FIDE Chess Download

Anuj Dahiya

January 13, 2020

Introduction

The purpose of this document is to show how I downloaded all of the standard rating files from the FIDE organizations website.

Import libraries

I don't think any packages are necessary to download any of the files. Still though, I use `knitr` and `dplyr` to display a table of urls that we will call on to download from later on.

```r
library(knitr)
library(dplyr)
```

Set up working directory

Before proceeding, we need to set up a working directory to store all of the downloaded files. Make sure to have an empty folder you can dump the files into. I've created a `Downloaded files` subfolder within the folder of this `.Rmd` document.

IMPORTANT: Please adjust `path` to the path of the folder you want the data to be in.

If you're not sure how to set your directory, you can choose `path` interactively using `choose.dir()`. If you choose to do so, make sure to comment out the first line and uncomment out the 2nd line.

```r
path = "~/GitHub/FIDE/Chess Scripts/Step 1 - Download/Downloaded files"
#path = choose.dir()

opts_knit$set(root.dir = path)
```

FIDE file format

R's default is to adjust all numbers, beginning with 0, and truncatte the leading zeros.

Below we setup many useful vectors of strings that will be useful for creating urls that we will download the `.zip` files from.

```
year_vector <- as.character(1:as.numeric(substr(Sys.Date(), 3, 4)))
month_vector <- tolower(substr(month.name, 1, 3))
url_vector = url_vect_destfile = rep(0, length(month_vector)*length(year_vector))

for(i in 1:length(year_vector)){
  if(nchar(year_vector[i]) == 1){
    year_vector[i] = paste("0", year_vector[i], sep = "")
  }
}

latest <- format(Sys.Date(), format="%b%Y")
latest <- paste(tolower(substr(latest, 1, 1)), substr(latest, 2, 3), substr(latest, nchar(latest)-1, ncl

latest
```

```
## [1] "jan20frl.zip"
```

`latest` tells us the most recent file FIDE should have hosted on their website.

Below, we put it all together using a somewhat ugly for loop, but it accomplishes our job nicely to create the urls we desire. Note that in August 2014 (which explains the `else if` statement at the 140th iteration), FIDE added an extra word, "standard_" to the URLs to be downloaded from.

```
for(i in seq_along(year_vector)){
  for(j in seq_along(month_vector)){
    if(12*(i-1)+j <= 140){
      url_vector[12*(i-1)+j] = paste("http://ratings.fide.com/download/", month_vector[j], year_vector[i
    }
    else if(12*(i-1)+j > 140){
      url_vector[12*(i-1)+j] = paste("http://ratings.fide.com/download/standard_", month_vector[j], yea
    }
    url_vect_destfile[12*(i-1)+j] <- substr(url_vector[12*(i-1)+j], nchar(url_vector[12*(i-1)+j])- 11 ,
  }
}

url_vect_destfile <- url_vect_destfile[1:which(url_vect_destfile == latest)]
url_vector <- url_vector[1:which(url_vect_destfile == latest)]

rm(list=setdiff(ls(), c("url_vector", "url_vect_destfile")))
```

URLs and their files

URL	File
http://ratings.fide.com/download/jan01frl.zip	jan01frl.zip
http://ratings.fide.com/download/feb01frl.zip	feb01frl.zip
http://ratings.fide.com/download/mar01frl.zip	mar01frl.zip
http://ratings.fide.com/download/apr01frl.zip	apr01frl.zip
http://ratings.fide.com/download/may01frl.zip	may01frl.zip
http://ratings.fide.com/download/standard_sep19frl.zip	sep19frl.zip
http://ratings.fide.com/download/standard_oct19frl.zip	oct19frl.zip
http://ratings.fide.com/download/standard_nov19frl.zip	nov19frl.zip
http://ratings.fide.com/download/standard_dec19frl.zip	dec19frl.zip
http://ratings.fide.com/download/standard_jan20frl.zip	jan20frl.zip

Above is a trimmed table of URLs and their corallary files. This is what what will be inputted in the `download_all()` function down below.

Download all chess files silently

I'm fairly sure you can remove the two lines involving `old`, but I'd keep them in case you start to get spammed with messages.

`download_all()` quietly checks each URL we will visit and downloads the file present at each URL. If a file doesn't exist at a URL (many of the early year URLs don't), then the function skips over the error.

Note: Download times vary!

```r
download_all <- function(link, dest){
  if (!file.exists(dest)) {
    tryCatch({
      download.file(link, dest, method="auto", quiet = TRUE)
    }, error=function(e){})
  }
}


old <- getOption("warn"); options(warn = -1)
invisible(mapply(download_all, url_vector, url_vect_destfile))
options(warn = old)
```

Unzipping and cleaning

Some brief housecleaning is taken care of below After the step below, you should only have text files in the directory you set at the beginning. All that's done below is unzipping the `.zip` files and deleting the `.zip` files.

```r
invisible(sapply(list.files(pattern = "*.zip"), function(x) unzip(x, exdir = getwd())))

unlink(list.files(pattern = "*.zip"))
```

Lastly, we can verify what is in our directory.

```r
list.files(pattern = "*.txt")%>%
head()
```

```
## [1] "apr08frl.txt" "apr09frl.txt" "aug12frl.txt" "jan03frl.txt" "jan09frl.txt"
## [6] "jan10frl.txt"
```

As we can see, we can see the downloaded and unzipped text files: Success!

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.

```r
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).

```r
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.

```r
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.

Cleaning

Anuj Dahiya

December 25, 2019

The purpose of this document is to clean the `.csv` files stored in the *Step 2 - Reformat/Data csvs/* folder.

Before beginning, we need to import several packages that help run the code below.

```
library(knitr)
library(dplyr)
library(data.table)
library(lubridate)
```

Access the data

In order to access the data we need to specify the folder we want to access the files from. We do this by defining a path to find them and set the path with `opts_knit$set(root.dir = path)`. In an R-Markdown document, it is required that you set the directory using this function. We can see the files in the directory with `head()`.

We also create a destination path with `dest` for later use so that we can export the files properly. `country_path` is a path to a dataset from which we will reference.

```
path = "~/GitHub/FIDE/Chess Scripts/Step 2 - Reformat/Data csvs/"
country_path = "~/GitHub/FIDE/Chess Scripts/Step 3 - FIDE Country Codes/Country Data/FIDE_codes.csv"
dest =  "~/GitHub/FIDE/Chess Scripts/Step 4 - Cleaning/Cleaned csvs/"
opts_knit$set(root.dir = path)

temp = list.files(path = path)
full_path <- paste(path, temp, sep = "")

head(temp)
```

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

Create functions to rename datasets

Below, I define a few functions that help us rename the datasets.

`month` gets the first 3 letters of the temp elements, the month name `num` converts each month into a number `add_zero` converts each month number into a usable form when we create dates later on.

```
month <- function(x){return(substr(x, 1, 3))}
num <- function(x) match(tolower(x), tolower(month.abb))
add_zero <- function(x){if (x <= 9){x = paste("0", x, sep = "")}; return(x)}
```

Get the month number of all of the files in the dataset

Below, we rename create the variables names when they are assigned and imported.

```r
temp%>%
sapply(month)%>%
sapply(num)%>%
sapply(add_zero)%>%
paste("20", substr(temp, 4, 5), "-", ., "-", "01", sep = "")-> month_num

head(month_num)
```

```
## [1] "2001-04-01" "2002-04-01" "2003-04-01" "2004-04-01" "2005-04-01"
## [6] "2006-04-01"
```

We can see that the datasets correspond to dates on which the is recorded.

Import all datasets

Below, we import and assign all of the datasets into memory. The only objects we need to hold onto is the data and several folder path references for later on. Therefore, we will remove everything unneeded with `rm(list=setdiff(ls())` below.

```r
for(i in 1:length(full_path)) {
assign(month_num[i], fread(full_path[i], sep = "*", data.table = FALSE,
                           strip.white = TRUE, blank.lines.skip = TRUE))
}

FIDE <- mget(ls(pattern = "[0-9][0-9]-[0-9][0-9]"))

rm(list=setdiff(ls(), c("FIDE", "path", "temp", "dest", "country_path")))
```

Data prep

In order to get the data to have useful and common values, we need to rename dozens of columns and values.

```r
vector_months <- c(month.abb, tolower(month.abb),toupper(month.abb))
string = ""
for (i in 1:length(vector_months)){
if (i == 1){string = vector_months[i]}
else if (i > 1) {string = paste(string, vector_months[i], sep = "|")}
}
string = paste(string, "RATING", sep = "|")


new = c("CM", "WCM", "WCM", "WGM", "WFM", "WFM", "GM", "IM", "FM", "WIM", "GM")
old = c("c", "wc", "WC", "wg", "WF", "wf", "g", "m", "f", "wm", "gm" )

dates = as.Date(names(FIDE))
```

```
months_vec <- months(dates)%>%toupper()%>%substr(., 1, 3)
year_vec <- year(dates)%>%as.character()%>%substr(3,4)
files <- paste(months_vec, year_vec, ".csv", sep = "")

codes <- fread(country_path,
               sep =  ",", header = TRUE)


country_codes <- c("BDI", "BHU", "BUR", "CAF", "CAM", "CGO", "CIV", "CMR", "COD",
                   "CPV", "CUR", "DJI", "ERI", "FID", "FIE", "GAB", "GUM", "Ind",
                   "IVC", "KOR", "KOS", "KSA", "LAO", "LBN", "LBR", "LCA", "LES",
                   "MDV", "MTN", "NET", "NRU", "OMA", "PLW", "ROU", "SCG", "SGP",
                   "SLE", "SOL", "SSD", "STP", "SWZ", "TLS", "TPE", "TTO")

countries <- c("Berundi", "Bhutan", "Burkina Faso", "Central African Republic", "Cambodia",
            "Republic of the Congo", "Cote d'Ivoire", "Cameroon", "Democratic Congo",
            "Cape Verde", "Curaçao", "Djibouti", "Eritrea", "Finland", "FIE", "Gabon",
            "Guam", "India", "Côte d'Ivoire", "South Korea", "Kosovo", "Saudi Arabia",
            "Laos", "Lebanon", "Liberia", "Saint Lucia", "Lesotho", "Maldives", "Mauritania",
            "NET", "Nauru", "Oman", "Palau", "Romania", "Serbia and Montenegro", "Singapore",
            "Sierra Leonne", "Solomon Islands", "South Sudan", "Sao Tome and Principe",
            "Swaziland", "East Timor (Timor-Leste)  ", "Taiwan", "Trinidad and Tobago")
```

Ugly data cleaning

This is the ugly part of the document: a `for loop` that is really meaty. Essentially, we will iterate all of the FIDE datasets to adjust each data's columns and values. We need to do this because want common values to merge the data later.

You are free to use `data.table`'s `rbindlist()` function to merge all of the datasets within `FIDE`, but I chose not to in this file.

```
for(i in 1:length(FIDE)){
  colnames(FIDE[[i]])[grepl("Name|NAME|name", colnames(FIDE[[i]]))] <- "Name"
  colnames(FIDE[[i]])[grepl("NUMBER", colnames(FIDE[[i]]))] <- "ID_Number"
  colnames(FIDE[[i]])[grepl("Fed|FED|COUNTRY", colnames(FIDE[[i]]))] <- "Country"
  colnames(FIDE[[i]])[grepl("Gms|GAMES|GM|Game|GAME", colnames(FIDE[[i]]))] <- "Games"
  colnames(FIDE[[i]])[grepl("K", colnames(FIDE[[i]]))] <- "K_factor"
  colnames(FIDE[[i]])[grepl("FLAG|Flag|flag", colnames(FIDE[[i]]))] <- "Activity"
  colnames(FIDE[[i]])[colnames(FIDE[[i]]) %in% c("Wtit","wtit","WTIT", "WTit")] <- "Womens_Title"
  colnames(FIDE[[i]])[colnames(FIDE[[i]]) %in% c("TITLE","Title","title","Tit")] <- "Title"
  colnames(FIDE[[i]])[grepl(string, colnames(FIDE[[i]]))] <- "Rating"
  colnames(FIDE[[i]])[grepl("Born|Age|age|BIRTHDAY|B-day|Bday", colnames(FIDE[[i]]))] <- "Age_Birthday"
  colnames(FIDE[[i]])[grepl("SEX", colnames(FIDE[[i]]))] <- "Sex"
  colnames(FIDE[[i]])[grepl("FOA", colnames(FIDE[[i]]))] <- "FIDE_Online_Arena"
  colnames(FIDE[[i]])[grepl("OTit", colnames(FIDE[[i]]))] <- "Other_Titles"
  FIDE[[i]] <- FIDE[[i]] %>%
               mutate(Date = as.POSIXct(names(FIDE)[i], format="%Y-%m-%d"),
                      Date_numeric = year(Date)+yday(Date)/366,
                      Rating = as.numeric(Rating),
                      Title= c(new, Title)[match(Title, c(old, Title))],
                      Country = c(codes$Country, Country)[match(Country, c(codes$Code, Country))],
```

```r
                      Country = c(countries, Country)[match(Country, c(country_codes, Country))])%>%
                      filter(!Country %in% c("Fed", "Col"))%>%
                      select(-one_of("V1"))
  fwrite(FIDE[[i]] , file = paste(dest, files[i], sep = ""), sep = "*")
}
```

Example data after modifications

Name	Country	Rating	Title	Date
A C J John	India	1063		2019-12-01
A Chakravarthy	India	1151		2019-12-01
A E M, Doshtagir	Bangladesh	1840		2019-12-01
A hamed Ashraf, Abdallah	Egypt	1728		2019-12-01
A Hamid, Harman	Malaysia	1325		2019-12-01
A K M, Sourab	Bangladesh	1598		2019-12-01

```r
list.files(path = dest, pattern = "*.csv")%>%head()
```

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

Analysis

January 09, 2020

The purpose of this document is to visually analyze all of the FIDE data files collected in the previous step's folder.

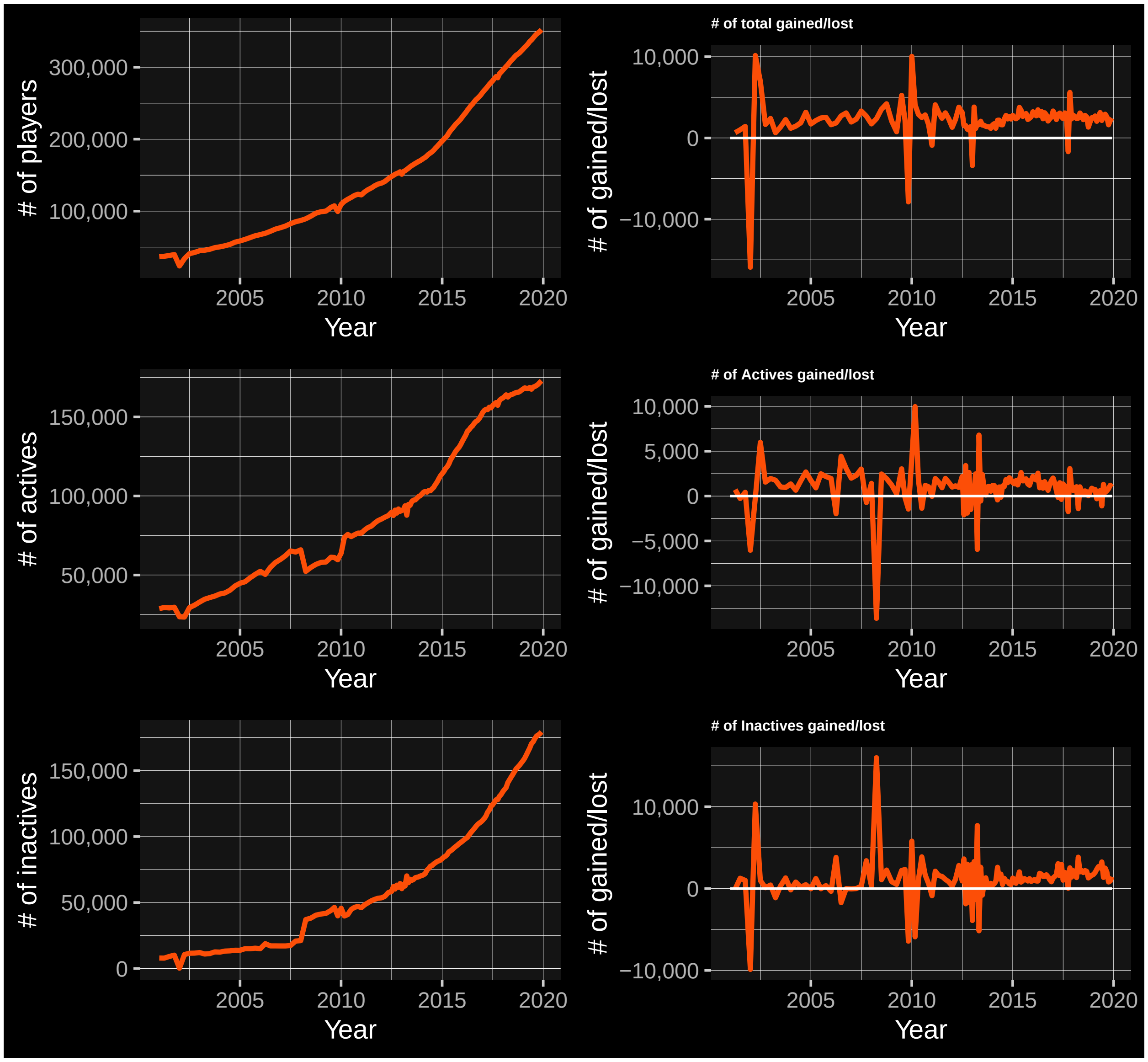
Irregular values by year

Date	# of irregular values
2002.249	7435
2002.003	5455
2001.003	306
2001.249	305
2001.497	304
2005.497	244
2005.003	194
2004.751	170
2004.500	136
2004.251	114

As we can see from the table above, most of irregular values in the files come from early on (2001 - 2005) rather than the latest files.

I'll look to address many of the values in the early datasets eventually. For now though, over 99.9% of the data is interpretable.

Total player count over time

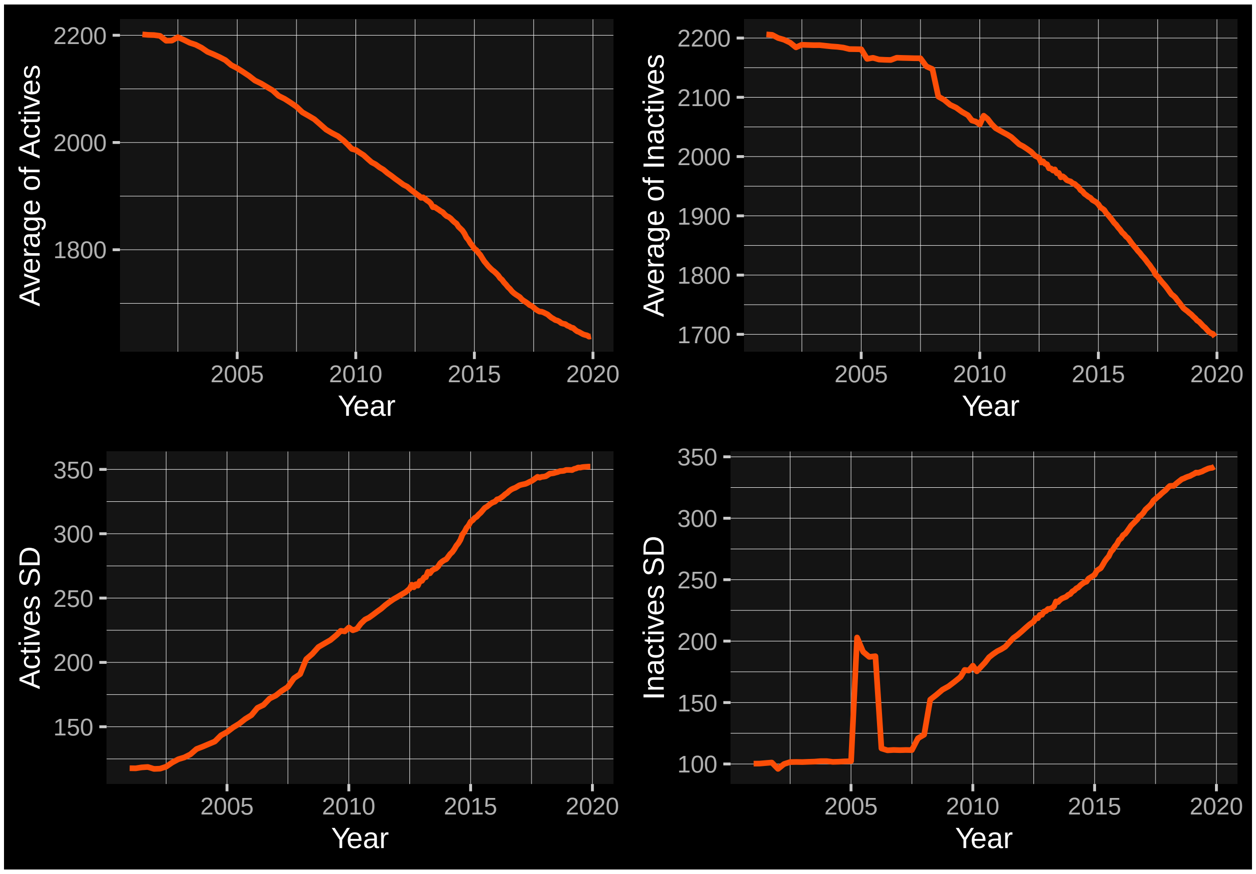


The four charts above reveal the total count of FIDE's members over time. Chart `A` shows a smooth gradual increasing growth curve among total players. This may lead you to believe that more chess players are playing tournaments, but a player's **activity** is a better metric to go by.

Activity is defined by if a given FIDE player had played a rated FIDE game within the past 12 months. If we take this into account, charts `C` and `E` show how the total active & inactive player count increase over time. Both charts show a fairly linear trend over time, but from 2007 to 2010 in each graph, there was a noticeable drop off in the active player base and increase in the inactive player base. I have my doubts on if there is faulty data here because chart `A` shows no irregularity during that time. This dip may be due to the economic crash during that time, but I need to do more exploration on this topic before making any definitive statements. Exploring variation by country may also be worth doing.

Charts `B`, `D` and `F` show the number of players gained and lost over time. The most relevant of the 3 graphs is `B` which shows several instances where total player counts dropped off. I genuinely don't if the data is faulty because of my doing or if FIDE is providing incomplete data sets based off of the charts.

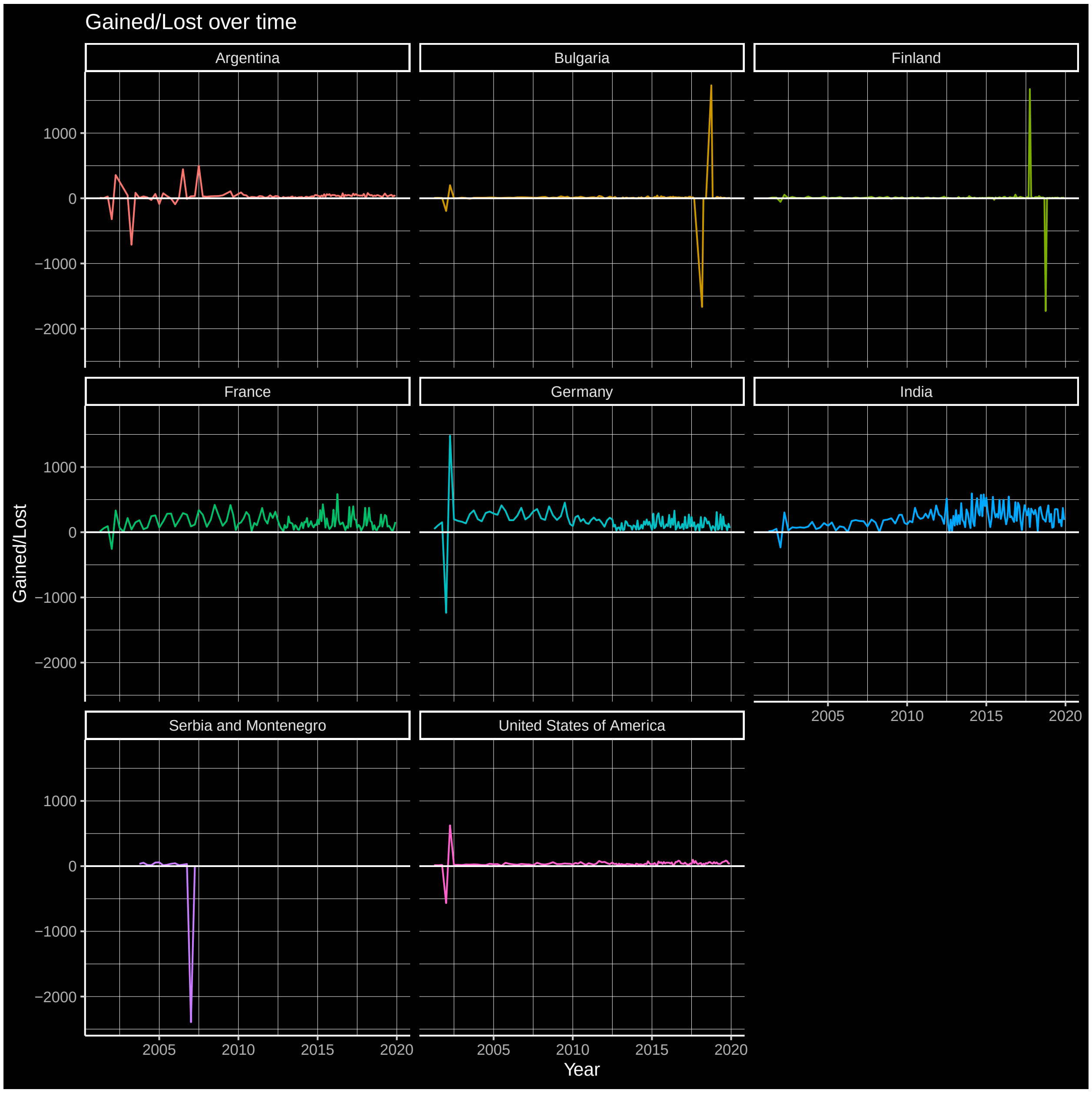
Rating stability over time (misleading)



Each chart above shows how the Active and Inactive player's average rating and rating standard deviation have progressed over time. For the most part, it is a meaningless metric because FIDE has brought in more **lower** rated chess players into the player pool over time.

This explains the steady decrease in average rating and increase in standard deviation over time.

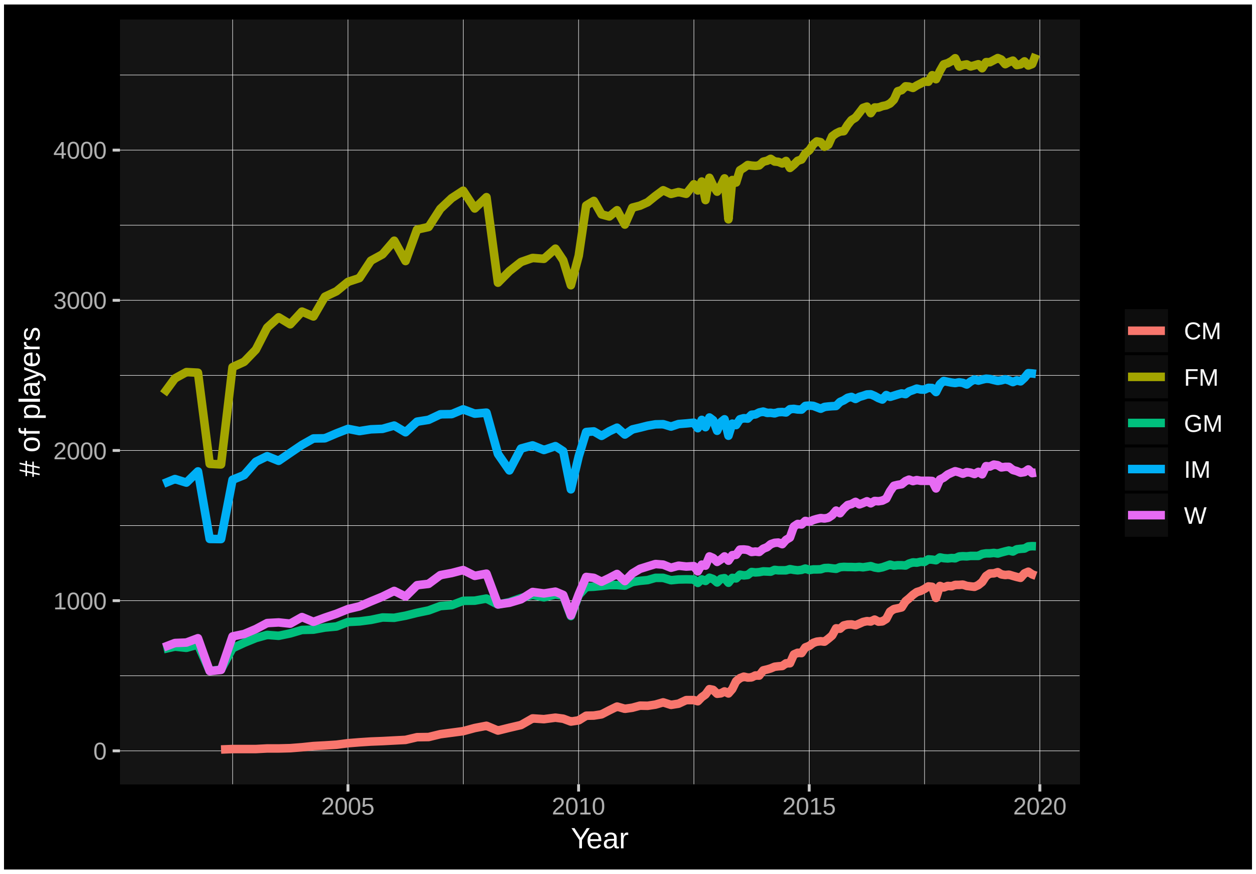
Which countries have seen the greatest changes in player counts?



A consistent theme we will see in every graph is a potential corruption of data. The graph above displays the of players gained and lost over time by country.

- Bulgaria, Finland, Germany, Serbia and Montenegro, and the United State of America saw unusual spikes & dips at various junctures (2002, 2007, 2017)

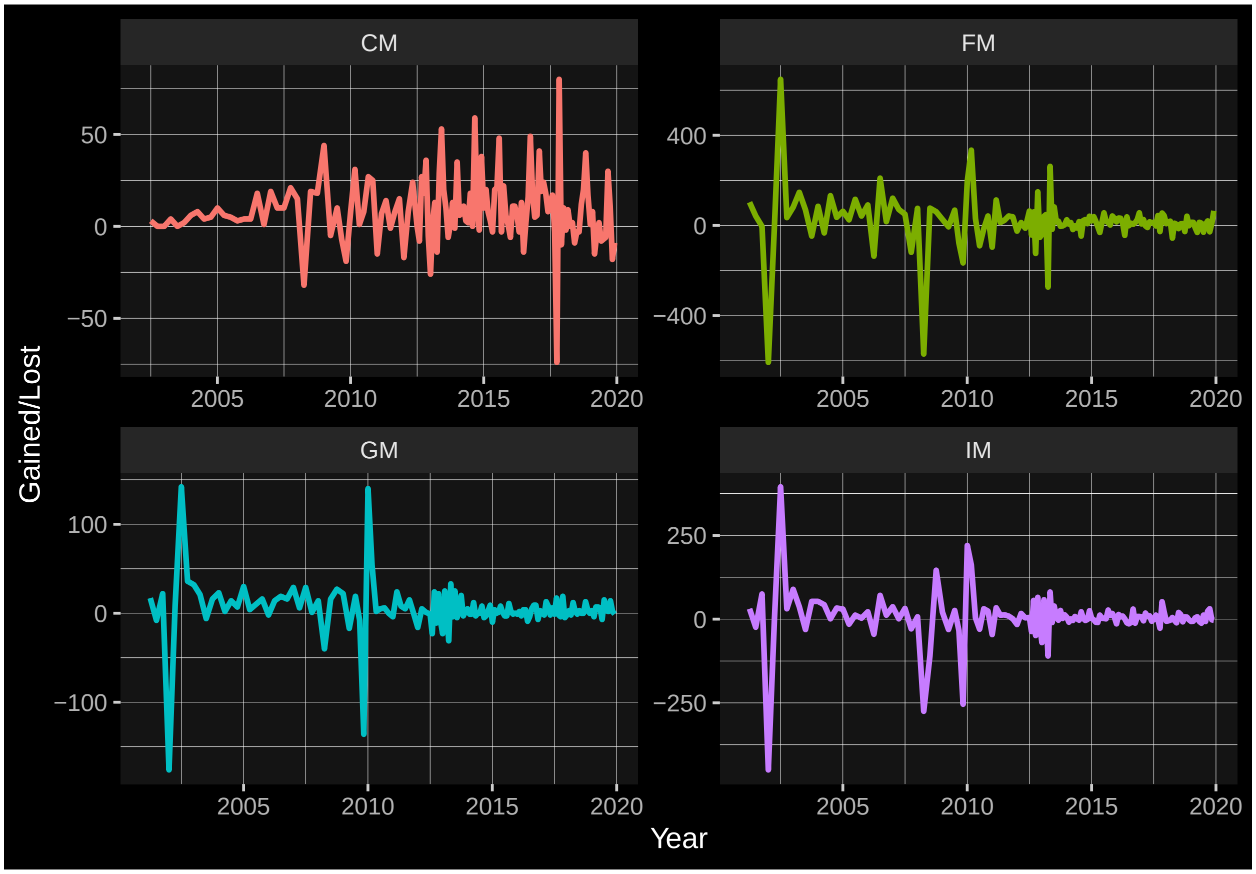
Titled player count over time



The plot above shows a few interesting trends (W = All women's titles):

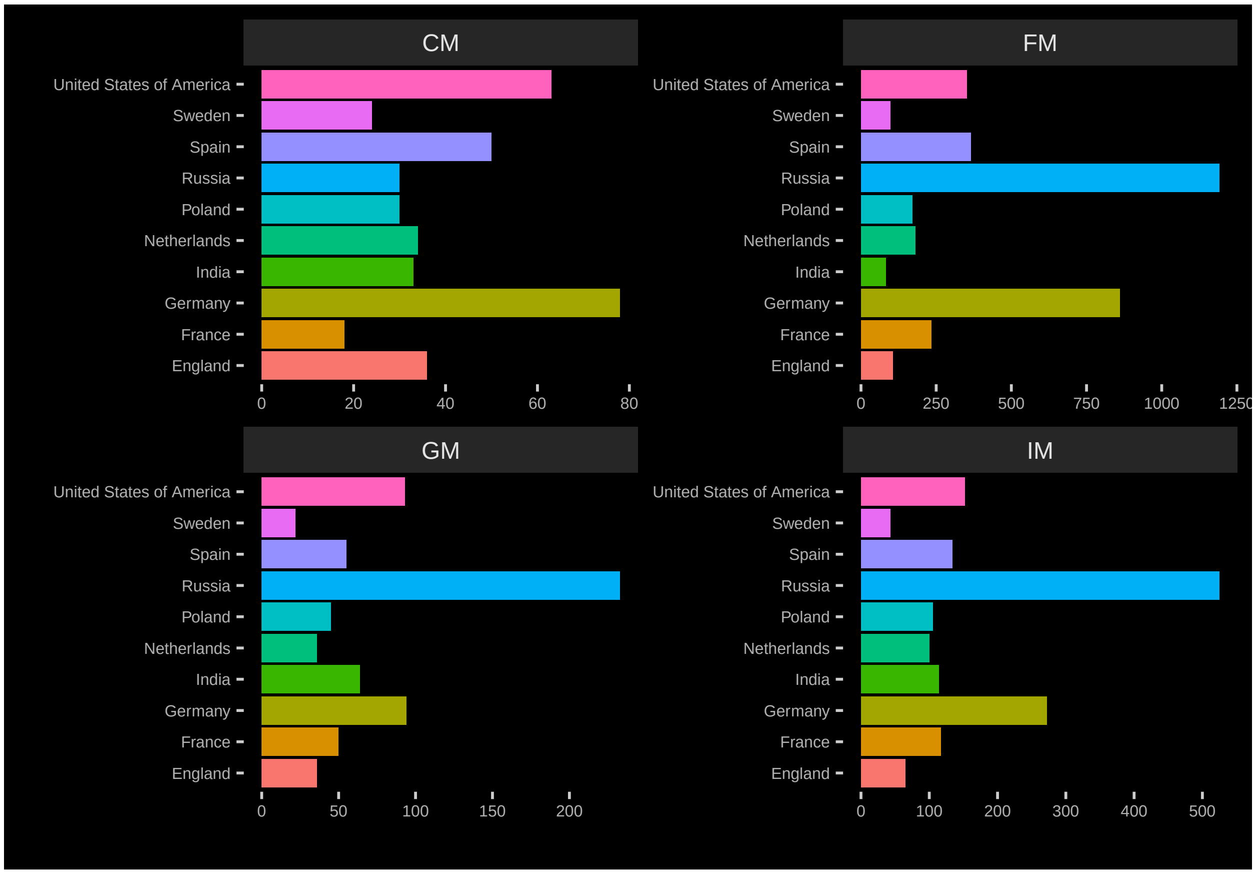
- All titled player groups have significantly increased in size over the past 20 years
- Around both 2002 and 2007, there were significant dips in all categories, except in `GM`s and `CM`s.
- There are about as many `GM`s as `CM`s. It goes to show how stigmatized the CM title is.

Titled players gained/lost over time



The graph above is graphs the rolling difference of titled player counts over time. As we saw before, there are a few noticeable dips in the graph. Most notably they occur around 2002, 2008 and 2010 in varying degrees.

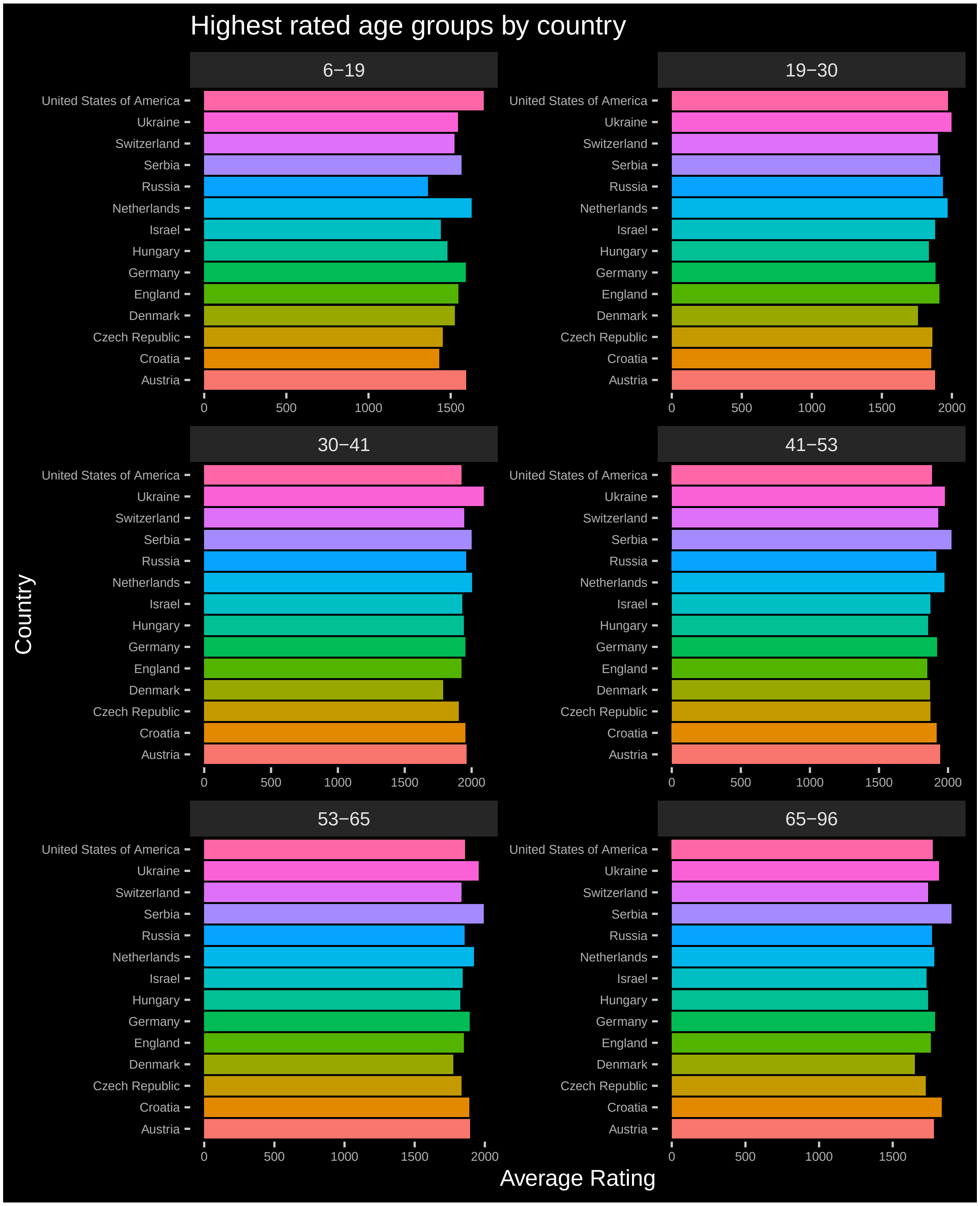
Titled player counts by country



Notable observations:

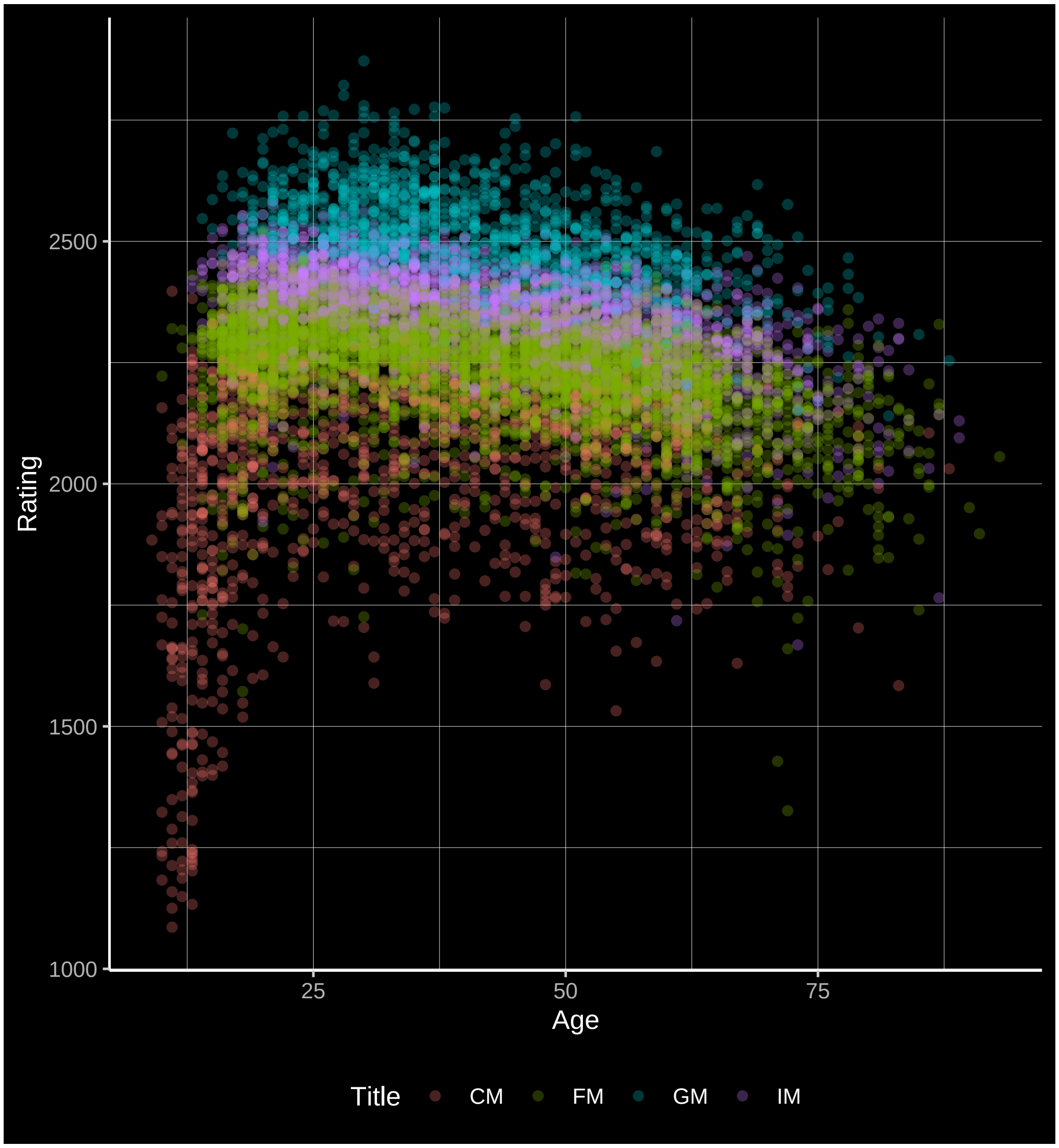
- Clearly, among `FM`, `IM` and `GM` players, Russia has the most titled players by a large margin. This is not surprising because of Russia's longstanding history (USSR and Soviet Union) of chess.
- Germany is a clear 2nd amongst `FM`, `IM` and `GM` players. They also lead the pack in `CM` players. This is surprising to me since I've never thought of Germany as being a bastion of strong players.
- Every other country lags behind these top-tier powerhouses.

Strongest countries by age group (Work in progress)



This plot would be substantially better at showing relative strengths among countries if I could scale it differently. This will adjusted in future iterations, probably through scaling.

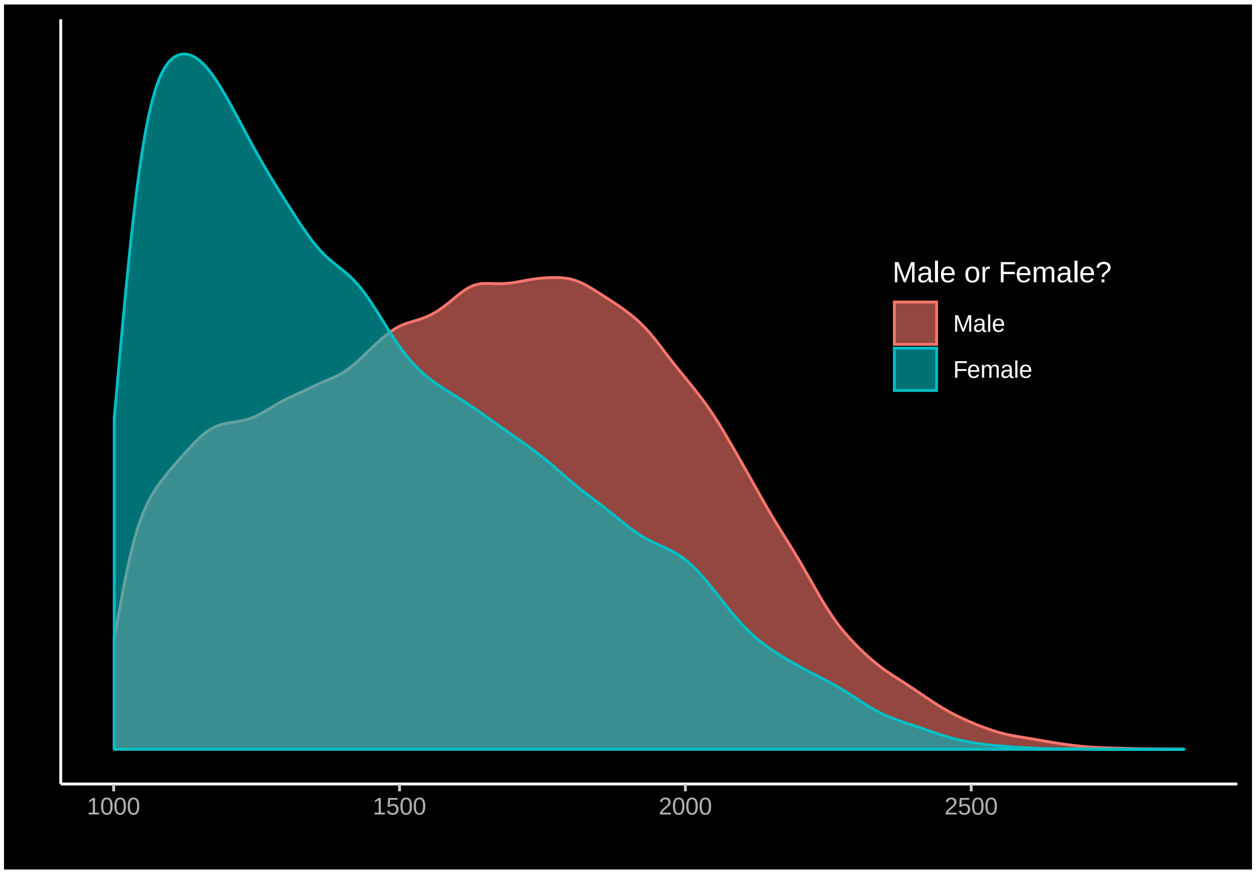
Age vs Rating of titled players (December 2019)



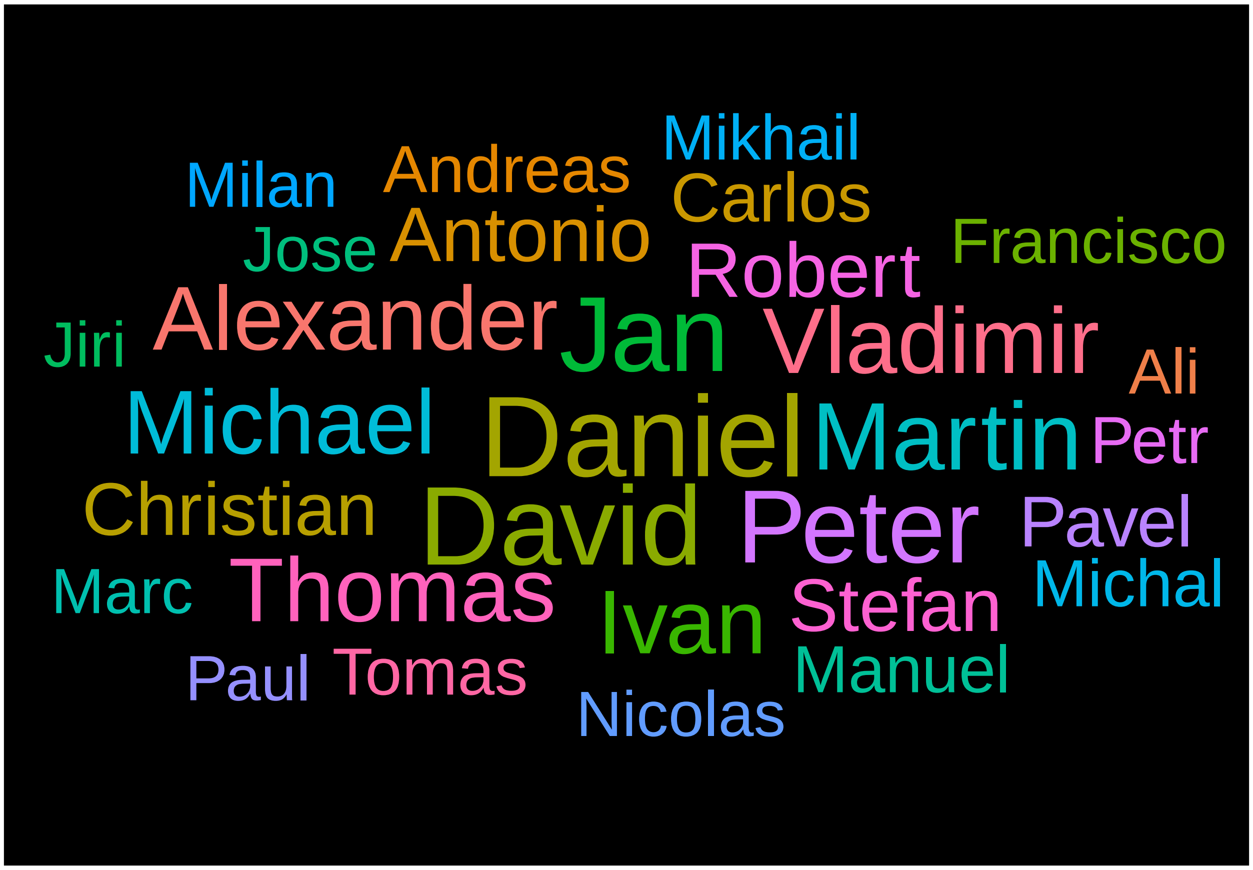
The most aesthetically pleasing graph to look at is the one above. It reveals a few aesthetically pleasing observations:

- Bands of players can be separated by titles categories: an obvious points is that the top blue band is all GMs, the highest rated group. Below the blue band are IMs (purple band), FMs (green band) and CMs (red band).
- CMs and FMs vary greatly across rating categories because it has become much easier for lower rated players to acquire titles in youth tournaments and via inter zonal tournaments.
- There is a slight negative correlation, among all titled player groups, between Age and Rating.

Males/Females rating distributions (December 2019)



Above is a plot showing the stark difference between male and female rating distributions for the previous month. I'll look to explore why the female rating distribution is sharply skewed right compared to the male counterpart.


Mikhail
Milan
Andreas
Carlos
Jose
Antonio
Francisco
Robert
Jiri
Alexander
Jan
Vladimir
Ali
Michael
Daniel
Martin
Petr
Christian
David
Peter
Pavel
Marc
Thomas
Michal
Ivan
Stefan
Paul
Tomas
Manuel
Nicolas

Andersson
Martin
Johansson
Horvath
Garcia
Toth
Schmidt
Mueller
Szabo
Silva
Nielsen
Wang
Larsen
Kim
Chen
Hansen
Lee
Tan
Liu
Nguyen
Li
Andersen
Pedersen
Jensen
Ivanov
Christensen
Zhang
Nagy
Sorensen
Da Silva