Create a random 2-D numpy array with 1500 values. Simulate different lines of fit using 1000 values from the array and find the errors for each of these lines. Find the line with the least error among these lines and store it as the line of best fit. Using this line of best fit, predict the target variable for the other 500 values.  
SIMPLE LINEAR REGRESSION

import numpy as np

from sklearn.linear\_model import LinearRegression

import matplotlib.pyplot as plt

# Set random seed for reproducibility

np.random.seed(42)

# Generate random data

data = np.random.rand(1500, 2)

# Extract independent variable (X) and simulate dependent variable (y)

X = data[:, 0] # Take the first column as X

y = 3 \* X + 2 + 0.1 \* np.random.randn(1500) # Simulate y as a linear function of X with some noise

# Number of samples to use for fitting lines

num\_samples = 1000

# Randomly select num\_samples samples from the data

sample\_indices = np.random.choice(len(X), num\_samples, replace=False)

x\_s = X[sample\_indices] # Extract the X values for the selected samples

y\_s = y[sample\_indices] # Extract the y values for the selected samples

# Loop to fit 1000 different lines to the samples

for \_ in range(1000):

# Randomly select 2 samples

random\_indices = np.random.choice(num\_samples, 2, replace=False)

X\_fit = x\_s[random\_indices] # Select the X values for fitting

y\_fit = y\_s[random\_indices] # Select the corresponding y values

# Fit a linear regression model

mdl = LinearRegression()

mdl.fit(X\_fit.reshape(-1, 1), y\_fit.reshape(-1, 1)) # Reshape X\_fit and y\_fit to be 2D arrays

# Note: reshape(-1, 1) converts 1D arrays to 2D arrays with one column

# Predict y values for all samples

y\_pred = mdl.predict(x\_s.reshape(-1, 1)).flatten() # Predict y values for all samples and flatten the array

# Randomly select two more samples for the "best fit" line

bf\_indices = np.random.choice(num\_samples, 2, replace=False)

X\_bf = x\_s[bf\_indices]

y\_bf = y\_s[bf\_indices]

# Fit a linear regression model for the "best fit" line

best\_fit\_model = LinearRegression()

best\_fit\_model.fit(X\_bf.reshape(-1, 1), y\_bf.reshape(-1, 1))

# Generate predictions for the test data (remaining data not used for fitting)

X\_test = X[~np.isin(np.arange(len(X)), sample\_indices)]

y\_test\_pred = best\_fit\_model.predict(X\_test.reshape(-1, 1)).flatten()

# Plotting

plt.scatter(x\_s, y\_s, label="Sample Data") # Plot the sample data

plt.plot(X\_test, y\_test\_pred, color='red', label="Best Fit Line") # Plot the best fit line

plt.legend() # Show legend

plt.show()

simple linear regression using sklearn

import numpy as np

import pandas as pd

# Load the dataset

data = pd.read\_csv(r"C:\Users\ADMIN\Downloads\data1 (2).csv")

# Extract the independent variable (X) and dependent variable (y)

X = data['x'].values

y = data['y'].values

# Calculate the mean of X and y

mean\_X = np.mean(X)

mean\_y = np.mean(y)

# Calculate the total number of samples

n = len(X)

# Using the equations provided in the slides

# Calculate the slope (beta1)

numerator = np.sum((X - mean\_X) \* (y - mean\_y))

denominator = np.sum((X - mean\_X) \*\* 2)

beta1 = numerator / denominator

# Calculate the intercept (beta0)

beta0 = mean\_y - beta1 \* mean\_X

# Predict the values of y using the model

y\_pred = beta0 + beta1 \* X

# Calculate Root Mean Squared Error (RMSE)

rmse = np.sqrt(np.sum((y\_pred - y) \*\* 2) / n)

# Calculate Mean Absolute Error (MAE)

mae = np.sum(np.abs(y\_pred - y)) / n

print("Manual Implementation:")

print("Intercept (beta0):", beta0)

print("Slope (beta1):", beta1)

print("RMSE:", rmse)

print("MAE:", mae)

# Using scikit-learn's LinearRegression model

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error

# Reshape X to a 2D array

X = X.reshape(-1, 1)

# Fit the model

model = LinearRegression()

model.fit(X, y)

# Get the coefficients

beta0\_sklearn = model.intercept\_

beta1\_sklearn = model.coef\_[0]

# Predict the values of y using the model

y\_pred\_sklearn = model.predict(X)

# Calculate RMSE using scikit-learn

rmse\_sklearn = np.sqrt(mean\_squared\_error(y, y\_pred\_sklearn))

# Calculate MAE using scikit-learn

mae\_sklearn = mean\_absolute\_error(y, y\_pred\_sklearn)

print("\nUsing scikit-learn:")

print("Intercept (beta0):", beta0\_sklearn)

print("Slope (beta1):", beta1\_sklearn)

print("RMSE:", rmse\_sklearn)

print("MAE:", mae\_sklearn)