Error (MSE)**: Measures the **average of the squares of errors** (differences between actual and predicted values). A lower MSE indicates better accuracy.
- **R^2 Score (Coefficient of Determination)**: Indicates how well the model explains the variability in the dependent variable. A value closer to **1** signifies a good model fit.

1. **What is the role of the train_test_split() function in building a linear regression model, and why is it important? Ans.** The `train_test_split()` function is used to **divide the dataset** into training and testing sets.

- This allows the model to be **trained on one portion** of the data and **tested on another**, which helps:
  - Evaluate the model's performance on **unseen data**.
  - Prevent **overfitting**, ensuring better **generalization**.

GitHub Link: https://github.com/yash69420/DWM