R Basics I

R. Montgomerie, Queen's University

This is a guide to R for the complete novice. My philosophy here is to tell you only about things that I use often, and to tell you only one way to perform each task. One of the hallmarks of R—and a frustration for the beginner—is that there are many ways to do exactly the same thing in R. Some of the procedures and practices I describe below would be scoffed at by the seasoned R user, but they work and are easiest for the beginner, in my opinion.

I begin with a brief background and overview then the rest of this guide is in 2 parts, each designed to be covered in a 2-hour workshop 1-2 weeks apart. In this guide, information that you type into the console is in blue text, and results that are output on the console are in brown text, both shown with a pale green background, like this where I typed 1+2 and R says it equals 3 (notice the number in [] which is the i^{th} item of the vector shown on the screen):

```
> 1+2
> [1] 3
```

R functions and commands etc in the text of this guide are shown in **bold blue font**.

Though R works on any platform, the interfaces for Mac OSX and Windows are slightly different. This guide shows the OSX version in the Overview section below, but I will add an appendix showing the Windows version as soon as I get a chance.

Background

R is an object-oriented programming language that can be used as a statistical and graphics application. R was created by Ross Ihaka and Robert Gentleman in 1996 as an an open-source implementation of the S Language that was developed earlier at the Bell Labs—S is the basis for S-Plus which is commercially available (and expensive). R is freely available at the CRAN (Comprehensive R Archive Network) website and is updated frequently.

A couple of years ago, James Holland Jones published the following list of 7 reasons to use R, on his Monkey's Uncle (http://monkeysuncle.stanford.edu/?p=367#more-367) blog, and I agree. Have at look at the blog article for more details:

(1) R is what is used by the majority of academic statisticians.

(2) R is effectively platform independent. If you live in an all-windows environment, this may not be such a big deal but for those of us who use Linux/Mac and work with people who use windows, it's a tremendous advantage.

(3) R has unrivaled help resources. There is absolutely nothing like it.

(4) R makes the best graphics. Full stop.

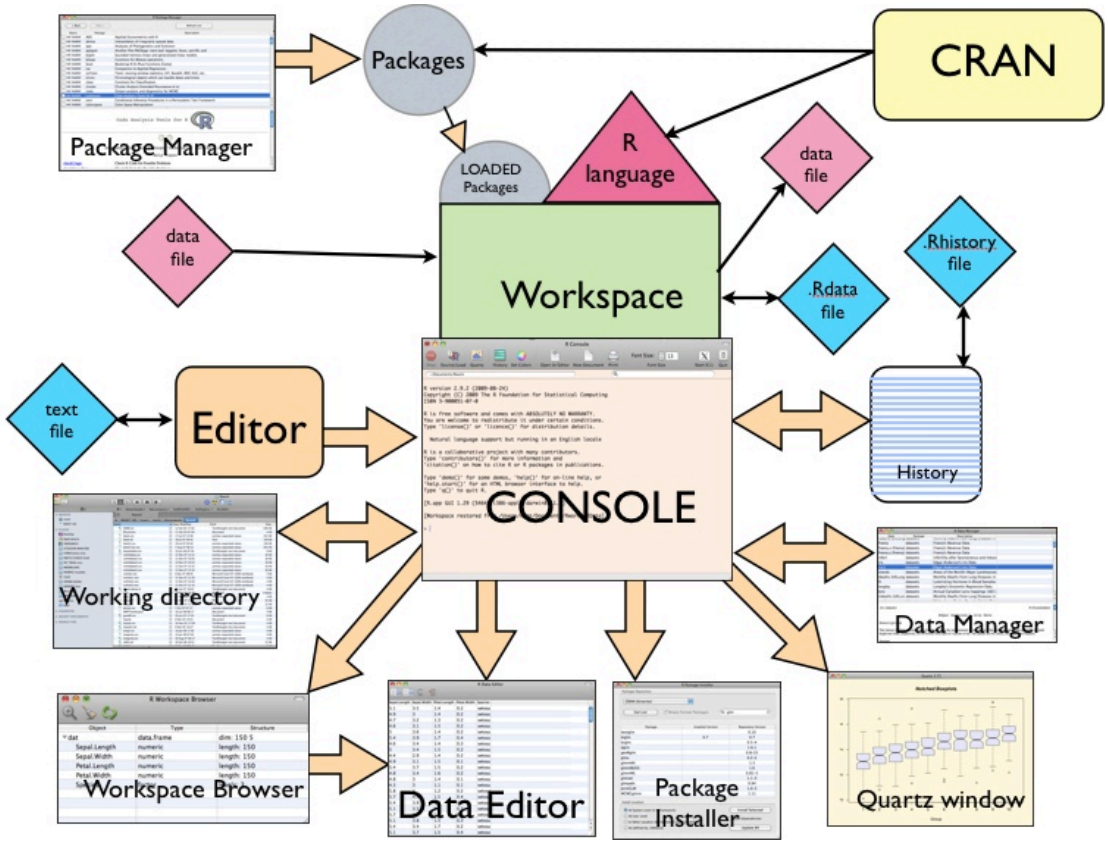
(5) The command-line interface — perhaps counterintuitively — is much, much better for teaching. You can post your code exactly and students can reproduce your work exactly.

(6) R is more than a statistics application. It is a full programming language.

(7) The online distribution system beats anything out there.

Overview

The diagram below shows the relation between the R programming language and various internal and external components, all described below:



While this probably looks a bit daunting, I think it helps to understand what is linked to the console (fat arrows) and what is loaded from elsewhere (thin arrows). The console is what you will see and work with most of the time.

Console

This is your main window that you see when you open the R application, where you will type commands for R to implement, and where you will see the text responses to those commands. The console has a toolbar at the top for quick access to some commands.

R Language

This is the (invisible) compiler that is the heart and soul, or more correctly the brain, of R.

Working Directory

This is the default folder (directory)where R will look for files that you specify in commands, unless you give the full path to a file. Set this in the Preferences.

Packages

R is complemented by packages that contain specific functions and datasets that you can use once a Package is loaded. Many packages are built in and are immediately loaded when you download and install R; others can be loaded as needed; still others can be downloaded and installed into R, then loaded as needed. See http://CRAN.R-project.org for a long list of available packages.

Editor
This is an internal module to compose and edit R commands and scripts, and then to save them in a text file for later use. The R editor highlights functions, commands and other things in different colours and automatically closes brackets for you etc. You can also use an external program (that you specify in the Preferences) as an editor, if you prefer. R is easiest to work with if you do all your basic work in an editor, then transfer commands to the console, then save your editor files for easy later reference.

Package Installer
Use this to download new packages so that they can be loaded as needed using the Package Manager.

Quartz Window
All of your graphs will appear here. Size and shape can be adjusted by just changing the window size using the handle in the lower right corner of the window. You can save the window as a pdf file that can be opened and edited in Adobe Illustrator or other vector drawing programs.

Package Manager
Use this to load the packages that have been downloaded and installed. You can also use the Package Manager to view help files for each package.

Data Manager
Lets you see all datasets and variables in your workspace, including in loaded Packages.

Data Editor
This is a window where you can edit your data. Use **fix(dat)** to see your data in the Data Editor, where **dat** is the name of a dataset object in your workspace (note that I use **dat** for the generic name of your dataset throughout this document)

Workspace
This is an invisible place where all of the work gets done, using the R Language to manipulate objects. When you are working in an R session, all of the objects that you have loaded or created are stored in a workspace that can be saved at the end of your session. You can also set a Preference so that R always saves the current workspace when you close R, then opens that same workspace when you re-open R.

Workspace Browser
Use this menu item to see a listing of the objects in the current workspace in spreadsheet format. You can use the toolbar on this window to delete objects or edit them, which will open the Data Editor window.

History
Use the History icon in the toolbar to see all the commands used since you last cleared the history. Double click on a command in this list to have it entered anew at the prompt. You can also load previous history that you might have saved.

External Files
These are text files containing data, commands, functions, history or other information that is either saved during an R session or read into the workspace during a session. These are shown as diamonds in the diagram

Session 1: Getting started

1.1 Download and Install

Download and install R using the 'R-2.13.0.dmg (latest version)' file at the Mac OSX page under http://cran.r-project.org/ The number after the R- will change periodically as the program is updated. If you have previously installed R on your computer using this method, you can download and install using the 'R-2.13.0-mini.dmg' file instead. Installing R will put the app in your Applications folder and a bunch of other things in the Library folder.

1.2 Startup and Customize the Working Environment

Double click on the R icon to start it up. If you see any warning messages you should probably quit, restart your computer and start R again.

You could just start using R but your experience will be better if you customize the working environment to your personal taste. First, customize the toolbar by control clicking on it and choosing Customize Toolbar and setting it up the way you like it. Here's mine:

Second, select the R>Preferences menu item and customize each of the settings. I like to make the background colour pale brown to make it easy to distinguish from other windows on my screen, and I like a bigger font than the default. Your mileage may vary. Come back here after you have used R a bit and understand better what all those things mean.

1.3 RStudio

RStudio (http://) is an open source, freely available IDE (independent development environment) for R, that makes R much easier to use. When you run RStudio it opens the R console in one of its panes (no need to also run R) and provides other convenient panes for all of the other useful R windows, all in a single large window. RStudio works best with large monitors and tends to run a little slower than R itself but it can be a tremendous time-saver overall. Most useful, I think, it will save and reopen, automatically any editor files that you open to show your lists of useful commands etc.

The rest of this document assumes you are working in the RStudio environment,

1.4 Basic Information and Commands

- each command entered by hitting <return> is executed immediately and often generates a response; commands and responses etc can be shown in different colours (see Preferences) and this really helps when you are trying to find things on the console after a long session
- R is case sensitive, so **House** is a different from **house**
- **>** is the prompt from R on the console, indicating that it is waiting for you to enter something
- **citation()** tells you how to cite R in your paper
- when you hit <return> before a command is completely typed a **+** will appear at the

beginning of the next line, to remind you that more is needed

- separate commands on the same line with a semicolon
- don't worry about spaces between words and symbols in a typed line; they do not have any effect, so **win=3+4** is the same as **win = 3 + 4**
- **#** begins a comment and these can either follow a command, or begin a new line
- R functions are followed by **()** with or without something inside the brackets
- the console and editor will automatically close your brackets and quotes; this is very useful but can sometimes be annoying when you are editing a previous line of typing
- **=** is the assignment operator; **xxx = yyy** assigns **yyy** to a new object **xxx; most advanced** R users and books use **<-** but this is (almost) equivalent to **=** and requires two keystrokes so I prefer to use **=**
- **names(dat)** gives you an ordered list of the variable (column) names in your data frame
- the up arrow ↑ takes you to the previously entered command and scrolls back through previous commands that you have entered on the console each time you press it; you can either press <return> to execute the command again or edit it to correct a mistake etc. Very handy.

1.5 R Objects and Variables

R is an object-oriented programming language. Thus the entities that R creates and manipulates are called 'objects', and these can be variables, arrays of numbers, character strings, functions, or more general structures built from such components like the entire output of a statistical analysis. To create an object just type its name and assign something to it, like:

```
> dat = 1+2 # makes dat the number 3

> my.data = read.csv(file.choose()) # a data frame created from the data in a file that you navigate to and open

> MyExcellentResults = lm(my.data$y ~ my.data$x) # the complete regression output from a regression analysis of the variables y on x in the object (data.frame) my.data
```

Each of the items on the left of the **=** sign is an object that is assigned the stuff to the right of the equals sign; on each line the stuff after the # is a comment that describes what each object is. to see what's in an object just type its name and hit <return>. You can call an object anything you like EXCEPT one of the reserved words or function in R. There are lots of these reserved words so it's best to make odd names for objects. If you use one of the reserved words as an object name you'll get a syntax error. R experts seem to like object names with dots in them to separate words, but I do not really see the point of this.

Variables are objects that are either vectors that you have entered yourself or columns (which are vectors, of course) in a data frame. Specify variables like this **my.data$Spermlength** where **my.data** is the name of the data frame and **SpermLength** is the exact name of the

column (variable) of interest, and the **$** sign separates the two. Because typing all this can be tedious it's good to keep the names of your data frames and variables short, and make sure your variable names do not contain a space.

Note that you need to specify both the data frame and the variable name every time you use a variable. This can be done either as **my.data$Spermlength** or by using the **with()** command as in **with(my.data, plot (x,y))**.

Use **ls()** to display the names of (most of) the objects that are currently stored in the active workspace. To remove each object from the workspace use **rm(dat)**, or delete using the Workspace Pane. To remove all objects from the workspace use **rm(list=ls())**.

1.6 Data Input

You can either create datasets using R—which I would not recommend except for very small datasets—or import them.

Creating Datasets

Create datasets in R as follows:

- to create vectors: e.g., **>bob = c(12,13,14,15,16)** puts these numbers into a vector 'bob'
- to create matrices: e.g., **>andy = matrix(c(1,2,3, 11,12,13), nrow = 2, ncol=3, byrow=TRUE, dimnames = list(c("row1", "row2"), c("A", "B", "C")))** makes the following matrix:

	A	B	C
row1	1	2	3
row2	11	12	13

  where most of this syntax should be fairly obvious, except that **c()** means combine the stuff inside the brackets into a vector, **dimnames** defines the row and column names, and **list()** sets up the list of row and column names. Remember to put words you want to use in quotes, otherwise R might think these are commands or functions.

Data frames are matrices with row and column labels. The columns can be of different types (e.g., numeric, categorical, etc) whereas matrices contain only one data type. Two examples:

```
> L3 = LETTERS[1:3]
> d = data.frame(cbind(x=1, y=1:10), fac=sample(L3, 10, repl=TRUE))
```

where:

L3 is a vector of upper case letters from 1-3 in the alphabet; LETTERS is a built-in constant referring to upper case letters, and 1:3 means 'from 1 to 3'

data.frame() creates a data frame

cbind(x=1, y=1:10) binds together, by column, the two vectors assigned inside the brackets

fac=sample(L3, 10, repl=TRUE) creates a variable called 'fac' calculated using the **sample(x)** function which takes a sample of the specified size from the elements defined by x using either with **(repl=TRUE)** or without **(repl=FALSE)** replacement.

Importing Datasets

Alternatively, and most of the time, you can import datasets into the R workspace using the Data Manager to get data from the installed packages, or using the following syntax:

```
> dat = read.csv(file.choose())
```

where

dat is the name of the object where you want to store this file

read.csv tells R that the file to be read in is a csv (comma-separated variable) file

file.choose() tells R to open an Open/Save dialog box so you can find the file

To see and change this dataset in a spreadsheet format, use **fix(dat),** or **View(dat)** or click on the object name in the `Workspace Pane.

To see all of the built-in datasets, use **data()**, which will open a new tab in the Source Pane called 'R data sets' [OR use **Packages & Data > Data Manager** to show a different window called 'R Data Manager']. These built-in datasets are very handy for testing your stats commands and programs, especially as many of them are standards for stats textbooks. You can just refer to the datasets by name if their package is loaded (use Package Manager to see if a package is loaded, and simply click the check box to load ones you want available in the workspace.

1.7 Data Manipulation

My philosophy and general practice is to do ALL data manipulation in Excel, or NeoOffice, or JMP, or Numbers, and to save the spreadsheets from those applications as a .csv file that I then import into R. This has, for me, several advantages compared to manipulating your data within R: (i) you can keep the spreadsheet open so that you can readily see what your data look like—in R you have to either enter the dataset name and then see the data on the console, which for big datasets can be very confusing, or to look at your data using the R Data Editor but then you have to close the editor before you can use the console again, (ii) you can directly see your manipulations and so reduce the chance that you make a mistake, and (iii) you don't have to save the dataset from R after it's manipulated so you never accidentally lose a manipulated dataset. So I do all transformations, sorting, data entry, formulae to make new columns, subsetting, and column naming in my spreadsheet program, and I don't then manipulate it at all when it's in R.

Having said that, it's sometimes useful to be able to refer to subsets of your data and that can be done by using **bobs.normal[xx:yy]** to select elements xx-yy in the vector bobs.normal, or make a vector if you need to select some nonconsecutive elements from the vector, like **bobs.normal[c(44,55, 61:70)]**, which takes the 44th, 55th, and 61st-70th elements. For arrays, use the same syntax in general except that rows and columns are separated by a comma as in **bobs.array[13:17, 3:8]**, with rows first and columns after the comma.

1.8 Plotting Graphs

R can draw all the basic graphs used by scientists and will put them in the Plots pane from where you can save them so they can be opened and further tweaked in your favourite drawing application. Enter **demo(graphics)** in the console and press <return> to see a selection of graph examples in the Plots pane, and the code that produced them in the console.

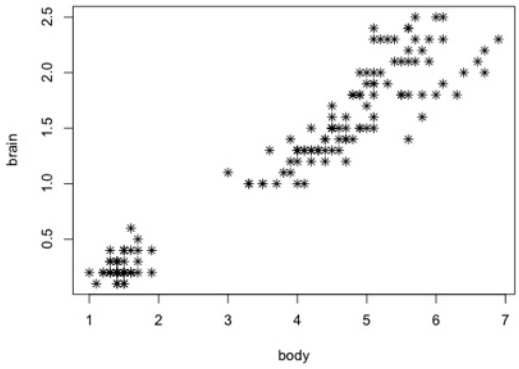
The simplest graphs just use the defaults in R:

```
> plot(x,y) #gives a scatterplot of the variable y on the variable x
> plot(y,x) #also gives a scatterplot of the variable y on the variable x
> boxplot(x) #gives a boxplot of the variable x
> boxplot(x, y) #gives a boxplot of the variables x and y side by side
> boxplot(x~a) #gives a boxplot of the variable x at each a (categorical)
```

You can change just about every aspect of a graph to suit your needs. For example:

```
> plot(x,y,xlab="body",ylab="brain",pch=8) #try different numbers instead of 8
to see the different plot symbols
```

gives



where **xlab** and **ylab** are axis labels, **pch** is the type of data point (* here),

The details of how your graph will look are determined by both high and low level functions. Low level commands are included inside the brackets of any of the graphing functions [e.g., **plot(), hist(), boxplot(),** etc.], whereas high level commands are set for all subsequent graphs drawn within a given session. High level graph functions are set by the function **par()** and can be set and reset any time during an R session. Enter **par()** to list all of the high level commands and their current values.

Graphs are drawn relative to the Plots window so if you want to make your graphs a different size or shape, the easiest thing to do is to adjust the size and shape of the Plots pane when you Export your graph. To look at graphs previously plotted in the current session, use ← and → above the Plots pane

1.9 Simple Statistical Analysis

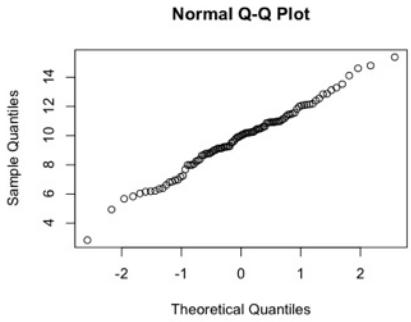
Note that I use the with function in these examples to specify the dataset (dat frame) name, as I think this is generally most easy and good practice (see 1.4). Below I also use **dat** as the generic name for my data frame, when needed.

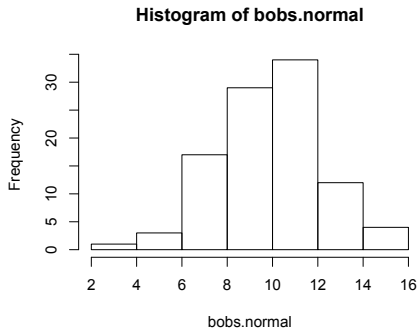
Distributions

R can be used to generate distributions, which can be very useful in both modeling and statistical analysis. For example:

```
> bobs.normal = rnorm(100, mean=10, sd=2.5) #generates 100 numbers from a normal distrib with mean 10 and SD 2.5 and stores those numbers in a vector in the object bobs.normal
> hist(bobs.normal) #plots histogram of those generated data
> bobs.binomial = binomial0.5 = rbinom(100, 20, 0.5) # as above
> bobs.poisson10 = rpois(100, 10)
> qqnorm(normal) #plots a quantile-quantile plot of the data in bobs.normal; useful
```

and here are those plots:





To test if a distribution is normal use **shapiro.test(bobs.normal)** and get this output:

```
Shapiro-Wilk normality test
data:  bobs.normal
W = 0.9894, p-value = 0.6149
```

Descriptive Stats

The basic, and self-explanatory, functions for descriptive statistics are: **mean(x)**, **var(x)**, **sd (x)**, **median(x)**

In addition:

- **summary(filename)** gives the min, first quartile (25th percentile), median (50th percentile), third quartile (75th percentile), and max of all variables in a file named **filename** in the workspace
- **fivenum(x)** creates a vector continuing the min, lower hinge (the median of the numbers from the min to the median), median (50th percentile), upper hinge (the median of the numbers from the in median to the max), and max of the object **x** [NOTE that this is slightly different from what the summary gives you.

and here are the outputs (note I have my prefs set so that what I type is blue, and output is brown:

```
> summary(bobs.normal)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  2.844   8.571  10.010   9.771  11.020  15.370
> fivenum(bobs.normal)
[1]  2.843724  8.504896 10.010404 11.050354 15.373265
```

Two-sample comparisons

For parametric analyses use:

```
> with(dat, t.test(x,y, var.equal = TRUE)) #a simple t-test comparing x and y, assuming equal variances; note that you can just use T instead of TRUE
> with(dat, t.test(x, y)) #a t-test assuming unequal variances (Welch's test)
> with(dat, t.test(x,y,paired = T)) #paired t-test
```

NOTE. in the preceding. that I am using the **with()** command to specify which data frame contains the variables (**x** and **y**) used in the **t.test** function. This my preferred method and I use it through the rest of this workshop. Here is an example of output:

```
> with(dat, t.test(x,y, var.equal = TRUE))

        Two Sample t-test

data:  x and y
t = 16.2974, df = 298, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 2.249700 2.867633
sample estimates:
mean of x mean of y
```

For nonparametric analyses use:

```
> with(dat, wilcox.test(x, y))
        Wilcoxon rank sum test with continuity correction
data:  x and y
W = 19348.5, p-value < 2.2e-16
alternative hypothesis: true location shift is not equal to 0
> with(dat, wilcox.test(x, y, paired = TRUE))
        Wilcoxon signed rank test with continuity correction
data:  x and y
V = 11325, p-value < 2.2e-16
alternative hypothesis: true location shift is not equal to 0
```

NOTE: that (continuity correction is default but can be turned off by using correct=FALSE

ANOVA

For very simple anova use:

```
> with(dat, aov(formula))
```

where **formula** is in the form response ~ predictor(s) and **dat** is the name of the data frame where these variables can be found. And this is what you get:

```
> with(iris, aov(Sepal.Length~Species))
Call:
   aov(formula = Sepal.Length ~ Species)

Terms:
                Species Residuals
Sum of Squares  63.21213  38.95620
Deg. of Freedom       2       147

Residual standard error: 0.5147894
```

some more complicated models etc:

```
> with(dat, aov(y~a*b)) #full model including interaction

> with(dat, aov(y~a+b)) #as above but no interaction term; note that the + sign
here just means 'and' and no sign is implied in the model

> with(dat, aov(y~a+Error(b))) #repeated measures with b as the blocking variable

> with(dat, aov(y~as.factor(a))) #ensures that a is treated as a factor; especially
important when a is a numeric variable where the numbers specify different levels

> my.results = with(dat, aov(y~a*b))
> plot(my.results) #provides 4 diagnostic graphs for the anova in the previous line
```

For generalized linear models use:

```
> with(dat, glm(Sepal.Length~Species)
Call:  glm(formula = Sepal.Length ~ Species)

Coefficients:
     (Intercept)  Speciesversicolor   Speciesvirginica
           5.006             0.930             1.582

Degrees of Freedom: 149 Total (i.e. Null);  147 Residual
Null Deviance:            102.2
Residual Deviance: 38.96    AIC: 231.5
```

Correlation

For Pearson correlation use **cor(x,y, method =z)** where **z** is **'pearson'**, **'spearman'**, or **'kendal'** and **'pearson'** is the default (so if you want Pearson you don't have to say so. This give just the correlation coefficient that you asked for. To get a more comprehensive set of results use:

```
 > with(dat, cor.test(x, y, alternative =  "two.sided" OR "less" OR "greater"), method
= "pearson" OR "kendall" OR "spearman"), conf.level = 0.95))
```

where **alternative** defines the alternative hypothesis and thus whether the test in one- or two-tailed; **conf.level** specifies the size of the confidence interval to be reported. The output is:

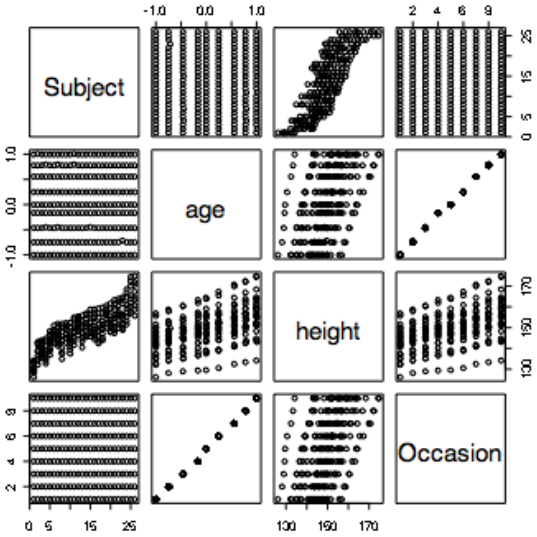
```
> with(dat, cor.test(x, y, alternative = "two.sided", method = "pearson", conf.level =
0.95))
        Pearson's product-moment correlation
data:  x and y
t = 43.3872, df = 148, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.9490525 0.9729853
sample estimates:
     cor
```

To calculate a correlation matrix from several variables, simply use **cor(dat)** where **dat** is a data frame object in the current workspace. For example, **>cor(Oxboys)** gives:

```
Subject         age                  height     Occasion
Subject   1.0000000000 -0.0004189085 -0.03895067 0.0000000
age      -0.0004189085  1.0000000000  0.46416066 0.9988791
height   -0.0389506695  0.4641606632  1.00000000 0.4644676
Occasion  0.0000000000  0.9988790961  0.46446765 1.0000000
```

If you want to test hypotheses about these correlations, however, you will need to use **cor.test ()** on each pair-wise relation as above.

To see what this all looks like in graphs, use **>pairs(object)** as above, to get a scatterplot matrix like this:



Regression

Ordinary Least Squares (OLS) regression can be calculated using **>lm(y~x)** which asks for y as a function of x and gives the coefficients. The output looks like this:

```
> with(dat, lm(y~x))
Call:
lm(formula = y ~ x)
Coefficients:
(Intercept)            x
   -0.3631       0.4158
```

To plot this use **> with(dat, plot(x,y))** to get the basic scatter diagram, then **> with(dat, abline(lm(y~x)))** to plot the regression line on that scatter diagram, as shown in the graph to the right.

If you assign the **lm()** function to an object, then you can query that object for more info, like this:

```
> myreg= with(dat, lm(y~x))
> xxx(myreg)
```

where **xxx** can be **summary**, **coefficients**, **residuals**, **fitted.values**, **weights**, **df.residual**, **formula**, **influence**. NOTE that this becomes very handy in resampling stats when you need to calculate a single coefficient many times.

For example, **>summary(myreg)** gives:

```
> summary(myreg)
Call:
lm(formula = y ~ x)
Residuals:
    Min      1Q  Median      3Q     Max
-0.56515 -0.12358 -0.01898  0.13288  0.64272

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.363076   0.039762  -9.131  4.7e-16 ***
x            0.415755   0.009582  43.387  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 0.2065 on 148 degrees of freedom
Multiple R-squared: 0.9271, Adjusted R-squared: 0.9266
F-statistic:  1882 on 1 and 148 DF,  p-value: < 2.2e-16
```

1.10 Getting Help

Select **Help>R for Mac OS X FAQ** to get some general questions answered

Select **Help>R help** or type **help.start()** for some basic info, not your typical Mac help page though.

Select **Help>Vignettes** for access to some built-in pdfs in Packages

Type **help(xxx)** or **?xxx** for help on R functions etc. Most help topics have examples that can be seen by typing **example(xxx)** where **xxx** is the topic. The help files for R functions are set up in a standard format that looks like this at the top of the document:

aov {stats} R Documentation

Fit an Analysis of Variance Model

Description

Fit an analysis of variance model by a call to `lm` for each stratum.

Usage

```
aov(formula, data = NULL, projections = FALSE, qr = TRUE,
    contrasts = NULL, ...)
```

Arguments

`formula`	A formula specifying the model.
`data`	A data frame in which the variables specified in the formula will be found. If missing, the variables are searched for in the standard way.
`projections`	Logical flag: should the projections be returned?
`qr`	Logical flag: should the QR decomposition be returned?
`contrasts`	A list of contrasts to be used for some of the factors in the formula. These are not used for any `Error` term, and supplying contrasts for factors only in the `Error` term will give a warning.
`...`	Arguments to be passed to `lm`, such as `subset` or `na.action`. See 'Details' about `weights`.

Details

This provides a wrapper to `lm` for fitting linear models to balanced or unbalanced experimental designs.

The main difference from `lm` is in the way `print`, `summary` and so on handle the fit: this is expressed in the traditional language of the analysis of variance rather than that of linear models.

in the upper left is the name of the function and **{xxx}** the package that contains that function, **Description** is brief and often incomprehensible to the beginner; **Usage** shows the syntax with all of the defaults that are assumed if you do not include them in the command; **Arguments** explains each of the terms in the command; **Details** is a little more expansive than the **Description**. This is all followed by various other things and some examples which, for some bizarre reason are almost always too complex and incomprehensible to be useful to the beginner. In fact much of what is in these help files will be very difficult to understand so at first focus mainly on the **Usage** and **Arguments** and how to understand those.

Go to the different special interest group (SIG) sites for questions that people have asked and usually some excellent answers. You can subscribe to any of these lists so that you get regular delivery of postings to your email. Here are a few of them that I use:

R-SIG-MAC: for Mac OSX questions, though these often (usually?) are about general stats and R questions; people get a bit shirty here if you ask a question that's already been

answered. Go to https://stat.ethz.ch/mailman/listinfo/r-sig-mac

r-sig-ecology: an excellent compilation of interesting questions with particular reference to analyses that ecologists and evolutionary biologist perform. Even very basic and simple questions are answered politely here. Go to https://stat.ethz.ch/pipermail/r-sig-ecology/

r-sig-phylo: as for r-sig-ecology but here the questions are about phylogenetic analyses. Go to https://stat.ethz.ch/pipermail/r-sig-phylo/

1.11 Finishing a Session

To quit R, you just **File>Quit**, of course, but before you do that think about whether you want to:

(i) **File>Save As** to save the entire console session as a .txt file,

(ii) **Workspace>Save Workspace File** to save all the objects and Packages in your workspace so you can load them all at once some other time,

(iii) save the current or any of the previous Quartz windows as pdfs,

(iv) save the History file as a .history file using the button at the bottom of the History drawer, or

(v) **Workspace>Save Default Workspace** to save your current workspace (with all the loaded packages) as your default workspace that will be loaded automatically whenever you open R

1.12 Some Guiding Principles

There is a lot of variation in how different people use R. My approach here is based on three principles:

- learn only one way to do each thing and stick to that method/menu item/command/style, at least until you are very proficient at using R.
- use a spreadsheet or other stats program to organize, store and manipulate your dataset. Use R only for analysis and drawing graphs. Save your dataset as a .csv file and read it into R as needed
- do all your typing in the editor first then transfer it to R. Save these editor files with your project and include only the correct commands that worked. Make editor files that include the commands for each time of analysis that you perform and document these commands carefully. Use these editor files to remind you how to do different kinds of analyses, and simply copy/paste the relevant commands as needed

1.13 Homework

Before our next session I suggest you do the following:

- work through this entire document using R and make sure you can do everything that is described here; chance are good that we did not get through all of section 1.9 so work through the rest of that on your own
- import one of your own datasets into R and do as many of the statistical analyses as you can as described in section 1.9

- pick a couple (or all, if you are ambitious) of the stats shown in 1.9 and redo one or more of the examples from your favourite stats text. Completely document your editor file and compare your results.
- make a list of any questions you might have about anything in this document or how R works

1.14 Getting More Information

Web pages:

Quick R http://www.statmethods.net/ a great website for examples and basic commands; a use this site all the time to remind me how to do things

An Introduction to R http://cran.r-project.org/doc/manuals/R-intro.html also available as a downloadable pdf (see below), this is quite a comprehensive introduction to R with lots of useful code

R and statistics http://www.biw.kuleuven.be/vakken/statisticsbyR/index.htm some good regression and anova examples

Kickstarting R http://cran.r-project.org/doc/contrib/Lemon-kickstart/kr_intro.html some simple examples for the beginner and a pretty good introduction to setting up R to make it most useful

Downloadable pdfs:

Simple R by John Verzani is a more extensive guide to standard statistics from one- way hypothesis testing, to ANOVA's and regression. (http://cran.r-project.org/doc/contrib/Verzani-SimpleR.pdf)

R Primer by Christopher Green is an old (2004) primer for an upper year stats course, but has a more detailed introduction to logic tests and looping (as well as statistical functions). (http://www.stat.washington.edu/cggreen/rprimer/first-edition/rprimer.09212004.firsted.pdf)

An Introduction to R by Venables & Smith is a great resource for matrices and matrix functions, as well as writing your own functions and more advanced plotting functions. (cran.r-project.org/doc/manuals/R-intro.pdf) Note this is the same material as at the web page above.

Statistics Using R with Biological Examples by Seefeld & Linder is a nice reference for statistical methods that go beyond frequentist statistics. (http://cran.r-project.org/doc/ contrib/Seefeld_StatsRBio.pdf)

Books that I have, use, and like:

Adler J. 2009. R in a Nutshell. OReilly Media <learn how to program in R, but also how to find the right user-contributed R packages for statistical modeling, visualization, and bioinformatics.>

Bolker BM. 2008 Ecological Models and Data in R. Princeton University Press. ISBN 978-0-691-12522-0 <lots of useful stats as well as info on model building>

Crawley MJ. 2005 Statistics: An Introduction using R. Wiley. ISBN 0-470-02297-3 <basic intro to statistics that uses R for the examples. Crawley's approach and writing can sometimes be quirky.>

Crawley MJ. 2007 The R Book. John Wiley, New York <this used to be the bible, and just as thick, for R users; lots of details on how to do a wide variety of things in R and lots of stats>

Faraway JJ. 2004 Linear Models with R. Chapman & Hall/CRC, Boca Raton, FL. ISBN 1-584-88425-8. <very detailed and comprehensive on the basics of linear models>

Faraway JJ. 2006 Extending Linear Models with R: Generalized Linear, Mixed Effects and Nonparametric Regression Models. Chapman & Hall/CRC, Boca Raton, FL. ISBN 1-584-88424-X. <more complex models than his previous book, with worked examples and downloadable datasets from his website>

Wright DB, London K. 2009 Modern Regression Techniques Using R: A Practical Guide. SAGE, London, UK. ISBN: 9781847879035.

Zuur AF, Ieno EN, Walker N, Saveiliev AA, Smith GM. 2009 Mixed Effects Models and Extensions in Ecology with R. Springer, New York. ISBN 978-0-387-87457-9. <all about mixed models and model building; very readable and easy to follow>

Zuur AF, Ieno EN, Meesters E. 2009 A Beginner's Guide to R. Use R. Springer. ISBN: 978-0-387-93836-3. <I just got this and it looks to be very readable and easy to follow, but not as easy as this workshop guide that you just read!>