**INTRODUCTION TO R**

In this text we use the R Statistical software. R includes numerous useful packages for the analysis of textual information. Excellent specific stand-alone packages for the analysis of textual information are available as well (see RapidMiner, KNIME, MALLET, VOYANT, AntConc, PhiloLogic and many others).

This note is meant as a brief introduction to R. The review will help readers follow the R code we have written to analyze textual information. This review summarizes common tools for entering numerical and textual information into the R workspace, and describes useful R packages and R commands that allow the reader to carry out the analysis of both numerical (that is, numbers) and textual (that is, words) information. While our coverage is certainly not comprehensive, it should help the reader to get started with R.

The website that accompanies this book includes detailed information on R and the programs that we have written for the analysis of our two corpora. The website includes valuable information that will be useful when you analyze your own texts. You can cut and paste snippets of the programs to adapt the programs for your own analysis.

Several excellent introductory books on how to work with R have been written. Among them, Adler J.: R in a Nutshell (O’Reilly Media Inc, 2012); Crawley, M. J.: The R Book (Wiley, 2007); Everitt, B. S. and Hothorn, T.: A Handbook of Statistical Analyses Using R (Chapman and Hall, 2006), and Venables, W. N. and Smith, D. M.: An Introduction to R (Network Theory, 2002). A detailed list of additional books is given in the reference section at the end of this introduction.

R is a free software which is available through the internet; it can be downloaded from <http://cran.us.r-project.org/>. R is a language and an environment for statistical computing and graphics. It can be used under a variety of platforms, including Windows, Unix, Linux and Python. As of October 2020, the latest Windows version is R-4.0.2.

Here we explain how to run R under Windows. We assume that you have downloaded the most recent version of R from one of the websites. Running R will open up an R window (RGui) and within it an R Console window with its prompt ">". R commands may be issued at this point and run by clicking the enter tab. We will omit the prompt in our subsequent discussion.

You can use **RStudio** as well. RStudio sessions allow you to see both your script and the results on your console when you run your syntax (by clicking run in script pane, or Ctrl+Enter). It makes it easy to set your working directory and access files on your computer, and you can view and interact with the objects when you run your syntax.

R has an extensive help facility. You can get information from the Help function on the top right of your R window. You can get information on specific R functions (such as the histogram function hist or the log transformation log) by typing the following instruction at the prompt in the R console

help(hist)

? hist

help(log)

R has an extensive set of program libraries, also referred to as packages. It is very likely that the most common packages (such as base, stats, graphics) are loaded automatically when you install the software. If they are not, or if you want to install other more advanced packages, go to the top menu of your R window, look for Packages > Install package(s) (you may want to go to a website that is close to you) and click on the packages that you want to install (for example tm, a key package for analyzing text data).

Then go to Packages > Load package, and load that package. Alternatively, you can use the command install.packages("tm"). While you need to install a package only once, you need to load the package for future uses through the command

library(tm)

**Basic R commands**

R is case-sensitive, so x and X refer to different variables. R operates on named data structures. Data can be entered at the terminal or can be read from an external file. Entering the elements of a vector x – consisting of the four numbers 2, 4, 5, and 7 – one uses the R command

x <- c(2,4,5,7) or x = c(2,4,5,7)

This is an assignment statement using the function c(). Notice that the assignment operator "<-" (which is the same as the "=" operator) consists of the two characters < ("less than") and - ("minus") and points to the object receiving the value of the expression. For simplicity we use "=" here. Here and in the following we use the font **Ariel** to indicate what is entered by the user.

Let us consider the thickness measurements on the tabs used to close 5-gallon paint cans

(*n* = 135, measured in microns). We enter the data into the object "thick".

thick=c(29,36,39,34,34,29,29,28,32,31,34,34,39,38,37,

35,37,33,38,41,30,29,31,38,29,34,31,37,39,36,

30,35,33,40,36,28,28,31,34,30,32,36,38,38,35,

35,30,37,35,31,35,30,35,38,35,38,34,35,35,31,

34,35,33,30,34,40,35,34,33,35,34,35,38,35,30,

35,30,35,29,37,40,31,38,35,31,35,36,30,33,32,

35,34,35,30,36,35,35,31,38,36,32,36,36,32,36,

36,37,32,34,34,29,34,33,37,35,36,36,35,37,37,

36,30,35,33,31,35,30,29,38,35,35,36,30,34,36)

print(thick) prints out the data

thick[12] is the 12th observation in the list.

thickordered=order(thick) orders the data set from smallest to the largest.

print(thickordered) displays the ordered observations

thickordered[12] is the observation with rank 12

Once we have entered the data, we can perform various operations on the data. We can obtain summary statistics, construct a histogram, dot plot, box plot, and so on.

summary(thick) calculates summary statistics (min, first quartile, median, mean, third quartile, max)

quantile(thick,probs=c(0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9))

calculates the percentiles of order 0.1, 0.2, …, 0.9

a1=mean(thick)

a2=median(thick)

a3=sd(thick)

a4=var(thick)

These commands calculate and store in the object to the left of the

equality, the mean, median, standard deviation, and variance of the 135 observations.

a4 prints out the variance

boxplot(thick) creates the box and whisker diagram

hist(thick) creates the histogram

for further options, look at the help command for hist

dotPlot(thick) creates the dotplot. Note that the package BHH2 is needed.

**Entering data and text into an R-session**

Let us consider the data weight (in 1000 pounds) and the fuel efficiency (in gallons per 100 miles) of *n* = 10 cars listed below (see Abraham and Ledolter (2006)). The third column represents the name of the car.

Weight GPM Car

3.4 5.5 AMC Concord

3.8 5.9 Chevy Caprice

4.1 6.5 Ford Country

2.2 3.3 Chevette

2.6 3.6 Toyota Corona

2.9 4.6 Ford Mustang

2.0 2.9 Mazda GLC

2.7 3.6 AMC Sprint

1.9 3.1 VW Rabbit

3.4 4.9 Buick Century

You can enter the data by reading in each variable individually:

ygpm=c(5.5,5.9,6.5,3.3,3.6,4.6,2.9,3.6,3.1,4.9) gallons per 100 miles

xweight=c(3.4,3.8,4.1,2.2,2.6,2.9,2.0,2.7,1.9,3.4) weight in 1000 pounds

ygpm prints out ygpm

xweight prints out xweight

You can also read the data from the external text file that is stored on our website/your computer. For example, one can use the read.csv command to read comma-separated values (CSV) Excel files into R. If you have stored the data file FuelEfficiency.csv on your computer in directory C:/Data/, use the command

data=read.csv("C:/Data/FuelEfficiency.csv")

You can also use the read.table command if your data are stored in an external file such as the file hooker.txt (a text file) shown below. Dr. Joseph Hooker collected a set of 31 measurements on the boiling temperature of water (in degrees Fahrenheit) and the atmospheric pressure (in inches of mercury) at various locations in the Himalaya Mountains.

Temp AP

210.8 29.211

210.2 28.559

208.4 27.972

The first line of the file hooker.txt specifies a name for each variable in the data frame. The subsequent lines include the values for each variable. To read an entire data frame, we use the command

hook = read.table("C:/Data/hooker.txt",header=T)

The commands

Temp = hook[,1]

AP=hook[,2]

define the first column of the matrix “hook” as Temp (the boiling temperature of water) and the second column as AP (the atmospheric pressure). The statement

LnAP = 100\*log(AP)

? log

results in a transformation of the variable AP; log(AP) is the natural log of AP.

We use this data set to explain R commands for fitting simple and multiple linear regression models, lm(). For instance, a simple linear regression of Temp (the response or y-variable) on LnAP (the explanatory or x-variable) can be fit by issuing the command

hookfit = lm(Temp~LnAP)

The output object from the lm() command, “hookfit”, is a fitted model object. Information about the fitted model can be extracted from this file. For example,

summary(hookfit)

prints a comprehensive summary of the results of the regression analysis including the estimated coefficients, their standard errors, t–values and p-values. The command

anova(hookfit)

supplies the analysis of variance (ANOVA) table. The command

plot(LnAP,Temp)

plots Temp (the y-coordinate) against LnAP (the x-coordinate). A graphics window opens automatically. The fitted line can be superimposed on the scatter plot by issuing the command

abline(hookfit)

The command

qqnorm(hookfit$residuals)

leads to a normal probability plot of the residuals where “residuals” is in the fitted model object “hookfit”.

Multiple linear regression models can be fit quite easily with R. Suppose we have data in the vectors y, x1, x2 and x3. We can fit a multiple linear regression of y on x1, x2, and x3 by using the command

mregfit=lm(y~x1+x2+x3)

Information about the model is contained in the fitted model object “mregfit”, and can be displayed through the commands

mregfit

summary(mregfit)

One can restrict the intercept to be zero through

mulregfit=lm(y~x1+x2+x3-1)

The above commands can be fine-tuned according to specific requirements. Many other commands are available to perform various statistical analyses and plots (such as residual analysis, leverages, Cook’s D, various residual plots).

**Useful R commands for textual data analytics**

Commands specific to the analysis of text are shown next. Additional explanation is given in Chapter 1. We suggest that you make use of the excellent on-line help functions on packages and commands to learn more about the details of the commands.

R refers to text data as **string** objects. A string can be created by using either simple or double quotes. Create a string as follows:

mytext = "the weather of today is nice" ## string object mytext is created with double quotes

mytext

mytext = 'the weather of today is nice' ## string object mytext is created using simple quotes

mytext

Various commands help us process and analyze the text data, such as counting the number of characters, and transferring text to all lower (upper) case.

mytext = "the weather of TODAY is nice"

nchar(mytext) ## count the number of characters

toupper(mytext) ## change all the characters of the string to upper case

tolower(mytext) ## change all the characters of the string to lower case

We can also create a vector of strings (that is, a vector of several elements where each element contains a string) by using c() and separating strings by comma.

myvector = c("the weather of today is nice", "Today is a good day")

myvector

## creates the vector myvector with two elements; each element contains a string

nchar(myvector)

## returns the number of characters of each string in the vector

length(myvector) ## the number of elements (strings) in the vector

We can combine multiple strings into one by using the command paste. The strings are separated by a space by default. If want to separate strings by a specific character, use the sep argument in the command.

mytext = paste("the weather", "of today", "is nice")

## paste the three strings into one string and separate strings by a space

mytext

length(mytext)

mytext = paste("the weather", "of today", "is nice", sep = ",")

## paste the three strings into one and separate strings by “,”

mytext

length(mytext)

If the information is already in a vector of strings (such as the vector myvector below), use collapse rather sep:

myvector = c("the weather", "of today", "is nice")

## creates a vector containing three strings

mytext= paste(myvector, collapse=" ")

## the three strings in the vector are pasted into one and separated by a space

mytext

length(mytext)

We can extract a sub-string from a string using the command substring. By providing the beginning and ending character’s position in the string, we can extract the substring between these two positions.

mytext = "the weather of today is nice"

substr = substring(mytext, 3, 8)

## extract a sub-string from the 3rd to 8th character

substr

In a vector of strings, we may want to locate strings that contain a certain pattern. The command grep helps us to do this. It returns the indices of the elements that match the pattern. The match is case sensitive. The first argument of the grep function is the pattern that will be matched, the second argument is the string vector that is being searched.

myvector= c("The WEATHER", "the weather", "THE WEather")

grep("EA", myvector)

## find the strings that contain “EA”

grep("EA", myvector, ignore.case=TRUE)

 ## since grep is case sensitive, use ignore.case = TRUE.

grep("EA", myvector, value=TRUE)

## use the argument value=TRUE to return the strings

When processing text data, the replacement of a regular pattern with a certain specified string is very useful. This can be achieved with the function gsub. The first argument in the function gsub specifies the pattern that is to be matched; the second argument contains the replacement string for the pattern to be matched. The argument ignore.case, with its default FALSE, specifies that the pattern matching is case sensitive; if it is TRUE, the matching ignores the case.

myvector= c("the weather of today is NICE", "it is nice")

myvecrepl=gsub("nice", "good", myvector)

## replace “nice” with “good”, the “NICE” is not replaced

myvecrepl

myvecrepl=gsub("nice", "good", ignore.case= TRUE, myvector)

## both “NICE” and “nice” are replaced

myvecrepl

Splitting strings into individual parts (words) is useful when processing text data. The command strsplit splits a string into substrings (this is done within each string if we process a vector of strings).

myvector= c("the weather of today is nice", "it is nice")

strsplit(myvector, " ")

## split the strings by a space " "

## strsplit returns a list. If we want a character vector, combine the list

unlist(strsplit(myvector, " "))

The above commands should get you started with the text processing. Each command has many more arguments that what we have mentioned here; they can be fine-tuned to achieve specific goals.

**Useful R commands for reading text data from files and writing text data to files**

Next, we discuss useful commands for importing into R information of external textual files stored in different file formats, such as .csv, .txt, and .xml files.

**CSV (comma-separated values) file format** uses the comma character to separate (or delimit) the data. The R command read.csv helps us read csv-formatted data into an R table or data frame. One of its arguments, stringsAsFactors, is a logical argument with default TRUE. If the data should not be converted to factors, the argument should be set to FALSE. Another of its arguments is the header argument. If the first row of the input file is the header of a data frame, then header is set to TRUE, otherwise FALSE. Let us read the external csv file “test.csv” (which is used in Chapter 1) into a data frame. The CSV file has no header line, and we did not want to convert characters to factors.

data = read.csv("C:\\Users\\ledolter\\Desktop\\test.csv", header = FALSE, stringsAsFactors = FALSE)

dim(data)

data[,1]

str(data) ## shows the class of each column

dim(data) ## shows the dimension of the data frame

nrow(data) ## number of rows

ncol(data) ## number of columns

summary(data) ## shows the statistical summary

**TXT file format** is another frequently-used format for storing textual data. We use the R command readLines to convert the contents of a TXT file with *m* rows into an *m*-dimensional R vector of strings. As illustration we use the file combine39.txt, a file of more than 100,000 rows with each row containing a speech in front of the 39th U.S. Congress. Each string in the vector data represents one row (speech) of the TXT file.

data = readLines("C:\\Users\\ledolter\\Desktop\\combine39.txt" )

class(data) ## examine the class of mytext

length(data) ## the number of elements in the vector

data[11:20]

We can also write a string or a vector of strings to a TXT file, using the R command writeLines.

writeLines("The weather of today is nice", "C:\\Users\\ledolter\\Desktop\\mytext.txt")

mytext = "The weather of today is nice"

writeLines(mytext, "C:\\Users\\ledolter\\Desktop\\mytext.txt")

 ## two ways of writing the string vector “mytext” into a file

 ## note that the initially-created file is overwritten

**XML (eXtensible Markup Language) file format** is another useful file format for storing textual information. Information in an XML file is characterized according to a tree structure that contains at least one simple root element. The root element may have several child elements. Each element is tagged with angle brackets < >. The bracket <> represents the beginning of a root, while </> represents the ending of a root. For illustration we use the file 23.xml. The file contains the letters of volumes 2 and 3 (Northwest Territories) of the Territorial Papers.

To scrap the data from the XML file, we first need to parse the file by using the command xmlParse (a command from the XML library). You need to load the package first, using the command library(XML). Next, we use the command xpathSApply to select the node from the tree structure by providing the path of the node. If double forward slashes "//" are used for a node, such as "//year", then all nodes called "year" in the file are extracted. The argument xmlValue specifies that the text of the node is to be extracted. The logical argument recursive is used in connection with xmlValue. If it is FALSE, only the node “year” is processed. If it is TRUE, all sub-nodes are processed.

library(XML)

root=xmlParse("C:\\Johannes Ledolter\\2019SageBook\\TerrPapers\\LatestData\\23.xml")

year=xpathSApply(root,"//year",xmlValue,recursive=FALSE) ## year letter written

aut=xpathSApply(root,"//from",xmlValue,recursive=FALSE) ## author of letter

rec=xpathSApply(root,"//to",xmlValue,recursive=FALSE) ## recipient of letter

header=xpathSApply(root,"//header",xmlValue,recursive=FALSE) ## header of letter

text=xpathSApply(root,"//record/body",xmlValue,recursive=FALSE) ## text of letter

aut

rec

year

text

**Data Sets**

Data/text files used in this book can be downloaded from our book’s website. The files are stored as comma-separated values (CSV) Excel files, plain text TXT) files, and XML files. Download files and store them on your own computer.

Files used in this brief R tutorialare:

FuelEfficiency.csv

hooker.txt

test.csv

combine39.txt

23.xml

**R packages used**

In this text we use numerous R packages which have been written and extensively tested by researchers in this field. These packages must be installed and loaded before they can be used. Below is an incomplete list of the most important packages:

tm, topicmodels, stm, slam, mallet, text2vec, skmeans, wordcloud, poweRlaw, lattice, ggplot2, pdftools, qdpa, tidytext, quanteda, and many others.

**Reference Materials for R**

There are many helpful books on how to use R. References that we have found useful are listed below. You can also use the help function in R to learn about packages and commands.

Adler, J.: R In a Nutshell: A Desktop Quick Reference. O’Reilly Media, 2010.

Albert, J. and Rizzo, M.: R by Example (Use R!). New York: Springer, 2012.

Crawley, M.J.: The R Book. New York: Wiley, 2007.

Kabacoff, R.I.: R In Action: Data Analysis and Graphics with R. Greenwich, CT:

Manning Publications, 2011.

Ledolter, J.: Data Mining and Business Analytics with R. New York: Wiley, 2013.

Maindonald, J.H.: Using R for Data Analysis and Graphics: Introduction, Code and

Commentary, 2008. <http://cran.r-project.org/doc/contrib/usingR.pdf> (free resource)

Matloff, N.: The Art of R Programming: A Tour of Statistical Software Design. No

Starch Press, 2011.

Murrell, P.: R Graphics. Chapman & Hall, 2005.

<http://www.stat.auckland.ac.nz/~paul/RGraphics/rgraphics.html> (free resource)

Spector, P.: Data Manipulation with R (Use R!). New York: Springer, 2008.

Teetor, P.: R Cookbook. O’Reilly Media, 2011.

Torgo, L.: Data Mining with R: Learning with Case Studies. Chapman & Hall, 2010.

Venables, W.N., Smith, D.M., and the R Core Team: An Introduction to R, 2012.

<http://cran.r-project.org/doc/manuals/R-intro.pdf> (free resource)

**Files relevant for this tutorial**

* Introduction to R.docx: Word file containing the appendix
* Introduction to R.Rmd: RMD file when running R-Studio

An RMD file is an R Markdown file created using RStudio, an open source Integrated Development Environment (IDE) for the R programming language. It contains YAML (Ain't Markup Language) metadata, markdown-formatted plain text, and chunks of R code that, when rendered using RStudio, combine to form a sophisticated data analysis document

* + Introduction to R.html: HTML (Hypertext Markup Language) file

containing R code and R output

* FuelEfficiency.csv: Fuel efficiency data file
* hooker.txt: data file
* test.csv: text data for the illustration
* combine39.txt: Original text/data source of the speeches of the 39th Congress
* 23.xml: XML file for North West Territories of the Territorial Papers