An introduction to R

William Revelle
Swift 315
email: revelle@northwestern.edu

March 29, 2009

Contents

This short course will meet in Swift Hall 107 from 5-7 on Monday, Tuesday, Wednesday (March 30-April 1) and Monday, Tuesday (April 6-7).

1 Objectives

There are many possible statistical programs that can be used in psychological research. They differ in multiple ways, at least some of which are ease of use, generality, and cost. Some of the more common programs used are SAS, SPSS, and Systat. These programs have GUIs (Graphical User Interfaces) that are relatively easy to use but that are unique to each package. These programs are also very expensive and limited in what they can do. Although convenient to use, GUI based operations are difficult to discuss in written form. When teaching statistics or communicating results, it is helpful to use examples that others may use, perhaps in other computing environments. This course describes an alternative approach that is widely used by practicing statisticians, the statistical environment R.

R is used in various courses here at NU and has been adopted as the primary stats program for teaching at the University of Virginia and the University of Colorado (among others). I use it in teaching Psych 205, 371, 405, and 454.

The objective of this short course is very simple: to have you learn enough about R to start using it to facilitate your teaching and research. You will not, however be fluent in R. But, by the end of the course you should be wondering why you ever used SPSS or SAS. For "this is R. There is no if. Only how." (R `fortune`).

2 Requirements and readings

A willingness to learn and to ask questions. Bringing a personal computer to class would not be a bad idea.

Handouts of the lecture notes will be linked from this outline. Most of the handouts will be either pdfs of slides or pdfs of example code.

There are a number of tutorials on learning R, ranging from the short to the extensive. The definitive short text is An Introduction to R by Venables et al. (2008). This may either be purchased (proceeds go to the R Foundation) or downloaded. For those who are familiar with SPSS or SAS, the book, R for SAS and SPSS Users by Muenchen (2009), is a good introduction. (See his webpage at http://rforsasandspssusers.com/). For psychologists, my tutorial Using R for psychological research:A simple guide to an elegant package is not a bad beginning. See also the short and very short versions of that for undergraduates. As an example of what a bright undergraduate can do to help other undergraduates use R, see K. Funkhouser's Using R to analyze a simple data set.

There are a number of other very good tutorials on the web. An essential aid is the R reference card and the search engines R seek: a search engine for R and Jonathan Baron's search engine of the R help archives.

3 Day 1: What is R? An introduction

3.1 What is it?

The R Development Core Team (2008) has developed an extremely powerful "language and environment for statistical computing and graphics" and a set of **packages** that operate within this programming environment (R). The R program is an open source version of the statistical program S and is very similar to the statistical program based upon S, S-PLUS (also known as S+). Although described as merely "an effective data handling and storage facility [with] a suite of operators for calculations on arrays, in particular, matrices" R is, in fact, a very useful interactive package for data analysis. When compared to most other stats packages used by psychologists, R has at least three compelling advantages: it is free, it runs on multiple platforms (e.g., Windows, Unix, Linux, and Mac OS X and Classic), and combines many of the most useful statistical programs into one quasi integrated environment. R is free[1], open source software as part of the GNU[2] Project. That is, users are free to use, modify, and distribute the program, within the limits of the GNU non-license). The program itself and detailed installation instructions for Linux, Unix, Windows, and Macs are available through CRAN (Comprehensive R Archive Network) at `http://www.r-project.org`

The R Development Core Team (2008) releases an updated version of R about every six months. That is, as of March, 2009, the current version of 2.8.1 will be replaced with 2.9.0 sometime in April. Bug fixes are then added with a sub version number (e.g. 2.8.1 fixed minor problems with 2.8.0). It is recommended to use the most up to date version, as it will incorporate various improvements and operating efficiencies. Although many run R as a language and text oriented programming environment, there are GUIs available for PCs, Linux and Macs. See for example, *R Commander* by John Fox or *R-app* for the Macintosh developed by Stefano Iacus and Simon Urbanek. Compared to the basic PC environment, the Mac GUI is to be preferred.

R is an integrated, interactive environment for data manipulation and analysis that includes functions for standard descriptive statistics (means, variances, ranges) and also includes useful graphical tools for Exploratory Data Analysis. In terms of inferential statistics R has many varieties of the General Linear Model including the conventional special cases of Analysis of Variance, MANOVA, and linear regression. Statisticians and statistically minded people around the world have contributed **packages** to the R Group and maintain a very active news group offering suggestions and help. The growing collection of **packages** and the ease with which they interact with each other and the core R is perhaps the greatest advantage of R. Advanced features include correlational **packages** for multivariate

[1] Free as in speech rather than as in beer. See `http://www.gnu.org`

[2] GNU's Not Unix

analyses including Factor and Principal Components Analysis, and cluster analysis. Advanced multivariate analyses **packages** that have been contributed to the R-project include one for Structural Equation Modeling (**sem**, Hierarchical Linear Modeling (referred to as non linear mixed effects in the **nlme4** package) and taxometric analysis. All of these are available in the (>1400) free **packages** distributed by the R group at CRAN. Many of the functions described in this book are incorporated into the **psych** package. Other **packages** useful for psychometrics are described in a task-view at CRAN. In addition to being a environment of prepackaged routines, R is a interpreted programming language that allows one to create specific functions when needed.

R is also an amazing program for producing statistical graphics. A collection of some of the best graphics is available at the webpage addictedtoR with a complete gallery of thumbnail of figures.

3.2 How to get it: CRAN (Comprehensive R Archive Network)

Although it is possible that your local computer lab already has R, it is most useful to do analyses on your own machine. In this case you will need to download the R program from the R project and install it yourself. Go to the R home page at `http://www.r-project.org` and then choose the Download from CRAN (Comprehensive R Archive Network) option. This will take you to list of mirror sites around the world. You may download the Windows, Linux, or Mac versions at this site. For most users, downloading the binary image is easiest and does not require compiling the program.

3.3 Packages and Task Views

One of the advantages of R is that it can be supplemented with additional programs that are included as *packages* using the `package manager.` (e.g., *sem* does structural equation modeling) or that can be added using the `source` command. Most packages are directly available through the CRAN repository. Others are available at the BioConductor `http://www.bioconductor.org` repository. Yet others are available at "other" repositories. The *psych* package Revelle (2009) may be downloaded from CRAN or from the `http://personality-project.org/r` repository. The concept of a "task view" has made downloading relevant packages very easy. For instance, the `install.views("psychometrics")` command will download over 20 packages that do various types of psychometrics.

For any other than the default packages to work, you must activate it by either using the Package Manager or the `library` command:

- e.g., `library`(**psych**) or `library`(**sem**)

- entering ?**psych** will give a list of the functions available in the **psych** package as well as an overview of their funtionality.
- `objects(package:psych)` will list the functions available in a package (in this case, **psych**).

3.4 Help and Guidance

R is case sensitive and does not give overly useful diagnostic messages. If you get an error message, don't be flustered but rather be patient and try the command again using the correct spelling for the command.

When in doubt, use the `help(somefunction)` function. This is identical to `?` somefunction where some function is what you want to know about. e.g.,
`?read.table` #ask for help in using the read.table function – see the answer in the `help` window, or
`help(read.table)` #another way of asking for help. - see the `help` window

`RSiteSearch`("keyword") will open a browser window and return a search for "keyword" in all functions available in Rand the associated packages as well (if desired) the R-Help News groups.

3.5 Package vignettes

All packages have help pages for each function in the package. These are meant to help you use a function that you already know about, but not to introduce you to new functions. An increasing number of packages have a package "vignettes" that give more of an overview of the program than a detailed description of any one function. These vignettes are accessible from the help window and sometimes as part of the help index for the program. The two vignettes for the **psych** package are also available from the personality project web page. (An overview of the psych package and Using the psych package as a front end to the sem package).

3.6 Basic R commands and syntax

There are more than 10,000 possible one line commands that one can enter when using R [3] and no one can be expected to know them all. Even those of us who write packages need help remembering the possible commands and syntax of our own packages. In fact,

[3]This is based on the observation that there are > 1500 packages and that each package probably has at least 6 functions.

the Rpad handout is just one of many ways of remembering the appropriate command for core R. Even so, the basic concept of all commands is fairly easy to grasp in terms of the following simple analogy: R is just a fancy calculator that draws graphics and has built in statistic tables.

3.6.1 R is just a fancy calculator

One can think of R as a fancy graphics calculator. Enter a command and look at the output. Thus,

```
> 2 + 2
[1] 4
> 3^4
[1] 81
> pi
[1] 3.141593
```

In the above example, the > symbol is the R prompt and the next line [1] is the answer. When copying a line like this, do not include the > symbol. The # symbol is used to add comments to lines (and will not show when running R to prepare documents!). It is helpful to use a text editor (perhaps the one available in R, perhaps another one) to write the commands out before copying them into R. The up arrow command will echo the previous command on the terminal and allow for editing.

At the abstract level, almost all operations in R consists of executing a function on an object. The result is a new object. This very simple idea allows the output of any operation to be operated on by another function.

Command syntax tends to be of the form:
`variable = function (parameters)` or
`variable <- function (parameters)`
The = and the <- symbol imply replacement, not equality. The preferred style is to use the <- symbol to avoid confusion with the test for equality (==).

The result of an operation will not necessarily appear unless you ask for it. The command

```
x <- c(1, 3, 5, 7)
m <- mean(x)
```

will create the vector x made up of the numbers 1, 3, 5, 7, and set m equal to the mean of x but will not print anything on the console without the additional request
`m`.

however, just asking `mean(x)`
will find the mean and print it.

```
> x <- c(1, 3, 5, 7)
> m <- mean(x)
> m
[1] 4
> mean(x)
[1] 4
> sd(x)
[1] 2.581989
```

In addition to simple arithmetic, R allows you to create vectors or matrices and do operations on these matrices. The first example forms the vector V made up of the numbers from 1 to 10. The second finds the 3 x 5 matrix, m, made up of randomly chosen numbers sampled (with replacement) from the numbers 0-9. In this later example, to make the example replicable for the reader, the random number seed is set to a well known but arbitrary value (Adams, 1980). Using the fancy desk calculator ability of R, the last operation is a matrix operation that finds the sum of the cross products of M. That is, the sum of the squares of the columns of M (the diagonals) and the sum of the products of each column with each other column.

```
> set.seed(42)
> V <- seq(1:5)
> M <- matrix(sample(5, 15, replace = TRUE), ncol = 3, nrow = 5)
> V
[1] 1 2 3 4 5
> M
     [,1] [,2] [,3]
[1,]    5    3    3
[2,]    5    4    4
[3,]    2    1    5
[4,]    5    4    2
[5,]    4    4    3
> V * M
```

```
     [,1] [,2] [,3]
[1,]    5    3    3
[2,]   10    8    8
[3,]    6    3   15
[4,]   20   16    8
[5,]   20   20   15
```

```
> t(M) %*% M
```

```
     [,1] [,2] [,3]
[1,]   95   73   67
[2,]   73   58   50
[3,]   67   50   63
```

3.6.2 R as a graphing calculator

Suppose we want to present a graph comparing two normal distributions with different means and sigmas:

3.6.3 R is also a statistics table

It has been suggested by some that you should never buy a statistics book that has probability tables in it, because that means that the author did not know about modern statistics and the various distributions in R. Many statistics books include tables of the t or F or χ^2 distribution. By using R this is unnecessary since these and many more distributions can be obtained directly. Consider the normal distribution as an example. `dnorm(x, mean=mu, sd=sigma)` will give the probability density of observing that x in a distribution with mean=mu and standard deviation= sigma. `pnorm(q,mean=0,sd=1)` will give the probability of observing the value q or less. `qnorm(p, mean=0, sd=1)` will give the quantile value of a value with probability p. `rnorm(n,mean,sd)` will generate n random observations sampled from the normal distribution with specified mean and standard deviation. Thus, to find out what z value has a .05 probability we ask for `qnorm(.05)`. Or, to evaluate the probability of observing a z value of 2.5, specify `pnorm(2.5)`. (These last two examples are one side p values).

Applying these prefixes (d,p,q, r) to the various distributions available in R allows us to evaluate or simulate many different distributions (Table 1).

Consider the following examples.

```
> pt(2, 6)
```

```
> curve(dnorm(x, 1, 0.5), -3, 3, ylab = "Probability of x", main = "Comparing two distribut
> curve(dnorm(x, 0, 1), add = TRUE)
```

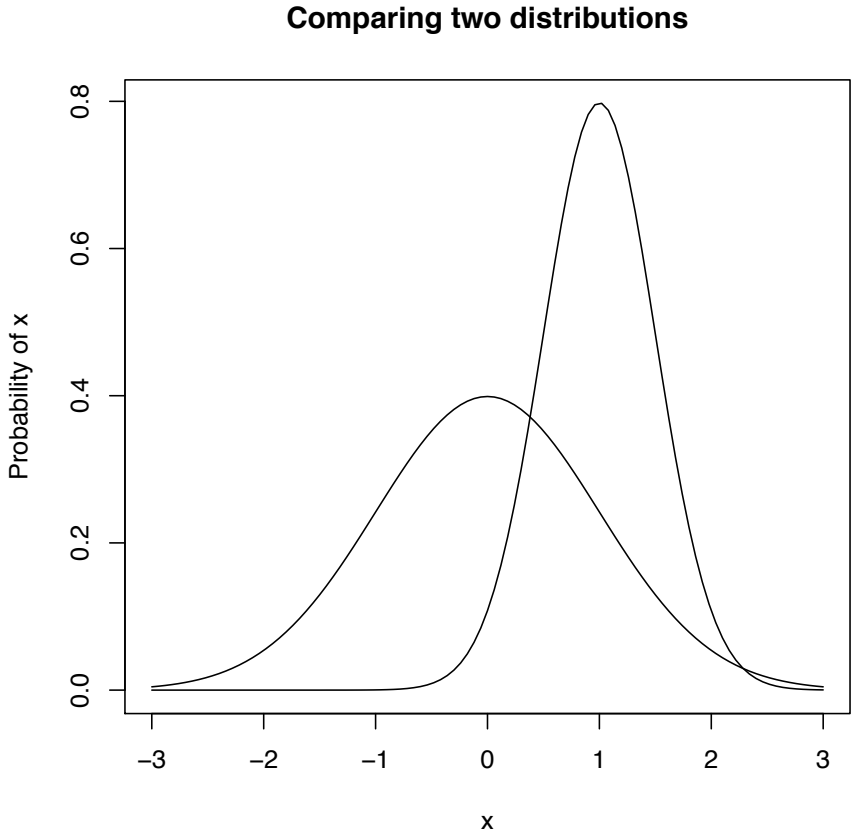

Figure 1: Two normal distributions drawn using the curve function.

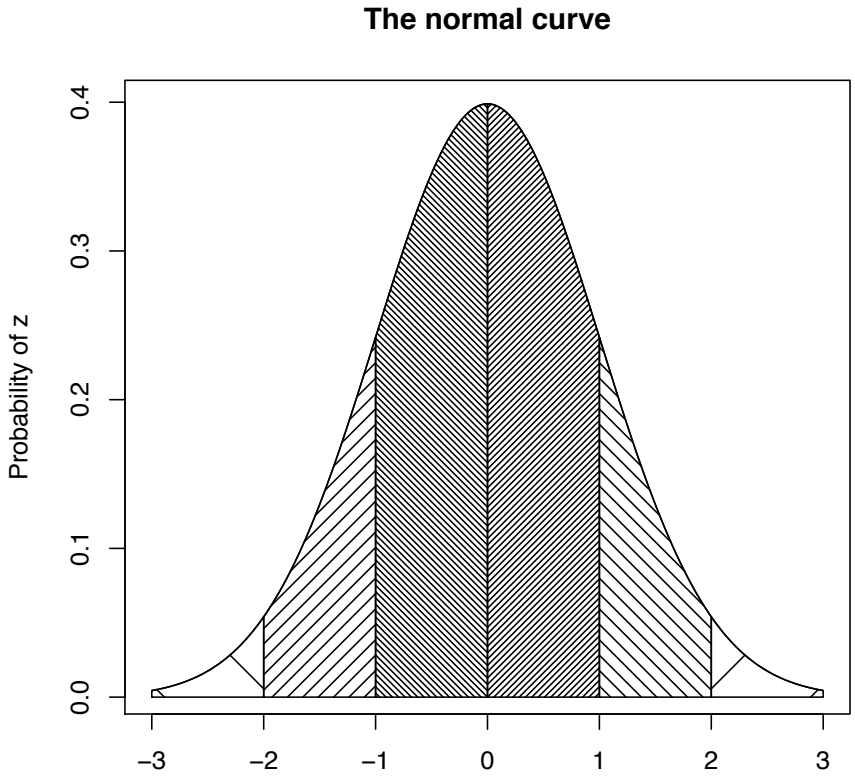

Figure 2: The normal curve with various colorings is a nice example of a simple but useful graphic.

```
[1] 0.9537868
> pnorm(2)
[1] 0.9772499
> dnorm(-1)
[1] 0.2419707
> pf(3.5, 1, 20)
[1] 0.923926
> qf(0.95, 1, 60)
[1] 4.001191
> qchisq(0.95, 1)
[1] 3.841459
```

3.6.4 R will make up data

Although making up data is normally considered a bad thing for a researcher to do, when we call it "simulation" it is considered scientific. All of the distributions listed in Table 1 can be prefaced with "r" to create (pseudo) random data with that particular shape. Consider the following example where the data are generated and then their histogram is drawn (Figure 3).

This ability to simulate data is particularly useful when teaching statistics, or when trying out a new method. For if you know what the underlying model is, it is easier to understand what how well the method works. When teaching about distributions, it is useful to show what happens if we take progressively larger samples of the data. This is shown in Figure 4 where we plot the means of samples of size 1, 2, 4, and 8. This uses the `replicate` and `rowMeans` functions. The last two panels show how to combine multiple commands into one line.

4 Entering or getting the data

For most data analysis, rather than manually enter the data into R, it is probably more convenient to use a spreadsheet (e.g., Excel or OpenOffice) as a data editor, save as a tab or comma delimited file, and then read the data from the file. Many of the examples in this tutorial assume that the data have been entered this way. Many of the examples in

Table 1: Some of the most useful distributions for psychometrics that are available as functions. To obtain the density, prefix with *d*, probability with *p*, quantiles with *q* and to generate random values with *r*. (e.g., the normal distribution may be chosen by using dnorm, pnorm, qnorm, or rnorm.) Each function has specific parameters, some of which take default values, some of which require being specified. Use *help* for each function for details.

Distribution	base name	P 1	P 2	P 3	example application
Normal	norm	mean	sigma		Most data
Multivariate normal	mvnorm	mean	r	sigma	Most data
Log Normal	lnorm	log mean	log sigma		income or reaction time
Uniform	unif	min	max		rectangular distributions
Binomial	binom	size	prob		Bernuilli trials (e.g. coin flips)
Student's t	t	df		nc	Finding significance of a t-test
Multivariate t	mvt	df	corr	nc	Multivariate applications
Fisher's F	f	df1	df2	nc	Testing for significance of F test
χ^2	chisq	df		nc	Testing for significance of χ^2
Beta	beta	shape1	shape2	nc	distribution theory
Cauchy	cauchy	location	scale		Infinite variance distribution
Exponential	exp	rate			Exponential decay
Gamma	gamma	shape	rate	scale	distribution theoryh
Hypergeometric	hyper	m	n	k	
Logistic	logis	location	scale		Item Response Theory
Poisson	pois	lambda			Count data
Weibull	weibull	shape	scale		Reaction time distributions

```
> op <- par(mfrow = c(2, 2))
> n <- 1000
> x <- rnorm(n)
> hist(x, main = "Normal")
> x <- runif(n)
> hist(x, main = "Rectangular")
> x <- rpois(n, 3)
> hist(x, main = "Poisson")
> x <- rlnorm(n)
> hist(x, main = "Log Normal")
> op <- par(mfrow = c(1, 1))
```

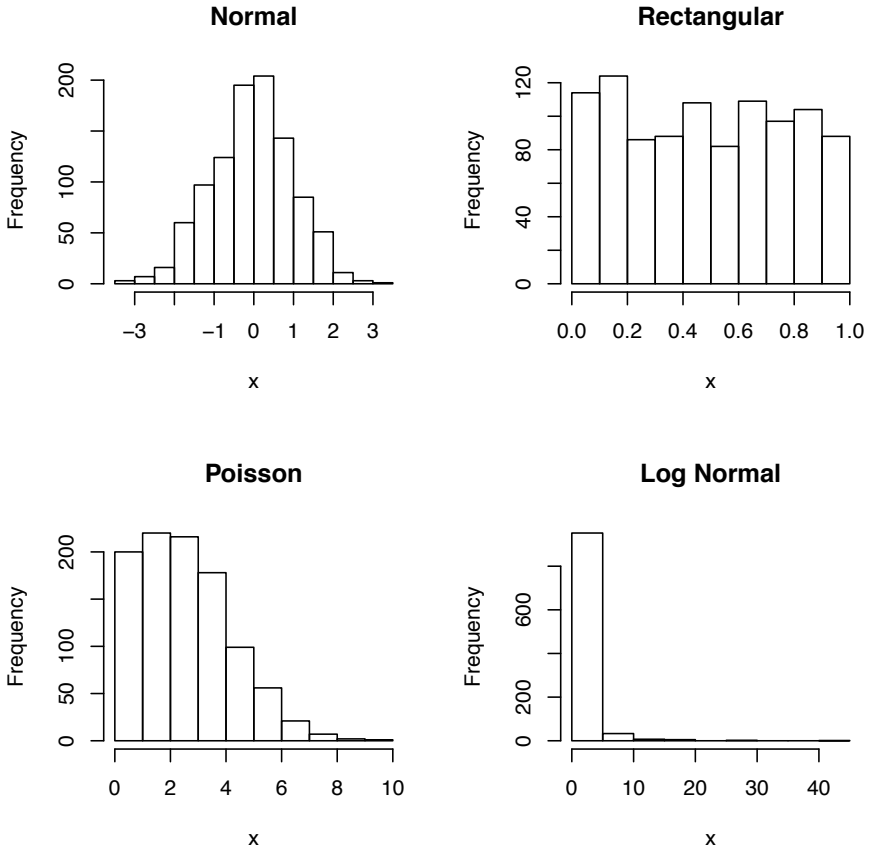

Figure 3: Histograms of four different random distributions, the normal (rnorm), the rectangular or uniform (runif), the Poisson (rpois), and the lognormal (rlnorm). The first op command specifies that we want a 2 x 2 plot, the second one returns us to the normal 1 x 1 plot.

```
> op <- par(mfrow = c(2, 2))
> n <- 1000
> x <- runif(n)
> hist(x, main = "1 case")
> x <- rowMeans(replicate(2, runif(n)))
> hist(x, main = "2 cases", xlim = c(0, 1))
> hist(rowMeans(replicate(4, runif(n))), main = "4 cases", xlim = c(0, 1))
> hist(rowMeans(replicate(8, runif(n))), main = "8 cases", xlim = c(0, 1), xlab = "Mean of
> op <- par(mfrow = c(1, 1))
```

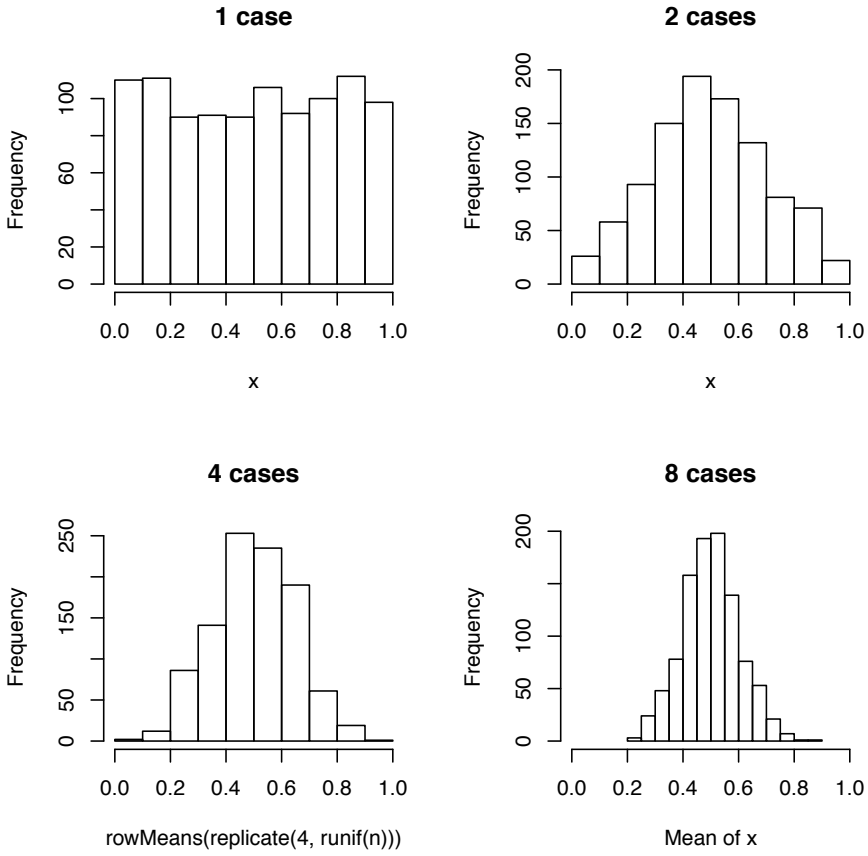

Figure 4: Histograms of the means of four different sample sizes where the samples are taken from a uniform distribution. This makes use of the replicate and rowSums functions, as well as the runif and hist functions.

the help menus have small data sets entered using the c() command or created on the fly. It is also possible to read data in from a remote file server. Alternatively, if you have data in a SAS or SPSS file, there are functions in the *foreign* package to import them.

Using the `copy.clipboard()` function from the **psych** package, it is also possible to have a data file open in a text editor or spreadsheet program, copy the relevant lines to the clipboard, and then read the clipboard directly into R.

Finally, many packages include example data sets that can be accessed directly using the `data` command. Thus, to get a number of factor analysis examples with a bifactor structure, the `bifactor` data set is called to make the seven data sets within it available.

4.1 Getting data from a remote file server

For the first example, we read data from a file server in the Personality-Motivation-Cognition lab at Northwestern University that contains the responses for several hundred subjects on 13 personality scales (5 from the Eysenck Personality Inventory (EPI), 5 from a Big Five Inventory (BFI) , one Beck Depression, and two anxiety scales). The data are taken from a study in the Personality, Motivation, and Cognition Laboratory. The file is structured normally, i.e. rows represent different subjects, columns different variables, and the first row gives subject labels. Had we saved this file as comma delimited, we would add the separation (sep=",") parameter.

```
#specify the name and address of the remote file
> datafilename <- "http://personality-project.org/r/datasets/maps.mixx.epi.bfi.data"
#now read the data file
> my.data <-  read.table(datafilename,header=TRUE)  #read the data file
```

4.2 Getting data from a local file

More typically, the data are stored somewhere on your computer in a tab delimited or comma delimited file. The process is equally easy, in that you first locate the file and then read it.

```
> datafilename <- file.choose()  #where you dynamically can go to find the file
> my.data <-  read.table(datafilename,header=TRUE)  #read the data file
```

4.3 Copying the data from the clipboard

Yet another alternative is to directly access the data file outside of R, copy the data to the clipboard, and then read the clipboard using the `read.clipboard` function.

```
> my.data <- read.clipboard()  # if there are complete cases and each column has an identif
> my.data <- read.clipboard(header=FALSE)  # if there are complete cases and columns do not
> my.data <- read.clipboard(sep="t") #if the data come from a spreadsheet with blank cells
> my.data <- read.clipboard.csv() #if the data were copied from a comma delimited file
```

4.4 Reading from an SPSS file

To read from an SPSS or SAS data file, it is necessary to first load the *foreign*.

```
> library(foreign)
> my.spss.file.name  <- file.choose()  #where you dynamically can go to find the file
> my.data <- read.spss(my.spss.file.name,to.data.frame=TRUE)
```

4.5 Getting the data from a built in data set

Because we want to demonstrate the same data set many times and the user is not necessarily connected to the internet, many packages have built in data sets. A list of all available sets can be found by the `data()` command. The data sets within a package are found by specifying the package. For the next examples, we use the epi.bfi data set.

```
> data(package = "psych")
> data(epi.bfi)
> my.data <- epi.bfi
```

The data are now in the `data.frame` "my.data". Data.frames allow one to have columns that are either numeric or alphanumeric. They are conceptually a generalization of a matrix in that they have rows and columns, but unlike a matrix, some columns can be of different "types" (integers, reals, characters, strings) than other columns. But how do you know they are there? R is a somewhat reticent program and will not give some affirmative message that it has worked, but just do it. The functions `dim` and `names` can be used to find out how many variables and cases were read, and what their names are.

```
> dim(my.data)

[1] 231  13

> names(my.data)

 [1] "epiE"     "epiS"     "epiImp"   "epilie"   "epiNeur"  "bfagree"  "bfcon"    "bfext"
[12] "traitanx" "stateanx"
```

4.6 Entering data manually–understanding data structures

Everything in R is an *object*. Some of these objects are *functions*, some are the *results of functions*. Thinking very abstractly, the process of analysis is to apply some function to some object and return a new object. Knowing the structures of these objects allows a better understanding of how to use them. In particular, if it is necessary to enter data manually, it is useful to know the various types of data structures .

4.6.1 Data structures

elements : These are single values which may be integers, reals, logicals, factors, or character (strings). They are actually thought of as vectors of length one and have no dimensions.

vectors These are the basic object in R and are ordered sets of values. They have length and are of dimension one. They may be formed by the concatenation function `c`. E.g., x <- c(1,4, 6), y <- c("apples","oranges"), z <- c(TRUE, FALSE, TRUE). Elements of a vector may be addressed by location (x[2] has the value of 4).

matrices Matrices are just vectors of vectors. That is to say, they are two dimensional arrays where the elements are all of the same type. Elements may be addressed by location (X[i,j] is the element in the ith row and the jth column of X). More useful, X[i,] is the entire ith column, X[,j] is the entire jth row.

data.frames A data.frame appears to be the same as a matrix, but may have columns of different types. Each column must be of the same length. Elements may be addressed by location (X.df[i,j])

lists The most general way of aggregating objects. Lists have members of the list, each member may itself be a list, matrix, vector or element.

Below are examples of creating vectors (x, y, z), matrices (X, Y), a data.frame (y.z.df), and then finally, a list (L) made up of all of the prior objects.

```
> x <- c(1, 2, 4)
> y <- c(letters[1:6], LETTERS[1:4])
> z <- seq(10, 28, 2)
> X <- matrix(1:20, ncol = 4)
> Y <- matrix(c(11, 22, 44, 4, 15, 42), ncol = 3, byrow = TRUE)
> yz.df <- data.frame(A = y, b = z)
> L <- list(a = x, b = y, c = z, d = X, e = Y, f = yz.df)
> x
[1] 1 2 4
```

```
> y
 [1] "a" "b" "c" "d" "e" "f" "A" "B" "C" "D"
> z
 [1] 10 12 14 16 18 20 22 24 26 28
> X
     [,1] [,2] [,3] [,4]
[1,]    1    6   11   16
[2,]    2    7   12   17
[3,]    3    8   13   18
[4,]    4    9   14   19
[5,]    5   10   15   20
> Y
     [,1] [,2] [,3]
[1,]   11   22   44
[2,]    4   15   42
> yz.df
   A  b
1  a 10
2  b 12
3  c 14
4  d 16
5  e 18
6  f 20
7  A 22
8  B 24
9  C 26
10 D 28
> L
$a
[1] 1 2 4

$b
 [1] "a" "b" "c" "d" "e" "f" "A" "B" "C" "D"

$c
```

```
 [1] 10 12 14 16 18 20 22 24 26 28

$d
     [,1] [,2] [,3] [,4]
[1,]    1    6   11   16
[2,]    2    7   12   17
[3,]    3    8   13   18
[4,]    4    9   14   19
[5,]    5   10   15   20

$e
     [,1] [,2] [,3]
[1,]   11   22   44
[2,]    4   15   42

$f
   A  b
1  a 10
2  b 12
3  c 14
4  d 16
5  e 18
6  f 20
7  A 22
8  B 24
9  C 26
10 D 28
```

Thinking analogically, the data.frame is the most similar to the standard spreadsheet way of organizing data, and most statistical analysis will make use of data.frames or of matrices. The list structure is a particularly appealing way of storing the results of any particular analysis, for it can hold different types of information as one, high level, object.

If you do know, or can remember the structure of a particular object, then the `str` function will retrieve it for you. This is particularly useful when running a complex statistical analysis, for only some results will be shown, even though far more information is hidden in the object.

```
> str(L)

List of 6
 $ a: num [1:3] 1 2 4
 $ b: chr [1:10] "a" "b" "c" "d" ...
```

```
 $ c: num [1:10] 10 12 14 16 18 20 22 24 26 28
 $ d: int [1:5, 1:4] 1 2 3 4 5 6 7 8 9 10 ...
 $ e: num [1:2, 1:3] 11 4 22 15 44 42
 $ f:'data.frame':        10 obs. of  2 variables:
  ..$ A: Factor w/ 10 levels "a","A","b","B",..: 1 3 5 7 9 10 2 4 6 8
  ..$ b: num [1:10] 10 12 14 16 18 20 22 24 26 28
```

4.6.2 Entering data into a data.frame

A data.frame is just a collection of objects, each of the same length where subjects are rows and variables are columns. Thus, an experiment might have one or more condition variables, and one or more outcome variables. Consider a simple example of two conditions (control vs. experimental) and a measured variable.

```
> condition = c("e", "e", "e", "c", "c", "c")
> result <- c(2, 3, 4, 1, 2, 3)
> my.data <- data.frame(condition, result)
> my.data
```

```
  condition result
1         e      2
2         e      3
3         e      4
4         c      1
5         c      2
6         c      3
```

4.7 Basic data manipulation

As one would expect, data can be selected, recoded, sorted, and merged using the appropriate commands.

4.7.1 Editing the data in a data.frame

To change just one or two values in small data frame or matrix, use the functions `edit` and `fix`.

```
x <- edit(my.data)  #will open an edit window and allow changes, which are then put into x,
fix(my.data)     #immediately changes my.data.
```

4.7.2 Modifying particular cases by a formula

An alternative to using the `edit` or `fix` functions is to operate on the data directly. Consider the data.frame my.data. Select those cases for which condition is equal to "e" and increase the values by 2

```
> my.data[my.data$condition == "e", 2] <- my.data[my.data$condition == "e", 2] + 2
> my.data
```

```
  condition result
1         e      4
2         e      5
3         e      6
4         c      1
5         c      2
6         c      3
```

This ability to manipulate data that meet certain logical conditions is very powerful, for it can be done to turn particular observations into missing data (NA), or to select just certain cases.

4.7.3 Selecting particular cases or conditions

Just as one can modify certain cases that meet certain conditions, so can one select just those cases for a new object.

```
> my.subset <- subset(my.data, my.data$condition == "e")
> my.subset
```

```
  condition result
1         e      4
2         e      5
3         e      6
```

4.7.4 Sorting the data by a particular variable

A frequently asked question is how to sort the data file according to some criterion. Consider the data.frame, people, made up of names and numbers. The order of names can be found by the `order` function, and then a new data.frame can be created using these orders.

```
> names <- c("Roger", "Ellen", "Anne", "Mary", "Carolyn")
> numb <- c(9, 10, 32, 35, 39)
> people <- data.frame(names, numbers = numb)
> people

    names numbers
1   Roger       9
2   Ellen      10
3    Anne      32
4    Mary      35
5 Carolyn      39

> sorted <- people[order(people$names), ]
> sorted

    names numbers
3    Anne      32
5 Carolyn      39
2   Ellen      10
4    Mary      35
1   Roger       9
```

4.7.5 Merging two data.frames

Suppose we have another data.frame, gender, that catalogs the genders of the same people. To merge these two data.frames together, use the `merge` function specifying the variable to use to combine the two data.frames.

```
> names <- c("Roger", "Ellen", "Anne", "Mary", "Carolyn")
> gen <- c("M", rep("F", 4))
> gender <- data.frame(names, gender = gen)
> gender

    names gender
1   Roger      M
2   Ellen      F
3    Anne      F
4    Mary      F
5 Carolyn      F

> merge(gender, people, by = "names")

    names gender numbers
1    Anne      F      32
```

```
2 Carolyn      F      39
3   Ellen      F      10
4    Mary      F      35
5   Roger      M       9
```

Using this technique, we routinely merge data files of 40-60,000 records with 300 variables with other files with 100 variables.

5 Basic descriptive statistics

Basic descriptive statistics are most easily reported by using the `summary`, `mean` and Standard Deviations (`sd`) commands. Using the `describe` function available in the **psych** package produces output more useful to most psychologists. Graphical displays that also capture the data are available as a boxplot.

```
> describe(my.data)
```

```
           var n mean   sd median trimmed  mad min max range skew kurtosis   se
condition*   1 6  1.5 0.55    1.5     1.5 0.74   1   2     1    0    -2.31 0.22
result      2 6  3.5 1.87    3.5     3.5 2.22   1   6     5    0    -1.80 0.76
```

It is sometimes useful to report statistics by a particular group. This can be done using the `describe.by` function. We use a different data set `sat.act` which has self reported SAT Verbal, Quant and ACT scores for 700 participants collected on the personality-project.org web site. In addition, we have gender, education and age. The first `describe` is for all the cases, the second is broken down by gender (males=1, females=2). To make the output shorter, the option to not show the skews and kurtosi is set.

```
> data(sat.act)
> describe(sat.act, skew = FALSE)
```

```
          var   n   mean     sd median trimmed    mad min max range   se
gender      1 700   1.65   0.48      2    1.68   0.00   1   2     1 0.02
education   2 700   3.16   1.43      3    3.31   1.48   0   5     5 0.05
age         3 700  25.59   9.50     22   23.86   5.93  13  65    52 0.36
ACT         4 700  28.55   4.82     29   28.84   4.45   3  36    33 0.18
SATV        5 700 612.23 112.90    620  619.45 118.61 200 800   600 4.27
SATQ        6 687 610.22 115.64    620  617.25 118.61 200 800   600 4.41
```

```
> describe.by(sat.act, sat.act$gender, skew = FALSE)
```

```
$`1`
          var   n   mean     sd median trimmed    mad min max range   se
```

```
gender      1 247   1.00   0.00      1    1.00   0.00   1   1      0 0.00
education   2 247   3.00   1.54      3    3.12   1.48   0   5      5 0.10
age         3 247  25.86   9.74     22   24.23   5.93  14  58     44 0.62
ACT         4 247  28.79   5.06     30   29.23   4.45   3  36     33 0.32
SATV        5 247 615.11 114.16    630  622.07 118.61 200 800    600 7.26
SATQ        6 245 635.87 116.02    660  645.53  94.89 300 800    500 7.41

$`2`
          var   n   mean     sd median trimmed    mad min max range   se
gender      1 453   2.00   0.00      2    2.00   0.00   2   2      0 0.00
education   2 453   3.26   1.35      3    3.40   1.48   0   5      5 0.06
age         3 453  25.45   9.37     22   23.70   5.93  13  65     52 0.44
ACT         4 453  28.42   4.69     29   28.63   4.45  15  36     21 0.22
SATV        5 453 610.66 112.31    620  617.91 103.78 200 800    600 5.28
SATQ        6 442 596.00 113.07    600  602.21 133.43 200 800    600 5.38
```

`describe.by` is just an example of a more basic R function, `by`. As can be seen in the help page for `by`, an alternative way to do the preceding analysis is just

```
> by(sat.act, sat.act$gender, describe, skew = FALSE)

sat.act$gender: 1
          var   n   mean     sd median trimmed    mad min max range   se
gender      1 247   1.00   0.00      1    1.00   0.00   1   1      0 0.00
education   2 247   3.00   1.54      3    3.12   1.48   0   5      5 0.10
age         3 247  25.86   9.74     22   24.23   5.93  14  58     44 0.62
ACT         4 247  28.79   5.06     30   29.23   4.45   3  36     33 0.32
SATV        5 247 615.11 114.16    630  622.07 118.61 200 800    600 7.26
SATQ        6 245 635.87 116.02    660  645.53  94.89 300 800    500 7.41
------------------------------------------------------------------------------------------
sat.act$gender: 2
          var   n   mean     sd median trimmed    mad min max range   se
gender      1 453   2.00   0.00      2    2.00   0.00   2   2      0 0.00
education   2 453   3.26   1.35      3    3.40   1.48   0   5      5 0.06
age         3 453  25.45   9.37     22   23.70   5.93  13  65     52 0.44
ACT         4 453  28.42   4.69     29   28.63   4.45  15  36     21 0.22
SATV        5 453 610.66 112.31    620  617.91 103.78 200 800    600 5.28
SATQ        6 442 596.00 113.07    600  602.21 133.43 200 800    600 5.38
```