Reformat

Anuj Dahiya

December 24, 2019

The purpose of this document is to how to process the text files we have previously seen in `Download.pdf` within the Step 1 folder.

Libraries

The first steps to succesfully reformatting all of the text files in the `Downloads files` folder is importing all of the necessary libraries.

```
library(knitr)
library(tidyverse)
library(data.table)
library(Kmisc)
library(foreach)
library(doParallel)
library(stringi)
library(lubridate)
```

A very quick summary of why each package is used:

- `knitr` is imported to set the working directory for the R-Markdown document.
- `tidyverse` is useful for its `%>%` (pipe) operator.
- `data.table` is extremely fast for writing dataframes to stored files.
- `Kmisc` is used for its `readlines()` which the fast version of R's base function, `readLines()` .
- `foreach` and `doParallel` are useful for parallel processing exporting data.frames to csv files

Directories

In order to work with the data files in Step 1 and rework them, we need to:

1. Assign a path we will import the text files
2. Assign a path we will to store the created .csv files

```
path = "~/GitHub/FIDE/Chess Scripts/Step 1 - Download/Downloaded files"
destination = "~/GitHub/FIDE/Chess Scripts/Step 2 - Reformat/Data csvs/"
opts_knit$set(root.dir = path)
```

Function

Reformat

This is by far the most cumbersome function to read, let alone work through. Long story short, many of the text files are hopelessly formatted:

- Some have improperly labeled columns
- Some have misspelled columns
- Some have their columns out line
- Some have blank & missing rows
- Some don't even have column headers to begin with

I would like to thank Kirsan Ilyumzhinov and his aliens for bestowing the challenge of fixing his organization's publically available files.

```
reformat <- function(file_csv, df){

#Reformat October 2002
if(file_csv == "OCT02FRL.TXT"){df[1] <- df[1]%>%
                                        gsub("COUNTRY", "  Fed ", .)%>%
                                        gsub("GAMES", "Gms", .)%>%
                                        gsub("BIRTHDAY", " BIRTHDAY", .)}

#Reformat April 2003
else if(file_csv == "APR03FRL.TXT"){df[1] <- df[1]%>%
                                       gsub("   CODE  ","ID_NUMBER",.)%>%
                                       gsub("COUNTRY","  FED  ",.)%>%
                                       gsub(" APR03", "APR03 ", .)%>%
                                       gsub(" GAMES", "GMS   ", .)%>%
                                       gsub("  BIRTHDAY", "BIRTHDAY  ", .)%>%
                                       gsub(" FLAG", "FLAG ", .)}

#Reformat April, July, October 2004 & January, July 2005
else if(file_csv %in% c("APR04FRL.TXT","JUL04FRL.TXT","OCT04FRL.TXT",
                        "JAN05FRL.TXT", "JUL05FRL.TXT")){
                                            df[1] <- df[1]%>%
                                            gsub("COUNTRY", "   FED ", . )%>%
                                            gsub("GAMES", "GAME ", .)%>%
                                            gsub(" BIRTHDAY", "BIRTHDAY", .)}

#Reformat January 2006 to July 2012
else if(file_csv == "JAN06FRL.TXT"){df <- df[-2]}
else if(file_csv %in% c("APR06FRL.TXT", "JUL06FRL.TXT", "OCT06FRL.TXT",
                      list.files(pattern = "[0][7-9][Ff][Rr][Ll].[Tt][Xx][Tt]"),
                      list.files(pattern = "[1][0-1][Ff][Rr][Ll].[Tt][Xx][Tt]"),
                      list.files(pattern = "[1][2][Ff][Rr][Ll].[Tt][Xx][Tt]")[1:5])){
                      df[1] <- df[1] %>% gsub("Titl", "Tit ", .)%>%
                                         gsub("Games", "Game ", .)%>%
                                         gsub("July", "Jul", .)}

#Insert underscore to make sure that proper columns are created in the functions afterwards
df[1] <- gsub("ID Number", "ID_NUMBER", df[1])%>%
         gsub("ID number", "ID_NUMBER", .)

return(df)
}
```

Indexes

All of the text files lack a proper delimeter to import the data on. Any normal dataset with have a comma or tab delimeter, but these have nothing of the sort. Given this, the function below helps tackle this problem. It grabs the column headers of a given text file and finds the indexes at which we need to insert delimeters at.

```
indexes <- function(df){
column_vector<- df[1]
indexes <- rep(0, nchar(df[1]))
for(i in 1:nchar(column_vector)){
    index = grep("\\s[A-z]", substr(column_vector, i, i+1))
    if (identical(index, integer(0)) == TRUE){indexes[i] = 0}
    else {indexes[i] = 1}
                                    }
return(which(indexes == 1))
                        }
```

Insert delimeters quickly

As discussed above, we need to insert delimeters at these indexes. Using R's base functions, we can use `utf8ToInt()` and `intToUtf8()`to quickly break down every string into vectors and replace vectors indexes with the delimeter we want. In this case, we will insert a commma.

```
utf_func <- function(df, indexed){
string <- utf8ToInt(df)
string[indexed] <- utf8ToInt("*")
return(intToUtf8(string))
}
```

File rename

Each exported file needs to be renamed so the function is helpful for that.

```
filenamer <- function(Year_num) {
  Year_num%>%
  substr(., nchar(.)-11, nchar(.)-7)%>%
  toupper()%>%
  paste(destination, ., ".csv", sep = "")
}
```

Write files

Lastly, we need to export the text files using `data.table`'s speedy `fwrite()` function.

```
fwrite_wrapper <- function(filename, df){
if (file.exists(filename)) {unlink(filename)}
fwrite(list(df), file = filename, quote = FALSE)
}
```

All files write

Below is a wrapper function that builds on all of the previously stated functions and puts it all together. Some of the files may have blank lines initially so unfortunately, we have to use `readLines()` (a somewhat slow version of `readlines()`, but allows users to read in files beginnign with blank lines).

```
All_files_fwrite <- function(year_vector){
text_insert_first = c("jul03frl.txt","OCT03FRL.TXT", "JAN04FRL.TXT",
                      "APR05FRL.TXT", "jan03frl.txt")
columns = "ID_NUMBER NAME                              TITLE FED  RATING GM  Bday      Flag "

if(year_vector %in% text_insert_first){
df = readLines(year_vector)
  if(nchar(df[1]) != 0) {
    cat("", df, file = year_vector, sep = "\n")
    df <- readLines(year_vector)
    df[1] <- columns} else{df[1] <- columns}
} else {df = readlines(year_vector)}

df <- reformat(year_vector, df)
indexed = indexes(df)
df = sapply(df , utf_func, indexed = indexed, USE.NAMES = FALSE)
filename <- filenamer(year_vector)
fwrite_wrapper(df, file = filename)
}
```

Multi Processing

The first time I ran a for loop that executes `All_files_fwrite()` on every single text file, it took ~20-25 minutes. The following function runs `All_files_fwrite()` in parallel. It took me a while to put it all together because of a few reasons:

1. `.export = functions` is necessary to import functions from R's global environment into the parallel processing function environment. Otherwise, global functions & variables aren't detected.
2. `detectCores()` is a useful function to check how many cores your computer has. My machine has 8 and it cuts down the original ~20-25 minutes to ~4-5 minutes!
3. `.packages = c("dplyr", "data.table", "Kmisc")` is needed because these packages are heavily involved in many of the functions defined above.
4. The rest is setting up a cluster network, which allows for parallel processing.

```
multi_processing <- function(Year_num){
                    functions = ls(globalenv())
                    cl <- makeCluster(detectCores())
                    clusterExport(cl, functions)
                    registerDoParallel(cl)
                    foreach(i = Year_num,
                            .export = functions,
                            .packages = c("dplyr",
                                          "data.table",
                                          "Kmisc")) %dopar% {All_files_fwrite(i)}
```

```
                    stopCluster(cl)
                                    }
```

A final touch and output

The final chunk here is divided into a few steps:

- `Year_num` gathers all of the text files into a vector.
- `system.time()` captures how long this process takes. If `multi_processing(Year_num)` does not run, you can run the commented out line below it.
- The last line prints out example `.csv` files that been created in the destination directory.

```
Year_num <- list.files(path = path, pattern = "[Ff][Rr][Ll].[Tt][Xx][Tt]")

system.time(multi_processing(Year_num))
```

```
##    user  system elapsed
##    0.33    0.26  315.66
```

```
##Run this if parallel processing doesn't work
# system.time(invisible(mapply(All_files_fwrite, Year_num)))

head(list.files(path = destination, pattern = "*.csv"))
```

```
## [1] "APR01.csv" "APR02.csv" "APR03.csv" "APR04.csv" "APR05.csv" "APR06.csv"
```

We can see the `.csv` files have been created, as a result.

I hope you found this document helpful. Writing this file was extremely time consuming. Many of the functions have only been optimized after several iterations and refinements.