Introduction to R (BaRC Hot Topics)

George Bell

September 30, 2011

This document accompanies the slides from BaRC's Introduction to R and shows the use of some simple commands. See the accompanying slides and data files at `http://iona.wi.mit.edu/bio/education/R2011/` or material from other Hot Topics talks at `http://iona.wi.mit.edu/bio/hot_topics/`. All of these commands should work similarly if run on Windows/Mac (using the typical R installation or RStudio) or Linux operating systems (such as tak).

Our example data is the number of tumors in wild-type and knockout mice, each assayed in triplicate. For variable names, we use one-word (no spaces, anyway) names that can include any combination of lowercase and uppercase letters and numbers (although they must begin with a letter). It can be helpful to include dots to make the variables easier to read, like `MGF.ko.trial.1`

R will convert many other special characters (even dashes and underscores) into dots.

1 Entering data by hand

The `c()` (combine) command is used to concatenate several pieces of data into a vector. These can be numbers or text (but the latter must be surrounded by quotes).

```
> # These are the tumor counts for the WT animals.
> wt = c(5, 6, 7)
> # These are the tumor counts for the KO animals.
> ko = c(8, 9, 11)
> wt

[1] 5 6 7

> ko

[1]  8  9 11
```

Note that typing the variable by itself prints the values. The [1] at the beginning of each output row shows that the first value in the row is the first entry in the vector (list). This becomes useful if we have a vector containing more values that can be printed on one line.

We've also included some comments. Whenever R sees a pound sign `#`, it ignores the rest of the line. A liberal use of comments is very helpful to remind you or show others what each command does and perhaps why you chose the specific options you did.

2 Running a statistical test

Let's do some basic statistics like a t-test. In R by default the two-sample t-test is performed without any assumption that the variances of both variables are the same. This is sometimes referred to as Welch's test.

```
> t.test(wt, ko)
```

```
        Welch Two Sample t-test

data:  wt and ko
t = -3.1623, df = 3.448, p-value = 0.04191
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -6.4542173 -0.2124494
sample estimates:
mean of x mean of y
 6.000000  9.333333
```

Unlike Excel, the output for a statistical test provides more information than just the p-value. But what if we don't want to use the default t-test options. How can we find out how to change the default settings? Try entering the command, preceded by a question mark, as in `?t.test`

Running this command on a Windows or Mac computer will usually open a documentation page in your web browser, which can be easier to read than the text-only Unix way of printing documentation.

We can also save the results of a statistical analysis as another variable which will then contain all the information that you saw printed to the screen (and sometimes a lot more). For example, let's run a standard t-test (note the `var.equal=T`, which calculates separate variances for each variable).

```
> wt.vs.ko = t.test(wt, ko, var.equal=T)
> wt.vs.ko

        Two Sample t-test

data:  wt and ko
t = -3.1623, df = 4, p-value = 0.03411
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -6.2599634 -0.4067032
sample estimates:
mean of x mean of y
 6.000000  9.333333
```

If we want to print or save just part of this output, we need to know how to access its different parts. Try the `names()` command to see what each part is called.

```
> names(wt.vs.ko)

[1] "statistic"   "parameter"   "p.value"     "conf.int"    "estimate"
[6] "null.value"  "alternative" "method"      "data.name"
```

Now we can access each part of this variable using the syntax `VARIABLE$PART`, such as

```
> wt.vs.ko$p.value

[1] 0.03410942

> wt.vs.ko$conf.int

[1] -6.2599634 -0.4067032
attr(,"conf.level")
[1] 0.95
```

What if we want to get the commands we used, to put them into a script to document our analyis pipeline? Use the "history" command, as in `history(max.show=Inf)`.

3 Reading data from a file

To get started you probably want to get R and your data in the same place. To get R's working directory, try

```
> getwd()
```

```
[1] "/nfs/BaRC/Hot_Topics_2011-2012/R_Bioconductor/class_1/vignette"
```

To change the working directory on your computer, go to File => Change dir... menu. On tak (or you can do it this way on your computer, too), use the `setwd()` command, like `setwd("/lab/solexa_public/Graceland_Lab/Elvis")`

We can see what files we have in our directory. (You won't have all of these.)

```
> dir()
```

```
 [1] "R_intro_Hot_Topics-017.eps" "R_intro_Hot_Topics-017.pdf"
 [3] "R_intro_Hot_Topics-019.eps" "R_intro_Hot_Topics-019.pdf"
 [5] "R_intro_Hot_Topics.aux"     "R_intro_Hot_Topics.log"
 [7] "R_intro_Hot_Topics.out"     "R_intro_Hot_Topics.pdf"
 [9] "R_intro_Hot_Topics.R"       "R_intro_Hot_Topics.Snw"
[11] "R_intro_Hot_Topics.tex"     "Rplots.pdf"
[13] "Thumbs.db"                  "tumor_boxplot.pdf"
[15] "tumor_boxplot.png"          "tumor_pvals.txt"
[17] "tumors_wt_ko.txt"
```

Let's assume that we have a tab-delimited text file of our dataset ("tumors_wt_ko.txt"), with one column for `wt` and another for `ko`, with column labels in the first row. One way to read the file is with the `read.delim()` command, Note that we've included `header=T` to indicate that the first row has our column names. If we use `header=F` then the columns will just be numbered (and the data will include the first row of the file).

```
> tumors = read.delim("tumors_wt_ko.txt", header=T)
> tumors
```

```
  wt ko
1  5  8
2  6  9
3  7 11
```

Our data looks OK: we have one (labeled) column for each group of mice.

4 Accessing a data matrix

How do we access subsets of `tumors`? We can use column names or row and column numbers.

```
> tumors$wt
```

```
[1] 5 6 7
```

```
> tumors$ko
```

```
[1]  8  9 11
```

```
> # Or subset the matrix with [rows, columns]
> tumors[1:3,1]
```

```
[1] 5 6 7
```

```
> tumors[1:2,1:2]

  wt ko
1  5  8
2  6  9

> # If we're missing rows or columns, R assumes we want them all.
> tumors[,1]

[1] 5 6 7
```

So now we can do the t-test using this set of data.

```
> t.test(tumors$wt, tumors$ko)

        Welch Two Sample t-test

data:  tumors$wt and tumors$ko
t = -3.1623, df = 3.448, p-value = 0.04191
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -6.4542173 -0.2124494
sample estimates:
mean of x mean of y
 6.000000  9.333333
```

5 Creating an output table

Unless we have just a little data, we may want to run several analyses and create a file to hold the output. One way to do that is to create a table the hold the output data. In this case, let's say we want to try two variations of two statistical tests. We should note that normally we'd decide on a statistical test when designing an experiment, rather than trying a bunch of them after we already have the data. Let's start by creating an empty matrix of two rows and two columns, labeled to show what we want each cell to hold.

```
> pvals.out = matrix(data=NA, ncol=2, nrow=2)
> colnames(pvals.out) = c("two.tail", "one.tail")
> rownames(pvals.out) = c("Welch", "Wilcoxon")
> pvals.out

         two.tail one.tail
Welch          NA       NA
Wilcoxon       NA       NA
```

Now let's run some statistical tests and put the output p-values into our table. We'll start with Welch's test for the first row, once as a two-sided test and once as a one-sided test.

```
> pvals.out[1,1] = t.test(tumors$wt, tumors$ko)$p.value
> pvals.out[1,2] = t.test(tumors$wt, tumors$ko, alt="less")$p.value
```

Now let's try the Wilcoxon rank sum test, a non-parametric alternative to the t-test.

```
> pvals.out[2,1] = wilcox.test(tumors$wt, tumors$ko)$p.value
> pvals.out[2,2] = wilcox.test(tumors$wt, tumors$ko, alt="less")$p.value
> pvals.out

           two.tail   one.tail
Welch    0.04191452 0.02095726
Wilcoxon 0.10000000 0.05000000
```

Since we have row names, we can also use them to access subsets of our matrix

```
> pvals.out["Welch",]

  two.tail   one.tail
0.04191452 0.02095726
```

6 Printing an output table

Our table looks fine, except for the fact that we have so many digits that some are meaningless. We could either round them now or later. If we choose to do it now,

```
> pvals.out.rounded = round(pvals.out, 4)
> pvals.out.rounded

         two.tail one.tail
Welch      0.0419    0.021
Wilcoxon   0.1000    0.050
```

Now let's print the table.

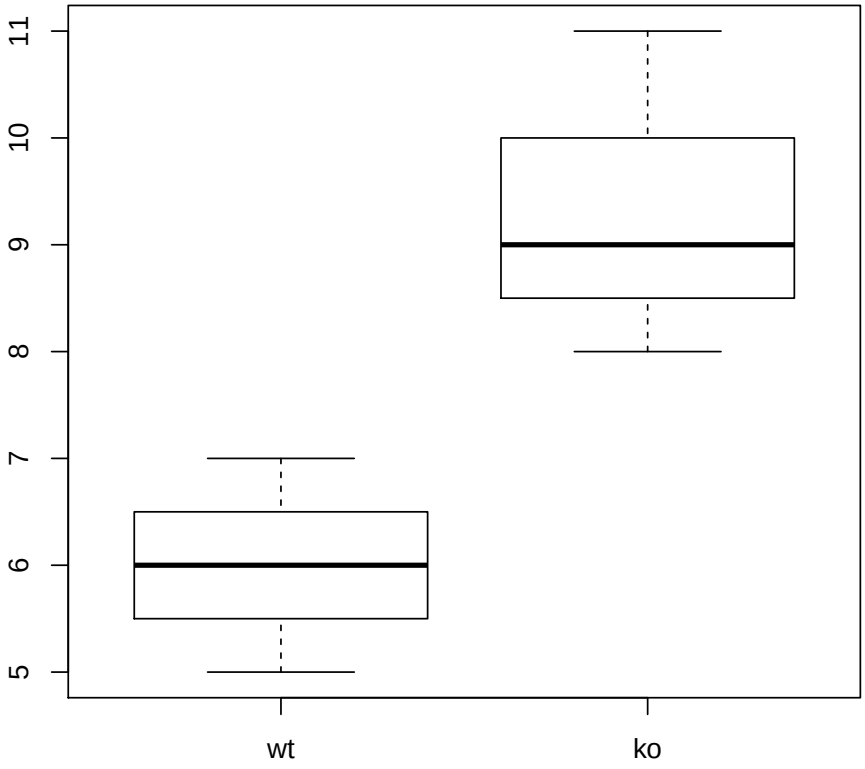
```
> write.table(pvals.out.rounded, file="tumor_pvals.txt", sep="\t", quote=F)
```

We've included our output filename, that we want the file to be tab-delimited (`sep=""`), and that we don't want quotes around any text (`quote=F`). Now we can open `"Tumor_pvals.txt"` in any text editor or Excel. If we do so in Excel, however, our column labels will need to be shifted over one column. To avoid this, we could have created a 3-column table, with a first column containing "Welch" and "Wilcoxon" – and then print the table with the additional option `row.names=F`.

7 Creating figures

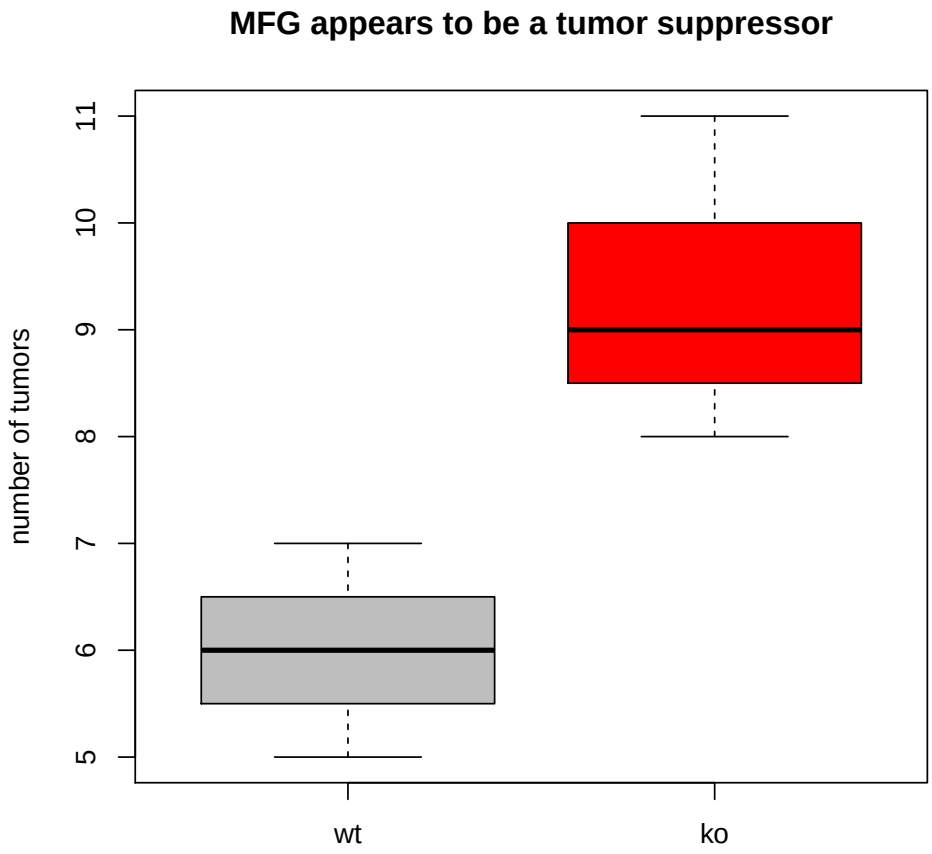
R is very powerful and very flexible with its figure generation. Besides statistics, graphics are another great reason to learn R. We can start by creating a simple boxplot of our data.

```
> boxplot(tumors)
```



Normally if we only had three points of data, a scatterplot would be more informative than a boxplot. Nevertheless, let's add some more details the make the figure more readable.

```
> boxplot(tumors, col=c("gray", "red"), main="MFG appears to be a tumor suppressor",
        ylab="number of tumors")
```



If we run R on our computer, a figure will pop up and can then be saved by right-clicking on it. On tak, a figure will end up in a file called `"Rplots.pdf"`.

Normally it's more useful to include filenames in our code, as this makes it more reproducible and more automated. To do this, we need to first issue a command telling R what graphical format to use. Then we write our plotting command(s). A PDF file can hold multiple figues, by default one per page. Finally, we need to "close" the file when we're finished with it.

```
> # Create a PDF file 11 inches wide and 8.5 inches high
> pdf("tumor_boxplot.pdf", w=11, h=8.5)
> boxplot(tumors)
> dev.off()

pdf
  2
```

Another common format is PNG. The syntax is similar, except the figure dimensions are in pixels rather than inches.

```
> png("tumor_boxplot.png", w=1800, h=1200)
> boxplot(tumors)
> dev.off()

pdf
  2
```

What variables do we have so far?

```
> ls()
```

```
[1] "ko"                "pvals.out"         "pvals.out.rounded"
[4] "tumors"            "wt"                "wt.vs.ko"
```

And what files have we created?

```
> dir(pattern="tumor*")
```

```
[1] "tumor_boxplot.pdf" "tumor_boxplot.png" "tumor_pvals.txt"
[4] "tumors_wt_ko.txt"
```

In the interests of totally reproducible research, we'll end by printing our R environment.

```
> sessionInfo()
```

```
R version 2.12.1 (2010-12-16)
Platform: x86_64-pc-linux-gnu (64-bit)

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=C              LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

loaded via a namespace (and not attached):
[1] tools_2.12.1
```

This is just a start! There's lots more.... To exit R, type `q()`