```
import pandas as pd

df = pd.read_csv('/content/Salary_dataset.csv')
display(df.head())
```

	Unnamed: 0	YearsExperience	Salary
0	0	1.2	39344.0
1	1	1.4	46206.0
2	2	1.6	37732.0
3	3	2.1	43526.0
4	4	2.3	39892.0

```
print("First 5 rows:")
display(df.head())

print("\nLast 5 rows:")
display(df.tail())
```

```
First 5 rows:
```

	Unnamed: 0	YearsExperience	Salary
0	0	1.2	39344.0
1	1	1.4	46206.0
2	2	1.6	37732.0
3	3	2.1	43526.0
4	4	2.3	39892.0

```
Last 5 rows:
```

	Unnamed: 0	YearsExperience	Salary
25	25	9.1	105583.0
26	26	9.6	116970.0
27	27	9.7	112636.0
28	28	10.4	122392.0
29	29	10.6	121873.0

```
# Input variable (YearsExperience)
X = df[['YearsExperience']]
```

```
# Output variable (Salary)
y = df['Salary']

print("Input (X) head:")
display(X.head())

print("\nOutput (y) head:")
display(y.head())
```

```
Input (X) head:
```

	YearsExperience
0	1.2
1	1.4
2	1.6
3	2.1
4	2.3

```
Output (y) head:
```

	Salary
0	39344.0
1	46206.0
2	37732.0
3	43526.0
4	39892.0

dtype: float64

```
# Independent variable (input)
X = df[['YearsExperience']]

# Dependent variable (output)
y = df['Salary']

print("Independent variable (X) head:")
display(X.head())

print("\nDependent variable (y) head:")
display(y.head())
```

```
Independent variable (X) head:
```

	YearsExperience
0	1.2
1	1.4
2	1.6
3	2.1
4	2.3

```
Dependent variable (y) head:
```

	Salary
0	39344.0
1	46206.0
2	37732.0
3	43526.0
4	39892.0

dtype: float64

```
# Convert X and y to NumPy arrays
X_np = X.values
y_np = y.values

print("X as a NumPy array (first 5 elements):")
print(X_np[:5])

print("\ny as a NumPy array (first 5 elements):")
print(y_np[:5])
```

```
X as a NumPy array (first 5 elements):
[[1.2]
 [1.4]
 [1.6]
 [2.1]
 [2.3]]

y as a NumPy array (first 5 elements):
[39344. 46206. 37732. 43526. 39892.]
```

```
print(f"Original shape of X_np: {X_np.shape}")
print(f"Original shape of y_np: {y_np.shape}")

# Reshape y_np to a 2D array (n_samples, 1)
# X_np is already in (n_samples, n_features) format, so no reshape needed for X
```

```
y_np_reshaped = y_np.reshape(-1, 1)

print(f"\nReshaped shape of X_np: {X_np.shape}")
print(f"Reshaped shape of y_np: {y_np_reshaped.shape}")
```

```
Original shape of X_np: (30, 1)
Original shape of y_np: (30,)

Reshaped shape of X_np: (30, 1)
Reshaped shape of y_np: (30, 1)
```

```
# Initialize slope (m) and intercept (b) to 0
m = 0
b = 0

print(f"Initial Slope (m): {m}")
print(f"Initial Intercept (b): {b}")
```

```
Initial Slope (m): 0
Initial Intercept (b): 0
```

```
# Calculate predictions using the current slope (m) and intercept (b)
# y_pred = m * X_np + b
y_pred = m * X_np + b

print("First 5 predicted values (y_pred):")
print(y_pred[:5])
```

```
First 5 predicted values (y_pred):
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]]
```

```
# Calculate the Mean Squared Error (MSE)
# MSE = (1/n) * sum((y_pred - y_actual)^2)

# Calculate the error (difference between predicted and actual)
error = y_pred - y_np_reshaped

# Square the error
squared_error = error**2

# Sum the squared errors and divide by the number of samples (n)
mse = squared_error.mean()

print(f"Mean Squared Error (MSE): {mse}")
```

```
Mean Squared Error (MSE): 6503107277.733334
```

```
# Calculate the Mean Squared Error (MSE)
# MSE = (1/n) * sum((y_pred - y_actual)^2)

# Calculate the error (difference between predicted and actual)
error = y_pred - y_np_reshaped

# Square the error
squared_error = error**2

# Sum the squared errors and divide by the number of samples (n)
mse = squared_error.mean()

print(f"Mean Squared Error (MSE): {mse}")
```

```
Mean Squared Error (MSE): 6503107277.733334
```

```
learning_rate = 0.01
epochs = 1000

print(f"Learning Rate: {learning_rate}")
print(f"Number of Epochs: {epochs}")
```

```
Learning Rate: 0.01
Number of Epochs: 1000
```

Implement Gradient Descent

Subtask:

Create a loop to perform gradient descent. In each iteration, calculate predictions, compute the error, calculate the gradients for 'm' and 'b', and update 'm' and 'b' using the learning rate.

Reasoning: The subtask requires creating a loop to perform gradient descent by calculating predictions, errors, gradients, and updating the slope and intercept using the defined learning rate and number of epochs.

```
n = len(X_np)

for epoch in range(epochs):
    # a. Calculate predictions
    y_pred = m * X_np + b

    # b. Calculate the error
    error = y_pred - y_np_reshaped
```

```
        # c. Calculate the gradient for m (dm)
        dm = (1/n) * (error * X_np).sum()

        # d. Calculate the gradient for b (db)
        db = (1/n) * error.sum()

        # e. Update m
        m = m - learning_rate * dm

        # f. Update b
        b = b - learning_rate * db

        # Optional: Print MSE every 100 epochs to observe progress
        if epoch % 100 == 0:
            mse = (error**2).mean()
            print(f"Epoch {epoch}: MSE = {mse:.2f}, m = {m:.2f}, b = {b:.2f}")

    print(f"\nFinal Slope (m): {m:.2f}")
    print(f"Final Intercept (b): {b:.2f}")
```

```
Epoch 0: MSE = 6503107277.73, m = 4850.04, b = 760.04
Epoch 100: MSE = 104651526.54, m = 12188.33, b = 6190.31
Epoch 200: MSE = 79900672.92, m = 11679.18, b = 9659.42
Epoch 300: MSE = 63498142.50, m = 11264.69, b = 12483.51
Epoch 400: MSE = 52628092.99, m = 10927.27, b = 14782.51
Epoch 500: MSE = 45424449.81, m = 10652.59, b = 16654.05
Epoch 600: MSE = 40650554.81, m = 10428.99, b = 18177.61
Epoch 700: MSE = 37486867.64, m = 10246.95, b = 19417.89
Epoch 800: MSE = 35390274.30, m = 10098.77, b = 20427.57
Epoch 900: MSE = 34000850.05, m = 9978.13, b = 21249.51

Final Slope (m): 9880.81
Final Intercept (b): 21912.59
```

Predict Salaries with Trained Model

Subtask:

Once the model is trained (i.e., 'm' and 'b' are optimized), use the final 'm' and 'b' values to predict salaries for the given 'YearsExperience' (X_np).

Reasoning: The subtask requires predicting salaries using the final optimized 'm' and 'b' values. I will apply the linear regression formula to calculate 'y_pred_final'.

```
y_pred_final = m * X_np + b

print("Predicted salaries (first 5 values):")
print(y_pred_final[:5])
```

```
Predicted salaries (first 5 values):
[[33769.56598975]
 [35745.72879067]
 [37721.8915916 ]
 [42662.2985939 ]
 [44638.46139482]]
```

Start coding or generate with AI.

```
# Calculate the Mean Squared Error (MSE) for the trained model
# MSE = (1/n) * sum((y_pred_final - y_actual)^2)

# Calculate the error (difference between predicted and actual salaries)
error_final = y_pred_final - y_np_reshaped

# Square the error
squared_error_final = error_final**2

# Sum the squared errors and divide by the number of samples (n)
mse_final = squared_error_final.mean()

print(f"Mean Squared Error (MSE) with trained model: {mse_final:.2f}")
```

```
Mean Squared Error (MSE) with trained model: 33080070.75
```

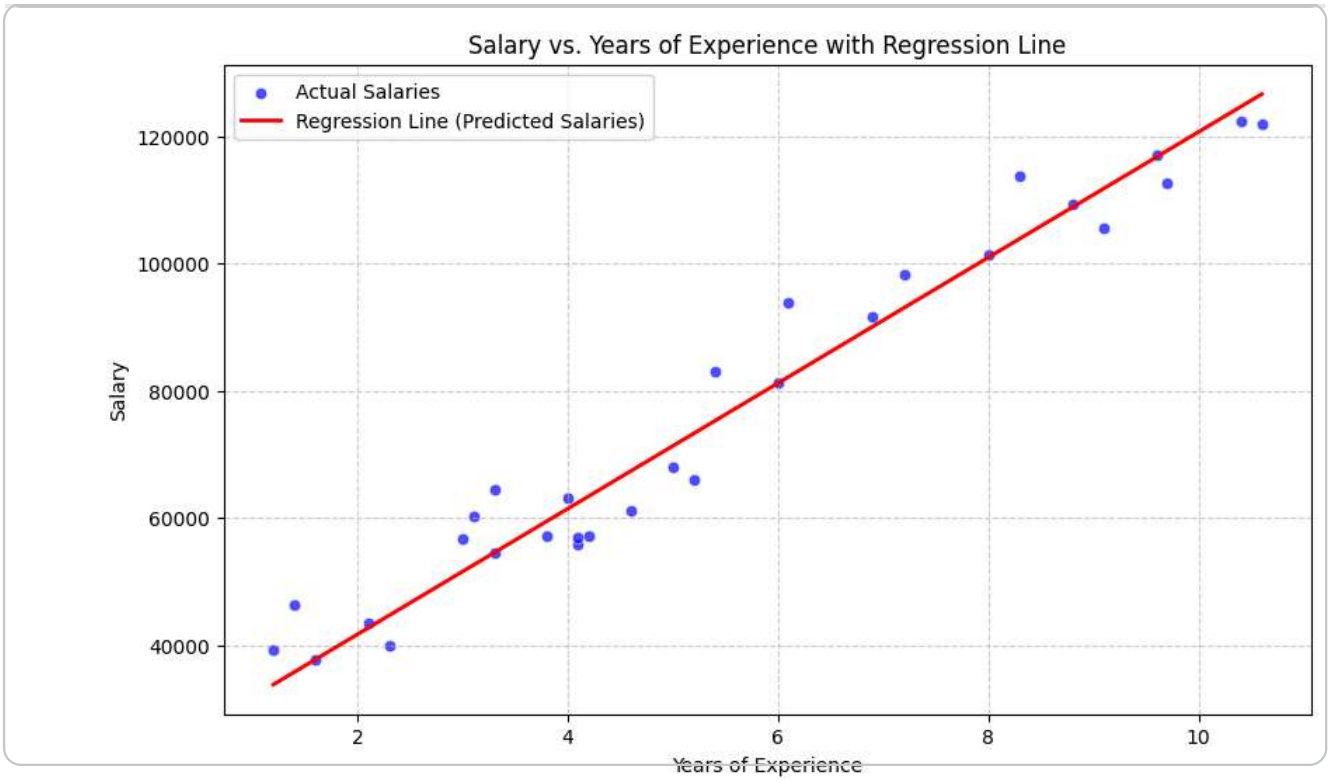
Start coding or generate with AI.

```
import matplotlib.pyplot as plt
import seaborn as sns

# Create a scatter plot of the actual data points
plt.figure(figsize=(10, 6))
sns.scatterplot(x=X_np.flatten(), y=y_np_reshaped.flatten(), label='Actual S

# Plot the regression line
sns.lineplot(x=X_np.flatten(), y=y_pred_final.flatten(), label='Regression l

# Add labels and title
plt.title('Salary vs. Years of Experience with Regression Line')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()
```




```
print(f"Final Optimized Slope (m): {m:.2f}")
print(f"Final Optimized Intercept (b): {b:.2f}")

# Predict salary for a new number of years of experience
new_years_experience = 15.0  # Example: Predict for 15 years of experience
predicted_salary = m * new_years_experience + b

print(f"\nPredicted Salary for {new_years_experience} Years of Experience: {pre
```

```
Final Optimized Slope (m): 9880.81
Final Optimized Intercept (b): 21912.59

Predicted Salary for 15.0 Years of Experience: 170124.80
```