Pandas - Compute z-score for all columns

Asked 7 years, 1 month ago Active 6 months ago Viewed 153k times

62

28

I have a dataframe containing a single column of IDs and all other columns are numerical values for which I want to compute z-scores. Here's a subsection of it:

```
ID      Age    BMI    Risk Factor
PT 6    48     19.3    4
PT 8    43     20.9    NaN
PT 2    39     18.1    3
PT 9    41     19.5    NaN
```

Some of my columns contain NaN values which I do not want to include into the z-score calculations so I intend to use a solution offered to this question: [how to zscore normalize pandas column with nans?]

```
df['zscore'] = (df.a - df.a.mean())/df.a.std(ddof=0)
```

I'm interested in applying this solution to all of my columns except the ID column to produce a new dataframe which I can save as an Excel file using

```
df2.to_excel("Z-Scores.xlsx")
```

So basically; how can I compute z-scores for each column (ignoring NaN values) and push everything into a new dataframe?

SIDENOTE: there is a concept in pandas called "indexing" which intimidates me because I do not understand it well. If indexing is a crucial part of solving this problem, please dumb down your explanation of indexing.

python pandas indexing statistics Edit tags

Share Edit Follow Close Flag

edited May 23 '17 at 12:18
Community ♦
1 1

asked Jul 15 '14 at 15:18
Slavatron
1,986 5 25 33

What don't you understand about indexing? – EdChum Jul 15 '14 at 15:43

I think its something like the concept of a primary key in SQL databases where you set an identifier that will let you refer to values within a row; but I'm not even sure about that. I don't really understand when I would want to set an index either. – Slavatron Jul 15 '14 at 15:47

The concept of the index is no different to an SQL table but unlike a clustered index say, a multi-index will have different levels, think grouping by say gender, then age, then weight for example. The other concept is label indexing, your index can be anything, strings, dates, integers etc.. You can

index using label indexing or by integer value: pandas.pydata.org/pandas-docs/stable/... – EdChum Jul 15 '14 at 15:51

Interesting; that sounds like it could be really useful once I get the hang of it. I'm still intimidated by all the unfamiliar jargon in the documentation but it definitely feels a little more accessible now. Thanks again. – Slavatron Jul 15 '14 at 16:13

(Probably much too late to help you.) One big advantage of `df.set_index(['ID'])` is that all of the hassles you have with treating that column separately now go away. – Teepeemm Feb 14 '20 at 21:26

8 Answers

Active | Oldest | Votes

0

To calculate a z-score for an entire column quickly, do as follows:

```
from scipy.stats import zscore
import pandas as pd

df = pd.DataFrame({'num_1': [1,2,3,4,5,6,7,8,9,3,4,6,5,7,3,2,9]})
df['num_1_zscore'] = zscore(df['num_1'])

display(df)
```

Share Edit Follow Flag

answered Feb 26 at 22:47
GbG
176 2 8

7

for Z score, we can stick to documentation instead of using 'apply' function

```
from scipy.stats import zscore
df_zscore = zscore(cols as array, axis=1)
```

Share Edit Follow Flag

edited Nov 26 '20 at 16:52
andrea_grioni
165 1 9

answered Sep 30 '18 at 20:29
ibozkurt79
195 2 7

Which package would the zscore be in ? – whisperer Feb 20 '19 at 1:33

I just fixed it. It is scipy library – ibozkurt79 Feb 21 '19 at 5:49

6

Here's other way of getting Zscore using custom function:

```
In [6]: import pandas as pd; import numpy as np

In [7]: np.random.seed(0) # Fixes the random seed

In [8]: df = pd.DataFrame(np.random.randn(5,3), columns=["randomA",
```

```
"randomB","randomC"])

In [9]: df # watch output of dataframe
Out[9]:
    randomA   randomB   randomC
0  1.764052  0.400157  0.978738
1  2.240893  1.867558 -0.977278
2  0.950088 -0.151357 -0.103219
3  0.410599  0.144044  1.454274
4  0.761038  0.121675  0.443863

## Create custom function to compute Zscore
In [10]: def z_score(df):
   ....:         df.columns = [x + "_zscore" for x in df.columns.tolist()]
   ....:         return ((df - df.mean())/df.std(ddof=0))
   ....:

## make sure you filter or select columns of interest before passing dataframe to
function
In [11]: z_score(df) # compute Zscore
Out[11]:
   randomA_zscore  randomB_zscore  randomC_zscore
0        0.798350       -0.106335        0.731041
1        1.505002        1.939828       -1.577295
2       -0.407899       -0.875374       -0.545799
3       -1.207392       -0.463464        1.292230
4       -0.688061       -0.494655        0.099824
```

Result reproduced using scipy.stats zscore

```
In [12]: from scipy.stats import zscore

In [13]: df.apply(zscore) # (Credit: Manuel)
Out[13]:
    randomA   randomB   randomC
0  0.798350 -0.106335  0.731041
1  1.505002  1.939828 -1.577295
2 -0.407899 -0.875374 -0.545799
3 -1.207392 -0.463464  1.292230
4 -0.688061 -0.494655  0.099824
```

Share Edit Follow Flag

edited Mar 6 '18 at 15:54

answered Mar 6 '18 at 15:42

Surya

8,834 3 51 34

1

When we are dealing with time-series, calculating z-scores (or anomalies - not the same thing, but you can adapt this code easily) is a bit more complicated. For example, you have 10 years of temperature data measured weekly. To calculate z-scores for the whole time-series, you have to know the means and standard deviations for each day of the year. So, let's get started:

Assume you have a pandas DataFrame. First of all, you need a DateTime index. If you don't have it yet, but luckily you do have a column with dates, just make it as your index. Pandas will try to guess the date format. The goal here is to have DateTimeIndex. You can check it out by trying:

```
type(df.index)
```

If you don't have one, let's make it.

```
df.index = pd.DatetimeIndex(df[datecolumn])
df = df.drop(datecolumn,axis=1)
```

Next step is to calculate mean and standard deviation for each group of days. For this, we use the groupby method.

```
mean = pd.groupby(df,by=[df.index.dayofyear]).aggregate(np.nanmean)
std = pd.groupby(df,by=[df.index.dayofyear]).aggregate(np.nanstd)
```

Finally, we loop through all the dates, performing the calculation (value - mean)/stddev; however, as mentioned, for time-series this is not so straightforward.

```
df2 = df.copy() #keep a copy for future comparisons
for y in np.unique(df.index.year):
    for d in np.unique(df.index.dayofyear):
        df2[(df.index.year==y) & (df.index.dayofyear==d)] = (df[(df.index.year==y) &
(df.index.dayofyear==d)]- mean.ix[d])/std.ix[d]
        df2.index.name = 'date' #this is just to look nicer

df2 #this is your z-score dataset.
```

The logic inside the for loops is: for a given year we have to match each dayofyear to its mean and stdev. We run this for all the years in your time-series.

Share Edit Follow Flag

edited Apr 12 '17 at 5:16

answered Apr 11 '17 at 8:12

Deninhos

579 5 8

80

Using Scipy's zscore function:

```
df = pd.DataFrame(np.random.randint(100, 200, size=(5, 3)), columns=['A', 'B', 'C'])
df
```

	A	B	C
0	163	163	159
1	120	153	181
2	130	199	108
3	108	188	157
4	109	171	119

```
from scipy.stats import zscore
df.apply(zscore)
```

	A	B	C
0	1.83447	-0.708023	0.523362
1	-0.297482	-1.30804	1.3342
2	0.198321	1.45205	-1.35632
3	-0.892446	0.792025	0.449649
4	-0.842866	-0.228007	-0.950897

If not all the columns of your data frame are numeric, then you can apply the Z-score function only to the numeric columns using the `select_dtypes` function:

```
# Note that `select_dtypes` returns a data frame. We are selecting only the columns
numeric_cols = df.select_dtypes(include=[np.number]).columns
df[numeric_cols].apply(zscore)
```

	A	B	C
0	1.83447	-0.708023	0.523362
1	-0.297482	-1.30804	1.3342
2	0.198321	1.45205	-1.35632
3	-0.892446	0.792025	0.449649
4	-0.842866	-0.228007	-0.950897

Share Edit Follow Flag

edited Feb 14 '17 at 9:51

answered Jan 18 '17 at 7:30

Manuel
1,926 1 15 26

how to `apply` in place rather than returning a new copy? – CKM Jan 3 '18 at 5:55

1 @chandresh, `apply` does not have an `inplace` parameter, so you cannot replace the column data with the function result.You should check this question: stackoverflow.com/questions/45570984/... – Manuel Jan 4 '18 at 10:37

21

If you want to calculate the zscore for all of the columns, you can just use the following:

```
df_zscore = (df - df.mean())/df.std()
```

Share Edit Follow Flag

answered Feb 1 '17 at 14:09

Joe Bathelt
4,033 2 12 14

1 Oddly, for me anyway, this calculation of score gives slightly different results than "from scipy.stats import zscore; df.apply(zscore)". Anyone know why? – pitosalas Apr 28 '18 at 22:14

2 @pitosalas: Could be different default `ddof` for the `std` function – ascripter Jun 21 '18 at 13:10

3 @pitosalas: @ascripter, you are correct. Passing `df.std(ddof=0)` produces the same result as `df.apply(scipy.stats.zscore)` – roob Sep 26 '18 at 21:24

pandas probably won't be happy about the non-numerical ID column, but it should be an index anyway. I like that this one operates on the entire dataframe, instead of column by column like the other answers. – Teepeemm Feb 14 '20 at 21:32

4

The almost one-liner solution:

```
df2 = (df.ix[:,1:] - df.ix[:,1:].mean()) / df.ix[:,1:].std()
df2['ID'] = df['ID']
```

Share Edit Follow Flag

answered Sep 22 '16 at 19:45

Josh Chartier
41 1

9 almost one-liner aka two-liner :) – Wtower Dec 26 '16 at 22:30

2 One liner `df2 = df2.assign(ID=(df.ix[:,1:] - df.ix[:,1:].mean()) / df.ix[:,1:].std())` – Little Bobby Tables Nov 6 '17 at 9:18

85

Build a list from the columns and remove the column you don't want to calculate the Z score for:

```
In [66]:
cols = list(df.columns)
cols.remove('ID')
df[cols]

Out[66]:
   Age  BMI  Risk  Factor
0    6   48  19.3       4
1    8   43  20.9     NaN
2    2   39  18.1       3
3    9   41  19.5     NaN
In [68]:
# now iterate over the remaining columns and create a new zscore column
for col in cols:
    col_zscore = col + '_zscore'
    df[col_zscore] = (df[col] - df[col].mean())/df[col].std(ddof=0)
df
Out[68]:
   ID  Age  BMI  Risk  Factor  Age_zscore  BMI_zscore  Risk_zscore  \
0  PT    6   48  19.3       4   -0.093250    1.569614    -0.150946
1  PT    8   43  20.9     NaN    0.652753    0.074744     1.459148
2  PT    2   39  18.1       3   -1.585258   -1.121153    -1.358517
3  PT    9   41  19.5     NaN    1.025755   -0.523205     0.050315

   Factor_zscore
0              1
1            NaN
2             -1
3            NaN
```

Share Edit Follow Flag

answered Jul 15 '14 at 15:29

EdChum
308k 177 693 500

1 is there a way to do this without the for loop? (assime you don't need to remove one of the columns...) – Alex Lenail Jul 24 '17 at 13:55

2 @AlexLenail looking at this again 3 years later you could just define a `func` and call this func using `apply` as this is syntactic sugar for a `for` loop – EdChum Jul 24 '17 at 18:08

Could you provide some explanation for the `ddof` parameter? We can either compute sample standard deviation (`ddof=1`) or population standard deviation (`ddof=0`). I would assume that in most of the cases we are looking at samples, thus we should use the `ddof=1` – Ryszard Cetnarski Mar 20 '18 at 13:03

3 @RyszardCetnarski see an explanation statsdirect.com/help/basics/degrees_freedom.htm and stats.stackexchange.com/questions/58230/... it depends on your use case – EdChum Mar 20 '18 at 13:15

1 @Khashir No, that is incorrect, what should happen is that over time the better answer will accumulate more votes (I upvoted that answer myself), and that will be viewed as the best answer. The accepted answer is not necessarily the best answer. Also one should not start incrementally updating their answer, effectively cannibalising another person's answer. This is bad form and annoys other's, people are free to vote however they like, the most votes has never meant it's always the best, one only needs to look around this planet to see that isn't patently true – EdChum Jul 28 '19 at 19:52