**Descriptive statistics!**

**Some useful commands:**

> getwd()

tells you where R is looking for input and where R is storing output

> list.files()

tells you what files are in this location

> hurr<-read.csv("<http://csivc.csi.cuny.edu/Lisa.Manne/files/classes/biol78002/NOAA_hurricane>s.csv”, header=T)

## Note: you can abbreviate TRUE to T.

> head(hurr)

This dataset has the following columns:

1. Year
2. Named\_Storms (Storms that achieve wind speeds of 39 mph)
3. Hurricanes (Saffir score 1-5, wind speeds of 74 mph)
4. Major\_Hurricanes (Saffir score 3-5)
5. ACE (Accumulated Cyclonic energy)
6. Notes

Tropical cyclones are the only storms that forecasters name. Organized naming began in the 1950s.

Hurricanes are tropical cyclones in the Atlantic Basin (Atlantic, Caribbean Sea and Gulf of Mexico) and in the Pacific east of the International Date Line. Typhoons are tropical cyclones over the northern Pacific west of the date line. Tropical cyclones in the South Pacific and over the Indian Ocean are just called cyclones.

**More inquiries about files:**

What do these commands do or tell you?

> str(hurr)

>  dim(hurr)

>   names(hurr)

>   hurr[1,]

>   hurr[1:20,4]

>   sum(hurr[1:20,3])

> tail(hurr)

> nrow(hurr)

Perhaps you want to query for information about your file without having to scan through the data file with your own eyes. You can do this.

> sum(is.na(hurr[,4])) # counts how many NAs are in the 4th column

>  which(is.na(hurr[,4])) # what does this command do?

Note also: “#” tells R that the following text is a comment. This is useful for when you are writing scripts: files containing lists of commands to be completed as a batch.

“apply” is a useful trick to be aware of. It applies a function to a dataframe or subset of a data frame.

The syntax is: apply(dataframe, flag, function) where *flag* indicates by rows or columns (1 if by rows, 2 if by columns)

 >  apply(hurr, 2, max) # what does this return? Does this make sense?

 > apply(hurr, 2, min)

 >  apply(hurr, 2, median)

>   apply(hurr, 2, mean)

But what if you don’t want to be calculating averages (etc) on a data set that has missing values? There is an easy solution – remove any rows that have missing values:

hurr2 <- na.omit(hurr) # na.omit gets rid of all rows that contain missing values

Repeat the apply commands above, has anything changed?

**Standard errors, confidence intervals**

The standard error of an estimate (say, the mean) is an estimate of the standard deviation of that estimate. Let us consider the *Named\_Storms* variable in the *hurr2* dataframe. We will calculate the mean of Named\_Storms:

> mean(hurr2[,2]) # Named\_Storms is in the 2nd column; alternately, could use *apply* construction above

The standard error of this estimate is  
  
>   sd(hurr2[,2]) / sqrt(166)

How would you change this standard error calculation to take advantage of the *nrow* command?

The Central Limit Theorem tells us that the sample mean has a roughly normal distribution centered at the true population mean. Thus, we may use the fact that 95% of the data from any Gaussian distribution is contained within 1.96 standard deviations of its mean to construct the following 95% confidence interval for the true population mean of Named\_Storms:

 > mean(hurr2[,2]) + c(-1.96,1.96)\*sd(hurr2[,2]) / sqrt(166)

For any sample of data [the column containing the Named\_Storms data is one such sample], 95% of confidence intervals constructed as above will contain the population mean.

**Investigating data graphically with histograms**

One of the characteristics of normal data is that the median and the mean will have similar values. Alternately, skewed data will show a mean value that is quite different from the median.

Let us examine a frequency distribution of the Named\_Storms data:

> attach(hurr2)

# *attach* allows us to refer to column names without using either the *hurr2$Named\_Storms* notation, or the *hurr2[,2]* notation

> hist(Named\_Storms)

Note that there is a small bit of skew evident, but a comparison of the mean and median shows that the skew is actually less than one named storm

> mean(Named\_Storms)

[1] 9.204819

> median(Named\_Storms)

[1] 8.5

So, not too much skew, really.

**Coefficient of Variation:**

Use CV to determine which variable is most variable. [Why can’t you use standard deviation or variance for this? Two reasons.]

> cv(Named\_Storms)

This didn’t work because, although the relevant command is “cv”, “cv” is not part of the “base” set of commands in R.

> ?cv

No documentation for 'cv' in specified packages and libraries:

you could try '??cv'

> ??cv

In the dialog box that pops up, you’ll see commands (and the packages that they are in) that might fit what you want. In this particular case, all the commands that pop up are cross-validation commands (not what we want), so let us construct a function to calculate cv. The syntax of function definitions is:

> Myfunction <- function(x,…) {

calculate stuff

return (stuff) } # notice that the curly braces enclose the commands of the function; both *function* and *return* are reserved words.

So the particular command would be:

> cv<-function(x) { result <- sd(x)\*100/mean(x) ; return(result) } # the semicolon allows us to put multiple commands on a single line, and if those commands should spill over to multiple lines, the semicolon helps R to recognize where commands begin and end

> cv(Named\_Storms)

[1] 22.81925

Calculate the cv’s of the hurricanes and major hurricanes. Which measures are most variable?

**New York weekly incomes**

Compare to a dataset of average NY weekly incomes. Download the ny\_income.csv dataset to your local drive and read it into R

> h<-“http://csivc.csi.cuny.edu/Lisa.Manne/files/classes/biol78002/ny\_income.csv”

> income<-read.csv(h, header=T)

> head(income)

County Income

1 Albany 907

2 Allegany 656

3 Bronx 828

4 Broome 692

5 Cattaraugus 658

6 Cayuga 658

> attach(income)

> hist(Income)

Somewhat more skewed.

> mean(Income)

[1] 724.541

> median(Income)

[1] 685

> cv(Income)

[1] 15.25843

**Installing and loading a new package**

One of the very powerful things about R is that it is open-source (so that people are always writing new packages or modules to use with it, and these are freely available to you), and that you only need load the packages that you anticipate needing for a particular session. This makes the running of R very streamlined. To use a new package, you need to install it (once per computer) and then load it (in every session).

We now need to install and load the *MASS* package.

Packages -> Install package(s)

The dialog will ask you to select a location from which the download will come; select one near us (PA or MD). Then another dialog will pop up asking you to choose a package (it is a long list). Choose *MASS*.

This will result in the package being installed to your local machine (in your GC account) (but not loaded into your current session – you’ll have to do that step too.)

Packages -> Load package [then choose MASS] -- alternately > library(MASS)

**Other versions of histograms:**

*truehist*, in the MASS package. Now that the MASS package is loaded, ask R to draw histograms of the NY income and Named\_Storms data using the *hist* and *truehist* commands. What are the differences between *hist* and *truehist*?

**Sampling distributions from the human gene lengths dataset**

Download the human gene lengths dataset: http://csivc.csi.cuny.edu/Lisa.Manne/files/classes/biol78002/HumanGeneLengths.csv, and load it into R.

The *sample* command will choose a number of elements randomly from a column of data.

> mean(sample(g$gene.length, 100)) # I named the gene length dataset “g”; you may have named yours something different. Alter the command accordingly. This command chooses 100 values randomly from the gene lengths in the gene.length column in your gene length dataset

[1] 2431.59

Repeat this command 10 times (up arrow –> return, up arrow –> return, etc.) and note the change in the mean of your random sample.

You can also view a histogram of your results:

> hist(sample(gene.length, 100))

Let’s automate this process. We are going to take the mean of a random sample (n=100) of gene lengths, 20 times; record the 20 means that you get, and calculate their mean.

> outvec<-numeric(30) # create empty space to store sample means

> for (i in 1:30) { outvec[i]<-mean(sample(gene.length, 100)) } ## {} contain for loop instructions

> mean\_of\_means<-mean(outvec)

Note that you can take the mean of data you enter yourself, inline, but why would we do that:

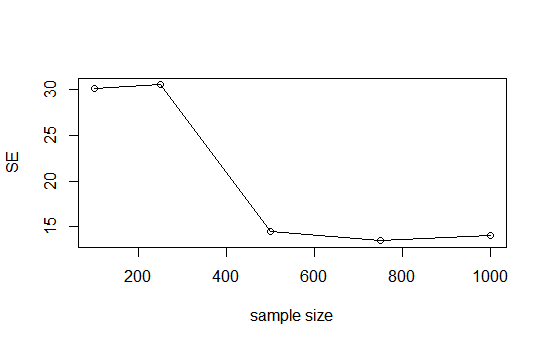
> mean(2431.59, 2754.46, 2528.71, 2697.01) # (etc.)

We’ll now repeat this process for n=250 (30 random samples of size 250), n=500 (30 random samples of size 500), n=750 (30 random samples of size 750), and n=1000 (30 random samples of size 1000). How large does sample size need to be, in order to insure the precision that we want? To answer this, calculate standard errors (of your mean of the 30 sample means) as we did earlier in this exercise. Perhaps generate a plot of the size of your random sample (x) versus the SE for that size sample (y).

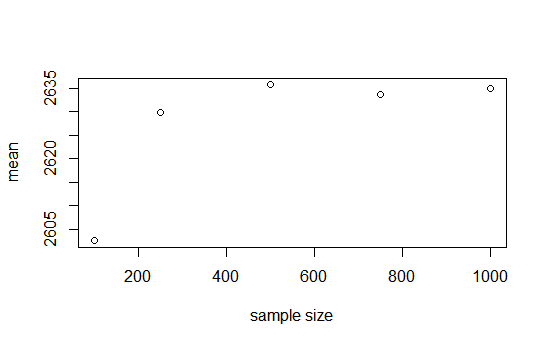
Let’s script this. You will need two loops. The outer loop will loop over the size of the random sample (100, 250, … , 1000). The inner loop will sample randomly from the gene.length column and calculate the mean of the random sample (the for loop in line 197 above).

|  |  |  |
| --- | --- | --- |
| **Size of random sample** | **Mean of the 30 sample means** | **SE** |
| 100  250  500  750  1000 |  |  |

Here is what I get:



and



The mean stabilizes around 2635 nucleotides, with a stable and small SE of 15 nucleotides, by the time you have taken 500 samples.

Code that will do this:

## I had already attached the genes dataframe

reps<-c(100,250,500,750,1000)

outvec<-numeric(30) # create empty space to store sample means

mns<-numeric(length(reps)) ## mean of means

ses<-numeric(length(reps)) ## SE of means

for (j in 1:length(reps)) {

for (i in 1:30) { outvec[i]<-mean(sample(gene.length, reps[j])) }

mns[j]<-mean(outvec)

ses[j]<-sd(outvec)/sqrt(30) }

plot(reps, ses, xlap="sample size", ylab="SE")

##plot(reps, ses, xlab="sample size", ylab="SE", type="l") ## type = ell

plot(reps, mns, xlab="sample size", ylab="mean")

##plot(reps, mns, xlab="sample size", ylab="mean", type="l")