Iteration

Goal

Fit model, compute r.squared, collect coefficient ***for every country.***

One possible solution

```r
canada <- gapminder %>%
  filter(country == "Canada")
canada_lm <- lm(lifeExp ~ year, data = canada)
canada_lm %>% glance() %>% pull(r.squared)
canada_lm %>% tidy() %>%
  filter(term == "year") %>%
  pull(estimate)
```

Adapted from 'Master the tidyverse' CC by RStudio

One possible solution

```r
canada <- gapminder %>%
  filter(country == "Canada")
canada_lm <- lm(lifeExp ~ year, data = canada)
canada_lm %>% glance() %>% pull(r.squared)
canada_lm %>% tidy() %>%
  filter(term == "year") %>%
  pull(estimate)
```

Copy and paste

Adapted from 'Master the tidyverse' CC by RStudio

One possible solution

```r
nz <- gapminder %>%
  filter(country == "New Zealand")
nz_lm <- lm(lifeExp ~ year, data = nz)
nz_lm %>% glance() %>% pull(r.squared)
nz_lm %>% tidy() %>%
  filter(term == "year") %>%
  pull(estimate)
```

Edit for another country

Repeat 140 more times!

Is there a better way?

[Adapted from 'Master the tidyverse' CC by RStudio] purrr

purrr

Functions for solving **iteration problems**

```
# install.packages("tidyverse")
library(tidyverse)
```

Iteration problems look like:

For each ______ do ______

Adapted from 'Master the tidyverse' CC by RStudio

Your Turn 1

First, we need to learn about another data structure.

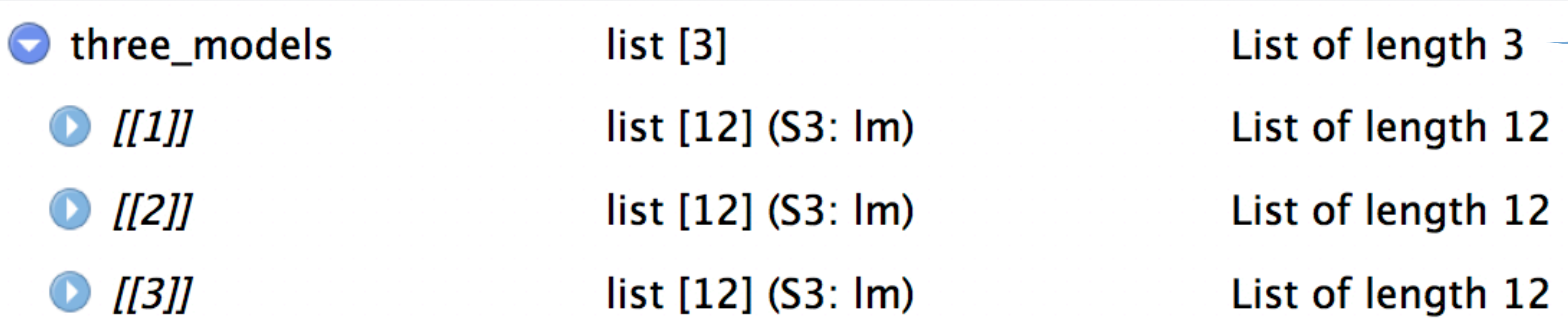
Run the setup chunk.

Look for `three_models` in your environment. Click on it to open a Viewer.

What kind of object is this? Does its contents look familiar?

01:00

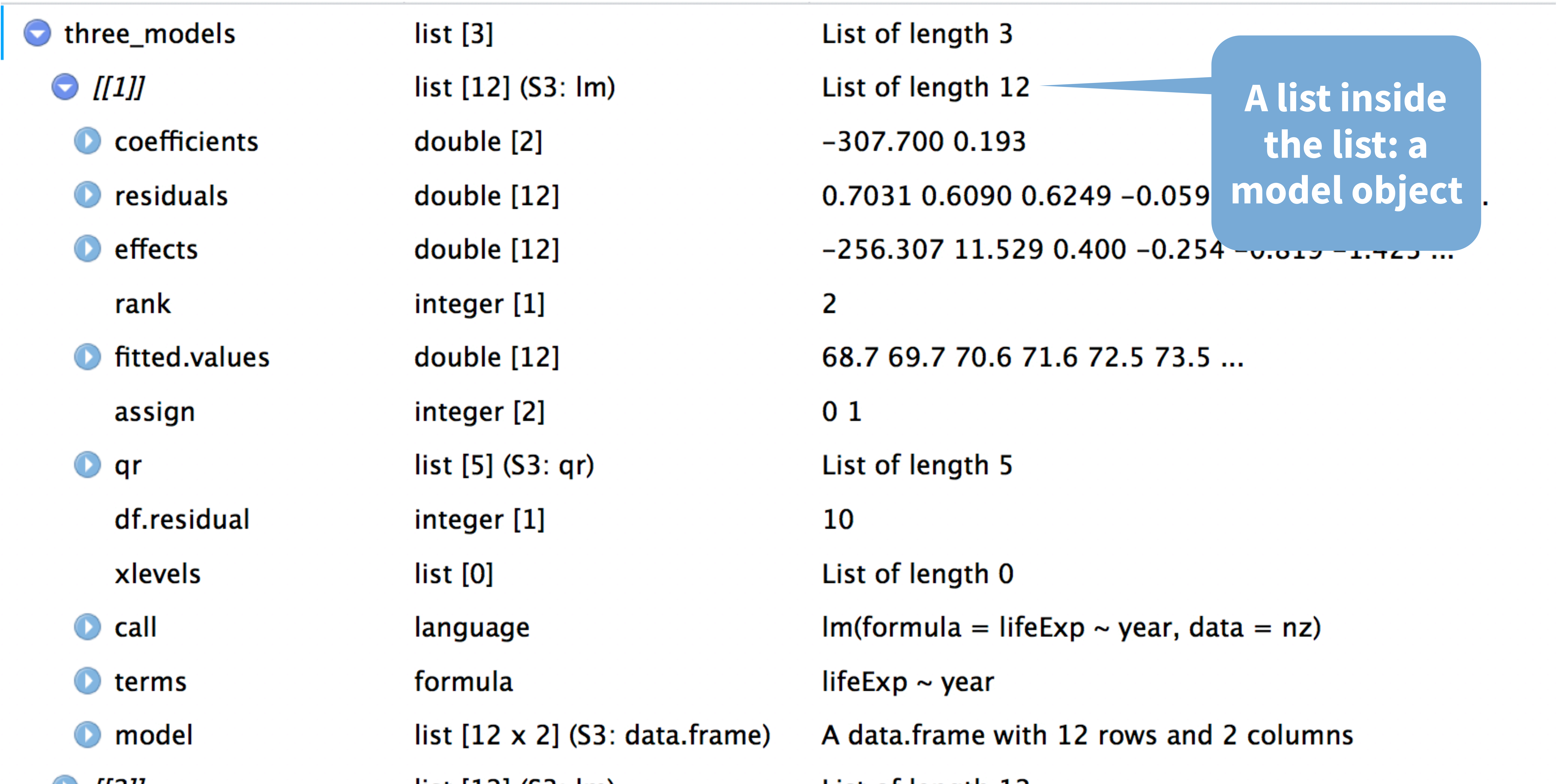
```
View(three_models)
```



three_models	list [3]	List of length 3
[[1]]	list [12] (S3: lm)	List of length 12
[[2]]	list [12] (S3: lm)	List of length 12
[[3]]	list [12] (S3: lm)	List of length 12

A list

```
View(three_models)
```



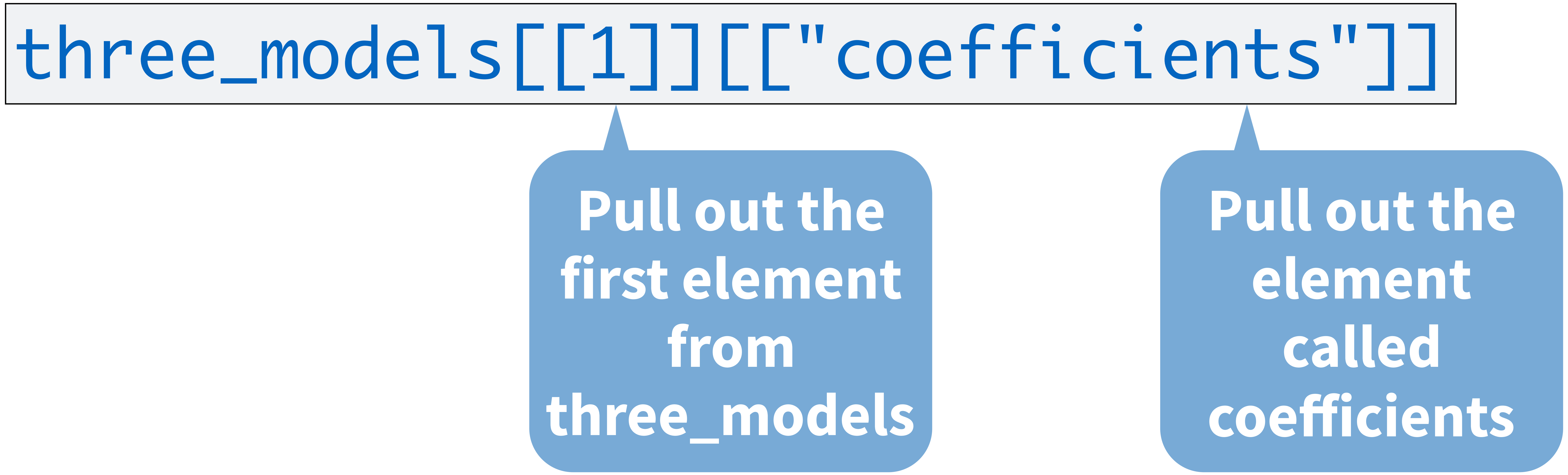
```
three_models      list [3]                         List of length 3
  [[1]]           list [12] (S3: lm)               List of length 12
    coefficients  double [2]                       -307.700 0.193
    residuals     double [12]                      0.7031 0.6090 0.6249 -0.059
    effects       double [12]                      -256.307 11.529 0.400 -0.254 -0.819 -1.425 ...
    rank          integer [1]                      2
    fitted.values double [12]                      68.7 69.7 70.6 71.6 72.5 73.5 ...
    assign        integer [2]                      0 1
    qr            list [5] (S3: qr)                List of length 5
    df.residual   integer [1]                      10
    xlevels       list [0]                         List of length 0
    call          language                         lm(formula = lifeExp ~ year, data = nz)
    terms         formula                          lifeExp ~ year
    model         list [12 x 2] (S3: data.frame)   A data.frame with 12 rows and 2 columns
  [[2]]           list [12] (S3: lm)               List of length 12
```

A list inside the list: a model object

```
View(three_models)
```

three_models	list [3]	List of length 3
[[1]]	list [12] (S3: lm)	List of length 12
coefficients	double [2]	-307.700 0.193
residuals	double [12]	0.7031 0.6090 0.6249 -0.0592 -0.6533 -1.2874 ...
effects	double [12]	-256.307 11.529 0.400 -0.254 -0.819 -1.423 ...
rank	integer [1]	2
fitted.values	double [12]	68.7 69.7 70.6 71.6 72.5 73.5 ...
assign	integer [2]	0 1
qr	list [5] (S3: qr)	List of length 5
df.residual	integer [1]	10
xlevels	list [0]	List of length 0
call	language	lm(formula = lifeExp ~ year, data = nz)
terms	formula	lifeExp ~ year
model	list [12 x 2] (S3: data.frame)	A data.frame with 12 rows and 2 columns

click on this to get code to access this element



```r
three_models[[1]][["coefficients"]]
```

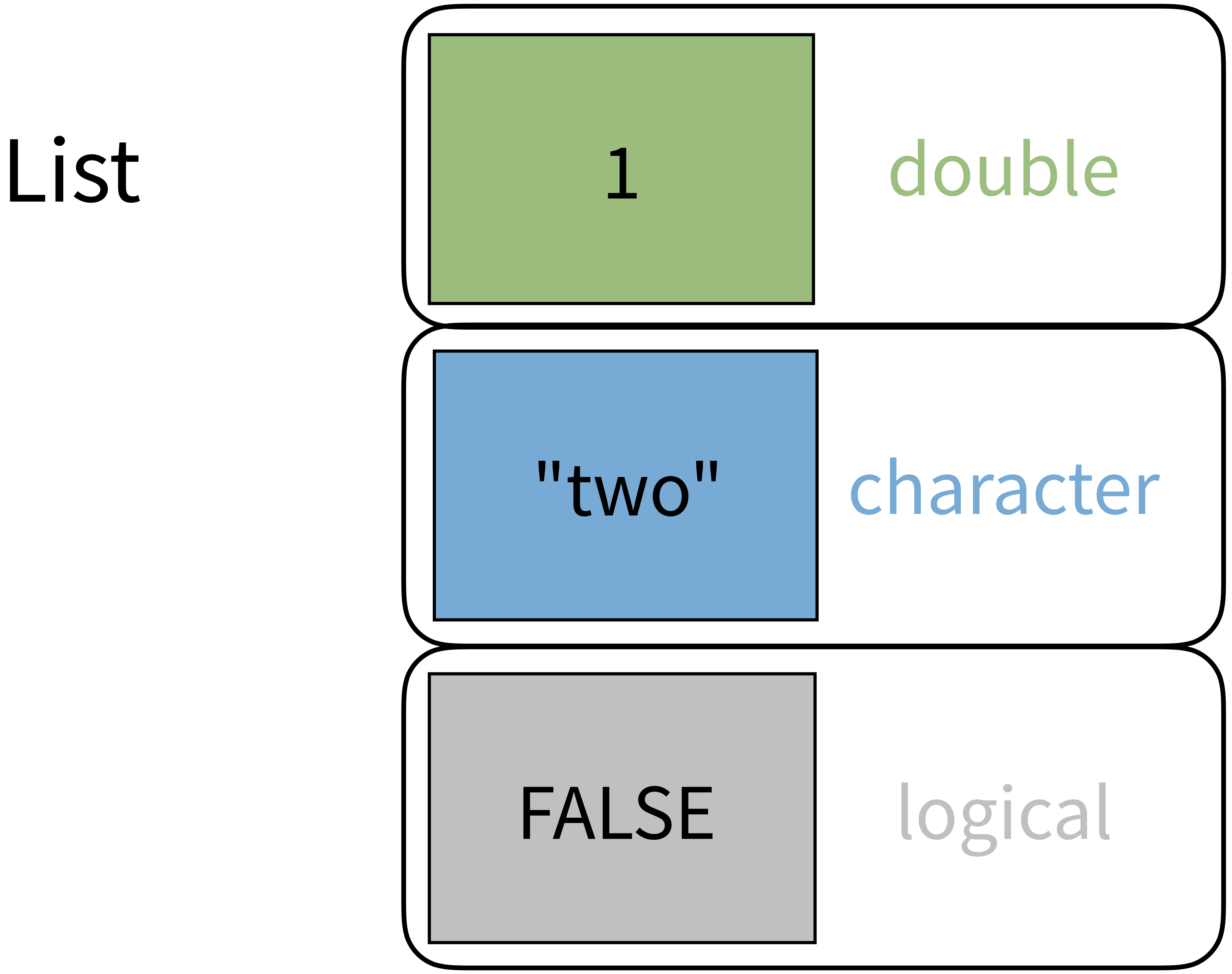
Pull out the first element from three_models

Pull out the element called coefficients

```
(Intercept)          year
-307.699628     0.192821
```

Lists

A way to store complicated and possibly heterogeneous objects



List

1 double

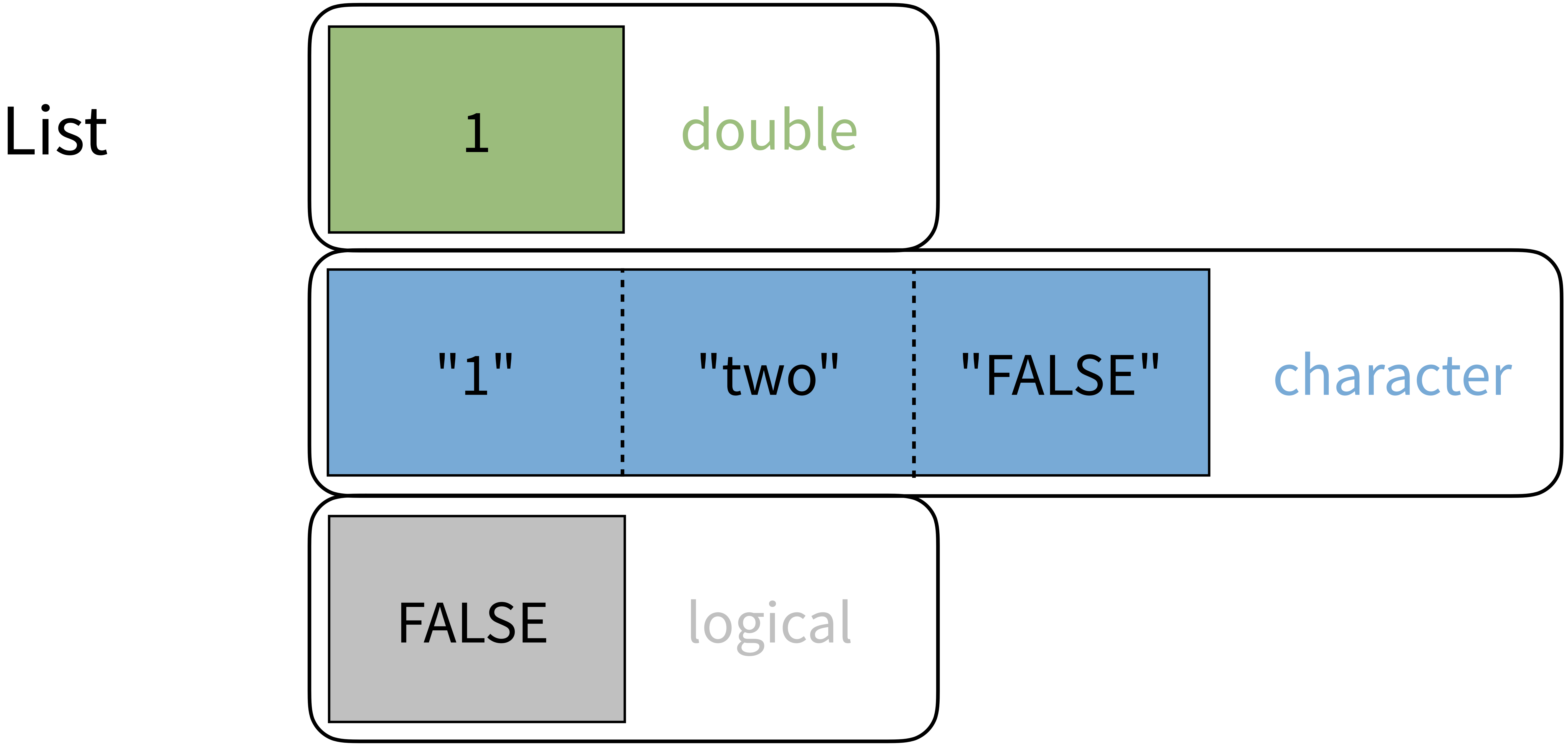
"two" character

FALSE logical

[Adapted from 'Master the tidyverse' CC by RStudio]

Lists

A way to store complicated and possibly heterogeneous objects



List

1

double

"1"

"two"

"FALSE"

character

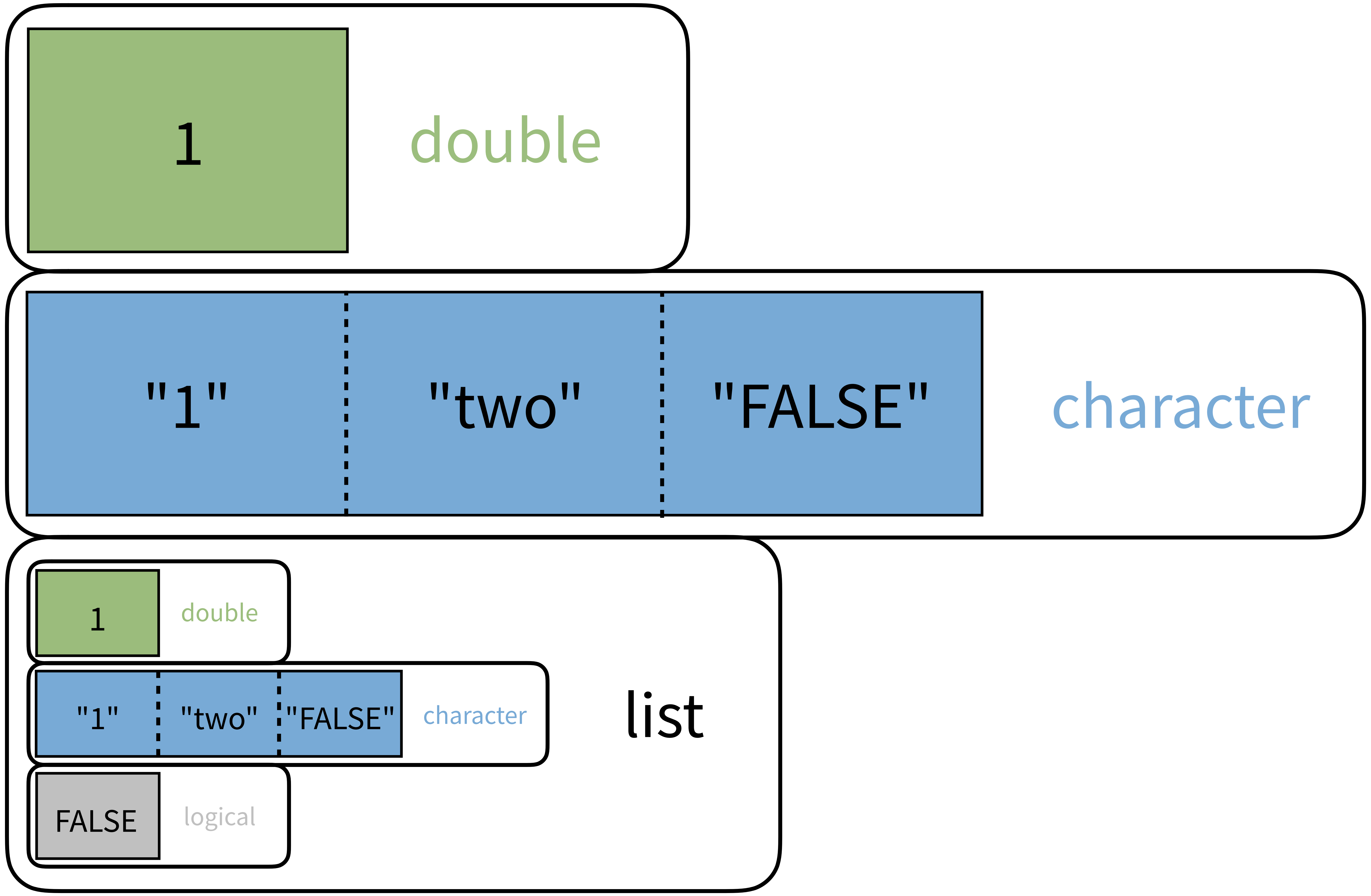
FALSE

logical

Lists

A way to store complicated and possibly heterogeneous objects

List



Introducing purrr::map

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

for each element of `.x` do `.f`

`.x`

- a vector
- **a list**
- a data frame (for each column)

`.f`

We'll get to that...

CC BY Charlotte Wickham

WHAT ARE THE COEFFICIENTS FOR EACH MODEL?

for each model in `three_models`, **tidy the model**

```
map(three_models, ____ )
```

What should we put here?

STRATEGY

1. Do it for one element
2. Turn it into a recipe
3. Use `map()` to do it for all elements

Your Turn 2

```
nz_model <- three_models[[1]]
```

Do it for one element:

Tidy nz_model.

DO IT FOR ONE

Solve the problem for one element

```
nz_model <- three_models[[1]]
```

```
nz_model %>% tidy()
```

CC BY Charlotte Wickham

DO IT FOR ONE

Solve the problem for one element

```
nz_model <- three_models[[1]]
```

```
nz_model %>% tidy()
```

DO IT FOR ONE

Solve the problem for one element

```
canada_model <- three_models[[2]]
```

```
canada_model %>% tidy()
```

DO IT FOR ONE

Solve the problem for one element

```
___ <- three_models[[?]]

___ %>% tidy()
```

TURN IT INTO A RECIPE

Make it a formula

Use .x as a pronoun

```
~ .x %>% tidy()
```

A formula

purrr's "pronoun" for one element of our vector

DO IT FOR ALL!

Your recipe is the second argument to map

```
map( three_models  ,
   ~ .x  %>% tidy()          )
```

A formula

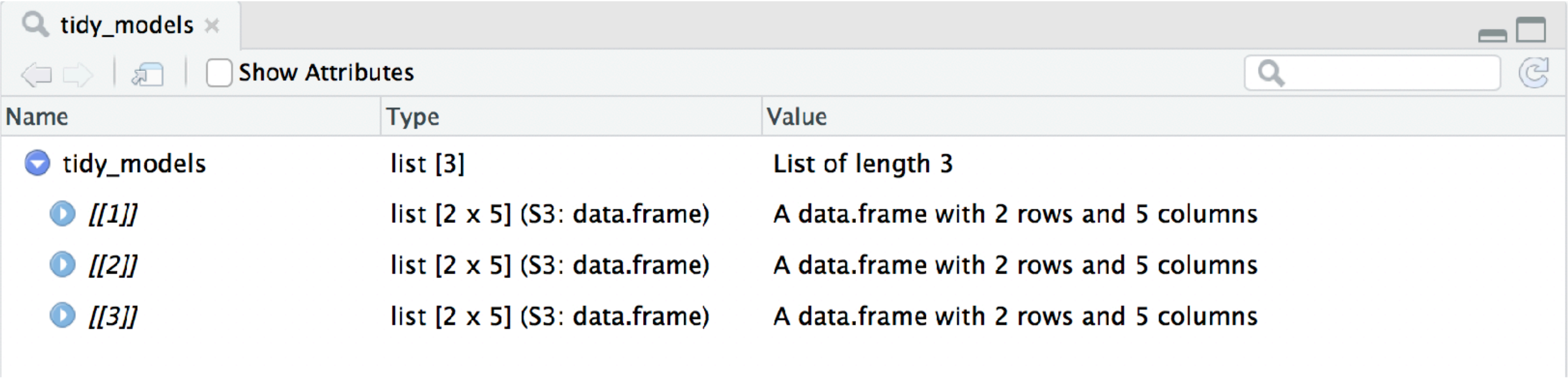
purrr's "pronoun" for
one element of our vector

Your Turn 3

Run the code, what kind of object is tidy_models?

02:00

```
View(tidy_models)
```



Name	Type	Value
tidy_models	list [3]	List of length 3
[[1]]	list [2 x 5] (S3: data.frame)	A data.frame with 2 rows and 5 columns
[[2]]	list [2 x 5] (S3: data.frame)	A data.frame with 2 rows and 5 columns
[[3]]	list [2 x 5] (S3: data.frame)	A data.frame with 2 rows and 5 columns

Another list!

The output of map is **always** another list, the same length as the input.

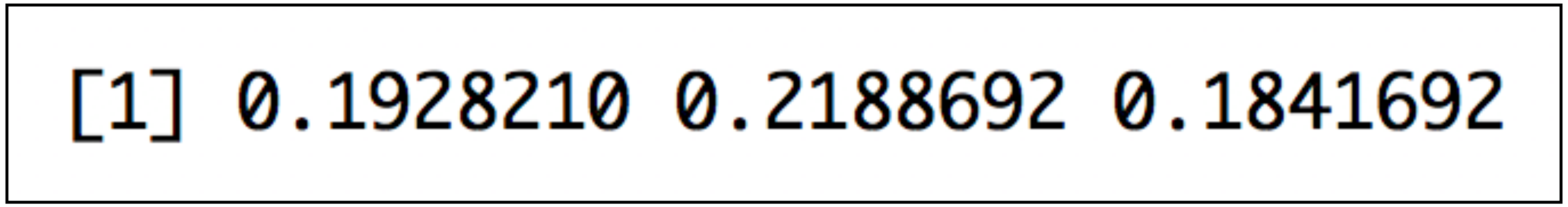
map functions

function	returns results as
map()	list
map_chr()	character vector
map_dbl()	double vector (numeric)
map_int()	integer vector
map_lgl()	logical vector
map_df()	data frame
walk()	nothing

map_dbl()

If we are asking for output that could be a vector:

```
map_dbl(three_models, ~ tidy(.x) %>%
                       filter(term == "year") %>%
                       pull(estimate)
)
```



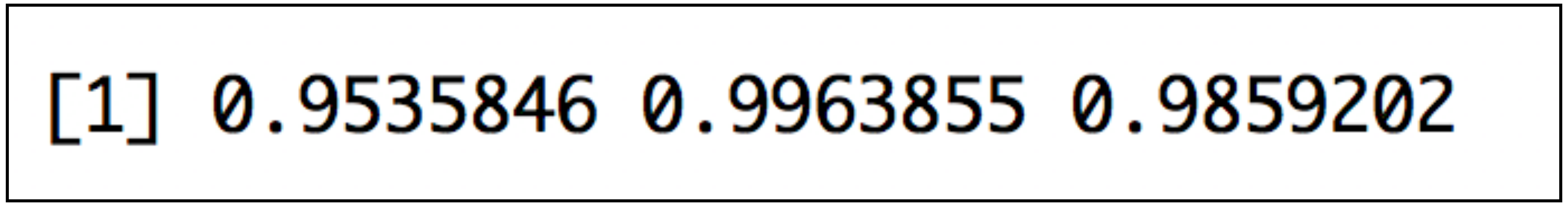
```
[1] 0.1928210 0.2188692 0.1841692
```

Your Turn 4

Edit the code to instead get the r.squared for each model.

02 : 00

```r
map_dbl(three_models, ~ glance(.x) %>%
                        pull(r.squared))
```



```
[1] 0.9535846 0.9963855 0.9859202
```

map functions

single list	two lists	returns results as
map()	map2()	list
map_chr()	map2_chr()	character vector
map_dbl()	map2_dbl()	double vector
map_int()	map2_int()	integer vector
map_lgl()	map2_lgl()	logical vector
map_df()	map2_df()	data frame
walk()	walk2()	nothing

map functions

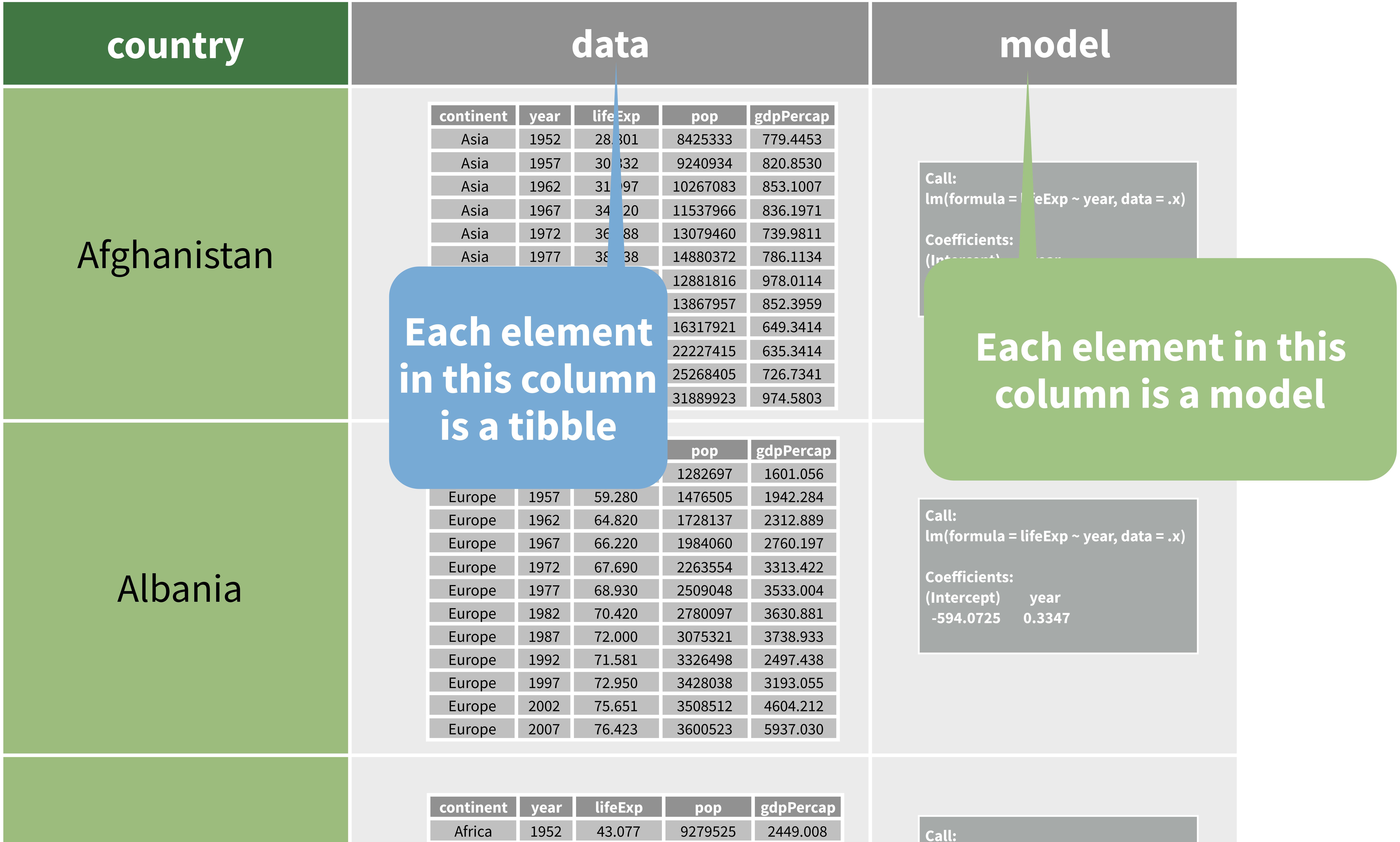
single list	two lists	many lists	returns results as
map()	map2()	pmap()	list
map_chr()	map2_chr()	pmap_chr()	character vector
map_dbl()	map2_dbl()	pmap_dbl()	double vector
map_int()	map2_int()	pmap_int()	integer vector
map_lgl()	map2_lgl()	pmap_lgl()	logical vector
map_df()	map2_df()	pmap_df()	data frame
walk()	walk2()	pwalk()	nothing

List columns

You can put more than just:

- numbers
- logicals, and
- character strings

in tibbles!



country	data	model

Afghanistan

continent	year	lifeExp	pop	gdpPercap
Asia	1952	28. 01	8425333	779.4453
Asia	1957	30 32	9240934	820.8530
Asia	1962	31 97	10267083	853.1007
Asia	1967	34 20	11537966	836.1971
Asia	1972	36 88	13079460	739.9811
Asia	1977	38 38	14880372	786.1134
			12881816	978.0114
			13867957	852.3959
			16317921	649.3414
			22227415	635.3414
			25268405	726.7341
			31889923	974.5803

```
Call:
lm(formula = l eExp ~ year, data = .x)

Coefficients:
(In
```

Albania

			pop	gdpPercap
			1282697	1601.056
Europe	1957	59.280	1476505	1942.284
Europe	1962	64.820	1728137	2312.889
Europe	1967	66.220	1984060	2760.197
Europe	1972	67.690	2263554	3313.422
Europe	1977	68.930	2509048	3533.004
Europe	1982	70.420	2780097	3630.881
Europe	1987	72.000	3075321	3738.933
Europe	1992	71.581	3326498	2497.438
Europe	1997	72.950	3428038	3193.055
Europe	2002	75.651	3508512	4604.212
Europe	2007	76.423	3600523	5937.030

```
Call:
lm(formula = lifeExp ~ year, data = .x)

Coefficients:
(Intercept)       year
  -594.0725     0.3347
```

continent	year	lifeExp	pop	gdpPercap
Africa	1952	43.077	9279525	2449.008

```
Call:
```

Each element in this column is a tibble

Each element in this column is a model

CC by RStudio

Why?

<table>
<tr><th>country</th><th>data</th><th>model</th><th>r.squared</th></tr>
<tr><td>Afghanistan</td><td><table>
<tr><th>continent</th><th>year</th><th>lifeExp</th><th>pop</th><th>gdpPercap</th></tr>
<tr><td>Asia</td><td>1952</td><td>28.801</td><td>8425333</td><td>779.4453</td></tr>
<tr><td>Asia</td><td>1957</td><td>30.332</td><td>9240934</td><td>820.8530</td></tr>
<tr><td>Asia</td><td>1962</td><td>31.997</td><td>10267083</td><td>853.1007</td></tr>
<tr><td>Asia</td><td>1967</td><td>34.020</td><td>11537966</td><td>836.1971</td></tr>
<tr><td>Asia</td><td>1972</td><td>36.088</td><td>13079460</td><td>739.9811</td></tr>
<tr><td>Asia</td><td>1977</td><td>38.438</td><td>14880372</td><td>786.1134</td></tr>
<tr><td>Asia</td><td>1982</td><td>39.854</td><td>12881816</td><td>978.0114</td></tr>
<tr><td>Asia</td><td>1987</td><td>40.822</td><td>13867957</td><td>852.3959</td></tr>
<tr><td>Asia</td><td>1992</td><td>41.674</td><td>16317921</td><td>649.3414</td></tr>
<tr><td>Asia</td><td>1997</td><td>41.763</td><td>22227415</td><td>635.3414</td></tr>
<tr><td>Asia</td><td>2002</td><td>42.129</td><td>25268405</td><td>726.7341</td></tr>
<tr><td>Asia</td><td>2007</td><td>43.828</td><td>31889923</td><td>974.5803</td></tr>
</table></td><td>Call:
lm(formula = lifeExp ~ year, data = .x)

Coefficients:
(Intercept)   year
-507.5343   0.2753</td><td>0.034</td></tr>
<tr><td>Albania</td><td><table>
<tr><td>Europe</td><td>1982</td><td>70.420</td><td>2780097</td><td>3630.881</td></tr>
<tr><td>Europe</td><td>1987</td><td>72.000</td><td>3075321</td><td>3738.933</td></tr>
<tr><td>Europe</td><td>1992</td><td>71.581</td><td>3326498</td><td>2497.438</td></tr>
<tr><td>Europe</td><td>1997</td><td>72.950</td><td>3428038</td><td>3193.055</td></tr>
<tr><td>Europe</td><td>2002</td><td>75.651</td><td>3508512</td><td>4604.212</td></tr>
<tr><td>Europe</td><td>2007</td><td>76.423</td><td>3600523</td><td>5937.030</td></tr>
</table></td><td>(Intercept)   year
-594.0725   0.3347</td><td>0.493</td></tr>
<tr><td></td><td><table>
<tr><th>continent</th><th>year</th><th>lifeExp</th><th>pop</th><th>gdpPercap</th></tr>
<tr><td>Africa</td><td>1952</td><td>43.077</td><td>9279525</td><td>2449.008</td></tr>
</table></td><td>Call:</td><td></td></tr>
</table>

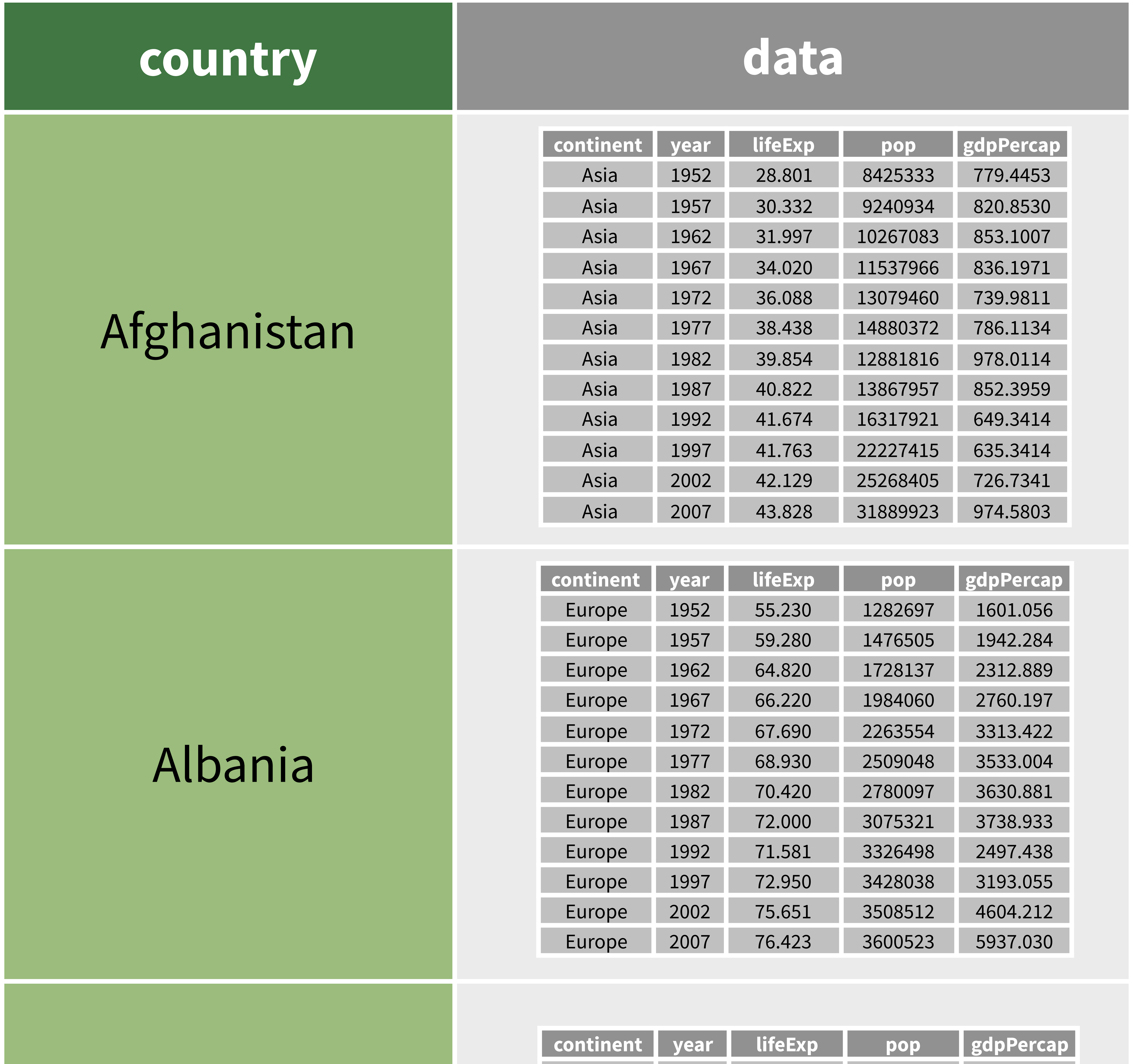
Organization.

We keep the things that are related together.

nest()

Places grouped cases into a list column.

```
gapminder %>%
  group_by(country) %>%
  nest()
```



country	data
Afghanistan	see table below

continent	year	lifeExp	pop	gdpPercap
Asia	1952	28.801	8425333	779.4453
Asia	1957	30.332	9240934	820.8530
Asia	1962	31.997	10267083	853.1007
Asia	1967	34.020	11537966	836.1971
Asia	1972	36.088	13079460	739.9811
Asia	1977	38.438	14880372	786.1134
Asia	1982	39.854	12881816	978.0114
Asia	1987	40.822	13867957	852.3959
Asia	1992	41.674	16317921	649.3414
Asia	1997	41.763	22227415	635.3414
Asia	2002	42.129	25268405	726.7341
Asia	2007	43.828	31889923	974.5803

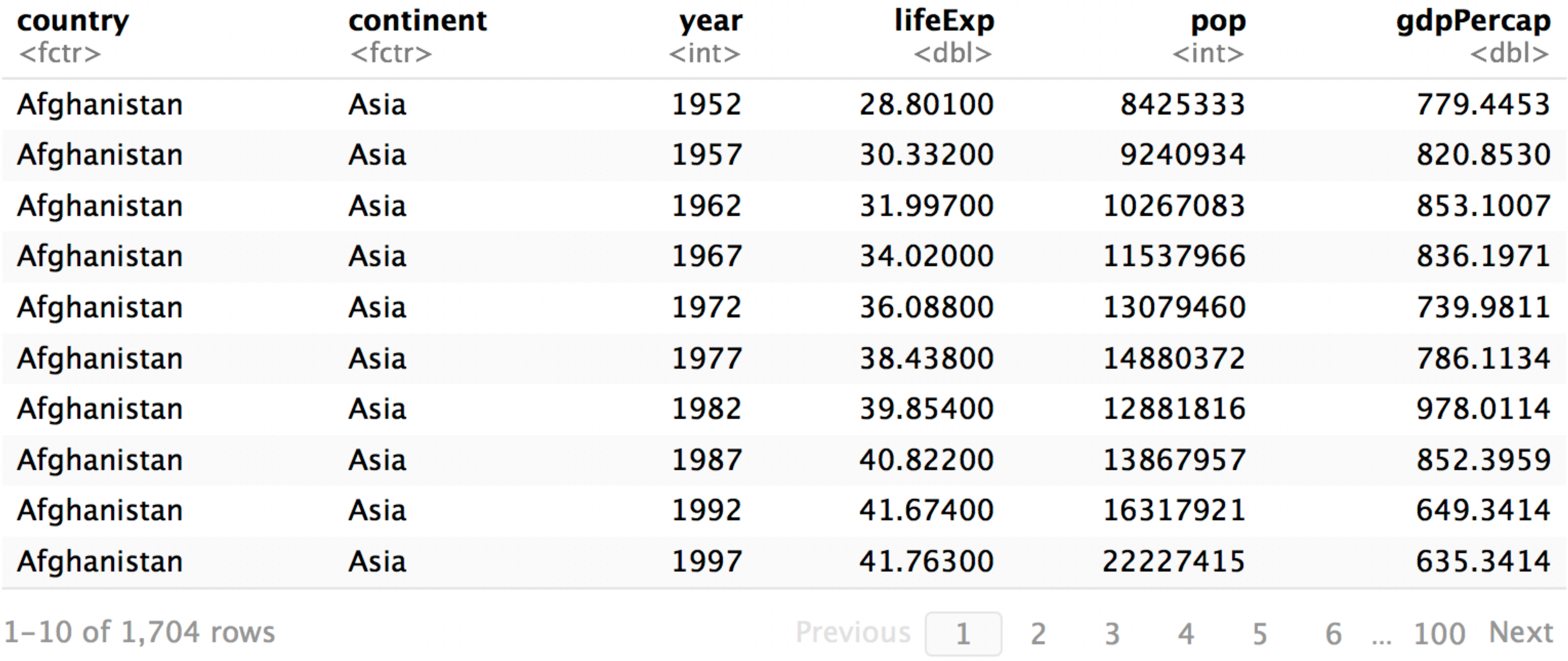
Albania

continent	year	lifeExp	pop	gdpPercap
Europe	1952	55.230	1282697	1601.056
Europe	1957	59.280	1476505	1942.284
Europe	1962	64.820	1728137	2312.889
Europe	1967	66.220	1984060	2760.197
Europe	1972	67.690	2263554	3313.422
Europe	1977	68.930	2509048	3533.004
Europe	1982	70.420	2780097	3630.881
Europe	1987	72.000	3075321	3738.933
Europe	1992	71.581	3326498	2497.438
Europe	1997	72.950	3428038	3193.055
Europe	2002	75.651	3508512	4604.212
Europe	2007	76.423	3600523	5937.030

continent	year	lifeExp	pop	gdpPercap

Adapted from 'Master the tidyverse' CC by RStudio

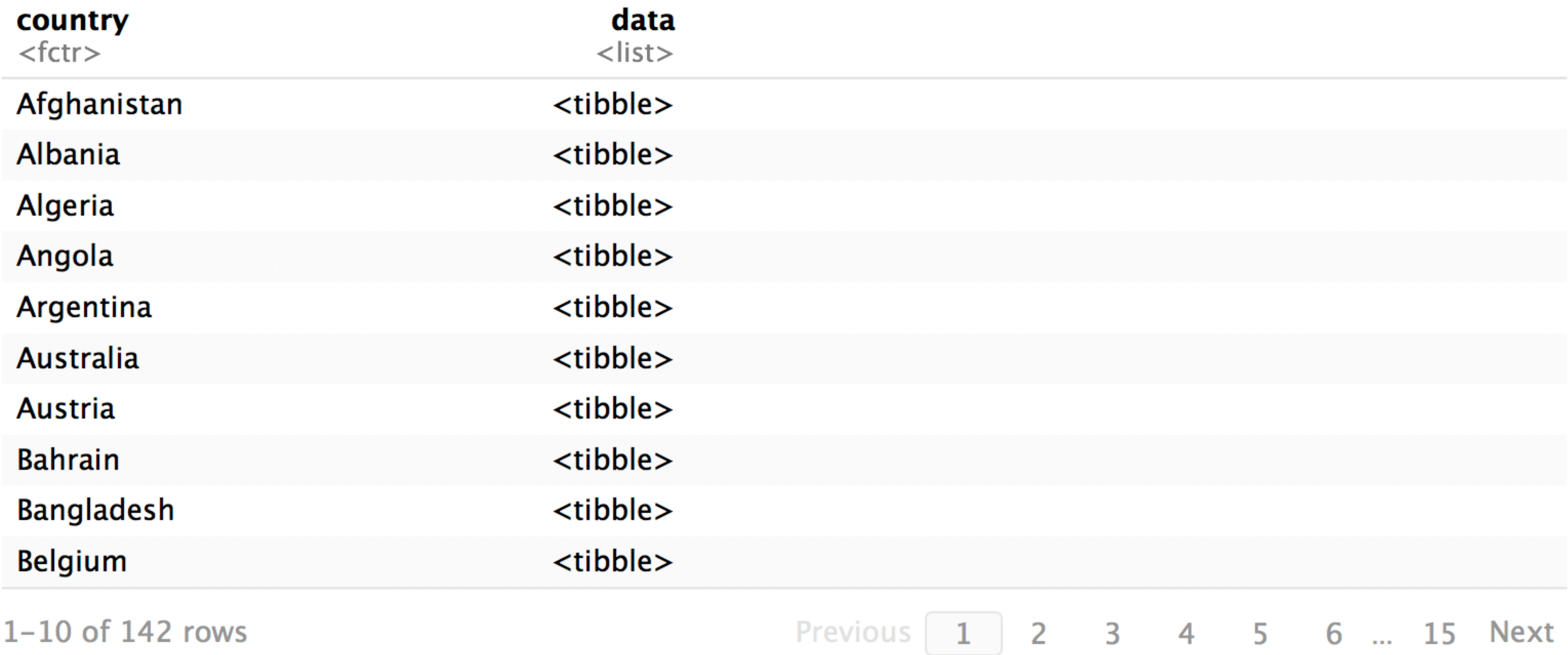
```
gapminder
```



country	continent	year	lifeExp	pop	gdpPercap
<fctr>	<fctr>	<int>	<dbl>	<int>	<dbl>
Afghanistan	Asia	1952	28.80100	8425333	779.4453
Afghanistan	Asia	1957	30.33200	9240934	820.8530
Afghanistan	Asia	1962	31.99700	10267083	853.1007
Afghanistan	Asia	1967	34.02000	11537966	836.1971
Afghanistan	Asia	1972	36.08800	13079460	739.9811
Afghanistan	Asia	1977	38.43800	14880372	786.1134
Afghanistan	Asia	1982	39.85400	12881816	978.0114
Afghanistan	Asia	1987	40.82200	13867957	852.3959
Afghanistan	Asia	1992	41.67400	16317921	649.3414
Afghanistan	Asia	1997	41.76300	22227415	635.3414

1–10 of 1,704 rows

```r
gapminder_nested <- gapminder %>%
  group_by(country) %>%
  nest()
gapminder_nested
```



country <fctr>	data <list>
Afghanistan	<tibble>
Albania	<tibble>
Algeria	<tibble>
Angola	<tibble>
Argentina	<tibble>
Australia	<tibble>
Austria	<tibble>
Bahrain	<tibble>
Bangladesh	<tibble>
Belgium	<tibble>

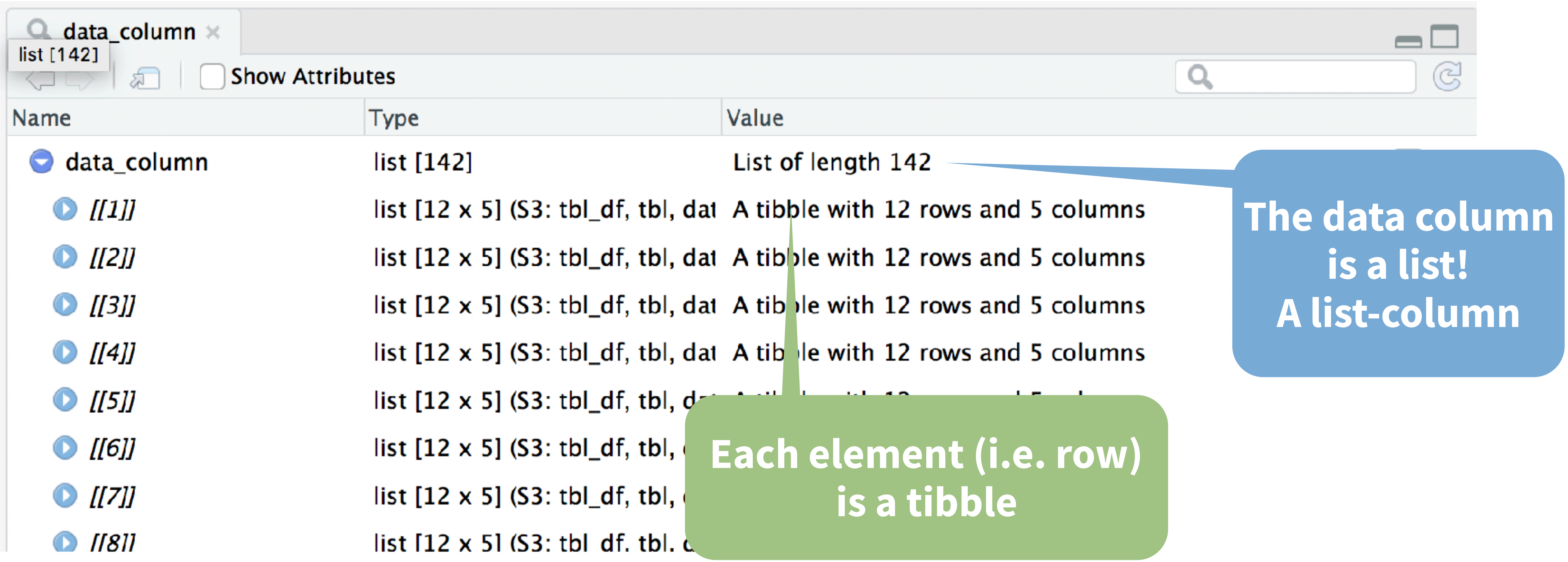
1-10 of 142 rows

Adapted from

Your Turn 5

Run the chunk: it pulls out the data column into `data_column`.

Use the object explorer to take a look at `data_column`. What kind of object is it?



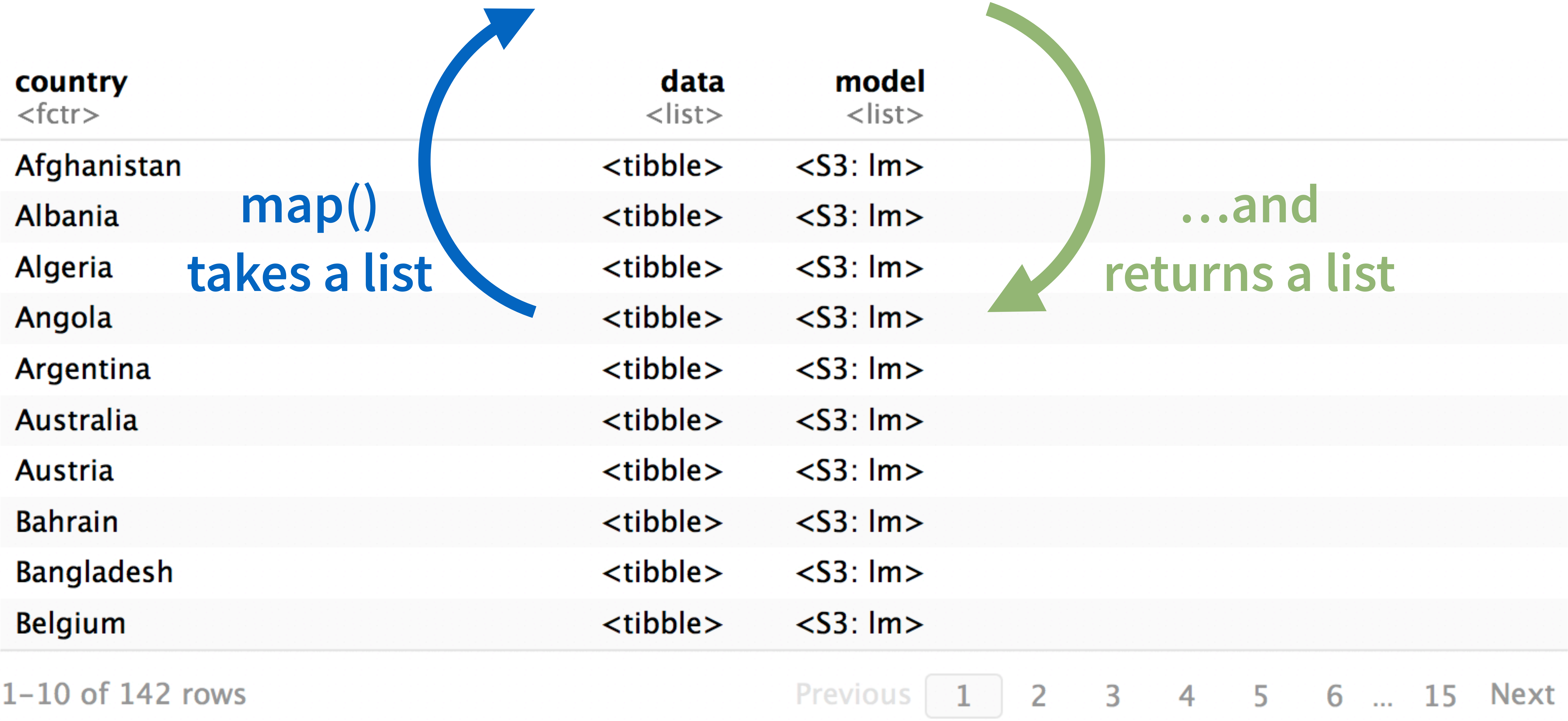
Name	Type	Value
data_column	list [142]	List of length 142
[[1]]	list [12 x 5] (S3: tbl_df, tbl, dat	A tibble with 12 rows and 5 columns
[[2]]	list [12 x 5] (S3: tbl_df, tbl, dat	A tibble with 12 rows and 5 columns
[[3]]	list [12 x 5] (S3: tbl_df, tbl, dat	A tibble with 12 rows and 5 columns
[[4]]	list [12 x 5] (S3: tbl_df, tbl, dat	A tibble with 12 rows and 5 columns
[[5]]	list [12 x 5] (S3: tbl_df, tbl, d	
[[6]]	list [12 x 5] (S3: tbl_df, tbl,	
[[7]]	list [12 x 5] (S3: tbl_df, tbl,	
[[8]]	list [12 x 5] (S3: tbl_df, tbl,	

The data column is a list!
A list-column

Each element (i.e. row) is a tibble

CC by RStudio

```r
gapminder_nested <- gapminder_nested %>%
  mutate(model = map(data, ~ lm(lifeExp ~ year, data = .x)))
```



map()
takes a list

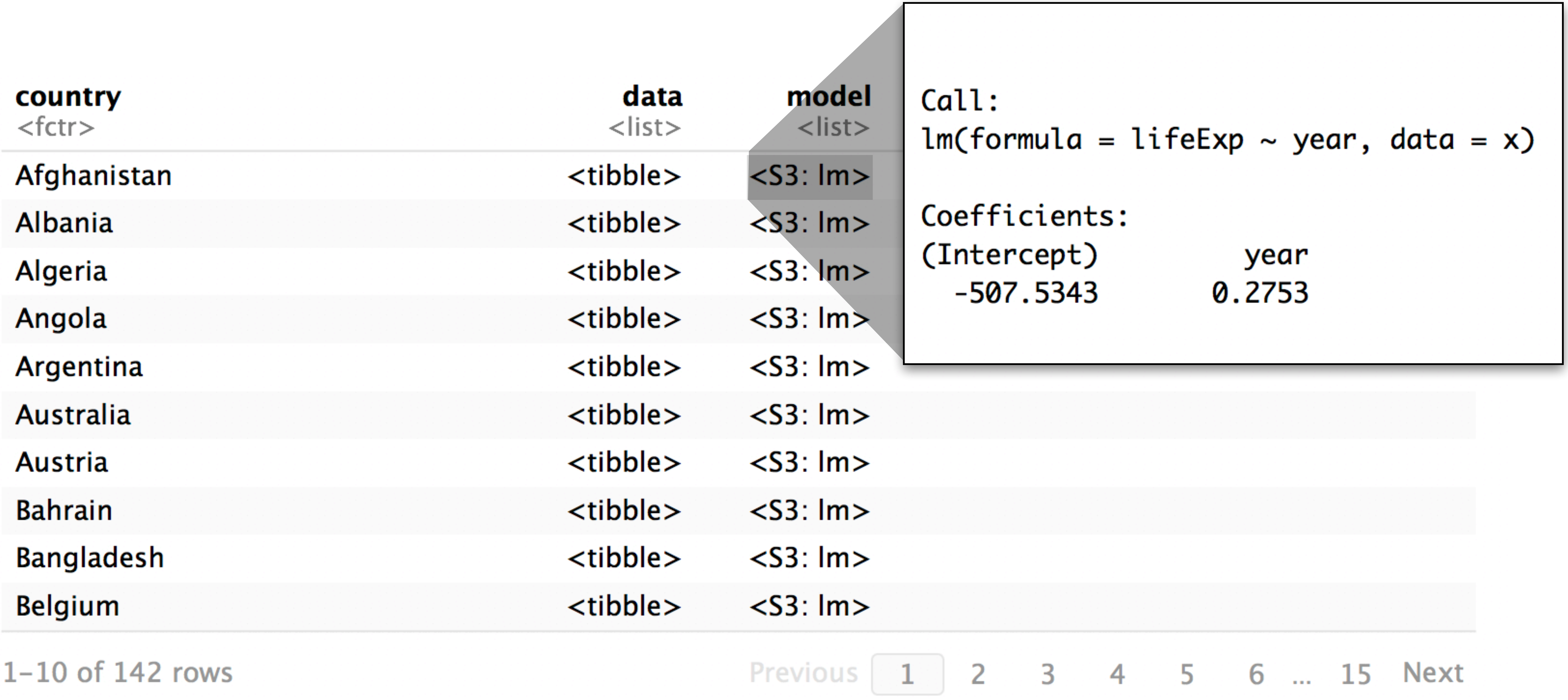
...and
returns a list

country <fctr>	data <list>	model <list>
Afghanistan	<tibble>	<S3: lm>
Albania	<tibble>	<S3: lm>
Algeria	<tibble>	<S3: lm>
Angola	<tibble>	<S3: lm>
Argentina	<tibble>	<S3: lm>
Australia	<tibble>	<S3: lm>
Austria	<tibble>	<S3: lm>
Bahrain	<tibble>	<S3: lm>
Bangladesh	<tibble>	<S3: lm>
Belgium	<tibble>	<S3: lm>

1–10 of 142 rows

Adap

```r
gapminder_nested %>% pull(model) %>% pluck(1)
```



country	data	model
<fctr>	<list>	<list>
Afghanistan	<tibble>	<S3: lm>
Albania	<tibble>	<S3: lm>
Algeria	<tibble>	<S3: lm>
Angola	<tibble>	<S3: lm>
Argentina	<tibble>	<S3: lm>
Australia	<tibble>	<S3: lm>
Austria	<tibble>	<S3: lm>
Bahrain	<tibble>	<S3: lm>
Bangladesh	<tibble>	<S3: lm>
Belgium	<tibble>	<S3: lm>

```

Call:
lm(formula = lifeExp ~ year, data = x)

Coefficients:
(Intercept)         year
  -507.5343       0.2753
```

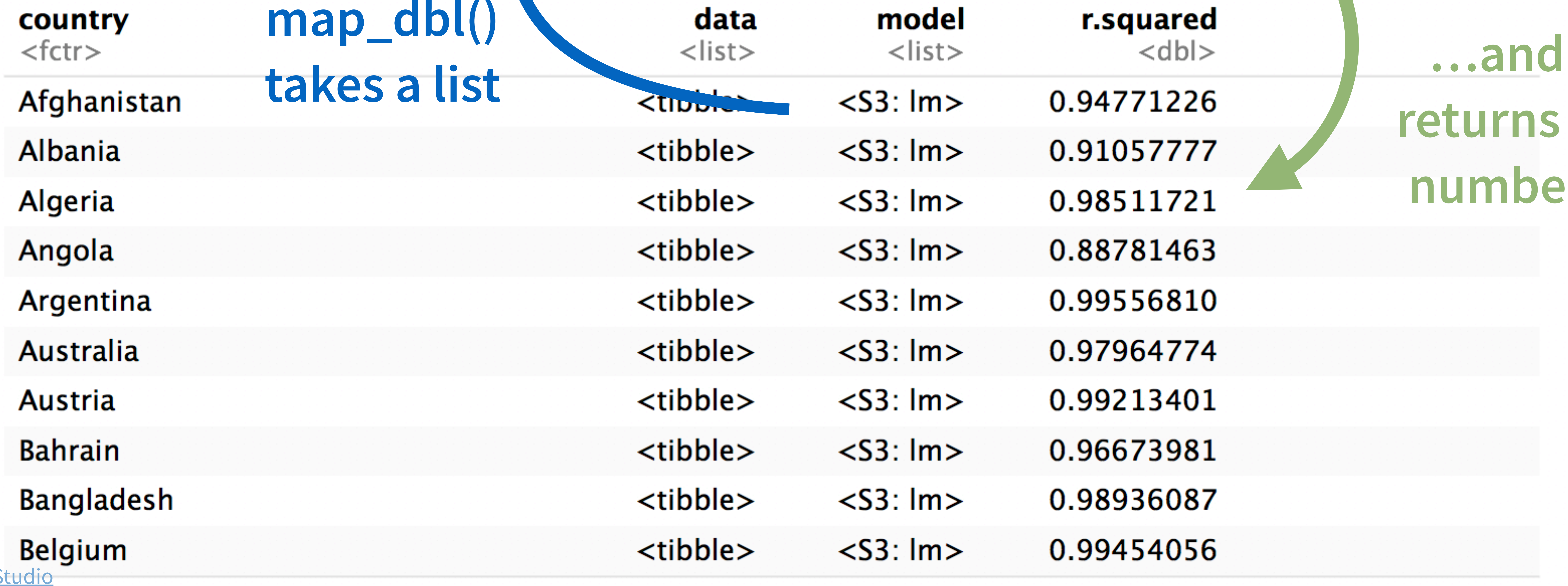
1–10 of 142 rows

CC by RStudio

```
gapminder_nested <- gapminder_nested %>%
  mutate(r.squared = map_dbl(model,
                             ~ glance(.x) %>% pull(r.squared)))
```

map_dbl() takes a list

…and returns a number



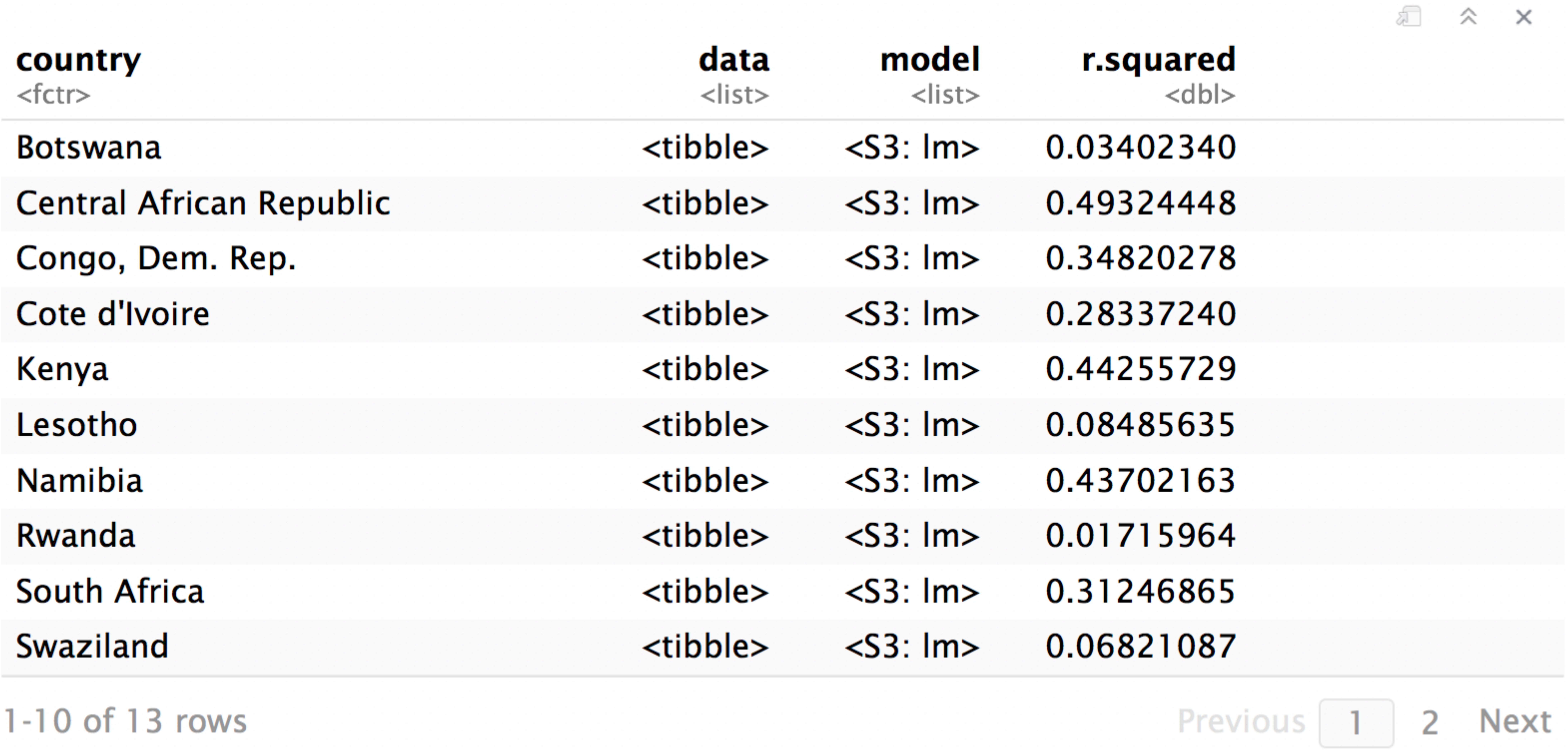
country <fctr>	data <list>	model <list>	r.squared <dbl>
Afghanistan	<tibble>	<S3: lm>	0.94771226
Albania	<tibble>	<S3: lm>	0.91057777
Algeria	<tibble>	<S3: lm>	0.98511721
Angola	<tibble>	<S3: lm>	0.88781463
Argentina	<tibble>	<S3: lm>	0.99556810
Australia	<tibble>	<S3: lm>	0.97964774
Austria	<tibble>	<S3: lm>	0.99213401
Bahrain	<tibble>	<S3: lm>	0.96673981
Bangladesh	<tibble>	<S3: lm>	0.98936087
Belgium	<tibble>	<S3: lm>	0.99454056

Your Turn 6

(Make sure you run all the chunks before this one, then)

Filter gapminder_nested to find the countries with r.squared less than 0.5.

```
gapminder_nested %>%
  filter(r.squared < 0.5)
```



country <fctr>	data <list>	model <list>	r.squared <dbl>
Botswana	<tibble>	<S3: lm>	0.03402340
Central African Republic	<tibble>	<S3: lm>	0.49324448
Congo, Dem. Rep.	<tibble>	<S3: lm>	0.34820278
Cote d'Ivoire	<tibble>	<S3: lm>	0.28337240
Kenya	<tibble>	<S3: lm>	0.44255729
Lesotho	<tibble>	<S3: lm>	0.08485635
Namibia	<tibble>	<S3: lm>	0.43702163
Rwanda	<tibble>	<S3: lm>	0.01715964
South Africa	<tibble>	<S3: lm>	0.31246865
Swaziland	<tibble>	<S3: lm>	0.06821087

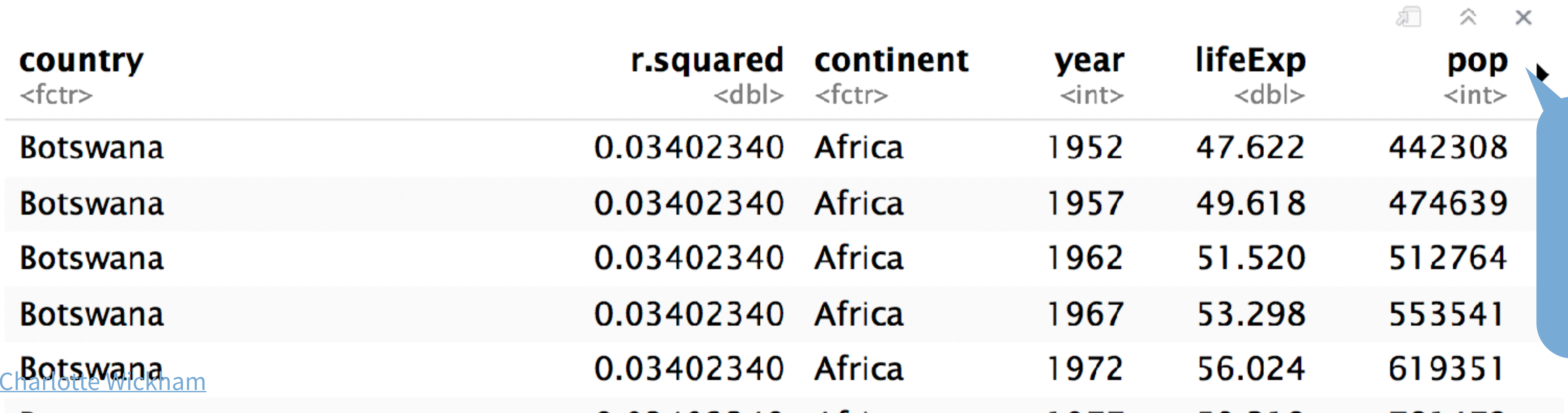
1-10 of 13 rows

unnest()

```
poor_fit <- gapminder_nested %>%
  filter(r.squared < 0.5)

gapminder_nested %>% unnest(data)
```

Column to unnest

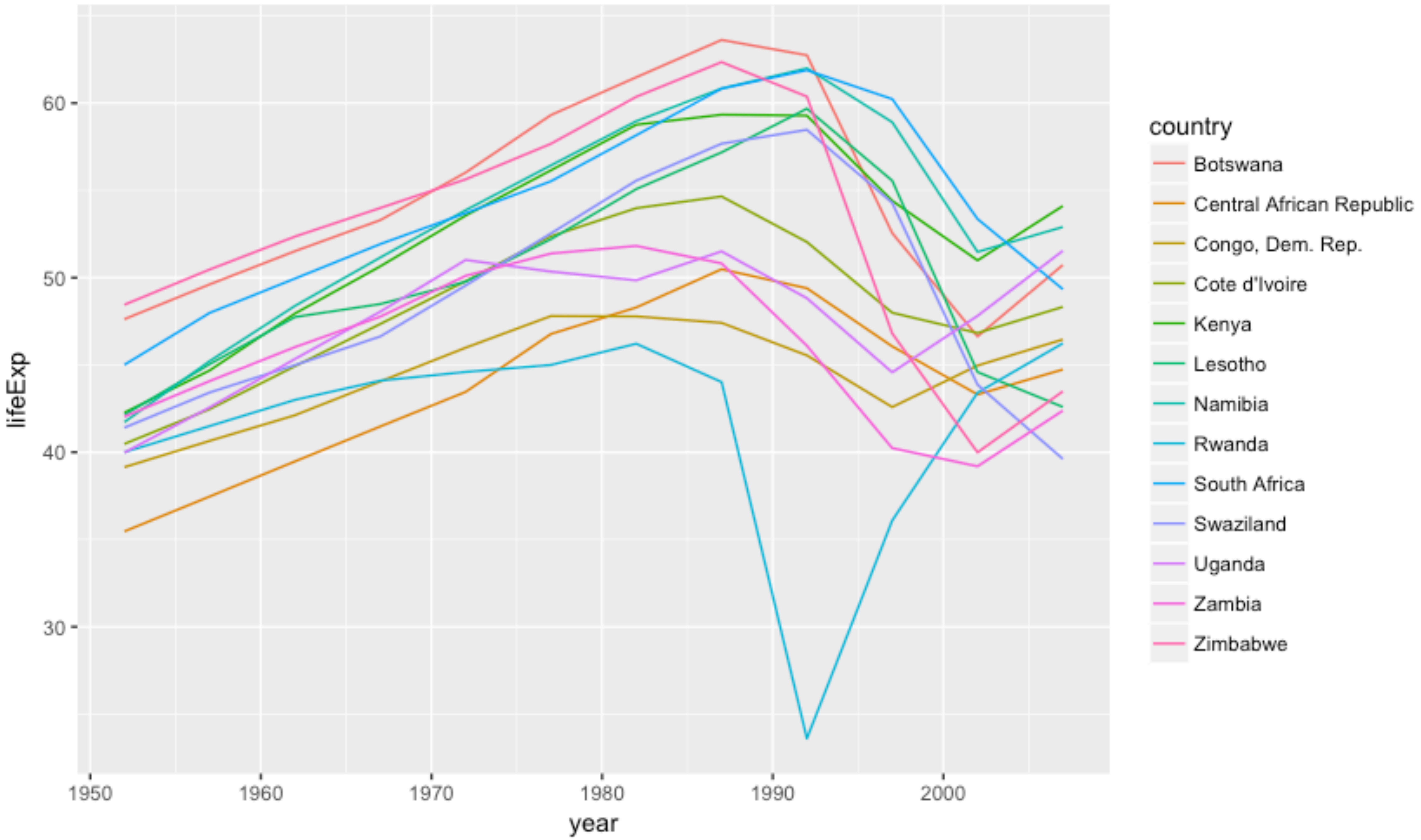


country	r.squared	continent	year	lifeExp	pop
<fctr>	<dbl>	<fctr>	<int>	<dbl>	<int>
Botswana	0.03402340	Africa	1952	47.622	442308
Botswana	0.03402340	Africa	1957	49.618	474639
Botswana	0.03402340	Africa	1962	51.520	512764
Botswana	0.03402340	Africa	1967	53.298	553541
Botswana	0.03402340	Africa	1972	56.024	619351

Columns from inside data

CC BY Charlotte Wickham

```
unnest(poor_fit, data) %>%
  ggplot(aes(x = year, y = lifeExp)) +
    geom_line(aes(color = country))
```

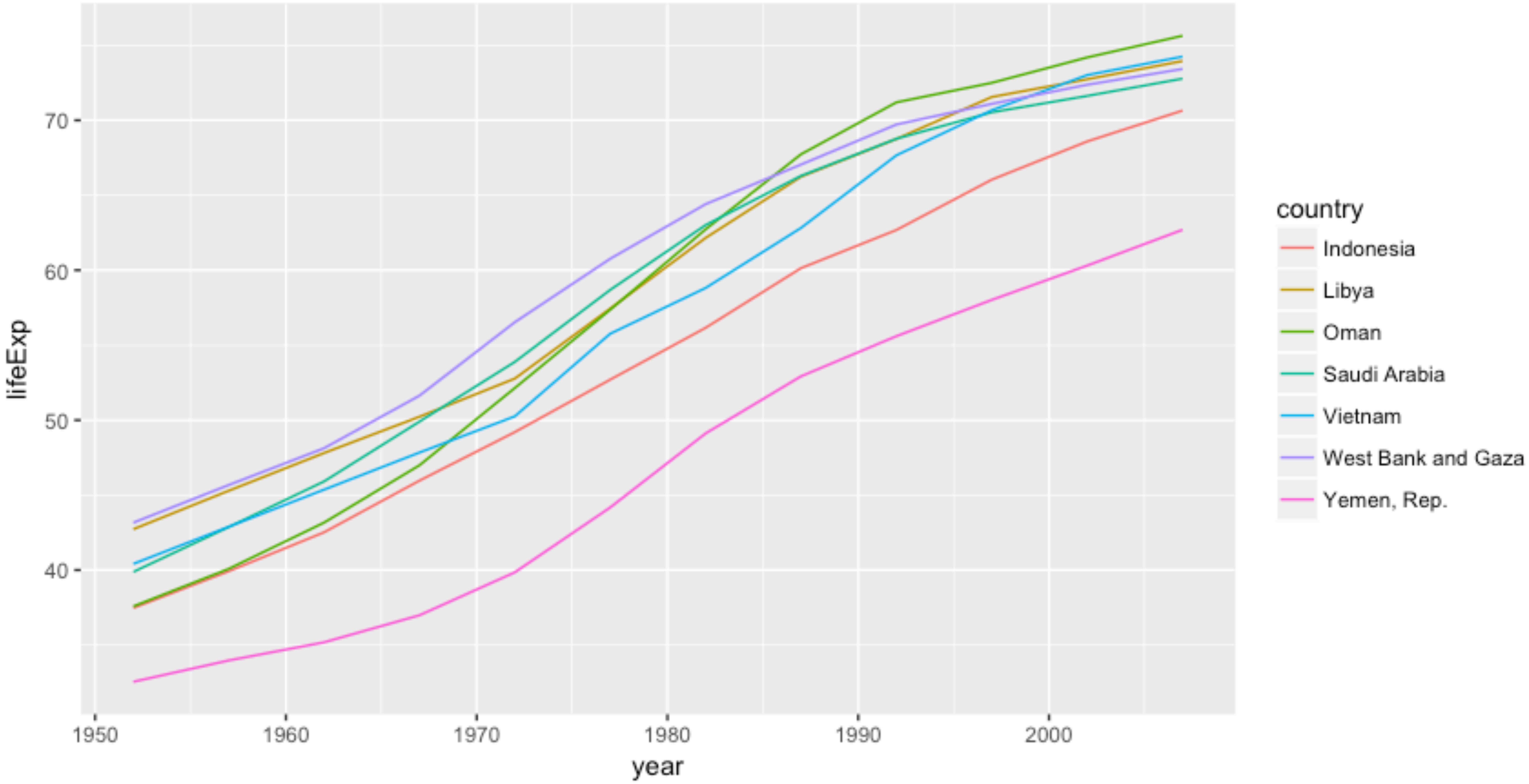


Your Turn 7

Write code to:

- Add a column to `gapminder_nested` with the slope coefficient from the model
- Filter to find countries with a slope above 0.6 years/year.
- Plot these countries over time.

```
gapminder_nested <- gapminder_nested %>%
  mutate(slope = map_dbl(model,
                        ~ tidy(.x) %>% filter(term == "year") %>%
pull(estimate)))
gapminder_nested %>%
  filter(slope > 0.60) %>%
  unnest(data) %>%
  ggplot(aes(x = year, y = lifeExp)) +
    geom_line(aes(color = country))
```



CC BY Charlotte Wickham

Take Away

A table is …an organizational structure …that you can manipulate.

country	r.squared	data	model
Botswana	0.03	year .resid 1952 -5.3071154 1957 -3.6144580 1962 -2.0158007 1967 -0.5411434 1972 1.8815140 1977 4.8731713 1982 6.7348287 1987 8.5694860 1992 7.3891434 1997 -3.1031993 2002 -9.3285420 2007 -5.5378846	Call: lm(formula = lifeExp ~ year, data = .) Coefficients: (Intercept) year -65.49586 0.06067
Lesotho	0.08	year .resid 1952 -5.2410256 1957 -2.8098543 1962 -0.5876830 1967 -0.3205117 1972 0.4766597 1977 2.4398310 1982 4.8320023 1987 6.4561737 1992 8.4833450 1997 3.8785163 2002 -7.5643124 2007 -10.0431410	Call: lm(formula = lifeExp ~ year, data = .) Coefficients: (Intercept) year -139.16529 0.09557