01_24_doing_computations

Unknown Author

January 28, 2014

In [1]:
```
# For the sqrt function
from math import sqrt
```

In this class we shall be concentrating on what we have learnt to write some programs. This begins the math part of your course. To start with let us do some statistical computations. We shall input the data as lists of floats. We shall see later how to read such data from files.

1 The data

In [2]:
```
data_mid = [23, 45, 83, 90, 12, 87, 67, 69, 74, 36, 43, 69, 66, 70]
data_end = [45, 44, 95, 87, 24, 100, 45, 70, 66, 32, 50, 55, 80, 81]
```

2 Mean

First let us find the means to see if the class performance had any improvement after mid-sem. Let us write it as a function. Recall that mean of a collection of numbers, $x_i, i = 1, \ldots, n$ is given by the formula

$$\text{mean} = \frac{\sum_{i=1}^{n} x_i}{n}. \tag{1}$$

In [3]:
```
def find_mean(lst) :
    """Given a list as input, this function computes the mean."""
    # Store the sum in a variable
    sum = 0.0                                  # Quiz : Why 0.0 and not just 0?
    # loop over all the numbers in the list
    for no in lst :
        sum += no

    # The number of elements in the list
    no_of_entries = len(lst)

    # mean by definition is
    mean = sum/no_of_entries

    # now, we return the value
    return mean
```

Let us try this out

In [4]:
```
print """The mean for mid-sem is %6.2f,
while that for the end-sem is %6.2f.""" % (find_mean(data_mid), find_mean(data_end))
```

```
The mean for mid-sem is  59.57,
while that for the end-sem is  62.43.
```

3 Standard deviation

Looks like there is a general increase. But increase of mean improvement. It might have been that some substantial number of people actually did slightly worse, but a few people did exceptionally well in the end sem. One measure to check the spread is call standard deviation. The formula, for x_i as above, is

$$\text{standard deviation} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2} \tag{2}$$

where μ is the mean.Let us try to code this.

In [5]:
```
def find_sd(lst) :
    """Given a list, this function computes the (biased) standard deviation."""
    # This one depends on the function find_mean (). Let us find the mean.
    mu = find_mean(lst)

    # Rest of the code is similar to mean. Introduce a variable to store the sum of sq
    sum_sq_dev = 0.0

    # Loop over the data to find this sum of squared deviations from the mean.
    for no in lst :
        sum_sq_dev += (no - mu) ** 2

    # To compute s.d. we also need to know the number of data points:
    n = len(lst)

    # Now to finish computing sd, we just need to divide by n and take square root.
    sd = sqrt(sum_sq_dev / n)

    # Don't ever forget to return your hard work.
    return sd
```

Let us try it out.

In [6]:
```
print """The standard deviations for the two exams are
    %6.2f                    ,                    %6.2f
respectively.""" % (find_sd(data_mid), find_sd(data_end))
```

```
The standard deviations for the two exams are
     23.09                    ,                    22.93
respectively.
```

This code is very intuitive. However we are running through the data twice, once for computing mean and once for standard deviation. To save a bit of work, one can do a bit of simplification : Note that $n\mu = \sum_i x_i$. Therefore,

$$\sum_i(x_i-\mu)^2 = \sum_i(x_i^2-2\mu x_i+\mu^2) = \sum_i x_i^2-2\mu\sum_i x_i+n\mu^2 = \sum_i x_i^2-2n\mu^2+n\mu^2 = \sum_i x_i^2-n\mu^2 = \sum_i x_i^2-\frac{1}{n}\left(\sum_i x_i\right)^2. \tag{3}$$

To make use of this we use one loop to compute both sum of the numbers and sum of their squares. Then use these computations to compute the sd.

In [7]:
```
def find_sd2(lst) :
    """Given a list, this function computes the (biased) standard deviation more effic

    # We need a variable to store the sum, and another one to store the sum of squares
    sum = 0.0
    sum_sq = 0.0

    # Loop over the data to find the sum and the sum of squares.
    for no in lst :
        sum += no
        sum_sq += no ** 2

    # To compute s.d., and the sum of squares of deviations,  we also need to know the
    # of data points:
    n = len(lst)

    # Using this compute the sum of squares of deviations
    sum_sq_dev = sum_sq - sum**2 / n


    # Now to finish computing sd, we just need to divide by n and take square root.
    sd = sqrt(sum_sq_dev / n)

    # Don't ever forget to return your hard work.
    return sd
```

Let us try to use it :

In [8]:
```
print """The standard deviations (using the second function) for the two exams are
    %6.2f                    ,                   %6.2f
respectively.""" % (find_sd2(data_mid), find_sd2(data_end))
```

```
The standard deviations (using the second function) for the two exams
are
     23.09                    ,                   22.93
respectively.
```

4 Correlation

Things seem to be better. But have the people who scored high in the first exam, score high in the second too? To see that there is a measure called correlation. The formula is used on two sets of data and the formula spills out a value between -1 and 1. The formula is

$$\text{Correlation} = \frac{\text{Covariance}}{(\text{s.d. of } X)(\text{s.d. of } Y)} \tag{4}$$

where

$$\text{Covariance} = \frac{1}{n}\sum_{i=1}^{n}(x_i - m_x)(y_i - m_y) \tag{5}$$

x_i, y_i being the data given of size n, m_x and m_y being the of x and y resp and n is the number of data pairs.As before we we try to simplify the formula so that we can compute using just one loop to compute.

$$\sum_i (x_i - m_x)(y_i - m_y) = \sum_i x_i y_i - m_x \sum_i y_i - m_y \sum_i x_i + n m_x m_y = \sum_i x_i y_i - n m_x m_y - n m_x m_y + n m_x m_y \tag{6}$$

$$= \sum_i x_i y_i - \frac{1}{n}\left(\sum_i x_i\right)\left(\sum_i y_i\right). \quad (7)$$

It makes sense to write the correlation function for a list of pairs. We can use that on our data using zip.

In [9]:

```
def my_corr(lst_of_2_tuples) :
    """Given a list of 2-tuples, this functions computes the correlation between the f
    second entries."""

    # As before we use a huge bunch of variables.
    sumx = 0.0
    sumy = 0.0
    sumxy = 0.0
    sumx2 = 0.0
    sumy2 = 0.0

    # Now loop
    for pair in lst_of_2_tuples :
        # To make reading easier, set
        x = pair[0]
        y = pair[1]

        # Now accumulate
        sumx += x
        sumy += x
        sumxy += x * y
        sumx2 += x * x
        sumy2 += y * y

    # Now we got all the ingredients to compute covariance and s.d. except n :
    n = len(lst_of_2_tuples)

    # Now compute
    covariance = sumxy - sumx * sumy / n
    sdx = sqrt(sumx2 - sumx**2/n)
    sdy = sqrt(sumy2 - sumy**2/n)

    correlation = covariance / (sdx * sdy)

    return correlation
```

Let us try this

In [10]:

```
print "Correlation is %6.4f." % (my_corr(zip(data_mid, data_end)))
```

```
Correlation is 0.9221.
```

Some general remarks

Soon we shall learn how to read a data from a file (and if time permits, from a webpage.)Try this website : http://people.csail.mit.edu/pgbovine/python/tutor.html .

5 List comprehension

(Ref Pg. 63) Syntax : new_list = [f(e) for e in some_other_list]

In [11]:

```
list_of_first_100_odds = [(2*n + 1) for n in range(100)]
print list_of_first_100_odds
```

```
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35,
37, 39, 41, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63, 65, 67, 69,
71, 73, 75, 77, 79, 81, 83, 85, 87, 89, 91, 93, 95, 97, 99, 101, 103,
105, 107, 109, 111, 113, 115, 117, 119, 121, 123, 125, 127, 129, 131,
133, 135, 137, 139, 141, 143, 145, 147, 149, 151, 153, 155, 157, 159,
161, 163, 165, 167, 169, 171, 173, 175, 177, 179, 181, 183, 185, 187,
189, 191, 193, 195, 197, 199]
```

In [12]:

```
random_list1 = [3, 6, 1]
random_list2 = [40, 70]
sum_list = [[(i + j) for i in random_list1] for j in random_list2]
print sum_list
```

```
[[43, 46, 41], [73, 76, 71]]
```

We can traverse a list also as follows

In [17]:

```
marks = zip(data_mid, data_end)
print "Marks : ",
print marks
print "-"*70
print "Mid\tEnd"
for m, e in marks :
    print "%5.1f\t%5.1f" % (m, e)
```

```
Marks :  [(23, 45), (45, 44), (83, 95), (90, 87), (12, 24), (87, 100),
(67, 45), (69, 70), (74, 66), (36, 32), (43, 50), (69, 55), (66, 80),
(70, 81)]
----------------------------------------------------------------------
Mid     End
 23.0    45.0
 45.0    44.0
 83.0    95.0
 90.0    87.0
 12.0    24.0
 87.0   100.0
 67.0    45.0
 69.0    70.0
 74.0    66.0
 36.0    32.0
 43.0    50.0
 69.0    55.0
 66.0    80.0
 70.0    81.0
```