Data Input and Output

Chapter 8

Stats 20: Introduction to Statistical Programming with R

UCLA

Contents

Acknowledgements: Vivian Lew and Juana Sanchez

Learning Objectives

After studying this chapter, you should be able to:

- Import and export R script files using `dump()` and `source()`.
- Import and export R objects using `save()` and `load()`.
- Differentiate between `.R` files and `.RData` files.
- Import datasets into R using `read.table()` and `read.csv()`.
- Export data tables using `write.table()` and `write.csv()`.

1 Importing and Exporting R Script Files

Recall that an R script (or script file) is a text file containing a set of R commands (i.e., a program). Script files are often stored as .R files.

1.1 The `dump()` Function

Suppose we have constructed an R object that we want to save and recall in a later R session. For example, we can create the `parks_df` data frame from the previous chapter.


```
parks_df <- data.frame(
  "Name" = c("Leslie", "Ron", "April"), "Height" = c(62, 71, 66),
  "Weight" = c(115, 201, 119), "Income" = c(4000, NA, 2000)
)
parks_df
```



```
    Name Height Weight Income
1 Leslie     62    115   4000
2    Ron     71    201     NA
3  April     66    119   2000
```


The **`dump()`** function will store the command necessary to create the input R object(s) in a text/script file on your computer's hard drive. The first argument, called `list`, is a character vector of object names to store in the file, whose name we specify in the second argument called `file`.

```
dump("parks_df", "Parks.R")
```

The created file will be saved to your current working directory.

Question: How can we check what is the current working directory?

1.2 The `source()` Function

The **`source()`** function reads in the file name of an R script and executes all the commands inside the file. The input file for the `source()` function should only contain R code (so any comments should be on a line with `#`). This can be useful for transferring the code for functions and objects between people or computers. It can also be helpful to compartmentalize a long program into a few separate .R files and use `source()` to combine all the code together at the end.

The `file` argument in `source()` should contain the name of the script file in quotations. The file can either be in a local directory (i.e., on your computer) or online. If the file is in the current working directory, then the file name is sufficient. If the file is outside the current working directory, the whole path name (or website address) has to be specified. Specifying a file name without a path name that is not in the working directory will throw an error.

```
source("whoops.R")
```

```
Warning in file(filename, "r", encoding = encoding): cannot open file
'whoops.R': No such file or directory
```

```
Error in file(filename, "r", encoding = encoding): cannot open the connection
```

As an example, we can remove the `parks_df` object from the workspace and use `source()` to create the object from the saved `Parks.R` file.

```
rm(parks_df)
source("Parks.R")
parks_df
```

```
    Name Height Weight Income
1 Leslie     62    115   4000
2    Ron     71    201     NA
3  April     66    119   2000
```


Caution: Running code using the `source()` function differs from entering code in the command line in a few ways.

- The `source()` function will read and parse the entire script file *before* any line is executed. This means that a syntax error in a single line of code in the script will cause none of the code in the script to be executed.

- Printing results from R commands will not be done automatically. For example, if we type `1:10` in the command line, the vector of values will be printed on the screen. If we use `source()` to read a script file containing only the line `1:10`, the command will run but the vector will not be printed. To force the script file to print the results from a command, use the `print()` function.

- Objects created in the `source()` file will be created in the workspace. Any objects with the same name that were previously in the workspace will be overwritten.

As an example, suppose we write the following commands and save it in a .R file called `Example Script.R`:

```
first_ten <- 1:10
first_ten

first_five <- 1:5
print(first_five) # Print the first_five vector

mean_fn <- function(x) {
  sum(x) / length(x)
}

mean_vec <- mean_fn(first_five + first_ten)
mean_vec
```

By using `source()` to run the script file, only `print(first_five)` will be printed to the console, but all the objects created in the script will appear in the workspace.

```
source("Example Script.R")
```

```
[1] 1 2 3 4 5
```

```
ls()
```

```
[1] "first_five" "first_ten"  "mean_fn"    "mean_vec"   "parks_df"
```

2 Importing and Exporting R Data Files

Recall that all objects created and stored in an R session are collectively called the workspace or global environment. When quitting an R session, a popup will ask if you want to save an image of the current workspace. When saving the workspace image, a file called `.RData` wil be created in the current working directory that contains all the R objects in the workspace. If a new R session is opened in the same working directory, then the workspace will automatically be restored using the `.RData` file. If R is started in a different directory that does not contain a `.RData` file, we would need to manually load the workspace image.

2.1 The `save()` Function

In general, **RData** files are binary files that contain R objects. The **`save()`** function can be used to save specific R objects from the workspace into an RData file. The objects to be saved can be listed as separate arguments in the `save()` function. The `file` argument then specifies the name of the resulting RData file.

```
save(parks_df, first_ten, file = "Chapter 8.RData")
```

To save all of the objects in the workspace (i.e., make a workspace image), we can use the **`save.image()`** function, which is an example of a wrapper function for `save()`. The `save.image()` function actually calls `save()` to save objects, but a few of the default arguments are changed to be more convenient for saving the entire workspace. In particular, `save.image()` runs the command `save(list = ls(all.names = TRUE), file = ".RData", envir = .GlobalEnv)`.

```
save.image("Workspace.RData") # Same as save.image()
```


2.2 The `load()` Function

The **`load()`** function reads in the file name of an RData file and loads the contained objects into the workspace. The path of the file name needs to be specified if the file is not in the current working directory.

```
rm(list = ls()) # Kaboom!
load("Chapter 8.RData") # Load the parks_df and first_ten objects
ls()
```

```
[1] "first_ten" "parks_df"
```

```
load("Workspace.RData") # Load the workspace image
ls()
```

```
[1] "first_five" "first_ten"  "mean_fn"    "mean_vec"   "parks_df"
```

Objects loaded from the RData file will be stored in the workspace, and any objects with the same name that were previously in the workspace will be overwritten.

Caution: *RData files are not the same as R script files.* Script files contain R commands which are read and executed. RData files contain R objects that can be directly loaded into the workspace. For long programs, it may be more efficient to just load the saved results from an RData file rather than rerunning the entire long program to recreate the same results.

3 Importing and Exporting Data Tables

3.1 The `read.table()` Function

Typing data directly into R can be cumbersome and prone to data entry errors. It is often useful to import data from files outside of R.

The **`read.table()`** function is used to import data that is stored in a table format (typically a plain text or tab delimited file). If possible, be sure to look at the data to see if there is a **header** (i.e., variable names) and how values are separated (spaces, commas, tabs, etc.). The output of `read.table()` is a data frame, so it allows for data with mixed types of variables.

The first argument of `read.table()` is `file`, which specifies the name and location of the file containing the data table. For online datasets, the `file` argument can input the complete web address (URL) in quotations.

The `header` argument inputs a logical value that specifies if the first line (row) of the data contains the names of the variables in the data. The default for `read.table()` is `header = FALSE`.

The `sep` argument inputs a character value that specifies the type of **separator** that is used to separate values within a line of the data table. The default value is `sep = ""`, which is the "white space" separator, meaning values are separated by one or more spaces, tabs, newlines, or carriage returns. The tab separator is specified by the special character `"\t"`.

As an example of an online dataset, we will use the LA ozone data found at:
https://web.stanford.edu/~hastie/ElemStatLearn/datasets/LAozone.data

The documentation for this data is available at:
https://web.stanford.edu/~hastie/ElemStatLearn/datasets/LAozone.info.txt

By looking at the data, we see that the first line consists of variable names, so there is a header, and the values in each line are separated by commas.


```
# Read the data into R
oz <- read.table("https://web.stanford.edu/~hastie/ElemStatLearn/datasets/LAozone.data",
  header = TRUE, sep = ","
)
```


Once the data is loaded into R, make sure that the data is loaded correctly. For example, make sure the dimensions of the data correspond to the proper number of observations and variables.

```
head(oz)
```

```
  ozone   vh wind humidity temp  ibh dpg ibt vis doy
1     3 5710    4       28   40 2693 -25  87 250   3
2     5 5700    3       37   45  590 -24 128 100   4
3     5 5760    3       51   54 1450  25 139  60   5
4     6 5720    4       69   35 1568  15 121  60   6
5     4 5790    6       19   45 2631 -33 123 100   7
6     4 5790    3       25   55  554 -28 182 250   8
```

```
names(oz)
```

```
 [1] "ozone"    "vh"       "wind"     "humidity" "temp"     "ibh"
 [7] "dpg"      "ibt"      "vis"      "doy"
```

```
dim(oz)
```

```
[1] 330  10
```

Question What happens if we do not specify `header = TRUE` (or specify it incorrectly)? What about `sep = ","`?

Caution: The `read.table()` function expects that each row in the data has the same length (i.e., each line has the same number of values), and it will throw an error if some rows have a different length. This may occur if there are blank/missing values in the data. To allow for missing values, set the optional argument `fill = TRUE`, which will then fill in blank or missing fields as `NA`. If there are some rows with a different length, the `count.fields()` function counts the number of fields (separated by the `sep` value) in each line of the data table.

Suppose we downloaded the LA ozone dataset to our computer and placed the file in the current working directory. Then the following command will work:

```
oz <- read.table("LAozone.data", header = TRUE, sep = ",")
```

3.2 The `read.csv()` Function

Many datasets have fields that are separated by commas and saved as .csv files (CSV stands for **comma separated values**). The `read.csv()` function can be used to import data from .csv files.

The `read.csv()` function is a wrapper for `read.table()`. In particular, the standard default arguments are `header = TRUE`, `sep = ","` and `fill = TRUE`.

The `births.csv` file contains data on sample of babies born in North Carolina. Suppose we downloaded the `births.csv` to our computer and placed the file in the current working directory.

```
births <- read.csv("births.csv")
head(births, 3)
```

```
  Gender Premie weight Apgar1 Fage Mage Feduc Meduc TotPreg Visits   Marital
1   Male      No    124      8   31   25    13    14       1     13   Married
2 Female      No    117      8   36   26     9    12       2     11 Unmarried
3   Male      No    107      3   30   16    12     8       2     10 Unmarried
  Racemom Racedad Hispmom Hispdad Gained     Habit MomPriorCond BirthDef
1   White   White NotHisp NotHisp     40 NonSmoker          None     None
2   White   White Mexican Mexican     20 NonSmoker          None     None
3   White Unknown Mexican Unknown     40 NonSmoker At Least One     None
     DelivComp BirthComp
1 At Least One      None
2 At Least One      None
3 At Least One      None
```

```
names(births)
```

```
 [1] "Gender"       "Premie"       "weight"       "Apgar1"       "Fage"
 [6] "Mage"         "Feduc"        "Meduc"        "TotPreg"      "Visits"
[11] "Marital"      "Racemom"      "Racedad"      "Hispmom"      "Hispdad"
[16] "Gained"       "Habit"        "MomPriorCond" "BirthDef"     "DelivComp"
[21] "BirthComp"
```

```
dim(births)
```

```
[1] 1998   21
```

There are other wrapper functions of `read.table()` with similarly convenient default arguments. Some common ones are `read.csv2()`, `read.delim()`, and `read.fwf()`, which import data tables from semicolon separated values, tab delimited data, and fixed width formatted files, respectively.

3.3 The `write.table()` Function

The **`write.table()`** and **`write.csv()`** functions are used to export a matrix or data frame object (i.e., a data table) from R into a plain text or CSV file. If the input is not a matrix or data frame, the object will be coerced into a data frame (which is likely not what you want).

The first argument, called `x`, is the matrix or data frame to write into a .txt or .csv file, whose name we specify in the second argument called `file`.

```
write.table(parks_df, "Parks.txt")
```

The created file will be saved to your current working directory.

The `Parks.txt` file will contain the following lines (if opened in a plain text editor):

```
"Name" "Height" "Weight" "Income"
"1" "Leslie" 62 115 4000
"2" "Ron" 71 201 NA
"3" "April" 66 66 2000
```


Notice that the row and column names of the data frame are preserved, and the default separator is a single space `sep = " "`. To exclude row or column names, set `row.names = FALSE` or `col.names = FALSE`. The separator can be set with the `sep` argument.

Question: How would you read the data from `Parks.txt` into R?

3.4 The `write.csv()` Function

The **`write.csv()`** function, similar to `read.csv()`, is a wrapper function for CSV files.

```
write.csv(parks_df, "Parks.csv")
```


The `Parks.csv` file will contain the following lines:

```
"","Name","Height","Weight","Income"
"1","Leslie",62,115,4000
"2","Ron",71,201,NA
"3","April",66,66,2000
```


Caution: By convention, CSV files include a blank column name for the row names, which helps with formatting the CSV file when read by spreadsheet software (like Excel). However, using `read.csv()` to import the `Parks.csv` data into R will interpret the row names as the first column in the data.

```
read.csv("Parks.csv")
```

```
  X   Name Height Weight Income
1 1 Leslie     62    115   4000
2 2    Ron     71    201     NA
3 3  April     66    119   2000
```

One way to fix this is to set the argument `row.names=1`.

```
read.csv("Parks.csv", row.names = 1)
```

```
    Name Height Weight Income
1 Leslie     62    115   4000
2    Ron     71    201     NA
3  April     66    119   2000
```

3.5 Reading Other File Formats

3.5.1 Excel Files

Microsoft Excel is a widely used appliation for data entry, data manipulation, and basic data analysis that is ubiquitous in most industries. There are several packages that have functions to read and write Excel files. Some common packages are `xlsx`, `gdata`, `XLConnect`, and `readxl`. There may be optional arguments, such as `sheet`, that need to be specified to successfully import from .xls or .xlsx files.

3.5.2 The `foreign` and `tidyverse` Packages

The **`foreign`** package contains functions to read files from SPSS, Stata, SAS, Minitab, and other common statistical software programs.

- The **`read.spss()`** function converts SPSS (.sav) files.
- The **`read.dta()`** function converts Stata (.dta) files.
- The **`read.xport()`** function converts transport (.xport) files from SAS, but cannot read SAS (.sas7bdat) files directly.

Since the `foreign` package was written in 2000, newer versions of SPSS, Stata, and SAS may have incompatible file formats. The **`haven`** package, written in 2015, is able to handle newer files.

The `haven` package is part of the **`tidyverse`**, a system of R packages designed by Hadley Wickham (creator of `ggplot2` and current Chief Scientist at RStudio) for data manipulation, exploration, and visualization in R.