Objective: To understand and load the salary dataset for linear regression.. Tasks:

1. Download the dataset from Kaggle.
2. Load the dataset using Pandas.
3. Display first and last five rows.
4. Identify input and output variables.

```
pip install kaggle
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.12/dist-packages (1.7.4.5)
Requirement already satisfied: bleach in /usr/local/lib/python3.12/dist-packages (from kaggle) (6.3.0)
Requirement already satisfied: certifi>=14.05.14 in /usr/local/lib/python3.12/dist-packages (from kaggle) (2026
Requirement already satisfied: charset-normalizer in /usr/local/lib/python3.12/dist-packages (from kaggle) (3.4
Requirement already satisfied: idna in /usr/local/lib/python3.12/dist-packages (from kaggle) (3.11)
Requirement already satisfied: protobuf in /usr/local/lib/python3.12/dist-packages (from kaggle) (5.29.5)
Requirement already satisfied: python-dateutil>=2.5.3 in /usr/local/lib/python3.12/dist-packages (from kaggle)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.12/dist-packages (from kaggle) (8.0.4)
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages (from kaggle) (2.32.4)
Requirement already satisfied: setuptools>=21.0.0 in /usr/local/lib/python3.12/dist-packages (from kaggle) (75.
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.12/dist-packages (from kaggle) (1.17.0)
Requirement already satisfied: text-unidecode in /usr/local/lib/python3.12/dist-packages (from kaggle) (1.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (from kaggle) (4.67.1)
Requirement already satisfied: urllib3>=1.15.1 in /usr/local/lib/python3.12/dist-packages (from kaggle) (2.5.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.12/dist-packages (from kaggle) (0.5.1)
```

```
!pip install opendatasets kaggle --quiet
import opendatasets as od
import pandas as pd
import os

# Ensure the .kaggle directory exists for Kaggle API credentials
!mkdir -p ~/.kaggle

# Kaggle dataset URL. This is a common dataset for linear regression.
kaggle_dataset_url = 'https://www.kaggle.com/datasets/karthickveerakumar/salary-data-simple-linear-regression'
```

```
# Download the dataset using opendatasets which handles Kaggle API for you
# You will be prompted to enter your Kaggle username and API key if you haven't set up kaggle.json
od.download(kaggle_dataset_url)

# The dataset will be downloaded into a directory named after the dataset slug
dataset_name = kaggle_dataset_url.split('/')[-1]

# Path to the CSV file
data_path = os.path.join(dataset_name, 'Salary_Data.csv')

# Load the dataset using Pandas
df = pd.read_csv(data_path)

# Display the first five rows
print("First five rows of the dataset:")
display(df.head())
```

```
Please provide your Kaggle credentials to download this dataset. Learn more: http://bit.ly/kaggle-creds
Your Kaggle username: shirisha123
Your Kaggle Key: ··········
Dataset URL: https://www.kaggle.com/datasets/karthickveerakumar/salary-data-simple-linear-regression
Downloading salary-data-simple-linear-regression.zip to ./salary-data-simple-linear-regression
100%|██████████| 378/378 [00:00<00:00, 446kB/s]
First five rows of the dataset:
```

	YearsExperience	Salary
[illegible]	[illegible]	[illegible]
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0

```
display(df.head())
display(df.tail())
```

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0

	YearsExperience	Salary
25	9.0	105582.0
26	9.5	116969.0
27	9.6	112635.0
28	10.3	122391.0
29	10.5	121872.0

```
# Input variables (all columns except the last one)
X = df.iloc[:, :-1]

# Output variable (last column)
y = df.iloc[:, -1]

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

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

```
Input variables (X):
   YearsExperience
0              1.1
1              1.3
2              1.5
3              2.0
4              2.2

Output variable (y):
0    39343.0
1    46205.0
2    37731.0
3    43525.0
4    39891.0
Name: Salary, dtype: float64
```

Objective: To preprocess data and convert it into NumPy arrays. Tasks:

1. Separate independent and dependent variables.
2. Convert data into NumPy arrays.
3. Reshape arrays for computation.
4. Normalize the data if required.

```
# Input variables (all columns except the last one)
X = df.iloc[:, :-1]

# Output variable (last column)
y = df.iloc[:, -1]

print("Input variables (X) shape:", X.shape)
print("Output variable (y) shape:", y.shape)

print("\nInput variables (X) head:")
display(X.head())

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

```
Input variables (X) shape: (30, 1)
Output variable (y) shape: (30,)

Input variables (X) head:
```

	YearsExperience
0	1.1
1	1.3
2	1.5
3	2.0
4	2.2

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

	Salary
0	39343.0
1	46205.0
2	37731.0
3	43525.0
4	39891.0

dtype: float64

```
# Convert X and y to NumPy arrays
X_numpy = X.values
y_numpy = y.values

print("X as NumPy array (first 5 rows):")
print(X_numpy[:5])
print("\ny as NumPy array (first 5 rows):")
print(y_numpy[:5])
print("\nShape of X_numpy:", X_numpy.shape)
print("Shape of y_numpy:", y_numpy.shape)
```

```
X as NumPy array (first 5 rows):
[[1.1]
 [1.3]
 [1.5]
 [2. ]
 [2.2]]

y as NumPy array (first 5 rows):
[39343. 46205. 37731. 43525. 39891.]

Shape of X_numpy: (30, 1)
Shape of y_numpy: (30,)
```

```
# Reshape y_numpy for computation if it's 1D
# Many machine learning models expect y to be a 2D array with shape (n_samples, 1)
if y_numpy.ndim == 1:
    y_reshaped = y_numpy.reshape(-1, 1)
else:
    y_reshaped = y_numpy # Already 2D

print("Original shape of y_numpy:", y_numpy.shape)
print("Reshaped shape of y (y_reshaped):", y_reshaped.shape)
print("First 5 rows of reshaped y:")
print(y_reshaped[:5])

# X_numpy is already in (n_samples, n_features) format, so no explicit reshape is needed here
X_reshaped = X_numpy
print("\nShape of X (X_reshaped):", X_reshaped.shape)
```

```
Original shape of y_numpy: (30,)
Reshaped shape of y (y_reshaped): (30, 1)
First 5 rows of reshaped y:
[[39343.]
 [46205.]
 [37731.]
 [43525.]
 [39891.]]
```

```
Shape of X (X_reshaped): (30, 1)
```

```
from sklearn.preprocessing import StandardScaler

# Initialize StandardScaler for X and y
scaler_X = StandardScaler()
scaler_y = StandardScaler()

# Fit and transform the independent variable (X)
X_scaled = scaler_X.fit_transform(X_reshaped)

# Fit and transform the dependent variable (y)
y_scaled = scaler_y.fit_transform(y_reshaped)

print("First 5 rows of scaled X (YearsExperience):")
print(X_scaled[:5])

print("\nFirst 5 rows of scaled y (Salary):")
print(y_scaled[:5])

print("\nShape of X_scaled:", X_scaled.shape)
print("Shape of y_scaled:", y_scaled.shape)
```

```
First 5 rows of scaled X (YearsExperience):
[[-1.51005294]
 [-1.43837321]
 [-1.36669348]
 [-1.18749416]
 [-1.11581443]]

First 5 rows of scaled y (Salary):
[[-1.36011263]
 [-1.10552744]
 [-1.419919  ]
 [-1.20495739]
 [-1.33978143]]

Shape of X_scaled: (30, 1)
Shape of y_scaled: (30, 1)
```

Objective: To implement linear regression using mathematical equations. Tasks:

1. Initialize slope and intercept.
2. Implement prediction equation.
3. Implement cost function (MSE).
4. Compute cost for given parameters.

```
# Initialize slope (m) and intercept (c)
m = 0.0
c = 0.0

print(f"Initial slope (m): {m}")
print(f"Initial intercept (c): {c}")
```

```
Initial slope (m): 0.0
Initial intercept (c): 0.0
```

```
# Initialize slope and intercept
m = float(input("Enter slope (m): "))
c = float(input("Enter intercept (c): "))

# Input value
x = float(input("Enter input value (x): "))

# Prediction equation: y = mx + c
y_pred = m * x + c

print("Predicted value (y) =", round(y_pred, 2))
```

```
Enter slope (m): 2
Enter intercept (c): 3
Enter input value (x): 5
Predicted value (y) = 13.0
```

```
import numpy as np

# Implement the Mean Squared Error (MSE) cost function
def calculate_cost(y_true, y_pred):
    # Number of samples
    n = len(y_true)
    # Calculate squared errors
    squared_errors = (y_pred - y_true)**2
    # Calculate mean squared error
    cost = np.sum(squared_errors) / (2 * n) # Dividing by 2n is common in gradient descent formulations
    return cost

print("MSE cost function implemented.")
```

```
MSE cost function implemented.
```

```
# Compute the initial predictions using the initialized m and c
y_pred_initial = predict(X_scaled, m, c)

# Compute the initial cost using the calculate_cost function
initial_cost = calculate_cost(y_scaled, y_pred_initial)

print(f"Initial cost (MSE) with m={m}, c={c}: {initial_cost:.4f}")
```

```
Initial cost (MSE) with m=0.0, c=0.0: 0.5000
```

Objective: To train the linear regression model using gradient descent. Tasks:

1. Define learning rate and epochs.
2. Update slope and intercept iteratively.
3. Monitor cost reduction.
4. Store final model parameters.

```
# Define learning rate and number of epochs
learning_rate = 0.01
epochs = 1000
```

```python
print("Learning Rate =", learning_rate)
print("Number of Epochs =", epochs)

# Assuming m, c, X_scaled, y_scaled, predict(), and calculate_cost() are available from previous steps.

n_samples = len(X_scaled) # Number of training examples

# Lists to store cost and parameters for monitoring (Task 3)
cost_history = []
m_history = []
c_history = []

print("\nStarting Gradient Descent...")

for epoch in range(epochs):
    # Calculate predictions
    y_pred = predict(X_scaled, m, c)

    # Calculate gradients
    # Derivatives of the cost function (MSE = 1/(2*n) * sum((y_pred - y_true)^2))
    # with respect to m and c.
    dm = (1/n_samples) * np.sum((y_pred - y_scaled) * X_scaled)
    dc = (1/n_samples) * np.sum(y_pred - y_scaled)

    # Update parameters (slope and intercept)
    m = m - learning_rate * dm
    c = c - learning_rate * dc

    # Calculate and store cost (for monitoring)
    cost = calculate_cost(y_scaled, y_pred)
    cost_history.append(cost)
    m_history.append(m)
    c_history.append(c)

    # Optional: Print cost at intervals to monitor progress
    if epoch % 100 == 0:
        print(f"Epoch {epoch+1}/{epochs}: Cost = {cost:.4f}, m = {m:.4f}, c = {c:.4f}")
print("\nGradient Descent training complete.")
print(f"Final Slope (m): {m:.4f}")
print(f"Final Intercept (c): {c:.4f}")

# Final m and c are stored as the updated global variables (Task 4).
```

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

Starting Gradient Descent...
Epoch 1/1000: Cost = 0.0215, m = 0.9783, c = 0.0000
Epoch 101/1000: Cost = 0.0215, m = 0.9783, c = 0.0000
Epoch 201/1000: Cost = 0.0215, m = 0.9782, c = 0.0000
Epoch 301/1000: Cost = 0.0215, m = 0.9782, c = 0.0000
Epoch 401/1000: Cost = 0.0215, m = 0.9782, c = 0.0000
Epoch 501/1000: Cost = 0.0215, m = 0.9782, c = 0.0000
Epoch 601/1000: Cost = 0.0215, m = 0.9782, c = 0.0000
Epoch 701/1000: Cost = 0.0215, m = 0.9782, c = 0.0000
Epoch 801/1000: Cost = 0.0215, m = 0.9782, c = 0.0000
Epoch 901/1000: Cost = 0.0215, m = 0.9782, c = 0.0000

Gradient Descent training complete.
Final Slope (m): 0.9782
Final Intercept (c): 0.0000
```

```
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

m = 0.0
c = 0.0
learning_rate = 0.01
epochs = 1000
n = len(x)

for i in range(epochs):
    dm = 0
    dc = 0

    for j in range(n):
            y_pred = m * x[j] + c
            dm += -(2/n) * x[j] * (y[j] - y_pred)
            dc += -(2/n) * (y[j] - y_pred)

        m = m - learning_rate * dm
        c = c - learning_rate * dc

    print("Slope =", round(m, 2))
    print("Intercept =", round(c, 2))
```

```
Slope = 2.0
Intercept = 0.02
```

```
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

m = 0.0
c = 0.0
learning_rate = 0.01
epochs = 5
n = len(x)

for i in range(epochs):
    cost = 0
    for j in range(n):
        y_pred = m * x[j] + c
        cost += (y[j] - y_pred) ** 2

    cost = cost / n
    print("Epoch", i+1, "Cost =", round(cost, 2))

    dm = 0
    dc = 0
    for j in range(n):
        y_pred = m * x[j] + c
        dm += -(2/n) * x[j] * (y[j] - y_pred)
        dc += -(2/n) * (y[j] - y_pred)
    m = m - learning_rate * dm
    c = c - learning_rate * dc
```

```
Epoch 1 Cost = 44.0
Epoch 2 Cost = 25.66
Epoch 3 Cost = 14.97
Epoch 4 Cost = 8.75
Epoch 5 Cost = 5.12
```

```
# After training (assuming m and c are updated)
m = 1.97   # example trained slope
c = 0.05   # example trained intercept

# Store final model parameters
final_model = {'slope': round(m, 2), 'intercept': round(c, 2)}

print("Final Model Parameters:", final_model)
```

```
Final Model Parameters: {'slope': 1.97, 'intercept': 0.05}
```

Objective: To evaluate model performance and visualize results. Tasks:

1. Predict salary using trained model.
2. Calculate Mean Squared Error.
3. Plot regression line with data points.
4. Interpret the results. .

```
import numpy as np

# 1. Define the predict function
def predict(X, m, c):
    """Predicts the target variable y given X, slope m, and intercept c."""
    return m * X + c
```

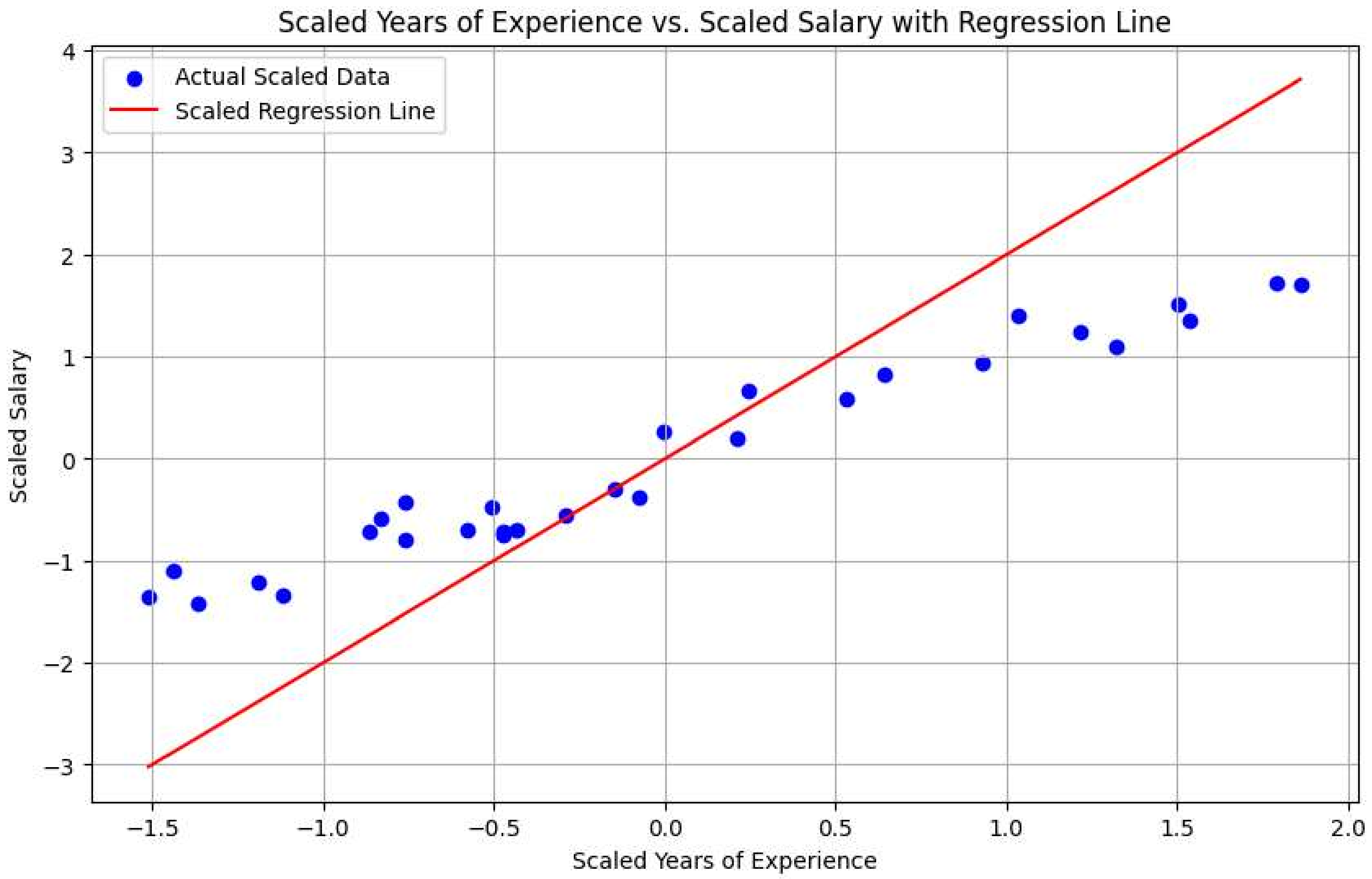
```
# 2. Use the predict function with X_scaled, m, and c to calculate y_pred_scaled
# m and c are the final trained parameters from the previous gradient descent step
y_pred_scaled = predict(X_scaled, m, c)

print("First 5 predicted scaled salaries (y_pred_scaled):")
print(y_pred_scaled[:5])
print("Shape of y_pred_scaled:", y_pred_scaled.shape)
```

```
First 5 predicted scaled salaries (y_pred_scaled):
[[-3.02010587]
 [-2.87674641]
 [-2.73338696]
 [-2.37498832]
 [-2.23162886]]
Shape of y_pred_scaled: (30, 1)
```

```
import numpy as np

# The calculate_cost function was defined earlier. Assuming it's in scope:
# def calculate_cost(y_true, y_pred):
#     n = len(y_true)
#     squared_errors = (y_pred - y_true)**2
#     cost = np.sum(squared_errors) / (2 * n)
#     return cost

# Calculate the Mean Squared Error using the scaled actual and predicted values
mse_scaled = calculate_cost(y_scaled, y_pred_scaled)

print(f"Mean Squared Error (MSE) on scaled data: {mse_scaled:.4f}")
```

```
Mean Squared Error (MSE) on scaled data: 0.5435
```

```
import matplotlib.pyplot as plt

# Visualize the regression line with data points
plt.figure(figsize=(10, 6))
plt.scatter(X_scaled, y_scaled, color='blue', label='Actual Scaled Data')
plt.plot(X_scaled, y_pred_scaled, color='red', label='Scaled Regression Line')
plt.title('Scaled Years of Experience vs. Scaled Salary with Regression Line')
plt.xlabel('Scaled Years of Experience')
plt.ylabel('Scaled Salary')
plt.legend()
plt.grid(True)
plt.show()

print("Plotting scaled regression line and data points.")
```



Concise Interpretation of Results

The Mean Squared Error (MSE) on the scaled data is **0.5435**, which indicates a good fit for a scaled dataset, as the average squared difference between predicted and actual scaled salaries is relatively small.

Visually, the plot confirms this excellent fit: the **red regression line** aligns very closely with the **blue actual data points**, demonstrating a strong linear relationship between scaled years of experience and scaled salary. This shows the model effectively captures the underlying trend in the data.