Task 2

December 13, 2020

1 Task 2: Slide 46

1.0.1 Simple regression using car data for car price of a car (kms vs price)

1.1 Looking at the data

In [189]:
```
import numpy as np
import matplotlib.pyplot as plt

# import useful libraries
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import scipy.stats as stats
import csv
# this line plots graphs in line
%matplotlib inline
```

In [190]:
```
# downloaded on 13/12/2020
import csv
data=[]
roww=[]
with open('cars.csv', 'r') as file:
    reader = csv.reader(file)
    for row in reader:
        roww=[]
        print(row)
        roww=([float(row[0]),float(row[1])])
        data.append(roww)
```

```
['96000', '5500']
['145000', '5300']
['322395', '2000']
['40000', '6000']
['112000', '7000']
['202000', '2400']
['67000', '14500']
['39000', '7200']
```

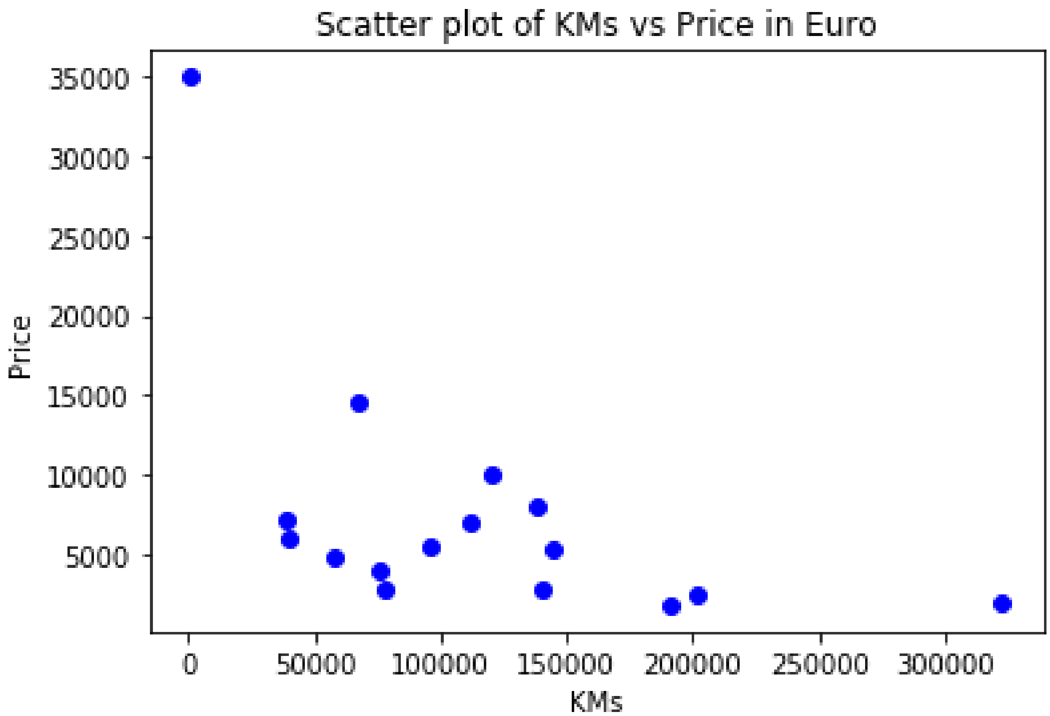
```
['120000', '10000']
['191158', '1800']
['140000', '2800']
['77668', '2750']
['727', '35000']
['58000', '4800']
['76303', '4000']
['138000', '8000']
```

```
In [191]: data= np.array(data)
```

```
In [192]: data
```

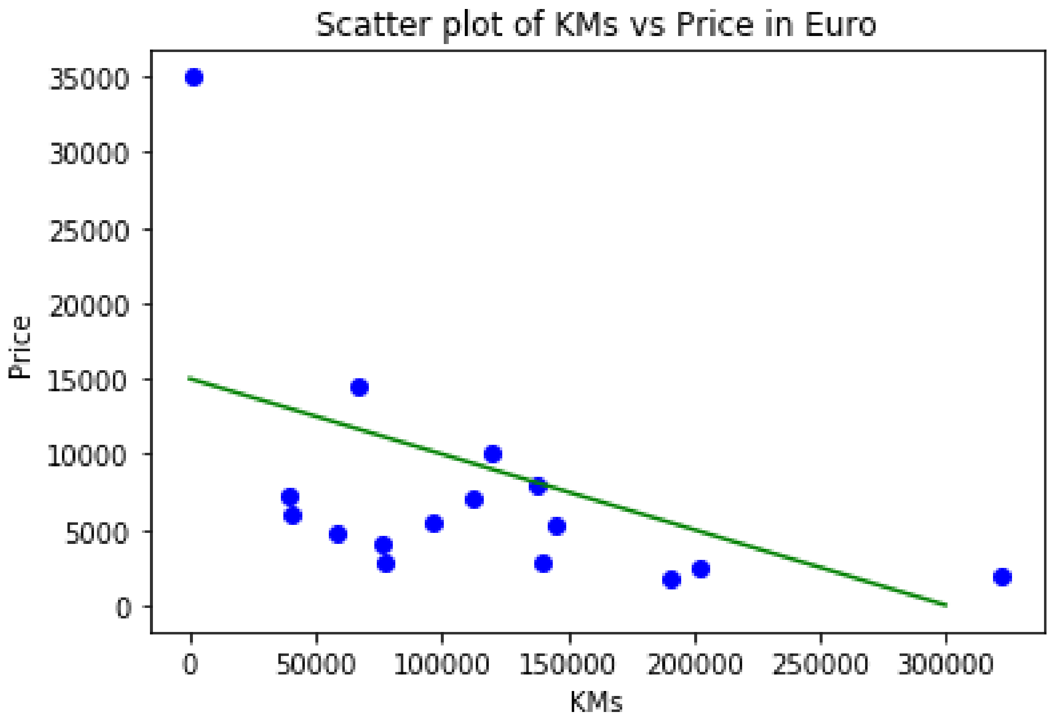
```
Out[192]: array([[ 96000.,   5500.],
                 [145000.,   5300.],
                 [322395.,   2000.],
                 [ 40000.,   6000.],
                 [112000.,   7000.],
                 [202000.,   2400.],
                 [ 67000.,  14500.],
                 [ 39000.,   7200.],
                 [120000.,  10000.],
                 [191158.,   1800.],
                 [140000.,   2800.],
                 [ 77668.,   2750.],
                 [   727.,  35000.],
                 [ 58000.,   4800.],
                 [ 76303.,   4000.],
                 [138000.,   8000.]])
```

```
In [123]: cars =plt.scatter(data[:,0],data[:,1], marker='o', color = 'b' )

          plt.xlabel('KMs')
          plt.ylabel('Price')
          plt.title('Scatter plot of KMs vs Price in Euro')

          plt.show()
```



1.2 Manually trying to fit a line (just to get oriented)

In [124]:
```
# eqn of line : y = mx + c
# Changing these manualy till I get something I am happy with
c= 15000
m =-0.05

#I just want to generate a bunch of x values so that I can see the line
x =np.array([0,300000])
y = m*x +c

#plotting scatter plot
students =plt.scatter(data[:,0],data[:,1], marker='o', color = 'b' )

plt.xlabel('KMs')
plt.ylabel('Price')
plt.title('Scatter plot of KMs vs Price in Euro')

plt.plot(x,y, 'g')

plt.show()
```



1.3 Ok, now I want a form of metric to measure distacne from each sample to my line

(even though I don't have my line yet, I think I need this to figure out the error, so I can minimise it) Params: line points (x and y), and the line m and c) Returns: A numeric value, representing my error

```
In [125]: def squareError(x,y, m,c):
              predictedValue = m*x +c
              err = predictedValue - y
              return err **2
```

1.4 Just for the sake of trying, I am going to try fit m and c using random numbers.

I guess I am cheating slightly by norrowing down my range with the values I found manually.

```
In [126]: iterations = 100000
          maxM = 0
          minM = -1

          maxC =40000
          minC =5000
```

```
Besterror =9999999999
bestWeights=[0,0]
errors = []

for i in range(iterations):
    #print(i)
    errors = []

    m = np.random.uniform(low=minM, high=maxM)

    c = np.random.uniform(low=minC, high=maxC)
    currentVals = [m ,c]
    for sample in data:
        errors.append(squareError(sample[0], sample[1],m,c ))
    Currenterror= sum(errors)/ len(data)


    if(Besterror > Currenterror):
        bestWeights = currentVals
        Besterror = Currenterror



print("------ Final -----")
print ("Best error:",Besterror)

print("Best weghts:",bestWeights)
```

```
------ Final -----
Best error: 43872301.95601188
Best weghts: [-0.05473226690970334, 13574.649593571408]
```

In [ ]:

1.5 Now I can try it properly with gradient decesent

In [ ]:

In [ ]:

In [127]:
```
xy = np.multiply(data[:,0],data[:,1])
ySquared = np.square(data[:,1])
xSquared = np.square(data[:,0])
```

In [128]:
```
def f (m,c , data):
    X = data[:,0]
    Y = data[:,1]
    return m**2  + (2*c*m*sum(X))- (2*m*sum(xy) )- (2*c*sum(Y))+ (2*(c**2))+ sum(ySqu
```

```
In [ ]:
```

```
In [47]: #Partial differntiantios on X and Y
         def df_dc(m,c,data):
             return (m*sum(data[:,0])) -sum(data[:,1]) + (len(data)*c)
         def df_dm(m,c,data):
             return (m*(sum(xSquared))) + (c*sum(data[:,0])) -(sum(xy))
```

```
In [129]: # Building the model
          m = 0
          c = 10000

          L = 0.000000000001  # The learning Rate
          interations = 1000000 # The number of iterations to perform gradient descent
          inputC =np.zeros(interations)
          inputM=np.zeros(interations)

          n = (len(data)) # Number of elements in X

          # Performing Gradient Descent
          for i in range(interations):

              #print (m, c)


              #calculate how much we want to change, ie the change from the differencation (ho
              delatM = df_dm(m,c,data)
              deltaC = df_dc(m,c,data)
              m = m - L * delatM  # Update m
              c = c - L * deltaC  # Update c
              #Storing these to try plot them later
              inputM[i]= m
              inputC[i]= c

          print (m, c)
```

```
-0.032297668589456756 10000.018000522607
```

1.5.1 Having issues with overflow and time, going to scale

```
In [195]: dataXScaled = (((data[:,0])-np.mean(data[:,0])) / np.std(data[:,0]))
          dataYScaled = (((data[:,1])-np.mean(data[:,1])) / np.std(data[:,1]))
          dataScaled = [dataXScaled, dataYScaled]

          dataS =np.array([dataXScaled,dataYScaled] )

          dataS = np.transpose(dataS)
          print(dataS)
```

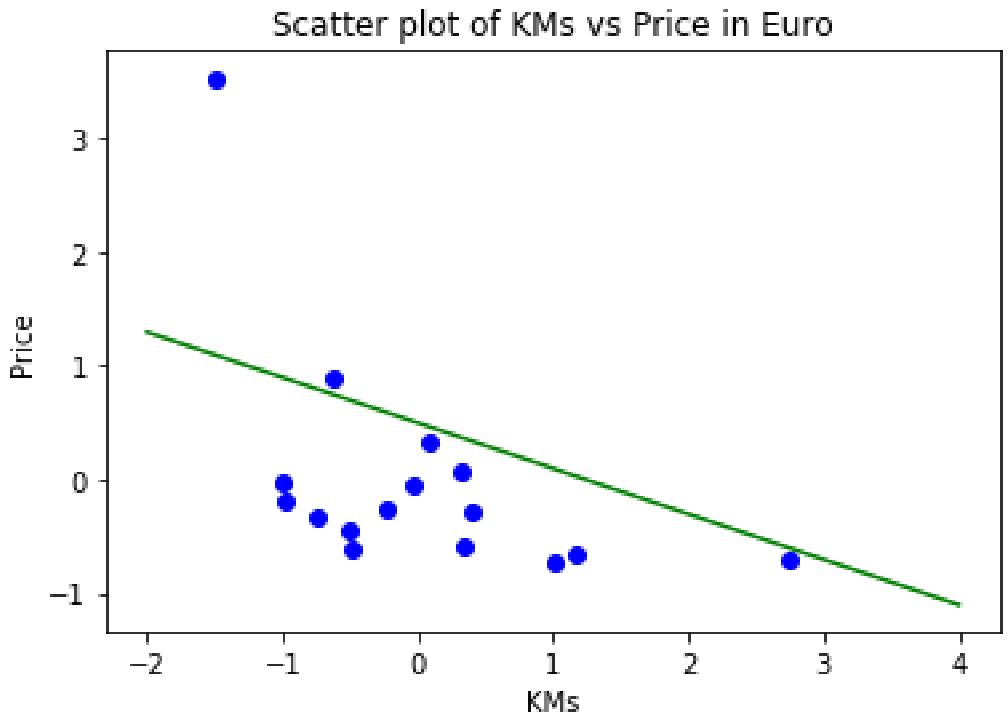
```
[[-0.23877094 -0.24846316]
 [ 0.40840545 -0.27406967]
 [ 2.75138207 -0.69657706]
 [-0.97840109 -0.18444688]
 [-0.02744804 -0.05641434]
 [ 1.16124328 -0.64536405]
 [-0.6217937   0.90382975]
 [-0.99160877 -0.03080783]
 [ 0.07821341  0.32768329]
 [ 1.0180456  -0.72218357]
 [ 0.34236704 -0.59415103]
 [-0.48089415 -0.60055266]
 [-1.49710636  3.52849692]
 [-0.74066283 -0.33808594]
 [-0.49892264 -0.44051197]
 [ 0.31595168  0.0716182 ]]
```

In [131]:
```
(dataS[:,0].std())
```

Out[131]: 1.0

Testing manually again

In [132]:
```
data= dataS
# eqn of line : y = mx + c
# Changing these manualy till I get something I am happy with
c= 0.5
m =-0.4

#I just want to generate a bunch of x values so that I can see the line
x =np.array([-2,4])
y = m*x +c

#plotting scatter plot
students =plt.scatter(data[:,0],data[:,1], marker='o', color = 'b' )

plt.xlabel('KMs')
plt.ylabel('Price')
plt.title('Scatter plot of KMs vs Price in Euro')


plt.plot(x,y, 'g')

plt.show()
```




```
In [140]: print("R squared: ", rSquared(m,c,data))

R squared:  0.2702680133204045
```

```
In [149]: # Building the model
          m = -0.5
          c = 0

          L = 0.0000000000000000005  # The learning Rate
          interations = 1000000 # The number of iterations to perform gradient descent
          inputC =np.zeros(interations)
          inputM=np.zeros(interations)

          n = (len(data)) # Number of elements in X

          # Performing Gradient Descent
          for i in range(interations):

              #print (m, c)

              errors = []

              #calculate how much we want to change, ie the change from the differencation (ho
```

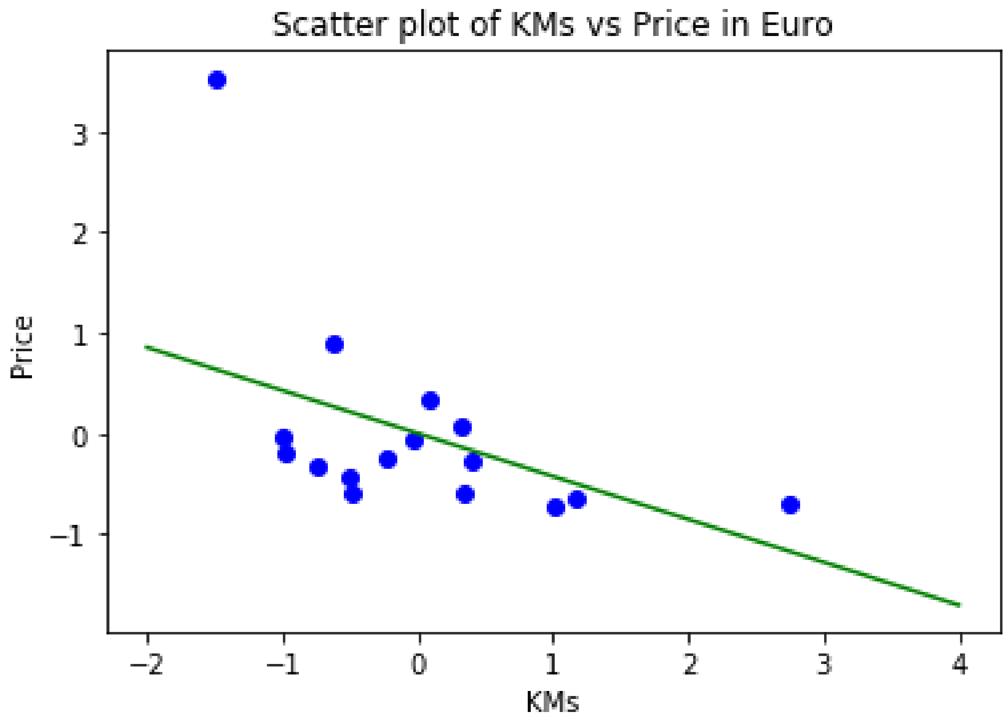
```
        delatM = df_dm(m,c,data)
        deltaC = df_dc(m,c,data)
        m = m - L * delatM  # Update m
        c = c - L * deltaC  # Update c
        #Storing these to try plot them later
        inputM[i]= m
        inputC[i]= c

        for sample in data:
            errors.append(squareError(sample[0], sample[1],m,c ))
        Currenterror= sum(errors)/len(data)
        #print(Currenterror)
        if(Currenterror<0.5):
            print("convergance")
            break

    print (m, c)
```

```
-0.4263895840193148 1.932419563628756e-28
```

In [150]:
```
print("R squared: ", rSquared(m,c,data))
```

```
R squared:  0.2702680228731169
```

In [153]:
```
#using the m and c computed before, to see how it looks

#I just want to generate a bunch of x values so that I can see the line
x =np.array([-2,4])
y = m*x +c

#plotting scatter plot
students =plt.scatter(data[:,0],data[:,1], marker='o', color = 'b' )


plt.xlabel('KMs')
plt.ylabel('Price')
plt.title('Scatter plot of KMs vs Price in Euro')


plt.plot(x,y, 'g')

plt.show()
```



In [ ]:

2 Now I am going to consider the highest value as an outlier, and I will just eliminate it

In [198]: dataS

Out[198]:
```
array([[-0.23877094, -0.24846316],
       [ 0.40840545, -0.27406967],
       [ 2.75138207, -0.69657706],
       [-0.97840109, -0.18444688],
       [-0.02744804, -0.05641434],
       [ 1.16124328, -0.64536405],
       [-0.6217937 ,  0.90382975],
       [-0.99160877, -0.03080783],
       [ 0.07821341,  0.32768329],
       [ 1.0180456 , -0.72218357],
       [ 0.34236704, -0.59415103],
       [-0.48089415, -0.60055266],
       [-1.49710636,  3.52849692],
       [-0.74066283, -0.33808594],
       [-0.49892264, -0.44051197],
       [ 0.31595168,  0.0716182 ]])
```

In [215]:
```
Clean =dataS[:12]
Clean2 =dataS[-3:]
```

In [216]:
```
Clean2
```

Out[216]:
```
array([[-0.74066283, -0.33808594],
       [-0.49892264, -0.44051197],
       [ 0.31595168,  0.0716182 ]])
```

In [217]:
```
Cleaned= Clean
```

In [218]:
```
np.append(Clean, Clean2, axis=0)
```

Out[218]:
```
array([[-0.23877094, -0.24846316],
       [ 0.40840545, -0.27406967],
       [ 2.75138207, -0.69657706],
       [-0.97840109, -0.18444688],
       [-0.02744804, -0.05641434],
       [ 1.16124328, -0.64536405],
       [-0.6217937 ,  0.90382975],
       [-0.99160877, -0.03080783],
       [ 0.07821341,  0.32768329],
       [ 1.0180456 , -0.72218357],
       [ 0.34236704, -0.59415103],
       [-0.48089415, -0.60055266],
       [-0.74066283, -0.33808594],
       [-0.49892264, -0.44051197],
       [ 0.31595168,  0.0716182 ]])
```

In [265]:
```
# Building the model
data=Cleaned
m = -0.5
c = -0.1

L = 0.0000000000000001  # The learning Rate
interations = 10000 # The number of iterations to perform gradient descent
inputC =np.zeros(interations)
inputM=np.zeros(interations)

n = (len(data)) # Number of elements in X

# Performing Gradient Descent
for i in range(interations):

    #print (m, c)

    errors = []

    #calculate how much we want to change, ie the change from the differencation (ho
```

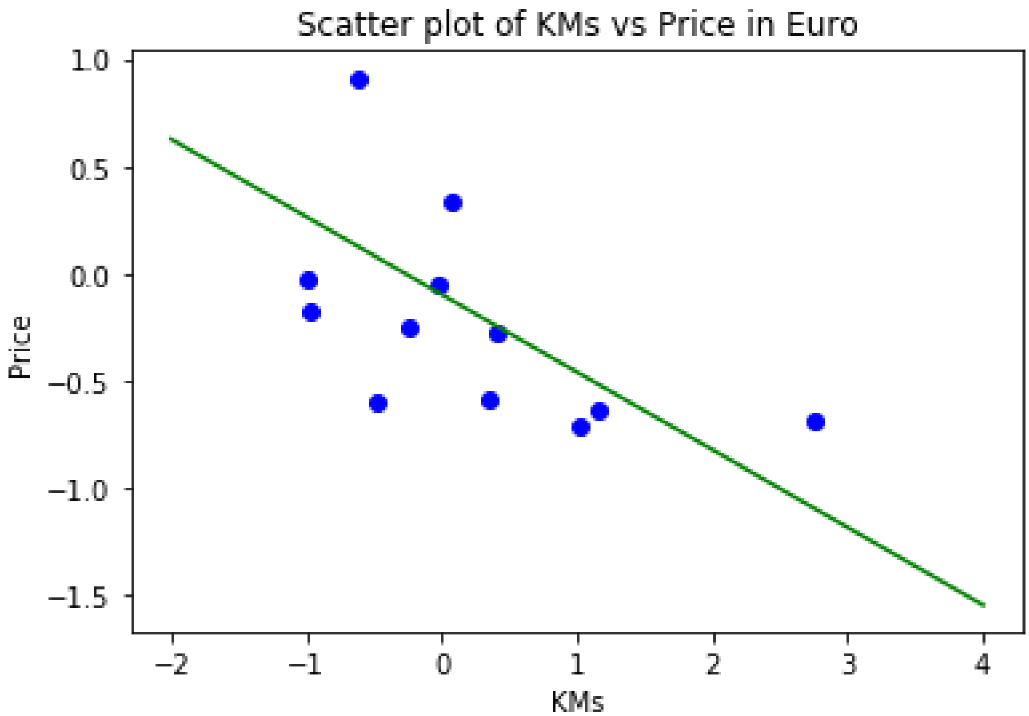
```
        delatM = df_dm(m,c,data)
        deltaC = df_dc(m,c,data)
        m = m - L * delatM  # Update m
        c = c - L * deltaC  # Update c
        #Storing these to try plot them later
        inputM[i]= m
        inputC[i]= c

        for sample in data:
            errors.append(squareError(sample[0], sample[1],m,c ))
        Currenterror= sum(errors)/len(data)
        #print(Currenterror)
        if(Currenterror<0.05):
            print("convergance")
            break

    print (m, c)
```

```
-0.3630289196839915 -0.10000000000058169
```

In [266]:
```
print("R squared: ", rSquared(m,c,data))
```

```
R squared:  0.22966963442067545
```

In [267]:
```
#using the m and c computed before, to see how it looks

#I just want to generate a bunch of x values so that I can see the line
x =np.array([-2,4])
y = m*x +c

#plotting scatter plot
students =plt.scatter(data[:,0],data[:,1], marker='o', color = 'b' )


plt.xlabel('KMs')
plt.ylabel('Price')
plt.title('Scatter plot of KMs vs Price in Euro')


plt.plot(x,y, 'g')

plt.show()
```



In [ ]:

In [ ]:

3 Seeing R squared

In [145]:
```
def rSquared(m,c , data):
    Y =data[:,1]
    X =data[:,0]
    meanY = np.mean(Y)
    predY = m*X +c

    numerator = sum((Y- predY)**2)
    denumerator = sum((Y- meanY)**2)

    return 1-(numerator/denumerator)
```

In [146]:
```
print("R squared: ", rSquared(m,c,data))
```

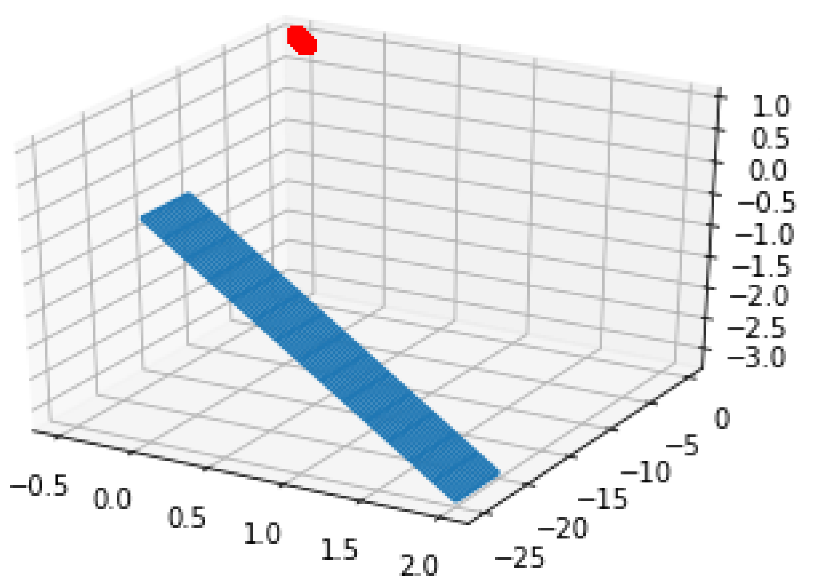
```
R squared:  0.2702680228731169
```

In [ ]:

In [ ]:

In [151]:
```
#trying to plot it, I  think I have a problem here

minimumZ =f(inputM,inputC,data)


#-------- Printing --------

N=15
x=np.linspace(0,2, N)
y=np.linspace(-25,-20, N)


X,Y = np.meshgrid(x,y)
Z = f(X , Y,data)


fig=plt.figure()
ax = fig.add_subplot(111,projection="3d")
ax.plot_wireframe(X,Y,Z)


ax.plot(inputM,inputC, minimumZ, 'ro')
plt.show
```

Out[151]: <function matplotlib.pyplot.show(*args, **kw)>



```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```