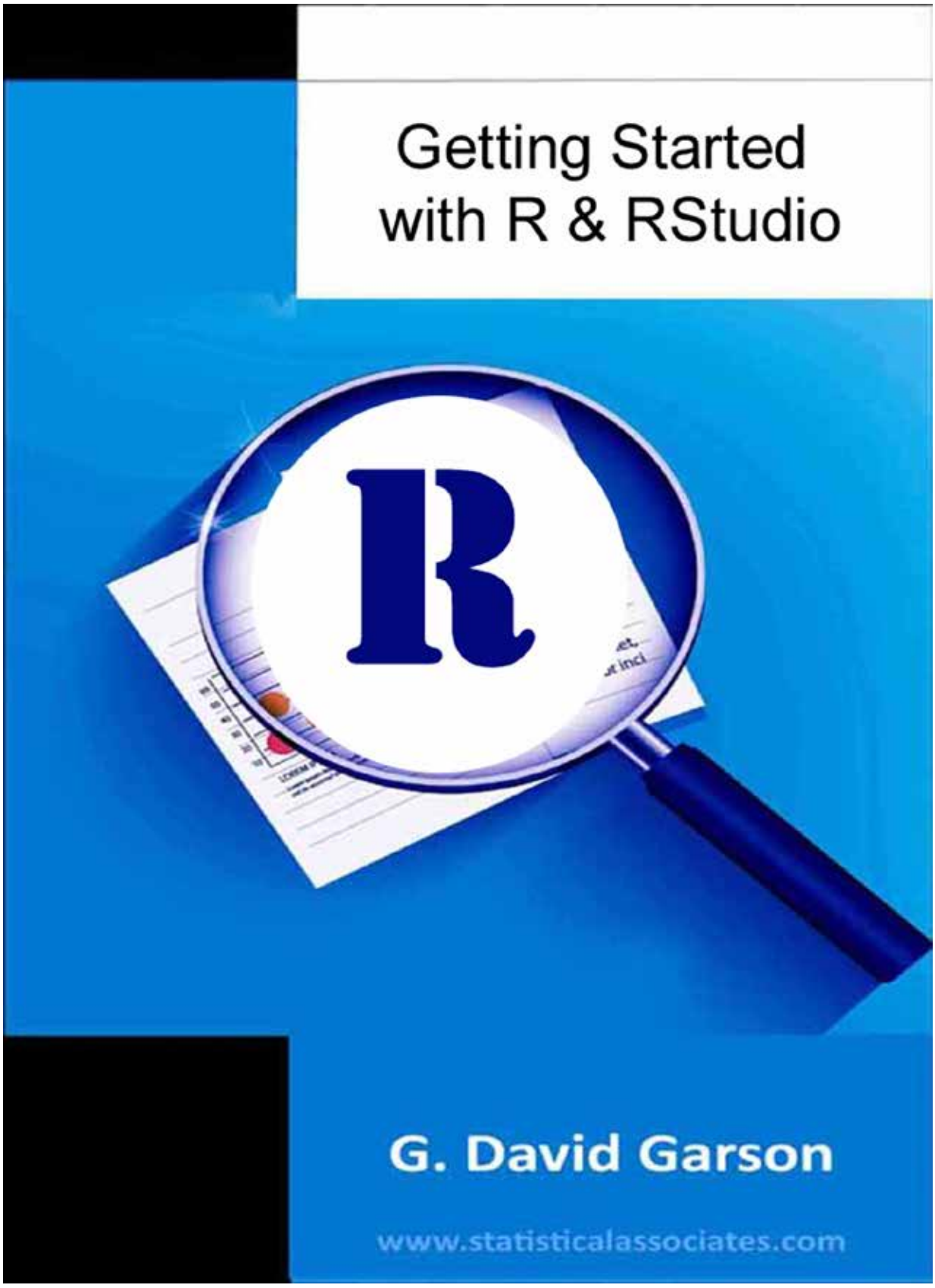


Copyright @c 2015 by G. David Garson and Statistical Associates Publishing

This work is copyrighted and no one has been given permission to display it online. If you are viewing this on the Internet, you are viewing an illegal copy. Please report the offending url to sa.publishers@gmail.com and to the host of the url.

@c 2018 by G. David Garson and Statistical Associates Publishing. All rights reserved worldwide in all media. No permission is granted to any user to copy or post this work in any format or any media.

Version 1/25/2018

The author and publisher of this eBook and accompanying materials make no representation or warranties with respect to the accuracy, applicability, fitness, or completeness of the contents of this eBook or accompanying materials. The author and publisher disclaim any warranties (express or implied), merchantability, or fitness for any particular purpose. The author and publisher shall in no event be held liable to any party for any direct, indirect, punitive, special, incidental or other consequential damages arising directly or indirectly from any use of this material, which is provided "as is", and without warranties. Further, the author and publisher do not warrant the performance, effectiveness or applicability of any sites listed or linked to in this eBook or accompanying materials. All links are for information purposes only and are not warranted for content, accuracy or any other implied or explicit purpose. This eBook and accompanying materials is © copyrighted by G. David Garson and Statistical Associates Publishing. No part of this may be copied, or changed in any format, sold, or used in any way under any circumstances other than reading by the downloading individual.

Contact:

G. David Garson, President
Statistical Publishing Associates
274 Glenn Drive
Asheboro, NC 27205 USA

Email: sa.publishers@gmail.com
Web: www.statisticalassociates.com

Table of Contents

Copyright @c 2015 by G. David Garson and Statistical Associates Publishing

Getting started with R and R Studio.

DATA

In this appendix we use a small dataset called mydata.sav. We import this SPSS-format data into R using code described further below. A link to mydata.sav is found on the Companion Website for this book.

INSTALLATION

To install R for the first time, go to any of the CRAN (Comprehensive R Archive Network) websites at https://cran.r-project.org/mirrors.html. Installation is free. To uninstall, update, or for other frequently asked questions, go to https://cran.cnr.berkeley.edu/ and click on FAQs. There are links on the same page for the R Homepage, The R Journal, R Sources, R Binaries, R Packages, and R Manuals.

1. Select a site in your country and click on it.
2. On the next page, select your operating system (Windows, Mac, Linux) and click appropriately (e.g., "Download for Windows").
3. On the subsequent page, click on the link marked "base".
4. On the next page, click "Download R 3.4.1 for Windows (75 megabytes, 32/64 bit)" or similar. This page also has links to download instructions and new features. Note that R changes versions frequently and keeps increasing in the size of the base package.
5. The file "R-3.4.1-win.exe" is downloaded to your default download directory. When requested, click "Save". When you open the file, an installation wizard opens. Accept the defaults but on the screen which offers 32-bit or 64-bit files, select according to your machine's processor.
6. After you clicking "Finish" in the installation wizard, there will be a desktop icon titled "R x64 3.4.1" or similar. By default, it will be located in C:\Program Files\R\R-3.4.1. If you installed prior versions, they will have folders in the same R directory, unless you

Copyright @c 2015 by G. David Garson and Statistical Associates Publishing

have deleted earlier versions (recommended). Files with the extension .RData will be associated with this version of R.

While you could now click on the R desktop icon, causing the "R Console" to open, ready to accept your menu choices or commands, most R users also install an integrated development environment (IDE) with a graphical user interface (GUI) for R. The leading one is "R Studio", also free to install. Among alternatives are "R Companion" or "RCommander". Here, we use R Studio, the functionality of which is summarized at https://www.rstudio.com/products/rstudio/features/. To install R Studio, follow the steps below.

1. To install R Studio, go to https://www.rstudio.com/products/rstudio/download/.
2. Click the "Download" button for the "Free" version of "RStudio Desktop".
3. Then click the installer link appropriate to your computer (e.g., "RStudio 1.0.153 - Windows Vista/7/8/10 "). Choose "Save File" from the ensuing popup window. Open the downloaded file.
4. A setup wizard starts. Accept the defaults and click the "Finish" button at the end.
5. R Studio will now be in the Windows start menu. You can right-click its icon and choose to pin it to the taskbar if you wish.

THE R CONSOLE

The R Console, which appears as soon as R is launched, is shown below in Figure Appendix 2.1, with the Help menu selection open to its options. In actual practice, the researcher is apt to use RStudio, which incorporates the R Console. RStudio is described in the next section. For the R Console, the R prompt is ">", indicating R is ready to accept a command. After the command is executed, another ">" prompt is supplied. If instead a plus sign ("+") is returned, this indicates the prior command was incomplete (e.g., often a trailing parenthesis has been omitted).

[Figure Appendix2.1 here; caption: 'The R Console']

The menu choices are:

File: Load and save workspaces, display files, change directories, load and save histories, create or open scripts, save to file, print, exit

Copyright @c 2015 by G. David Garson and Statistical Associates Publishing

Edit: Launch data editor, clear console, configure the graphical user interface (GUI), select all, copy, paste
View: Toggle toolbar or status bar on or off
Misc: Stop computations, list or remove objects, toggle word or filename completion on and off, toggle buffered output on and off
Packages: Install, load, or update packages; set CRAN mirror, select repositories
Windows: Cascade or tile windows, arrange icons
Help: FAQs, manuals, and other helps as shown in Figure Appendix2.1
Below the menu choices in the R Console is the toolbar (unless toggled off in the View menu) as shown in Figure Appendix 2.2.

[Figure Appendix2.2 here; caption: 'R Console Icons']

For convenience here, the toolbar is shown above in enlarged and numbered format.

1. Open script

2. Load workspace

3. Save workspace

4. Copy

5. Paste

6. Copy and paste

7. Stop current computation

8. Print

THE RSTUDIO INTERFACE

Many of the foregoing operations and, indeed, almost all operations are easier if an integrated development environment such as RStudio is used. This appendix is not meant to serve as a manual for either R or RStudio, but some of the functionality of the latter can be illustrated in this section. As an example, we will use RStudio to import a SPSS data file, then we will run Pearsonian correlations on its variables. We continue to use the small dataset mydata (specifically, the SPSS version, mydata.sav) mentioned in the "Data" section above.

After starting RStudio, the basic user interface appears as shown in Figure Appendix2.3 below.

It is a good idea to do two things right at the start and one thing at the end:

Copyright @c 2015 by G. David Garson and Statistical Associates Publishing

1. Select Session > New Session from the RStudio menu. This creates a new instance of RStudio with a cleared environment.
2. Choose Session > Set Working Directory. Many researchers make a practice of creating a separate file folder for each R project. Set the working directory to point to your project folder.
3. At the end, select Session > Quit Session. Respond "Yes" when asked if you want to save your workspace image in a .RDATA file in your working directory.

We next load in the example SPSS format mydata.sav dataset by selecting "From SPSS" from the "Import Dataset" drop-down menu.

[Figure Appendix2.3 here' caption: Caption: 'The RStudio User Interface']

After browsing to and selecting mydata.sav, the "Import Statistical Data" window appears, as shown in Figure Appendix2.4. Note that the "Code Preview" area in the lower right helpfully shows the R syntax needed to accomplish the current task, here importing the SPSS file. Click the "Import" button in the lower right to execute data importation. For more on data importation in RStudio, go to https://support.rstudio.com/hc/en-us/articles/218611977-Importing-Data-with-RStudio.

[Figure Appendix2.4 here' caption: Caption: 'The Import Window in RStudio']

After data are imported, RStudio appears as shown in Figure Appendix2.5 below.

[Figure Appendix2.5 here' caption: Caption: 'RStudio with Imported Data']

Steps for running an actual statistics command, which will be the `cor` command for correlation in this example, are given below. As `cor` is in the stats package, which is loaded by default, we may use it directly without having to use the `library` command. Variants of the command are shown below, including both Pearson and Kendall correlation for continuous and rank data respectively.

```
# Assume the environment has been cleared
# Read in mydata from a text file, as explained previously:
mydata <- read.csv("c://Data/mydata.csv", header=TRUE, sep = ",")

# You must use full variable names to do cor (unless you attach mydata).
# The cor command defaults to output of the usual Pearsonian correlation:
cor(happy,gender)
Error in is.data.frame(y) : object 'gender' not found
cor(mydata$happy,mydata$gender)
```

Copyright @c 2015 by G. David Garson and Statistical Associates Publishing

```
[1] -0.0727393

# However, mydata may be used with the summary command
# without attaching mydata:
summary(mydata)
     happy          gender
 Min.   :1.00   Min.   :0.0
 1st Qu.:2.25   1st Qu.:0.0
 Median :3.00   Median :0.5
 Mean   :3.10   Mean   :0.5
 3rd Qu.:4.00   3rd Qu.:1.0
 Max.   :5.00   Max.   :1.0

# Likewise, simple variable names may be used to get the correlation matrix,
# without having attached mydata:
cor(mydata)
              happy     gender
 happy   1.0000000 -0.0727393
 gender -0.0727393  1.0000000

# Other types of correlation are available besides Pearsonian:
cor(mydata,method="kendall")
              happy      gender
happy   1.00000000 -0.06405126
gender -0.06405126  1.00000000

# View other options for the correlation command:
help(cor)
```

As in the R Console, if a package has not been loaded then it is necessary to load it in RStudio. We illustrate for the package `rel`, which includes the command `kra`, implementing Krippendorff's alpha, a reliability measure preferred for ordinal data. A simple Internet search for "Rstudio krippendorff's alpha" immediately turns up the existence of the `rel` package. In the lower right-hand side of the RStudio interface, shown above, click on the "Packages" tab to list available packages. There will be two such lists: "User Library" and "System Library". Inspection of both lists shows the `rel` package is not installed on either one. Later, after installation, `rel` will be listed in the "User Library". Click on the "Install" button for the "Packages" tab and the "Install Packages" window will appear, as shown below in Figure Appendix2.6

[Figure Appendix2.6 here; caption: '"Install Packages Window in RStudio"]

The R syntax for actual execution of the `rel` package's `kra` command is shown below. Note that even though the package is installed in the User Library, it must be invoked with the `library(rel)` command before issuing the `kra` command. The `library(rel)` command will be issued automatically by RStudio is one checks the package's checkbox in the User Library, or `library(rel)` may be added explicitly as in the script below.

```
# If not installed, install the rel package:
install.packages("rel")
# Invoke the rel package:
library(rel)

# Load sample data: 4 raters, 12 subjects, and 5 ordinal scale response categories.
myratings <- cbind(rbind(1,2,3,3,2,1,4,1,2,NA,NA,NA),
rbind(1,2,3,3,2,2,4,1,2,5,NA,3),
rbind(NA,3,3,3,2,3,4,2,2,5,1,NA),
rbind(1,2,3,3,2,4,4,1,2,5,1,NA))

#Krippendorff's alpha for ordinal scale data
kra(data = myratings, metric = "ordinal", conf.level = 0.95, R =
0)
```

Output:

```
Call:
kra(data = myratings, metric = "ordinal", conf.level = 0.95,
    R = 0)

      Estimate LowerCB UpperCB
Const  0.81539      NA      NA

Confidence level = 95%
Observations = 4
Sample size = 12
```

FORMAT FOR R CODE SEGMENTS

In the segments of R code above and below, note that the hashtag ("#") starts a comment for the command. Comments are ignored when R executes. Output lines start with a right arrow symbol (>) , which is the prompt in the R console. Commands are in Courier New font but are not preceded by a symbol. It is possible to enter multiple command and comment lines as a block by the cut-and-paste method.

ENTERING OR IMPORTING DATA

As in other statistical packages, data for R may be loaded in a variety of ways, only a few of which are described below. Note lines starting with hashtags (#) are comment lines, ignored by R. The commands below are executed directly from the R console (not R Studio). In R Studio, loading data is made easier because one can select File > Import Dataset > From SPSS (or From SAS, From Stata, From Excel, or From CSV). Note that loading one data file does not erase others from memory.

Data format requirements

For manually created data such as text files and spreadsheet data the researcher will export, keep in mind these requirements:

- One variable per column.
- All columns have the same number of rows.
- No interior spaces in variable names if names are in row 1.
- Names are case-sensitive.
- Do not use reserved words in names (e.g., 'numeric','factor','vector', 'summary', and others. Use of such terms may lead to unpredictable results.
- For missing data values, enter "NA" without the quote marks. If the original data contain some other missing code, such as -99, cases can be recoded:

```
# recode -99 to missing for variable happy
# select rows where happy is -99 and recode happy
mydata$happy[mydata$happy==-99] <- NA
```

- For continuous data entered as vectors, enter an integer (e.g., 76) or a decimal value (e.g., 76.832).
- For categorical data entered as factors, enter integer values (e.g., 1, 2, 3; use of 0 is permissible but not recommended) or text values (e.g., female, male) or logical values (e.g., T, F)

Direct manual entry

Define and assign values to gender and happy as variables (objects)

Assign values for 10 cases to happy

The c() function below creates a vector of values

Use <- for assignment of values; "=" does not always work

```
happy <- c(4,1,3,5,3,3,2,4,5,1)

#Assign values for 10 cases to gender, repeating values five times each
# Gender is coded 0=male, 1=female but there are no value labels at this point
gender <- c(rep(0, 5), rep(1, 5))

# Create the data frame 'mydata' containing happy and gender
# You could call it mydata.df  but then you will always have to use the ".df" extension
# but some like the ".df" extension as a reminder the object is a data frame
mydata <- data.frame(happy, gender)
# Display a table of the mydata data frame
# Initially, the table has no value labels
table(mydata)

# The factor() function is used to assign factor labels, here for the gender variable
mydata$gender <- factor(mydata$gender,levels = c(0,1),
    labels = c("male", "female"))

# The ordered() function can assign ordinal factor labels, here for the happy variable
mydata$happy <- ordered(mydata$happy,
   levels = c(1, 2, 3, 4, 5),
   labels = c("very unhappy", "unhappy",
      "neither happy nor unhappy", "happy", "very happy"))

# Create a table of happy by gender
# There are now labels for gender and happy
table(mydata$happy,mydata$gender)

                          female male
  very unhappy                 0    0
  unhappy                      0    0
  neither happy nor unhappy    0    0
  happy                        0    0
  very happy                   0    0

# We can get a more complete table by using the describe() function from the "psych"
# package, then using the kable() function from the "knitr" package.

install.packages(psych)
library(psych)
install.packages(knitr)
library(knitr)
#Get information on kable, a table generation function
```

help(kable)
# Example:
```
out.table <- mydata
kable(describe(out.table, IQR = TRUE), digits = 3)
```

#Optionally, clear the workspace of a specific dataset (remove data):
```
rm(data=mydata)
```

#Optionally, clear the workspace of all datasets. The working directory is not cleared.
```
rm(list=ls())
```

Importing a text file

From Excel or some other program, save a comma separated values (.csv) text file. Let row 1 contain variable names (the header row, hence header=TRUE below). If tab-delimited, change the separator specification to "\t". Text files are the most universal file format and are sometimes preferred for this reason.
Set the working directory
```
setwd("c:/Data")
```
Read in the text file called mydata.csv into a data object called "mydata" from
the c:/Data directory or substitute your own file location
```
mydata <- read.csv("c://Data/mydata.csv", header=TRUE, sep =
","")
```
The attach() function below optionally links mydata to the R search path.
After attaching, mydata objects can be accessed by simply giving their names (e.g., happy)
Otherwise full variable name must be used (e.g., mydata$happy)
```
attach(mydata)
```
Optionally, list the file
```
list(mydata)
```
Optionally, get mean, median, and quartile values
```
summary(mydata)
```

It is possible to use a Windows-style file selection window to browse to the text file that is wanted (e.g., .txt, .csv). The commands below will enter the selected file, mydata.csv, into the object "mydata".
```
mydata <- read.table("mydata.csv", header = TRUE, sep = ",")
```

or

```
mydata <- read.csv("mydata.csv", header=TRUE, sep = ",")
```

Type `help("read.table")` to view more options for this command, which is not restricted to comma-separated values format.

Copyright @c 2015 by G. David Garson and Statistical Associates Publishing

Importing a Stata .dta *file

Importing Stata files by the method below requires the 'haven' package be installed
If it isn't installed, see the following section on installing packages from the R console
If using R Studio, this will be taken care of for you after selecting from the menu
File > Import Dataset > From Stata
Note: The alternative library 'foreign' only loads Stata 12 and earlier .dta files

```
library(haven)
mydata <- read_dta("C:/Data/mydata.dta")
# The View() function below must be capitalized
# View() displays a spreadsheet-like listing of the data
View(mydata)

# Assigning labels, list(), summary(), and table()
# can be implemented as in the section above on csv files
```

Importing a SPSS .sav file

Importing SPSS files by the method below requires the 'foreign' package be installed

```
library(foreign)
# Import SPSS sav file with value labels
library(foreign)
mydata <-read.spss("C:/Data/mydata.sav", use.value.labels=TRUE,
to.data.frame=TRUE,max.value.labels=Inf,trim.factor.names=FALSE)
# Now view and table the data, which will have value labels
View(mydata2)
table(mydata2$happy,mydata2$gender)
```

Importing a SAS .sas7bdat file

Importing SAS files by the method below requires the 'haven' package be installed. There is an alternative method using the 'sas7bdat' package, but it is much slower. Also, one may save to another format and then import into R (for csv files, use the read.csv function; for Excel files, use the read.xls function in the xlsReadWrite package; or in SAS, save a SAS xport file (.xpt), then use the 'Hmisc' package (not done here).

```
library(Hmisc)
mydata <- sasxport.get("c:/Data/mydata.xpt")
# character variables are converted to R factors
```

Here we use the haven package:

```
# The haven method
library(haven)
mydata<- read_sas("c:/Data/mydata.sas7bdat")
View(mydata)

# Assigning labels, list(), summary(), and table()
# can be implemented as in the section above on csv files
```

INSPECTING YOUR DATA

As shown in output below for the object "mydata", the `ls()` function shows objects in the current environment. The `View()` function, which must be capitalized, puts the table in the RStudio View window. The `mode()` function shows the data type of the object. The `names()` function lists the variables. The `str()` function gives basic information about the data structure. The `structure()` function, not shown, gives even more information. These and other data inspection functions are shown below.

```
ls()                    # Lists objects in the environment, like mydata
                            [1] "gender"    "happy"     "mydata"
View(mydata)            # Shows mydata as a table; capitalize "View"
dim(mydata)             # Shows dimension of mydata, here 10 rows,2 cols:
                        [1] 10  2
mode(mydata)            # Shows the data type of mydata
                        [1] "list"
class(mydata)           # Shows data class (data frame, matrix, numeric, etc.)
                        [1] "tbl_df"  "tbl"  "data.frame"
names(mydata)           # Shows the variable names in mydata
                        [1] "happy"  "gender"
str(mydata)             # Shows structure of mydata in brief format
                        'data.frame':  10 obs. of  2 variables:
                        $ happy : int  4 1 3 5 3 3 2 4 5 1
                        $ gender: int  0 0 0 0 0 1 1 1 1 1
head(mydata, n=10)      # Print the first 10 rows of mydata
                                              happy gender
                        1                     happy   male
                        2               very unhappy   male
                        3  neither happy nor unhappy   male
                        ...
                        9                 very happy female
                        10              very unhappy female
tail(mydata, n=10)      # print last 10 rows of mydata
                        # Similar output not shown here
```

One may check for rows with missing data using this command:

```
mydata[!complete.cases(mydata),]
```

If missing data are found, one may create a new dataset in which rows with missing data have been dropped listwise:

```
mydata2 <- na.omit(mydata)
```

SAVING AND RETRIEVING DATA

Saving/retrieving in text format

The comma-separated values (.csv) text format is a format which is generally supported across platforms. Therefore some researchers prefer to save data in .csv format with the write.csv() function and retrieve it with the read.table() function, as shown below. The read.table() function supports even large tables.

```
# Set the working directory
setwd("c:/Data")
# Assuming mydata is in memory, write it to a .csv file
write.csv(mydata, file = "mydata2.csv", row.names = FALSE)
# For demonstration purposes, erase mydata from memory
rm(mydata)
# Retrieve mydata from the saved mydata2.csv file
mydata <- read.table("mydata2.csv", header = TRUE, sep = ",")
# Optionally , view mydata (or use View(mydata))
table(mydata)
```

Saving/retrieving in R format

As in other statistical packages, data in memory in R will not be saved to the C: drive or other location unless the researcher acts to do so. R's native file format extension is .rds, as in mydata.rds. There are a variety of ways to save and retrieve data in R. A common method is illustrated below.

```
# Set the working directory
setwd("c:/Data")
# Save an object in memory to a file
saveRDS(mydata, file = "mydata.rds")
# Optionally, remove mydata from memory
rm(mydata)
```

Copyright @c 2015 by G. David Garson and Statistical Associates Publishing

```
# Restore the mydata from disk into data frame mydata
mydata <- readRDS("mydata.rds")
# Optionally, view mydata
View(mydata)
```

PUTTING AN OBJECT INTO R'S SEARCH PATH

Note that an object in the View window is not necessarily in R's search path. The `attach()` function puts an object in the search path. When in the search path, variables like "happy" can be referenced with their simple name. If not in the path, full variable names must be used with the data frame as part of the reference, separated by a dollar sign (e.g., mydata$happy).

Warnings: The `attach()` method does not update changes made to variables in the data frame, requiring the researcher to detach and then re-attach. Also, the `attach()` method may find more than one object of the same name in more complex projects, requiring the more conservative practice of using full variable names. For these reasons, and for variable-naming clarity, the `attach()` method often is derogated in favor of use of full variable names (e.g., mydata$gender).

We illustrate below by examining the search path using the `search()` function. Upon finding mydata is not listed we add it with the `attach()` function; then we search again, showing mydata is now in the search path.

```
> search()
 [1] ".GlobalEnv"        "package:bitops"    "tools:rstudio"
 [4] "package:stats"     "package:graphics"  "package:grDevices"
 [7] "package:utils"     "package:datasets"  "package:methods"
[10] "Autoloads"         "package:base"
> attach(mydata)
> search()
 [1] ".GlobalEnv"        "mydata"            "package:bitops"
 [4] "tools:rstudio"     "package:stats"     "package:graphics"
 [7] "package:grDevices" "package:utils"     "package:datasets"
[10] "package:methods"   "Autoloads"         "package:base"
```

R DATA STRUCTURES: VECTORS, FACTORS, AND DATA FRAMES

R supports very complex data structures, which is one reason for its popularity. For instance, R is popular for handling "big data" involving captured social media. For purposes here, however, we discuss only the very basics.

- A "vector" is what social scientists usually think of as a numeric variable, corresponding to a column in a dataset, containing numeric data. In R parlance, this is an "atomic vector".
- A "factor" is what social scientists usually think of as a categorical variable, corresponding to a column in a dataset, containing alphanumeric data.
- A "data frame" is what social scientists usually think of as a dataset, containing vectors and/or factors, each with the same number of observations, thus forming a rectangular dataset.
- A "tibble" is a "new take" on data frames, providing slightly different functionality. For instance, tibbles do not convert character strings to factors. Tibbles are an advanced topic not discussed in this book but see https://cran.r-project.org/web/packages/tibble/vignettes/tibble.html
- R has other data structures, such as lists (like a data frame but different variables may have different numbers of observations and be of different data types), matrices (different variables but of the same type), and arrays (the most general), which also are not discussed here.
- Vectors, factors, and data frames are all "objects".

In the examples above on importing data into a data frame, the reader will have noticed that commands which reference a variable (vector or factor) must use the full variable name with the data frame name included, followed by a dollar sign: e.g., `mydata$gender.` After data have been brought into a data frame (here, mydata) , use of simple variable names will lead to an error message such as the following, if the data frame is not in R's search path:

```
# Assume the data frame mydata and the vectors happy and gender
# aren't in the R environment. Then load mydata from a text file
mydata <- read.csv("mydata.csv", header = TRUE, sep = ",")
```

The table() function gives an error message
```
table(happy,gender)
Error in table(happy, gender) : object 'happy' not found
```

However, table() works if full variable names are used:
```
table(mydata$happy,mydata$gender)
```

```
                            female male
  happy                          1    1
  neither happy nor unhappy      1    2
  unhappy                        1    0
  very happy                     1    1
  very unhappy                   1    1
```

However, it is also possible to separate a data frame into its vectors and factors, which is another way (apart from using full variable names like mydata$happy) that simple variable names may be used. The `as.vector()` and `factor()` functions are used for this purpose.

I am not a huge fan of this personally. It's difficult to "unfactor" something as it doesn't retain the original values. So when running 'as.vector' you end up creating a character rather than a numeric vector, which you cannot use in running the correlations below. I think it would be better to say that those original values should be held in value even after you factor them.

INSTALLING ADDITIONAL PACKAGES

Optionally, view packages installed by default on your machine.
```
getOption("defaultPackages")
[1] "datasets"  "utils"     "grDevices" "graphics"  "stats"
"methods"
```

Optionally, view all currently installed packages:
```
installed.packages()
```

output is too long to list here

Check to see if the desired package exists using the require() function. Below, the stats package is already installed but the lme4 package (for multilevel modeling) was not:
```
require("stats")
require("lme4")
> Loading required package: lme4
> Warning message:
> In library(package, lib.loc = lib.loc, character.only = TRUE,
> logical.return = TRUE,  :
```

Copyright @c 2015 by G. David Garson and Statistical Associates Publishing

```
> there is no package called ‘lme4’
```

Alternatively, to see if a specific package is installed, one may use the "Packages" tab in RStudio, as illustrated in figure Appendix 2.7. Simply look down the list of packages and see if the one your want (e.g., lme4) is there. If it is, as in this example, check its checkbox to issue the library command automatically. Double-click on it to view its documentation, which will include a list of functions it contains (e.g., the lme4 package supports the lmer function for multilevel modeling).

[Figure Appendix2.7 here; caption: ‘“Packages Tab in RStudio”]

The the wanted package is not found, it must be installed. To install the lme4 package:

```
install.packages("lme4")
> Installing package into ‘C:/Users/David/Documents/R/win->
library/3.4’
> (as ‘lib’ is unspecified)
> --- Please select a CRAN mirror for use in this session ---
> also installing the dependencies ‘minqa’, ‘nloptr’,
> ‘RcppEigen’

# At this point you are asked to pick a mirror site from a drop-down list. It will then load
# the requested package and all dependent packages.
> trying URL
'https://mirrors.nics.utk.edu/cran/bin/windows/contrib/3.4/minqa
_1.2.4.zip'
> Content type 'application/zip' length 667956 bytes (652 KB)
> downloaded 652 KB

> trying URL
'https://mirrors.nics.utk.edu/cran/bin/windows/contrib/3.4/nlopt
r_1.0.4.zip'
> Content type 'application/zip' length 1173353 bytes (1.1 MB)
> downloaded 1.1 MB

. . .

> trying URL
'https://mirrors.nics.utk.edu/cran/bin/windows/contrib/3.4/lme4_
1.1-13.zip'
> Content type 'application/zip' length 4733443 bytes (4.5 MB)
> downloaded 4.5 MB

> package ‘minqa’ successfully unpacked and MD5 sums checked
> package ‘nloptr’ successfully unpacked and MD5 sums checked
> package ‘RcppEigen’ successfully unpacked and MD5 sums checked
> package ‘lme4’ successfully unpacked and MD5 sums checked
```

Copyright @c 2015 by G. David Garson and Statistical Associates Publishing

```
# The lme4 package supports the multilevel analysis  function
lmer()
```

After installing a package, the next step is to load it with the `library()function:`

```
library(lme4)
```

Note that for multilevel modeling, an earlier package than `lme4` is available:

```
install.packages("nlme")
library(nlme)
# The nlme package supports the multilevel analysis function
lme()
```

Installing new packages is also discussed and illustrated in the previous section on "Using the RStudio Interface".

MORE USER BASICS

Using the R Console

From the R Console menu, select Help > Console to view a listing of keystrokes used for scrolling and editing (e.g., the Esc (Escape) key aborts the R interpreter and returns to the R prompt to ">"). Keep in mind that R is case-sensitive (e.g., Ctrl + L is not the same as Ctrl + l). Set the working directory where your data are located:

```
setwd("C:/Data")
```

Note that one must use a forward slash in the path name. You can also type `getwd()` to list your current working directory.

Saving and loading your workspace

Your workspace contains your current R working environment, including any objects you have created, such as vectors, data frames, and functions . At the end of any R session you will be prompted to save your workspace. If you respond affirmatively, the workspace as configured when you saved it will be reloaded automatically the next time you start R. The workspace is saved in the current working directly with the default name ".RData". You can also give it a full filename if you desire.

In the R code below, the same small data frame as illustrated earlier (under "Direct Manual Entry") is recreated and stored in the data frame mydata, and two of its variables are used to output a table. The workspace is then saved and R is exited. In the lower section of R code, after R is started up again, the workspace is loaded and it is demonstrated that the mydata data frame is still available and the same table may be created.

```
setwd("c:/Data")
```

```
happy = c(4,1,3,5,3,3,2,4,5,1)
gender = c(rep(0, 5), rep(1, 5))
mydata.df = data.frame(happy, gender)
table(happy,gender)
>     gender
> happy 0 1
>    1 1 1
>    2 0 1
>    3 2 1
>    4 1 1
>    5 1 1
# Below, save the workspace to "c:/Data/.RData"
save.image()
# Exit R
quit()
```

[The R Console is closed, then is re-started]

```
setwd("c:/Data")
load(".RData")
table(happy,gender)
>     gender
> happy 0 1
>    1 1 1
>    2 0 1
>    3 2 1
>    4 1 1
>    5 1 1
```

Saving and loading your command history

By default, R can save and load the history of your commands. Among other purposes, the resulting list of commands may be used to copy and paste commands for re-use. Below, the working directory is set, two commands used elsewhere in this appendix are given, the savehistory() command is given, and the R session is terminated. History will be saved to "c:/Data/.RHistory".

```
setwd("c:/Data")
require(utils)
data()
savehistory()
quit()
```

R is closed and then restarted. The `loadhistory()` command loads the saved history of commands. The `history()` command displays these commands.

```
setwd("c:/Data")
loadhistory()
history()

savehistory(file="myfile") # default is ".Rhistory"

It is also possible to save or load under specific filenames.
Files go to the current working directory.
savehistory(file="Appendix2.Rhistory")
loadhistory(file="Appendix2.Rhistory")
```

Clearing RStudio

As you work, your work environment may become cluttered and you may wish to start fresh. If you have just selected Session > New Session from the RStudio menu, this is not a problem as the work environment starts cleared, in a new instance of RStudio. However, re-running RStudio for the same session will not accomplish this as the saved environment is reinstated. Instead, consider using the following commands (but be careful!).

- Clear the console: Ctrl-L - This will clear the "Console" window in the lower left of RStudio, which is where commands are entered. However, command history is not cleared. The up/down arrows may still be used to scroll through past commands. This is equivalent to selecting Edit > Clear Console from the RStudio menu.
- Clear all datasets from the environment: `rm(list=ls())` – This empties all datasets from the "Global Environment"> Environment tab, in the upper right of RStudio.
- Clear history: Select the "History" tab and click the broom icon. This will clear the history of all commands. The up/down arrow keys will have nothing to scroll through until you enter more commands.
- Clear datasets shown in upper left "View" window: Click the "x" option for each dataset to close it. When RStudio is closed, answer "yes" when queried whether

Copyright @c 2015 by G. David Garson and Statistical Associates Publishing

you wish to save the current environment. When RStudio is run again, it will not list the datasets that were closed. Datasets are not removed from disk.

Getting help

From the R prompt, the following commands illustrate how to get help:
Get basic information about file manipulation functions in R:

```
help(files)
```

Get basic information about R functions for base stats:

```
help(stats)
Get a listing of commands available for the stats package:
library(help = "stats")
```

Get help for the cor (correlation), one of the commands in the stats library:

```
help(cor)
```

You can also get help for other package names. For instance, `help(graphics)` gives help on the graphics package and `help(data)` gives help on the data function of the utils package, which includes help on loading data, formats supported, etc. Note that after typing help, in the upper left corner of the help screen you are told the corresponding package:

graphics-package {graphics} - `help(graphics)` leads to the graphics package

data {utils} - `help(data)` leads to the data function of the utils package

To list all installed packages:

```
installed.packages()
```

A listing of thousands of R packages for statistical analysis is found at https://advanceddataanalytics.net/r-packages/.

Sometimes help() fails, in which case the double-question mark method may be used, causing the R console to search more broadly, as illustrated below.

```
help(lme4)
> No documentation for 'lme4' in specified packages and libraries:
> you could try '??lme4'
```

```
??lme4
> Search Results
```

[Search results appear web-style in the lower right window, with hypertext links.]

Vignettes:

lme4::lmer	Fitting Linear Mixed-Effects Models using lme4	PDF	source	R code

lme4::lme4	Linear, generalized linear, and nonlinear mixed models

Tutorial on R

While there are many tutorials available for learning R, here we mention one: the swirl package, which is an interactive platform for simultaneous learning about R, the R Console, and statistics. It contains a variety of lessons, some of which are video-based, some are self-review tests, and some are hands-on exercises in using the R console. User responses are tested for correctness and hints are given if appropriate. Progress is automatically saved so that a user may quit at any time and later resume, without losing work, by typing swirl().

```
install.packages("swirl")
> Installing package into 'C:/Users/David/Documents/R/win-
library/3.4'
> (as 'lib' is unspecified)
> --- Please select a CRAN mirror for use in this session ---
[You select mirror site from the drop-down list.]
> also installing the dependencies 'stringi', 'crayon',
'praise', 'jsonlite', 'mime',
> 'curl', 'openssl', 'bitops', 'stringr', 'testthat', 'httr',
'yaml', 'RCurl', 'digest'
```

[Swirl and its many supporting packages are installed.]

```
# Now invoke swirl
library(swirl)
> | Hi! Type swirl() when you are ready to begin.

swirl()
> Welcome to swirl! Please sign in. If you've been here before,
use the same name as you did then. If you are new, call yourself
something unique.
>  What shall I call you? >
```

[The swirl tutorial starts. Later, type swirl() to resume.]

Sample data and demos

R comes with sample data and demonstrations. To list all data sets in the package 'datasets':

```
require(utils)
data()
```

R also comes with built-in demonstrations. To bring up a listing of the many demos available in the base, graphics, grDevices, and stats packages, all of which aare part of default installation of R, type:

```
demo()
```

For a larger list of demos, type:

```
demo(package = .packages(all.available = TRUE))
```

To run an available demo in the listing, such as that for one of the linear and generalized linear modeling demos from `An Introduction to Statistical Modelling' by Annette Dobson (available in the stats package), type:

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

Naturally, only some statistical programs have demos.

Error Messages

Problem: "Unexpected input" error when using quote marks
R wants you to use straight quotes: "text".
Unfortunately, Word defaults to smart quotes: “text”.
To change this Word default, do this:

1. On the File tab, click Options.
2. Click Proofing, and then click AutoCorrect Options.
3. In the AutoCorrect dialog box, do the following: Click the AutoFormat As You Type tab, and under Replace as you type, select or clear the "Straight quotes" with “smart quotes” check box. ...
4. Click OK.

Other resources

R and RStudio tutorials and resources abound on the Internet and are found with any search engine. However, readers may find particularly useful the collection of “Cheat Sheets” for R and RStudio put out by RStudio itself. These pdf documents concisely summarize many widely used R packages and commands, with very brief illustrations of R code. The url is https://www.rstudio.com/resources/cheatsheets/ .

Copyright @c 2015 by G. David Garson and Statistical Associates Publishing

FIGURES

Copyright @c 2015 by G. David Garson and Statistical Associates Publishing

Figure Appendix2.1

Caption: “The R Console”

File: garson_Appendix2-1.tif

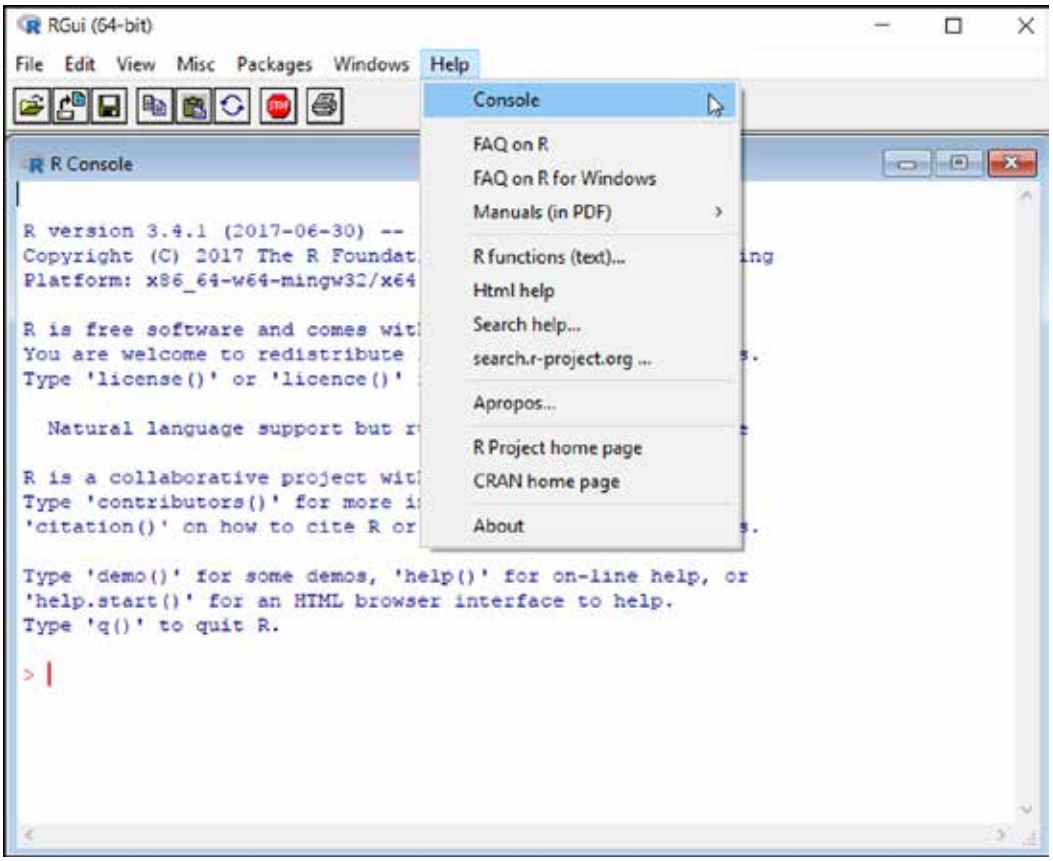


Figure Appendix2.2

Caption: “R Console Icons”

File: garson_Appendix2-2.tif

Figure Appendix2.3

Caption: “The RStudio User Interface”

File: garson_Appendix2-3.tif

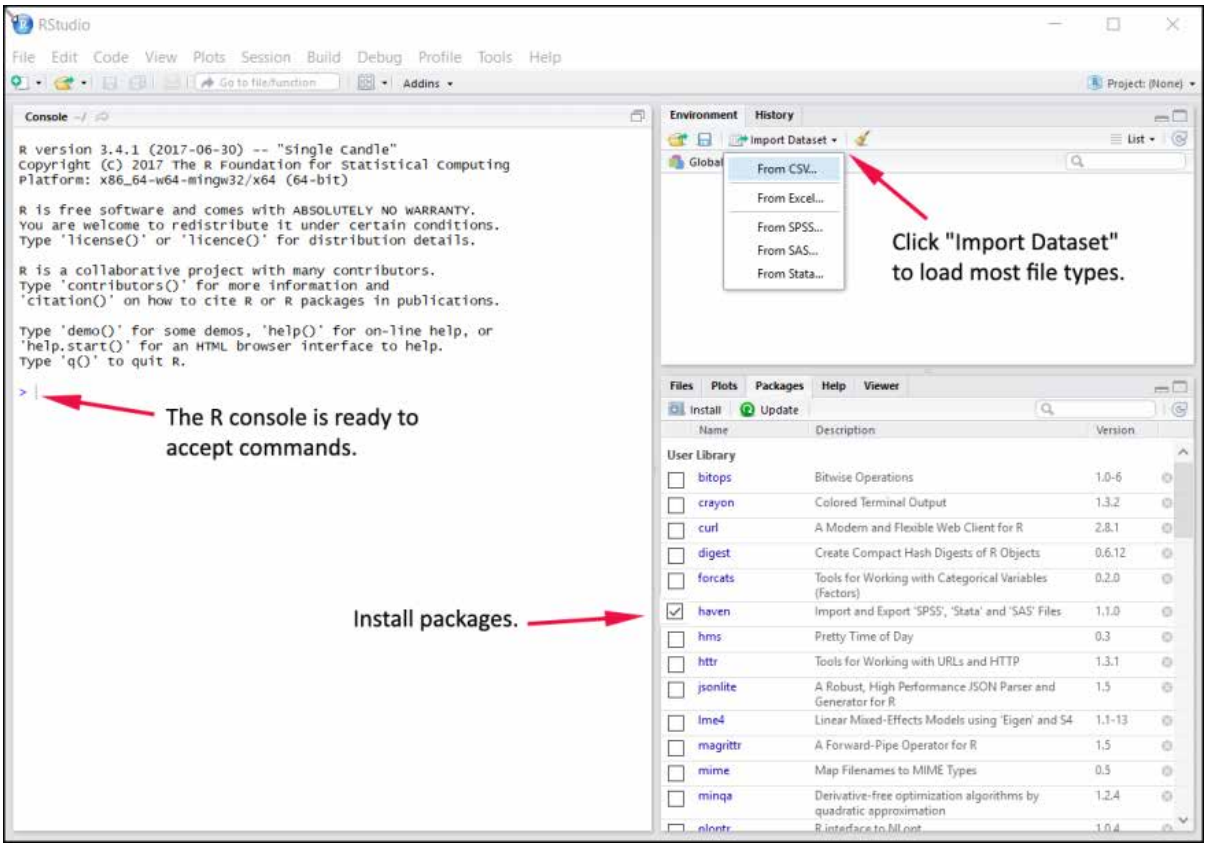


Figure Appendix2.4

Caption: “The Import Window in RStudio”

File: garson_Appendix2-4.tif

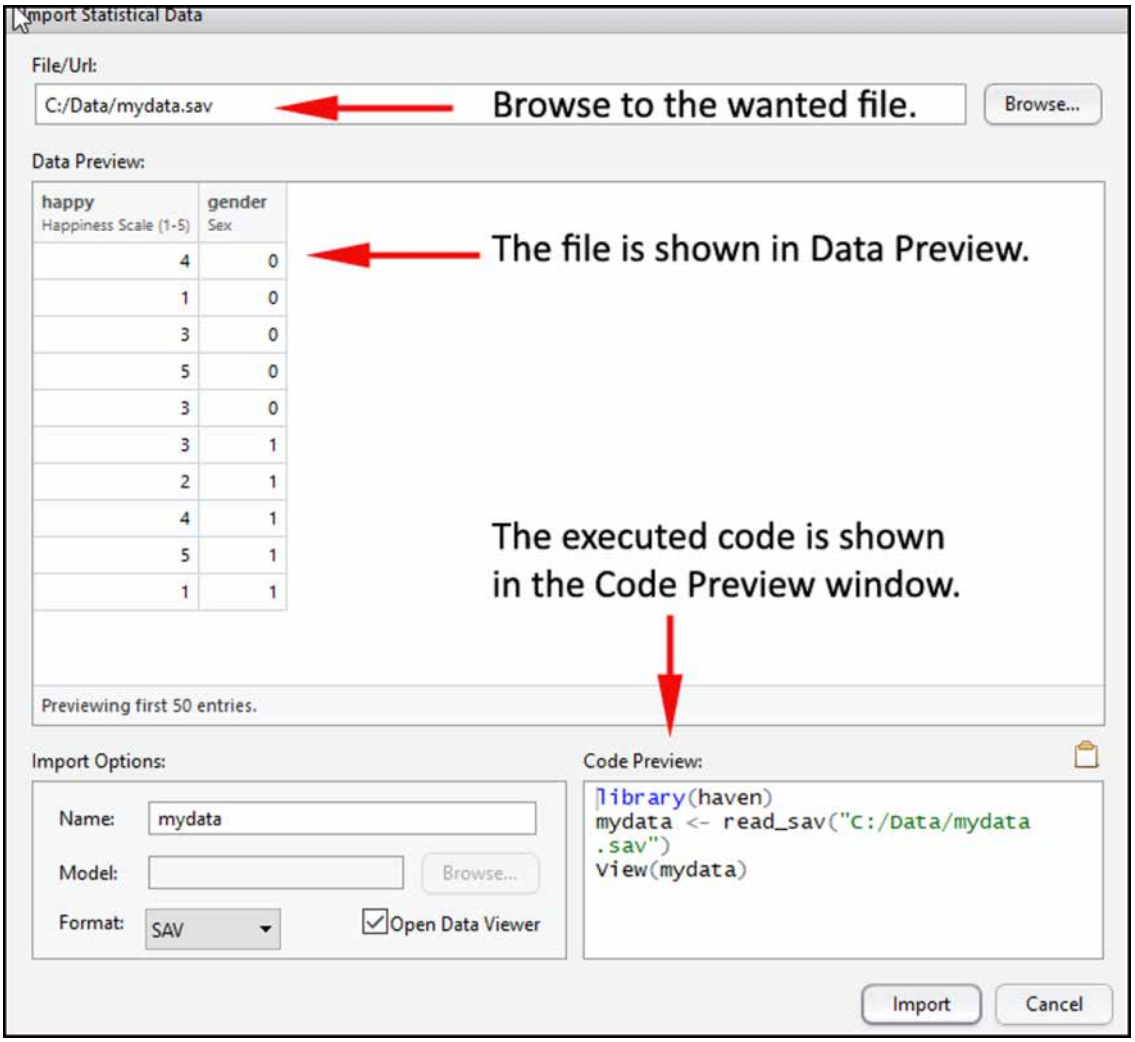


Figure Appendix2.5

Caption: “RStudio with Imported Data”

File: garson_Appendix2-5.tif

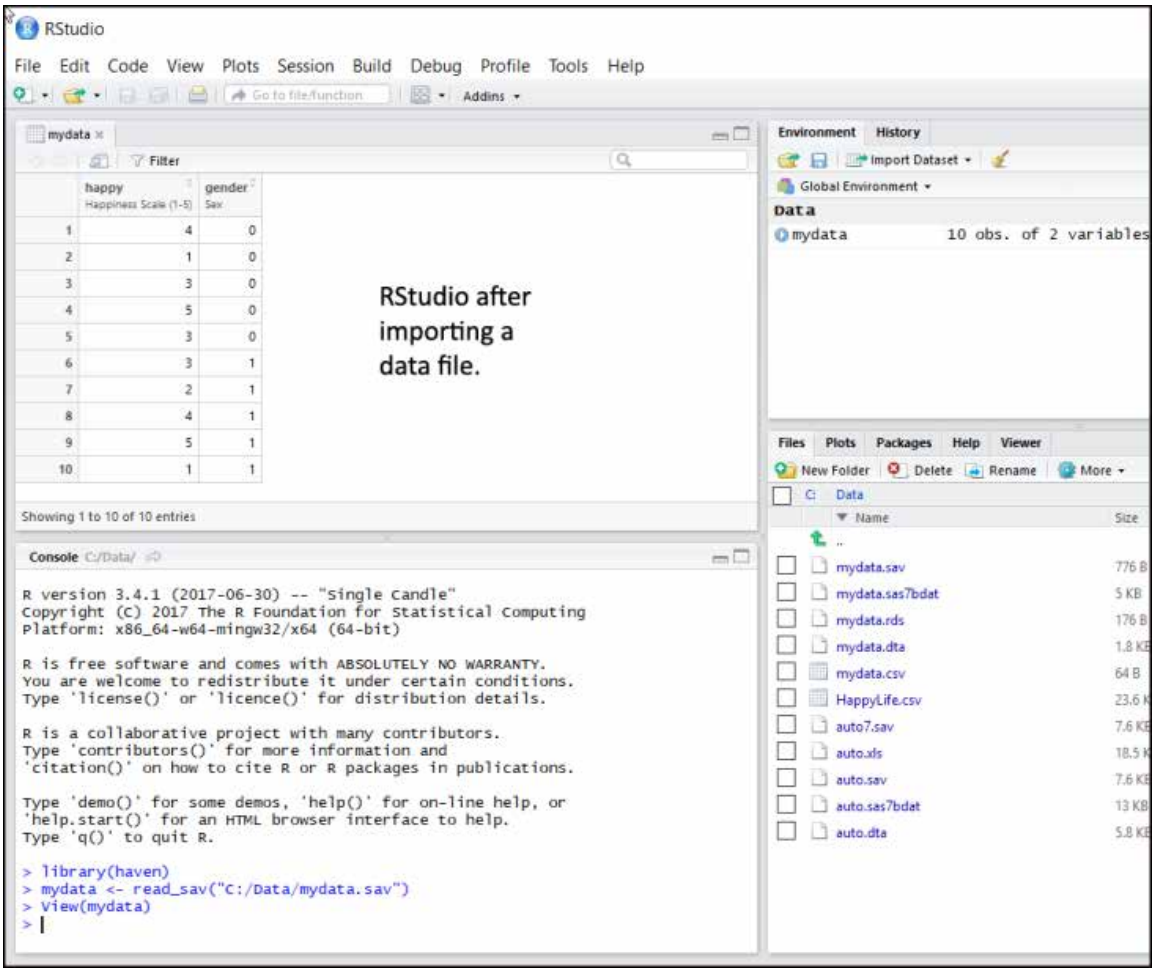
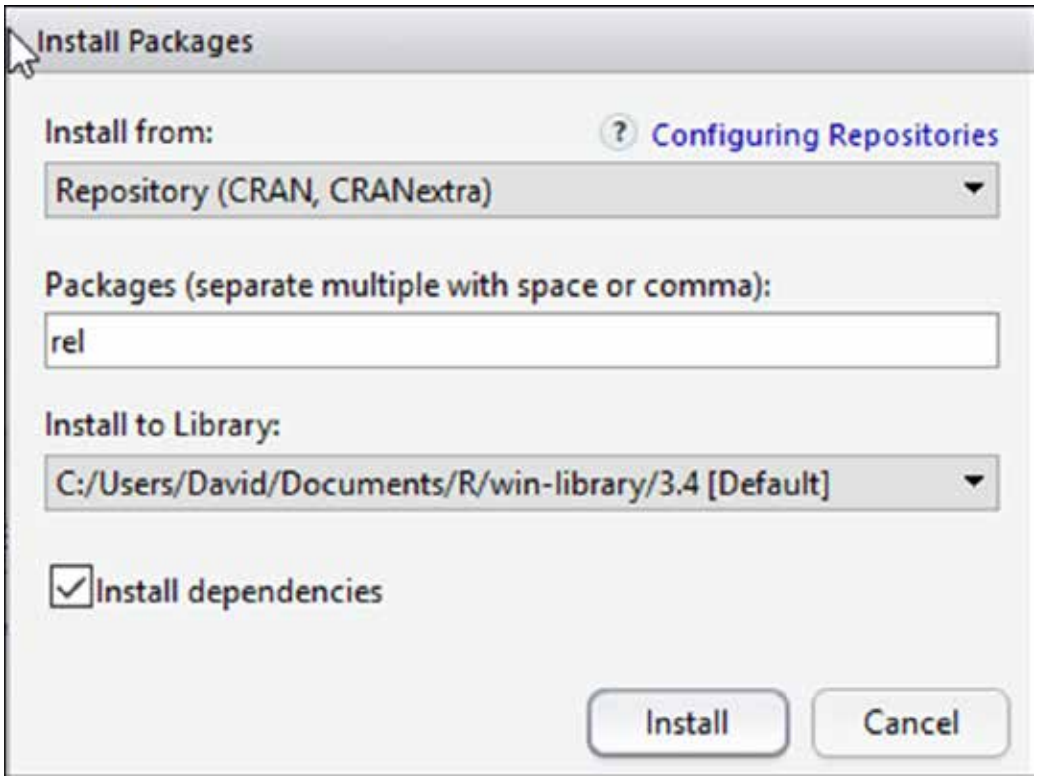


Figure Appendix2.6

Caption: “Install Packages Window in RStudio”

File: garson_Appendix2-6.tif

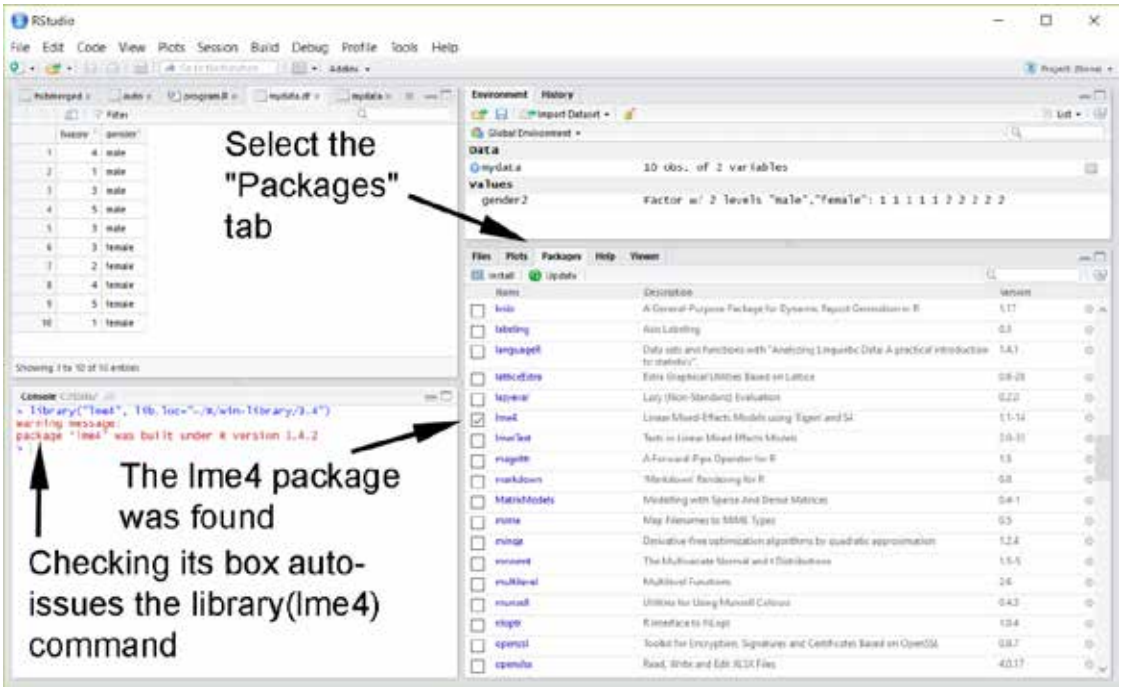


Copyright @c 2015 by G. David Garson and Statistical Associates Publishing

Figure Appendix2.7

Caption: “Packages Tab in RStudio”

File: garson_Appendix2-7.tif



Statistical Associates Publishing

Blue Book Series

NEW! For use by a single individual, our entire current library is available at Amazon in no-password pdf format on DVD for $120 plus shipping. Click on http://www.amazon.com/dp/1626380201 . Includes one year of free updates when email address is provided.

FREE! If you buy the Kindle version and send an email with the receipt to sa.publishers@gmail.com, we will send you a free no-password pdf version if you want that format also. Note that our home page links to the free Kindle for PC Reader if you do not own a Kindle.

NEW FOR CLASS USE! If you are requesting this for class use, consider recommending site licensing so the ebook is free for everyone at your institution and is always available. For class use, see our new low-cost site license policy for university libraries and others at http://statisticalassociates.com/FAQ.htm#sales . Site license for a university is $100 per title.

Association, Measures of
Case Study Research
Cluster Analysis
Correlation
Correlation, Partial
Correspondence Analysis
Cox Regression
Creating Simulated Datasets
Crosstabulation
Curve Estimation & Nonlinear Regression
Data Management, Introduction to
Delphi Method in Quantitative Research
Discriminant Function Analysis
Ethnographic Research
Factor Analysis
Focus Group Research
Game Theory
Generalized Linear Models/Generalized Estimating Equations
GLM Multivariate, MANOVA, & Canonical Correlation
GLM: Univariate & Repeated Measures
Grounded Theory
Life Tables & Kaplan-Meier Survival Analysis
Literature Review in Research and Dissertation Writing
Logistic Regression: Binary & Multinomial

Log-linear Models,
Longitudinal Analysis
Missing Values & Data Imputation
Multidimensional Scaling
Multiple Regression
Narrative Analysis
Network Analysis
Neural Network Models
Ordinal Regression
Parametric Survival Analysis
Partial Correlation
Partial Least Squares Regression
Participant Observation
Path Analysis
Power Analysis
Probability
Probit and Logit Response Models
Research Design
Scales and Measures
Significance Testing
Social Science Theory in Research and Dissertation Writing
Structural Equation Modeling
Survey Research & Sampling
Testing Statistical Assumptions
Two-Stage Least Squares Regression
Validity & Reliability
Variance Components Analysis
Weighted Least Squares Regression

Statistical Associates Publishing
http://www.statisticalassociates.com
sa.publishers@gmail.com