Pollutantmean Solution Discussion

No tags yet. + Add Tag

Sort replies by: Oldest first Newest first Most popular

Derek Franks · 17 days ago

Now that the hard deadline for assignment 1 has passed, I wanted to pull together a bit of a walkthrough for the pollutantmean() function.

First, I want to show how to "solve" the assignment. Second, I want to show some more advanced concepts that will hopefully help you with the remaining assignments and with your usage of R in the rest of the Data Science specialization.

Keep in mind that I'm only going to cover the pollutantmean() function. I'm not going to post solutions for any of the other assignments.

So, if you followed the tutorial, you ended up with this function:

```
weightmedian <- function(directory, day)  {
        files_list <- list.files(directory, full.names=TRUE)
        dat <- data.frame()
        for (i in 1:5) {
                dat <- rbind(dat, read.csv(files_list[i]))
        }
        dat_subset <- dat[which(dat[, "Day"] == day),]
        median(dat_subset[, "Weight"], na.rm=TRUE)
}
```

I purposely made this function very similar to a working solution to pollutantmean(), but different in a few ways so that it's not possible to just blindly copy and paste your way to a solution.

Solution 1

So here is a solution for pollutantmean() that is based on what is covered in the tutorial:

```
pollutantmean <- function(directory, pollutant, id = 1:332) {
        files_list <- dir(directory, full.names=TRUE)
        dat <- data.frame()
        for (i in id) {
                dat <- rbind(dat, read.csv(files_list[i]))
        }
        mean(dat[, pollutant], na.rm=T)
}
```

When you look at it laid out like that, it's pretty simple, isn't it? What I've done is to build a file list of

everything in the specdata directory. I then create an empty data frame. Then I loop through the files specified by `id` and use `rbind()` to merge them all into a single data frame. Finally I take the `mean()` of the column of that data frame defined by `pollutant`.

We don't put any quotes around `pollutant` because it's not a literal name of a column. That's also why `dat$pollutant` doesn't work.

You'll also notice that I'm not trying to combine every single csv file and then subset by `id`. You actually can do that, but a) it will be very, very slow with this approach, and b) `dat_id <- dat[which(dat$id == id), ]` isn't going to work. Actually it will work, but only if `id` is a single number (same thing goes for `day` in `weighmedian()`).

If you have `dat[which(dat$id == id), ]` and 'id = 1:10', you need to use `%in%` instead of `==`. Otherwise, R will compare 1 with row 1, 2 with row 2... 1 with row 11, 2 with row 12 and so on. This is exactly the same as the SWIRL tutorial showing how R recycles vectors of different lengths.

So the solution above is certainly a valid approach and many of you probably ended up with this or something very similar such as changing `dir(directory, full.names = TRUE)` to `dir(directory, full.names = TRUE)[id]` and then changing your for loop to `i in seq_along(files_list)`. Or maybe you used `sprintf` and `paste` to create a list of file names. You could even just build a list of file names in excel, save them in a csv file and read that list into R. As you'll see, there are a lot of different ways to create a working solution for `pollutantmean()`

The issue with this approach in general though is that it's very slow. If you run `pollutantmean("specdata","nitrate")` you'll find that it probably takes anywhere from 45 seconds to several minutes or more to complete.

We can shorten this significantly by not actually creating an entire data frame and then subsetting it to take the mean. If we only are interested in the mean of a specific pollutant, why not just create a vector of that pollutant and take the mean?

Solution 2

Here's an example that does just that:

```
pollutantmean <- function(directory, pollutant, id = 1:332) {
        files_list <- dir(directory, full.names=TRUE)
        dat <- as.numeric()
        for (i in id) {
                dat <- c(dat, read.csv(files_list[i])[, pollutant])
        }
        mean(dat, na.rm=T)
}
```

You can see that the basic structure is unchanged. But by creating a vector that contains only the column specified by `pollutant` instead of an entire dataframe, this approach will run `pollutantmean("specdata","nitrate")` in a matter of seconds. On my system, this approach yields a roughly 17x performance improvement.

As a general rule, it's much faster to work with simpler objects (vectors) in R and build more complex objects (data frames) as a final step if needed. But also keep in mind that maximum absolute performance isn't usually your top priority when working in R. This also isn't a very "R-like" way of approaching this problem.

Solution 3

A much more "R-like" approach would be to take advantage of the apply family of functions. The biggest problem with the approaches above is that they involve growing an object inside of a loop by copying that object over and over. It works, but it's a very inefficient way of building an object - particularly as they get really big.

There are several ways to address this in your code this but I'm going to focus on `lapply()` which in my experience tends to be the "workhorse" of the apply family. The "l" stands for "list". Data frames are just lists in which each element has the same length.

The basic idea with `lapply()` is to take a list and iterate over that list with a specified function. The output of `lapply()` will always be a list. Under the covers it's still a loop, but it doesn't build the output by copying and re-copying the data. A such, it works much faster and usually requires far less code than implementing the same functionality within a for loop. This means it's easier to use interactively at the command line.

We're going to start out the same way, by using `dir()` to create a list of all our files. Then we'll use `lapply()` to create a list that contains the data of each csv file. The syntax is `lapply(object_you_want_to_iterate_over, function_you_want_to_use)`. So our code is going to be `dat_temp <- lapply(files_list, read.csv)`. This reads in each file from `files_list` and stores it as an element of `dat_temp`. We can subset `files_list` by `id` in order to limit our data to the range called in the function.

To better understand what's going on with `lapply()` you can run the following commands at the console:

```
files_list <- dir("specdata", full.names=TRUE)
dat_temp <- lapply(files_list[1:10], read.csv)
summary(dat_temp)
str(dat_temp)
```

Once we have that list, we can use a function called `do.call()` to combine them into a single data frame. `do.call` lets you specify a function and then passes a list as if each element of the list were an argument to the function. The syntax is `do.call(function_you_want_to_use, list_of_arguments)`. In our case, we want to `rbind()` the output of our `lapply` list. So we'll use `dat <- do.call(rbind, dat_temp)`. This gives us a data frame, `dat` that is just like what we created in our original approach.

The function looks like this:

```
pollutantmean <- function(directory, pollutant, id = 1:332) {
        files_list <- dir(directory, full.names=TRUE)
        dat_temp <- lapply(files_list[id], read.csv)
        dat <- do.call(rbind, dat_temp)
        mean(dat[, pollutant], na.rm=T)
}
```

There are several benefits to this approach to `pollutantmean()`. First, you could do this interactively at the command line. While the goal of the assignment is to create a function to calculate the mean of a given pollutant, in the real-world, you're probably more likely to want to create a dataframe that contains all of your data and then manipulate and analyze that data frame as needed. You can do that with a couple of lines of code very easily.

Second, it's fast. It's not as fast as the vector approach above, but it's way faster than the original approach. On my system, it's 8x faster than the first approach and about 2x slower than the vector approach. It's not absolutely as fast as we can achieve, but it strikes a good balance between speed and utility.

This is the approach that I would personally use if you handed me the specdata folder and started asking me questions about it. I'd create a dataframe with everything in it using `lapply()` and `do.call()` and then start conducting my analysis.

So for example:

```
files_list <- dir("specdata", full.names=TRUE)
dat <- do.call(rbind, lapply(files_list, read.csv))
summary(dat)

##          Date           sulfate          nitrate             ID
##  2004-01-01:   250   Min.   : 0       Min.   : 0       Min.   :  1
##  2004-01-02:   250   1st Qu.: 1       1st Qu.: 0       1st Qu.: 79
##  2004-01-03:   250   Median : 2       Median : 1       Median :168
##  2004-01-04:   250   Mean   : 3       Mean   : 2       Mean   :164
##  2004-01-05:   250   3rd Qu.: 4       3rd Qu.: 2       3rd Qu.:247
##  2004-01-06:   250   Max.   :36       Max.   :54       Max.   :332
##  (Other)   :770587   NA's   :653304   NA's   :657738
```

That gets me a data frame `dat` with all of my data in it. You could actually collapse the code down to 1 line if you wanted to.

```
dat <- do.call(rbind, lapply(dir("specdata", full.names=TRUE), read.csv))
```

Then I can work with that data frame as needed:

```
dat_1_10 <- dat[which(dat$ID %in% 1:10),]
summary(dat_1_10)


##          Date          sulfate         nitrate            ID
##  2003-01-01:    9   Min.   : 0      Min.   :0       Min.   : 1.00
##  2003-01-02:    9   1st Qu.: 2      1st Qu.:0       1st Qu.: 3.00
##  2003-01-03:    9   Median : 3      Median :1       Median : 5.00
##  2003-01-04:    9   Mean   : 4      Mean   :1       Mean   : 5.09
##  2003-01-05:    9   3rd Qu.: 5      3rd Qu.:1       3rd Qu.: 7.00
##  2003-01-06:    9   Max.   :28      Max.   :7       Max.   :10.00
##  (Other)   :23685   NA's   :20177   NA's   :20095


mean(dat_1_10$nitrate, na.rm=T)


## [1] 0.7976
```

If I wanted to be a little more "sophisticated" with how I was doing things, I might use a package like

`dplyr` to handle the data manipulation or `tidyr` to tidy the data up a bit first.

So for instance, I'll tidy up `dat`:

```
library("tidyr")
dat_tidy <- gather(dat, pollutant, reading, 2:3)
str(dat_tidy)



## 'data.frame':    1544174 obs. of  4 variables:
##  $ Date     : Factor w/ 4018 levels "2003-01-01","2003-01-02",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ ID       : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ pollutant: Factor w/ 2 levels "sulfate","nitrate": 1 1 1 1 1 1 1 1 1 1 ...
##  $ reading  : num  NA NA NA NA NA NA NA NA NA NA ...

head(dat_tidy)



##         Date ID pollutant reading
## 1 2003-01-01  1   sulfate      NA
## 2 2003-01-02  1   sulfate      NA
## 3 2003-01-03  1   sulfate      NA
## 4 2003-01-04  1   sulfate      NA
## 5 2003-01-05  1   sulfate      NA
## 6 2003-01-06  1   sulfate      NA
```

To use "tidy data" terminology, I've just melted the data set and turned the nitrate and sulfate columns into rows. Now each row of the data is a unique observation. As you'll see in later courses, one of the benefits of this format is that it makes it easier to visualize the data using a package like ggplot2.

Once I have tidy data, I might then use `dplyr` to do some data munging. So for instance:

```
library("dplyr")
dat_tidy %>% group_by(pollutant) %>% summarise(mean_reading = mean(reading, na.rm=T))



## Source: local data frame [2 x 2]
##
##   pollutant mean_reading
## 1   sulfate        3.189
## 2   nitrate        1.703
```

Essentially, I've told `dplyr` to group the data by pollutant and show me the mean reading value for each. The `%>%` just allows me to chain together commands. This is just a really simple example but it should give you some ideas about how some of these packages start working together. A basic understanding of `tidyr` and `dplyr` will also help out in the Getting and Cleaning Data class.

I should also add that it's possible to replicate what I just did above with `dplyr` and `tidyr` using functions in base R such as `reshape()`, `by()`, `aggregate()` and `tapply()`. It just usually takes longer and requires more code.

Solution 4

I want to show one more example of how you can "solve" the assignment. This is an implementation that uses the logic from our vector approach and combines it with `lapply()`. Here's what it looks like:

```
pollutantmean <- function(directory, pollutant, id = 1:332) {
        files_list <- dir(directory, full.names=TRUE)
        dat <- lapply(files_list[id], function(x) read.csv(x)[[pollutant]])
        mean(unlist(dat), na.rm=T)
}
```

This is the fastest computationally of all the approaches. I'm using an anonymous function in combination with `lapply`. Anonymous functions are a topic of their own, but essentially, instead of feeding `lapply` a function such as `read.csv` like we did before, I've written a short "function" that subsets the ouput of `read.csv` to produce a numeric vector. If I removed the `[[pollutant]]`, this anonymous function would work exactly the same as our `lapply(file_list, read.csv)` in the previous function.

However because of that `[[pollutant]]`, instead of each element of the output list being an entire data frame like in the last example, in this case, each element of the list is a numeric vector that contains the data for the specified pollutant. Then I use the function `unlist()` to take the data out of our list and combine it into one big numeric vector and then take the mean of that vector.

In Conclusion

Don't worry if you can't follow everything (espcially the last version of the pollutantmean - anonymous functions are very useful but they can be confusing at first). There are also about a million different ways to "solve" the assignment, but hopefully this has cleared up some questions you may have had and given you some ideas about other ways to approach this if you have to do something similar in the future.

Generally speaking, the apply family of functions are very, very useful in R. They can be a bit confusing but if you can get comfortable with them (or at least a couple like `apply()` and `lapply()`), get comfortable with a few data manipulation packages such as `dplyr` and `tidyr`, and then learn `ggplot2`, you'll pretty much be ready for what you'll be doing in the rest of the Data Science specialization.

↑ 10 ↓ · flag

Anonymous · 17 days ago

Wow, great stuff, Derek! Thank you for posting.

With respect to the huge difference in speed between looping `rbind` and `do.call(rbind...)` -- is this a memory allocation issue? The thing I do not quite understand is how `lapply` gets around this. In the lectures, I thought it was suggested that the `lapply` family of functions was more semantic and produced better code, but was not necessarily much faster -- and also that they were implemented using loops anyway.

Does it work out because the loop happens at a lower level (i.e, in C), or because the implementation automatically accounts for the memory allocation (i.e, initializing the return data frame to whatever size it needs to *before* looping)?

Solution 4 required the writing of an anonymous function. In this case, that function was one line. But how long can anonymous functions be? Is it bad form to have to write your own custom anonymous function separately? i.e, `lapply(x, myfunction(elem))`

And, is it okay to simply write a lengthy anonymous function? For example, in javascript you frequently see this with callbacks. In R, I suppose it could be:

```
a <- lapply(x, function(e) {
    ...
    ...
    b <- lapply (e, function(e2) {
        ...
    }
    ...
    ... ##etc
}, ...)
```

↑ 1 ↓ · flag

Derek Franks · 17 days ago

I don't know that memory allocation is technically the right way to phrase it, but yes, that's basically why it's faster. Here's essentially the same thing as solution 3 using a for loop:

```
pollutantmean <- function(directory, pollutant, id = 1:332) {
    fileList <- list.files(directory, full.names = TRUE)
    tmp <- vector(mode = "list", length = length(id))
    for (i in seq_along(fileList)) {
      tmp[[i]] <- read.csv(fileList[[i]])
    }
    output <- do.call(rbind, tmp)
    mean(output[[pollutant]])
}
```

Essentially, each file gets read in once and moved one time, rather than the entire data frame being re-copied on each pass through the loop In theory, I believe lapply should be faster than a for loop, even when pre-allocated. But in practice, the real benefit is that it's easier to write.

Hadley Wickham has written about this fairly extensively: http://adv-r.had.co.nz/memory.html#modification

An anonymous function can be as long as you want it to be. They're really just time savers meant for situations where you don't want to take the time to write out a formal function. Personally, if I'm going to use multiple lines to write an anonymous function, I'll probably just give it a name so it's easier to reference again later.

If you're defining your own "anonymous" function outside of the lapply call, then you've basically just written a regular function that you're referencing with lapply. That's perfectly fine and probably a good idea if it's a function that you will need to use repeatedly.

↑ 1 ↓ · flag

Anonymous · 17 days ago

Thank you so much! You've been a great resource to so many of us in this class. Much appreciated.

0 · flag

+ Comment

Anonymous · 17 days ago

Hi Derek!
Thank you so much for posting the solution. I tried the exact code you put here for solution 1 and I still get the same answer...

rror in file(file, "rt") : cannot open the connection In addition: Warning message:
In file(file, "rt") : cannot open file 'NA': No such file or directory

Note:
- when I typed dir(), it shows all 332 csv files
- getwd() shows C:/Users/.../Specdata
- list.files() shows all 332 csv files

What have I done wrong? please help. Thank you.

Em

0 · flag

 Leonard Greski Signature Track · 17 days ago

Working directory needs to be the parent of specdata, not specdata.

Len

0 · flag

Anonymous · 17 days ago

Thanks Len. I have changed my WD...now I got this answer instead

Error in `[.data.frame`(dat, , pollutant) : undefined columns selected

does this mean the function doesn't recognize pollutant in the column of files? I just don't understand how R knows what pollutant is? I mean, do we set pollutant <- c("Nitrate", "sulfate") first?

Thanks for your time :)

0 · flag

 Leonard Greski Signature Track · 17 days ago

Since the pollutant argument does not have a default value, if you fail to specify one you

receive an "undefined columns" error message. You need to include a pollutant as an argument to the pollutant mean() function, either sulfate or nitrate, as follows:

```
[Workspace loaded from ~/RProgramming/.RData]

> source('~/RProgramming/derek-pollutantmean1.R')
> pollutantmean("specdata","nitrate",1:5)
[1] 0.9030249
>
```

Len

↑ 0 ↓ · flag

+ Comment

Pierre Tourigny · 17 days ago

Thank you, Derek.
I learned a lot from your practice assignment and even more from this post.

↑ 1 ↓ · flag

+ Comment

Eric Philippon Signature Track · 16 days ago

Thanks a lot Derek, I've learned a lot by reading your code. I personally think that it's one of the best way to improve a language knowledge, to be able to read someone else code (I love the shortcut to filter directly the read.csv function output).

Here is my version using a for loop which is somewhat faster on my computer than your "vector" function when I loop over all files. It doesn't compete in terms of length (mine is too long), but I think it's ok :)

```
pollutantmean <- function(directory, pollutant, id=1:332) {
  # intermediate vectors, contains values needed for mean
  intermediateSum <- numeric(length = length(id))
  intermediateNbValues <- numeric(length = length(id))

  fileslist <- list.files(directory, full.names = T)
  #iteration variable for vectors
  iter <- 1

  #calculation over all files ids
  for (ii in fileslist[id]) {
    filepath <- ii
    #print(filepath)

    #get data
    data  <- read.csv(filepath, header = TRUE,quote = '"')
```

```
    #sample data on the selected polluant
    data2 <- data[,pollutant]

    #remove NAs
    dataset <- data2[!is.na(data2)]

    #calculate&store values needed
    intermediateSum[iter] <-sum(dataset)
    intermediateNbValues[iter] <- length(dataset)

    #increment iteration number for next loop
    iter <- iter+1
  }

  #calculte global sum&number of valid values
  sumAllValues <- sum(intermediateSum)
  nbValues <- sum(intermediateNbValues)
  #computes the mean
  sumAllValues / nbValues
}

# my version :
# > system.time(pollutantmean("data_cours/specdata", "nitrate"))
# utilisateur     système      écoulé
# 3.742       0.053       3.795
# your version with data frame :
# > system.time(pollutantmeanDataframe("data_cours/specdata", "nitrate"))
# utilisateur     système      écoulé
# 56.793       8.280      65.037
# your version with vector
# > system.time(pollutantmeanVector("data_cours/specdata", "nitrate"))
# utilisateur     système      écoulé
# 4.408       0.628       5.038
# your version with apply
# > system.time(pollutantmeanApply("data_cours/specdata", "nitrate"))
# utilisateur     système      écoulé
# 7.322       1.956       9.280
```

2 · flag

+ Comment

 Pierre Tourigny · 15 days ago

Looking at different versions of a program, especially one on which I worked myself, is very educational.

After studying Eric (ou vraisemblablement Éric)'s code, I came up with this streamlined version:

```
pollutantmean_stream <- function(directory, pollutant, id = 1:332) {  # Eric's version streamlin
ed
  files_list <- dir(directory, full.names=TRUE)
  sums <- array(dim = c(2, length(id)))
  for (x in id) {
    y <- read.csv(files_list[x])[[pollutant]]
    sums[,x] <- c(sum(y, na.rm = T), sum(!is.na(y)))
  }
  sum(sums[1,]) / sum(sums[2,]
}
```

I then realized that I need keep track of only two numbers: the current cumulative sum for numerator and denominator.

```
pollutantmean_accu <- function(directory, pollutant, id = 1:332) {  # Only keep track of 2 numbe
rs
  files_list <- dir(directory, full.names=TRUE)
  sums <- c(0, 0)
  for (x in id) {
    y <- read.csv(files_list[x])[[pollutant]]
    sums <- sums + c(sum(y, na.rm = T), sum(!is.na(y)))
  }
  sums[1] / sums[2]
}
```

I wanted to time these versions, but system.time gives a different answer each time for the same code. So I decided to run each version multiple times and calculate the average CPU time.

```
compare.pollutantmean <- function(x = 1) {  # time each pollutantmean version x times
  once.each <- function(y) {
    c(  # system.time()[[1]] is user time, aka cpu time
      Solution_1 = system.time(pollutantmean_sol1("specdata","nitrate"))[[1]],
      Solution_2 = system.time(pollutantmean_sol2("specdata","nitrate"))[[1]],
      Solution_3 = system.time(pollutantmean_sol3("specdata","nitrate"))[[1]],
      Solution_4 = system.time(pollutantmean_sol4("specdata","nitrate"))[[1]],
      Eric = system.time(pollutantmean_Eric("specdata","nitrate"))[[1]],
      Streamlined = system.time(pollutantmean_stream("specdata","nitrate"))[[1]],
      Accumulator = system.time(pollutantmean_accu("specdata","nitrate"))[[1]]
    )
  }
  sapply(1:x, once.each)  # sapply simplifies the list of 7-element vectors to a 7 by x matrix
}
```

Here are the results on my admittedly old machine. I let compare.pollutantmean run unattended so I don't know how long it took.

```
> times <- compare.pollutantmean(50)
> data.frame(Mean.CPU.Time = a <- rowMeans(times), Compared.to.Solution.2 = a/a[2])
            Mean.CPU.Time Compared.to.Solution.2
Solution_1        85.5980             17.0134362
Solution_2         5.0312              1.0000000
Solution_3        11.5688              2.2994117
```

```
Solution_4          4.2642              0.8475513
Eric                4.2444              0.8436158
Streamlined         4.1838              0.8315710
Accumulator         4.1792              0.8306567
```

Derek's solution 4 is not the fastest, but it cannot be beaten for simplicity and elegance.

1 · flag

Eric Philippon Signature Track · 14 days ago

Wonderful version here Pierre! I love the simplicity of your last version. And thanks a lot for the comparison process, I think I might use it some other time :)

When I see this post, I think it's a pity that we can't share/discuss assignments, I've learned so much reading!

0 · flag

+ Comment

Raj Bissessur Signature Track · 14 days ago

Hi Derek,

can you please help me on understanding why I cannot subset my data frame?

Here are the codes to the 1st assignment (yes, i didn't make the deadline as I was struggling to get this working). But then i followed your steps above, but I'm still having this error message on execution. Here are my functions (the first one is mine) then I copied and pasted yours in the file....I cannot understand this error on undefined column selection. I'm running R on a mac through R studio.

```
pollutantmean <-function(directory,pollutant,id=1:332){
  files_list <-list.files(directory,full.names=TRUE)
  dat <-data.frame()
  files_list <-list.files(directory,full.names=TRUE)
  tmp <-vector(mode="list",length=length(files_list))
  tmp <-lapply(files_list[id],read.csv)
  dat<-do.call(rbind,tmp)

  dat_sub <-dat[dat$ID %in% id,]
  mean(dat_sub[,pollutant] ,na.rm=T)

}
pollutantmean2 <- function(directory, pollutant, id = 1:332) {
  files_list <- dir(directory, full.names=TRUE)
  dat_temp <- lapply(files_list[id], read.csv)
  dat <- do.call(rbind, dat_temp)
  mean(dat[, pollutant], na.rm=T)
}
pollutantmean3 <- function(directory, pollutant, id = 1:332) {
  files_list <- dir(directory, full.names=TRUE)
```

```
  dat <- lapply(files_list[id], function(x) read.csv(x)[[pollutant]])
  mean(unlist(dat), na.rm=T)
}
```

The error messages:

```
> pollutantmean2("specdata","sulphate",1:30)
 Error in `[.data.frame`(dat, , pollutant) : undefined columns selected
```

↑ 0 ↓ · flag

Derek Franks · 14 days ago

It's probably because you're misspelling `sulfate`.

↑ 0 ↓ · flag

 Leonard Greski Signature Track · 14 days ago

...and we hadn't yet learned

```
validPollutants <- c("sulfate","nitrate")
if (!pollutant %in% validPollutants) {
   stop(paste("invalid pollutant:",pollutant))
}
```

:)

Len

↑ 0 ↓ · flag

+ Comment

Anonymous · 14 days ago

Doh.....that's the price to pay when you learn English in England.! Thanks

↑ 0 ↓ · flag

 Leonard Greski Signature Track · 14 days ago

Those cheeky Americans, first they revolt, then they ruin the language :).

↑ 0 ↓ · flag

+ Comment

 Al Warren · 14 days ago

How about the data.table way:

```
library(data.table)
```

```
pollutantmean <- function(directory="specdata", pollutant="nitrate", id = 1:332) {
    files <- list.files(directory, full.names = T)[id]
    dt <- rbindlist(lapply(files, fread, header=T, na.strings="NA"))
    mean(dt[[pollutant]], na.rm=T)
}

system.time(replicate(10, pollutantmean("specdata", "sulfate", 1:10)))
##    user  system elapsed
##    0.44    0.03    0.46

system.time(replicate(10, pollutantmean("specdata", "nitrate", 70:72)))
##    user  system elapsed
##    0.14    0.01    0.15

system.time(replicate(10, pollutantmean("specdata", "sulfate", 34)))
##    user  system elapsed
##    0.05    0.02    0.06

system.time(pollutantmean("specdata", "nitrate"))
##    user  system elapsed
##    1.57    0.17    1.74
```

Compared to lapply:

```
pollutantmean <- function(directory="specdata", pollutant="nitrate", id = 1:332) {
    files_list <- dir(directory, full.names=TRUE)
    dat <- lapply(files_list[id], function(x) read.csv(x)[[pollutant]])
    mean(unlist(dat), na.rm=T)
}

system.time(replicate(10, pollutantmean("specdata", "sulfate", 1:10)))
##    user  system elapsed
##    1.52    0.06    1.73

system.time(replicate(10, pollutantmean("specdata", "nitrate", 70:72)))
##    user  system elapsed
##    0.45    0.04    0.52

system.time(replicate(10, pollutantmean("specdata", "sulfate", 34)))
##    user  system elapsed
##    0.11    0.02    0.15

system.time(pollutantmean("specdata", "nitrate"))
##    user  system elapsed
##    5.49    0.26    7.77
```

↑ 4 ↓ · flag

 Pierre Tourigny · 13 days ago

Thank you Al for introducing me to the blazingly fast data.table package.
I wondered if your version could be even faster if only the pollutant column was used.

```
pollutantmean_select <- function(directory="specdata", pollutant="nitrate", id = 1:332)
{
  files <- list.files(directory, full.names = T)[id]
  dt <- rbindlist(lapply(files, fread, select = pollutant))
  mean(dt[[pollutant]], na.rm=T)
}
```

```
> compare.pollutantmean(30)
           Mean.CPU.Time Compared.to.First
Solution_4     4.2846667         1.0000000
Al             1.1993333         0.2799129
Select         0.5033333         0.1174732
```

1 · flag

+ Comment

New post

To ensure a positive and productive discussion, please read our forum posting policies before posting.

B | I | Link | <code> | Pic | Math | Edit: Rich | Preview

Make this post anonymous to other students

Subscribe to this thread at the same time

Add post