Day 2, afternoon

Jennifer Bryan
RStudio, University of British Columbia
@JennyBryan @jennybc

Deep Thoughts

Get good at iteration

Get good at iteration

as in, really, really good

#rstats lists via lego

Hadley Wickham
Lionel Henry

+ dplyr
+ tidyr
+ tibble
+ broom

https://cran.r-project.org/package=purrr
https://github.com/hadley/purrr

An API Of Ice And Fire | https://anapioficeandfire.com

```
{
 "url": "http://www.anapioficeandfire.com/api/characters/1303",
 "id": 1303,
 "name": "Daenerys Targaryen",
 "gender": "Female",
 "culture": "Valyrian",
 "born": "In 284 AC, at Dragonstone",
 "died": "",
 "alive": true,
 "titles": [
    "Queen of the Andals and the Rhoynar and the First Men,
     Lord of the Seven Kingdoms",
    "Khaleesi of the Great Grass Sea",
    "Breaker of Shackles/Chains",
    "Queen of Meereen",
    "Princess of Dragonstone"
],
"aliases": [
    "Dany",
    "Daenerys Stormborn",
```

`x`

`x[i]`

`x[[i]]`

from

http://r4ds.had.co.nz/vectors.html#lists-of-condiments

```
library(repurrrsive)
```

Let's inspect got_chars!

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

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

for every element of **`.x`**

apply **`.f`**

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

how to specify `.f`?

existing function

name & position shortcuts

concise ~ formula syntax

```
.x = minis
```

```
map(minis, "pants")
```

"return results **like so**"

```
map_lgl(.x, .f, ...)
map_int(.x, .f, ...)
map_dbl(.x, .f, ...)
map_chr(.x, .f, ...)
```

```
map_dfr(minis, `[`,
        c("pants", "torso", "head")
```

go over to R ...

apply name (and position)
shortcuts to got_chars

plain old map
type-specific map
data-frame-producing map

```
.x = minis
```

```
map(minis, antennate)
```

workflow

1 do something easy with the iterative machine

2 do the real, hard thing with one representative unit

3 insert logic from 2 into template from 1

```r
library(glue)

glue_data(
  list(name = "Jenny", born = "in Atlanta"),
  "{name} was born {born}."
)
#> Jenny was born in Atlanta.
```

```r
glue_data(got_chars[[2]], "{name} was born {born}.")
#> Tyrion Lannister was born In 273 AC, at Casterly Rock.
```

```r
glue_data(got_chars[[9]], "{name} was born {born}.")
#> Daenerys Targaryen was born In 284 AC, at Dragonstone.
```

```
map_chr(got_chars, ~ glue_data(.x, "{name} was born {born}."))
#>  [1] "Theon Greyjoy was born In 278 AC or 279 AC, at Pyke."
#>  [2] "Tyrion Lannister was born In 273 AC, at Casterly Rock."
#>  [3] "Victarion Greyjoy was born In 268 AC or before, at Pyke."
#>  [4] "Will was born ."
#>  [5] "Areo Hotah was born In 257 AC or before, at Norvos."
#>  [6] "Chett was born At Hag's Mire."
#>  [7] "Cressen was born In 219 AC or 220 AC."
#>  [8] "Arianne Martell was born In 276 AC, at Sunspear."
#>  [9] "Daenerys Targaryen was born In 284 AC, at Dragonstone."
```

go over to R ...

no more shortcuts

apply a general function .f

explore various forms





.y = hair

.x = minis

```
map2(minis, hair, enhair)
```

.y = weapons

.x = minis

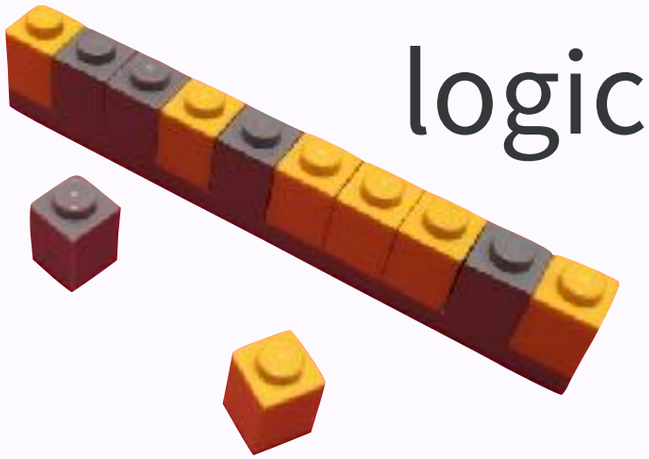
```
map2(minis, weapons, arm)
```

```
minis %>%
   map2(hair, enhair) %>%
   map2(weapons, arm)
```

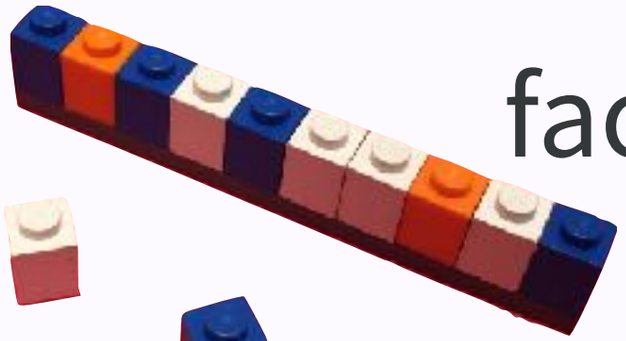
```
df <- tibble(pants, torso, head)
embody <- function(pants, torso, head)
  insert(insert(pants, torso), head)
```

pmap(df, embody)

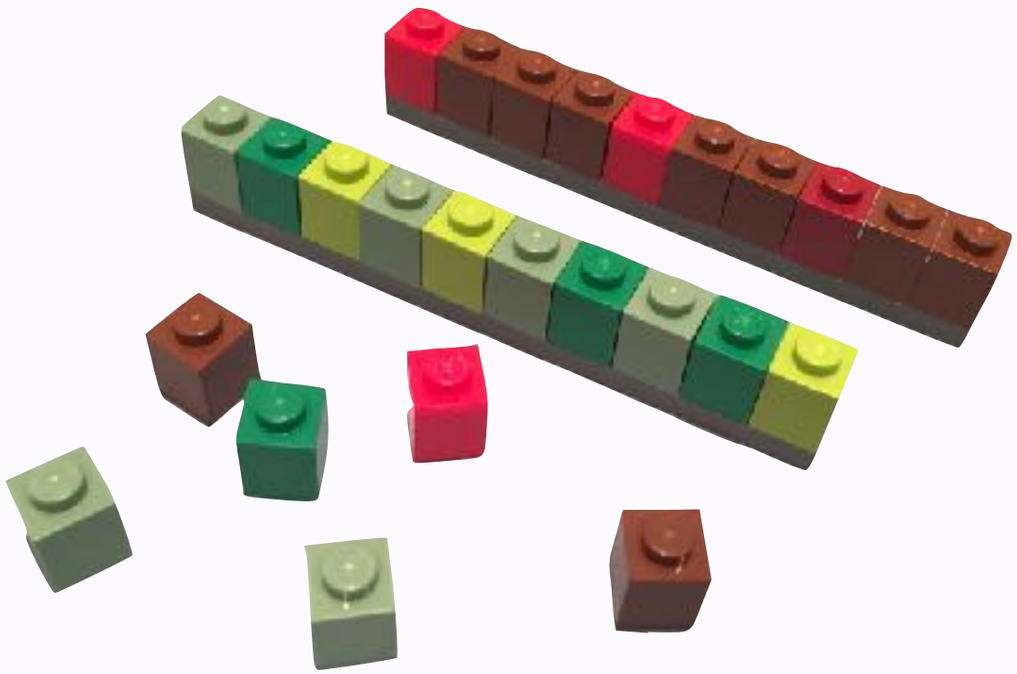
atomic vectors



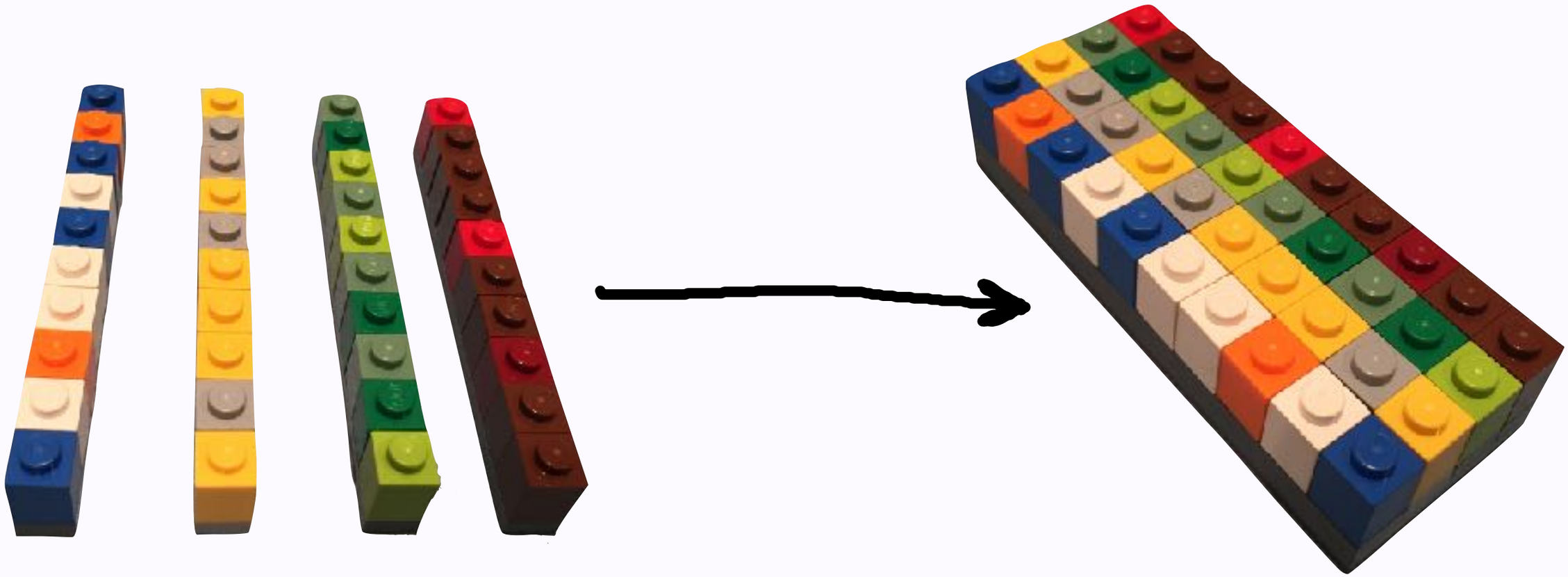
logical



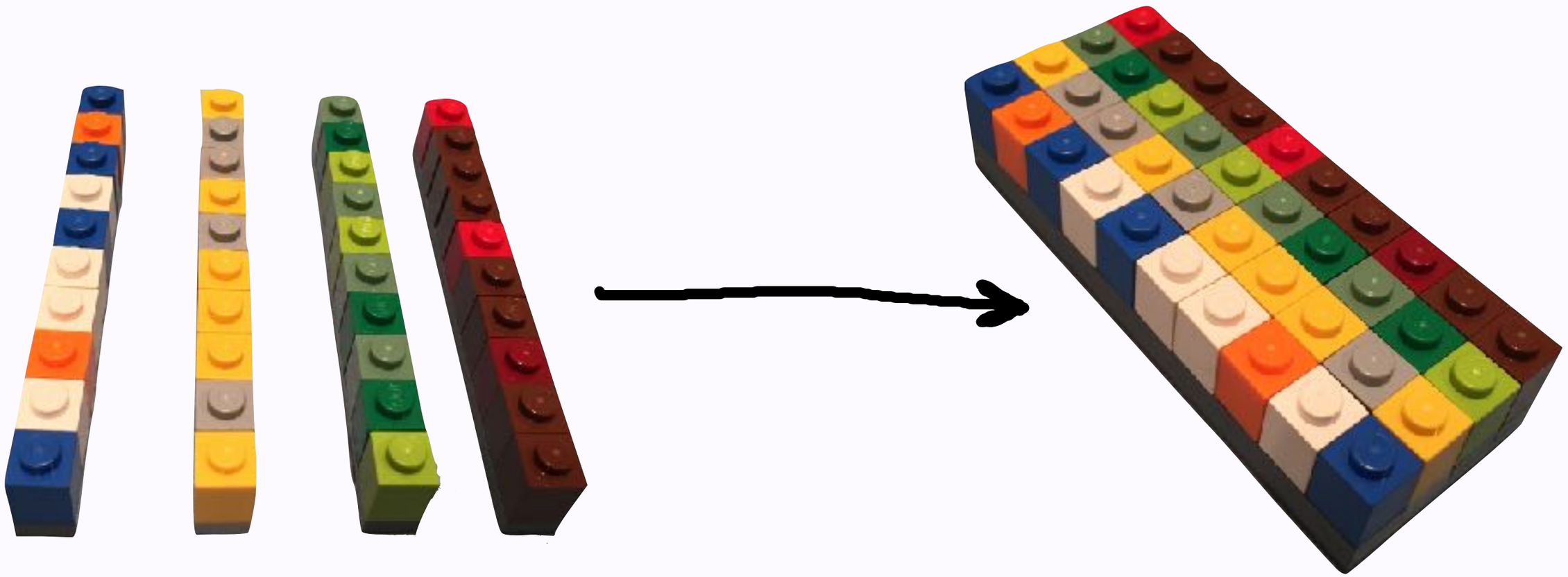
factor



integer, double



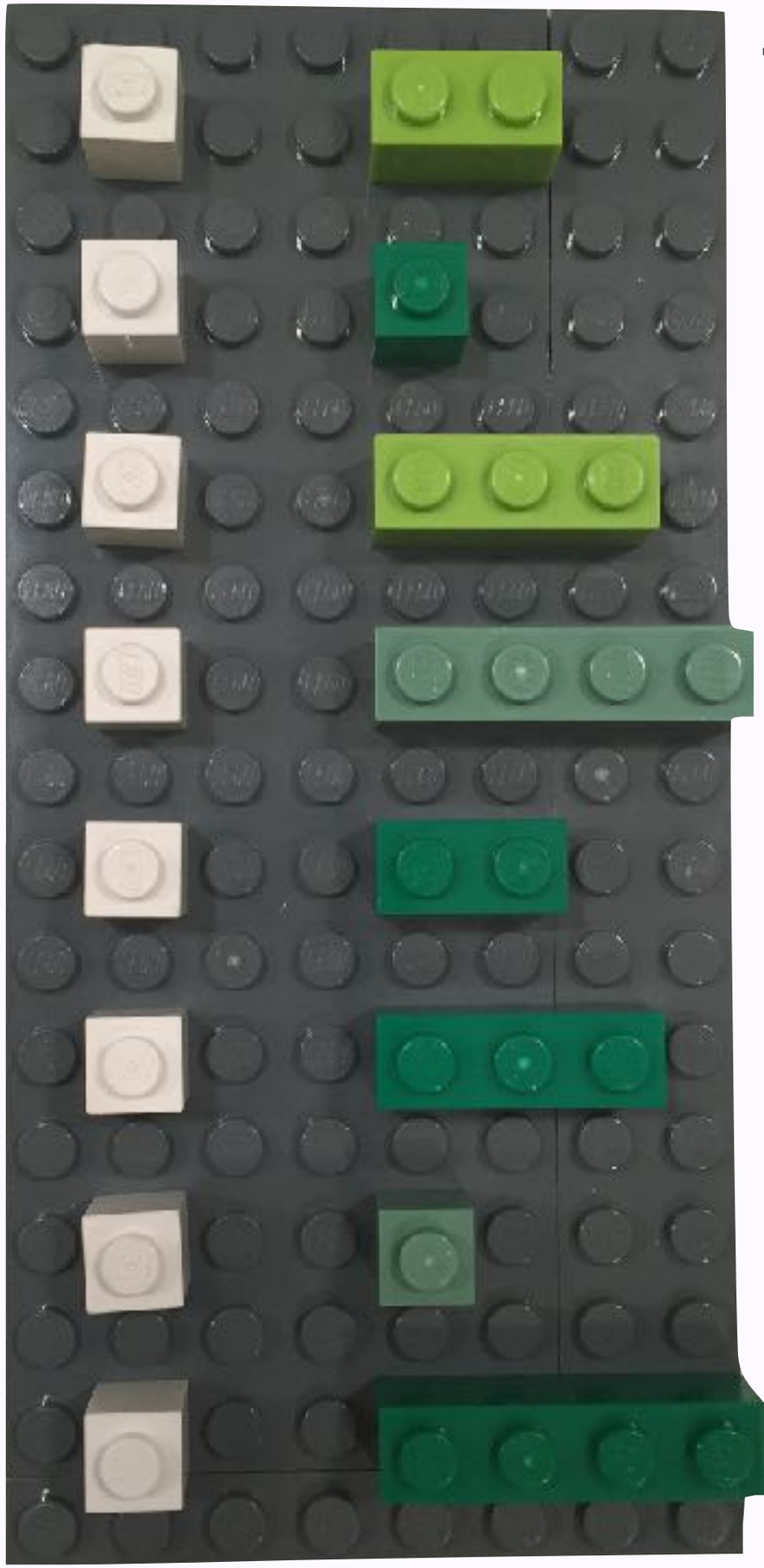
vectors of same length? **DATA FRAME!**



vectors don’t have to be atomic

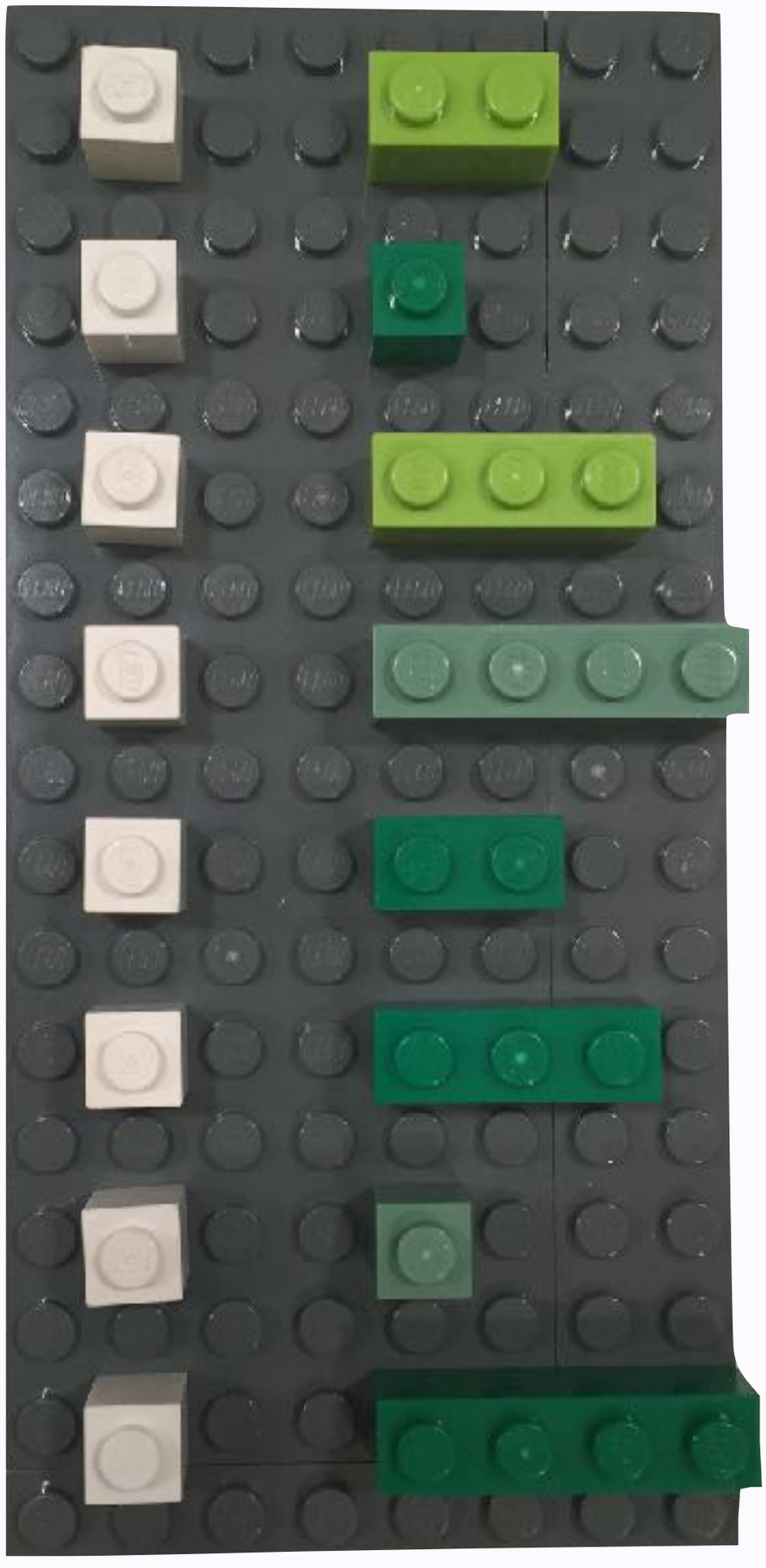
works for lists too! LOVE THE **LIST COLUMN!**

atomic vector



this is a data frame!

list column



you can use all the previous techniques to manipulate and simplify list-columns in a data frame

Why would you do this to yourself?

Lists are often forced upon you.

- String processing, e.g., splitting
- JSON or XML, e.g., from web API
- Split-Apply-Combine

But why lists in a data frame?

All the usual reasons!

- Keep multiple vectors intact and “in sync”
- Use existing toolkit for filter, select, ….

What happens in the

DATA FRAME

Stays in the data frame

```
gt <- tibble(
  name = map_chr(got_chars, "name"),
  houses = map(got_chars, "allegiances")
)
gt %>%
  mutate(n_houses = map_int(houses,length)) %>%
  filter(n_houses > 1) %>%
  unnest()
#> # A tibble: 15 x 3
#>    name              n_houses houses
#>    <chr>                <int> <chr>
#>  1 Davos Seaworth           2 House Baratheon of Dragonstone
#>  2 Davos Seaworth           2 House Seaworth of Cape Wrath
#>  3 Asha Greyjoy             2 House Greyjoy of Pyke
#>  4 Asha Greyjoy             2 House Ironmaker
#>  5 Barristan Selmy          2 House Selmy of Harvest Hall
#>  6 Barristan Selmy          2 House Targaryen of King's Landing
#>  7 Brienne of Tarth         3 House Baratheon of Storm's End
#>  8 Brienne of Tarth         3 House Stark of Winterfell
#>  9 Brienne of Tarth         3 House Tarth of Evenfall Hall
#> 10 Catelyn Stark            2 House Stark of Winterfell
#> 11 Catelyn Stark            2 House Tully of Riverrun
#> 12 Jon Connington           2 House Connington of Griffin's Roost
#> 13 Jon Connington           2 House Targaryen of King's Landing
#> 14 Sansa Stark              2 House Baelish of Harrenhal
#> 15 Sansa Stark              2 House Stark of Winterfell
```

```
gt <- tibble(
  name = map_chr(got_chars, "name"),
  houses = map(got_chars, "allegiances")
)
gt %>%
  mutate(n_houses = map_int(houses,length)) %>%
  filter(n_houses > 1) %>%
  unnest()
#> # A tibble: 15 x 3
#>    name                n_
#>    <chr>
#>  1 Davos Seaworth
#>  2 Davos Seaworth
#>  3 Asha Greyjoy
#>  4 Asha Greyjoy
#>  5 Barristan Selmy
#>  6 Barristan Selmy
#>  7 Brienne of Tarth
#>  8 Brienne of Tarth
#>  9 Brienne of Tarth
#> 10 Catelyn Stark
#> 11 Catelyn Stark
#> 12 Jon Connington
#> 13 Jon Connington
#> 14 Sansa Stark
#> 15 Sansa Stark
```

```
atheon of Dragonstone
worth of Cape Wrath
yjoy of Pyke
nmaker
my of Harvest Hall
garyen of King's Landing
3 House Baratheon of Storm's End
3 House Stark of Winterfell
3 House Tarth of Evenfall Hall
2 House Stark of Winterfell
2 House Tully of Riverrun
2 House Connington of Griffin's Roost
2 House Targaryen of King's Landing
2 House Baelish of Harrenhal
2 House Stark of Winterfell
```

extract, filter, unnest