Introduction to Statistics (MAT/SST 115.03 2008S)

Getting Started with R

Summary: We begin to explore how one uses the R statistics environment. Along the way, we consider the basic data types R works with and the techniques you can use to explore data in R.

Preliminaries

Unlike many modern programs, R does not have a fancy graphical user interface (aka GUI). Rather, you type commands in R and it responds to those commands, most typically by presenting a result in textual or graphical form.

You start R by opening a terminal window and then entering **R** at the prompt. Try doing so now. You should see some welcome message from the R system. Eventually, you will see the R prompt, a greater-than sign.

```
% R

R version 2.4.0 Patched (2006-11-25 r39997)
Copyright (C) 2006 The R Foundation for Statistical Computing
ISBN 3-900051-07-0

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

Even without knowing anything else about R, you can use R as a simple calculator. That is you can type arithmetic expressions and it will provide the results. For example, Activity 2-2 asks us to compute the proportion of men and women in a study on hand washing who washed their hands.

```
> 2393/3206
[1] 0.746413
> 2802/3103
[1] 0.9029971
```

Similarly, we might find the overall proportion of people who wash their hands by summing the two sub-populations.

```
> (2393+2802)/(3206+3103)
[1] 0.8234269
>
```

Of course, many of us prefer to work with names of values, rather than with values. (What does that 2393 stand for, anyway?) In R, you can associate names with values (including the results of computations) by writing

```
> name = expression
```

For example, we might write:

```
> Men = 3206
> Women = 3130
> MenWashed = 2393
> WomenWashed = 2802
> MenProportion = MenWashed/Men
> MenProportion
[1] 0.746413
> WomenProportion = WomenWashed/Women
> WomenProportion
[1] 0.8952077
> OverallProportion = (MenWashed + WomenWashed)/(Men + Women)
> OverallProportion
[1] 0.819918
```

Sometimes, you'll need more than one line to type an R expression. If you hit **Enter** in the middle of an expression, R will print a plus sign to indicate that it is expecting more input.

```
> (MenWashed + WomenWashed) /
+ (Men + Women)
[1] 0.819918
```

At times, R will think that you're continuing a command, and you can't figure out why. *You can always cancel a command in R by hitting **Ctrl-C***. (That is, hold down the key marked **Ctrl** and, while still holding that key, also press the key marked **C**).

R also lets you revisit prior commands by using the up-arrow and down-arrow keys.

Exercise 1: Simple Computations

a. Start R.

b. Verify that the computations above worked as advertised. Note that for the longer expressions, you may want to copy and paste from the online version of this lab.

Exercise 2: Simple Computations, Continued

As you may recall, Activity 2-6 reports on two sections of Introductory Statistics, one that they term a "Regular Section" and one that the term a "Sports section". In the Regular section, 16 students got good scores, 11 students got fair scores, and 2 students got poor scores. In the Sports section, 7 students got good scores, 15 students got fair scores, and 6 students got poor scores.

a. Write R expressions that assign names to those six values.

b. Write an R expression that computes the total number of students in the Regular section and assigns that value to `RegularStudents`.

c. Write an R expression that computes the proportion of students who passed the Regular section (good and fair grades are passing grades; poor grades are not).

d. Repeat the previous two steps using the grades of the Sports section.

R's Data Types

Clearly, R can work with simple numbers. However, as you've already seen, the practice of statistics involves more computation than just a few numbers. Hence, R supports a variety of kinds of data. For this class, we can focus on four kinds: numbers, strings, vectors, and frames. We've already seen a bit of what you can do with numbers, so we'll focus on the other three kinds.

Strings

Strings are pieces of text that you might use in your analysis. They can be values associated with variables (e.g., the name of a state) or they can be titles you use in your diagrams (e.g., the title of an axis or of the whole diagram). In R, you surround each string with quotation marks, typically double quotation marks. For example, `"Connecticut"`, `"Usage Percentage"`, and `" February High Temperature"` are all strings.

Vectors

As you've noted, we often gather a large number of values in response to a single question. For example, in our study of states that students visited, we had over 100 values spread across the four sections of introductory statistics that participated. Clearly, it would not be sensible to name each of those values separately. R supports *vectors* for sequences of values.

The easiest way to create a vector is with the $c(v_1, v_2, ..., v_n)$ command. Here's a short vector giving a few counts of states visited.

```
> StatesVisited = c(20,37,15,20,20,15,4,18,25,4,4,4)
> StatesVisited
[1] 20 37 15 20 20 15  4 18 25  4  4  4
```

Similarly, here are two vectors, one of which contains the ratios of performances in the regular section and one of which contains strings that describe those ratios.

```
> RegularGood = 16
> RegularFair = 11
> RegularPoor = 2
> RegularTotal = RegularGood + RegularFair + RegularPoor
> RegularProportions = c(RegularGood/RegularTotal, RegularFair/RegularTotal, RegularPoor/RegularTotal)
> RegularLabels = c("Good Grades", "Fair Grades", "Poor Grades")
```

You will use vectors in a variety of ways. For example, you can use the two vectors we've just created to create a simple bar graph. (We'll return to the details of bar graphs a little bit later.)

```
> barplot(RegularProportions, names.arg=RegularLabels, main="Grades in Regular Section")
```

Similarly, you can use the vector of states visited to create a dot plot. (You may find the first command a little confusing. We'll return to the details of dot plots a little later.)

```
> library(BHH2, lib="/home/rebelsky/Stats115/Packages")
> dotPlot(StatesVisited,main="States Visited by Stats Students")
```

More importantly, you can also compute summary statistics for a vector of values. (Don't worry if you don't understand all of the things we compute in the next examples ... they are intended for demonstration purposes only.) In particular, we can find the `length` (number of entries), `max` (maximum value), `min` (minimum value), `mean` (average value), `median` (middle value), and `sum` (total of values) of a vector.

```
> length(StatesVisited)
[1] 12
> max(StatesVisited)
[1] 37
> min(StatesVisited)
[1] 4
> mean(StatesVisited)
[1] 15.5
> median(StatesVisited)
[1] 16.5
> sum(StatesVisited)
[1] 186
```

We can even produce a table that summarizes the values in the vector.

```
> table(StatesVisited)
StatesVisited
 4 15 18 20 25 37
 4  2  1  3  1  1
```

At times, you will need vectors with regularly-spaced values. For example, you might want to create the vector (0,5,10,15,20,25,30,35,40,45,50) as labels for a diagram or as boundaries for ranges. To create vectors of regularly-spaced values, you use the `seq(from=`*`val`*`, to=`*`val`*`,by=`*`val`*`)` command.

```
> seq(from=0,to=50,by=5)
[1]  0  5 10 15 20 25 30 35 40 45 50
> seq(from=10,to=20,by=1.2)
[1] 10.0 11.2 12.4 13.6 14.8 16.0 17.2 18.4 19.6
> seq(from=10,to=1,by=-1)
[1] 10  9  8  7  6  5  4  3  2  1
```

Exercise 3: Vectors and Graphs

a. Try the example above in which we created vectors for the regular section of introductory statistics and then created a bar graph from them. We've repeated those commands here:

```
> RegularGood = 16
> RegularFair = 11
> RegularPoor = 2
> RegularTotal = RegularGood + RegularFair + RegularPoor
> RegularProportions = c(RegularGood/RegularTotal, RegularFair/RegularTotal, RegularPoor/RegularTotal)
> RegularLabels = c("Good Grades", "Fair Grades", "Poor Grades")
> barplot(RegularProportions, names.arg=RegularLabels, main="Grades in Regular Section")
```

b. Create a similar graph for the sports section of the class.

c. Create a similar graph for both sections. (Hint: You'll need to build bigger vectors.)

Exercise 4: Tabulating and Graphing Data

The data in the previous exercise were already tabulated for us. What if the data are not already tabulated? As we've just seen, the `table` command will tabulate for you. You can even tell R to graph that table.

a. Verify that R tabulates the simulated states visited data.

```
> StatesVisited = c(20,37,15,20,20,15,4,18,25,4,4,4)
> table(StatesVisited)
```

b. Build a bar graph from the data by first tabulating it and then building a bar graph from the table.

```
> barplot(table(StatesVisited), main="States Visited by Intro Stats Students")
```

Exercise 5: Tabulating String Data

Here's a vector of strings that represent the grades that students got in a class.

```
> LetterGrades = c("C","A","B","A","A","B","C","C","B","A","C","C")
```

a. Build that vector.

b. Using `table`, build a tabular summary of the grades.

c. We can use `barplot` to plot that table. In particular, try the following and see what happens.

```
> barplot(table(LetterGrades))
```

d. Create a similar vector, table, and plot of the genders of the ten or so students nearest you.

Exercise 6: Computing with Vectors

Interestingly (or perhaps not so interestingly), R lets you include vectors in computations. In this exercise, you'll explore those computations a bit.

a. Create the vector `95,85,90,78,70,90,95,100,92` and name it `Grades`. This vector is intended to represent a group of sample grades.

b. Create the vector `50,20,33,90,60` and name it `Answers`. This vector intended to represent how many students selected answers a, b, c, d, and e on a multiple-choice problem.

c. What do you expect the result of the following command to be? (Don't enter it yet: Describe what you *think* should happen.)

```
> Grades + 10
```

d. Check your answer experimentally. (That is, type in the command and see what the result is.)

e. What do you expect the result of the following command to be? (Once again, don't enter it yet: Describe what you *think* should happen, now that you have a little more experience.)

```
> 2 * Answers
```

f. Check your answer experimentally.

g. Of course, neither of those computations makes sense. Here's one that might. What do you expect the result of the following computation to be?

```
> Grades - mean(Grades)
```

h. Check your answer experimentally.

i. Why might someone want to do that computation?

j. Create a bar graph from those modified data.

```
> students = c("Adam","Beth","Charles","Dani","Eric","Fiona","Greg","Hanna","Iggy")
> barplot(Grades - mean(Grades), names.arg=students)
```

k. What do you expect the output of the following command to be?

l. Check your answer experimentally.

m. Why might someone want to do that computation?

n. Create a bar graph for those scaled counts.

```
> Letters = c("A","B","C","D","E")
> barplot(Answers/sum(Answers), names.arg=Letters, main="Responses to Question 3")
```

Data Frames

Vectors are a great way to accumulate all the values associated with a single variable (as we did for grades and states visited) or to gather data for a bar graph (as we did for numbers of students getting certain categories of grades in different statistics sections). However, most of the time we're analyzing data, we work with more than one variable per observational unit. For example, in Activity 2-4, each OU had three variables: The state name, the kind of law, and the percentage of usage.

When you deal with data with more than one variable, R prefers to organize those data into what R calls a *data frame*. (In fact, there are times when you're dealing with only one variable that R still prefers to organize those data into a data frame.) A frame is a table-like object (in fact, R often refers to it as a table) in which each column represents a variable and each row an observational unit.

We can build data frames at the command line, using the `data.frame` procedure, and creating a vector for each column. For example, here's a data frame for the start of the seatbelt data.

```
> SeatbeltUsage = data.frame(State=c("Alabama","Alaska","Arizona","Arkansas"),
+   Law=c("Primary","Secondary","Secondary","Secondary"),
+   Compliance=c(81.8,78.4,94.2,68.3))
> SeatbeltUsage
     State       Law Compliance
1  Alabama   Primary       81.8
2   Alaska Secondary       78.4
3  Arizona Secondary       94.2
4 Arkansas Secondary       68.3
```

Of course, that mechanism for building frames is both inconvenient and awkward (since, for example, you enter different variables for the same OU at different places), and almost no one every uses it except for quick tests. Instead, we either read pre-existing data from a file or build the data using a data editor.

To use the data editor, you write

```
> TableName = edit(data.frame())
```

The specifics of the data editor vary from system to system. On our Linux boxes, it's a bit clumsy, but gets the job done.

Once you've built a data frame and want to update it, you use the command `fix(`*`TableName`*`)`. This command brings up the same editor, but on an existing frame. Many R users find it convenient to build a small table at the command line and then update it using `fix`.

A lot of the time, you'll simply want to read data from a file. The `read.table("`*`filename`*`", header=TRUE)` command is the most typical command for reading from a file. If the data are in CSV (comma-separated values) format, as they often are for this class, you use `read.csv` instead of `read.table`.

```
> SeatbeltUsage = read.csv("/home/rebelsky/Stats115/Data/SeatBeltUsage05.csv")
```

So, once you have a frame (that you've created at the command line, that you've built in the data editor, or that you've read from a file), what can you do with it? You can look at the first few lines with `head(`*`table`*`)`.. You can look at the whole thing (not usually advisable) by entering the name of the table. Early on, you are likely to want to extract particular columns from the table to use as vectors. For example, you might do that to analyze them individually. You can extract a column by writing *`Table`*`$`*`ColName`*. For example, we might get the Law column from the SeatbeltUsage frame with `SeatbeltUsage$Law`.

R also contains a wealth of techniques for extracting particular rows from a frame. You can, for example, follow the name of a frame with square brackets and, within those square brackets, give a vector of rows and then a comma. We get rows 1, 2, and 4 of `SeatbeltUsage` with

```
> SeatbeltUsage[c(1,2,4),]
     State       Law Compliance
1  Alabama   Primary       81.8
2   Alaska Secondary       78.4
4 Arkansas Secondary       68.3
```

As importantly, instead of the vector, we can substitute a *predicate* (test) based on some column of the table. For example, the following selects all the rows of the `SeatbeltUsage` table for which the law is Primary. (We use `==` rather than `=` because `=` is used for assigning names to values, and we don't want R to be confused.)

```
> SeatbeltUsage[SeatbeltUsage$Law == "Primary",]
            State     Law Compliance
1         Alabama Primary       81.8
5      California Primary       92.5
7     Connecticut Primary       81.6
8        Delaware Primary       83.8
..
```

We can also use inequality tests to get states with above or below a certain compliance.

```
> SeatbeltUsage[SeatbeltUsage$Compliance < 70,]
              State       Law Compliance
4          Arkansas Secondary       68.3
16           Kansas Secondary       69.0
17         Kentucky Secondary       66.7
21    Massachusetts Secondary       64.8
24      Mississippi Secondary       60.8
NA              <NA>      <NA>         NA
40   South Carolina Secondary       69.7
41     South Dakota Secondary       68.8
NA.1            <NA>      <NA>         NA
```

The two *NA* rows are for the states for which there was no compliance data. (It's a strange quirk of R that the whole row gets turned to NA.) For most of the data we use, we won't have that problem.

Finally, if you've created (or modified) a table in R, you may want to save it for later use. You do so with the `write.table("filename")` command. You can also use the `write.csv("filename")` command.

Exercise 7: Building Frames

a. Using the `data.frame(Name=c(...),...)` construct, create the following data frame, named `GradeBook`.

```
   Name      Year Grade
1 Adams Sophomore     90
2 Brown    Junior     80
3 Davis Sophomore     90
4   Doe Freshling     85
5 Jones    Senior     70
6 Smith    Senior    100
```

b. Create a vector, `Years`, that contains just the years from that frame.

c. Create a summary table from `Years`. (Remember, you use `table(...)` to create a summary table.)

d. Can you find a way to create a summary table of student years directly from `GradeBook`?

e. Edit `GradeBook` using `fix(GradeBook)` and add a few more rows.

f. Repeat steps c and d.

g. Edit `GradeBook` using `fix(GradeBook)` and add another column entitled `Major`. You may choose whatever major you want for our observational units.

h. Build a summary table of majors from the modified `GradeBook`.

i. Write `GradeBook` to a text file and a CSV file using

```
> write.table(GradeBook, "~/Desktop/GradeBook.txt")
> write.csv(GradeBook, "~/Desktop/GradeBook.csv", row.names=FALSE)
```

j. Read each of those files back into variables called, respectively `GB1` and `GB2`.

k. Two files should have appeared on the desktop, corresponding to the names that you gave above (`GradeBook.txt` and `GradeBook.csv`). Double click on each to open it in some editor. What differences do you see between the ways in which you edit the files?

l. Add a few more rows to each file, save it, and then redo step j. Note that when you're editing the CSV file (in Gnumeric on our Linux systems), you'll need to remind it to save the file as CSV.

Exercise 8: Fun with Frames

The file `/home/rebelsky/Stats115/Data/survey-a.csv` contains five of the variables from the introductory statistics survey:

- `Penny`: Choice of abolishing or retaining the penny.
- `States`: Number of states visited.
- `Countries`: Number of countries visited.
- `StatsValue`: Value of statistics on a 1-9 scale.
- `Goal`: Preferred honor (Nobel Prize, Olympic Gold Medal, or Academy Award).

a. Read in that table and store it in a data frame named `Survey`.

```
> Survey = read.csv("/home/rebelsky/Stats115/Data/survey-a.csv", header=TRUE)
```

b. Look at the first few lines of that frame with `head(Survey)`.

c. Write a command to extract rows 5, 10, 15, 20, ... 100. (*Hint:* Think about ways to construct the vector 5, 10, 15, ..., 100.)

d. Write a command to build a bar graph of responses to the question of whether to retain or abolish the penny.

e. Write a command to determine the mean number of states respondents to the survey have visited.

f. Write a command to extract the rows of the table in which the goal variable has the value `"Olympic gold medal"`.

g. Write a command to assign the name `GoldMedalists` to the result of the previous computation.

h. Find the mean number of states that respondents who selected `"Olympic gold medal"` have visited.

Constructing Bar Graphs

Okay, you've constructed a few bar graphs above. Are there general principles you should know? A few. (We'll also cover more principles when you need more information.)

You build bar graphs with the `barplot(...)` command. The simplest bar graphs are constructed with a vector of values. Each value gets a corresponding bar, and the height of the bar depends on the value. For example, the following command builds a bar plot with a bar of height 1, a bar of height 5, and a bar of height 10.

```
> barplot(c(1,5,10))
```

Frequently, you want a title at the top of your bar graph. If you add `main="..."` to the `barplot` command, it will add that string as a title.

```
> barplot(c(1,5,10), main="A Sample Bar Graph")
```

As frequently, you want to add titles to the bars. You must create a vector of strings to use as the titles, and then add them to the plot with `names.arg=...`.

```
> barplot(c(1,5,10), main="A Sample Bar Graph",
+ names.arg=c("Alpha", "Beta", "Gamma"))
```

Once in a while, you'll have a vector of variables that you need to first tally, and then plot. For example, the `Goal` column of the `Survey` frame described above contains three possible factors. We can refer to that column as `Survey$Goal`, tally it with `table`, and then graph it with `barplot`. We should probably add a name, too.

```
> barplot(table(Survey$Goal), main="Goals of Intro Stats Students")
```

Printing Graphs

At times, you will find it useful to print the graphs you create, or to copy them to other applications. Both printing and copying start the same way: You save the graph to a file. That is, create the graph as you normally would. Then, you use a series of arcane commands to build the file. The most important of these commands is `dev.copy`, which copies the graph to a file. You specify the type of file (usually `jpeg`), the file name, and the width and height. You must also follow the `dev.copy` command with `dev.off()`.

For example, the following saves the current image to the file `graph.jpg` on your desktop.

```
> dev.copy(jpeg, filename="~/Desktop/graph.jpg", width=400, height=400)
> dev.off()
```

You can also save to PostScript and a wide variety of other formats.

Exercise 9: Printing

a. Create a bar plot of the goals of introductory statistics students with the following commands.

```
> Survey = read.csv("/home/rebelsky/Stats115/Data/survey-a.csv", header=TRUE)
> barplot(table(Survey$Goal), main="Goals of Intro Stats Students")
```

b. Save that bar plot to your desktop with the following commands.

```
> dev.copy(jpeg, filename="~/Desktop/goals.jpg", width=600, height=400)
> dev.off()
```

c. On your desktop, you should now see a file called `goals.jpg`. Double-click on it.

Constructing Dot Plots

Although *Workshop Statistics* makes a lot of use of dot plots, those kinds of dot plots are not standard in R. Why not? Because there are other ways to represent data, such as histograms, that are often more useful. So why does *Workshop Statistics* use these simple dot plots? Because they're easy to construct by hand.

Fortunately, R is extensible. Hence, some folks who decided that they needed simple dot plots added support for them in R. However, you need to load the BHH2 package, which adds the `dotPlot` command.

If you're working at home, on your own computer, you can probably add the package with the following commands:

```
> install.packages("BHH2")
> library(BHH2)
```

When you enter the first command, R will probably ask you where to look for R packages. We generally use the Iowa State server, which appears in most lists as "USA (Iowa)".

Unfortunately, our Linux computers are not configured to allow you to extend R. Hence, you need a somewhat different command.

```
> library(BHH2, lib="/home/rebelsky/Stats115/Packages")
```

Once you've loaded this library, the `dotPlot` command is available to you.

As in the case of `barplot`, you can call `dotPlot` with just the data you want plotted. (In this case, you don't have to turn it in to a table first.)

```
> dotPlot(Survey$States)
```

Of course, you'll want to add a title to the figure.

```
> dotPlot(Survey$States, main="States visited by Intro Stats Students")
```

One thing that you may notice is that the label on the X axis is a bit awkward. It reads "Survey$States", which is unlikely to be meaningful to anyone but the person who generated the plot. We can change that text by adding `xlab="..."` to the command.

```
> dotPlot(Survey$States, main="States visited by Intro Stats Students",
+ xlab="Number of States")
```

You may also notice that the tick marks on the X axes are somewhat awkward. Unfortunately, it's not nearly so easy to fix those. The solution is to draw the dotplot without any axes and then to add them back in. You tell R not to draw axes by adding `axes=FALSE` to the command. You tell R what ticks to use when drawing a horizontal axis with `axis(1,`*`vector`*`)`, where the vector has the tick marks.

```
> dotPlot(Survey$States, axes=FALSE,
+ main="States visited by Intro Stats Students",
+ xlab="Number of States")
> axis(1, seq(from=0,to=50,by=5))
```

Exercise 10: Dot Plots

a. Repeat the previous set of commands to verify that the plots are created as described. Note that you will need to run the `library` command first. Note also that you will need to have the `Survey` frame defined. See earlier in the lab for how to read in the survey.

b. Create a dot plot for the `StatsValue` column of the `Survey` frame.

c. Create a bar graph for the `StatsValue` column of the `Survey` frame.

d. What are the relative advantages and disadvantages of these two ways of displaying the responses to the value of statistics question?

e. Create a bar graph for the `States` column of the `Survey` frame.

f. What problems, if any, do you think one would have reading the bar graph rather than the dot plot for the number of states visited data?
