Non Linear Regression Analysis

If the data shows a curvy trend, then linear regression will not produce very accurate results when compared to a non-linear regression because, as the name implies, linear regression presumes that the data is linear. Let's learn about non linear regressions and apply an example on python. In this notebook, we fit a non-linear model to the datapoints corrensponding to China's GDP from 1960 to 2014.

Importing required libraries

In [1]:

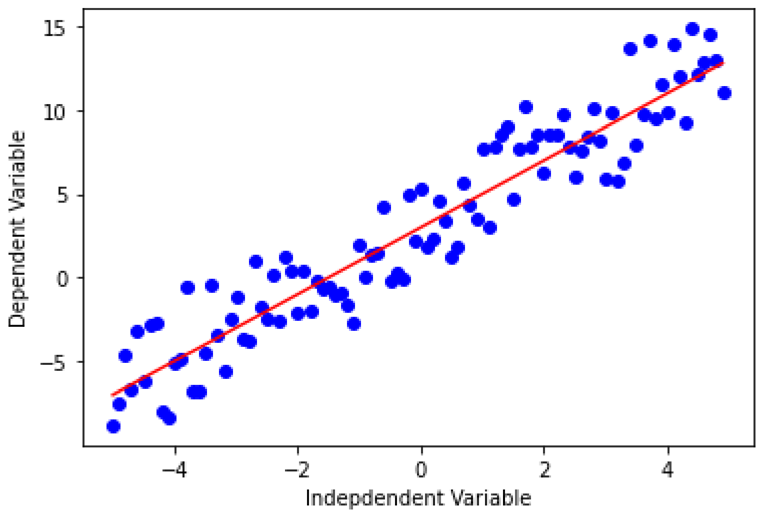
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Though Linear regression is very good to solve many problems, it cannot be used for all datasets. First recall how linear regression, could model a dataset. It models a linear relation between a dependent variable y and independent variable x. It had a simple equation, of degree 1, for example y = $2x$ + 3.

In [2]:

```
x = np.arange(-5.0, 5.0, 0.1)

##You can adjust the slope and intercept to verify the changes in the graph
y = 2*(x) + 3
y_noise = 2 * np.random.normal(size=x.size)
ydata = y + y_noise
#plt.figure(figsize=(8,6))
plt.plot(x, ydata,  'bo')
plt.plot(x,y, 'r')
plt.ylabel('Dependent Variable')
plt.xlabel('Indepdendent Variable')
plt.show()
```



Non-linear regressions are a relationship between independent variables x and a dependent variable y which result in a non-linear function modeled data. Essentially any relationship that is not linear can be termed as non-linear, and is usually represented by the polynomial of k degrees (maximum power of x).

$$y = ax^3 + bx^2 + cx + d$$

Non-linear functions can have elements like exponentials, logarithms, fractions, and others. For example:

$$y = \log(x)$$

Or even, more complicated such as :

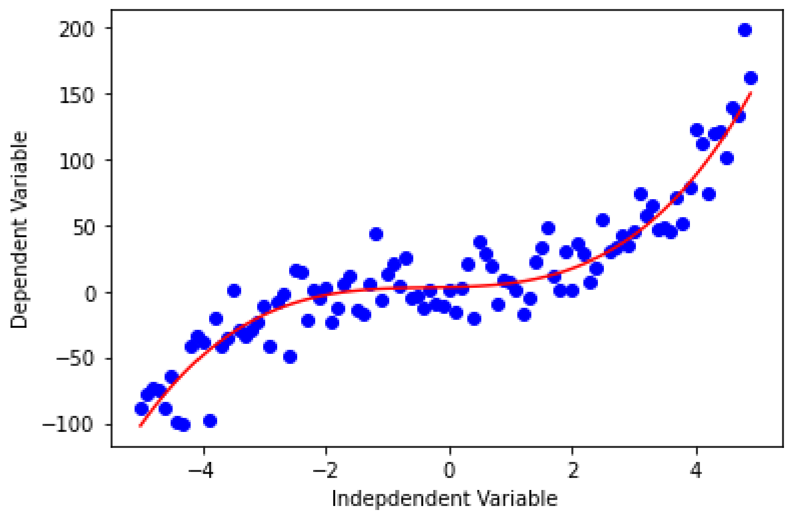
$$y = \log(ax^3 + bx^2 + cx + d)$$

Let's take a look at a cubic function's graph.

In [3]:

```
x = np.arange(-5.0, 5.0, 0.1)

##You can adjust the slope and intercept to verify the changes in the graph
y = 1*(x**3) + 1*(x**2) + 1*x + 3
y_noise = 20 * np.random.normal(size=x.size)
ydata = y + y_noise
plt.plot(x, ydata,  'bo')
plt.plot(x,y, 'r')
plt.ylabel('Dependent Variable')
plt.xlabel('Indepdendent Variable')
plt.show()
```



In [4]:

```
y_noise
```

Out[4]:

```
array([ 1.39307931e+01,  1.81904083e+01,  1.65858529e+01,  9.12214011e+00,
       -1.00427185e+01,  8.27389865e+00, -3.17631910e+01, -3.84330189e+01,
        1.66034262e+01,  1.88019852e+01,  1.07895096e+01, -5.28427727e+01,
        2.14193637e+01, -3.76042017e+00, -3.58522580e-01,  3.25167261e+01,
       -1.25646900e+00, -9.20963020e+00, -6.69306400e+00, -3.31881777e+00,
        7.51275971e+00, -2.60122548e+01,  6.36674398e+00,  1.02805921e+01,
       -3.93683780e+01,  2.45362196e+01,  2.22695232e+01, -1.59653603e+01,
        5.43845903e+00, -1.70022436e+00,  6.19312224e+00, -2.10738372e+01,
       -1.07859316e+01,  6.02166494e+00,  1.17167799e+01, -1.50116667e+01,
       -1.80606022e+01,  4.70062396e+00,  4.26592363e+01, -8.59493924e+00,
        1.13968175e+01,  1.83682833e+01,  2.43493894e+00,  2.30911092e+01,
       -7.01627198e+00, -6.08943596e+00, -1.59788715e+01, -1.95123428e+00,
       -1.19095358e+01, -1.36986262e+01, -1.24463185e+00, -1.83487328e+01,
       -1.24072402e+00,  1.71174923e+01, -2.39917030e+01,  3.28529494e+01,
        2.39334665e+01,  1.48184003e+01, -1.50818310e+01,  2.99170658e+00,
        8.31319693e-01, -5.94534917e+00, -2.44989990e+01, -1.30651678e+01,
        1.35270969e+01,  2.32218151e+01,  3.63032012e+01, -1.01702649e+00,
       -1.30209249e+01,  1.37999426e+01, -1.58308183e+01,  1.66210581e+01,
        8.28552062e+00, -1.53940963e+01, -6.85201903e+00,  2.67183750e+01,
       -6.27675789e-01, -2.14617802e-01,  7.09334731e+00, -4.71482078e+00,
        3.63524211e+00,  2.83064522e+01,  7.27651264e+00,  1.16840824e+01,
       -1.11980474e+01, -1.29880109e+01, -2.06645577e+01, -5.19355238e-02,
       -2.54141106e+01, -2.50534199e+00,  3.56278879e+01,  1.90893523e+01,
       -2.52553555e+01,  1.38466705e+01,  9.67085086e+00, -1.72255858e+01,
        1.33366065e+01, -1.01653091e+00,  5.64343881e+01,  1.25838987e+01])
```

As you can see, this function has x^3 and x^2 as independent variables. Also, the graphic of this function is not a straight line over the 2D plane. So this is a non-linear function.

Some other types of non-linear functions are:

Quadratic

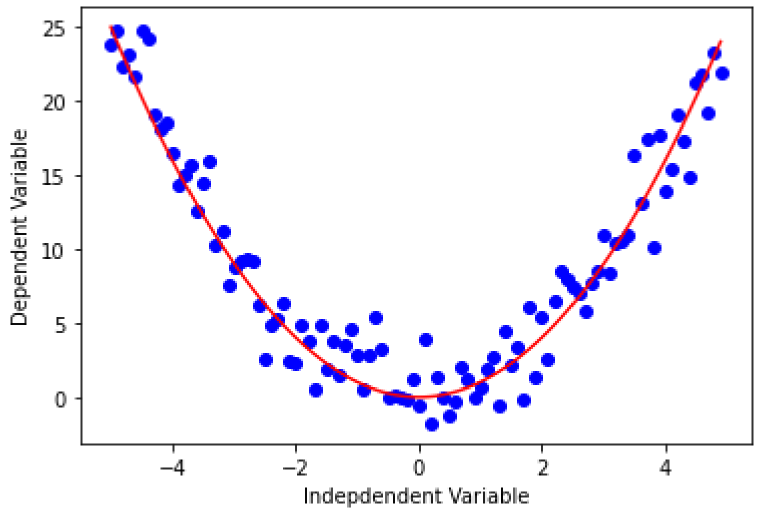
$$Y = X^2$$

In [5]:

```
x = np.arange(-5.0, 5.0, 0.1)

##You can adjust the slope and intercept to verify the changes in the graph

y = np.power(x,2)
y_noise = 2 * np.random.normal(size=x.size)
ydata = y + y_noise
plt.plot(x, ydata,  'bo')
plt.plot(x,y, 'r')
plt.ylabel('Dependent Variable')
plt.xlabel('Indepdendent Variable')
plt.show()
```



Exponential

An exponential function with base c is defined by

$$Y = a + bc^X$$

where b ≠0, c > 0 , c ≠1, and x is any real number. The base, c, is constant and the exponent, x, is a variable.

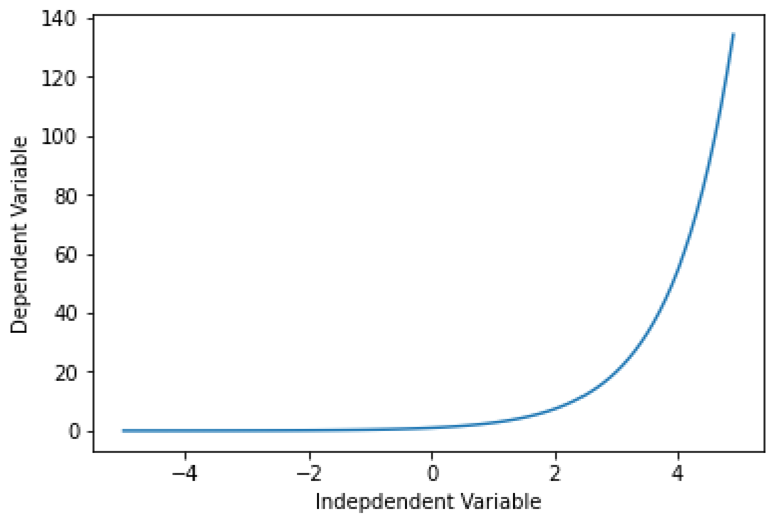
In [6]:

```
X = np.arange(-5.0, 5.0, 0.1)

##You can adjust the slope and intercept to verify the changes in the graph

Y= np.exp(X)

plt.plot(X,Y)
plt.ylabel('Dependent Variable')
plt.xlabel('Indepdendent Variable')
plt.show()
```



Logarithmic

The response y is a results of applying logarithmic map from input x's to output variable y. It is one of the simplest form of **log()**: i.e.

$$y = \log(x)$$

Please consider that instead of x, we can use X, which can be polynomial representation of the x's. In general form it would be written as

$$y = \log(X)$$

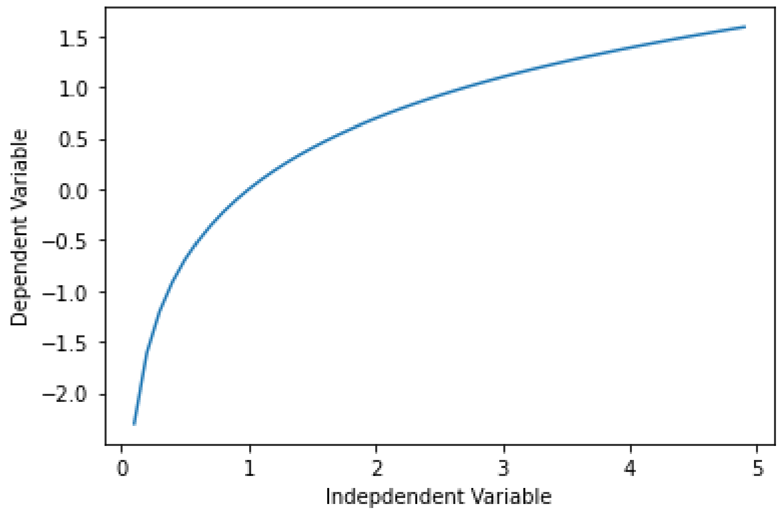
In [7]:

```
X = np.arange(-5.0, 5.0, 0.1)

Y = np.log(X)

plt.plot(X,Y)
plt.ylabel('Dependent Variable')
plt.xlabel('Indepdendent Variable')
plt.show()
```

```
<ipython-input-7-0b6a51fd782b>:3: RuntimeWarning: invalid value encountered
in log
  Y = np.log(X)
```



Sigmoidal/Logistic

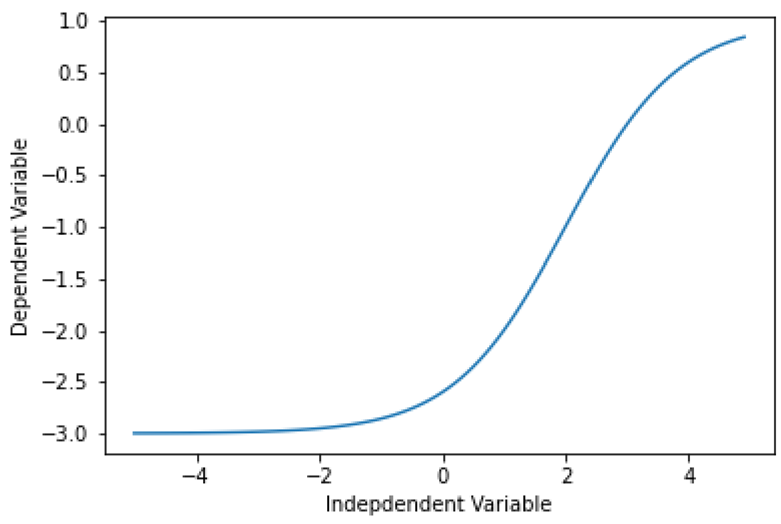
$$Y = a + \frac{b}{1 + c^{(X-d)}}$$

In [8]:

```
X = np.arange(-5.0, 5.0, 0.1)


Y = 1-4/(1+np.power(3, X-2))

plt.plot(X,Y)
plt.ylabel('Dependent Variable')
plt.xlabel('Indepdendent Variable')
plt.show()
```


Dependent Variable
Indepdendent Variable