Cairo University
Faculty of Computers and Artificial Intelligence

Machine Learning

Assignment 1

Section IS S1&S2

Team Members

Name	ID
Salma Mamdoh Sabry	**20210162**
Roaa Talat Mohamed	**20210138**
Youssef Ehab Mohamed	**20210466**
Zeyad Ehab Maamoun	**20211043**
Youssef Mohamed Salah Eldin Anwar	**20210483**

Requirements:

Write a Python program in which you do the following:

a) Load the "co2_emissions_data.csv" dataset.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

plt.rcParams["figure.figsize"] = (10, 6)
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
df = pd.read_csv("co2_emissions_data.csv")
df.head()
```

Out[2]:

	Make	Model	Vehicle Class	Engine Size(L)	Cylinders	Transmission	Fuel Type	Fuel Consumption City (L/100 km)	Fuel Consumption Hwy (L/100 km)	Fuel Consumption Comb (L/100 km)	Fuel Consumption Comb (mpg)	CO2 Emissions(g/km)	Emission Class
0	ACURA	ILX	COMPACT	2.0	4	AS5	Z	9.9	6.7	8.5	33	196	MODERATE
1	ACURA	ILX	COMPACT	2.4	4	M6	Z	11.2	7.7	9.6	29	221	HIGH
2	ACURA	ILX HYBRID	COMPACT	1.5	4	AV7	Z	6.0	5.8	5.9	48	136	MODERATE
3	ACURA	MDX 4WD	SUV - SMALL	3.5	6	AS6	Z	12.7	9.1	11.1	25	255	HIGH
4	ACURA	RDX AWD	SUV - SMALL	3.5	6	AS6	Z	12.1	8.7	10.6	27	244	HIGH

In [3]:
```
df.shape
```

Out[3]: (7385, 13)

b) Perform analysis on the dataset to:

i) check whether there are missing values

In [6]:
```
print("\nMissing Values in Users Data:")
print(df.isnull().sum())
```

```

Missing Values in Users Data:
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
```

In [7]:
```
# Duplicated data
print("Duplicates in df :", df.duplicated().sum())
```

```
Duplicates in df : 1103
```

In [8]:
```
# drop all duplicate data
df = df.drop_duplicates()
df.shape
```

Out[8]: (6282, 13)

In [9]:
```
print(df.columns)
```

```
Index(['Make', 'Model', 'Vehicle Class', 'Engine Size(L)', 'Cylinders',
       'Transmission', 'Fuel Type', 'Fuel Consumption City (L/100 km)',
       'Fuel Consumption Hwy (L/100 km)', 'Fuel Consumption Comb (L/100 km)',
       'Fuel Consumption Comb (mpg)', 'CO2 Emissions(g/km)', 'Emission Class'],
      dtype='object')
```

The dataset was checked for missing values using the .isnull().sum() method. The results indicate that all columns have zero missing values, confirming the dataset is complete and does not require imputation or handling of missing data.

ii) check whether numeric features have the same scale

In [10]:
```
# Basic statistics summary of Numerical features

df.describe().T
```

Out[10]:

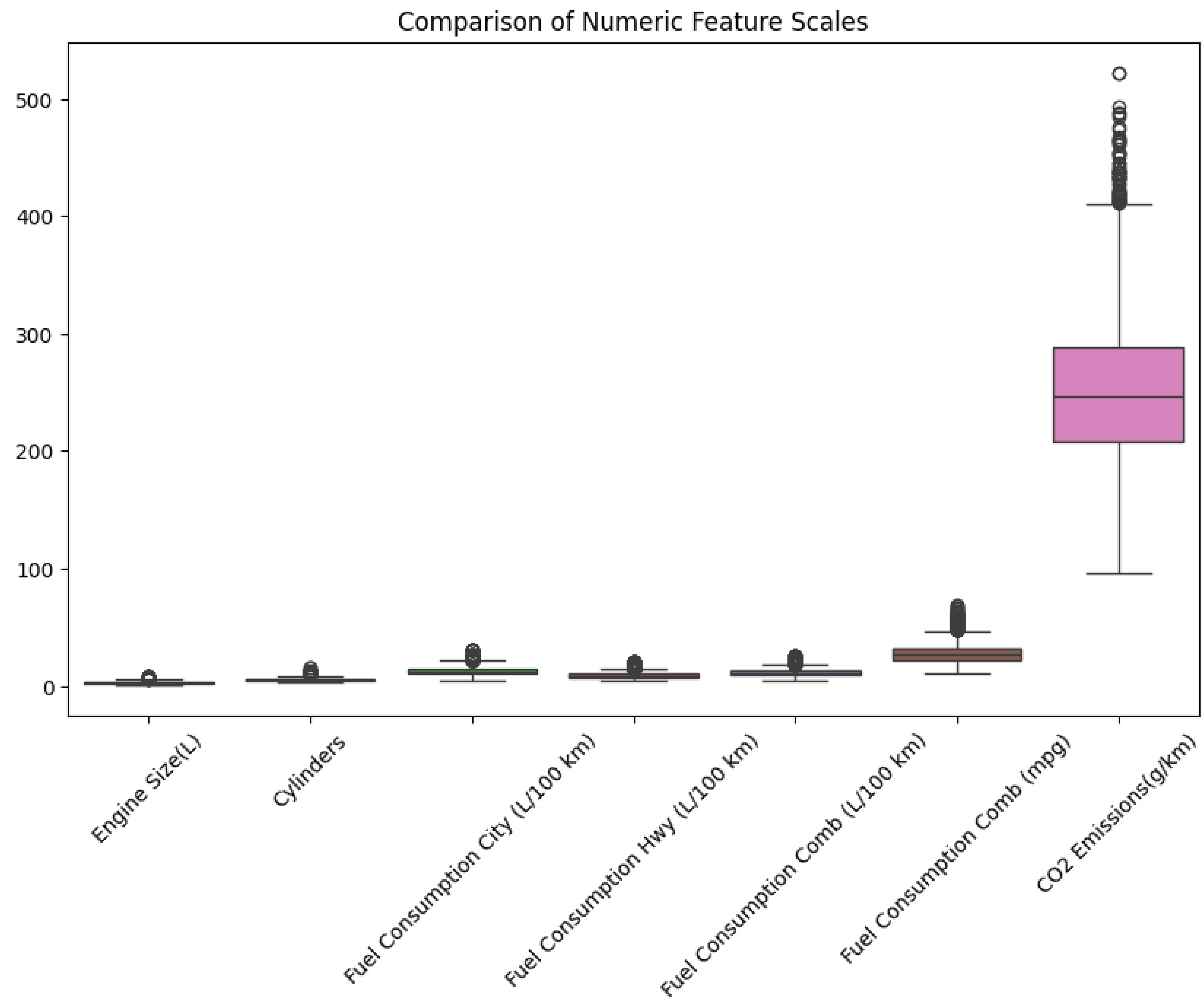
	count	mean	std	min	25%	50%	75%	max
Engine Size(L)	6282.0	3.161812	1.365201	0.9	2.0	3.0	3.7	8.4
Cylinders	6282.0	5.618911	1.846250	3.0	4.0	6.0	6.0	16.0
Fuel Consumption City (L/100 km)	6282.0	12.610220	3.553066	4.2	10.1	12.1	14.7	30.6
Fuel Consumption Hwy (L/100 km)	6282.0	9.070583	2.278884	4.0	7.5	8.7	10.3	20.6
Fuel Consumption Comb (L/100 km)	6282.0	11.017876	2.946876	4.1	8.9	10.6	12.7	26.1
Fuel Consumption Comb (mpg)	6282.0	27.411016	7.245318	11.0	22.0	27.0	32.0	69.0
CO2 Emissions(g/km)	6282.0	251.157752	59.290426	96.0	208.0	246.0	289.0	522.0

In [11]:
```
# Basic statistics summary of Object features

df.describe(include= 'object').T
```

Out[11]:

	count	unique	top	freq
Make	6282	42	FORD	577
Model	6282	2053	F-150 FFV	32
Vehicle Class	6282	16	SUV - SMALL	1006
Transmission	6282	27	AS6	1139
Fuel Type	6282	5	X	3039
Emission Class	6282	4	HIGH	4953

In [12]:
```
# Boxplot to visualize distributions and scales
plt.figure(figsize=(10, 6))
sns.boxplot(data=df.select_dtypes(include='number'))
plt.xticks(rotation=45)
plt.title("Comparison of Numeric Feature Scales")
plt.show()
```



Check whether numeric features have the same scale

Metric	Mean	Min	Max
Engine Size (L)	3.161812	0.9	8.4
Cylinders	5.618911	3.0	16.0
Fuel Consumption City (L/100 km)	12.610220	4.2	30.6
Fuel Consumption Hwy (L/100 km)	9.070583	4.0	20.6
Fuel Consumption Comb (L/100 km)	11.017876	4.1	26.1
Fuel Consumption Comb (mpg)	27.411016	11.0	69.0
CO2 Emissions (g/km)	251.157752	96.0	522.0

The numeric features do not appear to be on the same scale. Here's why:

The features have different ranges, means, and standard deviations. This confirms that they are not on the same scale, which could affect certain analyses and models especially models sensitive to feature scaling like linear regression.

iii) visualize a pairplot in which diagonal subplots are histograms

Pairplot Analysis:
A pairplot was generated to analyze the relationships between key numerical features in the dataset. Each subplot displays a scatterplot of the interactions between two variables, while the diagonal subplots present histograms showing the distribution of individual features.

- **Key Observations:**

    1. **Engine Size (L), Cylinders, and CO2 Emissions:** A strong positive correlation is visible, indicating that larger engines and higher cylinder counts are associated with greater CO2 emissions.

    2. **Fuel Consumption Metrics:** Significant linear relationships are observed between city, highway, and combined fuel consumption, which is expected since they measure related aspects of vehicle efficiency.

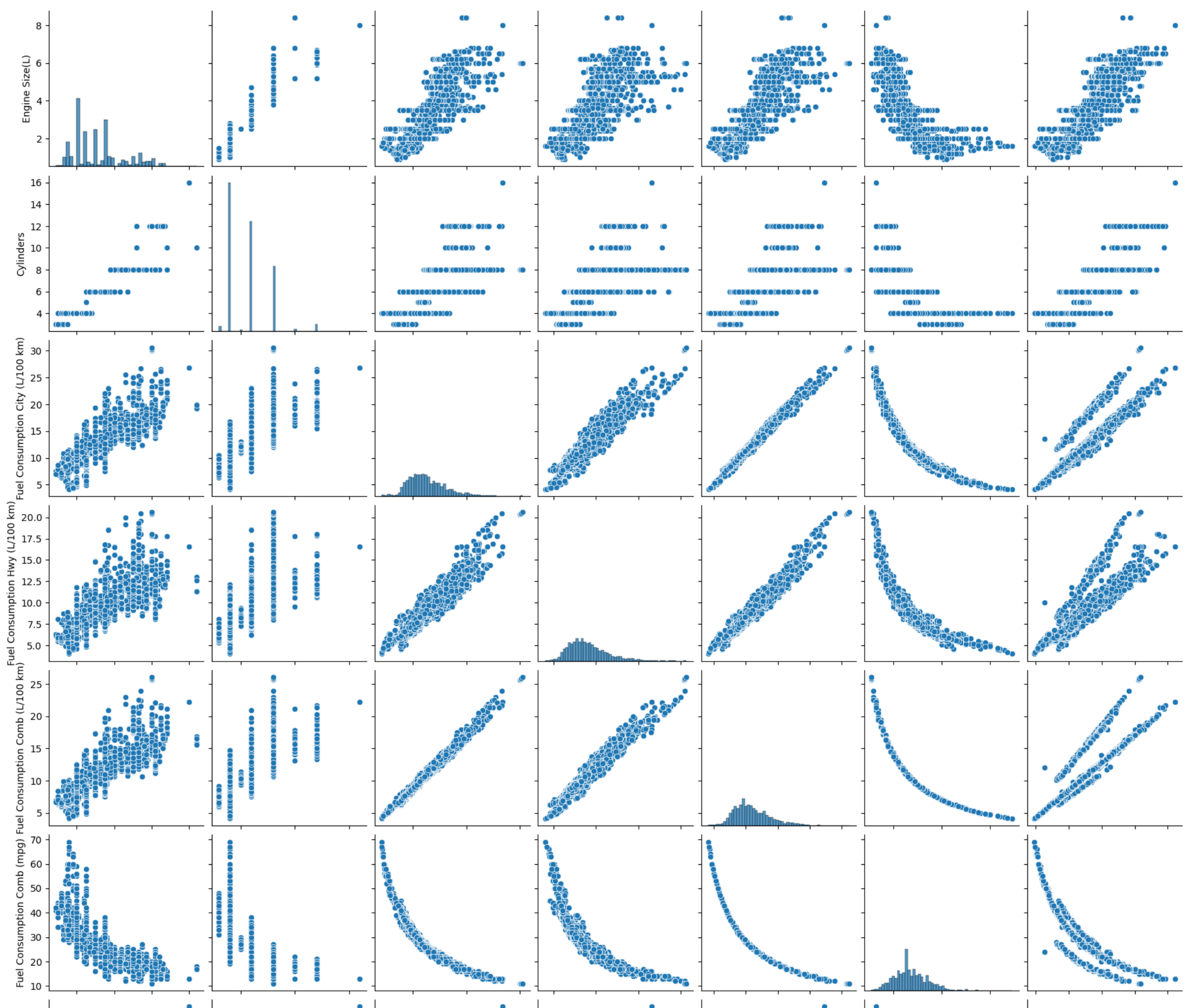
    3. **CO2 Emissions and Fuel Efficiency:** CO2 emissions show an inverse relationship with combined fuel consumption measured in mpg, as higher mpg implies lower emissions.

    4. **Histograms:** The diagonal plots reveal the distribution of individual features, such as engine size having a right-skewed distribution and fuel consumption values being tightly clustered.

Visualize a pairplot in which diagonal subplots are histograms

Each subplot shows the relationship between two features in the dataset, allowing us to see trends, patterns, and correlations across the different pairs of features.

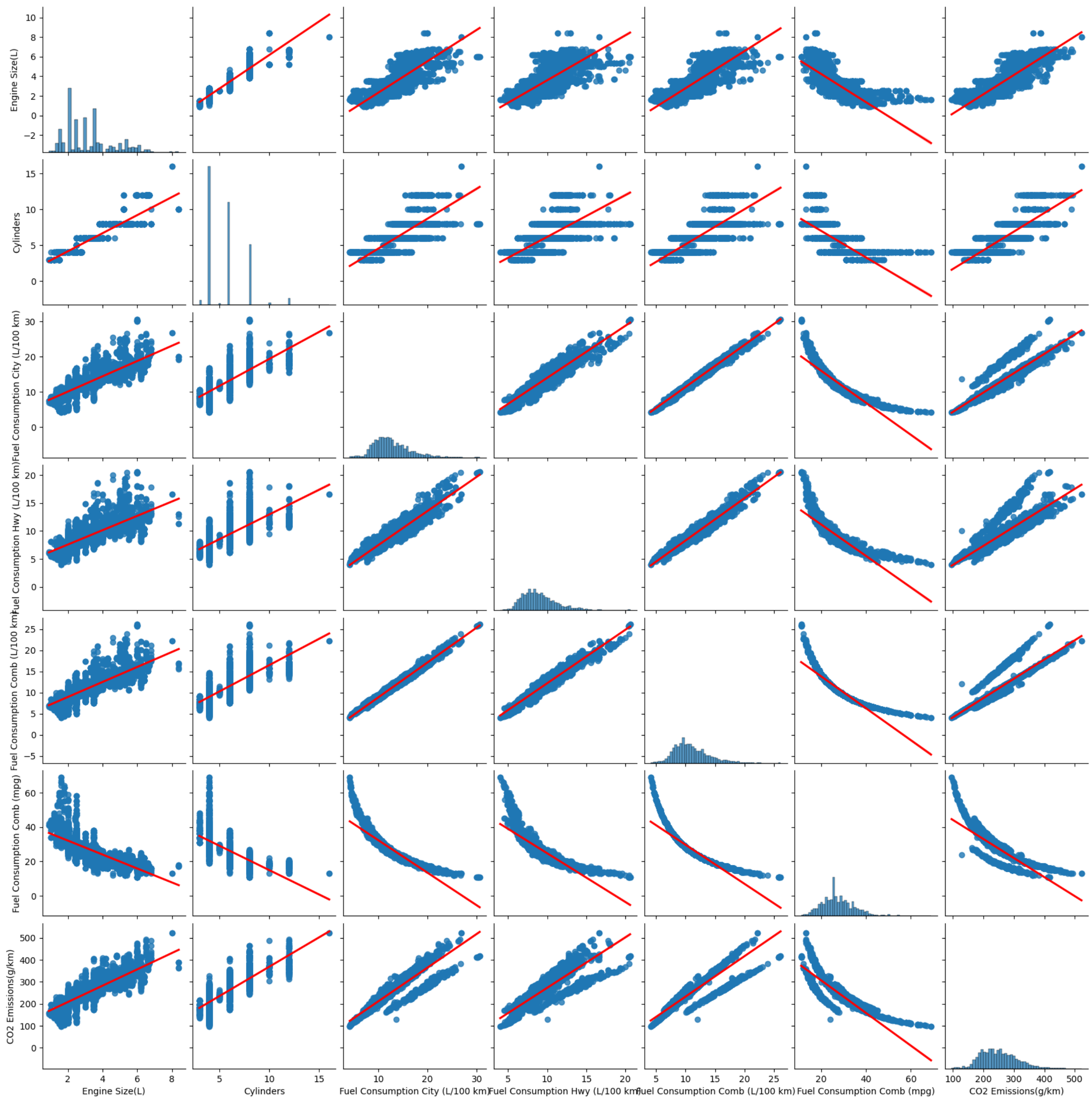
In [15]:
```
sns.pairplot(df, diag_kind="hist")

plt.show()
```


Engine Size(L)
Cylinders
Fuel Consumption City (L/100 km)
Fuel Consumption Hwy (L/100 km)
Fuel Consumption Comb (L/100 km)
Fuel Consumption Comb (mpg)
CO2 Emissions(g/km)

In [16]:

```python
# Pairplot for the dataframe

sns.pairplot(df,
             kind="reg",     # scatter plots with regression lines
             diag_kind="hist",
             plot_kws={"line_kws": {"color": "red"}}
            )
```

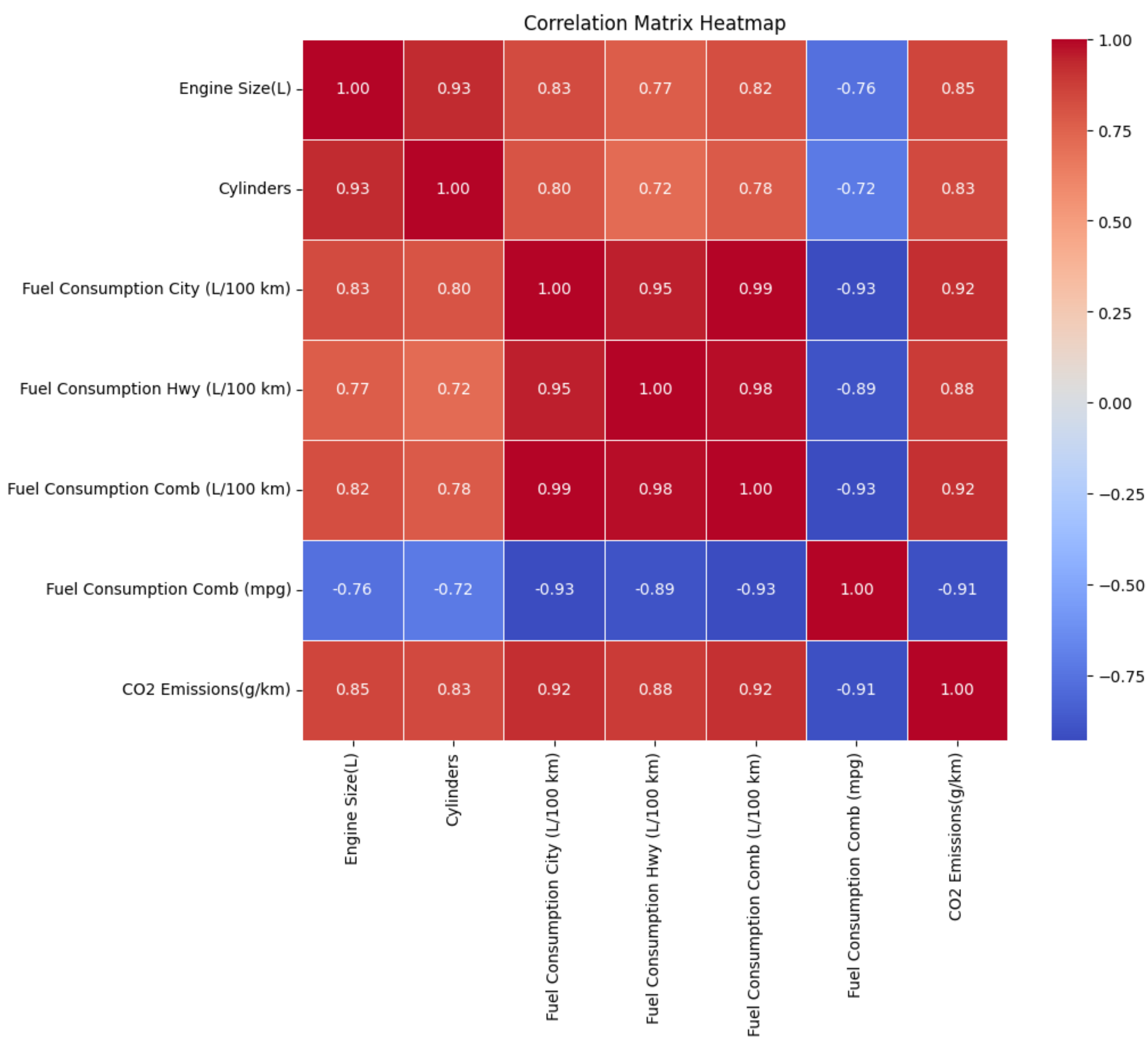
Out[16]: <seaborn.axisgrid.PairGrid at 0x143929ba5d0>



iv) visualize a correlation heatmap between numeric columns

visualize a correlation heatmap between numeric columns

In [17]:
```python
# Select only numeric columns
numeric_features = df.select_dtypes(include=['number'])

# Calculate the correlation matrix

correlation_matrix = numeric_features.corr()

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)

plt.title('Correlation Matrix Heatmap')
plt.show()
```



This is a **correlation matrix heatmap**, which visualizes the correlation coefficients between pairs of numerical features in your dataset. The values range from -1 to 1:

- **1** indicates a perfect positive correlation.

- **-1** indicates a perfect negative correlation.
- **0** indicates no correlation.

Key Insights

1. **Strong Positive Correlations**:
    - **Fuel Consumption Comb (L/100 km)** has a strong positive correlation with **Fuel Consumption City (L/100 km)** (0.99) and **Fuel Consumption Hwy (L/100 km)** (0.98). This indicates that combined fuel consumption is closely related to both city and highway consumption.
    - **Engine Size (L)** has a strong positive correlation with **Cylinders** (0.93), meaning that vehicles with larger engines tend to have more cylinders.
    - **CO_2 Emissions (g/km)** has a high positive correlation with **Fuel Consumption Comb (L/100 km)** (0.92), **Fuel Consumption City (L/100 km)** (0.92), and **Fuel Consumption Hwy (L/100 km)** (0.88). This suggests that higher fuel consumption is associated with higher CO_2 emissions.
2. **Strong Negative Correlations**:
    - **Fuel Consumption Comb (mpg)** has strong negative correlations with **Fuel Consumption City (L/100 km)** (-0.93), **Fuel Consumption Hwy (L/100 km)** (-0.89), and **CO_2 Emissions (g/km)** (-0.91). This indicates that higher miles per gallon (better fuel efficiency) is associated with lower fuel consumption and lower CO_2 emissions.
3. **Moderate Positive Correlations**:
    - **Engine Size (L)** and **CO_2 Emissions (g/km)** (0.85): Larger engine sizes are moderately associated with higher CO_2 emissions.
    - **Engine Size (L)** and **Fuel Consumption Comb (L/100 km)** (0.82): Larger engines tend to have higher combined fuel consumption.

Summary of Insights

- **Fuel efficiency (mpg)** is inversely related to fuel consumption and CO_2 emissions, **confirming that more fuel-efficient vehicles emit less CO_2**.
- **Engine size** and **cylinders** are directly linked; larger engines tend to have more cylinders and higher fuel consumption.

- **Fuel consumption metrics** (city, highway, combined) are highly correlated with each other and with CO_2 emissions, indicating that higher fuel consumption generally leads to higher emissions.

These insights suggest that **reducing fuel consumption** (improving fuel efficiency) could be a key strategy for reducing CO_2 emissions in vehicles.

Preprocess the data such that:

i) the features and targets are separated

Seperate Features and Targets

```python
#Sepearating the features and the targets
X = df.drop(columns=['CO2 Emissions(g/km)', 'Emission Class'])
y = df[['CO2 Emissions(g/km)', 'Emission Class']]

print("Features : \n")
print(X.head())
print("\n Targets :")
print(y.head())
```

✓ 0.0s Python

Output

```
Features :

    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)
0                           33
1                           29
2                           48
3                           25
4                           27
```

```
 Targets :
   CO2 Emissions(g/km) Emission Class
0                  196       MODERATE
1                  221           HIGH
2                  136       MODERATE
3                  255           HIGH
4                  244           HIGH
```

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

The data is shuffled and split into training and testing sets

```
#Training and Testing sets

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


print(df.shape)
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)
```

71] ✓ 0.0s

```
(6282, 13)
X_train shape: (5025, 11)
X_test shape: (1257, 11)
y_train shape: (5025, 2)
y_test shape: (1257, 2)
```

iii) categorical features and targets are encoded

```
from sklearn.preprocessing import LabelEncoder
X_train_encoded = pd.get_dummies(X_train, drop_first=True)
X_test_encoded = pd.get_dummies(X_test, drop_first=True)

X_train_encoded, X_test_encoded = X_train_encoded.align(X_test_encoded, join='left', axis=1, fill_value=0)

label_encoder = LabelEncoder()

y_train['Emission Class'] = label_encoder.fit_transform(y_train['Emission Class'])
y_test['Emission Class'] = label_encoder.transform(y_test['Emission Class'])

print("\nEncoded Training Features (X_train):")
print(X_train_encoded.head())
print("\nEncoded Test Features (X_test):")
print(X_test_encoded.head())
print("\nEncoded Training Target (y_train):")
print(y_train.head())
print("\nEncoded Test Target (y_test):")
print(y_test.head())
```

Output

```
Encoded Training Features (X_train):
      Engine Size(L)  Cylinders  Fuel Consumption City (L/100 km)  \
6622             2.0          4                              10.0
106              4.4          8                              15.0
1503             5.0          8                              14.9
3144             2.7          6                              10.7
1528             6.0          8                              22.0

      Fuel Consumption Hwy (L/100 km)  Fuel Consumption Comb (L/100 km)  \
6622                              8.2                               9.2
106                               9.8                              12.7
1503                              9.5                              12.4
3144                              7.4                               9.2
1528                             14.9                              18.8

      Fuel Consumption Comb (mpg)  Make_ALFA ROMEO  Make_ASTON MARTIN  \
6622                           31            False              False
106                            22            False              False
1503                           23            False              False
3144                           31            False              False
1528                           15            False              False

      Make_AUDI  Make_BENTLEY  ...  Transmission_AV10  Transmission_AV6  \
6622      False         False  ...              False             False
106       False         False  ...              False             False
1503      False         False  ...              False             False
3144      False         False  ...              False             False
1528      False         False  ...              False             False
```

```
      Transmission_AV7  Transmission_AV8  Transmission_M5  Transmission_M6  \
6622             False             False            False            False
106              False             False            False            False
1503             False             False            False            False
3144             False             False            False            False
1528             False             False            False            False

      Transmission_M7  Fuel Type_E  Fuel Type_X  Fuel Type_Z
6622            False        False        False         True
106             False        False        False         True
1503            False        False         True        False
3144            False        False        False         True
1528            False        False         True        False

[5 rows x 1982 columns]

Encoded Test Features (X_test):
      Engine Size(L)  Cylinders  Fuel Consumption City (L/100 km)  \
3003             3.0          6                              13.6
5970             3.0          6                              11.9
2394             3.6          6                              14.8
6020             3.6          6                              12.9
3416             3.0          6                              11.8

      Fuel Consumption Hwy (L/100 km)  Fuel Consumption Comb (L/100 km)  \
3003                             10.0                              12.0
5970                              8.5                              10.4
2394                             10.4                              12.8
6020                             10.2                              11.7
3416                              8.9                              10.5
```

```
     Fuel Consumption Comb (mpg)  Make_ALFA ROMEO  Make_ASTON MARTIN  \
3003                          24            False              False
5970                          27            False              False
2394                          22            False              False
6020                          24            False              False
3416                          27            False              False

     Make_AUDI  Make_BENTLEY  ...  Transmission_AV10  Transmission_AV6  \
3003     False         False  ...                  0             False
5970     False         False  ...                  0             False
2394     False         False  ...                  0             False
6020     False         False  ...                  0             False
3416     False         False  ...                  0             False

     Transmission_AV7  Transmission_AV8  Transmission_M5  Transmission_M6  \
3003            False             False            False            False
5970            False             False            False            False
2394            False             False            False            False
6020            False             False            False            False
3416            False             False            False            False

     Transmission_M7  Fuel Type_E  Fuel Type_X  Fuel Type_Z
3003           False        False        False        False
5970           False        False        False         True
2394           False        False         True        False
6020           False        False         True        False
3416           False        False        False         True
```

```
Encoded Training Target (y_train):
      CO2 Emissions(g/km)  Emission Class
6622                  215               0
106                   292               0
1503                  285               0
3144                  217               0
1528                  432               0

Encoded Test Target (y_test):
      CO2 Emissions(g/km)  Emission Class
3003                  322               0
5970                  242               0
2394                  300               0
6020                  275               0
3416                  245               0
```

iv) numeric features are scaled

```
from sklearn.preprocessing import StandardScaler

numeric_features = X.select_dtypes(include=['float64', 'int64']).columns


scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train[numeric_features])
X_test_scaled = scaler.transform(X_test[numeric_features])

X_train[numeric_features] = X_train_scaled
X_test[numeric_features] = X_test_scaled

print("Scaled Training Data:\n", X_train_scaled)
print("\nScaled Test Data:\n", X_test_scaled)
```

Output

```
0.0s

Scaled Training Data:
 [[-0.84946717 -0.87380374 -0.72924759 -0.37876225 -0.61200082  0.49144898]
 [ 0.91379859  1.30528499  0.67715608  0.32219726  0.57438064 -0.7518122 ]
 [ 1.35461502  1.30528499  0.649028    0.19076735  0.4726908  -0.61367207]
 ...
 [-0.84946717 -0.87380374 -0.78550374 -0.729242   -0.78148388  0.62958911]
 [-0.84946717 -0.87380374 -1.06678447 -0.90448188 -1.01876018  1.04400951]
 [-0.48212013 -0.87380374 -0.86988796 -0.729242   -0.8153805   0.76772925]]

Scaled Test Data:
 [[-0.1147731   0.21574063  0.28336305  0.4098172   0.33710435 -0.47553194]
 [-0.1147731   0.21574063 -0.1948142  -0.24733234 -0.20524146 -0.06111154]
 [ 0.32604334  0.21574063  0.62089993  0.58505707  0.60827725 -0.7518122 ]
 ...
 [ 1.20767621  1.30528499  0.78966837  0.32219726  0.64217386 -0.7518122 ]
 [-0.99640598 -0.87380374 -0.84175988 -1.03591178 -0.91707034  0.90586938]
 [-0.84946717 -0.87380374 -0.89801603 -0.86067191 -0.91707034  0.90586938]]
```

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

Based on the correlation heatmap, select two features to be the independent variables of your model.

```python
# Select the two features based on the correlation heatmap
X_train_selected = X_train_encoded[['Fuel Consumption Comb (L/100 km)', 'Engine Size(L)']].values
X_test_selected = X_test_encoded[['Fuel Consumption Comb (L/100 km)', 'Engine Size(L)']].values
```
✓ 0.0s

```python
# Add a bias column (intercept) for each feature
X_train_selected = np.c_[np.ones(X_train_selected.shape[0]), X_train_selected]
X_test_selected = np.c_[np.ones(X_test_selected.shape[0]), X_test_selected]
```
✓ 0.0s Python

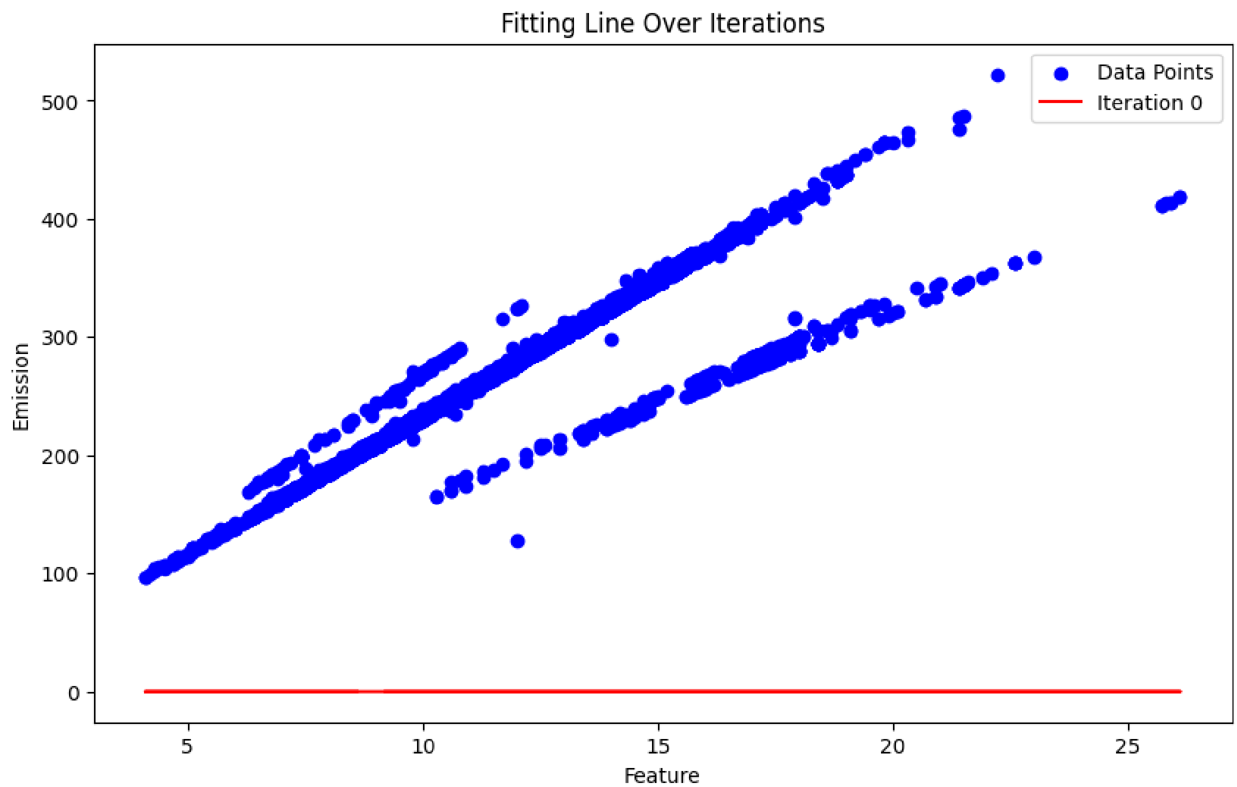
```python
# Function to compute the hypothesis (prediction)
def hypothesis(X, theta):
    return np.dot(X, theta) # The np.dot(X, theta) operation calculates the dot product between the feature matrix X and the parameter vector theta "which is
```
✓ 0.0s Python

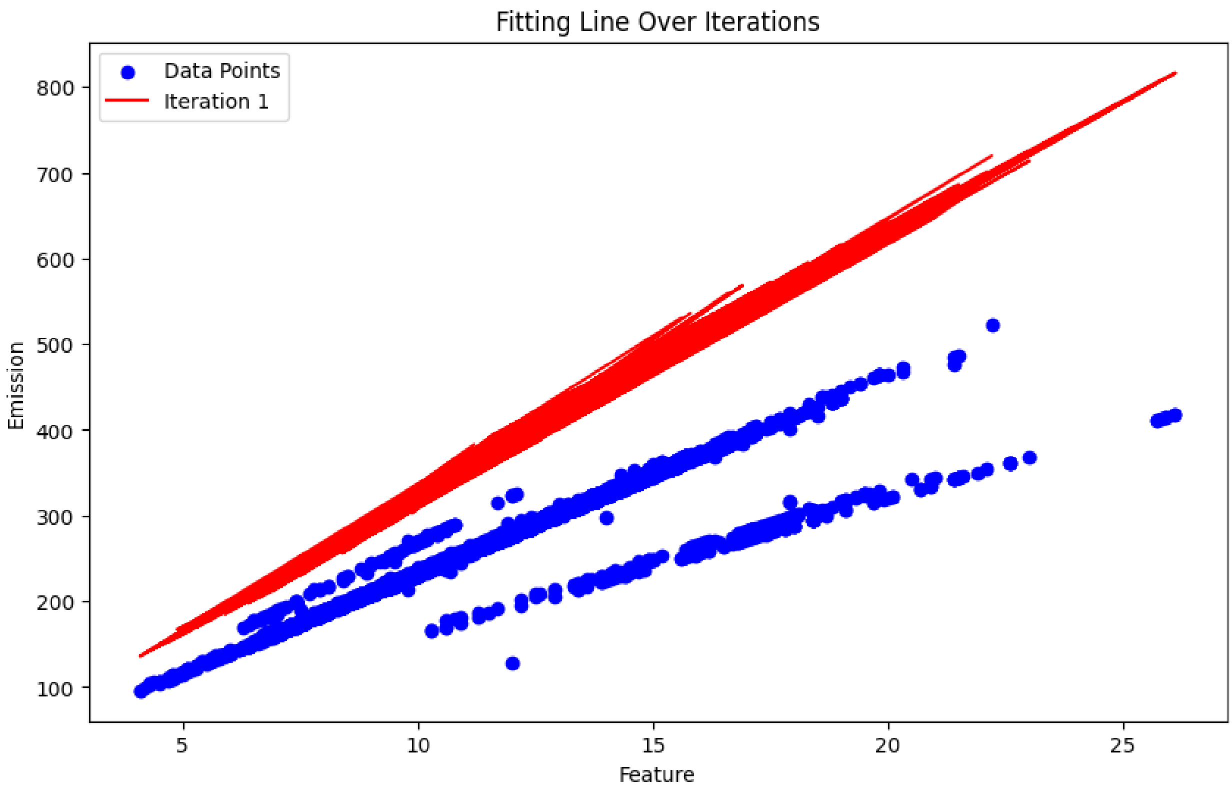
```python
# Cost function (Mean Squared Error)
def cost_function(X, y, theta):
    m = len(y)  # Number of training examples
    predictions = hypothesis(X, theta)
    cost = (1/(2*m)) * np.sum(np.square(predictions - y))  # MSE formula
    return cost
```
0.0s

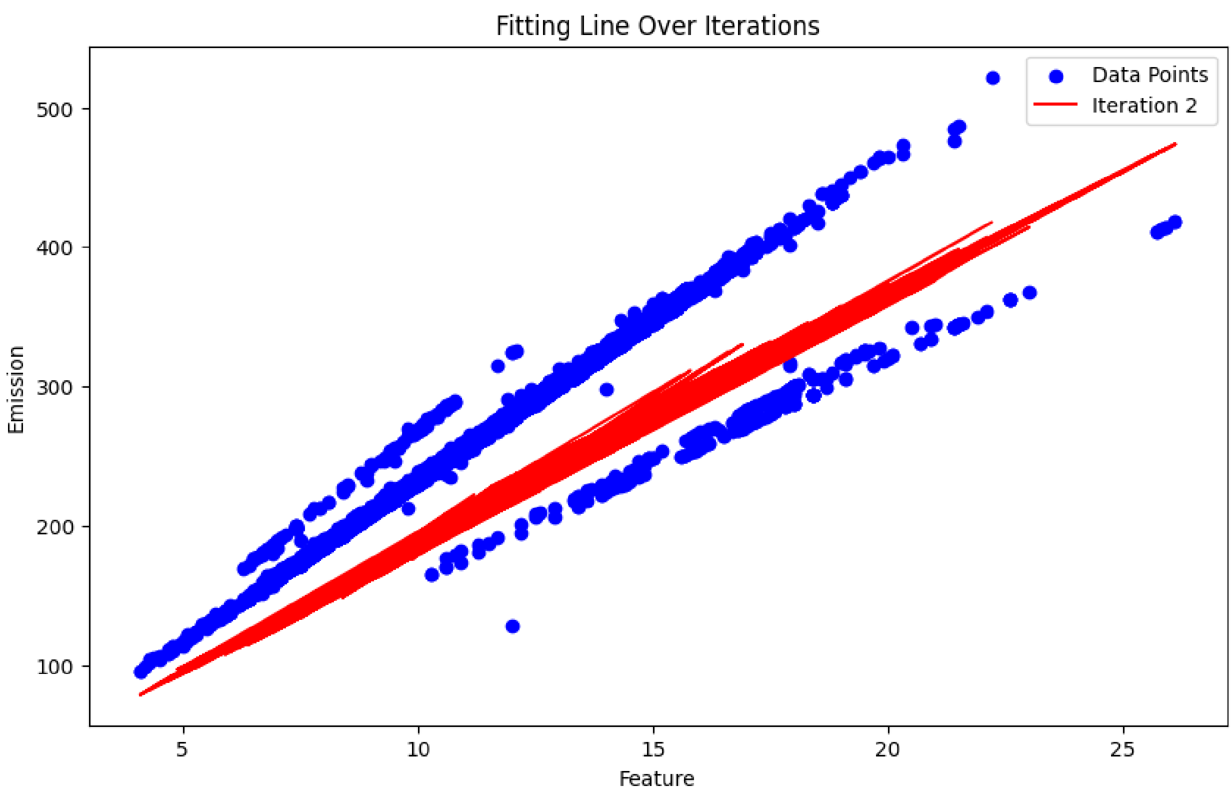
```python
# Gradient Descent function
def gradient_descent(X, y, theta, learning_rate, iterations):
    m = len(y)
    cost_history = []  # Store the cost at each iteration

    # Set up the plot for the iterations
    plt.figure(figsize=(10, 6))

    for i in range(iterations):
        predictions = hypothesis(X, theta)
        errors = predictions - y
        gradient = (1 / m) * np.dot(X.T, errors)
        theta -= learning_rate * gradient
        cost_history.append(cost_function(X, y, theta))

        # Plotting the line fit after every certain number of iterations
        if i < 10 or i == 2000:  # Adjust the frequency of the plot updates
            plt.scatter(X[:, 1], y, color='blue', label='Data Points')
            plt.plot(X[:, 1], predictions, color='red', label=f'Iteration {i}')
            plt.xlabel('Feature')
            plt.ylabel('Emission')
            plt.title('Fitting Line Over Iterations')
            plt.legend()
            plt.pause(0.5)  # Pause to view each plot

    plt.show()
    return theta, cost_history
```

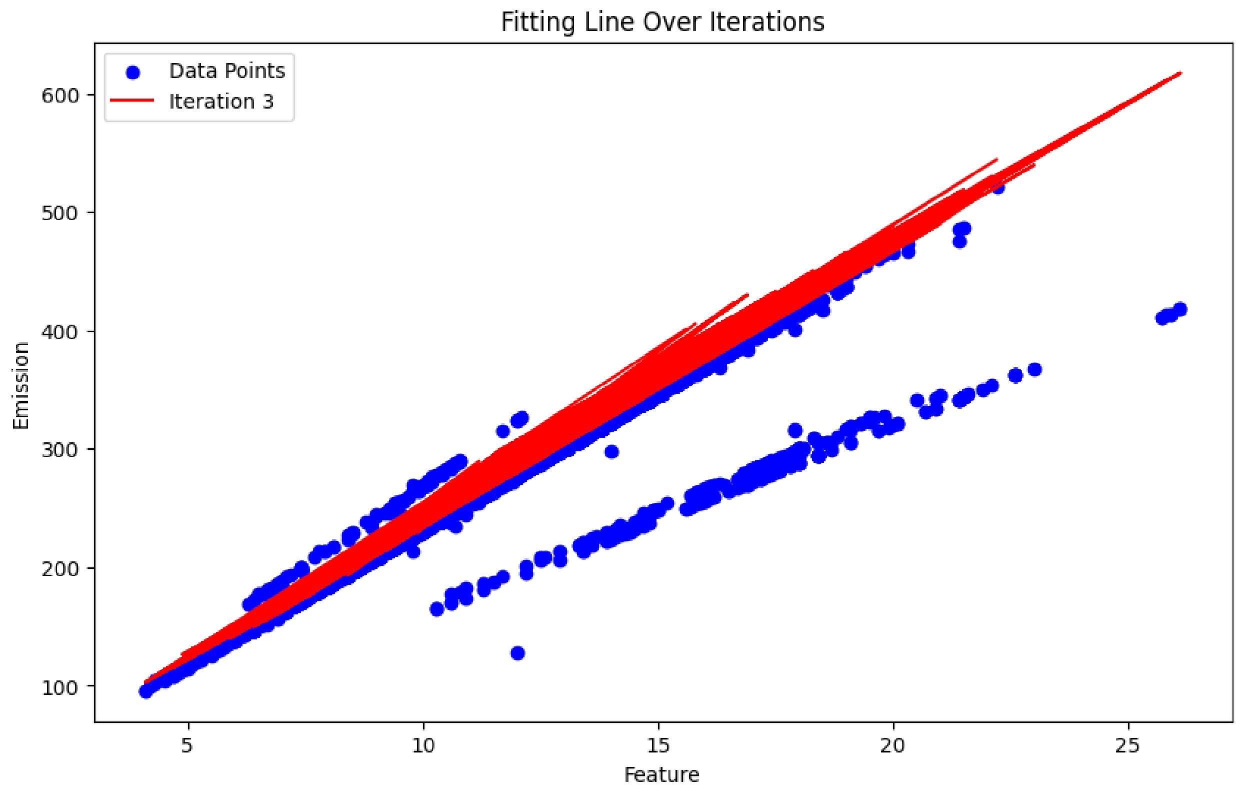
```
y_train_values = y_train['CO2 Emissions(g/km)'].values
learning_rate = 0.01
iterations = 2000

theta_initial = np.zeros(X_train_selected.shape[1])  # Initialize theta to zeros
theta_final, cost_history = gradient_descent(X_train_selected, y_train_values, theta_initial, learning_rate, iterations)
# Plot the cost function to visualize error improvement
plt.plot(range(iterations), cost_history)
plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.title('Cost Function vs. Iterations (Gradient Descent)')
plt.show()
```

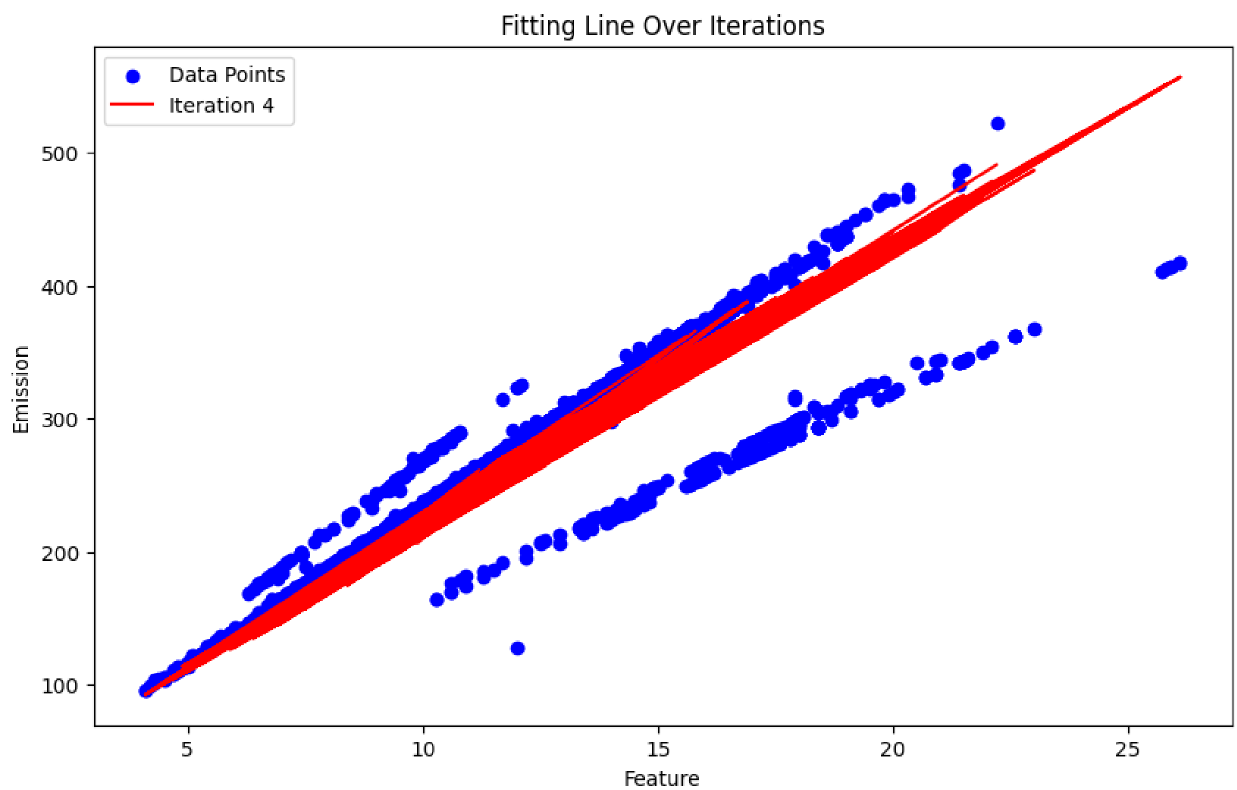
Output

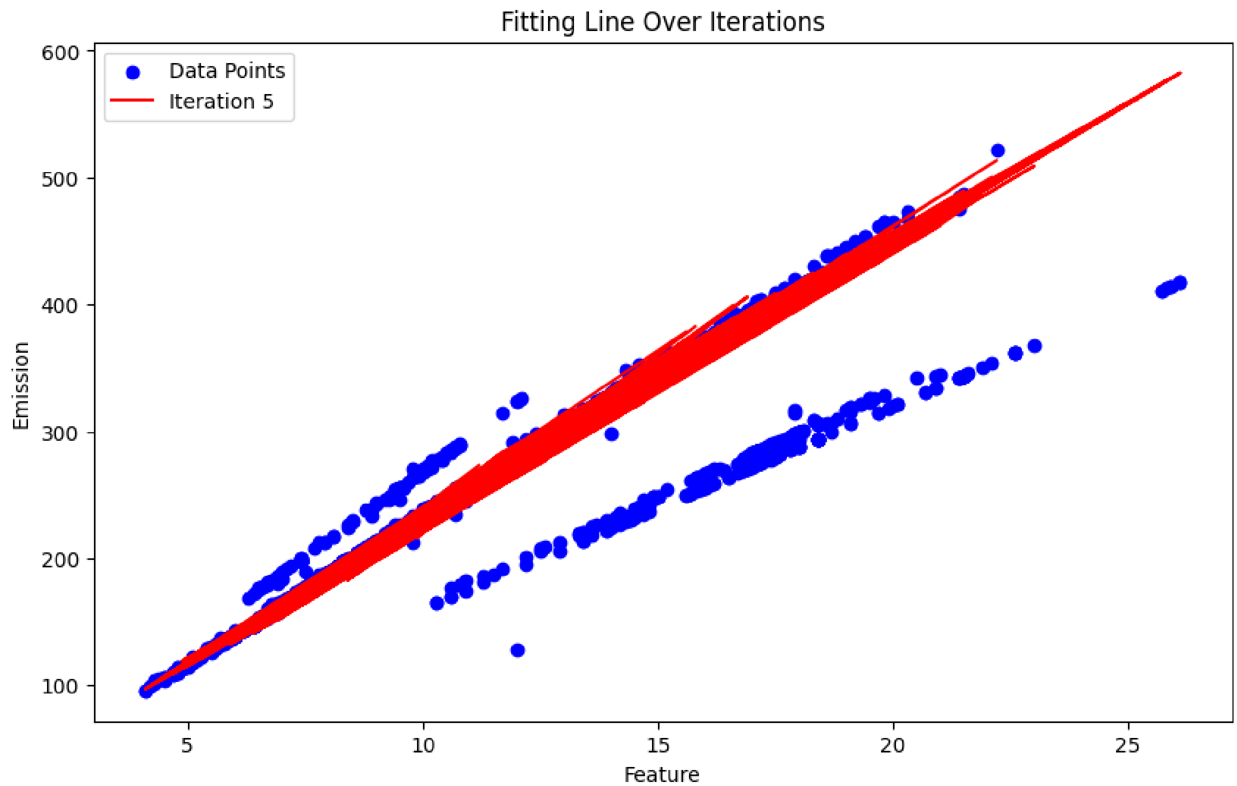



Fitting Line Over Iterations
Data Points
Iteration 1
800
700
600
500
400
300
200
100
Emission
5
10
15
20
25
Feature

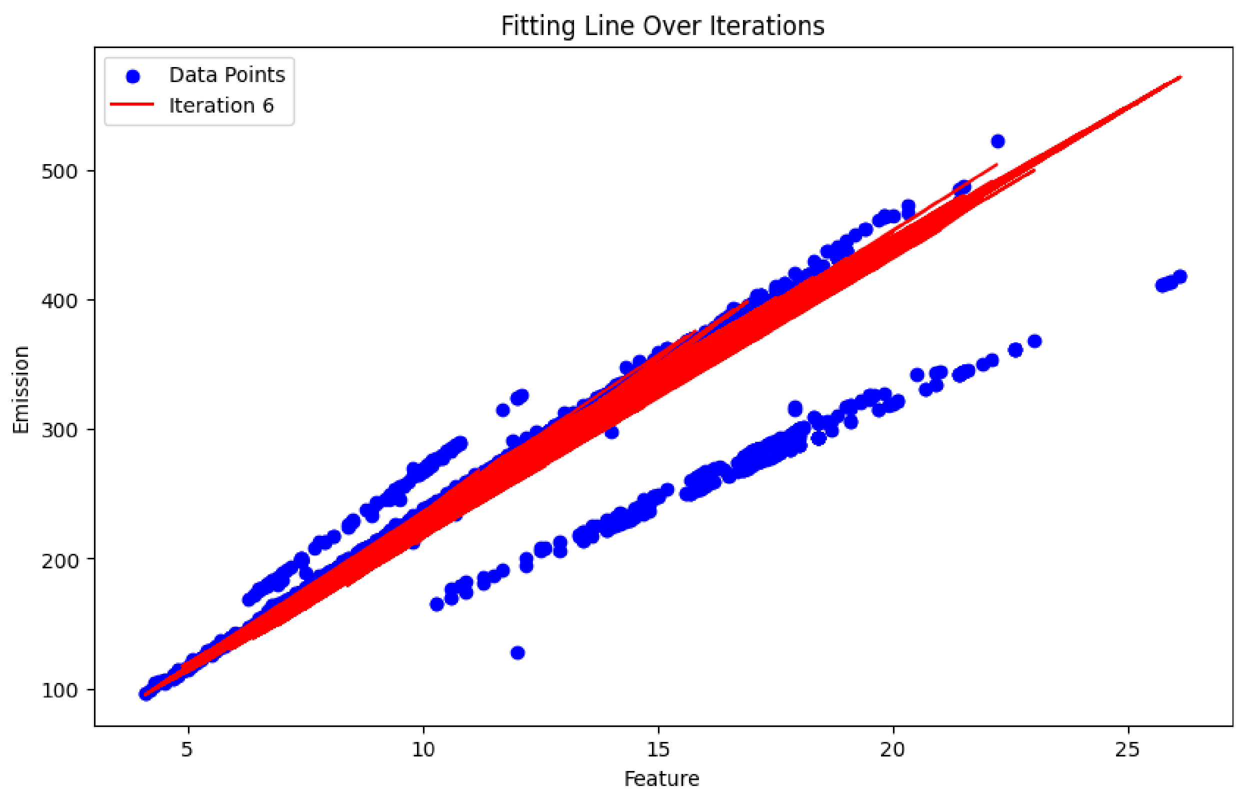

Fitting Line Over Iterations
Data Points
Iteration 2
500
400
300
200
100
Emission
5
10
15
20
25
Feature


Fitting Line Over Iterations
Data Points
Iteration 3
Emission
Feature
600
500
400
300
200
100
5
10
15
20
25

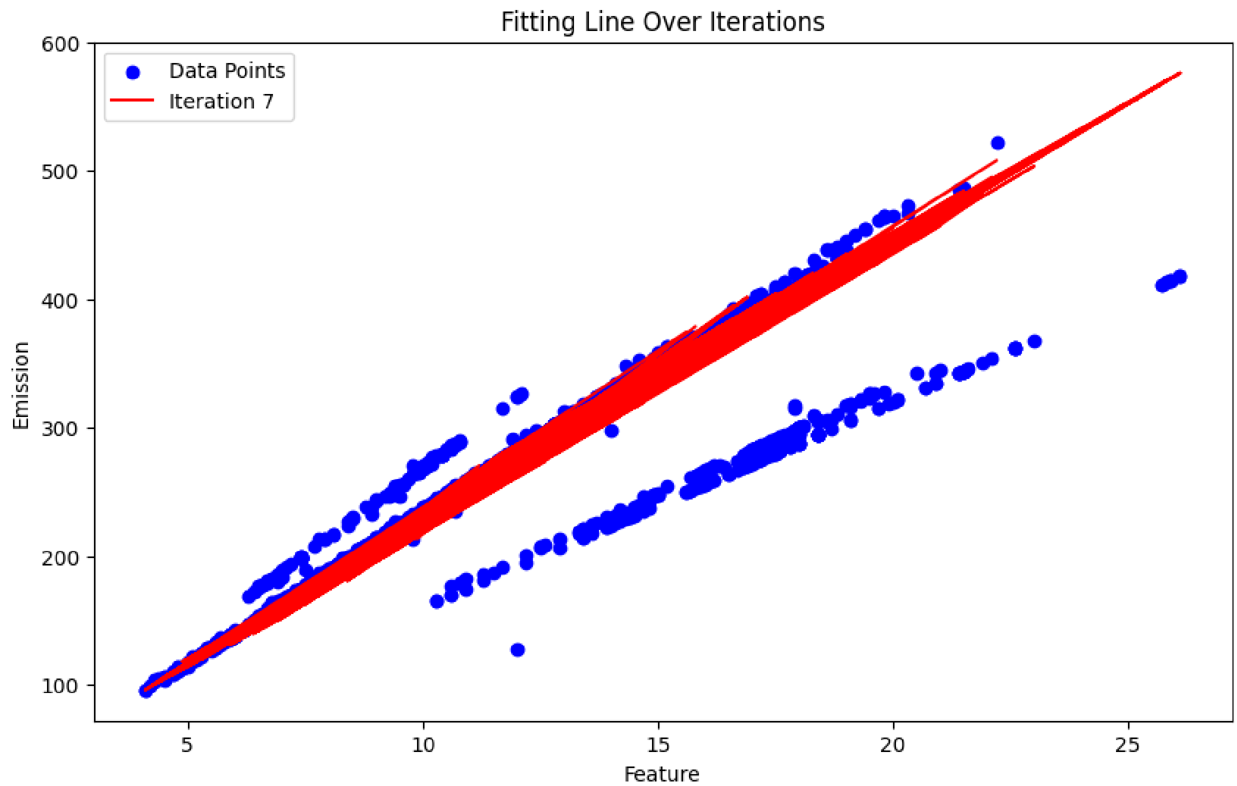

Fitting Line Over Iterations
Data Points
Iteration 4
Emission
Feature
500
400
300
200
100
5
10
15
20
25



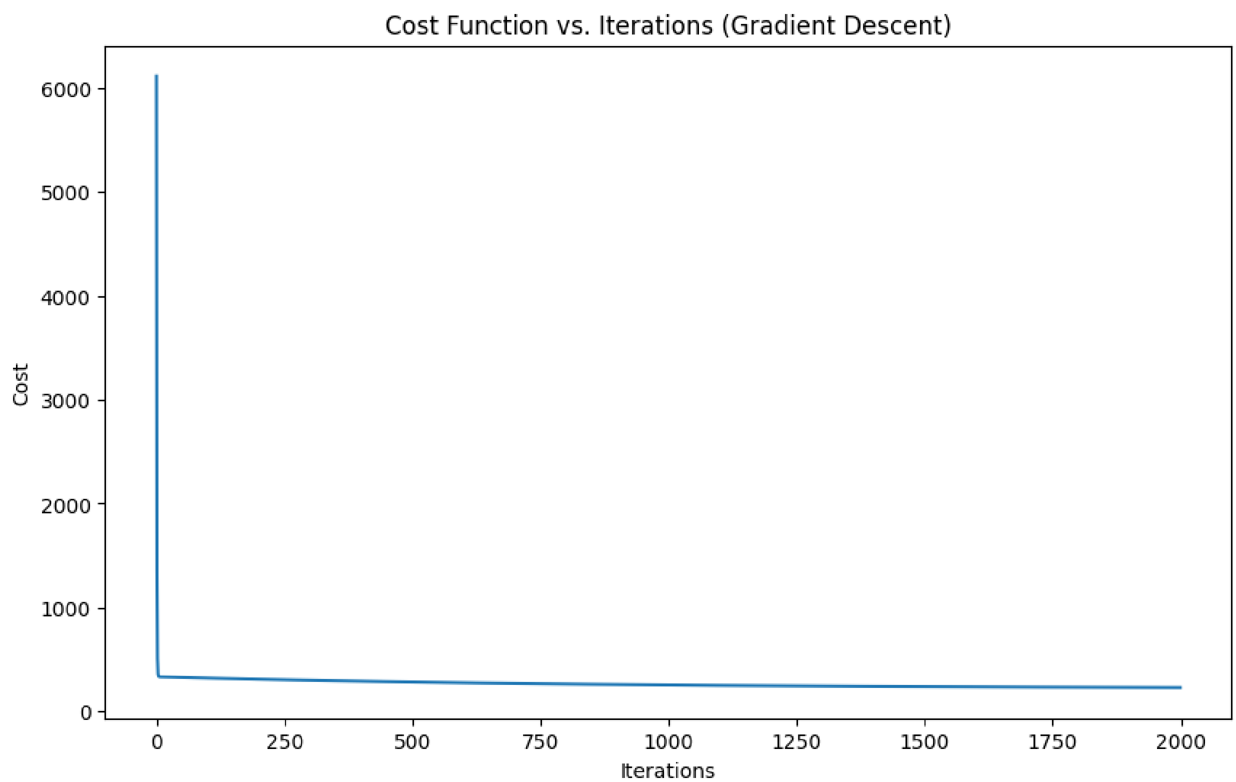
Fitting Line Over Iterations
Data Points
Iteration 5
Emission
Feature



Fitting Line Over Iterations
Data Points
Iteration 6
Emission
Feature



Fitting Line Over Iterations
Data Points
Iteration 7
Emission
Feature
100
200
300
400
500
600
5
10
15
20
25



Cost Function vs. Iterations (Gradient Descent)
Cost
Iterations
0
1000
2000
3000
4000
5000
6000
0
250
500
750
1000
1250
1500
1750
2000

- **Evaluate the model on the test set using Scikit-learn's R2 score.**
  Used **r2_score** to compare the actual values to the predicted values and the **R2 score = 0.8675010769944115.**

Evaluate Model

```
from sklearn.metrics import r2_score

y_test_values = y_test['CO2 Emissions(g/km)'].values

# Compute R² score using the predictions and actual values
r2 = r2_score(y_test_values, y_test_pred)

print(f"R² Score: {r2}")
```

```
R² Score: 0.8675010769944115
```

Logistic Regression Model

```
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import accuracy_score


sgd_clf = SGDClassifier(loss="log_loss", max_iter=2000, tol=1e-3, random_state=42)


sgd_clf.fit(X_train_selected, y_train['Emission Class'].values)


y_test_pred = sgd_clf.predict(X_test_selected)

accuracy = accuracy_score(y_test['Emission Class'].values, y_test_pred)
print(f'Accuracy of the logistic regression model on the test set: {accuracy:.4f}')
```

[90] ✓ 0.0s

```
Accuracy of the logistic regression model on the test set: 0.9769
```

Imported the scikit-learn library then defined
the sdc classifier (loss="log_loss": Configures the classifier to use the logistic regression loss function and max_iter=2000: Limits the number of iterations the SGD will perform during training),
then gave it the x and y required for training then used predict function to make predictions then printed the output.

```python
from sklearn.metrics import  classification_report

print("Classification Report:")
print(classification_report(y_test['Emission Class'].values,y_test_pred))
```

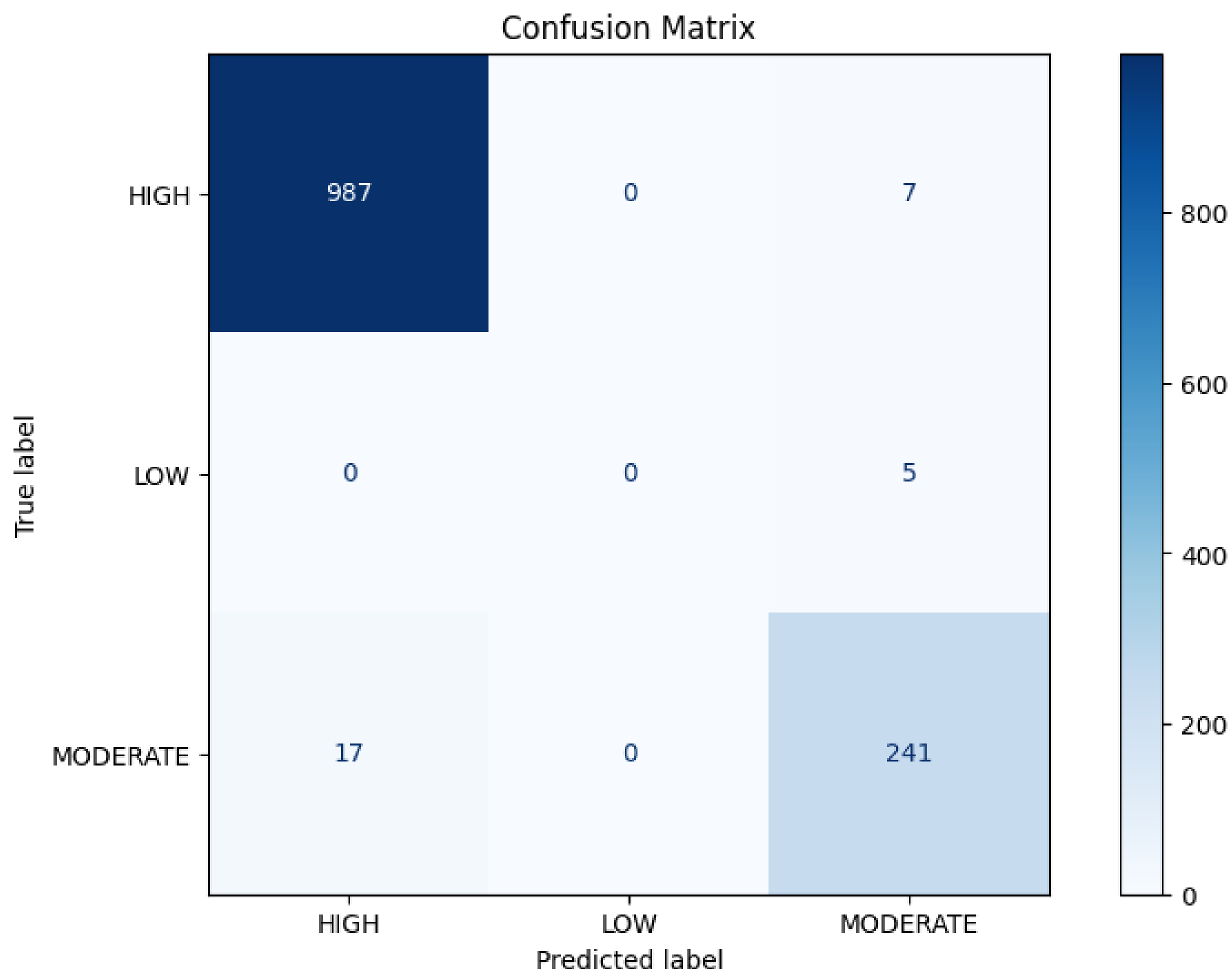
✓ 0.0s

```
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       994
           1       0.00      0.00      0.00         5
           2       0.95      0.93      0.94       258

    accuracy                           0.98      1257
   macro avg       0.65      0.64      0.64      1257
weighted avg       0.97      0.98      0.97      1257
```

Key Observations

1. **Class 0**:
   - Excellent performance (high precision, recall, F1-score).
2. **Class 1**:
   - Poor performance (model fails to predict this class).
   - Possible reason: **Class imbalance** (very few instances of class 1).
3. **Class 2**:
   - Good performance (slightly lower than class 0)

```python
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

y_true_original = label_encoder.inverse_transform(y_test['Emission Class'].values)
y_pred_original = label_encoder.inverse_transform(y_test_pred)

cm = confusion_matrix(y_true_original, y_pred_original)

unique_classes = sorted(set(y_true_original) | set(y_pred_original))

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=unique_classes)
disp.plot(cmap='Blues')
plt.title("Confusion Matrix")
plt.show()
```



Key Observations:

1. **HIGH Class**:
   - Excellent performance with **987 correctly classified** and only **7 misclassified** as MODERATE.
   - Indicates that the model predicts HIGH with high accuracy.
2. **LOW Class**:
   - Poor performance, as no instances of LOW were correctly classified.
   - Suggests that the model struggles significantly with this class.
   - Likely due to class imbalance (LOW has very few samples).
3. **MODERATE Class**:
   - Good performance overall, with **241 correctly classified**.
   - Some misclassifications: 17 as HIGH, and no instances predicted as LOW.