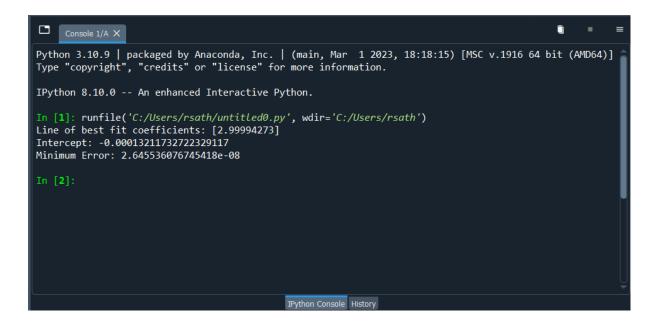
Ex 1	
	Linear Regression
27.12.2023	

Q1: 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.

CODE:

```
import numpy as np
from sklearn.linear model import LinearRegression
np.random.seed(42)
random array = np.random.rand(1500)
random_array = random_array.reshape(-1, 1)
X_train = random_array[:1000]
y_train = 3 * X_train.squeeze() + np.random.normal(0, 0.1,
size=1000)
best_fit_line = None
min_error = float('inf')
for _ in range(1000):
   indices = np.random.choice(1000, size=1000, replace=True)
```

```
X_subset = X_train[indices]
   y_subset = y_train[indices]
    model = LinearRegression()
    model.fit(X_subset, y_subset)
   X_test = random_array[1000:]
   y_test = 3 * X_test.squeeze()
    error = np.mean((model.predict(X test) - y test) ** 2)
   if error < min error:</pre>
       min error = error
        best_fit_line = model
print(f"Line of best fit coefficients: {best_fit_line.coef_}")
print(f"Intercept: {best fit line.intercept }")
print(f"Minimum Error: {min_error}")
X remaining = random array[1000:]
predicted_values = best_fit_line.predict(X_remaining)
```



Q2: Use the data1.csv to build a simple linear regression from scratch without using sklearn libraries and print the RMSE and mean absolute error values. Use both the equations available in the slides (in theory page) to build the model and compare the intercept and coefficient values.

CODE:

```
import csv
import math
import numpy as np
data = []
with open('data1.csv', 'r') as file:
    csv_reader = csv.reader(file)
    next(csv_reader)
    for row in csv_reader:
        data.append([float(row[0]), float(row[1])])

def calculate_coefficients_partial_derivative(data):
    x = [row[0] for row in data]
    y = [row[1] for row in data]

    x_mean = sum(x) / len(x)
    y_mean = sum(y) / len(y)
```

```
numerator = sum((x[i] * y[i] - y_mean * x[i])) for i in
range(len(data)))
    denominator = sum((x[i] ** 2 - x_mean * x[i])) for i in
range(len(data)))
   b1 = numerator / denominator
   b0 = y mean - b1 * x mean
    return b0, b1
def calculate_metrics(data, b0, b1):
   mse = 0
   mae = 0
   n = len(data)
   for i in range(n):
        x, y = data[i]
        y_pred = b0 + b1 * x
       mse += (y - y_pred) ** 2
       mae += abs(y - y pred)
   mse /= n
    rmse = math.sqrt(mse)
    mae /= n
    return rmse, mae
def calculate_coefficients_correlation(data):
   x = [row[0]] for row in data]
   y = [row[1] for row in data]
    x mean, y mean = np.mean(x), np.mean(y)
   r = np.corrcoef(x, y)[0, 1]
   Sx = np.std(x)
   Sy = np.std(y)
   b1 = r * Sy / Sx
   b0 = y_mean - b1 * x_mean
    return b0, b1
```

```
intercept partial, coefficient partial =
calculate coefficients partial derivative(data)
intercept correlation, coefficient correlation =
calculate coefficients correlation(data)
rmse partial, mae partial = calculate metrics(data,
intercept partial, coefficient partial)
rmse correlation, mae correlation = calculate metrics(data,
intercept correlation, coefficient correlation)
print("Using Partial Derivative Equations:")
print(f"Intercept (b0): {intercept partial}")
print(f"Coefficient (b1): {coefficient partial}")
print(f"Root Mean Squared Error (RMSE): {rmse partial}")
print(f"Mean Absolute Error (MAE): {mae_partial}")
print("\nUsing Correlation Coefficient and Standard Deviation:")
print(f"Intercept (b0): {intercept correlation}")
print(f"Coefficient (b1): {coefficient correlation}")
print(f"Root Mean Squared Error (RMSE): {rmse correlation}")
print(f"Mean Absolute Error (MAE): {mae_correlation}")
```

