CHAPTER 10

Exporting from R

Being able to export from R makes R more useful. Objects may be exported to files or connections. Since R Studio does not have specialized methods for exporting objects, only command line R methods are covered here. In this chapter, we cover exporting to external files on the hard drive and to the console. You can find information about exporting to connections by entering **?connections** at the R prompt or by using the "Help" tab in R Studio.

There are a number of functions that export to external text files, eight of which we will discuss in this chapter. The first is the function `dump()`. The function `dump()` can write named objects of any kind to an external file in text format.

The next function is `sink()`. The function `sink()` can sink output that would normally be displayed at the console to an external file in text format. Next is the function `write()`. The function `write()` can write atomic data to an external file in text format. Next comes the function `write.matrix()`. For matrices and data frames, the function `write.matrix()` exports the matrix or data frame in tabular text format.

The next two functions are `write.table()` and `write.csv()`. For objects that can be coerced to a data frame, `write.table()` and `write.csv()` can write the object to an external file while maintaining the data frame structure. The functions are slower but more sophisticated than `write.matrix()` and write tabular text data.

M. Tollefson, *R Quick Syntax Reference*, https://doi.org/10.1007/978-1-4842-4405-0_10

The last two functions we cover are save() and saveRDS(). These functions save objects in binary format by default and are the functions of choice to transfer data sets and functions between workspaces.

There are also functions that convert data frames to Excel, SPSS, SAS, and Stata formats, which we briefly cover in this chapter. Also, output at the console can be cut and pasted to an external file.

A table of importing and exporting functions covered in the book is given in Table 10-1, where some functions are paired.

The Function dump()

The function `dump()` takes a vector of object names and exports the contents of the objects to a file. The file will have a text format. (The function `source()` reads the dumped file into a two element list containing the value read and a logical value indicating if the result is visible. If more than one object is dumped, only the last object is sourced.)

The first argument to `dump()` is **list** and is a collection of the objects to be dumped. To enter the objects into the function, the object names are collected into a character vector with the object names in quotes.
For example:

```
> a = function(){ print( 1:4 ) }
> b = expression( x~y )
> c = list( 1:4, "a" )
> d = c( 1, 2, 3, 4 )

> dump( c( "a", "b", "c", "d" ), file="" )
a <-
function(){print(1:4)}
b <-
expression(x ~ y)
c <-
```

```
list(1:4, "a")
d <-
c(1, 2, 3, 4)
.
```

Other than the vector of named objects, the function takes the arguments **file**, **append**, **control**, **envir**, and **evaluate**.

The argument **file** contains the location to which the function writes. If the argument is set to **“”**, the dump goes to the console or stdout() if stdout() is not the console. A hard drive address is an option for **file** and can be either relative to the working directory or an absolute address. For a hard drive address, the location is a character string or a character object. The default value is **“dumpdata.R”**.

The argument **append** is a logical variable. If **append** is **TRUE** and **file** equals a file name, `dump()` appends the dump to the existing file. If **FALSE**, the existing file is overwritten. The default value is **FALSE**.

The argument **envir** is an argument of mode `environment` and tells `dump()` in which environment to look for the objects to be dumped. The default value is **parent.frame()**.

The arguments **control** and **evaluate** have to do with saving and reloading functions, where `dump()` is used to save and `source()` is used to load the function. **Control** gives the deparse options used by dump(), by default **“all,”** and **evaluate** is a logical variable that tell R whether to evaluate promises, by default **TRUE**.

You can access the help page by entering **?dump** at the R prompt or under the “Help” tab in R Studio.

The Function sink()

The function **`sink()`** sends output from command line commands to a file or connection. The function **`sink()`** continues writing until **sink()** or **sink(file=NULL)** is entered at the R prompt. The function takes four arguments: **file**, **append**, **type**, and **split**.

The **file** argument tells `sink()` where to write the output. If writing to a hard drive file, the write location is a character argument, which is a hard drive address within quotes. The address can be relative to the workspace folder or absolute. The option **file=""** does not work for `sink()`.

The second argument, **append**, tells `sink()` whether to append or overwrite the file. The argument is a logical argument. For **append** equal to **TRUE**, the file is appended. For **FALSE**, the file is overwritten. The default value is **FALSE**.

The third argument, **type**, tells `sink()` which of two possible streams to sink. The argument is a character argument, which can take on one of two values: **output** or **message**. For **output**, the output stream is sent to the file. For **message**, any messages generated by the command are sent to the file. The default value is **output**.

The fourth argument, **split**, is a logical argument that tells `sink()` how to split the stream. If **FALSE**, the output stream is not sent to the console. If **TRUE** the output stream is sent to both the file and the console. The default value is **FALSE**.

Following is an example of the use of `sink()`:

```
> sink( "test.txt" )
> rnorm( 10 )
> sink()
```

The file `"test.txt"` is relative to the folder containing the R workspace. The contents of `test.txt` are

```
[1] -0.30618294 -0.52505474  0.47243057 -0.89954490
    -1.06653790  0.03690703
[7]  1.81562861 -0.74177999 -0.28352208 -1.28133196
```

Note that the command lines are not output.

For more information, enter **?sink** at the R prompt or use the "Help" tab in R Studio.

The Function write()

The function `write()` can write atomic objects to a file, and it writes in tabular text format. The objects are entered as a single vector, for example, as a collection of objects collected using `c()`. If the data are in a matrix or array, `write()` reads the data down the columns or dimensions of the matrix or array, but writes across rows in the two-dimensional output.

The first argument is **x,** the vector to be exported. The argument is usually any object of an `atomic mode`. (See the help page for cat() for more information on acceptable modes.)

Other than the vector to be exported, there are four more arguments to `write()`. The first is the character argument **file**, which tells `write()` where to write the output. The argument can be a connection or a location on a hard drive, relative to the workspace or absolute. If **""** is given for **file**, the output is sent to the console or to the value of stdout() if stdout() is not the console. The default value is **"data."** The object can also be piped to a command in R.

The second argument is **ncolumns**. The argument **ncolumns** can be logical, numeric, or complex, and if it is not an integer, it is coerced to an integer. The argument gives the number of columns for the exported table. By default, the argument takes on the value **if(is.character(x)) 1 else 5**. So, if the data is of mode `character`, the output matrix has one column by default. Otherwise, the output matrix has five columns by default.

The input file does not have to be of a length divisible by **ncolumns**. In other words, the last row does not have to be complete.

The third argument, **append**, is a logical argument. If set to **TRUE**, the output is appended to the file. If set to **FALSE**, the file is overwritten. The default value is **FALSE**.

The fourth argument, **sep**, is a character string that gives the characters to be placed between the elements of the output matrix. The default value is a white space.

An example follows:

```
> x=1:4
> y=5:8
> z=rbind( x, y )
> w=paste( "a", 1:3, sep="" )
> b = rep( " ", 4 )

> write( c( x, y, b, z, b, w ), file="", ncol=4, sep=" + " )
1 + 2 + 3 + 4
5 + 6 + 7 + 8
  +   +   +
1 + 5 + 2 + 6
3 + 7 + 4 + 8
  +   +   +
a1 + a2 + a3
```

Note that when entered separately, **x** and **y** each exports as a row. When **x** and **y** are bound together into a matrix using `rbind()`, `write()` goes down the two columns to read and writes the result across the rows. Also note that there are four columns as specified by **ncol** and that there are only three elements in the last row.

You can find more information about `write()` by entering **?write** at the R prompt or by using the "Help" tab in R Studio.

The Function write.matrix()

The function `write.matrix()` is in the package **MASS**, which is not a package that is loaded by default. **MASS** can be loaded by entering **library(MASS)** at the R prompt since **MASS** is installed by default when R is installed. According to the CRAN writers, `write.matrix()` is much faster than `write.table()` for large data sets, so the function may be preferable if the matrix or data.frame is large and the data frame is appropriate.

The function has the arguments **x**, **file**, **sep**, and **blocksize**. The argument **x** is the object to be exported and should be a matrix or a data. frame containing objects of just one mode. If modes are mixed, some strange things can happen. The function only exports in one mode, which is why `write.matrix()` is faster than `write.table()`.

The argument **file** gives the location to which to write. For addresses on the hard drive, the argument is of mode character and is either relative to the workspace or absolute. The default value is **“”**, which directs output to the console or to the value of stdout() if it is not the console.

The argument **sep** is a character string that gives the separator between the outputted elements. The argument defaults to white space.

The argument **blocksize** has no default value and does not need to be entered. If entered, the argument tells `write.matrix()` the size of the block of data to be transferred at one time. According to the CRAN writers, the value should be as large as possible for the amount of memory available.

Here is an example. The object `mat` is a matrix, the object `mat.df` is a data frame of one mode, the object `mat.df.x` is a data frame of mixed numeric and character modes. The default value of **file** is used as follows, so the outputs goes to the console.

```
> mat = matrix( 1:4, 2, 2, dimnames=list( c( "r1", "r2"),
c( "c1", "c2" ) ) )
> mat
   c1 c2
r1  1  3
r2  2  4

> write.matrix( mat )
c1 c2
1 3
2 4
```

```
> mat.df=data.frame( mat )
> mat.df
   c1 c2
r1  1  3
r2  2  4

> write.matrix( mat.df )
c1 c2
1 3
2 4

> mat.df.x = data.frame( mat, c( "art", "birth" ) )
> mat.df.x
   c1 c2 c..art....birth..
r1  1  3               art
r2  2  4             birth

> write.matrix(mat.df.x)
c1 c2 c..art....birth..
1     3     art
2     4     birth
```

More about `write.matrix()` can be found by entering **?MASS::write.matrix** at the R prompt or by loading MASS in R Studio, then using the "Help" tab.

The Functions write.table() and write.csv()

The functions `write.table()` and `write.csv()` also export matrices and data frames in tabular text format. The two are essentially the same function but with different defaults. All of the defaults for `write.table()` can be changed. For `write.csv()`, the defaults **append**, **col.names**, **sep**, **dec**, and **qmethod** cannot be changed. (As with `read.csv()` there is also

the function `write.csv2()` for European users. The function `write.csv2()` uses a semicolon for the separator and a comma for the decimal point, but otherwise is the same as `write.csv()`.)

The functions take the arguments **x**, **file**, **append**, **quote**, **sep**, **eol**, **na**, **dec**, **row.names**, **col.names**, **qmethod**, and **fileEncoding**. The argument **x** is the object to be exported and must be an object that can be coerced to a data frame.

The argument **file** gives the location to which to export. For external files, **file** is of mode `character` and the address is either relative to the workspace or absolute. If **file** equals **“”**, then the functions export to the console or to stdout() if stdout() is not the console. The value of **file** is **“”** by default.

The argument **append** is a logical argument. If **append** is **TRUE**, then the file is appended with the new data frame. If **FALSE**, the file is overwritten. The default value is **FALSE**.

The argument **quote** is either logical or a numeric vector of column numbers and gives rules for placing quotes around elements. The default value is **TRUE**. If set to **FALSE**, nothing is quoted.

The argument **sep** is a character argument and gives the separator to be used between the elements of the exported data. The separator is entered within quotes. For `write.table()`, the default value is a white space. For `write.csv()`, the value is a comma.

The argument **eol** is an argument of mode `character` and gives the end of line delineator. By default, **eol** is equal to “\n”. The correct value for **eol** varies with operating system. Use “\n” for Windows, “\r” for OS X, and “\r\n” for Linux.

The argument **na** is also a character argument and gives the string to be output where data is missing. The default value is “NA.”

The argument **dec** is another character argument and gives the character to be used as the decimal point. By default, **dec** equals “.”.

The argument **row.names** is either a logical value or a character vector of row names. Note that `write.table()` and `write.csv()` treat the row names differently if **row.names** is set to **TRUE** or to a character vector of names. If a column of row names is in the exported data frame, the function `write.table()` does not create a blank character string for the name of the row name column, while `write.csv()` does. If **row.names** is equal to **FALSE**, there is no difference between the two with regard to row names since no row names are exported.

If no row names are given, row names are not present in the data. frame (for example, if a matrix without row names is entered for **x**) and **row.names** is **TRUE**, then the rows are given names, starting with "1" and incrementing by one with each row. By default, **row.names** equals **TRUE**.

The argument **col.names** is either logical or a character vector of column names. For `write.table()`, if **col.names** is set equal to **TRUE**, either the column names are taken from the data frame or, if no names are present in the data frame, column names are created starting with "V1" and incrementing the integer by one for each new column. If column names are supplied, the column names are set equal to the supplied names.

As noted previously, for `write.table()`, by default, no column name value is given for the column of row names if the row name column exists in the exported file. However, if **col.names** is set equal to **NA**, then columns are treated the same as for **col.names** set equal to **TRUE** except that a blank character string is added for the row name column. If **row.names** equals **FALSE**, then setting **col.names** equal to **NA** gives an error. If **col.names** is set equal to **FALSE**, no column names are assigned in the exported file.

For `write.csv()`, the default for **col.names** depends on the value of **row.names**. The default cannot be changed. If **row.names** equals **TRUE**, **col.names** is set to **NA**. Otherwise, **col.names** is set equal to **TRUE**.

In either case, column names are given by either the names in the data frame or, if there are no column names in the data frame, names starting with "V1" and with the integer incrementing by one for each new column.

The next argument is **qmethod** and can take on the values "escape" or "double". The default value is "escape". The argument gives instructions for double quoted values. See the help page for `write.table()` for more information. The last argument is **fileEncoding**, which need not be assigned, but if assigned tell R how to encode the output, for example in UTF-8 format.

Here are some examples. The object `mat.df.x` is a data frame with row and column names. The object `mat` is a matrix that does not have row or column names.

```
> mat.df.x
   c1 c2    C3
r1  1  3   art
r2  2  4 birth

> write.table( mat.df.x )
"c1" "c2" "C3"
"r1" 1 3 "art"
"r2" 2 4 "birth"

> write.table( mat.df.x, sep="," )
"c1","c2","C3"
"r1",1,3,"art"
"r2",2,4,"birth"

> write.table( mat.df.x, sep=",", col.names=NA )
"","c1","c2","C3"
"r1",1,3,"art"
"r2",2,4,"birth"
```

```
> write.table( mat.df.x, col.names=F )
"r1" 1 3 "art"
"r2" 2 4 "birth"

> write.table( mat.df.x, row.names=F, col.names=F )
1 3 "art"
2 4 "birth"

> write.table( mat.df.x, sep=",", row.names=F )
"c1","c2","C3"
1,3,"art"
2,4,"birth"

> write.csv( mat.df.x )
"","c1","c2","C3"
"r1",1,3,"art"
"r2",2,4,"birth"

> write.csv( mat.df.x, row.names=F )
"c1","c2","C3"
1,3,"art"
2,4,"birth"

> mat
     [,1] [,2]
[1,]    1    3
[2,]    2    4

> write.table( mat )
"V1" "V2"
"1" 1 3
"2" 2 4
```

```
> write.table( mat, row.names=c( "r1", "r2" ), col.names=NA )
"" "V1" "V2"
"r1" 1 3
"r2" 2 4

> write.table( mat, row.names=F, col.names=F )
1 3
2 4

> write.csv( mat )
"","V1","V2"
"1",1,3
"2",2,4

> write.csv( mat, row.names=c( "r1", "r2" ) )
"","V1","V2"
"r1",1,3
"r2",2,4
```

To access the help page for `write.table()` and `write.csv()`, enter **?write.table** at the R prompt or use the "Help" tab in R Studio.

The Function save()

The function save() saves R objects, by default in binary form, to a file. The saved objects can be loaded into a workspace using load() or sometimes data() or attached to a workspace using attach(). See the previous chapter for information about load(), data(), and attach().

The function save() takes the arguments **...** , **list**, **file**, **ascii**, **version**, **envir**, **compress, compression_level**, **eval.promises**, and **precheck**. The names of the objects to be saved can be entered in two ways: symbols or character strings containing the object names separated by commas or a

character vector containing the names of the objects (or both).
The argument **file** gives the location where the objects are to be saved.
For example:

```
> save( "ClintonCorpus", "mat", list=c( "junk", "trst" ),
file="save.bin" )

> load( "save.bin", ver=T )
Loading objects:
  junk
  trst
  ClintonCorpus
  mat

> class( junk )
[1] "list"

> class( trst )
[1] "asS4"
attr(,"package")
[1] ".GlobalEnv"

> class( ClintonCorpus )
[1] "SimpleCorpus" "Corpus"

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

Here, four objects are saved to the file “save.bin,” which is then reloaded. The four objects belong to different classes.

Any types of objects can be saved using save(). When loaded, the objects are loaded into the workspace under their original names and are not displayed at the console.

The argument "ascii" tells save() to write an ASCII file if given the value TRUE. If given FALSE—the default—a binary file is created. For NA, see the help page for save(). From the help page for save(), the argument "version" tells save() which version of the workspace format to use. The choices are NULL—for the current default format and 1, 2, or 3 for the default formats in R 0.99.0 to R 1.3.1, R 1.4.0, and from R 3.5.0 on respectively.

The argument "envir" tells save() the environment in which to find the object(s). The mode of the argument is environment, and the default value is "parent.frame()." The argument "compress" indicates what kind of compression to do or if to do compression. If FALSE, no compression is done. If TRUE, "gzip" compression is done. Setting the value equal to "gzip," "bzip2," or "xz" tells save() to use that method of compression. The default value is "isTRUE(!ascii)," so if "ascii" is FALSE, compression is done by default. According to the help page for save(), this argument is ignored if the file argument is a connection or if the workspace format is version 1.

The argument "compression_level" gives the level of compression if "compress" is not equal to FALSE. If the compression method is "gzip," the default level is "6." For "bzip2" or "xz," the default level is "9."

The argument "precheck" is a logical argument that when set equal to TRUE, the default tells save() to check to see if an object exists before opening a file or connection. If set equal to FALSE, the file or connection is opened even if nothing is saved. For version 1, "precheck" does not apply—according to the help page for save().

The argument "safe" is a logical argument that, when set equal to TRUE, tells save() to open a temporary file when saving a workspace in case the save fails. TRUE is the default value but causes the save to use more disk space during the saving. If set equal to FALSE, the workspace can be lost if the save fails.

For more information about save(), enter ?save at the R prompt or use the "Help" tab in R Studio.

The Function saveRDS()

The function saveRDS() saves a single object to a file. Objects saved with saveRDS() can be loaded with readRDS()—see the previous chapter. The arguments to saveRDS() are "object," "file," "ascii," "compress," and "refhook."

The argument "object" is set equal to the name of the object, which is not quoted. The argument "file" is the name to be assigned to the file, which is a character string or a connection. The argument "ascii" behaves the same as for save().

The next argument is "version." From the help page for saveRDS(), setting "version" equal to NULL tells the function to use the default value—currently 2—since R 1.4.0. For R 3.5.0 and later, the legal options for "version" are 2 and 3.

The argument compress behaves like in save(). See the help page for information about the argument "refhook."

For more information, enter ?saveRDS at the R prompt or use the "Help" tab in R Studio.

Matching Importing and Exporting Functions

Many of the importing and exporting functions are paired with each other. For example: source() with dump(); save() with load(), data() or attach(); dput() with dget(); and write.table() with read.table(). Table 10-1 gives importing and exporting functions based on pairing.

Table 10-1. *Paired Import and Export Functions*

Importing	Exporting	Use
source()	dump()	Create and source external files in a text format
scan()		Read textual data as a vector
	sink()	Write textual output from commands
	write()	Write textual data in tabular form
	write.matrix()	Write a matrix or data frame using one atomic mode, maintains the original structure
read.table() read.csv()	write.table() write.csv()	Read and write a matrix or data frame in textual form, maintains the original structure
load() data() attach()	save()	Read and write objects, mainly in binary format, used to transfer objects
readRDS()	saveRDS()	Read and write an object, mainly in binary format, used to transfer an object
dget()	dput()	Of historical interest, uses the text format

Other Exporting Functions

Like the functions that read in data, there are a variety of functions that write data. The CRAN page on the package **rio** for importing and exporting data lists many of the packages and what they do. The CRAN vignette can be found at https://cran.r-project.org/web/packages/rio/vignettes/rio.html.

For SPSS, SAS, and Stata, the function `write.foreign()`, which can be found in the package **foreign**, can import and export in the correct format. The function `write.foreign()` also exports in some other formats. Other exporting functions can also be found in the package **foreign**.

The package **foreign** is one of the packages installed by default. To see the contents of **foreign**, enter **help(package=foreign)** at the R prompt or use the "Packages" tab in R Studio. Click on "foreign" in the list of packages. To load **foreign**, enter **library(foreign)** at the R prompt or check the box to the left of "foreign" under the "Packages" tab in R Studio.

A newer package to read and write SPSS, SAS, and Stata files is the package **haven**. The package is not installed by default, unlike the package **foreign**, so **haven** must be installed before you can load it. After installing **haven**, you can see the contents of **haven** by entering **help(package=haven)** at the R prompt or by using the "Packages" tab in R Studio. Click on "haven" in the list of packages. To load **haven**, enter **library(haven)** at the R prompt or check the box to the left of "haven" under the "Packages" tab in R Studio.

For Excel, there is a package, **xlsx**, specifically for working with Excel. The package **xlsx** is not a default package in R, so it must be installed. After **xlsx** is installed, information about **xlsx** can be found by entering **help(package=xlsx)** at the R prompt. For older Excel files, the package **readxl** has functions to write and read the Excel files. Like the package **xlsx, readxl** is not installed by default, so must be installed before it is loaded.