Yale University

StatLab
SOCIAL SCIENCE RESEARCH SERVICES

Introduction To R

Breanne Chryst
Sara Bastomski

February 19, 2016

Contents

Overview of Workshop

The purpose of this workshop is to provide a basic introduction to R, orienting you to the R environment and some basic things you can do within it. This is not intended to be a comprehensive introduction to R; instead, its purpose is to give you a focused, first-hand taste of R so you can see what it is like to do basic statistics within the R environment. A more detailed outline of the contents of this document is given below:

Section 1: What is R?

- A brief description of the R programming language – including advantages and drawbacks – installing R and R Studio.

Section 2: Introduction to the R Environment

- Issues relating to writing R code are discussed here. Ways of getting help in R are introduced.
- Discuss Rstudio, a user friendly IDE for R.

Section 3: An Introductory R workshop

- We demonstrate simple data-analysis tasks using R. Some basic issues relating to R-programming are discussed as they arise

Appendix:

- In sections A.1-A.7 we introduce some of the most common types of objects that we encounter as we learn to program with R. A.8 deals with downloading and using packages in R.

Prior Assumptions

Because this is a basic introductory workshop to R, we designed it with minimal assumptions about your statistical and computing background. So don't worry if you've never used R, programmed, or used a command line to do your statistics. We'll cover all you need to know right here! And, for ease of presentation and learning the software, we'll restrict our statistical examples to content that would be covered in any graduate or undergraduate introductory statistics course (e.g., descriptive statistics, ANOVA, regression). You should know, however, that R can do incredibly complex, customized statistical operations, often in just a few lines of code.

1 What is R?

1.1 The essence of R

R is both a statistical package (like SPSS, Stata, SAS, etc.) and a programming language. More accurately, R is an environment within which you can do statistical programming and graphics. Just as your operating system (Windows, Mac OS X, Unix, Linux, etc.) acts as an environment within which you can use software and hardware to manage files, R provides an environment and a language that allow you to compute, analyze, and graph your data - and so much more.

R is an object-oriented programming language. Everything is an object in R. Everything you do and create in R, including statistical models, functions, graphics, and output, is an object stored in memory that can be manipulated, saved, and reused for greater efficiency and power.

1.2 Why use R?

Ultimately, you will have to decide this for yourself, but here are a few reasons why many people are learning to use R:

- R is completely free software-and always will be!
- R runs on all major operating systems, so if you ever have to switch you'll still be able to use it.
- R is open source and completely transparent: if you really want to, you can access the source code for a statistical function-even R itself-and modify it to your liking without breaking any laws or having to pay a cent in licensing fees.
- R is highly customizable and extendible: R can probably do any kind of statistical analysis or create any kind of statistical graphic you can think of, and then some. There is a growing community of individuals who love R so much they spend their spare time contributing, modifying, and critiquing new statistical functions, as well as writing tutorials and giving workshops.
- R is constantly improving and up-to-date. It is actively maintained by a core team of statisticians and programmers who are responsible for rolling out regular updates to the program. This translates into greater flexibility and a kind of peer-review process to ensure that statistical functions operate with minimal error. And errors often are caught and fixed quickly, even more so now that the user base is growing rapidly.
- R's command-line interface allows you to work interactively with your data in a natural question and answer format, and scripts allow you to log and later replicate exactly what you did with your data (somewhat like syntax in SPSS, for example).

- R allows you to produce beautiful publication-quality graphics with very little effort.
- Compared to statistical programs with a primarily ‘point-and-click’ interface (e.g., SPSS), R’s command-line interface facilitates accurate and efficient communication of the process of statistical analysis. Entire statistical sessions can be saved in a text file called a script, a log of the commands you issued to R. Not only do scripts help you keep track of and replicate what you did with your data, but others with access to the same data can run the script in R and reproduce the same output. For this reason, it is easier for others to provide technical assistance with R, compared to a point-and-click interface, since the problem can easily be described and replicated through scripts as opposed to descriptions of mouse clicks. Later in the workshop, we will demonstrate the use of R-Studio as a text editor for writing R scripts.

1.3 Why not use R?

- Why reinvent the wheel? Depending on your statistical and graphical needs, R may provide little incremental utility beyond what you already can do efficiently in another statistical program. R may not be the best application for every kind of statistical analysis. It really depends on your needs and preferences.
- Learning curve. Depending on how comfortable you are with command line interfaces and programming languages, R may require a significant investment of time up front before you can use it efficiently. At the same time, people are often surprised at how intuitive the translation from a statistical question to the appropriate command is within the R language. And an increasing number of undergraduate students are being exposed to R through intro-level statistics courses.

1.4 Installing R

R can be freely downloaded from the site `www.r-project.org`. Click on the link for CRAN in the left hand frame and select a URL from which to begin the download. Then select the version of R appropriate for your operating system from the links under ‘precompiled binary downloads’.

R-Studio can be freely downloaded from the site `https://www.rstudio.com`. R-Studio is an open source IDE (integrated development environment) that provides a user-friendly interface for R. We will be using R-Studio for the in class examples today.

2 Introduction to the R Environment

2.1 Getting Help

Getting help is easy in R, and it’s one of the first things you should get used to doing as you learn on your own.

- help(function): Open help file on function within R. For example, help(t.test) opens help on the function for performing t-tests. Alternatively we could write "?function".
- help.start(): Opens help within a HTML() browser. This can be advantageous over method mentioned above, since we are able to search within this browser.
- help.search("string") Returns functions that contain what you type in for "string" in their help files. This is a great tool for finding what help topics are available when you don't know the exact name of the function you're looking for help on.
- `stackoverflow.com`: "Stack Overflow is a question and answer site for professional and enthusiast programmers. It's 100% free." You can post questions or see if someone else has already asked your question. Excellent resource!

2.2 Other Resources

Due to an explosion of interest in R across the world, excellent references, tutorials, and other material are appearing nearly every month on the web and in print. If you plan to use R at all after this workshop, it will be well worth your time to consult one or more of these resources (which are free, unless otherwise noted).

2.2.1 Online resources

- `http://www.r-project.org` This should be your very first step for all things related to R, including obtaining it, downloading the manuals, finding support through the R listsery, and finding links to free online resources for learning R.
- `http://www.r-project.org/other-docs.html` A collection of online resources ranging from introductory texts to R to advanced statistical procedures and graphics within R.
- **R Site Search:** (website hosted by Jonathan Baron at UPenn) Lets you search all help files, documentation, functions, and R-help listsery archives:

  `http://finzi.psych.upenn.edu/search.html`
- **R-Commander:** `http://socserv.mcmaster.ca/jfox/Misc/Rcmdr/index.html` (by John Fox, McMaster University) R-Commander is a graphical front-end to R. It allows you to click through menus and dialogue boxes and produces the necessary R code to complete the task. It also allows you to submit your own code interactively. It's a great way to learn how to use R to do your own statistics, as you can study the code resulting from your menu selections (much like clicking on 'Paste' in SPSS to learn the syntax).

- **Quick-R:** `http://www.statmethods.net/` (by Robert I. Kabacoff) This website is for R users and also for the experienced users of other statistical packages (e.g., SAS, SPSS, Stata) who would like to transition to R. This website provides easy-to-use code for various things we can do with R with explanation and graphics.

2.2.2 Books

- Dalgaard, Peter J. (2002). *Introductory statistics with R.* A clear and concise introductory text that already is becoming a classic among new R users. It focuses on how to use R to do statistical analyses rather than on R programming per se. Best of all, Yale offers an electronic version of this book for free on Ebrary. Just search for the book on Orbis and click on the link to the electronic resource (NB: Ebrary is not accessible through Linux at present).
- Crawley, Michael J. (2007) *The R Book.* An in-depth easy-to-use reference book covering a wide variety of graphical and modeling procedures in R. A copy is kept in the consultant's office at the Statlab.
- Adler, Joseph (2010) *R in a Nutshell.* A good introductory book. A copy is kept at the consultants' office at the Statlab.

3 An Introductory R Session

The code for the following session is available for download at: `github.com/bchryst/IntroductionToR2016`. Before we start

- Please use a plain text editor (Notepad, TextEdit, and etc) to write your codes; advanced editing tools like Word will have undesired format screwing up your codes.
- Do not save your 'workspace'; save your codes. Also, you could save results in data files or copy and paste output from R console to a text file.

3.1 Reading in data

The first thing we need to do is read our data set into R. Suppose we have stored the data in the folder:

`C:\Users\NetID\Desktop\IntroductionToR2016-master.`

```
getwd() # Look at the current directory.
```

Method 1: Click by mouse.

- Windows: File->Change dir

- Mac: Misc->Change Working Directory...
- RStudio: Tools->Set Working Directory->Choose Directory

To check:

```
getwd() # Just to check that the working directory has changed.
```

Method 2:

```
setwd("C:\Users\NetID\Desktop\IntroductionToR2016-master")
getwd()
```

Note that R ignores text after a '#' sign. This is a good way to add comments into your code. Let's use the read.table command. We specify that the first row contains column names (header = T), and that the data is white space separated set="". The option row.names=NULL indicates that the rows of the data set are not named; in this case, R will number the rows.

```
data <- read.table("remote_weight.txt", header=T, sep="",
                   row.names=NULL)
```

For 'csv' files (one type of Excel files), use read.csv command; for more complicated data formats would be to use scan(), but we will not need that command here.

3.2 Extracting data from the data frame

The default way of storing data in R is with a data frame. This is like a generalized matrix where the columns can be either text or numeric. The output from the read.table command is a data frame which we save as an object imaginatively titled 'data'. Note the use of the assignment operator '<-' in the above expression (alternatively, one can use a single equals sign, "="). Some easy manipulations one can perform with a data frame are listed below.

```
dim(data)
names(data)
str(data)
data
data[1:10,]
data[,"weight"]
data[,3]
data$weight
data[1:10,"weight"]
# See only the first 10 values of the weight column.
data[,-1] # See all EXCEPT the first column of data
data.o <- data # Backup data frame
```

More commands for data frames are listed in the appendix. Descriptive Statistics and Graphics One of the most useful commands in R is summary(). Depending on the object one is summarizing, the command will do different things. For a data frame, it gives the mean for each variable and the 5 number summary of that variable: (minimum, first quantile, median, third quantile, and maximum).

```
summary(data)
id remote weight gender
Min. : 1.00 Min. : 0.00 Min. :119.0 Min. :0.0
1st Qu.: 25.75 1st Qu.: 5.00 1st Qu.:139.8 1st Qu.:0.0
Median : 50.50 Median :14.00 Median :152.0 Median :0.5
Mean :50.50 Mean :17.59 Mean :156.4 Mean :0.5
3rd Qu.: 75.25 3rd Qu.:30.00 3rd Qu.:174.3 3rd Qu.:1.0
Max. :100.00 Max. :37.00 Max. :203.0 Max. :1.0
```

Of course, commands exist to extract only the sample standard deviations or similar statistics:

```
sd(data) # Calculate standard deviations of all variables
var(data) # Variance on diagonal, covariance off diagonal
mean(data) # Calculate the mean of all variables
```

A useful function to graphically summarize all variables simultaneously is pairs().

```
pairs(data[,-1], main="Pairwise␣scatter␣plots␣of␣variables␣except␣id")
```

Notice that the use of '[,-1]' above. We are in fact asking R to make pair-wise scatterplots using all columns except column 1 (id). As a variation, we could have also used '[,c(2,3,4)]' to achieve the same effect. The argument main allows us to give a title to the plot. We can also produce scatterplots of individual variables using the plot function. Note how we use '$' to access specific (named) columns of data.

```
plot(data$weight, data$remote) # Scatterplot of 'weight' vs. 'remote'.
plot(data[,2], data[,3]) # Another way of making the same plot.
plot(data$weight, data$remote, xlab="Weight",
    ylab="Remote", main="Scatter␣Plot␣of␣Weight␣and␣Remote",
    col="green") # Use more arguments.
dim(data) # Find out how many rows and columns are in the data set.
names(data) # List all variable names in the dataset .
str(data) # Look at the structure of your data.
data # See the data frame on the screen.
data[1:10,] # See the first 10 rows.
data[,"weight"] # See only the weight column.
data[,3] # Same as above.
data$weight # Yet another way
```

By 'attaching' the data frame, we can directly refer to the variables, without the need to use '$'.

```
attach(data) # Attach the data frame
```

Note that now we can ask R to print out an object by using just the object name. For example:

```
remote
```

will print out the data entries for the variable remote. Since we have just attached the data frame, we no longer need to use the more long-winded:

```
data$remote.
```

If you wanted to stop using this dataset and attach a different one, you can use the command

```
detach(data).
```

More examples of basic Graphics Histograms - use the hist() command:

```
par(mfrow=c(2,1))
# Allows us to display 2 graphs on the same plotting window
hist(remote, main = "distribution of remote usage")
hist(weight, main = "distribution of weight usage")
```

Boxplots can be created with the boxplot() command

```
par(mfrow=c(1,1)) # Change plotting window back to default
boxplot(remote, weight) # Boxplot of 'remote' and 'weight'
boxplot(remote ~ gender) # Boxplot of 'remote' conditioned on 'gender'
boxplot(weight ~ gender) # Boxplot of 'weight' conditioned on 'gender'
```

The last two commands here generate separate male and female boxplots for the variables weight and gender.

3.3 Inferential statistics

We briefly illustrate some of the most basic inferential statistics techniques in R

```
cor(remote, weight) # Find the correlation coefficient
[1] 0.8116673
t.test(remote ~ gender)
# Test if frequency of remote usage differs by gender
Welch Two Sample t-test
data: remote by gender
t = -31.3195, df = 84.845, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
```

```
-25.92777 -22.83223
sample estimates:
mean in group 0 mean in group 1
5.40 29.78
```

Note that the result of any analysis in R is technically an object which can be stored in the workspace. Let us save the t.test as a variable named 'rem.t'

```
rem.t <- t.test(remote ~ gender) # Save results of last analysis
```

R stores rem.t as a type of list – a collection of named objects (See the appendix for more details). One can see the names of these sub-objects by using the names() function. To access any sub-object we use the operator '$'. For example, the realized value of the t–statistic is one such sub-object; to find its value use the command

```
rem.t$statistic
names(rem.t) # See the names of variables in the object rem.t
rem.t$statistic # Get the value of the t-statistic
[1] -31.31951
```

The p-value is another available sub-object.

```
rem.t$p.val
[1] 2.139944e-48
```

3.4 Regression commands

We use the command

```
lm()
```

in R to fit linear regression models. Remember again that R creates an object (this time an lm or linear model object) when we use this command. To manipulate the results of the regression, it is best if we create a variable to store this object. For example,

```
mod1 <- lm(remote ~ gender)
```

stores the linear regression that predicts remote usage by gender (remote usage is the response) as an object called mod1. Note the syntax used to specify regression models in R, "response ∼ independent". The same syntax is used in much more general regression models than lm. We can think of the English meaning of the '∼' operator as 'as a function of'. To summarize the results of this regression with an ANOVA table, we can use the anova() function:

```
anova(mod1) # ANOVA table of the previous model
```

Output:

```
Analysis of Variance Table
Response: remote
Df Sum Sq Mean Sq F value Pr(>F)
gender 1 14859.6 14859.6 980.91 < 2.2e-16 ***
Residuals 98 1484.6 15.1
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

To fit a regression model with multiple independent variables (but no interactions), we separate the variables on the right hand side of the '∼' operator, by a '+' sign. E.g:

```
mod3 <- lm(remote ~ weight + gender) # Model 'remote' as a linear
# function of 'weight' & 'gender'
```

If we want to include a weight-gender interaction, we can simply replace the '+' by a '*' as illustrated below. Also, notice how the summary() command can be used on a regression object to give us the table of coefficients.

```
mod4 <- lm(remote ~ weight * gender)
mod4
```

This outputs the following information:

```
Call:
lm(formula = remote ~ weight * gender)
Residuals:
Min 1Q Median 3Q Max
-5.85304 -2.77067 0.04217 2.54726 5.03316 Coefficients:
Estimate Std. Error t value Pr(>|t|)
summary(mod4)
(Intercept) 16.72442 6.32943 2.642 0.00962 **
weight
gender
weight:gender
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 3.169 on 96 degrees of freedom
Multiple R-Squared: 0.941, Adjusted R-squared: 0.9392
F-statistic: 510.4 on 3 and 96 DF, p-value: < 2.2e-16
```

One can test whether the extra parameters that are fitted here for mod4 (compared with mod3) are really necessary using the ANOVA command. The output indicates that this may be the case.

```
anova(mod3,mod4)
```

Output:

```
-0.08030 -22.96669
0.04477 -1.794 0.07601 . 8.18338 -2.807 0.00606 **
0.28988
0.05393 5.375 5.36e-07 ***
Analysis
Model 1:
Model 2:
  Res.Df
2 96 964.23 1 290.20 28.893 5.359e-07 ***
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Saving Results:

If you want '.txt' files, use write.table() command; want 'csv' files, use write.csv() command.

```
write.table(data, "newdata.txt", row.names=FALSE)
```

3.5 Packages in R:

For complicated tasks, sometimes the functions that are accessible from a default installation of R are not sufficient. Fortunately, many enthusiasts have contributed other packages that can be downloaded from the CRAN network simply by clicking on Packages in the menu bar and clicking Install packages from CRAN.

If you want to browse through some of the packages available and what they do, take a look at www.r-project.org and select 'Packages' in the left hand window. Once a package has been downloaded you'll need to attach it to the search path using the command library(package) to have it available to use.

4 Appendix

4.1 Common Type of Objects

The most common objects in R are vectors, arrays, lists, factors, data frames, and functions. We will proceed by listing commands for each object.

4.1.1 Vectors

Vectors can either have numeric, character or logical strings. They are assigned using the "concatenation" or "combine" function, c().

```
x <- c(1,2,3,4,5,6) # Assignment of numeric vectors.
y <- c(2,4,8,16,32,64)
3:10 # the colon operator produces a numeric vector of
# sequential integers.
x+y # Vector arithmetic is done element by element.
x*y
sqrt(x) # basic functions tend to work on each element of a vector
seq(1:4) * rep(3,5) # Recycles elements from the shorter vector if
# necessary, but returns a warning with the
# result.
# A character vector - elements must be enclosed inside quotes.
n <- c("red", "orange", "yellow", "green", "blue", "purple")
# Examples of logical vectors:
x <= 3 & x > 1 # & is the 'and' operator
x <= 2 | x > 5 # | is the 'or' operator
x==3
n=="red"
x[1] # Vector elements can be accessed by the element number.
x[1:2] # A vector of element numbers will return a vector
x[-1] # A negative element number will return all but the elements
# given.
x[x > 4] # A logical vector will return the elements
# that correspond to  true elements
y[(x < 3) | (x > 5)]
names(x) # A vector can have names.
names(x) <- n x["red"]
seq(from = 1, to = 10, by = .1) # Produces regular sequences.
rep(3, 6) # Produces repeated patterns.
rep(x, 3) length(x) # Returns the number of elements of the vector.
sum(x)/length(x) # The mean of the vector x.
mean(x) # Also the mean of the vector x.
a = seq(1,4) # Caution: a single equals sign, '=', is an assignment
b = seq(4,7)
a[b == 5] # Logical comparison.
a[b = 5] # Notice the difference.
x[x<-6] # Sets the value of x to 6.
x[x < -6] # Returns elements of x less than -6 (an empty
# vector in this case).
```

4.1.2 Matrices and Arrays

Matrices and Arrays have similar characteristics to vectors.

```
m <- matrix(y, 2, 3) # Assignment of a numeric matrix.
t(m) # Matrix transpose.
mm <- matrix(y, 3, 2, byrow = T) # New assignment.
dim(mm)
dim(mm) <- c(1,6)
dim(mm) <- c(3,2)
# An array is an extension of a matrix of length(dim)>=3:
a <- array(c(1:8, 11:18, 111:118), dim = c(2,4,3)) # Array
arr <- 1:8
dim(arr) <- c(2, 2, 2)
m[1,2] # Accessing matrices is similar to vectors.
m[1,] # You can specify all the row or columns by leaving it blank.
# rbind and cbind add rows and columns to matrices:
rbind(1:2, mm)
cbind(1:3, mm)
cbind(matrix(4:7, 2, 2), matrix(rep(6, 4), 2)) # Note: One dimension.
rbind(matrix(4:7, 2, 2), matrix(rep(6, 4), 2)) # See help on matrix.
# Some Arithmetic
m + x # Adding a vector to a matrix does element by element addition;
# the vector is matched to the matrix going down the columns
# first.
m + m # Matrices with the same dimensions can be added.
m + m == 2*m # Logical matrices can be generated.
#CAUTION: ONLY MATRICES OF THE SAME DIMENSIONS CAN BE ADDED.
m + mm
m*m # * also works element by element (not as you may expect).
m %*% m # %*% is the matrix multiplication operator.
c(3,1) %*% c(2,2) # Vectors are made into rows or columns as nec.
m <- matrix(c(1, 7, 2, 3), 2, 2)
solve(m) # "solve" function inverts square matrices.
m %*% solve(m) # Verification.
```

4.1.3 Lists

A list is a collection of items of different types.

```
l1 <- list(vec1 = x, vec2 = c(3, 49, 54),
           l.matrx=matrix(c(T, T, F, T),2,2),
            state = c("Maine", "New␣Hampshire",
                      "Massachusetts","Vermont"))
ll[1:3]
ll[1]
ll[[1]]
ll$vec1
ll$vec2[3]
unlist(ll) # Converts a list into a vector.
unlist(ll, use.names = F) # Converts into a vector without names.
```

4.1.4 Factors

A factor is a special kind of a vector that is really useful when a categorical variable is considered:

```
sex <- rep(c("Male","Female"), c(6,4)) # Produces a character vector.
sex.fac <- factor(sex) # Changes it to a factor object.
sex.fac # Quotes are not printed.
as.numeric(sex.fac) # The underlining vector.
levels(sex.fac) # The levels of the vector
sum(sex.fac == "Male") # Counts the number of "Male" values.
sex.fac1 <- factor(sex, levels=c("Male", "Female"))
# Changes the levels.
levels(sex.fac1) # The first level gets the value 1.
```

4.1.5 Data Frames

Data are stored in R as data frames.

```
dat.frm <- data.frame(age=c(12,32,43,55,43,55),
                      sex=c(rep("M",3),rep("F",3)))
dat.frm[,2] # Data frames can be addressed as either a matrix
dat.frm$sex # or a list.
# Using rbind and cbind for adding observations and columns
rbind(dat.frm, data.frame(age=43,sex="M")) # Names need to match up.
cbind(dat.frm, data.frame(Novalue=rep("NA",6)))
dat.frm$older <- dat.frm$age > 40 # or new columns can be assigned
# With the I() command vectors retain their original mode
# (i.e. character):
dat.frm1 = data.frame(age=c(12,32,43,55,43,55),
sex=I(c(rep("M",3),rep("F",3))))
dat.frm$sex #notice the difference
```

4.1.6 Functions

We have used functions before in assigning objects. More generally, functions have input arguments that can either be entered in order in which they occur or be specified by the argument's name. For example:

```
seq(1,10,2) # No names, the arguments must be in the right order.
seq(by = 2, from = 1, to = 10) # With names, the order doesn't matter.
seq(1, 10, length = 5) # Mixing order and names.
seq(from = 1, by = 3, length = 12)
# Arguments not given are set to default values.

# A number of other useful basic functions are:
table(x) # for tabulations
apply(x,y,..) # is used to apply a function to sections of an array.
tapply(x,y,...), lapply(x,..), & sapply(x,...)# apply a function to a list.
```

Even though R has many built-in functions (that with everyday use of the program you may become familiar with), it is extremely convenient to create your own functions especially for operations repeated frequently. This is done using the function() command:

```
fname <- function(AnyArguments){the body}
# Example 1:
Range = function(vec){ # body is enclosed in curly brackets
  minvec <- min(vec)
  maxvec <- max(vec)
  c(minvec,maxvec)
}
# Note that the last (unassigned) object is the output of fname

Range(1:20) # We call the function with its name and the
# necessary input arguments.

# Example 2:
# Create a function to compare the ranges of several variables
# simultaneously using a histogram.
stackhist <- function(X) {
  par(mfrow = c(ncol(X),1))
  for(i in 1:ncol(X)) {
    hist(X[,i], xlim = c(min(X), max(X)))
  }
}
stackhist(cbind(rexp(1000,1), rnorm(1000, mean = 1)))
```

```
# Example 3:
# Create a func. to calculate the probability from birthday problem
bday <- function(k){ # k is the variable
  top <- seq(365,length=k,by=-1)
  bottom <- rep(365,k)
  return(1-prod(top/bottom))
}

bday(40)
```

4.1.7 Graphical Parameters

The main function for diagnostic plots is the “plot()” function. In order to find out all the arguments that are involved with this function, look at the help menu. A number of other useful functions for simple plotting are the following:

```
plot(x,y,...)       # Creates a plot.
lines(x,y,...)      # Adds lines to an existing plot .
points(x,y,...)     # Adds points to a current plot.
abline(x,y,...)     # Draws a line to a current plot.
title(...)          # Adds a title.
axes(...)           # Adds axes.
legend(...)         # Adds a legend to a current plot.
barplot(x,...)      # Bar graphs.
qqplot(...)         # Compares the distribution of two samples.
qqnorm(...)         # Compares the distribution of a sample to a Normal.
pairs(x..)          # All pairwise scatter plots between multiple variables.
hist(x,..)          # Histogram -- frequency plot -- of a variable x.
par(mfrow=c(i,j))# To put multiple figures in one plot in an i*j array,
# filling rows first.
par(mfcol=c(i,j))# Or filling columns first.
```