CSE343/ECE343: Machine Learning
Assignment-1 Linear and Logistic Regression, ML in Practice

Megha_2021337

1. (10 points) Section A (Theoretical)

(a) (2 marks) If two variables exhibit a strong correlation with a third variable, does this necessarily imply that they will also display a high degree of correlation with each other? Provide a reasoned explanation, supported by an illustrative example.

Sol: No,it is not necessary.This is known as '**Third Variable Problem**'.An observed correlation between two variables is actually explained by a third variable.And so,the initial two variables are not related to each other.

Example: A study found a positive correlation between the number of temples and number of crimes in a city. But that does not mean that temples are causing crimes or vice versa.Rather they both are related to population size.The larger the population, the more temples are built .And crime rates also increase due to larger population.

(b) (2 marks) What is the defining criteria(s) for a mathematical function to be categorized as a logistic function? Briefly explain which of the following functions are valid logistic functions: sinh(x), cosh(x), tanh(x), and signum(x).

Sol:Logistic function models involve limiting growth.It is a type of exponential function that models the growth of a population or a phenomenon that has a limit or a saturation point .

$$f(x) = L / (1 + e^{-k(x-xo)}) \quad \{ \text{sigmoid curve} \}$$

L : max value of function K : growth rate Xo : midpoint of the curve

1. The curve starts from zero 2. Increases rapidly
3. Approaches L asymptotically

$\sinh(x) = (e^x - e^{-x})/ 2$

Grows exponentially as x increases or decreases .Unbounded Function.Not a logistic function.

$\cosh(x) = (e^x + e^{-x})/ 2$

Grows exponentially as x increases and approaches 1 as x decreases.

Unbounded function.Not a logistic function.

tanh(x) = sinh(x) / cosh(x) = $(e^x - e^{-x}) / (e^x + e^{-x})$

After rationalizing : $(2/ (1+e^{-2x})) -1$.

This is a scaled version of logistic function.

signum(x) = {-1 x<0

0 x=0

1 x>0} This is an oscillating function from [-1 to 1]

Not a logistic function.

(c) (2 marks) Which validation technique is beneficial for very sparse datasets and why? Briefly explain how it is different from the traditional K-Fold cross-validation technique.

Sol :(LOOCV) Leave-one-out cross validation is beneficial for a very sparse dataset.It is a modified version of K-fold cross validation, where

(No. of folds) k = N (The no. of data points in the set).

In LOOCV, we maximize the size of the training set and minimize the bias of the performance.So, each observation is considered the validation set and the rest (N-1) observations are considered as the training set. We repeat this for N times for each observation as the validation set. This reduces the Bias and randomness. This aims at reducing Mean Squared Error rate and preventing overfitting.

Source : LOOCV

(d) (2 marks) Find the coefficients of the least square regression line for a set of 'n' data points (xi, yi) in slope-intercept form.

Sol : Y = a + b*X

Y = predicted value /estimated value

X = independent variable value

b = slope of line

a = y-intercept

The least square regression line minimizes the sum of the squared differences between the observed values 'yi' and values predicted by the line 'a + bX '

Mean of x : $\bar{x} = (\Sigma xi) / n$

Mean of y : $\bar{y} = (\Sigma yi) / n$

$b = \Sigma[(xi - \bar{x}) * (yi - \bar{y})] / \Sigma(xi - \bar{x})^2$

a = Y - b*X

(e) (1 mark) The parameters to be estimated in the simple linear regression

model Y = $\alpha+\beta x+\epsilon$ ϵ N(0,σ) are:

(a) α, β, σ

(b) α, β, ϵ

(c) a, b, s

(d) ϵ, 0, σ

Sol : a.) α, β, σ .

α = intercept , β = the slope and σ = standard deviation of the error term.

'ϵ' is not a parameter ,but a random variable that follows a normal distribution having (mean =0 , variance = σ)

(f) (1 mark) In a study of the relationship between X=mean daily temperature for the month and Y=monthly charges on the electric bill, the following data was gathered: X=[20, 30, 50, 60, 80, 90], Y= [125, 110, 95, 90,110, 130]. Which of the following seems the most likely model?

(a) Y= $\alpha + \beta x + \epsilon$ $\beta<0$

(b) Y= $\alpha + \beta x + \epsilon$ $\beta>0$

(c) Y= $\alpha + \beta_1 x + \beta_2 x^2 + \epsilon$ $\beta_2 < 0$

(d) Y= $\alpha + \beta_1 x + \beta_2 x^2 + \epsilon$ $\beta_2 > 0$

Sol: We can see that as the temperature increases , Y decreases .But after a certain value it increases at a higher rate. Parabolic relation
So,correct ans : (d) Y= α + β1x + β2x2+ϵ β2 > 0
, where β shows a positive relationship.

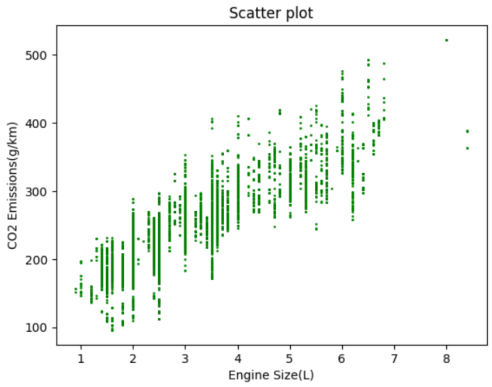
3. (15 points) Section C (Algorithm implementation using packages)

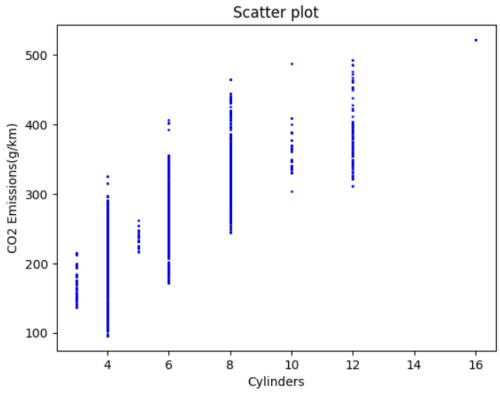
Implementation of linear regression using libraries:- Split the dataset into 80:20 (train: test)

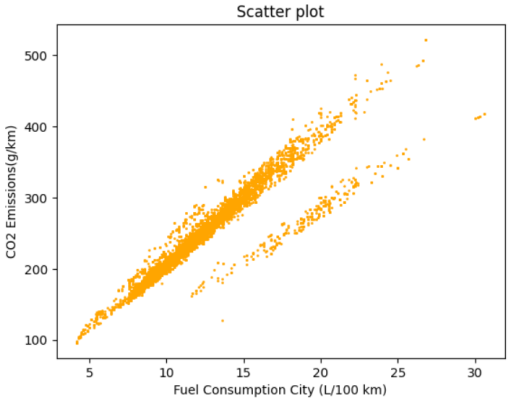
Dataset: CO2 Emissions Dataset

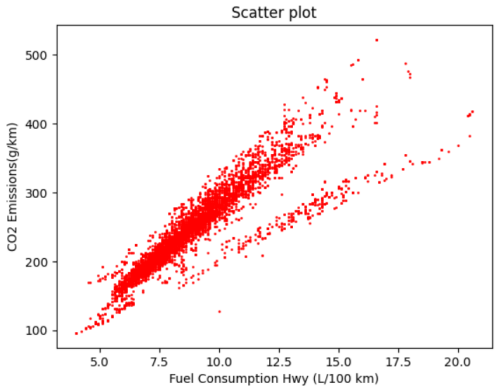
(a) (2 marks) Visualize the dataset. Make scatter plots, pair plots, box plots, and correlation heatmap. Distribution plots, i.e. histograms, pie charts etc. for categorical features. Give at least five insights on the dataset.

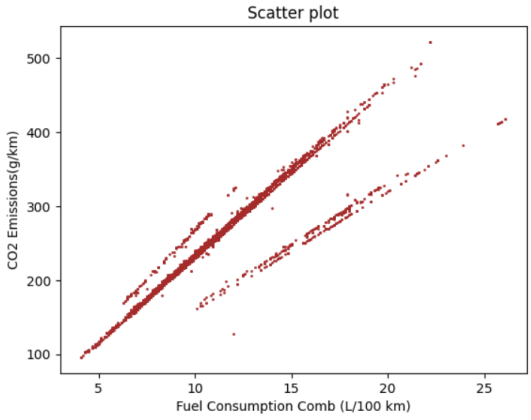
Scatter plots. Various parameters VS CO2 emission

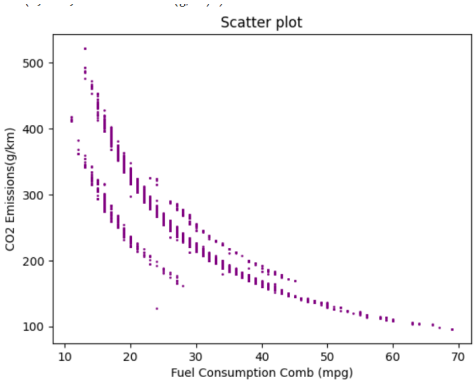




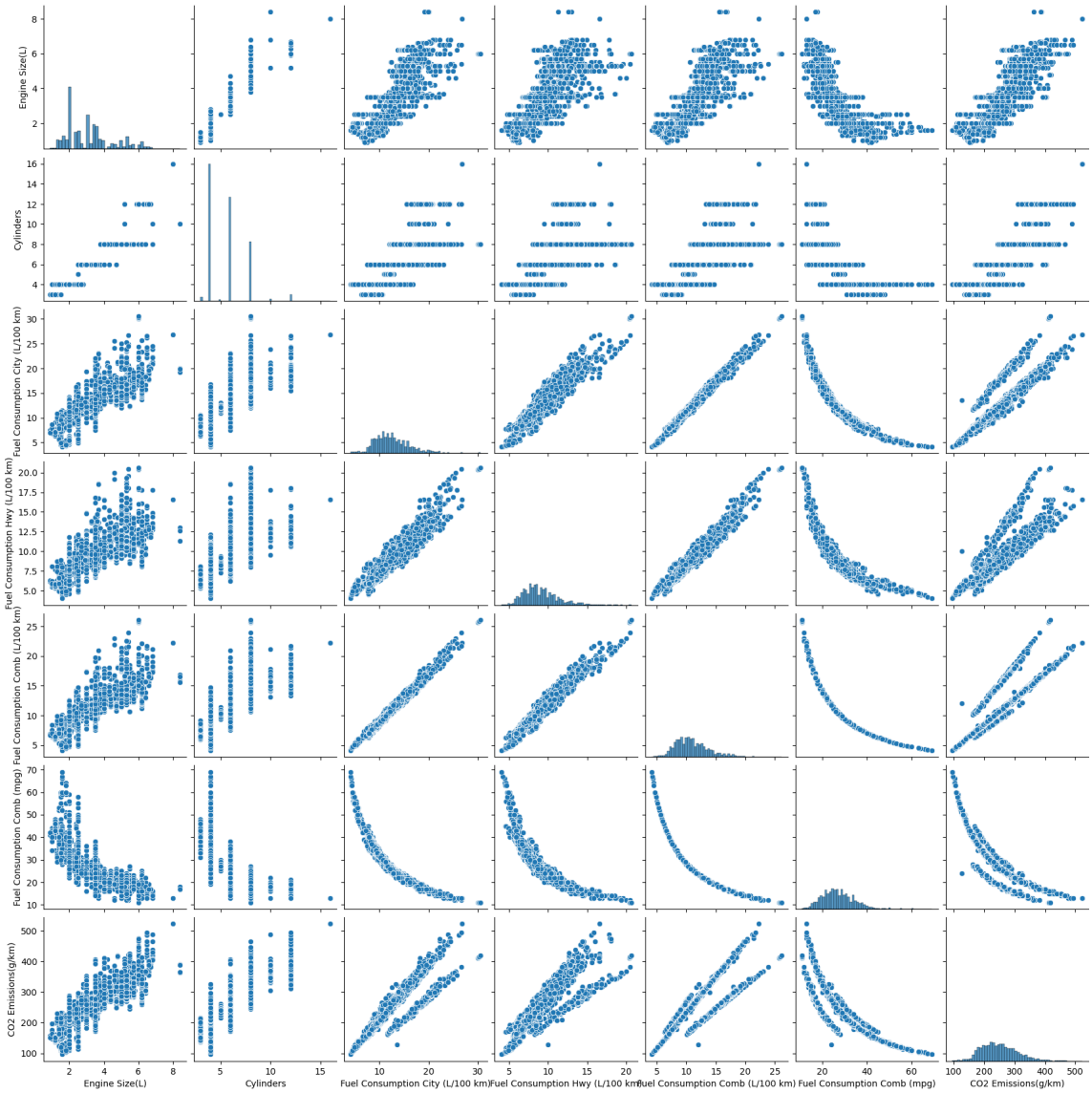






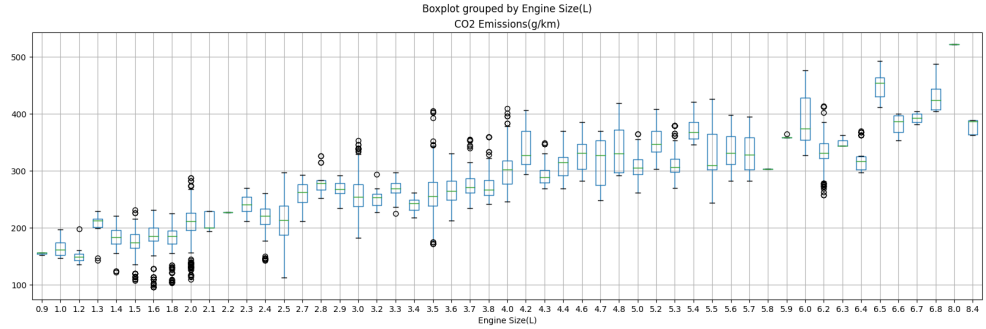


Pair plots.

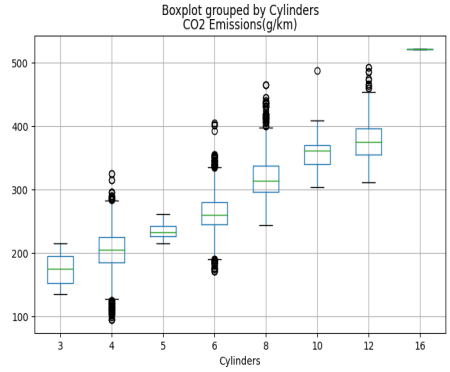


Box plot

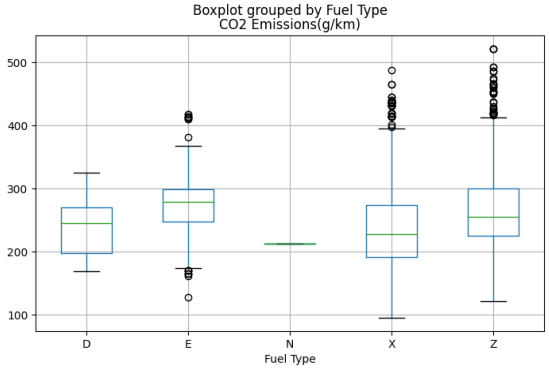
CO2 emission/grouped by Engine Size



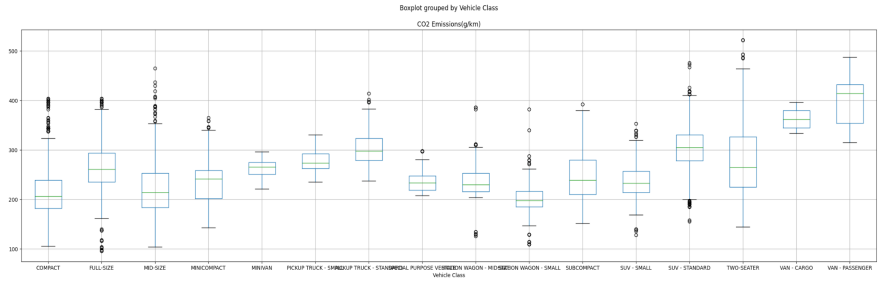
CO2 emission/ grouped by Cylinder



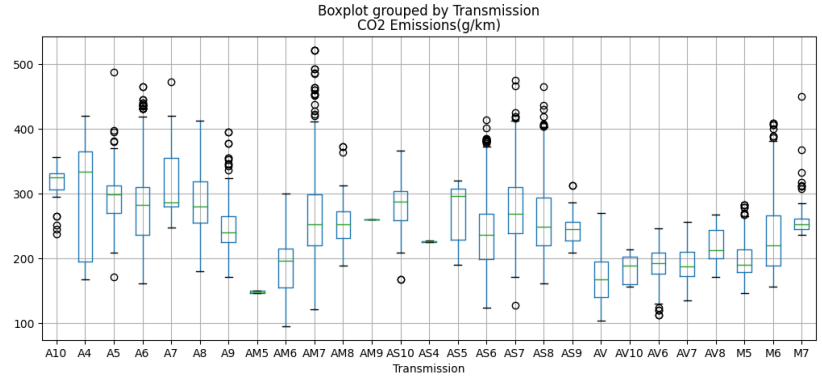
CO2 emission/ grouped by Fuel Type



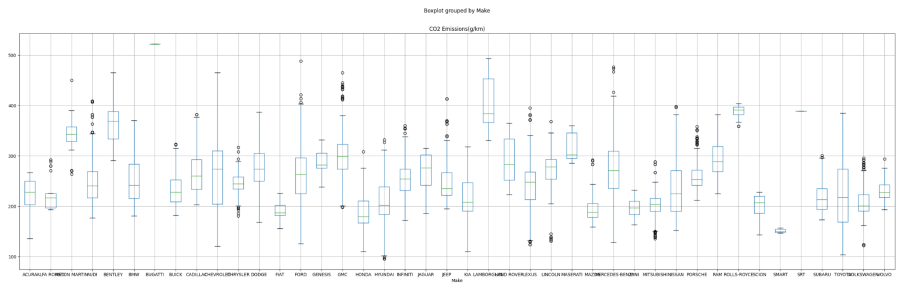
CO2 emission /Vehicle class



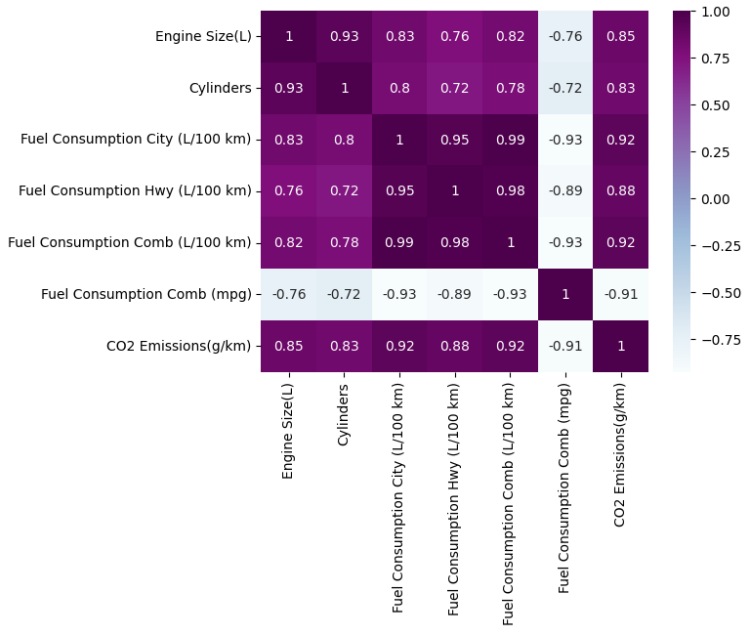
CO2 emission /grouped by Transmission



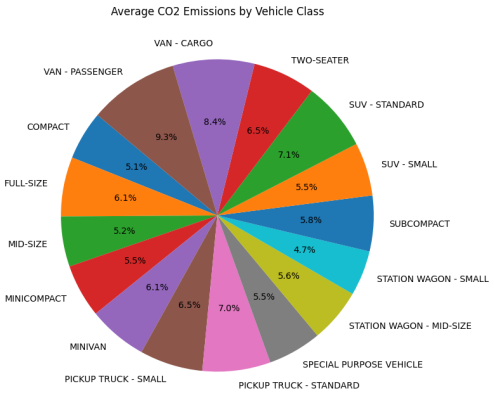
CO2 emission /grouped by Make



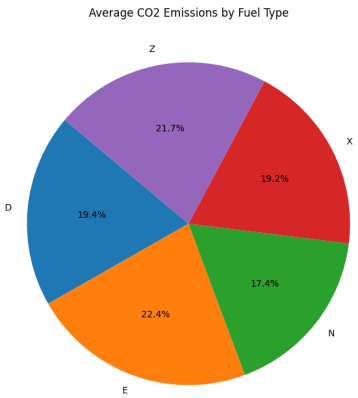
Correlation heatmap



Average CO2 emission by different vehicle classes.



Average CO2 emission by different Fuel types.



5 insights :

1.From pie chart : Fuel Type E has the highest average CO2 emission.

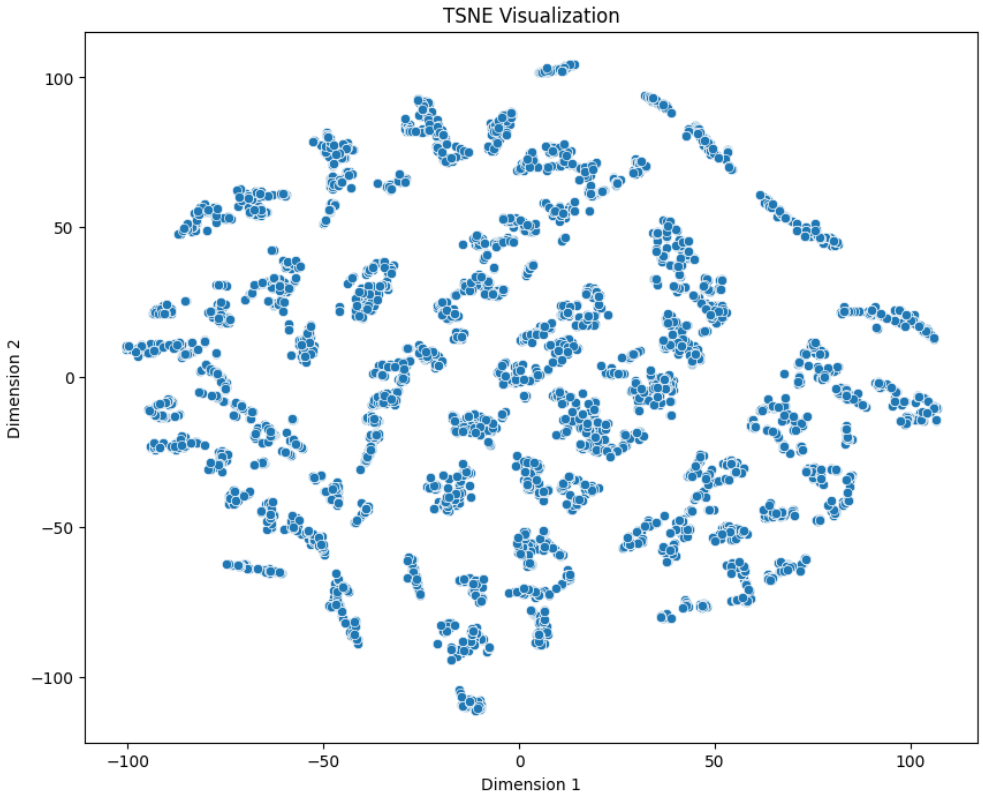
2. From the scattered plot we learned the more the cylinders are, the more CO2 is released.

3.From Correlation heatmap ,we observed that CO2 emission is highly related to fuel consumption in the city.Moreover the pair plot also signifies the same.

4.Every feature is negligible dependent on Fuel consumption comb.

5.From pie chart : Van passenger has highest average CO2 emission.

(b) (2 marks) Use TSNE (t-distributed stochastic neighbour embedding) algorithm to reduce data dimensions to 2 and plot the resulting data as a scatter plot. Comment on the separability of the data.



(c) (2 marks) Perform the necessary preprocessing steps, and for categorical features use label-based encoding. Perform linear regression on the preprocessed data. Report MSE, RMSE, R2 score, Adjusted R2 score, MAE on the train and test data.

Sol: We have performed label-based encoding for categorical features.This will convert categorical data into numerical values.Because Linear regression uses

mathematical expression to predict outcome.Therefore we need to encode categories.

Categorical column has all columns with non numerical values.

Our X comprises both numerical and categorical columns.

```
categorical_col = ['Make', 'Model', 'Vehicle Class', 'Transmission', 'Fuel Type']
numerical_col = ['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)']

X = data[categorical_col + numerical_col]
y = data['CO2 Emissions(g/km)']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=20)
```

Label Encoder(): This is a preprocessing technique used to convert text-based categories into numerical values.

pd.concat(): Then we have concatenated the values from both the training and testing set.So, that we will have all the unique categories.

label_encoder.fit(): Model will learn the mapping between each category and its numerical label.

```
#Label encoder on (training and testing data) for categorical features
label_encoder = LabelEncoder()
for col in categorical_col:
    combined_data = pd.concat([X_train[col], X_test[col]], axis=0)
    label_encoder.fit(combined_data)
    X_train[col] = label_encoder.transform(X_train[col])
    X_test[col] = label_encoder.transform(X_test[col])

#Feature scaling for numerical columns
scaler = StandardScaler()
X_train[numerical_col] = scaler.fit_transform(X_train[numerical_col])
X_test[numerical_col] = scaler.transform(X_test[numerical_col])
```

Then we calculated the MSE, RMSE, R2 score, Adjusted R2 score, MAE on Training and Testing data.The results are as follows:

```
Train Data Metrics:
MSE: 285.59
RMSE: 16.90
R2 Score: 0.92
MAE: 11.01

Test Data Metrics:
MSE: 296.58
RMSE: 17.22
R2 Score: 0.91
MAE: 11.22
```

:

(d) (2 marks) Use Principal Component Analysis (PCA) on the original dataset to reduce the number of features and then train the model with the reduced feature dataset. Vary the number of components, i.e. 4, 6, 8, 10 and compare the results (MSE, RMSE, R2 score, Adjusted R2 score, MAE) on the train and test dataset.

Sol: Principal Component Analysis (PCA) is dimensionality reduction technique.It is used to simplify complex datasets by reducing the number of features ,while preserving essential information.

We are using this on pre-processed data we got after label-based encoding.

Code: We run a 'for' loop for exploring different numbers of components in the array [2,4,6,8,10].

PCA(n_components=n_components) :We are specifying the number of components.

pca.fit_transform(X_train) : This will compute principal components and transform training data into new feature space, with lower dimension.

pca.transform(X_test) : Same, it will use principal components learned from training data,and transform this.

```
from sklearn.decomposition import PCA

for n_components in [2, 4, 6,8,10]:

    pca = PCA(n_components=n_components)
    X_train_pca = pca.fit_transform(X_train)
    X_test_pca = pca.transform(X_test)

    #fitting linear regression model
    model = LinearRegression()
    model.fit(X_train_pca, y_train)

    # Predict Training and testing result
    y_train_pred = model.predict(X_train_pca)
    y_test_pred = model.predict(X_test_pca)

    #Performance metrices for train and test data
    mse_train = mean_squared_error(y_train, y_train_pred)
    rmse_train = np.sqrt(mse_train)
    r2_train = r2_score(y_train, y_train_pred)
    mae_train = mean_absolute_error(y_train, y_train_pred)

    mse_test = mean_squared_error(y_test, y_test_pred)
    rmse_test = np.sqrt(mse_test)
    r2_test = r2_score(y_test, y_test_pred)
    mae_test = mean_absolute_error(y_test, y_test_pred)
```

Then we calculated the MSE, RMSE, R2 score, Adjusted R2 score, MAE for different numbers of components on Training and Testing data.

The results are as follows:

We can clearly observe that as the no. of components increases

MSE /RMSE/ MAE decreases. But R2 Score increases.

```
Number of Components: 2
TRAIN DATA METRICES :
MSE: 3285.24
RMSE: 57.32
R2 Score: 0.05
MAE: 45.26

Test DATA METRICES:
MSE: 3106.68
RMSE: 55.74
R2 Score: 0.06
MAE: 43.97

Number of Components: 4
TRAIN DATA METRICES :
MSE: 2758.28
RMSE: 52.52
R2 Score: 0.20
MAE: 40.93

Test DATA METRICES:
MSE: 2627.42
RMSE: 51.26
R2 Score: 0.20
MAE: 39.73

Number of Components: 6
TRAIN DATA METRICES :
MSE: 306.78
RMSE: 17.52
R2 Score: 0.91
MAE: 12.11

Test DATA METRICES:
MSE: 314.19
```

```
RMSE: 17.52
R2 Score: 0.91
MAE: 12.11

Test DATA METRICES:
MSE: 314.19
RMSE: 17.73
R2 Score: 0.90
MAE: 12.28

Number of Components: 8
TRAIN DATA METRICES :
MSE: 286.56
RMSE: 16.93
R2 Score: 0.92
MAE: 11.03

Test DATA METRICES:
MSE: 298.13
RMSE: 17.27
R2 Score: 0.91
MAE: 11.27

Number of Components: 10
TRAIN DATA METRICES :
MSE: 285.67
RMSE: 16.90
R2 Score: 0.92
MAE: 11.01

Test DATA METRICES:
MSE: 296.45
RMSE: 17.22
R2 Score: 0.91
MAE: 11.22
```

(e) (2 marks) Encode the categorical features of the original dataset with one-hot encoding and perform all tasks of part c again, i.e. apply linear regression and report MSE, RMSE etc. Compare the results obtained with part c

Sol: I had used Hot encoding for converting categorical features .But I needed to drop one table, as it had a wide variety of different values.

```
data = data.drop([ 'Model'], axis=1)
categorical_col = ['Make','Vehicle Class', 'Transmission', 'Fuel Type']
```

OneHotEncoder class : This will assign each unique categorical feature a unique binary(0 or 1) value.

drop='first' : For each categorical feature, the first category encountered should be dropped to avoid redundant information.

Sparse =False : Ensure the output will not a sparse matrix.

Then we compute one-hot encoded representation of categorical features for the training set and apply the same scheme in the testing set.

```
onehot_encoder = OneHotEncoder(drop='first', sparse=False)
X_train_encoded = onehot_encoder.fit_transform(X_train[categorical_col])
X_test_encoded = onehot_encoder.transform(X_test[categorical_col])
```

```
One-Hot Encoding Results:
Train Data Metrics:
MSE: 21.89
RMSE: 4.68
R2 Score: 0.99
MAE: 2.89

Test Data Metrics:
MSE: 19.95
RMSE: 4.47
R2 Score: 0.99
MAE: 2.92
```

:

Comparing with result obtained in c.

```
Train Data Metrics:
MSE: 285.59
RMSE: 16.90
R2 Score: 0.92
MAE: 11.01

Test Data Metrics:
MSE: 296.58
RMSE: 17.22
R2 Score: 0.91
MAE: 11.22
```

The one hot encoding is performing better in evaluation metrics..

(f) (2 marks) Perform PCA on the one-hot encoded dataset and choose the appropriate number of components (try 5 different values). Compare the results (MSE, RMSE, R2 score, Adjusted R2 score, MAE) on the train and test dataset.

Sol: In this we had performed PCA on one hot encoded dataset: The code is same as in part d.

```python
for n_components in [2,4,6,8,10]:

    pca = PCA(n_components=n_components)
    X_train_pca = pca.fit_transform(X_train_scaled)
    X_test_pca = pca.transform(X_test_scaled)

   #fitting linear regression model
    model = LinearRegression()
    model.fit(X_train_pca, y_train)

    # Predict Training and testing result
    y_train_pred = model.predict(X_train_pca)
    y_test_pred = model.predict(X_test_pca)

    #Performance metrices for train and test data
    mse_train = mean_squared_error(y_train, y_train_pred)
    rmse_train = np.sqrt(mse_train)
    r2_train = r2_score(y_train, y_train_pred)
    mae_train = mean_absolute_error(y_train, y_train_pred)

    mse_test = mean_squared_error(y_test, y_test_pred)
    rmse_test = np.sqrt(mse_test)
    r2_test = r2_score(y_test, y_test_pred)
    mae_test = mean_absolute_error(y_test, y_test_pred)
```

```
Number of Components: 2
Train Data Metrics:
MSE: 553.28
RMSE: 23.52
R2 Score: 0.84
MAE: 16.20

Test Data Metrics:
MSE: 563.67
RMSE: 23.74
R2 Score: 0.83
MAE: 16.40

Number of Components: 4
Train Data Metrics:
MSE: 548.54
RMSE: 23.42
R2 Score: 0.84
MAE: 16.11

Test Data Metrics:
MSE: 562.12
RMSE: 23.71
R2 Score: 0.83
MAE: 16.26

Number of Components: 6
Train Data Metrics:
MSE: 538.08
RMSE: 23.20
R2 Score: 0.84
MAE: 15.84

Test Data Metrics:
MSE: 547.64
RMSE: 23.40
R2 Score: 0.83
MAE: 15.96

Number of Components: 8
Train Data Metrics:
MSE: 540.19
RMSE: 23.24
R2 Score: 0.84
MAE: 15.87

Test Data Metrics:
MSE: 551.15
RMSE: 23.48
R2 Score: 0.83
MAE: 16.00

Number of Components: 10
Train Data Metrics:
MSE: 515.37
RMSE: 22.70
R2 Score: 0.85
MAE: 15.65

Test Data Metrics:
MSE: 533.92
RMSE: 23.11
R2 Score: 0.84
MAE: 15.90
```

(g) (1.5 marks) Use L1 and L2 regularization while training the linear model (use the preprocessed dataset of part c). Compare the MSE, RMSE, R2 score, Adjusted R2 score, and MAE on the test dataset for both regularization techniques.

Sol: L1 regularization (Lasso) : It shrinks the less important features coefficient to zero.Used for feature selection in case we have a huge number of features.

L2 regularization (Ridge) : Prevent overfitting by penalizing the sum of squared coefficients to prevent them from becoming too large.

Alpha (Controls the strength of regularization)

If alpha is << 1.Then both L1&L2 behave like an ordinary linear regression problem.

If Alpha == 1, it means applying a moderate level of regularization.

Then we are fitting the model to the training data.

During training the Lasso model it adjusts its coefficient to find best linear fit, which causes some coefficients to become zero.Same with Ridge .

```python
from sklearn.linear_model import Lasso, Ridge

#L1 (Lasso) regularization
lasso_m = Lasso(alpha=1.0)
lasso_m.fit(X_train, y_train)
lasso_pred = lasso_m.predict(X_test)

#L2 (Ridge) regularization
ridge_m = Ridge(alpha=1.0)
ridge_m.fit(X_train, y_train)
ridge_pred = ridge_m.predict(X_test)
```

```
Lasso Regression Metrics:
MSE: 296.66
RMSE: 17.22
R-squared: 0.91
Adjusted R-squared: 0.91
MAE: 11.29

Ridge Regression Metrics:
MSE: 296.49
RMSE: 17.22
R-squared: 0.91
Adjusted R-squared: 0.91
MAE: 11.22
```

(h) (1.5 marks) Use SGDRegressor library to perform linear regression on the preprocessed dataset of part c. Report the evaluation metrics and compare the results.

Sol: SGDRegressor is used when we deal with large datasets where loading the dataset into memory is not feasible.It is faster than traditional gradient descent, and can be controlled by maximum no of iterations /regularization strength / seed for randomness.

I had set the maximum no. of iterations to 10,000 ,meaning that the algorithm will stop after going through the training data 10,000 times (or when convergence is met).

```
#Q3. h. SGD Regressor and calculating MSE, RMSE, R2 score, Adjusted R2 score, MAE on Training and Testing data
from sklearn.linear_model import SGDRegressor

sgd_m = SGDRegressor(max_iter=10000, random_state=20)
sgd_m.fit(X_train, y_train)
sgd_pred = sgd_m.predict(X_test)

#Performance metrices for train and test data
sgd_mse = mean_squared_error(y_test, sgd_pred)
sgd_rmse = np.sqrt(sgd_mse)
sgd_r2 = r2_score(y_test, sgd_pred)
sgd_mae = mean_absolute_error(y_test, sgd_pred)
```

```
SGDRegressor Metrics:
MSE: 340541836943968789582367948 8.00
RMSE: 58355962586865.86
R2 Score: -10319960791107658527539 20.00
MAE: 50152961015500.28
```