Name : Andi Izzat Zaky Ashari
NIM : 2501977172

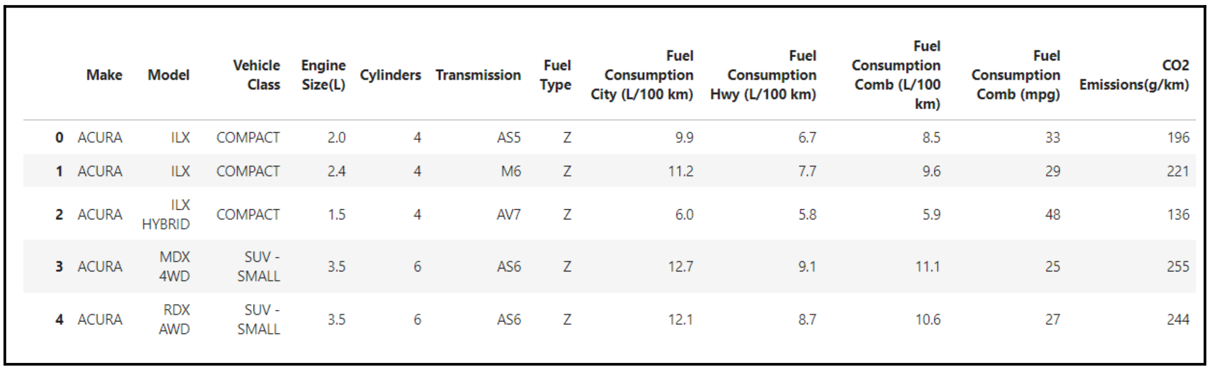
CO2 Emission by Vehicles

(https://www.kaggle.com/datasets/debajyotipodder/co2-emission-by-vehicles/data)

About the dataset

This dataset captures the details of how CO2 emissions by a vehicle can vary with the different features. The dataset has been taken from Canada Government official open data website. There are total 7385 rows and 12 columns.

Here is the first 5 rows of the data contains in this dataset



	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)
0	ACURA	ILX	COMPACT	2.0	4	AS5	Z	9.9	6.7	8.5	33	196
1	ACURA	ILX	COMPACT	2.4	4	M6	Z	11.2	7.7	9.6	29	221
2	ACURA	ILX HYBRID	COMPACT	1.5	4	AV7	Z	6.0	5.8	5.9	48	136
3	ACURA	MDX 4WD	SUV - SMALL	3.5	6	AS6	Z	12.7	9.1	11.1	25	255
4	ACURA	RDX AWD	SUV - SMALL	3.5	6	AS6	Z	12.1	8.7	10.6	27	244

Columns that needed to be explained:

Model

• 4WD/4X4	= Four-wheel drive
• AWD	= All-wheel drive

FFV	= Flexible-fuel vehicle
SWB	= Short wheelbase
LWB	= Long wheelbase
EWB	= Extended wheelbase

Transmission

A	= Automatic
AM	= Automated manual
AS	= Automatic with select shift
AV	= Continuously variable
M	= Manual
3 - 10	= Number of gears

Fuel type

X	= Regular gasoline
Z	= Premium gasoline
D	= Diesel
E	= Ethanol (E85)
N	= Natural gas

Fuel Consumption

City and highway fuel consumption ratings are shown in litres per 100 kilometres (L/100 km) - the combined rating (55% city, 45% hwy) is shown in L/100 km and in miles per gallon (mpg)

CO2 Emissions

The tailpipe emissions of carbon dioxide (in grams per kilometre) for combined city and highway driving

Code

(https://drive.google.com/file/d/11GNoqoiuEQ5PLf88N3teUSP-36Y4kHDQ/view?usp=sharing)

EDA - Exploratory Data Analysis

Firstly, lets see the basic information of this dataset

```
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   Make                              7385 non-null   object
 1   Model                             7385 non-null   object
 2   Vehicle Class                     7385 non-null   object
 3   Engine Size(L)                    7385 non-null   float64
 4   Cylinders                         7385 non-null   int64
 5   Transmission                      7385 non-null   object
 6   Fuel Type                         7385 non-null   object
 7   Fuel Consumption City (L/100 km)  7385 non-null   float64
 8   Fuel Consumption Hwy (L/100 km)   7385 non-null   float64
 9   Fuel Consumption Comb (L/100 km)  7385 non-null   float64
 10  Fuel Consumption Comb (mpg)       7385 non-null   int64
 11  CO2 Emissions(g/km)               7385 non-null   int64
```

As mentioned before, this dataset contains 12 columns, there are 5 columns of strings, and 7 columns of integers and float. From this output, we can see that there are no missing values on this dataset, but I wonder, is there any data duplication here? Let's see..

```
print("Duplicated data: ",df.duplicated().sum())
> Duplicated data:  1103
```

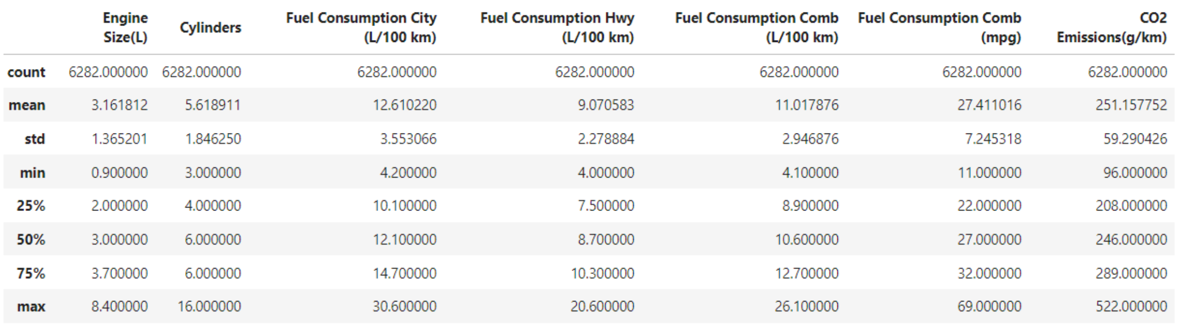
Based on the result, there are 1,103 data that are duplicated, lets handle it!

```
workingDF = df.drop_duplicates()
print("Number of duplicates after removal:", df.duplicated().sum())
> Number of duplicates after removal: 0
```

In this code, I created a dataframe named 'workingDF' and I assign a `.drop_duplicates()` function to delete all duplicated data.

Now, I want to see the statistical summary on this data by running this code:

```
workingDF.describe()
```



	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)
count	6282.000000	6282.000000	6282.000000	6282.000000	6282.000000	6282.000000	6282.000000
mean	3.161812	5.618911	12.610220	9.070583	11.017876	27.411016	251.157752
std	1.365201	1.846250	3.553066	2.278884	2.946876	7.245318	59.290426
min	0.900000	3.000000	4.200000	4.000000	4.100000	11.000000	96.000000
25%	2.000000	4.000000	10.100000	7.500000	8.900000	22.000000	208.000000
50%	3.000000	6.000000	12.100000	8.700000	10.600000	27.000000	246.000000
75%	3.700000	6.000000	14.700000	10.300000	12.700000	32.000000	289.000000
max	8.400000	16.000000	30.600000	20.600000	26.100000	69.000000	522.000000

- **Count** : Total number of observations in the dataset for each category (6282 in this case).
- **Mean** : The average value for each category.

- **Std** : Measures the amount of variation or dispersion of a set of values. A low standard deviation indicates that the values tend to be close to the mean, while a high standard deviation indicates that the values are spread out over a wider range.
- **Min** : The lowest value in each category.
- **25%** : 25% of the data is below this value. (Q1)
- **50%** : The middle value when the data set is ordered from lowest to highest. (Median)
- **75%** : 75% of the data is below this value. (Q3)
- **Max** : The highest value in each category.

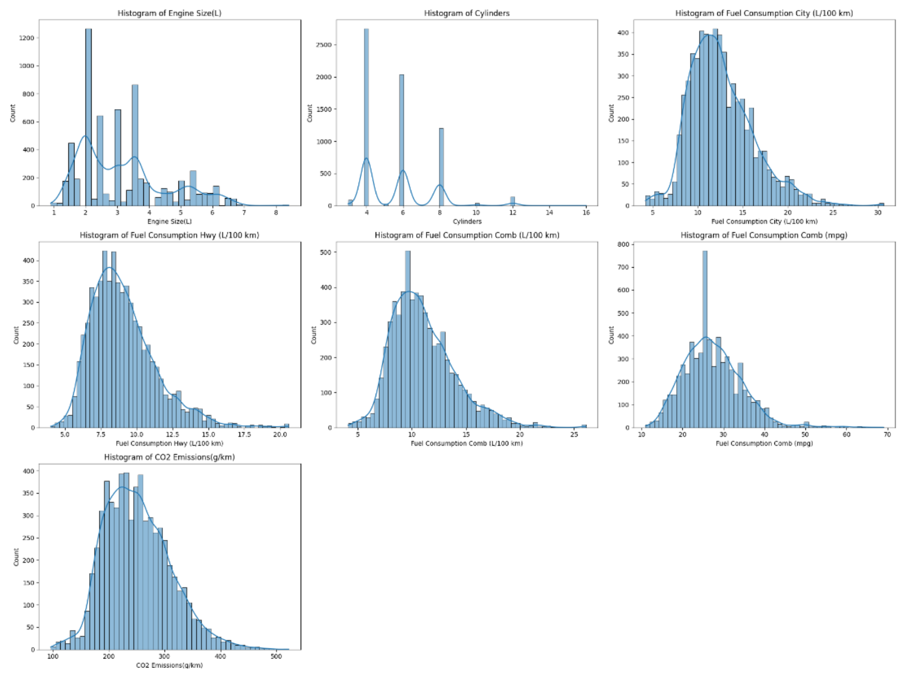
Now i want to show the histogram for each numerical columns, but firstly, I wanna create a list that contain column names of all numerical values

```
num_cols = ['Engine Size(L)', 'Cylinders', 'Fuel Consumption City (L/100 km)',
            'Fuel Consumption Hwy (L/100 km)', 'Fuel Consumption Comb   (L/100km)',
            'Fuel Consumption Comb (mpg)', 'CO2 Emissions(g/km)']
```

This num_cols will store all column names that are identified as numerical data. This list then will be used to show histogram for all numerical columns.

```
# Number of rows and columns for the subplot grid
n_rows = len(num_cols) // 3 + int(len(num_cols) % 3 != 0)
n_cols = 3

# Create subplots
fig, axes = plt.subplots(n_rows, n_cols, figsize=(20, n_rows * 5))

# Flatten the axes array for easy indexing
axes = axes.flatten()

# Plot each histogram
for i, col in enumerate(num_cols):
    sns.histplot(workingDF[col], kde=True, ax=axes[i])
    axes[i].set_title(f'Histogram of {col}')

# Remove empty subplots
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

plt.tight_layout()
plt.show()
```

Firstly, I set the plot to use a grid to show all plots in one cell, specifying the column is 3 plots for each row.



Histogram of Engine Size (L)

This histogram shows a right-skewed distribution, indicating that most vehicles have smaller engines, with fewer vehicles having larger engine sizes. The peak seems to be around the 2 to 3-liter engine size range.

Histogram of Cylinders

The distribution of cylinders among vehicles shows distinct peaks at 4, 6, and 8 cylinders, which are common configurations in passenger vehicles. There's a significant drop-off in vehicles with more than 8 cylinders, which are less common and typically found in high-performance or heavy-duty vehicles.

Histogram of Fuel Consumption City (L/100 km)

This histogram is also right-skewed, showing that most vehicles have lower fuel consumption in the city, with a peak around 10 to 15 L/100 km. The tail extends towards higher fuel consumption, indicating fewer vehicles with very high city fuel consumption.

Histogram of Fuel Consumption Hwy (L/100 km)

The highway fuel consumption shows a similar distribution to city consumption but shifted to the left, indicating overall lower fuel consumption on the highway. The peak is around 7.5 to 10 L/100 km.

Histogram of Fuel Consumption Comb (L/100 km)

The combined fuel consumption histogram is roughly symmetric with a slight right skew. The peak is between 8 and 12 L/100 km, suggesting that most vehicles are optimized for a balance between city and highway driving conditions.

Histogram of Fuel Consumption Comb (mpg)

This histogram shows the distribution of combined fuel consumption in miles per gallon. It is left-skewed, with a peak around 25 to 30 mpg. The tail extends to higher mpg values, which indicates that fewer vehicles achieve very high fuel efficiency.

Histogram of CO2 Emissions (g/km)

The CO2 emissions histogram is also right-skewed, with most vehicles emitting between 200 to 300 g/km. The tail to the right indicates fewer vehicles with very high emissions.

Insight

★ Most vehicles in the dataset have smaller engines and fewer cylinders, which generally correlate with lower fuel consumption and emissions.

★ The distributions of city and highway fuel consumption suggest that vehicles are more fuel-efficient on the highway.

★ There's a wider spread of values in city fuel consumption compared to the highway, possibly due to varying traffic conditions affecting city driving more significantly.

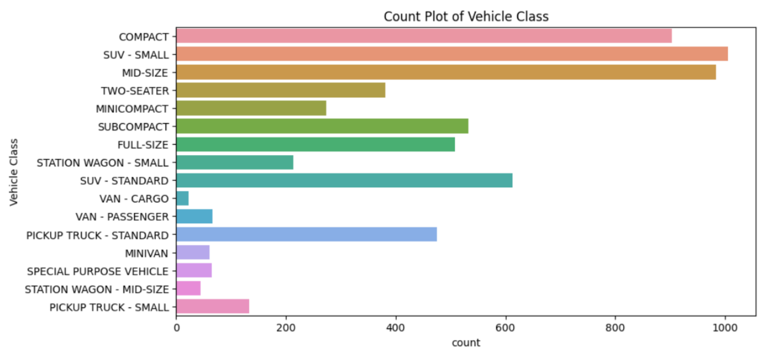
★ The combined fuel consumption and CO2 emissions histograms suggest that vehicles with mid-range engine sizes are most common, striking a balance between performance and efficiency.

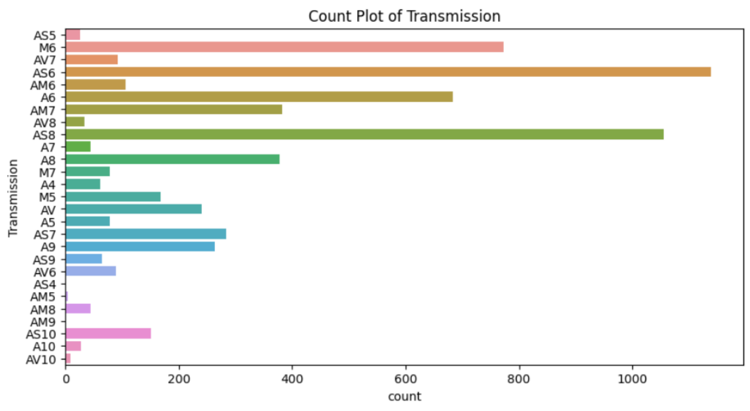
After analysed the numerical columns, now i want to see the plots for categorical columns using barplot, but for categorical columns, i only use ‘Vehicle Class’, ‘Transmission’, and ‘Fuel Type’, since those are the only columns that i will working on with

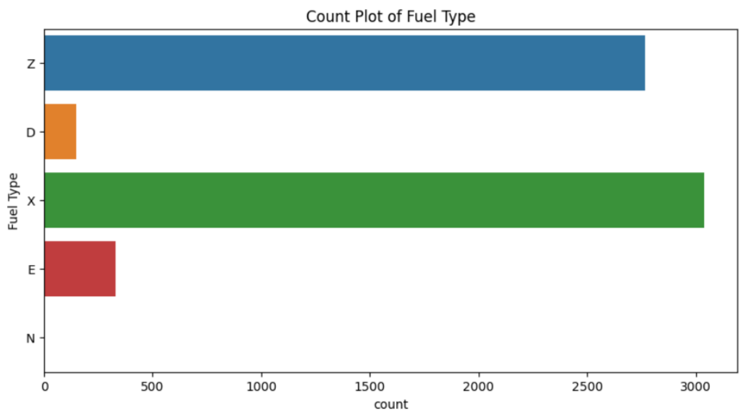
```
cat_cols = ['Vehicle Class', 'Transmission', 'Fuel Type']
```

Same as before, I created a list named ‘cat_cols’ to store the column names of relevant categorical columns. Here are the plots:

```
for col in cat_cols:
    plt.figure(figsize=(10, 5))
    sns.countplot(y=workingDF[col])
    plt.title(f'Count Plot of {col}')
    plt.show()
```







Count Plot of Vehicle Class

This plot shows the distribution of vehicles across different classes.
Compact vehicles have the highest count, indicating that they are the most common vehicle class in this dataset. SUVs, particularly small and standard sizes, are also common, which reflects a strong market presence or consumer preference for these types of vehicles. Special purpose vehicles, mid-size station wagons, and small pickup trucks have the fewest counts, suggesting these are less common or niche vehicle classes.

Count Plot of Transmission

The distribution of transmissions is categorized by type (Automatic - A, Manual - M) and the number of gears (5, 6, 7, etc.). Automatic transmissions with 6 speeds (A6) are the most common, followed closely by manual transmissions with 6 speeds (M6). There's a wide variety of transmission types, but those with 5 to 8 gears dominate the dataset. Transmissions with 10 gears (A10 and AM10) are the least common, which could indicate they are newer, more expensive, or less preferred options.

Count Plot of Fuel Type

This plot illustrates the types of fuel used by the vehicles in the dataset, with categories likely representing gasoline (X), diesel (D), electric (E), and other types (Z and N). Gasoline (X) is by far the most common fuel type, indicating a market dominated by gasoline-powered vehicles.Electric vehicles (E) have the lowest count, which may reflect the emerging status of electric vehicles in the market relative to traditional fuel types. Other fuel types (Z and N) have moderate counts, which could include hybrids or alternative fuels like LPG, CNG, etc.

Insights

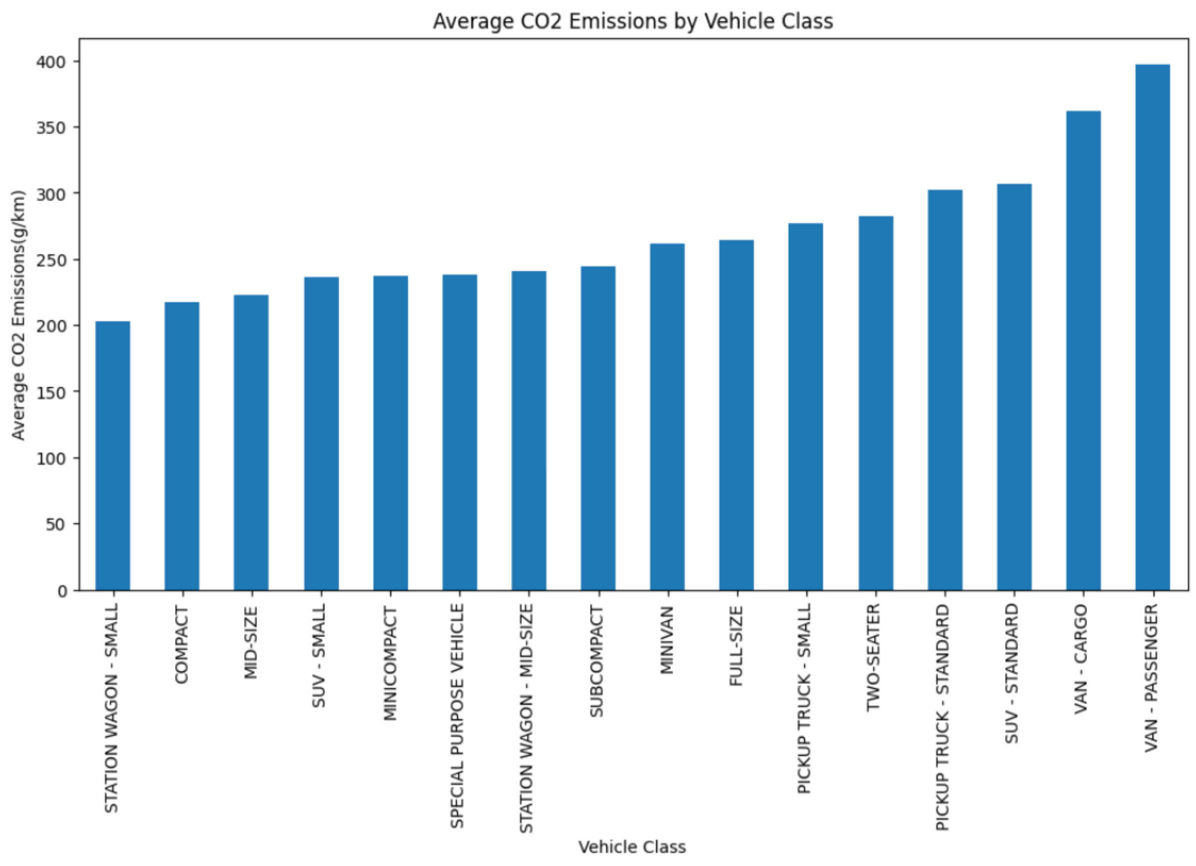
★ The vehicle class distribution suggests that compact cars are popular, possibly due to their affordability, fuel efficiency, or suitability for urban environments.

★ The prevalence of automatic transmissions, especially 6-speed, might indicate a preference for convenience and the widespread adoption of automatic gearboxes in modern vehicles.

★ The dominance of gasoline vehicles points to the established infrastructure and technology supporting this fuel type, whereas the low number of electric vehicles may reflect barriers to adoption like cost, range anxiety, or lack of charging infrastructure. (**note that this dataset is from 2017, the fuel type insight might not relevant anymore since electric vehicle is a thing now**)

Now i want to see the bar plot for average CO2 Emission for each Vehicle Class, lets analyze it!

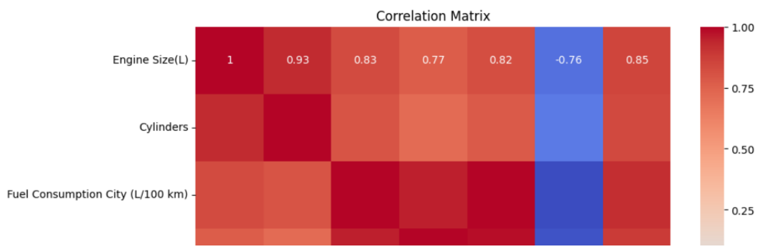
```
grouped_data = workingDF.groupby('Vehicle Class')['CO2 Emissions(g/km)'].mean()
plt.figure(figsize=(12, 6))
grouped_data.sort_values().plot(kind='bar')
plt.title('Average CO2 Emissions by Vehicle Class')
plt.ylabel('Average CO2 Emissions(g/km)')
plt.show()
```



The vehicle class is a significant factor in determining CO2 emissions, with larger or performance-oriented vehicles typically emitting more CO2. Environmental policies aimed at reducing emissions could target the classes with higher averages, such as "Van - Passenger" and "SUV - Standard." Manufacturers may focus on improving fuel efficiency in classes with higher emissions to meet regulatory standards and consumer demand for more environmentally friendly vehicles. Consumers looking to reduce their carbon footprint might consider purchasing vehicles from classes with lower average emissions, such as "Station Wagon - Small" or "Compact."

Lastly, i want to analyze the correlation between all numerical columns, lets analyze it!

```
plt.figure(figsize=(10, 8))
sns.heatmap(workingDF[num_cols].corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```



Engine Size and Cylinders (0.93): A very high positive correlation, suggesting that vehicles with larger engines tend to have more cylinders.
Engine Size and Fuel Consumption (City, Hwy, Comb): High positive correlations (ranging from 0.77 to 0.83), indicating that larger engines generally consume more fuel in all contexts.
Engine Size and CO2 Emissions (0.85): Another high positive correlation, indicating that larger engines are associated with higher CO2 emissions.
Cylinders and Fuel Consumption (City, Hwy, Comb): Similarly high correlations as with engine size, showing that more cylinders typically mean higher fuel consumption.
Cylinders and CO2 Emissions (0.82): A high positive correlation, meaning more cylinders contribute to higher emissions.
Fuel Consumption (City, Hwy, Comb) and CO2 Emissions: All show high positive correlations, which is expected since fuel consumption is directly related to CO2 emissions.
Fuel Consumption Comb (mpg) and all other factors: These show high negative correlations (especially with CO2 Emissions at -0.76), as the mpg is an inverse measure of consumption (higher mpg means lower fuel use and emissions).

Insights

- ★ Larger, more powerful engines with more cylinders tend to consume more fuel and produce more emissions.
- ★ Improving fuel efficiency, as indicated by higher miles per gallon (mpg), is associated with lower CO2 emissions.
- ★ There is a consistent pattern across all variables that suggests a trade-off between vehicle power/performance and fuel efficiency/emissions.

Feature Engineering

That's all for the data exploration, before jumping into model building, the data needed to be pre-processed, now I will implement feature engineering to encode categoricalvariables, creating the interaction features, binning the CO2 emission for classification task, and standarized the values using feature scaling.
Lets do it!

```
# Encoding Categorical Variables
categorical_cols = ['Vehicle Class', 'Transmission', 'Fuel Type']
workingDF_encoded = pd.get_dummies(workingDF, columns=categorical_cols)

# Creating Interaction Features
workingDF_encoded['EngineSize_Cylinders'] = workingDF['Engine Size(L)'] *
workingDF['Cylinders']

# Binning CO2 Emissions
bins = [0, 200, 300, max(workingDF['CO2 Emissions(g/km)'])]
labels = ['Low', 'Medium', 'High']
```

```
workingDF_encoded['CO2 Emissions Category'] = pd.cut(workingDF['CO2
Emissions(g/km)'], bins=bins, labels=labels)

# Feature Scaling
scaler = StandardScaler()
scaled_columns = ['Engine Size(L)', 'Cylinders', 'Fuel Consumption Comb (L/100 km)',
'CO2 Emissions(g/km)']
workingDF_encoded[scaled_columns] =
scaler.fit_transform(workingDF_encoded[scaled_columns])
```

Encoding Categorical Variables

- I converts categorical columns ('Vehicle Class', 'Transmission', 'Fuel Type') into dummy/indicator variables (one-hot encoding) using pd.get_dummies(). This is necessary because machine learning algorithms require numerical input.
- After encoding, each unique category value in the original columns will become a separate column in workingDF_encoded with a binary indicator of 0 or 1. (False - True)

Creating Interaction Features

- An interaction feature 'EngineSize_Cylinders' is created by multiplying the 'Engine Size(L)' and 'Cylinders' columns. This is based on the hypothesis that the interaction between these two features may have a significant effect on the target variable (possibly CO2 emissions or fuel efficiency).
- Interaction features can capture the combined effect of two variables that the individual variables might not show on their own.

Binning CO2 Emissions for classification

- CO2 emissions are categorized into bins representing 'Low', 'Medium', and 'High' emissions.
- The pd.cut() function is used to segment and sort the data values into these bins. The range of values is determined by the specified bins list, which uses 0 and 200 as lower bounds for 'Low' and 'Medium', 300 as the lower bound for 'High', and the maximum CO2 emissions value in the dataset as the upper bound for 'High'.

Feature Scaling

- The StandardScaler from a machine learning library is used to scale features. Scaling features to have zero mean and unit variance is important for many machine learning algorithms, as it ensures that all features contribute equally to the result and improves the convergence of algorithms.
- Only selected columns ('Engine Size(L)', 'Cylinders', 'Fuel Consumption Comb (L/100 km)', 'CO2 Emissions(g/km)') are scaled, likely because these are continuous variables and could have very different ranges.

The encoded dataset can be accessed here:
https://drive.google.com/file/d/1ezk6th2b2MfDCH2F1F-ZPDNxKUm-icma/view?usp=sharing

Regression Model

After performing the feature engineering, we can now proceed to the model building, firstly lets do the Regression task:

```
# Selecting relevant features for the model
selected_features = [
     "EngineSize_Cylinders",
    "Fuel Consumption City (L/100 km)",
    "Fuel Consumption Hwy (L/100 km)",
    "Fuel Consumption Comb (L/100 km)",
    "Fuel Consumption Comb (mpg)",
    # Transmission encoded
    "Transmission_AS5", "Transmission_M6","Transmission_AV7","Transmission_AS6",
"Transmission_AM6", "Transmission_A6", "Transmission_AM7",
    "Transmission_AV8", "Transmission_AS8", "Transmission_A7", "Transmission_A8",
"Transmission_M7", "Transmission_A4", "Transmission_M5",
    "Transmission_AV", "Transmission_A5", "Transmission_AS7", "Transmission_A9",
"Transmission_AS9", "Transmission_AV6", "Transmission_AS4",
    "Transmission_AM5", "Transmission_AM8", "Transmission_AM9", "Transmission_AS10",
"Transmission_A10", "Transmission_AV10",
    # FuelType encoded
    "Fuel Type_Z", "Fuel Type_D", "Fuel Type_X", "Fuel Type_E", "Fuel Type_N",
    # Vechicle Class encoded
    "Vehicle Class_COMPACT" , "Vehicle Class_SUV - SMALL" , "Vehicle Class_MID-SIZE",
"Vehicle Class_TWO-SEATER", "Vehicle Class_MINICOMPACT",
    "Vehicle Class_SUBCOMPACT", "Vehicle Class_FULL-SIZE", "Vehicle Class_STATION WAGON
- SMALL",
    "Vehicle Class_SUV - STANDARD", "Vehicle Class_VAN - CARGO", "Vehicle Class_VAN -
PASSENGER",
    "Vehicle Class_PICKUP TRUCK - STANDARD", "Vehicle Class_MINIVAN", "Vehicle
Class_SPECIAL PURPOSE VEHICLE",
     "Vehicle Class_STATION WAGON - MID-SIZE", "Vehicle Class_PICKUP TRUCK - SMALL"
]

# Target variable
target = "CO2 Emissions(g/km)"

# Separating the features and the target variable
X = workingDF_encoded[selected_features]
y = workingDF_encoded[target]

# Splitting the dataset into training (80%) and testing (20%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

Selecting relevant features for the model

I created a list of features selected_features that are relevant for regression model, which includes:

- EngineSize_Cylinders: The interaction term from earlier preprocessing.
- Different measures of fuel consumption: These features will likely have strong predictive power for CO2 emissions since fuel consumption directly impacts emission levels.

- Encoded categorical variables: Various dummy variables for transmission type, fuel type, and vehicle class. Each of these has been previously one-hot encoded to turn categorical data into numerical format suitable for modeling.

Target variable

- The target variable is set to "CO2 Emissions(g/km)", which is the outcome the model aims to predict.

Separating features and the target variable

- The dataset workingDF_encoded is divided into a set of features X (the independent variables) and the target y (the dependent variable).

Splitting the dataset into training and testing sets

The train_test_split function is used to split X and y into training and testing sets. The test_size=0.2 argument means that 20% of the data will be used as a test set, and the remaining 80% will be used for training the model. random_state=42 is set to ensure reproducibility. This means that the random number generator that controls the shuffling and partitioning of the data will produce the same results each time the code is run.

After selecting relevant features for target and predictor, and also splitting the dataset, lets now proceed to train the regression models (Decision Tree Regressor, KNN Regressor, and Linear Regression)
I will give the code for all models first, then I will give the conclusion of performance metrics later on after all regression models are printed out.

Decision Tree Regressor

```
# Training the Decision Tree Regressor model
dt_reg = DecisionTreeRegressor(random_state=42)
dt_reg.fit(X_train, y_train)

# Predicting on the test set
y_pred_dt = dt_reg.predict(X_test)

# Calculating evaluation metrics
rmse_DT = np.sqrt(mean_squared_error(y_test, y_pred_dt))
r2_DT = r2_score(y_test, y_pred_dt)
```

KNN Regressor

```
# Training the KNN Regressor model
knn_reg = KNeighborsRegressor()
knn_reg.fit(X_train, y_train)

# Predicting on the test set
y_pred_knn = knn_reg.predict(X_test)
```

```
# Calculating evaluation metrics
rmse_KNN = np.sqrt(mean_squared_error(y_test, y_pred_knn))
r2_KNN = r2_score(y_test, y_pred_knn)
```

Linear Regression

```
# Predicting on the test set
y_pred_lr = model.predict(X_test)

# Calculating evaluation metrics
rmse_lr = np.sqrt(mean_squared_error(y_test, y_pred_lr))
r2_lr = r2_score(y_test, y_pred_lr)
```

Regression Model Performance

- Decision Tree Regressor

  ```
  RMSE          : 0.06701629048607645
  R-squared     : 0.9956257118949741
  ```

  RMSE of **0.067**: This value is quite low, indicating that the model's predictions are very close to the actual values. A lower RMSE is always desirable as it signifies less prediction error.

  R-squared of **0.9956**: This is an excellent score, reflecting that the model explains about 99.56% of the variance in the dependent variable (CO2 emissions). It suggests a very high level of fit between the model and the data.

- KNN Regressor

  ```
  RMSE          : 0.11228009740048296
  R-squared     : 0.9877213060300837
  ```

  RMSE of **0.112**: This is higher than the Decision Tree Regressor, indicating slightly less accuracy in predictions. However, it's still a good score, especially considering the complexity of real-world data.

  R-squared of **0.9877**: This score is also high, showing that the model explains approximately 98.77% of the variance. While it's slightly lower than the Decision Tree, it's still indicative of a strong model.

- Linear Regression

  ```
  RMSE          : 0.09361229005230334
  R-squared     : 0.9914648265131405
  ```

  RMSE of **0.0936**: This is lower than the KNN but higher than the Decision Tree, placing it in the middle in terms of prediction accuracy among the three models.

R-squared of **0.9915**: This score is very close to that of the Decision Tree, suggesting that Linear Regression is also a very effective model for this dataset, explaining around 99.15% of the variance.

Conclusion

- Model Suitability
  The Decision Tree Regressor seems to be the most suitable model for this dataset with the lowest RMSE and highest R-squared.
- Dataset Complexity
  Given the nature of the dataset, all three models perform exceptionally well. This suggests that the features in the dataset have a strong and clear relationship with the CO2 emissions, making it relatively easier for models to make accurate predictions.

Classification Model

After getting the model performance for regression, now i want to proceed to the classification task, for classification, i already set the target variable as 'CO2 Emissions Category' that already created before on the feature engineering progress.

```
# Target variable for classification
target_classification = "CO2 Emissions Category"

# Separating the features and the target variable for classification
y_classification = workingDF_encoded[target_classification]

# Splitting the dataset into training (80%) and testing (20%) sets for
classification
X_train_class, X_test_class, y_train_class, y_test_class = train_test_split(
    X, y_classification, test_size=0.2, random_state=42
)
```

Target variable for classification

- target_classification is assigned the name of the column in workingDF_encoded that contains the categorical target variable for the classification task, "CO2 Emissions Category".
- This column was previously created by binning continuous CO2 emissions data into categories.

Separating features and target variable for classification

- y_classification is set to the workingDF_encoded DataFrame's target variable column, storing the CO2 emissions categories for each observation in the dataset

  .

Splitting the dataset into training and testing sets for classification

- The train_test_split function is used to divide the dataset into feature sets (X_train_class, X_test_class) and target sets (y_train_class, y_test_class) for the training and testing phases, respectively.

- The test_size=0.2 parameter specifies that 20% of the data should be reserved for the test set, which is a standard practice to evaluate the model's performance on unseen data.
- The random_state=42 ensures that the split will be reproducible; that is, running this code multiple times will always produce the same split, which is important for debugging and for others to replicate the study.

After setting the target variable, separating the feature and target, and splitting the dataset, we can now proceed to train the classification models (Decision Tree Classifier, Logistic Regression (Binary/Multinomial, KNN Classifier, Naive bayes)

Decision Tree Classifier

```
# Training the Decision Tree Classifier
dt_class = DecisionTreeClassifier(random_state=42)
dt_class.fit(X_train_class, y_train_class)

# Predicting on the test set
y_pred_dt_class = dt_class.predict(X_test_class)

# Calculating classification metrics
accuracy_DTC =  accuracy_score(y_test_class, y_pred_dt_class)
classification_report_result_DTC = classification_report(y_test_class,
y_pred_dt_class)
```

Logistic Regression (Binary / Multinomial)

```
# Training the Logistic Regression Classifier
log_reg = LogisticRegression(max_iter=1000, random_state=42)
log_reg.fit(X_train_class, y_train_class)

# Predicting on the test set
y_pred_log = log_reg.predict(X_test_class)

# Calculating classification metrics
accuracy_log  = accuracy_score(y_test_class, y_pred_log)
classification_report_result_log = classification_report(y_test_class,
y_pred_log)
```

KNN Classifier

```
# Training the KNN Classifier
knn_class = KNeighborsClassifier()
knn_class.fit(X_train_class, y_train_class)

X_test_classKNNCLS = np.ascontiguousarray(X_test_class)
# Predicting on the test set
y_pred_knn_class = knn_class.predict(X_test_classKNNCLS)

# Calculating classification metrics
```

```
accuracy_KNNCLS = accuracy_score(y_test_class, y_pred_knn_class)
classfication_report_result_KNNCLS = classification_report(y_test_class,
y_pred_knn_class)
```

Naive Bayes

```
# Training the Naive Bayes Classifier
nb_class = GaussianNB()
nb_class.fit(X_train_class, y_train_class)

# Predicting on the test set
y_pred_nb = nb_class.predict(X_test_class)

# Calculating classification metrics
accuracy_nb = accuracy_score(y_test_class, y_pred_nb)
classifaction_report_result_nb = classification_report(y_test_class, y_pred_nb)
```

Classification Model Performance

- Decision Tree Classifier

```
Decision Tree Classifier Accuracy            :  0.9832935560859188
Decision Tree Classifier Classification Report      :
              precision    recall  f1-score   support

        High       0.97      0.97      0.97       245
         Low       0.99      0.98      0.99       263
      Medium       0.99      0.99      0.99       749

    accuracy                           0.98      1257
   macro avg       0.98      0.98      0.98      1257
weighted avg       0.98      0.98      0.98      1257
```

Accuracy 98.33%: This is a very high accuracy, indicating excellent overall performance. **Classification Report**: Shows high precision, recall, and F1-score across all classes (High, Low, Medium), with almost equal performance for each class. This indicates the model is not only accurate overall but also balanced in classifying each category.

- Logistic Regression (Binary / Multinomial)

```
Logistic Regression Accuracy                 :  0.9673826571201273
Logistic Regression Classification Report    :
              precision    recall  f1-score   support

        High       0.96      0.95      0.95       245
         Low       0.96      0.97      0.96       263
      Medium       0.97      0.97      0.97       749
```

```
    accuracy                           0.97      1257
   macro avg       0.96      0.96      0.96      1257
weighted avg       0.97      0.97      0.97      1257
```

Accuracy 96.74%: Slightly lower than the Decision Tree but still very high.
Classification Report: Consistent precision, recall, and F1-scores across all classes, though slightly lower than the Decision Tree. This model also shows a balanced performance across different classes.

- KNN Classifier

```
KNN Classifier Accuracy                 :  0.9681782020684169
KNN Classifier Classification Report    :
               precision    recall  f1-score   support

        High       0.96      0.96      0.96       245
         Low       0.96      0.97      0.97       263
      Medium       0.97      0.97      0.97       749

    accuracy                           0.97      1257
   macro avg       0.96      0.97      0.96      1257
weighted avg       0.97      0.97      0.97      1257
```

Accuracy 96.82%: Comparable to Logistic Regression, indicating strong performance.
Classification Report: Similar to Logistic Regression, KNN shows high and balanced scores across precision, recall, and F1-score for all classes, confirming its effectiveness in classifying each category accurately.

- Naive Bayes

```
Naive Bayes Classifier Accuracy                 :  0.45584725536992843
Naive Bayes Classifier CLassification Report    :
               precision    recall  f1-score   support

        High       0.38      0.97      0.55       245
         Low       0.47      0.97      0.63       263
      Medium       0.88      0.11      0.19       749

    accuracy                           0.46      1257
   macro avg       0.58      0.68      0.46      1257
weighted avg       0.70      0.46      0.35      1257
```

Accuracy 45.58%: Significantly lower than the other models, indicating a poor fit for this dataset.
Classification Report: Shows a very high precision for the 'Medium' class but extremely low recall, indicating it correctly identifies a small fraction of actual 'Medium' cases. Conversely, it has high recall but lower precision for 'High' and 'Low', meaning it over-classifies instances into these categories.

Conclusion

- **Model Efficacy**: The Decision Tree Classifier emerges as the most effective model for this dataset, with Logistic Regression and KNN following closely. Their high accuracy and balanced classification reports suggest that they are suitable for this task.
- **Naive Bayes' Limitation**: The poor performance of Naive Bayes could be due to its underlying assumption of feature independence, which might not hold true in this dataset. It also might not be capturing the relationships between features and classes as effectively as the other models.
- **Model Selection**: Decision Trees are more interpretable, Logistic Regression is a robust and well-understood model, and KNN is effective but can be computationally intensive.

Model Comparison

Regression Models	Classification Models

● Decision Tree Regressor
`RMSE         : 0.067`
`R-squared    : 0.995`
 	
● KNN Regressor
`RMSE         : 0.112`
`R-squared    : 0.987`
 	
● Linear Regression
`RMSE         : 0.093`
`R-squared    : 0.991`
	● Decision Tree Classifier
	`Accuracy  : 0.98`
	`Precision : 0.98`
	`Recall    : 0.98`
	`F1-score  : 0.98`
	● Logistic Regression (Binary / Multinomial)
	`Accuracy  : 0.97`
	`Precision : 0.96`
	`Recall    : 0.96`
	`F1-score  : 0.96`
	● KNN Classifier
	`Accuracy  : 0.97`
	`Precision : 0.96`
	`Recall    : 0.97`
	`F1-score  : 0.96`
	● Naive Bayes
	`Accuracy  : 0.46`
	`Precision : 0.58`
	`Recall    : 0.68`
	`F1-score  : 0.46`

Conclusion

- The regression models are performing exceptionally well, with high R-squared values indicating that a significant portion of the variance in the CO2 emissions data is being captured by the models. The decision tree regressor is the best-performing model among the regression techniques, suggesting that the relationship between the features and the CO2 emissions is quite complex and possibly non-linear.

- On the classification side, except for Naive Bayes, the models are showing high performance across all metrics. The Decision Tree Classifier and Logistic Regression are particularly effective, indicating that the features provide strong indicators for classifying the emissions into discrete categories which correspond to different levels of emissions on the binning process.

- The poor performance of the Naive Bayes classifier could be due to the assumption of feature independence in the model, which is likely violated in this dataset. The features that determine CO2 emissions are likely correlated (e.g., engine size, fuel types, etc.), which would negatively impact the Naive Bayes performance.

- Overall, the dataset appears to be well-suited for both regression and classification tasks, with the exception of Naive Bayes. The strong performance of the other models suggests that the dataset contains meaningful patterns that these models are able to learn and make accurate predictions from.