StdFunction

January 22, 2024

1 Childhood Obesity

1.1 Introduction.

According to the WHO website, "Overweight and obesity are defined as abnormal or excessive fat accumulation that presents a risk to health. A body mass index (BMI) over 25 is considered overweight, and over 30 is obese. The issue has grown to epidemic proportions, with over 4 million people dying each year as a result of being overweight or obese in 2017 according to the global burden of disease." (https://www.who.int/health-topics/obesity#tab=tab_1). This is particularly concerning in the case of children, where the prevalence of obesity has increased worldwide over the past 50 years. (Abarca-Gomez et al.)

In order to understand this phenomena we will take a look at some representative data compiled by the CDC and NHANES (National Health and Nutrition Estimation Survey). We will be using some standard statistical techniques to analyze the overall BMI trends in the population namely: the mean and the standard deviation. We will compute these values ourselves although the data could easily be obtained through other means as a way to learn about control loops and functions.

To begin with

1.1.1 Setup

Import some python libraries needed for this module

[1]:
```
import numpy as np
import pandas as pd
import matplotlib as plot
```

1.1.2 Data

Let's first suppose we have a list of weights from a set of people in a study. For now we'll just randomly generate such a data set. In step two we will read it in from some known data source (CDC/NHANES).

For the time being however, let's just generate a random sample of numbers taken from a Gaussian distribution with mean 150 and standard deviation of 5.

[134]:
```
num_samples = 10000
```

[135]:
```
x = [np.random.normal(150,5) for _ in range(num_samples)]
```

Let's check to see what the values look like

[136]:
```
truemean = np.mean(x)
truemean
```

[136]: 149.9707803247167

[137]:
```
truestd = np.std(x)
truestd
```

[137]: 5.0164199270455985

[153]:
```
# According to the python documentation the std is computed from the formula:
# In general both the mean and standard deviation are divided by N =␣
 ↪num_samples unless
# the degree of freedom (dof) variable is set to some value different than the
# default of dof=0.

xmean = np.sum(x)/num_samples
std = np.sqrt(np.sum(np.abs(x - xmean)**2)/num_samples)

std
```

[153]: 5.0164199270455985

2 Two different algorithms for computing standard deviation

The standard deviation of a sample data set is helpful in understanding the spread of the data from the mean (average). Like the mean, it is computed through standard formulas although there are variations on how one can compute it.

Here we will look at two different versions that are mathematically equivalent, and yet will sometimes give different results.

2.1 Algorithm 1

Let's now try what is known as the two-pass version for computing the standard deviation. The first step is use to compute the mean. In the second control loop, the standard deviation is computed as sum of squares of the original data after normalizing by subtracting the mean.

The equations we will use are:

$$\bar{x} = \frac{1}{m}\sum_{i}^{m} x_i$$

$$s_2^2 = \frac{1}{m-1}\sum_{i}^{m}(x_i - \bar{x})^2$$

N.B. These are the standard equations used in most statistics textbooks.

[158]:
```
#
# Note that in order to be consistent with python we will use m instead of m-1␣
 ↪in the
# denominator for the standard deviation formula

m = len(x)

sumx = 0.0
for i in range(0, m) :
    sumx += x[i]
meanx = sumx/m

sumx2 = 0.0
for i in range(0, m) :
    sumx2 += (x[i] - meanx)**2
var = sumx2/(m)
stdx = np.sqrt(var)
print ("the mean of the %d numbers is %20.16f and the std is %20.16f" % (m,␣
 ↪meanx, stdx))
```

```
the mean of the 10000 numbers is 149.9707803247164577 and the std is
5.0164199270455967
```

2.2 Algorithm 2

It is not hard to show that the above formula for the standard deviation can be written as:

$$\bar{x} = \frac{1}{m}\sum_{i}^{m} x_i$$

$$s_2^2 = \frac{1}{m-1}\left(\sum_{i}^{m} {x_i}^2\right) - \bar{x}^2$$

Computationally, this seems like a good choice, because we can write code for this formulation that computes everything in one for loop, which increases its computational efficiency.

[160]:
```
#
# Note that in order to be consistent with python we will use m instead of m-1␣
 ↪in the
# denominator for the standard deviation formula

m = len(x)

sumx = 0.0
sumx2 = 0.0
for i in range(0, m) :
    sumx += x[i]
    sumx2 += x[i]**2
meanx = sumx/m
```

```
#var   = (sumx2/(m-1)) - meanx**2
var = (sumx2 - meanx*sumx)/(m)
stdx  = np.sqrt(var)

print ("the mean of the %d numbers is %20.16f and the std is %20.16f" % (m,␣
 ↪meanx, stdx))
```

```
the mean of the 10000 numbers is 149.9707803247164577 and the std is
5.0164199270594567
```

Notice that while the mean of the numbers appears to be the same up to at least 15 digits, the standard deviation is different at around the 12th digit. As the number of samples increases, we can expect that this difference might become even larger.

2.2.1 Part 2: NHANES Data

Now let's try to compute the mean and standard deviation for a real-world data set. The file we will use is a slightly cleaned up version of a data set downloaded from the NHANES (National Health and Nutrition Examination Survey) web site: https://www.cdc.gov/nchs/nhanes/index.htm

The particular data set chosen was the 2017- March 2020 Pre-Pandemic Examination Data - Continuous NHANES for Body Measures, P_BMX.xpt (https://wwwn.cdc.gov/Nchs/Nhanes/2017-2018/P_BMX.XPT). The main changes to the csv file here is that only 5 variables were chosen from the original data set and all of the "NA"s were removed to make it easier to compute the statistics.

Note: We can supply the short python file that created the csv file from the xpt file on the NHANES web pages if needed.

[161]:
```
# Read in the data set and take a quick look at it.

import pandas as pd
bmidata = pd.read_csv('NHANESChildObesity.csv')
print(bmidata)
```

```
      Unnamed: 0  BMDBMIC  BMXBMI  BMXHT  BMXWAIST  BMXWT
0              2        2    17.6  154.7      63.8   42.2
1              3        2    15.0   89.3      41.2   12.0
2              6        4    30.9  156.0      91.4   75.3
3             13        2    18.6  162.0      67.6   48.7
4             14        2    16.3  112.0      59.2   20.5
…            …       …      …     …        …     …
4485       14270        4    21.2   94.5      53.8   18.9
4486       14276        2    15.8  117.2      57.7   21.7
4487       14285        2    16.0   98.8      48.7   15.6
4488       14297        3    17.5   93.7      48.4   15.4
4489       14298        2    15.1  123.3      57.5   22.9

[4490 rows x 6 columns]
```

[162]:
```
# Let's select only the BMI data for the computation for this example.
# We'll grab all of the rows using the ":" notation, and we select the BMXBMI␣
 ↪column through the second parameter.
# Note: In general, we wouldn't go to this much trouble and instead just␣
 ↪reference the correct column,
# but in the interests of clarity and exposition we'll pull out the data from␣
 ↪the original data set.

x = bmidata.loc[:,['BMXBMI']]
x = x.to_numpy()
m = len(x)
print ("the number of BMI measurements is %d." % (m))
```

```
the number of BMI measurements is 4490.
```

[166]:
```
# Now let's compute the mean and std. deviation for the BMI using Algorithm 1.
m = len(x)

sumx = 0.0
for i in range(0, m) :
    sumx += x[i]
bmimean = sumx/m

sumx2 = 0.0
for i in range(0, m) :
    sumx2 += (x[i] - bmimean)**2
var = sumx2/m
bmistd = np.sqrt(var)

print ("the mean of the BMI is %15.12f and the std dev is %15.12f" % (bmimean,␣
 ↪bmistd))
```

```
the mean of the BMI is 20.808953229399 and the std dev is  6.260679785489
```

[167]:
```
# Now let's compute the mean and std. deviation for the BMI using Algorithm 2.
m = len(x)

sumx = 0.0
sumx2 = 0.0
for i in range(0, m) :
    sumx += x[i]
    sumx2 += x[i]**2

bmimean2 = sumx/m
bmivar2 = (sumx2 - bmimean2*sumx)/m
bmistd2  = np.sqrt(var)
```

```
print ("the mean of the BMI is %15.12f and the std dev is %15.12f" % (bmimean2,␣
 ↪bmistd2))
```

```
the mean of the BMI is 20.808953229399 and the std dev is  6.260679785489
```

2.2.2 Summary

It appears that both versions of the algorithm will return similar values for the mean and the standard deviation. However, this is not always the case, and care should be taken when using the formulas.

2.2.3 To Do

Other things to try: 1. Convert either/both algorithms to a function, callable with just the variable containing the data as an input variable. 2. Convert to a function that computes the mean and std dev for all of the variables/columns in a data frame 3. Create a data set for which the two algorithms do in fact return different values 4. Allow the inclusion of NAs in the data set when calling the function.

2.2.4 References

1. Worldwide trends in body-mass index, underweight, overweight, and obesity from 1975 to 2016: a pooled analysis of 2416 population-based measurement studies in $128 \cdot 9$ million children, adolescents, and adults. Lancet. 2017 Dec 16;390(10113):2627-2642. doi: 10.1016/S0140-6736(17)32129-3. Epub 2017 Oct 10. PMID: 29029897; PMCID: PMC5735219.
2. Computing standard deviations: accuracy, Chan and Lewis, Communications of the ACM, Vol. 22, No. 9, September 1979.

NSF Dubois Project

Module C: Vivian Carter, Deb Nolan, Roummel Marcia, Juan C. Meza Date: September 11, 2023