# Polynomial Regression

## Objectives

After completing this lab you will be able to:

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

# Table of contents



---

### Importing Needed packages

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

## Downloading Data

To download the data, we will use !wget to download it from IBM Object Storage.

In [2]:
```
#!wget -O FuelConsumption.csv https://cf-courses-data.s3.us.cloud-object-storage.appd

import urllib.request
url = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloper
filename = 'FuelConsumption.csv'
urllib.request.urlretrieve(url, filename)
```

Out[2]:
```
('FuelConsumption.csv', <http.client.HTTPMessage at 0x1953dd965b0>)
```

## 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 [3]:
```
df = pd.read_csv("FuelConsumption.csv")

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

Out[3]:

| | MODELYEAR | MAKE | MODEL | VEHICLECLASS | ENGINESIZE | CYLINDERS | TRANSMISSION | FUELTYPE | FUELCC |
|---|---|---|---|---|---|---|---|---|---|
| **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 | |

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

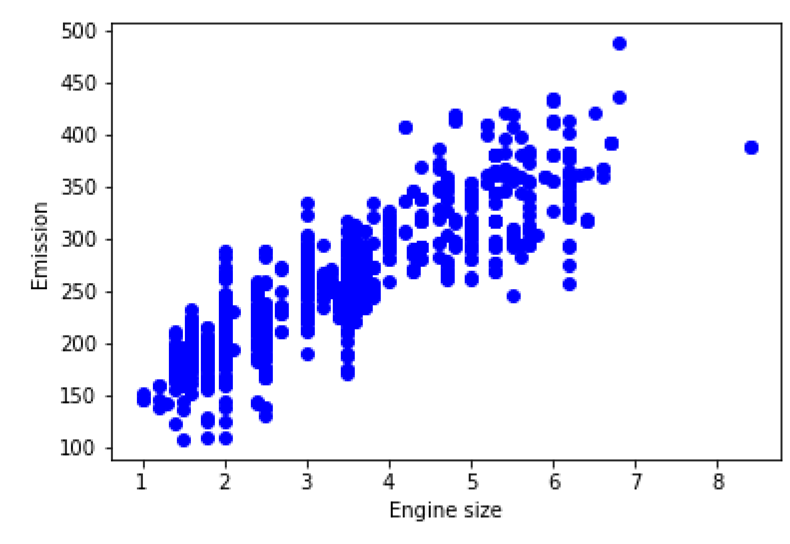
In [4]:
```
cdf = df[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_COMB','CO2EMISSIONS']]
cdf.head(9)
```

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 |

Lets plot Emission values with respect to Engine size:

In [5]:
```
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.

In [6]:
```
msk = np.random.rand(len(df)) < 0.8
train = cdf[msk]
test = cdf[~msk]
```

## Polynomial regression

Sometimes, the trend of data is not really linear, and looks curvy. In this case we can use Polynomial regression methods. In fact, many different regressions exist that can be used to fit whatever the dataset looks like, such as quadratic, cubic, and so on, and it can go on and on to infinite degrees.

In essence, we can call all of these, polynomial regression, where the relationship between the independent variable x and the dependent variable y is modeled as an nth degree polynomial in x. Lets say you want to have a polynomial regression (let's make 2 degree polynomial):

$$y = b + \theta_1 x + \theta_2 x^2$$

Now, the question is: how we can fit our data on this equation while we have only x values, such as **Engine Size**? Well, we can create a few additional features: 1, $x$, and $x^2$.

**PolynomialFeatures()** function in Scikit-learn library, drives a new feature sets from the original feature set. That is, a matrix will be generated consisting of all polynomial combinations of the features with degree less than or equal to the specified degree. For example, lets say the original feature set has only one feature, *ENGINESIZE*. Now, if we select the degree of the polynomial to be 2, then it generates 3 features, degree=0, degree=1 and degree=2:

In [7]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn import linear_model
train_x = np.asanyarray(train[['ENGINESIZE']])
train_y = np.asanyarray(train[['CO2EMISSIONS']])

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


poly = PolynomialFeatures(degree=2)
train_x_poly = poly.fit_transform(train_x)
train_x_poly
```

Out[7]:
```
array([[ 1.  ,  2.  ,  4.  ],
       [ 1.  ,  3.5 , 12.25],
       [ 1.  ,  3.5 , 12.25],
       ...,
       [ 1.  ,  3.  ,  9.  ],
       [ 1.  ,  3.2 , 10.24],
       [ 1.  ,  3.2 , 10.24]])
```

**fit_transform** takes our x values, and output a list of our data raised from power of 0 to power of 2 (since we set the degree of our polynomial to 2).

The equation and the sample example is displayed below.

$$\begin{bmatrix} v_1\\ v_2\\ \vdots\\ v_n \end{bmatrix} \longrightarrow \begin{bmatrix} [1 & v_1 & v_1^2]\\ [1 & v_2 & v_2^2]\\ \vdots & \vdots & \vdots\\ [1 & v_n & v_n^2] \end{bmatrix}$$

$$\begin{bmatrix} 2.\\ 2.4\\ 1.5\\ \vdots \end{bmatrix} \longrightarrow \begin{bmatrix} [1 & 2. & 4.]\\ [1 & 2.4 & 5.76]\\ [1 & 1.5 & 2.25]\\ \vdots & \vdots & \vdots \end{bmatrix}$$

It looks like feature sets for multiple linear regression analysis, right? Yes. It Does. Indeed, Polynomial regression is a special case of linear regression, with the main idea of how do you select your features. Just consider replacing the $x$ with $x_1$, $x_1^2$ with $x_2$, and so on. Then the degree 2 equation would be turn into:

$$y = b + \theta_1 x_1 + \theta_2 x_2$$

Now, we can deal with it as 'linear regression' problem. Therefore, this polynomial regression is considered to be a special case of traditional multiple linear regression. So, you can use the same mechanism as linear regression to solve such a problems.

so we can use **LinearRegression()** function to solve it:

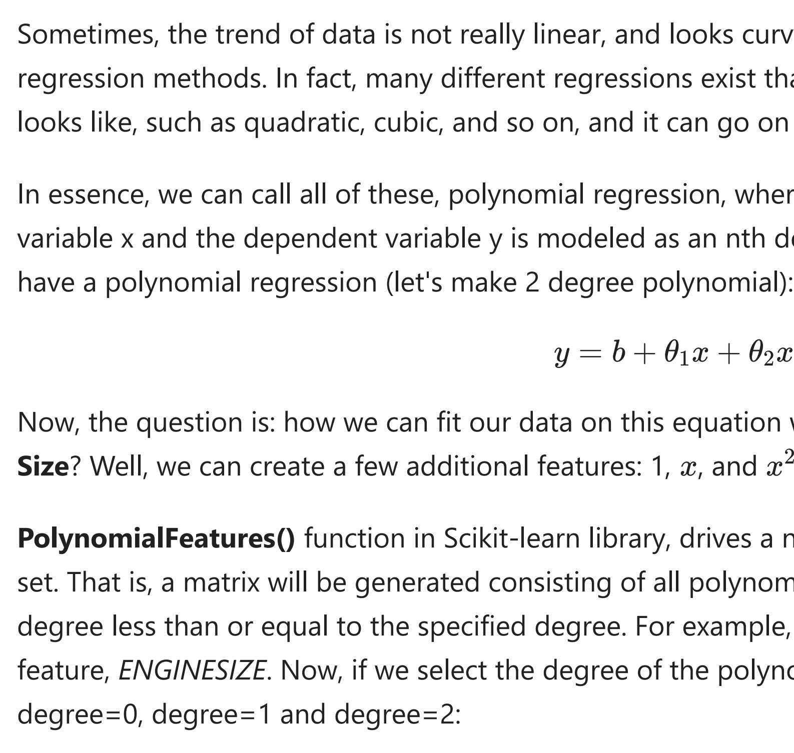
In [8]:
```
clf = linear_model.LinearRegression()
train_y_ = clf.fit(train_x_poly, train_y)
# The coefficients
print ('Coefficients: ', clf.coef_)
print ('Intercept: ',clf.intercept_)
```

```
Coefficients:  [[ 0.         46.23082214 -0.9891437 ]]
Intercept:  [114.51245477]
```

As mentioned before, **Coefficient** and **Intercept** , are the parameters of the fit curvy line. Given that it is a typical multiple linear regression, with 3 parameters, and knowing that the parameters are the intercept and coefficients of hyperplane, sklearn has estimated them from our new set of feature sets. Lets plot it:

In [9]:
```
plt.scatter(train.ENGINESIZE, train.CO2EMISSIONS,  color='blue')
XX = np.arange(0.0, 10.0, 0.1)
yy = clf.intercept_[0]+ clf.coef_[0][1]*XX+ clf.coef_[0][2]*np.power(XX, 2)
plt.plot(XX, yy, '-r' )
plt.xlabel("Engine size")
plt.ylabel("Emission")
```

Out[9]:
```
Text(0, 0.5, 'Emission')
```




## Evaluation

In [10]:
```
from sklearn.metrics import r2_score

test_x_poly = poly.fit_transform(test_x)
test_y_ = clf.predict(test_x_poly)

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: 25.26
Residual sum of squares (MSE): 1079.76
R2-score: 0.62
```

## Practice

Try to use a polynomial regression with the dataset but this time with degree three (cubic). Does it result in better accuracy?

In [11]:
```
poly3 = PolynomialFeatures(degree=3)
train_x_poly3 = poly3.fit_transform(train_x)
clf3 = linear_model.LinearRegression()
train_y3_ = clf3.fit(train_x_poly3, train_y)

# The coefficients
print ('Coefficients: ', clf3.coef_)
print ('Intercept: ',clf3.intercept_)
plt.scatter(train.ENGINESIZE, train.CO2EMISSIONS,  color='blue')
XX = np.arange(0.0, 10.0, 0.1)
yy = clf3.intercept_[0]+ clf3.coef_[0][1]*XX + clf3.coef_[0][2]*np.power(XX, 2) + clf
plt.plot(XX, yy, '-r' )
plt.xlabel("Engine size")
plt.ylabel("Emission")
test_x_poly3 = poly3.fit_transform(test_x)
test_y3_ = clf3.predict(test_x_poly3)
print("Mean absolute error: %.2f" % np.mean(np.absolute(test_y3_ - test_y)))
print("Residual sum of squares (MSE): %.2f" % np.mean((test_y3_ - test_y) ** 2))
print("R2-score: %.2f" % r2_score(test_y3_ , test_y) )
```

```
Coefficients:  [[ 0.         33.87409783  2.48501583 -0.29425684]]
Intercept:  [127.39644721]
Mean absolute error: 25.15
Residual sum of squares (MSE): 1067.16
R2-score: 0.63
```