Multinomial Goodness-of-Fit Test (ts_multinomial_gof)

Author: P. Stikker
Website: https://peterStatistics.com
YouTube: https://www.youtube.com/stikpet
Version: 0.1 (2023-01-12)

Introduction

The *ts_multinomial_gof* function (and *ts_multinomial_gof_arr* in VBA) performs an exact Goodness-of-Fit test. The test could be used to compare the proportions from different categories. The null-hypothesis is roughly that the proportions are all the same. If the p-value is too small (usually below 0.05) the assumption is rejected, indicating that at least two categories will have a different proportion in the population.

This document contains the details on how to use the functions, and formulas used in them.

1 About the Function

1.1 Input parameters:

- **data**
  The data to be used. Note for Python this needs to be a pandas data series.

- *Optional parameters*
  - **expCount** (default is none)
    A table with two columns. One with the categories and another with the expected counts. In Pandas this needs to be a dataframe.

  - **out** (default is "pvalue") – only applies to VBA *ts_multinomial_gof*
    Choice what to show as result. Either:
    - "pvalue": show the p-value (significance)
    - "pobs": the probability of the observed data
    - "ncomb": the number of possible combinations with equal sum as in the data

1.2 Output:

- **p-obs**. The probability of the observed data
- **n comb**. The number of combinations with an equal sum as the observed data
- **p-value**. The two-sided significance (p-value) of the test
- **test** used. Except for the non-array version in VBA (Excel) which will only show the requested p-value.

- The array version in VBA (*ts_multinomial_gof_arr*) requires **two rows** and **four columns**.

1.3 Dependencies

- **Excel**
  The function makes use of the Excel functions *COUNTIF*, *ROUNDDOWN*, *MULTINOMIAL*, and *FACT*.
  You can run the **ts_multinomial_gof_addHelp** macro so that the function will be available with some help in the 'User Defined' category in the functions overview.

- **Python**
  The following additional libraries will have to be installed/loaded:
  - *pandas as pd*
    the data input needs to be a pandas data series, and the output is also a pandas dataframe.

  - *itertools as it*
    the *product* function from Python's itertools library is needed

  - *numpy as np*
    the *arrange, asarray,* and *sum* functions are used

  - *scipy.stats*
    the *multinomial* function is used.

- **R**
  No other libraries required.

2 Examples

2.1 Excel

	A	B	C	D	E	F	G	H	I	J
1	Marital		Expected counts							
2	MARRIED		MARRIED	5						
3	DIVORCED		DIVORCED	5						
4	MARRIED		NEVER MARRIED	5						
5	SEPARATED		SEPARATED	5						
6	DIVORCED									
7	NEVER MARRIED									
8	DIVORCED		pvalue	0,388712	=ts_multinomial_gof($A$2:$A$20;;C8)					
9	DIVORCED		pobs	0,002541	=ts_multinomial_gof($A$2:$A$20;;C9)					
10	NEVER MARRIED		ncomb	1540	=ts_multinomial_gof($A$2:$A$20;;C10)					
11	MARRIED									
12	MARRIED		pvalue	0,388712	=ts_multinomial_gof($A$2:$A$20;C2:D5)					
13	MARRIED									
14	SEPARATED		p-obs	n comb.	p-value	test				
15	DIVORCED		0,002540683	1540	0,388712	one-sample multinomial exact goodness-of-fit test				
16	NEVER MARRIED									
17	NEVER MARRIED		C14:F15 ->	=ts_multinomial_gof_arr(A2:A20)						
18	DIVORCED									
19	DIVORCED									
20	MARRIED									
21										

2.2 Python

[2]:
```
data = pd.DataFrame(["MARRIED", "DIVORCED", "MARRIED", "SEPARATED", "DIVORCED",
                     "NEVER MARRIED", "DIVORCED", "DIVORCED", "NEVER MARRIED",
                     "MARRIED", "MARRIED", "MARRIED", "SEPARATED", "DIVORCED",
                     "NEVER MARRIED", "NEVER MARRIED", "DIVORCED", "DIVORCED", "MARRIED"],
                    columns=["marital"])

ts_multinomial_gof(data)
```

[2]:

	p obs.	n combs.	p-value	test
0	0.002541	1540	0.388712	one-sample multinomial exact goodness-of-fit test

[3]:
```
eCounts = pd.DataFrame({'category' : ["MARRIED", "DIVORCED", "NEVER MARRIED", "SEPARATED"], 'count' : [5,5,5,5]})

ts_multinomial_gof(data, eCounts)
```

[3]:

	p obs.	n combs.	p-value	test
0	0.002541	1540	0.388712	one-sample multinomial exact goodness-of-fit test

2.3 R

```
> data <- c("MARRIED", "DIVORCED", "MARRIED", "SEPARATED", "DIVORCED",
+           "NEVER MARRIED", "DIVORCED", "DIVORCED", "NEVER MARRIED",
+           "MARRIED", "MARRIED", "MARRIED", "SEPARATED", "DIVORCED",
+           "NEVER MARRIED", "NEVER MARRIED", "DIVORCED", "DIVORCED", "MARRIED")
> eCounts = data.frame(c("MARRIED", "DIVORCED", "NEVER MARRIED", "SEPARATED"),
+                      c(5,5,5,5))
> ts_multinomial_gof(data)
        pObs ncomb      pval                                                testUsed
1 0.002540683  1540 0.3887116 one-sample multinomial exact goodness-of-fit test
> ts_multinomial_gof(data, eCounts)
        pObs ncomb      pval                                                testUsed
1 0.002540683  1540 0.3887116 one-sample multinomial exact goodness-of-fit test
> |
```

3 Details of Calculations

The test can be done using the following four steps:

1. Determine the probability of the observed counts using the probability mass function of the multinomial distribution.

$$pmf(F_1, F_2 \dots F_k; p_1, p_2 \dots p_k) = \frac{n!}{\prod_{i=1}^{k} F_i!} \times \prod_{i=1}^{k} p_i^{F_i}$$

With:

$$n = \sum_{i=1}^{k} F_i$$

$$\sum_{i=1}^{k} p_i = 1$$

2. Determine all possible permutations with repetition that create a sum equal to the sample size over the k-categories.

3. Determine the probability of each of these permutations using the probability mass function of the multinomial distribution.

4. Sum all probabilities found in step 3 that are equal or less than the one found in step 1.

Step 2 is quite tricky. We could create all possible permutations with replacement. If our sample size is n and the number of categories is k, this gives $(n+1)^k$ permutations. The '+ 1' comes from the option of 0 to be included. Most of these permutations will not sum to the sample size, so they can be removed.

If the expected probability for each category is the same, we could use another approach. We could then create all possible combinations with replacement. This would give fewer results:

$$\binom{n+k}{k} = \frac{(n+k)!}{n!\,k!}$$

Again we can then remove the ones that don't sum to the sample size. Then perform step 3, but now multiply each by how many variations this can be arranged in. If for example we have 5 categories, and a total sample size of 20, one possible combination is [2, 2, 3, 3, 10]. This would be the same as [2, 3, 3, 2, 10], [2, 3, 10, 2, 3], etc. We could determine the count (frequency) of each unique score, so in the example 2 has a frequency of 2, 3 also and 10 only one. Now the first 2 we can arrange in:

$$\binom{5}{2} = \frac{5!}{(5-2)!\,2!} = 10$$

The 5 is our number of categories, the 2 the frequency. For the two 3's we now have 5 – 2 = 3 spots left, so those can only be arranged in:

$$\binom{3}{2} = \frac{3!}{(3-2)!\,2!} = 3$$

Combining these 3 with the 10 we had earlier gives $3 \times 10 = 30$ possibilities. The single 10 only can now go to one spot so that's it.

In general, if we have k categories, m different values and F_i is the i-th frequency of those values, sorted from high to low, we get:

$$\binom{k}{F_1}\prod_{i=2}^{m}\binom{k-\sum_{j=1}^{m-i+1}F_j}{F_j} = \binom{k}{F_1}\binom{k-F_1}{F_2}\binom{k-\sum_{j=1}^{2}F_j}{F_3}\dots\binom{k-\sum_{j=1}^{m-1}F_j}{F_k}$$

Where:

$$\binom{a}{b} = \frac{a!}{(a-b)!\,b!}$$

Symbols used:

- k the number of categories
- F_i the (absolute) frequency of category i
- E_i the expected frequency of category i
- n the sample size, i.e. the sum of all frequencies