Simple Linear Regression Lab

importing needed packages

In [1]:
```
import matplotlib.pyplot as plt
import pandas as pd
import pylab as pl
import numpy as np
%matplotlib inline
```

Understanding the Data

`FuelConsumption.csv`:

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

- **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 [2]:
```
df = pd.read_csv("FuelConsumptionCo2.csv")

#take a look at the dataset

df.head()
```

Out[2]:

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

In [3]:

```
# summerize the data df is our dataframe
df.describe()
```

Out[3]:

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

Let's select some features to explore more.

cdf = features hayee ke mikhahim roye anha kar konim

In [4]:

```
cdf = df[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_COMB','CO2EMISSIONS']]
cdf.head(9)
```

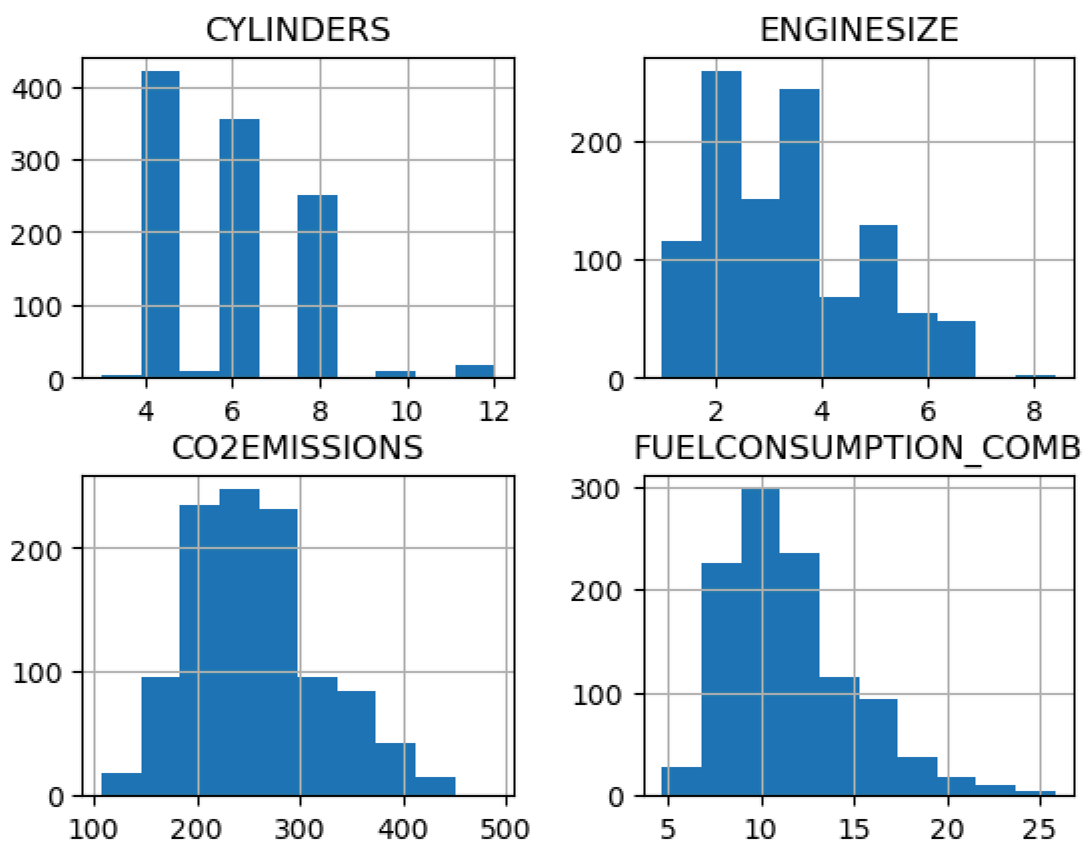
Out[4]:

	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
7	3.7	6	11.1	255
8	3.7	6	11.6	267

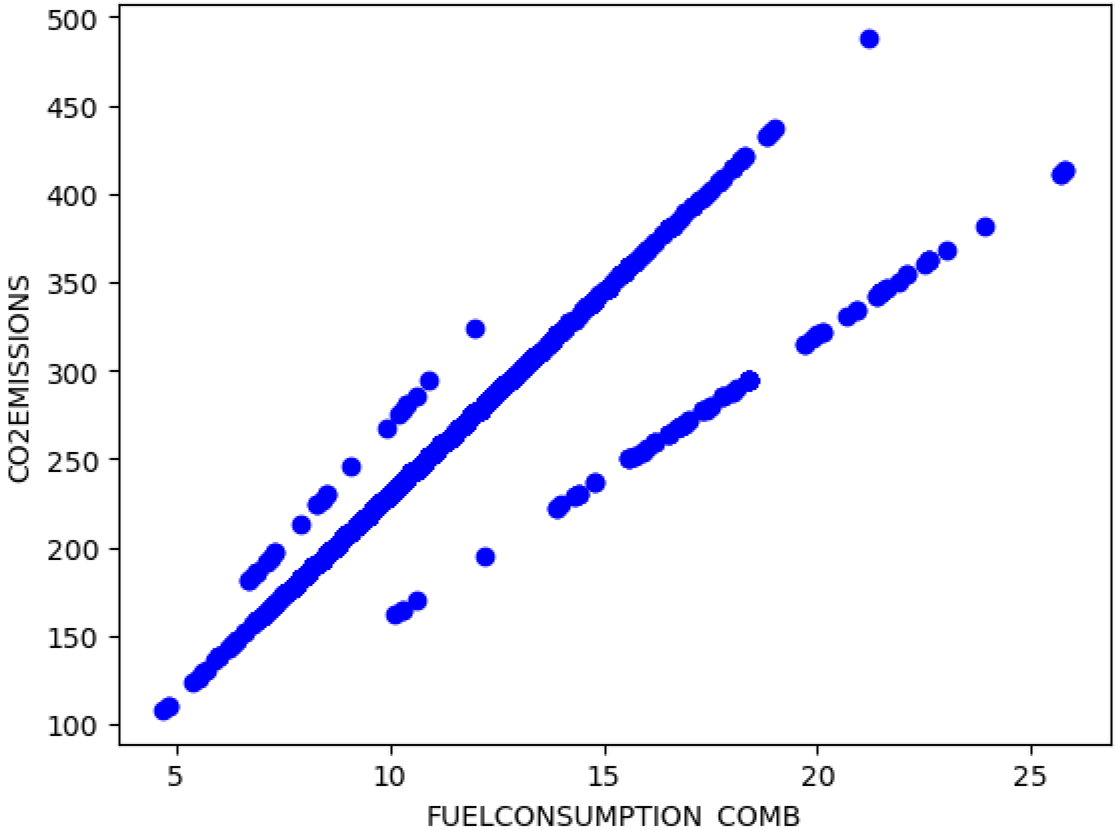
We can plot each of these features:

viz = visualization

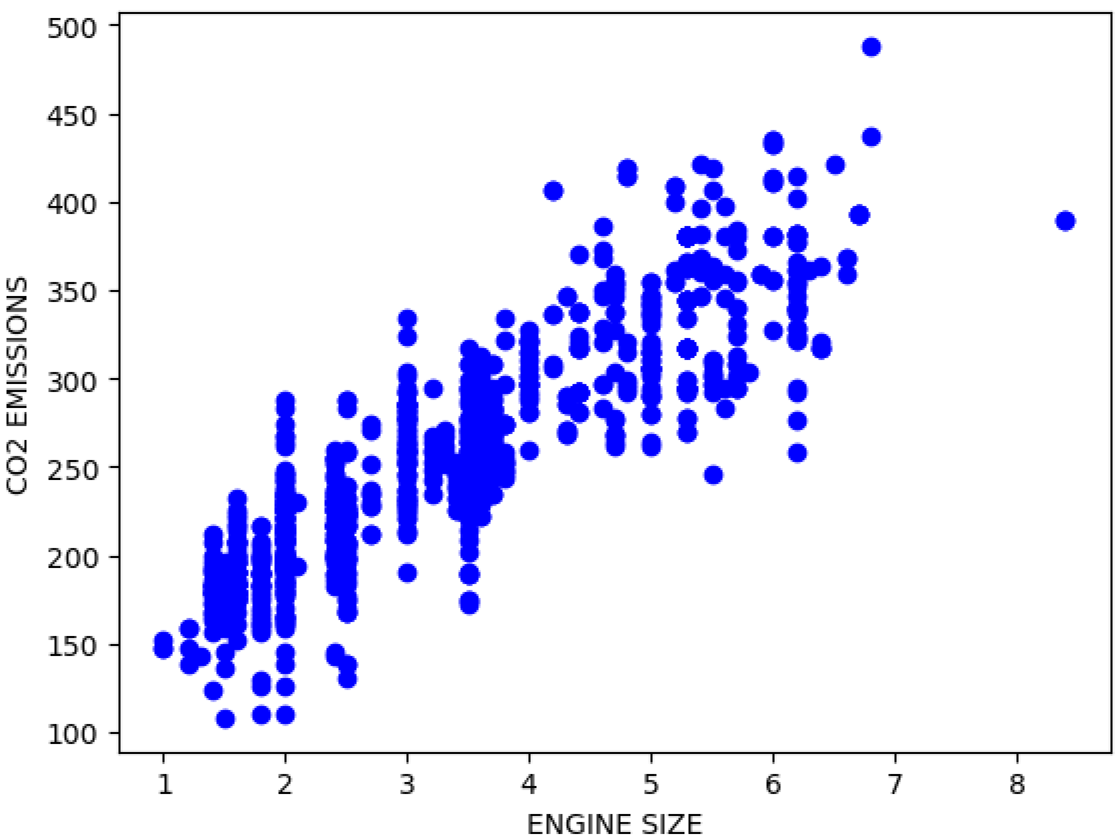
In [5]:
```
viz = cdf[['CYLINDERS','ENGINESIZE','CO2EMISSIONS','FUELCONSUMPTION_COMB']]
viz.hist() #hist = histogram
plt.show()
```



In [6]:
```
plt.scatter(cdf.FUELCONSUMPTION_COMB, cdf.CO2EMISSIONS, color = 'blue')
plt.xlabel('FUELCONSUMPTION_COMB')
plt.ylabel('CO2EMISSIONS')
plt.show()
```



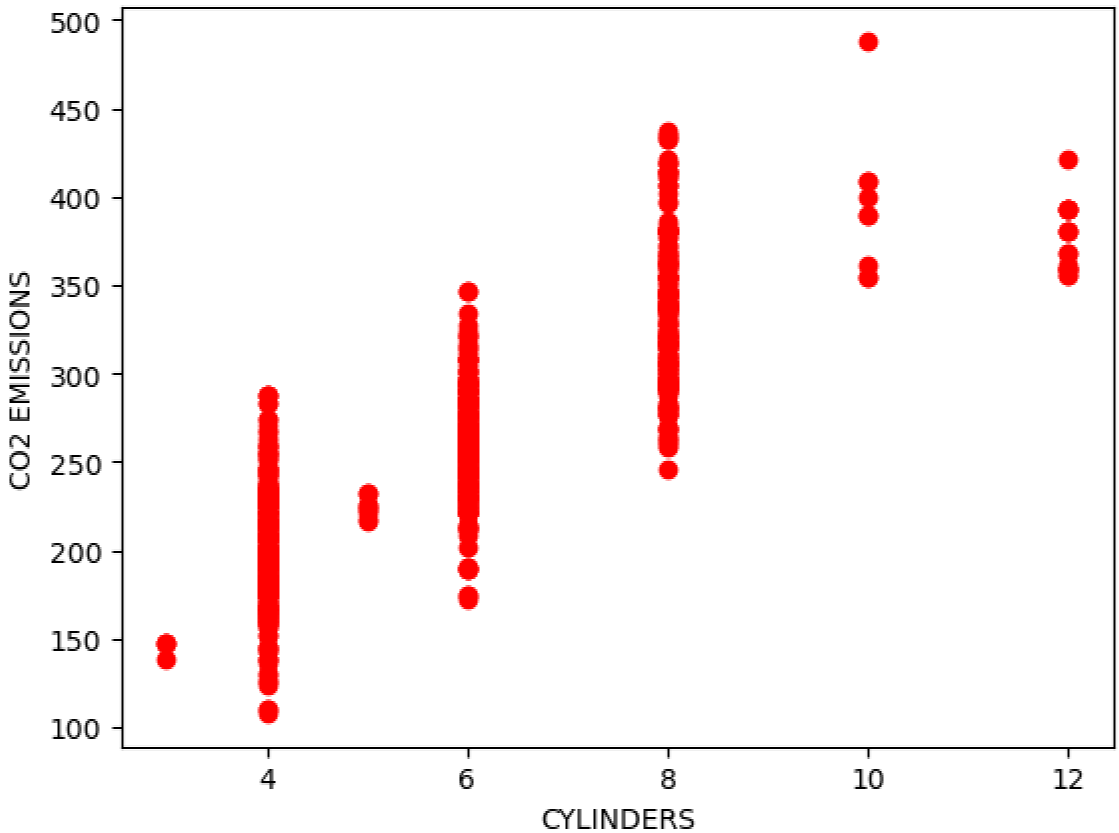
In [7]:
```
plt.scatter(cdf.ENGINESIZE, cdf.CO2EMISSIONS, color = 'Blue')
plt.xlabel('ENGINE SIZE')
plt.ylabel('CO2 EMISSIONS')
plt.show()
```



Practice

Plot **CYLINDER** vs the Emission, to see how linear is their relationship is:

In [8]:
```
plt.scatter(cdf.CYLINDERS, cdf.CO2EMISSIONS, color = "red")
plt.xlabel('CYLINDERS')
plt.ylabel('CO2 EMISSIONS')
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 [9]:
```
msk = np.random.rand(len(df)) < 0.8
train = cdf[msk]
test = cdf[~msk]
print(msk)
print(~msk)
print(cdf)
print(train)
print(test)
```

```
[ True False  True ...  True  True False]
[False  True False ... False False  True]
      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
...          ...        ...                   ...           ...
1062         3.0          6                  11.8           271
1063         3.2          6                  11.5           264
1064         3.0          6                  11.8           271
1065         3.2          6                  11.3           260
1066         3.2          6                  12.8           294

[1067 rows x 4 columns]
      ENGINESIZE  CYLINDERS  FUELCONSUMPTION_COMB  CO2EMISSIONS
0            2.0          4                   8.5           196
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
...          ...        ...                   ...           ...
1060         3.0          6                  11.5           264
1061         3.2          6                  11.2           258
1062         3.0          6                  11.8           271
1064         3.0          6                  11.8           271
1065         3.2          6                  11.3           260

[861 rows x 4 columns]
      ENGINESIZE  CYLINDERS  FUELCONSUMPTION_COMB  CO2EMISSIONS
1            2.4          4                   9.6           221
10           2.4          4                   9.8           225
17           4.7          8                  15.4           354
31           4.0          8                  11.3           260
33           3.0          6                  11.2           258
...          ...        ...                   ...           ...
1035         1.8          4                   8.2           189
1038         2.0          4                   9.0           207
1046         2.5          5                   9.8           225
1063         3.2          6                  11.5           264
1066         3.2          6                  12.8           294

[206 rows x 4 columns]
```

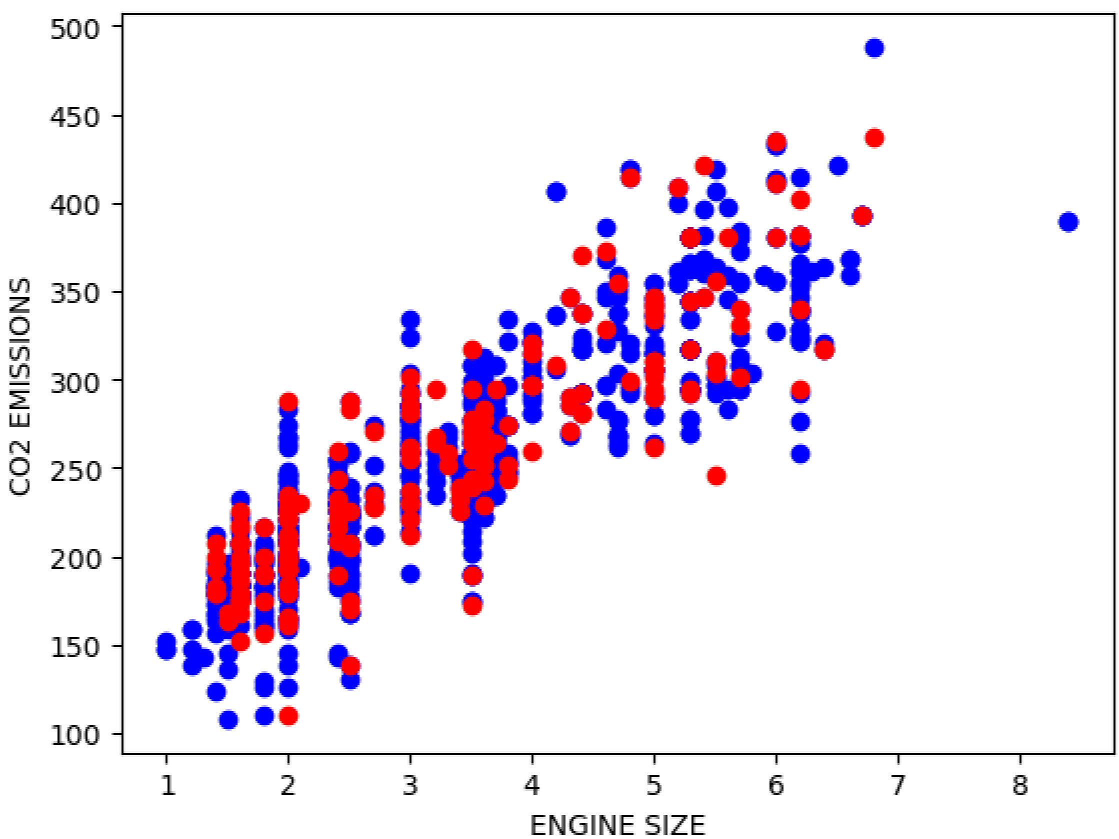
Simple Regression Model

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

whit tow args

In [10]:
```
fig = plt.figure()
ax1 = fig.add_subplot(111)
ax1.scatter(train.ENGINESIZE, train.CO2EMISSIONS, color = 'Blue')
ax1.scatter(test.ENGINESIZE, test.CO2EMISSIONS, color = 'red')
plt.xlabel('ENGINE SIZE')
plt.ylabel('CO2 EMISSIONS')
plt.show()
```



Modeling

Using sklearn package to model data.

In [13]:
```
from sklearn import linear_model
regr = linear_model.LinearRegression()
train_x = np.asanyarray(train[['ENGINESIZE']])
train_y = np.asanyarray(train[['CO2EMISSIONS']])
regr.fit(train_x,train_y)
# The coefficients
print(regr)
print ('Coefficients: ', regr.coef_)
print ('Intercept: ',regr.intercept_)
```

```
LinearRegression()
Coefficients:  [[38.96579805]]
Intercept:  [125.3523798]
```

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.

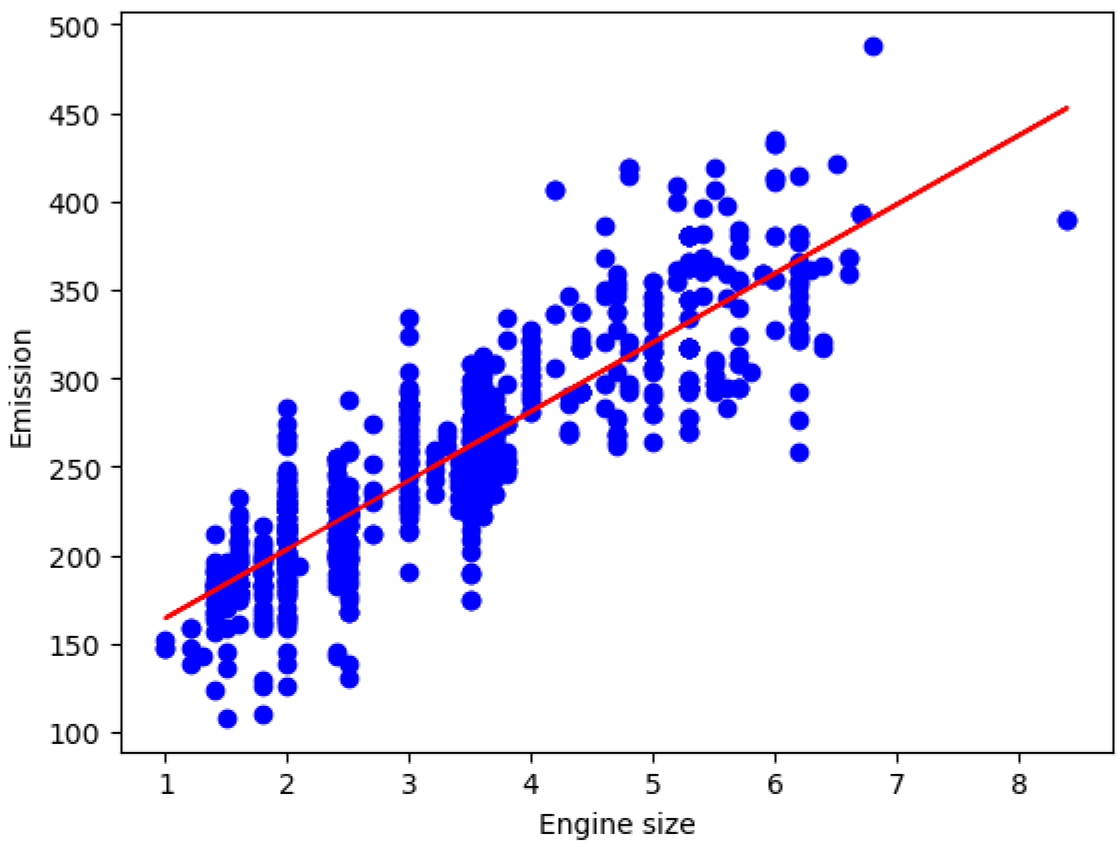
Plot outputs

In [16]:

```
plt.scatter(train.ENGINESIZE, train.CO2EMISSIONS, color = 'Blue')
plt.plot(train_x, regr.coef_[0][0]*train_x + regr.intercept_[0], '-r')
plt.xlabel("Engine size")
plt.ylabel("Emission")
```

Out[16]:

```
Text(0, 0.5, 'Emission')
```



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 [19]:

```
test_x = np.asanyarray(test[['ENGINESIZE']])
test_y = np.asanyarray(test[['CO2EMISSIONS']])
test_y_ = regr.predict(test_x)
print("Mean absolute error: %.2f" % np.mean(np.absolute(test_y_ - test_y)))
print("Residual sum of squares (MSE): %.2f" % np.mean((test_y_ - test_y) ** 2))
print("R2-score: %.2f" % r2_score(test_y , test_y_) )
```

```
Mean absolute error: 24.39
Residual sum of squares (MSE): 1042.78
R2-score: 0.76
```