Univariate Linear Regression by Gradient Descent

Importing the libraries

In [1]:
```
#libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import math
import seaborn as sns
```

Importing the Dataset

In [2]:
```
dataset=pd.read_csv("Life Expectancy Data.csv")
```

In [3]:
```
dataset.head()
```

Out[3]:

	Country	Year	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatit
0	Afghanistan	2015	Developing	65.0	263.0	62	0.01	71.279624	65
1	Afghanistan	2014	Developing	59.9	271.0	64	0.01	73.523582	62
2	Afghanistan	2013	Developing	59.9	268.0	66	0.01	73.219243	64
3	Afghanistan	2012	Developing	59.5	272.0	69	0.01	78.184215	67
4	Afghanistan	2011	Developing	59.2	275.0	71	0.01	7.097109	68

5 rows × 22 columns

In [4]:
```
#displaying the dataset
dataset
```

Out[4]:

	Country	Year	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hep
0	Afghanistan	2015	Developing	65.0	263.0	62	0.01	71.279624	
1	Afghanistan	2014	Developing	59.9	271.0	64	0.01	73.523582	
2	Afghanistan	2013	Developing	59.9	268.0	66	0.01	73.219243	
3	Afghanistan	2012	Developing	59.5	272.0	69	0.01	78.184215	
4	Afghanistan	2011	Developing	59.2	275.0	71	0.01	7.097109	
...	...	...	...	...	...	...	...	...	
2933	Zimbabwe	2004	Developing	44.3	723.0	27	4.36	0.000000	
2934	Zimbabwe	2003	Developing	44.5	715.0	26	4.06	0.000000	
2935	Zimbabwe	2002	Developing	44.8	73.0	25	4.43	0.000000	
2936	Zimbabwe	2001	Developing	45.3	686.0	25	1.72	0.000000	
2937	Zimbabwe	2000	Developing	46.0	665.0	24	1.68	0.000000	

2938 rows × 22 columns

Checking for the Null Values

In [5]:
```
pd.isna(dataset).any()
```

Out[5]:
```
Country                             False
Year                                False
Status                              False
Life expectancy                      True
Adult Mortality                      True
infant deaths                       False
Alcohol                              True
percentage expenditure              False
Hepatitis B                          True
Measles                             False
 BMI                                 True
under-five deaths                   False
Polio                                True
Total expenditure                    True
Diphtheria                           True
 HIV/AIDS                           False
GDP                                  True
Population                           True
 thinness  1-19 years                True
 thinness 5-9 years                  True
Income composition of resources      True
Schooling                            True
dtype: bool
```

In [6]:
```
# checking details of the dataset
dataset.describe()
```

Out[6]:

	Year	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hep
count	2938.000000	2928.000000	2928.000000	2938.000000	2744.000000	2938.000000	2385
mean	2007.518720	69.224932	164.796448	30.303948	4.602861	738.251295	80
std	4.613841	9.523867	124.292079	117.926501	4.052413	1987.914858	25
min	2000.000000	36.300000	1.000000	0.000000	0.010000	0.000000	1
25%	2004.000000	63.100000	74.000000	0.000000	0.877500	4.685343	77
50%	2008.000000	72.100000	144.000000	3.000000	3.755000	64.912906	92
75%	2012.000000	75.700000	228.000000	22.000000	7.702500	441.534144	97
max	2015.000000	89.000000	723.000000	1800.000000	17.870000	19479.911610	99

Seeing the correlation matrix to select the feature

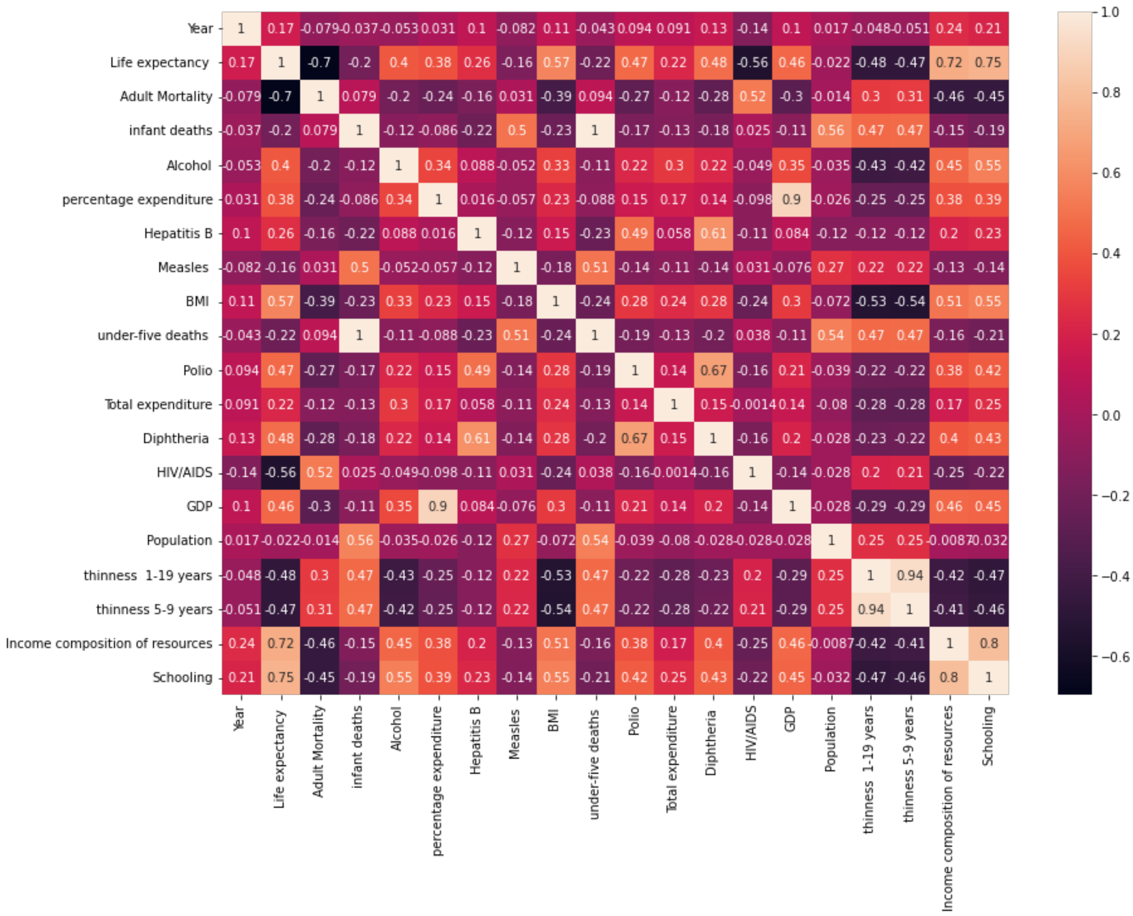
In [7]:
```
dataset.corr()
```

Out[7]:

	Year	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B
Year	1.000000	0.170033	-0.079052	-0.037415	-0.052990	0.031400	0.104333
Life expectancy	0.170033	1.000000	-0.696359	-0.196557	0.404877	0.381864	0.256762
Adult Mortality	-0.079052	-0.696359	1.000000	0.078756	-0.195848	-0.242860	-0.162476
infant deaths	-0.037415	-0.196557	0.078756	1.000000	-0.115638	-0.085612	-0.223566
Alcohol	-0.052990	0.404877	-0.195848	-0.115638	1.000000	0.341285	0.087549
percentage expenditure	0.031400	0.381864	-0.242860	-0.085612	0.341285	1.000000	0.016274
Hepatitis B	0.104333	0.256762	-0.162476	-0.223566	0.087549	0.016274	1.000000
Measles	-0.082493	-0.157586	0.031176	0.501128	-0.051827	-0.056596	-0.120529
BMI	0.108974	0.567694	-0.387017	-0.227279	0.330408	0.228700	0.150380
under-five deaths	-0.042937	-0.222529	0.094146	0.996629	-0.112370	-0.087852	-0.233126
Polio	0.094158	0.465556	-0.274823	-0.170689	0.221734	0.147259	0.486171
Total expenditure	0.090740	0.218086	-0.115281	-0.128616	0.296942	0.174420	0.058280
Diphtheria	0.134337	0.479495	-0.275131	-0.175171	0.222020	0.143624	0.611495
HIV/AIDS	-0.139741	-0.556556	0.523821	0.025231	-0.048845	-0.097857	-0.112675
GDP	0.101620	0.461455	-0.296049	-0.108427	0.354712	0.899373	0.083903
Population	0.016969	-0.021538	-0.013647	0.556801	-0.035252	-0.025662	-0.123321
thinness 1-19 years	-0.047876	-0.477183	0.302904	0.465711	-0.428795	-0.251369	-0.120429
thinness 5-9 years	-0.050929	-0.471584	0.308457	0.471350	-0.417414	-0.252905	-0.124960
Income composition of resources	0.243468	0.724776	-0.457626	-0.145139	0.450040	0.381952	0.199549
Schooling	0.209400	0.751975	-0.454612	-0.193720	0.547378	0.389687	0.231117

In [8]:
```
#Visuilizing the correlation matrix
plt.figure(figsize = (14, 10))
correlation = dataset.corr()
sns.heatmap(correlation,
        xticklabels=correlation.columns,
        yticklabels=correlation.columns,
           annot=True)
```

Out[8]: <AxesSubplot:>



Note: After analyzing the coorelation matrix , I am considering "Income composition of resources" as my feature

Defining feature and target variable

In [9]:
```
x=dataset.iloc[:,20:21].values
y=dataset.iloc[:,3:4].values
```

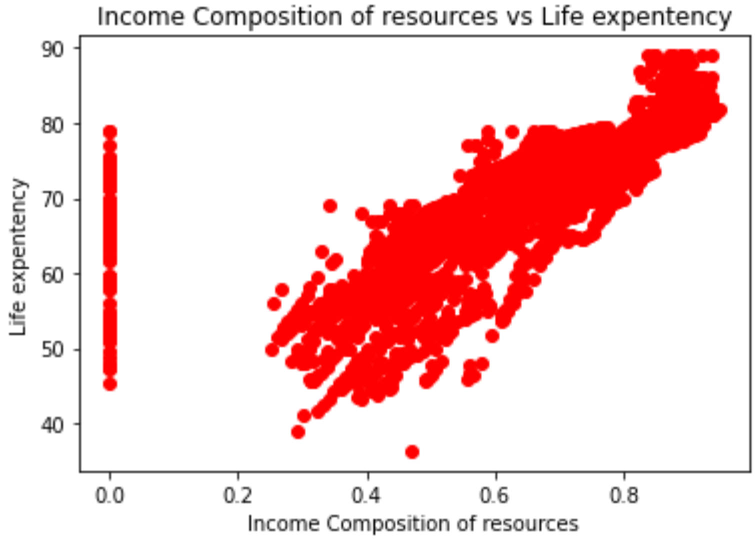
In [10]:
```
x
```

Out[10]:
```
array([[0.479],
       [0.476],
       [0.47 ],
       ...,
       [0.427],
       [0.427],
       [0.434]])
```

In [11]:
```
y
```

Out[11]:
```
array([[65. ],
       [59.9],
       [59.9],
       ...,
       [44.8],
       [45.3],
       [46. ]])
```

In [12]:
```
#plotting the actual data
plt.scatter(x,y,color='red')
plt.title('Income Composition of resources vs Life expentency')
plt.xlabel('Income Composition of resources')
plt.ylabel("Life expentency")
plt.show()
```



Taking care of missing values

Using SimpleImputer Class of sklearn.impute library to fill the missing values

In [13]:
```
# Using mean strategy to impute missing values
from sklearn.impute import SimpleImputer
imputer=SimpleImputer(missing_values=np.nan,strategy='mean')
imputer.fit(y[:,:])
y[:,:]=imputer.transform(y[:,:])
```

In [14]:
```
# reshaping the feature into 2-D so that it can be passed in the fit
x=np.reshape(x,(len(x),1))
```

In [15]:
```
from sklearn.impute import SimpleImputer
imputer=SimpleImputer(missing_values=np.nan,strategy='mean')
imputer.fit(x[:,:])
x[:,:]=imputer.transform(x[:,:])
```

In [16]:
```
x
```

Out[16]:
```
array([[0.479],
       [0.476],
       [0.47 ],
       ...,
       [0.427],
       [0.427],
       [0.434]])
```

In [17]:
```
y
```

Out[17]:
```
array([[65. ],
       [59.9],
       [59.9],
       ...,
       [44.8],
       [45.3],
       [46. ]])
```

In [18]:
```
len(x)
```

Out[18]:
```
2938
```

In [19]:
```
len(y)
```

Out[19]:
```
2938
```

In [20]:
```
#adding extra column(of value 1) in the feature for the symmetry. Do
#the matrix multiplication between our transpose of parameter vector
x = x[:,0:len(x)]
ones = np.ones([x.shape[0],1])
x = np.concatenate((ones,x),axis=1)
theta = np.zeros([1,2])
```

In [21]:
```
x.shape,theta.shape,y.shape
```

Out[21]:
```
((2938, 2), (1, 2), (2938, 1))
```

Splitting Dataset into Training Set and Test Set

In [22]:
```
#giving 20% to the test set
x_test=x[0:587,:]
```

In [23]:
```
x_train=x[587:,:]
```

In [24]:
```
y_test=y[0:587,:]
y_train=y[587:,:]
```

In [25]:
```
len(x_train)+len(x_test)
```

Out[25]:
```
2938
```

In [26]:
```
len(y_train)+len(y_test)
```

Out[26]:
```
2938
```

In [27]:
```
print(x_train)
```

```
[[1.    0.658]
 [1.    0.659]
 [1.    0.656]
 ...
 [1.    0.427]
 [1.    0.427]
 [1.    0.434]]
```

In [28]:
```
x_train
```

Out[28]:
```
array([[1.   , 0.658],
       [1.   , 0.659],
       [1.   , 0.656],
       ...,
       [1.   , 0.427],
       [1.   , 0.427],
       [1.   , 0.434]])
```

In [29]:
```
print(x_test)
```

```
[[1.    0.479]
 [1.    0.476]
 [1.    0.47 ]
 ...
 [1.    0.675]
 [1.    0.669]
 [1.    0.658]]
```

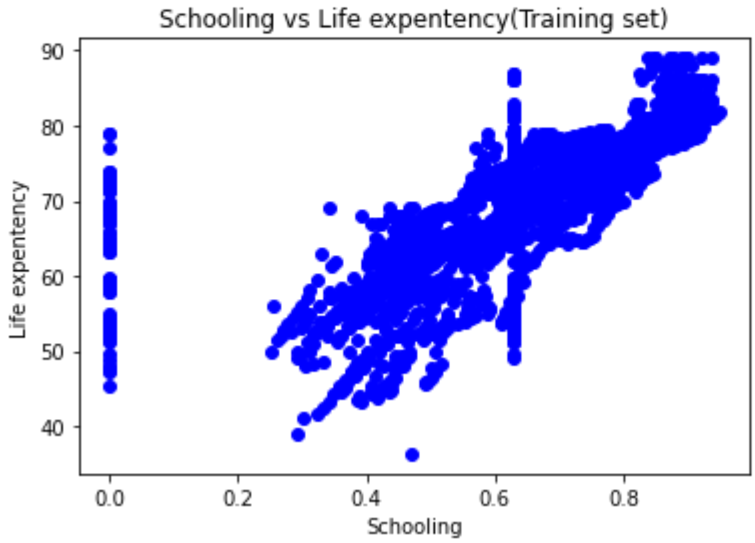
In [30]:
```
print(y_train)
```

```
[[72.8]
 [72.4]
 [71.8]
 ...
 [44.8]
 [45.3]
 [46. ]]
```

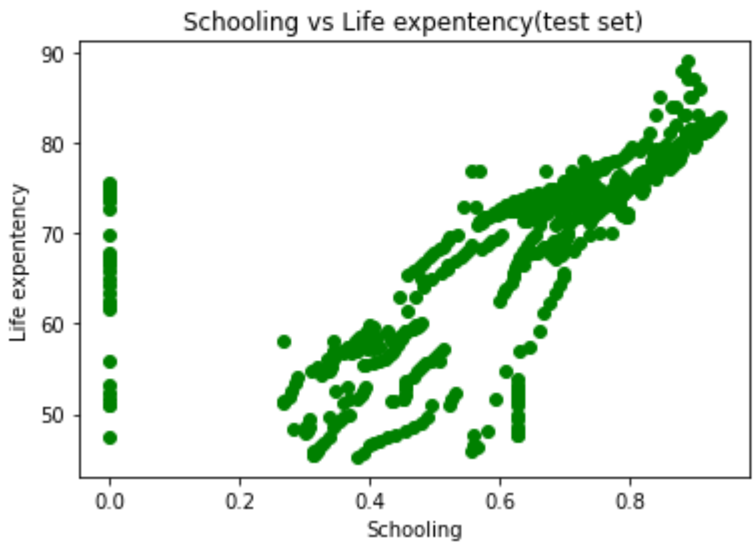
In [31]:
```
print(y_test)
```

```
[[65. ]
 [59.9]
 [59.9]
 [59.5]
 [59.2]
 [58.8]
 [58.6]
 [58.1]
 [57.5]
 [57.3]
 [57.3]
 [57. ]
 [56.7]
 [56.2]
 [55.3]
 [54.8]
 [77.8]
 [77.5]
 [77.2]
```

In [32]:
```
#Plotting the scatter plot to see the trainig set
plt.scatter(x_train[:,1:],y_train,color='blue') #plotting the actual
plt.title('Schooling vs Life expentency(Training set)')
plt.xlabel('Schooling')
plt.ylabel("Life expentency")
plt.show()
```



In [33]:
```
#plotting the scatter plot to see the test set
plt.scatter(x_test[:,1:],y_test,color='green') #plotting the actual
plt.title('Schooling vs Life expentency(test set)')
plt.xlabel('Schooling')
plt.ylabel("Life expentency")
plt.show()
```



Training our model on the training set

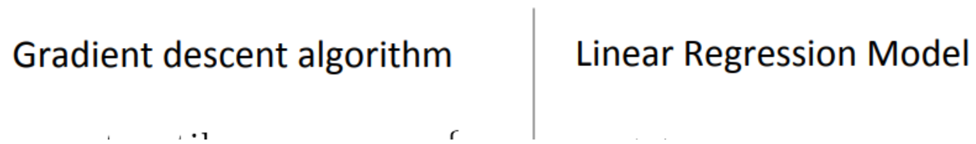
In [34]:
```
#defining the cost function
def Cost(x,y,theta):
    summed = np.power(((x @ theta.T)-y),2)
    return np.nansum(summed)/(2 * len(x))

#defining the gradient descent
def gradientDescent(x,y,theta,iters,alpha):
    cost = np.zeros(iters)
    for i in range(iters):
        theta = theta - (alpha/len(x)) * np.nansum(x * (x @ theta.T -
        #print(theta)
        cost[i] = Cost(x, y, theta)

    return theta,cost
```

Cost Function Formula

$$\frac{1}{2m}\sum_{1}^{m}(h(x^{(i)})-y^{(i)})^2$$



$$
\begin{aligned}
&\text{repeat until convergence } \{ \\
&\quad \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \\
&\qquad (\text{for } j = 1 \text{ and } j = 0) \\
&\}
\end{aligned}
$$

$$h_\theta(x) = \theta_0 + \theta_1 x$$

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

In [35]:
```
#setting values of gradient descent parameters
alpha = 0.01
iters = 1500
```

In [36]:
```
m,cost = gradientDescent(x_train,y_train,theta,iters,alpha)
print(m)

finalCost = Cost(x_train,y_train,m)
print(finalCost)
```

```
[[49.21601509 32.10382897]]
22.03915281306857
```

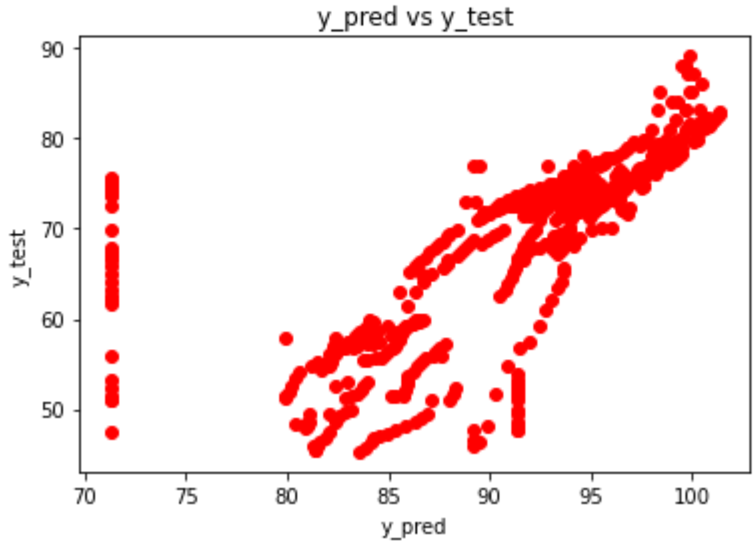
Predicting on the test set

In [37]:
```
#predicting on the test set
def predict(m, w, b):
    return m.dot(w.T) + b
y_pred = predict(x_test, m, finalCost)
```

In [38]:
```
y_pred
```

Out[38]:
```
array([[ 86.63290198],
       [ 86.53659049],
       [ 86.34396752],
       [ 86.11924072],
       [ 85.83030626],
       [ 85.63768328],
       [ 85.18822968],
       [ 85.15612585],
       [ 84.57825693],
       [ 84.25721864],
       [ 83.96828418],
       [ 83.48672674],
       [ 83.22989611],
       [ 82.20257358],
       [ 82.17046975],
       [ 82.10626209],
       [ 95.71828558],
       [ 95.68618175],
       [ 95.62197409],
```

In [39]:
```
#Plotting the scatter plot to analyze test target variable and predi
plt.scatter(y_pred,y_test,color='red')
plt.title('y_pred vs y_test')
plt.xlabel('y_pred')
plt.ylabel("y_test")
plt.show()
```



In [40]:
```
print(np.concatenate((y_pred.reshape(1,len(y_pred)),y_test.reshape(1,
```

```
[[86.63290198 86.53659049 86.34396752 ... 92.92525246 92.73262948
  92.37948737]
 [65.         59.9        59.9        ... 73.5        73.1
  73.1       ]]
```

Defining the RMSE

In [41]:
```
def rmse(predictions, targets):
    return np.sqrt(((predictions - targets) ** 2).mean())
rmse(y_pred, y_test)
```

Out[41]: 23.961915896753368

RMSE

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

Defining the MAE

In [42]:
```python
def MAE(y_pred,y):
    res=np.sum(abs(y_pred-y))/len(y)
    return res
MAE(y_pred, y_test)
```

Out[42]:
```
22.849594889689822
```

MAE

$$\mathrm{MAE} = \underbrace{\frac{1}{n}\sum_{i=1}^{n}}_{\text{test set}} |\underbrace{y_i}_{\text{predicted vaue}} - \underbrace{\hat{y}_i}_{\text{actual value}}|$$

Univariate Linear Regression by Closed Form

Defining feature and target variable

In [43]:
```python
x1=dataset.iloc[:,-1].values
y1=dataset.iloc[:,3:4].values
print(x)
print(y)
```

```
[[1.    0.479]
 [1.    0.476]
 [1.    0.47 ]
 ...
 [1.    0.427]
 [1.    0.427]
 [1.    0.434]]
[[65. ]
 [59.9]
 [59.9]
 ...
 [44.8]
 [45.3]
 [46. ]]
```

Taking care of missing values

Using SimpleImputer Class of sklearn.impute library to fill the missing values

In [44]:
```python
# Using mean strategy to impute missing values
from sklearn.impute import SimpleImputer
imputer=SimpleImputer(missing_values=np.nan,strategy='mean')
imputer.fit(y1[:,:])
y1[:,:]=imputer.transform(y1[:,:])
```

In [45]:
```
x1=np.reshape(x1,(len(x1),1))
```

In [46]:
```
from sklearn.impute import SimpleImputer
imputer=SimpleImputer(missing_values=np.nan,strategy='mean')
imputer.fit(x1[:,:])
x1[:,:]=imputer.transform(x1[:,:])
```

In [47]:
```
x1.shape,theta.shape,y1.shape
```

Out[47]:
```
((2938, 1), (1, 2), (2938, 1))
```

Splitting dataset into training set and test set

In [48]:
```
x1_test=x[0:587,:]
x1_train=x[587:,:]
y1_test=y[0:587,:]
y1_train=y[587:,:]
```

Training my model on training set

In [49]:
```
def find_Theta(x, y):

    m = x.shape[0] # Number of training examples.
    # Appending a cloumn of ones in X to add the bias term.
    x = np.append(x, np.ones((m,1)), axis=1)
    # reshaping y to (m,1)
    y = y.reshape(m,1)

    # The Normal Equation
    theta = np.dot(np.linalg.inv(np.dot(x.T, x)), np.dot(x.T, y))

    return theta
```

$$y = \sum_{j=0}^{m} W^T X$$

$$W = \left(X^T X\right)^{-1} X^T y$$

In [50]:
```
def predict(x,theta):

    # Appending a cloumn of ones in X to add the bias term.
    x = np.append(x, np.ones((x.shape[0],1)), axis=1)

    # preds is y_hat which is the dot product of X and theta.
    preds = np.dot(x, theta)

    return preds
```

Predicting value on test set

In [51]:
```
theta1 = find_Theta(x1_train, y1_train)
theta1
y1_pred = predict(x1_test,theta1)
```

In [52]:
```
y1_pred
```

Out[52]:
```
array([[0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
```

Defining RMSE

In [53]:
```
def rmse(predictions, targets):
    return np.sqrt(((predictions - targets) ** 2).mean())
rmse(y1_pred, y1_test)
```

Out[53]:
```
69.21591689502061
```

Defining MAE

In [54]:
```
def MAE(y_pred,y):
    res=np.sum(abs(y_pred-y))/len(y)
    return res
MAE(y1_pred, y1_test)
```

Out[54]:
```
68.43935264054514
```