INTRODUCTION

This analysis aims to explore the relationship between vehicle characteristics and CO2 emissions in Canada. Using a dataset containing information on vehicle, make, model, engine specifications, fuel type, and fuel consumption, we will investigate the factors influencing CO2 emissions. By examining trends and patterns within the data, this analysis seeks to identify potential areas for reducing carbon footprint in the Canadian automotive sector.

transmission

A = Automatic

AM = Automated manual

AS = Automatic with select shift

AV = Continuously variable

M = Manual

Fuel type

X = Regular gasoline

Z = Premium gasoline

D = Diesel

E = Ethanol (E85)

N = Natural gas

In [1]:
```
# importing needed libraries

import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score,mean_squared_error, mean_absolute_error
from sklearn.preprocessing import StandardScaler
%matplotlib inline
```

In [2]:
```
df = pd.read_excel("C://Users//quays//OneDrive//Desktop//CO2 Emissions_Canada.xlsx"
```

In [3]:
```
df.head() # displaying the top 5 rows of the dataset
```

Out[3]:

	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

In [4]:
```python
df.tail()  # displaying the down 5 rows of the dataset
```

Out[4]:

	Make	Model	Vehicle Class	Engine Size(L)	Cylinders	Transmission	Fuel Type	Fuel Consumption City (L/100 km)	F Consumpt Hwy (L/ l
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	

EXPLORATORY DATA ANALYSIS (E.D.A)

In [5]:
```python
((df.isnull().sum()) / len(df)) * 100 # checking for percentage null, nan, N/A valu
```

Out[5]:
```
Make                                0.0
Model                               0.0
Vehicle Class                       0.0
Engine Size(L)                      0.0
Cylinders                           0.0
Transmission                        0.0
Fuel Type                           0.0
Fuel Consumption City (L/100 km)    0.0
Fuel Consumption Hwy (L/100 km)     0.0
Fuel Consumption Comb (L/100 km)    0.0
Fuel Consumption Comb (mpg)         0.0
CO2 Emissions(g/km)                 0.0
dtype: float64
```

In [6]:
```
df.dtypes # checking for the data types of the columns
```

Out[6]:
```
Make                                 object
Model                                object
Vehicle Class                        object
Engine Size(L)                      float64
Cylinders                             int64
Transmission                         object
Fuel Type                            object
Fuel Consumption City (L/100 km)    float64
Fuel Consumption Hwy (L/100 km)     float64
Fuel Consumption Comb (L/100 km)    float64
Fuel Consumption Comb (mpg)           int64
CO2 Emissions(g/km)                   int64
dtype: object
```

In [7]:
```
df.describe(include = 'all') # making an initial statistical description of the dat
```

Out[7]:

	Make	Model	Vehicle Class	Engine Size(L)	Cylinders	Transmission	Fuel Type	Fuel Consumption City (L/100 km)	Cons Hw
count	7385	7385	7385	7385.000000	7385.000000	7385	7385	7385.000000	738
unique	42	2047	16	NaN	NaN	27	5	NaN	
top	FORD	F-150 FFV	SUV - SMALL	NaN	NaN	AS6	X	NaN	
freq	628	34	1217	NaN	NaN	1324	3637	NaN	
mean	NaN	NaN	NaN	3.160068	5.615030	NaN	NaN	12.556534	
std	NaN	NaN	NaN	1.354170	1.828307	NaN	NaN	3.500274	
min	NaN	NaN	NaN	0.900000	3.000000	NaN	NaN	4.200000	
25%	NaN	NaN	NaN	2.000000	4.000000	NaN	NaN	10.100000	
50%	NaN	NaN	NaN	3.000000	6.000000	NaN	NaN	12.100000	
75%	NaN	NaN	NaN	3.700000	6.000000	NaN	NaN	14.600000	1
max	NaN	NaN	NaN	8.400000	16.000000	NaN	NaN	30.600000	2

In [8]:
```
df.nunique() # checking for the total number of unique values in each field or colu
```

Out[8]:
```
Make                                   42
Model                                2047
Vehicle Class                          16
Engine Size(L)                         51
Cylinders                               8
Transmission                           27
Fuel Type                               5
Fuel Consumption City (L/100 km)      211
Fuel Consumption Hwy (L/100 km)       143
Fuel Consumption Comb (L/100 km)      181
Fuel Consumption Comb (mpg)            54
CO2 Emissions(g/km)                   331
dtype: int64
```

In [9]:
```
df.shape # checking for the shape of the data, (rows, columns)
```

Out[9]:
```
(7385, 12)
```

In [10]:
```
df.duplicated().sum() # checking for the sum of the duplicated rows in the data
```

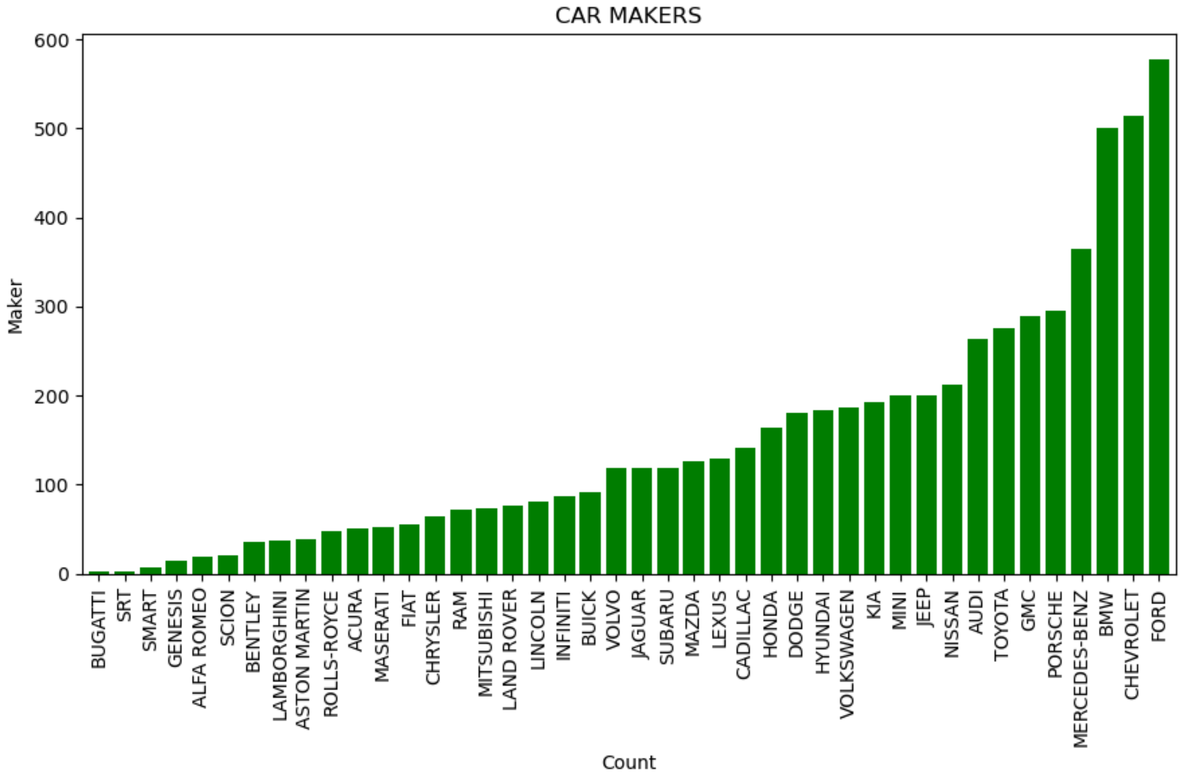
Out[10]:
```
1103
```

In [11]:
```
df.drop_duplicates(inplace = True)  # dropping the duplicated values or rows in the
```

In [12]:
```
df.columns.tolist() # viewing the columns in to list format
```

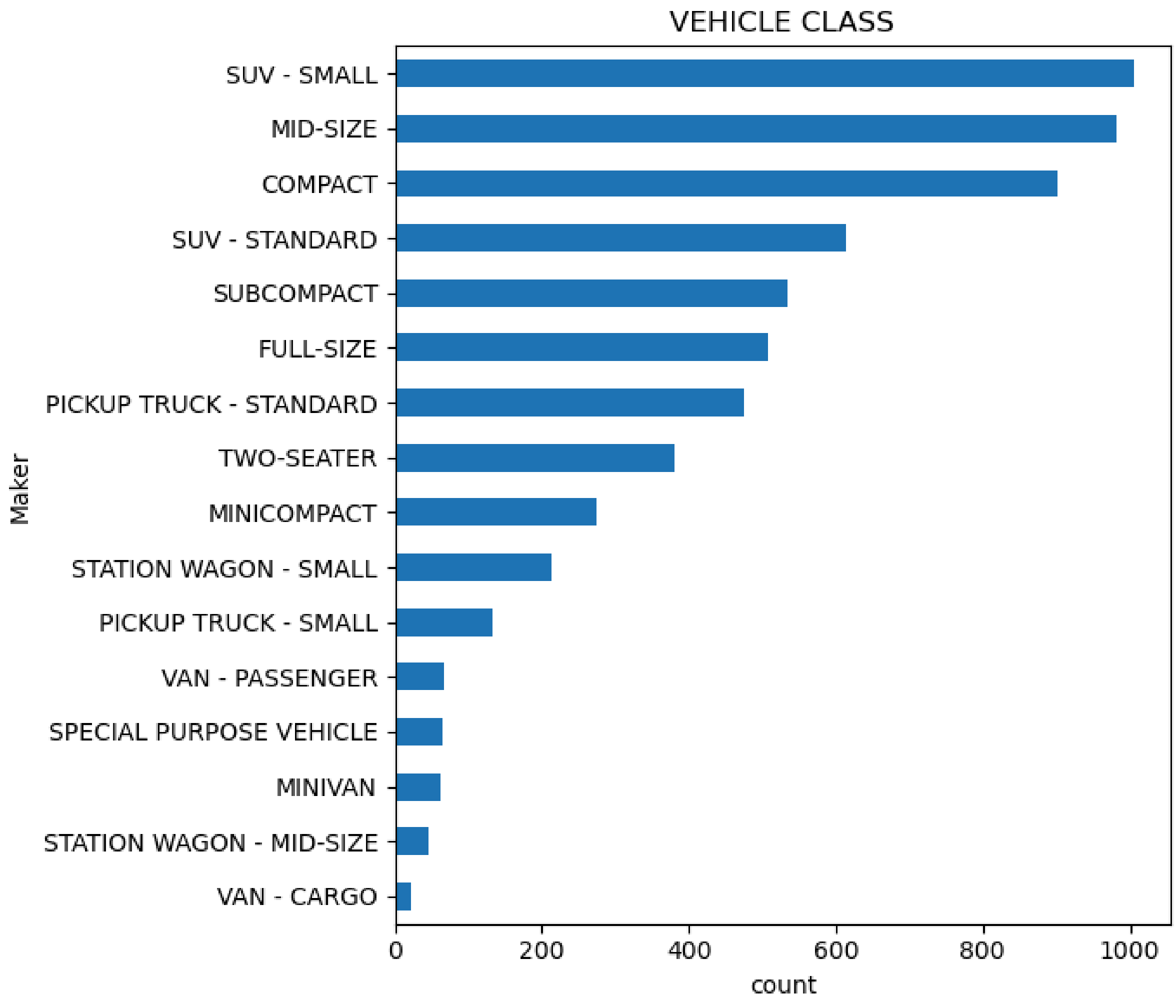
Out[12]:
```
['Make',
 'Model',
 'Vehicle Class',
 'Engine Size(L)',
 'Cylinders',
 'Transmission',
 'Fuel Type',
 'Fuel Consumption City (L/100 km)',
 'Fuel Consumption Hwy (L/100 km)',
 'Fuel Consumption Comb (L/100 km)',
 'Fuel Consumption Comb (mpg)',
 'CO2 Emissions(g/km)']
```

In [13]:
```
# bar plot of car makers and the car counts

plt.figure(figsize = (9, 6))
value_counts = df['Make'].value_counts()
plot = value_counts.sort_values(ascending = True)
plot.plot(kind = 'bar', width = 0.8, color = 'green')
plt.xlabel('Count', color = 'black')
plt.ylabel('Maker', color = 'black')
plt.title('CAR MAKERS', color = 'black')
plt.xticks(rotation = 90)
plt.tight_layout()
plt.show()
```



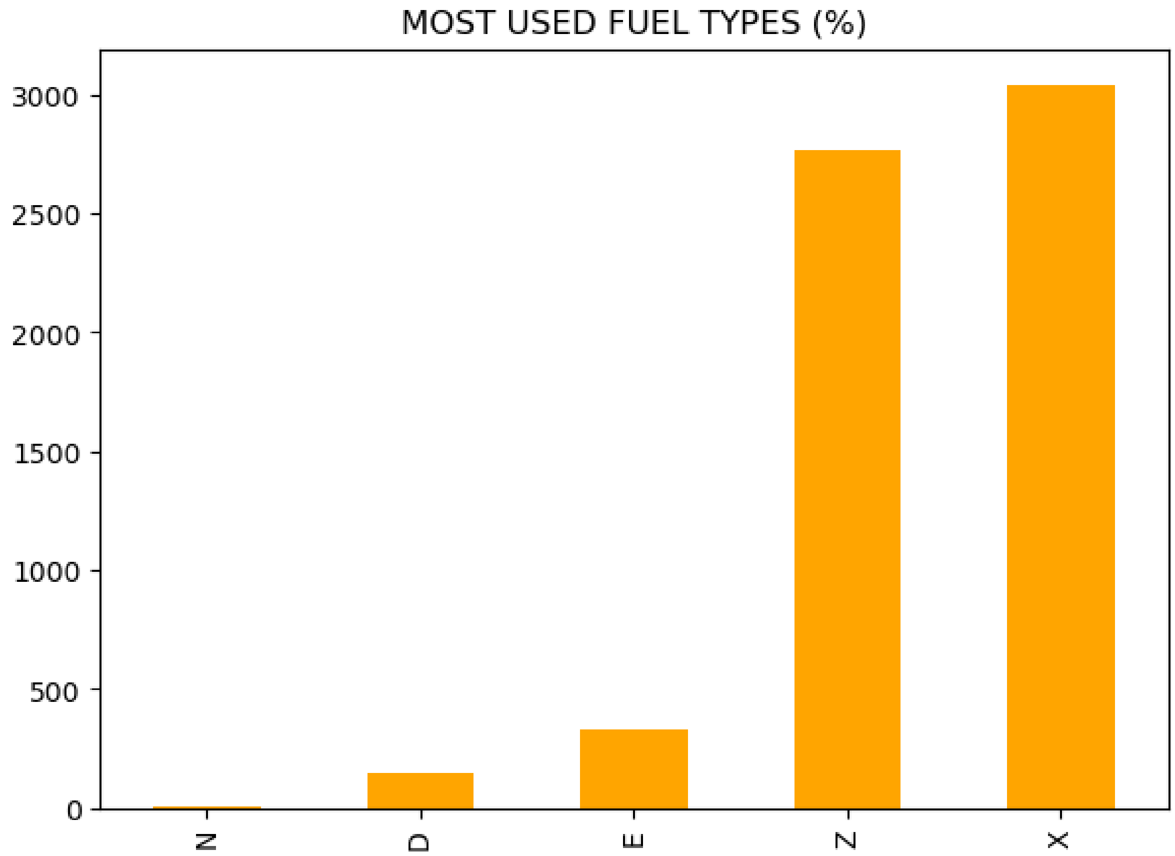
In [14]:

```
# bar plot of vehicle class and the vehicle counts

plt.figure(figsize = (7, 6))
value_counts = df['Vehicle Class'].value_counts()
plot = value_counts.sort_values(ascending = True)
plot.plot(kind = 'barh')
plt.xlabel('count', color = 'black')
plt.ylabel('Maker', color = 'black')
plt.title('VEHICLE CLASS', color = 'black')
plt.tight_layout()
plt.show()
```



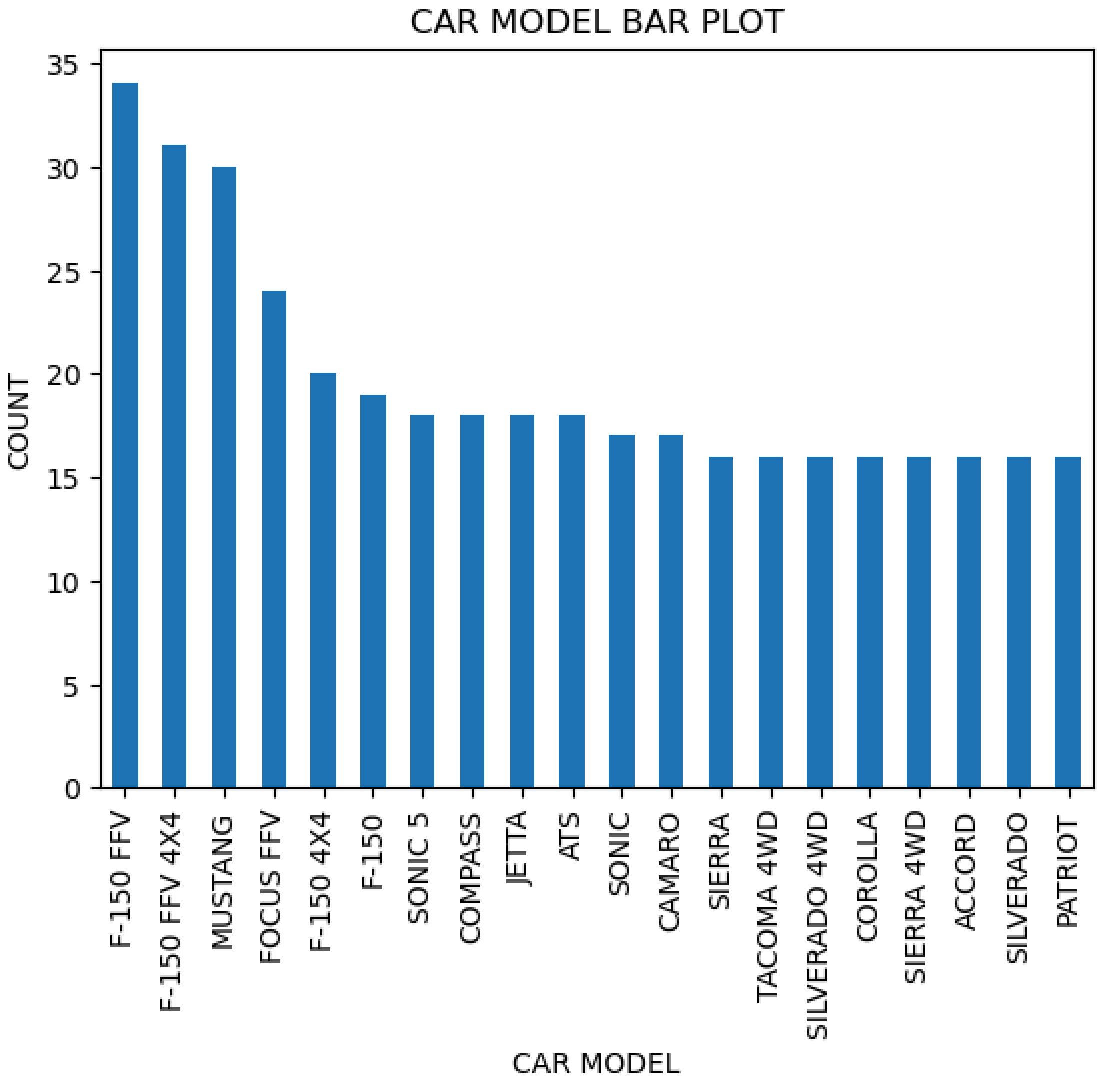
In [15]:

```
# bar chart of the fuel types

plt.figure(figsize = (7, 5))
value_counts = df['Fuel Type'].value_counts()
plot = value_counts.sort_values(ascending = True)
plot.plot(kind = 'bar', color = 'orange')
plt.title('MOST USED FUEL TYPES (%)', color = 'black')
plt.show()
```



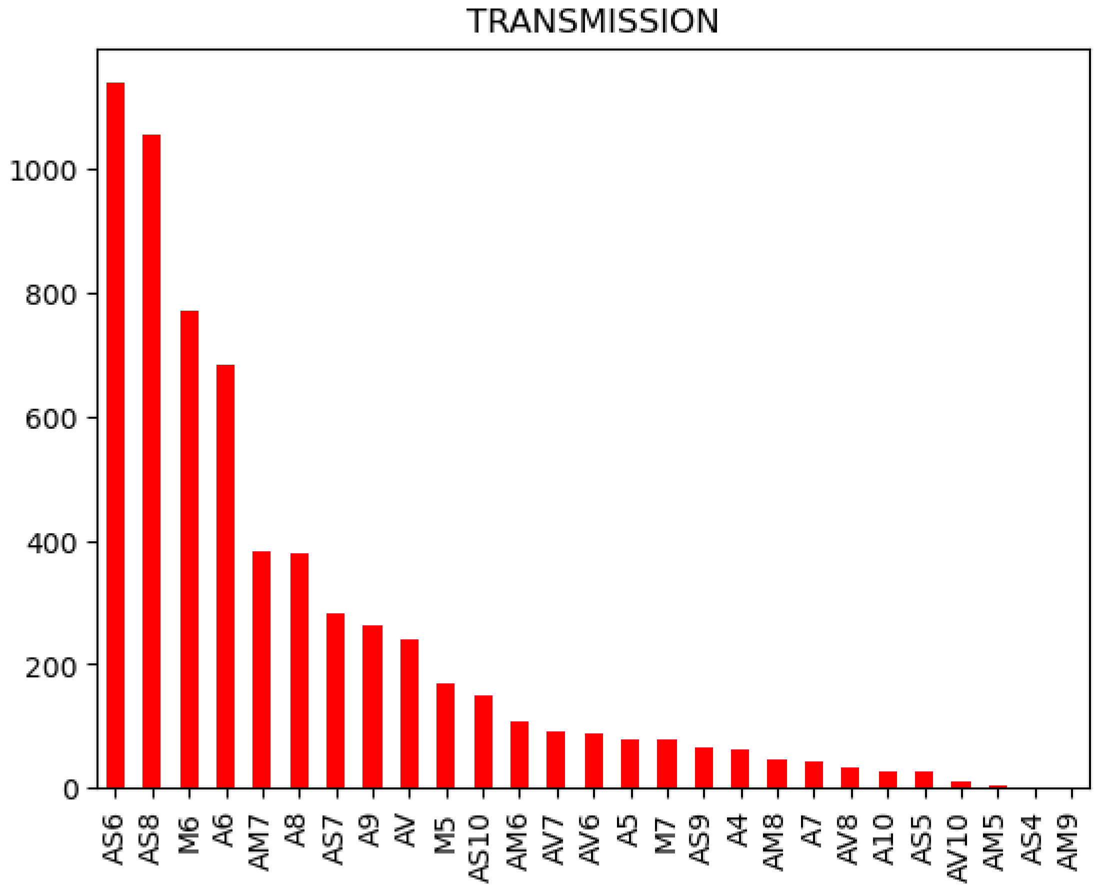
In [16]:

```
# bar plot of the model column

df['Model'].value_counts()[:20].plot(kind = 'bar')
plt.xlabel('CAR MODEL')
plt.ylabel('COUNT')
plt.title('CAR MODEL BAR PLOT')
plt.show()
```



In [17]:

```
# Transmission bar plot

df['Transmission'].value_counts().plot(kind = 'bar', color = 'red')
plt.title('TRANSMISSION', color = 'black')
plt.show()
```



Label Encoding

In [18]:
```
encoder = LabelEncoder()
```

In [19]:
```
df['Model'] = df['Model'].astype(str) # since it contains int and str, all values c
```

In [20]:
```
# converting all the string data types to numeric or int dtype

df['Make'] = encoder.fit_transform(df['Make'])
df['Model'] = encoder.fit_transform(df['Model'])
df['Vehicle Class'] = encoder.fit_transform(df['Vehicle Class'])
df['Transmission'] = encoder.fit_transform(df['Transmission'])
df['Fuel Type'] = encoder.fit_transform(df['Fuel Type'])
```

feature engineering

In [21]:
```
# creating new columnS called fuel efficiency in Km/L

df['fuel efficiency city'] = round(100 / df['Fuel Consumption City (L/100 km)'], 1)
df['fuel efficiency Highway'] = round(100 / df['Fuel Consumption Hwy (L/100 km)'],
```

In [22]:
```
df # reviewing the dataset
```

Out[22]:

	Make	Model	Vehicle Class	Engine Size(L)	Cylinders	Transmission	Fuel Type	Fuel Consumption City (L/100 km)	Fuel Consumption Hwy (L/100 km)
0	0	1055	0	2.0	4	14	4	9.9	6.7
1	0	1055	0	2.4	4	25	4	11.2	7.7
2	0	1056	0	1.5	4	22	4	6.0	5.8
3	0	1231	11	3.5	6	15	4	12.7	9.1
4	0	1497	11	3.5	6	15	4	12.1	8.7
...	...	...	...	...	...	...	...	...	...
7380	41	1947	11	2.0	4	17	4	10.7	7.7
7381	41	1953	11	2.0	4	17	4	11.2	8.3
7382	41	1955	11	2.0	4	17	4	11.7	8.6
7383	41	1963	12	2.0	4	17	4	11.2	8.3
7384	41	1964	12	2.0	4	17	4	12.2	8.7

6282 rows × 14 columns

In [23]:
```
df.describe() # statistics of the fields or the columns
```

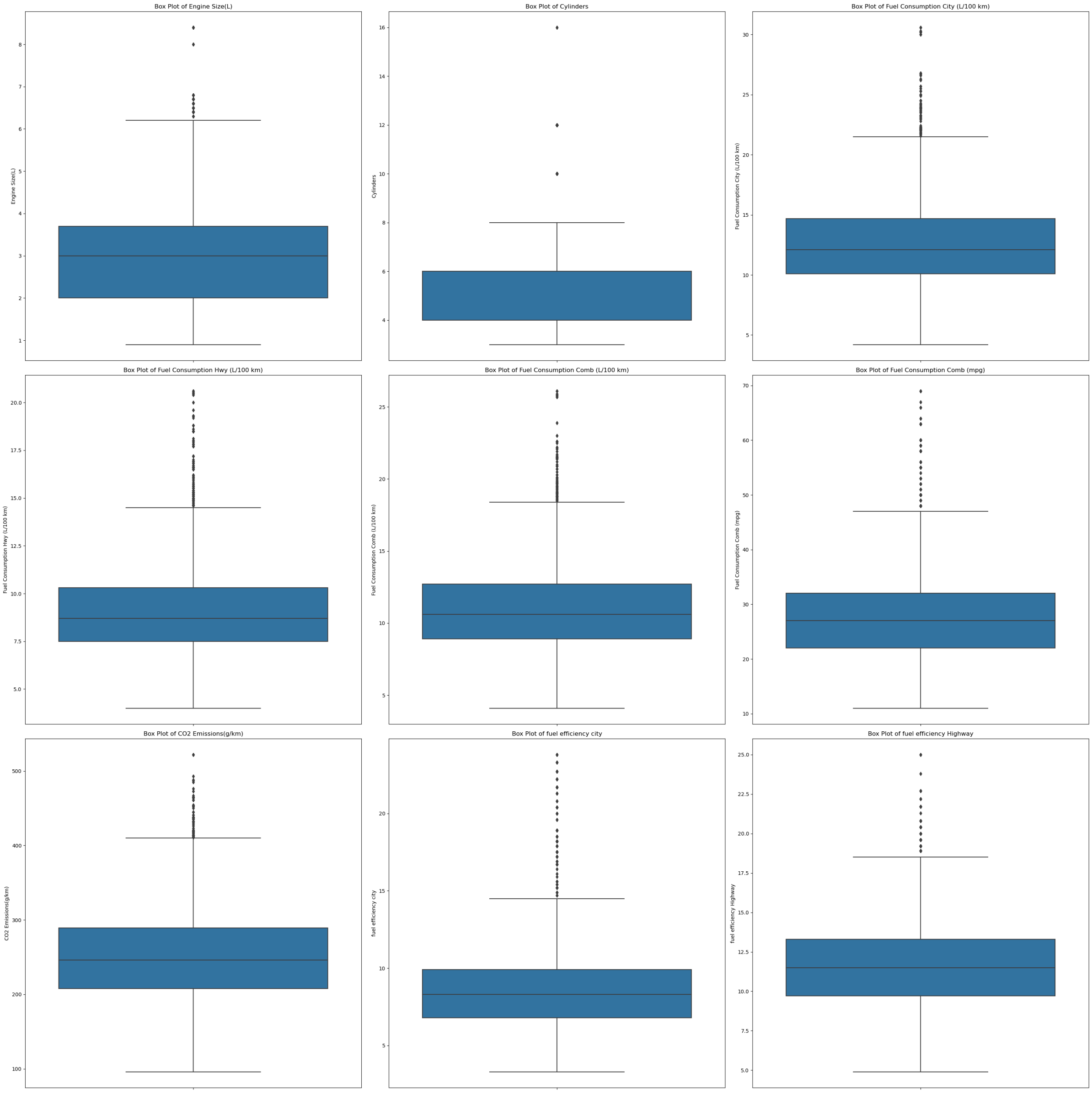
Out[23]:

	Make	Model	Vehicle Class	Engine Size(L)	Cylinders	Transmission	Fuel Type
count	6282.000000	6282.000000	6282.000000	6282.000000	6282.000000	6282.000000	6282.00000C
mean	19.463706	1021.653136	6.333811	3.162066	5.619230	14.078478	3.264725
std	11.438964	575.865335	4.828190	1.365134	1.846144	7.251199	0.88942€
min	0.000000	0.000000	0.000000	0.900000	3.000000	0.000000	0.00000C
25%	9.000000	531.000000	2.000000	2.000000	4.000000	8.000000	3.00000C
50%	17.000000	990.500000	6.000000	3.000000	6.000000	15.000000	3.00000C
75%	29.000000	1523.000000	11.000000	3.700000	6.000000	17.000000	4.00000C
max	41.000000	2046.000000	15.000000	8.400000	16.000000	26.000000	4.00000C

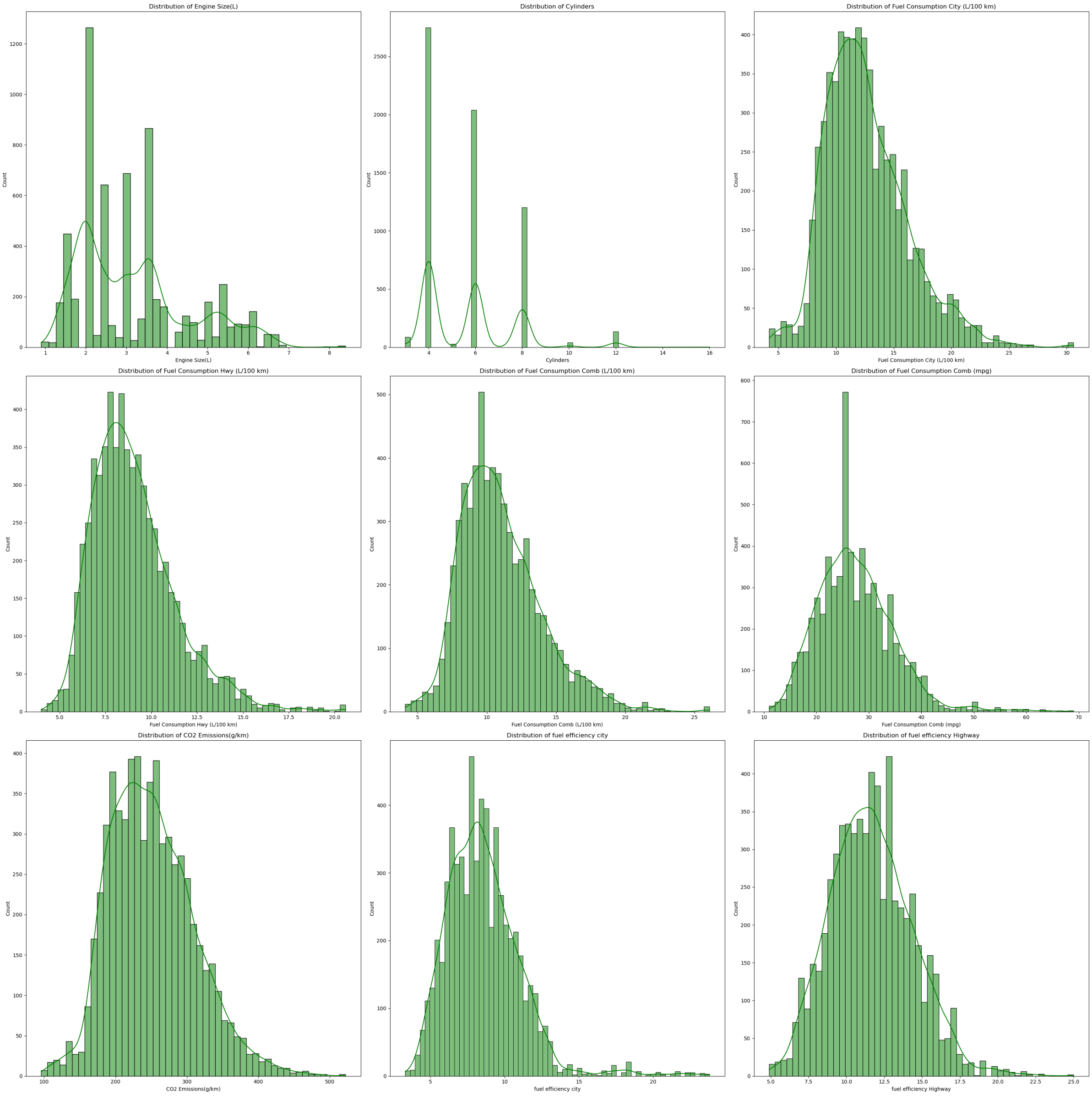
In [24]:
```
# Identify numerical and categorical columns
numerical_cols = df.select_dtypes(include=['int64', 'float64']).columns
```

In [25]:
```
# Box plots for numerical columns

plt.figure(figsize=(30, 30))
for i, col in enumerate(numerical_cols, 1):
    plt.subplot(3, 3, i)
    sns.boxplot(y=df[col])
    plt.title(f'Box Plot of {col}')
```

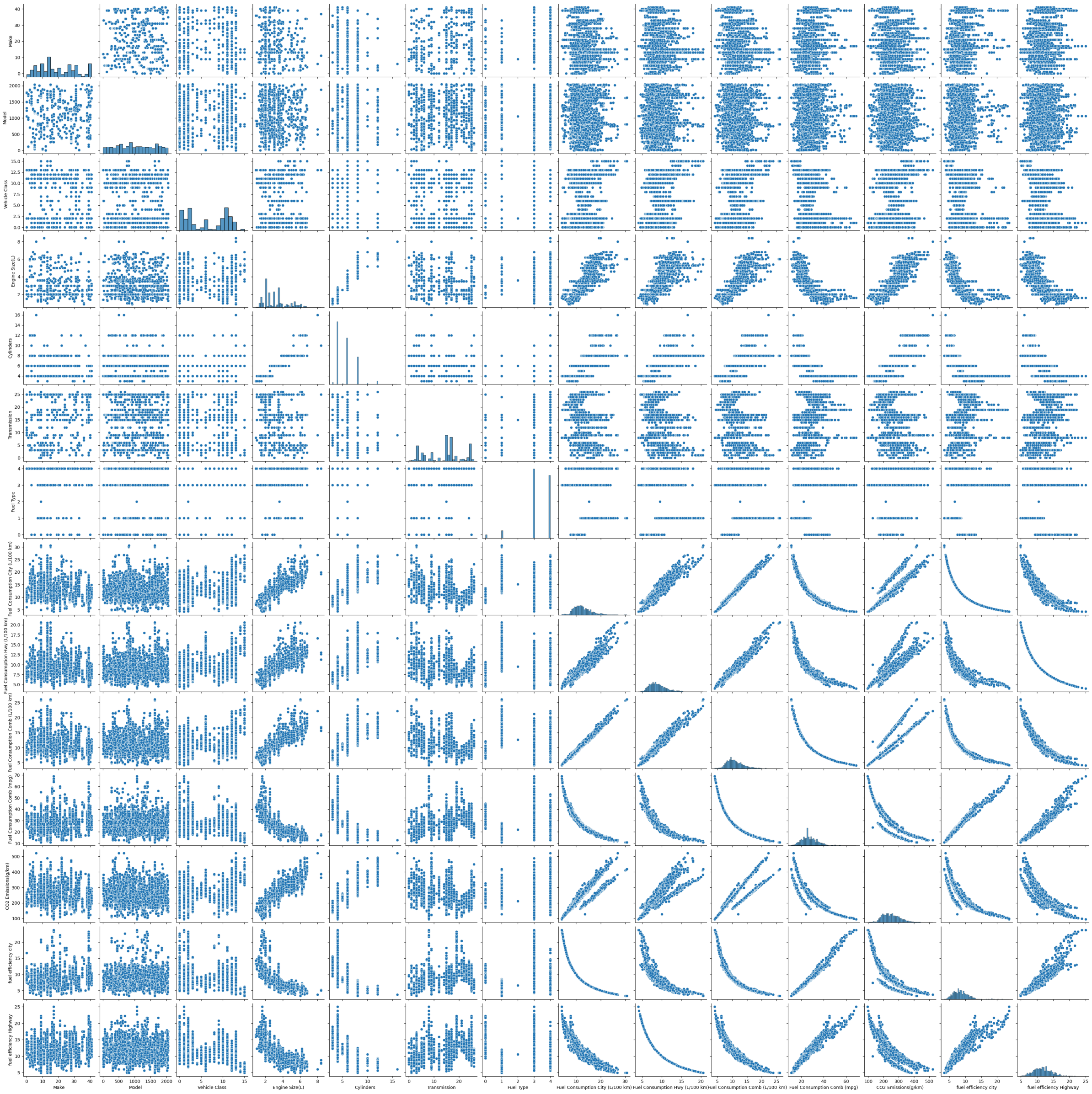
```
plt.tight_layout()
plt.show()
```



In [26]:
```
# Histograms for numerical columns to show values distribution

plt.figure(figsize=(30, 30))
for i, col in enumerate(numerical_cols, 1):
    plt.subplot(3, 3, i)
    sns.histplot(df[col], kde=True, color = 'g')
    plt.title(f'Distribution of {col}')
plt.tight_layout()
plt.show()
```



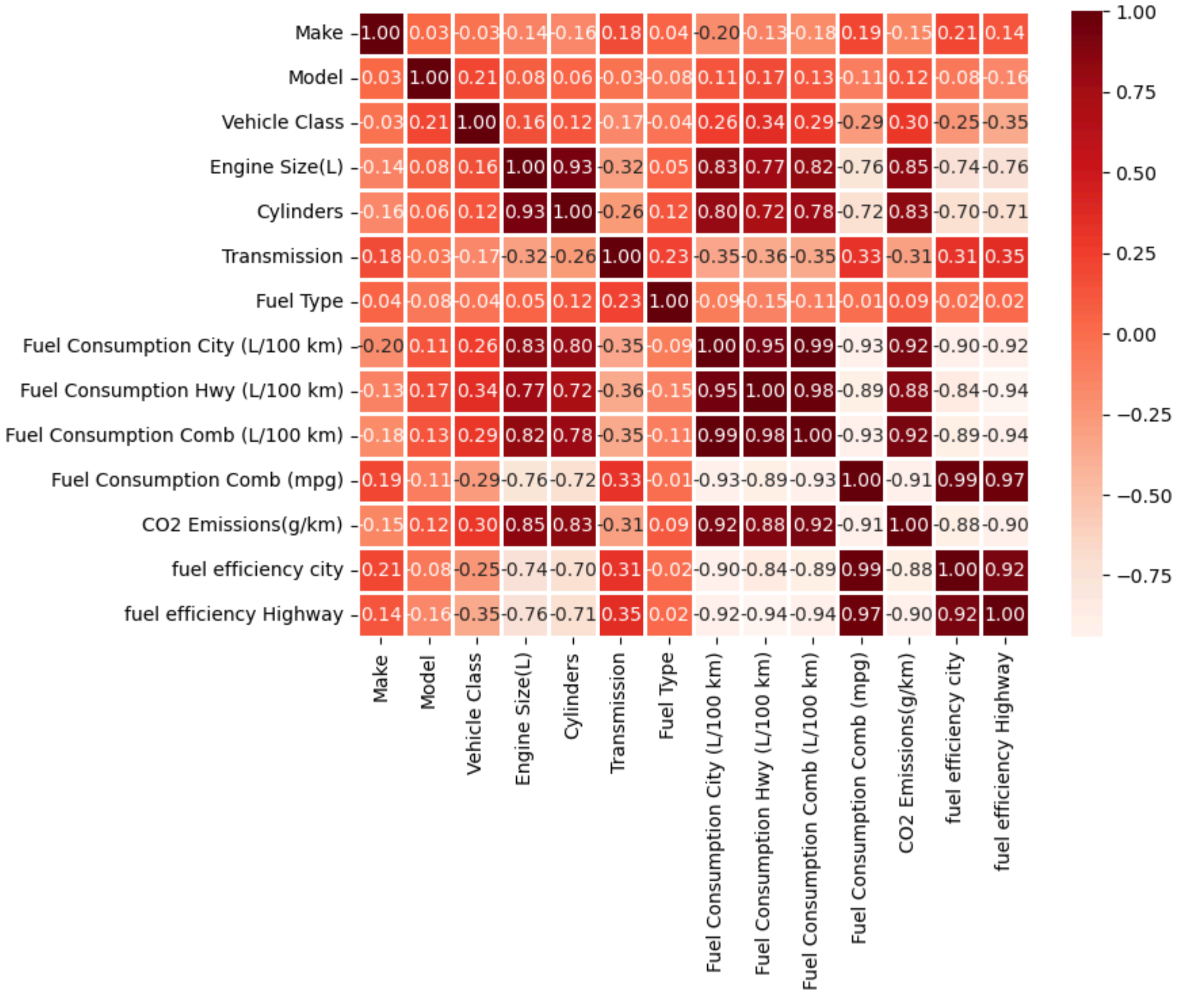
In [27]:
```
# pairplot to show relationship between fields

sns.pairplot(df)
plt.show()
```



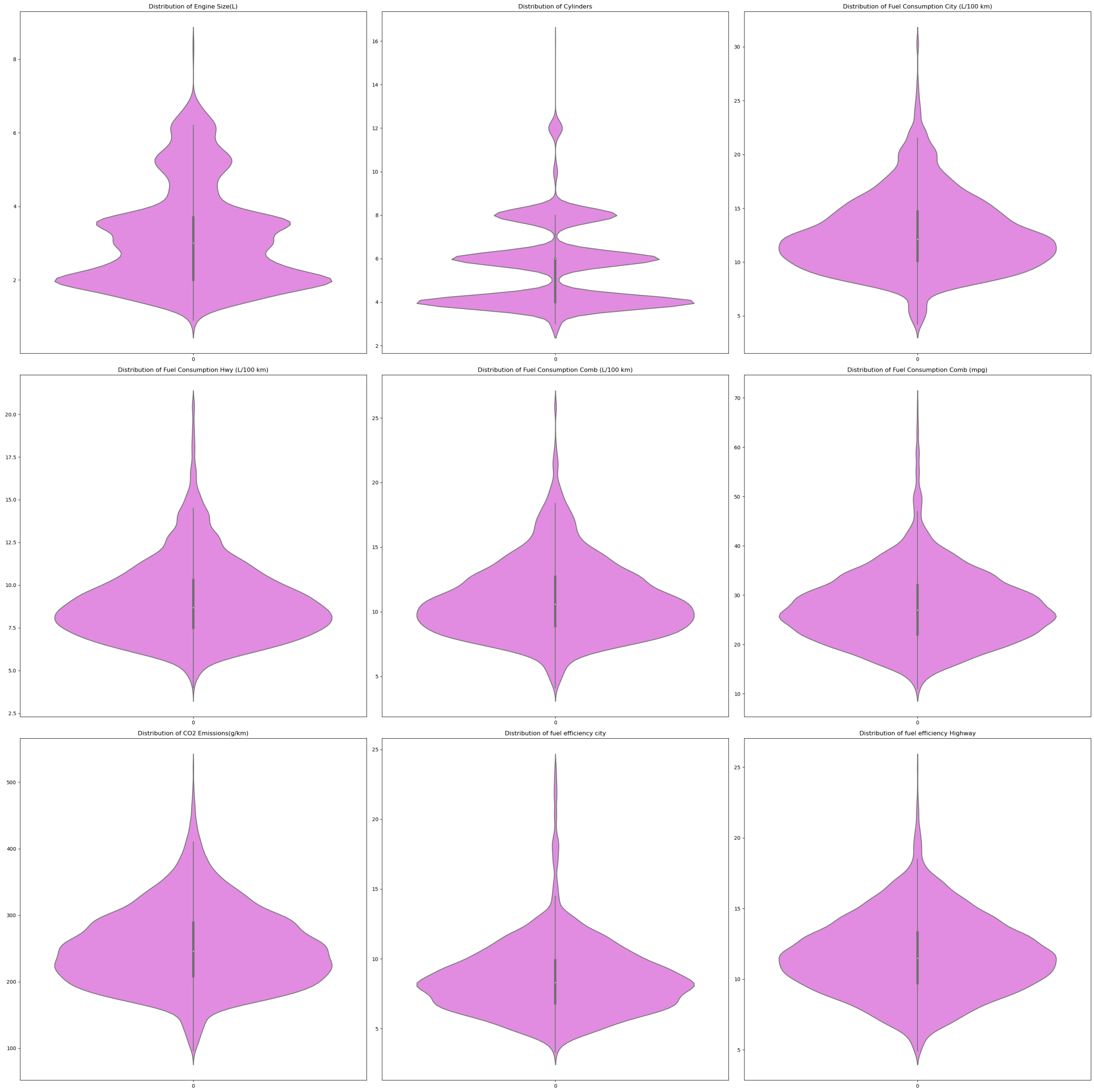
In [28]:

```
# heatmap to show fields or columns correlation with each other


plt.figure(figsize = (20, 10))
plt.figure(figsize = (8, 6))
sns.heatmap(df.corr(), annot = True,linewidth = 1,fmt='.2f', cmap = 'Reds')
plt.show()
```

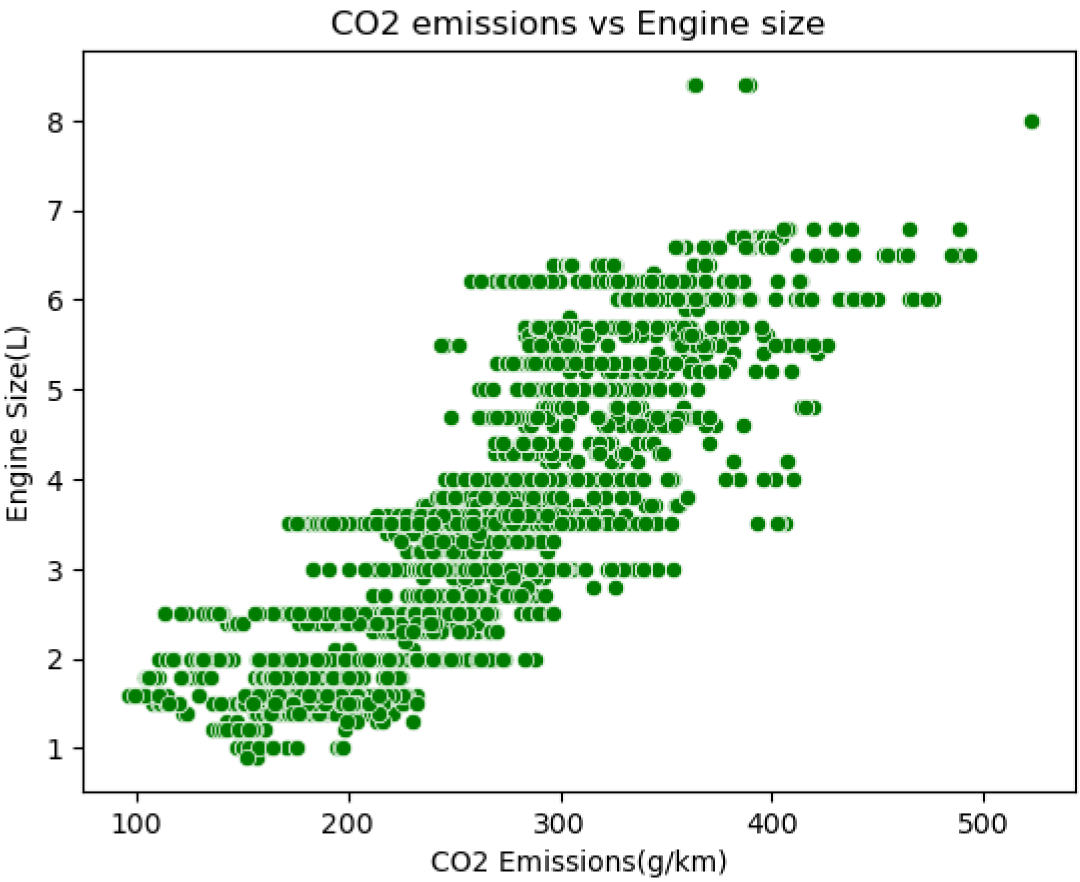
```
<Figure size 2000x1000 with 0 Axes>
```



In [29]:
```
# violinplot to show distribution and density of values in columns

plt.figure(figsize=(30, 30))
for i, col in enumerate(numerical_cols, 1):
    plt.subplot(3, 3, i)
    sns.violinplot(df[col], kde=True, color = 'violet')
    plt.title(f'Distribution of {col}')
plt.tight_layout()
plt.show()
```



In [30]:
```
# scatter plot to show how engine size contributes to CO2 emissions

sns.scatterplot(x = df['CO2 Emissions(g/km)'], y = df['Engine Size(L)'], color ='gr
plt.title('CO2 emissions vs Engine size')
plt.show()
```



MODEL CONSTRUCTION

In [31]:
```
# splitting data data into dependent and independent variable

x = df[['Make','Model', 'Vehicle Class', 'Engine Size(L)', 'Cylinders', 'Transmissi
       'Fuel Type', 'Fuel Consumption City (L/100 km)',
       'Fuel Consumption Hwy (L/100 km)', 'Fuel Consumption Comb (L/100 km)',
       'Fuel Consumption Comb (mpg)','fuel efficiency city', 'fuel efficiency Highw

y = df['CO2 Emissions(g/km)'] # dependent
```

In [32]:
```
# standard scalling the independent columns to have a common scale

sc = StandardScaler()
x = sc.fit_transform(x)
```

In [33]:
```
# splitting data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size= 0.2, random_st
```

In [34]:
```
X_train[:5] # viewing the top 5 rows of the X_train
```

Out[34]:
```
array([[ 1.70800862,  0.01623228,  0.96652387,  0.24756598,  0.20626821,
         0.40293394, -0.29765909, -0.22818518, -0.16262875, -0.24371622,
        -0.05662489, -0.02852091, -0.06127902],
       [-0.5651054 , -0.29462902, -0.0691434 ,  1.34644685,  1.28969373,
         0.12709555, -0.29765909,  0.92590087,  0.97837123,  0.94412484,
        -1.02287363, -0.89884634, -1.05437848],
       [ 1.00858892,  0.64663817,  0.96652387,  0.24756598,  0.20626821,
         0.67877233, -0.29765909, -0.14373986, -0.07485952, -0.10796296,
        -0.19466043, -0.1076414 , -0.1716234 ],
       [-0.30282301, -0.90940501, -1.10481068, -1.21760852, -0.8771573 ,
         1.50628751, -0.29765909, -1.29782591, -1.25974412, -1.29580402,
         1.59980151,  1.55388896,  1.63066822],
       [ 0.92116146,  0.54938547,  0.96652387, -0.55827999, -0.8771573 ,
         0.95461073, -0.29765909, -0.76300555, -0.42593644, -0.65097602,
         0.49551724,  0.60444304,  0.23297268]])
```

In [35]:
```
X_test[:5] # viewing the top 5 rows of the X_test
```

Out[35]:
```
array([[ 1.79543608,  0.10827501, -1.31194413, -1.29086724, -0.8771573 ,
         0.12709555, -0.29765909, -1.15708371, -1.25974412, -1.22792739,
         1.46176598,  1.27696724,  1.63066822],
       [-0.73996032, -0.54123408,  1.17365733,  1.85925792,  1.28969373,
        -1.25209642, -0.29765909,  1.15108839,  0.80283277,  1.04593979,
        -1.02287363, -1.01752708, -0.90725264],
       [ 0.1343143 ,  0.4938125 ,  0.55225696, -1.14434979, -0.8771573 ,
        -0.83833883, -0.29765909, -2.28302132, -1.87412872, -2.14426193,
         4.49854773,  5.39123291,  3.35939693],
       [ 0.74630653, -1.24284283,  1.17365733,  0.6138596 ,  1.28969373,
        -1.11417722,  0.82675124,  1.54516655,  2.86540966,  2.03015096,
        -1.43698023, -1.21532832, -1.93713357],
       [ 0.74630653, -0.48913442, -0.89767722,  1.12667067,  1.28969373,
         0.26501474,  0.82675124,  0.08144766,  0.23233278,  0.12960525,
        -0.33269596, -0.30544264, -0.4658751 ]])
```

In [36]:
```
y_train[:5] # viewing the top 5 rows of the y_train
```

Out[36]:
```
4292    243
3673    323
6286    250
3800    167
6258    212
Name: CO2 Emissions(g/km), dtype: int64
```

In [37]:
```
y_test[:5] # viewing the top 5 rows of the y_test
```

Out[37]:
```
4338    175
4672    331
3934    110
6167    396
2999    270
Name: CO2 Emissions(g/km), dtype: int64
```

LINEAR REGRESSION

In [38]:
```
lr_model = LinearRegression() # constructing the linear regression model
```

In [39]:
```
lr_model.fit(X_train, y_train) # fitting the training datasets to the model for tra
```

Out[39]:
```
▾ LinearRegression
LinearRegression()
```

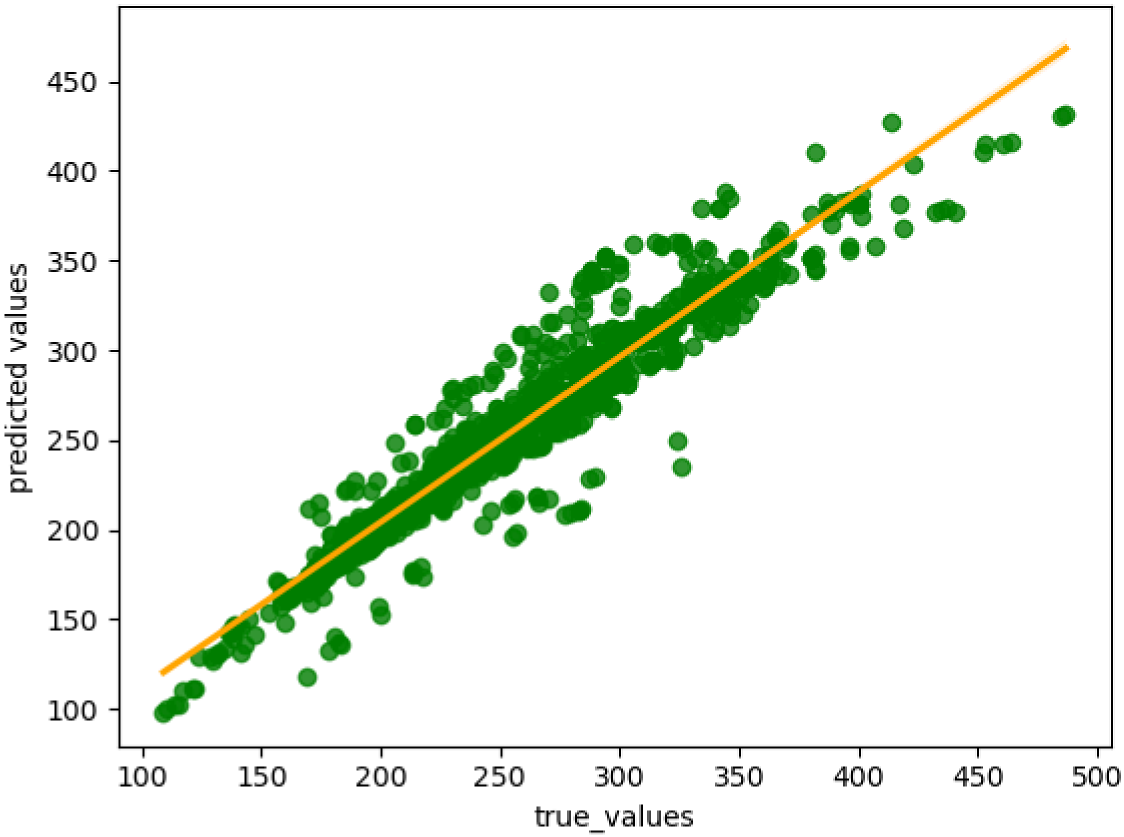
In [40]:
```
# showing the coefficients and intercept of the model

print('computed model coefficients =  ', lr_model.coef_)
print('\ncomputed model intercept =  ', lr_model.intercept_)
```

```
computed model coefficients =   [ 0.91435469  0.19550024  3.49060435  7.58731682 1
0.39860407 -0.4507636
  6.70487182 -6.37809534  1.51148312 27.95448817 -7.41845556 -7.1922284
 -3.76384871]

computed model intercept =   251.15542431061957
```

In [41]:
```
# using the algorithm to predict values

y_pred = lr_model.predict(X_test)
y_pred
```

Out[41]:
```
array([172.25424537, 321.98784645, 100.87620936, ..., 224.79875991,
       171.35385329, 253.33197662])
```

In [42]:
```
# visualization of the correlation between the actual and predicted dependent value

sns.regplot(x = y_test, y = y_pred, color = 'g', line_kws = {"color":'orange'})
plt.xlabel('true_values', color = 'black')
plt.ylabel('predicted values', color = 'black')
plt.show()
```



model's metrics

In [43]:
```
# displaying metrics about the linear regression algorithm on the dataset

lrr2_score = r2_score(y_test, y_pred) * 100
MSE = mean_squared_error(y_test, y_pred)
MAE = mean_absolute_error(y_test, y_pred)
```

```
print(f"r2_score: {round(lrr2_score, 4)} %\n\nMean Squared Error: {round(MSE, 4)}\r
```

```
r2_score: 90.8586 %

Mean Squared Error: 308.3711

mean absolute error:11.458081048203125
```

RANDOM FOREST REGRESSOR

In [44]:
```
rf_model = RandomForestRegressor() # constructing the random forest regressor model
```

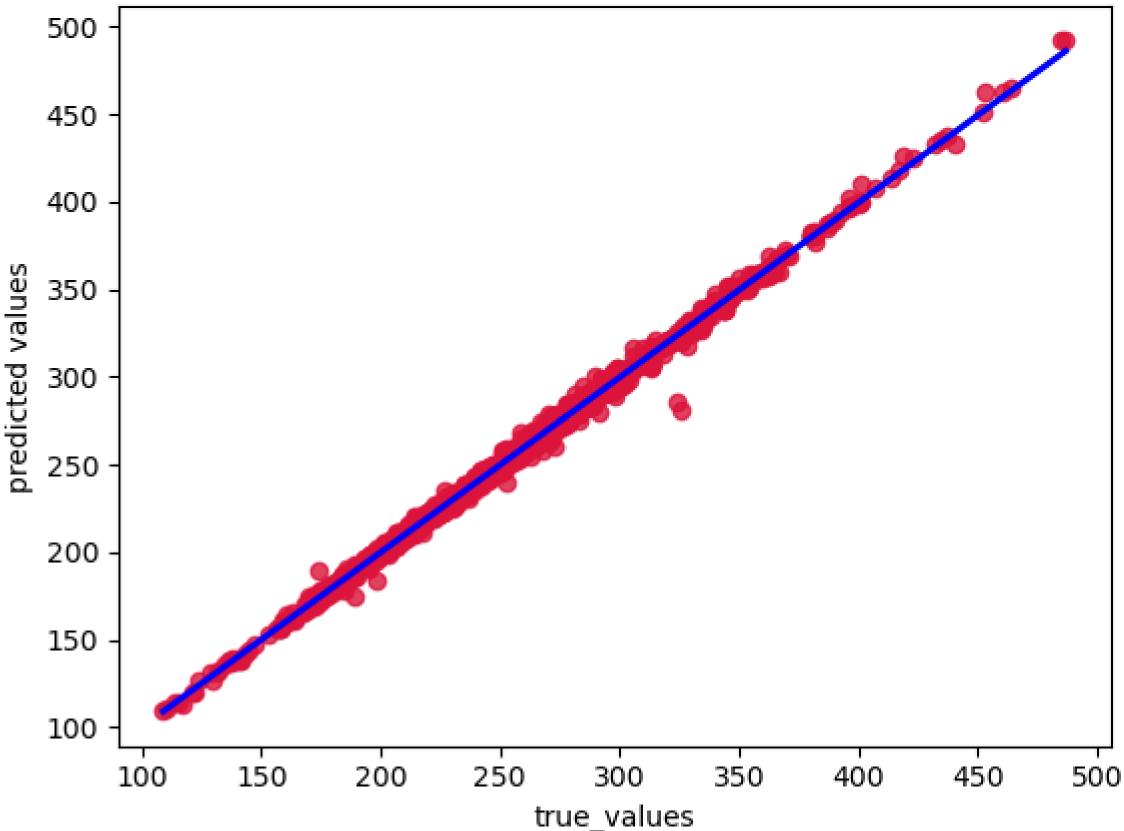
In [45]:
```
rf_model.fit(X_train, y_train) # fitting the training datasets to the model for tra
```

Out[45]:
```
▾ RandomForestRegressor
RandomForestRegressor()
```

In [46]:
```
# using the algorithm to predict values

rf_pred = rf_model.predict(X_test)
rf_pred
```

Out[46]:
```
array([172.8195    , 330.98916667, 111.31      , ..., 226.90333333,
       172.87      , 259.7875    ])
```

In [47]:
```
# visualization of the correlation between the actual and predicted dependent value

sns.regplot(x = y_test, y = rf_pred, color = 'crimson', line_kws = {"color":'blue'}
plt.xlabel('true_values', color = 'black')
plt.ylabel('predicted values', color = 'black')
plt.show()
```



model's metrics

In [48]:
```
# displaying metrics about the linear regression algorithm on the dataset

r_score = r2_score(y_test, rf_pred) * 100
rfMSE = mean_squared_error(y_test, rf_pred)
rfMAE = mean_absolute_error(y_test, rf_pred)

print(f"r2_score: {round(r_score, 4)} %\n\nMean Squared Error: {round(rfMSE, 4)}\n\
```

```
r2_score: 99.6866 %

Mean Squared Error: 10.5721

mean absolute error:2.050224504678562
```

DECISION TREE REGRESSOR

In [49]:
```
dt_model = DecisionTreeRegressor() # constructing the decision tree regressor model
```

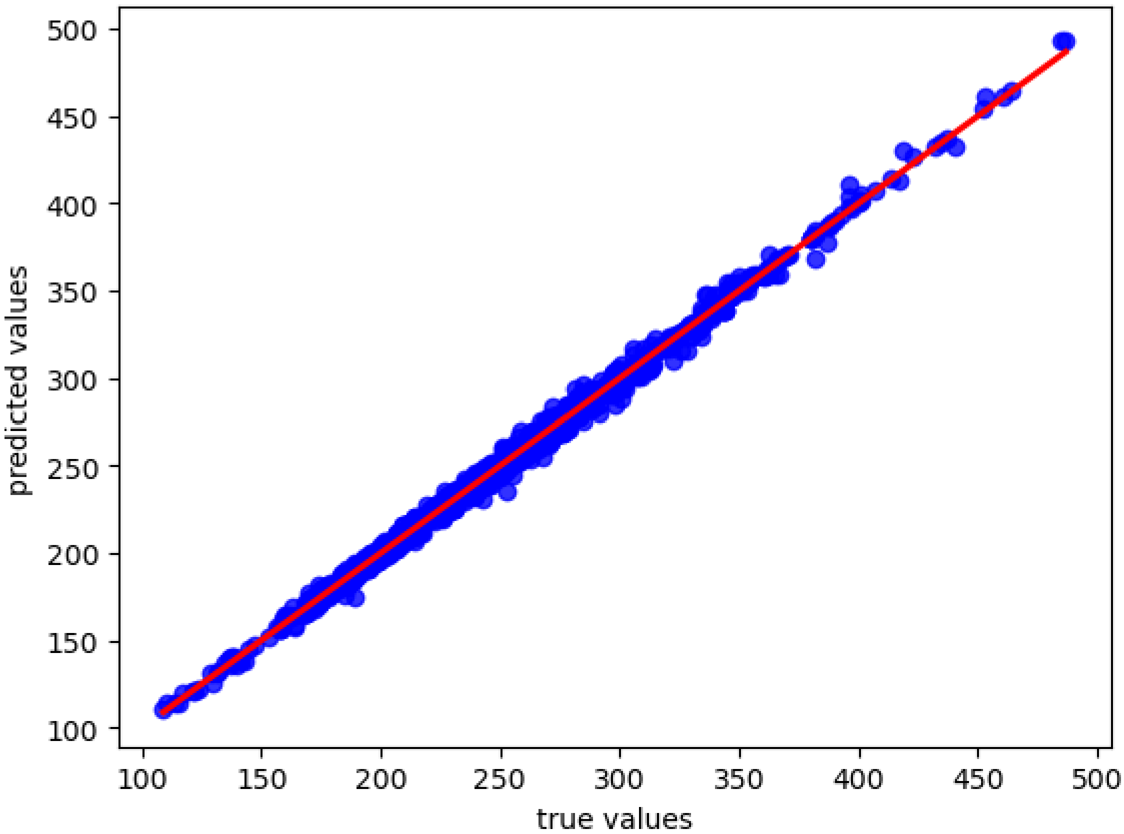
In [50]:
```
dt_model.fit(X_train, y_train) # fitting the training datasets to the model for tra
```

Out[50]:
```
▾ DecisionTreeRegressor
DecisionTreeRegressor()
```

In [51]:
```
# using the algorithm to predict values

dt_pred = dt_model.predict(X_test)
dt_pred
```

Out[51]:
```
array([173., 331., 114., ..., 228., 173., 258.])
```

In [52]:
```
# visualization of the correlation between the actual and predicted dependent value

sns.regplot(x = y_test, y = dt_pred, color = 'blue', line_kws = {"color":'red'})
plt.xlabel('true values', color = 'black')
plt.ylabel('predicted values', color = 'black')
plt.show()
```



model metrics

In [53]:
```
# displaying metrics about the linear regression algorithm on the dataset

dt_MSE = mean_squared_error(y_test, dt_pred)
r_score1 = r2_score(y_test, dt_pred) * 100
dtMAE = mean_absolute_error(y_test, dt_pred)

print(f"r2_score: {round(r_score1, 4)}\n\nMean Squared Error: {round(dt_MSE, 4)}\n\
```

```
r2_score: 99.651

Mean Squared Error: 11.7743

mean absolute error:2.1913285600636434
```

In [54]:
```
# my_prediction = lr/dt_model.predict([[]])
# my_prediction
```

OBSERVATIONS AND FINDINGS

1. most of the car makers are ford followed by chevrolet, bmw, mercedes benz, others.

2. The car model preferences are F-150 FFFV with the highest, followed by F-150 FFV 4X4, MUSTANG, FOCUS FFV, and others.

3. the most used vehicle class was SUV-SMALL, MID-SIZE,COMPACT,SUV-STANDARD, and others.

4. The overwhelming majority of vehicles have automatic transmissions. This reflects consumer preferences for ease of driving, especially in urban environments.

5. The most common fuel type is regular gasoline, which may indicate cost considerations for consumers. The presence of premium gasoline vehicles suggests a niche market for performance-oriented or luxury vehicles.

6. Most vehicles have engine sizes between 2 and 4 liters, which is typical for modern vehicles aiming for a balance between power and fuel efficiency.

7. The data shows that fuel consumption in the city is generally higher than on the highway, which is typical due to stop-and-go traffic conditions. This insight can help consumers make informed decisions based on their driving habits.

8. The distribution of CO_2 emissions indicates that most vehicles emit between 200-250 g/km. This information is crucial for understanding the environmental impact of the vehicle fleet and can inform policy decisions regarding emissions standards.

9. Factors that contribute to CO_2 emissions highly are:

a. engine size

b. Cylinders

c. Fuel consumption city

d. Fuel consumption highway

e. Fuel consumption comb