CSV to SAV Instructions

Dina Sinclair

January 16, 2018

If you want to convert a file from CSV (a format you can get from the server) to SAV (a format you can load into SPSS), this document will help you get started.

Overarching Code Format

When you change a file format from csv to sav in R, your code will follow the following basic steps:

1. Import the data into R from your csv file
2. Clean the data. This might mean fixing the variable names or changing the type (numeric, string, factor, etc) of desired columns.
3. Export the data into an sav file.

The following is an example piece of code. Lines starting with '#' are comments, meaning that R ignores them.

```r
# STEP ONE: read (import) the data into R. Here we use the function read.csv, and the
# first entry '18 01 08 donnee.csv' is the name of the csv file we want to use.
# More on the import step in a later section.
d <- read.csv("18 01 08 donnee.csv", na.strings = "---", check.names=FALSE)

# STEP TWO: clean the data before saving it as an sav file. Here that means shortening
# the variable names and making sure the number column of the data is saved as an integer.
# More on the cleaning step in a later section.
names(d) <- gsub(".*\\.", "",names(d))
names(d) <- make.names(names(d))
names(d) <- gsub("\\.\\.\\.",'_',names(d))
d$Number <- as.integer(d$Number)

# STEP THREE: export (save) the data as an sav file using the write_sav function from
# the haven library.
# More on the export step in a later section.
library(haven)
write_sav(d,"exported_donnee.sav")
```

Reading/Importing the Data

To successfully read the data from a csv, there are three important questions to ask.

1. How are pieces of data separated in this csv file? By commas (the default) or by some other means (semicolons, spaces, etc)?
2. How are NAs represented in this csv file?
3. Do I want R to keep the original variable names, or can it fix the variable names so that they are readible in R?

Data Entry Separation

To start to answer these questions, a good first step is to import the data into R using the read_csv command, then look at the first six lines of data using the head command.

```
ex1 <- read.csv("Example.csv")
head(ex1)
```

```
##   ex.pays ex.interviewer ex.temps ex.registrer ex.nombre_denfants
## 1    Mali            ---       EB         0.82                  1
## 2    Mali        adoptee       EF         0.56                  2
## 3      GB    participant       EF         0.44                ---
## 4    Mali         simple       EB         0.55                  5
## 5     SEN        adoptee       EF         0.29                ---
## 6     SEN    participant       EF         0.45                  2
```

Above, the result of the head command gives you the correct number of columns and the columns are filled with data as expected. Great - the file likely uses the comma to separate data values. If you open "Example.csv" in a text editor, you will indeed see that all of the entries are separated by commas.

But what happens if the data is separated by a different character? Below, we see one such example.

```
ex2 <- read.csv("Example - Semicolons.csv")
head(ex2)
```

```
##   ex.pays.ex.interviewer.ex.temps.ex.registrer.ex.nombre_denfants
## 1                                                 Mali;---;EB;0.82;1
## 2                                             Mali;adoptee;EF;0.56;2
## 3                                           GB;participant;EF;0.44;---
## 4                                              Mali;simple;EB;0.55;5
## 5                                            SEN;adoptee;EF;0.29;---
## 6                                          SEN;participant;EF;0.45;2
```

If you see the data only loads into one column or row (or maybe no data shows up at all), open the csv file in a text editor. How are the data entries separated? If a character other than a comma is used, we'll need to tell R so that it knows how to read the data properly. If you open up "Example - Semicolons.csv" in a text editor, you'll see that it's separated by semicolons. We can use the argument 'sep' in the read.csv function to tell R that we need to use semicolons to separate data entries, then again use head to look at the data and see if the problem has been fixed.

```
ex3 <- read.csv("Example - Semicolons.csv", sep = ';')
head(ex3)
```

```
##   ex.pays ex.interviewer ex.temps ex.registrer ex.nombre_denfants
## 1    Mali            ---       EB         0.82                  1
## 2    Mali        adoptee       EF         0.56                  2
## 3      GB    participant       EF         0.44                ---
## 4    Mali         simple       EB         0.55                  5
## 5     SEN        adoptee       EF         0.29                ---
## 6     SEN    participant       EF         0.45                  2
```

Sure enough, now the data loads in correctly.

Non Applicables

After loading in the data using the correct data entry separation, we also need to check that R has identified the NA entries in the data. The default way to write NA in R is 'NA', and if your data uses a different set of characters to represent NA entries, you need to tell R to look for that different set of characters. Find an NA element in the head entry of your dataset. Do you see it represented as the text NA, or something else?

```r
ex4 <- read.csv("Example.csv")
head(ex4)
```

```
##   ex.pays ex.interviewer ex.temps ex.registrer ex.nombre_denfants
## 1    Mali            ---       EB         0.82                  1
## 2    Mali        adoptee       EF         0.56                  2
## 3      GB    participant       EF         0.44                ---
## 4    Mali         simple       EB         0.55                  5
## 5     SEN        adoptee       EF         0.29                ---
## 6     SEN    participant       EF         0.45                  2
```

In this example, we see that the NA elements are represented as '—'. Since R doesn't know that '—' is NA, it will see '—' as a string, and therefore read any columns with '—' as strings or factors, even if the rest of the elements in the column are numeric. To fix this, we can use the na.strings argument in the read.csv function.

```r
ex5 <- read.csv("Example.csv", na.strings = '---')
head(ex5)
```

```
##   ex.pays ex.interviewer ex.temps ex.registrer ex.nombre_denfants
## 1    Mali           <NA>       EB         0.82                  1
## 2    Mali        adoptee       EF         0.56                  2
## 3      GB    participant       EF         0.44                 NA
## 4    Mali         simple       EB         0.55                  5
## 5     SEN        adoptee       EF         0.29                 NA
## 6     SEN    participant       EF         0.45                  2
```

Now, the NA entries show up as NA, like we want.

Keeping or Fixing Variable Names

Sometimes, the variable names (column headers) in the csv file will use characters R can't use as variable names. Example characters that are invalid in R variable names are -,*,$,+ and spaces. This is because these characters represent operations in r. For example, if you had variables a, b and a-b, how would R know if a-b means the variable 'a-b' or the variable 'a' minus the variable 'b'?

If you don't tell R to keep the original variable names, it will fix all of the variable names by changing all the characters R can't use to periods. Sometimes, though, you'll want to keep the original variable names (we'll talk about when in the cleaning data section). If you want to keep the original variable names, you can do so using the check.names argument in the read.csv function.

```r
ex6 <- read.csv("Example.csv", check.names = FALSE)
head(ex6)
```

```
##   ex-pays ex-interviewer ex-temps ex-registrer ex-nombre_denfants
## 1    Mali            ---       EB         0.82                  1
## 2    Mali        adoptee       EF         0.56                  2
## 3      GB    participant       EF         0.44                ---
## 4    Mali         simple       EB         0.55                  5
## 5     SEN        adoptee       EF         0.29                ---
## 6     SEN    participant       EF         0.45                  2
```

Here, we can see that variable names that used to show up as 'ex.pays' and ex.interviewer' now are 'ex-pays' and 'ex-interviewer', as they were originally in the csv file.

Cleaning the Data

Changing Variable Names

Changing Column Types