CO_2 emission by vehicles

CO_2 emissions from vehicles are a significant contributor to global warming and climate change. The widespread use of internal combustion engine (ICE) vehicles powered by gasoline and diesel fuels has led to substantial CO_2 emissions, affecting air quality and public health. Various factors, including driving habits, fuel type, and vehicle efficiency, influence the emission of high amounts of CO_2. Understanding these factors is crucial to developing effective strategies to mitigate the impact of vehicle emissions on the environment and climate.

Objective

The objective of this project is to analyze the factors influencing high CO_2 emissions from vehicles, develop predictive models to assess and identify vehicles emitting high amounts of CO_2 and propose strategies for mitigating these emissions. This aims to contribute towards addressing global warming and climate change by promoting more sustainable transportation practices.

There are a few abbreviations that have been used to describe the

features.

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 liters per 100 kilometers (L/100 km) - the combined rating (55% city, 45% highway) is shown in L/100 km and in miles per gallon (mpg)

CO_2 Emissions

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

Importing the nescessary libraries

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder,MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression,Lasso,Ridge
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import GradientBoostingRegressor
from xgboost import XGBRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV
```

In [ ]:
```
df=pd.read_csv('/content/drive/MyDrive/Dataset/CO2_Emissions_Canada[1].csv')
```

In [ ]:
```
df.head()
```

Out[ ]:

	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

EDA

In [ ]:
```
df.describe()
```

Out[ ]:

	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)	Emissi
count	7385.000000	7385.000000	7385.000000	7385.000000	7385.000000	7385.000000	73
mean	3.160068	5.615030	12.556534	9.041706	10.975071	27.481652	2
std	1.354170	1.828307	3.500274	2.224456	2.892506	7.231879	
min	0.900000	3.000000	4.200000	4.000000	4.100000	11.000000	
25%	2.000000	4.000000	10.100000	7.500000	8.900000	22.000000	2
50%	3.000000	6.000000	12.100000	8.700000	10.600000	27.000000	2
75%	3.700000	6.000000	14.600000	10.200000	12.600000	32.000000	2
max	8.400000	16.000000	30.600000	20.600000	26.100000	69.000000	5

In [ ]:

```
df.shape
```

Out[ ]:

```
(7385, 12)
```

Checking for missing values

In [ ]:

```
df.isna().sum()
```

Out[ ]:

```
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
dtype: int64
```

Checking for duplicates

In [ ]:

```
df.duplicated().sum()
```

Out[ ]:

```
1103
```

In [ ]:

```
df.drop_duplicates(inplace=True)
df
```

Out[ ]:

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

6282 rows × 12 columns

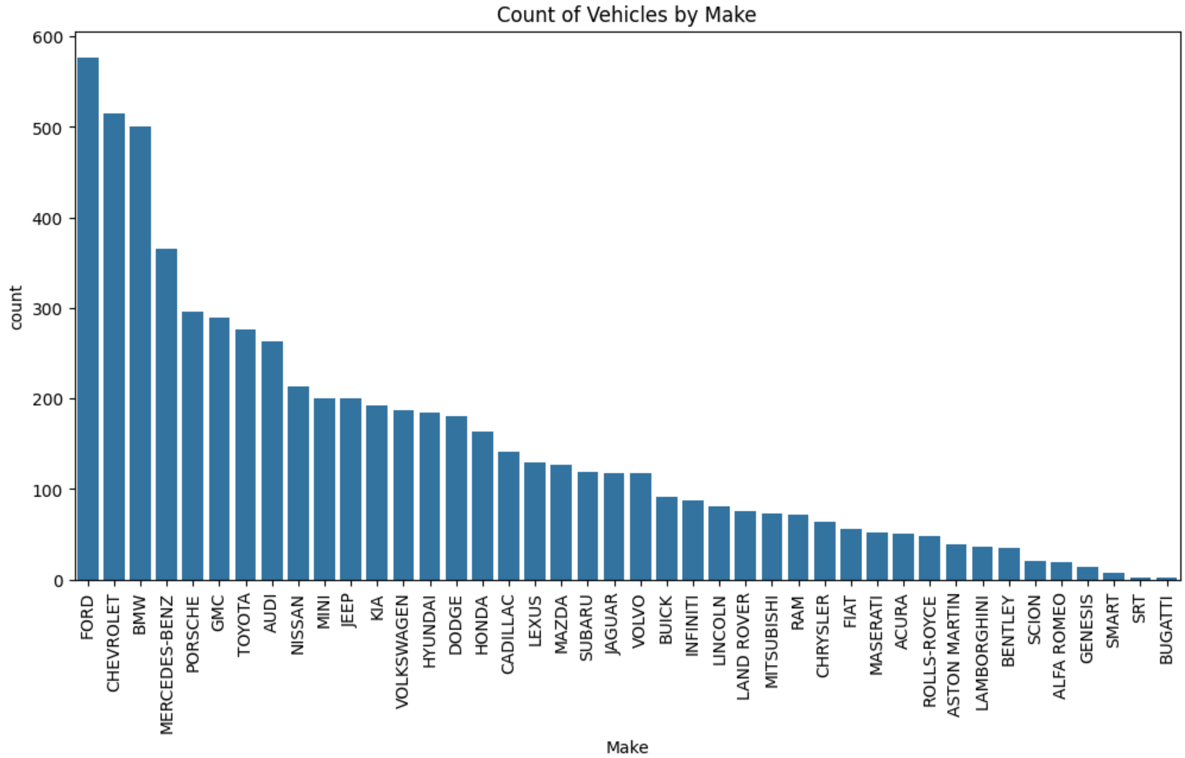
In [ ]:
```
df.duplicated().sum()
```

Out[ ]:
```
0
```

In [ ]:
```
# df['Make'].value_counts()
```

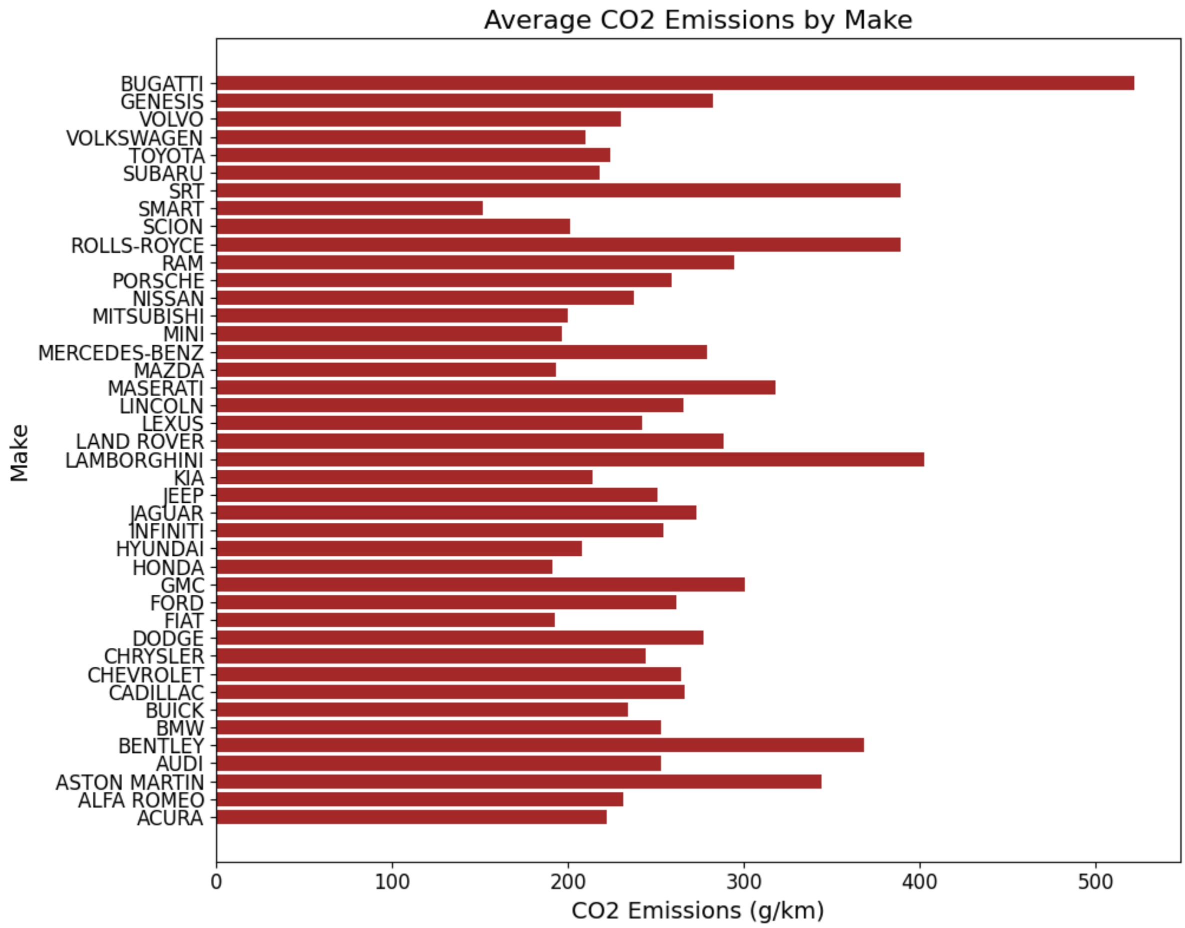
Univariate,Bivariate and multivariate analysis

In [ ]:
```
plt.figure(figsize=(12, 6))
sns.countplot(data=df, x='Make', order=df['Make'].value_counts().index)
plt.title('Count of Vehicles by Make')
plt.xticks(rotation=90)
plt.show()
```



In [ ]:

```
make=df['Make'].unique()
co2_means=[]
for i in make:
  co2_mean=df[df['Make']==i]['CO2 Emissions(g/km)'].mean()
  co2_means.append(co2_mean)
plt.figure(figsize=(10, 8))
plt.barh(make, co2_means, color='brown')
plt.title('Average CO2 Emissions by Make', fontsize=16)
plt.xlabel('CO2 Emissions (g/km)', fontsize=14)
plt.ylabel('Make', fontsize=14)
plt.tight_layout()
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.show()
```



In [ ]:
```
df['Model'].value_counts()
```

Out[ ]:
```
Model
F-150 FFV             32
F-150 FFV 4X4         31
MUSTANG               27
FOCUS FFV             24
F-150 4X4             20
                      ..
LS 500                 1
LS 500h                1
NX 300 AWD F SPORT     1
RX 350 L AWD           1
XC40 T4 AWD            1
Name: count, Length: 2053, dtype: int64
```

In [ ]:
```
df['Vehicle Class'].value_counts()
```

Out[ ]:

```
Vehicle Class
SUV - SMALL                 1006
MID-SIZE                     983
COMPACT                      903
SUV - STANDARD               613
SUBCOMPACT                   533
FULL-SIZE                    508
PICKUP TRUCK - STANDARD      475
TWO-SEATER                   381
MINICOMPACT                  274
STATION WAGON - SMALL        214
PICKUP TRUCK - SMALL         133
VAN - PASSENGER               66
SPECIAL PURPOSE VEHICLE       65
MINIVAN                       61
STATION WAGON - MID-SIZE      45
VAN - CARGO                   22
Name: count, dtype: int64
```

In [ ]:

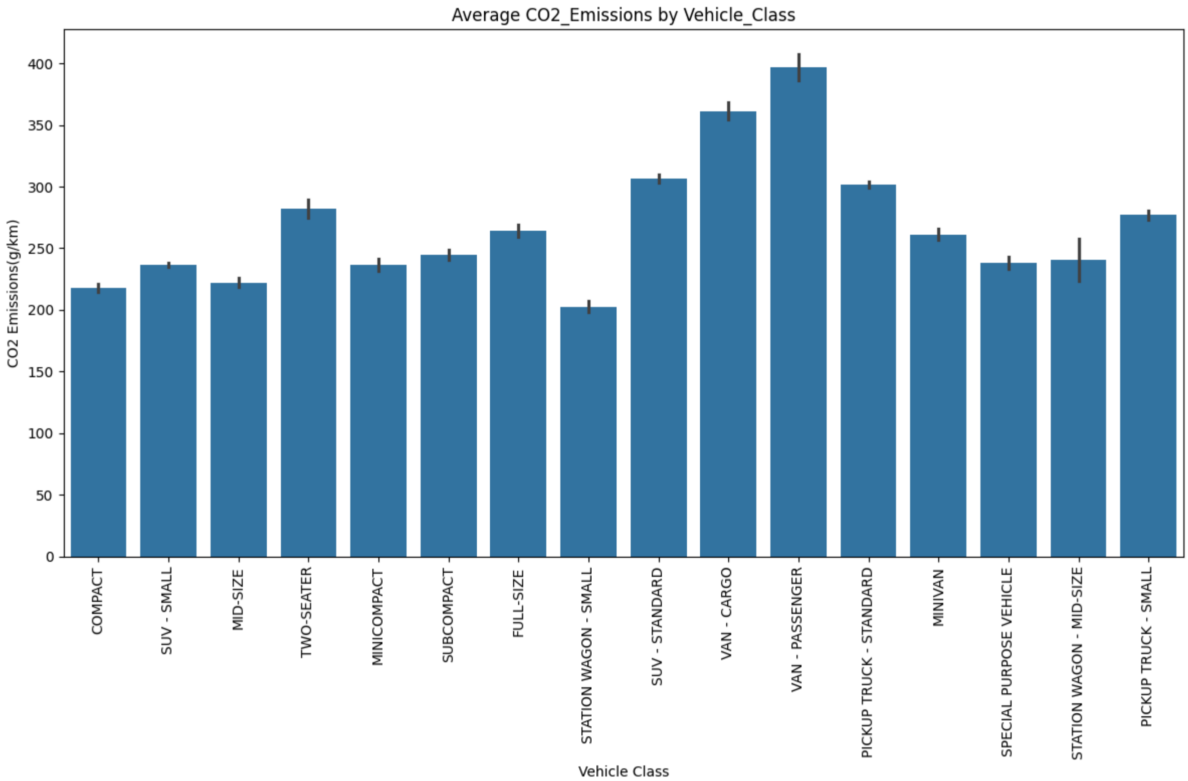
```
df['Vehicle Class'].unique()
```

Out[ ]:

```
array(['COMPACT', 'SUV - SMALL', 'MID-SIZE', 'TWO-SEATER', 'MINICOMPACT',
       'SUBCOMPACT', 'FULL-SIZE', 'STATION WAGON - SMALL',
       'SUV - STANDARD', 'VAN - CARGO', 'VAN - PASSENGER',
       'PICKUP TRUCK - STANDARD', 'MINIVAN', 'SPECIAL PURPOSE VEHICLE',
       'STATION WAGON - MID-SIZE', 'PICKUP TRUCK - SMALL'], dtype=object)
```

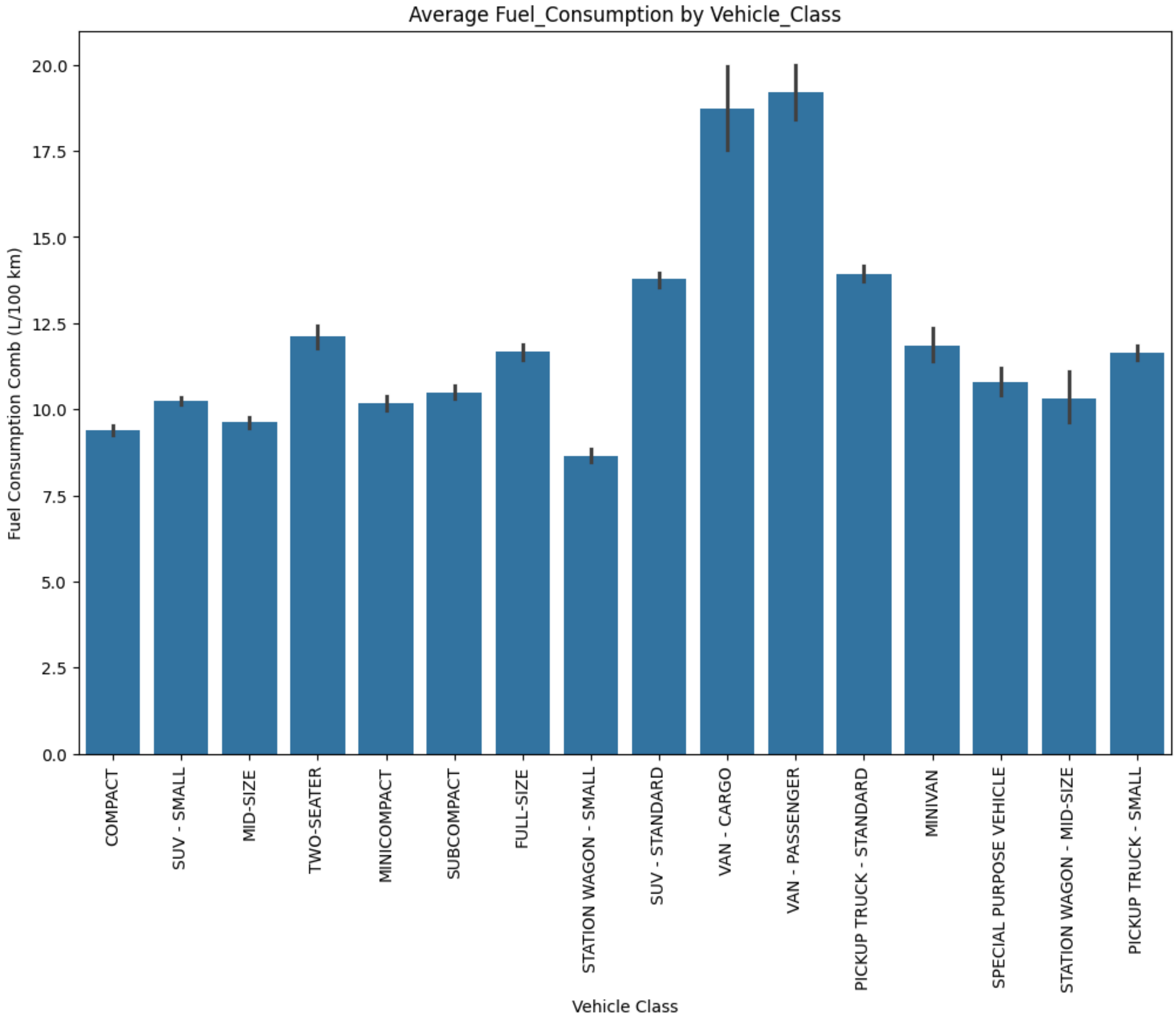
In [ ]:

```
plt.figure(figsize=(12,8))
sns.barplot(data=df, x='Vehicle Class', y='CO2 Emissions(g/km)')
plt.title('Average CO2_Emissions by Vehicle_Class')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



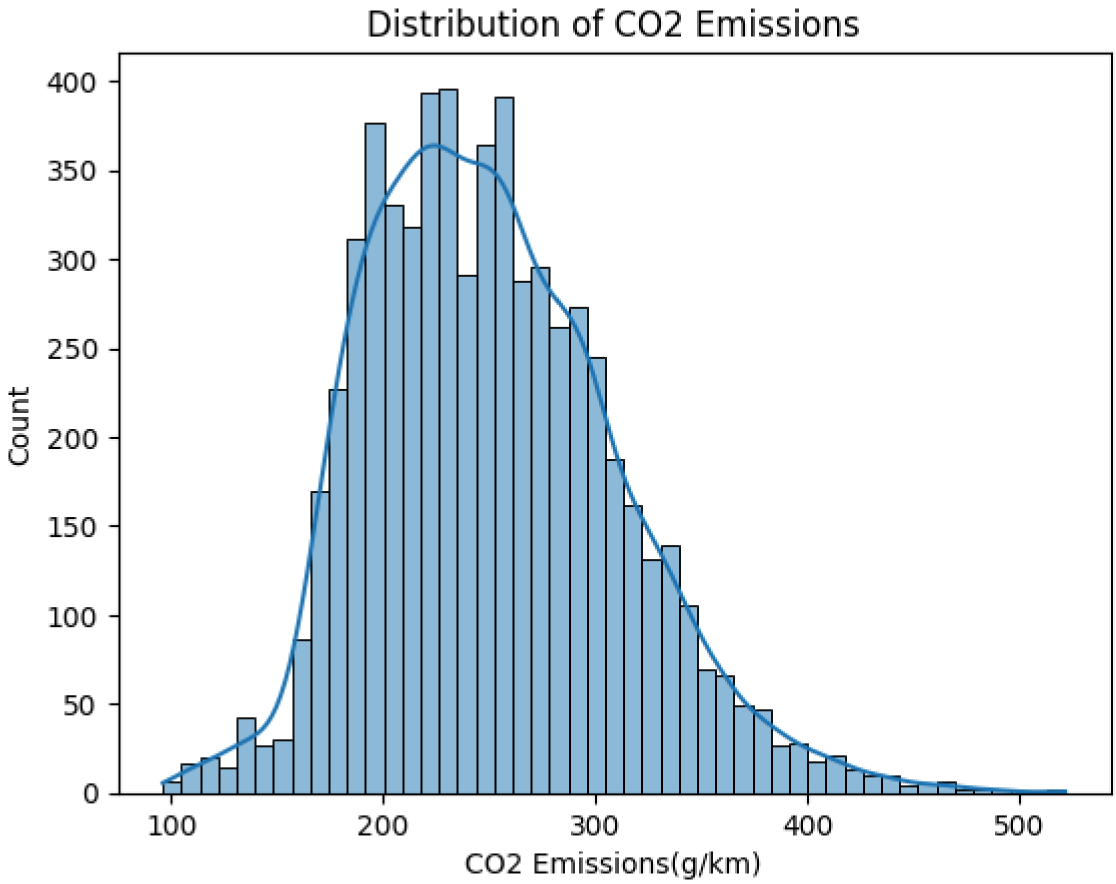
In [ ]:

```
plt.figure(figsize=(12, 8))
sns.barplot(data=df, x='Vehicle Class', y='Fuel Consumption Comb (L/100 km)')
plt.title('Average Fuel_Consumption by Vehicle_Class')
plt.xticks(rotation=90)
plt.show()
```



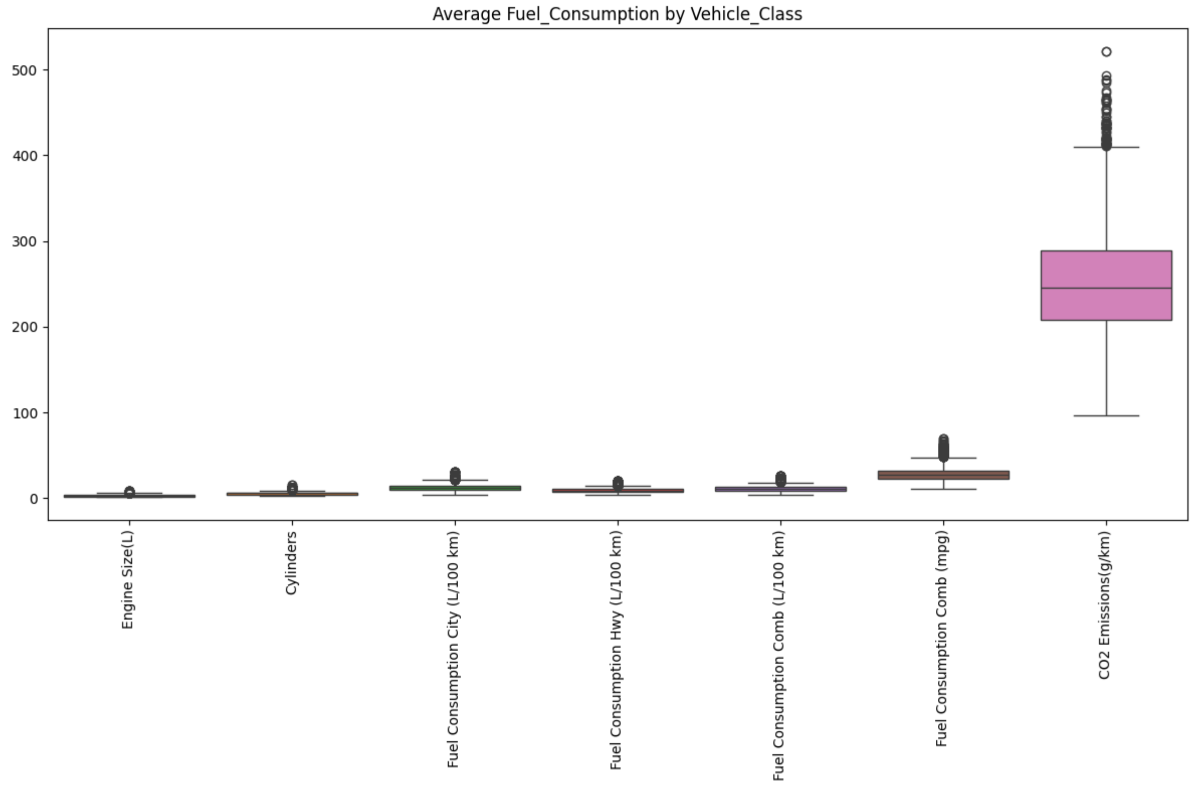
In [ ]:

```
sns.histplot(data=df, x='CO2 Emissions(g/km)', kde=True)
plt.title('Distribution of CO2 Emissions')
plt.show()
```

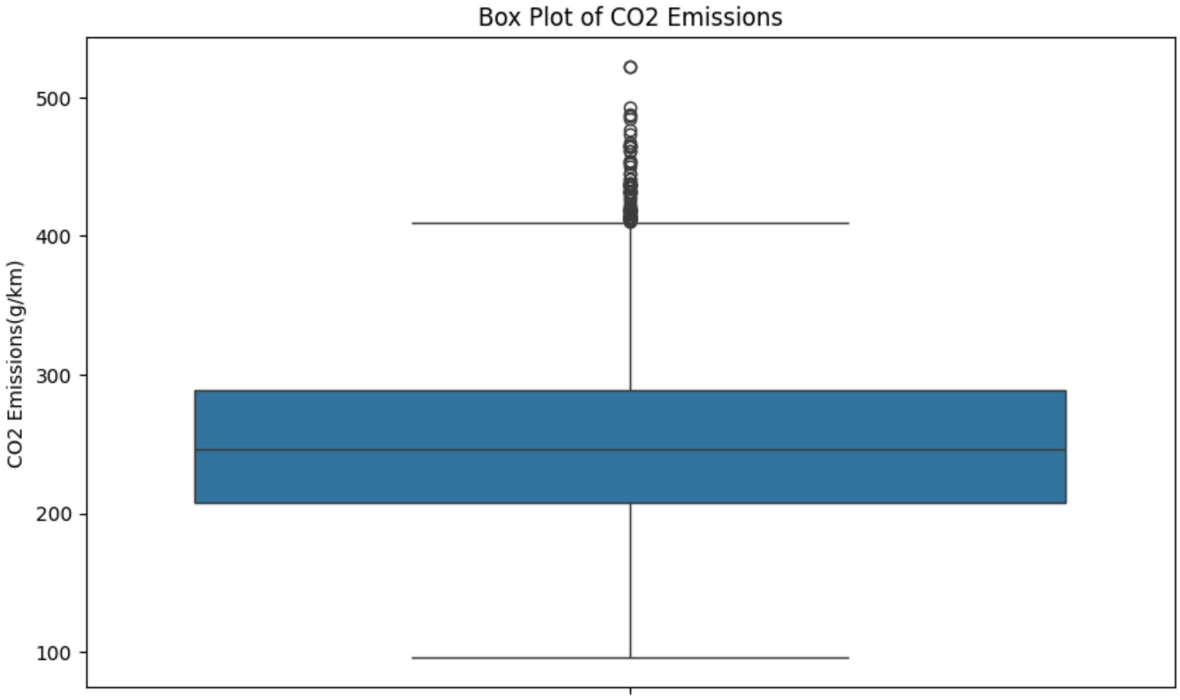


Detection of Outliers

```
In [ ]: plt.figure(figsize=(12, 8))
        sns.boxplot(data=df)
        plt.title('Average Fuel_Consumption by Vehicle_Class')
        plt.xticks(rotation=90)
        plt.tight_layout()
        plt.show()
```



In [ ]:
```python
plt.figure(figsize=(10, 6))
sns.boxplot(y=df['CO2 Emissions(g/km)'])
plt.title('Box Plot of CO2 Emissions')
plt.show()
```



In [ ]:
```python
Q1 = df['CO2 Emissions(g/km)'].quantile(0.25)
Q3 = df['CO2 Emissions(g/km)'].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# outliers=df[(df['CO2 Emissions(g/km)']<lower_bound) | (df['CO2 Emissions(g/km)']>
# outliers
```

In [ ]:
```python
df= df[(df['CO2 Emissions(g/km)'] >= lower_bound) & (df['CO2 Emissions(g/km)'] <= u
df
```

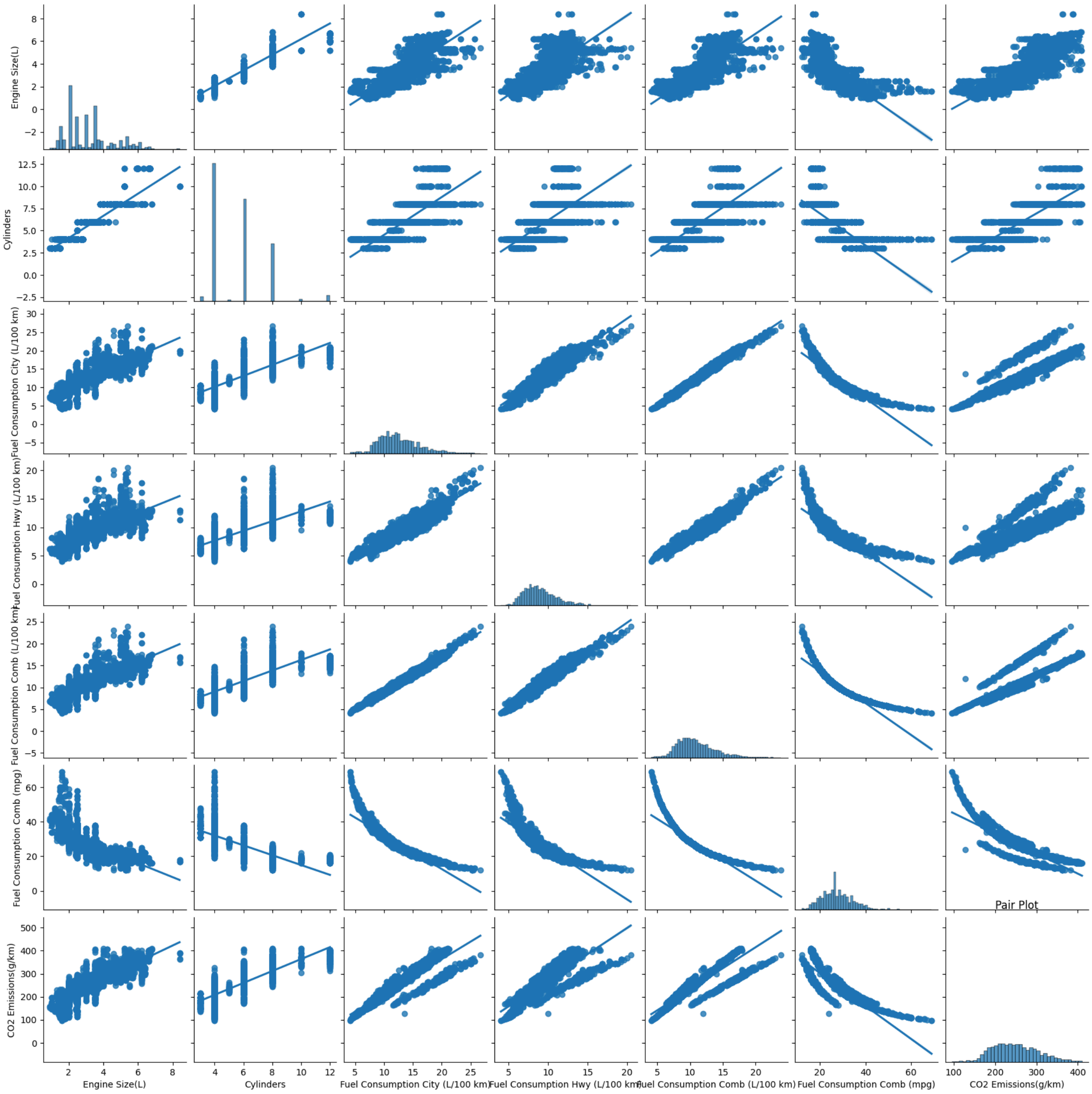
Out[ ]:

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

6208 rows × 12 columns

In [ ]:

```
sns.pairplot(df,kind='reg')
plt.title('Pair Plot', y=1.02)
plt.show()
```




```
In [ ]: df['Transmission'].value_counts()
```

Out[ ]:

```
Transmission
AS6      1138
AS8      1052
M6        773
A6        648
A8        376
AM7       365
AS7       276
A9        263
AV        241
M5        168
AS10      151
AM6       107
AV7        92
AV6        89
A5         77
M7         77
AS9        65
A4         60
AM8        45
A7         41
AV8        34
A10        28
AS5        26
AV10        9
AM5         4
AS4         2
AM9         1
Name: count, dtype: int64
```

In [ ]:

```
df['Transmission'].unique()
```

Out[ ]:

```
array(['AS5', 'M6', 'AV7', 'AS6', 'AM6', 'A6', 'AM7', 'AV8', 'AS8', 'A7',
       'A8', 'M7', 'A4', 'M5', 'AV', 'A5', 'AS7', 'A9', 'AS9', 'AV6',
       'AS4', 'AM5', 'AM8', 'AM9', 'AS10', 'A10', 'AV10'], dtype=object)
```

In [ ]:

```
def categorize_transmission(transmission):
    if transmission in ['AV7', 'AV6', 'AV8', 'AV', 'AV10', 'AM5', 'AM6', 'AM7', 'AM
        return 'Automated Manual'
    if transmission in ['AS6', 'AS8', 'AS9', 'AS10', 'AS4', 'AS7', 'AS5']:
        return 'Automatic'
    if transmission in ['A5', 'A6', 'A7', 'A8', 'A9', 'A10', 'A4', 'M6', 'M7', 'M5'
        return 'Manual'
    # else:
        # return 'Unknown'
df['Transmission_Category'] = df['Transmission'].apply(categorize_transmission)
```

```
<ipython-input-26-9849cac08ab3>:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['Transmission_Category'] = df['Transmission'].apply(categorize_transmission)
```

In [ ]:

```
def categorize_vehicle_class(vehicle_class):
    suvs=['SUV - SMALL', 'SUV - STANDARD']
    cars=['MID-SIZE', 'COMPACT', 'SUBCOMPACT', 'FULL-SIZE', 'TWO-SEATER', 'MINICOMP
    trucks=['PICKUP TRUCK - STANDARD', 'PICKUP TRUCK - SMALL']
    others=['VAN - PASSENGER', 'SPECIAL PURPOSE VEHICLE', 'MINIVAN', 'VAN - CARGO']

    if vehicle_class in suvs:
        return 'SUVs'
    elif vehicle_class in cars:
```

```
        return 'Cars'
    elif vehicle_class in trucks:
        return 'Trucks'
    else:
        return 'Others'
df['Vehicle Class Category'] = df['Vehicle Class'].apply(categorize_vehicle_class)
```

```
<ipython-input-27-5c31f26e50a7>:15: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['Vehicle Class Category'] = df['Vehicle Class'].apply(categorize_vehicle_clas
s)
```

In [ ]:
```
df
```

Out[ ]:

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

6208 rows × 14 columns

In [ ]:
```
df['Vehicle Class Category'].value_counts()
```

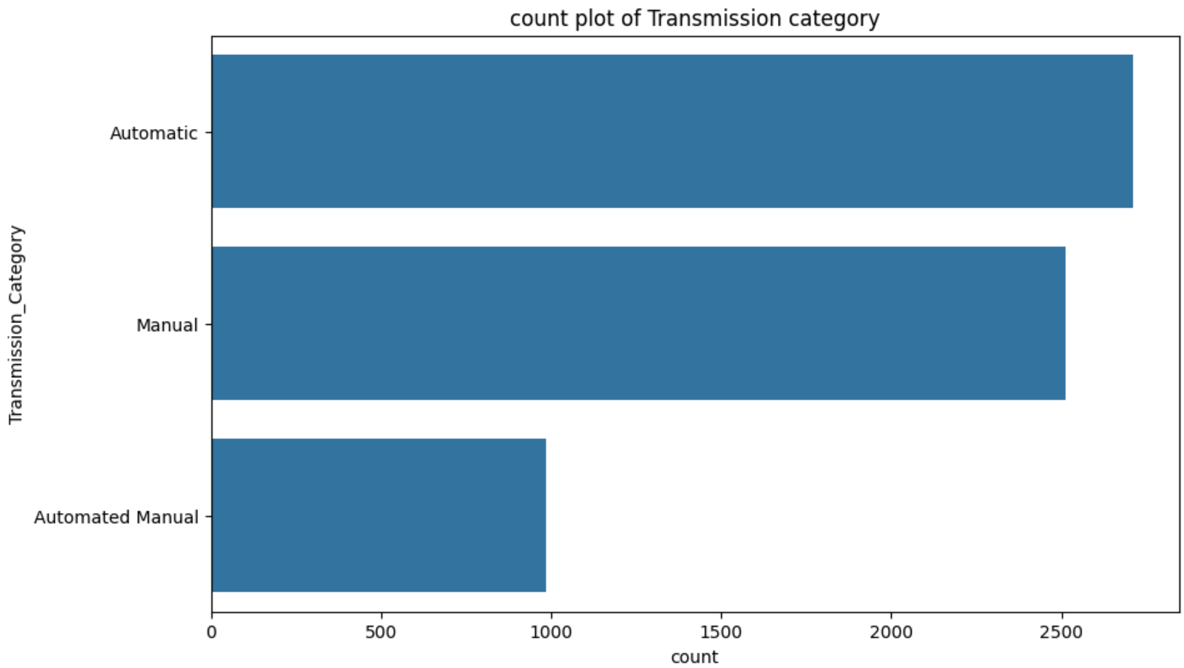
Out[ ]:
```
Vehicle Class Category
Cars       3817
SUVs       1608
Trucks      607
Others      176
Name: count, dtype: int64
```

In [ ]:
```
df['Transmission_Category'].value_counts()
```

Out[ ]:
```
Transmission_Category
Automatic           2710
Manual              2511
Automated Manual     987
Name: count, dtype: int64
```

In [ ]:
```
plt.figure(figsize=(10,6))
sns.countplot(df.Transmission_Category)
plt.title(label='count plot of Transmission category')
```

Out[ ]:
```
Text(0.5, 1.0, 'count plot of Transmission category')
```



In [ ]:
```
le_transmission = LabelEncoder()
df['Transmission_Category']=le_transmission.fit_transform(df['Transmission_Category
le_vehicle_class = LabelEncoder()
df['Vehicle Class Category']=le_vehicle_class.fit_transform(df['Vehicle Class Categ
```

```
<ipython-input-32-027448899426>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['Transmission_Category']=le_transmission.fit_transform(df['Transmission_Categ
ory'])
<ipython-input-32-027448899426>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['Vehicle Class Category']=le_vehicle_class.fit_transform(df['Vehicle Class Ca
tegory']);df
```

Out[ ]:

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

6208 rows × 14 columns

In [ ]:

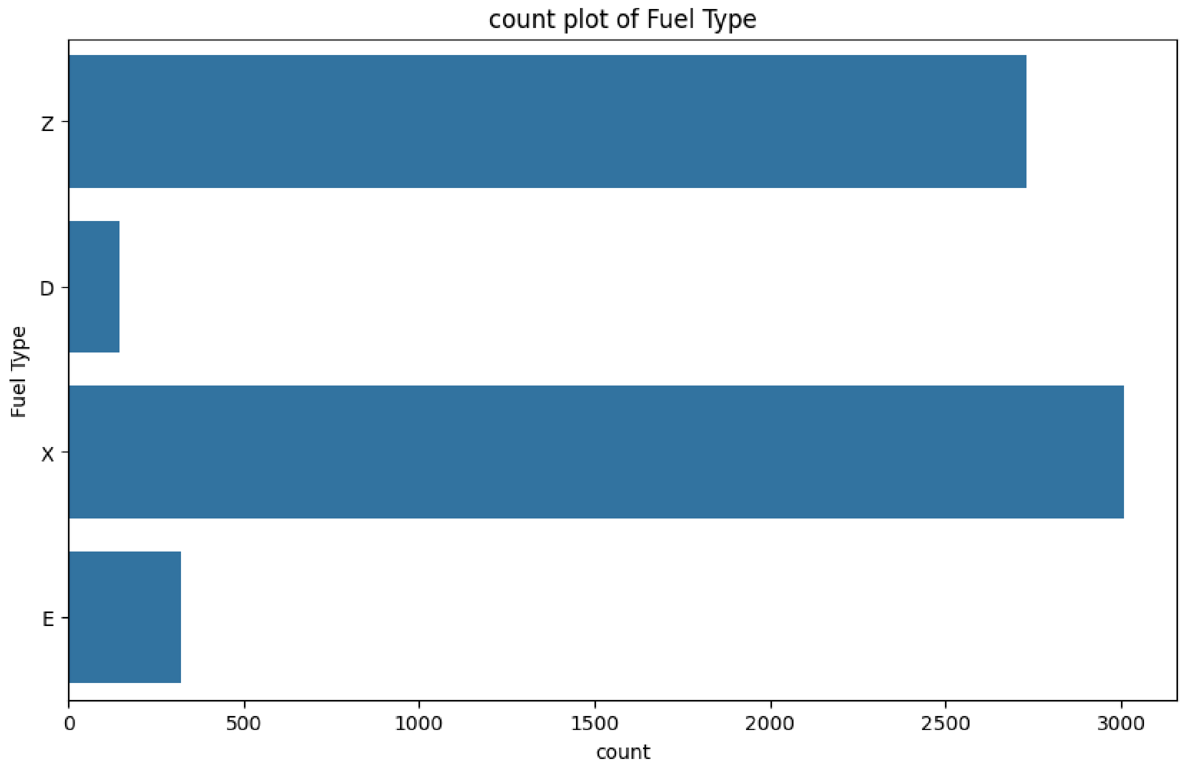
```
df['Fuel Type'].value_counts()
df= df[df['Fuel Type']!='N']
```

In [ ]:

```
plt.figure(figsize=(10,6))
sns.countplot(df['Fuel Type'])
plt.title(label='count plot of Fuel Type')
```

Out[ ]:

```
Text(0.5, 1.0, 'count plot of Fuel Type')
```



In [ ]:
```
onehot_res=pd.get_dummies(df['Fuel Type'],dtype='int',drop_first=True)
onehot_res.columns = ['ethanol','regular gasoline','premium gasoline']
onehot_res
```

Out[ ]:

	ethanol	regular gasoline	premium gasoline
0	0	0	1
1	0	0	1
2	0	0	1
3	0	0	1
4	0	0	1
...	...	...	...
7380	0	0	1
7381	0	0	1
7382	0	0	1
7383	0	0	1
7384	0	0	1

6207 rows × 3 columns

In [ ]:
```
pd.concat([df,onehot_res],axis=1)
df=df.join(onehot_res)
```

In [ ]:
```
df.columns
```

Out[ ]:
```
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)',
       'Transmission_Category', 'Vehicle Class Category', 'ethanol',
       'regular gasoline', 'premium gasoline'],
      dtype='object')
```

In [ ]:
```
df
```

Out[ ]:

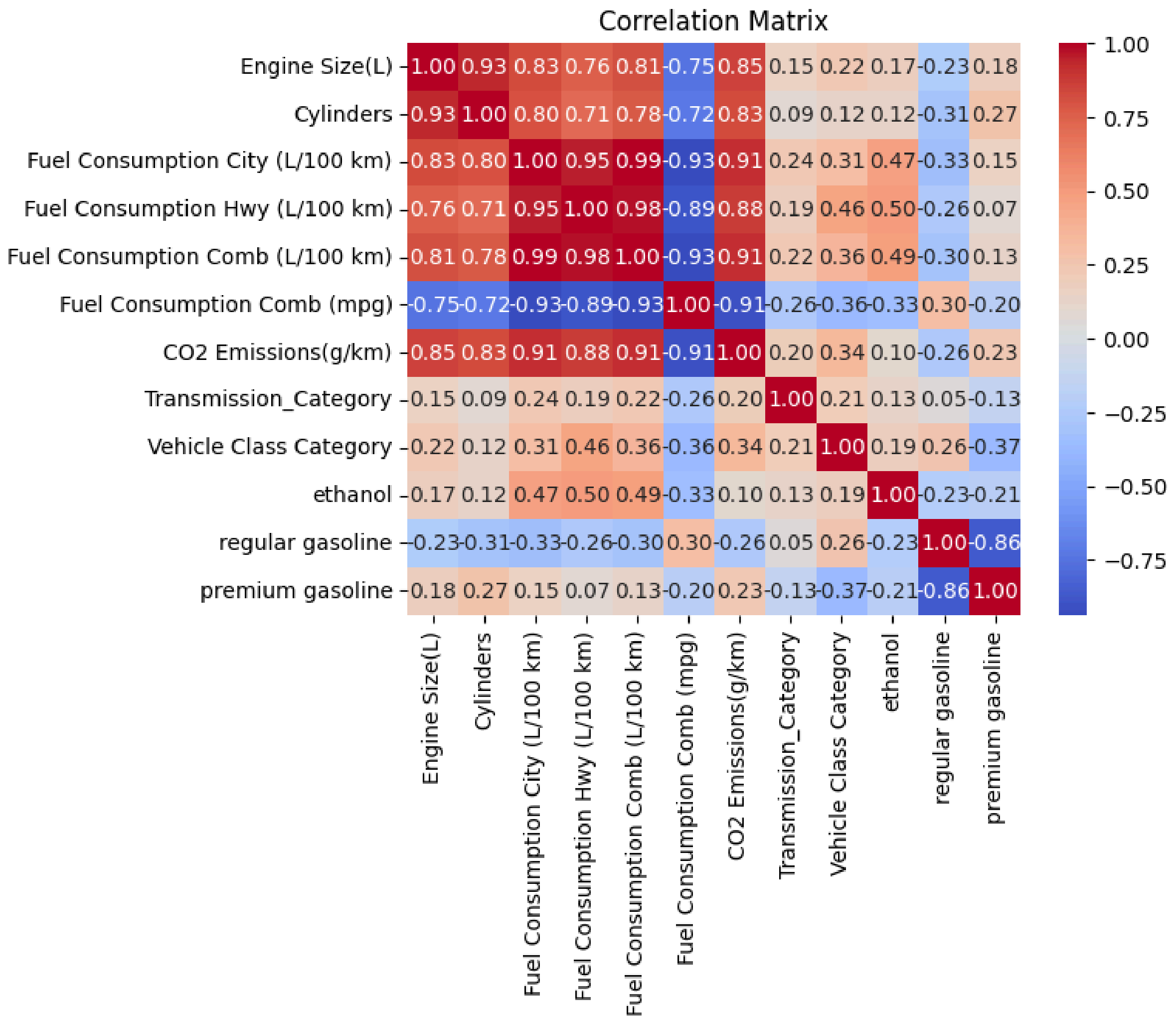
	Make	Model	Vehicle Class	Engine Size(L)	Cylinders	Transmission	Fuel Type	Fuel Consumption City (L/100 km)	Consump Hwy (L,
0	ACURA	ILX	COMPACT	2.0	4	AS5	Z	9.9	
1	ACURA	ILX	COMPACT	2.4	4	M6	Z	11.2	
2	ACURA	ILX HYBRID	COMPACT	1.5	4	AV7	Z	6.0	
3	ACURA	MDX 4WD	SUV - SMALL	3.5	6	AS6	Z	12.7	
4	ACURA	RDX AWD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

6207 rows × 17 columns

In [ ]:
```
df.drop(columns=['Make','Model','Vehicle Class','Transmission','Fuel Type'],inplace
```

In [ ]:
```
cor=df.corr()
sns.heatmap(cor,annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
```

Out[ ]:
```
Text(0.5, 1.0, 'Correlation Matrix')
```



In [ ]:

```
df
```

Out[ ]:

	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)	CO Emissions(g/km
0	2.0	4	9.9	6.7	8.5	33	19
1	2.4	4	11.2	7.7	9.6	29	22
2	1.5	4	6.0	5.8	5.9	48	13
3	3.5	6	12.7	9.1	11.1	25	25
4	3.5	6	12.1	8.7	10.6	27	24
...	...	...	...	...	...	...	
7380	2.0	4	10.7	7.7	9.4	30	21
7381	2.0	4	11.2	8.3	9.9	29	23
7382	2.0	4	11.7	8.6	10.3	27	24
7383	2.0	4	11.2	8.3	9.9	29	23
7384	2.0	4	12.2	8.7	10.7	26	24

6207 rows × 12 columns

In [ ]:
```python
x=df.drop(columns=['CO2 Emissions(g/km)'])
y= df['CO2 Emissions(g/km)']
```

Training and Testing

In [ ]:
```python
x_train,x_test,y_train,y_test=train_test_split(x, y, test_size=0.3, random_state=42
```

In [ ]:
```python
minmax=MinMaxScaler()
x_train_scaled=minmax.fit_transform(x_train)
x_test_scaled=minmax.transform(x_test)
```

In [ ]:
```python
model=LinearRegression()
model.fit(x_train_scaled, y_train)
y_pred=model.predict(x_test_scaled)
y_pred_train=model.predict(x_train_scaled)
```

In [ ]:
```python
mae = mean_absolute_error(y_train, y_pred_train)
mse = mean_squared_error(y_train, y_pred_train)
rmse = np.sqrt(mse)
r_squared = r2_score(y_train, y_pred_train)

print("Mean Absolute Error (MAE):", mae)
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("R-squared:", r_squared)
```

```
Mean Absolute Error (MAE): 2.932416155449537
Mean Squared Error (MSE): 21.688716360117162
Root Mean Squared Error (RMSE): 4.657114595982921
R-squared: 0.9930518798158267
```

In [ ]:
```python
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r_squared = r2_score(y_test, y_pred)

print("Mean Absolute Error (MAE):", mae)
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("R-squared:", r_squared)
```

```
Mean Absolute Error (MAE): 3.0488703440899885
Mean Squared Error (MSE): 24.536631118416263
Root Mean Squared Error (RMSE): 4.95344638796225
R-squared: 0.9921897554352468
```

In [ ]:
```python
lasso_model = Lasso(alpha=0.01)
lasso_model.fit(x_train_scaled, y_train)
```

Out[ ]:
```
▾ Lasso
Lasso(alpha=0.01)
```

In [ ]:
```python
y_pred_lasso_train = lasso_model.predict(x_train_scaled)
y_pred_lasso_test = lasso_model.predict(x_test_scaled)
```

In [ ]:
```python
lasso_train_rmse =np.sqrt(mean_squared_error(y_train, y_pred_lasso_train, squared=F
lasso_test_rmse = np.sqrt(mean_squared_error(y_test, y_pred_lasso_test, squared=Fal
lasso_train_r2 = r2_score(y_train, y_pred_lasso_train)
lasso_test_r2 = r2_score(y_test, y_pred_lasso_test)
lasso_train_MAE = mean_absolute_error(y_train, y_pred_lasso_train)
```

```
lasso_test_MAE = mean_absolute_error(y_test, y_pred_lasso_test)
lasso_train_MSE = mean_squared_error(y_train, y_pred_lasso_train)
lasso_test_MSE = mean_squared_error(y_test, y_pred_lasso_test)
```

In [ ]:
```
print(f'Lasso Regression - Training RMSE: {lasso_train_rmse}, Testing RMSE: {lasso_
print(f'Lasso Regression - Training R2: {lasso_train_r2}, Testing R2: {lasso_test_r
print(f'Lasso Regression - Training MAE: {lasso_train_MAE}, Testing MAE: {lasso_tes
print(f'Lasso Regression - Training MSE: {lasso_train_MSE}, Testing MSE: {lasso_tes
```

```
Lasso Regression - Training RMSE: 2.162139589845112, Testing RMSE: 2.2298105502041
47
Lasso Regression - Training R2: 0.9929988660198148, Testing R2: 0.9921309634351295
Lasso Regression - Training MAE: 2.9574269432226044, Testing MAE: 3.07617909085082
57
Lasso Regression - Training MSE: 21.854200139095703, Testing MSE: 24.7213318160232
02
```

Hyper parameter tuning

In [ ]:
```
alpha_values = [0.01, 0.1, 1.0, 10.0, 100.0]
lasso = Lasso()
grid_params = {
    'alpha': alpha_values,
}
grid_search = GridSearchCV(lasso, param_grid=grid_params, cv=5, scoring='neg_mean_s
grid_search.fit(x, y)
best_alpha = grid_search.best_params_['alpha']
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_coordinate_descent.p
y:631: ConvergenceWarning: Objective did not converge. You might want to increase
the number of iterations, check the scale of the features or consider increasing r
egularisation. Duality gap: 1.683e+03, tolerance: 1.509e+03
  model = cd_fast.enet_coordinate_descent(
```

In [ ]:
```
best_alpha
```

Out[ ]:
```
0.01
```

In [ ]:
```
mlr=LinearRegression()
rf=RandomForestRegressor()
svr=SVR()
dt=DecisionTreeRegressor()
xgb=XGBRegressor()
gb=GradientBoostingRegressor()
lasso=Lasso(alpha=0.01)
```

In [ ]:
```
model_names = ['Multiple Linear Regression', 'RandomForest','Support Vector Regress
train_scores = []
mae_values = []
mse_values = []
rmse_values = []
r_squared_values = []
```

In [ ]:
```
models=[mlr,rf,svr,dt,xgb,gb,lasso]
for model in models:
    model.fit(x_train_scaled,y_train)
    train_predictions = model.predict(x_train_scaled)
    train_score = r2_score(y_train,train_predictions)
    y_pred=model.predict(x_test_scaled)
    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
```

```
    r_squared = r2_score(y_test, y_pred)

    train_scores.append(train_score)
    mae_values.append(mae)
    mse_values.append(mse)
    rmse_values.append(rmse)
    r_squared_values.append(r_squared)
```

In [ ]:

```
metrics_df = pd.DataFrame({
    'Model': model_names,
    'Train R2 Score': train_scores,
    'MAE': mae_values,
    'MSE': mse_values,
    'RMSE': rmse_values,
    'R-squared': r_squared_values
})
print(metrics_df)
```

```
                      Model  Train R2 Score       MAE         MSE  \
0  Multiple Linear Regression        0.993052  3.048870   24.536631
1                RandomForest        0.999060  2.223861   14.383373
2    Support Vector Regressor        0.956425  7.571255  138.186436
3                DecisionTree        0.999622  2.293122   20.930668
4                     XGBoost        0.999049  2.104387   11.106306
5            GradientBoosting        0.996143  2.742368   16.865813
6                       Lasso        0.992999  3.076179   24.721332

        RMSE  R-squared
0   4.953446   0.992190
1   3.792542   0.995422
2  11.755273   0.956014
3   4.575005   0.993338
4   3.332613   0.996465
5   4.106801   0.994631
6   4.972055   0.992131
```

In [ ]:

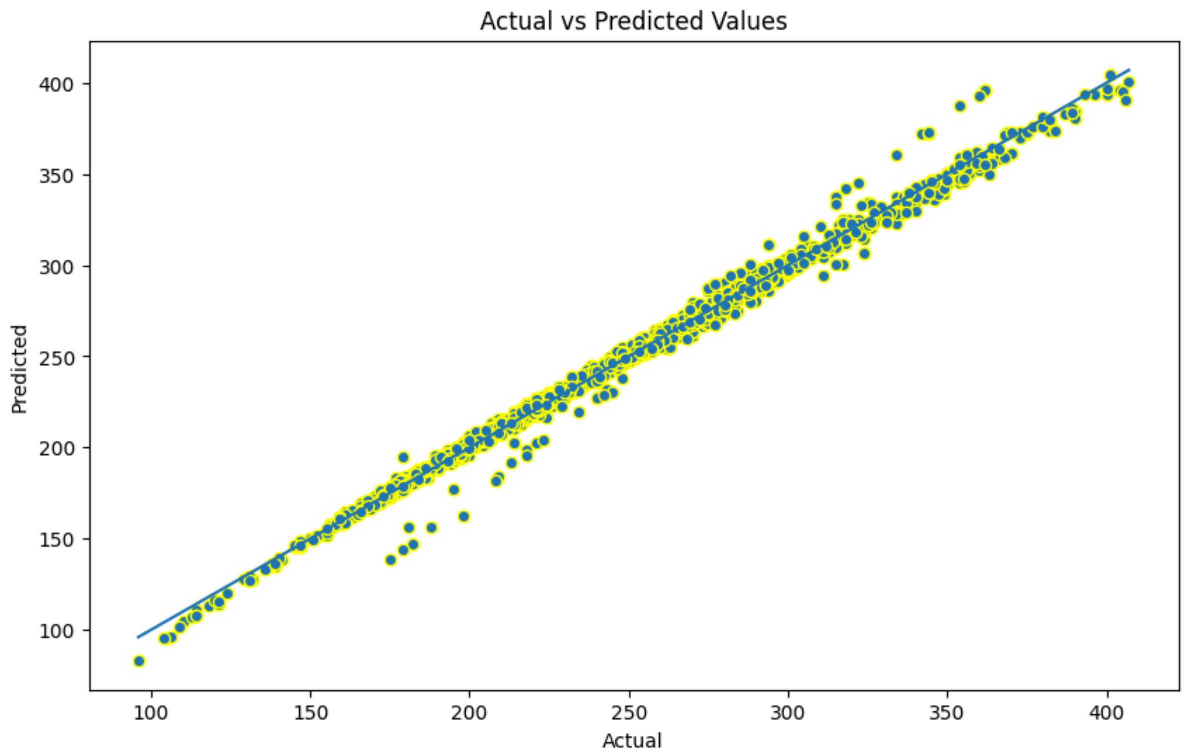
```
rf=RandomForestRegressor(n_estimators=300,max_features='auto',max_depth=50,random_s
rf.fit(x_train_scaled,y_train)
y_pred_rf=rf.predict(x_test_scaled)
print('mae', mean_absolute_error(y_test, y_pred))
print('mse', mean_squared_error(y_test, y_pred))
print('rmse',np.sqrt(mse))
print('r_squared',r2_score(y_test, y_pred))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWar
ning: `max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3.
To keep the past behaviour, explicitly set `max_features=1.0` or remove this param
eter as it is also the default value for RandomForestRegressors and ExtraTreesRegr
essors.
  warn(
mae 3.0761790908508257
mse 24.721331816023202
rmse 4.972055089801721
r_squared 0.9921309634351295
```

Plot actual vs. predicted values

In [ ]:

```
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred,edgecolors=(1,1,0))
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)])
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Actual vs Predicted Values')
plt.show()
```



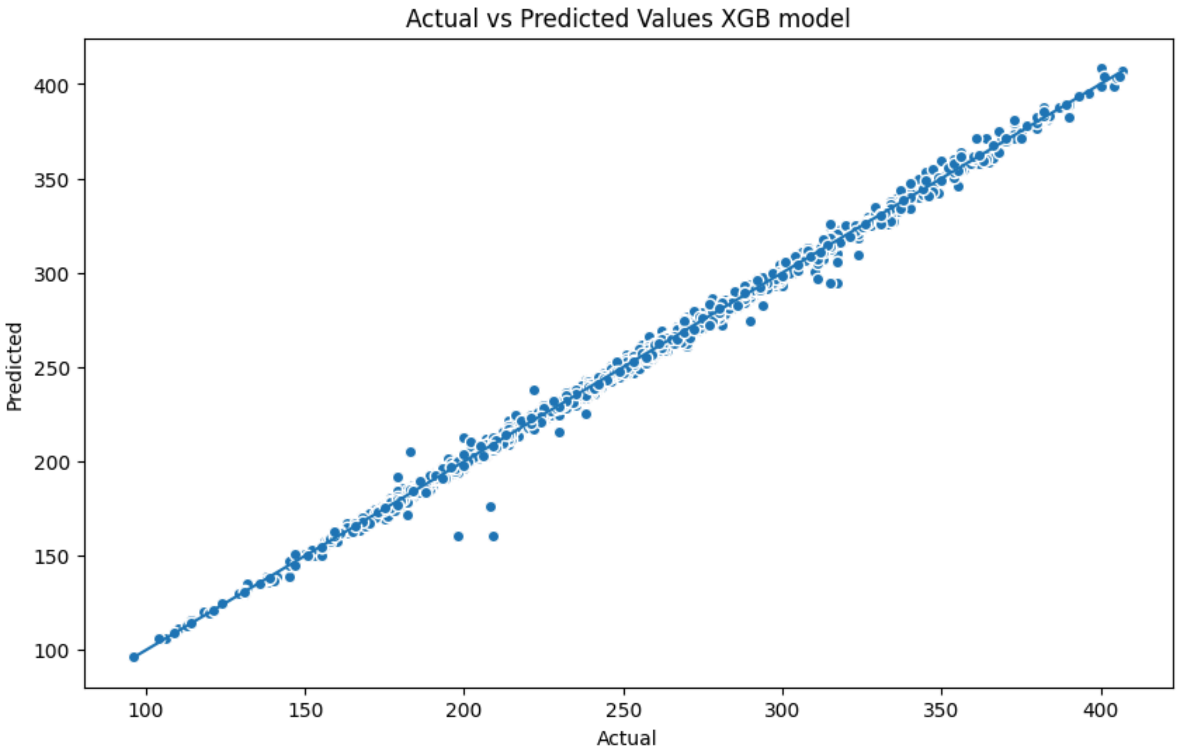
Best Model

```
In [ ]: xgb=XGBRegressor()
        xgb.fit(x_train_scaled,y_train)
        y_pred_xgb=xgb.predict(x_test_scaled)
        print('Mean Absolute Error:', mean_absolute_error(y_test, y_pred_xgb))
        print('Mean Squared Error:', mean_squared_error(y_test, y_pred_xgb))
        print('Root Mean Square Error:',np.sqrt(mse))
        print('R2_squared:',r2_score(y_test, y_pred_xgb))
```

```
Mean Absolute Error: 2.1043865676855313
Mean Squared Error: 11.106306404307675
Root Mean Square Error: 4.972055089801721
R2_squared: 0.9964647563550962
```

Actual Vs Predicted of XGB

```
In [ ]: plt.figure(figsize=(10, 6))
        plt.scatter(y_test, y_pred_xgb, edgecolors=(1,1,1))
        plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)])
        plt.xlabel('Actual')
        plt.ylabel('Predicted')
        plt.title('Actual vs Predicted Values XGB model')
        plt.show()
```



XGB is the best model

Deployment

In [ ]:

```
needed_files={'Minmax':minmax,'model':xgb,'Label_Encoder_Transmission': le_transmis
    'Label_Encoder_Vehicle_Class': le_vehicle_class,'dataframe':df}
import pickle
file=open('file.pkl','wb')
pickle.dump(needed_files,file)
```

In [ ]:

```
file1=open('file.pkl','rb')
res=pickle.load(file1)
res['model']
```

Out[ ]:

```
▾ XGBRegressor
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=
None,
             enable_categorical=False, eval_metric=None, feature_types=
None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=
None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
```

In [ ]:

```
import pandas
import numpy as np
import sklearn
import pickle
import streamlit as st
file1=open(r"C:\Users\Nanz\Downloads\file (7).pkl",'rb')
dict1=pickle.load(file1)
# print(dict1)
```

```
le_transmission_cat=dict1['Label_Encoder_Transmission']
le_vehicle_cat=dict1['Label_Encoder_Vehicle_Class']
model=dict1['model']
minmax= dict1['Minmax']
st.title('CO2 emission by vehicles')
st.header('Deployed model')
Engine=st.slider('Engine capacity in Litres',min_value=0.8,max_value=10.0,step=0.1)
Cylinders=st.selectbox('Enter the number of cylinders:',options=range(2,21))
fuel_cons_city=st.slider('Fuel Consumption in city:',min_value=3.0,max_value=35.0,s
fuel_cons_hwy=st.slider('Fuel Consumption in Highway:',min_value=3.0,max_value=35.0
fuel_cons_comb=st.slider('Fuel Consumption in Combined:',min_value=3.0,max_value=35
fuel_cons_comb_mpg=st.slider('Fuel Consumption in Combined in mpg:',min_value=3.0,m
transmission=st.selectbox('Transmission category',['Automated Manual','Automatic','
Vehicle_cat=st.selectbox('Vehicle class category',['SUVs','Cars','Trucks','Others']
st.subheader('Fuel Type Categorization')
st.write('Note: Choose any one of the fuel type listed below..If your fuel type is
ethanol = st.selectbox('Ethanol',[0,1])
regular_gasoline = st.selectbox('Regular Gasoline',[0,1])
premium_gasoline = st.selectbox('Premium Gasoline',[0,1])



def prediction(engine_size,cylinders,fuel_consumption_city,fuel_consumption_hwy,fue
    transmission_cat=le_transmission_cat.transform([transmission_category])[0]
    vehicle_cat=le_vehicle_cat.transform([vehicle_class_category])[0]

    features = np.array([engine_size, cylinders, fuel_consumption_city, fuel_consum
                         fuel_consumption_comb_mpg, transmission_cat, vehicle_cat,e
                         regular_gasoline,premium_gasoline]).reshape(1, -1)
    scaled_features = minmax.transform(features)
    return model.predict(scaled_features)[0]

def categorize_co2_level(co2_emission):
    if co2_emission < 150:
        return 'Low'
    elif 150 <= co2_emission <= 300:
        return 'Medium'
    else:
        return 'High'

if st.button('Predict CO2 Emission'):
    result = prediction(Engine, Cylinders, fuel_cons_city, fuel_cons_hwy, fuel_cons
    co2_level = categorize_co2_level(result)
    st.write(f'The predicted CO2 emission of the vehicle is: {result:.2f}')
    st.write(f'The CO2 emission level is: {co2_level}')
```

In [14]:
```
!jupyter nbconvert --to pdf '/content/drive/MyDrive/Colab Notebooks/Copy of CO2_emi
```

```
[NbConvertApp] Converting notebook /content/drive/MyDrive/Colab Notebooks/Copy of
CO2_emission_by_vehicles.ipynb to html
[NbConvertApp] Writing 2245877 bytes to /content/drive/MyDrive/Colab Notebooks/Cop
y of CO2_emission_by_vehicles.html
```

In [ ]: