ML Assignment 1

Name	ID
Ahmed Hagag	20216010
Abdulrahman Hijazy	20216056
Ahmed Bakry	20216012

Requirements:

a) Load the "co2_emissions_data.csv" dataset.

```
[2] data = pandas.read_csv("data/co2_emissions_data.csv")
```

b) Perform analysis on the dataset to:

i) check whether there are missing values

```
# i) Checking if there are any missing values in the data, we found out that there's none
print("     Missing data in each feature:")
print(data.isnull().sum())
```

```
     Missing data in each feature:
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
```

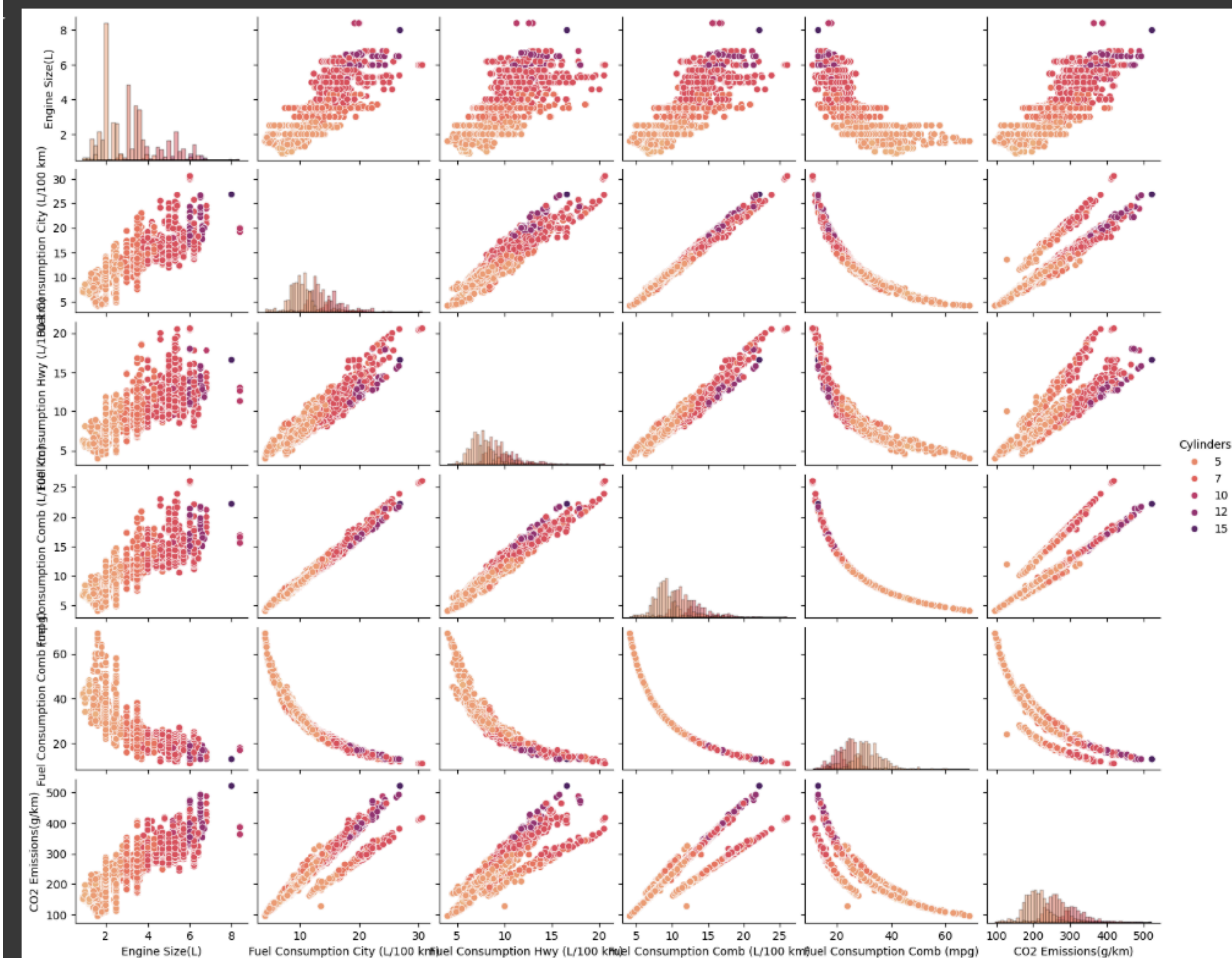
ii) check whether numeric features have the same scale

```
# ii) Checking if numeric values are scaled, after printing we found out they need sclaing.
numeric_data = data.select_dtypes(include=['number'])
print(numeric_data.describe())
```

```
       Engine Size(L)    Cylinders  Fuel Consumption City (L/100 km)  \
count     7385.000000  7385.000000                       7385.000000
mean         3.160068     5.615030                         12.556534
std          1.354170     1.828307                          3.500274
min          0.900000     3.000000                          4.200000
25%          2.000000     4.000000                         10.100000
50%          3.000000     6.000000                         12.100000
75%          3.700000     6.000000                         14.600000
max          8.400000    16.000000                         30.600000

       Fuel Consumption Hwy (L/100 km)  Fuel Consumption Comb (L/100 km)  \
count                      7385.000000                       7385.000000
mean                          9.041706                         10.975071
std                           2.224456                          2.892506
min                           4.000000                          4.100000
25%                           7.500000                          8.900000
50%                           8.700000                         10.600000
75%                          10.200000                         12.600000
max                          20.600000                         26.100000

       Fuel Consumption Comb (mpg)  CO2 Emissions(g/km)
count                  7385.000000          7385.000000
mean                     27.481652           250.584699
std                       7.231879            58.512679
min                      11.000000            96.000000
25%                      22.000000           208.000000
50%                      27.000000           246.000000
75%                      32.000000           288.000000
max                      69.000000           522.000000
```

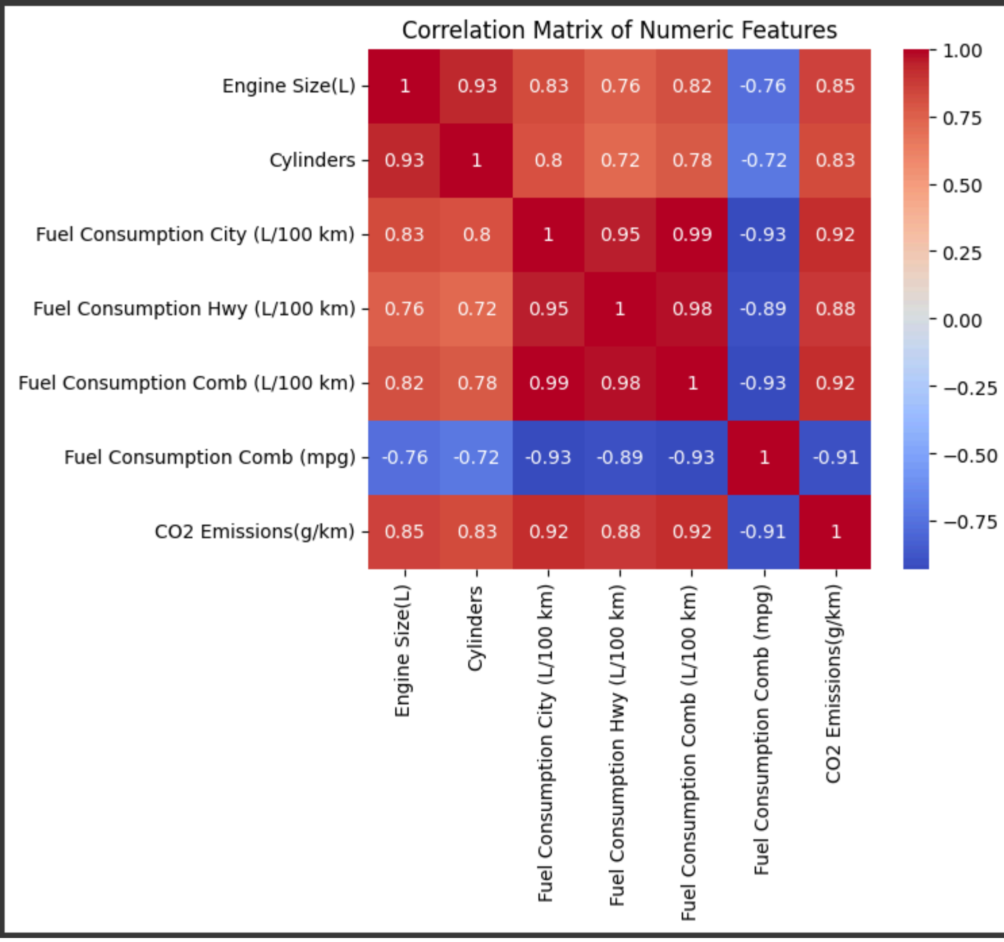
iii) visualize a pairplot in which diagonal subplots are histograms

```
# iii) visualizing a pairplot in which diagonal subplots are histograms
sns.pairplot(data, hue='Cylinders', diag_kind='hist', height=2, aspect=1.2, palette="flare")
plot.show()
```



iv) visualize a correlation heatmap between numeric columns

```
# iv) visualizing a correlation heatmap between numeric columns

plot.figure(figsize=(6, 5))
sns.heatmap(numeric_data.corr(), annot=True, cmap='coolwarm')
plot.title('Correlation Matrix of Numeric Features')
plot.show()
```



c) Preprocess the data such that:

i) the features and targets are separated

```
# Preprocessing the data:
# i) here separate the features and the targets

y_emission_class = data['Emission Class']
y_co2_emissions = data['CO2 Emissions(g/km)']
features = ["Engine Size(L)", "Fuel Consumption Comb (L/100 km)"]
X = data[features]
Y = data[['Emission Class', 'CO2 Emissions(g/km)']]


print("Features (X) shape:", X.shape)
print("Labels (Y) shape Emisssion Class:", y_emission_class.shape)
print("Labels (Y) shape CO2 Emission amount:", y_co2_emissions.shape)
```

```
Features (X) shape: (7385, 2)
Labels (Y) shape Emisssion Class: (7385,)
Labels (Y) shape CO2 Emission amount: (7385,)
```

ii) categorical features and targets are encoded

```
[10] # encoding the categorical data
     categorical_cols = X.select_dtypes(include=['object']).columns
     X = pandas.get_dummies(X, columns=categorical_cols)
     categorical_cols = Y.select_dtypes(include=['object']).columns
     y_emission_class = pandas.get_dummies(y_emission_class, columns=categorical_cols)
```

iii) the data is shuffled and split into training and testing sets

```
[8]  # Shuffling the data
     X, y_emission_class = shuffle(X, y_emission_class, random_state=42)

     # Splitting data into training and testing sets
     X_train, X_test, Y_train, Y_test = train_test_split(X, y_emission_class, test_size=0.2, random_state=42)
```

```
[9]  # Seeing the data shape after splitting
     print("X_train shape:", X_train.shape)
     print("X_test shape:", X_test.shape)
     print("Y_train shape:", Y_train.shape)
     print("Y_test shape:", Y_test.shape)
```

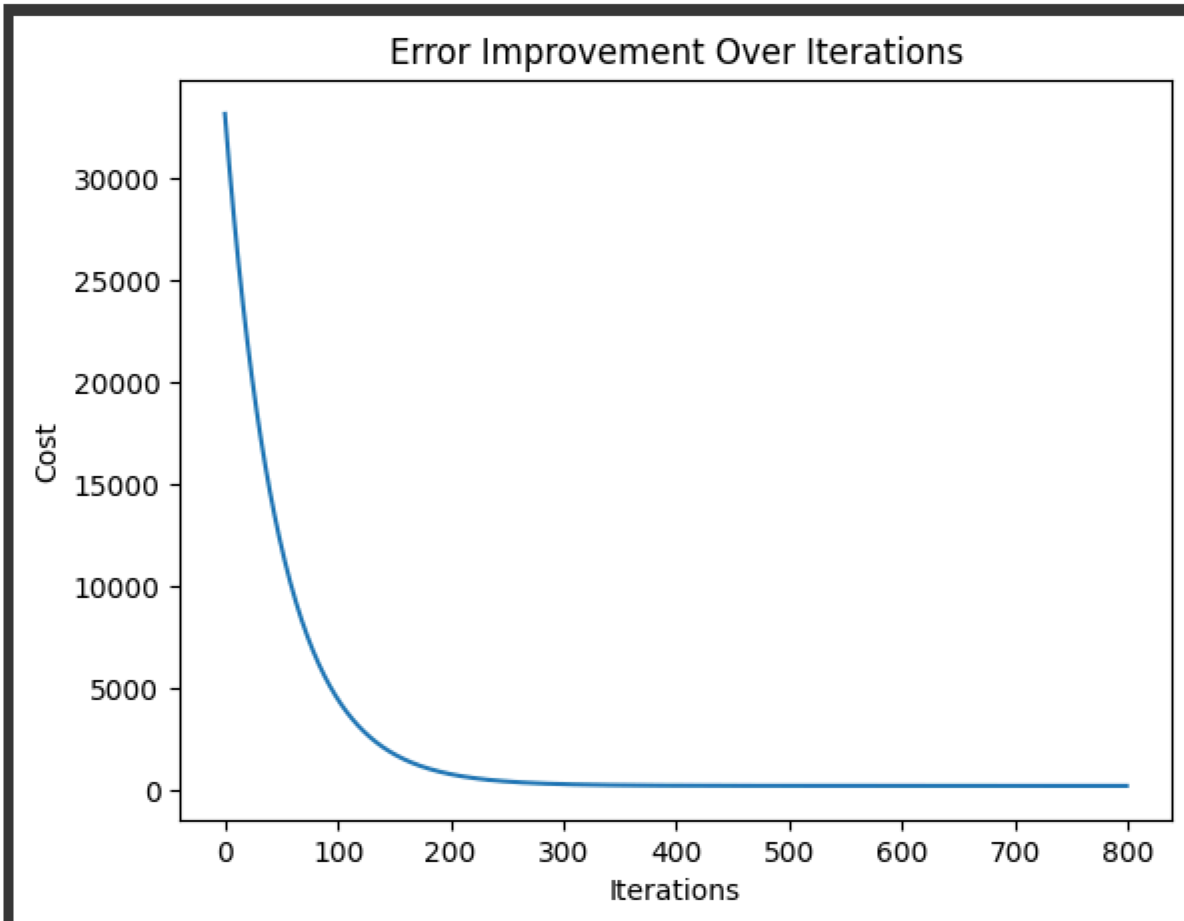
```
X_train shape: (5908, 2)
X_test shape: (1477, 2)
Y_train shape: (5908,)
Y_test shape: (1477,)
```

iv) numeric features are scaled

```
[11] # Scaling numerical features in the training set
     scaler = StandardScaler()
     numerical_cols = X_train.select_dtypes(include=['number']).columns
     X_train[numerical_cols] = scaler.fit_transform(X_train[numerical_cols])

     # Scaling numerical features in the test set using the same scaler
     X_test[numerical_cols] = scaler.transform(X_test[numerical_cols])
```

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

```
# Linear Regression:
X = data[['Engine Size(L)', 'Fuel Consumption Comb (L/100 km)']].values
y = data['CO2 Emissions(g/km)'].values

# Scaling numeriec features
X = (X - np.mean(X, axis=0)) / np.std(X, axis=0)

# Add a column of ones to X for the bias term
X = np.hstack([np.ones((X.shape[0], 1)), X])

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize parameters needed for the linear regression equation
theta = np.zeros(X_train.shape[1])
alpha = 0.01
iterations = 800
m = len(y_train)
cost_history = []

# Gradient descent Implementation
for i in range(iterations):
  # Predictions of the model (hypothesis)
  predictions = X_train.dot(theta)
  # Errors made by the model (predictions - actual)
  errors = predictions - y_train
  # Cost function (mean squared error) of the model
  cost = (1 / (2 * m)) * np.dot(errors.T, errors)
  # Save cost history for the visualization
  cost_history.append(cost)
  # Update the parameters using gradient descent for the next iteration
  theta -= (alpha / m) * X_train.T.dot(errors)

# Plotting the cost history illustrateing how the error of the hypothesis function improves with every iteration of gradient descent.
plot.plot(range(iterations), cost_history)
plot.xlabel("Iterations")
plot.ylabel("Cost")
plot.title("Error Improvement Over Iterations")
plot.show()

# Evaluate the model on the test set using R² score
y_test_pred = X_test.dot(theta)
print("\nR2 Score on Test Set:", r2_score(y_test, y_test_pred))
```




```
R2 Score on Test Set: 0.870745532352562
```

e) Fit a logistic regression model to the data to predict the emission class.

```
# Here we create the stochastic gradient classifier model and fitting the model
regressor = SGDClassifier()
regressor.fit(X_train, Y_train)

# Testing the model using the test set and get accuracy
y_pred = regressor.predict(X_test)
accuracy = accuracy_score(Y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.970886932972241
```