Loading data: more about csv-files

Alla Tambovtseva

Working with different separators

As we already discussed, by default, in csv-files R takes a comma as a columns separator:

```
df1 <- read.csv("example1.csv")
head(df1)
```

```
##    A B C
## 1  4 7 8
## 2 10 0 1
## 3  2 6 9
## 4  5 5 3
```

However, sometimes a separator might be different. For instance, this usually happens when we save data as a csv-file via Excel or open a downloaded csv-file via Excel. If R return an error while loading a file (something about incorrect number of columns), it might serve as a sign of a different columns separator. We can specify it with the option sep:

```
df2 <- read.csv("example2.csv", sep=";")
head(df2)
```

```
##    A B C
## 1  4 7 8
## 2 10 0 1
## 3  2 6 9
## 4  5 5 3
```

An incorrect separator does not necessarily lead to errors that stop code execution. A file can be loaded correctly, but it will not look as expected:

```
df3 <- read.csv("example2.csv")
head(df3)
```

```
##    A.B.C
## 1  4;7;8
## 2 10;0;1
## 3  2;6;9
## 4  5;5;3
```

Since R takes a comma as a separator by default, a semi-colon is not recognized as a column separator, so we get one large and strange column instead of multiple ones.

Another problem can occur if a decimal separator in a data set is different from a point. If decimal fractions contain commas instead of points, such columns will be regarded as character (or factor), not numeric variables:

```
df4 <- read.csv("example3.csv")
head(df4)
```

```
##    A   B   C
## 1  4   7 8,6
## 2 10   0 1,2
## 3  2 6,9   9
```

```
## 4  5   5   3
```

```
str(df4)
```

```
## 'data.frame':    4 obs. of  3 variables:
##  $ A: int  4 10 2 5
##  $ B: Factor w/ 4 levels "0","5","6,9",..: 4 1 3 2
##  $ C: Factor w/ 4 levels "1,2","3","8,6",..: 3 1 4 2
```

How to handle this problem? Just specify a decimal separator by adding the option dec:

```
df5 <- read.csv("example3.csv", dec=",")
str(df5) # now C is numeric
```

```
## 'data.frame':    4 obs. of  3 variables:
##  $ A: int  4 10 2 5
##  $ B: num  7 0 6.9 5
##  $ C: num  8.6 1.2 9 3
```

Handling missing values that are not NA's

One more useful option is na.strings. It can be helpful if missing values in a file are encoded differently from NA. For example, in example5.csv missing values are represented by cells with one space inside. Thus, we can tell R that values with `' '` should be regarded as NAs:

```
# compare
df6 <- read.csv("example4.csv")
df7 <- read.csv("example4.csv", na.strings = "' '")
```

```
# look at B type and number of non-null values
str(df6)
```

```
## 'data.frame':    4 obs. of  3 variables:
##  $ A: int  4 10 2 5
##  $ B: Factor w/ 4 levels "' '","5","6.9",..: 4 1 3 2
##  $ C: Factor w/ 4 levels "' '","2","8.6",..: 3 2 4 1
```

```
str(df7)
```

```
## 'data.frame':    4 obs. of  3 variables:
##  $ A: int  4 10 2 5
##  $ B: num  7 NA 6.9 5
##  $ C: num  8.6 2 9 NA
```

```
print(sum(complete.cases(df6)))
```

```
## [1] 4
```

```
print(sum(complete.cases(df7)))
```

```
## [1] 2
```

As it can be seen, in df7 loaded with option na.strings cells with spaces are treated as missing values (NAs).

Characters vs factors in files

Usually R reads text variables in files as factor ones. In most cases it does not cause any obstactles since character variables often contain qualitative information that can be coded with numbers (for example, 1 for

"male" and 0 for "female"). However, if it is important to handle character columns as text ones, it is better to keep their format. This might be particularly useful if we have region names that we plan to change somehow (search patterns and replace parts of texts) or if we have invalid text cells we want to delete so as to freely covert the rest of values to numeric. Let's see the illustration for the second case.

A researcher, which rarely works with numbers, instead of leaving cells with no data blank, filled them with the text "no data".

```
data <- read.csv("houses.csv")
head(data)
```

```
##   id    year floors
## 1  1    1989     12
## 2  2    1902      2
## 3  3    2015     30
## 4  4 no data      5
```

```
str(data)
```

```
## 'data.frame':    4 obs. of  3 variables:
##  $ id    : int  1 2 3 4
##  $ year  : Factor w/ 4 levels "1902","1989",..: 2 1 3 4
##  $ floors: int  12 2 30 5
```

If we remove such values from the column year, and convert the rest valid values to numeric, we will be surprised:

```
data <- data[data$year!="no data", ] # filter rows
data$year <- as.integer(data$year) # convert the rest
head(data) # years have changed!
```

```
##   id year floors
## 1  1    2     12
## 2  2    1      2
## 3  3    3     30
```

This occurs because R automatically treated year as factors. R assigned text values its own numeric labels and then kept them, so after type conversion we got namely these labels in text format, not years! To avoid such problems, we can force R to load text variables as character ones (later we can change some of them to factors manually if needed).

```
data <- read.csv("houses.csv", stringsAsFactors = FALSE)
str(data)
```

```
## 'data.frame':    4 obs. of  3 variables:
##  $ id    : int  1 2 3 4
##  $ year  : chr  "1989" "1902" "2015" "no data"
##  $ floors: int  12 2 30 5
```

```
data <- data[data$year!="no data", ] # filter rows
data$year <- as.integer(data$year) # convert the rest
head(data) # years unchanged
```

```
##   id year floors
## 1  1 1989     12
## 2  2 1902      2
## 3  3 2015     30
```

Now everything is correct.

Working with encodings

If we work with English texts, usually there are no problem with file encodings since latin letters look the same in different encodings on different platforms. However, when we have non-latin characters, we can have difficulties. For instance, if a file with cyrillics was created on Windows, it might look strange on Mac/Linux and vice versa. This is because a file on Windows was saved with Windows-1251 or CP-1251 encoding (or other, these are most commmon) and the default encoding on Mac/Linux is UTF-8. To cope with this problem, we should add an option encoding while uploading files:

```
df_cyr <- read.csv("with_cyr.csv", encoding = "Windows-1251")
head(df_cyr) # now it is ok
```

```
##   X                region          region_trans
## 1 1 Архангельская область Arkhangelskaya oblast
## 2 2    Республика Якутия   Respublika Yakutiya
## 3 3   Краснодарский край     Krasnodarsky krai
```

That's all for this overview!