Looping Variants

Daniel Anderson

Week 5, Class 2

Agenda

- `walk()` and friends
- `modify()`
- `safely()`
- `reduce()`

Learning Objectives

- Know when to apply `walk` instead of `map`, and why it may be useful
- Understand the parallels and differences between `map` and `modify`
- Diagnose errors with `safely` and understand other situations where it may be helpful
- Collapsing/reducing lists with `purrr::reduce()` or `base::Reduce()`

Setup

Let's go back to our plotting example from last class.

First we'll load our libraries

```
library(tidyverse)
library(fivethirtyeight)
library(glue)
library(english)
```

Prep the data

```
pulitzer <- pulitzer %>%
  select(newspaper, starts_with("num")) %>%
  pivot_longer(
    -newspaper,
    names_to = "year_range",
    values_to = "n",
    names_prefix = "num_finals"
  ) %>%
  mutate(year_range = str_replace_all(year_range, "_", "-")) %>%
  filter(year_range != "1990-2014") %>%
    group_by(newspaper) %>%
    mutate(
      tot = sum(n),
      label = glue(
        "{str_to_title(as.english(tot))} Total Pulitzer Awards"
    )
)
```

Produce plots

```
final_plots <- pulitzer %>%
    group_by(newspaper, label) %>%
    nest() %>%
    mutate(plots = pmap(list(newspaper, label, data), ~{
    ggplot(..3, aes(n, year_range)) +
      geom_col(aes(fill = n)) +
      scale_fill_distiller(type = "seq",
                           limits = c(0, max(pulitzer$n)),
                           palette = "BuPu",
                           direction = 1) +
        scale_x_continuous(limits = c(0, max(pulitzer$n)),
                           expand = c(0, 0)) +
        guides(fill = "none") +
        labs(title = glue("Pulitzer Prize winners: {..1}"),
             x = "Total number of winners",
             y = "",
             caption = ..2)
      })
)
```

Saving

- We saw last time that we could use `nest_by()`
  - Required a bit of awkwardness with adding the paths to the data frame
  - Instead, we'll do it again but with the `walk()` family

Why `walk()` for saving instead of `map()`?

> Walk is an alternative to map that you use when you want to call a function for its side effects, rather than for its return value. You typically do this because you want to render output to the screen or save files to disk – the important thing is the action, not the return value.
>
> –r4ds

More practical

If you use `walk()`, nothing will get printed to the screent. This is particularly helpful for RMarkdown files.

Example

Please do the following

- Create a new RMarkdown document
- Paste the code you have for creating the plots in a code chunk there (along with the library loading, data cleaning, etc.)

01:00

Create a directory

```
fs::dir_create(here::here("plots", "pulitzers"))
```

Create file paths

```
newspapers <- str_replace_all(tolower(final_plots$newspaper), " '
paths <- here::here("plots", "pulitzers", glue("{newspapers}.png'
```

Challenge

- Use a `map()` family function to loop through `paths` and `final_plots$plots` to save all plots.
- Render (knit) your file. What do you notice?

03:00

`walk()`

Just like `map()`, we have parallel variants of `walk()`, including, `walk2()`, and `pwalk()`

These work just like `map()` but don't print to the screen

Try replacing your prior code with a `walk()` version.

How does the rendered output change?

02:00

Save plots

```
walk2(paths, final_plots$plots, ggsave,
      width = 9.5,
      height = 6.5,
      dpi = 500)
```

modify

> Unlike `map()` and its variants which always return a fixed object type (list for `map()`, integer vector for `map_int()`, etc), the `modify()` family always returns the same type as the input object.

map vs modify

map

```
map(mtcars, ~as.numeric(scale(.x)))
```

```
## $mpg
##  [1]  0.15088482  0.15088482  0.44954345  0.21725341 -0.23073453 -0.3302
##  [8]  0.71501778  0.44954345 -0.14777380 -0.38006384 -0.61235388 -0.4630
## [15] -1.60788262 -1.60788262 -0.89442035  2.04238943  1.71054652  2.2912
## [22] -0.76168319 -0.81145962 -1.12671039 -0.14777380  1.19619000  0.9804
## [29] -0.71190675 -0.06481307 -0.84464392  0.21725341
##
## $cyl
##  [1] -0.1049878 -0.1049878 -1.2248578 -0.1049878  1.0148821 -0.1049878
##  [9] -1.2248578 -0.1049878 -0.1049878  1.0148821  1.0148821  1.0148821
## [17]  1.0148821 -1.2248578 -1.2248578 -1.2248578 -1.2248578  1.0148821
## [25]  1.0148821 -1.2248578 -1.2248578 -1.2248578  1.0148821 -0.1049878
##
## $disp
##  [1] -0.57061982 -0.57061982 -0.99018209  0.22009369  1.04308123 -0.0461
##  [8] -0.67793094 -0.72553512 -0.50929918 -0.50929918  0.36371309  0.3637
## [15]  1.94675381  1.84993175  1.68856165 -1.22658929 -1.25079481 -1.2879
## [22]  0.70420401  0.59124494  0.96239618  1.36582144 -1.22416874 -0.8909
## [29]  0.97046468 -0.69164740  0.56703942 -0.88529152
##
## $hp
```

modify

```
modify(mtcars, ~as.numeric(scale(.x)))
```

```
##                           mpg        cyl        disp          hp
## Mazda RX4            0.15088482 -0.1049878 -0.57061982 -0.53509284  0.56
## Mazda RX4 Wag        0.15088482 -0.1049878 -0.57061982 -0.53509284  0.56
## Datsun 710           0.44954345 -1.2248578 -0.99018209 -0.78304046  0.47
## Hornet 4 Drive       0.21725341 -0.1049878  0.22009369 -0.53509284 -0.96
## Hornet Sportabout   -0.23073453  1.0148821  1.04308123  0.41294217 -0.83
## Valiant             -0.33028740 -0.1049878 -0.04616698 -0.60801861 -1.56
## Duster 360          -0.96078893  1.0148821  1.04308123  1.43390296 -0.72
## Merc 240D            0.71501778 -1.2248578 -0.67793094 -1.23518023  0.17
## Merc 230             0.44954345 -1.2248578 -0.72553512 -0.75387015  0.60
## Merc 280            -0.14777380 -0.1049878 -0.50929918 -0.34548584  0.60
## Merc 280C           -0.38006384 -0.1049878 -0.50929918 -0.34548584  0.60
## Merc 450SE          -0.61235388  1.0148821  0.36371309  0.48586794 -0.98
## Merc 450SL          -0.46302456  1.0148821  0.36371309  0.48586794 -0.98
## Merc 450SLC         -0.81145962  1.0148821  0.36371309  0.48586794 -0.98
## Cadillac Fleetwood  -1.60788262  1.0148821  1.94675381  0.85049680 -1.24
## Lincoln Continental -1.60788262  1.0148821  1.84993175  0.99634834 -1.11
## Chrysler Imperial   -0.89442035  1.0148821  1.68856165  1.21512565 -0.68
## Fiat 128             2.04238943 -1.2248578 -1.22658929 -1.17683962  0.90
## Honda Civic          1.71054652 -1.2248578 -1.25079481 -1.38103178  2.49
## Toyota Corolla       2.29127162 -1.2248578 -1.28790993 -1.19142477  1.16
## Toyota Corona        0.23384555 -1.2248578 -0.89255318 -0.72469984  0.19
## Dodge Challenger    -0.76168319  1.0148821  0.70420401  0.04831332 -1.56
## AMC Javelin         -0.81145962  1.0148821  0.59124494  0.04831332 -0.83
## Camaro Z28          -1.12671039  1.0148821  0.96239618  1.43390296  0.24
## Pontiac Firebird    -0.14777380  1.0148821  1.36582144  0.41294217 -0.96
```

```
modify2(LETTERS[1:3], letters[1:3], paste0)
```

```
## [1] "Aa" "Bb" "Cc"
```

```
map2(LETTERS[1:3], letters[1:3], paste0)
```

```
## [[1]]
## [1] "Aa"
##
## [[2]]
## [1] "Bb"
##
## [[3]]
## [1] "Cc"
```

safely

Iterating when errors are possible

Sometimes a loop will work for most cases, but return an error on a few

Often, you want to return the output you can

Alternatively, you might want to diagnose *where* the error is occurring

`purrr::safely`

Example

```
by_cyl <- mpg %>%
  group_by(cyl) %>%
  nest()
by_cyl
```

```
## # A tibble: 4 x 2
## # Groups:   cyl [4]
##     cyl data
##   <int> <list>
## 1     4 <tibble[,10] [81 × 10]>
## 2     6 <tibble[,10] [79 × 10]>
## 3     8 <tibble[,10] [70 × 10]>
## 4     5 <tibble[,10] [4 × 10]>
```

```
by_cyl %>%
  mutate(mod = map(data, ~lm(hwy ~ displ + drv, data = .x)))
```

```
## Error: Problem with `mutate()` input `mod`.
## x contrasts can be applied only to factors with 2 or more levels
## ℹ Input `mod` is `map(data, ~lm(hwy ~ displ + drv, data = .x))`.
## ℹ The error occurred in group 2: cyl = 5.
```

Safe return

- First, define safe function – note that this will work for any function

```
safe_lm <- safely(lm)
```

- Next, loop the safe function, instead of the standard function

```
safe_models <- by_cyl %>%
  mutate(safe_mod = map(data, ~safe_lm(hwy ~ displ + drv, data =
safe_models
```

```
## # A tibble: 4 x 3
## # Groups:   cyl [4]
##     cyl data                  safe_mod
##   <int> <list>                <list>
## 1     4 <tibble[,10] [81 × 10]> <named list [2]>
## 2     6 <tibble[,10] [79 × 10]> <named list [2]>
## 3     8 <tibble[,10] [70 × 10]> <named list [2]>
## 4     5 <tibble[,10] [4 × 10]>  <named list [2]>
```

What's returned?

```
safe_models$safe_mod[[1]]
```

```
## $result
##
## Call:
## .f(formula = ..1, data = ..2)
##
## Coefficients:
## (Intercept)        displ         drvf
##      37.370       -5.289        3.882
##
##
## $error
## NULL
```

```
safe_models$safe_mod[[4]]
```

```
## $result
## NULL
##
## $error
## <simpleError in `contrasts<-`(`*tmp*`, value = contr.funs[1 + isOF[nn]])
```

Inspecting

I often use `safely()` to help me de–bug. Why is it failing *there*.

First – create a new variable to filter for results with errors

```
safe_models %>%
  mutate(error = map_lgl(safe_mod, ~!is.null(.x$error)))
```

```
## # A tibble: 4 x 4
## # Groups:   cyl [4]
##     cyl data                  safe_mod         error
##   <int> <list>                <list>           <lgl>
## 1     4 <tibble[,10] [81 × 10]> <named list [2]> FALSE
## 2     6 <tibble[,10] [79 × 10]> <named list [2]> FALSE
## 3     8 <tibble[,10] [70 × 10]> <named list [2]> FALSE
## 4     5 <tibble[,10] [4 × 10]>  <named list [2]> TRUE
```

Inspecting the data

```
safe_models %>%
  mutate(error = map_lgl(safe_mod, ~!is.null(.x$error))) %>%
  filter(error) %>%
  select(cyl, data) %>%
  unnest(data)
```

```
## # A tibble: 4 x 11
## # Groups:   cyl [1]
##     cyl manufacturer model      displ  year trans      drv     cty   hwy
##   <int> <chr>        <chr>      <dbl> <int> <chr>      <chr> <int> <int>
## 1     5 volkswagen   jetta        2.5  2008 auto(s6)   f        21    29
## 2     5 volkswagen   jetta        2.5  2008 manual(m5) f        21    29
## 3     5 volkswagen   new beetle   2.5  2008 manual(m5) f        20    28
## 4     5 volkswagen   new beetle   2.5  2008 auto(s6)   f        20    29
```

The `displ` and `drv` variables ar constant, so no relation can be estimated.

Pull results that worked

```
safe_models %>%
  mutate(results = map(safe_mod, "result"))
```

```
## # A tibble: 4 x 4
## # Groups:   cyl [4]
##     cyl data                      safe_mod         results
##   <int> <list>                    <list>           <list>
## 1     4 <tibble[,10] [81 × 10]> <named list [2]> <lm>
## 2     6 <tibble[,10] [79 × 10]> <named list [2]> <lm>
## 3     8 <tibble[,10] [70 × 10]> <named list [2]> <lm>
## 4     5 <tibble[,10] [4 × 10]>  <named list [2]> <NULL>
```

Now we can `broom::tidy()` or whatevs

Notice that there is no `cyl == 5`.

```
safe_models %>%
  mutate(results = map(safe_mod, "result"),
         tidied = map(results, broom::tidy)) %>%
  select(cyl, tidied) %>%
  unnest(tidied)
```

```
## # A tibble: 11 x 6
## # Groups:   cyl [3]
##      cyl term          estimate std.error  statistic      p.value
##    <int> <chr>            <dbl>     <dbl>      <dbl>        <dbl>
##  1     4 (Intercept) 37.37023   3.537572  10.56381   1.052943e-16
##  2     4 displ       -5.288562  1.436068  -3.682668  4.235795e- 4
##  3     4 drvf         3.882134  0.9971876  3.893083  2.073699e- 4
##  4     6 (Intercept) 27.96536   2.347630  11.91217   5.718039e-19
##  5     6 displ       -2.333261  0.6373304 -3.660991  4.651570e- 4
##  6     6 drvf         4.570840  0.6012367  7.602397  6.789988e-11
##  7     6 drvr         6.384355  1.229277   5.193585  1.713129e- 6
##  8     8 (Intercept) 14.82265   2.887289   5.133759  2.708515e- 6
##  9     8 displ        0.3060487 0.5719058  0.5351383 5.943528e- 1
## 10     8 drvf         8.555294  2.679129   3.193311  2.156229e- 3
## 11     8 drvr         3.709336  0.7319048  5.068058  3.473594e- 6
```

When else might we use this?

Any sort of web scraping – pages change and URLs don't always work

Example

```
library(rvest)
links <- list(
  "https://en.wikipedia.org/wiki/FC_Barcelona",
  "https://nosuchpage",
  "https://en.wikipedia.org/wiki/Rome"
)
pages <- map(links, ~{
  Sys.sleep(0.1)
  read_html(.x)
})
```

```
## Error in open.connection(x, "rb"): Failed to connect to nosuchpage port
```

The problem

I can't connect to https://nosuchpage because it doesn't exist

BUT

That also means I can't get *any* of my links because *one* page errored (imagine it was 1 in 1,000 instead of 1 in 3)

`safely()` to the rescue

Safe version

```
safe_read_html <- safely(read_html)
pages <- map(links, ~{
  Sys.sleep(0.1)
  safe_read_html(.x)
})
str(pages)
```

```
## List of 3
##  $ :List of 2
##   ..$ result:List of 2
##   .. ..$ node:<externalptr>
##   .. ..$ doc :<externalptr>
##   .. ..- attr(*, "class")= chr [1:2] "xml_document" "xml_node"
##   ..$ error : NULL
##  $ :List of 2
##   ..$ result: NULL
##   ..$ error :List of 2
##   .. ..$ message: chr "Failed to connect to nosuchpage port 443: Operati
##   .. ..$ call   : language open.connection(x, "rb")
##   .. ..- attr(*, "class")= chr [1:3] "simpleError" "error" "condition"
##  $ :List of 2
##   ..$ result:List of 2
##   .. ..$ node:<externalptr>
##   .. ..$ doc :<externalptr>
##   .. ..- attr(*, "class")= chr [1:2] "xml_document" "xml_node"
```

Non–results

In a real example, we'd probably want to double–check the pages where we got no results

```
errors <- map_lgl(pages, ~!is.null(.x$error))
links[errors]
```

```
## [[1]]
## [1] "https://nosuchpage"
```

reduce

Reducing a list

The `map()` family of functions will always return a vector the same length as the input `reduce()` will collapse or reduce the list to a single element

Example

```
l <- list(
  c(1, 3),
  c(1, 5, 7, 9),
  3,
  c(4, 8, 12, 2)
)

reduce(l, sum)
```

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

What's going on?

The code `reduce(l, sum)` is the same as

```
sum(l[[4]], sum(l[[3]], sum(l[[1]], l[[2]])))
```

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

Or slidghlty differently

```
first_sum <- sum(l[[1]], l[[2]])
second_sum <- sum(first_sum, l[[3]])
final_sum <- sum(second_sum, l[[4]])
final_sum
```

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

Why might you use this?

What if you had a list of data frames like this

```
l_df <- list(
  tibble(id = 1:3, score = rnorm(3)),
  tibble(id = 1:5, treatment = rbinom(5, 1, .5)),
  tibble(id = c(1, 3, 5, 7), other_thing = rnorm(4))
)
```

We can join these all together with a single loop – we want the output to be of length 1!

```
reduce(l_df, full_join)
```

```
## # A tibble: 6 x 4
##      id       score treatment other_thing
##   <dbl>       <dbl>     <int>       <dbl>
## 1     1 -0.3782499           1   0.1678145
## 2     2 -0.02004990          0  NA
## 3     3 -1.763696            0   1.260830
## 4     4 NA                   1  NA
## 5     5 NA                   0   0.2522675
## 6     7 NA                  NA   0.3740336
```

Note – you have to be careful on directionality

```
reduce(l_df, left_join)
```

```
## # A tibble: 3 x 4
##      id       score treatment other_thing
##   <dbl>       <dbl>     <int>       <dbl>
## 1     1 -0.3782499           1   0.1678145
## 2     2 -0.02004990          0  NA
## 3     3 -1.763696            0   1.260830
```

```
reduce(l_df, right_join)
```

```
## # A tibble: 4 x 4
##      id       score treatment other_thing
##   <dbl>       <dbl>     <int>       <dbl>
## 1     1 -0.3782499           1   0.1678145
## 2     3 -1.763696            0   1.260830
## 3     5 NA                   0   0.2522675
## 4     7 NA                  NA   0.3740336
```

More common

You probably just want to `bind_rows()`

```
l_df2 <- list(
  tibble(id = 1:3, scid = 1, score = rnorm(3)),
  tibble(id = 1:5, scid = 2, score = rnorm(5)),
  tibble(id = c(1, 3, 5, 7), scid = 3, score = rnorm(4))
)
reduce(l_df2, bind_rows)
```

```
## # A tibble: 12 x 3
##       id  scid       score
##    <dbl> <dbl>       <dbl>
##  1     1     1  0.8564613
##  2     2     1 -0.7238976
##  3     3     1 -0.2668617
##  4     1     2  1.358469
##  5     2     2 -1.181592
##  6     3     2  0.5089259
##  7     4     2 -0.4892010
##  8     5     2 -0.2350173
##  9     1     3  0.01672748
## 10     3     3  0.2884129
## 11     5     3 -0.01951237
## 12     7     3 -1.682676
```

Wrap up

- Lots more to `{purrr}` but we've covered a lot
- Functional programming can *really* help your efficiency, and even if it slows you down initially, I'd recommend always striving toward it, because it will ultimately be a huge help.

Questions?

If we have any time left – let's work on the homework