Importing required packages

In [29]:
```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import numpy as np
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
from sklearn.linear_model import LinearRegression
%matplotlib inline
```

Let's download and import the data on Fuel Consumption using *pandas* `read_csv()` method.

Download Dataset

Understanding the Data

`FuelConsumption.csv`:

We have downloaded a fuel consumption dataset, **`FuelConsumptionData.csv`**, which contains model-specific fuel consumption ratings and estimated carbon dioxide emissions for new light-duty vehicles for retail sale in Canada.

- **MODELYEAR** e.g. 2014
- **MAKE** e.g. Acura
- **MODEL** e.g. ILX
- **VEHICLE CLASS** e.g. SUV
- **ENGINE SIZE** e.g. 4.7
- **CYLINDERS** e.g 6
- **TRANSMISSION** e.g. A6
- **FUEL CONSUMPTION in CITY(L/100 km)** e.g. 9.9
- **FUEL CONSUMPTION in HWY (L/100 km)** e.g. 8.9
- **FUEL CONSUMPTION COMB (L/100 km)** e.g. 9.2
- **CO2 EMISSIONS (g/km)** e.g. 182 --> low --> 0

Reading the data in

In [3]:
```
df = pd.read_csv("FuelConsumptionData.csv")

# take a look at the dataset
df.head()
```

Out[3]:

	MODELYEAR	MAKE	MODEL	VEHICLECLASS	ENGINESIZE	CYLINDERS	TRANSMISSION	FUELTYPE	FUELCONSUMPTION_CITY	FUEL
0	2014	ACURA	ILX	COMPACT	2.0	4	AS5	Z	9.9	
1	2014	ACURA	ILX	COMPACT	2.4	4	M6	Z	11.2	
2	2014	ACURA	ILX HYBRID	COMPACT	1.5	4	AV7	Z	6.0	
3	2014	ACURA	MDX 4WD	SUV - SMALL	3.5	6	AS6	Z	12.7	
4	2014	ACURA	RDX AWD	SUV - SMALL	3.5	6	AS6	Z	12.1	

Data Exploration

Let's first have a descriptive exploration on our data.

In [4]:
```
# summarize the data
df.describe()
```

Out[4]:

	MODELYEAR	ENGINESIZE	CYLINDERS	FUELCONSUMPTION_CITY	FUELCONSUMPTION_HWY	FUELCONSUMPTION_COMB	FUELCON
count	1067.0	1067.000000	1067.000000	1067.000000	1067.000000	1067.000000	
mean	2014.0	3.346298	5.794752	13.296532	9.474602	11.580881	
std	0.0	1.415895	1.797447	4.101253	2.794510	3.485595	
min	2014.0	1.000000	3.000000	4.600000	4.900000	4.700000	
25%	2014.0	2.000000	4.000000	10.250000	7.500000	9.000000	
50%	2014.0	3.400000	6.000000	12.600000	8.800000	10.900000	
75%	2014.0	4.300000	8.000000	15.550000	10.850000	13.350000	
max	2014.0	8.400000	12.000000	30.200000	20.500000	25.800000	

Let's select some features to explore more.

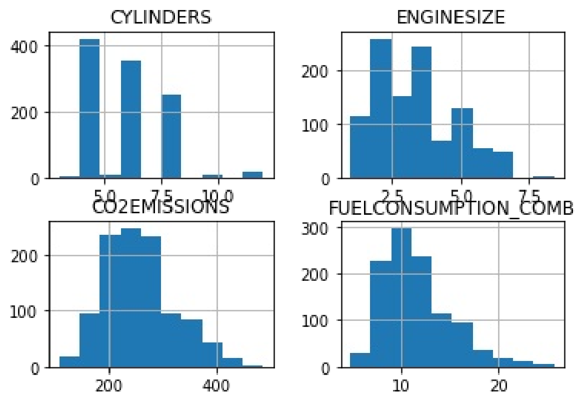
In [6]:
```
new_df = df[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_COMB','CO2EMISSIONS']]
new_df.head(7)
```

Out[6]:

	ENGINESIZE	CYLINDERS	FUELCONSUMPTION_COMB	CO2EMISSIONS
0	2.0	4	8.5	196
1	2.4	4	9.6	221
2	1.5	4	5.9	136
3	3.5	6	11.1	255
4	3.5	6	10.6	244
5	3.5	6	10.0	230
6	3.5	6	10.1	232

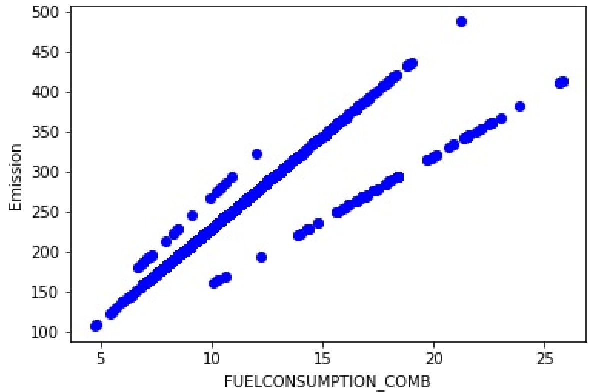
We can plot each of these features:

In [7]:
```
vis = new_df[['CYLINDERS','ENGINESIZE','CO2EMISSIONS','FUELCONSUMPTION_COMB']]
vis.hist()
plt.show()
```



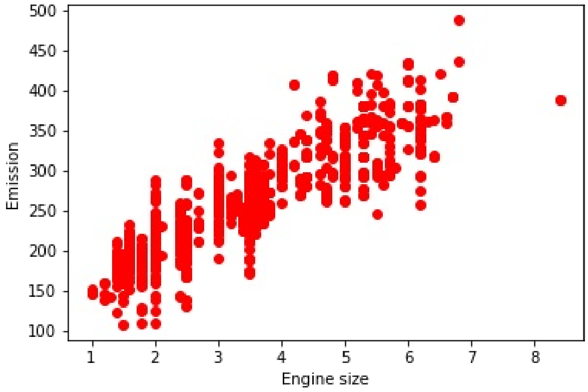
Now, let's plot each of these features against the Emission, to see how linear their relationship is:

In [9]:
```
plt.scatter(new_df.FUELCONSUMPTION_COMB, new_df.CO2EMISSIONS,  color='blue')
plt.xlabel("FUELCONSUMPTION_COMB")
plt.ylabel("Emission")
plt.show()
```

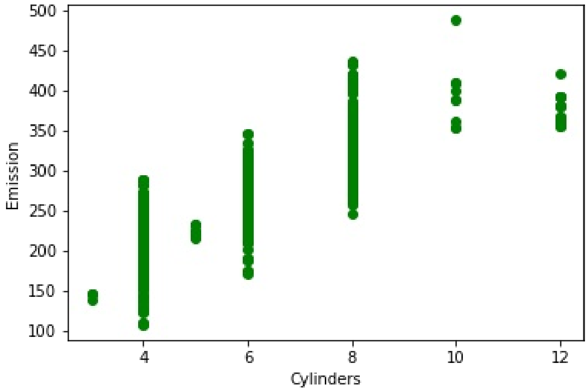


In [11]:
```
plt.scatter(new_df.ENGINESIZE, new_df.CO2EMISSIONS,  color='red')
plt.xlabel("Engine size")
plt.ylabel("Emission")
```

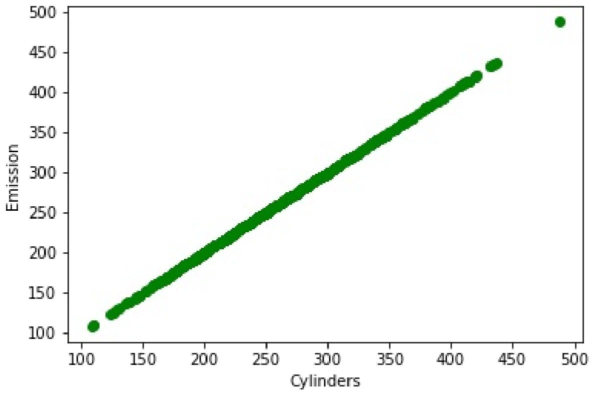
```
plt.show()
```



In [14]:
```
plt.scatter(new_df.CYLINDERS, new_df.CO2EMISSIONS, color='green')
plt.xlabel("Cylinders")
plt.ylabel("Emission")
plt.show()
```



In [15]:
```
plt.scatter(new_df.CO2EMISSIONS, new_df.CO2EMISSIONS, color='green')
plt.xlabel("Cylinders")
plt.ylabel("Emission")
plt.show()
```



Creating train and test dataset

Train/Test Split involves splitting the dataset into training and testing sets that are mutually exclusive. After which, you train with the training set and test with the testing set. This will provide a more accurate evaluation on out-of-sample accuracy because the testing dataset is not part of the dataset that have been used to train the model. Therefore, it gives us a better understanding of how well our model generalizes on new data.

This means that we know the outcome of each data point in the testing dataset, making it great to test with! Since this data has not been used to train the model, the model has no knowledge of the outcome of these data points. So, in essence, it is truly an out-of-sample testing.

Let's split our dataset into train and test sets. 80% of the entire dataset will be used for training and 20% for testing. We create a mask to select random rows using **np.random.rand()** function:

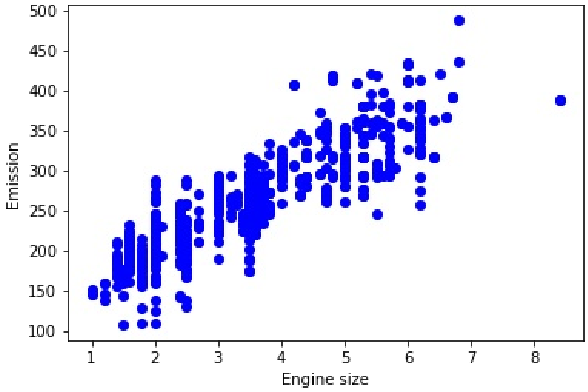
In [18]:
```
mask = np.random.rand(len(df)) < 0.8
train = new_df[mask]
test = new_df[~mask]
```

Simple Regression Model

Linear Regression fits a linear model with coefficients B = (B1, ..., Bn) to minimize the 'residual sum of squares' between the actual value y in the dataset, and the predicted value yhat using linear approximation.

Train data distribution

In [19]:
```
plt.scatter(train.ENGINESIZE, train.CO2EMISSIONS,  color='blue')
plt.xlabel("Engine size")
plt.ylabel("Emission")
plt.show()
```



Modeling

Using sklearn package to model data.

In [27]:
```
regr = LinearRegression()
train_x = np.asanyarray(train[['ENGINESIZE']])
train_y = np.asanyarray(train[['CO2EMISSIONS']])
regr.fit(train_x, train_y)
# The coefficients
print ('Coefficients: ', regr.coef_)
print ('Intercept: ',regr.intercept_)
```

```
Coefficients:  [[39.02421253]]
Intercept:  [126.05993983]
```

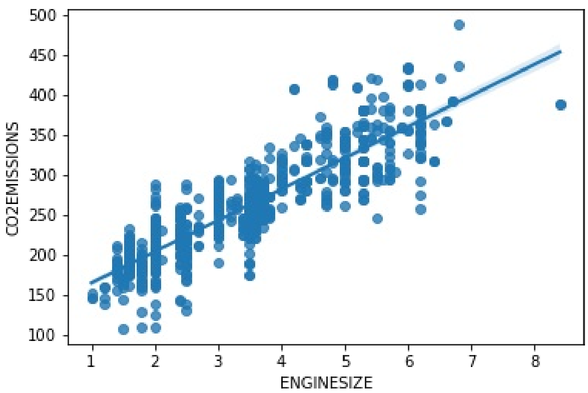
As mentioned before, **Coefficient** and **Intercept** in the simple linear regression, are the parameters of the fit line. Given that it is a simple linear regression, with only 2 parameters, and knowing that the parameters are the intercept and slope of the line, sklearn can estimate them directly from our data. Notice that all of the data must be available to traverse and calculate the parameters.

Plot outputs

We can plot the fit line over the data:

In [26]:
```
sns.regplot(x="ENGINESIZE", y="CO2EMISSIONS", data=train)
```

Out[26]:
```
<AxesSubplot:xlabel='ENGINESIZE', ylabel='CO2EMISSIONS'>
```



Evaluation

We compare the actual values and predicted values to calculate the accuracy of a regression model. Evaluation metrics provide a key role in the development of a model, as it provides insight to areas that require improvement.

There are different model evaluation metrics, lets use MSE here to calculate the accuracy of our model based on the test set:

- Mean Absolute Error: It is the mean of the absolute value of the errors. This is the easiest of the metrics to understand since it's just average error.

- Mean Squared Error (MSE): Mean Squared Error (MSE) is the mean of the squared error. It's more popular than Mean Absolute Error because the focus is geared more towards large errors. This is due to the squared term exponentially increasing larger errors in comparison to smaller ones.
- Root Mean Squared Error (RMSE).
- R-squared is not an error, but rather a popular metric to measure the performance of your regression model. It represents how close the data points are to the fitted regression line. The higher the R-squared value, the better the model fits your data. The best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse).

In [31]:
```
test_x = np.asanyarray(test[['ENGINESIZE']])
test_y = np.asanyarray(test[['CO2EMISSIONS']])
pred_y=regr.predict(test_x)
mse = mean_squared_error(test_y, pred_y)
mae = mean_absolute_error(test_y, pred_y)
score=r2_score(test_y, pred_y)
# display
print("Mean absolute error : " + str(mae))
print("Mean squared error : " + str(mse))
print("r2_score : " + str(score))
```

```
Mean absolute error : 23.67951423989859
Mean squared error : 948.3883780931031
r2_score : 0.7490554265559102
```

Another Way

In [33]:
```
from sklearn.metrics import r2_score

test_x = np.asanyarray(test[['ENGINESIZE']])
test_y = np.asanyarray(test[['CO2EMISSIONS']])
pred_y = regr.predict(test_x)

print("Mean absolute error: %.2f" % np.mean(np.absolute(pred_y - test_y)))
print("Residual sum of squares (MSE): %.2f" % np.mean((pred_y - test_y) ** 2))
print("R2-score: %.2f" % r2_score(test_y , pred_y) )
```

```
Mean absolute error: 23.68
Residual sum of squares (MSE): 948.39
R2-score: 0.75
```

Exercise

Lets see what the evaluation metrics are if we trained a regression model using the `CYLINDERS` feature.

Hints:

Start by selecting `CYLINDERS` as the train_x data from the `train` dataframe, then select `FUELCONSUMPTION_COMB` as the test_x data from the `test` dataframe. Now train a Linear Regression Model using the `train_x` you created and the `train_y` created previously.Find the predictions using the model's `predict` function and the `test_x` data. Finally use the `predictions` and the `test_y` data and find the Mean Absolute Error value using the `np.absolute` and `np.mean` function like done previously.

Thank you

Author

Moazzam Ali
