Data I/O

January 16, 2019

We can write a dataset to disk in comma-separated format using the `write.csv` function. Using our previous "states" example…

```
write.csv(states, file="states.csv")
```

Note the named argument to indicate the filename. We could have specified any other folder/directory by passing a path. In the previous call, the file will be written to our current working directory. We can see which one it is with:

```
getwd()
```

```
## [1] "H:/teaching/intro_to_R/src"
```

and we can use the `setwd()` to set it.

Unsurprisingly, we can read our dataset back using `read.csv`.

```
states <- read.csv("states.csv")
states
```

What about other delimiters and even formats? For other delimiters, we can use the more general function `read.table` and `write.table` that allows us to specify which delimiter we want to use. Actually, `read.csv` is just `read.table` with a predefined delimiter

```
states <- read.table("states.csv", sep=",")
```

and we could have, for instance, defined that our input is tab separated by specifying

```
states <- read.table("states.csv", sep="\t")
```

The most common formats for R uses the extensions `.RData` (or `RDS`), using the functions `save` (`saveRDS`) and `load` (`readRDS`):

```
save(states, file="states.RData")
load("states.RData")
states

saveRDS(states, file = "states.RDS")
states <- readRDS("states.RDS")
```

RDS is generally safer to use than RData because `readRDS` is functional, whereas `load` attaches the data to the calling environment (by default), i.e., `load` uses non-local assignment, which can have unintended consequences.

But R can also read (and sometimes write) data in other binary formats: data coming from Stata, SAS, SPSS, or even Excel. The functions to handle these foreign formats are provided in the `foreign` package that comes with `R` but a much better alternative is the `haven` package which gives us access to functions like `read_sas` to read SAS sas7bdat files or `read_stata` to read Stata dta files.

This package is not provided with R but we need instead to install it and loaded, which gives us a good opportunity to look at importing new functionality into R.

We first need to install the library using:

```
install.packages("haven")
```

The function will hit a CRAN mirror, download the file for the package that we want, and perform the installation routine (which includes a number of checks). Now the package is available in our system, we can load it into our session:

```
library(haven)
```

For instance, we can now load an *uncompressed* SAS file using

```
path <- system.file("examples", "iris.sas7bdat", package="haven")
read_sas(path)
```

```
## # A tibble: 150 x 5
##    Sepal_Length Sepal_Width Petal_Length Petal_Width Species
##           <dbl>       <dbl>        <dbl>       <dbl> <chr>
##  1          5.1         3.5          1.4         0.2 setosa
##  2          4.9         3            1.4         0.2 setosa
##  3          4.7         3.2          1.3         0.2 setosa
##  4          4.6         3.1          1.5         0.2 setosa
##  5          5           3.6          1.4         0.2 setosa
##  6          5.4         3.9          1.7         0.4 setosa
##  7          4.6         3.4          1.4         0.3 setosa
##  8          5           3.4          1.5         0.2 setosa
##  9          4.4         2.9          1.4         0.2 setosa
## 10          4.9         3.1          1.5         0.1 setosa
## # ... with 140 more rows
```

or we could read a Stata file using

```
path <- system.file("examples", "iris.dta", package="haven")
read_stata(path)
```

```
## # A tibble: 150 x 5
##    sepallength sepalwidth petallength petalwidth species
##          <dbl>      <dbl>       <dbl>      <dbl> <chr>
##  1        5.10       3.5         1.40      0.200 setosa
##  2        4.90       3           1.40      0.200 setosa
##  3        4.70       3.20        1.30      0.200 setosa
##  4        4.60       3.10        1.5       0.200 setosa
##  5        5          3.60        1.40      0.200 setosa
##  6        5.40       3.90        1.70      0.400 setosa
##  7        4.60       3.40        1.40      0.300 setosa
##  8        5          3.40        1.5       0.200 setosa
##  9        4.40       2.90        1.40      0.200 setosa
## 10        4.90       3.10        1.5       0.100 setosa
## # ... with 140 more rows
```

As a matter of fact, one of the good things about `R` is that it interacts nicely with many other programs. For instance, it can read Excel spreadsheets in several different ways. One option is `readxl` library, which does not require Java. So first, like before, we need to install the package and load it:

```
install.packages("readxl")
library(readxl)
```

Now we will be able to call a function like `read_excel`

```
read_excel(path)
```

```
## # A tibble: 3 x 3
##   B3    C3    D3
##   <chr> <chr> <chr>
## 1 B4    C4    D4
## 2 B5    C5    D5
## 3 B6    C6    D6
```

We can also query data from databases. There are a number of packages for this, but the one I prefer is RODBC (some database systems may be incompatible and require a different package).

```
library(RODBC)

db_str <- sprintf("driver={SQL Server};server=%s;database=%s;uid=%s;pwd=%s",
                   server, database, username, password)
db_handle <- odbcDriverConnect(db_str)

data <- sqlQuery(dbhandle, query_str)

odbcClose(db_handle)
```

Each of these function has a number of options (Do we want to use a catalog file for SAS? In which sheet is the data in our Excel file? How to deal with user generated missing values in SPSS?) and of course we can interact with many other formats (feather for data exchange with Python, fixed width files, hd5, . . .), streams (for real-time analysis), . . . but this is sufficient for now.