You need to understand these metrics in order to determine whether regression models are accurate or misleading.

These first metrics are just a few of them. Other concepts, like **bias** and **overtraining** models, also yield misleading results and incorrect predictions.

([Learn more in Bias and Variance in Machine Learning.](https://www.bmc.com/blogs/bias-variance-machine-learning/" \t "_self))

What is variance?

In terms of linear regression, **variance** is a measure of how far observed values differ from the average of predicted values, i.e., their difference from the **predicted value mean**. The goal is to have a value that is low. What **low** means is quantified by the **r2 score** (explained below).

In the code below, this is **np.var(err)**, where **err** is an array of the differences between observed and predicted values and **np.var()** is the numpy array variance function.

What is r2 score?

The **r2 score**varies between 0 and 100%. It is closely related to the **MSE** (see below), but not the same. [Wikipedia](https://en.wikipedia.org/wiki/Coefficient_of_determination) defines **r2** as

*” …the proportion of the variance in the dependent variable that is predictable from the independent variable(s).”*

Another definition is “(total variance explained by model) / total variance.” So if it is 100%, the two variables are perfectly correlated, i.e., with no variance at all. A low value would show a low level of correlation, meaning a regression model that is not valid, but not in all cases.

Reading the code below, we do this calculation in three steps to make it easier to understand. **g** is the sum of the differences between the observed values and the predicted ones. **(ytest[i] – preds[i]) \*\*2**. **y** is each observed value **y[i]** minus the average of observed values **np.mean(ytest)**. And then the results are printed thus:

Copy

print ("total sum of squares", y)

print ("ẗotal sum of residuals ", g)

print ("r2 calculated", 1 - (g / y))

We can of course let scikit-learn to this with the r2\_score() method:

Copy

print("R2 score : %.2f" % r2\_score(ytest,preds))

What is mean square error (MSE)?

**Mean square error (MSE)**is the average of the square of the errors. The larger the number the larger the error. **Error** in this case means the difference between the observed values y1, y2, y3, … and the predicted ones pred(y1), pred(y2), pred(y3), … We square each difference (pred(yn) – yn)) \*\* 2 so that negative and positive values do not cancel each other out.

So here is the complete code:

import matplotlib.pyplot as plt

from sklearn import linear\_model

import numpy as np

from sklearn.metrics import mean\_squared\_error, r2\_score

reg = linear\_model.LinearRegression()

ar = np.array([[[1],[2],[3]], [[2.01],[4.03],[6.04]]])

y = ar[1,:]

x = ar[0,:]

reg.fit(x,y)

print('Coefficients: \n', reg.coef\_)

xTest = np.array([[4],[5],[6]])

ytest = np.array([[9],[8.5],[14]])

preds = reg.predict(xTest)

print("R2 score : %.2f" % r2\_score(ytest,preds))

print("Mean squared error: %.2f" % mean\_squared\_error(ytest,preds))

er = []

g = 0

for i in range(len(ytest)):

print( "actual=", ytest[i], " observed=", preds[i])

x = (ytest[i] - preds[i]) \*\*2

er.append(x)

g = g + x

x = 0

for i in range(len(er)):

x = x + er[i]

print ("MSE", x / len(er))

v = np.var(er)

print ("variance", v)

print ("average of errors ", np.mean(er))

m = np.mean(ytest)

print ("average of observed values", m)

y = 0

for i in range(len(ytest)):

y = y + ((ytest[i] - m) \*\* 2)

print ("total sum of squares", y)

print ("ẗotal sum of residuals ", g)

print ("r2 calculated", 1 - (g / y))

Results in:

Coefficients:

[[2.015]]

R2 score : 0.62

Mean squared error: 2.34

actual= [9.] observed= [8.05666667]

actual= [8.5] observed= [10.07166667]

actual= [14.] observed= [12.08666667]

MSE [2.34028611]

variance 1.2881398892129619

average of errors 2.3402861111111117

average of observed values 10.5

total sum of squares [18.5]

ẗotal sum of residuals [7.02085833]

r2 calculated [0.62049414]

You can see by looking at the data **np.array([[[1],[2],[3]], [[2.01],[4.03],[6.04]]])** that every dependent variable is roughly twice the independent variable. That is confirmed as the calculated coefficient **reg.coef\_** is 2.015.

There is no correct value for **MSE**. Simply put, the lower the value the better and 0 means the model is perfect. Since there is no correct answer, the MSE’s basic value is in selecting one prediction model over another.

Similarly, there is also no correct answer as to what **R2** should be. 100% means perfect correlation. Yet, there are models with a low R2 that are still good models.

Our take away message here is that you cannot look at these metrics in isolation in sizing up your model. You have to look at other metrics as well, plus understand the underlying math. We will get into all of this in subsequent blog posts.

SSE Sum of Squares

SSO Sum of residuals

R2 = 1 - SSE/SS0 1 - (N\* MSE) SSO / SSO = 1 – N \* MSE ¿???

SSE = N \* MSE SS0 = (N-1)\*VAR(Target)

VAR(T) = SS0/(N-1) = [SSE/(1-R2)]/(N-1)

= [N/N-1]\*MSE/(1-R2)

~ 5000/0.55 ~ 9091

# Mean Squared Error

print('MSE train: %.3f, test: %.3f' **%** (mean\_squared\_error(y\_train, y\_train\_pred),

                 mean\_squared\_error(y\_test, y\_test\_pred)))

# R-Squared

print('R^2 train: %.3f, test: %.3f' **%** (r2\_score(y\_train, y\_train\_pred),

                 r2\_score(y\_test, y\_test\_pred)))