This R Data Import Tutorial Is Everything You Need

July 21st, 2015 in R Programming

Karlijn Willems

You might find that loading data into R can be quite frustrating. Almost every single type of file that you want to get into R seems to require its own function, and even then you might get lost in the functions' arguments. In short, you might agree that it can be fairly easy to mix up things from time to time, whether you are a beginner or a more advanced R user.

To cover these needs, DataCamp decided to publish a comprehensive, yet easy tutorial to quickly importing data into R, going from simple text files to the more advanced SPSS and SAS files. Keep on reading to find out how to easily import your files into R!

(Try this interactive course: Importing Data in R (Part 1), to work with CSV and Excel files in R.)

Contents

Checking Your Data

To import data into R, you first need to have data. This data can be saved in a file onto your computer in an Excel, SPSS, or some other type of file. When your data is saved locally, you can go back to it later to edit, to add more data or to change them, preserving the formulas that you maybe used to calculate the data, etc.

However, data can also be found on the Internet or can be obtained through other sources.

Where to find these data are out of the scope of this tutorial, so for now it's enough to mention this list of data sets, and DataCamp's interactive tutorial, which deals with how to import and manipulate Quandl data sets.

Before you move on and discover how to load your data into R, it might be useful to go over the following checklist that will make it easier to import the data correctly into R:

- If you work with spreadsheets, the first row is usually reserved for the header, while the first column is used to identify the sampling unit;
- Avoid names, values or fields with blank spaces, otherwise each word will be interpreted as a separate variable, resulting in errors that are related to the number of elements per line in your data set;
- If you want to concatenate words, inserting a . in between to words instead of a space;
- Short names are prefered over longer names;
- Try to avoid using names that contain symbols such as `?`, `$`, `%`, `^`, `&`, `*`, `(`, `)`, `-`, `#`, `?`, `,`, `<`, `>`, `/`, `|`, `\`, `[`, `]`, `{`, and `}`;
- Delete any comments that you have made in your Excel file to avoid extra columns or NA's to be added to your file; and
- Make sure that any missing values in your data set are indicated with `NA`.

Preparing Your R Workspace

Make sure to go into RStudio and see what needs to be done before you start your work there. You might have an environment that is still filled with data and values, which you can all delete using the following line of code:

```
rm(list=ls())
```

The `rm()` function allows you to "remove objects from a specified environment". In this case, you specify that you want to consider a list for this function, which is the outcome of the `ls()` function. This last function returns you a vector of character strings that gives the names of the objects in the specified environment. Since this function has no argument, it is assumed that you mean the data sets and functions that you as a user have defined.

Next, you might also find it handy to know where your working directory is set at the moment:

script.R R Console

```
1 getwd()
```

And you might consider changing the path that you get as a result of this function, maybe to the folder in which you have stored your data set:

```
setwd("<location of your dataset>")
```

Read CSV, TXT, HTML, and Other Common Files into R

You will see that the following basic R functions focus on getting spreadsheets into R, rather than Excel or other type of files. If you are more interested in the latter, scroll a bit further to discover the ways of importing other files into R.

Read TXT files with read.table()

If you have a `.txt` or a tab-delimited text file, you can easily import it with the basic R function `read.table()`. In other words, the contents of your file will look similar to this

```
1  6  a
2  7  b
3  8  c
4  9  d
5  10 e
```

and can be imported as follows:

script.R R Console

```
df <- read.table("https://s3.amazonaws.com/assets.datacamp.com/blog_assets/test.txt",
                 header = FALSE)
df
```

Run

Note that ideally, you should just pass in the file name and the extension because you have set your working directory to the folder in which your data set is located. You'll have seen in the code chunk above that the first argument isn't always a filename, but could possibly also be a webpage that contains data. The `header` argument specifies whether or not you have specified column names in your data file. Lastly, you'll see that, by using this function, your data from the file will become a `data.frame` object.

Inspect the final result of your importing in the DataCamp Light chunk!

It's good to know that the `read.table()` function is the most important and commonly used function to import simple data files into R. It is easy and flexible. That is why you should check out our previous tutorial on reading and importing Excel

These variants are almost identical to the `read.table()` function and differ from it in three a

- The separator symbol;
- The `header` argument is always set at TRUE, which indicates that the first line of the file being read contains the header with the variable names;
- The `fill` argument is also set as TRUE, which means that if rows have unequal length, blank fields will be added implicitly.

Read CSV Files into R

If you have a file that separates the values with a `,` or `;`, you usually are dealing with a `.csv` file. Its contents looks somewhat like this:

```
Col1,Col2,Col3
1,2,3
4,5,6
7,8,9
a,b,c
```

Make sure that you have saved the file as a regular csv file without a Byte Order Mark (BOM). If you have done this, you'll see weird characters appearing at the start of your imported data if you don't add the extra argument `fileEncoding = "UTF-8-BOM"` to your importing function!

To successfully load this file into R, you can use the `read.table()` function in which you specify the separator character, or you can use the `read.csv()` or `read.csv2()` functions. The former function is used if the separator is a `,`, the latter if `;` is used to separate the values in your data file.

Remember that the `read.csv()` as well as the `read.csv2()` function are almost identical to the `read.table()` function, with the sole difference that they have the `header` and `fill` arguments set as `TRUE` by default.

script.R R Console

```
# Read in csv files
df <- read.table("https://s3.amazonaws.com/assets.datacamp.com/blog_assets/test.csv",
                 header = FALSE,
                 sep = ",")

df <- read.csv("https://s3.amazonaws.com/assets.datacamp.com/blog_assets/test.csv",
               header = FALSE)

df <- read.csv2("https://s3.amazonaws.com/assets.datacamp.com/blog_assets/test.csv",
               header= FALSE)

# Inspect the result
df
```

Run

`read.table()`, `read.csv()` or `read.csv2()` functions.

Note that if you get a warning message that reads like "incomplete final line found by reac and "stand" on the cell that contains the last value (`c` in this case) and press ENTER. Th because the message indicates that the last line of the file doesn't end with an End Of Line (EOL) character, which can be a linefeed or a carriage return and linefeed. Don't forget to save the file to make sure that your changes are saved!

Pro-Tip: use a text editor like NotePad to make sure that you add an EOL character without adding new rows or columns to your data.

Also note that if you have initialized other cells than the ones that your data contains, you'll see some rows or columns with `NA` values appear. The best case is then to remove those rows and columns!

read.delim() for Delimited Files

In case you have a file with a separator character that is different from a tab, a comma or a semicolon, you can always use the `read.delim()` and `read.delim2()` functions. These are variants of the `read.table()` function, just like the `read.csv()` function.

Consequently, they have much in common with the `read.table()` function, except for the fact that they assume that the first line that is being read in is a header with the attribute names, while they use a tab as a separator instead of a whitespace, comma or semicolon. They also have the `fill` argument set to `TRUE`, which means that blank field will be added to rows of unequal length.

You can use the `read.delim()` and `read.delim2()` functions as follows:

script.R R Console

```
# Read a delimited file
df <- read.delim("https://s3.amazonaws.com/assets.datacamp.com/blog_assets/test_delim.txt", sep="$")
df <- read.delim2("https://s3.amazonaws.com/assets.datacamp.com/blog_assets/test_delim.txt", sep="$")

# Inspect the result
df
```

Run

XLConnect Package for Reading Excel Files

To load Excel files into R, you first need to do some further prepping of your workspace in the sense that you need to install packages.

Simply run the following piece of code to accomplish this:

```
install.packages("<name of the package>")
```

When you have installed the package, you can just type in the following to activate it in your workspace:

script.R R Console

```
# Fill in a package name
library("<name of the package>")
```

Importing Excel Files With The XLConnect Package

The first way to get Excel files directly into R is by using the `XLConnect` package. Install the package and if you're not sure whether or not you already have it, check if it is already there.

Next, you can start using the `readWorksheetFromFile()` function, just like shown here below:

```
library(XLConnect)
df <- readWorksheetFromFile("<file name and extension>",
                            sheet = 1)
```

Note that you need to add the `sheet` argument to specify which sheet you want to load into R. You can also add more specifications. You can find these explained in our tutorial on reading and importing Excel files into R.

You can also load in a whole workbook with the `loadWorkbook()` function, to then read in worksheets that you desire to appear as data frames in R through `readWorksheet()` :

```
wb <- loadWorkbook("<name and extension of your file>")
df <- readWorksheet(wb,
                    sheet=1)
```

Note again that the `sheet` argument is not the only argument that you can use in `readWorkSheetFromFile()` . If you want more information about the package or about all the arguments that you can pass to the `readWorkSheetFromFile()` function or to the two alternative functions that were mentioned, you can visit the package's RDocumentation page.

Importing Excel Files With The Readxl Package

The `readxl` package has only recently been published and allows R users to easily read in Excel files, just like this:

```
library(readxl)
df <- read_excel("<name and extension of your file>")
```

Note that the first argument specifies the path to your `.xls` or `.xlsx` file, which you can set by using the `getwd()` and `setwd()` functions. You can also add a `sheet` argument, just like with the XLConnect package, and many more arguments on which you can read up here or in this blog post.

Read JSON Files Into R

To get JSON files into R, you first need to install or load the rjson package. If you want to know how to install packages or how to check if packages are already installed, scroll a bit up to the section of importing Excel files into R :)

Once you have done this, you can use the `fromJSON()` function. Here, you have two options:

1. Your JSON file is stored in your working directory:

```
# Import data from json file
JsonData <- fromJSON(file= "<filename.json>" )
```

2. Your JSON file is available through a URL:

```
# Activate `rjson`
library(rjson)

# Import data from json file
JsonData <- fromJSON(file= "<URL to your JSON file>" )
```

Read XML Data Into R

If you want to get XML data into R, one of the easiest ways is through the usage of the XML package. First, you make sure you install and load the XML package in your workspace, just like demonstrated above. Then, you can use the `xmlTreeParse()` function to parse the XML file directly from the web:

```
# Activate the `XML` library
library(XML)

# Parse the XML file
xmlfile <- xmlTreeParse("<Your URL to the XML data>")
```

Next, you can check whether R knows that `xmlfile` is in XML by entering:

```
# Result is usually similar to this: [1] "XMLDocument"         "XMLAbstractDocument"
class(xmlfile)
```

Tip: you can use the `xmlRoot()` function to access the top node:

```
topxml <- xmlRoot(xmlfile)
```

You will see that the data is presented kind of weirdly when you try printing out the `xmlfile` vector. That is because the XML file is still a real XML document in R at this point. To put the data in a data frame, you first need to extract the XML values. You can use the `xmlSApply()` function to do this:

```
topxml <- xmlSApply(topxml,
                    function(x) xmlSApply(x, xmlValue))
```

The first argument of this function will be `topxml`, since it is the top node on whose children you want to perform a certain function. Then, you list the function that you want to apply to each child node. In this case, you want to extract the contents of a leaf XML node. This, in combination with the first argument `topxml`, will make sure that you will do this for each leaf XML node.

Lastly, you put the values in a dataframe!

You use the `data.frame()` function in combination with the matrix transpostition function `t()` to do this. Additionally you also specify that no row names should be included:

```
xml_df <- data.frame(t(topxml),
                     row.names=NULL)
```

```
url <- <a URL with XML data>
data_df <- xmlToDataFrame(url)
```


Importing Data From HTML Tables Into R

From HTML tables into R is pretty straightforward:

```
# Assign your URL to `url`
url <- "<a URL>"

# Read the HTML table
data_df <- readHTMLTable(url,
                         which=3)
```

Note that the `which` argument allows you to specify which tables to return from within the document.

If this gives you an error in the nature of “failed to load external entity”, don’t be confused: this error has been signaled by many people and has been confirmed by the package’s author here.

You can work around this by using the `RCurl` package in combination with the `XML` package to read in your data:

```
# Activate the libraries
library(XML)
library(RCurl)

# Assign your URL to `url`
url <- "YourURL"

# Get the data
urldata <- getURL(url)

# Read the HTML table
data <- readHTMLTable(urldata,
                      stringsAsFactors = FALSE)
```

Note that you don’t want the strings to be registered as factors or categorical variables! You can also use the httr package to accomplish exactly the same thing, except for the fact that you will want to convert the raw objects of the URL’s content to characters by using the `rawToChar` argument:

```
# Activate `httr`
library(httr)

# Get the URL data
urldata <- GET(url)

# Read the HTML table
data <- readHTMLTable(rawToChar(urldata$content),
                      stringsAsFactors = FALSE)
```

Read SAS, SPSS, and Other Datasets into R

As you will know already, R is a programming language and software environment for statistical computing. That's why it probably won't come as a surprise when I say that many people use R as an open-source alternative to commercial statistical programs, such as SPSS, SAS, etc.

In this section, you'll see how you can import data from advanced statistical software programs: you'll see which packages that you need to install to read your data files into R, just like you have done with data that was stored in Excel or JSON files.

Read SPSS Files into R

If you're a user of SPSS software and you are looking to import your SPSS files into R, firstly install the foreign package. After loading the package, run the `read.spss()` function that is contained within it and you should be good to go!

```
# Activate the `foreign` library
library(foreign)

# Read the SPSS data
mySPSSData <- read.spss("example.sav")
```

Tip if you wish the result to be displayed in a data frame, make sure to set the `to.data.frame` argument of the `read.spss()` function to `TRUE`. Furthermore, if you do NOT want the variables with value labels to be converted into R factors with corresponding levels, you should set the `use.value.labels` argument to `FALSE`:

```
# Activate the `foreign` library
library(foreign)

# Read the SPSS data
mySPSSData <- read.spss("example.sav",
                        to.data.frame=TRUE,
                        use.value.labels=FALSE)
```

Remember that factors are variables that can only contain a limited number of different values. As such, they are often called "categorical variables". The different values of factors can be labeled and are therefore often called "value labels"

Read Stata Files into R

To import Stata files, you keep on using the `foreign` package. Use the `read.dta()` function to get your data into R:

```
# Activate the `foreign` library
library(foreign)

# Read Stata data into R
mydata <- read.dta("<Path to file>")
```

```
# Activate the `foreign` library
library(foreign)

# Read Systat data
mydata <- read.systat("<Path to file>")
```

Read SAS Files into R

For those R users that also want to import SAS file into R, it's very simple! For starters, install the `sas7bdat` package. Load it, and then invoke the `read.sas7bdat()` function contained within the package and you are good to go!

```
# Activate the `sas7bdat` library
library(sas7bdat)

# Read in the SAS data
mySASData <- read.sas7bdat("example.sas7bdat")
```

Does this function interest you and do you want to know more? Visit the Rdocumentation page.

Note that you can also use the `foreign` library to load in SAS data in R. In such cases, you'll start from a SAS Permanent Dataset or a SAS XPORT Format Library with the `read.ssd()` and `read.xport()` functions, respectively. For more information, click here.

Read Minitab Files into R

Is your software of choice for statistical purposes Minitab? Look no further if you want to use Minitab data in R!

Importing `.mtp` files into R is pretty straightforward. It won't come as a surprise any more that you'll need to install the foreign package and load it. Then simply use the `read.mtp()` function from that package:

```
# Activate the `foreign` library
library(foreign)

# Read the Minitab data
myMTPData <- read.mtp("example2.mtp")
```

Read RDA or RData Files into R

If your data file is one that you have saved in R as an `.rdata` file, you can read it in as follows:

```
load("<FileName>.RDA")
```

Read Databases and Other Sources Into R

Since this tutorial focuses on importing data from different types of sources, it is only right to also briefly mention that you can import data into R that comes from databases, webscraping, etc.

Read Relational and Non-Relational Databases into R

Importing Data From Relational Databases

If, however, you want to load data from **MySQL** into R, you can follow this tutorial, which u
the data into R.

If you are interested in knowing more about this last package, make sure to check out Da
which is definitely a must for everyone that wants to use dplyr to access data stored outside of R in a database. Furthermore, the course also teaches you how to perform sophisticated data manipulation tasks using dplyr!

Importing Data From Non-Relational Databases

For more information on loading data from non-relational databases into R, like data from **MongoDB**, you can read this blogpost from "Yet Another Blog in Statistical Computing" for an overview on how to load data from MongoDB into R.

Importing Data Through Webscraping

You can read up on how to scrape JavaScript data with R with the use of PhantomJS and the `rvest` package in this DataCamp tutorial. If you want to use APIs to import your data, you can easily find one here.

Tip: you can check out this set of amazing tutorials which deal with the basics of webscraping.

Importing Data Through The TM Package

For those of you who are interested in importing textual data to start mining texts, you can read in the text file in the following way after having installed and activated the `tm` package:

```
text <- readLines("<filePath>")
```

Then, you have to make sure that you load these data as a corpus in order to get started correctly:

```
docs <- Corpus(VectorSource(text))
```

You can find an accessible tutorial on text mining with R here.

This Is Just The Beginning…

Loading your data into R is just a small step in your exciting data analysis, manipulation and visualization journey. DataCamp is here to guide you through it!

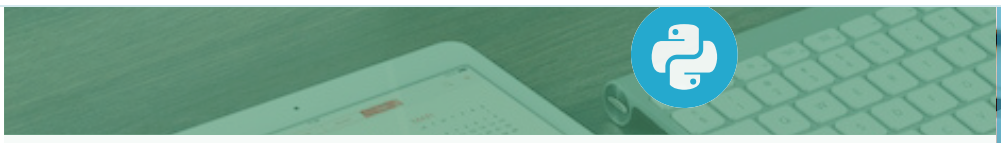
Continue with our Importing Data in R course or go and build models with your data: our machine learning will definitely come in handy, just like our Introduction to Machine Learning.

If you want to continue with data manipulation, go through tutorial on 15 Easy Solutions To Your Data Frame Problems In R or consider taking DataCamp's data.table course.

Are you not sure where to start? Check out DataCamp's course curriculum and discover what lies ahead in your data science journey with DataCamp!

What do you think?

R Programming | Importing & Cleaning Data



Asyncio: An Introduction

Python 1,506 views

A short introduction to asynchronous I/O with the asyncio package.

May 8th, 2017 in Tutorial

Scikit-Learn Tutorial: Baseball Analytics in Python Pt 1

Machine Learning 1,740 views

This scikit-learn tutorial will show you how to predict MLB wins per season by modeling data to KMeans clustering model and linear regress…

May 4th, 2017 in Tutorial

Keras Tutorial: Deep Learning in Python

Machine Learning 9,016 views

This Keras tutorial introduces you to deep learning in Python: learn o preprocess your data, model, evaluate and optimize neural networks.

May 2nd, 2017 in Tutorial

View All