

# BISO560R Spring 2014 Homework 2

Pittard

Due by 11:59 PM on February 14, 2014

## Instructions

Send responses in a plain text file name LastName\_Firstname\_HW2.txt or LastName\_Firstname\_HW2.R. You can use RStudio to create the latter. We will run your commands at the R console to verify the statements. Email to BOTH [dvandom@emory.edu](mailto:dvandom@emory.edu) and [wsp@emory.edu](mailto:wsp@emory.edu)

## 1 Data Frames

Write a function called `my.sampler` that fulfills the following requirements: It will take a data frame, such as `mtcars`, and “split” it up based on a given grouping variable. Then it will process each group and sample a specified number of records, (without replacement), from each group. When it is finished processing all groups it will combine those results into a data frame and return it. You need only accommodate a single group/factor per function call. As an example:

```
my.sampler(mtcars,mtcars$cyl,3)
```

|                    | mpg  | cyl | disp  | hp  | drat | wt    | qsec  | vs | am | gear | carb |
|--------------------|------|-----|-------|-----|------|-------|-------|----|----|------|------|
| Toyota Corona      | 21.5 | 4   | 120.1 | 97  | 3.70 | 2.465 | 20.01 | 1  | 0  | 3    | 1    |
| Honda Civic        | 30.4 | 4   | 75.7  | 52  | 4.93 | 1.615 | 18.52 | 1  | 1  | 4    | 2    |
| Merc 230           | 22.8 | 4   | 140.8 | 95  | 3.92 | 3.150 | 22.90 | 1  | 0  | 4    | 2    |
| Merc 280           | 19.2 | 6   | 167.6 | 123 | 3.92 | 3.440 | 18.30 | 1  | 0  | 4    | 4    |
| Mazda RX4          | 21.0 | 6   | 160.0 | 110 | 3.90 | 2.620 | 16.46 | 0  | 1  | 4    | 4    |
| Merc 280C          | 17.8 | 6   | 167.6 | 123 | 3.92 | 3.440 | 18.90 | 1  | 0  | 4    | 4    |
| Hornet Sportabout  | 18.7 | 8   | 360.0 | 175 | 3.15 | 3.440 | 17.02 | 0  | 0  | 3    | 2    |
| AMC Javelin        | 15.2 | 8   | 304.0 | 150 | 3.15 | 3.435 | 17.30 | 0  | 0  | 3    | 2    |
| Cadillac Fleetwood | 10.4 | 8   | 472.0 | 205 | 2.93 | 5.250 | 17.98 | 0  | 0  | 3    | 4    |

  

```
my.sampler(ChickWeight,ChickWeight$Diet,2) # ChickWeight is built-in to R
```

|     | weight | Time | Chick | Diet |
|-----|--------|------|-------|------|
| 20  | 138    | 14   | 2     | 1    |
| 57  | 197    | 16   | 5     | 1    |
| 253 | 145    | 16   | 23    | 2    |

|     |     |    |    |   |
|-----|-----|----|----|---|
| 270 | 49  | 2  | 25 | 2 |
| 361 | 221 | 16 | 32 | 3 |
| 389 | 41  | 0  | 35 | 3 |
| 513 | 135 | 12 | 45 | 4 |
| 554 | 322 | 21 | 48 | 4 |

Your function will have to work with the other potential grouping variables from the given data set (e.g. gear, transmission, carb, and vs ). Once you have split the data frame on a factor you should be able to, as part of your looping or apply logic, determine how many rows are in each category using functions you've learned about in class. We will also run your function on the Chick-Weight data and the diamonds dataset, (in the ggplot2 package), to insure that there is nothing specific in it to mtcars

THINGS TO CHECK: Insure that the number of unique values that the grouping variable takes on is less than 10. So if you are given a factor to split on then check that it assumes at most 10 unique values - otherwise complain to the user with an appropriate error message.

You should also do error checking to make sure that the number of records you are attempting to sample is not larger than the available number of records in the group. In that case then your program will stop with an informative message. For example in looking at the mtcars data, one can see that there is only one car that has a value of 6 in the carb column, thus

```
my.sampler(mtcars,mtcars$carb,2) would fail:
```

```
Error in my.sampler(mtcars, mtcars$carb, 2) :
  Number of requested samples 2 exceeds available records
```

# As would the following because MPG really isn't a grouping variable:

```
my.sampler(mtcars,mtcars$mpg,5)
Error in my.sampler(mtcars, mtcars$mpg, 5) :
  Grouping factor has 25 unique values. Must be less than 10
```

Table 1: Arguments to my.sampler function

| Argument    | Purpose   |
|-------------|---|
| my.df       | a dataframe (e.g. mtcars, ChickWeight, etc)                 |
| my.group    | a grouping variable from my.df                              |
| numtosample | the number of records to sample from each value of my.group |

## 2 DNA

### 2.1 seqgen

Write a function called `seqgen` that returns a randomly generated DNA nucleotide sequence. Here we will restrict ourselves to using four characters - ACGT. For this exercise all 4 bases occur with the same probability. Your function should take the following arguments:

Table 2: Arguments to `seqgen` function

| Argument   | Purpose  |
|--|--|
| <code>size</code>                                    | the number of characters in the desired sequence                         |
| <code>collapsed</code> (default <code>TRUE</code> )  | return a collapsed character string or, if <code>FALSE</code> , a vector |
| <code>upper.case</code> (default <code>TRUE</code> ) | return upper or lower case   |

As an example, a collapsed string would look like "AGTTGG" whereas an uncollapsed string would look like "A" "G" "T" "T" "G" "G". Your function should validate the size of the requested sequence to make sure it makes sense. That is check for negative numbers or character strings. The second and third arguments should have defaults of `TRUE` and `TRUE` respectively. Here are some examples of how the function might be called:

```
seqgen(20,T,F)
[1] "ggtatacaatccccgtgggt"
```

```
seqgen(20,T,T)
[1] "GCTGCAATCTCGGCCTTGAT"
```

```
seqgen(20,F,T)
[1] "G" "T" "A" "A" "G" "C" "A" "T" "C" "A" "G" "A" "A" "C" "G" "C" "T" "C" "C" "C"
```

```
seqgen(0)
Error in seqgen(0) : Sorry - size needs to be numeric and greater than or equal to 1
```

### 2.2 revcomp

Write a companion function called `revcomp` that, given any valid sequence generated by `seqgen`, will return the complimentary sequence. *Note that for this assignment use of the "chartr" function is forbidden.* In DNA terminology certain bases have specific compliments. The character A changes to T, C changes to G, G changes to C, and T changes to A. As an example the string "ACGTTGA" would have a compliment of: TGCAACT.

Moreover, if we wanted to take the “reverse” compliment of a string we would first reverse the given string and then apply the compliment substitution as indicated. So given the same string as above, “ACGTTGA”, we would first reverse it to get “AGTTGCA” after which we would apply the compliment substitutions to get a reverse compliment of “TCAACGT”. Use [http://www.bioinformatics.org/sms/rev\\_comp.html](http://www.bioinformatics.org/sms/rev_comp.html) to check your work. Make sure to use the drop down menu to reflect whatever it is you are testing: “complement”, “reverse compliment”, etc. Your function will take the following arguments:

Table 3: Arguments to revcomp function

| Argument                | Purpose  |
|-------------------------|--|
| sequence                | A sequence string like that returned by seqgen               |
| reverse (default FALSE) | If TRUE then return the compliment of the reverse DNA string |

Here are some examples of how revcomp might be called:

```
my.seq = seqgen(20)
```

```
my.seq
[1] "C" "A" "T" "T" "A" "T" "C" "A" "T" "C" "G" "C" "A" "A" "T" "T" "C" "T" "T"
[20] "A"
```

```
revcomp(my.seq)
[1] "g" "t" "a" "a" "t" "a" "g" "t" "a" "g" "c" "g" "t" "t" "a" "a" "g" "a" "a"
[20] "t"
```

```
revcomp(seqgen(20),collapsed=T)
[1] "ccactcctattgatattata"
```

If you want a specific test sequence then use the following. However there should be nothing in your code that is specific to this string:

```
TAACGGCGGCAGGCGTATGG
```

```
revcomp("TAACGGCGGCAGGCGTATGG", reverse= F, collapsed=T )
[1] "attgccgccgtccgcatacc"
```

```
revcomp("TAACGGCGGCAGGCGTATGG", reverse=T, collapsed=T)
[1] "ccatacgctgccgccgtta"
```

### 3 Quartile

Write a function called “quartile.nist” which computes the requested percentiles of a given vector `x`. The calling sequence should look like this:

```
quartile.nist(x, probs=c(0.25,0.50,0.75))
```

Table 4: Arguments to quartile.nist function

| Argument           | Purpose  |
|--------------------|--|
| <code>x</code>     | a numeric vector of any size                                       |
| <code>probs</code> | a vector of desired percentile(s) where ( $0 < \text{probs} < 1$ ) |

The default value for `probs` should be `c(.25,.50,.75)`. Check for values in `probs` that are less than or equal to zero or greater than or equal to 1. Some examples are:

```
x = c(19,15.5,15.0,20.5,18.3,20.9,11.7,24.3,23.9)
quartile.nist(x)
```

```
 25%  50%  75%
15.3 19.0 22.4
```

```
quartile.nist(x,c(.40,.60))
 40%  60%
18.3 20.5
```

```
set.seed(145)
testx = rnorm(1000)
quartile.nist(testx)
 25%    50%    75%
-0.5810 0.0756 0.7543
```

#### 3.1 Background

Suppose that we have a data vector  $x_1, x_2, \dots, x_n$ . The  $p$ th percentile is conceptually meant to be a value  $Q_p$  such that  $p$  proportion of the observations fall below  $Q_p$ . There are different ways to compute the percentiles / quartiles. Here we will use the “n+1” method. Consider the data set:

```
Xn = 19.0 15.5 15.0 20.5 18.3 20.9 11.7 24.3 23.9
```

```
x = c(19,15.5,15.0,20.5,18.3,20.9,11.7,24.3,23.9)
```

Here  $n = 9$ . We will first need to order these values:

```
sortedx = 11.7 15.0 15.5 18.3 19.0 20.5 20.9 23.9 24.3
```

To compute the 20th percentile of the data we calculate  $Q_{20}$  as follows:

$$(n + 1) * p = (9 + 1) * .20 = 2$$

So the value representing the 10th percentile is `sortedx[2]` which is 15. Thus for computed Q values that wind up being an integer then you simply use the Q value to index into the sorted vector and you are done. Now, let's compute the 33rd percentile. Here we must use interpolation which will require some adjustment to our approach.

$$Q = (n + 1) * p = (9 + 1) * 0.33 = 3.3$$

Note that we have a number with a fractional part. HINT: It will be very useful for you to code up some logic to break up this number into its whole component (3) and its fractional component (0.3). You will need these values early on in your function to do comparisons. So here we take the third observation (the whole part of Q) of the SORTED vector and add to it 0.3 (the fractional part of Q) times the distance between it and the next highest (4th) observation. So:

$$Q_{33} = \text{sortedx}[3] + 0.3 * (\text{sortedx}[4] - \text{sortedx}[3]) \text{ is } 16.3$$

```
quartile.nist(x,.33)
      33%
16.34
```

So that is the approach you implement for computed Q values that have a fractional part. Well almost. There are “boundary conditions” that we must consider. For low or high values of prob (like less than .10 or greater than .90) you need to pay attention to the computed Q value. Consider the case where prob is 0.95.  $Q = (9 + 1) * .95 = 9.5$ . Since there are only 9 elements in the original vector you can't interpolate it since there are no other elements above element 9. So in this case you would subtract 1 from the whole part (giving 8) and then apply the given interpolation formula:

$$Q_{95} = \text{sortedx}[8] + 0.5 * (\text{sortedx}[9] - \text{sortedx}[8]) = 24.1$$

```
quartile.nist(x,.95)
      95%
24.1
```

Also consider the case where prob is 0.05.  $Q = (9 + 1) * .05 = 0.5$ . Note that there is no *zero-th* element of the vector. So we add 1 to Q and apply the given interpolation formula. So Q is 0.5 but we add 1 to it to get 1.5 and then apply the formula:

```
Q05 = sortedx[1] + 0.5*(sortedx[2] - sortedx[1]) = 13.3
```

```
quartile.nist(x,.05)
```

```
5%
```

```
13.35
```