Tidying Up With ~~Marie Kondo~~ `purrr` et al.

or: How I Learned to Stop Worrying and Love the Tidyverse

Daniel Bride

Tidyverse Pt. II

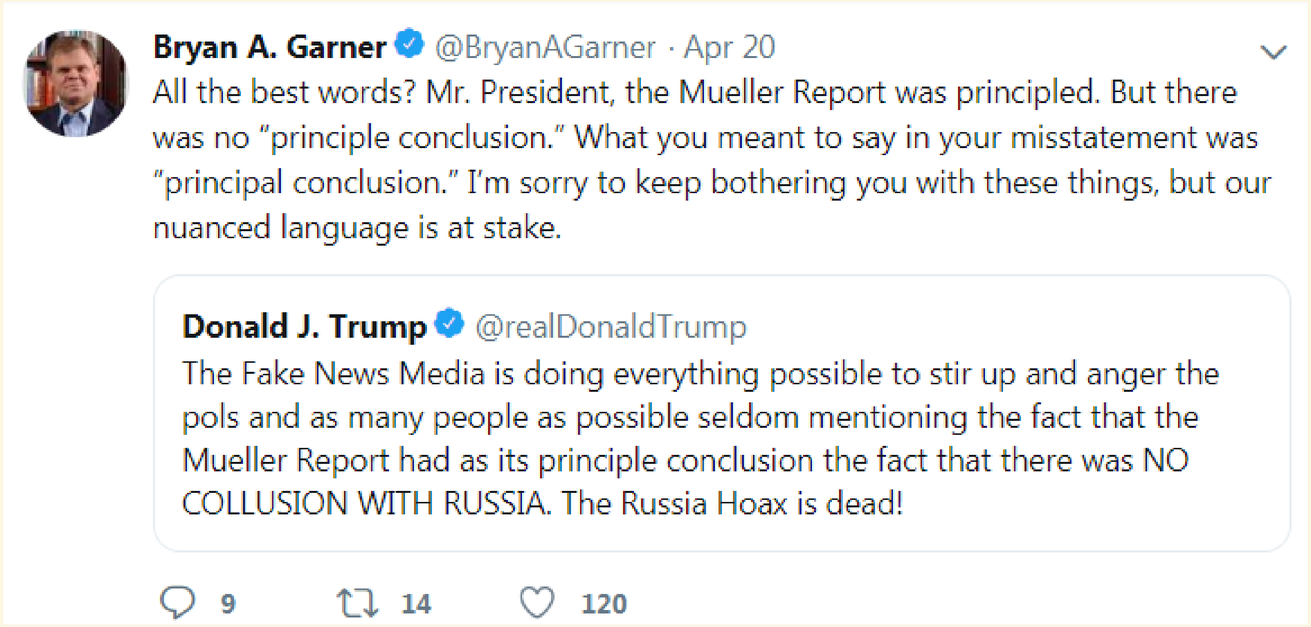
- Part I covers many cool packages
  - https://github.com/slc-rug/tidyverse-explore
- I'll only cover one cool package: `purrr`
- That's okay.

Roadmap

- Tidyverse philosophy lesson
- Apply functions with `map()`, `modify()`
- Modify functions with `safely()`, `quietly()`, `possibly()`
- Reduce vectors with `reduce()`, `accumulate()`

Interesting Diversion

An Impassioned Defense



Bryan A. Garner @BryanAGarner · Apr 20

All the best words? Mr. President, the Mueller Report was principled. But there was no "principle conclusion." What you meant to say in your misstatement was "principal conclusion." I'm sorry to keep bothering you with these things, but our nuanced language is at stake.

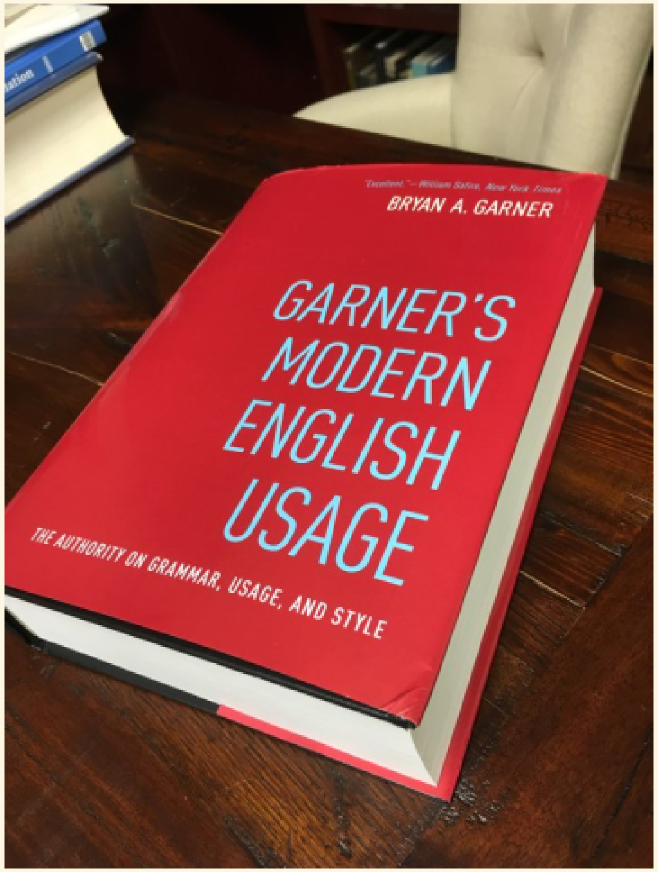
Donald J. Trump @realDonaldTrump

The Fake News Media is doing everything possible to stir up and anger the pols and as many people as possible seldom mentioning the fact that the Mueller Report had as its principle conclusion the fact that there was NO COLLUSION WITH RUSSIA. The Russia Hoax is dead!

9 14 120

Interesting Diversion

Garner’s other work


"Excellent." — William Safire, New York Times
BRYAN A. GARNER
GARNER'S MODERN ENGLISH USAGE
THE AUTHORITY ON GRAMMAR, USAGE, AND STYLE

Interesting(?) Diversion

Descriptive vs. Prescriptive Grammar

Descriptive:

- Rules derived from usage
- All usage equally valid
- Comprehensive

Prescriptive:

- Rules inform usage
- One correct/best usage
- Intentionally constrained

Tidyverse Philosophy

Why Are We Talking About Grammar?

- Prescriptivism is good for beginners
- Prescriptivism can outsource cognitive burden
- Prescriptivism can train a way of thinking
- The Tidyverse is prescriptive

Tidyverse Philosophy

Tidy Tendencies

- Reuse existing data structures
- Compose simple functions with the pipe (%>%)
- Embrace functional (vs. imperative) programming
- Design for humans

Reference: https://tidyverse.tidyverse.org/articles/manifesto.html

Learn to Love the Tidyverse

1. Try it
2. Ask why
3. Branch out confidently

The `purrr` Package (Finally!)

- A "functional programming toolkit"
- Built for complex, iterative tasks
- `install.packages("tidyverse")`

Apply Functions

`map()`

- Do something to/with each element of a list or vector
- Return a list* of results
- *Hold your horses. There are ways to return other types.

Apply Functions: `map()`

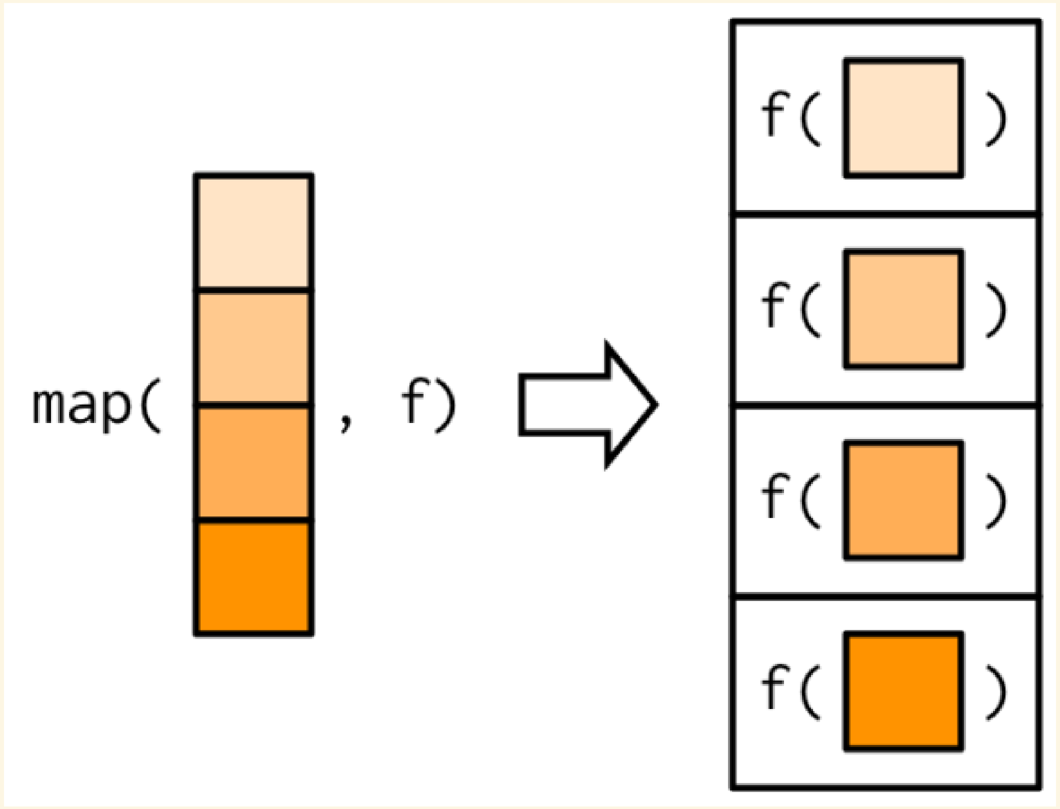
`map(.x, .f, ...)`

Do something to/with each element of a list or vector

- `.x`: A vector or list
- `.f`:
  - A function name or formula to apply a function iteratively, or
  - A vector or list to specify elements to extract from `.x`
- `...`: additional arguments passed to `.f`

```
map(.x, .f, ...)
```

Visually:



Reference: https://adv-r.hadley.nz/functionals.html

`map(.x, .f, ...)`

Three Ways to Specify Functions

- `purrr`-style anonymous function shortcut (`~ mean(.x)`)
- Named function (`.f = mean`)
- Full function definition (`.f = function(x) mean(x)`)

map(.x, .f, ...)

Let's make some unique example data

- Using `babynames` and `lexicon`
- Unpopular baby names, popular sayings, inscrutable internet slang

```r
baby_pool <-
  babynames::babynames %>%
  group_by(sex, first_letter = str_sub(name, 1L, 1L)) %>%
  top_n(-10, prop) %>%
  pull(name) %>%
  sort()

cliches <- lexicon::cliches

web_slang <-
  lexicon::hash_internet_slang
```

`map(.x, .f, ...)`

Write function with `purrr` shortcut

Organize baby names by first letter

```r
chosen_babies <-
  map(.x = LETTERS,
      .f = ~ {str_subset(baby_pool, paste0("^", .x))}) %>%
  set_names(LETTERS)
```

- ~ is shorthand for function(x) { }
- When specifying this way, use `.` or `.x` to refer to data

map(.x, .f, ...)

Write named function

Output first three names for each letter

- Notice we're using `...` to pass addtl argument (3) to `head()`

```
chosen_babies %>% map(head, 3)
```

```
## $A
## [1] "Aaban" "Aahna" "Aajah"
##
## $B
## [1] "Baani"   "Bacilio" "Badr"
##
## $C
## [1] "Cabella" "Cabrina" "Cace"
##
## $D
## [1] "Dacorian" "Daegan"   "Daemion"
##
## $E
## [1] "Eadie"   "Eanna"   "Earlene"
##
## $F
## [1] "Fabrisio" "Faelynn"  "Fahd"
```

##

map(.x, .f, ...)

Write full function definition

Match cliches to first letter

```
alphabetical_cliches <-
  LETTERS %>%
  map(function(ltr) {
    cliches_sub <- str_subset(cliches,
                              regex(paste0("^", ltr), ignore_case =
 TRUE))
    if(length(cliches_sub)) cliches_sub
    else "you're indescribable"}) %>%
  set_names(LETTERS)
```

...Pardon the Interruption...

We'll Use This Later

```r
everything <- list(babies = chosen_babies,
                   cliches = alphabetical_cliches,
                   web_slang = deframe(web_slang)[1:26])
```

map(.x, .f, ...)

Use map() to extract elements

- An integer n extracts nth item from each element of the list

```
alphabetical_cliches %>% map(15, .default = "NOTHIN' TO SEE HERE!")
```

```
## $A
## [1] "all fun and games"
##
## $B
## [1] "beat a dead horse"
##
## $C
## [1] "clear as a bell"
##
## $D
## [1] "don\\'t step on anyone\\'s toes"
##
## $E
## [1] "NOTHIN' TO SEE HERE!"
##
## $F
## [1] "fit as a fiddle"
##
```

map(.x, .f, ...)

Use map() to extract elements

- For named vectors/lists, a string extracts thus-named items from each

```
everything %>%
  map("Q", .default = "---No match---")
```

```
## $babies
##  [1] "Qualen"      "Qualon"      "Quamari"     "Quameir"     "Quanisha
##  [6] "Quanta"      "Quantasia"   "Quantay"     "Quashaun"    "Quasim"
## [11] "Quayvon"     "Quenia"      "Quentasia"   "Quindarrius" "Quiniya'
## [16] "Quinlen"     "Quinlynn"    "Quinnlan"    "Quinnlyn"    "Quinter
## [21] "Quintus"     "Quinya"      "Quion"       "Quynn"
##
## $cliches
## [1] "quick as a bunny"     "quick as a lick"      "quick as a wink"
## [4] "quick as lightning"   "quiet as a dormouse"
##
## $web_slang
## [1] "---No match---"
```

`map(.x, .f, ...)`

Use `map()` to extract elements

- Use multiple values to index multiple levels
- Allowing you to access deeply nested values
- Combine characters and integers in a list

```r
everything %>%
  map(list("Q", 3), .default = "***I got nothin'***")
```

```
## $babies
## [1] "Quamari"
##
## $cliches
## [1] "quick as a wink"
##
## $web_slang
## [1] "***I got nothin'***"
```

`map(.x, .f, ...)`

Returning other types

Obligatory `mtcars` example. Ugly version:

```
map(mtcars, mean)
```

```
## $mpg
## [1] 20.09062
##
## $cyl
## [1] 6.1875
##
## $disp
## [1] 230.7219
##
## $hp
## [1] 146.6875
##
## $drat
## [1] 3.596563
##
## $wt
## [1] 3.21725
##
```

`map(.x, .f, ...)`

Returning other types

- Just add a suffix: `map_*()`
- E.g., `map_int()`, `map_lgl()`, ...

Better version:

```
map_dbl(mtcars, mean)
```

```
##        mpg        cyl       disp         hp       drat         wt 
##  20.090625   6.187500 230.721875 146.687500   3.596563   3.217250 
##       qsec         vs         am       gear       carb 
##  17.848750   0.437500   0.406250   3.687500   2.812500
```

- Names are conveniently preserved

Apply Functions Across Two Lists/Vectors

`map2()`

- Do something with each element of two lists or vectors
- Return a list* of results
- *Or character vector (`map2_chr()`), or double vector (`map2_dbl()`), or…

Apply Functions Across Two Lists/Vectors

```
map(.x, .y, .f, ...)
```

Do something to/with each element of two lists/vectors

- `.x`: A vector or list
- `.y`: Another vector or list (length 1 or same as .x)
- `.f`: A function name or formula to apply a function iteratively
- `...`: additional arguments passed to `.f`

map2(.x, .y, .f, ...)

Celebrate the babies of the alphabet with alliterative cliches

```r
map2_chr(.x = chosen_babies,
         .y = alphabetical_cliches,
         ~ paste0(str_to_sentence(sample(.y, 1, TRUE)), ", ",
                  str_to_sentence(sample(.x, 1)), "!"))
```

```
##                                             A 
##                     "A loose cannon, Adayla!" 
##                                             B 
## "Bitten off more than he can chew, Baeleigh!" 
##                                             C 
##                "Cool as a cucumber, Cabella!" 
##                                             D 
##                "Dark before the dawn, Drean!" 
##                                             E 
##                      "Eleventh hour, Eilene!" 
##                                             F 
##    "For all intents and purposes, Folashade!" 
##                                             G 
##                "Go him one better, Galileah!" 
##                                             H 
##                       "High and dry, Haleia!" 
##                                             I 
##                    "In a nutshell, Itayetzi!" 
```

So, `map3()`?

Of course not!

Apply Functions With Many Arguments

`pmap()`

- Do something with each element of many lists/vectors
- Return a list or typed vector of results

Apply Functions With Many Arguments

`pmap(.l, .f, ...)`

Do something to/with each element of many lists/vectors

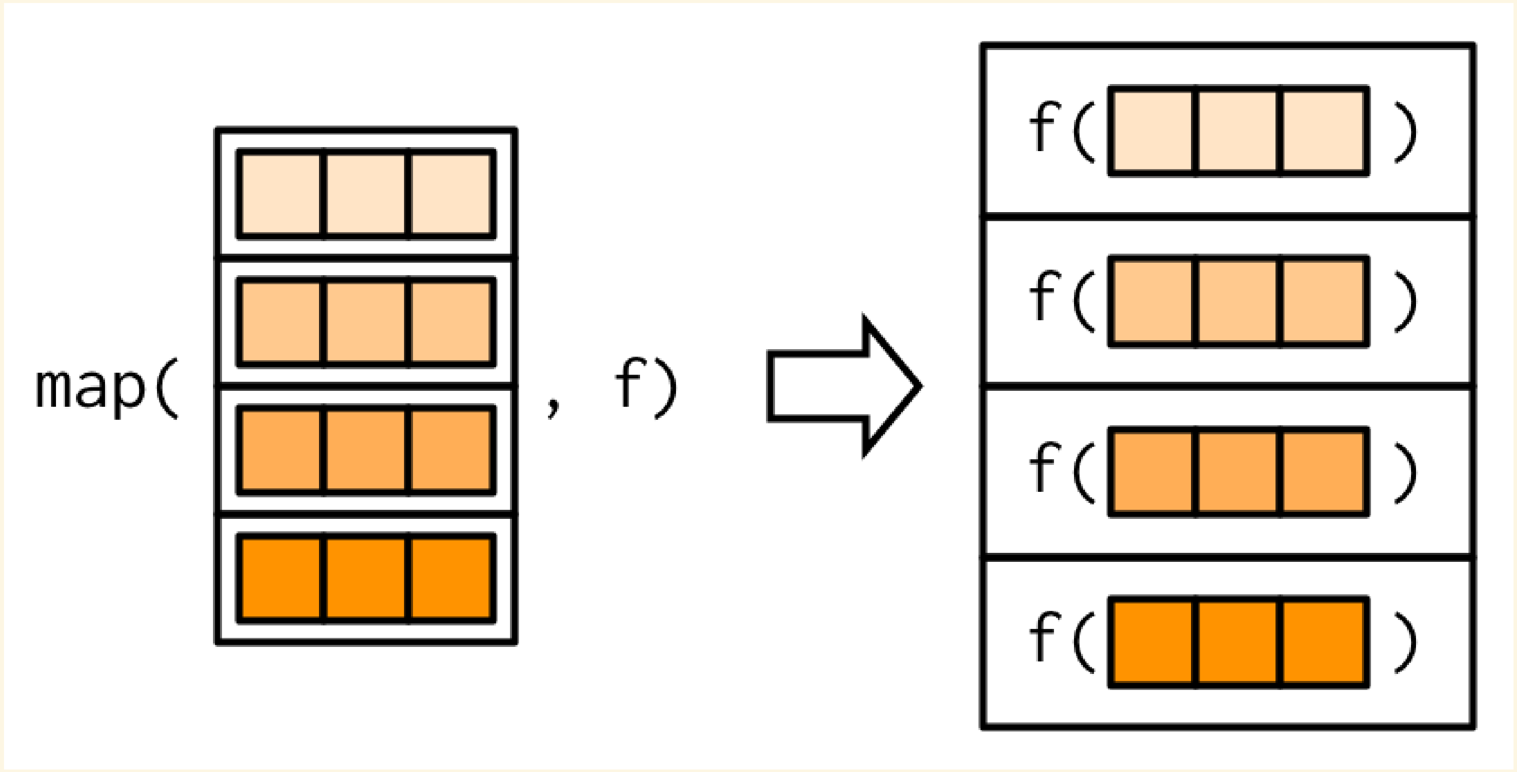
- `.l`: A **list** of vectors/lists to supply as arguments to a function
  - All elements generally same length or length 1
- `.f`: A function name or formula to apply a function iteratively
- `...`: additional arguments passed to `.f`

pmap(.l, .f, ...)

```r
pmap(.l = everything, .f = function(babies, cliches, web_slang, ...) {
  paste0(str_to_sentence(web_slang),
                         ", ",
                         str_to_sentence(sample(babies, 1, TRUE)),
                         "--",
                         sample(cliches, 1),
                         "!")
  })
```

```
## $A
## [1] "Too fast for you, Amarely--add insult to injury!"
##
## $B
## [1] "Tomorrow, Brolin--ballpark figure!"
##
## $C
## [1] "Tonight, Chaeden--can\\'t cut the mustard!"
##
## $D
## [1] "For your eyes only, Daxten--dog and pony show!"
##
## $E
## [1] "As a matter of fact, Eleah--eat your heart out!"
##
## $F
## [1] "Acknowledgment, Frank--feather your nest!"
##
```

Using `map()` With Data Frames



Using `map()` With Data Frames

- (Fake) data similar to a study on cardiac events and air pollution
- For each event (case) in this set, need to compare PM2.5 (air pollution) on the same day of week in the same month

```
ACS_events
```

```
## # A tibble: 621 x 3
##       id ref_date   blood_type
##    <int> <date>     <chr>
##  1     1 2015-01-02 B
##  2     2 2015-01-03 O
##  3     3 2015-01-05 A
##  4     4 2015-01-07 A
##  5     5 2015-01-13 B
##  6     6 2015-01-14 A
##  7     7 2015-01-15 A
##  8     8 2015-01-16 A
##  9     9 2015-01-16 O
## 10    10 2015-01-19 O
## # ... with 611 more rows
```

Using `map()` With Data Frames

List columns

```
nested <-
  ACS_events %>%
  mutate(id_date = map(.x = ref_date,
                       .f = ~ {
                         row_refdate <- .x
                         month_dates <- seq(from = floor_date(.x,
  "month"),
                                            to =
  rollback(ceiling_date(.x, "month")),
                                            by = "day")
                         keep(month_dates, ~ wday(.x) ==
  wday(row_refdate))
                       }))
```

Using `map()` With Data Frames

List columns

- Each cell in `id_date` contains a list of 4 or 5 dates.

```
## # A tibble: 621 x 4
##       id ref_date   blood_type id_date
##    <int> <date>     <chr>      <list>
##  1     1 2015-01-02 B          <date [5]>
##  2     2 2015-01-03 O          <date [5]>
##  3     3 2015-01-05 A          <date [4]>
##  4     4 2015-01-07 A          <date [4]>
##  5     5 2015-01-13 B          <date [4]>
##  6     6 2015-01-14 A          <date [4]>
##  7     7 2015-01-15 A          <date [5]>
##  8     8 2015-01-16 A          <date [5]>
##  9     9 2015-01-16 O          <date [5]>
## 10    10 2015-01-19 O          <date [4]>
## # ... with 611 more rows
```

Using `map()` With Data Frames

Nesting and unnesting

```
unnested <-
  nested %>%
  unnest(id_date) %>%
  arrange(id, id_date) %>%
  mutate(event = ref_date == id_date)
unnested
```

```
## # A tibble: 2,725 x 5
##       id ref_date   blood_type id_date    event
##    <int> <date>     <chr>      <date>     <lgl>
##  1     1 2015-01-02 B          2015-01-02 TRUE
##  2     1 2015-01-02 B          2015-01-09 FALSE
##  3     1 2015-01-02 B          2015-01-16 FALSE
##  4     1 2015-01-02 B          2015-01-23 FALSE
##  5     1 2015-01-02 B          2015-01-30 FALSE
##  6     2 2015-01-03 O          2015-01-03 TRUE
##  7     2 2015-01-03 O          2015-01-10 FALSE
##  8     2 2015-01-03 O          2015-01-17 FALSE
##  9     2 2015-01-03 O          2015-01-24 FALSE
## 10     2 2015-01-03 O          2015-01-31 FALSE
## # ... with 2,715 more rows
```

Using `map()` With Data Frames

(Here's how it ends):

```r
unnested %>%
  left_join(PM25, by = c("id_date" = "date"))
```

```
## # A tibble: 2,725 x 6
##         id ref_date   blood_type id_date    event  PM25
##      <int> <date>     <chr>      <date>     <lgl> <dbl>
##   1      1 2015-01-02 B          2015-01-02 TRUE  11.9
##   2      1 2015-01-02 B          2015-01-09 FALSE 11.9
##   3      1 2015-01-02 B          2015-01-16 FALSE 15.9
##   4      1 2015-01-02 B          2015-01-23 FALSE  4.05
##   5      1 2015-01-02 B          2015-01-30 FALSE  6.22
##   6      2 2015-01-03 O          2015-01-03 TRUE   6.06
##   7      2 2015-01-03 O          2015-01-10 FALSE 15.3
##   8      2 2015-01-03 O          2015-01-17 FALSE 15.1
##   9      2 2015-01-03 O          2015-01-24 FALSE  8.48
## 10      2 2015-01-03 O          2015-01-31 FALSE 16.6
## # ... with 2,715 more rows
```

Modify a List

```
modify(.x, .f, ...)
```

- Behaves like `map()`
- **Except** returns the type of object it receives

Modify Functions

`safely()`, `quietly()`, `possibly()`, et al.

- "Adverbs": modify how functions ("verbs") behave
- Handy complement to `map()` and friends

Modify Functions

- `safely(.f, otherwise = NULL, quiet = TRUE)`
- Suppose I want to convert a bunch of things to dates
- If I happen to have some errant values, I want to know about them
- I don’t want to stop everything because one date didn’t work

Modify Functions

- `safely(.f, otherwise = NULL, quiet = TRUE)`

```r
bunch_o_ints <- list(1:2, "A", 4)

safe_date <- safely(.f = as_date,
                    otherwise = "This position is empty")

map(bunch_o_ints, safe_date, origin = origin)
```

```
## [[1]]
## [[1]]$result
## [1] "1970-01-02" "1970-01-03"
##
## [[1]]$error
## NULL
##
##
## [[2]]
## [[2]]$result
## [1] "This position is empty"
##
## [[2]]$error
## <simpleError in .local(x, ...): unused argument (origin = origin)>
##
##
## [[3]]
## [[3]]$result
```

Reduce/Accumulate

- `reduce(.x, .f, ..., .init .dir = c("forward", "ba`
- `accumulate(.x, .f, ..., .init, .dir = c("forward"`

Simplest examples:

```r
reduce(1:10, `+`)
```

```
## [1] 55
```

```r
accumulate(1:10, `+`)
```

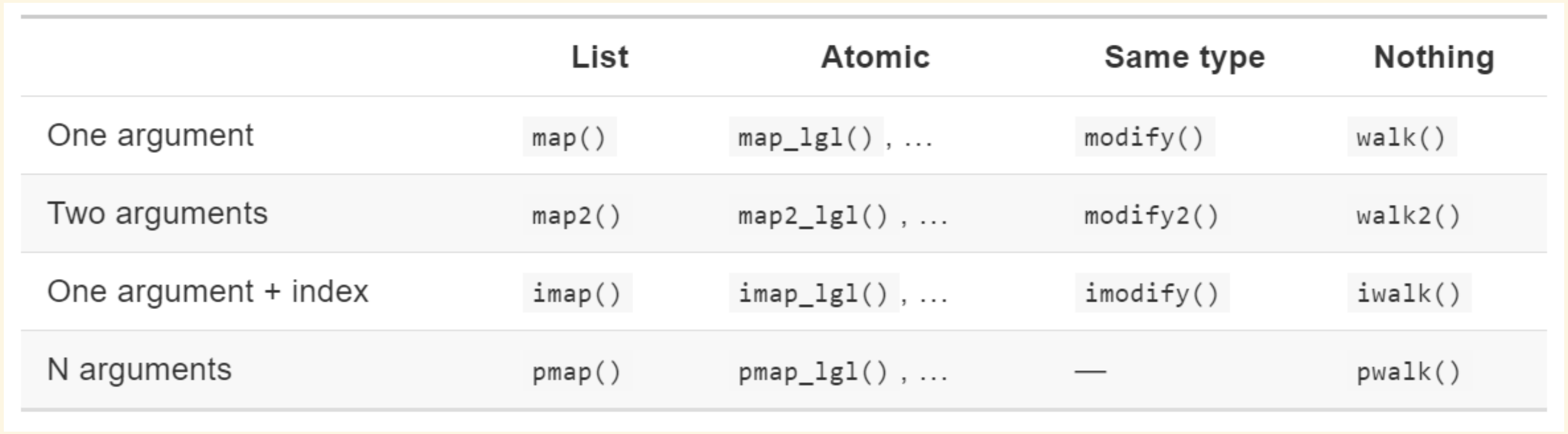
```
##  [1]  1  3  6 10 15 21 28 36 45 55
```

Reduce/Accumulate

Bigger, more complex:

```r
list(df1, df2, df3, df4) %>%
  reduce(.f = full_join)
```

A Map of `map()` Varieties



	List	Atomic	Same type	Nothing
One argument	`map()`	`map_lgl()` , …	`modify()`	`walk()`
Two arguments	`map2()`	`map2_lgl()` , …	`modify2()`	`walk2()`
One argument + index	`imap()`	`imap_lgl()` , …	`imodify()`	`iwalk()`
N arguments	`pmap()`	`pmap_lgl()` , …	—	`pwalk()`

Reference: https://adv-r.hadley.nz/functionals.html

Learn More

- https://r4ds.had.co.nz/iteration.html
- https://adv-r.hadley.nz/functionals.html
- https://purrr.tidyverse.org