Data manipulation in R

APAM E4990
Modeling Social Data

Jake Hofman

Columbia University

February 8, 2019

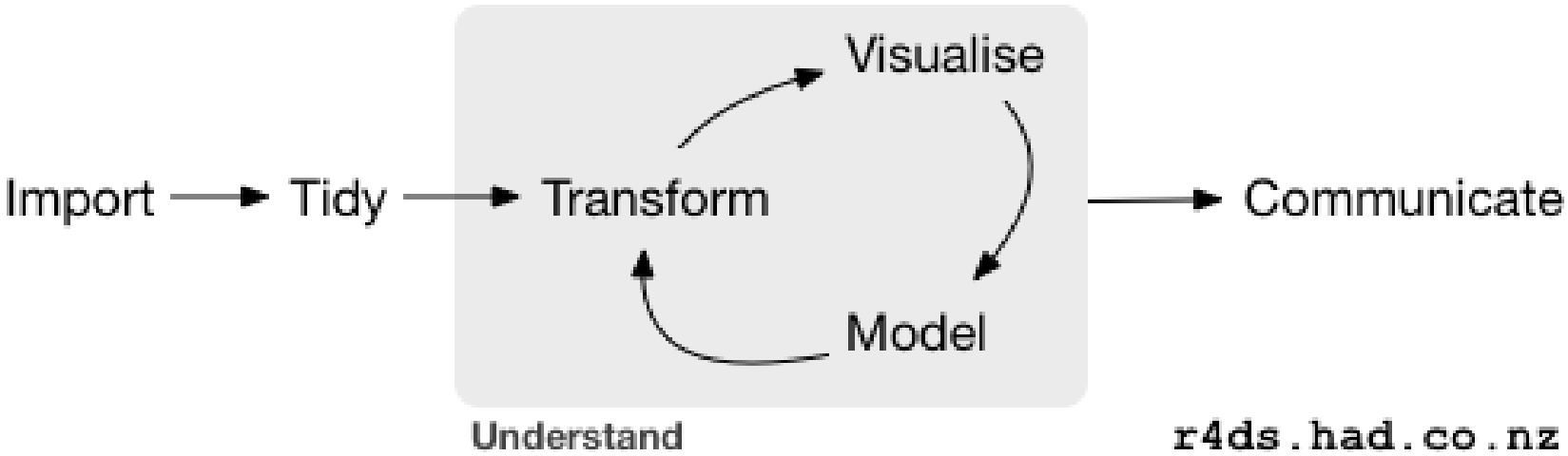
The good, the bad, & the ugly

- R isn't the best programming language out there
- But it happens to be great for data analysis
- The result is a steep learning curve with a high payoff

For instance . . .

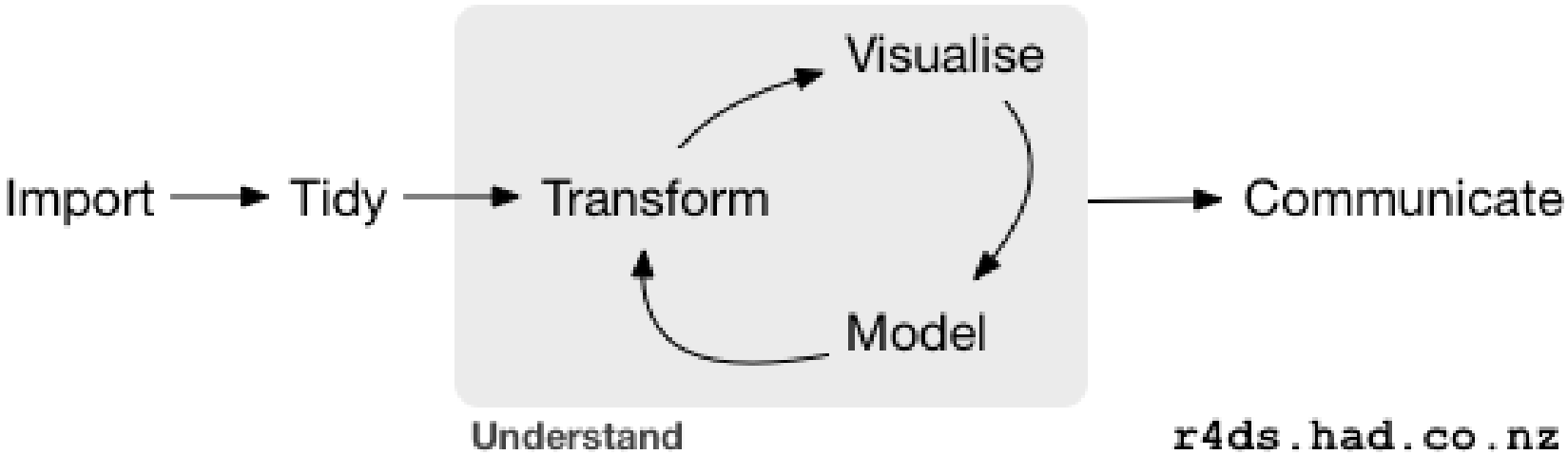
- You'll see a mix of `camelCase`, `this.that`, and `snake_case` conventions
- Dots (`.`) (mostly) don't mean anything special
- Likewise, `$` gets used in funny ways
- R is loosely typed, which can lead to unexpected coercions and silent fails
- It also tries to be clever about variable scope, which can backfire if you're not careful

But it will help you . . .



- Do extremely fast exploratory data analysis
- Easily generate high-quality data visualizations
- Fit and evaluate pretty much any statistical model you can think of

But it will help you . . .



- Do extremely fast exploratory data analysis
- Easily generate high-quality data visualizations
- Fit and evaluate pretty much any statistical model you can think of

This will change the way you do data analysis, because you'll ask questions you wouldn't have bothered to otherwise

Basic types

- `int, double`: for numbers
- `character`: for strings
- `factor`: for categorical variables ($\sim$ `struct` or `ENUM`)

Basic types

- `int, double`: for numbers
- `character`: for strings
- `factor`: for categorical variables ($\sim$ `struct` or `ENUM`)

Factors are handy, but take some getting used to

Containers

- `vector`: for multiple values of the same type ($\sim$ array)
- `list`: for multiple values of different types ($\sim$ dictionary)
- `data.frame`: for tables of rectangular data of mixed types ($\sim$ matrix)

Containers

- `vector`: for multiple values of the same type ($\sim$ array)
- `list`: for multiple values of different types ($\sim$ dictionary)
- `data.frame`: for tables of rectangular data of mixed types ($\sim$ matrix)

We'll mostly work with data frames, which themselves are lists of vectors

The tidyverse

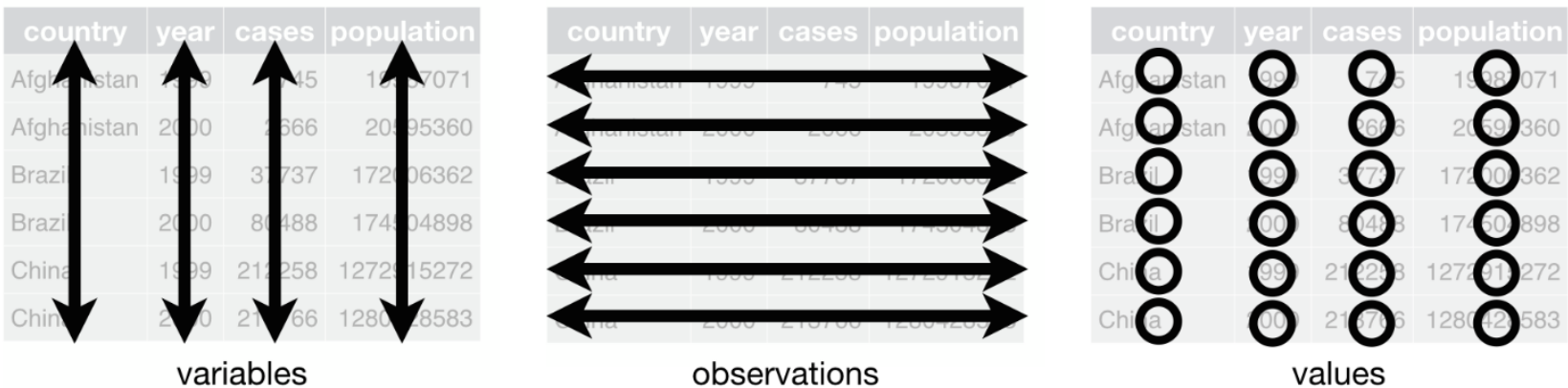
The tidyverse is a collection of packages that work together to make data analysis easier:

- dplyr for split / apply / combine type counting
- ggplot2 for making plots
- tidyr for reshaping and "tidying" data
- readr for reading and writing files
- ...

Tidy data

The core philosophy is that your data should be in a "tidy" table with:

- One variable per column
- One observation per row
- One measured value per cell



Tidy data

	tripduration	starttime	stoptime	start_station_id	start_station_name	start_station_latitude	start_station_longitude
1	382	2014-02-01 00:00:00	2014-02-01 00:06:22	294	Washington Square E	40.73049	-73.99572
2	372	2014-02-01 00:00:03	2014-02-01 00:06:15	285	Broadway & E 14 St	40.73455	-73.99074
3	591	2014-02-01 00:00:09	2014-02-01 00:10:00	247	Perry St & Bleecker St	40.73535	-74.00483
4	583	2014-02-01 00:00:32	2014-02-01 00:10:15	357	E 11 St & Broadway	40.73262	-73.99158
5	223	2014-02-01 00:00:41	2014-02-01 00:04:24	401	Allen St & Rivington St	40.72020	-73.98998
6	541	2014-02-01 00:00:46	2014-02-01 00:09:47	152	Warren St & Church St	40.71474	-74.00911

- Most of the work goes into getting your data into shape
- After which descriptives statistics, modeling, and visualization are easy

`dplyr`: a grammar of data manipulation

`dplyr` implements the split / apply / combine framework discussed in the last lecture

- Its "grammar" has five main verbs used in the "apply" phase:
  - `filter`: restrict rows based on a condition ($N \to N'$)
  - `arrange`: reorder rows by a variable ($N \to N'$)
  - `select`: pick out specific columns ($K \to K'$)
  - `mutate`: create new or change existing columns ($K \to K'$)
  - `summarize`: collapse a column into one value ($N \to 1$)
- The `group_by` function creates indices to take care of the split and combine phases

`dplyr`: a grammar of data manipulation

`dplyr` implements the split / apply / combine framework discussed in the last lecture

- Its "grammar" has five main verbs used in the "apply" phase:
  - `filter`: restrict rows based on a condition ($N \to N'$)
  - `arrange`: reorder rows by a variable ($N \to N'$)
  - `select`: pick out specific columns ($K \to K'$)
  - `mutate`: create new or change existing columns ($K \to K'$)
  - `summarize`: collapse a column into one value ($N \to 1$)
- The `group_by` function creates indices to take care of the split and combine phases

The cost is that you have to think "functionally", in terms of "vectorized" operations

filter

```
filter(trips, start_station_name == "Broadway & E 14 St")
```

	tripduration	starttime	stoptime	start_station_id	start_station_name	start_station_latitude	start_station_longitude
1	372	2014-02-01 00:00:03	2014-02-01 00:06:15	285	Broadway & E 14 St	40.73455	-73.99074
2	439	2014-02-01 00:02:14	2014-02-01 00:09:33	285	Broadway & E 14 St	40.73455	-73.99074
3	636	2014-02-01 00:08:25	2014-02-01 00:19:01	285	Broadway & E 14 St	40.73455	-73.99074
4	914	2014-02-01 00:43:21	2014-02-01 00:58:35	285	Broadway & E 14 St	40.73455	-73.99074
5	906	2014-02-01 00:43:36	2014-02-01 00:58:42	285	Broadway & E 14 St	40.73455	-73.99074
6	468	2014-02-01 00:57:12	2014-02-01 01:05:00	285	Broadway & E 14 St	40.73455	-73.99074

arrange

```
arrange(trips, starttime)
```

	tripduration	starttime	stoptime	start_station_id	start_station_name	start_station_latitude	start_station_longitude
1	382	2014-02-01 00:00:00	2014-02-01 00:06:22	294	Washington Square E	40.73049	-73.99572
2	372	2014-02-01 00:00:03	2014-02-01 00:06:15	285	Broadway & E 14 St	40.73455	-73.99074
3	591	2014-02-01 00:00:09	2014-02-01 00:10:00	247	Perry St & Bleecker St	40.73535	-74.00483
4	583	2014-02-01 00:00:32	2014-02-01 00:10:15	357	E 11 St & Broadway	40.73262	-73.99158
5	223	2014-02-01 00:00:41	2014-02-01 00:04:24	401	Allen St & Rivington St	40.72020	-73.98998
6	541	2014-02-01 00:00:46	2014-02-01 00:09:47	152	Warren St & Church St	40.71474	-74.00911

select

```
select(trips, starttime, stoptime,
               start_station_name, end_station_name)
```

	starttime	stoptime	start_station_name	end_station_name
1	2014-02-01 00:00:00	2014-02-01 00:06:22	Washington Square E	Stanton St & Chrystie St
2	2014-02-01 00:00:03	2014-02-01 00:06:15	Broadway & E 14 St	E 4 St & 2 Ave
3	2014-02-01 00:00:09	2014-02-01 00:10:00	Perry St & Bleecker St	Mott St & Prince St
4	2014-02-01 00:00:32	2014-02-01 00:10:15	E 11 St & Broadway	Greenwich Ave & 8 Ave
5	2014-02-01 00:00:41	2014-02-01 00:04:24	Allen St & Rivington St	E 4 St & 2 Ave
6	2014-02-01 00:00:46	2014-02-01 00:09:47	Warren St & Church St	Pike St & Monroe St

mutate

```
mutate(trips, time_in_min = tripduration / 60)
```

	tripduration	starttime	stoptime	time_in_min
1	382	2014-02-01 00:00:00	2014-02-01 00:06:22	6.366667
2	372	2014-02-01 00:00:03	2014-02-01 00:06:15	6.200000
3	591	2014-02-01 00:00:09	2014-02-01 00:10:00	9.850000
4	583	2014-02-01 00:00:32	2014-02-01 00:10:15	9.716667
5	223	2014-02-01 00:00:41	2014-02-01 00:04:24	3.716667
6	541	2014-02-01 00:00:46	2014-02-01 00:09:47	9.016667

mutate

```
summarize(trips, mean_duration = mean(tripduration) / 60,
                 sd_duration = sd(tripduration) / 60)
```

mean_duration	sd_duration
14.57533	91.43487

group_by

```
trips_by_gender <- group_by(trips, gender)
```

```
Source: local data frame [224,736 x 4]
Groups: gender [3]

   tripduration           starttime             stoptime gender
          <int>              <dttm>               <dttm>  <int>
1           382 2014-02-01 00:00:00 2014-02-01 00:06:22      1
2           372 2014-02-01 00:00:03 2014-02-01 00:06:15      2
3           591 2014-02-01 00:00:09 2014-02-01 00:10:00      2
4           583 2014-02-01 00:00:32 2014-02-01 00:10:15      1
5           223 2014-02-01 00:00:41 2014-02-01 00:04:24      1
6           541 2014-02-01 00:00:46 2014-02-01 00:09:47      1
7           354 2014-02-01 00:01:01 2014-02-01 00:06:55      1
8           916 2014-02-01 00:01:11 2014-02-01 00:16:27      1
9           277 2014-02-01 00:01:33 2014-02-01 00:06:10      1
10          439 2014-02-01 00:02:14 2014-02-01 00:09:33      2
# ... with 224,726 more rows
```

group_by

```
trips_by_gender <- group_by(trips, gender)
```

```
Classes ‘grouped_df’, ‘tbl_df’, ‘tbl’ and 'data.frame': 224736 obs. of  4 variables:
 $ tripduration: int  382 372 591 583 223 541 354 916 277 439 ...
 $ starttime   : POSIXct, format: "2014-02-01 00:00:00" "2014-02-01 00:00:03" ...
 $ stoptime    : POSIXct, format: "2014-02-01 00:06:22" "2014-02-01 00:06:15" ...
 $ gender      : int  1 2 2 1 1 1 1 1 1 2 ...
 - attr(*, "vars")=List of 1
  ..$ : symbol gender
 - attr(*, "drop")= logi TRUE
 - attr(*, "indices")=List of 3
  ..$ : int  31 55 222 266 293 302 306 329 393 413 ...
  ..$ : int  0 3 4 5 6 7 8 10 11 12 ...
  ..$ : int  1 2 9 18 19 22 24 26 34 49 ...
 - attr(*, "group_sizes")= int  6731 176526 41479
 - attr(*, "biggest_group_size")= int 176526
 - attr(*, "labels")='data.frame':      3 obs. of  1 variable:
  ..$ gender: int  0 1 2
  ..- attr(*, "vars")=List of 1
  .. ..$ : symbol gender
  ..- attr(*, "drop")= logi TRUE
```

group_by + summarize

```
summarize(trips_by_gender,
          count = n(),
          mean_duration = mean(tripduration) / 60,
          sd_duration = sd(tripduration) / 60)
```

	gender	count	mean_duration	sd_duration
1	Unknown	6731	29.01385	92.76851
2	Male	176526	13.56721	83.67627
3	Female	41479	16.52268	118.57922

%>%: the pipe operator

```
trips %>%
  group_by(gender) %>%
  summarize(count = n(),
            mean_duration = mean(tripduration) / 60,
            sd_duration = sd(tripduration) / 60)
```

	gender	count	mean_duration	sd_duration
1	Unknown	6731	29.01385	92.76851
2	Male	176526	13.56721	83.67627
3	Female	41479	16.52268	118.57922

gather: wide to long

```
trips %>%
    gather("variable", "value", starttime, stoptime)
```

trip_id	starttime	stoptime
1	2014-02-01 00:00:00	2014-02-01 00:06:22
2	2014-02-01 00:04:02	2014-02-01 00:08:54
3	2014-02-01 00:06:53	2014-02-01 00:18:28
4	2014-02-01 00:09:03	2014-02-01 00:23:41

trip_id	variable	value
1	starttime	2014-02-01 00:00:00
2	starttime	2014-02-01 00:04:02
3	starttime	2014-02-01 00:06:53
4	starttime	2014-02-01 00:09:03
1	stoptime	2014-02-01 00:06:22
2	stoptime	2014-02-01 00:08:54
3	stoptime	2014-02-01 00:18:28
4	stoptime	2014-02-01 00:23:41

gather: wide to long

```
trips %>%
    gather("variable", "value", starttime, stoptime) %>%
    arrange(value)
```

trip_id	starttime	stoptime
1	2014-02-01 00:00:00	2014-02-01 00:06:22
2	2014-02-01 00:04:02	2014-02-01 00:08:54
3	2014-02-01 00:06:53	2014-02-01 00:18:28
4	2014-02-01 00:09:03	2014-02-01 00:23:41

trip_id	variable	value
1	starttime	2014-02-01 00:00:00
2	starttime	2014-02-01 00:04:02
1	stoptime	2014-02-01 00:06:22
3	starttime	2014-02-01 00:06:53
2	stoptime	2014-02-01 00:08:54
4	starttime	2014-02-01 00:09:03
3	stoptime	2014-02-01 00:18:28
4	stoptime	2014-02-01 00:23:41

gather: wide to long

```
trips %>%
    gather("variable", "value", starttime, stoptime) %>%
    arrange(value) %>%
    mutate(delta = ifelse(variable == "starttime", 1, -1))
```

trip_id	starttime	stoptime
1	2014-02-01 00:00:00	2014-02-01 00:06:22
2	2014-02-01 00:04:02	2014-02-01 00:08:54
3	2014-02-01 00:06:53	2014-02-01 00:18:28
4	2014-02-01 00:09:03	2014-02-01 00:23:41

trip_id	variable	value	delta
1	starttime	2014-02-01 00:00:00	1
2	starttime	2014-02-01 00:04:02	1
1	stoptime	2014-02-01 00:06:22	-1
3	starttime	2014-02-01 00:06:53	1
2	stoptime	2014-02-01 00:08:54	-1
4	starttime	2014-02-01 00:09:03	1
3	stoptime	2014-02-01 00:18:28	-1
4	stoptime	2014-02-01 00:23:41	-1

spread: long to wide

```
trips_long %>%
    spread(variable, value)
```

trip_id	variable	value
1	starttime	2014-02-01 00:00:00
2	starttime	2014-02-01 00:04:02
1	stoptime	2014-02-01 00:06:22
3	starttime	2014-02-01 00:06:53
2	stoptime	2014-02-01 00:08:54
4	starttime	2014-02-01 00:09:03
3	stoptime	2014-02-01 00:18:28
4	stoptime	2014-02-01 00:23:41

trip_id	starttime	stoptime
1	2014-02-01 00:00:00	2014-02-01 00:06:22
2	2014-02-01 00:04:02	2014-02-01 00:08:54
3	2014-02-01 00:06:53	2014-02-01 00:18:28
4	2014-02-01 00:09:03	2014-02-01 00:23:41

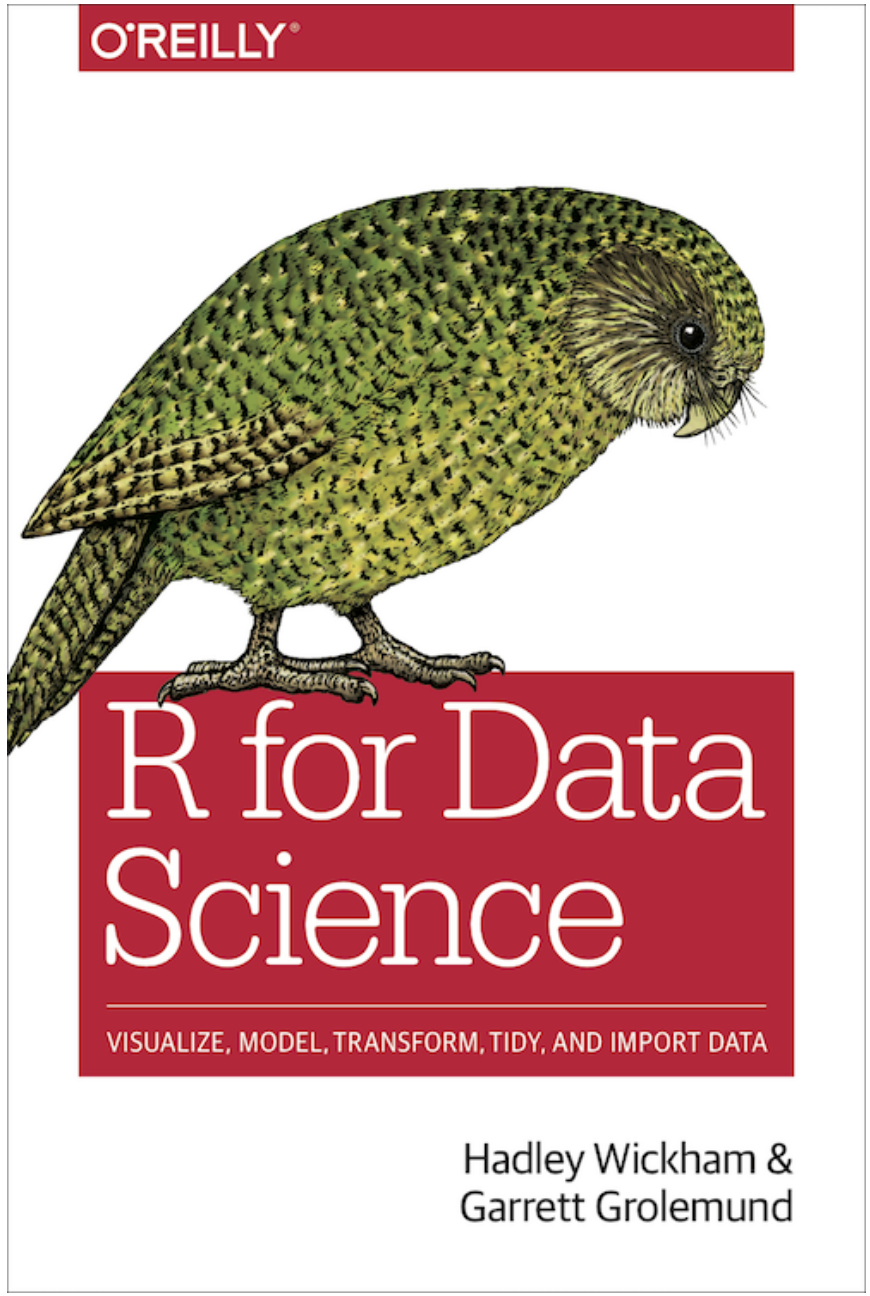
spread: long to wide

```
trips_long %>%
    head %>%
    spread(variable, value)
```

trip_id	variable	value
1	starttime	2014-02-01 00:00:00
2	starttime	2014-02-01 00:04:02
1	stoptime	2014-02-01 00:06:22
3	starttime	2014-02-01 00:06:53
2	stoptime	2014-02-01 00:08:54
4	starttime	2014-02-01 00:09:03

trip_id	starttime	stoptime
1	2014-02-01 00:00:00	2014-02-01 00:06:22
2	2014-02-01 00:04:02	2014-02-01 00:08:54
3	2014-02-01 00:06:53	*NA*
4	2014-02-01 00:09:03	*NA*

r4ds.had.co.nz



style.tidyverse.org

The tidyverse style guide

Hadley Wickham

Welcome

Good coding style is like correct punctuation: you can manage without it, butitsuremakesthingseasiertoread. This site describes the style used throughout the tidyverse. It was originally derived from Google's R style guide, but has evolved and expanded considerably over the years.

All style guides are fundamentally opinionated. Some decisions genuinely do make code easier to use (especially matching indenting to programming structure), but many decisions are arbitrary. The most important thing about a style guide is that it provides consistency, making code easier to write because you need to make fewer decisions.