Import and Packages

Irfan Emrah Kanat, Arianna Sammarchi

2023-03-02

Data Import

In this document I will try to walk you through the most common data import functions. Once you learn the basics, you can use the same principles to any type of analysis task at hand. *Make sure you are using the same project as we did in the introduction.*

Through out this document you will see two types of code blocks. First group (exact representation may vary depending on media type) is blue text in light gray boxes. Those are R commands you can type in to the R console. Second group is the black text on white back ground, preceded by ##. Which is the R output of said command. Below is an example.

```
print("Hello Students")
```

```
## [1] "Hello Students"
```

Before Beginning

Make sure you are using the same project as we did in the introduction. Copy the files in the zip file provided to your project directory.

Comma Separated Value Files

This is the most common file format there is to transfer data, with good reason I might add. It is human readable (not binary), and is compatible with almost any system out there. The basic idea is, the columns are separated by commas and observations are separated by new lines. I would recommend you use this format unless you have compelling reason to do otherwise.

R uses read.csv function to import this file type. Let us learn more about this function.

```
help(read.csv)
```

As you can see, the function takes quite a few parameters but most are optional with reasonable defaults (meaning you can safely leave them blank).

You are provided with a file named “country.csv” in the git repository. If you placed this file in your project directory you can import its contents as follows.

```
# Save file contents into a variable called csv
csv <- read.csv("country.csv")
# view the contents
csv
```

To view the contents of this new variable in a spreadsheet format, double click the csv in Environment pane of R studio (Area B), or write the command below.

Before running the command below,

install XQuartz

```
#View(csv)
# we utilize View function to display results in R Studio.
#However it is commented here because of incompatibility of documents and R knitting.
```

You can also explore the file contents via R functions.

```
str(csv) # Structure
```

```
## 'data.frame':    29 obs. of  3 variables:
##  $ ip_iso2: chr  "AU" "TR" "US" "GB" ...
##  $ Supply : int  20 80 100 50 70 40 20 10 50 30 ...
##  $ Those  : int  0 1 0 0 0 0 1 1 1 1 ...
```

```
summary(csv) # Descriptive Statistics
```

```
##    ip_iso2              Supply          Those       
##  Length:29          Min.   :  0.00   Min.   :0.0000  
##  Class :character   1st Qu.: 25.00   1st Qu.:0.0000  
##  Mode  :character   Median : 50.00   Median :0.0000  
##                     Mean   : 47.52   Mean   :0.4828  
##                     3rd Qu.: 64.00   3rd Qu.:1.0000  
##                     Max.   :100.00   Max.   :1.0000  
```

Excel Files

This section will be optional and not covered in the workshop as the functioning of the packages connected to the importation of Excel files relies on other technologies and their install may cause the arise of issues. However, for those interested in this part, it is possible to try it out by employing the currently commented lines of codes as well by using the country.xlsx file that can be found in the repository.

First method

This file format is very common due to the popularity of Microsoft's office suite.

To import this file type we need two additional steps, installing and loading packages of which we will learn quite a bit more later. Note that, by using this method, Java issues might arise as it is required to be installed to allow the machine to run these commands; therefore, the incoming part is commented and left to the reader.

```
# Below is commented out, since I do not want R to install package every time you run this.
# Uncomment the line the first time and run.
#install.packages("xlsx") # Install the xlsx package that enables XLSX import function.
#library(xlsx) # Enable the package
```

Now let us read the xlsx file into R. Note that I had to specify which worksheet to import.

```
#xlsx <- read.xlsx("country.xlsx", sheetIndex = 1) # Import the file
#xlsx # view contents
```

The package has other functionality that you can discover on your own.

Excel Files

Second method - Import Dataset

This method enables uploading xlxs files without incurring in the above mentioned issue. It consists of the following steps: - Going in the Environment" view, - Clicking on the Import Dataset icon, - Selecting "From Excel" and browsing the required Excel file.

Databases

R can connect to a variety of databases. For this workshop we currently do not have access to a database to demonstrate this functionality. Below is an example for MySQL.

```
#library(RMySQL) # Load the necessary package
#channel <- dbConnect(MySQL(), user = 'uname', password = 'pwd',
                      #host = '127.0.01', dbname='exampledata') # Establish database connection
#sql <- dbGetQuery(channel, "SELECT * FROM table;") # Execute query over connection, save results
```

Other Statistical Packages

R enables importing data from other statistical packages as well. I would recommend exporting the data from other packages in a csv and then importing it that way to minimize incompatibilities. If you have no access to the offending statistical package, then you can easily use one of the packages listed below.

I won't go into details as this may well turn out to be a fringe case not useful to all. For the purposes of this workshop, I will only discuss the packages and functions used and not actually import any data. You can use the principles used in earlier steps to import data on your own.

SAS

Package used for sas data format after version 7 is valled 'sas7bdat'. The function used is called read.sas7bdat().

If you have exported the data from SAS, you can use 'foreign' package with read.xport() function.

STATA

Stata data format has changed after version 13. If you wish to import a file of this sort use 'readstata13' package with read.dta13() function.

Earlier versions can be imported with 'foreign' package's read.dta() function.

SPSS

SPSS sav files can be imported with 'foreign' package's read.spss() function.

Exported por files can be imported with 'Hmisc' package's spss.get() function.

Packages

We have already encountered packages quite a few times so far. In introduction we discussed the packages and the opportunities they provided. In importing we have used them to read in data in various formats. Yet, we have so far avoided a serious discussion of what packages are and how we can use them.

Packages are a collection of functions, documentation, and in some cases data files. These are similar to libraries in other programing languages. **When loaded into memory**, they provide specific functionality, be it analysis, visualization, or reporting.

R being open source enabled a great number of packages serving any function imaginable, making it one of the the most popular tools for analytics. If a feature is missing, you can easily develop your own functions and bundle them into a package for others to use.

In this document we will go over the basics of package management in R.

Basics

Before getting into the details of package management, let us learn how to obtain and activate a package.

If the package you need is not already installed in your system, you can easily obtain it from CRAN via install.packages() function. Below you can see the command to install the plm package, a set of functions to carry out traditional econometric analysis.

```
install.packages("plm")  # Notice the quotes around package name
```

Pretty soon the package will be installed. Installing a package does not mean it is ready to use. You also have to activate it. For this purpose you can use the library() command.

```
library(plm) # Notice the lack of quotes around the package name
```

You most probably won't need this, still it is good to know. To de-activate a package, we use the detach() function. The parameters of this command is a bit more involved than the first two.

```
detach("package:plm", unload=TRUE) # Notice the package: prepended to the package name
```

Details

If you want to get a list of all the packages that are currently loaded in the memory (activated) use the .packages() command.

```
(.packages()) # Notice the parantheses
```

```
## [1] "stats"     "graphics"  "grDevices" "utils"     "datasets"  "methods"
## [7] "base"
```

If you want to get a list of all packages installed in the system use the parameter. I am omitting the output since I have about 150 packages installed.

```
.packages(all.available = T)
```

How to find out about packages?

If you want to find packages to carry out the tasks you have at hand the first place to start would be either Comprehensive R Archive Network (CRAN) or (for a better interface) R Forge. R site search (also available with command RSiteSearch()) provides a great search engine for documentation. R seek is a great search engine dedicated to scanning R related sources. Of course you can always use google as well but the letter R is a bit ambiguous in some cases.

R frequently offers a number of packages that serves similar purposes (lme4 and nlme for mixed models for example). There are subtle differences between the packages and picking the right tool for the task can save you a lot of time. I often find myself using documentation from cran pages of the packages before I settle for the one I want to use. I frequently spend more time browsing documentation, than on actual analysis. A careful reading of the documentation will allow you to better understand the options available to you.

CRAN is not only a repository for the code, but also for the documentation. Here is the plm package page for example. Beyond reference manuals and other documentation, R packages often come with vignettes and tutorials. These are often like quick start guides to analysis.

If you want to search a certain word in installed packages' documentation, you can always use ?? or help.search()

```
??mixed
help.search("mixed model")
```

Commonly Used Packages

Data Manipulation

data.table This package is a replacement of the data.frame data structure. It is not a drop-in replacement and is not fully compatible with all data.frame functionality. So you need to be careful when you use it.

It replaces data.frame with a data structure that is more efficient. Meaning it is faster when accessing and writing into data.tables.

It also comes with additional functionality that makes subsetting/aggregating a breeze.

dplyr This is a package that adds additional functions to efficiently manipulate data (subsetting/aggregate, etc.). It still uses data.frames so you do not have compatibility issues in your code.

Another benefit is, it can directly work with data stored in databases. So you can select, filter data from a database and use it in R.

Statistics

R base package (that comes loaded) includes your basic statistics such as multiple regressions and Generalized Linear Models.

Panel Data Econometrics Your traditional fixed effects / random effects econometric models are covered in plm package.

Hierarchical/Mixed Effects Models Two packages, nlme and lme4 handle this bit. The development on nlme has slowed down a bit, yet it incorporates functionality to model autoregressive correlation structures, weighting to take care of heteroskedasticity and so on. Both packages have unique strengths, you need to read the documentation to see which one fits you best.

Machine Learning

caret package provides a nice set of functions for almost all predictive analytics needs. It is discussed in deeper detail in Example Analysis.

Classifiers (KNN, LVQ) K Nearest Neighbor matching and Learning Vector Quantization and more are available through class package.

Support Vector Machines kernlab, e1071 packages are two among many packages that provide this functionality.

Clustering Base package has functionality to enable basic clustering techniques like K means (kmeans()), or hierarchical (hclust()) clustering.

For model based clustering there is the mclust package.

Neural Networks neuralnet package as well as others provide NN functionality.
