MONT 105N – Analyzing Environmental Data
Chapter 10 Project
March 18, 2019

Introduction

The goals of this chapter's project are:

- To introduce some basic features of the `R` statistics package. This is a much more extensive and powerful piece of software than the statistical routines in Google Sheets or Excel. In fact `R` is probably the go-to "industry standard" software for academic statisticians at the moment.
- To see how `R` computes some descriptive statistics and graphics for understanding the "shape" or "distribution" of a data set.
- To gain an intuitive understanding of the *Central Limit Theorem*, a basic result in statistics that underlies the wide applicability of normally-distributed random variables.

Background on `R` – Getting Started

The instructions here are geared toward the Windows version of `R`. `R` is a free, open source package and you can obtain a copy to run on your own computer, and there are Windows, Mac-OS, and Linux versions available. These have the same functionality but slightly different interfaces.

On PC's when you double-click the `R` shortcut from the standard desktop, you will launch the R-GUI (graphical user interface) window. Inside this is a workspace window with a white background. It *only* allows you to

- enter commands, and
- view numerical output.

It does not let you create integrated documents, and graphics are displayed in separate windows within the R-GUI, as you will see.

You can *print* the workspace window or any graphics window at any time. Just highlight that window and press the toolbar printer button.

To save your work, with comments and answers to questions (e.g. for the project report), I recommend

- opening another editor window (either Word or NotePad are fine for this on Windows systems), and
- using it to create a text file that will be your permanent record of your work.
- When you have a result you want to save from the `R` workspace, copy both the input command and the output and paste into the text file.

- Any time you want to add a comment to explain a calculation or answer a question, enter that directly into the text file.
- Save that text file print it when you want a hardcopy (from Word or NotePad).

A First `R` Session

In this first session, you will basically work through some examples illustrating features of `R` and some of the descriptive statistics we have talked about so far in the text. Keep a record of what you do for future reference.

In its most basic form, `R` provides a command-driven "statistical calculator" in which you type in data and commands to generate numerical and graphical output that are displayed. When you have the `R` window open, note the $>$ input prompt after the opening message. This is where you should start.

(1) The following data set records systolic blood pressure readings from 15 adults (not a very healthy group, on the whole!) First we need to get the systolic blood pressure readings into R so that we can work with them. The basic way is to enter (after the $>$ prompt):

```
bpdata <- c(172,140,123,130,155,148,138,129,137,161,123,152,133,128,142)
```

and press ENTER. Note: the $<-$ is formed by typing the $<$ and $-$ characters right next to each other. It is the *assignment operator* in R. This takes the data on the right and assigns it to the name on the left. The *c*, followed by the list of numbers, separated by commas, is `R`'s way of constructing *an ordered list or vector*. You should just get another input prompt if everything went OK with this (error messages if not). If you did get an error, try again, and try to follow the above exactly. You can verify that the numbers are entered correctly by typing in the name `bpdata` as a command; the numbers will be displayed in a row.

(a) We can compute the mean $\overline{y}$ and standard deviation s of the data in `R` (in several different ways!): Type in

```
ybar <- sum(bpdata)/length(bpdata)
mean(bpdata)
s <- sqrt(sum((bpdata - ybar)^2)/(length(bpdata) - 1))
sd(bpdata)
```

and press ENTER after each line to see the results. Note the way the sum function works in a very nice way here—in the computation of `ybar`, we only need to put in the name of the data list; the sum of all the elements is computed by default. Then in the computation of `s`, we can subtract *each entry* of the `bpdata` vector from `ybar` as shown! Of course we will usually want now to use the built-in functions `mean` and `sd`, but the first of each pair of computations shows how we can also use more primitive functions in `R` when we need them. The command `summary(bpdata)` automates the computation of the five number summary if you want to see that.

(b) Chebyshev's theorem mentioned in Chapter 9 is a result about the distribution of data sets. It says in a particular case that *there will always be at least* 75% *of the numbers in a data set within 2 standard deviations of the mean.* Let's use R to check that this is true here. In its most basic use, R can function as a simple numerical calculator.

```
upperb <- ybar + 2*s
lowerb <- ybar - 2*s
```

How many of the data values are in the range $[\overline{y} - 2s, \overline{y} + 2s]$, though? We could just count by hand, of course, with a data set this small. But R also contains a very rich collection of facilities for selecting subsets of a data set and massaging data in various ways. Here is a basic way to select the portion of the data set in the range we want:

```
middle <- subset(bpdata, bpdata <= upperb & bpdata >= lowerb)
length(middle)/length(bpdata)
```

(The first line picks out the subset of the data satisfying both of the inequalities; the second computes the fraction of the whole data set contained in the "middle.") Did Chebyshev's theorem "work" here?

(2) Now let's look at a different data set. This gives the number of parasites found in each of a sample of 100 individual foxes. The information is given there in table format:

Parasites	0	1	2	3	4	5	6	7	8
Foxes	69	17	6	3	1	2	1	0	1

So we want to create a data set containing 69 0 entries, 17 1 entries, etc. Clearly this is going to be tedious and error prone if we do it the most basic way(!) Fortunately, R lets us construct lists in more flexible ways besides just listing the entries. We can indicate how many times to repeat a given value if we want, like this:

```
foxpar <- rep(c(0,1,2,3,4,5,6,7,8),c(69,17,6,3,1,2,1,0,1))
```

The `rep` stands for "repetition" here. Note that the first list inside the `rep` gives the values of the number of parasites—the first row in the table format—and the second list gives number of times each value appears in the data. The first list could also be abbreviated as `0:8` (R's notation for a list of successive integer values).

(a) To construct a relative frequency histogram for the number of parasites per fox in R, let's begin by entering

```
hist(foxpar)
```

The histogram will be generated in a separate graphics window. This is a "default" version of the command. The `hist` command takes a number of options that control how the bins are selected and how the histogram is drawn as well. You will often want to modify the default results. For instance, looking at the output, how were the bins chosen here? Is that optimal for showing the different numbers of foxes with each number of parasites? Suppose we want bins centered on the integer values $0, 1, 2, 3, 4, 5, 6, 7, 8$ as we did in other examples. We can put in an option to make `R` do it that way too. The `breaks` option takes a list of values and makes those the boundaries of the bins for the histogram:

```
hist(foxpar,breaks=c(-0.5,0.5,1.5,2.5,3.5,4.5,5.5,6.5,7.5,8.5))
```

How is this different and why is it better than the default way that `R` drew the histogram?

(b) To practice using the commands we talked about before, calculate $\overline{y}$ and s for this data set. How many of the numbers were within two standard deviations of the mean in this case?

(c) The list of breaks for the histogram can be anything in `R`. What happens if you change the command above to

```
hist(foxpar,breaks=c(-0.5,0.5,1.5,2.5,3.5,4.5,5.5,8.5))
```

(deleting the 6.5 and 7.5 break points, so we're lumping together the counts for the values 6,7,8 parasites)? What is different about the new histogram that `R` drew in this case? Look at the graph carefully: do the heights of the boxes still represent *frequencies*? (Hint: what they do represent are *densities* that are computed by taking the frequency count and dividing by the width of the box in each case. What is true if we add the areas of the boxes drawn this way? We will see other reasons why this is a preferable way to draw histograms with unequal bin sizes later.) If we *really want frequencies*, we can make `R` do it that way:

```
hist(foxpar,breaks=c(-0.5,0.5,1.5,2.5,3.5,4.5,5.5,8.5),freq=TRUE)
```

However note that now `R` generates a warning message saying the AREAS in the plot are "wrong." Think about this. Why is the last box in this version of the histogram *misleading* if drawn this way?

Background on Probability Distributions in `R`

The R package contains built-in functions for computing probability functions for the discrete random variables we have discussed, plus functions for the probability density functions and cumulative distributions of the standard types of continuous random variables. Here's how it works. Each type of random variable is covered by a *family* of 4 functions distinguished by a *prefix letter* `d, p, q`, or `r`:

- `d` – the p.m.f. in the discrete case, or the p.d.f. in the continuous case,
- `p` – the c.d.f.
- `q` – the quantile function (it computes quartiles, deciles, percentiles; we'll see examples of how it is used later)
- `r` – random number generator (for sampling random values of a random variable with that distribution)

Following the prefix letter comes the rest of the function name and the inputs needed to compute the corresponding values.

Distributions of Sample Means

Work through the following example before and then follow the pattern here in the other parts of question A below. You do not need to hand in any work for this preliminary discussion, just for the parts of the question itself.

1. Begin by generating a list of 100 random numbers from a particular distribution.[1] Use the command:

   ```
   x <- rexp(100,1/3)
   ```

   to assign the result to the name `x`. We will think of this as a *sample* of values from that distribution.

2. Plot the density histogram for this data set `x` using the command

   ```
   hist(x,freq=FALSE)
   ```

   This should look roughly like the part of the graph of the p.d.f. for positive y, reflecting the fact that the random numbers are generated according to probabilities given by that density function.

3. Our main goal is to understand what happens if we generate *lots* of such samples, find their sample means, then consider *the distributions of those sample means.*

4. Here is one way to do this in `R` using a simple *for loop.* First we create space to store the means of 1000 different random samples:

   ```
   means <- array(1:1000)
   ```

   Next, we generate 1000 different random samples, find the means of each of them, and store them in the array created above:

   ```
   for (i in 1:1000)
      {
      x <- rexp(100,1/3)
      means[i] <- mean(x)
      }
   ```

[1] This is a type of continuous random variable that we did not discuss in the main part of the text. It is one of the exponential distributions from an example in the optional section §10.8. You will see roughly what its p.d.f. looks like below even if you did not look at the optional section.

(Note: This could all be entered as one input line provided you leave spaces in the appropriate places. However, I think it is more readable (and it is easier to identify typos if you happen to make one) if you press ENTER at the end of each of the lines as you are typing this. If you do it that way, you should note that `R` generates a new input prompt `+` each time, indicating that you are still in the body of the `for` loop. The final } will close off the loop and take you back to the > prompt.)

5. What does the distribution of the sample means look like? We can see that by generating a frequency histogram for the data in the `means` array:

   ```
   hist(means,breaks=20,freq=FALSE)
   ```

   This should look completely different from the histogram showing the distribution of the single sample above. In fact, what does this histogram remind you of?

6. If you said, "a normal p.d.f.," good. If you said something different, go back and look again.

7. Now, which normal density is it? Well, by Theorem 10.36 from the text, if we have 100 sampled values $Y_1, \ldots, Y_{100}$ from any distribution where $E(Y_i) = \mu$ for all i, and $V(Y_i) = \sigma^2$ for all i, and

   $$\overline{Y} = \frac{1}{100}(Y_1 + \cdots + Y_{100})$$

   is the sample mean, then we will have

   $$E(\overline{Y}) = \mu$$

   and

   $$V(\overline{Y}) = \frac{\sigma^2}{100},$$

   so the SD of the sample means will be $\sigma/10$. For the exponential random variables we are looking at, we will take it on faith that $\mu = 3$ and $\sigma = 3$. Hence the expected value of the sample means is also 3 and the SD of the sample means is $3/10 = .3$.

8. Let's overlay that normal density and see how well things match:

   ```
   curve(dnorm(x,3,.3),add=TRUE)
   ```

   (The `dnorm` is the normal p.d.f. The 3 is the μ and the .3 is the σ. You will use other normal densities in the questions below. They are specified in the same way.) This should match quite well (not perfectly, though, since there will always be variation in the sampled values as we know from the project for Chapter 9).

Assignment

Repeat the parts of the worked example below for the means of 1000 random samples of size 100 from each of the following distributions. Note: In each case to find the appropriate normal density curve, you will need to recall facts about expected values and variances of the Y_i individually, then use Theorem 10.36.

(A) A uniform distribution with $min = 4$ and $max = 10$. (These are the endpoints of the interval where the uniform density is different from zero.) Use `runif(100,4,10)` to generate a random sample of size 100 from this uniform distribution. See Theorem 10.33 in the text for the appropriate μ and σ^2 in this case. Then apply Theorem 10.36 as in the example above.

(B) A Poisson distribution with $\lambda = 2$. From the discussion in the text, we know $\mu = \lambda$ and $\sigma^2 = \lambda$ for Poisson random variables. The appropriate `R` function to generate the samples of size 100 is `rpois(100,2)`.

What is happening in each case? Explain. Is the distribution of the sample means approximately normal every time?