REGRESSION

Simple Linear Regression

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

In [6]:
```
df = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-
df.head()
```

Out[6]:

	MODELYEAR	MAKE	MODEL	VEHICLECLASS	ENGINESIZE	CYLINDERS	TRANSMISSION	FUELTYPE	FUELCONSUMPTION_CITY	FU
0	2014	ACURA	ILX	COMPACT	2.0	4	AS5	Z	9.9	
1	2014	ACURA	ILX	COMPACT	2.4	4	M6	Z	11.2	
2	2014	ACURA	ILX HYBRID	COMPACT	1.5	4	AV7	Z	6.0	
3	2014	ACURA	MDX 4WD	SUV - SMALL	3.5	6	AS6	Z	12.7	
4	2014	ACURA	RDX AWD	SUV - SMALL	3.5	6	AS6	Z	12.1	

In [8]:
```
df.describe()
```

Out[8]:

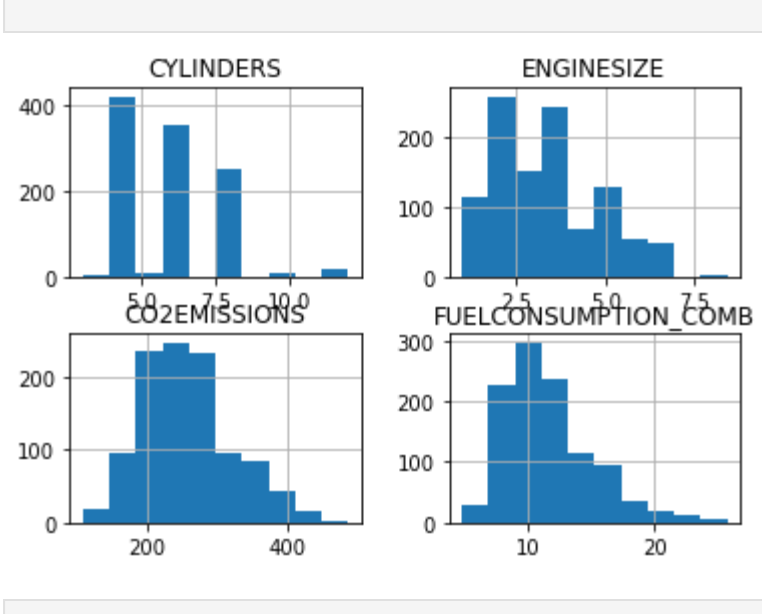
	MODELYEAR	ENGINESIZE	CYLINDERS	FUELCONSUMPTION_CITY	FUELCONSUMPTION_HWY	FUELCONSUMPTION_COMB	FUEL
count	1067.0	1067.000000	1067.000000	1067.000000	1067.000000	1067.000000	
mean	2014.0	3.346298	5.794752	13.296532	9.474602	11.580881	
std	0.0	1.415895	1.797447	4.101253	2.794510	3.485595	
min	2014.0	1.000000	3.000000	4.600000	4.900000	4.700000	
25%	2014.0	2.000000	4.000000	10.250000	7.500000	9.000000	
50%	2014.0	3.400000	6.000000	12.600000	8.800000	10.900000	
75%	2014.0	4.300000	8.000000	15.550000	10.850000	13.350000	
max	2014.0	8.400000	12.000000	30.200000	20.500000	25.800000	

In [9]:
```
#Exploing some of the features

cdf = df[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_COMB','CO2EMISSIONS']]
cdf.head(10)
```

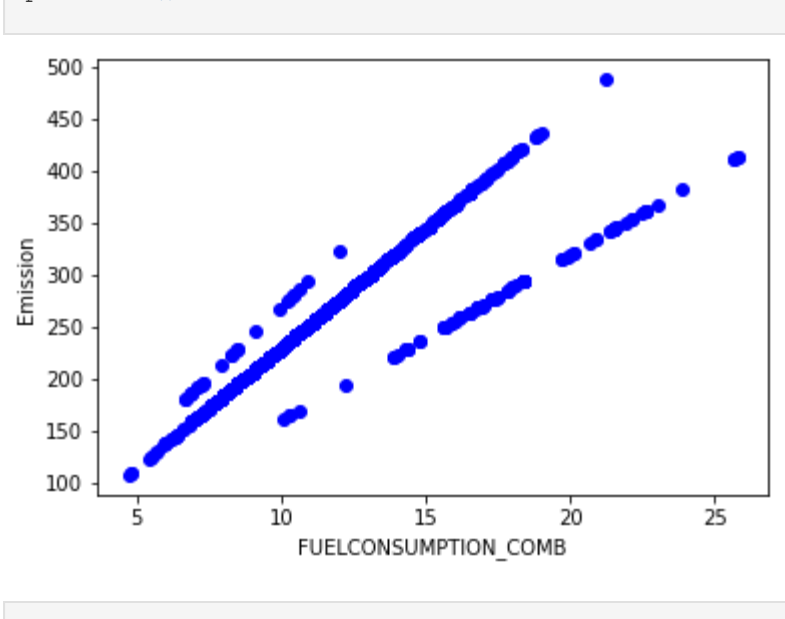
Out[9]:

	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
9	2.4	4	9.2	212

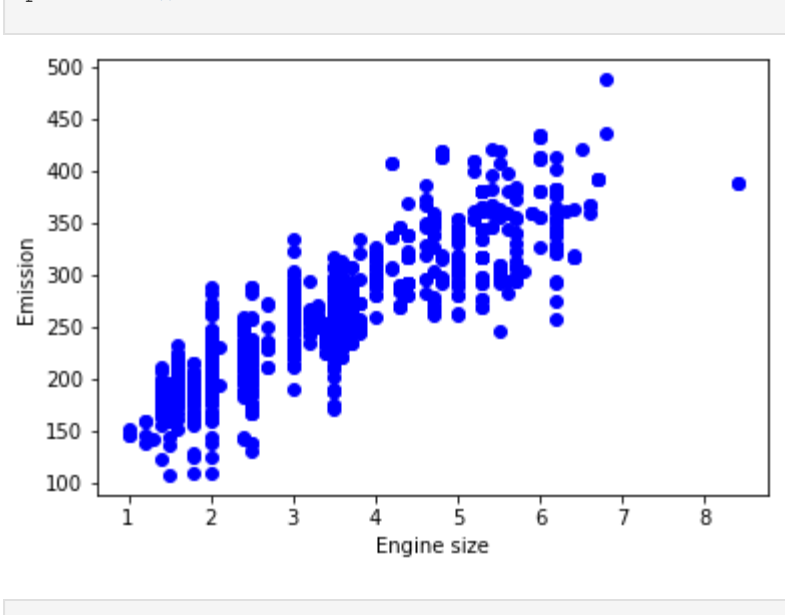
In [10]:
```
# visualizing the above features by plotting them

viz = cdf[['CYLINDERS','ENGINESIZE','CO2EMISSIONS','FUELCONSUMPTION_COMB']]
viz.hist()
plt.show()
```



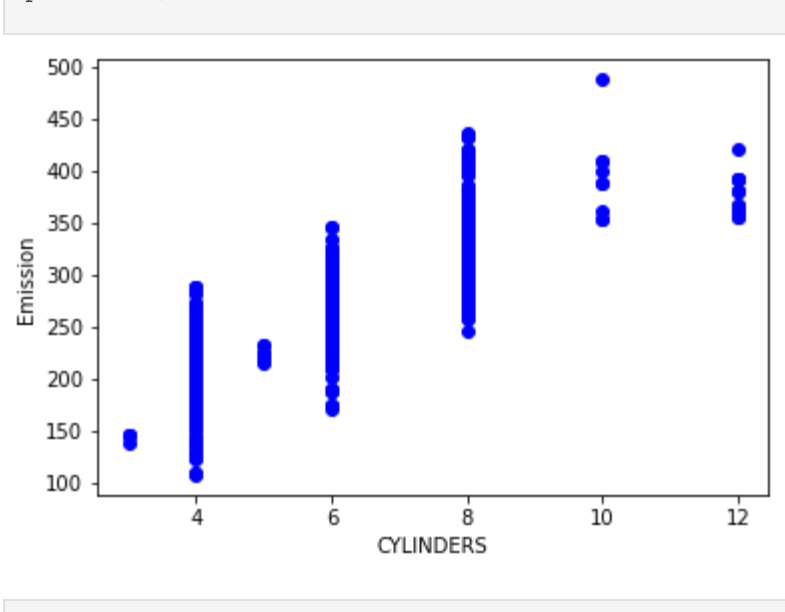
In [11]:
```
#ploting these features against the Emission, to see their linear relationship

plt.scatter(cdf.FUELCONSUMPTION_COMB, cdf.CO2EMISSIONS,  color='blue')
plt.xlabel("FUELCONSUMPTION_COMB")
plt.ylabel("Emission")
plt.show()
```



In [12]:
```
plt.scatter(cdf.ENGINESIZE, cdf.CO2EMISSIONS,  color='blue')
plt.xlabel("Engine size")
plt.ylabel("Emission")
plt.show()
```

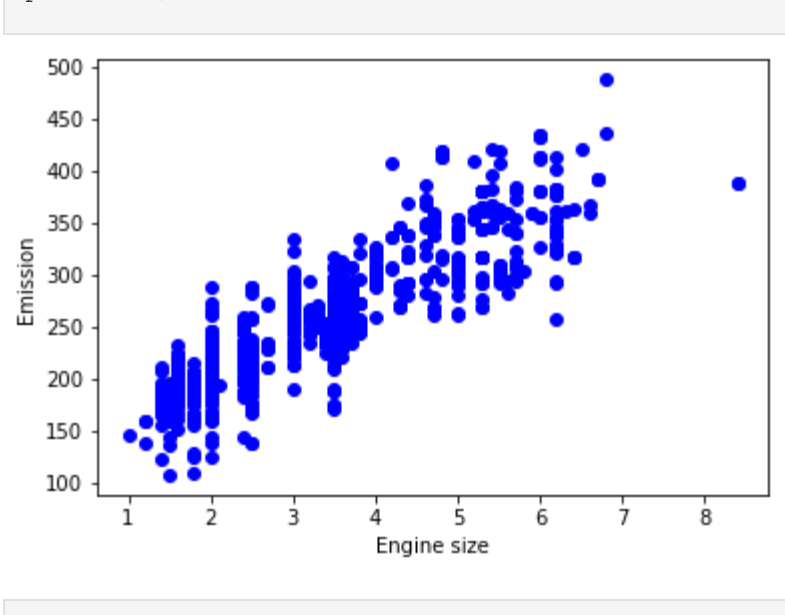


In [13]:
```
plt.scatter(cdf.CYLINDERS, cdf.CO2EMISSIONS,  color='blue')
plt.xlabel("CYLINDERS")
plt.ylabel("Emission")
plt.show()
```



In [14]:
```
#Training and testing the data seet

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

In [15]:
```
#For sinple linear regression model, Linear Regression fits a linear model with coefficients B = (B1, ..., Bn)
#Train data distribution

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

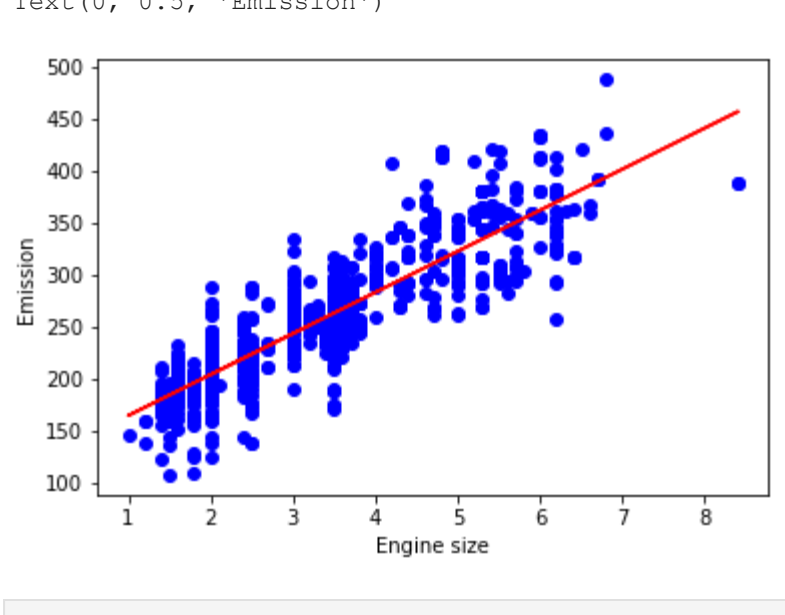


In [16]:
```
#Modeling the data using sklearn

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 ('Coefficients: ', regr.coef_)
print ('Intercept: ',regr.intercept_)
```

```
Coefficients:  [[39.48555416]]
Intercept:  [125.43722767]
```

In [17]:
```
#Plotting the fit line over the data

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



In [18]:
```
from sklearn.metrics import r2_score

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: 22.99
Residual sum of squares (MSE): 965.95
R2-score: 0.75
```

In [19]:
```
#To determine the evaluation metrics using the FUELCONSUMPTION_COMB feature
train_x = train[["FUELCONSUMPTION_COMB"]]

test_x = test[["FUELCONSUMPTION_COMB"]]
regr = linear_model.LinearRegression()
regr.fit(train_x, train_y)
predictions = regr.predict(test_x)
print("Mean Absolute Error: %.2f" % np.mean(np.absolute(predictions - 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: 21.12
Residual sum of squares (MSE): 965.95
R2-score: 0.75
```

Multiple Linear Regression

In [2]:
```
import matplotlib.pyplot as plt
import pandas as pd
import pylab as pl
import numpy as np
%matplotlib inline
```

In [4]:
```
df = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-
df.head(10)
```

Out[4]:

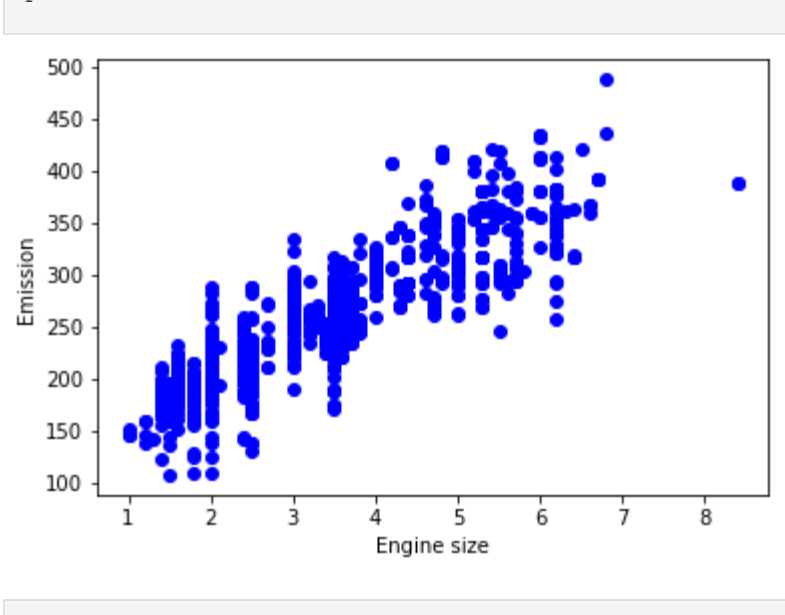
	MODELYEAR	MAKE	MODEL	VEHICLECLASS	ENGINESIZE	CYLINDERS	TRANSMISSION	FUELTYPE	FUELCONSUMPTION_CITY	FU
0	2014	ACURA	ILX	COMPACT	2.0	4	AS5	Z	9.9	
1	2014	ACURA	ILX	COMPACT	2.4	4	M6	Z	11.2	
2	2014	ACURA	ILX HYBRID	COMPACT	1.5	4	AV7	Z	6.0	
3	2014	ACURA	MDX 4WD	SUV - SMALL	3.5	6	AS6	Z	12.7	
4	2014	ACURA	RDX AWD	SUV - SMALL	3.5	6	AS6	Z	12.1	
5	2014	ACURA	RLX	MID-SIZE	3.5	6	AS6	Z	11.9	
6	2014	ACURA	TL	MID-SIZE	3.5	6	AS6	Z	11.8	
7	2014	ACURA	TL AWD	MID-SIZE	3.7	6	AS6	Z	12.8	
8	2014	ACURA	TL AWD	MID-SIZE	3.7	6	M6	Z	13.4	
9	2014	ACURA	TSX	COMPACT	2.4	4	AS5	Z	10.6	

In [5]:
```
cdf = df[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_CITY','FUELCONSUMPTION_HWY','FUELCONSUMPTION_COMB','CO2EMIS
cdf.head(10)
```

Out[5]:

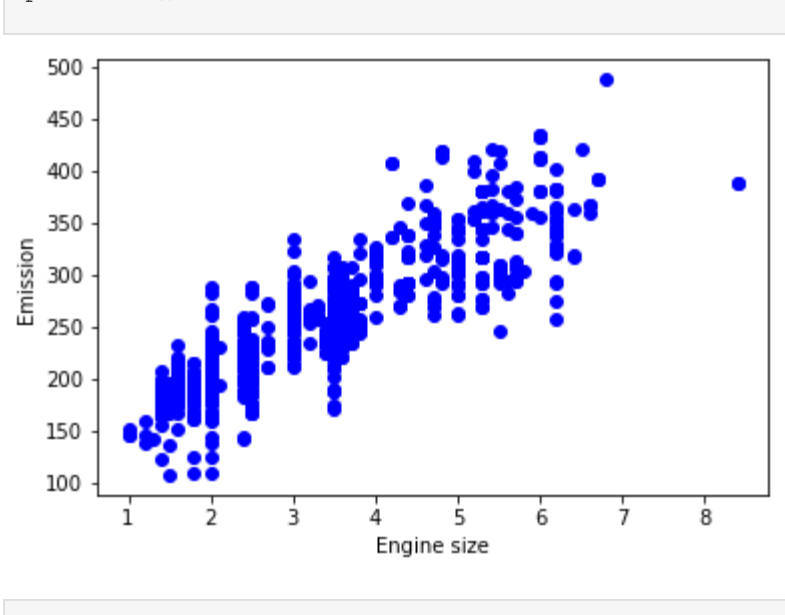
	ENGINESIZE	CYLINDERS	FUELCONSUMPTION_CITY	FUELCONSUMPTION_HWY	FUELCONSUMPTION_COMB	CO2EMISSIONS
0	2.0	4	9.9	6.7	8.5	196
1	2.4	4	11.2	7.7	9.6	221
2	1.5	4	6.0	5.8	5.9	136
3	3.5	6	12.7	9.1	11.1	255
4	3.5	6	12.1	8.7	10.6	244
5	3.5	6	11.9	7.7	10.0	230
6	3.5	6	11.8	8.1	10.1	232
7	3.7	6	12.8	9.0	11.1	255
8	3.7	6	13.4	9.5	11.6	267
9	2.4	4	10.6	7.5	9.2	212

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



In [7]:
```
#Test and training data
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()
```



In [9]:
```
# Using multiple independent variables such as FUELCONSUMPTION_COMB, EngineSize and Cylinders of cars to predic
#Multiple linear regression model is the extension of the simple linear regression model.

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)

print ('Coefficients: ', regr.coef_)
```

```
Coefficients:  [[10.30379996  7.88799292  9.32356035]]
```

In [10]:
```
#Estimating the unknown parameters in a linear regression model using Ordinary Least Squares (OLS) method.

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: 527.94
Variance score: 0.88
```

In [11]:
```
# Multiple linear regression using FLUELCONSUMPTION_CITY and FUELCONSUMPTION_HWY instead of FUELCONSUMPTION_CO
regr = linear_model.LinearRegression()
x = np.asanyarray(train[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_CITY','FUELCONSUMPTION_HWY']])
y = np.asanyarray(train[['CO2EMISSIONS']])
regr.fit (x, y)
print ('Coefficients: ', regr.coef_)
y_= regr.predict(test[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_CITY','FUELCONSUMPTION_HWY']])
x = np.asanyarray(test[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_CITY','FUELCONSUMPTION_HWY']])
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:  [[10.38108217  7.56387427  5.87764536  3.20849751]]
Residual sum of squares: 527.37
Variance score: 0.88
```

Polynomial Regression

In [12]:
```
df.head(10)
```

Out[12]:

	MODELYEAR	MAKE	MODEL	VEHICLECLASS	ENGINESIZE	CYLINDERS	TRANSMISSION	FUELTYPE	FUELCONSUMPTION_CITY	FU
0	2014	ACURA	ILX	COMPACT	2.0	4	AS5	Z	9.9	
1	2014	ACURA	ILX	COMPACT	2.4	4	M6	Z	11.2	
2	2014	ACURA	ILX HYBRID	COMPACT	1.5	4	AV7	Z	6.0	
3	2014	ACURA	MDX 4WD	SUV - SMALL	3.5	6	AS6	Z	12.7	
4	2014	ACURA	RDX AWD	SUV - SMALL	3.5	6	AS6	Z	12.1	
5	2014	ACURA	RLX	MID-SIZE	3.5	6	AS6	Z	11.9	
6	2014	ACURA	TL	MID-SIZE	3.5	6	AS6	Z	11.8	
7	2014	ACURA	TL AWD	MID-SIZE	3.7	6	AS6	Z	12.8	
8	2014	ACURA	TL AWD	MID-SIZE	3.7	6	M6	Z	13.4	
9	2014	ACURA	TSX	COMPACT	2.4	4	AS5	Z	10.6	

In [13]:
```
#Using ENGINESIZE and second degree plynomial for our model

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
```

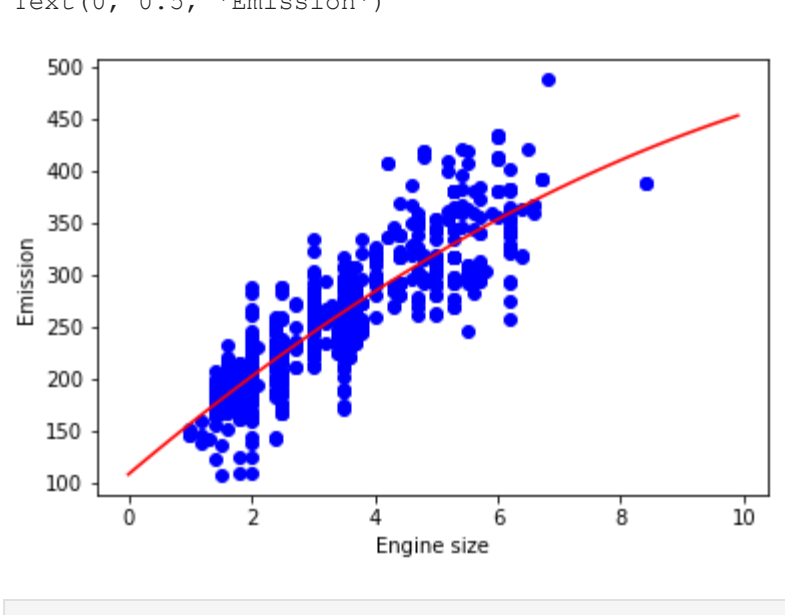
Out[13]:
```
array([[ 1.  ,  2.  ,  4.  ],
       [ 1.  ,  2.4 ,  5.76],
       [ 1.  ,  1.5 ,  2.25],
       ...,
       [ 1.  ,  3.  ,  9.  ],
       [ 1.  ,  3.2 , 10.24],
       [ 1.  ,  3.2 , 10.24]])
```

In [14]:
```
clf = linear_model.LinearRegression()
train_y_ = clf.fit(train_x_poly, train_y)
print ('Coefficients: ', clf.coef_)
print ('Intercept: ',clf.intercept_)
```

```
Coefficients:  [[ 0.         50.08730154 -1.54004369]]
Intercept:  [108.38613302]
```

In [15]:
```
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[15]: `Text(0, 0.5, 'Emission')`

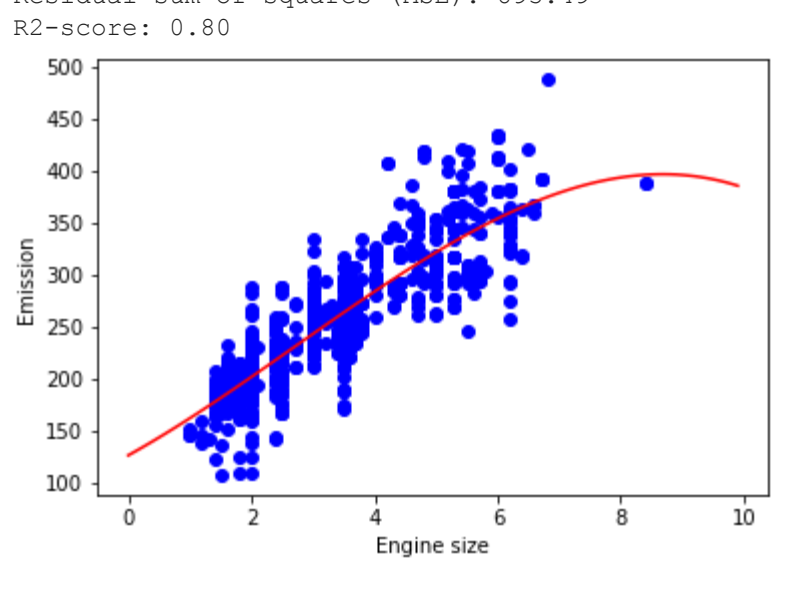


In [16]:
```
from sklearn.metrics import r2_score

test_x_poly = poly.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: 22.76
Residual sum of squares (MSE): 901.24
R2-score: 0.80
```

In [17]:
```
# Polynomial regression using above data with degree of three (cubic)

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)


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) + clf3.coef_[0][3]*np.power(XX,
plt.plot(XX, yy, '-r' )
plt.xlabel("Engine size")
plt.ylabel("Emission")
test_x_poly3 = poly3.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_y,test_y3_ ) )
```

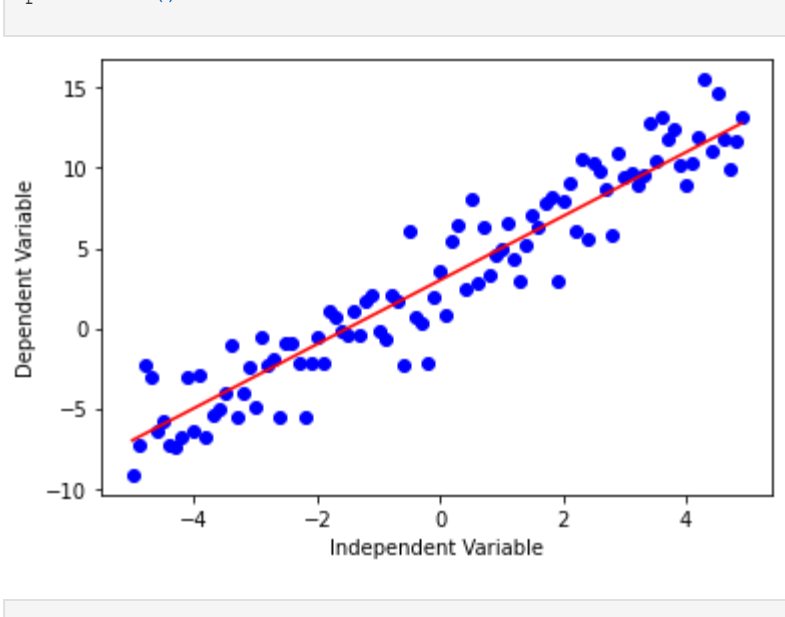
```
Coefficients:  [[ 0.         32.90349866  3.17660419 -0.38948774]]
Intercept:  [126.70421324]
Mean absolute error: 22.67
Residual sum of squares (MSE): 893.49
R2-score: 0.80
```



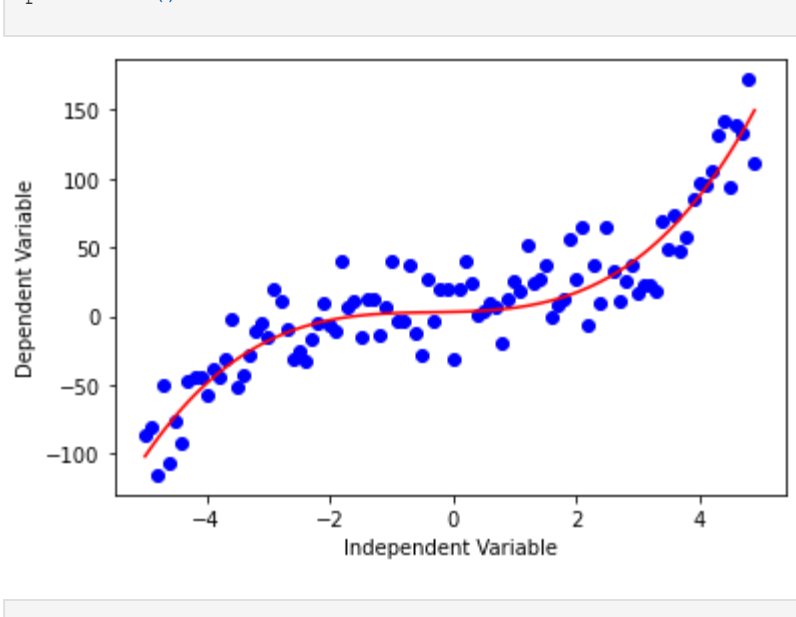
Non-Linear Regression Analysis

In [18]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

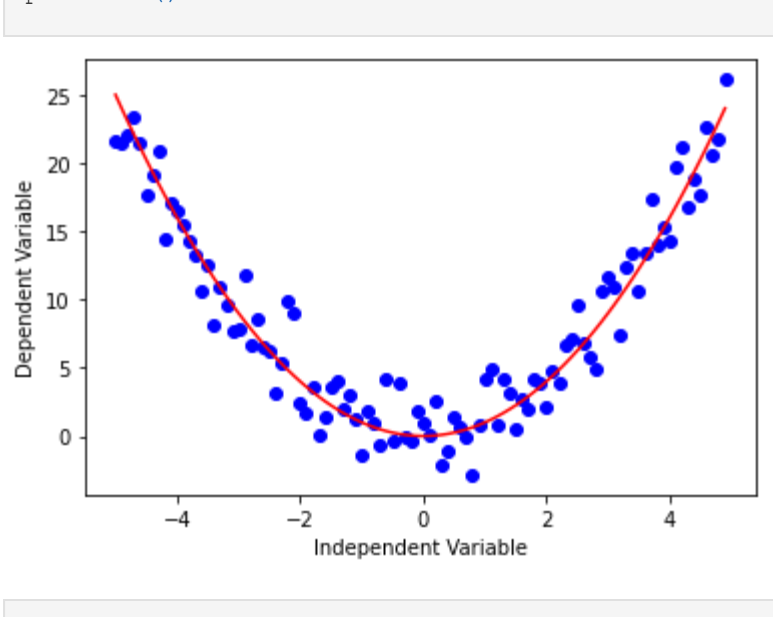
In [19]:
```
#Differentiating between linear and non-linear regression using China's GDP data from 1960 to 2014

#Linear regression models a linear relationship between the dependent and the independent variable with a simp

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('Independent Variable')
plt.show()
```



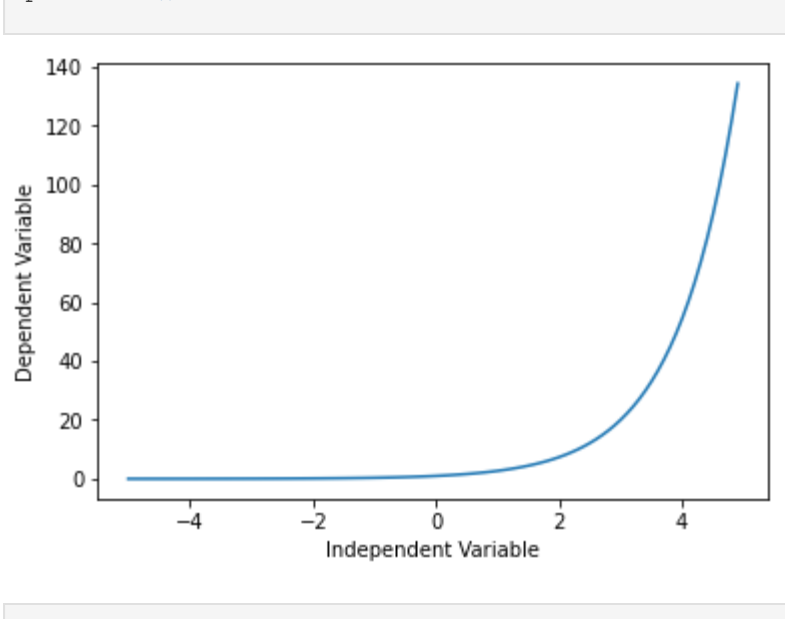
In [20]:
```
""""Non-linear regression models the non-linear relationship between the independent variables x and the depend
degrees (maximum power of  x). For example: y=ax3+bx2+cx+d

Non-linear functions can have elements like exponentials, logarithms, fractions, and so on.
For example:
y=log(x)
y=log(ax3+bx2+cx+d)
For cubic function's graph."""

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('Independent Variable')
plt.show()
```



In [21]:
```
#For quadratic function's graph (Y=X2(squared))

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()
```

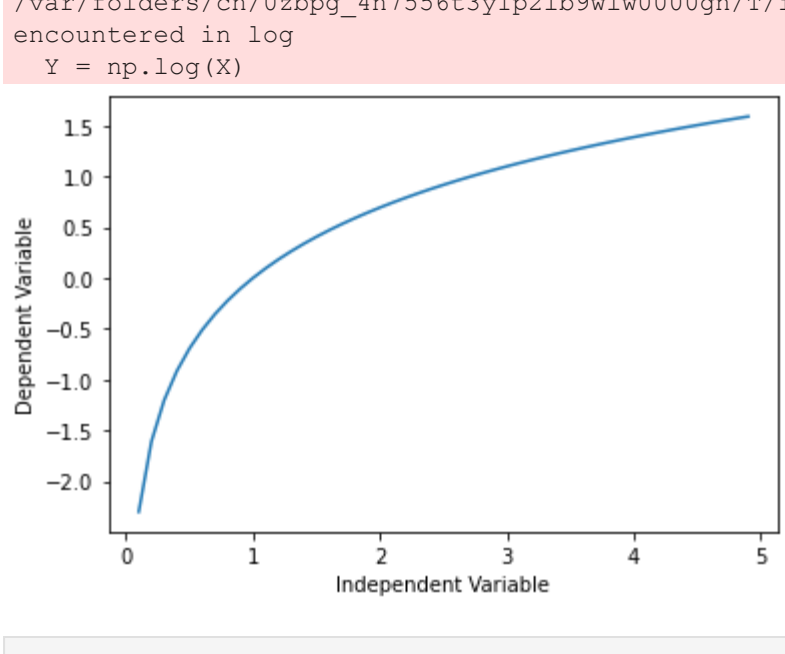


In [22]:
```
#Exponential: An exponential function with base c is defined by Y=a+bcX
#where b ≠0, c > 0 , c ≠1, and x is any real number. The base, c, is constant and the exponent, x, is a variab

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()
```

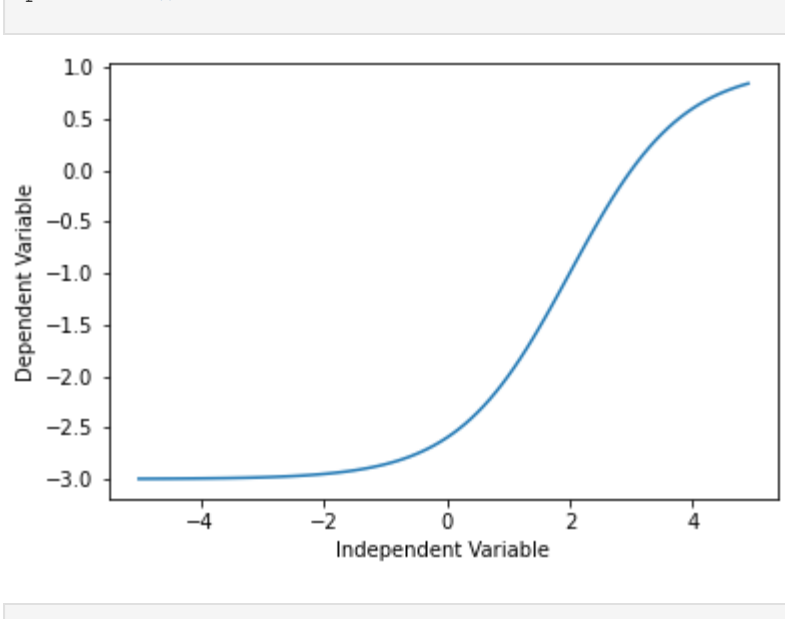


In [23]:
```
#Logarithmic: The response y is a results of applying the logarithmic map from the input x to the output y.
#y=log(x). Instead of x, we can use X, which can be a polynomial representation of the x values.
#In general form it would be written as y=log(X)

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()
```

```
/var/folders/cn/0zbpg_4n7556t3y1p21b9w1w0000gn/T/ipykernel_5094/1521300694.py:7: RuntimeWarning: invalid value
encountered in log
  Y = np.log(X)
```



In [24]:
```
#Sigmoidal/Logistic: Y= a + b/1+c(X-d) - (note: X-d is the power of c)

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()
```

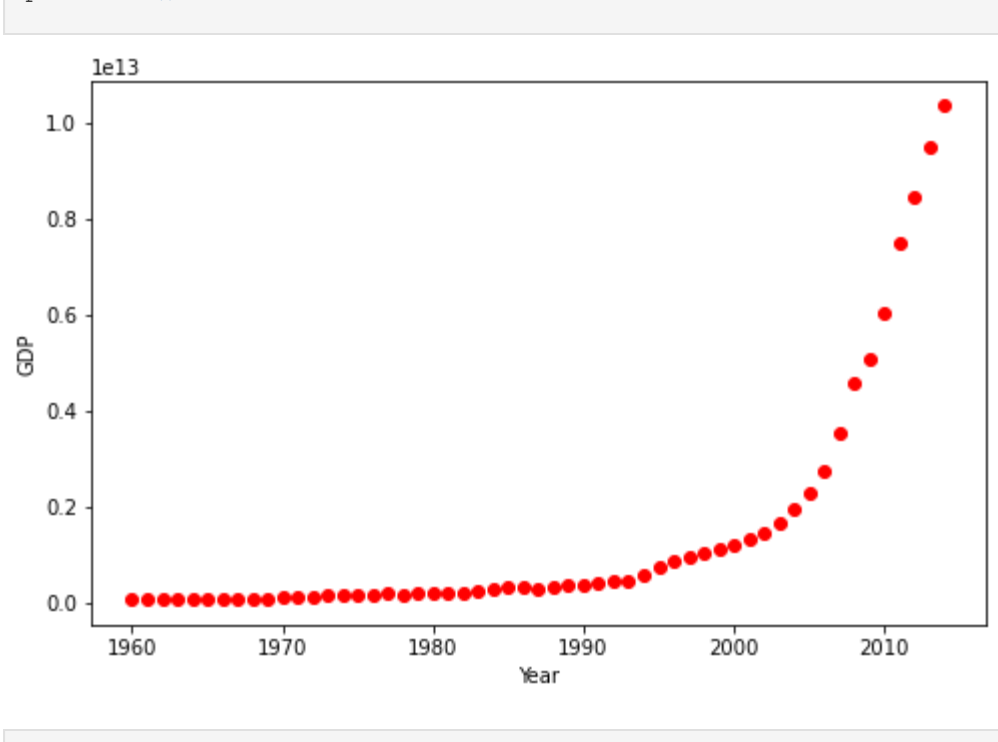


In [25]:
```
#Non-Linear Regression Example

import numpy as np
import pandas as pd

#downloading dataset

df = pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-
df.head(10)
```

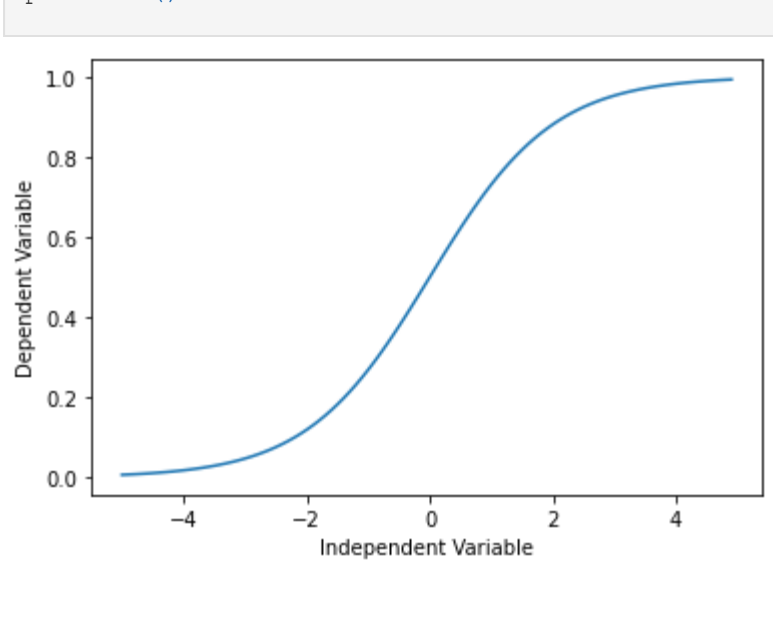
Out[25]:

	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
5	1965	6.970915e+10
6	1966	7.587943e+10
7	1967	7.205703e+10
8	1968	6.999350e+10
9	1969	7.871882e+10

In [26]:
```
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()
```



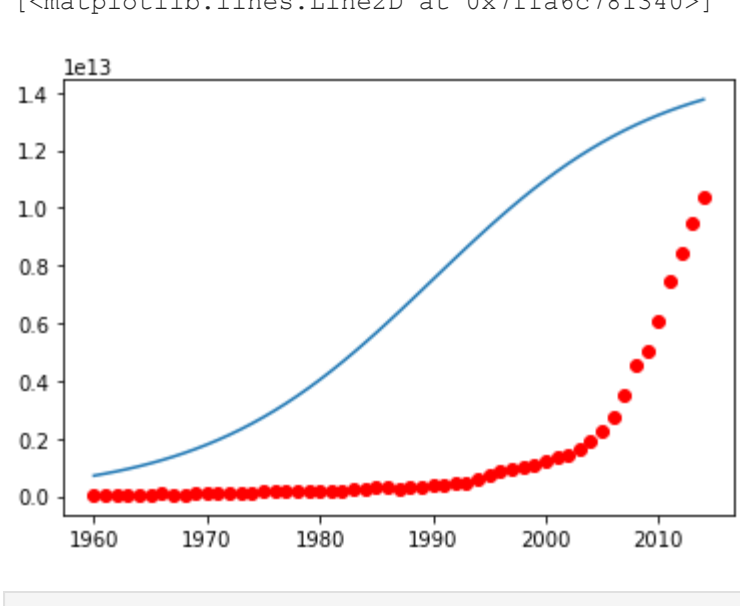
In [27]:
```
#The graph shows that a logistic function could be a better approximation. Compare the graph below and the data
# yhat=1/1+e-β_1(X-β_2)

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()
```



In [28]:
```python
#Building the model

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

In [29]:
```python
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[29]:
```
[<matplotlib.lines.Line2D at 0x7ffa6c781340>]
```

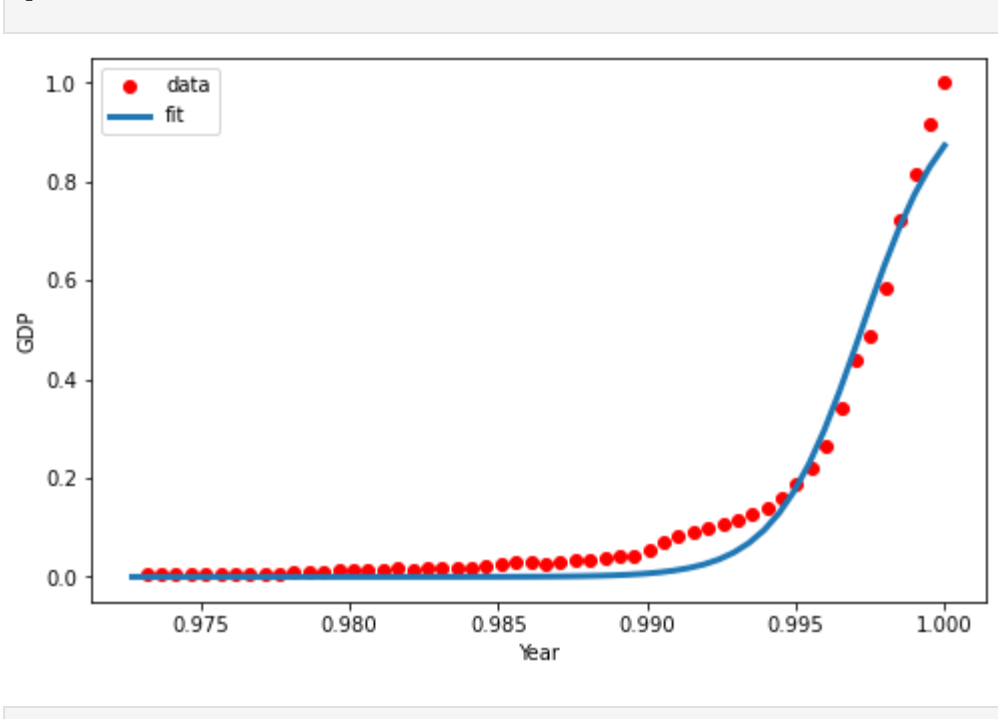


In [30]:
```python
# normalizing the data
xdata =x_data/max(x_data)
ydata =y_data/max(y_data)
```

In [31]:
```python
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 = 690.453017, beta_2 = 0.997207
```

In [32]:
```python
#Resulting regression

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 [33]:
```python
#Calculating the accuracy of the model

# split data into train/test
msk = np.random.rand(len(df)) < 0.8
train_x = xdata[msk]
test_x = xdata[~msk]
train_y = ydata[msk]
test_y = ydata[~msk]

# build the model using train set
popt, pcov = curve_fit(sigmoid, train_x, train_y)

# predict using test set
y_hat = sigmoid(test_x, *popt)

# evaluation
print("Mean absolute error: %.2f" % np.mean(np.absolute(y_hat - test_y)))
print("Residual sum of squares (MSE): %.2f" % np.mean((y_hat - test_y) ** 2))
from sklearn.metrics import r2_score
print("R2-score: %.2f" % r2_score(test_y,y_hat) )
```

```
Mean absolute error: 0.04
Residual sum of squares (MSE): 0.00
R2-score: 0.96
```

In [ ]: