Appendix A
Doing Things in R

This appendix is not an exhaustive list of all the things that R can do, but rather a list of some common steps you may want to take gathered in one convenient place, and a place to throw a few miscellaneous ideas about R that didn't fit neatly anywhere in the book.

Getting Started

I'm Looking at the R Console; Now What?

The really first thing to do is to get R Commander going so you will have a menu-driven graphical user interface (GUI) to help you. To get R Commander if you do not have the CD that came with this book, in R Console pull down the Packages menu and click on "Install packages." The first long skinny window that will pop up is labeled "CRAN mirror." You need to choose a CRAN mirror site before you can download packages. Choose any site, but preferably one near your geographical location. Press OK, and then another long skinny window will pop up. Scroll down to the "R"s and double click on "Rcmdr." Once that is finished installing, type this (exactly!) in the R Console window:

```
>library(Rcmdr)
```

The R Commander GUI will open.

Getting Data into R

I think the easiest way to do this is to use R Commander. Open R Commander (type `>library(Rcmdr)` into R Console to get R Commander to open). In R Commander, choose "Data" from the menu, and then "Import data." If you have an SPSS (.sav) file, choose the "from SPSS data set" option. The only thing I change in the dialogue box that pops up is the name of the data set.

If you have a .txt file, use the "from text file or clipboard" choice. This is a little trickier, and you'll have to know whether your numbers are separated by white spaces, commas, tabs, or something else. If a file is .csv, this means it is comma-delimited, so tick the "Commas" button under "Field Separator." You might have to experiment with this to get the correct format of your imported file. Once you click OK, R Commander lets you browse your computer to find the file.

With R Console, if the file you want is in the R/bin folder, just type:

```
>lafrance<-read.csv("lafrance.csv")
```

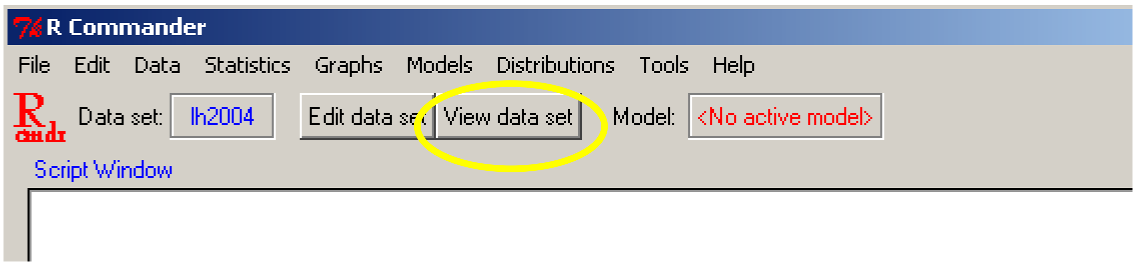
The first `lafrance` is necessary in order to put the .csv file into an object named lafrance that is in the workspace. If you only do the `read.csv("lafrance.csv")` command, it will print out the file on the R Console but you cannot manipulate the object.

Finding Out Names of a Data Set

```
>names(lafrance)
```

Looking at a Data Set

If you want to see what your data file looks like, press the "View data set" button on the R Commander window. This shows you the active file (lh2004 in this case).



Making R Work for You in the Long Run

Make it a habit to collect all the R commands and analyses that you do in some type of separate file, whether it be a Word document, TinnR (an ASCII file editor intended to be better than the Notepad program, and found at www.sciviews.org/Tinn-R), or R's own .Rhistory file (this can be opened with Notepad and contains a collection of the commands you made during your session). You can make comments on what you did (preceded by a # so that R will ignore them if you run the commands again) to help you remember your analyses years later.

Getting Help!

How to Pull Up Specific Help Files

Let's say you want to understand the lmRob function better, so you pull up the help file for it (and you already have opened the robust library). You can pull up this help file in the following ways:

```
>help("lmRob") #use of quotes optional here
>help("[[") #use of quotes absolutely necessary here
>?lmRob
```

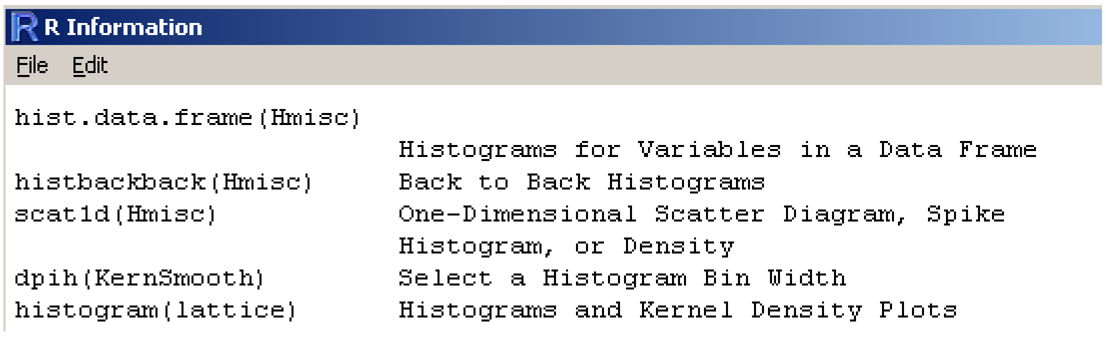
In R Console: Help>RFunctions (text); then type in lmRob

How to Find Help When You're Not Sure What You Want

Suppose you want to make a histogram but you're not sure how to do it in R. You've tried histogram(my.data) but that doesn't work, so you can type this to find out more about how to make histograms:

```
>help.search("histogram")
```

R will come up with a list of relevant topics. At the bottom of the box that pops up when you use the help.search() command it says "Type 'help(FOO, package=PKG)' to inspect entry 'FOO(PKG) TITLE.'" This command is hard to process at first! But, as an example, let's say you entered help.search("histogram"). The following is something like the list of topics you will see:



```
hist.data.frame(Hmisc)
                        Histograms for Variables in a Data Frame
histbackback(Hmisc)     Back to Back Histograms
scat1d(Hmisc)           One-Dimensional Scatter Diagram, Spike
                        Histogram, or Density
dpih(KernSmooth)        Select a Histogram Bin Width
histogram(lattice)      Histograms and Kernel Density Plots
```

Now you can type the name of a specific package into R to get more information about it. For example, let’s look at the first entry in this list:

```
>help(hist.data.frame, package=Hmisc)
```

Notice that FOO is just shorthand for the function name.

You can also try the apropos command, which is a partial name search and looks for commands relevant to the topic you put in parentheses:

```
>apropos ("ML", ignore.case=FALSE)
[1] "MLRplot"
```

How to Understand R Help Files

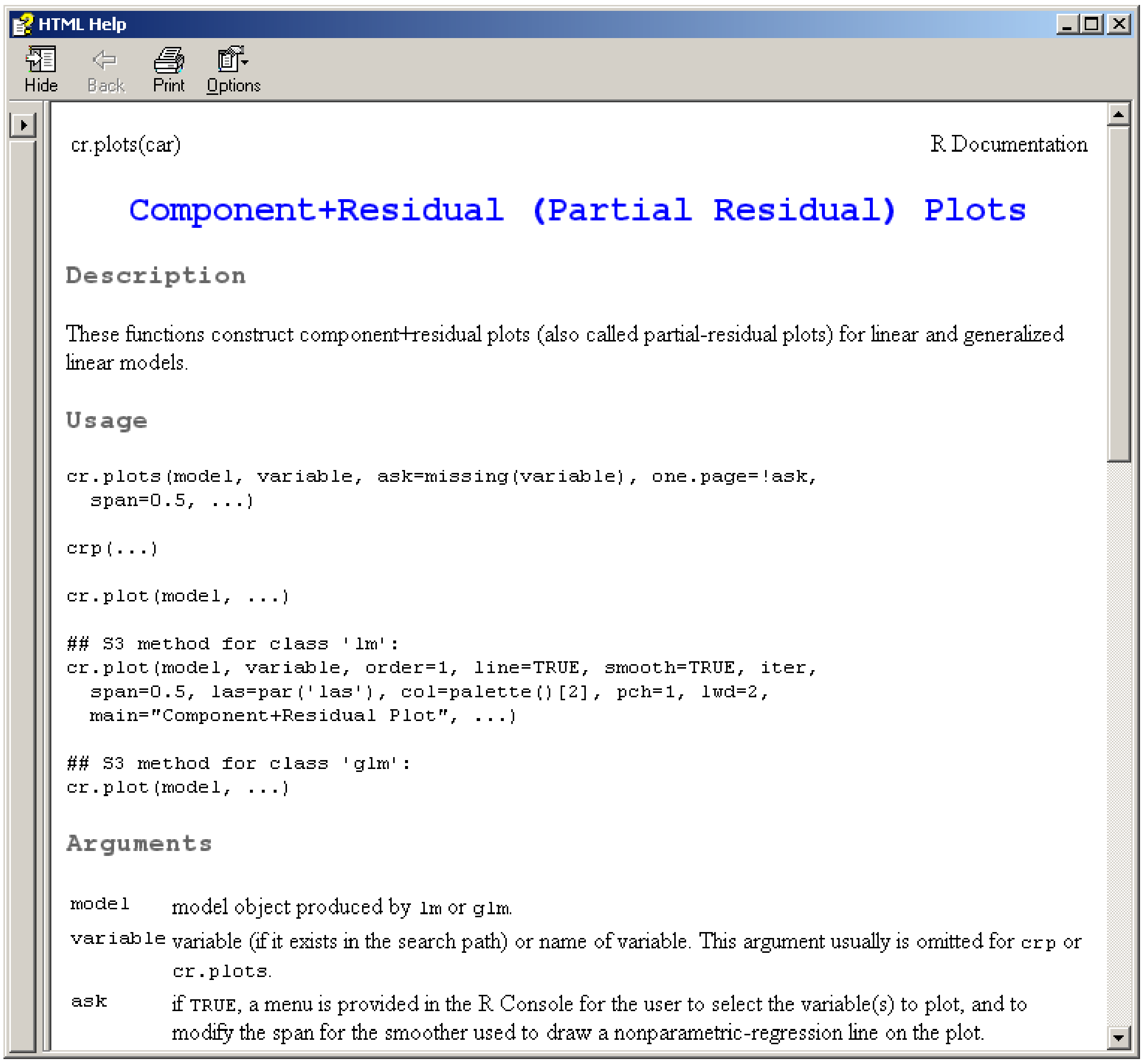
There are nine basic parts to help files: Description, Usage, Arguments, Details, Value, Author(s), References, See Also, and Examples. I often find it easiest to start with the examples. You can type:

```
>example(cr.plots)
```

and, if there are example files, they will run on R and show you what the function does. This is especially helpful if graphics are involved. To find out more about what cr.plots are and how they work, type one of these equivalent commands:

```
>help(cr.plots)
>?cr.plots
```

The following window will appear:



cr.plots(car) R Documentation

Component+Residual (Partial Residual) Plots

Description

These functions construct component+residual plots (also called partial-residual plots) for linear and generalized linear models.

Usage

```
cr.plots(model, variable, ask=missing(variable), one.page=!ask,
  span=0.5, ...)

crp(...)

cr.plot(model, ...)

## S3 method for class 'lm':
cr.plot(model, variable, order=1, line=TRUE, smooth=TRUE, iter,
  span=0.5, las=par('las'), col=palette()[2], pch=1, lwd=2,
  main="Component+Residual Plot", ...)

## S3 method for class 'glm':
cr.plot(model, ...)
```

Arguments

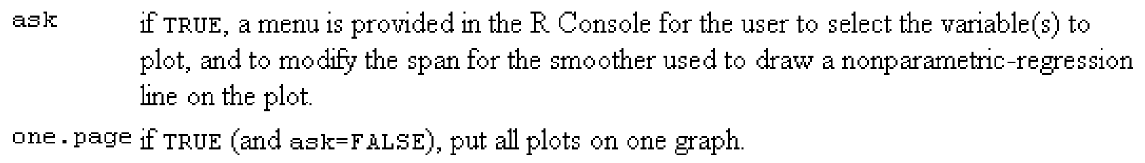
model	model object produced by lm or glm.
variable	variable (if it exists in the search path) or name of variable. This argument usually is omitted for crp or cr.plots.
ask	if TRUE, a menu is provided in the R Console for the user to select the variable(s) to plot, and to modify the span for the smoother used to draw a nonparametric-regression line on the plot.

We see that cr.plots are part of the car library (a very useful library). The Description section tells us that cr.plots are component+residual plots. The Usage part will show you exactly what arguments can be used in the function. For the cr.plots function, there are several different usages. The first is:

```
cr.plots(model, variable, ask=missing(variable), one.page=!ask,
  span=0.5, ...)
```

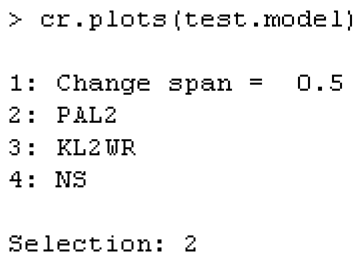
This is the most general syntax of the function. It tells us that we will need to put a model in the first argument position. The next argument is a variable, but it is not clear yet what that variable will be. Other arguments include one called “ask” and “one.page.” You can find out more about what these arguments are in the Arguments and Details sections. The default settings for these arguments are also given here. For example, in the “S3” method, the third argument, the default for order will be “1.” Don’t give up if you can’t understand the Usage right away, because there is more information later in the help document! The other variations in the Usage section tell us that this plot might also be called by crp. Its usage with lm models and glm models gives additional arguments that a user might be interested in setting. So they are all the same command, but different variations of it for different situations.

The Arguments section can help clarify the syntax. For example, Arguments tells us that the model in the first argument position needs to be a regression object created by lm or glm. Here are some details about "ask" and "one.page":



ask	if TRUE, a menu is provided in the R Console for the user to select the variable(s) to plot, and to modify the span for the smoother used to draw a nonparametric-regression line on the plot.
one.page	if TRUE (and ask=FALSE), put all plots on one graph.

It looks as though, for "ask," we can set it to ask=TRUE. However, it wasn't entirely clear to me how to use ask, since the Usage seemed to indicate I should put in a variable name there. To figure this out, I looked at R Commander (because this is a menu function in R Commander) as well as the Examples, and then I just fooled around with trying various kinds of syntax. I actually discovered that, if I put in a model name without any other arguments, then a menu popped up to ask me which variables I wanted to consider:



```
> cr.plots(test.model)

1: Change span =  0.5
2: PAL2
3: KL2WR
4: NS

Selection: 2
```

This led me to believe that "ask" is not so much something the user sets as an integral part of the command which asks the user which things to plot.

The Details section also often has some useful details that the author does not have any other place to put. For example, for cr.plots it is noted that the model cannot contain interactions. This is an important piece of information. Author and References are self-explanatory. The See Also has clickable terms that can lead you to related functions that will help you figure out how to use the function.

As you become more familiar with the R help pages they will make more sense and become more helpful!

Finding Out the Arguments That Are Needed for a Command

```
> args(lm)
function (formula, data, subset, weights, na.action, method = "qr",
 model = TRUE, x = FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE,
 contrasts = NULL, offset, ...)
```

When You Want More Information about a Certain Package

Let's say you want to do robust regression and you have noted that robustbase is a package that does robust regression. There are a couple of ways to get more information about this package:

```
>help.start()
```

In R Console: Help> HTML help

Find the Packages hyperlink under References.

Many packages have links here so that you can hyperlink from one function to another within the package. However, not all packages are found here. For example, say I want to know more about the prettyR package. Another thing to try is vignettes.

```
>vignette(all=T)
>vignette("grid") #for a vignette from a specific package
```

I still can't find anything about prettyR, so I go to this website: www.cran.r-project.org.

Click on "Packages" on the left-hand side bar. This has pdf documents from many packages which can be quite helpful, and it has one about prettyR.

Helpful Online Articles or Sites That Explain More about R

- On the R website (www.r-project.org), go to Manuals and you will find "An Introduction to R."
- One fun place to look at various kinds of graphs available in R is the web site http://addictedtor.free.fr/graphiques/.
- An introduction to statistical computing in R and S by John Fox at http://socserv.socsci.mcmaster.ca/jfox/Courses/R-course/index.html.
- Tom Short's R reference card, organizing R syntax by function and giving short explanations in a compact format: http://cran.r-project.org/doc/contrib/Short-refcard.pdf.
- A short online tutorial about using R for psychological research: http://www.personality-project.org/r/r.guide.html.
- Notes on the use of R for psychology experiments and questionnaires: http://www.psych.upenn.edu/~baron/rpsych/rpsych.html.

A general resource is on the R home page; go to Other, and then follow the link to Contributed Documentation, where you will find many articles.

Getting Help Online

If you are having a problem, the R help archives usually have some answers. You can also post a question to r-help@lists.R-project.org. However, before posting you must be sure to research your question by searching the archives and reading the posting guide, which can be found by clicking on Mailing Lists and reading the information there.

To search for answers on the R help archives, go to the R website (www.r-project.org) and go to Search. I usually go to Baron's R site search, which is the first hyperlink on the page. Insert your term and browse away.

When you post a question, the posting guide does ask you to state what version of R you are using. You can get this information by typing:

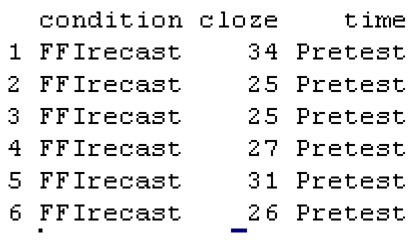
```
>sessionInfo()
```

This tells you what version of R you are using, what packages are attached, and what versions those packages are.

Types of Data

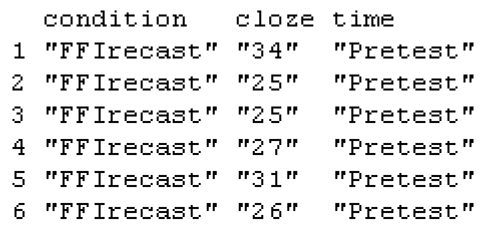
Understanding the Types of Data in R

If you import an SPSS file into R, it will be a “data frame.” A data frame is a collection of columns and rows of data, and the data can be numeric (made up of numbers) or character (a “string” in SPSS or, in other words, words that have no numerical value). It has only two dimensions, that of rows and columns. All of the columns must have the same length. Most of the data you will work with in this book is in data frames, and you can think of it as a traditional table. Here is an example of a data frame:



```
  condition cloze    time
1 FFIrecast    34 Pretest
2 FFIrecast    25 Pretest
3 FFIrecast    25 Pretest
4 FFIrecast    27 Pretest
5 FFIrecast    31 Pretest
6 FFIrecast    26 Pretest
```

A “matrix” is similar to a data frame, but all the columns must be of the same format, that is, either numeric or character. Like a data frame, a matrix has only two dimensions (row and column), and all columns must be of the same length. Here is an example of what happened when I turned the data frame above into a matrix:



```
  condition   cloze time
1 "FFIrecast" "34"  "Pretest"
2 "FFIrecast" "25"  "Pretest"
3 "FFIrecast" "25"  "Pretest"
4 "FFIrecast" "27"  "Pretest"
5 "FFIrecast" "31"  "Pretest"
6 "FFIrecast" "26"  "Pretest"
```

You can see that all the cells are now surrounded by quotation marks, meaning they are characters. A look at the structure of this matrix reveals indeed that all the variables are now regarded as character, meaning they do not have any numerical meaning.

A vector is just one line of data that might represent one column of a data frame or matrix. It can be composed of numeric, character, or logical entries.

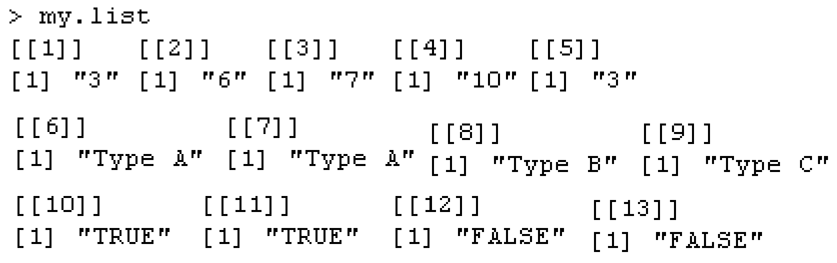
```
a=c(3,6,7,10,3) #numeric
b=c("Type A", "Type A", "Type B", "Type C") #character
c=c(TRUE, TRUE, FALSE, FALSE, FALSE, TRUE) #logical
```

If you take one column from a data frame, this is also a vector:

```
> lh2004$group
[1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 4 4
[36] 4 4 4 4 4 4
```

A list is a collection of vectors, which do not all need to be the same length. The elements of the list can be of different types as well. For example, we can create a list that holds all of the vectors presented above:

```
>my.list=as.list(c(a,b,c))
```

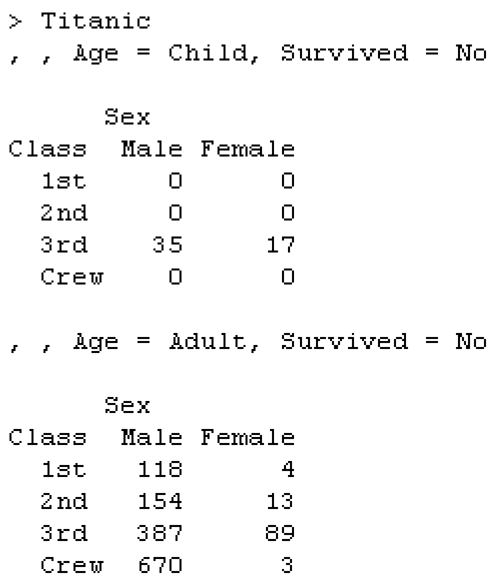


```
> my.list
[[1]]     [[2]]     [[3]]     [[4]]     [[5]]
[1] "3"   [1] "6"   [1] "7"   [1] "10"  [1] "3"

[[6]]          [[7]]          [[8]]          [[9]]
[1] "Type A"   [1] "Type A"   [1] "Type B"   [1] "Type C"

[[10]]       [[11]]       [[12]]        [[13]]
[1] "TRUE"   [1] "TRUE"   [1] "FALSE"   [1] "FALSE"
```

Each entry is referenced by its position in the list. For example, the 13th member of the list can be referenced by:

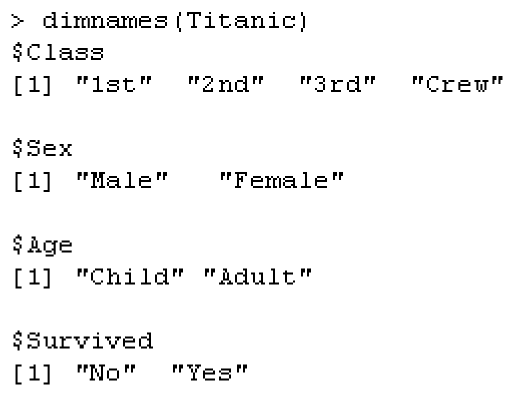
```
> my.list[[13]]
[1] "FALSE"
```

An array is similar to a data frame, but it can have more than two dimensions. The array for the *Titanic* data, found in the vcd library, is an array.



```
> Titanic
, , Age = Child, Survived = No

      Sex
Class  Male Female
  1st     0      0
  2nd     0      0
  3rd    35     17
  Crew    0      0

, , Age = Adult, Survived = No

      Sex
Class  Male Female
  1st   118      4
  2nd   154     13
  3rd   387     89
  Crew  670      3
```

This data set has four dimensions, which can be ascertained by their dimension names:



```
> dimnames(Titanic)
$Class
[1] "1st"  "2nd"  "3rd"  "Crew"

$Sex
[1] "Male"   "Female"

$Age
[1] "Child" "Adult"

$Survived
[1] "No"  "Yes"
```

A factor is a vector that R considers a categorical variable, and it contains two parts—a vector of integers as well as a vector of character strings that characterize those integers. Here I create a factor called "category":

```
> category=c(rep("A", 10), rep("B", 15))
> category=factor(category)
> category
[1] A A A A A A A A A A B B B B B B B B B B B B B B B
Levels: A B
> summary(category)
A B
10 15
```

In the summary, R knows that this is a factor and so it counts up the As and Bs separately. If we had not made "category" into a factor and performed a summary, it would simply count up 25 character entries.

Changing Data from One Type to Another

There are many commands to change data types (use methods(is) for a complete list), but essentially there are only a few that are likely to be useful. One is when you want to change your data from separate vectors to a data frame, like this:

```
>collect=as.data.frame(results, group, type)
```

Another may be when you want to change a data frame into a matrix:

```
>collect.matrix=as.matrix(collect)
```

Another common task is changing a non-factor into a factor:

```
> murphyLong$verbtype=as.factor(murphyLong$verbtype)
```

Remember, you can check the class of your data by typing:

```
>class(collect.matrix)
```

Manipulating Data

Finding Out What Class Your Data Is

```
>class(lafrance)
[1] "data.frame"
```

Possible classes include "data.frame," "matrix," "array," "vector," "table," and "list." For more information about what types of data classes mean, see the "Types of Data" section. If you use class on a vector, it will tell you whether it is numeric or character:

```
> class(lyster$cloze)
[1] "numeric"
```

Finding Out the Size of Your Data

```
>dim(lyster)
```

```
[1] 540 4
```

This specifies that the data frame has 540 rows and 4 columns.

Finding Out How Many Rows in One Column of Data

```
>length(murphy$REGULARV)
[1] 60
```

Finding Out about the Structure of Your Data

The `str` command will give you a wealth of information about your data.

```
> str(Forbes2000) #data found in HSAUR library
'data.frame': 2000 obs. of 8 variables:
$ rank : int 1 2 3 4 5 6 7 8 9 10 ...
$ name : chr "Citigroup" "General Electric" "American Intl Group"
$ country : Factor w/ 61 levels "Africa","Australia",..: 60 60 60 60 56 60 56 28
$ category : Factor w/ 27 levels "Aerospace & defense",..: 2 6 16 19 19 2 2 8 9 $
sales : num 94.7 134.2 76.7 222.9 232.6 ...
$ profits : num 17.85 15.59 6.46 20.96 10.27 ...
$ assets : num 1264 627 648 167 178 ...
$ marketvalue: num 255 329 195 277 174 ...
```

The first line tells us what class of object it is and also that there are eight variables.

The next eight lines tell us about each of the variables, whether they are character (chr; for the Forbes2000 data, these are not factors because none of the strings is repeated; they are all different companies), factors (and how many levels in each factor; you get a factor if you import with character strings), or just plain numbers (num).

Looking at Just a Few Rows of Your Data

```
>head(murphy) # lists the first few rows
>murphy[1:15,] #lists the first 15 rows
>murphy[sort(sample(60,5)),] #samples 5 rows from the 60 possible
```

Attaching and Detaching Data

You can find out what data sets are already attached by typing:

```
>search()
```

If any data sets are attached, they will be listed. If more than one data set is attached, the last one attached is in the highest position and will take precedence if they have any names that are the same.

Attach a data set by typing:

```
>attach(lyster)
```

One good reason to attach data is so that you can perform subsequent commands using just the names of the variables inside the attached data set instead of always having to append the name of the data set. In other words, if you have not attached the data, to calculate the mean of the "cloze" variable inside the lyster data set, you'll need to type:

```
>mean(lyster$cloze)
```

On the other hand, if you have attached the data set, you can just type:

```
>mean(cloze)
```

Note that, if you modify something in the data when it is attached, then changes will not be seen until you detach and reattach the data. To detach the data:

```
>detach(lyster)
```

or just:

```
>detach()
```

which detaches the most recently attached data set (the one highest up in the search path list).

Picking Out Certain Rows or Columns

With a data frame or matrix, there are a number of syntax equivalents you can use to pick out a certain column:

```
lyster[,3]
lyster[[3]]
lyster$time
```

To pick out a row:

```
lyster[15,]
```

Finding Out What Objects Are in Your Active Desktop

```
>ls()
```

This command lists all active objects, which are data sets and user-defined functions.

Finding Out What Data Sets Are in a Package or Library

```
>data(package="languageR")
```

This command causes a separate window with a list of the data sets to appear. To find out more about the data sets, in R Console choose Help > HTML help and, when a new window comes up, choose "Packages." Choose your package (such as "languageR"), click on hyperlink, then navigate to name of data set to find out more about it. To find out about all available data sets, type:

```
>data(package=.packages(all.available=TRUE))
```

Removing an Object from Your Active Desktop

Assume you have created an object that you now want to get rid of:

```
>rm(meanFun)
```

You can remove *all* active objects this way; just make sure that is really what you want to do!

```
>rm(list = ls())
```

Removing Specific Points from an Analysis

To remove just one point:

```
>model2=lm(growth[-7]~tannin[-7])
```

To remove more than one point from an analysis:

```
>model3=lm(growth[-c(1,7)]~tannin[-c(1,7)])
```

Removing All NAs from a Data Frame or Matrix

```
>newlh=na.omit(newlh)
>newlh=na.exclude(newlh)
```

The omit function will filter the NAs, while exclude will delete them. The following command also works by asking R to include any entries in which the logical answer to the question of whether there are NAs (is.na) is FALSE (! indicates a logical negation).

```
>newlh=newlh[!is.na(newlh)]
```

Ordering Factors of a Categorical Variable

First, find out if your variable is a factor:

```
> is.factor(beq$CATDOMIN)
[1] TRUE
```

If it is not, you can easily make it so:

```
>beq$CATDOMIN=as.factor(beq$CATDOMIN)
```

Next, check to see if it is already ordered:

```
> is.ordered(beq$CATDOMIN)
[1] FALSE
```

You'll need to know the exact names of the levels:

```
> levels(beq$CATDOMIN)
[1] "YES" "NO" "YESPLUS"
```

Now you are ready to order the levels to your own preference:

```
>beq$CATDOMIN=ordered(beq$CATDOMIN,levels=c("YES", "YESPLUS", "NO"))
```

If your factors came without names, you can set those in this command too:

```
>group=ordered(group,levels=c("2", "3", "1"), labels=c("NS adults", "NNS adults",
"NS children"))
```

Recoding Factors of a Categorical Variable into Different Factors

First, find out if your variable is a factor:

```
> is.factor(beq$FIRSTLAN)
[1] TRUE
```

You'll need to know the exact names of the levels:

```
> levels(beq$FIRSTLAN)
[1] " " "Afrikaans" "Albanian" "Arabic"
[5] "Armenian" "Basque" "Bengali" "Boobe"
```

(There are 108 levels! We want to consolidate.)

Now you can begin to form groups:

```
levels(beq$FIRSTLAN)[c(57,93,78,79,80)]= "Italic"
levels(beqmore$FIRSTLAN)[c(27,24,25,26,40,41,42,43,44)]= "Germanic"
```

etc.

Extracting One Column from a Matrix and Making a New Variable

```
>data=lyster[,'cloze']
>data=lyster$cloze #both of these commands are equivalent
```

Converting a Numeric Variable to a Character Variable (Changing a Variable into a Factor)

When your splitting variable is characterized as "character" it will be listed when you open the "Plot by groups" button in R Commander. If this doesn't work (and sometimes it seems not to), you can fix it by using R Console. Just make sure the data set is not attached (detach()) before you run the following code:

```
>lyster$condition=factor(lyster$condition)
>is.factor(lyster$condition)
[1] TRUE
```

Converting an Array into a Condensed Table

```
>library(vcd)
>(STD=structable(~Sex+Class+Age,data=Titanic[, ,2:1, ]))
```

Replacing Certain Cells in a Matrix with Something Else

Here I'm thinking of a situation where you might import a data set that had a "99" everywhere that data was not entered. To use this data set correctly in R you'll need to have NAs in those places. If we assume that m1 is a matrix or vector (this won't work with a data frame), you can use the following command:

```
>m1[m1==99]=NA
```

Creating a Vector of Data

```
>data<-c( 22, 23, . . . )
```

This will just put data into one variable.

Putting Vectors of Data Together into a Data Frame

```
>my.data.frame=data.frame(rt=data, subj=factor(subject), drug=factor(treatment))
```

Note that, if “treatment” were already a factor, there would be no need to specify it as such in the data frame.

Adding Columns to a Data Frame

```
>new.data.frame<-cbind.data.frame(participant=1:20, my.data.frame)
```

Naming the Columns of a Matrix or Data Frame

```
>dimnames(Ela.mul)[[2]]=
c("subj", "gp", "d11", "d12", "d13", "d21", "d22", "d23")
```

Here the [[2]] refers to the column names, so this is generalizable to any matrix or data frame.

```
Ela.mul
 subj gp d11 d12 d13 d21 d22 d23
1 1 1 19 22 28 16 26 22
2 2 1 11 19 30 12 18 28
```

Subsetting a Data Frame (Choosing Columns)

You can pick out the columns by their names (this data is in the HSAUR library):

```
>Forbes3=Forbes2000[1:3, c("name", "sales", "profits", "assets")]
```

or by their column number:

```
>Forbes3=Forbes2000[1:3, c(2,5,6,7)]
```

Note that this subset also chooses only the first three rows. To choose all of the rows, leave the space before the comma blank:

```
>Forbes3=Forbes2000[,c(2,5,6,7)]
```

This way of subsetting puts the columns into the order that you designate.

You can also subset according to logical criteria. The following command chooses only the rows where assets are more than 1000 (billion):

```
>Forbes2000[Forbes2000$assets > 1000, c("name", "sales", "profits", "assets")]
```

Subsetting a Data Frame (Choosing Groups)

To subset in R Commander, go to DATA > ACTIVE DATA SET > SUBSET ACTIVE DATA SET. You'll need to know the exact column name of the factor that divides groups, and the exact way each group is specified in that column to put into the "Subset expression" box. The R syntax is:

```
>lh2004.beg <- subset(lh2004, subset=LEVEL=="Beginner")
```

Note that a double equal sign (==) should be used after the index column name (LEVEL) and quotation marks should be around the name of the group. This syntax includes all columns in the data set.

```
>lh2004.beg <- subset(lh2004, subset=LEVEL=="Beginner",
select=c(PJ_P,R_L,SH_SHCH))
```

The "select" argument lets you choose only certain columns to subset.

Subsetting a Data Frame When There Are NAs

First, create a new variable composed of only rows which don't have NAs:

```
>library(HSAUR)
>na_profits=!is.na(Forbes2000$profits)
>table(na_profits)
>Forbes2000[na_profits, c("name", "sales", "profits", "assets")]
```

Subsetting a Data Frame to Remove Only One Row While Doing Regression

```
>model2=update(model1,subset=(tannin !=6)
```

The exclamation point means "not" that point, so this specifies to remove row 6 of the tannin data set for the updated model.

Ordering Data in a Data Frame

Say we want an index with sales in order from largest to smallest:

```
>order_sales=order(Forbes2000$sales) #in HSAUR library
```

Now we want to link the company name up with the ordered sales. First, rename the column:

```
>companies=Forbes2000$name
```

Now order the companies according to their sales:

```
>companies[order_sales]
```

Last, use the ordered sales as the rows and then choose the columns you want to see:

```
>Forbes2000[order_sales, c("name", "sales", "profits", "assets")]
```

Changing the Order of Rows or Columns in a Data Frame

```
>data(Animals,package="MASS")
> brain<-Animals[c(1:24,26:25,27:28),]
```

This takes rows 1–24, then row 26, then row 25, and then rows 27–28. To change the order of columns, just specify numbers *after* the comma.

```
>names(lh2004)
 [1] "LEVEL" "ZH_SH" "ZH_Z" "L_LJ" "R_L" "SH_S" "SH_SHCH"
 [8] "TS_T" "TS_S" "TS_CH" "PJ_P" "PJ_BJ" "MJ_M" "F_V"
[15] "F_X" "F_FJ" "R_RJ"
>lh2004.2=lh2004[,c(5,7,11,2:4,6,8:10,12:16)]
> names(lh2004.2)
 [1] "R_L" "SH_SHCH" "PJ_P" "ZH_SH" "ZH_Z" "L_LJ" "SH_S"
 [8] "TS_T" "TS_S" "TS_CH" "PJ_BJ" "MJ_M" "F_V" "F_X"
[15] "F_FJ"
```

Naming or Renaming Levels of a Variable

```
>levels(dataset$gender)
[1] "1" "2"
> levels(dataset$gender)=c("female", "male")
> levels(gender)
[1] "female" "male"
```

Creating a Factor

```
>TYPE=factor(rep(1:4, c(11,12,10,8)))
```

This creates a factor with four numbers, each repeated the number of times specified inside the "c" to create a vector of length 41.

```
>group=rep(rep(1:3, c(20,20,20)), 2)
```

This creates a factor where the numbers 1 through 3 are repeated 20 times each, and then cycled through again to create a vector of length 120. A factor can also be created with the gl function ("generate levels"). The following syntax creates the same results as the previous rep command:

```
>group=gl(4, 20, 120, labels=c("Quiet", "Medium", "Noisy", "Control")
```

The first argument specifies how many different factors are needed, the second specifies how many times each should be repeated, and the third specifies the total length. Note that this can work so long as each factor is repeated an equal number of times. What's nice about gl is that factor labels can be specified directly inside the command.

Adding a Created Factor to the End of a Data Frame

Assume that TYPE is a created factor.

```
>lh2003=cbind(lh2003,TYPE)
```

Changing the Ordering of Factor Levels without Using the "Ordered" Command

First, find out the current order of factor levels:

```
>group
```

```
[1] 1 1 1 1 1 1 1 1 . . .
[21] 2 2 2 2 2 2 2 . . .
[41] 3 3 3 3 3 3 3 . . .
attr(,"levels ")
[1] "NS children" "NS adults" "NNS adults"
```

Now change the factor order so that NS adults are first, NNS adults are second, and NS children are third.

```
>group=factor(rep(rep(c(3,1,2),c(20,20,20)),2))
```

Relabel “group” with level labels in the order of the numbers—the actual order in the list has not changed, but R will interpret the first name as corresponding to number 1, and so on.

```
>levels(group)=c("NS adults", "NNS adults", "NS children")
```

Dividing a Continuous Variable into Groups to Make Factors

This can be done in R Commander. Choose DATA > MANAGE VARIABLES IN ACTIVE DATA SET > BIN NUMERIC VARIABLE. R command is:

```
>ChickWeight$timecat = bin.var(ChickWeight$Time, bins=3, method='natural',
labels=c('early','mid','older'))
```

This command divides the continuous variable into three partitions.

Sampling Randomly from a List of Numbers

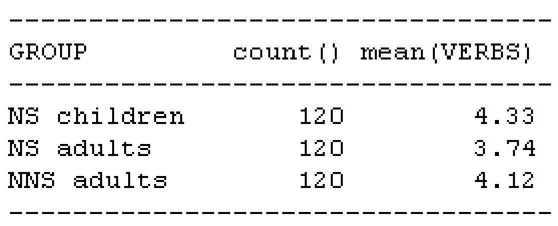
Suppose you want to generate a matrix that samples randomly from the numbers 1 through 5, which might represent a five-point Likert scale. The following command will generate a 100-row (assume that’s the number of respondents) and 10-column (assume that’s the number of items they responded to) matrix:

```
>v1=matrix(sample(c(1:5),size=1000,replace=T),ncol=10)
```

Obviously you could change this to specify the numbers that would be sampled, the number of columns, and the number of rows.

Creating Nicely Formatted Tables

```
>library(Epi)
>stat.table(GROUP, list(count(),mean(VERBS)),data=murphy)
```



```
---------------------------------
GROUP          count() mean(VERBS)
---------------------------------
NS children       120        4.33
NS adults         120        3.74
NNS adults        120        4.12
---------------------------------
```

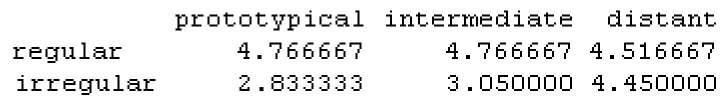
Creating a Table, Summarizing Means over Groups

The tapply function is quite useful in creating summary tables. The generic syntax is:

```
>tapply(X, INDEX, FUNCTION)
```

where X could be thought of as a response variable, INDEX as the explanatory variable(s), and FUNCTION as any kind of summary function such as mean, median, standard deviation, length, etc.

```
>tapply(murphy$VERBS, list(murphy$VERBTYPE, murphy$SIMILARI), mean)
```



```
           prototypical intermediate  distant
regular        4.766667     4.766667 4.516667
 irregular     2.833333     3.050000 4.450000
```

Graphics

Demonstration of Some Cool Graphics

```
>demo(graphics)
```

Other Interesting Demonstration Files

```
>demo(lm.glm)
```

Choosing a Color That You Like

```
>library(epitools)
>colors.plot(TRUE)
```

This command activates a locator, and you click on the colors you like from the R Graphics device. The names of the colors you click on will now show up in R Console.

Placing More Than One Graphic in the Graphics Device

Use the parameter setting to specify the number of rows and then columns:

```
>par(mfrow=c(2,3)) #2 rows, 3 columns
```

Getting Things out of R into Other Programs

Round Numbers

```
>round(t.test(v1)$statistic,2)
```

This command rounds off the value of *t* to two places. This can be useful for cutting and pasting R output into a manuscript.

Getting Data out of R

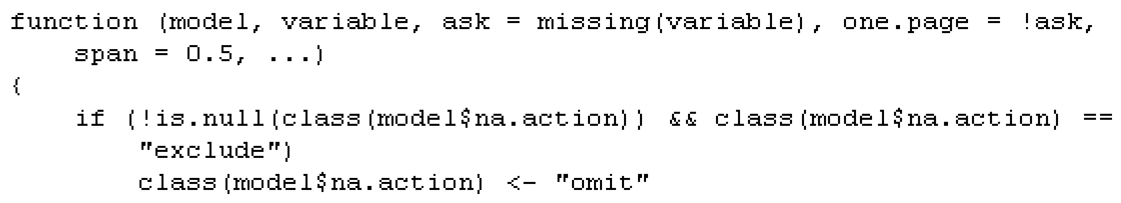
If you'd like to save a data set that you worked with in R, the `write.table(object1, "file1")` is the command to use. However, it is much easier to do this by using R Commander, which gives you a variety of options and lets you choose the file you will save it at. In R Commander, choose DATA > ACTIVE DATA SET > EXPORT ACTIVE DATA SET. If you choose "spaces" as a field separator, your file will be a .txt file. If you choose "commas," you can append the .csv designation to the name of your file, which means it is a comma-delimited .csv file. A program like Excel can open either of these types of files, but there are fewer steps if the file is .csv than if it is .txt.

Troubleshooting

Revealing Command Syntax

If you want to see the parts that go into any command and the arguments necessary for the command, just type the command, and they will be listed:

```
>cr.plots
```



```
function (model, variable, ask = missing(variable), one.page = !ask,
    span = 0.5, ...)
{
    if (!is.null(class(model$na.action)) && class(model$na.action) ==
        "exclude")
        class(model$na.action) <- "omit"
```

Understanding Error Messages

Many times error messages are cryptic. The first thing you should suspect is a typing error. Fox (2002b) says that, if you get an error telling you “object not found,” suspect a typing error or that you have forgotten to attach the object or the library that contains the object. Fox says the traceback command can provide some help, showing the steps of the command that were undertaken:

```
>traceback()
```

Getting Out of a Function That Won’t Converge or Whose Syntax You Have Typed Incorrectly

Press the “Esc” button on your keyboard.

Resetting the Contrasts Option for Regression

```
>options(contrast=c("contr.treatment", "contr.poly")
```

This is the default contrast setting in R.

Miscellaneous

Trimmed Means if There Are NAs

To get the 20% trimmed means, it is necessary to use the na.rm=T argument (which removes NA entries), as the mean command will not work with NAs in the data column:

```
>mean(smith$NEGDPTP,tr=.2,na.rm=T)
[1] 0.6152154
```

Using the Equals Sign instead of the Arrow Combination

According to Verzani (http://www.math.csi.cuny.edu/Statistics/R/simpleR) the arrow combination (“<-”) had to be used as the assignment operator prior to R version 1.5.0. I prefer to use the equals sign, as it takes less typing.

Calculating the Standard Error of the Means

R does not have a function for standard error of the means, but it is easy to calculate, because SE is just the square root of the variance, divided by the number of observations. The SE makes it easy to find the confidence interval for those means.

```
>se<-function(x)
{
y<-x[!is.na(x)] #remove the missing values
sqrt(var(as.vector(y))/length(y))
}
```

Once you have defined your function, you can use it with tapply to get a table:

```
>tapply(lyster$condition,IND=list(lyster$group, lyster$sex), FUN=se)
```

Suppressing Stars That Show "Statistical Significance"

```
>options(show.signif.stars=F)
```

This can be placed in the Rprofile file of R's etc subdirectory for permanent use, or can be used temporarily if simply entered into R Console in a session.

Loading a Data Set That Is Already in R, so R Commander Can See It

```
>data(ChickWeight)
```

Appendix B

Calculating Benjamini and Hochberg's FDR using R

pvalue<-c(.03, .002,.002,.93)	Insert all of your *p*-values for your multiple tests into the parentheses after "c."
sorted.pvalue<-sort(pvalue)	
j.alpha<-(1:4)*(.05/4)	Here I used 4 because I had 4 *p*-values; insert your own number of comparisons wherever there is a 4.
dif<-sorted.pvalue-j.alpha	
neg.dif<-dif[dif<0]	
pos.dif<-neg.dif[length(neg.dif)]	
index<-dif==pos.dif	
p.cutoff<-sorted.pvalue[index]	
p.cutoff	This line will return the cut-off value.
p.sig<-pvalue[pvalue<=p.cutoff]	
p.sig	This line will return the *p*-values which are significant.

This code is taken from Richard Herrington's calculations on the website http://www.unt.edu/benchmarks/archives/2002/april02/rss.htm. For more information about controlling the familywise error rate, please see this informative page.

Appendix C
Using Wilcox's R Library

Wilcox has an R library, but at the moment it cannot be retrieved from the drop-down list method in R Console. Instead, type this line directly into R:

```
install.packages("WRS",repos="http://R-Forge.R-project.org")
```

If you have any trouble installing this, go to the website http://r-forge.r-project.org/R/?group_id=468 for more information.

Once the library is downloaded, open it:

```
library(WRS)
```