Importing CSV files into R

As part of the WIPO Manual on Open Source Patent Analytics project we will be working with patent data in R using [RStudio](http://www.rstudio.com). If you do not have a copy of RStudio follow the simple instructions for installing on your platform [here](http://www.rstudio.com/products/rstudio/download/). There are lots of resources on the site to help you get started including [online learning](http://www.rstudio.com/resources/training/online-learning/), [videos](http://www.rstudio.com/resources/webinars/), and [cheatsheets](http://www.rstudio.com/resources/cheatsheets/). The excellent [R-Bloggers site](http://www.r-bloggers.com) will demonstrate why it is worth investing time in R when working with patent data.

Comma separated value files (or .csv) files are one of the most common and useful ways for sharing data. This includes patent data.

This walk through covers the basics of importing .csv files into R and writing .csv files. We will use the freely available [ritonavir](https://drive.google.com/open?id=0B4piiKOCkRPDRlBlcGpxR0tMTms&authuser=0) patent dataset as the example. You can grab the datasets by either forking or downloading the [GitHub repository](https://github.com/poldham/opensource-patent-analytics) or downloading the [zip file](https://github.com/poldham/opensource-patent-analytics/blob/master/2_datasets/datasets.zip?raw=true). While we use patent data as the example, this will work for other types of .csv data.

We will cover the following approaches to importing and writing .csv files here:

1. Importing .csv files from local directories using the standard read.table in the utils package.
2. Writing .csv files using write.csv()
3. Importing multiple .csv files using lapply() and ldply()
4. Importing and writing .csv files using the new [readr](http://blog.rstudio.org/2015/04/09/readr-0-1-0/) package.
5. Downloading a .csv file from a URL, focusing on https: connections, using Rcurl.

That should cover most needs. If you find that you are stuck with a function try calling the description for a particular function (?read.csv) or try [stackoverflow.com](http://stackoverflow.com/questions/tagged/r) results tagged R or from a Google search (often the quickest route).

## Reading in a file using read.table (utils package)

Reading in a .csv file is easy and is part of read.table in the R utils package (installed by default). We can simply read in a .csv by creating an object linked to the function read.csv() followed by the path to the local file as follows.

ritonavir <- read.csv("yourfilenamepath.csv")

In some European countries the delimiter in a .csv is a semicolon ";" and not a comma. In the unlikely event you come across these files use read.csv2() as above instead of read.csv.

You now have a dataset called ritonavir in R. That is how easy it is. You can take a look at the data by simply typing ritonavir into the console. What you will see is a mass of data.

We can improve on that by using head(ritonavir) but it is still a little difficult to view. We will come back to this in turning the data into a table data frame (tbl\_df()).

First, let's look at the function read.csv. R functions have settings called arguments that can be used to control what is going on.

?read.csv

read.csv(file, header = TRUE, sep = ",", quote = "\"", dec = ".", fill = TRUE, comment.char = "", ...)

The arguments for this function can be very useful, for example,

header = TRUE or FALSE. Determines whether or not to import column headings from the first row.

sep = "," . The separator for the values in each row.

The ... refers to additional arguments that might be applied. Among the most important of these using this function are:

1. stringsAsFactors = FALSE. To prevent character columns being converted to factors. This is actually a lot more important than it sounds, and will generally be your default.
2. na.strings = "NA". NA refers to not available. In some cases this needs to be expanded to cover blank cells in the source data. for example c("NA", " ", "") captures cells containing "NA", cells with only a space " " or empty cells "".
3. strip.white = TRUE. This will strip leading and trailing white space. Note that this will only work if you have specified sep = in the arguments.
4. skip = n. Specify the number of lines to skip before the data starts. Very useful for data tables with blank rows or text padding at the top of files.

This means that we would often want a read.csv() function with the following additional arguments for our file.

ritonavir1 <- read.csv("yourfilenamepath.csv", sep = ",", na.strings = "NA",   
 strip.white = TRUE, stringsAsFactors = FALSE)

Note here that the use of sep = "," is the condition for stripping leading and trailing white space on import using strip.white = TRUE. strip.white won't work without the sep.

If you intend to split the inventor and applicant data following import you may want to wait because the process will generate white space. It is always possible to write a .csv file after the cleaning process and reimport it with strip.white set to TRUE along with sep= ",". We will write a .csv file below.

We have not specified skip = n in the above as the column headers are in the first row in the original data. But, there are lots of occasions when skip can be useful.

Lets look at the type or class of object that has been created from our latest import.

class(ritonavir1) ##class is "data.frame"

If we print the ritonavir R object we will get the first 500 rows of data.

ritonavir1

That is not terribly helpful because we are overwhelmed with information and can't see everything as a snap shot. The solution to this is to install and load the recent dplyr package. We will be using this package a lot in the patent analysis tutorials so, if you don't have it already, now is a good time.

install.packages("dplyr")

load the package

library(dplyr)

We can now use the tbl\_df() function to create an easy to read dataframe table (tbl\_df()) using our ritonavir1 dataset that lists columns as characters.

ritonavir2 <- tbl\_df(ritonavir1)

This creates an easy to read table dataframe.

If we print the frame we will now have readable content.

ritonavir2

That is a lot easier to read than our original (try typing ritonavir into the console to take a look, then try ritonavir 2).

There are two points to note here.

1. Spaces in column names such as publication number are filled with full stops.
2. More importantly, by default character vectors are converted to factors (characters backed by a hidden number). It therefore makes sense to always use StringsAsFactors = FALSE unless you actually want columns to import as factors.

## Reading a .csv from the web

Reading a .csv file from the web is also easy but can involve some complications. We will cover a couple of cases here. If the URL begins with http: then it is as simple as entering the URL inside quotes. However, if it is the secure https: then it is a little bit more of a challenge.

For example if we try the following it will generally work with http: but not with https:

ritonavir\_url <- read.csv("https://drive.google.com/open?id=0B4piiKOCkRPDTGdSQmRMa1BOUEE&authuser=0")

To deal with this we need to install and load the package RCurl.

install.packages("RCurl")

load the library

library(RCurl)

Now let's try again and add a couple of arguments to make it work.

ritonavir\_url <- download.file("https://drive.google.com/open?id=0B4piiKOCkRPDTGdSQmRMa1BOUEE&authuser=0", "ritonavir\_url.csv", method = "curl")

In this case we use download.file and the URL in quotes, followed by the destination filename (which will download into the current working directory). For this to work without an error we need finally to specify method = "curl" or an error reading "unsupported URL scheme" will appear.

### Downloading from GitHub

In downloading from GitHub (where the project Google Drive datasets are also located), we have to go a step further. The URL that you see on the page in Github is basically a marker for the data location... not the actual dataset location. To access the actual dataset navigate to the relevant page [here](https://github.com/poldham/opensource-patent-analytics/blob/master/datasets/ritonavir/ritonavir.csv). However, then select the Raw button and copy that *URL*. The URL should begin with <https:raw>. as below.

ritonavir\_urlg <- download.file("https://raw.githubusercontent.com/poldham/opensource-patent-analytics/master/datasets/ritonavir/ritonavir.csv", "ritonavir\_urlg.csv", method = "curl")

As an alternative to this approach in Github it can be easier to simply navigate to the repository (such as <https://github.com/poldham/opensource-patent-analytics>) and then select Clone in Desktop (if you are using GitHub on your local machine) or 'Download ZIP'. That will download the repository including the relevant datasets.

## Writing a .csv file

If we wanted to write this data table to a new .csv file we would use the 'write.csv()' command.

Before we do that a critical point to remember is to give the file a new filename or it will overwrite your original data. It is also worth checking your working directory so that you know where it will be saved.

getwd()

If this is the wrong directory, locate and copy the path for the directory you want and then use setwd() to set the directory to save the file in.

To write the file with a new file name we will use write.csv().

write.csv(ritonavir2, "ritonavir2.csv")

This will write a file called ritonavir2.csv to your working directory. If you take a look at the file note that an extra column will have been added at the beginning (we will come back to that) and column names will now contain full stops instead of spaces.

Let's take a look at the options for writing .csv files by calling help.

?write.csv

write.csv is a function in write.table. Let's take a look at the arguments.

write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ", eol = "\n", na = "NA", dec = ".", row.names = TRUE, col.names = TRUE, qmethod = c("escape", "double"), fileEncoding = "")

A couple of these settings may be useful.

1. append = TRUE or FALSE. Do we want to append the data to an existing file of that name or not? If false and the same filename is used then it will overwrite the existing file. If TRUE it will append the data to the file of the same name.
2. na = "NA" the string that you want to use for missing data. This may need further definition depending on your data (e.g. na = c("NA", " ", "")).
3. row.names and col.names may be useful depending on your dataset or needs. Note that the default is TRUE. This is generally correct for columns with patent data but not for rows.

The row.names = FALSE argument is more important than it appears. As Garrett Grolemund points out in [Hands-On Programming with R](http://shop.oreilly.com/product/0636920028574.do) every time that you write a .csv file in R it will add a row names column and keep on writing a new column each time you write the file. No one seems to quite understand why that happens but you will not be wanting that to happen. So the suggested default for write.csv() is

write.csv(ritonavir2, "ritonavir2.csv", row.names = FALSE)

This will prevent R from writing an additional row names column.

We will now take a look at a somewhat unusual import case. That is importing multiple files. I have left this until after we have worked on writing .csv files because realising that we can import, export and then reimport files into R is an important part of creating effective workflows in future work. Otherwise, we may spend hours in R to achieve something that can be done easily outside R. This particular example provides a good illustration of this point.

## Reading in multiple .csv files

On some occasions we might want to read in multiple .csv files at the same time. Typically this will be where a patent dataset has been split into multiple files. If you would like to follow this discussion then download the [pizza\_sliced dataset](https://drive.google.com/folderview?id=0B4piiKOCkRPDVUxhVVhkbGtQLVU&usp=sharing) which contains five .csv files plus a ReadMe file.

Reading in multiple files is a task that is a little trickier than it should be. However, the approach below will work (assuming you have the files in a local folder). It is best to remove the Read Me file from the downloaded set for this exercise.

pizzasliced <- list.files("/Users/pauloldham17inch/Desktop/open\_source\_master/2\_DATASETS/pizza\_sliced", full.names = TRUE) ##create a vector of file names to read in

If we print pizza sliced we will now see a list of the full name of files including the full path. It should look something like that below.

pizzasliced

## [1] "/Users/pauloldham17inch/Desktop/open\_source\_master/2\_DATASETS/pizza\_sliced/pizza\_sliced\_20002002\_319.csv"  
## [2] "/Users/pauloldham17inch/Desktop/open\_source\_master/2\_DATASETS/pizza\_sliced/pizza\_sliced\_20032005\_366.csv"  
## [3] "/Users/pauloldham17inch/Desktop/open\_source\_master/2\_DATASETS/pizza\_sliced/pizza\_sliced\_20062008\_439.csv"  
## [4] "/Users/pauloldham17inch/Desktop/open\_source\_master/2\_DATASETS/pizza\_sliced/pizza\_sliced\_20092012\_428.csv"  
## [5] "/Users/pauloldham17inch/Desktop/open\_source\_master/2\_DATASETS/pizza\_sliced/pizza\_sliced\_20132014\_274.csv"

If we check the class of this object using class(pizzasliced) it will be a character type

class(pizzasliced)

## [1] "character"

What we now need to do is to transform this into a list. To do that we will use the function lapply() for list apply. In this case we are saying "apply list apply to the R character object pizzasliced and then apply the function read.csv ensuring that headings are included from column names" as below.

pizzasliced1 <- lapply(pizzasliced, read.csv, header = TRUE, stringsAsFactors = FALSE)

lapply will then iterate through and read in the five files to create a new object that we call pizzasliced1. Note that if doing this normally you would probably retain the same name and allow R to overwrite the object as you go along. Here we are showing the steps. If we now investigate the class of R object the answer will be a list.

class(pizzasliced1)

## [1] "list"

A list is a vector that groups together R objects (in this case the data from our files). To demonstrate this if we now try and print pizzasliced1 then we will see an overwhelming amount of information rush by as R prints the five objects in the list.

print(pizzasliced1)

What we now need to do is to convert the list into a data.frame. To do that however we need to turn to the plyr package and the ldplyr function.

install.packages("plyr")

Load the library

library(plyr)

We now simply apply ldplyr to pizzasliced1 to turn the list of objects into a data.frame. We will call that pizzasliced2.

pizzasliced2 <- ldply(pizzasliced1)

If we print pizzasliced2 into the console we will see another set of data rushing by. However, if we check the class we will see that we now have a data.frame object. A data.frame is actually a list of the class data.frame. To test that try typing typeof(pizzasliced2) into the console and the result will be "list").

class(pizzasliced2)

We can take a look at the top of the dataset using head(). This will show rather crunched data and column headings

head(pizzasliced2)

We could also take a look at the bottom of the dataset

tail(pizzasliced2)

If we use summary() we will gain a slightly cleaner view that tries to sum up all of the data.

summary(pizzasliced2)

What is needed is an easier to read form and that is where we can use tbl\_df again from the dplyr package that we used earlier.

pizzasliced3 <- tbl\_df(pizzasliced2)

Houston, we have a problem. That problem is that somewhere in the [esp@cenet](mailto:esp@cenet) data that makes up the pizza\_sliced dataset there is an issue with the character encoding that is preventing the dplyr function from working. Typically this will mean that the character encoding has become messed up in the underlying data. This of course is one of the reasons it is always worth checking the data first (as is suggested in this video [here](https://youtu.be/YYaMEbJW7Qw?list=PLsZOGmKUMi54n8R06U1HmxNywt0bAFays)). Clearly that is OK for small datasets but for large or multi-part datasets is a problem. So, another solution is needed. The problem as it turns out is in the inventor and the applicant fields in the space between the name and the country code entry which looks like this in Excel BIANCHI MARCO‰Ûâ[IT] and merely throws a ??? in Open Office Calc.

Working out how to solve this problem in R could take a while (although string replacement is a good bet). The issue therefore is how to get to a solution as quickly as possible. That solution is to take what we learned above to write the new data.frame to a .csv file, then open the file in Open Office or Excel and use find and replace on the corrupted strings (as is suggested in this video [here](https://youtu.be/YYaMEbJW7Qw?list=PLsZOGmKUMi54n8R06U1HmxNywt0bAFays)).

First check your working directory getwd() so you know where the file will go. Then setwd(pathtoyourdirectory) if that needs to change.

write.csv(pizzasliced2, "pizzasliced2.csv", row.names = FALSE)

Open the file in either Open Office or Excel. Copy the corrupted characters in the Inventors or Applicants columns, select those columns and then find and replace with a space as the replacement (see the video link above). Then save the file writing over the original download. Now let's reimport it.

pizzasliced3 <- read.csv("/Users/pauloldham17inch/Desktop/open\_source\_master/2\_DATASETS/pizzasliced2.csv", sep = ",", na.strings = "NA", strip.white = TRUE, stringsAsFactors = FALSE)

It can be worth printing the top of the file to check that the clean up has worked.

head(pizzasliced3)

## Title  
## 1 PIZZA TRAY  
## 2 COOKING METHOD OF GREEN-TEA ADDED PIZZA  
## 3 METHOD FOR COOKING A PIZZA  
## 4 Pizza preparation and delivery method and unit  
## 5 Method of making laminated pizza crust  
## 6 Container for transporting heated food, particularly pizza and the like  
## Publication.number Publication.date  
## 1 CA93181 (S) 24/08/01  
## 2 KR20010107868 (A) 07/12/01  
## 3 CA2731260 (A1); CA2731260 (C) 14/09/00  
## 4 US2002048624 (A1); US6858243 (B2) 25/04/02  
## 5 US6126977 (A) 03/10/00  
## 6 US2002040862 (A1); US6601758 (B2) 11/04/02  
## Inventor.s.  
## 1   
## 2 YOU YEN SIL [KR]  
## 3 HEDRINGTON JAMES ALAN [US]; DRESSEL BRENT WILLIAM [US]  
## 4 BLANCHET JEAN [FR]; CATHELIN HERVE [FR]; HEBERT CHRISTIAN [FR]; NOUYRIT OLIVIER [FR]  
## 5 BUBAR RONALD O [US]  
## 6 LIZZIO FILIPPO [IT]  
## Applicant.s.  
## 1 SCHWAN S FOOD MFG INC [US]  
## 2 YOU YEN SIL [KR]  
## 3 NAT PRESTO IND [US]  
## 4 NESTEC SA [US]  
## 5 PAULUCCI JENO F [US]  
## 6 TERMOPIZZA SRL [US]  
## International.classification  
## 1   
## 2 A21D13/00  
## 3 A23L1/01; A21B1/00; A47J37/04; A47J37/06  
## 4 A23L1/48; A21B1/00; A21D8/00; A21D13/00; A21D15/02; B60P3/025; B60P3/14; A23L1/00; A21D13/00  
## 5 A21C3/02; A21C11/00; A21D8/00; A21D8/02; A21D13/00; A21D13/08; A21D13/00  
## 6 B65D77/04; B65D81/26; B65D85/36; B65D85/36; B65D5/50  
## Cooperative.Patent.Classification  
## 1   
## 2   
## 3 A47J37/0611; A47J37/043  
## 4 A21D13/007; A21B1/00; A21D8/00; A21D15/02; B60P3/0257; B60P3/14  
## 5 A21D13/0061; A21C3/02; A21C11/004; A21D8/02; A21D13/007  
## 6 B65D77/0433; B65D81/262; B65D81/263; B65D85/36; B65D2585/366; Y10S229/906; Y10S229/902  
## Application.number Date.of.application  
## 1 CANDAT0093181F 0  
## 2 KR20010069326 20011105  
## 3 CA20002731260 20000310  
## 4 US20010982377 20011018  
## 5 US19970968900 19971106  
## 6 US20010963393 20010927  
## Priority.number.s.  
## 1 CANDAT0093181F 00000000  
## 2 KR20010069326 20011105  
## 3 US19990267981 19990312; CA20002363329 20000310  
## 4 EP20000122736 20001018  
## 5 US19970968900 19971106; US19950496894 19950630  
## 6 IT2000TO00900 20000928; IT2001TO00008U 20010119  
## Patents.cited.in.the.search.report  
## 1   
## 2   
## 3   
## 4 US3879564 (A); US4164591 (A); US4919477 (A); US5256432 (A)  
## 5 US628449 (A); US969173 (A); US1174826 (A); US1179294 (A); US1646921 (A); US2089396 (A); US2509035 (A); US2668767 (A); US3143424 (A); US3235390 (A); US3677769 (A); US3845219 (A); US3880069 (A); US4020184 (A); US4205091 (A); US4283424 (A); US4283431 (A); US4308286 (A); US4313961 (A); US4416910 (A); US4463020 (A); US4551337 (A); US4574090 (A); US4626188 (A); US4645673 (A); US4661361 (A); US4696823 (A); US4753813 (A); US4842882 (A); US4907501 (A); US5104669 (A); US5180603 (A); US5182123 (A); US5194273 (A); US5196223 (A); US5268188 (A); US5348751 (A); US5405626 (A); US5417150 (A); US5417996 (A); US5529799 (A); US5560946 (A); DE3704192 (A1); GB2241863 (A)  
## 6 US3026209 (A); US3515331 (A); US4197940 (A); US4441626 (A); US4883195 (A); US5052559 (A); US5385292 (A); US5445286 (A); US5482724 (A); US5588587 (A); US5895698 (A); EP0989067 (A1)  
## Literature.cited.in.the.search.report Patents.cited.during.examination  
## 1   
## 2   
## 3   
## 4   
## 5   
## 6   
## Literature.cited.during.examination Other.patent.citations  
## 1 NA NA  
## 2 NA NA  
## 3 NA NA  
## 4 NA NA  
## 5 NA NA  
## 6 NA NA  
## Other.literature.citations Patents.used.in.opposition  
## 1 NA <NA>  
## 2 NA <NA>  
## 3 NA <NA>  
## 4 NA <NA>  
## 5 NA <NA>  
## 6 NA <NA>  
## Literature.used.in.opposition  
## 1 NA  
## 2 NA  
## 3 NA  
## 4 NA  
## 5 NA  
## 6 NA  
## Patents.cited.by.the.applicant  
## 1   
## 2   
## 3   
## 4 US4361227 (A); US4791861 (A); JPH09299017 (A)  
## 5   
## 6   
## Literature.cited.by.the.applicant International.search.citation  
## 1 NA   
## 2 NA   
## 3 NA   
## 4 NA   
## 5 NA   
## 6 NA   
## International.search.NPL.citation  
## 1   
## 2   
## 3   
## 4   
## 5   
## 6   
## Supplementary.international.search.citation  
## 1 NA  
## 2 NA  
## 3 NA  
## 4 NA  
## 5 NA  
## 6 NA  
## Supplementary.international.search.NPL.citation  
## 1 NA  
## 2 NA  
## 3 NA  
## 4 NA  
## 5 NA  
## 6 NA

The inventors and applicant fields are now free of the encoding problem. We are now in a position to try again with the table data frame using tbl\_df().

pizzasliced4 <- tbl\_df(pizzasliced3)

Let's print pizzasliced4.

pizzasliced4

We can now see the first ten rows of a data frame that contains 1,707 rows and 24 columns. It will still be a little crunched but is much more readable. Also note that most of the columns are character fields (not factors).

Using dplyr we can call up a View of the dataset in a new pop up window. This can be very useful for inspecting datasets.

View(pizzasliced4)

Let's finish off here by writing pizzasliced4 back out to a .csv for practice.

write.csv(pizzasliced4, "pizzasliced4", row.names = FALSE)

While we would probably choose more informative filenames it is good practice to output work before moving on to other tasks or at some point it will be lost.

We have now covered the basics of importing a .csv file into R and then writing a file. We have also covered importing multiple .csv files and a quick work around character encoding problems before exporting a dplyr table data frame.

Some of these steps are not as easy as they could be. To address that we will look at the recent readr package.

## Using the new readr package.

If you don't have readr use the following to install it.

install.packages("readr")

If you do, or to check it has loaded, use:

library(readr)

If you have not done so already, let's install and load dplyr.

install.packages("dplyr")

library(dplyr)

Let's try loading our dataset again using the function read\_csv()

ritonavir3 <- read\_csv("/Users/pauloldham17inch/opensource-patent-analytics/2\_datasets/ritonavir/ritonavir.csv")

This will create a data frame and then display problems in red. The problems can be investigated by typing problems() in the console. We will ignore these in this case. As with read.csv2(), the readr function read\_csv2() will read files with the ";" as the separator.

To see the read\_csv arguments let's call help

?read\_csv

read\_csv(file, col\_names = TRUE, col\_types = NULL, na = "NA", skip = 0, n\_max = -1, progress = interactive())

This tells us that the function will assume that there are column names in the first row. col\_types = NULL tells us that the function will attempt to calculate the column type from the first thirty rows of data. You can however specify the column types as character, double, integer, logical etc.

skip will specify the number of rows to skip as before. n\_max will specify the maximum number of records to read. That can be helpful if the dataset is large and you just want to take a look at some of it to get a sense of the data.

The main advantages of read\_csv over read.csv are:

1. read\_csv does not automatically read in character vectors as factors. This means there is **no need** to specify stringsAsFactors = FALSE as part of the function's arguments. This will be a very great relief to many people as it is one less thing to remember!
2. The problems() prompt advises you that problems may exist with reading the file. You might be able to fix or ignore them.
3. For larger files a progress indicator will display on loading (in interactive mode) where the load is over 5 seconds.
4. Column names are left as is. That means that publication number stays as publication number rather than becoming publication.number and requiring renaming.
5. By default, readr turns imported data into a data.frame, and a table (tbl) and a table dataframe (tbl\_df). You can test this by typing class(ritonavir3) into the console. That means if you are running dplyr then it will automatically show the first ten rows and the column name. That may not sound exciting but it is a lot better than masses of data rushing past.

However, as readr is a new package that is being actively developed there are also some issues. You might want to check out the latest development version is available [here](https://github.com/hadley/readr/blob/master/README.md).

Let's take a look now at ritonavir3 but using the View() function to call up a dataset window.

View(ritonavir3)

If we scroll across then we can see that the date columns in the dataset have been transformed to NA. In some circumstances this is not a problem (remember that we still have the original dataset, what we see here is a data table). In other cases this could be a problem (if we wanted to use this data).

At the time of writing, there does not seem to be a clear way to deal with this issue (but see the development page read.me on precisely this issue). This reflects the difficulty of dealing with dates because they can be ambiguous. We will discuss this elsewhere.

For the moment, let's call ritonavir3 into the console.

ritonavir3

What we see here is the data with column names left as is. We can also see that most of the columns (vectors) are character vectors and have not been transformed into factors meaning **no more** stringsAsFactors = FALSE. The date fields have been recognised as dates, but as we have seen have been transformed to NA (not available) because of the lack of clarity on the kind of date.

We will update this part of the walkthrough as clarity on dealing with dates becomes available with readr.

## Writing a .csv file using write\_csv()

We can easily write a .csv file using the write\_csv() function as follows

write\_csv(ritonavir3, "ritonavir3.csv")

The output from this has the advantage of preserving the column names as is so that "publication number" stays as is and does not become "publication.number".

The full list of arguments for write\_csv() at the moment is

write\_csv(x, path, append = FALSE, col\_names = !append)

append = TRUE will append the table to the existing file. col\_names = TRUE will write column names at the top of the file. Expect more arguments to be added as readr develops.

Bear in mind that readr does not possess the functionality of read.csv or write.csv in read.table. Part of the aim of readr is simplification based on the idea of doing a limited number of things well. Therefore, it is unlikely that readr will ever be as comprehensive as the read.table equivalents in the future. However, readr is likely to become the go to package because of its simplicity for most needs and because it links with the wider family of tidyr, plyr and dplyr packages under development at RStudio to make data wrangling and analysis easier.

## Round Up

In this walkthrough we have covered the fundamentals of reading and writing .csv files in R. This is pretty much the easiest file format to work with for patent data and considerably better than Excel which we will cover next. I, like almost everyone else, would encourage you to start working with .csv files wherever possible for the straightforward reason that they are cleaner and better to share across platforms and programmes than Excel or other proprietary format files.