Iterating well with R (and the tidyverse)

rstd.io/raukr

Jennifer Bryan

RStudio

@jennybc

@JennyBryan

installs dplyr, tidyr, and purrr, among other things

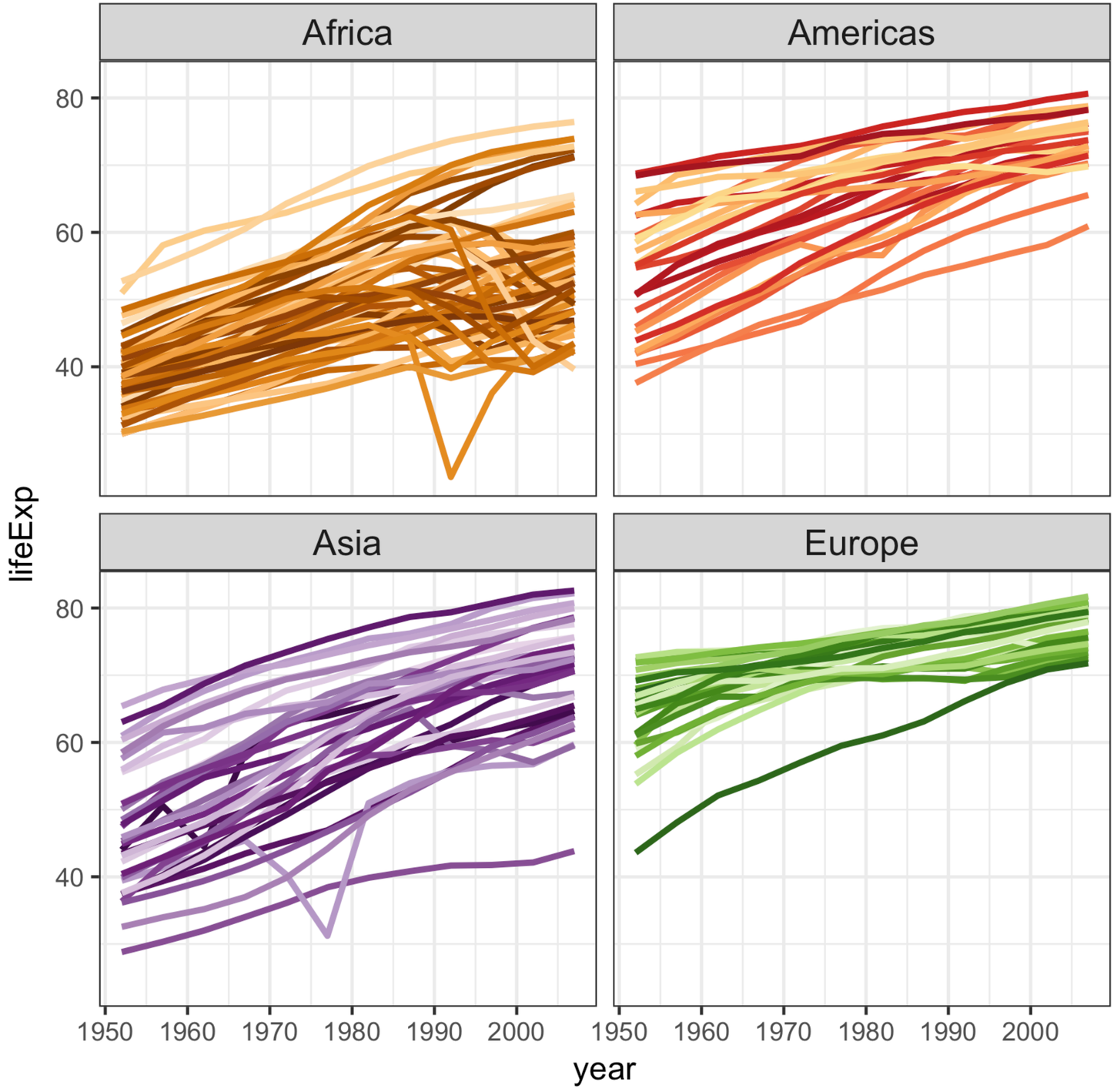
```
install.packages("tidyverse")
install.packages("repurrrsive")
install.packages("gapminder")
```

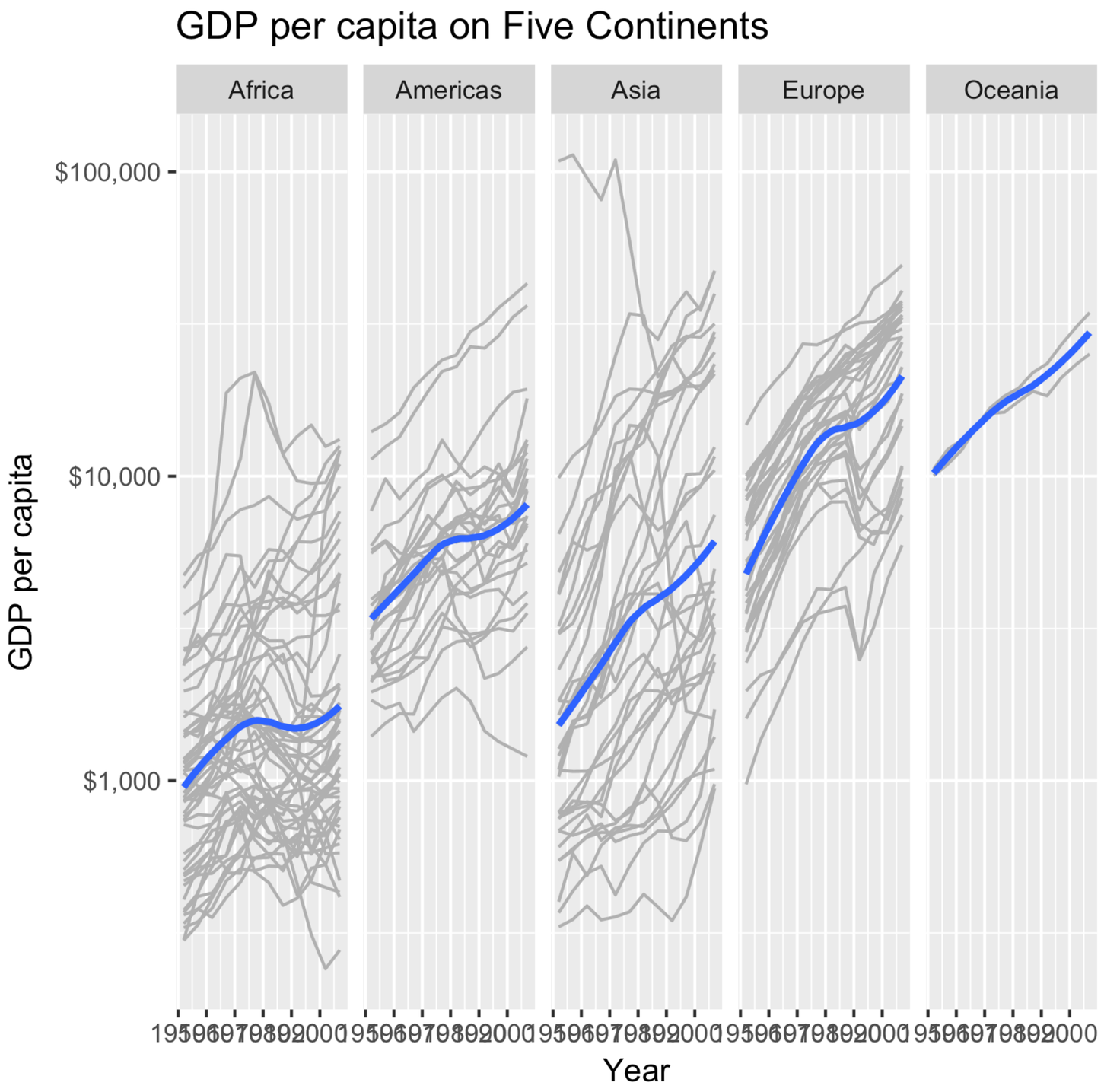
- What is the harm with copy/paste and repetitive code?
- What's the downside to explicit for loops?
- What should I do instead?
  - use purpose-built tools to express iteration in terms of your data
- (Self-study, later) Consult resources for prompts for concrete hands-on practice

```
library(gapminder)
library(tidyverse)

gapminder
#> # A tibble: 1,704 x 6
#>    country     continent  year lifeExp      pop gdpPercap
#>    <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
#>  1 Afghanistan Asia       1952    28.8  8425333      779.
#>  2 Afghanistan Asia       1957    30.3  9240934      821.
#>  3 Afghanistan Asia       1962    32.0 10267083      853.
#>  4 Afghanistan Asia       1967    34.0 11537966      836.
#>  5 Afghanistan Asia       1972    36.1 13079460      740.
#>  6 Afghanistan Asia       1977    38.4 14880372      786.
#>  7 Afghanistan Asia       1982    39.9 12881816      978.
#>  8 Afghanistan Asia       1987    40.8 13867957      852.
#>  9 Afghanistan Asia       1992    41.7 16317921      649.
#> 10 Afghanistan Asia       1997    41.8 22227415      635.
#> # … with 1,694 more rows
```

```
gapminder %>%
  count(continent)
#> # A tibble: 5 x 2
#>   continent     n
#>   <fct>     <int>
#> 1 Africa      624
#> 2 Americas    300
#> 3 Asia        396
#> 4 Europe      360
#> 5 Oceania      24
```


Africa
Americas
Asia
Europe
lifeExp
80
60
40
1950
1960
1970
1980
1990
2000
year


GDP per capita on Five Continents
Africa
Americas
Asia
Europe
Oceania
$100,000
$10,000
$1,000
GDP per capita
1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 2002 2007
Year

```r
africa ← gapminder[gapminder$continent == "Africa", ]
africa_mm ← max(africa$lifeExp) - min(africa$lifeExp)

americas ← gapminder[gapminder$continent == "Americas", ]
americas_mm ← max(americas$lifeExp) - min(americas$lifeExp)

asia ← gapminder[gapminder$continent == "Asia", ]
asia_mm ← max(asia$lifeExp) - min(africa$lifeExp)

europe ← gapminder[gapminder$continent == "Europe", ]
europe_mm ← max(europe$lifeExp) - min(europe$lifeExp)

oceania ← gapminder[gapminder$continent == "Oceania", ]
oceania_mm ← max(europe$lifeExp) - min(oceania$lifeExp)

cbind(
  continent = c("Africa", "Asias", "Europe", "Oceania"),
  max_minus_min = c(africa_mm, americas_mm, asia_mm,
                    europe_mm, oceania_mm)
)
```

What am I trying to do?

Have I even done it?*

* Can you find my mistakes?

How would *you* compute this?

for each continent

  max life exp - min life exp

put result in a data frame

Here's how I would use the dplyr package to do it.

```
gapminder %>%
  group_by(continent) %>%
  summarize(max_minus_min = max(lifeExp) - min(lifeExp))
#> # A tibble: 5 x 2
#>   continent max_minus_min
#>   <fct>             <dbl>
#> 1 Africa             52.8
#> 2 Americas           43.1
#> 3 Asia               53.8
#> 4 Europe             38.2
#> 5 Oceania            12.1
```

Conclusion: there are many ways to write a for loop in R!

This is not just a matter of taste.

Implementation is connected to (the likelihood of) correctness.

```
cbind(
  continent = c("Africa", "Asias", "Europe", "Oceania"),
  max_minus_min = c(africa_mm, americas_mm, asia_mm,
                    europe_mm, oceania_mm)
)
#> Warning in cbind(continent = c("Africa", "Asias", "Europe", "Oceania"), : number of
#> rows of result is not a multiple of vector length (arg 1)
#>      continent max_minus_min
#> [1,] "Africa"  "52.843"
#> [2,] "Asias"   "43.074"
#> [3,] "Europe"  "59.004"
#> [4,] "Oceania" "38.172"
#> [5,] "Africa"  "12.637"
```

"manual" split-apply-combine

incorrect result

```
#> 10 lines of repetitive copy/paste code ...
cbind(
  continent = c("Africa", "Asias", ...),
  max_minus_min = c( ... )
)
#> Warning in cbind( ... )
#>      continent max_minus_min
#> [1,] "Africa"  "52.843"
#> [2,] "Asias"   "43.074"
#> [3,] "Europe"  "59.004"
#> [4,] "Oceania" "38.172"
#> [5,] "Africa"  "12.637"
```

"conceptual" split-apply-combine

correct result

```
gapminder %>%
  group_by(continent) %>%
  summarize(max_minus_min = ...)
#> # A tibble: 5 x 2
#>   continent max_minus_min
#>   <fct>             <dbl>
#> 1 Africa             52.8
#> 2 Americas           43.1
#> 3 Asia               53.8
#> 4 Europe             38.2
#> 5 Oceania            12.1
```

sidebar on %>%*

* As of R 4.1, base R even has a native pipe! |>

```r
filter(gapminder, country == "Canada")
gapminder %>%
  filter (country == "Canada")

mean(x)
x %>% mean()
```

```
whatever(arg1, arg2, arg3, …)

arg1 %>%
  whatever(arg2, arg3, …)
```

```
foo_foo <- little_bunny()

bop_on(
  scoop_up(
    hop_through(foo_foo, forest),
    field_mouse
  ),
  head
)

# vs
foo_foo %>%
  hop_through(forest) %>%
  scoop_up(field_mouse) %>%
  bop_on(head)
```

from various Hadley Wickham talks

the magrittr package provides %>%

used heavily and re-exported in the tidyverse

As of R 4.1, base R even has a native pipe! |>

tidyverse can't use native pipe yet (backwards compatibility)

but %>% and |> do the same thing for most usage

https://www.tidyverse.org/blog/2020/11/magrittr-2-0-is-here/

New example: making strings

```
child ← c("Reed", "Wesley", "Eli", "Toby")
age   ← c(    17,       15,    15,      4)

s ← rep_len("", length(child))
for (i in seq_along(s)) {
  s[i] ← paste(child[i], "is", age[i], "years old")
}
s
#> [1] "Reed is 17 years old"   "Wesley is 15 years old" "Eli is 15 years old"
#> [4] "Toby is 4 years old"
```

Here's one way I might do it.

```
child ← c("Reed", "Wesley", "Eli", "Toby")
age   ← c(    17,       15,    15,      4)

paste(child, "is", age, "years old")
#> [1] "Reed is 17 years old"   "Wesley is 15 years old" "Eli is 15 years old"
#> [4] "Toby is 4 years old"
```

Lots of R functions, such as paste(), are natively vectorized.

You don't need to write the loop!

Here's another way I might do it.

```
child ← c("Reed", "Wesley", "Eli", "Toby")
age   ← c(    17,        15,    15,       4)
glue::glue("{child} is {age} years old")
#> Reed is 17 years old
#> Wesley is 15 years old
#> Eli is 15 years old
#> Toby is 4 years old
```

Some tasks -- like string interpolation -- are so important that it pays to educate yourself about purpose-built solutions, e.g. the glue package.

Here's yet another way I might do it.

```
dat ← tibble(child, age)
glue::glue_data(dat, "{child} is {age} years old")
#> Reed is 17 years old
#> Wesley is 15 years old
#> Eli is 15 years old
#> Toby is 4 years old
```

Again, the glue package rocks.

Foreshadows: iterating over rows of a data frame.

This is not just a matter of taste.

Implementation connected to readability / maintainability.

Consider syntax : substance ratio.

```r
s ← rep_len("", length(child))
for (i in seq_along(s)) {
  s[i] ← paste(child[i], "is", age[i], "years old")
}

paste(child, "is", age, "years old")

glue::glue("{child} is {age} years old")

glue::glue_data(dat, "{child} is {age} years old")
```

Of course, someone has to write loops.

It doesn't have to be you.

Try to express your intent in the language of your data.

- What is the harm with copy/paste and repetitive code?
- What's the downside to explicit for loops?
- What should I do instead?
  - use purpose-built tools to express iteration in terms of your data
- (Self-study, later) Consult resources for prompts for concrete hands-on practice

Current tidyverse vibe / philosophy:

Most users should be able to express themselves in dplyr:

Use group_by() and rowwise() to declare unit of iteration.

For more complicated problems and general iteration, use purrr.

Alone or together with dplyr.

This is a very typical "look" for dplyr::group_by().

```
gapminder %>%
  group_by(continent) %>%
  summarize(max_minus_min = max(lifeExp) - min(lifeExp))
#> # A tibble: 5 x 2
#>   continent max_minus_min
#>   <fct>             <dbl>
#> 1 Africa             52.8
#> 2 Americas           43.1
#> 3 Asia               53.8
#> 4 Europe             38.2
#> 5 Oceania            12.1
```

Input: many rows per continent

Output: 1 row per continent

group_by() "just" changes how subsequent dplyr verbs work

```
gapminder %>%
  group_by(continent)
#> # A tibble: 1,704 x 6
#> # Groups:   continent [5]
#>    country     continent  year lifeExp      pop gdpPercap
#>    <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
#>  1 Afghanistan Asia       1952    28.8  8425333      779.
#>  2 Afghanistan Asia       1957    30.3  9240934      821.
#>  3 Afghanistan Asia       1962    32.0 10267083      853.
#>  4 Afghanistan Asia       1967    34.0 11537966      836.
#>  5 Afghanistan Asia       1972    36.1 13079460      740.
#>  6 Afghanistan Asia       1977    38.4 14880372      786.
#>  7 Afghanistan Asia       1982    39.9 12881816      978.
#>  8 Afghanistan Asia       1987    40.8 13867957      852.
#>  9 Afghanistan Asia       1992    41.7 16317921      649.
#> 10 Afghanistan Asia       1997    41.8 22227415      635.
#> # … with 1,694 more rows
```

But sometimes the "unit of work" really is a row.

There's nothing to group by (other than row number).

```
sw ← starwars %>%
  slice(1, 4, 10) %>%
  select(name, starships)
sw
#> # A tibble: 3 x 2
#>   name           starships
#>   <chr>          <list>
#> 1 Luke Skywalker <chr [2]>
#> 2 Darth Vader    <chr [1]>
#> 3 Obi-Wan Kenobi <chr [5]>
```

```
sw %>%
  mutate(ships = paste(starships, collapse = "/"))
#> # A tibble: 3 x 3
#>   name          starships ships
#>   <chr>         <list>    <chr>
#> 1 Luke Skywal… <chr [2]> "c(\"X-wing\", \"Imperial shuttle\")/TIE Advanced x1/c(\"J…
#> 2 Darth Vader  <chr [1]> "c(\"X-wing\", \"Imperial shuttle\")/TIE Advanced x1/c(\"J…
#> 3 Obi-Wan Ken… <chr [5]> "c(\"X-wing\", \"Imperial shuttle\")/TIE Advanced x1/c(\"J…
```

```
sw %>%
  rowwise() %>%
  mutate(ships = paste(starships, collapse = "/"))
#> # A tibble: 3 x 3
#> # Rowwise:
#>   name          starships ships
#>   <chr>         <list>    <chr>
#> 1 Luke Skywal… <chr [2]> X-wing/Imperial shuttle
#> 2 Darth Vader  <chr [1]> TIE Advanced x1
#> 3 Obi-Wan Ken… <chr [5]> Jedi starfighter/Trade Federation cruiser/Naboo star skiff…
```

```
sw %>%
  rowwise() %>%
  mutate(ships = paste(starships, collapse = "/"))
#> # A tibble: 3 x 3
#> # Rowwise:
#>   name          starships ships
#>   <chr>         <list>    <chr>
#> 1 Luke Skywal… <chr [2]> X-wing/Imperial shuttle
#> 2 Darth Vader  <chr [1]> TIE Advanced x1
#> 3 Obi-Wan Ken… <chr [5]> Jedi starfighter/Trade Federation cruiser/Naboo star skiff…
```

Input: 1 row per person

Output: 1 row per person

rowwise() "just" changes how subsequent dplyr verbs work,

like group_by() does

```
sw %>%
  rowwise()
#> # A tibble: 3 x 2
#> # Rowwise:
#>   name           starships
#>   <chr>          <list>
#> 1 Luke Skywalker <chr [2]>
#> 2 Darth Vader    <chr [1]>
#> 3 Obi-Wan Kenobi <chr [5]>
```

```
sw %>%
  rowwise() %>%
  mutate(ships = paste(starships, collapse = "/"))
#> # A tibble: 3 x 3
#> # Rowwise:
#>   name          starships ships
#>   <chr>         <list>    <chr>
#> 1 Luke Skywal… <chr [2]> X-wing/Imperial shuttle
#> 2 Darth Vader  <chr [1]> TIE Advanced x1
#> 3 Obi-Wan Ken… <chr [5]> Jedi starfighter/Trade Federation cruiser/Naboo star skiff…

sw %>%
  rowwise() %>%
  summarize(ships = paste(starships, collapse = "/"))
#> `summarise()` has ungrouped output. You can override using the `.groups` argument.
#> # A tibble: 3 x 1
#>   ships
#>   <chr>
#> 1 X-wing/Imperial shuttle
#> 2 TIE Advanced x1
#> 3 Jedi starfighter/Trade Federation cruiser/Naboo star skiff/Jedi Interceptor/Belbu…
```

```
sw %>%
  rowwise() %>%
  mutate(ships = paste(starships, collapse = "/"))
#> # A tibble: 3 x 3
#> # Rowwise:
#>   name          starships ships
#>   <chr>         <list>    <chr>
#> 1 Luke Skywal… <chr [2]> X-wing/Imperial shuttle
#> 2 Darth Vader  <chr [1]> TIE Advanced x1
#> 3 Obi-Wan Ken… <chr [5]> Jedi starfighter/Trade Federation cruiser/Naboo star skiff…

sw %>%
  rowwise() %>%
  summarize(ships = paste(starships, collapse = "/"))
#> `summarise()` has ungrouped output. You can override using the `.groups` argument.
#> # A tibble: 3 x 1
#>   ships
#>   <chr>
#> 1 X-wing/Imperial shuttle
#> 2 TIE Advanced x1
#> 3 Jedi starfighter/Trade Federation cruiser/Naboo star skiff/Jedi Interceptor/Belbu…
```

```
sw %>%
  rowwise() %>%
  summarize(ships = paste(starships, collapse = "/"))
#> `summarise()` has ungrouped output. You can override using the `.groups` argument.
#> # A tibble: 3 x 1
#>   ships
#>   <chr>
#> 1 X-wing/Imperial shuttle
#> 2 TIE Advanced x1
#> 3 Jedi starfighter/Trade Federation cruiser/Naboo star skiff/Jedi Interceptor/Belbu…

sw %>%
  rowwise(name) %>%
  summarize(ships = paste(starships, collapse = "/"))
#> `summarise()` has grouped output by 'name'. You can override using the `.groups` argument.
#> # A tibble: 3 x 2
#> # Groups:   name [3]
#>   name          ships
#>   <chr>         <chr>
#> 1 Luke Skywalk… X-wing/Imperial shuttle
#> 2 Darth Vader   TIE Advanced x1
#> 3 Obi-Wan Keno… Jedi starfighter/Trade Federation cruiser/Naboo star skiff/Jedi Int…
```

```
sw %>%
  rowwise() %>%
  summarize(ships = paste(starships, collapse = "/"))
#> `summarise()` has ungrouped output. You can override using the `.groups` argument.
#> # A tibble: 3 x 1
#>   ships
#>   <chr>
#> 1 X-wing/Imperial shuttle
#> 2 TIE Advanced x1
#> 3 Jedi starfighter/Trade Federation cruiser/Naboo star skiff/Jedi Interceptor/Belbu…

sw %>%
  rowwise(name) %>%
  summarize(ships = paste(starships, collapse = "/"))
#> `summarise()` has grouped output by 'name'. You can override using the `.groups` argument.
#> # A tibble: 3 x 2
#> # Groups:   name [3]
#>   name          ships
#>   <chr>         <chr>
#> 1 Luke Skywalk… X-wing/Imperial shuttle
#> 2 Darth Vader   TIE Advanced x1
#> 3 Obi-Wan Keno… Jedi starfighter/Trade Federation cruiser/Naboo star skiff/Jedi Int…
```

dplyr::rowwise() vs. using purrr + dplyr

dplyr::rowwise() vs. using purrr + dplyr

```
sw %>%
  rowwise() %>%
  mutate(ships = paste(starships, collapse = "/"))
#> # A tibble: 3 x 3
#> # Rowwise:
#>   name          starships ships
#>   <chr>         <list>    <chr>
#> 1 Luke Skywal… <chr [2]> X-wing/Imperial shuttle
#> 2 Darth Vader  <chr [1]> TIE Advanced x1
#> 3 Obi-Wan Ken… <chr [5]> Jedi starfighter/Trade Federation cruiser/Naboo star skiff…

sw %>%
  mutate(ships = map_chr(starships, ~ paste(.x, collapse = "/")))
#> # A tibble: 3 x 3
#>   name          starships ships
#>   <chr>         <list>    <chr>
#> 1 Luke Skywal… <chr [2]> X-wing/Imperial shuttle
#> 2 Darth Vader  <chr [1]> TIE Advanced x1
#> 3 Obi-Wan Ken… <chr [5]> Jedi starfighter/Trade Federation cruiser/Naboo star skiff…
```

```
sw %>%
  mutate(n_ships = length(starships))
#> # A tibble: 3 x 3
#>   name           starships n_ships
#>   <chr>          <list>      <int>
#> 1 Luke Skywalker <chr [2]>       3
#> 2 Darth Vader    <chr [1]>       3
#> 3 Obi-Wan Kenobi <chr [5]>       3
```

```
sw %>%
  rowwise() %>%
  mutate(n_ships = length(starships))
#> # A tibble: 3 x 3
#> # Rowwise:
#>   name           starships n_ships
#>   <chr>          <list>      <int>
#> 1 Luke Skywalker <chr [2]>       2
#> 2 Darth Vader    <chr [1]>       1
#> 3 Obi-Wan Kenobi <chr [5]>       5
```

Here's a "wide" dataset that comes with tidyr.

```
billboard
#> # A tibble: 317 x 79
#>    artist  track  date.entered   wk1   wk2   wk3   wk4   wk5   wk6   wk7   wk8   wk9
#>    <chr>   <chr>  <date>       <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#>  1 2 Pac   Baby … 2000-02-26      87    82    72    77    87    94    99    NA    NA
#>  2 2Ge+her The H… 2000-09-02      91    87    92    NA    NA    NA    NA    NA    NA
#>  3 3 Door… Krypt… 2000-04-08      81    70    68    67    66    57    54    53    51
#>  4 3 Door… Loser  2000-10-21      76    76    72    69    67    65    55    59    62
#>  5 504 Bo… Wobbl… 2000-04-15      57    34    25    17    17    31    36    49    53
#>  6 98^0    Give … 2000-08-19      51    39    34    26    26    19     2     2     3
#>  7 A*Teens Danci… 2000-07-08      97    97    96    95   100    NA    NA    NA    NA
#>  8 Aaliyah I Don… 2000-01-29      84    62    51    41    38    35    35    38    38
#>  9 Aaliyah Try A… 2000-03-18      59    53    38    28    21    18    16    14    12
#> 10 Adams,… Open … 2000-08-26      76    76    74    69    68    67    61    58    57
#> # … with 307 more rows, and 67 more variables: wk10 <dbl>, wk11 <dbl>, wk12 <dbl>,
#> #   wk13 <dbl>, wk14 <dbl>, wk15 <dbl>, wk16 <dbl>, wk17 <dbl>, wk18 <dbl>,
#> #   and so on,
#> #   wk73 <lgl>, wk74 <lgl>, wk75 <lgl>, wk76 <lgl>
```

```
billboard %>%
  mutate(early_rank = min(wk1, wk2, wk3, na.rm = TRUE)) %>%
  select(artist, track, early_rank) %>%
  head(3)
#> # A tibble: 3 x 3
#>   artist       track                   early_rank
#>   <chr>        <chr>                        <dbl>
#> 1 2 Pac        Baby Don't Cry (Keep ...          6
#> 2 2Ge+her      The Hardest Part Of  ...          6
#> 3 3 Doors Down Kryptonite                       6
```

```
billboard %>%
  rowwise() %>%
  mutate(early_rank = min(wk1, wk2, wk3, na.rm = TRUE)) %>%
  select(artist, track, early_rank) %>%
  head(3)
#> # A tibble: 3 x 3
#> # Rowwise:
#>   artist       track                   early_rank
#>   <chr>        <chr>                        <dbl>
#> 1 2 Pac        Baby Don't Cry (Keep ...         72
#> 2 2Ge+her      The Hardest Part Of  ...         87
#> 3 3 Doors Down Kryptonite                       68
```

c_across() helps you target columns,

within a row,

using "tidy-select" syntax

```
billboard %>%
  rowwise() %>%
  mutate(avg_rank = mean(c_across(starts_with("wk")), na.rm = TRUE)) %>%
  select(artist, track, avg_rank) %>%
  head(3)
#> # A tibble: 3 x 3
#> # Rowwise:
#>   artist       track                   avg_rank
#>   <chr>        <chr>                      <dbl>
#> 1 2 Pac        Baby Don't Cry (Keep ...    85.4
#> 2 2Ge+her      The Hardest Part Of  ...    90
#> 3 3 Doors Down Kryptonite                  26.5
```

- What is the harm with copy/paste and repetitive code?
- What's the downside to explicit for loops?
- What should I do instead?
  - use purpose-built tools to express iteration in terms of your data
- (Self-study, later) Consult resources for prompts for concrete hands-on practice

Current tidyverse vibe / philosophy:

Most users should be able to express themselves in dplyr:

Use group_by() and rowwise() to declare unit of iteration.

For more complicated problems and general iteration, use purrr.

Alone or together with dplyr.


purrr
www.rstudio.com