SOLUTION DISCUSSION

By Nitin Kumar

IIT ROPAR

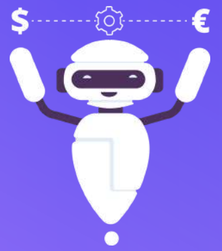
THE CHALLENGE

The question was very closely related to the arena of physics but still did not require much knowledge of the same. My first impression on seeing the question was not at all trivial. I approached the question in a way such that it could be done using a the most basic of all Data science models, Linear Regression. I have used a Bayesian approach(which was recommended in the problem statement), and combined it with the Linear model to get as precise output as possible.

PROJECT OBJECTIVES

OBJECTIVE 01

- To find the set of constants given in the formula by using the data given



OBJECTIVE 02

- To keep the error in the answer as low as possible

OBJECTIVE 03

- To analyze the values of the parameters and to check how accurate they are.

PROPOSED SOLUTION

- Here, I observed that the formula need can be reconstructed into a linear relation between input and output.

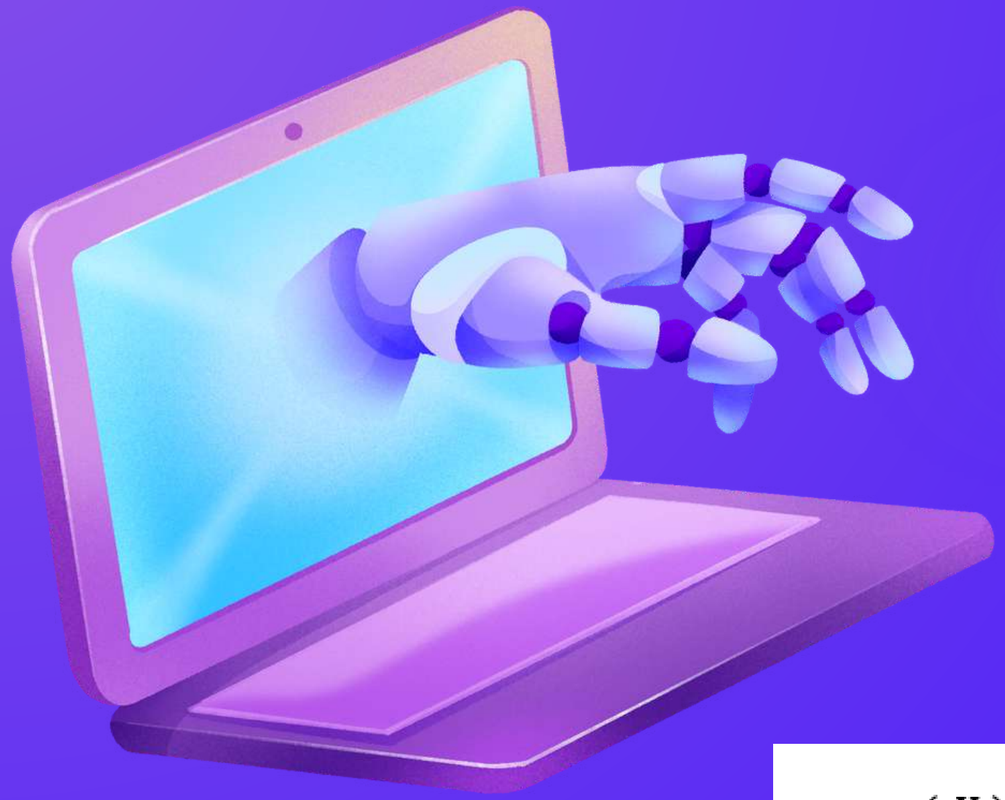
$$H(z) = H_0 \left[\Omega_m (1 + z)^3 + \Omega_k (1 + z)^2 + \Omega_\Lambda \right]^{1/2}$$

$$\left(\frac{H(z)}{H_0}\right)^2 = \Omega_m (1 + z)^3 + \Omega_k (1 + z)^2 + \Omega_\Lambda$$

$$y = \Omega_m x_1 + \Omega_k x_2 + \Omega_\Lambda$$

where, $x_1 = (1 + z)^3$, $x_2 = (1 + z)^2$, $y = \left(\frac{H(z)}{H_0}\right)^2$

- Now, one can easily observe that if we construct our feature vector so it contains the required powers of 1+z, and also reconstruct our output such that it is now the square of ratio of H and H0, we will be able to apply a simple linear regression and find the values of the parameters which is exactly what we need to find.



PARAMETER RECONSTRUCTION

Here I had to calculate the parameters and their errors seperately from each other rather than together.

So, I used the concept of errors from physics-

$$Z = \left(\frac{H}{H_0}\right)^2$$

$$\Delta Z = Z\left(\frac{\Delta H}{H} + \frac{\Delta H_0}{H_0}\right) = \left(\frac{H}{H_0}\right)^2\left(\frac{\Delta H}{H} + \frac{\Delta H_0}{H_0}\right)$$

I used these set of values for finding the parameters.

RESULT

- So, the final answer reaches to:
- Wm= 0.1646535 +- 0.0313167
- WK= 0.35964872 +- 0.01680515
- W= 0.2827349977938409 +- 0.5604636445724958

```
print(model1.coef_,model1.intercept_)
[7] ✓ 0.0s
... [0.1646535  0.35964872] 0.2827349977938409

print(model2d.coef_,model2d.intercept_)
[8] ✓ 0.0s
... [0.03131676 0.01680515] 0.5604636445724958

pred1=model1.predict(x.T)
pred2=model2d.predict(x.T)
[9] ✓ 0.0s

from sklearn.metrics import mean_absolute_error
error1=mean_absolute_error(y,pred1)
print(error1)
[10] ✓ 0.0s
... 0.43539831225997006

error2=mean_absolute_error(dely,pred2)
print(error2)
[11] ✓ 0.0s
... 0.4348458268086364
```

- I have reached these conclusions and have then verified the result by calculating the mean absolute for each term. And the error observed is fairly small indicating that the values of the parameters are very precise.

THANK YOU!

This is new project for me where i am given a data of a set of inputs and output corresponding to each input and also the formula which can be used to find the value of the output from the input with some error. Now my task is to find the value of the three constants which are used in the formula that gives the output from the input.

Importing the necessary libraries which are needed

In [3]:
```
import numpy as np
import pandas as pd
```

now we import the dataset

In [4]:
```
df= pd.read_csv('dataset.csv')
df.head()
```

Out[4]:

	Redshift	Hubble parameter(km/s/Mpc)	Error in Hubble Parameter(km/s/Mpc)
0	0.070	69.0	19.6
1	0.100	69.0	12.0
2	0.120	68.6	26.2
3	0.170	83.0	8.0
4	0.179	75.0	4.0

now we convert the pandas dataframe which contain the data into numpy arrays which are fairly easy to work with

In [5]:
```
xd=np.array(df['Redshift'])
yd=np.array(df['Hubble parameter(km/s/Mpc)'])
delyd=np.array(df['Error in Hubble Parameter(km/s/Mpc)'])
print(xd,yd,delyd)
```

```
[0.07  0.1   0.12  0.17  0.179 0.199 0.2   0.27  0.28  0.35  0.352 0.4
 0.44  0.48  0.593 0.6   0.68  0.73  0.781 0.875 0.88  0.9   1.037 1.3
 1.43  1.53  1.75  2.3  ] [ 69.   69.   68.6  83.   75.   75.   72.9  77.   88.8  7
6.3  83.   95.
  82.6  97.  104.   87.9  92.   97.3 105.  125.   90.  117.  154.  168.
 177.  140.  202.  224. ] [19.6 12.  26.2  8.   4.   5.  29.6 14.  36.6  5.6 14.  1
7.   7.8 62.
 13.   6.1  8.   7.  12.  17.  40.  23.  20.  17.  18.  14.  40.   8. ]
```

Now we have to make the necessary changes in the dataset which are mentioned in the presentation. So we convert the single feature input into a 2D array (feature1= (1+x)**3 and **feature2= (1+x)**2) and also change the output as per the formula. We will also separate the error in the output from its main value and calculate the errors saparately.

In [8]:
```
x1=(1+xd)**3
x2=(1+xd)**2
```

```
x=np.vstack((x1,x2))
# print(x)
y=yd/73.04
y=y**2
# print(y)
dely=delyd/yd
dely+=0.0142387732749179
dely*=2
dely*=y
print(x,y,dely)
```

```
[[ 1.225043    1.331       1.404928    1.601613    1.63885834  1.7236836
   1.728       2.048383    2.097152    2.460375    2.47132621  2.744
   2.985984    3.241792    4.04247486  4.096       4.741632    5.177717
   5.64926254  6.59179688  6.644672    6.859       8.45226465 12.167
  14.348907   16.194277   20.796875   35.937     ]
 [ 1.1449      1.21        1.2544      1.3689      1.390041    1.437601
   1.44        1.6129      1.6384      1.8225      1.827904    1.96
   2.0736      2.1904      2.537649    2.56        2.8224      2.9929
   3.171961    3.515625    3.5344      3.61        4.149369    5.29
   5.9049      6.4009      7.5625     10.89      ]] [0.89243512 0.89243512 0.8821180
4 1.29132231 1.05438932 1.05438932
 0.99617016 1.1113732  1.47810199 1.09125827 1.29132231 1.69170908
 1.27890583 1.76368873 2.02742664 1.4482923  1.58655132 1.77461501
 2.06660306 2.92885922 1.51832062 2.56596185 4.44549281 5.29050385
 5.87252675 3.673961   7.64858698 9.40534017] [0.53242098 0.33582658 0.69892501 0.28
570329 0.14249461 0.17061166
 0.83732971 0.43578489 1.26052814 0.19126106 0.4724005  0.6536295
 0.27795679 2.30483792 0.5645928  0.24225821 0.32110306 0.30587697
 0.5312182  0.88005643 1.39285637 1.081912   1.28127019 1.22135778
 1.36164737 0.8394176  3.24695635 0.93965102]
```

here we will not be doing any preprocessing as missing values, Nan, etc values are not present in the dataset and scaling will change the parameters from their original values which is not desirable in this case

Now we shall use the Bayesian Ridge Regression model as the size of the dataset is small and also a real world dataset requiring greater precision

In [9]:
```
from sklearn.linear_model import BayesianRidge
model1=BayesianRidge()
model2d=BayesianRidge()
```

we fit the model separately into the main values and the error term of the output

In [10]:
```
model1.fit(x.T,y)
model2d.fit(x.T,dely)
```

Out[10]:
```
▾ BayesianRidge
BayesianRidge()
```

here is the final answer for the main values of the parameters/constants

In [13]:
```
print(model1.coef_,model1.intercept_)
```

```
[0.1646535  0.35964872] 0.2827349977938409
```

here is the final answer for the error values of the parameters/constants

In [14]:
```
print(model2d.coef_,model2d.intercept_)
```

```
[0.03131676 0.01680515] 0.5604636445724958
```

Now we try to test the constants for their values. Although separate training and testing samples should have been divided but due to small data, we will use the same data for testing

In [15]:
```
pred1=model1.predict(x.T)
pred2=model2d.predict(x.T)
```

now we calculate the mean error in the output of the testing phase

In [16]:
```
from sklearn.metrics import mean_absolute_error
error1=mean_absolute_error(y,pred1)
print(error1)
```

```
0.43539831225997006
```

In [17]:
```
error2=mean_absolute_error(dely,pred2)
print(error2)
```

```
0.4348458268086364
```

We see that the mean errors are very small in comparison to the values of H and H0 and z, hence we can conclude that the values of the constants are highly precise