**Introduction**

Inspired by a recent post on how to import a directory of csv files at once [using purrr and readr](https://www.gerkelab.com/blog/2018/09/import-directory-csv-purrr-readr/) by Garrick, in this post we will try achieving the same using base R with no extra packages, and with data·table, another very popular package and as an added bonus, we will play a bit with benchmarking to see which of the methods is the fastest, including the tidyverse approach in the benchmark.

Let us show how to import all csvs from a folder into a data frame, with nothing but base R

To get the source data, download the zip file from [this link](https://www.gerkelab.com/data/ie-general-referrals-by-hospital.zip) and unzip it into a folder, we will refer to the folder path as data\_dir.

**Quick import of all csvs with base R**

To import all .csv files from the data\_dir directory and place them into a single data frame called result, all we have to do is:

filePaths <- list.files(data\_dir, "\\.csv$", full.names = TRUE)

result <- do.call(rbind, lapply(filePaths, read.csv))

# View part of the result

head(result)

## Month\_Year Hospital\_Name Hospital\_ID

## 1 Aug-15 AMNCH 1049

## 2 Aug-15 AMNCH 1049

## 3 Aug-15 AMNCH 1049

## 4 Aug-15 Bantry General Hospital 704

## 5 Aug-15 Bantry General Hospital 704

## 6 Aug-15 Bantry General Hospital 704

## Hospital\_Department ReferralType TotalReferrals

## 1 Paediatric ENT General Referral 2

## 2 Paediatric Gastroenterology General Referral 4

## 3 Paediatric General Surgery General Referral 4

## 4 Gastroenterology General Referral 12

## 5 General Medicine General Referral 18

## 6 General Surgery General Referral 43

**A quick explanation of the code:**

* list.files – produces a character vector of the names of the files in the named directory, in our case data\_dir. We have also passed a pattern argument "\\.csv$" to make sure we only process files with .csv at the end of the name and full.names = TRUE to get the file path and not just the name.
* read.csv – reads a file in table format and creates a data frame from its content
* lapply(X, FUN, ...)– Gives us a list of data.frames, one for each of the files found by list.files. More generally, it returns a list of the same length as X, each element of which is the result of applying FUN to the corresponding element of X. In our case X is the vector of file names in data\_dir (returned by list.files) and FUN is read.csv, so we are applying read.csv to each of the file paths
* rbind – in our case combines the rows of multiple data frames into one, similarly (even though a bit more rigidly) to UNION in SQL
* do.call – will combine all the data frames produced by lapply into one using rbind. More generally, it constructs and executes a function call from a name or a function and a list of arguments to be passed to it. In our case the function is rbind and the list is the list of data frames containing the data loaded from the csvs, produced by lapply.

**Reconstructing the results of the original post**

To fully reconstruct the results from the original post, we need to do two extra operations

* Add the source file names to the data frame
* Fix and reformat the dates

To do this, we will simply adjust the FUN in the lapply – in the above example, we have only used read.csv. Below, we will make a small function to do the extra steps:

filePaths <- list.files(data\_dir, "\\.csv$", full.names = TRUE)

result <- do.call(rbind, lapply(filePaths, function(path) {

df <- read.csv(path, stringsAsFactors = FALSE)

df[["source"]] <- rep(path, nrow(df))

df[["Month\_Year"]] <- as.Date(

paste0(sub("-20", "-", df[["Month\_Year"]], fixed = TRUE), "-01"),

format = "%b-%y-%d"

)

df

}))

# View part of the result

head(result)

## Month\_Year Hospital\_Name Hospital\_ID

## 1 2015-08-01 AMNCH 1049

## 2 2015-08-01 AMNCH 1049

## 3 2015-08-01 AMNCH 1049

## 4 2015-08-01 Bantry General Hospital 704

## 5 2015-08-01 Bantry General Hospital 704

## 6 2015-08-01 Bantry General Hospital 704

## Hospital\_Department ReferralType TotalReferrals

## 1 Paediatric ENT General Referral 2

## 2 Paediatric Gastroenterology General Referral 4

## 3 Paediatric General Surgery General Referral 4

## 4 Gastroenterology General Referral 12

## 5 General Medicine General Referral 18

## 6 General Surgery General Referral 43

## source

## 1 data/r005/ie-general-referrals-by-hospital//general-referrals-by-hospital-department-2015.csv

## 2 data/r005/ie-general-referrals-by-hospital//general-referrals-by-hospital-department-2015.csv

## 3 data/r005/ie-general-referrals-by-hospital//general-referrals-by-hospital-department-2015.csv

## 4 data/r005/ie-general-referrals-by-hospital//general-referrals-by-hospital-department-2015.csv

## 5 data/r005/ie-general-referrals-by-hospital//general-referrals-by-hospital-department-2015.csv

## 6 data/r005/ie-general-referrals-by-hospital//general-referrals-by-hospital-department-2015.csv

Lets look at the extra code in the lapply:

* Instead of just using read.csv, we have defined our own little function that will do the extra work for each of the file paths, which are passed to the function as path
* We read the data into a data frame called df using read.csv, and can we specify stringsAsFactors = FALSE, as the tidyverse packages do this by default, while base R’s default is different
* We add a new column source with the file name stored in path, repeated as many times as df has rows. This is a bit overkill here and could be done simpler, but it is quite robust and will also work with 0-row data frames
* We transform the Month\_Year into the requested date format with as.Date. Note that the relatively ugly sub() part is caused mostly by inconsistency in the source data itself
* Using [[ instead of $ is less pleasing to the eye, but we find it to be good practice, so sacrifice a bit of readability

**Alternatives to base R**

**Using data.table**

Another popular package that can help us achieve the same is data.table, so let’s have a look and reconstruct the results with data.table’s features:

library(data.table)

filePaths <- list.files(data\_dir, "\\.csv$", full.names = TRUE)

result <- rbindlist(lapply(filePaths, function(x) {

r <- fread(x)

r[, ':='(

source = x,

Month\_Year = as.Date(

paste0(sub("-20", "-", r[, Month\_Year], fixed = TRUE), "-01"),

format = "%b-%y-%d"

)

)]

}))

# View part of the result

head(result)

## Month\_Year Hospital\_Name Hospital\_ID

## 1: 2015-08-01 AMNCH 1049

## 2: 2015-08-01 AMNCH 1049

## 3: 2015-08-01 AMNCH 1049

## 4: 2015-08-01 Bantry General Hospital 704

## 5: 2015-08-01 Bantry General Hospital 704

## 6: 2015-08-01 Bantry General Hospital 704

## Hospital\_Department ReferralType TotalReferrals

## 1: Paediatric ENT General Referral 2

## 2: Paediatric Gastroenterology General Referral 4

## 3: Paediatric General Surgery General Referral 4

## 4: Gastroenterology General Referral 12

## 5: General Medicine General Referral 18

## 6: General Surgery General Referral 43

## source

## 1: data/r005/ie-general-referrals-by-hospital//general-referrals-by-hospital-department-2015.csv

## 2: data/r005/ie-general-referrals-by-hospital//general-referrals-by-hospital-department-2015.csv

## 3: data/r005/ie-general-referrals-by-hospital//general-referrals-by-hospital-department-2015.csv

## 4: data/r005/ie-general-referrals-by-hospital//general-referrals-by-hospital-department-2015.csv

## 5: data/r005/ie-general-referrals-by-hospital//general-referrals-by-hospital-department-2015.csv

## 6: data/r005/ie-general-referrals-by-hospital//general-referrals-by-hospital-department-2015.csv

Where

* rbindlist does the same as do.call("rbind", l) on data frames, but much faster
* fread is similar to read.table (and read.csv, which uses read.table) but faster and more convenient
* ':='() is the data.table syntax to create multiple new columns in a data.table (data frame)

**Using the tidyverse**

This is covered in much detail in the [post that inspired this one](https://www.gerkelab.com/blog/2018/09/import-directory-csv-purrr-readr/).

**Benchmarking for Twitter fun**

First off we are mostly looking at it for the fun of reacting [to Twitter discussion](https://twitter.com/_ColinFay/status/1046832479288676360), so take it for what it’s worth, by no means this is what we would call proper benchmarking.

Now that we have seen 3 ways to achieve the same goal, let’s look at speed. Note that we will be friendly to the tidyverse and not attach the entire package as is done in the [original post](https://www.gerkelab.com/blog/2018/09/import-directory-csv-purrr-readr/), however only those packages that we really need to get a more appropriate benchmark.

**Full script run benchmark**

First, we will perform an execution of an R script containing just the above code chunks (and [the tidyverse one](https://jozefhajnala.gitlab.io/r/post/data/r005/benchmarking/tidyverse.R)) a thousand times. The timing will also include overhead for launching the process, but this effect is present for all three scenarios and the variance should be safely covered by the fact that we execute 1000 times:

time for i in {1..1000};

do Rscript --vanilla data/r005/benchmarking/base.R &>/dev/null;

done

##

## real 3m51.477s

## user 3m16.364s

## sys 0m24.752s

time for i in {1..1000};

do Rscript --vanilla data/r005/benchmarking/datatable.R &>/dev/null;

done

##

## real 7m18.058s

## user 6m30.060s

## sys 0m36.888s

time for i in {1..1000};

do Rscript --vanilla data/r005/benchmarking/tidyverse.R &>/dev/null;

done

##

## real 20m43.588s

## user 19m21.888s

## sys 1m7.524s

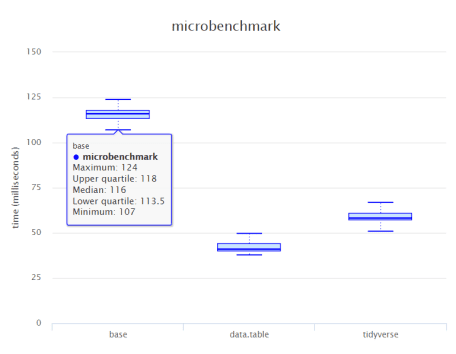
Visualizing the results shows that base R is the clear winner here, largely due to package loading overhead. Any performance benefits of the other packages are not enough to catch up in this very small use case:

If interested, you can look at the scripts ran above:

* [base.R](https://jozefhajnala.gitlab.io/r/post/data/r005/benchmarking/base.R)
* [datatable.R](https://jozefhajnala.gitlab.io/r/post/data/r005/benchmarking/datatable.R)
* [tidyverse.R](https://jozefhajnala.gitlab.io/r/post/data/r005/benchmarking/tidyverse.R)

**Benchmarking without package loading overhead**

We could argue that it is not fair to include the library statements in the benchmark, as the overhead can be relatively big considering how small the actual action done by the code is, as we are only processing 4 small files. Here is a benchmark omitting the overhead and only executing the relevant code with the packages pre-loaded, using microbenchmark with a 100 iterations:



boxplot

Visualizing the results in this case shows that data.table is a winner, with base R being the slowest of the options.