Lab 4: Purrr and a Bit of Dplyr

Name: Rufus petrie

This week's agenda: practicing how to use the map family of functions from `purrr`, and how to perform basic computations on data frames using `dplyr`.

Installing and loading packages

Below we install `tidyverse` which gives us the packages we need (`purrr` and `dplyr`) needed to complete this lab. We also install the `repurrrsive` package which has the Game of Thrones data set that we'll use for the first couple of questions. Since this may be the first time installing packages for some of you, we'll show you how. If you already have these packages installed, then you can of course skip this part. Note: *do not remove* `eval=FALSE` *from the above code chunk*, just run the lines below in your console. You can also select "Tools" –> "Install Packages" from the RStudio menu.

```
install.packages(tidyverse)
install.packages(repurrrsive)
```

Now we'll load these packages. Note: the code chunk below will cause errors if you try to knit this file without installing the packages first.

Post edit. Loading the `tidyverse` package in its entirety includes `plyr`, and this can cause namespace issues with the `dplyr` package. Better to just load only what you need.

```
library(purrr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(repurrrsive)
```

Game of Thrones data set

Below we inspect a data set on the 30 characters from Game of Thrones from the `repurrrsive` package. It's stored in a list called `got_chars`, which is automatically loaded into your R session when you load the 'repurrrsive" package.

```
class(got_chars)
```

```
## [1] "list"
```

```
length(got_chars)
```

```
## [1] 30
```

```
names(got_chars[[1]])
```

```
##  [1] "url"         "id"          "name"        "gender"      "culture"
##  [6] "born"        "died"        "alive"       "titles"      "aliases"
## [11] "father"      "mother"      "spouse"      "allegiances" "books"
## [16] "povBooks"    "tvSeries"    "playedBy"
```

```
got_chars[[1]]$name
```

```
## [1] "Theon Greyjoy"
```

```
got_chars[[1]]$aliases
```

```
## [1] "Prince of Fools" "Theon Turncloak" "Reek"            "Theon Kinslayer"
```

Q1. Warming up with map

- **1a.** Using the map functions from the `purrr` package, extract the names of the characters in `got_chars` so that you produce a character vector of length 30. Do this four different ways: (i) using `map()`, defining a custom function on-the-fly, and casting the resulting list into an appropriate data structure; (ii) using one of the `map_***()` functions, but still defining a custom function on-the-fly; (iii) using one of the `map_***()` functions, and using one of `` `[`() `` or `` `[[`() `` functions, as well as an additional argument; (iv) using one of the `map_***()` functions, and passing a string instead of a function (relying on its ability to define an appropriate extractor accordingly).

  Store each of the results in a different vector and check that they are all identical.

```
# YOUR CODE GOES HERE
```

- **1b.** Produce an integer vector that represents how many allegiances each character holds. Do this with whichever map function you'd like, and print the result to the console. Then use this (and your a saved object from the last question) to answer: which character holds the most allegiances? The least?

```
# YOUR CODE GOES HERE
```

- **1c.** Run the code below in your console. What does it do?

  ```
  1:5 %in% 3:6
  ```

  Using the logic you can infer about the `%in%` operator (you can also read its help file), craft a single line of code to compute a Boolean vector of length 6 that checks whether the first Game of Thrones character, stored in `got_chars[[1]]`, has appeared in each of the 6 TV seasons. Print the result to the console.

```
# YOUR CODE GOES HERE
```

- **1d.** Run the two lines of code below in their console. What do they do?

  ```
  rbind(1:5, 6:10, 11:15)
  do.call(rbind, list(1:5, 6:10, 11:15))
  ```

  Using the logic you can infer about the `do.call()` function (you can also read its help file), as well as the logic from the last question, complete the following task. Using `map()`, a custom-defined function, as well as some post-processing of its results, produce a matrix that has dimension 30 x 6, with each column representing a TV season, and each row a character. The matrix should have a value of `TRUE`

in position (i,j) if character i was in season j, and `FALSE` otherwise. Print the first 6 rows of the result to the console.

```
# YOUR CODE GOES HERE
```

- **Challenge.** Repeat the same task as in the last question, but using `map_df()` and no post-processing. The result will now be a data frame (not a matrix). Print the first 6 rows of the result to the console. Hint: `map_dfr()` will throw an error if it can't infer column names.

```
# YOUR CODE GOES HERE
```

Q2. Cultural studies

- **2a.** Using `map_dfr()`, create a data frame of dimension 30 x 5, whose columns represent, for each Game of Thrones character, their name, birth date, death date, gender, and culture. Store it as `got_df` and print the last 3 rows to the console.

```
# YOUR CODE GOES HERE
```

- *2b.* Using `got_df`, show that you can compute whether each character is alive or not, and compare this to what is stored in `got_chars`, demonstrating that the two ways of checking whether each character is alive lead to equal results.

```
# YOUR CODE GOES HERE
```

- **2c.** Using `filter()`, print the subset of the rows of `got_df` that correspond to Ironborn characters. Then print the subset that correspond to female Northmen.

```
# YOUR CODE GOES HERE
```

- **2d.** Create a matrix of dimension (number of cultures) x 2 that counts how many women and men there are in each culture appearing in `got_df`. Print the results to the console. Hint: what happens if you pass `table()` two arguments?

```
# YOUR CODE GOES HERE
```

- **2e.** Using `group_by()` and `summarize()` on `got_df`, compute how many characters in each culture have died. Which culture—aside from the unknown category represented by " "—has the most deaths?

```
# YOUR CODE GOES HERE
```

Rio Olympics data set

This is data set from the Rio Olympics data set that we saw in Lab 3. In the next question, we're going to repeat some calculations from Lab 3 but using `dplyr`.

```
rio = read.csv("http://www.stat.cmu.edu/~ryantibs/statcomp/data/rio.csv")
```

Q3. Practice with grouping and summarizing

- **3a.** Using `group_by()` and `summarize()`, compute how many athletes competed for each country in the `rio` data frame? Print the results for the first 10 countries to the console. Building off your here answer, use an additional call to `filter()` to compute which country had the most number of athletes and how many that was. Hint: consider using `n()` from the `dplyr` package for the first part here.

```
# YOUR CODE GOES HERE
```

- **3b.** Using `group_by()`, `summarize()`, and `filter()`, compute which country had the most numnber of total medals and many that was.

```
# YOUR CODE GOES HERE
```

- **3c.** Using `group_by()`, `summarize()`, and `filter()`, compute which country—among those with zero total medals—had the most number of athletes. Hint: you will need to modify your `summarize()` command to compute the number of athletes; and you might need two calls to `filter()`.

```
# YOUR CODE GOES HERE
```

- **3d.** Using—yes, you guessed it—`group_by()`, `summarize()`, and `filter()`, compute the average weight of athletes in each sport, separately for men and women, and report the two sport with the highest average weights (one for each of men and women). Hint: `group_by()` can accept more than one grouping variable. Also, consider using `na.rm=TRUE` as an additional argument to certain arithmetic summary functions so that they will not be thrown off by `NA` or `NaN` values.

```
# YOUR CODE GOES HERE
```

Fastest 100m sprint times

Below, we read two data sets of the 1000 fastest times ever recorded for the 100m sprint, in men's and women's track. We scraped this data from http://www.alltime-athletics.com/m_100ok.htm and http://www.alltime-athletics.com/w_100ok.htm, in early September 2021. (Interestingly, the 2nd, 3rd, 4th, 7th, and 8th fastest women's times were all set at the most recent Tokyo Olympics, or after! Meanwhile, the top 10 men's times are all from about a decade ago.)

```
sprint.m.df = read.table(
  file="http://www.stat.cmu.edu/~ryantibs/statcomp/data/sprint.m.txt",
  sep="\t", quote="", header=TRUE)
sprint.w.df = read.table(
  file="http://www.stat.cmu.edu/~ryantibs/statcomp/data/sprint.w.txt",
  sep="\t", quote="", header=TRUE)
```

More practice with data frame computations

- **4a.** Confirm that both `sprint.m.df` and `sprint.w.df` are data frames. Delete the `Rank` column from each data frame, then display the first and last 3 rows of each.

```
# YOUR CODE GOES HERE
```

- **4b.** Recompute the ranks for the men's data set from the `Time` column and add them back as a `Rank` column to `sprint.m.df`. Do the same for the women's data set. After adding back the rank columns, print out the first 10 rows of each data frame, but only the `Time`, `Name`, `Date`, and `Rank` columns. Hint: consider using `rank()`.

```
# YOUR CODE GOES HERE
```

- **4c.** Using base R functions, compute, for each country, the number of sprint times from this country that appear in the men's data set. Call the result `sprint.m.counts`. Do the same for the women's data set, and call the result `sprint.w.counts`. What are the 5 most represented countries, for the men, and for the women? (Interesting side note: go look up the population of Jamaica, compared to that of the US. Pretty impressive, eh?)

```
# YOUR CODE GOES HERE
```

- **4d.** Repeat the same calculations as in last part but using `dplyr` functions, and print out again the 5 most represented countries for men and women. (No need to save new variables.) Hint: consider using `arrange()` from the `dplyr` library.

```
# YOUR CODE GOES HERE
```

- **4e.** Are there any countries that are represented by women but not by men, and if so, what are they? Vice versa, represented by men and not women? Hint: consider using the `%in%` operator.

```
# YOUR CODE GOES HERE
```

Q5. Practice with grouping

- **5a.** Using `dplyr` functions, compute, for each country, the fastest time among athletes who come from that country. Do this for each of the men's and women's data sets, and display the first 10 rows of the result.

```
# YOUR CODE GOES HERE
```

- **5b.** With the most minor modification to your code possible, do the same computations as in the last part, but now display the first 10 results ordered by increasing time. Hint: recall `arrange()`.

```
# YOUR CODE GOES HERE
```

- **5c.** Rewrite your solution in the last part using base R. Hint: `tapply()` gives probably the easiest route here. Note: your code here shouldn't be too much more complicated than your code in the last part.

```
# YOUR CODE GOES HERE
```

- **5d.** Using `dplyr` functions, compute, for each country, the quadruple: name, city, country, and time, corresponding to the athlete with the fastest time among athletes from that country. Do this for each of the men's and women's data sets, and display the first 10 rows of the result, ordered by increasing time. If there are ties, then show all the results that correspond to the fastest time. Hint: consider using `select()` from the `dplyr` library.

```
# YOUR CODE GOES HERE
```

- **5e.** Rewrite your solution in the last part using base R. Hint: there are various routes to go; one strategy is to use `split()`, followed by `lapply()` with a custom function call, and then `rbind()` to get things in a data frame form. Note: your code here will probably be more complicated, or at least less intuitive, than your code in the last part.

```
# YOUR CODE GOES HERE
```