CO2 Emissions Data Analysis and Modeling

Name	ID
Rana Essam Ibrahim	20210133
Ahmed Yehia	20210049
Mohannad Hisham	20210413
NourEldin Ahmed	20210430
Merna Islam	20210500

TA: Eng. Sarah El-Nady

a. Loading the Dataset

first, we import these libraries.

```
import pandas
import numpy as np
import matplotlib.pyplot as plot
from sklearn.linear_model import  SGDClassifier
from sklearn.metrics import  r2_score, accuracy_score
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import  LabelEncoder
import seaborn as sns
from sklearn.preprocessing import StandardScaler
```

To begin, we load the dataset and inspect its structure.

```
# load the data
data = pandas.read_csv('co2_emissions_data.csv')
data

✓ 0.0s
```

	Make	Model	Vehicle Class	Engine Size(L)	Cylinders	Transmission	Fuel Type	Fuel Consumption City (L/100 km)	Fuel Consumption Hwy (L/100 km)
0	ACURA	ILX	COMPACT	2.0	4	AS5	Z	9.9	6.7
1	ACURA	ILX	COMPACT	2.4	4	M6	Z	11.2	7.7
2	ACURA	ILX HYBRID	COMPACT	1.5	4	AV7	Z	6.0	5.8
3	ACURA	MDX 4WD	SUV - SMALL	3.5	6	AS6	Z	12.7	9.1
4	ACURA	RDX AWD	SUV - SMALL	3.5	6	AS6	Z	12.1	8.7
...	...	...	...	...	...	...	...	...	...
7380	VOLVO	XC40 T5 AWD	SUV - SMALL	2.0	4	AS8	Z	10.7	7.7
7381	VOLVO	XC60 T5 AWD	SUV - SMALL	2.0	4	AS8	Z	11.2	8.3
7382	VOLVO	XC60 T6 AWD	SUV - SMALL	2.0	4	AS8	Z	11.7	8.6
7383	VOLVO	XC90 T5 AWD	SUV - STANDARD	2.0	4	AS8	Z	11.2	8.3
7384	VOLVO	XC90 T6 AWD	SUV - STANDARD	2.0	4	AS8	Z	12.2	8.7

7385 rows × 13 columns

b. Data Analysis

i. check whether there are missing values
ii. check whether numeric features have the same scale

```python
# step 1 : data analysis

# check nulls in the data
print(data.isnull().sum())

# get description of the data to detect the scale of the data
numericFeatures = data.select_dtypes(include=[np.number])
print(numericFeatures.describe())

```

Python

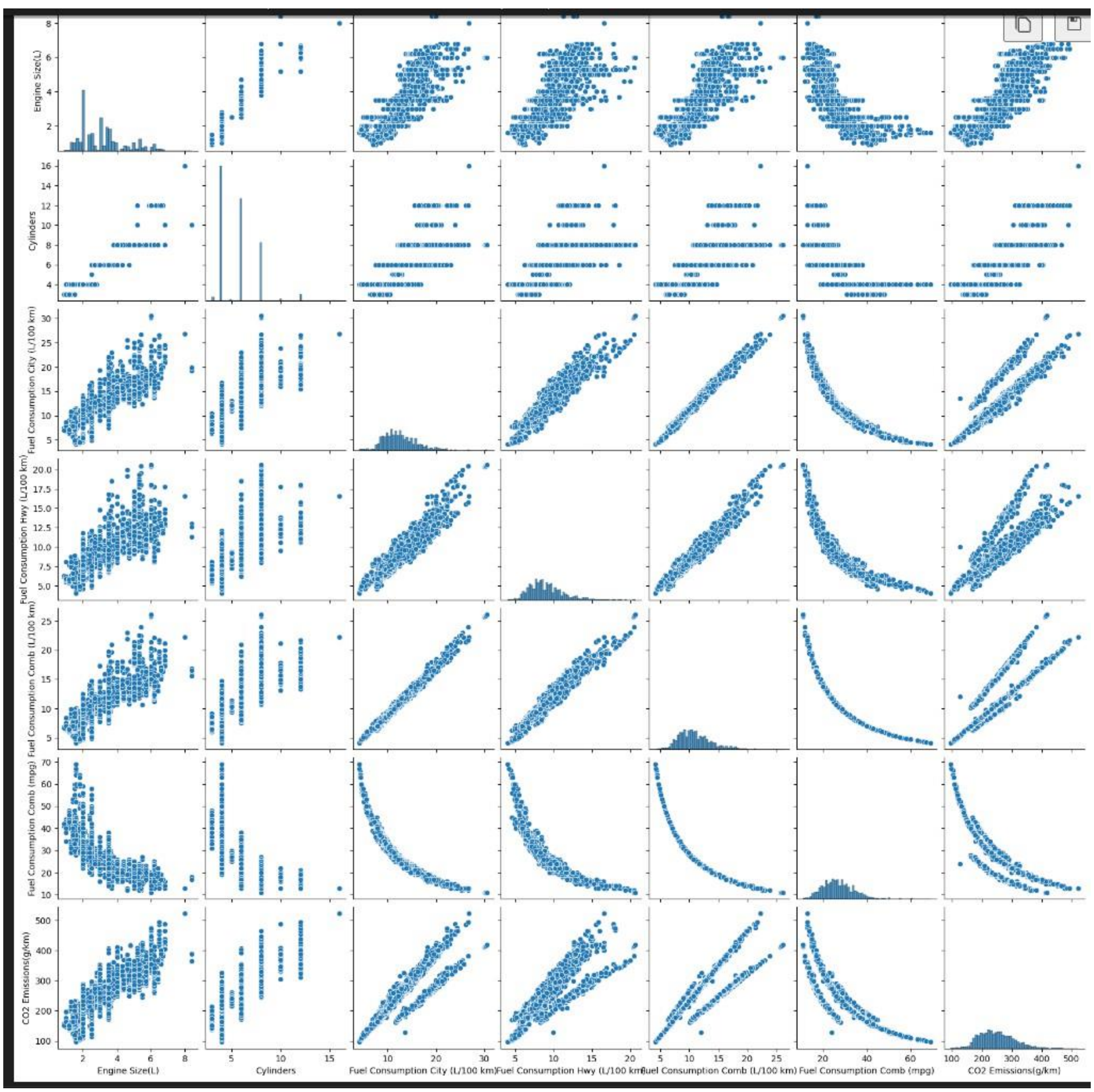
```
Make                                0
Model                               0
Vehicle Class                       0
Engine Size(L)                      0
Cylinders                           0
Transmission                        0
Fuel Type                           0
Fuel Consumption City (L/100 km)    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
       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)  \
...
25%                      22.000000          208.000000
50%                      27.000000          246.000000
75%                      32.000000          288.000000
max                      69.000000          522.000000
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

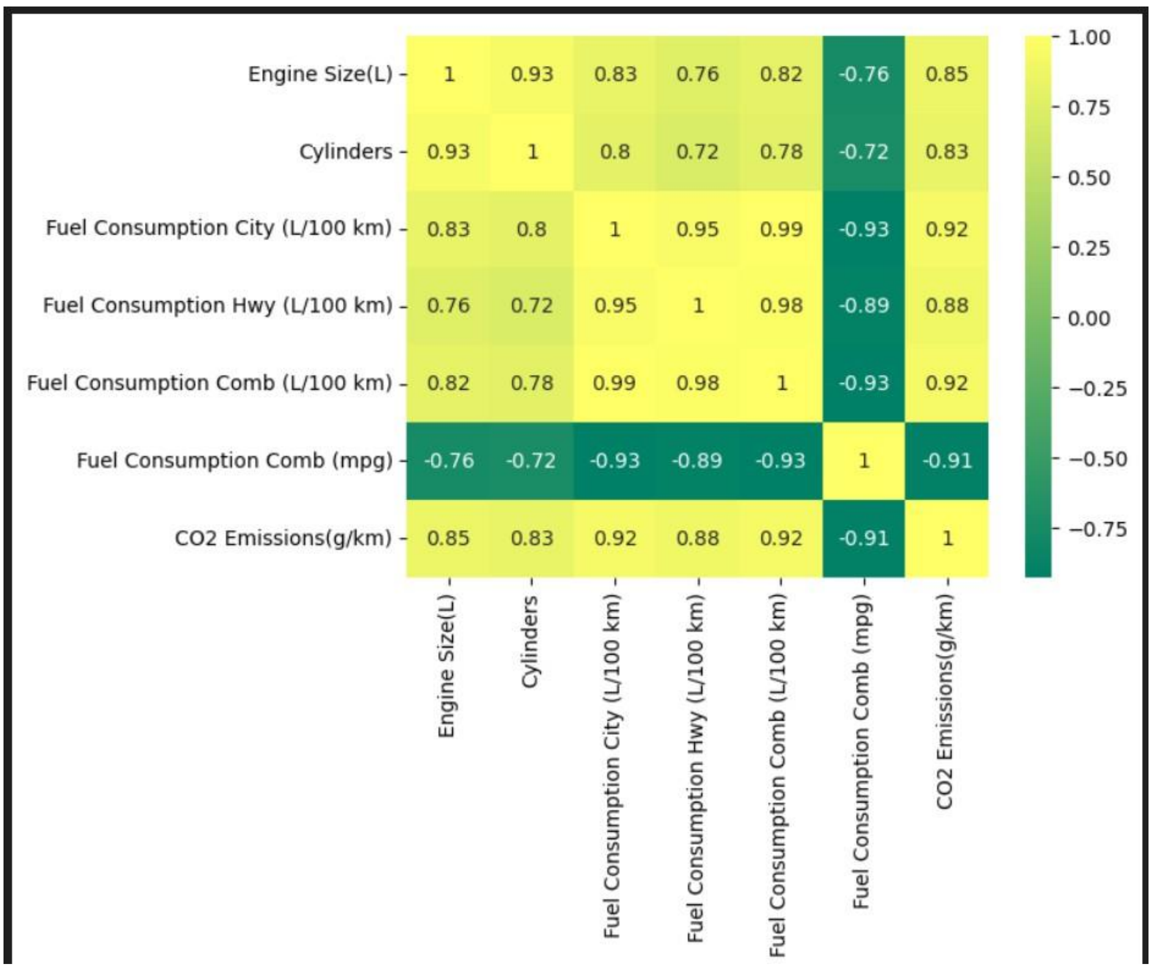
iii. Pairplot Visualization

```python
sns.pairplot(data, diag_kind='hist')
plot.show()
```



iv. Correlation Heatmap

```python
# as the heatmap works numerical values so we well select the numerical values only
numericCols = data.select_dtypes(include=[np.number])
correlation_matrix = numericCols.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='summer')
plot.show()
```



c.Data Preprocessing

First, Select the features that we work with them and any variable we will use in any model

```python
# this is the columns that we will use in the linear regression model and the logistic regression model as independent variables
feature1  = 'Cylinders'
feature2 = 'Fuel Consumption Comb (L/100 km)'
output = 'CO2 Emissions(g/km)'   # which is used in linear regression as dependent variable
alpha = 0.1
maxIterations = 1000
output2 = 'Emission Class'  # which is used in logistic regression as dependent variable
```

i. Target and Features separation has been done before training and testing for each model to be more readable as we are working on 2 models, not only one

iii. Shuffle and split the data into training and test data

```python
data = shuffle(data, random_state=100)  # shuffle the data

trainingData, testData = train_test_split(data, test_size=0.3, random_state=100) # split the data into training and testing data
```

iv. Scale the numerical values that we will use using Standard Scaler

```
scaler = StandardScaler()

#  fit and transform the training data
trainingDataScaled = pandas.DataFrame(scaler.fit_transform(trainingData[[feature1, feature2, output]]), columns=[feature1, feature2, output])

# transform only the test data
testDataScaled = pandas.DataFrame(scaler.transform(testData[[feature1, feature2, output]]), columns=[feature1, feature2, output])
```

ii. Encode the categorical values (this step could be done before splitting) but it will not affect the result in both cases

```
# label encoder to encod the emission class
labelEncoder = LabelEncoder()

# encode the output2 values to be numerical
output2EncodedTrain = pandas.DataFrame(labelEncoder.fit_transform(trainingData[output2]), columns=[output2])
output2EncodedTest = pandas.DataFrame(labelEncoder.fit_transform(testData[output2]), columns=[output2])
```

Finaly, concatenate both categorical values and numerical values after Preprocessing

```
# concatenat both numerical values with categorical values
trainingData = pandas.concat([trainingDataScaled, output2EncodedTrain], axis=1)
testData = pandas.concat([testDataScaled, output2EncodedTest], axis=1)
```

d. Linear Regression Implementation and Training

Select the Values that will enter the model

```python
feature1  = 'Cylinders'
feature2 = 'Fuel Consumption Comb (L/100 km)'
output = 'CO2 Emissions(g/km)'   # which is used in linear regression as dependent variable
alpha = 0.1
maxIterations = 1000
```

```python
x1 = trainingData[feature1].values
x2 = trainingData[feature2].values

x = np.column_stack((x1, x2))
y = trainingData[output].values
```

Create array of costs to store the cost after each iteration

```
costs = []   # to store the cost of each iteration
```

Implement the GD for linear regression model

```
def  fitGD(x, y, alpha, maxIterations):
    x = np.c_[np.ones(x.shape[0]), x]  # Add a column of ones to x for the bias term
    thetas = np.random.rand(x.shape[1])
    for i in range(maxIterations):
        h = np.dot( x, thetas)  # the hypothesis function
        for j in range(len(thetas)):
            partialDerivative = (1/len(y)) * np.sum((h - y) * x[:, j])  # the partial derivative of the cost function
            thetas[j] = thetas[j] - alpha * partialDerivative  # update the thetas values
            cost = (1/len(y)) * np.sum(np.square(h - y))  # the cost function
            costs.append(cost)  # store the cost of each iteration

    return thetas

```

Train the model to get thetas using training set

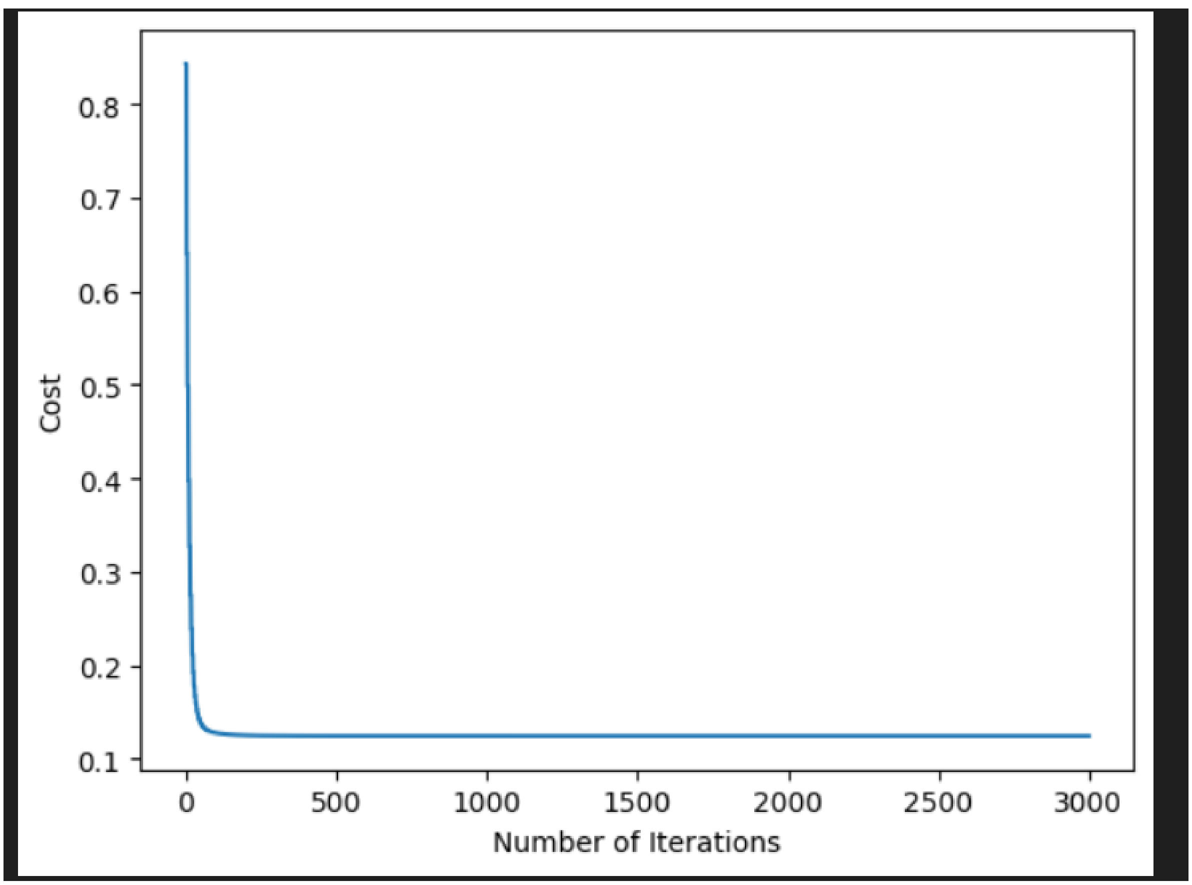
```python
thetas = fitGD(x, y, alpha, maxIterations) # fit the model using the training data to get the thetas values
print(thetas)
```

```
[-1.22506363e-16  2.93912439e-01  6.88954267e-01]
```

Draw the cost function

```python

# draw the cost function
plot.plot(costs)
plot.xlabel('Number of Iterations')
plot.ylabel('Cost')
plot.show()
```



Evaluate the Linear Regression Model

```python
#4.2 Predict the CO2 emissions for the test set using the linear regression model

def predictUsingLinearRegression(x, thetas):
    return np.dot( x, thetas)

# Prepare the test data
x1Test = testData[feature1].values
x2Test = testData[feature2].values


xTestTemp = np.column_stack((x1Test, x2Test))  # 2 diemtional X
xTest = np.c_[np.ones(xTestTemp.shape[0]), xTestTemp]  # Add a column of ones to x_test for the bias term

# Predict the CO2 emissions for the test set
yTest = testData[output].values

yPred = predictUsingLinearRegression(xTest, thetas)

# Calculate the R2 score
r2 = r2_score(yTest, yPred)
print('      "R2 score For"')

print(f'Linear Regression model: {r2}')
```

```
      "R2 score For"
Linear Regression model: 0.881053510923265
```

e. Train the logistic Regression Model using Stochastic Gradient Descent

```python
# Step 3.2 - Training the logistic regression model

# Predict the co2 emission class
yTest2 = testData[output2].values

# Stochastic Gradient Descent Classifier
logisticModel = SGDClassifier(loss='log_loss', random_state=99, max_iter=10000,tol=1e-3)

# Train the model using the training data
logisticModel.fit(x, trainingData[output2])

```

Evaluate the logistic regression model

```python
# Predict the CO2 emission class for the test set
yPred2 = logisticModel.predict(xTestTemp)


accuracyScore = accuracy_score(yTest2, yPred2)
print('  "accuracy_score For"')
print(f'Logistic Regression model: {accuracyScore}')


```

```
  "accuracy_score For"
Logistic Regression model: 0.9733754512635379
```