Using SciPy T.ppf to get p-value

Asked 4 years, 5 months ago Active 9 months ago Viewed 13k times

2

Trying to test code creating P-value manually against SciPy. The Scipy Documentation isn't the best, which makes it tought to know for sure what to do.

I am getting the correct t-stat and P-value with SciPy, but I'm not able to replicate the correct p-value manually - A friend steered me to `scipy.stats.t.ppf` - but I'm not getting a p-value from it.

What is the correct way to do `scipy.stats.t.ppf()` ?

my version:

```
def t_test(sample, mu):
    mean = np.mean(sample)
    var = np.var(sample)
    sem = (var / len(sample)) ** .5
    t = abs(mu - mean)/sem
    df = len(sample) - 1
    p = scs.t.ppf(.95, df)
    return (t, p)
```

returns `(0.081500599630942958, 1.7291328115213671)`

scipy version:

`scs.ttest_1samp(sample, 4.123)`

returns `(statistic=0.079436958358141435, pvalue=0.93751577779749051)`

for testing, I'm using the following sample set and sample mu.

```
sample = [4.15848606,  3.86146363,  4.31545726,  3.3748772,
          4.67023082,  4.45950272,  3.85894915,  4.41089417,
          3.82360986,  3.79889443,  4.75884172,  3.27100914,
          4.08939402,  4.08904694,  5.62589842,  3.71445656,
          3.58463792,  4.42426443,  3.9671448 ,  4.39339124]

mu = 4.123
```

hypothesis-testing python scipy

Share Cite Edit Follow Flag

edited Dec 2 '20 at 13:28
reox
241 2 10

asked Apr 1 '17 at 18:10
be-ns
23 1 1 4

I am unable to reproduce these results. For instance, the mean of the sample is 4.132523, which

@whuber `mu` is a made up estimation of the population mean - while `mean` is the mean of the sample found here in `sample` I just copied and pasted the code into my terminal and it ran correctly - I edited the above to run my version within a function so you could run that easily – be-ns Apr 1 '17 at 19:34

1 Answer

Active | Oldest | Votes

2

To get the same results, change two things:

1. Change the estimation of the variance such that the divisor is N-1
2. Calculate the p-value using the cdf, that is the probability of getting a more extreme value, here using that the t-distribution is symmetric around zero. Note that the function you're comparing with does a two-sided test, and therefore, so do I.

I've marked the relevant lines with ###. The result is now the same as from the ttest_1samp function.

```
def t_test(sample, mu):
mean = np.mean(sample)
var = np.var(sample, ddof = 1) ###
sem = (var / len(sample)) ** .5
t = abs(mu - mean)/sem
df = len(sample) - 1
p = 2*(1-scs.t.cdf(t, df)) ###
return (t, p)
```

Share Cite Edit Follow Flag

answered Apr 1 '17 at 20:03
swmo
2,052 12 19

worked like a charm. Just adding the definition of `ddof` for any future visitors. “Delta Degrees of Freedom”: the divisor used in the calculation is N - `ddof` , where N represents the number of elements. By default `ddof` is zero." – be-ns Apr 1 '17 at 23:07

I think the p value should be `2*min(1-scs.t.cdf(t, df), scs.t.cdf(t, df))` - depending on which side of the t-distribution you are. – reox Dec 2 '20 at 12:47

@reox: That would be correct if 't' were the signed t-statistic, but note that in swmo's code, it is the absolute value. This ensures that we are always on the right side of the distribution. – Ruben van Bergen Dec 2 '20 at 13:48

@RubenvanBergen ah yes you are right, I missed the abs. – reox Dec 3 '20 at 14:18