Principles of Computer Science II

Collections

Ioannis Chatzigiannakis

Sapienza University of Rome

Lecture 4

Simple Statistics

```
1 def main():
2     sum = 0.0
3     count = 0
4     xStr = input("Enter a number (<Enter> to quit) >> ")
5     while xStr != "":
6         x = eval(xStr)
7         sum = sum + x
8         count = count + 1
9         xStr = input("Enter a number (<Enter> to quit) >>
              ")
10    print( "\nThe average of the numbers is", sum / count)
11
12 main()
```

Simple Statistics: Observations

- The program itself doesnt keep track of the numbers that were entered it only keeps a running total.
- We want to extend the program to compute not only the mean, but also the median and standard deviation.

Median

The median is the data value that splits the data into equal-sized parts.

Standard Deviation

$$s = \sqrt{\frac{\Sigma(\bar{x} - x_i)^2}{n-1}}$$

Simple Statistics: Extensions

- We need to keep track of all the values inserted by the user
- We do not know how many variables the user will provide.

Lists

- Python provides List to store sequences of values
- Lists in python are dynamic.
  - They grow/shrink on demand.
- Lists are mutable
  - Values can change on demand
  - Data type of individual items can change

Lists: Basic Examples

```
lst = [1,5,15,7]
print(lst)

lst[2] = 22
lst

lst[1] =    Hello
lst

zeroes = [0] * 5
zerones = [0,1] * 3
zerones.append(2)
```

Lists: Operators

Operator	Meaning
seq + seq	Concatenation
seq * integer	Repetition
seq[]	Indexing
len(seq)	Length
sec[:]	Slicing
for var in sec:	Iteration
(expr) in sec	Membership (boolean)

Lists: Basic Examples

```
lst = lst + [22, 3]
len(lst)

15 in lst
3 in lst

sum = 0
for x in zerones:
    sum += x
print(sum)

X = zerones
zerones.append(2)

Y = lst[1:3]
Z = lst[3:-1]
K = lst[1:-3]
```

Lists: Methods

Method	Meaning
seq.append(x)	Add element x to end of list.
seq.sort()	Sort (order) the list. A comparison function may be passed as a parameter.
seq.reverse()	Reverse the list.
seq.index(x)	Returns index of first occurrence of x.
seq.insert(i, x)	Insert x into list at index i.
seq.count(x)	Returns the number of occurrences of x in list.
seq.remove(x)	Deletes the first occurrence of x in list.
seq.pop(i)	Deletes the ith element of the list and returns its value.

Lists: Basic Examples

```
1 lst = [3, 1, 4, 1, 5, 9]
2 lst.append(2)
3 lst
4
5 lst.sort()
6 lst
7
8 lst.reverse()
9
10 lst.index(4)
11
12 lst.insert(4, "Hello")
13
14 lst.count(1)
15
16 lst.remove(1)
17
18 lst.pop(3)
```

Simple Statistics: Modifications

- Collect input from user
- Store in a list

Simple Statistics: Modifications

- Collect input from user
- Store in a list

```
1 nums = []
2 x = input('Enter a number: ')
3 while x >= 0:
4     nums.append(x)
5     x = input('Enter a number: ')
```

Simple Statistics: Modifications

- Collect input from user
- Store in a list

```
nums = []
x = input('Enter a number: ')
while x >= 0:
    nums.append(x)
    x = input('Enter a number: ')
```

```
def mean(nums):
    sum = 0.0
    for num in nums:
        sum = sum + num
    return sum / len(nums)
```

Further Extensions

- How do we compute the standard deviation?
- Do we re-compute the mean?
  - Inefficient for large collections
- Do we pass the mean as a parameter?
  - Forced to invoke both functions sequentially

Further Extensions

- How do we compute the standard deviation?
- Do we re-compute the mean?
  - Inefficient for large collections
- Do we pass the mean as a parameter?
  - Forced to invoke both functions sequentially

```
def stdDev(nums, xbar):
    sumDevSq = 0.0
    for num in nums:
        dev = xbar - num
        sumDevSq = sumDevSq + dev * dev
    return sqrt(sumDevSq/(len(nums)-1))
```

Median

- How do we compute the median?
- Pseudocode
  1. sort the numbers into ascending order
  2. if the size of the data is odd:
  3. median = the middle value
  4. otherwise median = the average of the two middle values
  5. return median

Median

- How do we compute the median?
- Pseudocode
  1. sort the numbers into ascending order
  2. if the size of the data is odd:
  3. median = the middle value
  4. otherwise median = the average of the two middle values
  5. return median

```
def median(nums):
    nums.sort()
    size = len(nums)
    midPos = size / 2
    if size % 2 == 0:
        median = (nums[midPos] + nums[midPos-1]) / 2.0
    else:
        median = nums[midPos]
    return median
```

Simple Statistics: New Version

```
def main():
    print("This program computes mean, median and
        standard deviation.")

    data = getNumbers()
    xbar = mean(data)
    std = stdDev(data, xbar)
    med = median(data)

    print('\nThe mean is', xbar)
    print('The standard deviation is', std)
    print('The median is', med)
```

Range

- range creates a list of numbers in a specified range
- range([start,] stop[, step])
- When step is given, it specifies the increment (or decrement).

```
range(5)

range(5, 10)

range(0, 10, 2)

for i in range(0, len(lst), 2):
    print lst[i]
```

Zipping Lists

```
lst = [1,5,15,7]
zerones = [0,1] * 3

k = zip(lst, zerones)

for (i,j) in k:
    print (i,j)
```

Tuples

```
1 data = [("julius", 3),
2 ("maria", 2),
3 ("alice", 4)]
4
5 for (n, a) in data:
6     print("I met %s %s times" % (n, a))
7
8 data.sort()
```