STAT 5570 / 6570: Statistical Bioinformatics
Notes 1.2: Installing and Using R

In this course we will be using R (for Windows) for most of our work. These notes are to help students install R and then run a few simple commands. As of the beginning of the Spring 2014 semester, version 3.0.2 is the most recent (but slightly older versions should be fine). If you choose to install or use R in a non-Windows environment, you will be responsible for all non-Windows issues that arise.

1 Installing the most current version of R

Run the executable file available at the following website:

`http://cran.r-project.org/bin/windows/base/release.htm`

Follow the installation wizard that pops up. Note that you will probably choose to follow the English directions and accept the installation agreement. It's probably best to select the default "User installation" rather than customizing. If you want, you can put R on the start menu and create a desktop icon. This will all take just a few minutes.

It is not expected that students have their own personal or laptop computers in this class. You can install on a flash drive or even a rewritable CD, and then run R on any Windows machine (such as in the student computer labs). This just requires a few extra steps, easily available upon request to Dr. Stevens.

2 Running R: a simple example

You can run R for Windows by either double-clicking on the desktop icon or the shortcut you created to access the `bin\Rgui.exe` file in the folder created during installation. This will open up an interactive environment where we will be using many features of R. Use the "Introduction to R" file available through the course website as a good reference for the basic syntax of common commands in R. The search facility through CRAN (see course website) is also very useful. There are other interfaces to R for Windows (such as Tinn-R and some online resources), but they are unnecessary for our purposes in this course.

At the left side of each line of code presented here, `>` is the R command prompt; it is not part of the actual R code we type. Output here is boxed for convenience, but it is not boxed in R.

2.1 Example data

In a specific poultry population, a certain cell condition is not uncommon in bile ducts. Subjects were randomly assigned to one of four treatment groups (labelled I-IV).

After a predetermined period, each subject was evaluated for the cell condition on the following numeric scale:

1. Normal
2. Very Mild – rare
3. Mild – evident but not widespread
4. Moderate – evident in most areas
5. Severe – very evident in all areas

The scores at right were recorded for the four groups.

Group I	Group II	Group III	Group IV
1	5	1	3
1	3	1	2
1	5	1	3
1	3	1	2
1	2	1	3
1	3	1	4
1	2	1	3
1	4	1	3
1	3	1	2
1		1	5
			4

We can type these data in "by hand" by creating several objects, assigning them names (via `<-`), and piecing them together:

```
> g1 <- rep(1,10)
> g2 <- c(5,3,5,3,2,3,2,4,3)
> g3 <- rep(1,10)
> g4 <- c(3,2,3,2,3,4,3,3,2,5,4)
> score <- c(g1,g2,g3,g4)
> group <- c(rep(1,10),rep(2,9),rep(3,10),rep(4,11))
```

Note that the code `rep(1,10)` means to repeat the value 1 10 times. The function `c()` can be used to concatenate objects; when those objects are numbers, the result can be considered a vector. The following code makes the boxplot in Figure 1 to visualize the data:

```
> boxplot(score~group,xlab="Group",ylab="Score",cex.lab=1.5,cex.axis=1.5)
```

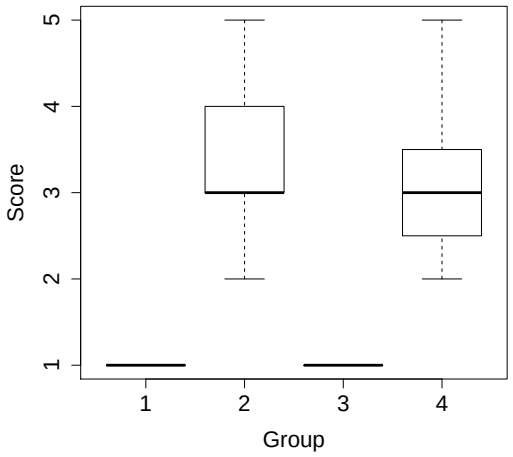

Figure 1: A visual summary of the distribution of score by group.

2.2 ANOVA Model

Now suppose we want to compare the mean in all four groups with an ANOVA model. That is, we want to fit the linear model

$$Y_{ij} = \mu_i + \epsilon_{ij}, \quad (1)$$

where $i = 1, \ldots, 4$, ϵ_{ij} are i.i.d. $N(0, \sigma^2)$, and we are interested in testing $H_0 : \mu_1 = \ldots = \mu_4$.

We can search the help documentation to find which function to use:

```
> help.search("Analysis of Variance")
```

In the resulting browser tab, we see (among other things) the following:

```
stats::aov              Fit an Analysis of Variance Model
stats::manova           Multivariate Analysis of Variance
stats::summary.aov      Summarize an Analysis of Variance Model
stats::summary.manova   Summary Method for Multivariate Analysis of Variance
```

Since we want to "Fit an Analysis of Variance Model", it looks like we can use the `aov` function from the `stats` package (more about packages later; for now, just know that packages contain specialized functions, and there are some packages that come with R by default, while others we'll need to download from certain repositories). We can use the `?` utility to learn more about this function, as in `?aov`. The resulting help file (which also opens in a browser) describes the function and gives examples of how to call it. We can do this for our data, and create an object called `result`. We can see this object by typing just its name:

```
> result <- aov(score~group)
> result
```

```
Call:
   aov(formula = score ~ group)

Terms:
                   group Residuals
Sum of Squares   9.15684  58.44316
Deg. of Freedom        1        38

Residual standard error: 1.240152
Estimated effects may be unbalanced
```

From this (notice the degrees of freedom for `group`), it looks like rather than the ANOVA model in Equation 1 above, we have instead fit the linear regression model

$$Score_i = \beta_0 + \beta_1 Group_i + \epsilon_i. \quad (2)$$

We need to try the `aov` function again, making sure that R knows `group` is not a numeric predictor variable. In R, the analog to SAS's "CLASS" statement is to make a numeric variable into either a "factor" or a "character" variable.

```
> fac.group <- as.factor(group)
> result <- aov(score~fac.group)
> result
```

```
Call:
   aov(formula = score ~ fac.group)

Terms:
                fac.group Residuals
Sum of Squares   48.69091  18.90909
Deg. of Freedom         3        36

Residual standard error: 0.7247431
Estimated effects may be unbalanced
```

We can call the `anova` function to get the ANOVA table for a fitted model object, and see what named components of the object exist.

```
> anova(result)
```

```
Analysis of Variance Table

Response: score
          Df Sum Sq Mean Sq F value    Pr(>F)
fac.group  3 48.691 16.2303    30.9 4.604e-10 ***
Residuals 36 18.909  0.5253
---
Signif. codes:  0 Ś***Š 0.001 Ś**Š 0.01 Ś*Š 0.05 Ś.Š 0.1 Ś Š 1
```

```
> names(result)
```

```
 [1] "coefficients"  "residuals"     "effects"       "rank"
 [5] "fitted.values" "assign"        "qr"            "df.residual"
 [9] "contrasts"     "xlevels"       "call"          "terms"
[13] "model"
```

We will use the square brackets [ ] to subset objects in R. For example, suppose we wanted to look at the first five elements of our `g4` object:

```
> g4[1:5]
```

```
[1] 3 2 3 2 3
```

We might also want to look at the elements of `g4` where `g2=3`:

```
> g4[g2==3]
```

```
[1] 2 2 4 2 4
```

If `A` is a matrix, then `A[i,j]` is the element in row `i` and column `j` of the matrix. We can use this if we want to extract the p-value for testing $H_0 : \mu_1 = \ldots = \mu_4$:

```
> anova.table <- as.matrix(anova(result))
> anova.table
```

```
          Df   Sum Sq    Mean Sq F value       Pr(>F)
fac.group  3 48.69091 16.2303030    30.9 4.603955e-10
Residuals 36 18.90909  0.5252525      NA           NA
```

```
> anova.table[1,5]
```

```
[1] 4.603955e-10
```

Note that `NA` refers to missing values in R.

2.3 Graphical diagnostics

We can use graphical diagnostics to check the major model assumptions of Equation 1. We can "point" to named components of an object using the `$` operator. Figure 2 shows the residual-vs-predicted plot as well as the normal quantile plot of the residuals.

```
> par(mfrow=c(1,2))  # put figures in a 1-by-2 layout
> resid <- result$residuals     # point to residuals within result object
> yhat <- result$fitted.values  # point to fitted.values within result object
> plot(yhat,resid)  # first argument is horizontal axis, second is vertical
> qqnorm(resid); qqline(resid)
```

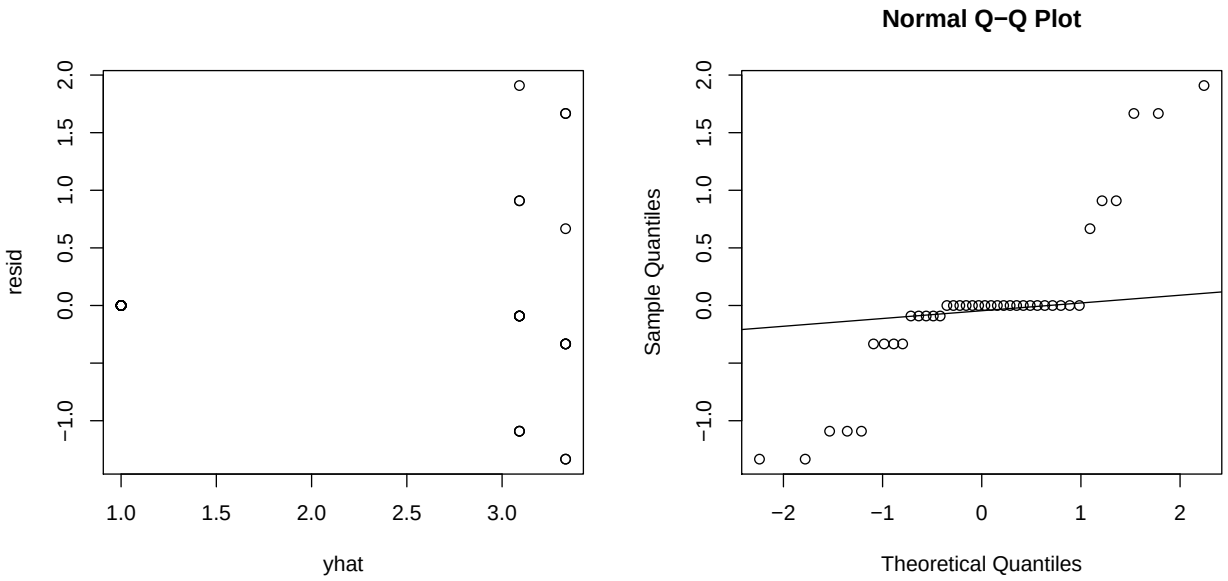

Figure 2: Two graphical diagnostics of the fitted model.

Note that the pound sign (`#`) comments out the rest of a line, and the semicolon (`;`) can be used to put multiple commands on the same line.

Clearly there are some major problems with this model fit. It turns out that the ANOVA approach to these data is completely wrong. For one thing, the observed data (Y, the cell condition) are not continuous, and there is zero variance in two of the groups.

2.4 "Programming" in R

It is straightforward to "program" in R using such approaches as for loops and if..then statements. For example, suppose that we wanted to define a new variable called `high` that is 1 if the animal has the highest `score` value, and 0 otherwise. There are several ways to do this; here are two:

```
> ## Approach 1 - just to show if..then and for loops
> high <- rep(0,length(score))
> # So high is a vector of 0s repeated 5 times
> for(i in 1:5)
    {
      if(score[i]==max(score))
        {
          high[i] <- 1
         }
    }

> ## Approach 2 - just to show T/F vector construction
> high <- rep(0,length(score)); t <- score==max(score)
> t
```

```
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
[13]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[25] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[37] FALSE FALSE  TRUE FALSE
```

```
> high[t] <- 1
```

Note that in general, explicit for loops and if..then statements are relatively inefficient (time-wise). Things will usually work fastest if we can do them with straightforward matrix manipulations.

We can also write functions to do specific things. Here is a simple function (called sq.med) that takes a vector and returns the square of the median of the vector's values:

```
> sq.med <- function(v)
   {
        med <- median(v)
        med.squared <- med^2
        return(med.squared)
     }
> # Call this function:
> sq.med(score)
```

```
[1] 2.25
```

3 Misc. Notes

- Code you type in the R Console window is not saved. For this reason, it is usually most convenient (and safe) to write your R commands in a text file (so you can save them; a common extension is .R), and then use one of the following:
  - Open your .R file in R as a script, and run individual lines or highlighted sections with CTRL-R.
  - With your .R file open in Notepad (or similar), copy and paste the code to R.
  - With your .R file open in Notepad++, and with NppToR installed (from sourceforge.net), run individual lines or highlighted sections with F8.
  - Use a different front-end to R, such as RStudio or Tinn-R, where your code is in an R Script (saved as a .R file). Run lines or sections with CTRL-R.

  However you do it, you need to be careful to save your code if you want to reproduce any results.

- Plots can be copied and pasted to word processing documents.

- To run a previous line of code in R, use the up-arrow until the desired line is displayed, then press enter.

- To clear the R Console window, press CTRL-L.

- If you have saved R commands in a text file, you can run the entire file using the `source` command in R, like this for a certain code chunk:

  ```
  > source("http://www.stat.usu.edu/jrstevens/stat5570/Rintro.R")
  ```

- To quit R (and not save any of the objects created during this session), type `q("no")`. Alternatively, you can click on the red 'X' as in other Windows applications. R will ask if you want to save your workspace. This is NOT the same as saving your code – it only saves the objects defined in your current R session. This is probably only useful if some of the objects you created would require substantial computation time to re-create. (So you will usually say 'No'.)

- If the objects created in your R session would require non-trivial computational time to re-create, you can save them for a later R session:

  ```
  > save(list=ls(),file="C:\\folder\\temp.Rdata")
  ```

  This .Rdata file is referred to as a workspace, and it contains all objects defined in the R session. Then you could load them in a later R session:

  ```
  > load("C:\\folder\\temp.Rdata")
  ```