How to save (and load) datasets in R: an overview

Sascha Wolfer

2023-04-08

Primary reference

http://rcrastinate.rbind.io/post/how-to-save-and-load-data-in-r-an-overview/

See also

https://statisticsglobe.com/r-save-load-rdata-workspace-file https://youtu.be/svgEpRzhG7M

What I will show you

In this post, I want to show you a few ways how you can save your datasets in R. Maybe, this seems like a dumb question to you. But after giving quite a few R courses mainly - but not only - for R beginners, I came to acknowledge that the answer to this question is not obvious and the different possibilites can be confusing. In this post, I want to give an overview over the different alternatives and also state my opinion which way is the best in which situation.

What are we going to do?

I will show you the following ways of saving or exporting your data from R: - Saving it as an R object with the functions save() and saveRDS() - Saving it as a CSV file with write.table() or fwrite() - Exporting it to an Excel file with WriteXLS()

For me, these options cover at least 90% of the stuff I have to do at work. So I hope that it'll work for you, too.

Preparation: Load some data

I will use some fairly (but not very) large dataset from the car package. The dataset is called MplsStops and holds information about stops made by the Minneapolis Police Department in 2017. Of course, you can access this dataset by installing and loading the car package and typing MplsStops. However, I want to simulate a more typical workflow here. Namely, loading a dataset from your disk (I will load it over the WWW). The dataset is also available from GitHub:

```
data <- read.table("https://vincentarelbundock.github.io/Rdatasets/csv/carData/MplsStops.csv",
                   sep = ",", header = T,
                   row.names = 1)

kableExtra::scroll_box(knitr::kable(head(data), row.names = F),
           width = "100%", height = "300px")
```

```
## Warning in !is.null(rmarkdown::metadata$output) && rmarkdown::metadata$output
## %in% : 'length(x) = 2 > 1' in coercion to 'logical(1)'
```

idNum	date	problem	MDC	citationIssued	personSearch	vehicleSearch	preRace	race
17-000003	2017-01-01 00:00:42	suspicious	MDC	NA	NO	NO	Unknown	Unkn
17-000007	2017-01-01 00:03:07	suspicious	MDC	NA	NO	NO	Unknown	Unkn
17-000073	2017-01-01 00:23:15	traffic	MDC	NA	NO	NO	Unknown	Whit
17-000092	2017-01-01 00:33:48	suspicious	MDC	NA	NO	NO	Unknown	East
17-000098	2017-01-01 00:37:58	traffic	MDC	NA	NO	NO	Unknown	Whit
17-000111	2017-01-01 00:46:48	traffic	MDC	NA	NO	NO	Unknown	East

We now have a dataset with over 50,000 rows (you can scroll through the first 6 of them in the box above) and 14 variables in our global environment (the 'workspace'). ust for the sake of simulating a real workflow, I will do some very light data manipulation. Here, I'm assigning a new column data*gender.not.knownwhichisTRUEwheneverdata*gender is "Unknown" or NA.

```
data$gender.not.known <- is.na(data$gender) | data$gender == "Unknown"
```

As I wrote above: Saving the current state of your dataset in R makes sense when all the preparations take a lot of time. If they don't, you can just run your pre-processing code every time you are getting back to analyzing the dataset. In the scope of this post, let's suppose that the calculation above took veeeery long and you absolutely don't want to run it everytime.

Option 1: Save as an R object

Whenever I'm the only one working on a project or everybody else is also using R, I like to save my datasets as R objects. Basically, it's just saving a variable/object (or several of them) in a file on your disk. There are two ways of doing this: 1. Use the function save() to create an .Rdata file. In these files, you can store several variables. 2. Use the function saveRDS() to create an .Rds file. You can only store one variable in it.

Option 1.1: save()

You can save your data simply by doing the following:

```
save(data, file = "data.Rdata")
```

By default, the parameter compress of the save() function is turned on. That means that the resulting file will use less space on your disk. However, if it is a really huge dataset, it could take longer to load it later because R first has to extract the file again. So, if you want to save space, then leave it as it is. If you want to save time, add a parameter compress = F.#

(First remove the file from the Global Environment to demonstrate that you are, in fact, reloading it)

```
rm(data)
```

If you want to load such an .Rdata file into your environment, simply do

```
load(file = "data.Rdata")
```

Then, the object is available in your workspace with its old name. Here, the new variable will also have the name data. With save() You can also save several objects in one file. Let's duplicate data to simulate this.

```
data2 <- data

save(list = c("data","data2"), file = "data.Rdata")
```

Now, if you do load("data.Rdata"), you will have two more objects in your workspace, namely data and data2.

```
rm(data,data2)

load(file = "data.Rdata")
```

Option 1.2: saveRDS()

This is the second option of saving R objects. saveRDS() can only be used to save one object in one file. The "loading function" for saveRDS() is readRDS(). Let's try it out.

```
saveRDS(data, file = "data.Rds")
data.copy <- readRDS(file = "data.Rds")
```

The difference between save() and saveRDS()

So, you might ask "why should I use saveRDS() instead of save()"? Actually, I like saveRDS() better - for one specific reason that you might not have noticed in the calls above. When we use load(), we do not assign the result of the loading process to a variable because the original names of the objects are used. But this also means that you have to "remember" the names of the previously used objects when using load().

When we use readRDS(), we have to assign the result of the reading process to a variable. This might mean more typing but it also has the advantage that you can choose a new name for the variable to integrate it in into the rest of the new script more smoothly. Also, it is more similar to the behavior of all the other "reading functions" like read.table(): for these, you also have to assign the result to a variable. The only advantage of save() really is that you can save several objects into one file - but in the end it might be better to have one file for one object. This might be more clearly organized.

(The rest of the blog entry is about saving csv and excel files. I tend to use write.csv for the former. I do not use the WriteXLS package as often because I prefer to output to csv, but I have used WriteLXS and it is good to be aware of.)