Unit 1: Simple Neural Networks

2. R, Rstudio, and GitHub

9/3/2020

Office hours poll

A Short technical primer

1. **R is an awesome language for rapid prototyping**

2. **RStudio makes it easy to integrate R with a bunch of other useful frameworks**

3. **GitHub is a powerful and flexible solution for version control and code sharing**

Why R?

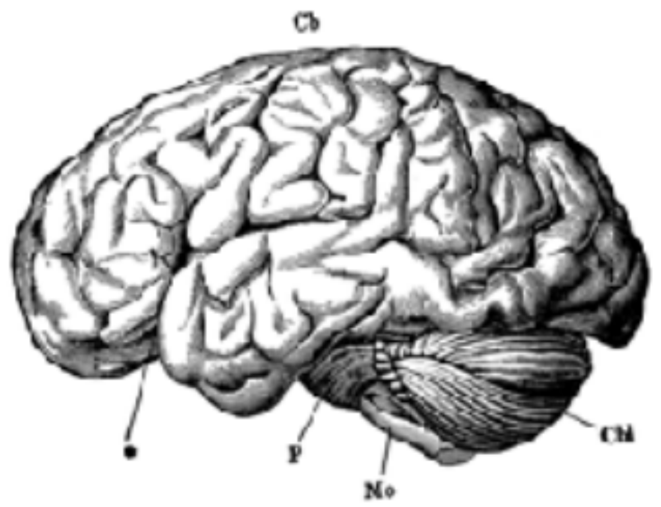
1. **Free and open source**
   good for you, and also *other people can use your code*.

2. **Big community of users across fields**

3. **Awesome plotting, data manipulation, and other packages**

5. **Tries to optimize for rapid prototyping**

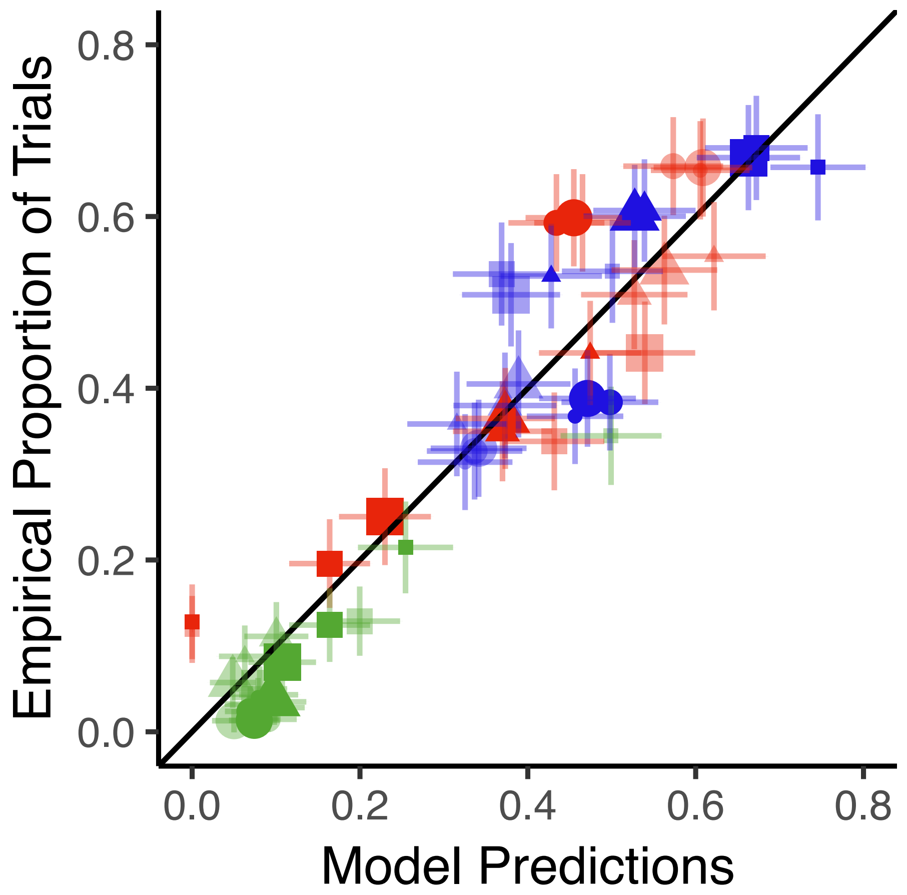
From an idea to a working model



A vague idea

specifying, clarifying, testing, iterating, etc…

computation



time

We should optimize for **human thought**, not **computation**

From an idea to a working model

R has most of the features you know and love:

- Iterative control structure (e.g. `for`, `while`)
- Functional programming (e.g. `map`)
- Objects (e.g. `structure`)

But, the best thing about R (in my opinion) is the suite of packages in the `tidyverse`

tidyverse
:
%>%
Ceci n'est pas un pipe.
TIBBLE
dplyr
ggplot2
readr
purrr

very basics of R

```
foo <- “hello"
```

This is how you assign values to variables.
You can use = instead of <-, but you shouldn’t!
The asymmetry between the two sides is cognitively clearer

```
foo <- c(1,2,3,4)
```

c is for concatenate. You use it to make lists

```
5 >= 7
```

Returns `FALSE`

```
is.character(“the”)
```

Returns TRUE

```
paste(foo, “world”)
```

Returns “hello world”. Because we `foo` has the value “hello”

magrittr: the pipe operator (%>%)

```
foo <- “hello"

bar <- paste(foo, “world”)

baz <- paste(bar, “from 85426”)
```

baz returns “hello world from 85426”

```
baz <- paste(paste(“hello”, “world”), “from 85427”)
```

```
y <- f(g(x))
```

magrittr: the pipe operator (%>%)

```
foo <- “hello"

bar <- paste(foo, “world”)

baz <- paste(bar, “from 85426”)
```

baz returns “hello world from 85426”

```
baz <- “hello” %>%
            paste(“world”) %>%
            paste(“from 85426”)
```

```
y <- x %>% f %>% g
```

tibble: a human readable, general datastructure

```
days <- c(“monday”, “tuesday”, “wednesday”)
meanings <- c(“moon”, “Tiu”, “Woden”)

origins <- tibble(day = days,
                  meaning = meanings)


origins$meaning
origins %>% pull(meaning)
```

day	*meaning*
monday	*moon*
tuesday	*Tiu*
wednesday	*Woden*

returns c(“moon”, “Tiu”, “Woden”)

dplyr: verbs for working with data

`group_by:` whatever you’re going to do next, do it separately for each group.

`select`: keep just a subset of the columns in a tibble

`filter`: keep just the rows of a tibble whose values in one or more columns match some truth condition (e.g. `day == “monday")`

`mutate`: apply an operation to one or more columns (e.g. `as.numeric`, `log, etc.`)

`summarise`: apply an operation to one or more columns that produces a single number (e.g. sum, mean, etc.)

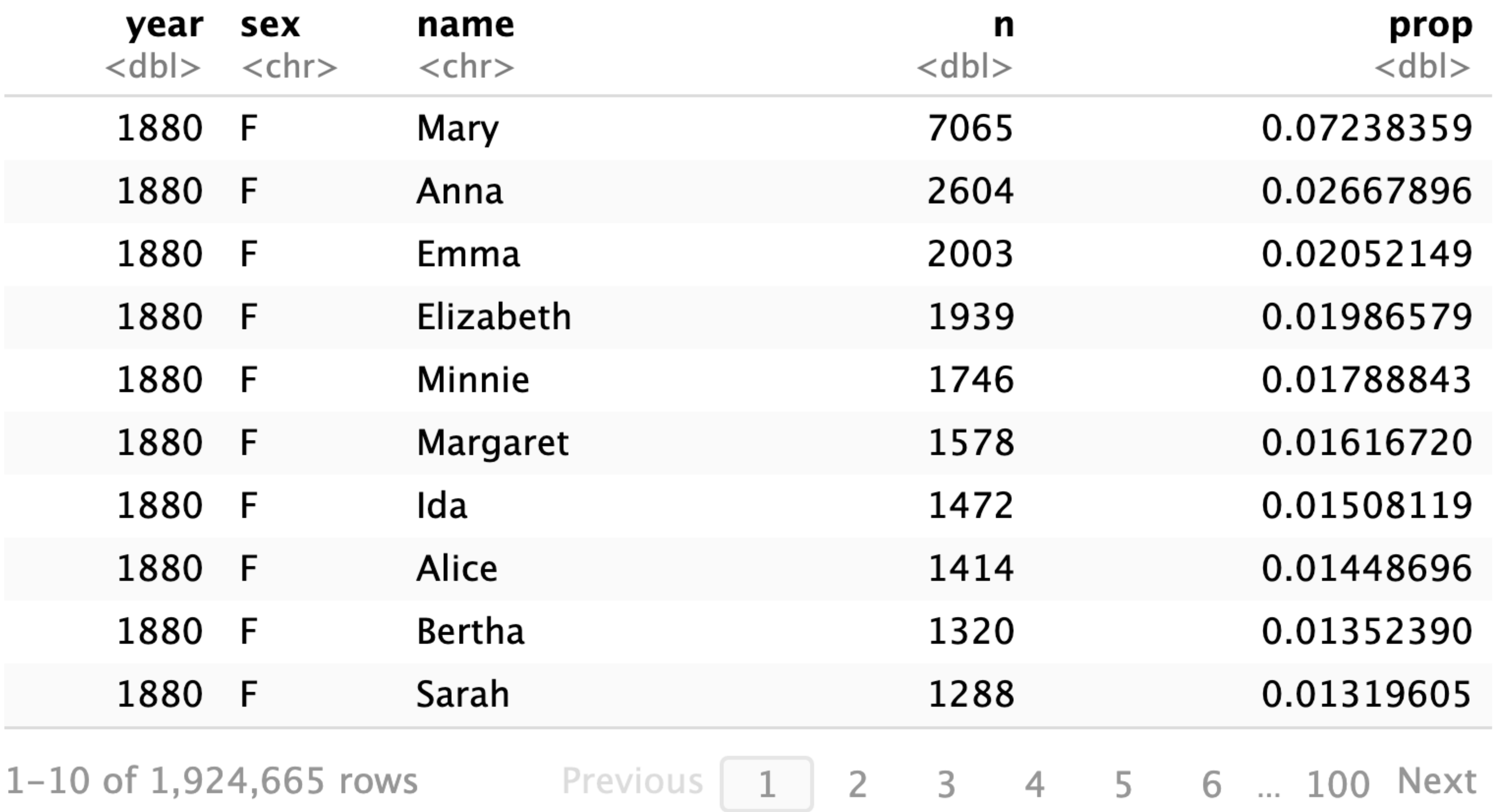
Transform Data with

Slides from “Remaster the tidyverse” by Garret Grolemund

https://github.com/rstudio-education/remaster-the-tidyverse

Slides

```
install.packages(“babynames”)
library(babynames)
babynames
```



year <dbl>	sex <chr>	name <chr>	n <dbl>	prop <dbl>
1880	F	Mary	7065	0.07238359
1880	F	Anna	2604	0.02667896
1880	F	Emma	2003	0.02052149
1880	F	Elizabeth	1939	0.01986579
1880	F	Minnie	1746	0.01788843
1880	F	Margaret	1578	0.01616720
1880	F	Ida	1472	0.01508119
1880	F	Alice	1414	0.01448696
1880	F	Bertha	1320	0.01352390
1880	F	Sarah	1288	0.01319605

1–10 of 1,924,665 rows

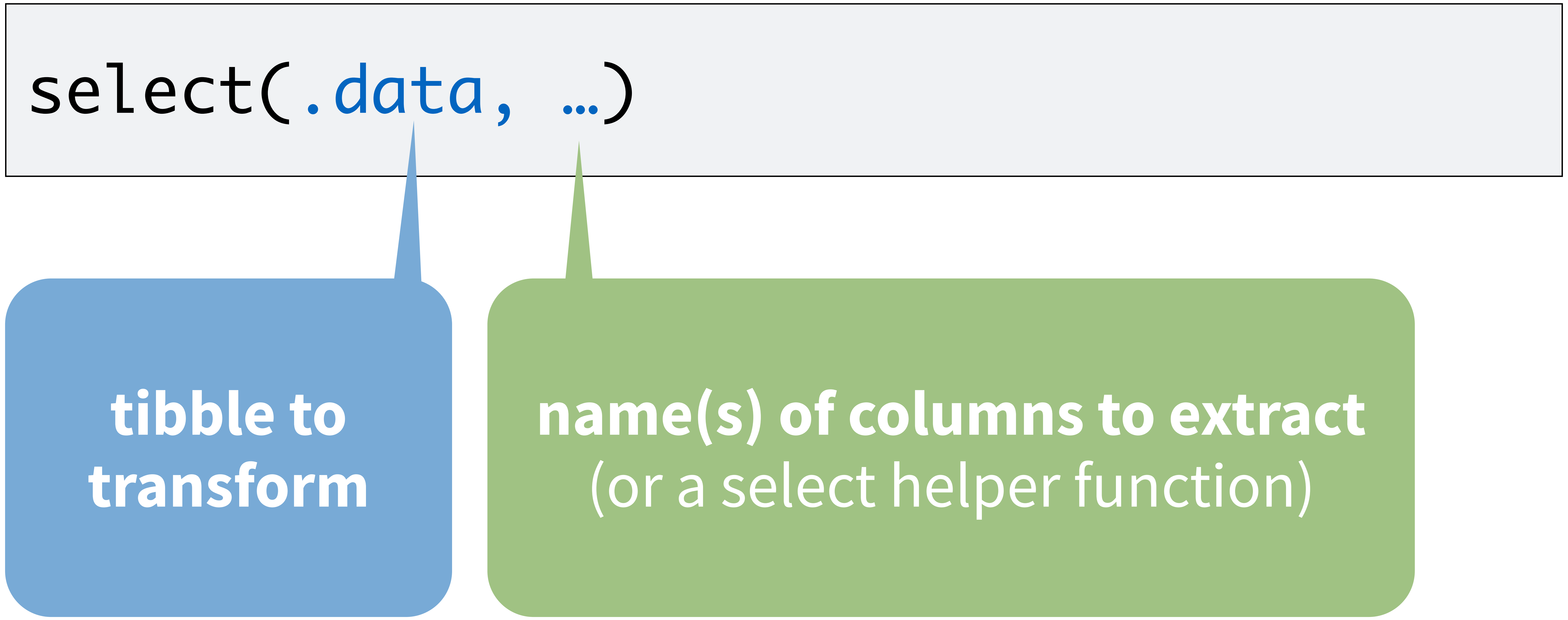
How to isolate?

year	sex	name	n	prop
1880	M	John	9655	0.0815
1880	M	William	9532	0.0805
1880	M	James	5927	0.0501
1880	M	Charles	5348	0.0451
1880	M	Garrett	13	0.0001
1881	M	John	8769	0.081
1881	M	William	8524	0.0787
1881	M	James	5442	0.0503
1881	M	Charles	4664	0.0431
1881	M	Garrett	7	0.0001
1881	M	Gideon	7	0.0001

year	sex	name	n	prop
1880	M	Garrett	13	0.0001
1881	M	Garrett	7	0.0001
…	…	Garrett	…	…

select()

Extract columns by name.



```
select(.data, …)
```

tibble to transform

name(s) of columns to extract
(or a select helper function)

select()

Extract columns by name.

```
select(babynames, name, prop)
```

babynames

year	sex	name	n	prop
1880	M	John	9655	0.0815
1880	M	William	9532	0.0805
1880	M	James	5927	0.0501
1880	M	Charles	5348	0.0451
1880	M	Garrett	13	0.0001
1881	M	John	8769	0.081

→

name	prop
John	0.0815
William	0.0805
James	0.0501
Charles	0.0451
Garrett	0.0001
John	0.081

select() helpers

: - Select range of columns

```
select(mpg, cty:class)
```

- - Select every column but

```
select(mpg, -c(cty, hwy))
```

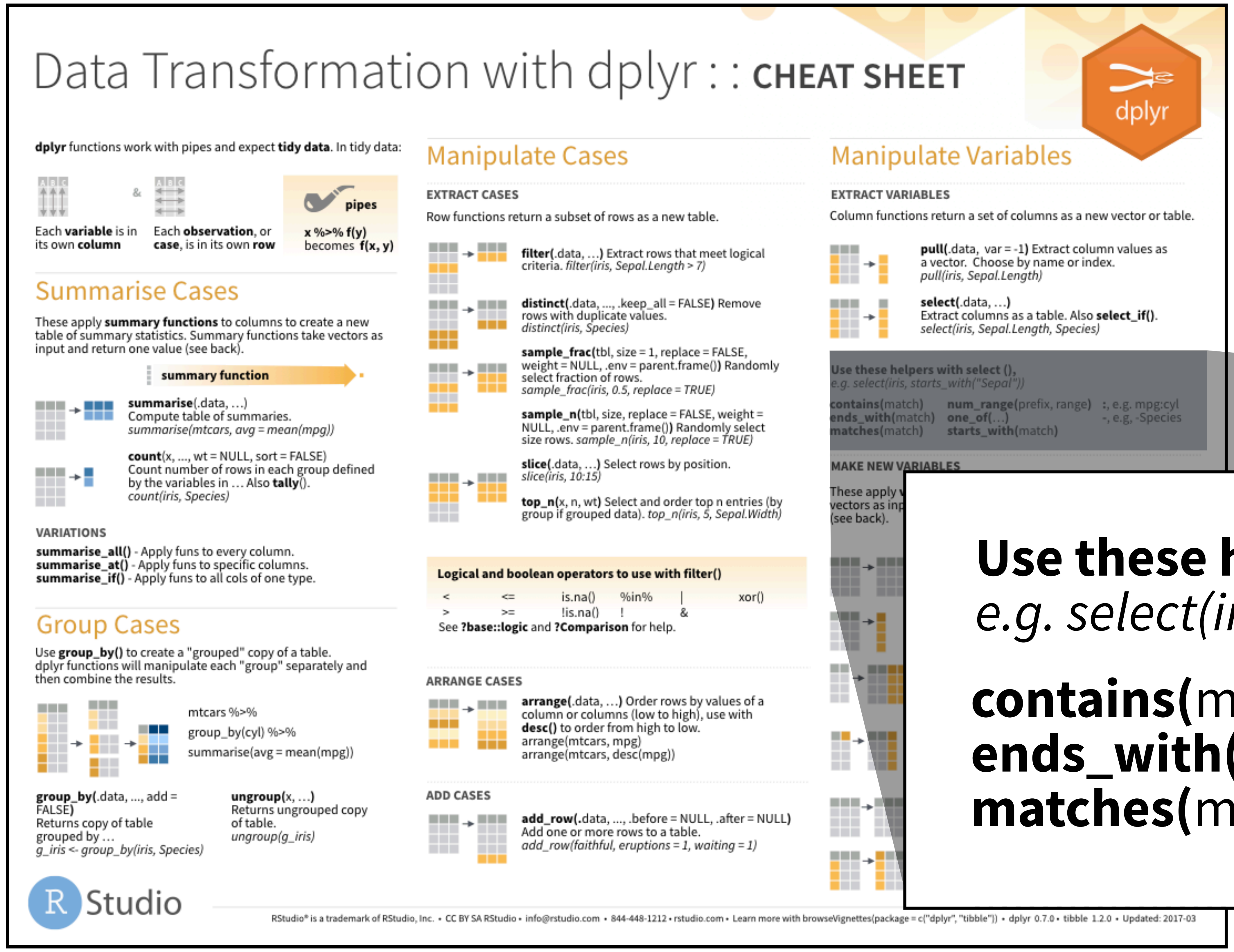
starts_with() - Select columns that start with…

```
select(mpg, starts_with("c"))
```

ends_with() - Select columns that end with…

```
select(mpg, ends_with("y"))
```

select() helpers



Data Transformation with dplyr : : CHEAT SHEET

dplyr functions work with pipes and expect **tidy data**. In tidy data:

Each **variable** is in its own **column**

Each **observation**, or **case**, is in its own **row**

x %>% f(y) becomes **f(x, y)**

Summarise Cases

These apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summarise(.data, …)
Compute table of summaries.
summarise(mtcars, avg = mean(mpg))

count(x, ..., wt = NULL, sort = FALSE)
Count number of rows in each group defined by the variables in … Also **tally**().
count(iris, Species)

VARIATIONS

summarise_all() - Apply funs to every column.
summarise_at() - Apply funs to specific columns.
summarise_if() - Apply funs to all cols of one type.

Group Cases

Use **group_by()** to create a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.

```
mtcars %>%
group_by(cyl) %>%
summarise(avg = mean(mpg))
```

group_by(.data, ..., add = FALSE**)**
Returns copy of table grouped by …
g_iris <- group_by(iris, Species)

ungroup(x, …**)**
Returns ungrouped copy of table.
ungroup(g_iris)

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table.

filter(.data, …**)** Extract rows that meet logical criteria. *filter(iris, Sepal.Length > 7)*

distinct(.data, ..., .keep_all = FALSE**)** Remove rows with duplicate values.
distinct(iris, Species)

sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame()**)** Randomly select fraction of rows.
sample_frac(iris, 0.5, replace = TRUE)

sample_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame()**)** Randomly select size rows. *sample_n(iris, 10, replace = TRUE)*

slice(.data, …**)** Select rows by position.
slice(iris, 10:15)

top_n(x, n, wt**)** Select and order top n entries (by group if grouped data). *top_n(iris, 5, Sepal.Width)*

Logical and boolean operators to use with filter()

```
<     <=     is.na()     %in%     |     xor()
>     >=     !is.na()    !        &
```

See **?base::logic** and **?Comparison** for help.

ARRANGE CASES

arrange(.data, …**)** Order rows by values of a column or columns (low to high), use with **desc()** to order from high to low.
arrange(mtcars, mpg)
arrange(mtcars, desc(mpg))

ADD CASES

add_row(.data, ..., .before = NULL, .after = NULL**)**
Add one or more rows to a table.
add_row(faithful, eruptions = 1, waiting = 1)

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

pull(.data, var = -1**)** Extract column values as a vector. Choose by name or index.
pull(iris, Sepal.Length)

select(.data, …**)**
Extract columns as a table. Also **select_if()**.
select(iris, Sepal.Length, Species)

Use these helpers with select (),
e.g. select(iris, starts_with("Sepal"))

contains(match**)**
ends_with(match**)**
matches(match**)**

num_range(prefix, range**)**
one_of(…**)**
starts_with(match**)**

:, e.g. mpg:cyl
-, e.g, -Species

MAKE NEW VARIABLES

These apply v
vectors as inp
(see back).

R Studio

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more with browseVignettes(package = c("dplyr", "tibble")) • dplyr 0.7.0 • tibble 1.2.0 • Updated: 2017-03

Use these helpers with select (),
e.g. select(iris, starts_with("Sepal"))

contains(match**)**
ends_with(match**)**
matches(match**)**

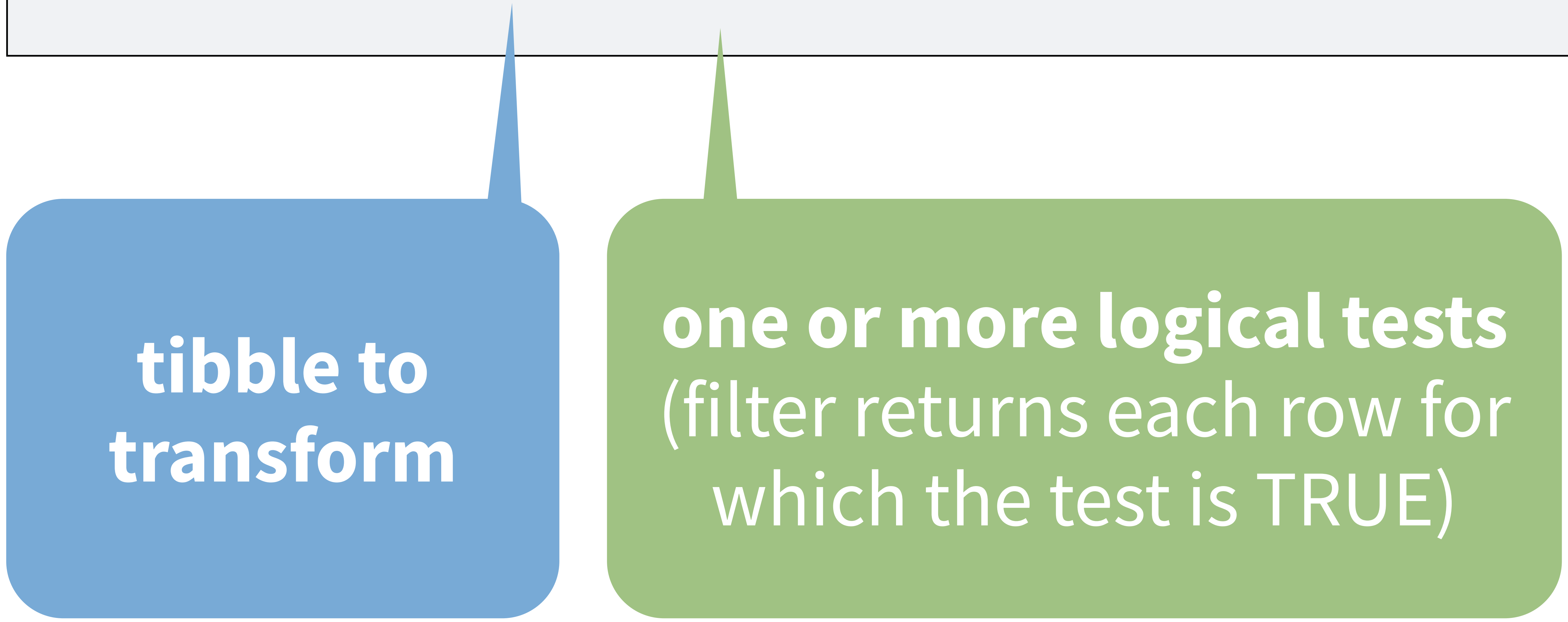
num_range(prefix, range**)**
one_of(…**)**
starts_with(match**)**

:, e.g. mpg:cyl
-, e.g, -Species

filter()

Extract rows that meet logical criteria.

```
filter(.data, … )
```

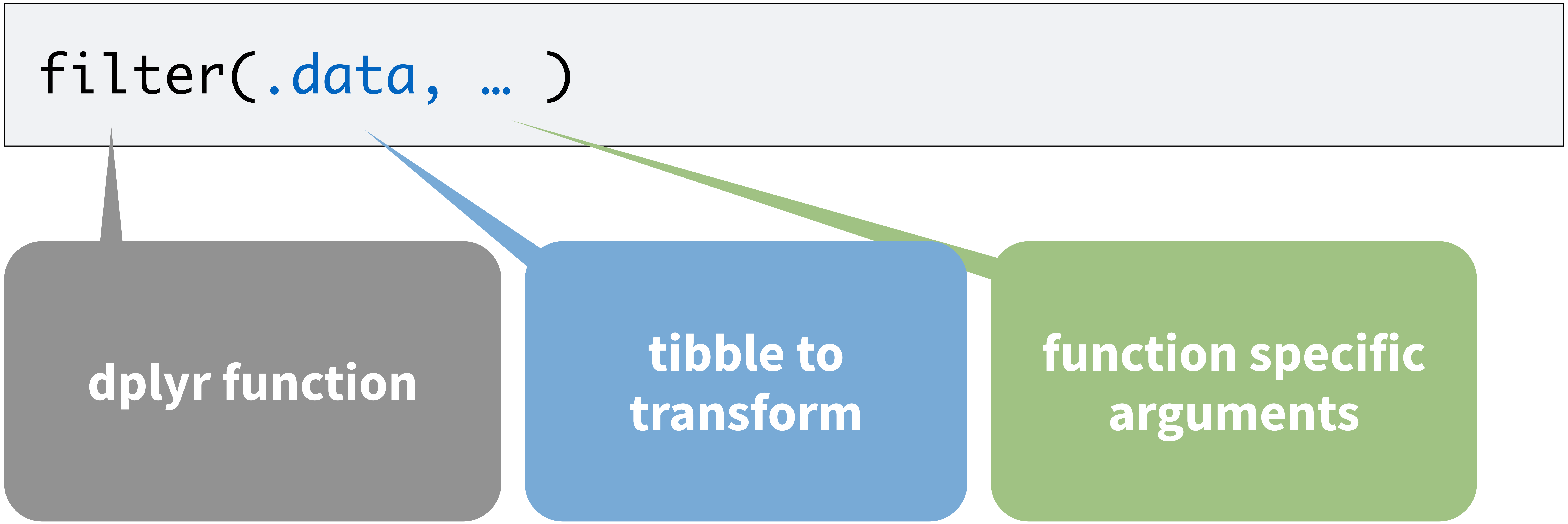


tibble to transform

one or more logical tests (filter returns each row for which the test is TRUE)

common syntax

Each function takes a data frame / tibble as its first argument and returns a data frame / tibble.



```
filter(.data, … )
```

dplyr function

tibble to transform

function specific arguments

filter()

Extract rows that meet logical criteria.

```
filter(babynames, name == "Garrett")
```

babynames

year	sex	name	n	prop
1880	M	John	9655	0.0815
1880	M	William	9532	0.0805
1880	M	James	5927	0.0501
1880	M	Charles	5348	0.0451
1880	M	Garrett	13	0.0001
1881	M	John	8769	0.081

→

year	sex	name	n	prop
1880	M	Garrett	13	0.0001
1881	M	Garrett	7	0.0001
…	…	Garrett	…	…

filter()

Extract rows that meet logical criteria.

```
filter(babynames, name == "Garrett")
```

babynames

year	sex	name	n	prop
1880	M	John	9655	0.0815
1880	M	William	9532	0.0805
1880	M	James	5927	0.0501
1880	M	Charles	5348	0.0451
1880	M	Garrett	13	0.0001
1881	M	John	8769	0.081

= sets
(returns nothing)

== tests if equal
(returns TRUE or FALSE)

Logical tests

?Comparison

`x < y`	Less than
`x > y`	Greater than
`x == y`	Equal to
`x <= y`	Less than or equal to
`x >= y`	Greater than or equal to
`x != y`	Not equal to
`x %in% y`	Group membership
`is.na(x)`	Is NA
`!is.na(x)`	Is not NA

```r
x <- 1
x >= 2
# FALSE
```

```r
x <- c(1, 2, 3)
x >= 2
# FALSE TRUE TRUE
```

```
filter(babynames, prop >= 0.08)
#    year   sex    name     n       prop
# 1  1880     M    John  9655 0.08154630
# 2  1880     M William  9531 0.08049899
# 3  1881     M    John  8769 0.08098299
```

```
filter(babynames, name == "Sea")
#    year   sex  name     n         prop
# 1  1982     F   Sea     5 2.756771e-06
# 2  1985     M   Sea     6 3.119547e-06
# 3  1986     M   Sea     5 2.603512e-06
# 4  1998     F   Sea     5 2.580377e-06
```

Two common mistakes

1. Using = instead of ==

```
filter(babynames, name = "Sea")
filter(babynames, name == "Sea")
```

2. Forgetting quotes

```
filter(babynames, name == Sea)
filter(babynames, name == "Sea")
```

filter()

Extract rows that meet *every* logical criteria.

```
filter(babynames, name == "Garrett", year == 1880)
```

babynames

year	sex	name	n	prop
1880	M	John	9655	0.0815
1880	M	William	9532	0.0805
1880	M	James	5927	0.0501
1880	M	Charles	5348	0.0451
1880	M	Garrett	13	0.0001
1881	M	John	8769	0.081

→

year	sex	name	n	prop
1880	M	Garrett	13	0.0001

filter()

Extract rows that meet every logical criteria.

```
filter(babynames, name == "Garrett" & year == 1880)
```

babynames

year	sex	name	n	prop
1880	M	John	9655	0.0815
1880	M	William	9532	0.0805
1880	M	James	5927	0.0501
1880	M	Charles	5348	0.0451
1880	M	Garrett	13	0.0001
1881	M	John	8769	0.081

→

year	sex	name	n	prop
1880	M	Garrett	13	0.0001

Boolean operators

?base::Logic

a & b	and
a | b	or
xor(a,b)	exactly or
!a	not
( )	To group tests . & evaluates before |

Two more common mistakes

3. Collapsing multiple tests into one

```
filter(babynames, 10 < n < 20)
filter(babynames, 10 < n, n < 20)
```

4. Stringing together many tests (when you could use %in%)

```
filter(babynames, n == 5 | n == 6 | n == 7 | n == 8)
filter(babynames, n %in% c(5, 6, 7, 8))
```

```
babynames %>%
  filter(name == "Garrett", sex == "M") %>%
  select(year, prop)
```

year	prop
1880	0.0001
1881	0.0001
1882	0.0001
1883	0.0001
1884	0.0001
…	…

Deriving information

summarise() - summarise **variables**

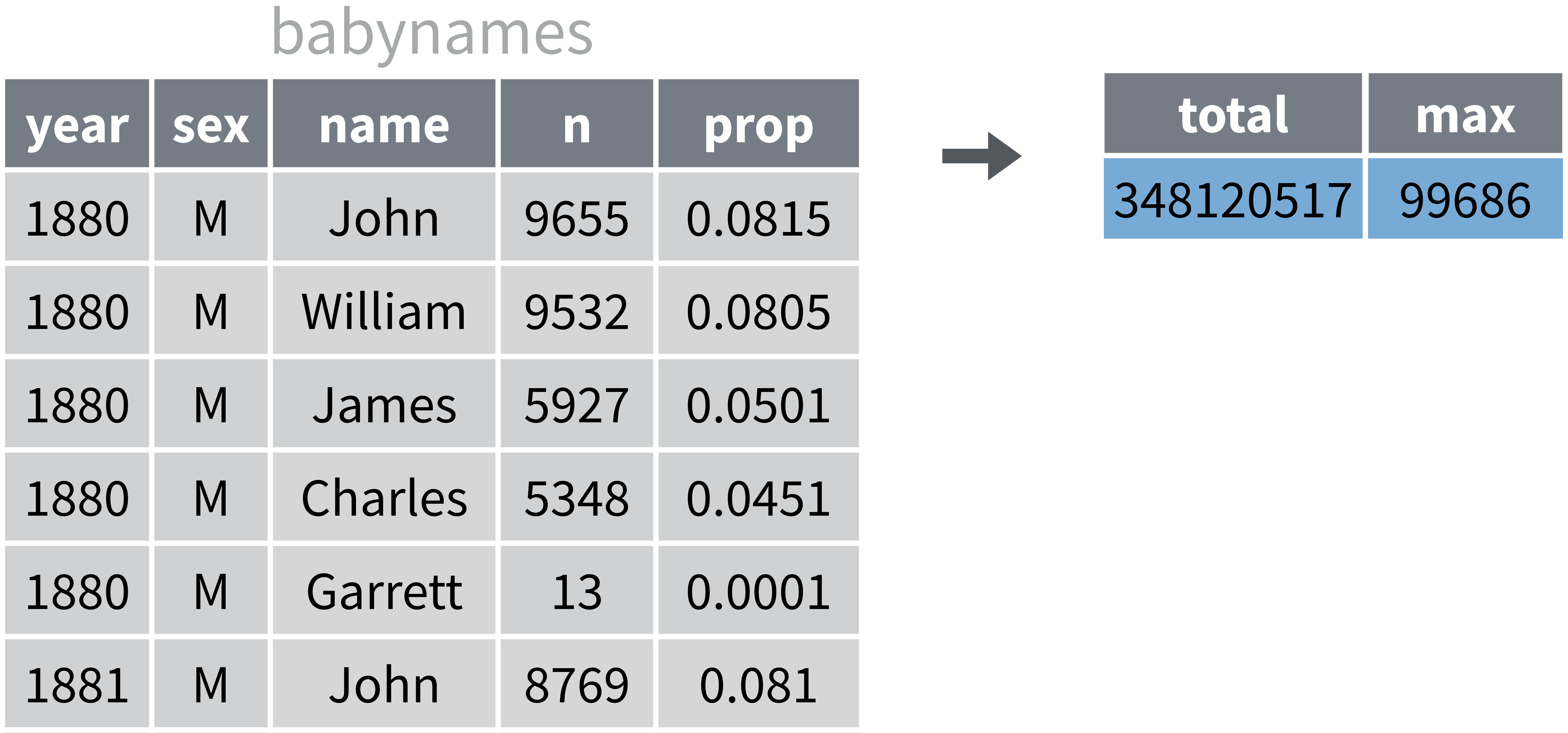
group_by() - group **cases**

mutate() - create new **variables**

summarise()

Compute table of summaries.

```
babynames %>%
     summarise(total = sum(n),
               max = max(n))
```



babynames

year	sex	name	n	prop
1880	M	John	9655	0.0815
1880	M	William	9532	0.0805
1880	M	James	5927	0.0501
1880	M	Charles	5348	0.0451
1880	M	Garrett	13	0.0001
1881	M	John	8769	0.081

→

total	max
348120517	99686

group_by()

Groups cases by common values.

```
babynames %>%
  group_by(sex) %>%
  summarise(total = sum(n))
```

sex	total
F	172371079
M	175749438

ungroup()

Removes grouping criteria from a data frame.

```
babynames %>%
  group_by(sex) %>%
  ungroup() %>%
  summarise(total = sum(n))
```

total
348120517

mutate()

Create new columns.

```
babynames %>%
  mutate(percent = round(prop*100, 2))
```

babynames

year	sex	name	n	prop
1880	M	John	9655	0.0815
1880	M	William	9532	0.0805
1880	M	James	5927	0.0501
1880	M	Charles	5348	0.0451
1880	M	Garrett	13	0.0001
1881	M	John	8769	0.081

→

year	sex	name	n	prop	percent
1880	M	John	9655	0.0815	8.15
1880	M	William	9532	0.0805	8.05
1880	M	James	5927	0.0501	5.01
1880	M	Charles	5348	0.0451	4.51
1880	M	Garrett	13	0.0001	0.01
1881	M	John	8769	0.081	8.1

mutate()

Create new columns.

```
babynames %>%
  mutate(percent = round(prop*100, 2), nper = round(percent))
```

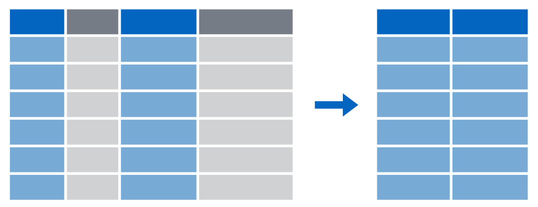
babynames

year	sex	name	n	prop
1880	M	John	9655	0.0815
1880	M	William	9532	0.0805
1880	M	James	5927	0.0501
1880	M	Charles	5348	0.0451
1880	M	Garrett	13	0.0001
1881	M	John	8769	0.081

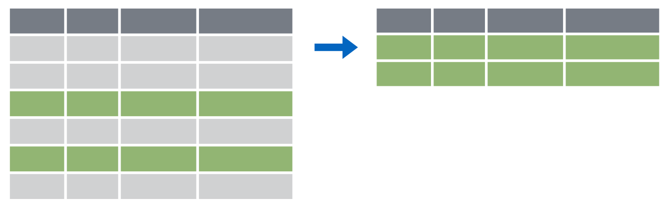
→

year	sex	name	n	prop	percent	nper
1880	M	John	9655	0.0815	8.15	8
1880	M	William	9532	0.0805	8.05	8
1880	M	James	5927	0.0501	5.01	5
1880	M	Charles	5348	0.0451	4.51	5
1880	M	Garrett	13	0.0001	0.01	0
1881	M	John	8769	0.081	8.1	8

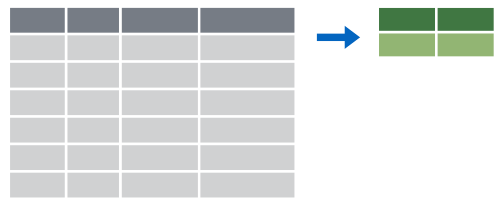
Recap: Single table verbs



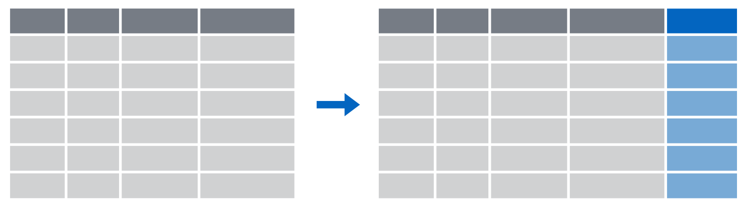
Extract variables with **select()**



Extract cases with **filter()**



Make tables of summaries with **summarise()**.



Make new variables, with **mutate()**.

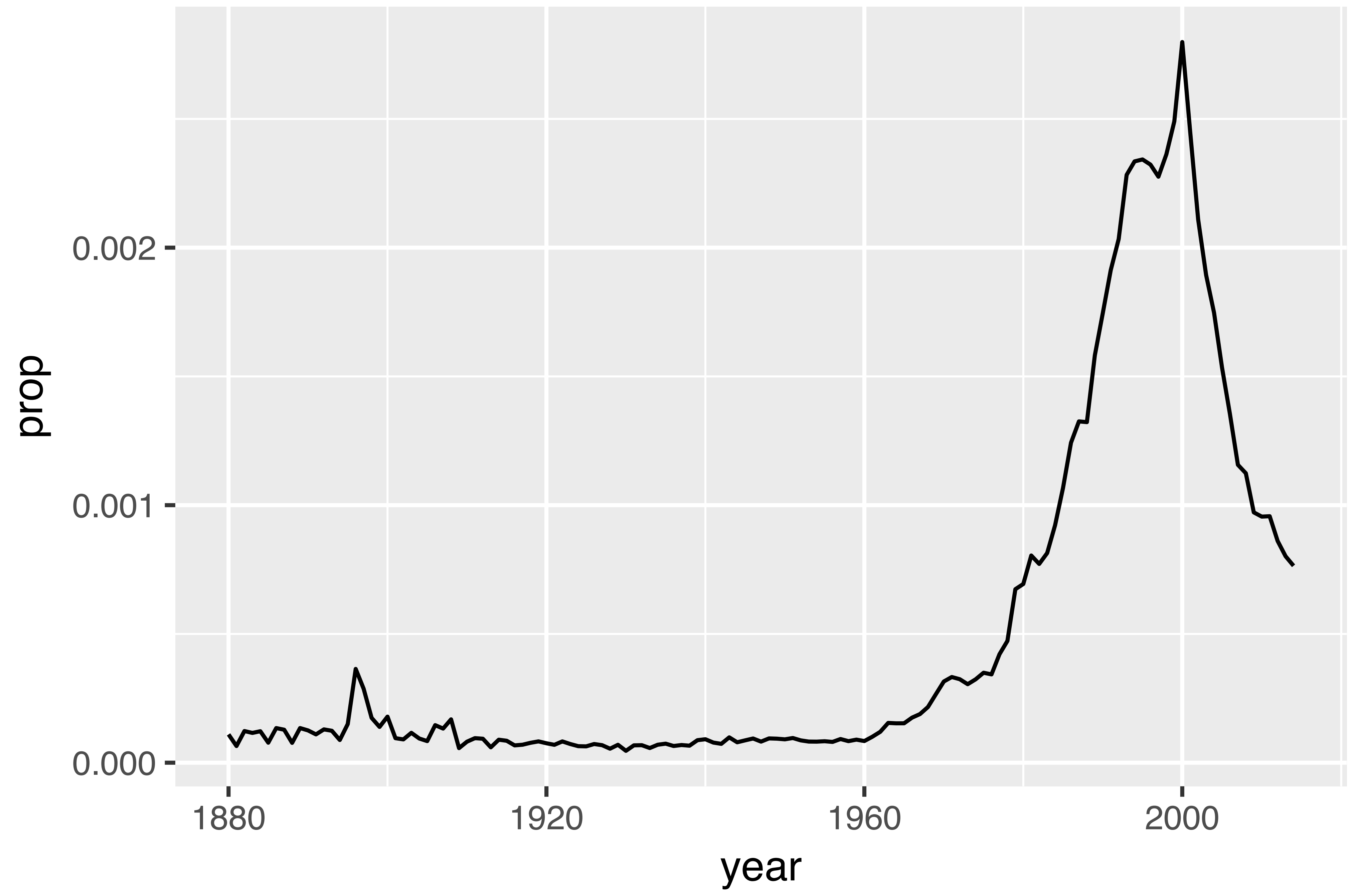
ggplot2: A grammar of graphics

ggplot()

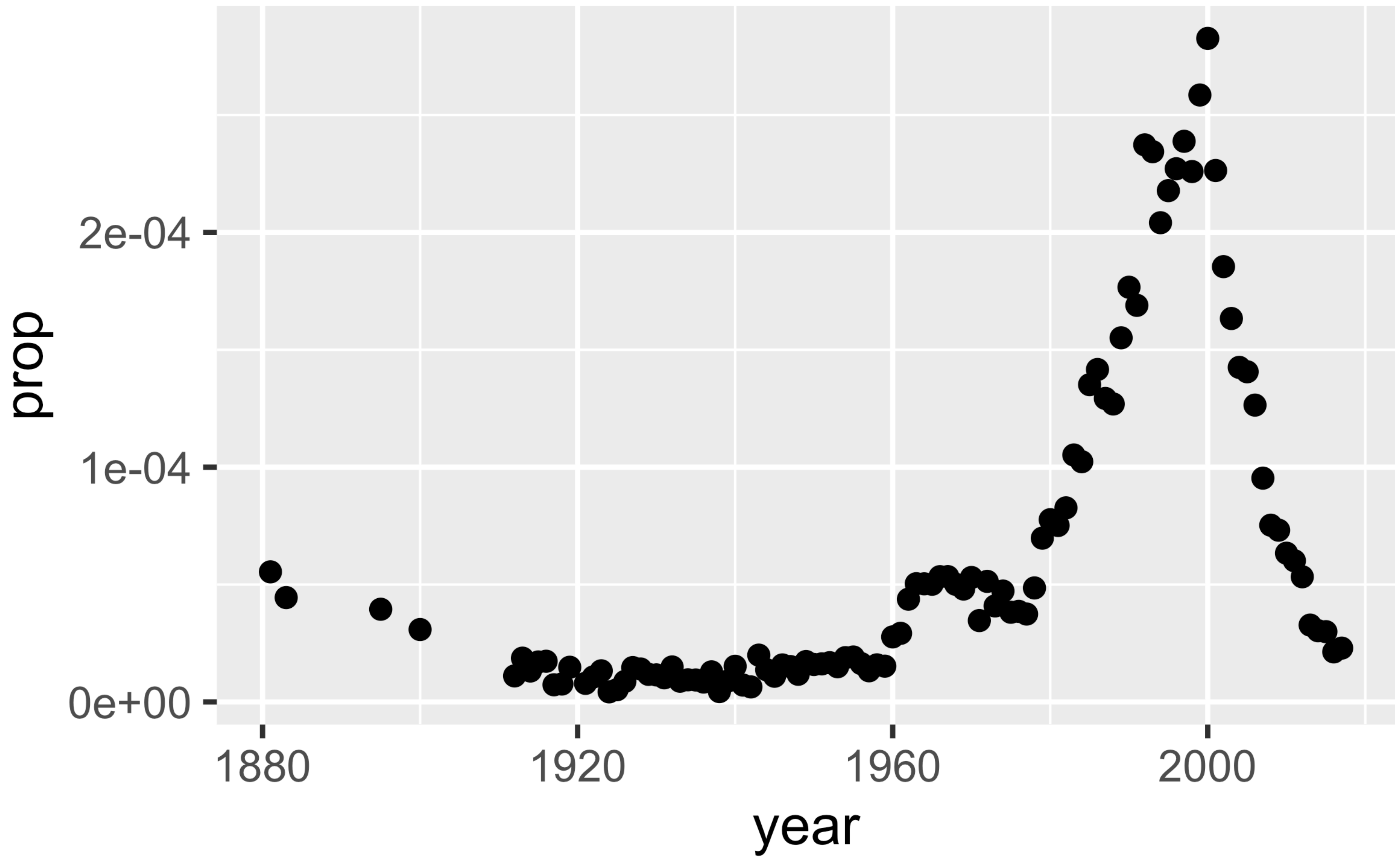
Key idea: You can compose a plot the same way you compose a sentence by following a grammar.

```
ggplot(data = <DATA>, mapping = aes(<MAPPINGS>) +
   <GEOM_FUNCTION>() +
   <GEOM_FUNCTION>() +
   ...
```

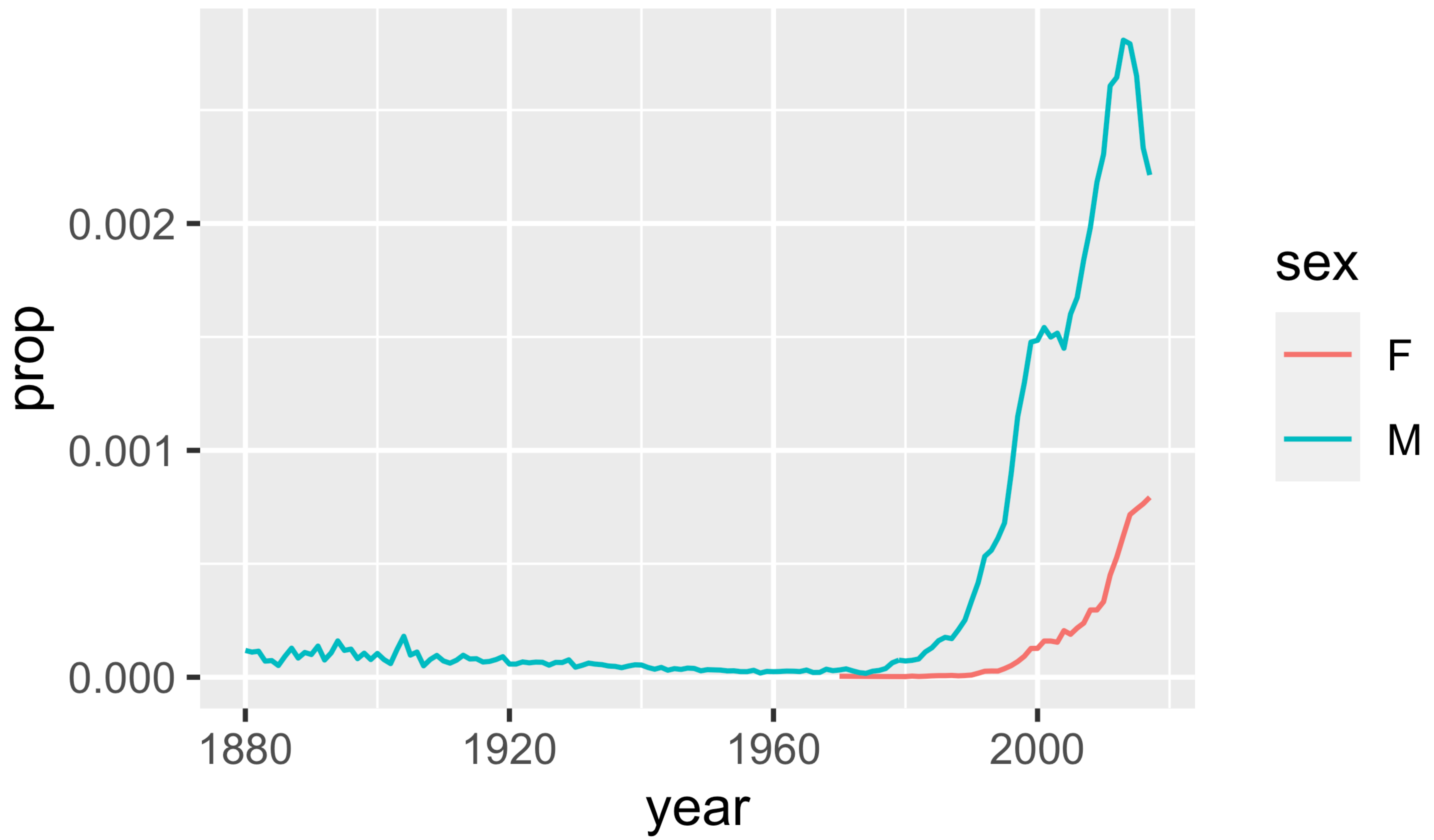
```
babynames %>%
  filter(name == "Garrett", sex == "M") %>%
  ggplot(aes(x = year, y = prop) +
    geom_line()
```



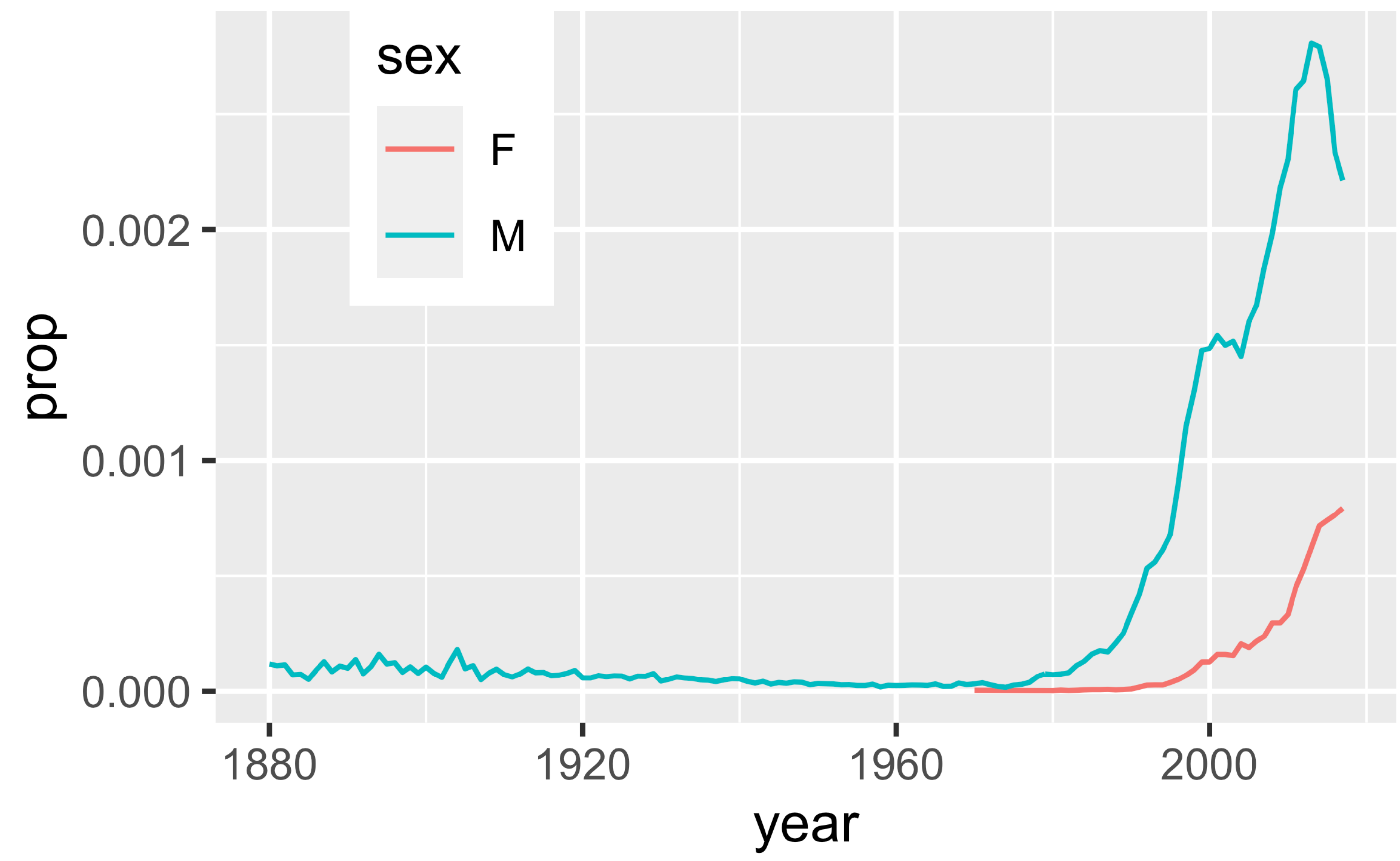
```
babynames %>%
  filter(name == "Garrett", sex == "M") %>%
  ggplot(aes(x = year, y = prop) +
    geom_point()
```



```
babynames %>%
  filter(name == "Parker") %>%
  ggplot(aes(x = year, y = prop, color = sex) +
    geom_line()
```



```
babynames %>%
  filter(name == "Parker") %>%
  ggplot(aes(x = year, y = prop, color = sex) +
    geom_line() +
    theme(legend.position = c(.2, .8)
```



Using RStudio

Git and github

"Always remember your first collaborator is your future self, and your past self doesn't answer emails"

- Christie Bahlai

From “Version control with git for scientists” by Max Joseph

https://github.com/mbjoseph/git-intro

An analogy from rock climbing

https://www.flickr.com/photos/magnezja/8587678264

From Hadley Wickham

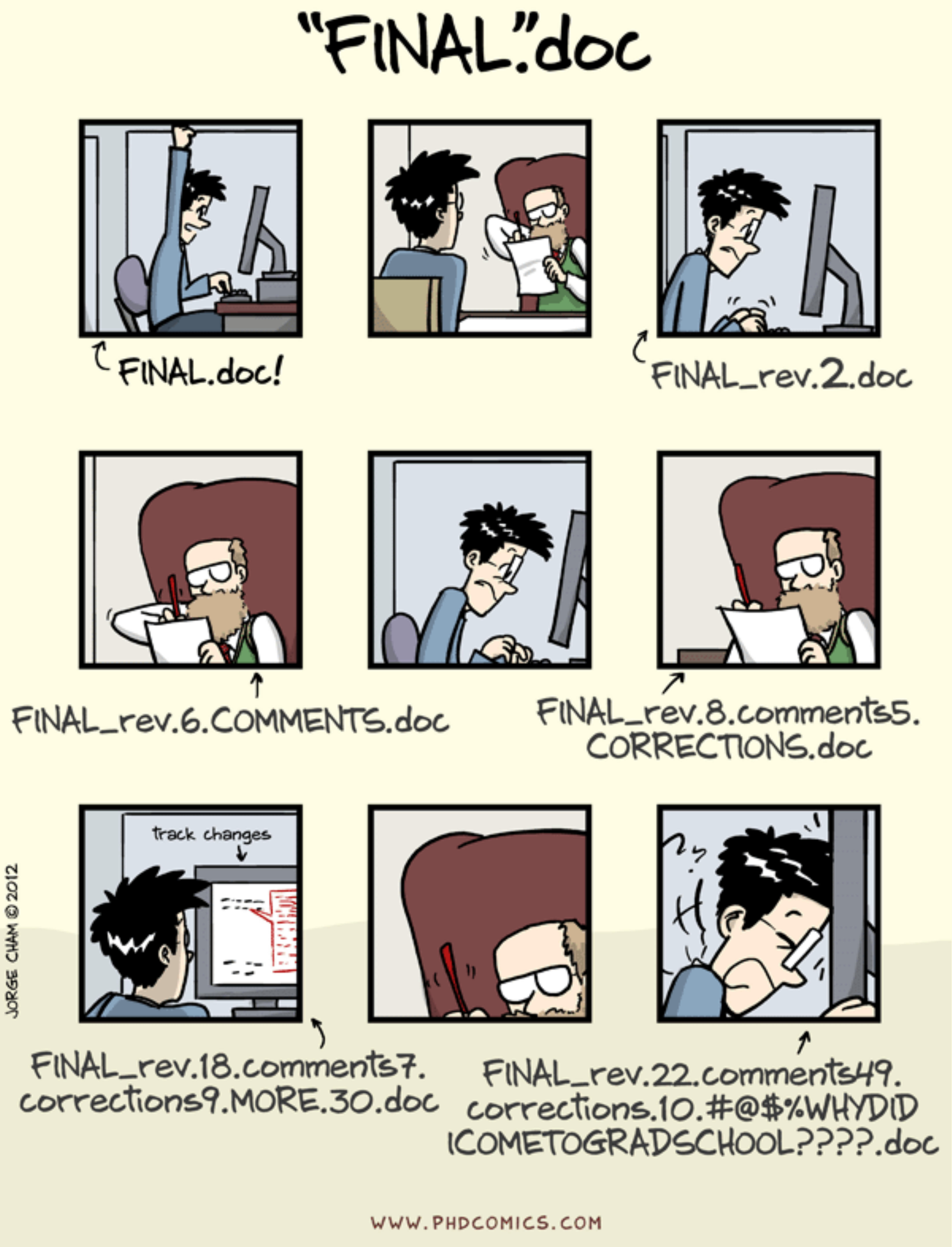
https://github.com/garrettgman/webinars/tree/master/06-Collaboration-and-time-travel-version-control

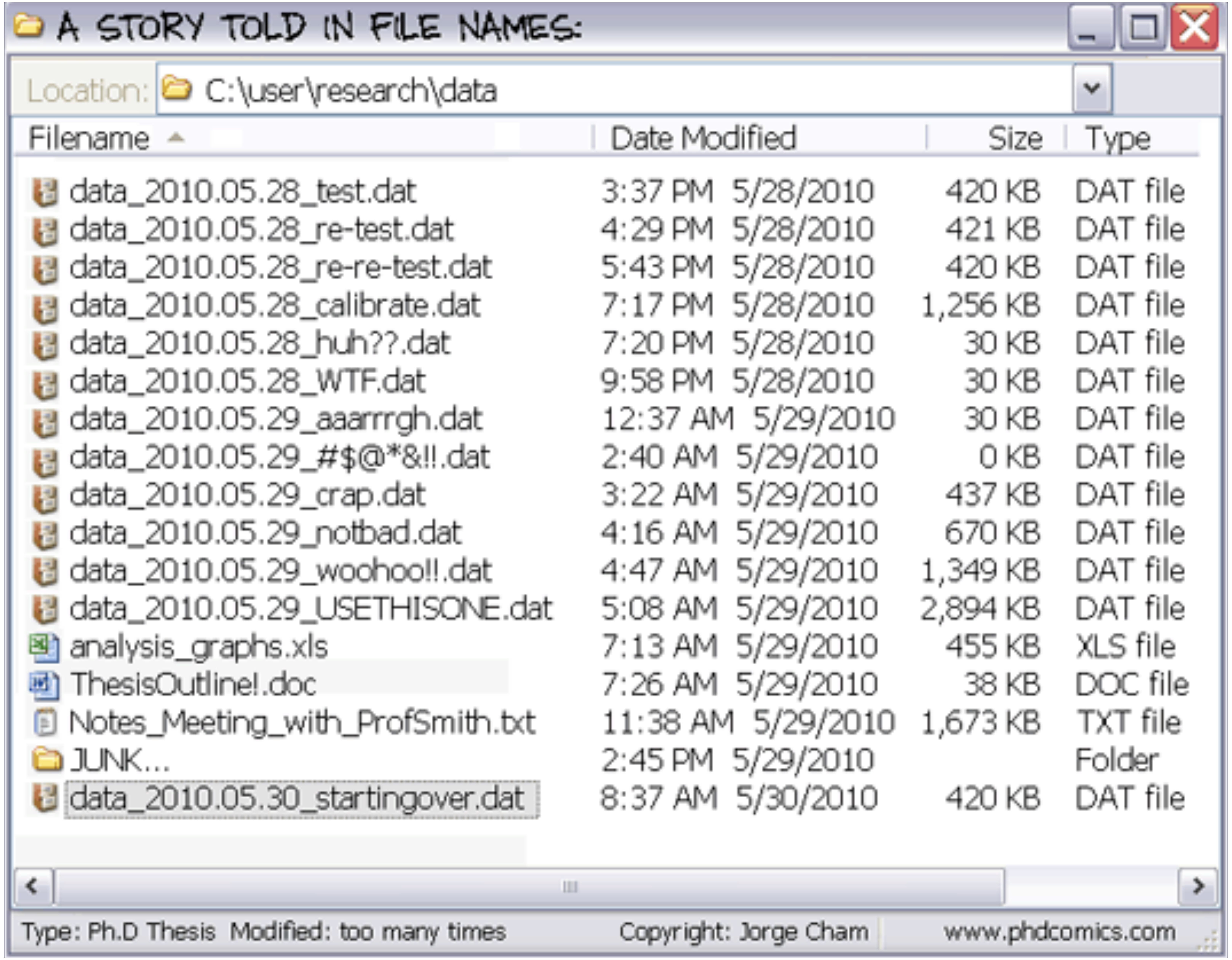
An analogy from rock climbing

https://www.flickr.com/photos/subflux/509039029

From Hadley Wickham

https://github.com/garrettgman/webinars/tree/master/06-Collaboration-and-time-travel-version-control


"FINAL".doc
FINAL.doc!
FINAL_rev.2.doc
FINAL_rev.6.COMMENTS.doc
FINAL_rev.8.comments5.
CORRECTIONS.doc
track changes
FINAL_rev.18.comments7.
corrections9.MORE.30.doc
FINAL_rev.22.comments49.
corrections.10.#@$%WHYDID
ICOMETOGRADSCHOOL????.doc
JORGE CHAM ©2012
WWW.PHDCOMICS.COM



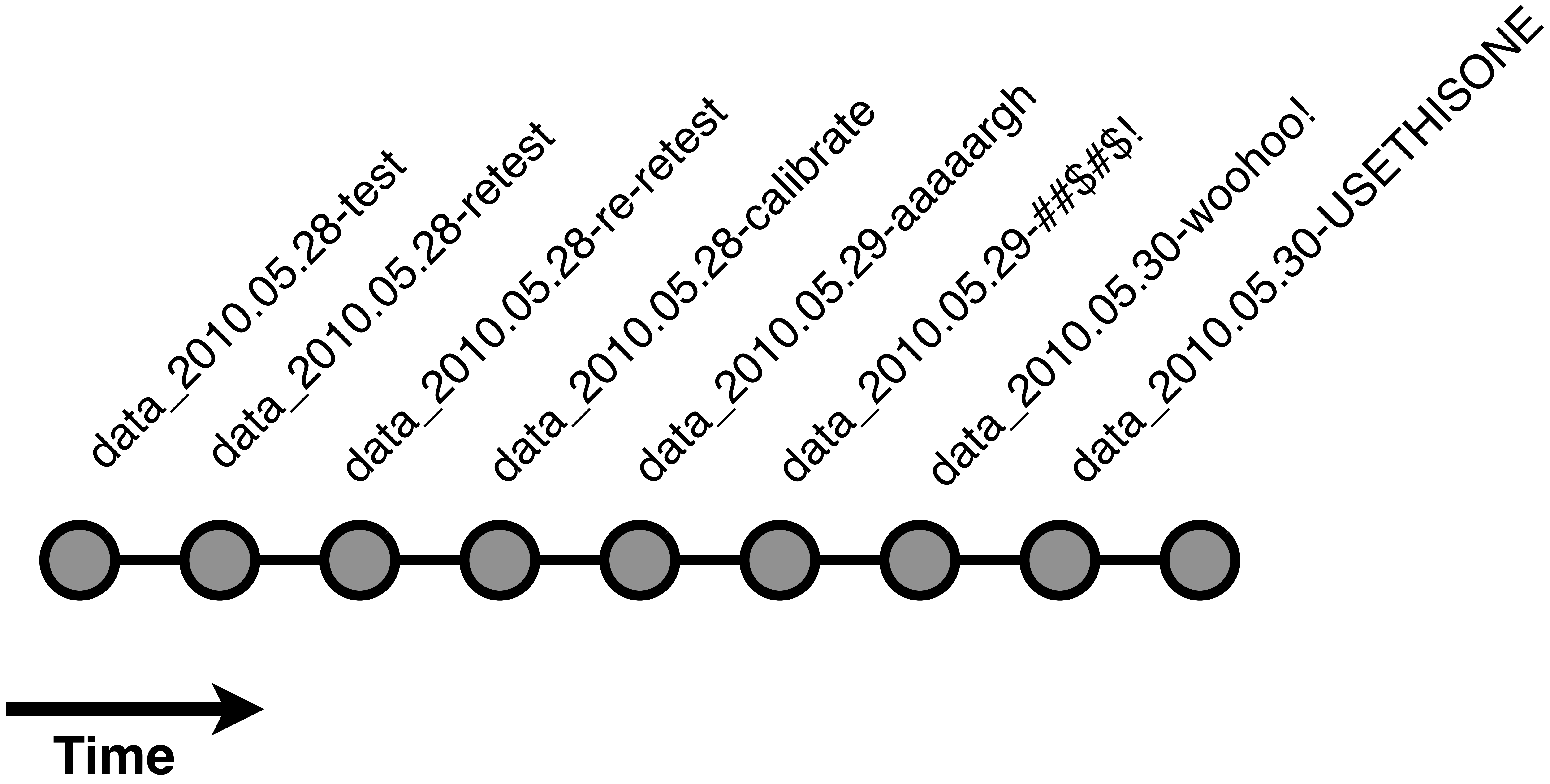
A STORY TOLD IN FILE NAMES:

Location: C:\user\research\data

Filename	Date Modified	Size	Type
data_2010.05.28_test.dat	3:37 PM 5/28/2010	420 KB	DAT file
data_2010.05.28_re-test.dat	4:29 PM 5/28/2010	421 KB	DAT file
data_2010.05.28_re-re-test.dat	5:43 PM 5/28/2010	420 KB	DAT file
data_2010.05.28_calibrate.dat	7:17 PM 5/28/2010	1,256 KB	DAT file
data_2010.05.28_huh??.dat	7:20 PM 5/28/2010	30 KB	DAT file
data_2010.05.28_WTF.dat	9:58 PM 5/28/2010	30 KB	DAT file
data_2010.05.29_aaarrrgh.dat	12:37 AM 5/29/2010	30 KB	DAT file
data_2010.05.29_#$@*&!!.dat	2:40 AM 5/29/2010	0 KB	DAT file
data_2010.05.29_crap.dat	3:22 AM 5/29/2010	437 KB	DAT file
data_2010.05.29_notbad.dat	4:16 AM 5/29/2010	670 KB	DAT file
data_2010.05.29_woohoo!!.dat	4:47 AM 5/29/2010	1,349 KB	DAT file
data_2010.05.29_USETHISONE.dat	5:08 AM 5/29/2010	2,894 KB	DAT file
analysis_graphs.xls	7:13 AM 5/29/2010	455 KB	XLS file
ThesisOutline!.doc	7:26 AM 5/29/2010	38 KB	DOC file
Notes_Meeting_with_ProfSmith.txt	11:38 AM 5/29/2010	1,673 KB	TXT file
JUNK...	2:45 PM 5/29/2010		Folder
data_2010.05.30_startingover.dat	8:37 AM 5/30/2010	420 KB	DAT file

Type: Ph.D Thesis Modified: too many times Copyright: Jorge Cham www.phdcomics.com

http://www.phdcomics.com/comics/archive.php?comicid=1323



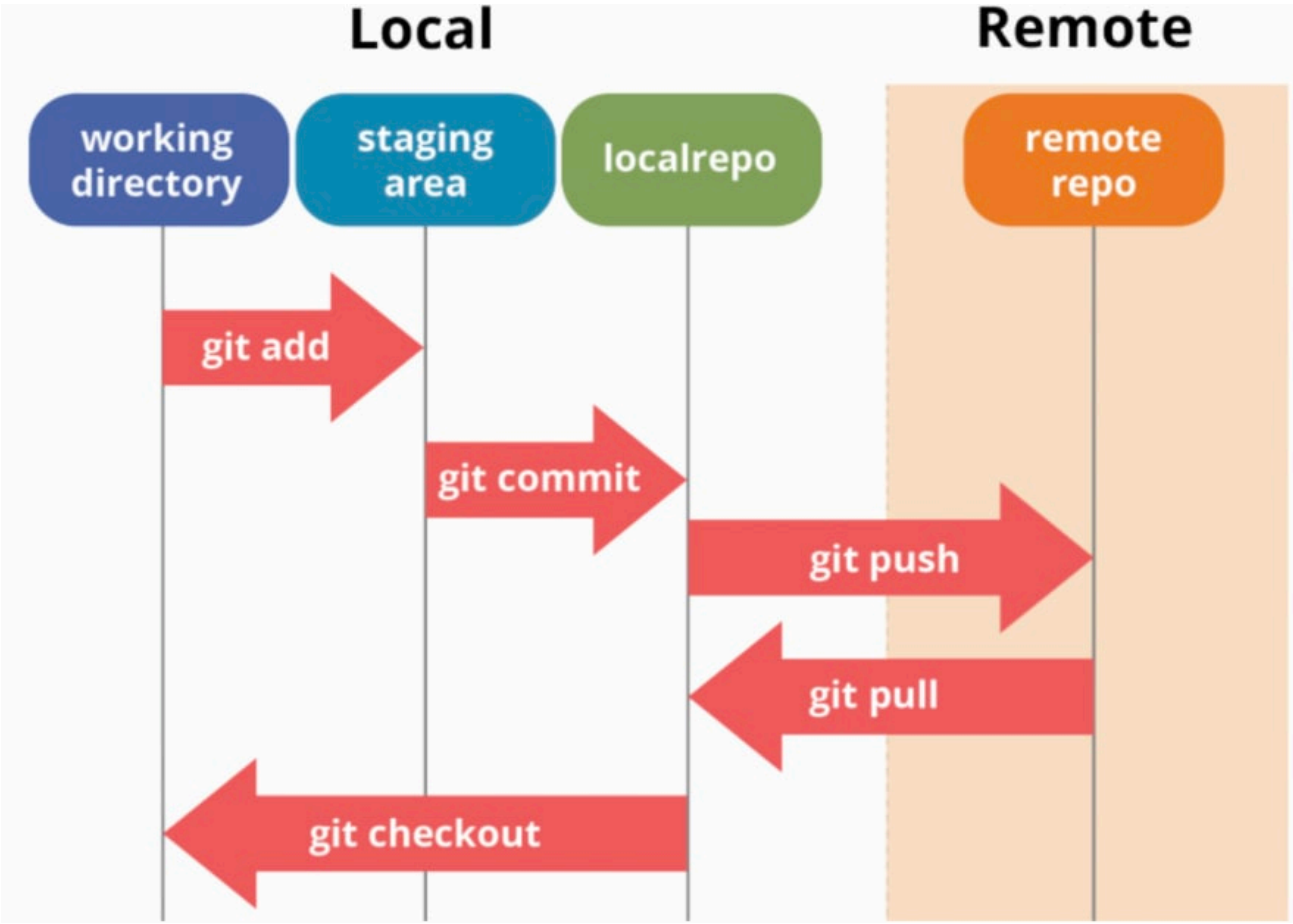
data_2010.05.28-test
data_2010.05.28-retest
data_2010.05.28-re-retest
data_2010.05.28-calibrate
data_2010.05.29-aaaaaargh
data_2010.05.29-##$#$!
data_2010.05.30-woohoo!
data_2010.05.30-USETHISONE
Time

The history of a project can be viewed as a series of changes

Changes

- A unique identifier
- What changed?
- When did it change?
- Who changed it?
- Why did it change?

The git workflow



The git workflow

`status`: see what is different between your current state and the last `commit`

`add`: add this file to the set of files that have been changed since last time

`commit`: flag that a set of additions mark a meaningful unit of progress

`push`: take everything in the last commit and push it to the cloud

`pull`: update your current status to be consistent with changes made **after** the last time you were consistent with the cloud