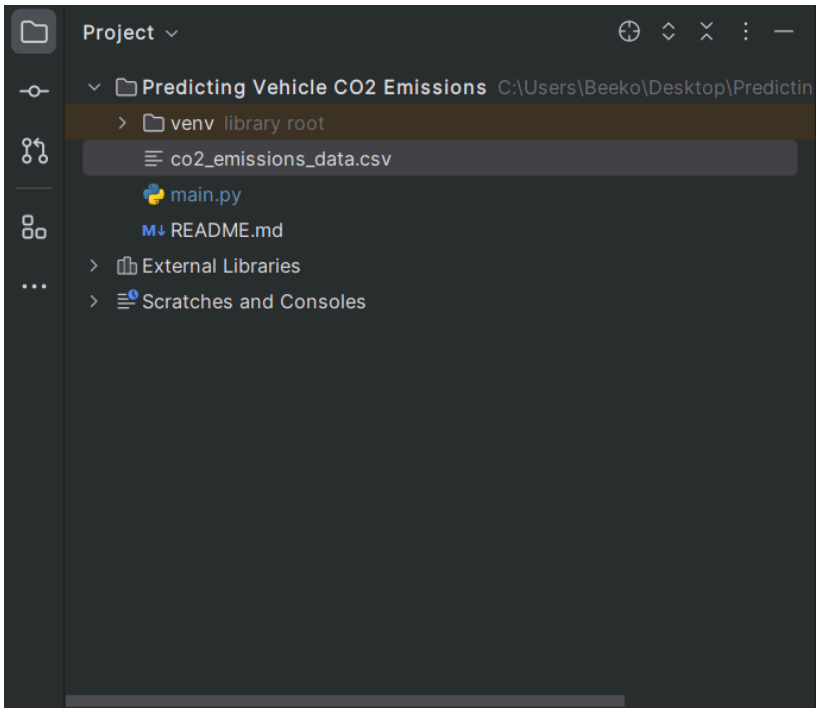
Assignment 1 Report

Students Name	Student ID
AbubakerElsiddig Omer Ali	20210733
Mohamed Ibrahim Ali	20180425
Jamal Bakr	
Mohammed Ali	

a. Loading the Dataset :



b. Data Analysis:

I. Check **whether** there are **missing values**

Answer → There are no missing values, here below the sum of empty values in each column

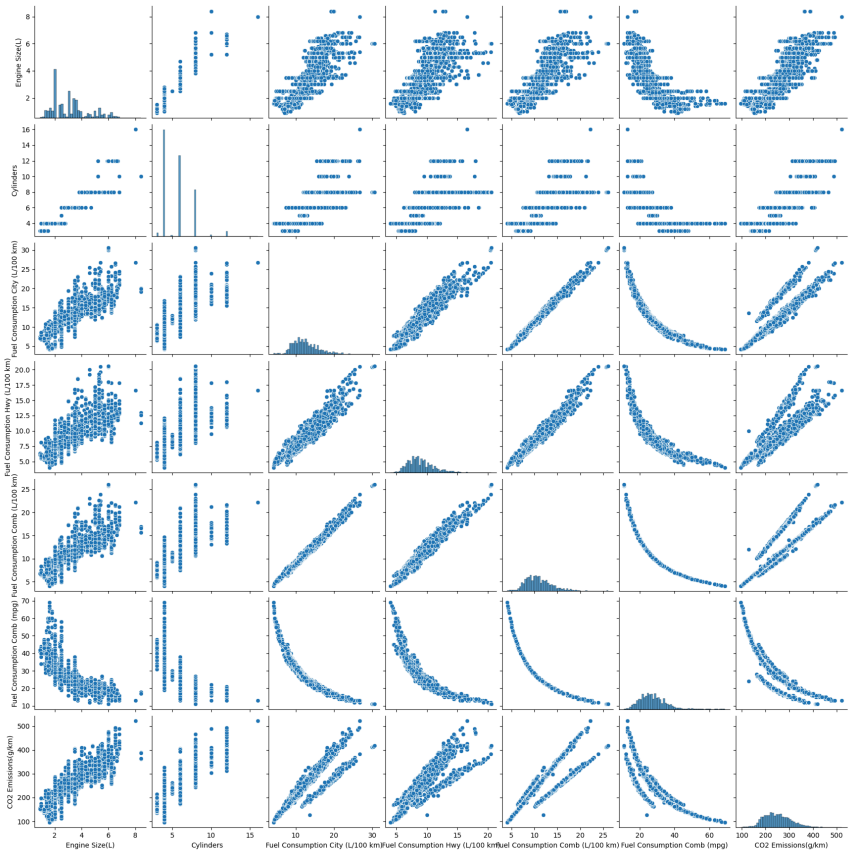
```
Make                                0
Model                               0
Vehicle Class                       0
Engine Size(L)                      0
Cylinders                           0
Transmission                        0
Fuel Type                           0
Fuel Consumption City (L/100 km)    0
Fuel Consumption Hwy (L/100 km)     0
Fuel Consumption Comb (L/100 km)    0
Fuel Consumption Comb (mpg)         0
CO2 Emissions(g/km)                 0
Emission Class                      0
dtype: int64
```

II. Check whether numeric features have the same scale

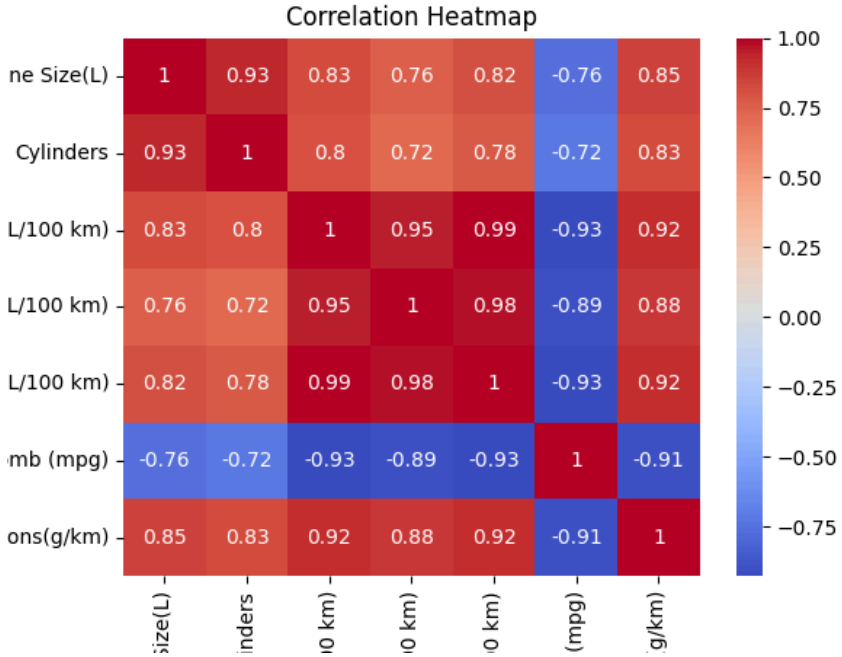
Answer → No they don't and they **need scaling**.

```
Range of each numeric feature:
Engine Size(L): 7.5
Cylinders: 13
Fuel Consumption City (L/100 km): 26.400000000000002
Fuel Consumption Hwy (L/100 km): 16.6
Fuel Consumption Comb (L/100 km): 22.0
Fuel Consumption Comb (mpg): 58
CO2 Emissions(g/km): 426

Process finished with exit code 0
```

III. **Visualize** a **pairplot** in which diagonal **subplots** are histograms



IV. **Visualize** a correlation heatmap between numeric columns



c).Preprocess the data such that:

I. the features and targets are separated

```
# Select features for linear regression
selected_features = ["Engine Size(L)", "Fuel Consumption Comb (L/100 km)", "Cylinders"]
X_train = features_train[selected_features].values
X_test = features_test[selected_features].values
y_train = target_train_reg.values
y_test = target_test_reg.values
```

II. categorical features and targets are encoded

```
Encoded Classes: ['HIGH', 'LOW', 'MODERATE', 'VERY LOW']
Example From The Encoded Targets: [2 0 2 0 0]
```

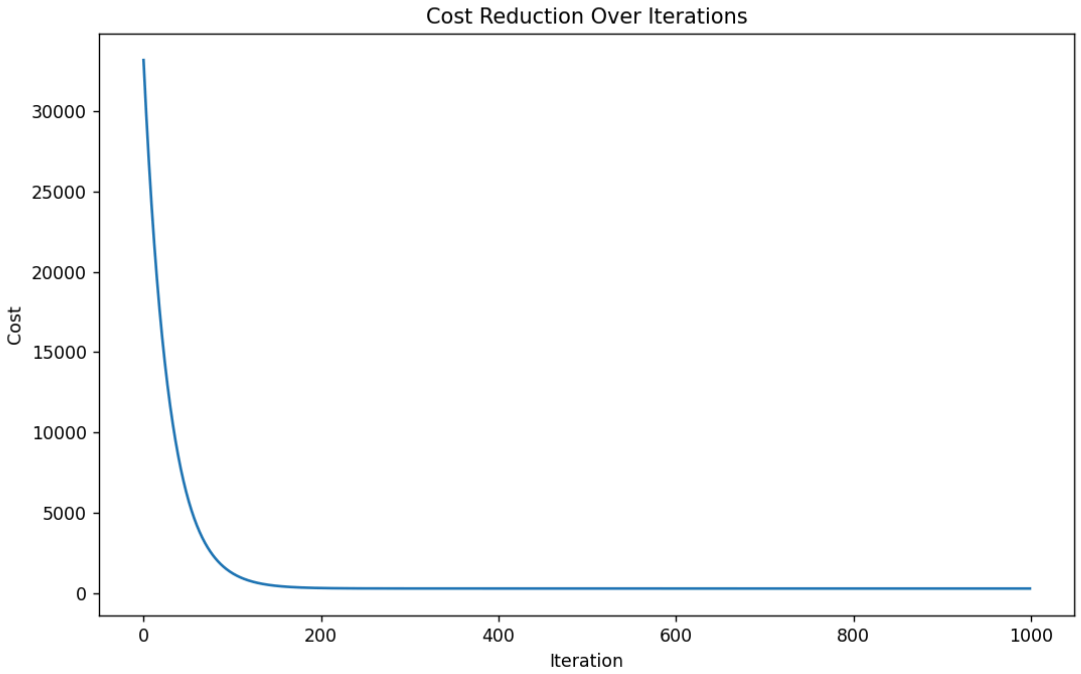
III. the data is shuffled and split into training and testing sets

```
# Split data
X_train_log, X_test_log, y_train_log, y_test_log = train_test_split(
    features_log, target_encoded, test_size=0.2, random_state=0
)
```

IV. numeric features are scaled

```
# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_log)
X_test_scaled = scaler.transform(X_test_log)
```

D). Implement linear regression using gradient descent from scratch to predict the CO2 emission amount.




```
# Select features for linear regression
selected_features = ["Engine Size(L)", "Fuel Consumption Comb (L/100 km)", "Cylinders"]
X_train = features_train[selected_features].values
X_test = features_test[selected_features].values
y_train = target_train_reg.values
y_test = target_test_reg.values
```

```
----------------------------------------
Linear Regression Results:
The Accuracy (R² Score) is:  0.8344458543680593
----------------------------------------
```

e)

```
data = pd.read_csv('co2_emissions_data.csv')
print(data.head())
```

[4] ✓ 0.0s

Actions...

	Make	Model	Vehicle Class	Engine Size(L)	Cylinders	Transmission
0	ACURA	ILX	COMPACT	2.0	4	AS5
1	ACURA	ILX	COMPACT	2.4	4	M6
2	ACURA	ILX HYBRID	COMPACT	1.5	4	AV7
3	ACURA	MDX 4WD	SUV - SMALL	3.5	6	AS6
4	ACURA	RDX AWD	SUV - SMALL	3.5	6	AS6

	Fuel Type	Fuel Consumption City (L/100 km)
0	Z	9.9
1	Z	11.2
2	Z	6.0
3	Z	12.7
4	Z	12.1

	Fuel Consumption Hwy (L/100 km)	Fuel Consumption Comb (L/100 km)
0	6.7	8.5
1	7.7	9.6
2	5.8	5.9
3	9.1	11.1
4	8.7	10.6

	Fuel Consumption Comb (mpg)	CO2 Emissions(g/km)	Emission Class
0	33	196	MODERATE
1	29	221	HIGH
2	48	136	MODERATE
3	25	255	HIGH
4	27	244	HIGH

```python
# Selecting two features
features = data[["Fuel Consumption Comb (L/100 km)", "Engine Size(L)"]]

# Selecting the target column (Emission class)
target = data["Emission Class"]
```

[5] ✓ 0.0s Python

+ Code + Markdown

```python
# Convert the categorical target values (Low, Moderate, High) into numerical values (0, 1, 2)
encoder = LabelEncoder()
target_encoded = encoder.fit_transform(target)

print("Encoded Classes:", list(encoder.classes_))
print("Example From The Encoded Targets:", target_encoded[:5])
```

[41] ✓ 0.0s Python

```
Encoded Classes: ['HIGH', 'LOW', 'MODERATE', 'VERY LOW']
Example From The Encoded Targets: [2 0 2 0 0]
```

```python
# Splitting data with 80% training, 20% test
X_train, X_test, y_train, y_test = train_test_split(features, target_encoded, test_size=0.2, random_state=0)

print("Training Features:", X_train.shape)
print("Testing Features:", X_test.shape)
```

[42] ✓ 0.0s Python

```
Training Features: (5908, 2)
Testing Features: (1477, 2)
```

```python
# Standardize the features to have mean 0 and standard deviation 1
scaler = StandardScaler()

# Fit the scaler on the training data and transform both train and test sets
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print("Scaled Features (Training):", X_train_scaled[:5])
```

[43] ✓ 0.0s Python

```
Scaled Features (Training): [[-0.71356506 -0.85513666]
 [ 2.10619149  1.58119868]
 [-0.64479051 -0.48599494]
 [ 0.52437684  0.91674359]
 [-0.09459411  0.17846015]]
```

```python
# Model Parameters
m, n = X_train_scaled.shape  # Number of training samples and features
theta = np.zeros(n)          # Start with weights of 0
learning_rate = 0.01         # Learning rate for gradient descent
iterations = 2000            # Number of training iterations
```

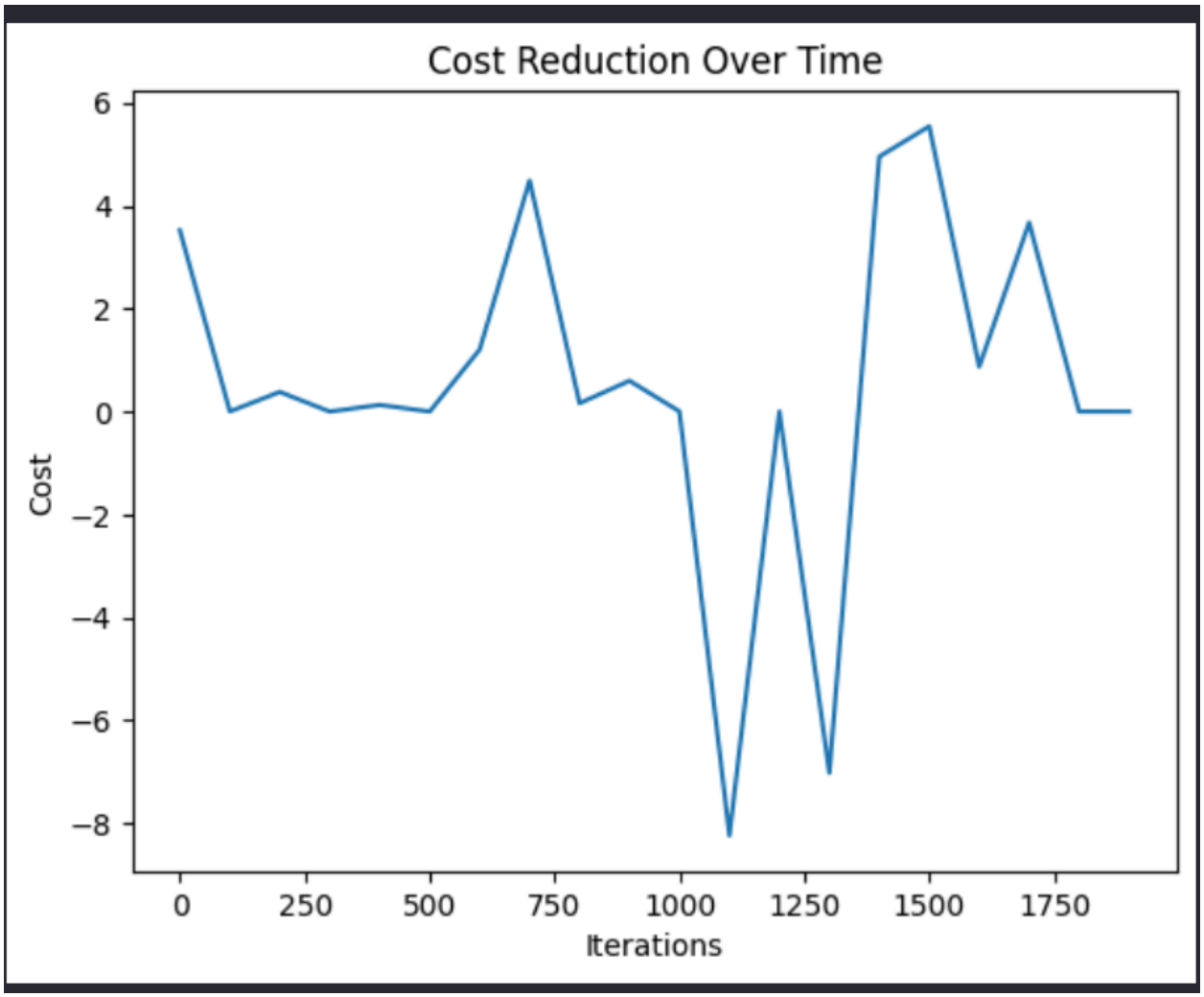
5] ✓ 0.0s Python

```python
# List to store cost
cost_history = []

# Perform training for the specified number of iterations
for iteration in range(iterations):

    random_index = np.random.randint(0, m)  # Randomly choose a sample index
    xi = X_train_scaled[random_index, :]    # Features for the chosen sample
    yi = y_train[random_index]              # True label for the chosen sample

    z = np.dot(xi, theta)          # Weighted sum
    prediction = sigmoid(z)        # Convert to probability

    error = prediction - yi        # Difference between prediction and true label
    gradient = error * xi          # Gradient for the weights
    theta -= learning_rate * gradient  # Update weights using the gradient


    if iteration % 100 == 0:
        cost = -yi * np.log(prediction) - (1 - yi) * np.log(1 - prediction)  # Binary cross-entropy loss
        cost_history.append(cost)
```

[47] ✓ 0.0s Python

```python
# Plotting the cost over time
plt.plot(range(0, iterations, 100), cost_history)
plt.xlabel("Iterations")
plt.ylabel("Cost")
plt.title("Cost Reduction Over Time")
plt.show()
```

[48] ✓ 0.1s Python




```python
# Using the trained weights to predict test set
z_test = np.dot(X_test_scaled, theta)  # Weighted sums for the test set
predicted_probabilities = sigmoid(z_test)

# Convert probabilities to class predictions using a threshold
threshold = 0.5  # Default threshold used in logistic regression
y_pred_class = predicted_probabilities >= threshold
```

✓ 0.0s Python

```python
# Accuracy
accuracy = np.mean(y_pred_class == y_test)
print("The Accuracy in this Logistic Regression is: ", accuracy)
```

[53] ✓ 0.0s Python

```
The Accuracy in this Logistic Regression is:  0.44617467840216657
```