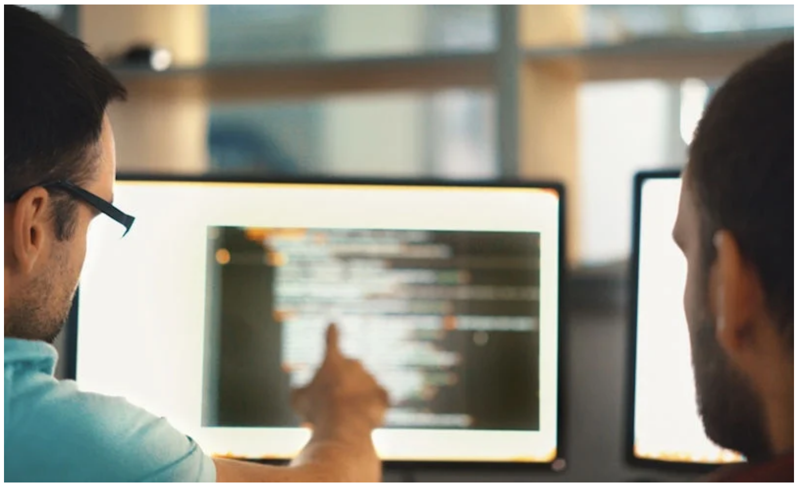
ImpoRting and ExpoRting: Getting Data Into and Out of R

Colby Ford

In any organization, data is housed in many locations and in many formats. Nowadays, many business analytics tools have the native ability to import from almost any data source imaginable. For example, Microsoft Power BI has the ability to get data from over 60 different sources, including many common file types and more esoteric data storage software.



From flat files to databases to proprietary file formats, we'll take a look at how simple it is to use these data sources in R. Many times, as you'll see, all it takes is the installation and use of a simple package. This includes many proprietary formats such as SAS and Microsoft Excel. The steps below will take you through each format and help you to master importing and exporting data!

Reading and Writing Common Flat Files

Base R

R comes equipped with the ability to read and write many common text-based flat files including `.csv` files, `.tsv` files, and more. For example, let's say I have a file called `mydata.csv` in my data folder. R has a function, `read.table`, and its children functions, `read.csv`, and `read.delim`.

```
#read.table works in the following manner: "read.table(file, header = FALSE, sep = "", quote = "\"'",...".
csvinput <- read.table("data/mydata.csv",header = TRUE, sep = ",",quote = "\"")

#Alternatively, using "read.csv"" will assume the separator is a comma, which may save some time.
#We used the quote parameter to tell R that there are some strings in the file surrounded by double quotes.
```

Now, let's say that you have done some data manipulation and calculations and you want to export your data back to your data folder. This is just as simple as reading things in.

```
#write.table works in the following manner: "write.table(x, file = "", col.names = TRUE, append = FALSE, quote = TRUE, sep = " ",...".
#Alternatively, using "write.csv"" will assume the separator is a comma, which may save some time.
#We use the append parameter to tell R to overwrite the file rather than just tack on rows to the end of the old version.
write.table(input,file="data/output.csv",col.names = TRUE, append = FALSE, quote = TRUE,  sep = ",")
```

[read.table documentation]

readr

In the realm of tidyverse, there is a package known as `readr` that can handle importing flat files, too. This package has multiple functions to handle different flat file types. The `readr` package runs a bit faster than the base R functions and handles factors and dates a bit better, too. It also supports seven file formats with seven `read_` functions:

- `read_csv()`: comma-separated (CSV) files
- `read_tsv()`: tab-separated files
- `read_delim()`: general delimited files
- `read_fwf()`: fixed-width files
- `read_table()`: tabular files where columns are separated by white-space
- `read_log()`: web log files

```
#install.packages("tidyverse") or #install.packages("readr")
library(readr)
#All of the readr functions work in this way: readr_csv("filename").

input <- read_csv("mydata.csv")
#readr assumes the filetype and delimiter by the function you choose.
#For example, when you use readr_tsv, the package assumes it's looking for a .tsv file and that the delimiter is a "\t".
```

[readr package documentation]

Excel-ling with Importing and Exporting

Similar to how we just imported a `.csv`, we can use almost identical functions to the previous section to import Microsoft `.xls` and `.xlsx` Excel files. There are many packages in the R universe that can do this. Here, we'll take a look at two: `openxlsx` and `readxl`. First, you'll need to install one of the packages. After it's installed, it is practically the same syntax as before.

openxlsx

```
#install.packages("openxlsx")
library(openxlsx)
#read.xlsx works in the following manner: "read.xlsx(xlsxFile, sheet = 1, startRow = 1, colNames = TRUE, detectDates = FALSE,...)".
input <- read.xlsx("data/mydata.xlsx")

#You can use the detectDates parameter to tell R that there are dates in the file and to convert them to date strings rather than numbers.
```

After you work with the imported data, writing back to Excel files is also possible in the same manner as before.

```
library(openxlsx)
#write.xlsx works in the following manner: "write.xlsx(x, file, asTable = FALSE,...)".
write.xlsx(input,"data/output.xlsx")
```

[openxlsx package documentation]

readxl

```
#install.packages("readxl")
library(readxl)
#readxl works the same way as the openxlsx package: read_excel("filename").
input <- read_excel("data/mydata.xlsx")
```

[realxl package documentation]

But What About My SAS Files?

Fear not! There are plenty of packages in the R universe to read in your SAS `.sasb7dat` files and other statistical software files as well. To start, the `sasb7dat` package will do the trick.

sasb7dat

```
#install.packages("sas7bdat")
library(sas7bdat)
#read.sas7bdat works in the following manner: "read.sas7bdat(filepath)".
input <-  read.sas7bdat("data/mydata.sas7bdat")
```

[sas7bdat package documentation]

haven

The tidyverse realm has a package to open SAS, SPSS, and Stata data as well. This package is known as `haven`.

```
#install.packages("haven")
library(haven)
#read_sas works in the following manner: read_sas("filepath");.
input <-  read_sas("data/mydata.sas7bdat")
```

The `haven` package can also write SAS files:

```
library(haven)
write_sas(input, "output.sas7bdat")
```

[haven package documentation]

Remember that the `haven` package works on SAS, SPSS, and Stata. So, this may be easier than installing different packages for each different statistical software's dataset. Currently, `haven` supports:

- SAS: `read_sas()` reads `.sas7bdat` plus `.sas7bcat` files and `read_xpt()` SAS transport files (version 5 and version 8). `write_sas()` writes `.sas7bdat` files.
- SPSS: `read_sav()` reads `.sav` files and `read_por()` reads the older `.por` files. `write_sav()` writes `.sav` files.
- Stata: `read_dta()` reads `.dta` files (up to version 14). `write_dta()` writes `.dta` files (versions 8-14).

Data in a Database

RODBC

Using the package `RODBC`, we can connect to a SQL-based database (such as Microsoft SQL Server) and run queries against it.

```
#install.packages("RODBC")
library(RODBC)

#odbcDriverConnect works in the following manner: "odbcDriverConnect("driver={DB Type}; server=servernamer; database=databasename; trusted_connection=true", uid = "username", pwd = "password")"
connection <- odbcDriverConnect("driver={SQL Server}; server=mysqlserver; database=finance; trusted_connection=true", uid = "admin", pwd = "password1234")

#sqlQuery works in the following manner: "sqlQuery(odbcDriverConnect String, query)"
input <- sqlQuery(connection, "select * from mydatatable")
```

This also works with Azure SQL Server, as well as Microsoft SQL Server on an Azure Virtual Machine. You just have to open each server up to external connections. This can work with Oracle, MySQL, and other databases as long as your machine has the correct ODBC drivers installed.

You can even write data back to a database. This comes in handy when you've changed or added to data (especially in predictive analytics). Instead of extracting the data out of R and then loading it into the database, you can do it from within the R console.

```
#install.packages("RODBC")
library(RODBC)

connection <- odbcDriverConnect("driver={SQL Server};server=mysqlserver;database=finance;trusted_connection=true", uid = "admin", pwd = "password1234")
#sqlwrite works in the following manner: "sqlwrite(odbcDriverConnect String, dataframe, tablename="table")"
sqlwrite(connection, input, tablename="outputtable")
```

[RODBC package documentation]

Data Online

Many public datasets live on the web. There's no need to download these to your local machine when you can just import them directly in R. This is especially useful when you need to reshape or manipulate data before it's useful to you. So, you can just import the data from the web, do whatever you need to do with it, and then write the useful data to your local machine.

RCurl

Using the `RCurl` package, we can connect directly to data using its web address. You will just use the `url` function in combination with any other `read.*` function listed earlier.

```
#install.packages("RCurl")
library(RCurl)

input <- read.csv(url("https://www.mywebsite.com/data/mydata.csv"))
```

[RCurl package documentation]

Getting Help

Some apprehension to using R comes from users not understanding how to get data into and out of the system. As you can see, importing and exporting are far from difficult. With just a few, simple (and notably very similar) commands, you can pull in data from any source you like.

There are thousands of packages available for R, many of which can do very similar functions to what have been shown here. These are some of the most popular, but each package has its strength. You can search through the CRAN package list for access to many more packages.

After you install any of the packages referenced here, you can view the documentation by adding a `?` before the package name or function name, as shown below.

```
#install.packages("haven")
library(haven)
#Get help on the entire haven package or list out the functions in the package.
?haven
#Get help on the syntax and options of the read_sas function.
?read_sas
```

Still looking for assistance? BlueGranite can help! We can work with you and your team to help you learn more about R through our instructor-led training featuring data science lessons and topics using Microsoft R Server for enterprise-class, big data analytics. Contact us today for more information.