Multiple Linear Regression

Objectives

After completing this lab you will be able to:

- Use scikit-learn to implement Multiple Linear Regression
- Create a model, train,test and use the model

Importing Needed packages

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

Insert data set

In [4]:
```
df = pd.read_csv("FuelConsumptionCo2.csv")
# take a look at the dataset
df.head()
```

Out[4]:

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

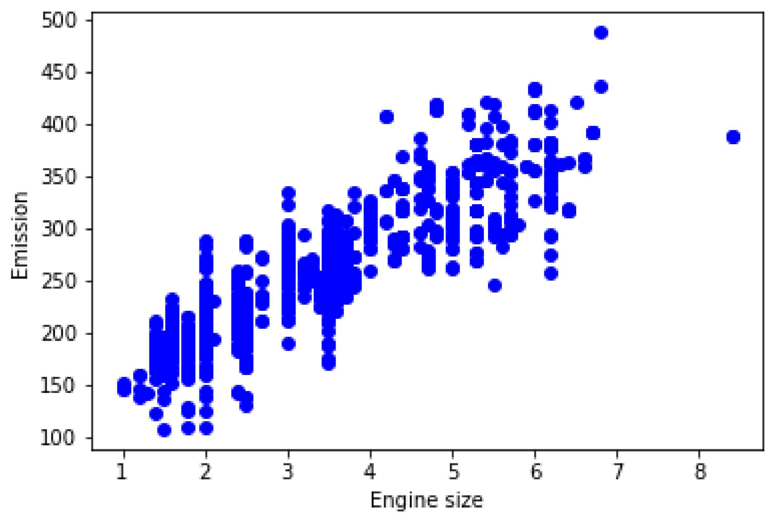
Lets select some features that we want to use for regression.

In [5]:
```
cdf = df[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_CITY','FUELCONSUMPTION_HWY','
cdf.head(9)
```

Out[5]:

	ENGINESIZE	CYLINDERS	FUELCONSUMPTION_CITY	FUELCONSUMPTION_HWY	FUELCONSUMP
0	2.0	4	9.9	6.7	
1	2.4	4	11.2	7.7	
2	1.5	4	6.0	5.8	
3	3.5	6	12.7	9.1	
4	3.5	6	12.1	8.7	
5	3.5	6	11.9	7.7	
6	3.5	6	11.8	8.1	
7	3.7	6	12.8	9.0	
8	3.7	6	13.4	9.5	

In [6]:
```
plt.scatter(cdf.ENGINESIZE, cdf.CO2EMISSIONS,  color='blue')
plt.xlabel("Engine size")
plt.ylabel("Emission")
plt.show()
```



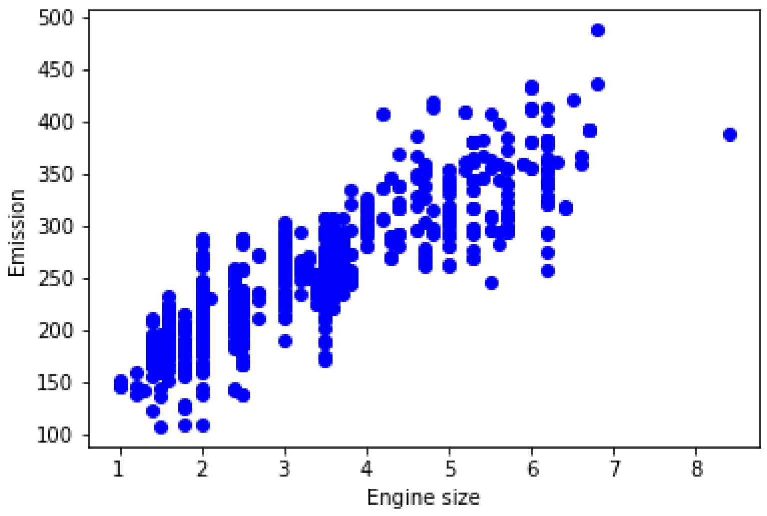
Creating train and test dataset

Train/Test Split involves splitting the dataset into training and testing sets respectively, which 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 data. It is more realistic for real world problems.

This means that we know the outcome of each data point in this dataset, making it great to test with! And 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's truly an out-of-sample testing.

In [7]:
```
msk = np.random.rand(len(df)) < 0.8
train = cdf[msk]
test = cdf[~msk]
```

In [8]:
```
plt.scatter(train.ENGINESIZE, train.CO2EMISSIONS,  color='blue')
plt.xlabel("Engine size")
plt.ylabel("Emission")
plt.show()
```



Multiple Regression Model

In reality, there are multiple variables that predict the Co2emission. When more than one independent variable is present, the process is called multiple linear regression. For example, predicting co2emission using FUELCONSUMPTION_COMB, EngineSize and Cylinders of cars. The good thing here is that Multiple linear regression is the extension of simple linear regression model.

In [10]:
```
from sklearn import linear_model
regr = linear_model.LinearRegression()
x = np.asanyarray(train[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_COMB']])
y = np.asanyarray(train[['CO2EMISSIONS']])
regr.fit (x, y)
# The coefficients
print ('Coefficients: ', regr.coef_)
```

```
Coefficients:  [[10.60503289  7.5526726   9.45344405]]
```

As mentioned before, **Coefficient** and **Intercept** , are the parameters of the fit line. Given that it is a multiple linear regression, with 3 parameters, and knowing that the parameters are the intercept and coefficients of hyperplane, sklearn can estimate them from our data. Scikit-learn uses plain Ordinary Least Squares method to solve this problem.

Ordinary Least Squares (OLS)

OLS is a method for estimating the unknown parameters in a linear regression model. OLS chooses the parameters of a linear function of a set of explanatory variables by minimizing the sum of the squares of the differences between the target dependent variable and those predicted by the linear function. In other words, it tries to minimizes the sum of squared errors (SSE) or mean squared error (MSE) between the target variable (y) and our predicted output ($\hat{y}$) over all samples in the dataset.

OLS can find the best parameters using of the following methods:

```
- Solving the model parameters analytically using closed-form equations
- Using an optimization algorithm (Gradient Descent, Stochastic Gradient
Descent, Newton’s Method, etc.)
```

Prediction

In [11]:
```
y_hat= regr.predict(test[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_COMB']])
x = np.asanyarray(test[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_COMB']])
y = np.asanyarray(test[['CO2EMISSIONS']])
print("Residual sum of squares: %.2f"
      % np.mean((y_hat - y) ** 2))

# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % regr.score(x, y))
```

```
Residual sum of squares: 571.73
Variance score: 0.86
```

explained variance regression score:
If $\hat{y}$ is the estimated target output, y the corresponding (correct) target output, and Var is Variance, the square of the standard deviation, then the explained variance is estimated as follow:

$$\texttt{explainedVariance}(y, \hat{y}) = 1 - \frac{Var{y - \hat{y}}}{Var{y}}$$

The best possible score is 1.0, lower values are worse.

Practice

Try to use a multiple linear regression with the same dataset but this time use __FUEL CONSUMPTION in CITY__ and __FUEL CONSUMPTION in HWY__ instead of FUELCONSUMPTION_COMB. Does it result in better accuracy?

```
In [14]: # write your code here

         regr = linear_model.LinearRegression()
         x = np.asanyarray(train[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_CITY','FUELCON
         y = np.asanyarray(train[['CO2EMISSIONS']])
         regr.fit (x, y)
         print ('Coefficients: ', regr.coef_)
         y_= regr.predict(test[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_CITY','FUELCONSU
         x = np.asanyarray(test[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_CITY','FUELCONS
         y = np.asanyarray(test[['CO2EMISSIONS']])
         print("Residual sum of squares: %.2f"% np.mean((y_ - y) ** 2))
         print('Variance score: %.2f' % regr.score(x, y))
```

```
Coefficients:  [[10.73560483  6.98061017  6.67199189  2.29915137]]
Residual sum of squares: 580.41
Variance score: 0.86
```

THE END