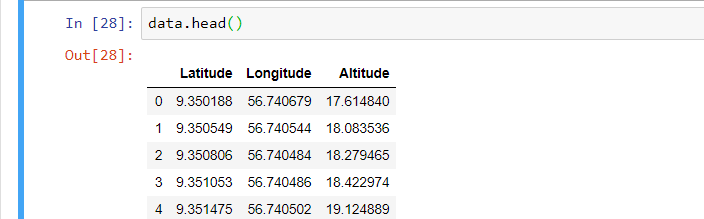
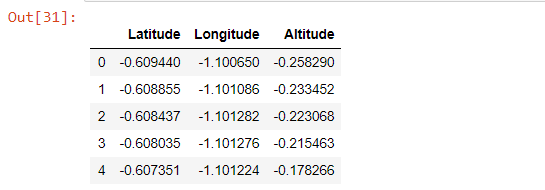
# Assignment 1

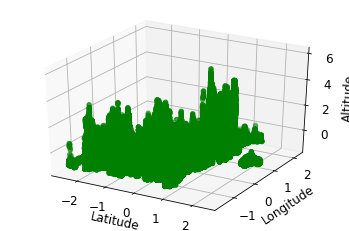
## Data Description:

The data set used in our analysis consists of 3 columns,which are as follows:  
1. Latitude(X1)  
2. Longitude(X2)   
3. Altitude(Y)

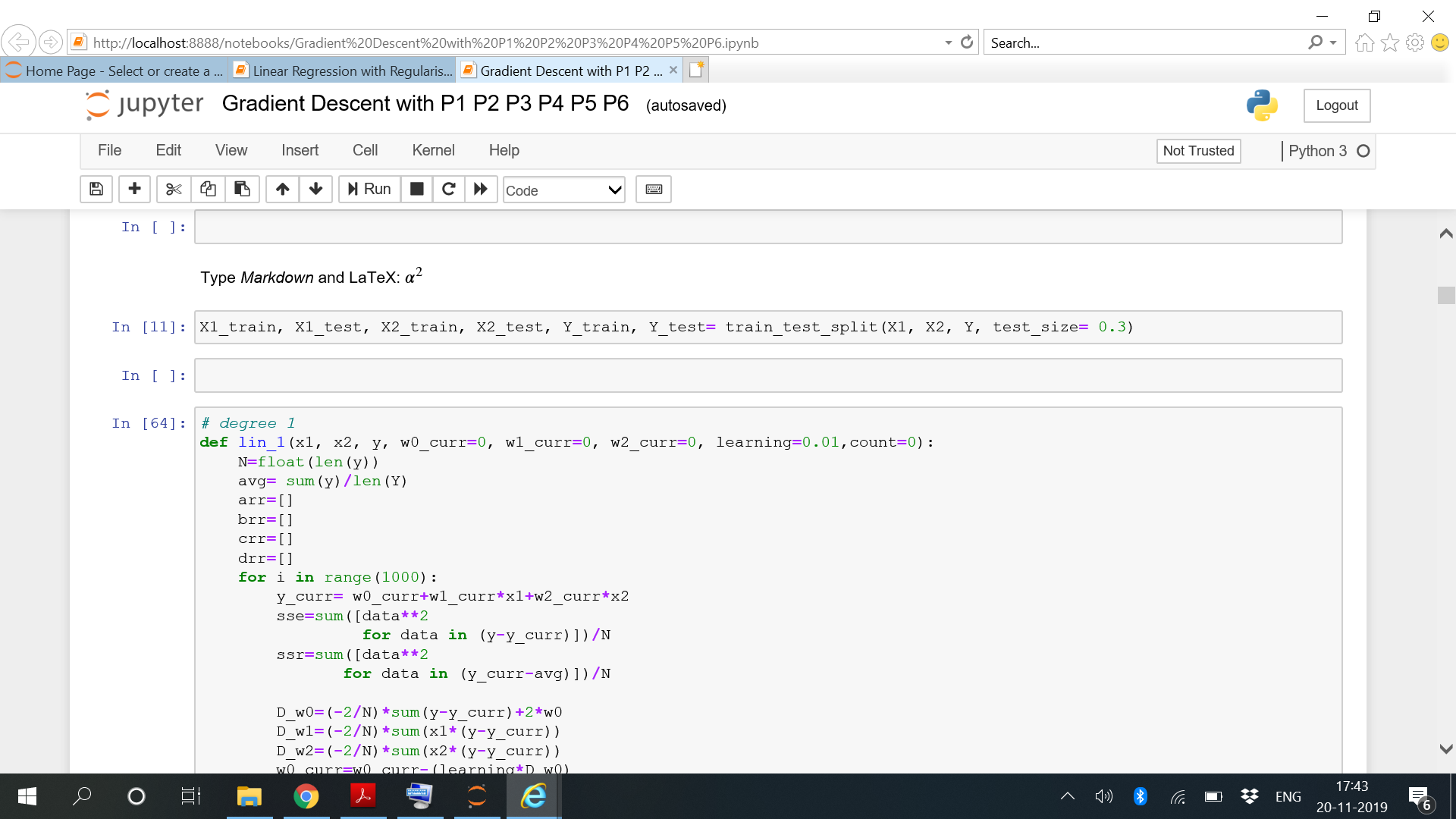


The size of our dataset is **1044628**  points. For more accurate results, we have **normalized** our data set. After normalization we got:

The scatter plot of the data is :



After the above mentioned step,the dataset was divided into 2 sets: Training Dataset, and Testing Dataset, with 30% of the data for testing purposes and 70% for training purposes.



# 

# Gradient Descent Method

Gradient descent is an optimization algorithm used to minimize some function by iteratively moving in the direction of steepest descent as defined by the negative of the gradient.

Math

Given the cost function:

f(m,b)=

The gradient can be calculated as:

f′(m,b)=⎡⎣df/dm/df/db⎤⎦=[1N∑−2xi(Yi−(mxi+b))1N∑−2(Yi−(mxi+b))]

To solve for the gradient, we iterate through our data points using our new m and b values and compute the partial derivatives. This new gradient tells us the slope of our cost function at our current position (current parameter values) and the direction we should move to update our parameters. The size of our update is controlled by the learning rate.Math

Given the cost function:

f(m,b)=

The gradient can be calculated as:

f′(m,b)=

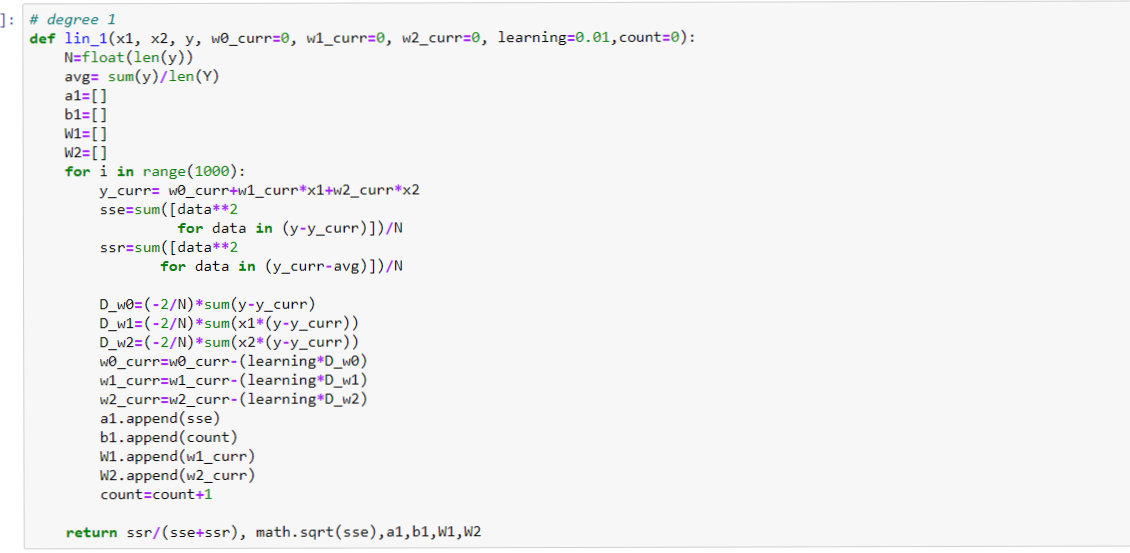
To solve for the gradient, we iterate through our data points using our new m and b values and compute the partial derivatives. This new gradient tells us the slope of our cost function at our current position (current parameter values) and the direction we should move to update our parameters. The size of our update is controlled by the learning rate.

Since we are building models for degree 1, here the implementation is done as shown below:

Learning rate=0.01

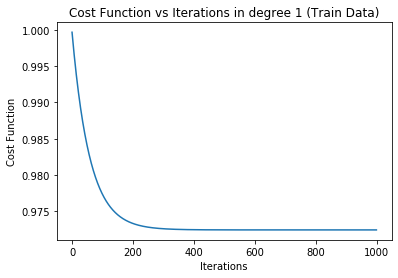
No of iterations =1000

Train: Test ratio= 70:30

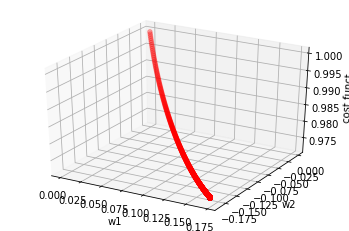


**TRAINING DATA:**

To see the variation of the cost function with no of iterations:



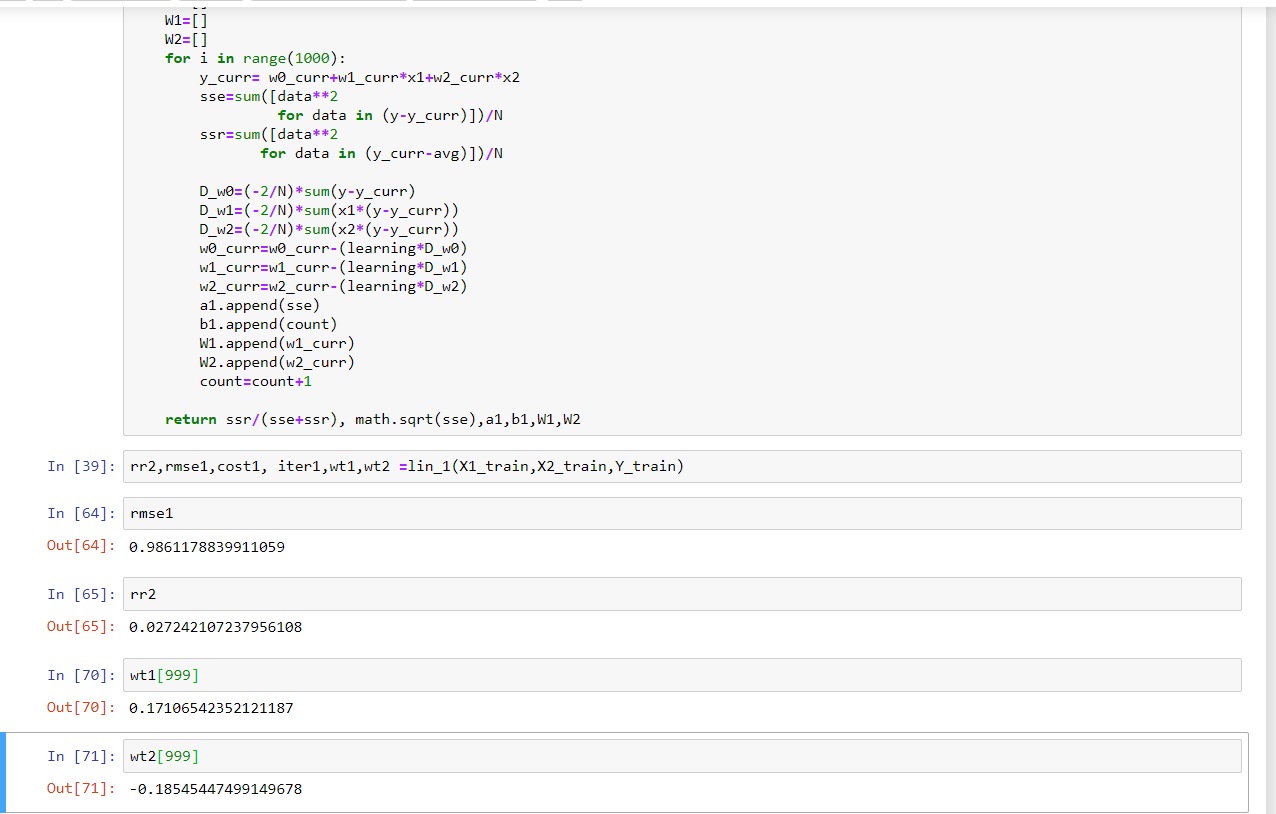
Our main aim is to get those coefficients of x1 and x2 that **minimize the cost function**. Those weights have been shown below:



The cost function is minimized as the  **w1 approaches 0.17106542352121187**

and **w2 approaches -0.18545447499149678.**

**The results of the code are shown below:**



**R Square =0.0272**

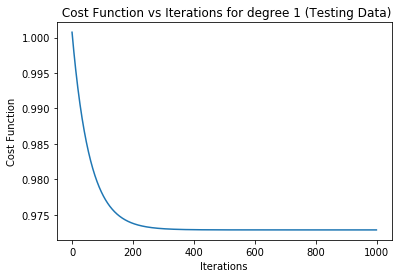
**RMSE=0.9861178839911059**

**Thus by using the gradient descent method for degree 1 on the training set, we get the equation of the curve as:**

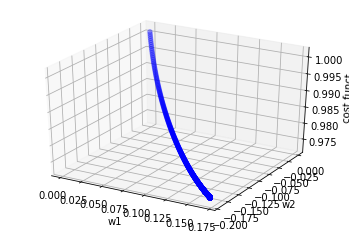
**Y= 0.17106542352121187\* X1 -0.18545447499149678\*X2+0.0002099386203973228**

**For Testing Data:**

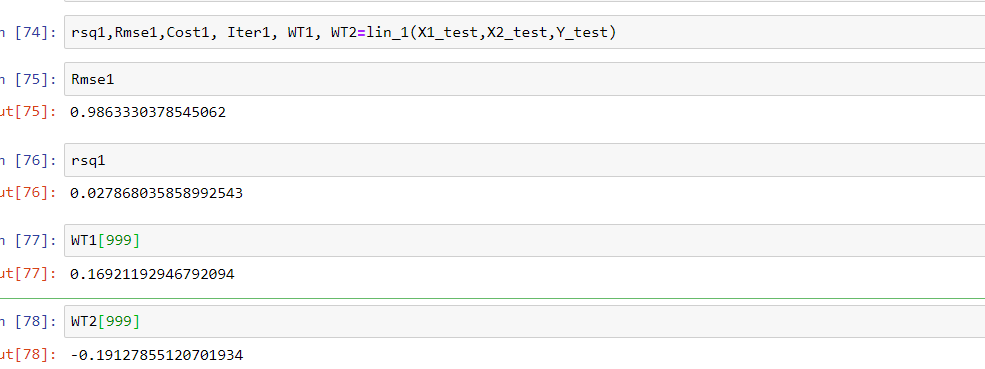
The variation in cost function with iterations is shown below:



The 3D plot of w1, w2, cost function shows that the cost function is minimized when **w1 approaches 0.16921192946792094** and **w2 approaches -0.19127855120701934**



The results for testing data were as follows:



**R Square= 0.027868035858992543**

**RMSE= 0.9863330378545062**

**Thus for the first degree, using the gradient descent method we get the equation as:**

**y= 0.16921192946792094\*X1 -0.19127855120701934\*X2 -0.0004846921239679693**

## Stochastic Gradient Descent

This method is used to optimize the method of gradient descent. In this method, we repeatedly run through the training set, and each time we encounter a training example, we update the parameters according to the gradient of the error with respect to that single training example only.

*Mini-batch stochastic gradient descent (mini-batch SGD)* is a compromise between full-batch iteration and SGD. A mini-batch is typically between 10 and 1,000 examples, chosen at random. Mini-batch SGD reduces the amount of noise in SGD but is still more efficient than full-batch.

The cost function is given as

**f=**

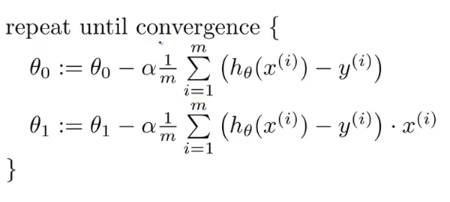
It’s derivative

**f’=**

And hence the weight updation for stochastic gradient is given as:

**Loop {**

**For i =1 :m {**

****

**}**

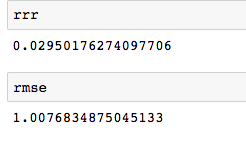
Learning rate=0.01

No of iterations =1000

Train: Test ratio= 70:30

Value of m=500

Code snippet:**For Training data-**



**=0.2950176**

**RMSE= 1.00768**

**Final coefficients:**

**w0=-0.0036**

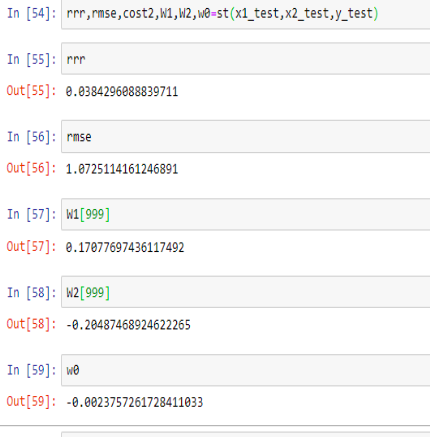
**w1=0.17788802282144467**

**w2=-0.19578726474990635**

**Thus by using the gradient descent method for degree 1 on the training set, we get the equation of the curve as:**

**Y= 0.17788802282144467\* X1 --0.19578726474990635\*X2-0.0036**

**For testing data**

****

**The value=0.0384296088829711**

**RMSE=1.0725114161246891**

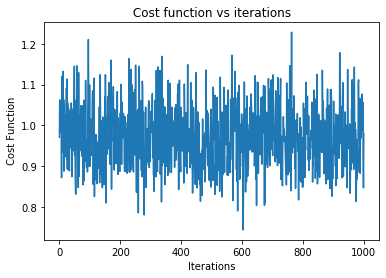
**The coefficients are:**

**w0=-0.0023757261728411033**

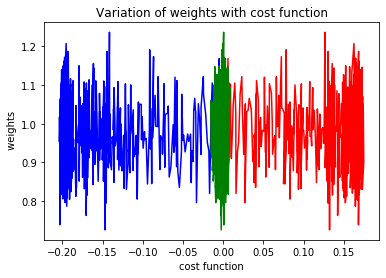
**w1=0.17077697436117492**

**w2=-0.20487468924622265**

The plot of cost function vs iterations is:



Variations of weights with cost function is shown below:



w1= red

w2= blue

w0= green

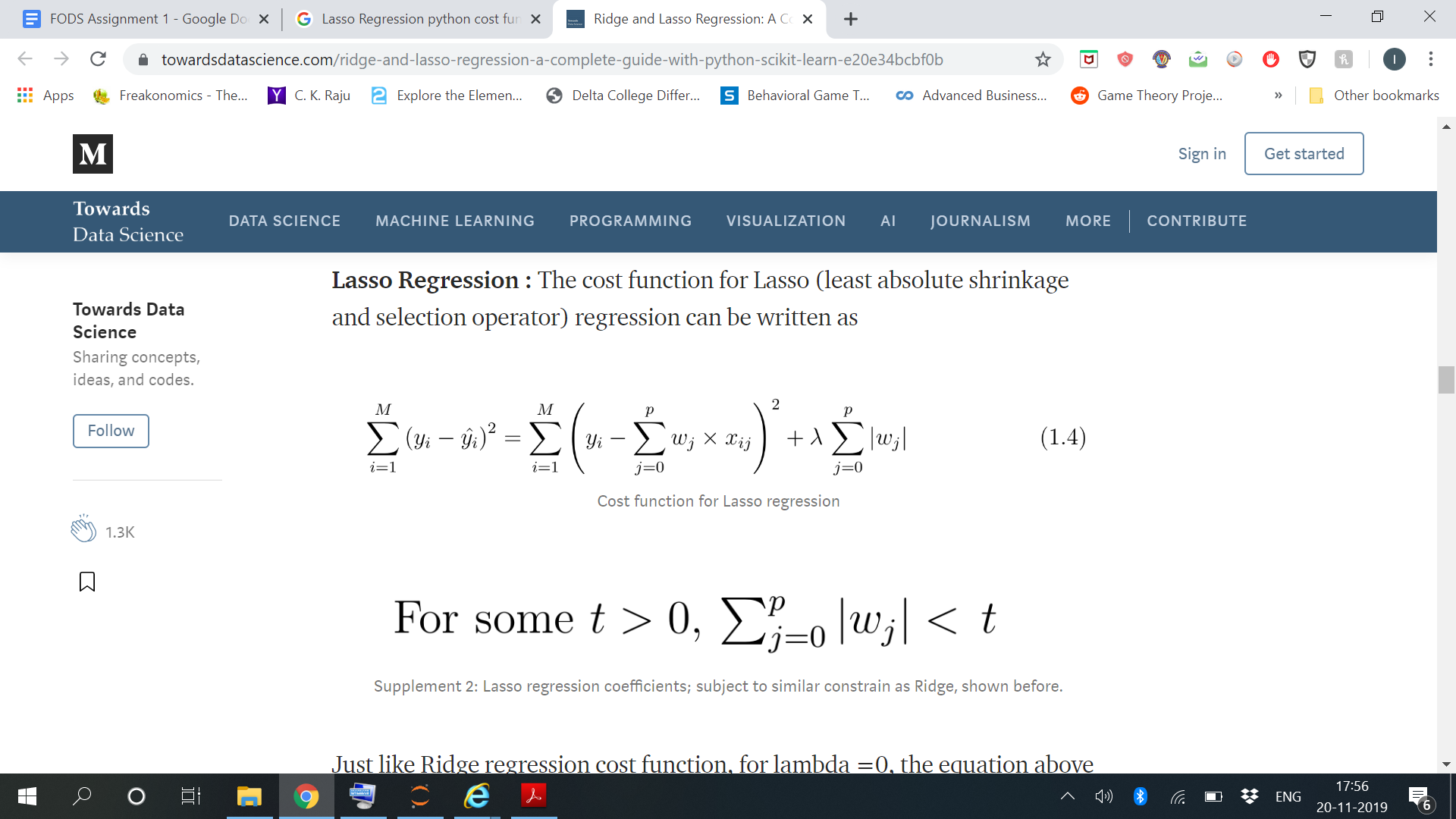
Gradient Descent Method with Regularisation  
  
The above methodology follows similar principle of optimisation as that of gradient descent. The only place where a fundamental difference arrives is in the development of Cost Function, and then subsequently, the Step Size. There are 2 types of regularisation:

1.L1 Regularisation (Lasso Regression)

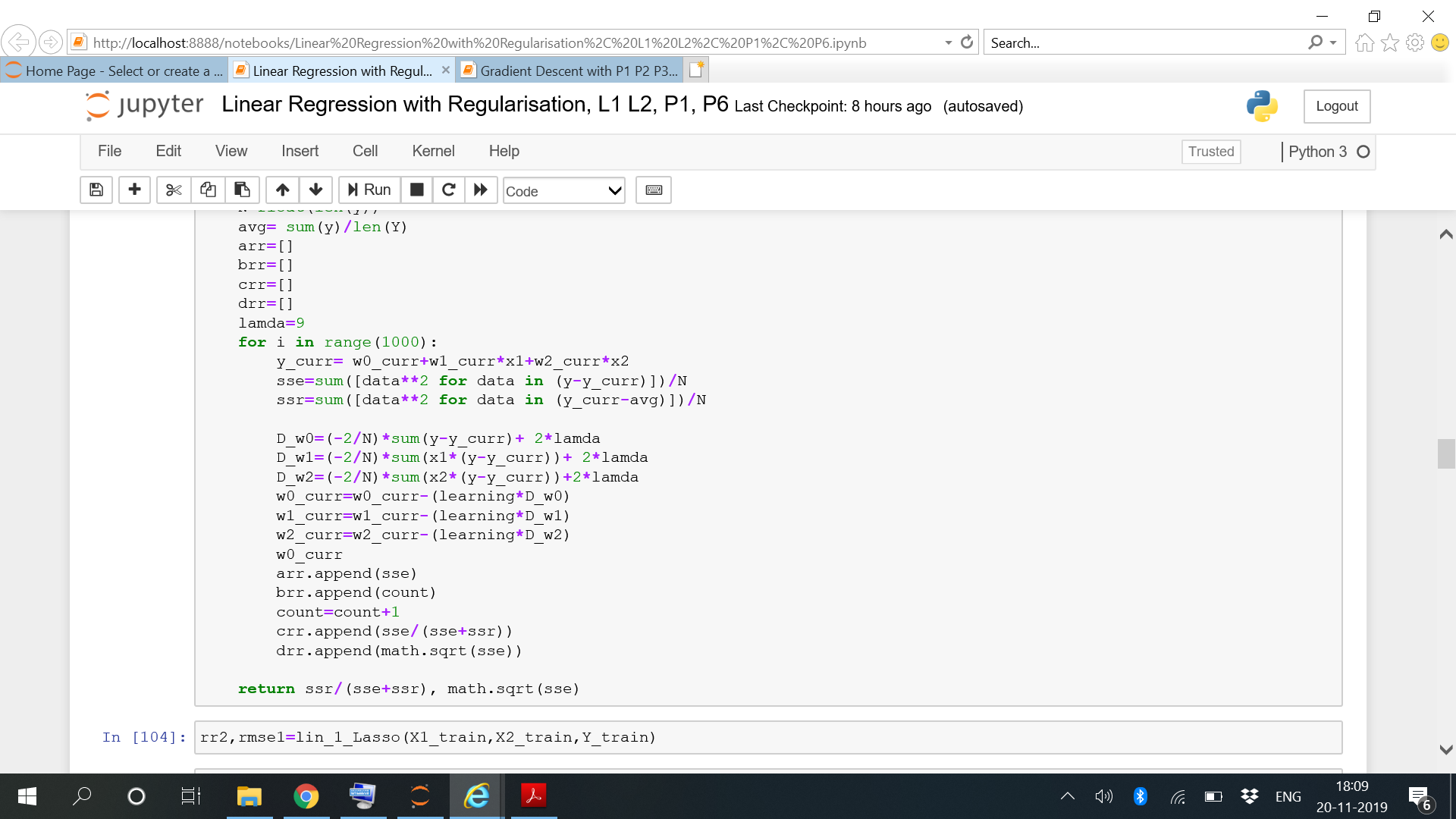
2. L2 Regularisation (Ridge Regression)

1. Lasso Regression

The Cost function of Lasso Regression is as follows



The following is the code snippet depicting the Cost Function and its derivative, and the subsequent updation of the parameters (The parameters to be evaluated have been initialised as 0):



The coefficients obtained after fitting the data are as follows:

w0= -9.362483235727861

w1= -6.537895301780843

w2= -5.567852915049161

The performance parameters of the model are as follows:

R2= 0.4985528815366945

RMSE= 13.94172707266491

After training the model on testing data, the final coefficients are as follows:

w0= -9.230759282507348

w1= -5.979521910680388

w2= -6.43621637175436

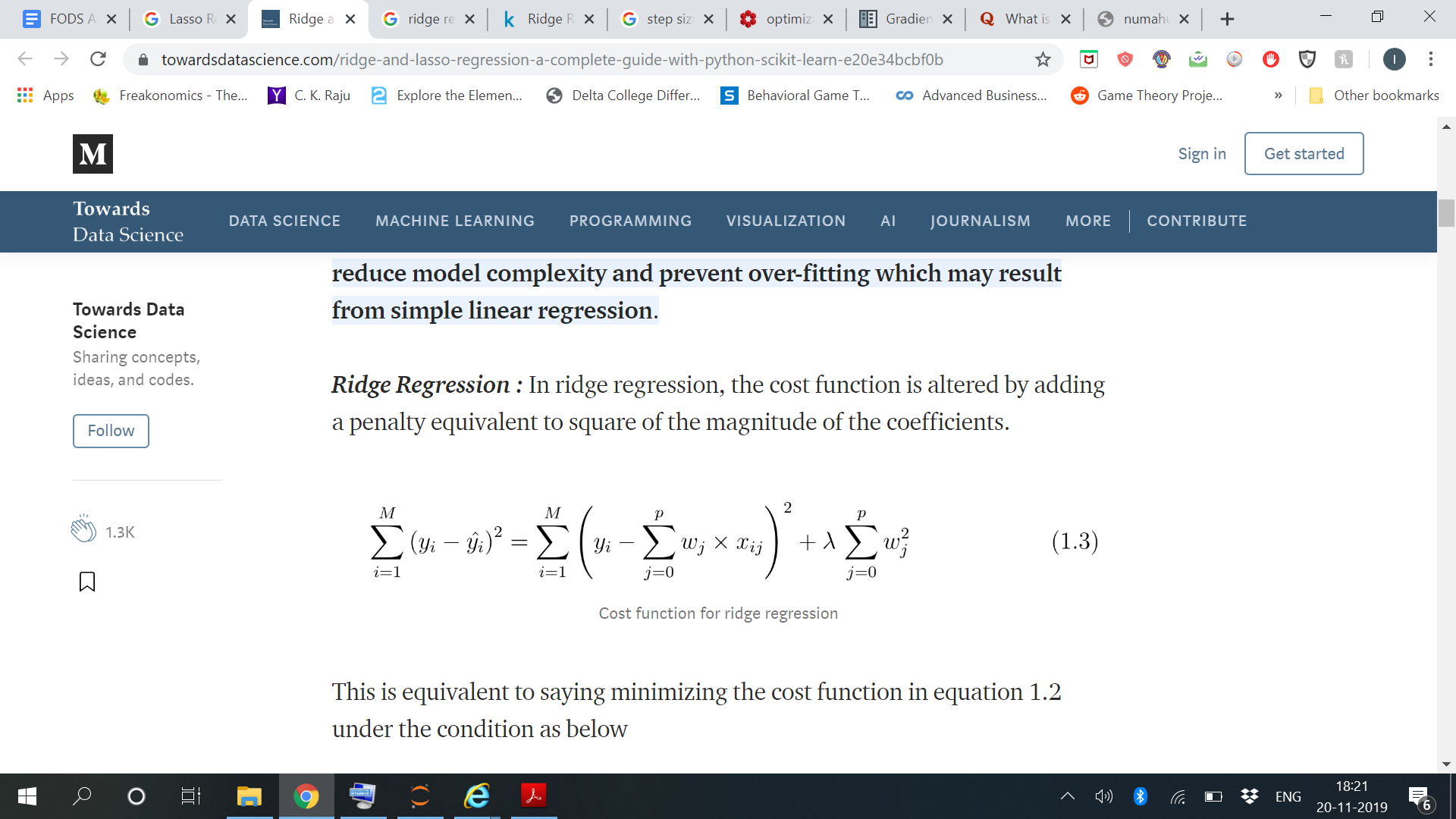
The final performance parameters of the model are as follows:

R2= 0.497338127729248

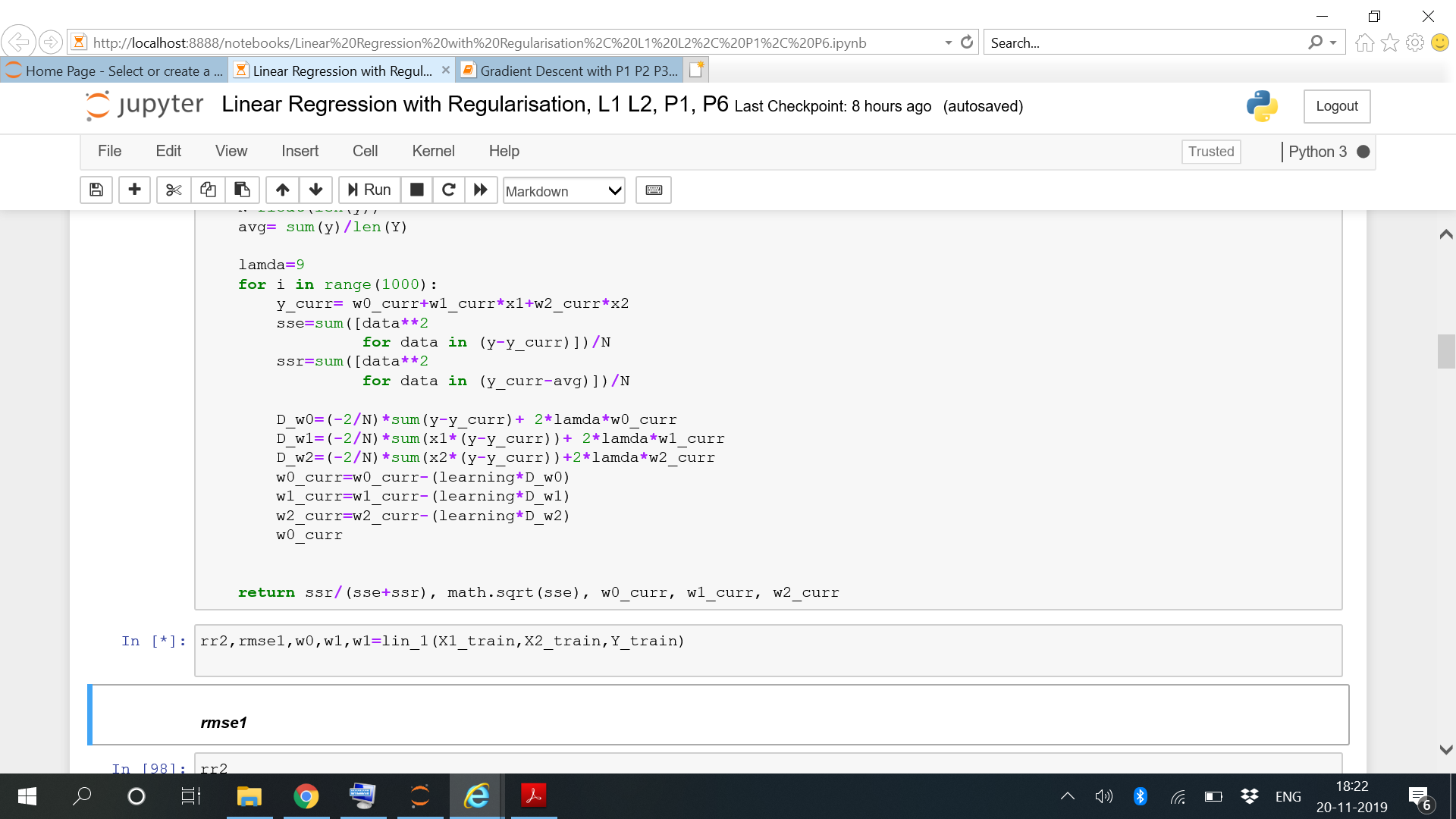
RMSE= 14.016972373035886

2. Ridge Regression

The Cost function of Ridge Regression is as follows:



The following is the code snippet depicting the Cost Function and its derivative, and the subsequent updation of the parameters (The parameters to be evaluated have been initialised as 0):



The coefficients obtained after fitting the training data are as follows:

w0= 0.0023363131020239323

w1= 0.007171253661183299

w2= -0.00954029030171071

The performance parameters of the model are as follows:

R2= 0.00011831895384501757

RMSE= 1.0079908690515829

After training the model on testing data, the final coefficients are as follows:

w0= 0.00522208282685183

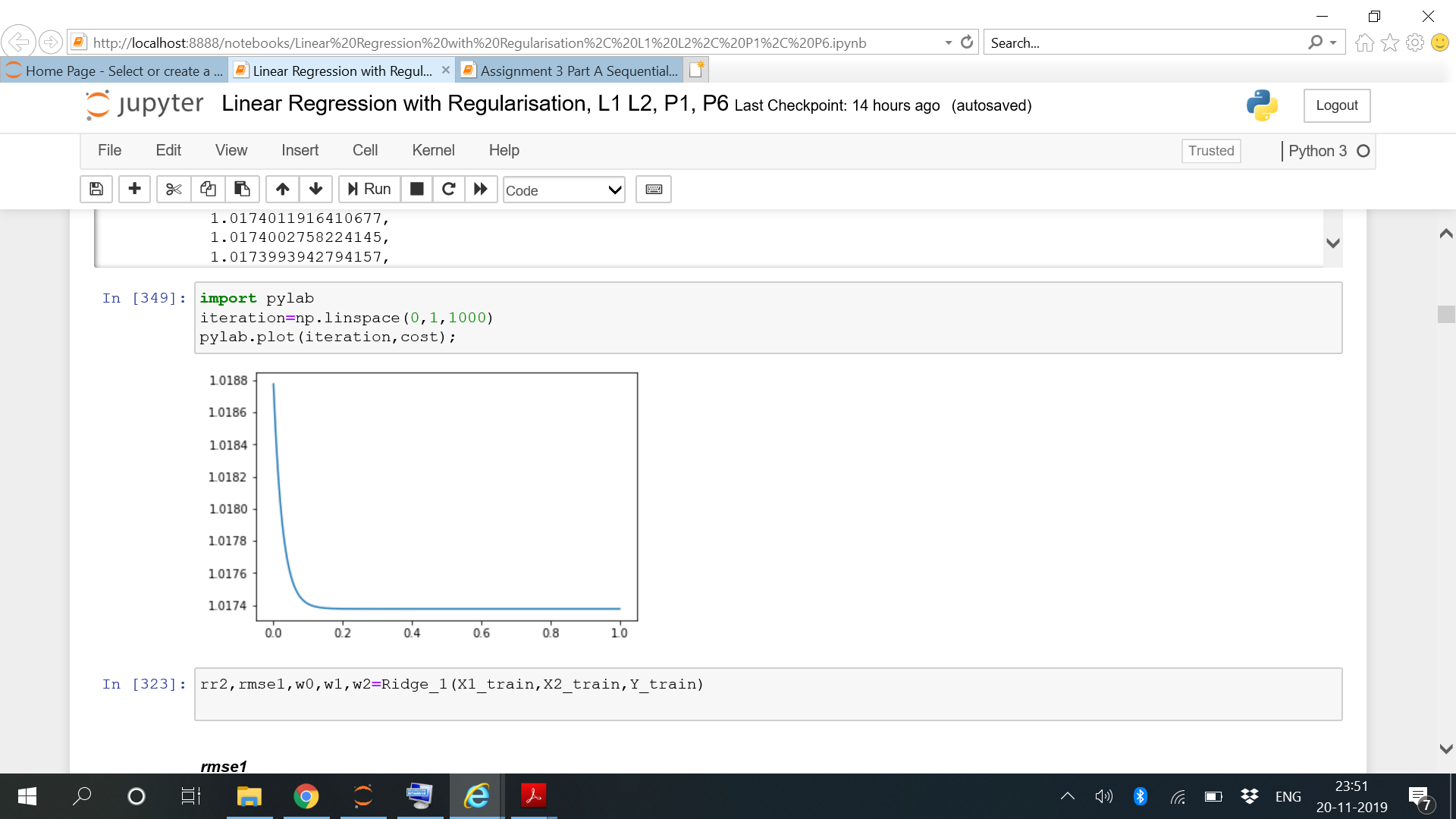
w1= 0.008658129090966611

w2= -0.006438931901060779

The final performance parameters of the model are as follows:

R2= 6.736090785399031e-05

RMSE= 1.030533960172263

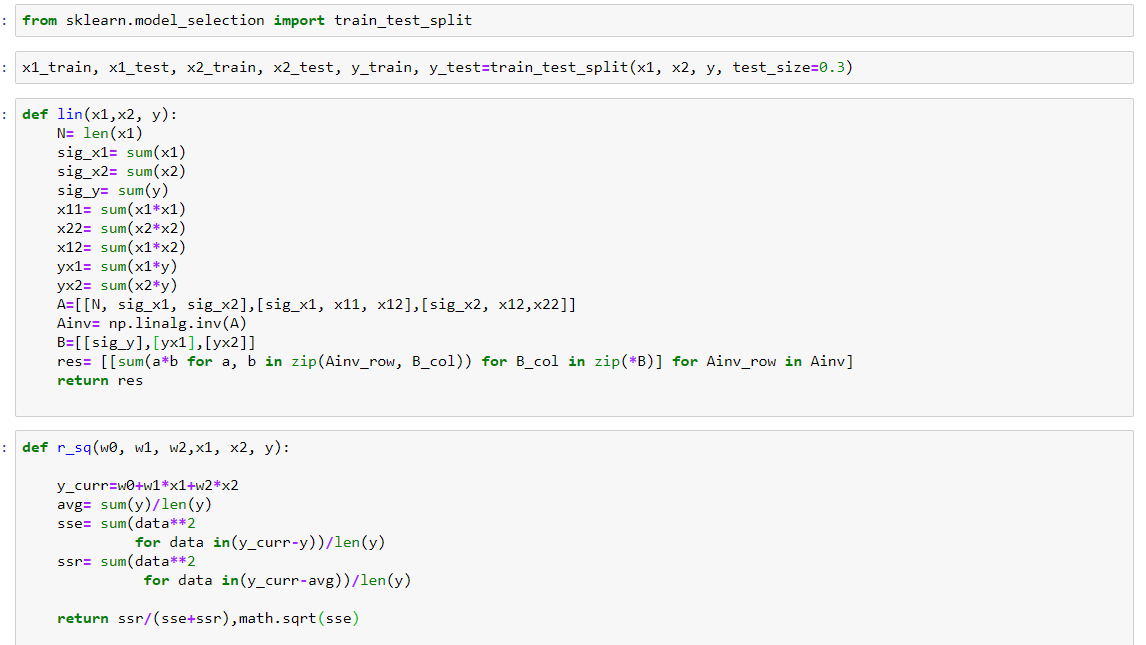


**Normal Equations Method:**

**Normal Equation** is an analytical approach to Linear Regression with a Least Square Cost Function. We can directly find out the value of θ without using Gradient Descent. It is an effective and time-saving option when are working with a dataset with small features.

The data is divided into **Training to testing in the ration 70:30**

The logic for the method is shown below:



For  **TRAINING DATA**  the results are:

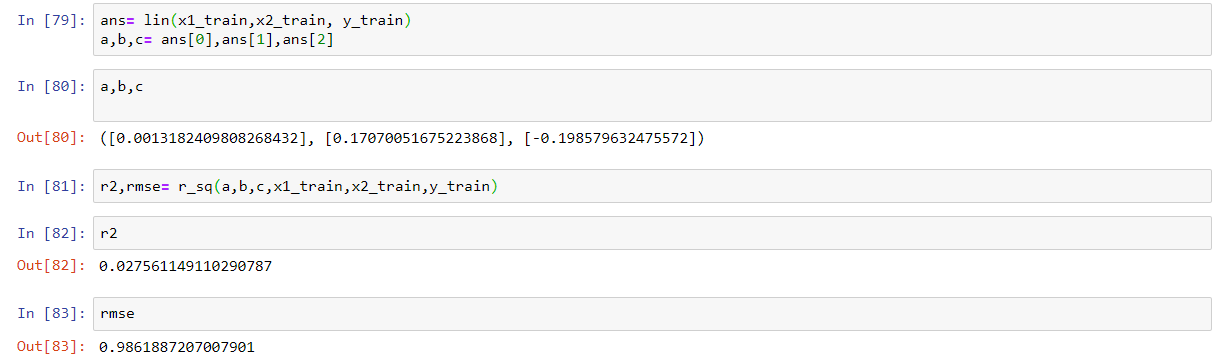
**w0= 0.0013182409808268432**

**w1= 0.17070051675223868**

**w2= -0.198579632475572**

**R sq= 0.027561149110290787**

**RMSE= 0.9861887207007901**



For  **TESTING DATA** the results are:

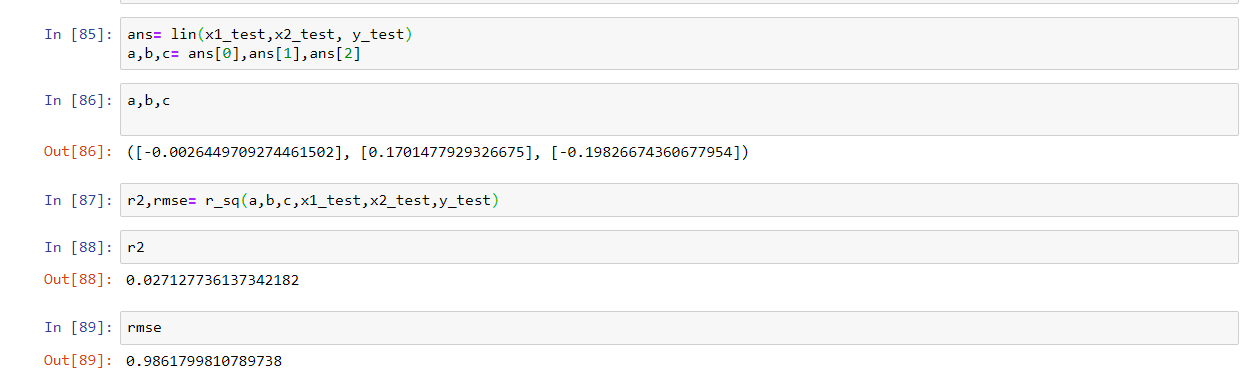
**W0=-0.0026449709274461502**

**w1=0.1701477929326675**

**w2=-0.19826674360677954**

**R sq=0.027127736137342182**

**RMSE=0.9861799810789738**

****

**Comparison of Models:-**

|  |  |  |
| --- | --- | --- |
| Model | R square | RMSE |
| Gradient Descent | 0.0272 | 0.986 |
| Stochastic Gradient Descent | 0.2950 | 1.0076 |
| Regularization(L1) | 0.4985 | 13.94 |
| Regularization(L2) | 0.005 | 1.007 |
| Normal equations | 0.02756 | 0.9861 |

**On the basis of R square : The best method is R square : Regularization(L1)> Stochastic Gradient> Normal Equation> Gradient Descent> Regularization(L2).**