Functions, loops and list-columns

Taking control of lists with purrr

bit.ly/kyiv-purrr

dperepolkin
dmi3k
ddrive.no
dmi3kno

Kyiv UseR! Group
15.08.2018

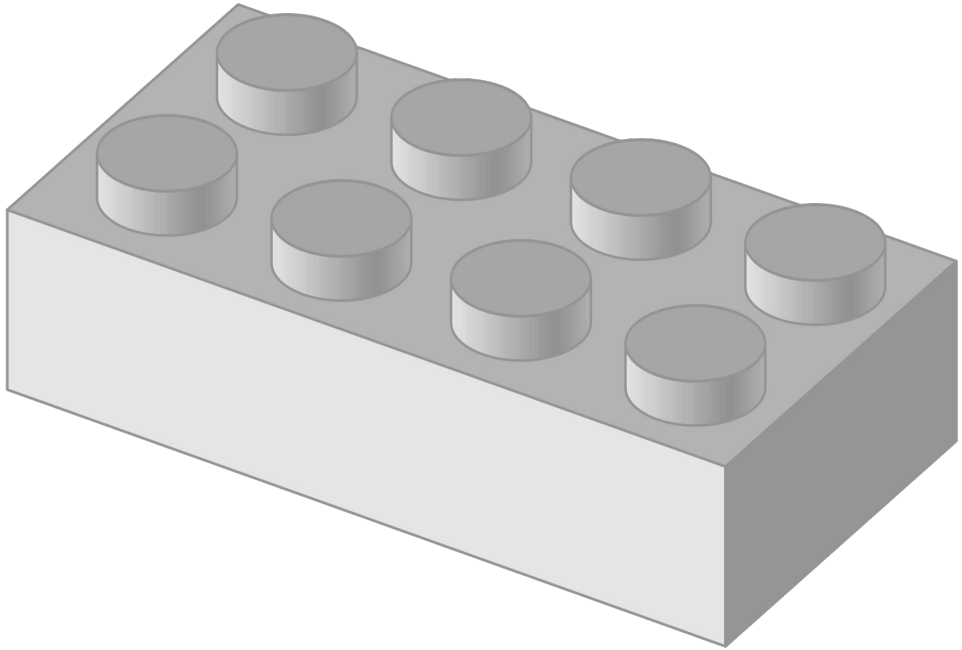
"If you find yourself copy-pasting a piece of code more than twice - it’s time to write a function.”

-- #rstats folklore

functions

Functions

1. Do one thing
2. Self-contained
3. Well-named
4. Data-tailored



```r
# nullcoalesce operator
`%||%` <- function(lhs, rhs) {
  if (!is.null(lhs) && length(lhs) > 0)
    lhs else rhs
}
```

Fully-connected neural network with identity activation function and zero hidden layers

1 Do one thing

hocr

```r
cheese_get_page <- function(i=NULL, page=NULL,
                            url=getOption("cheese.url")) {
  stopifnot(!is.null(url))

  if (!is.null(i) || !is.null(page)) {
    parsed_url <- httr::parse_url(url)
    parsed_url$query <- list(per_page = 100, i=i, page=page)
    url <- httr::build_url(parsed_url)
  }

  get_data(url)
}
```

2 Self-contained

cheese

```r
bbox_to_geometry <- function(x){
  coord_list <- lapply(strsplit(x, ",| "), as.numeric)
  sapply(coord_list, function(x){
    paste0(x[3]-x[1], "x", x[4]-x[2], "+", x[1], "+", x[2])
    })
}
```

3

Well-named

hocr

```
cheese_get_page <- function(i=NULL, page=NULL,
                            url=getOption("cheese.url")) {
  stopifnot(!is.null(url))

  if (!is.null(i) || !is.null(page)) {
    parsed_url <- httr::parse_url(url)
    parsed_url$query <- list(per_page = 100, i=i, page=page)
    url <- httr::build_url(parsed_url)
  }

  get_data(url)
}
```

4

Data-tailored

cheese

loops

Functional programming

Or, why for loops are "bad"

Instead of …

```
res <- vector(mode="double",
              length=ncol(df))

for(i in seq_along(df)){
  res[[i]] <- mean(df[[i]])
}
```

use…

```
library(purrr)
res <- map_dbl(df, mean)
```

```
map() map_if() map_at()
map_lgl() map_chr()
map_int() map_dbl()
map_raw() map_dfr()
map_dfc() walk()
```

```
map2() map2_lgl() map2_int()
map2_dbl() map2_chr()
map2_raw() map2_dfr()
map2_dfc() walk2() pmap()
pmap_lgl() pmap_int()
pmap_dbl() pmap_chr()
pmap_raw() pmap_dfr()
pmap_dfc() pwalk()
```

https://stackoverflow.com/questions/45101045/why-use-purrrmap-instead-of-lapply
https://purrr.tidyverse.org/reference/index.html

Embarrassingly parallel

```
tictoc::tic()
preproc_files <- list.files("data-raw/img", full.names = TRUE) %>%
   map_chr(img_preproc, save_to="data-raw/img_preproc", filter="Hamming")
tictoc::toc()
# 605.8 sec elapsed
```

Intel Core i5-6300U @ 2.40GHz + Compare

Average CPU Mark

Socket: *FCBGA1356*
Clockspeed: *2.4 GHz*
Turbo Speed: *3.0 GHz*
No of Cores: *2 (2 logical cores per physical)*
Typical TDP: *15 W*

4374

```
library(furrr)
plan(multiprocess)

tictoc::tic()
preproc_files <- list.files("data-raw/img", full.names = TRUE) %>%
 future_map_chr(img_preproc, save_to="data-raw/img_preproc", filter="Hamming")
tictoc::toc()
# 253.36 sec elapsed
```

```
plan(remote, workers = "IP ADDRESS HERE")
```

https://www.andrewheiss.com/blog/2018/07/30/disposable-supercomputer-future/

zoe

lists

miles[“releases”] vs miles[[“releases”]]

```
library(musicbrainz)

miles <- musicbrainz:::lookup_by
mbid="561d854a-6a28-4aa7-8c99-32
```

miles	list [17]	List of length 17
gender-id	character [1]	'36d3d30a-839d-3eda-8cb3-29be4384e4a9'
ipis	list [1]	List of length 1
gender	character [1]	'Male'
type-id	character [1]	'b6e035f4-3ce9-331c-97df-83397230b0df'
isnis	list [1]	List of length 1
country	character [1]	'US'
disambiguation	character [1]	'jazz trumpeter, bandleader, songwriter'
id	character [1]	'561d854a-6a28-4aa7-8c99-323e6ce46c2a'
releases	list [25]	List of length 25
name	character [1]	'Miles Davis'
life-span	list [3]	List of length 3
area	list [5]	List of length 5
end_area	list [4]	List of length 4
sort-name	character [1]	'Davis, Miles'
tags	list [17]	List of length 17
begin_area	list [4]	List of length 4
type	character [1]	'Person'

Tidying complex lists

```
lookup_artist_by_id <- function(mbid, includes=NULL) {
  available_includes <- c("recordings", "releases", "release-groups", "works", "tags")
  includes <- validate_includes (includes, available_includes)
  res <- lookup_by_id("artist", mbid, includes)
  parsers_df <- get_includes_parser_df(res, includes)

  # extract and bind
  res_df <- dplyr::bind_cols(
    purrr::map_dfr(get_main_parser_lst("artists"), ~ purrr::pluck(res, .x, .default = NA)),
    purrr::pmap_dfc(parsers_df, parse_includes)
  )
  res_df
}

parse_includes <- function(nm, lst_xtr, lst) {
  nm <- quo_name(nm)
  res_lst <- list(purrr::map_dfr(lst, ~ purrr::map(lst_xtr, function(i)
                                                    purrr::pluck(.x, i, .default = NA))))
  tibble::tibble(!!nm := res_lst)
}
```

parsers

musicbrainz

list-cols

Many models


```
library(tidyverse)
country_df <- gapminder::gapminder %>%
  group_by(country, continent) %>%
  nest() %>%
  mutate(model=map(data, ~lm(lifeExp ~ year, data=.x)),
         metrics=map(model, broom::glance))
#> # A tibble: 142 x 5
#>    country     continent data              model    metrics
#>   <fct>       <fct>     <list>            <list>   <list>
#> 1 Afghanistan Asia      <tibble [12 x 4]> <S3: lm> <tibble [1 x 11]>
#> 2 Albania     Europe    <tibble [12 x 4]> <S3: lm> <tibble [1 x 11]>
#> 3 Algeria     Africa    <tibble [12 x 4]> <S3: lm> <tibble [1 x 11]>
#> 4 Angola      Africa    <tibble [12 x 4]> <S3: lm> <tibble [1 x 11]>
#> 5 Argentina   Americas  <tibble [12 x 4]> <S3: lm> <tibble [1 x 11]>
# ... with 137 more rows

country_df %>%
  unnest(metrics) %>%
  arrange(adj.r.squared)
```

Nesting for relational data

```
search_artists("Onuka") %>%
  slice(1) %>%
  pull(mbid) %>%
  lookup_artist_by_id(includes=c("recordings", "releases", "release-groups", "works", "tags")) %>%
  select(name, releases, recordings, `release-groups`, works, tags)
#> # A tibble: 1 x 6
#>   name  releases          recordings        `release-groups` works            tags
#>   <chr> <list>            <list>            <list>           <list>           <list>
#> 1 Onuka <tibble [5 x 11]> <tibble [25 x 5]> <tibble [4 x 6]> <tibble [0 x 0]> <tibble [5 x 2]>
```

nest_join() is coming soon to dplyr.

Nesting joins create a list column of data.frames: nest_join() return all rows and all columns from x. Adds a list column of tibbles. Each tibble contains all the rows from y that match that row of x. When there is no match, the list column is a 0-row tibble with the same column names and types as y.

nest_join() is the most fundamental join since you can recreate the other joins from it. An inner_join() is a nest_join() plus an tidyr::unnest(), and left_join() is a nest_join() plus an unnest(drop = FALSE). A semi_join() is a nest_join() plus a filter() where you check that every element of data has at least one row, and an anti_join() is a nest_join() plus a filter() where you check every element has zero rows.

musicbrainz

https://stackoverflow.com/questions/45062277/join-across-within-nested-dataframes

adverbs

Traditional error handling `tryCatch`

```
…
content <- content %||% response$headers$`content-type`

res <- tryCatch(
  {
    httr::content(response, type = content)
  },
  error=function(cond){
    cat("<polite session> Encountered an error, while parsing content.\n",
        "There seems to be mismatch of content type or encoding or both.\n",
        "The server says it is serving: '", response$headers$`content-type`, "' \n",
        "But, please, do not despair! I will return a raw vector to you now,\n",
        "which you can parse with rawToChar(). Good luck!\n")
    return(httr::content(response, as = "raw"))
  }
)
return(res)
```

polite

Better error handling: Lionel Henry @ eRUM2018 https://www.youtube.com/watch?v=-v1tp41kizk

Error handling with adverbs `purrr::safely`

```
…
content <- content %||% response$headers$`content-type`

safe_content <- purrr::safely(httr::content)

res <- safe_content(response, type = content)

if (is.null(res$result)){
    cat("<polite session> Encountered an error, while parsing content.\n",
        "There seems to be mismatch of content type or encoding or both.\n",
        "The server says it is serving: '", response$headers$`content-type`, "' \n",
        "But, please, do not despair! I will r
        "which you can parse with rawToChar().
    return(httr::content(response, as = "raw")
  }

return(res$result)
```

polite

See also

`purrr::quietly()` - returns list(4) with result, output, messages and warnings.

`purrr::possibly()` - has argument otherwise

https://colinfay.me/purrr-adverb-tidyverse/

Links and references

Packages

- polite https://github.com/dmi3kno/polite
- hocr https://github.com/dmi3kno/hocr
- musicbrainz https://github.com/dmi3kno/musicbrainz
- cheese https://github.com/dmi3kno/cheese
- zoe https://github.com/dmi3kno/zoe

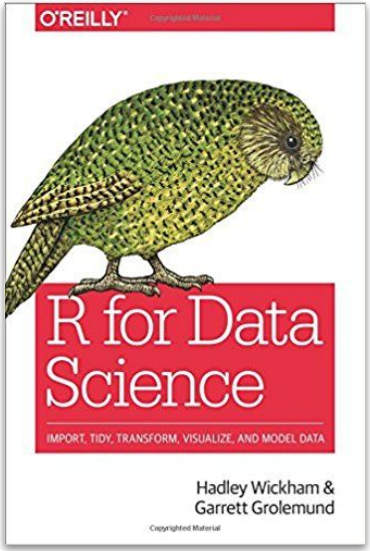
Learning from

https://jennybc.github.io/purrr-tutorial/

Inspired by

http://r4ds.had.co.nz

https://jrnold.github.io/r4ds-exercise-solutions/



Thank you!

dperepolkin
dmi3k
ddrive.no
dmi3kno

bit.ly/kyiv-purrr

Writing R package

- why write package

- anatomy of package

- usethis

- marketing your package and github collaboration