Day5 Self Learning

Jian Li

1/19/2019

Relationship to base and plyr functions

```
library(purrr)
library(repurrrsive)
```

lapply() vs. purrr::map() These are the core mapping functions of base and purrr,respectively. They are "list in,list out". The main(only?) difference is access to purrr's shortcuts for indexing by name or position and for creating anonymous functions.

What R base can do:

```
lapply(got_chars[1:3],function(x) x[["name"]])
```

```
## [[1]]
## [1] "Theon Greyjoy"
##
## [[2]]
## [1] "Tyrion Lannister"
##
## [[3]]
## [1] "Victarion Greyjoy"
```

What purrr can do:

```
map(got_chars[1:3],"name")
```

```
## [[1]]
## [1] "Theon Greyjoy"
##
## [[2]]
## [1] "Tyrion Lannister"
##
## [[3]]
## [1] "Victarion Greyjoy"
```

```
map_dfr(got_chars[23:25],`[`,c("name","playedBy"))
```

```
## # A tibble: 3 x 2
##   name            playedBy
##   <chr>           <chr>
## 1 Jon Snow        Kit Harington
## 2 Aeron Greyjoy   Michael Feast
## 3 Kevan Lannister Ian Gelder
```

```
tibble::tibble(
  name=map_chr(got_chars[23:25],"name"),
  id=map_int(got_chars[23:25],"id")
)
```

```
## # A tibble: 3 x 2
##   name               id
##   <chr>           <int>
## 1 Jon Snow          583
## 2 Aeron Greyjoy      60
## 3 Kevan Lannister   605
```

mapply() vs. map2(),pmap():

when you need to iterate over 2 ore more vectors/lists in parallel,the base option is mapply(). Unlike the other apply functions, the first arguments is FUN, the function to apply, and the multiple vector inputs are provided "loose" via ....
For exactly two vector inputs, purrr has map2(), with all the usual tyep-specific variants. For an arbitrary number of vector inputs, use purrr pmap() or type-specific variants, with the inputs packaged in a list. A very handy special case is when the input is a data frame, in which case pmap_*() applies .f to each row.

```
nms<- got_chars[16:18] %>% map_chr("name")
birth<- got_chars[16:18] %>% map_chr("born")

df<- tibble::tibble(nms,
                    connector="was born",
                    birth)
pmap_chr(df,paste)
```

```
## [1] "Brandon Stark was born In 290 AC, at Winterfell"
## [2] "Brienne of Tarth was born In 280 AC"
## [3] "Catelyn Stark was born In 264 AC, at Riverrun"
```

aggregate() vs. dplyr::summarise()

consider a data frame, as opposed to a nested list. How do you split it into pieces, according to one or more factors, apply a function to the pieces, and combine the results?
Create a tiny excerpt of the Gapminder dataset that contains a bit of data for Canada and Germany. Load dplyr, now that we are more in the data frame world.

```
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(gapminder)
mini_gap<- gapminder %>% filter(country %in% c("Canada","Germany"),year>2000) %>% droplevels()
```

```
mini_gap %>% group_by(country) %>% summarise_at(vars(lifeExp,gdpPercap),mean)
```

```
## # A tibble: 2 x 3
##   country lifeExp gdpPercap
##   <fct>     <dbl>     <dbl>
## 1 Canada     80.2    34824.
## 2 Germany    79.0    31103.
```

by() vs tidyr::nest()

```
by_obj<- by(gapminder,
            gapminder$country,
            function(df) lm(lifeExp~year,data=df))
str(by_obj[1:2],max.level=1)
```

```
## List of 2
##  $ Afghanistan:List of 12
##   ..- attr(*, "class")= chr "lm"
##  $ Albania    :List of 12
##   ..- attr(*, "class")= chr "lm"
##  - attr(*, "dim")= int 2
##  - attr(*, "dimnames")=List of 1
```

```
library(tidyr)
nested_df<- gapminder %>% group_by(country,continent) %>% nest() %>% mutate(fit=map(data,~lm(lifeExp~yea
str(nested_df$fit[1:2],max.level=1)
```

```
## List of 2
##  $ :List of 12
##   ..- attr(*, "class")= chr "lm"
##  $ :List of 12
##   ..- attr(*, "class")= chr "lm"
```

What if you want to inspect the fits for Oceania? On the tidyverse side, where the fits live in a data frame that carriers country and continent info, we can use our usual techniques for filtering rows based on the data.

```
nested_df %>% filter(continent=="Oceania") %>% .$fit
```

```
## [[1]]
##
## Call:
## lm(formula = lifeExp ~ year, data = .x)
##
## Coefficients:
## (Intercept)         year
##   -376.1163       0.2277
##
##
## [[2]]
##
## Call:
## lm(formula = lifeExp ~ year, data = .x)
##
## Coefficients:
```

```
## (Intercept)         year
##   -307.6996       0.1928
```

Final form a data frame with all info

```
nested_df %>% mutate(coefs=map(fit,coef),
                     intercept=map_dbl(coefs,1),
                     slope=map_dbl(coefs,2)) %>%
  select(country,continent,intercept,slope)
```

```
## # A tibble: 142 x 4
##    country     continent intercept slope
##    <fct>       <fct>         <dbl> <dbl>
##  1 Afghanistan Asia          -508. 0.275
##  2 Albania     Europe        -594. 0.335
##  3 Algeria     Africa       -1068. 0.569
##  4 Angola      Africa        -377. 0.209
##  5 Argentina   Americas      -390. 0.232
##  6 Australia   Oceania       -376. 0.228
##  7 Austria     Europe        -406. 0.242
##  8 Bahrain     Asia          -860. 0.468
##  9 Bangladesh  Asia          -936. 0.498
## 10 Belgium     Europe        -340. 0.209
## # ... with 132 more rows
```

Tomorrow to-do list:

Start take a peak at **gganimate** package

Click me