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.

```
# 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?

```
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.

```
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.

```
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

Before exporting to SPSS, we often want to make the data easier to use. This might mean changing the variable names or the type (numerics, string, factor, etc) of a column of data.

Changing Variable Names

To change the variable names, we first need to know what the variable names are. We can figure that out using the command names().

```
ex7 <- read.csv("Example.csv")
names(ex7)
```

```
## [1] "ex.pays"            "ex.interviewer"     "ex.temps"
## [4] "ex.registrer"       "ex.nombre_denfants"
```

Read through the names and decide: are you okay with the variable names, or would you like to reformat them? Reformatting them might mean shortening them, changing them from upper to lower case, or any other string manipulation you can come up with. If you're happy with the variable names the way they are, great - you can skip this section! If you're not happy with the variable names, ask yourself: can I write down clear instructions on how to change the names of these variables? Imagine you have to give these instructions to a stranger, along with the list of variable names. If they can use the instructions to change all of the variable correctly, then the instructions are good ones.

Examples of good (clear) instructions:

- Remove all text up to and including the last period, keep everything to the right of the last period.
- Change all letters to lowercase and change all '. . . ' to '_'

If in this case we might decide to remove all text up to and including the last period. To do so, we can use the gsub() function.

```
ex8 <- read.csv("Example.csv")
names(ex8) <- gsub('.*\\.','',names(ex8))
names(ex8)
```

```
## [1] "pays"            "interviewer"     "temps"           "registrer"
## [5] "nombre_denfants"
```

Note that we don't just call gsub, we assign the result of the gsub to names(ex8) to update the variable names. Make sure to update the variable names by assigning a new value to names() every time you want to make a change, otherwise your work won't be saved!

If you can't see any patterns in your variable names that will let you think of good change instructions, it might be easier to look at the original variable names instead (remember, unless you tell R otherwise, it replaces characters it can't read in variable names to periods). To do so, use the check.names option we mentioned in the importing section:

```
ex9 <- read.csv("Example.csv", check.names = FALSE)
names(ex9)
```

```
## [1] "ex-pays"            "ex-interviewer"     "ex-temps"
## [4] "ex-registrer"       "ex-nombre_denfants"
```

If you can see a good rule or set of instructions to use now, great. If not, R may not be able to help you, since you can't tell R what to do if you don't have instructions to give it!

Here, we might decide to remove everything in the variable names up through the last dash '-', or to simply remove all versions of the phrase 'ex-'. Either works just fine, and the code for both are below:

```
# Removing everything up through the last dash
 gsub('.*-','',names(ex9))
```

```
## [1] "pays"            "interviewer"     "temps"           "registrer"
## [5] "nombre_denfants"
```

```
# Removing all instances of the phrase 'ex-'
gsub('ex-','',names(ex9))
```

```
## [1] "pays"            "interviewer"     "temps"           "registrer"
## [5] "nombre_denfants"
```

Note that these are examples of us trying out code, but they haven't assigned the output to names(), so none of the work is saved. If we see that they both work, we can pick either one to use. Let's say we pick the first way. Then the code we would need to write to save the results of our gsub to the variable names is

```
names(ex9) <- gsub('.*-','',names(ex9))
```

If you're using the original variable names, after manipulating them you need to make sure R can read them. To do that, we use the make.names() function. If you forget this step, you might get errors in your R code or weird looking variables (v1, v2, etc) in your SPSS file.

```
names(ex9) <- make.names(ex9)
```

You can read more about the gsub function here and more about string manipulations in general here. There are a huge variety of commands you can use, but a summary of key commands likely to come up for Tostan data is

```
ex10 <- read.csv("Example - Names.csv", check.names = FALSE)
# The original names
names(ex10)
```

```
## [1] "EXAMPLE-DATA.COUNTRY"     "EXAMPLE-DATA.INTERVIEWER"
## [3] "EXAMPLE-DATA.PERIOD"
```

```
# Removing all characters up through the last period (here, \\. represents a period)
gsub('.*\\.','',names(ex10))
```

```
## [1] "COUNTRY"     "INTERVIEWER" "PERIOD"
```

```
# Removing all characters up through the last dash
gsub('.*-','',names(ex10))
```

```
## [1] "DATA.COUNTRY"     "DATA.INTERVIEWER" "DATA.PERIOD"
```

```
# Removing a specific set of characters
gsub('EXAMPLE-DATA.','',names(ex10))
```

```
## [1] "COUNTRY"     "INTERVIEWER" "PERIOD"
```

```
# Changing a set of characters to another set of characters
gsub('EXAMPLE-DATA','data', names(ex10))
```

```
## [1] "data.COUNTRY"     "data.INTERVIEWER" "data.PERIOD"
```

```
# Changing the words to lowercase (to change to uppercase, use toupper())
tolower(names(ex10))
```

```
## [1] "example-data.country"     "example-data.interviewer"
## [3] "example-data.period"
```

```
# Make the names readible by R
make.names(names(ex10))
```

```
## [1] "EXAMPLE.DATA.COUNTRY"     "EXAMPLE.DATA.INTERVIEWER"
## [3] "EXAMPLE.DATA.PERIOD"
```

You can also do several of these commands in a row. Note that order here matters! And remember to assign the new value to names() every time you use a command. For example, the following are the same string manipulation commands used at the example script at the top of this file.

```
# Read in the data, using the original variable names
ex11 <- read.csv("18 01 08 donnee.csv", check.names=FALSE)
# Look at the first 15 variable names
head(names(ex11),15)
```

```
##  [1] "Number"
##  [2] "Pays"
##  [3] "localization.departement"
##  [4] "localization.commune"
##  [5] "localization.village"
##  [6] "type_de_communaut"
##  [7] "type_evaluation"
##  [8] "interviewer"
##  [9] "ethnie.nbre_groupe_ethnique"
## [10] "demog_enquete.CE1_caracteristiques_enquete.CE2_sexe"
## [11] "demog_enquete.CE1_caracteristiques_enquete.CE3_groupe_ethnique"
## [12] "demog_enquete.CE1_caracteristiques_enquete.CE4_age_enquete.CE4x1_connait_age"
## [13] "demog_enquete.CE1_caracteristiques_enquete.CE4_age_enquete.CE4x1a_age_exact"
## [14] "demog_enquete.CE1_caracteristiques_enquete.CE4_age_enquete.CE4x1b_tranche_age"
## [15] "demog_enquete.CE1_caracteristiques_enquete.CE5_etat_matrimonial"
```

```
# Remove the text up through the last period
names(ex11) <- gsub(".*\\.", "",names(ex11))
# Change the variable names so that R can read them (this will create some '...')
names(ex11) <- make.names(names(ex11))
# Change instances of '...' to '_' to increase readability
names(ex11) <- gsub("\\.\\.\\.","_',names(ex11))
# Look at the first 15 resulting variable names
head(names(ex11),15)
```

```
##  [1] "Number"               "Pays"                 "departement"
##  [4] "commune"              "village"              "type_de_communaut"
##  [7] "type_evaluation"      "interviewer"          "nbre_groupe_ethnique"
## [10] "CE2_sexe"             "CE3_groupe_ethnique"  "CE4x1_connait_age"
## [13] "CE4x1a_age_exact"     "CE4x1b_tranche_age"   "CE5_etat_matrimonial"
```

Changing Column Types

Sometimes, R might not guess the type (integer, numeric, string, factor, etc) of each column correctly. You can fix that before you export to SPSS. If you notice in you SPSS file that a column isn't the right type, you can change it using the commands

```
ex12 <- read.csv("18 01 08 donnee.csv", na.strings = '---', check.names=FALSE)
# Convert to integer
ex12$Number <- as.integer(ex12$Number)
# Convert to factor
```

```
ex12$localization.commune <- as.factor(ex12$localization.commune)
# Convert to decimal
ex12$ethnie.nbre_groupe_ethnique <- as.numeric(ex12$ethnie.nbre_groupe_ethnique)
# Convert to character
ex12$interviewer <- as.character(ex12$interviewer)
```

Writing/Exporting the Data

Once your data is in a format you're happy with, you can export it to an sav file, which you can open in SPSS. To do so, we need to use a library (also known as a package) called haven. If you've never installed haven before, go ahead and click on Tools>Install Packages, type the word 'haven' into the packages bar, click install, and wait until the installation is complete. If you've already installed haven, or think you might have, go ahead and move on to the next step - if it turns out haven isn't installed, you'll get an error that reads

Error in library(haven) : there is no package called 'haven'

in which case you should go install haven using the instructions above.

Once haven is installed, we need to load it in a file if we want to use it. To load haven, we use the command library(haven). After that command, we can use the write_sav() file from the haven library to made an sav file, with the name of the data and new filename as arguments. The file will then appear in the same directory/folder as wherever the file executing the write_sav() command is saved.

```
library(haven)
ex13 <- read.csv("18 01 08 donnee.csv", na.strings = '---', check.names=FALSE)
write_sav(ex13,"new_sav_file.sav")
```

If the write_sav() command executes without errors, you should be able to open your new sav file in SPSS now! You may get an error or two the first time you run your file, and that's okay, since there are ways to fix those errors. Here are some of the most likely errors you'll encounter:

Error: SPSS only supports levels with <= 120 characters Problems: `Number`

This means that R thinks that the Number column should be a factor, but there are over 120 different items in that column which is too many options for a factor in SPSS. Number should actually be an integer, so to fix this, you can use the as.integer function on the problematic column

```
library(haven)
ex13 <- read.csv("18 01 08 donnee.csv", na.strings = '---', check.names=FALSE)
ex13$Number <- as.integer(ex13$Number)
write_sav(ex13,"new_sav_file.sav")
```

like we saw in the data cleaning section. If the column with the error should actually be a string, you can change it to

```
ex13$Number <- as.character(ex13$Number)
```

instead.