Iteration

January 24, 2024

1 Iteration Module: Childhood Obesity

1.1 Summary

According to the WHO website, "Overweight and obesity are defined as abnormal or excessive fat accumulation that presents a risk to health … 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 issue 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.)

One popular measure of obesity is the body mass index (BMI), which can be calculated from a simple formula given a person's height and weight. In particular, the BMI is given by:

$$BMI = \frac{weight}{height^2},$$

where weight is measured in kilograms and height is in meters.

Generally, a BMI over 25 is considered overweight, and over 30 is obese.

In order to understand this phenomena we will take a look at data compiled by the CDC and NHANES (National Health and Nutrition Estimation Survey). We will be using some standard statistical techniques to analyze BMI in the population namely: the mean and the standard deviation.

Although the data could easily be obtained through other means, we will compute these values ourselves as a way to learn about iterations, control, functions, and algorithms.

1.2 Module Goals

In this module we will cover the following concepts:

Area	Topics Covered
CS	Iterations
Social	Reliability and validity, fairness, precision
Ethics	Fairness, Privacy, Equity vs. Equality
Data Sets	Prepping and cleaning data

1.2.1 Research Questions

Some questions that naturally arise in the context of this data include: 1. Are there any differences in obesity among different age groups? 2. Are there any differences between different ethnic groups? 3. Are there any differences in gender, i.e. between boys and girls? 4. How do socio-economic differenes affect obesity? 5. How does weight vary over a six-month period? 6. How does weight vary between men based on height? 7. How does weight vary between women based on height? 8. How does weight vary among peri-menopausal women and pre-menopausal women?

Exercise:

Can you think of other research questions that you might want to consider. Use your own experiences and suggest 1-2 other questions that interest you.

1.2.2 Ethics Questions

1. What should the role of public health be (if any) in treating childhood obesity?
2. What ethical obligations do parents have to protect their children?
3. What is the right balance between individual rights and liberty-limiting public policies to promote the public health?

Discussion:

This is a good point to break the class down into smaller groups and discuss these (and other) ethics questions.

1.3 Setup

Let's now proceed in developing some basic codes that will help us start to answer some of the questions we posed.

In order to do the necessary calculations, we will have need of several python libraries. The following three in particular will be used in almost all of our notebooks. Running the cell below will allow us to access some of the necessary algorithms that will make our coding easier.

[9]:
```
import numpy as np
import pandas as pd
import matplotlib as plot
```

1.3.1 Data

In any analysis, we need to start with some data, for example let's suppose we have a list of weights from a set of people in a study.

In step 2 we will read data from a national survey administered by the CDC. For the time being however, let's just generate a random set of weights so we can more easily work with (and understand) our results.

For this smalle exercise, we will generate weigths taken from a Gaussian distribution with mean 120 (pounds) and standard deviation of 5.

[10]:
```
num_samples = 100
x = [np.random.normal(120,5) for _ in range(num_samples)]
```

Python allows us to easily compute the mean and standard deviation through some functions provided in the numpy library (more on functions in a subsequent module). Let's check to see what the values look like.

[11]:
```
truemean = np.mean(x)
truemean
```

[11]: 119.53996374481854

[12]:
```
truestd = np.std(x)
truestd
```

[12]: 5.180076626232874

As we can see, the values from the random sample of 100 generated values is close to what we specified, although not exact. This is the nature of a random sample after all.

Tip: If you like, you may want to run the cells above several times and see for yourself how the numbers change.

In the next step, we will demonstrate how to compute these values using a concept known as **iteration.**

2 Iterations: 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.

In order to compute these quantitites, we first introduce the concept of an ***iteration*** in python.

Let's first consider the computation of the mean.

[13]:
```
#
# 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

print ("the mean of the %d numbers is %20.16f" % (m, meanx))
```

```
the mean of the 100 numbers is 119.5399637448185644
```

Here we are using a construction known as a loop, specifically a ***"for loop"*** to compute the mean. The for loop specifies a variable known as an iterator, in this case the variable "i". The variable is given values in the ***"range"*** between 0 and m. The loop then proceeds to execute the lines inside the for loop, in each case taking on a new value.

In our example, we are simply adding the entry x[i] to a running sum (sumx). Once the for loop has evaluated that piece of code for each value within the range, the loop terminates.

At the end of the loop, the variable sumx contains the sum of all the entries of x[i], for i=0, 1, ..., m-1.

We then proceed to divide by the number of terms in the sum, which gives us the mean as described by the equation above.

We can do a similar exercise to compute the standard deviation given by the second equation. Note that we needed to compute the mean first in order to compute the quantity in this second loop.

[14]:
```
#
# 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)

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

```
the std of the 100 numbers is   5.1800766262328750
```

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.

[15]:
```
#
# 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 100 numbers is 119.5399637448185644 and the std is
5.1800766262324283
```

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.

[16]:
```
# Read in the data set and take a quick look at it.

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

```
       Unnamed: 0    SEQN  BMDBMIC  BMXBMI  BMXHT  BMXWAIST  BMXWT
0               1  109263      NaN     NaN    NaN       NaN    NaN
1               2  109264      2.0    17.6  154.7      63.8   42.2
2               3  109265      2.0    15.0   89.3      41.2   12.0
3               4  109266      NaN    37.8  160.2     117.9   97.1
4               5  109269      NaN     NaN    NaN       NaN   13.6
…             …       …       …      …      …         …     …
14295       14296  124818      NaN    38.2  168.7     114.7  108.8
14296       14297  124819      3.0    17.5   93.7      48.4   15.4
14297       14298  124820      2.0    15.1  123.3      57.5   22.9
14298       14299  124821      NaN    25.5  176.4      97.1   79.5
14299       14300  124822      NaN    21.3  167.5      86.9   59.7

[14300 rows x 7 columns]
```

[17]:
```
# 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 14300.
```

[18]:
```
# 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             nan and the std dev is             nan
```

[19]:
```
# 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             nan and the std dev is             nan
```

2.3 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.

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.

2.3.1 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: January 22, 2024