Importing required packages

In [24]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
import matplotlib.pyplot as plt
%matplotlib inline
```

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

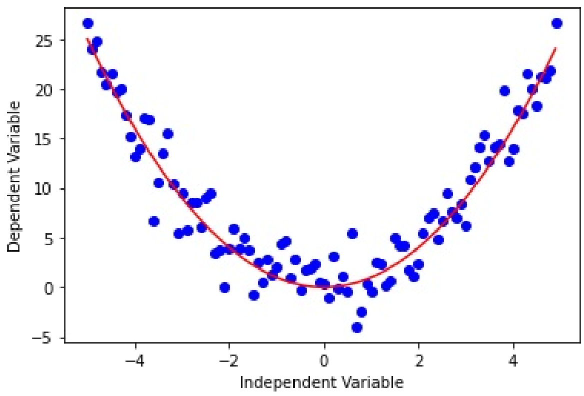
Types of non linear functions

Quadratic

In [7]:
```
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('Independent Variable')
plt.show()
```



Exponetial

An exponential function with base c is defined by

$$Y = a + bc^X$$

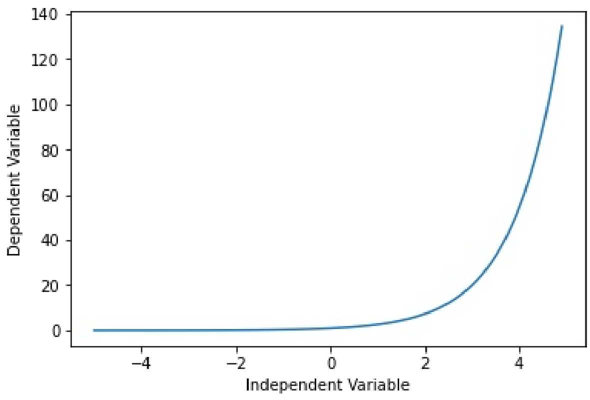
where b ≠0, c > 0 , c ≠1, and x is any real number. The base, c, is constant and the exponent, x, is a variable.

In [8]:
```
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('Independent Variable')
plt.show()
```



Logarithmic

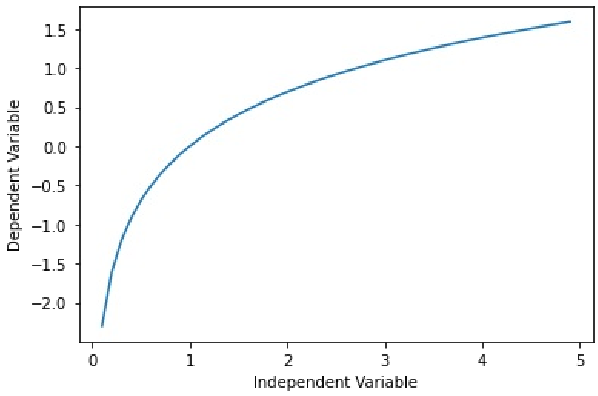
The response y is a results of applying the logarithmic map from the input x to the output y. It is one of the simplest form of **log()**:

In [9]:
```
X = np.arange(-5.0, 5.0, 0.1)

Y = np.log(X)

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

```
C:\Users\Meer Moazzam\AppData\Local\Temp\ipykernel_8368\2233958358.py:3: RuntimeWarning: invalid value encounte
red in log
  Y = np.log(X)
```



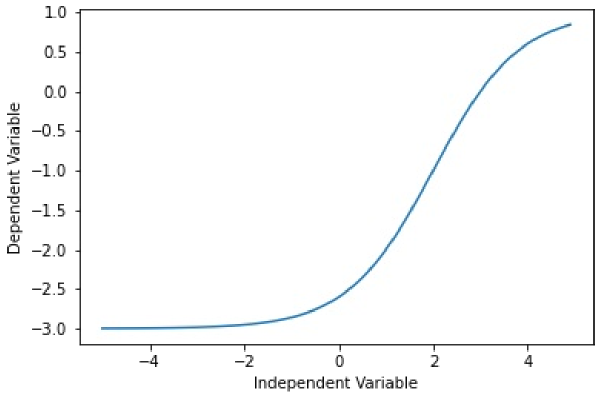
Sigmoidal/Logistic

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

In [10]:
```
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('Independent Variable')
plt.show()
```



Non-Linear Regression example

For an example, we're going to try and fit a non-linear model to the datapoints corresponding to China's GDP from 1960 to 2014. We download a dataset with two columns, the first, a year between 1960 and 2014, the second, China's corresponding annual gross domestic income in US dollars for that year.

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

Download Dataset

Reading the data

In [13]:
```
df = pd.read_csv("china_gdp.csv")

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

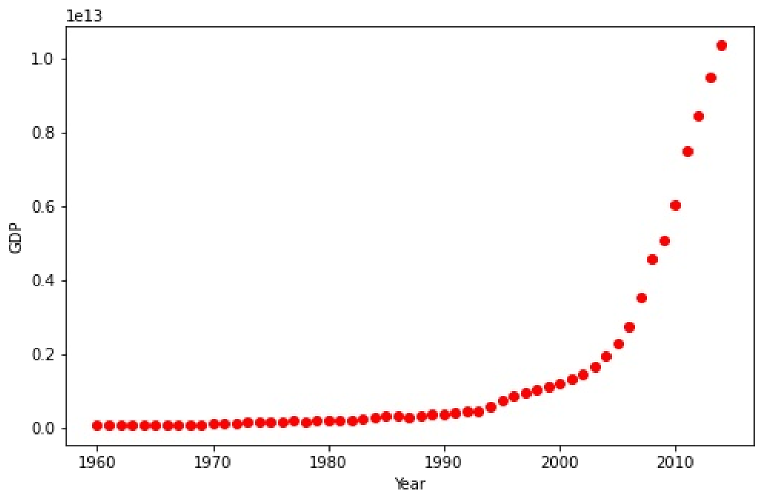
Out[13]:

	Year	Value
0	1960	5.918412e+10
1	1961	4.955705e+10
2	1962	4.668518e+10
3	1963	5.009730e+10
4	1964	5.906225e+10

Visualize the Dataset

This is what the datapoints look like. It kind of looks like an either logistic or exponential function. The growth starts off slow, then from 2005 on forward, the growth is very significant. And finally, it decelerates slightly in the 2010s.

In [14]:
```
plt.figure(figsize=(8,5))
x_data, y_data = (df["Year"].values, df["Value"].values)
plt.plot(x_data, y_data, 'ro')
plt.ylabel('GDP')
plt.xlabel('Year')
plt.show()
```

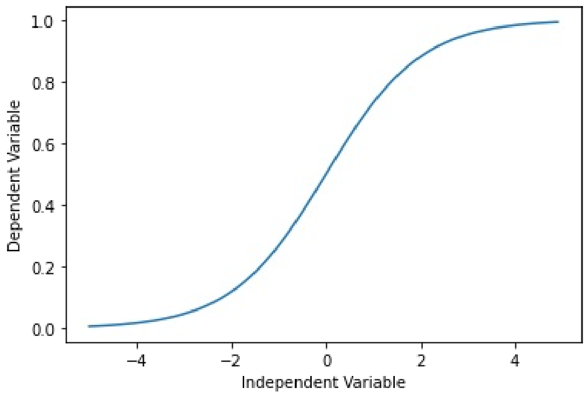


Choosing a model

From an initial look at the plot, we determine that the logistic function could be a good approximation, since it has the property of starting with a slow growth, increasing growth in the middle, and then decreasing again at the end; as illustrated below:

In [15]:
```
X = np.arange(-5.0, 5.0, 0.1)
Y = 1.0 / (1.0 + np.exp(-X))

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



The formula for the logistic function is the following:

$$\hat{Y} = \frac{1}{1 + e^{-\beta\_1(X-\beta\_2)}}$$

β_1: Controls the curve's steepness,

β_2: Slides the curve on the x-axis.

Building The Model

Now, let's build our regression model and initialize its parameters.

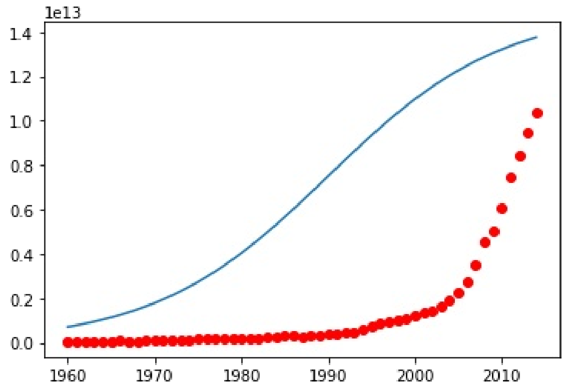
In [16]:
```
def sigmoid(x, Beta_1, Beta_2):
     y = 1 / (1 + np.exp(-Beta_1*(x-Beta_2)))
     return y
```

In [17]:
```
beta_1 = 0.10
beta_2 = 1990.0

#logistic function
Y_pred = sigmoid(x_data, beta_1 , beta_2)

#plot initial prediction against datapoints
plt.plot(x_data, Y_pred*15000000000000.)
plt.plot(x_data, y_data, 'ro')
```

Out[17]:
```
[<matplotlib.lines.Line2D at 0x1dc8a672670>]
```



Our task here is to find the best parameters for our model. Lets first normalize our x and y:

In [18]:
```
# Lets normalize our data
xdata =x_data/max(x_data)
ydata =y_data/max(y_data)
```

How we find the best parameters for our fit line?

we can use **curve_fit** which uses non-linear least squares to fit our sigmoid function, to data. Optimize values for the parameters so that the sum of the squared residuals of sigmoid(xdata, *popt) - ydata is minimized.

popt are our optimized parameters.

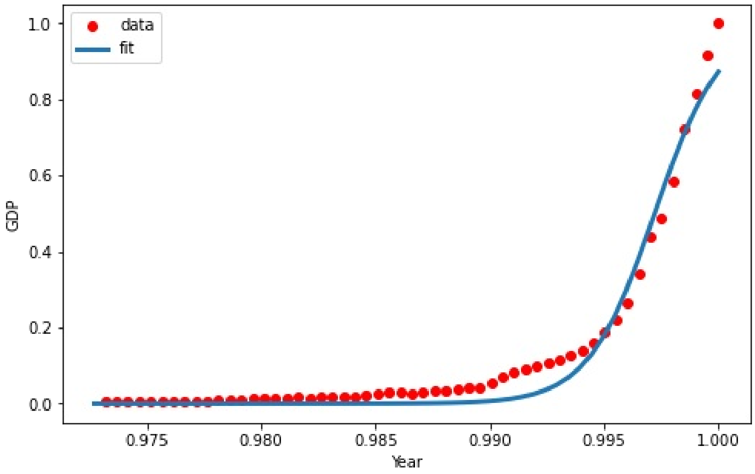
In [19]:
```
from scipy.optimize import curve_fit
popt, pcov = curve_fit(sigmoid, xdata, ydata)
#print the final parameters
print(" beta_1 = %f, beta_2 = %f" % (popt[0], popt[1]))
```

```
 beta_1 = 690.451711, beta_2 = 0.997207
```

Now we plot our resulting regression model.

Now we plot our resulting regression model.

In [21]:
```
x = np.linspace(1960, 2015, 55)
x = x/max(x)
plt.figure(figsize=(8,5))
y = sigmoid(x, *popt)
plt.plot(xdata, ydata, 'ro', label='data')
plt.plot(x,y, linewidth=3.0, label='fit')
plt.legend(loc='best')
plt.ylabel('GDP')
plt.xlabel('Year')
plt.show()
```



In [23]:
```
X_train, X_test, y_train, y_test=train_test_split(X,Y,test_size=0.2)
```

In [27]:
```
# build the model using train set
popt, pcov = curve_fit(sigmoid, X_train, y_train)

# predict using test set
y_pred = sigmoid(X_test, *popt)

# evaluation
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
score=r2_score(y_test, y_pred)
# display
print("Mean absolute error : " + str(mae))
print("Mean squared error : " + str(mse))
print("r2_score : " + str(score))
```

```
Mean absolute error : 5.533683932953948e-10
Mean squared error : 6.070838991856661e-19
r2_score : 1.0
```

Thank you

Author

Moazzam Ali
