Tidy data

Data Science in a Box

datasciencebox.org

Modified by Tyler George

Tidy data

> Happy families are all alike; every unhappy family is unhappy in its own way.
>
> Leo Tolstoy

Tidy data

> Happy families are all alike; every unhappy family is unhappy in its own way.
>
> Leo Tolstoy

Characteristics of tidy data:

- Each variable forms a column.
- Each observation forms a row.
- Each type of observational unit forms a table.

Tidy data

> Happy families are all alike; every unhappy family is unhappy in its own way.
>
> Leo Tolstoy

Characteristics of tidy data:

- Each variable forms a column.
- Each observation forms a row.
- Each type of observational unit forms a table.

Characteristics of untidy data:

!@#$%^&*()

What makes this data not tidy?

Airplanes on Hand in the AAF, By Major Type: Jul 1939 to Aug 1945

End of Month	Total	Very Heavy Bombers	Heavy Bombers	Medium Bombers	Light Bombers	Fighters	Recon-naissance	Transports	Trainers	Communi-cations
1939										
Jul	2,402	-	16	400	276	494	356	118	735	7
Aug	2,440	-	18	414	276	492	359	129	745	7
[Germany invades Poland, 1 Sep 1939]										
Sep	2,473	-	22	428	278	489	359	136	754	7
Oct	2,507	-	27	446	277	490	365	137	758	7
Nov	2,536	-	32	458	275	498	375	136	755	7
Dec	2,546	-	39	464	274	492	378	131	761	7
1940										
Jan	2,588	-	45	466	271	464	409	128	798	7
Feb	2,658	-	49	470	271	458	415	128	860	7
Mar	2,709	-	54	468	267	453	415	125	920	7
Apr	2,806	-	54	468	263	451	416	125	1,022	7
May	2,906	-	54	470	259	459	410	124	1,123	7
Jun	2,966	-	54	478	166	477	414	127	1,243	7
[France surrenders to Germany, 25 Jun 1940] [Battle of Britain begins, 10 July 1940]										
Jul	3,102	-	56	483	161	500	410	128	1,357	7
Aug	3,295	-	65	485	158	539	407	128	1,506	7

Source: Army Air Forces Statistical Digest, WW II

What makes this data not tidy?

	A	AA	AB	AC	AD	AE	AF	AG	AH
1	Estimated HIV Prevalence% - (Ages 15-49)	2004	2005	2006	2007	2008	2009	2010	2011
2	Abkhazia								
3	Afghanistan						0.06	0.06	0.06
4	Akrotiri and Dhekelia								
5	Albania								
6	Algeria	0.1	0.1	0.1	0.1	0.1			
7	American Samoa								
8	Andorra								
9	Angola	1.9	1.9	1.9	1.9	2	2.1	2.1	2.1
10	Anguilla								
11	Antigua and Barbuda								
12	Argentina	0.4	0.4	0.4	0.4	0.5	0.4	0.4	0.4
13	Armenia	0.1	0.1	0.1	0.1	0.1	0.2	0.2	0.2
14	Aruba								
15	Australia	0.1	0.1	0.1	0.1	0.1	0.2	0.2	0.2
16	Austria	0.2	0.2	0.2	0.3	0.3	0.3	0.4	0.4
17	Azerbaijan	0.06	0.06	0.06	0.1	0.1	0.1	0.1	0.1
18	Bahamas	3	3	3	3.1	3.1	2.9	2.8	2.8

Source: Gapminder, Estimated HIV prevalence among 15-49 year olds

What makes this data not tidy?

Subject	United States			
	Estimate	Margin of Error	Percent	Percent Margin of Error
EMPLOYMENT STATUS				
Population 16 years and over	255,797,692	+/-17,051	255,797,692	(X)
In labor force	162,184,325	+/-135,158	63.4%	+/-0.1
Civilian labor force	161,159,470	+/-127,501	63.0%	+/-0.1
Employed	150,599,165	+/-138,066	58.9%	+/-0.1
Unemployed	10,560,305	+/-27,385	4.1%	+/-0.1
Armed Forces	1,024,855	+/-10,363	0.4%	+/-0.1
Not in labor force	93,613,367	+/-126,007	36.6%	+/-0.1
Civilian labor force	161,159,470	+/-127,501	161,159,470	(X)
Unemployment Rate	(X)	(X)	6.6%	+/-0.1
Females 16 years and over	131,092,196	+/-11,187	131,092,196	(X)
In labor force	76,493,327	+/-75,824	58.4%	+/-0.1
Civilian labor force	76,350,498	+/-75,238	58.2%	+/-0.1
Employed	71,451,559	+/-79,007	54.5%	+/-0.1
Own children of the householder under 6 years	22,939,897	+/-14,240	22,939,897	(X)
All parents in family in labor force	14,957,537	+/-36,506	65.2%	+/-0.1
Own children of the householder 6 to 17 years	47,007,147	+/-19,644	47,007,147	(X)
All parents in family in labor force	33,238,793	+/-49,036	70.7%	+/-0.1

Source: US Census Fact Finder, General Economic Characteristics, ACS 2017

Displaying vs. summarising data

Output Code

```
## # A tibble: 87 x 3
##    name               height  mass
##    <chr>               <int> <dbl>
##  1 Luke Skywalker        172    77
##  2 C-3PO                 167    75
##  3 R2-D2                  96    32
##  4 Darth Vader           202   136
##  5 Leia Organa           150    49
##  6 Owen Lars             178   120
##  7 Beru Whitesun lars    165    75
##  8 R5-D4                  97    32
##  9 Biggs Darklighter     183    84
## 10 Obi-Wan Kenobi        182    77
## # ... with 77 more rows
```

```
## # A tibble: 3 x 2
##   gender    avg_ht
##   <chr>      <dbl>
## 1 feminine    165.
## 2 masculine   177.
## 3 <NA>        181.
```

Displaying vs. summarising data

Output | Code

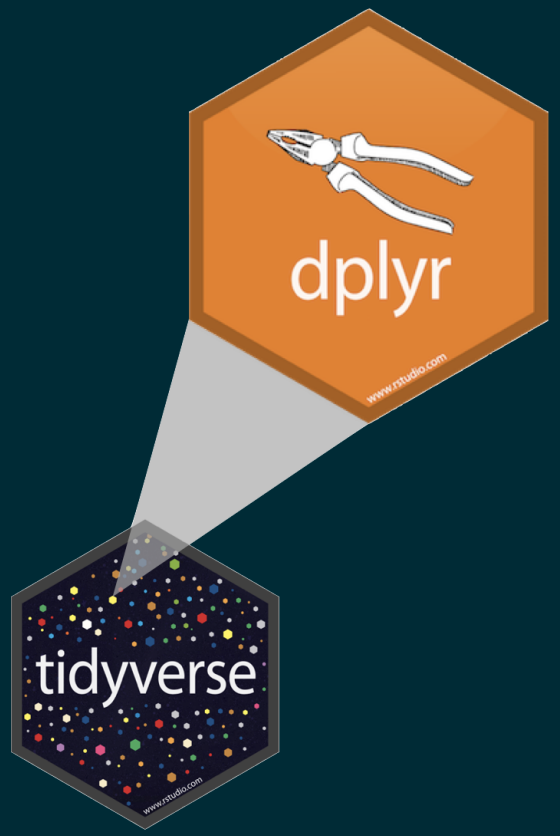
```
starwars %>%
  select(name, height, mass)
```

```
starwars %>%
  group_by(gender) %>%
  summarize(
    avg_ht = mean(height, na.rm = TRUE)
    )
```

Grammar of data wrangling

A grammar of data wrangling...

... based on the concepts of functions as verbs that manipulate data frames



- `select`: pick columns by name
- `arrange`: reorder rows
- `slice`: pick rows using index(es)
- `filter`: pick rows matching criteria
- `distinct`: filter for unique rows
- `mutate`: add new variables
- `summarise`: reduce variables to values
- `group_by`: for grouped operations
- ... (many more)

Rules of dplyr functions

- First argument is *always* a data frame
- Subsequent arguments say what to do with that data frame
- Always return a data frame
- Don't modify in place

Data: Hotel bookings

- Data from two hotels: one resort and one city hotel
- Observations: Each row represents a hotel booking
- Goal for original data collection: Development of prediction models to classify a hotel booking's likelihood to be cancelled (Antonia et al., 2019)

```
hotels <- read_csv("data/hotels.csv")
```

Source: TidyTuesday

First look: Variables

```
names(hotels)
```

```
##  [1] "hotel"
##  [2] "is_canceled"
##  [3] "lead_time"
##  [4] "arrival_date_year"
##  [5] "arrival_date_month"
##  [6] "arrival_date_week_number"
##  [7] "arrival_date_day_of_month"
##  [8] "stays_in_weekend_nights"
##  [9] "stays_in_week_nights"
## [10] "adults"
## [11] "children"
## [12] "babies"
## [13] "meal"
## [14] "country"
## [15] "market_segment"
## [16] "distribution_channel"
## [17] "is_repeated_guest"
## [18] "previous_cancellations"
...
```

Second look: Overview

```
glimpse(hotels)
```

```
## Rows: 119,390
## Columns: 32
## $ hotel                         <chr> "Resort Hotel", "Resort ~
## $ is_canceled                   <dbl> 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ lead_time                     <dbl> 342, 737, 7, 13, 14, 14,~
## $ arrival_date_year             <dbl> 2015, 2015, 2015, 2015, ~
## $ arrival_date_month            <chr> "July", "July", "July", ~
## $ arrival_date_week_number      <dbl> 27, 27, 27, 27, 27, 27, ~
## $ arrival_date_day_of_month     <dbl> 1, 1, 1, 1, 1, 1, 1, 1, ~
## $ stays_in_weekend_nights       <dbl> 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ stays_in_week_nights          <dbl> 0, 0, 1, 1, 2, 2, 2, 2, ~
## $ adults                        <dbl> 2, 2, 1, 1, 2, 2, 2, 2, ~
## $ children                      <dbl> 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ babies                        <dbl> 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ meal                          <chr> "BB", "BB", "BB", "BB", ~
## $ country                       <chr> "PRT", "PRT", "GBR", "GB~
## $ market_segment                <chr> "Direct", "Direct", "Dir~
## $ distribution_channel          <chr> "Direct", "Direct", "Dir~
...
```

Select a single column

View only `lead_time` (number of days between booking and arrival date):

```
select(hotels, lead_time)
```

```
## # A tibble: 119,390 x 1
##    lead_time
##        <dbl>
##  1       342
##  2       737
##  3         7
##  4        13
##  5        14
##  6        14
##  7         0
##  8         9
##  9        85
## 10        75
## # ... with 119,380 more rows
```

Select a single column

```
select(
  hotels,
  lead_time
  )
```

- Start with the function (a verb): `select()`

Select a single column

```
select(
  hotels,
  lead_time
  )
```

- Start with the function (a verb): `select()`
- First argument: data frame we're working with , `hotels`

Select a single column

```
select(
  hotels,
  lead_time
  )
```

- Start with the function (a verb): `select()`
- First argument: data frame we're working with , `hotels`
- Second argument: variable we want to select, `lead_time`

Select a single column

```
select(
  hotels,
  lead_time
  )
```

```
## # A tibble: 119,390 x 1
##    lead_time
##        <dbl>
##  1       342
##  2       737
##  3         7
##  4        13
##  5        14
##  6        14
##  7         0
##  8         9
##  9        85
## 10        75
## # ... with 119,380 more rows
```

- Start with the function (a verb): `select()`
- First argument: data frame we're working with , `hotels`
- Second argument: variable we want to select, `lead_time`
- Result: data frame with 119390 rows and 1 column

dplyr functions always expect a data frame and always yield a data frame.

```
select(hotels, lead_time)
```

```
## # A tibble: 119,390 x 1
##    lead_time
##        <dbl>
##  1       342
##  2       737
##  3         7
##  4        13
##  5        14
##  6        14
##  7         0
##  8         9
##  9        85
## 10        75
## # ... with 119,380 more rows
```

Select multiple columns

View only the `hotel` type and `lead_time`:

Select multiple columns

View only the `hotel` type and `lead_time`:

```
select(hotels, hotel, lead_time)
```

```
## # A tibble: 119,390 x 2
##    hotel        lead_time
##    <chr>            <dbl>
##  1 Resort Hotel       342
##  2 Resort Hotel       737
##  3 Resort Hotel         7
##  4 Resort Hotel        13
##  5 Resort Hotel        14
##  6 Resort Hotel        14
##  7 Resort Hotel         0
##  8 Resort Hotel         9
##  9 Resort Hotel        85
## 10 Resort Hotel        75
## # ... with 119,380 more rows
```

Select multiple columns

View only the `hotel` type and `lead_time`:

```
select(hotels, hotel, lead_time)
```

```
## # A tibble: 119,390 x 2
##    hotel        lead_time
##    <chr>            <dbl>
##  1 Resort Hotel       342
##  2 Resort Hotel       737
##  3 Resort Hotel         7
##  4 Resort Hotel        13
##  5 Resort Hotel        14
##  6 Resort Hotel        14
##  7 Resort Hotel         0
##  8 Resort Hotel         9
##  9 Resort Hotel        85
## 10 Resort Hotel        75
## # ... with 119,380 more rows
```

> What if we wanted to select these columns, and then arrange the data in descending order of lead time?

Data wrangling, step-by-step

Select:

```
hotels %>%
  select(hotel, lead_time)
```

```
## # A tibble: 119,390 x 2
##    hotel        lead_time
##    <chr>            <dbl>
##  1 Resort Hotel       342
##  2 Resort Hotel       737
##  3 Resort Hotel         7
##  4 Resort Hotel        13
##  5 Resort Hotel        14
##  6 Resort Hotel        14
##  7 Resort Hotel         0
##  8 Resort Hotel         9
##  9 Resort Hotel        85
## 10 Resort Hotel        75
## # ... with 119,380 more rows
```

Data wrangling, step-by-step

Select:

```
hotels %>%
  select(hotel, lead_time)
```

```
## # A tibble: 119,390 x 2
##    hotel        lead_time
##    <chr>            <dbl>
##  1 Resort Hotel       342
##  2 Resort Hotel       737
##  3 Resort Hotel         7
##  4 Resort Hotel        13
##  5 Resort Hotel        14
##  6 Resort Hotel        14
##  7 Resort Hotel         0
##  8 Resort Hotel         9
##  9 Resort Hotel        85
## 10 Resort Hotel        75
## # ... with 119,380 more rows
```

Select, then arrange:

```
hotels %>%
  select(hotel, lead_time) %>%
  arrange(desc(lead_time))
```

```
## # A tibble: 119,390 x 2
##    hotel        lead_time
##    <chr>            <dbl>
##  1 Resort Hotel       737
##  2 Resort Hotel       709
##  3 City Hotel         629
##  4 City Hotel         629
##  5 City Hotel         629
##  6 City Hotel         629
##  7 City Hotel         629
##  8 City Hotel         629
##  9 City Hotel         629
## 10 City Hotel         629
## # ... with 119,380 more rows
```

Pipes

What is a pipe?

In programming, a pipe is a technique for passing information from one process to another.

What is a pipe?

In programming, a pipe is a technique for passing information from one process to another.

- Start with the data frame `hotels`, and pass it to the `select()` function,

```
hotels %>%
  select(hotel, lead_time) %>%
  arrange(desc(lead_time))
```

```
## # A tibble: 119,390 x 2
##    hotel        lead_time
##    <chr>            <dbl>
##  1 Resort Hotel       737
##  2 Resort Hotel       709
##  3 City Hotel         629
##  4 City Hotel         629
##  5 City Hotel         629
##  6 City Hotel         629
##  7 City Hotel         629
##  8 City Hotel         629
##  9 City Hotel         629
## 10 City Hotel         629
## # ... with 119,380 more rows
```

What is a pipe?

In programming, a pipe is a technique for passing information from one process to another.

- Start with the data frame `hotels`, and pass it to the `select()` function,
- then we select the variables `hotel` and `lead_time`,

```
hotels %>%
  select(hotel, lead_time) %>%
  arrange(desc(lead_time))
```

```
## # A tibble: 119,390 x 2
##    hotel        lead_time
##    <chr>            <dbl>
##  1 Resort Hotel       737
##  2 Resort Hotel       709
##  3 City Hotel         629
##  4 City Hotel         629
##  5 City Hotel         629
##  6 City Hotel         629
##  7 City Hotel         629
##  8 City Hotel         629
##  9 City Hotel         629
## 10 City Hotel         629
## # ... with 119,380 more rows
```

What is a pipe?

In programming, a pipe is a technique for passing information from one process to another.

- Start with the data frame `hotels`, and pass it to the `select()` function,
- then we select the variables `hotel` and `lead_time`,
- and then we arrange the data frame by `lead_time` in descending order.

```
hotels %>%
  select(hotel, lead_time) %>%
  arrange(desc(lead_time))
```

```
## # A tibble: 119,390 x 2
##    hotel        lead_time
##    <chr>            <dbl>
##  1 Resort Hotel       737
##  2 Resort Hotel       709
##  3 City Hotel         629
##  4 City Hotel         629
##  5 City Hotel         629
##  6 City Hotel         629
##  7 City Hotel         629
##  8 City Hotel         629
##  9 City Hotel         629
## 10 City Hotel         629
## # ... with 119,380 more rows
```

Aside

The pipe operator is implemented in the package **magrittr**, though we don't need to load this package explicitly since **tidyverse** does this for us.

Aside

The pipe operator is implemented in the package **magrittr**, though we don't need to load this package explicitly since **tidyverse** does this for us.

> Any guesses as to why the package is called magrittr?

Aside

The pipe operator is implemented in the package **magrittr**, though we don't need to load this package explicitly since **tidyverse** does this for us.

> Any guesses as to why the package is called magrittr?

How does a pipe work?

- You can think about the following sequence of actions - find keys, unlock car, start car, drive to work, park.

How does a pipe work?

- You can think about the following sequence of actions - find keys, unlock car, start car, drive to work, park.
- Expressed as a set of nested functions in R pseudocode this would look like:

```
park(drive(start_car(find("keys")), to = "work"))
```

How does a pipe work?

- You can think about the following sequence of actions - find keys, unlock car, start car, drive to work, park.
- Expressed as a set of nested functions in R pseudocode this would look like:

```
park(drive(start_car(find("keys")), to = "work"))
```

- Writing it out using pipes give it a more natural (and easier to read) structure:

```
find("keys") %>%
  start_car() %>%
  drive(to = "work") %>%
  park()
```

A note on piping and layering

- %>% used mainly in **dplyr** pipelines, *we pipe the output of the previous line of code as the first input of the next line of code*

A note on piping and layering

- %>% used mainly in **dplyr** pipelines, *we pipe the output of the previous line of code as the first input of the next line of code*
- + used in **ggplot2** plots is used for "layering", *we create the plot in layers, separated by +*

dplyr

✗

```
hotels +
  select(hotel, lead_time)
```

```
## Error in select(hotel, lead_time): object 'hotel' not found
```

☑

```
hotels %>%
  select(hotel, lead_time)
```

```
## # A tibble: 119,390 x 2
##    hotel        lead_time
##    <chr>            <dbl>
##  1 Resort Hotel       342
##  2 Resort Hotel       737
##  3 Resort Hotel         7
...
```

ggplot2

✕

```
ggplot(hotels, aes(x = hotel, fill = deposit_type)) %>%
  geom_bar()
```

```
## Error: `mapping` must be created by `aes()`
## Did you use %>% instead of +?
```

☑

```
ggplot(hotels, aes(x = hotel, fill = deposit_type)) +
  geom_bar()
```

Code styling

Many of the styling principles are consistent across `%>%` and `+`:

- always a space before
- always a line break after (for pipelines with more than 2 lines)

✗

```
ggplot(hotels,aes(x=hotel,y=deposit_type))+geom_bar()
```

✓

```
ggplot(hotels, aes(x = hotel, y = deposit_type)) +
  geom_bar()
```

We...

Data: Hotel bookings

- Data from two hotels: one resort and one city hotel
- Observations: Each row represents a hotel booking

```
hotels <- read_csv("data/hotels.csv")
```

`select`, `arrange`, and `slice`

`select` to keep variables

```
hotels %>%
  select(hotel, lead_time)
```

```
## # A tibble: 119,390 x 2
##    hotel        lead_time
##    <chr>            <dbl>
##  1 Resort Hotel       342
##  2 Resort Hotel       737
##  3 Resort Hotel         7
##  4 Resort Hotel        13
##  5 Resort Hotel        14
##  6 Resort Hotel        14
##  7 Resort Hotel         0
##  8 Resort Hotel         9
##  9 Resort Hotel        85
## 10 Resort Hotel        75
## # ... with 119,380 more rows
```

`select` to exclude variables

```
hotels %>%
  select(-agent)
```

```
## # A tibble: 119,390 x 31
##    hotel  is_canceled lead_time arrival_date_ye~ arrival_date_mo~
##    <chr>        <dbl>     <dbl>            <dbl> <chr>
##  1 Resor~           0       342             2015 July
##  2 Resor~           0       737             2015 July
##  3 Resor~           0         7             2015 July
##  4 Resor~           0        13             2015 July
##  5 Resor~           0        14             2015 July
##  6 Resor~           0        14             2015 July
##  7 Resor~           0         0             2015 July
##  8 Resor~           0         9             2015 July
##  9 Resor~           1        85             2015 July
## 10 Resor~           1        75             2015 July
## # ... with 119,380 more rows, and 26 more variables:
## #   arrival_date_week_number <dbl>,
## #   arrival_date_day_of_month <dbl>,
## #   stays_in_weekend_nights <dbl>, stays_in_week_nights <dbl>,
## #   adults <dbl>, children <dbl>, babies <dbl>, meal <chr>,
...
```

`select` a range of variables

```
hotels %>%
  select(hotel:arrival_date_month)
```

```
## # A tibble: 119,390 x 5
##    hotel  is_canceled lead_time arrival_date_ye~ arrival_date_mo~
##    <chr>        <dbl>     <dbl>            <dbl> <chr>
##  1 Resor~           0       342             2015 July
##  2 Resor~           0       737             2015 July
##  3 Resor~           0         7             2015 July
##  4 Resor~           0        13             2015 July
##  5 Resor~           0        14             2015 July
##  6 Resor~           0        14             2015 July
##  7 Resor~           0         0             2015 July
##  8 Resor~           0         9             2015 July
##  9 Resor~           1        85             2015 July
## 10 Resor~           1        75             2015 July
## # ... with 119,380 more rows
```

`select` variables with certain characteristics

```
hotels %>%
  select(starts_with("arrival"))
```

```
## # A tibble: 119,390 x 4
##    arrival_date_year arrival_date_month arrival_date_week_number
##                <dbl> <chr>                                  <dbl>
##  1              2015 July                                      27
##  2              2015 July                                      27
##  3              2015 July                                      27
##  4              2015 July                                      27
##  5              2015 July                                      27
##  6              2015 July                                      27
##  7              2015 July                                      27
##  8              2015 July                                      27
##  9              2015 July                                      27
## 10              2015 July                                      27
## # ... with 119,380 more rows, and 1 more variable:
## #   arrival_date_day_of_month <dbl>
```

`select` variables with certain characteristics

```
hotels %>%
  select(ends_with("type"))
```

```
## # A tibble: 119,390 x 4
##    reserved_room_type assigned_room_t~ deposit_type customer_type
##    <chr>              <chr>            <chr>        <chr>
##  1 C                  C                No Deposit   Transient
##  2 C                  C                No Deposit   Transient
##  3 A                  C                No Deposit   Transient
##  4 A                  A                No Deposit   Transient
##  5 A                  A                No Deposit   Transient
##  6 A                  A                No Deposit   Transient
##  7 C                  C                No Deposit   Transient
##  8 C                  C                No Deposit   Transient
##  9 A                  A                No Deposit   Transient
## 10 D                  D                No Deposit   Transient
## # ... with 119,380 more rows
```

Select helpers

- `starts_with()`: Starts with a prefix
- `ends_with()`: Ends with a suffix
- `contains()`: Contains a literal string
- `num_range()`: Matches a numerical range like x01, x02, x03
- `one_of()`: Matches variable names in a character vector
- `everything()`: Matches all variables
- `last_col()`: Select last variable, possibly with an offset
- `matches()`: Matches a regular expression (a sequence of symbols/characters expressing a string/pattern to be searched for within text)

See help for any of these functions for more info, e.g. `?everything`.

`arrange` in ascending / descending order

```
hotels %>%
  select(adults, children, babies) %>%
  arrange(babies)
```

```
## # A tibble: 119,390 x 3
##    adults children babies
##     <dbl>    <dbl>  <dbl>
##  1      2        0      0
##  2      2        0      0
##  3      1        0      0
##  4      1        0      0
##  5      2        0      0
##  6      2        0      0
##  7      2        0      0
##  8      2        0      0
##  9      2        0      0
## 10      2        0      0
## # ... with 119,380 more rows
```

```
hotels %>%
  select(adults, children, babies) %>%
  arrange(desc(babies))
```

```
## # A tibble: 119,390 x 3
##    adults children babies
##     <dbl>    <dbl>  <dbl>
##  1      2        0     10
##  2      1        0      9
##  3      2        0      2
##  4      2        0      2
##  5      2        0      2
##  6      2        0      2
##  7      2        0      2
##  8      2        0      2
##  9      2        0      2
## 10      2        0      2
## # ... with 119,380 more rows
```

`slice` for certain row numbers

```
# first five
hotels %>%
  slice(1:5)
```

```
## # A tibble: 5 x 32
##   hotel   is_canceled lead_time arrival_date_ye~ arrival_date_mo~
##   <chr>         <dbl>     <dbl>            <dbl> <chr>
## 1 Resort~           0       342             2015 July
## 2 Resort~           0       737             2015 July
## 3 Resort~           0         7             2015 July
## 4 Resort~           0        13             2015 July
## 5 Resort~           0        14             2015 July
## # ... with 27 more variables: arrival_date_week_number <dbl>,
## #   arrival_date_day_of_month <dbl>,
## #   stays_in_weekend_nights <dbl>, stays_in_week_nights <dbl>,
## #   adults <dbl>, children <dbl>, babies <dbl>, meal <chr>,
## #   country <chr>, market_segment <chr>,
## #   distribution_channel <chr>, is_repeated_guest <dbl>,
## #   previous_cancellations <dbl>, ...
```

In R, you can use the # for adding comments to your code. Any text following # will be printed as is, and won't be run as R code. This is useful for leaving comments in your code and for temporarily disabling certain lines of code while debugging.

```
hotels %>%
  # slice the first five rows  # this line is a comment
  #select(hotel) %>%           # this one doesn't run
  slice(1:5)                   # this line runs
```

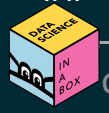
```
## # A tibble: 5 x 32
##   hotel   is_canceled lead_time arrival_date_ye~ arrival_date_mo~
##   <chr>         <dbl>     <dbl>            <dbl> <chr>
## 1 Resort~           0       342             2015 July
## 2 Resort~           0       737             2015 July
## 3 Resort~           0         7             2015 July
## 4 Resort~           0        13             2015 July
## 5 Resort~           0        14             2015 July
## # ... with 27 more variables: arrival_date_week_number <dbl>,
## #   arrival_date_day_of_month <dbl>,
...
```

filter

`filter` to select a subset of rows

```
# bookings in City Hotels
hotels %>%
  filter(hotel == "City Hotel")
```

```
## # A tibble: 79,330 x 32
##    hotel  is_canceled lead_time arrival_date_ye~ arrival_date_mo~
##    <chr>        <dbl>     <dbl>            <dbl> <chr>
##  1 City ~           0         6             2015 July
##  2 City ~           1        88             2015 July
##  3 City ~           1        65             2015 July
##  4 City ~           1        92             2015 July
##  5 City ~           1       100             2015 July
##  6 City ~           1        79             2015 July
##  7 City ~           0         3             2015 July
##  8 City ~           1        63             2015 July
##  9 City ~           1        62             2015 July
## 10 City ~           1        62             2015 July
## # ... with 79,320 more rows, and 27 more variables:
## #   arrival_date_week_number <dbl>,
## #   arrival_date_day_of_month <dbl>,
## #   stays_in_weekend_nights <dbl>, stays_in_week_nights <dbl>,
```



`filter` for many conditions at once

```
hotels %>%
  filter(
    adults == 0,
    children >= 1
    ) %>%
  select(adults, babies, children)
```

```
## # A tibble: 223 x 3
##    adults babies children
##     <dbl>  <dbl>    <dbl>
##  1      0      0        3
##  2      0      0        2
##  3      0      0        2
##  4      0      0        2
##  5      0      0        2
##  6      0      0        3
##  7      0      1        2
##  8      0      0        2
##  9      0      0        2
## 10      0      0        2
## # ... with 213 more rows
```

`filter` for more complex conditions

```
# bookings with no adults and some children or babies in the room
hotels %>%
  filter(
    adults == 0,
    children >= 1 | babies >= 1     # | means or
    ) %>%
  select(adults, babies, children)
```

```
## # A tibble: 223 x 3
##    adults babies children
##     <dbl>  <dbl>    <dbl>
##  1      0      0        3
##  2      0      0        2
##  3      0      0        2
##  4      0      0        2
##  5      0      0        2
##  6      0      0        3
##  7      0      1        2
##  8      0      0        2
##  9      0      0        2
## 10      0      0        2
## # ... with 213 more rows
```

Logical operators in R

operator	definition	operator	definition
`<`	less than	`x | y`	`x` OR `y`
`<=`	less than or equal to	`is.na(x)`	test if `x` is `NA`
`>`	greater than	`!is.na(x)`	test if `x` is not `NA`
`>=`	greater than or equal to	`x %in% y`	test if `x` is in `y`
`==`	exactly equal to	`!(x %in% y)`	test if `x` is not in `y`
`!=`	not equal to	`!x`	not `x`
`x & y`	`x` AND `y`		

Your turn!

Time to actually play around with the Hotels dataset!

- Go to RStudio and start `AE 04 - Hotels + Data wrangling`.
- Open the R Markdown document and complete Exercises 1 - 4.

`distinct` and `count`

`distinct` to filter for unique rows

... and `arrange` to order alphabetically

```
hotels %>%
  distinct(market_segment) %>%
  arrange(market_segment)
```

```
## # A tibble: 8 x 1
##   market_segment
##   <chr>
## 1 Aviation
## 2 Complementary
## 3 Corporate
## 4 Direct
## 5 Groups
## 6 Offline TA/TO
## 7 Online TA
## 8 Undefined
```

```
hotels %>%
  distinct(hotel, market_segment) %>%
  arrange(hotel, market_segment)
```

```
## # A tibble: 14 x 2
##    hotel        market_segment
##    <chr>        <chr>
##  1 City Hotel   Aviation
##  2 City Hotel   Complementary
##  3 City Hotel   Corporate
##  4 City Hotel   Direct
##  5 City Hotel   Groups
##  6 City Hotel   Offline TA/TO
##  7 City Hotel   Online TA
##  8 City Hotel   Undefined
##  9 Resort Hotel Complementary
## 10 Resort Hotel Corporate
...
```

`count` to create frequency tables

```
# alphabetical order by default
hotels %>%
  count(market_segment)
```

```
## # A tibble: 8 x 2
##   market_segment     n
##   <chr>          <int>
## 1 Aviation         237
## 2 Complementary    743
## 3 Corporate       5295
## 4 Direct         12606
## 5 Groups         19811
## 6 Offline TA/TO  24219
## 7 Online TA      56477
## 8 Undefined          2
```

`count` to create frequency tables

```
# alphabetical order by default
hotels %>%
  count(market_segment)
```

```
## # A tibble: 8 x 2
##   market_segment     n
##   <chr>          <int>
## 1 Aviation         237
## 2 Complementary    743
## 3 Corporate       5295
## 4 Direct         12606
## 5 Groups         19811
## 6 Offline TA/TO  24219
## 7 Online TA      56477
## 8 Undefined          2
```

```
# descending frequency order
hotels %>%
  count(market_segment, sort = TRUE)
```

```
## # A tibble: 8 x 2
##   market_segment     n
##   <chr>          <int>
## 1 Online TA      56477
## 2 Offline TA/TO  24219
## 3 Groups         19811
## 4 Direct         12606
## 5 Corporate       5295
## 6 Complementary    743
## 7 Aviation         237
## 8 Undefined          2
```

count and arrange

```
# ascending frequency order
hotels %>%
  count(market_segment) %>%
  arrange(n)
```

```
## # A tibble: 8 x 2
##   market_segment     n
##   <chr>          <int>
## 1 Undefined          2
## 2 Aviation         237
## 3 Complementary    743
## 4 Corporate       5295
## 5 Direct         12606
## 6 Groups         19811
## 7 Offline TA/TO  24219
## 8 Online TA      56477
```

```
# descending frequency order
# just like adding sort = TRUE
hotels %>%
  count(market_segment) %>%
  arrange(desc(n))
```

```
## # A tibble: 8 x 2
##   market_segment     n
##   <chr>          <int>
## 1 Online TA      56477
## 2 Offline TA/TO  24219
## 3 Groups         19811
## 4 Direct         12606
## 5 Corporate       5295
## 6 Complementary    743
## 7 Aviation         237
## 8 Undefined          2
```

`count` for multiple variables

```
hotels %>%
  count(hotel, market_segment)
```

```
## # A tibble: 14 x 3
##    hotel        market_segment     n
##    <chr>        <chr>          <int>
##  1 City Hotel   Aviation         237
##  2 City Hotel   Complementary    542
##  3 City Hotel   Corporate       2986
##  4 City Hotel   Direct          6093
##  5 City Hotel   Groups         13975
##  6 City Hotel   Offline TA/TO  16747
##  7 City Hotel   Online TA      38748
##  8 City Hotel   Undefined          2
##  9 Resort Hotel Complementary    201
## 10 Resort Hotel Corporate       2309
## 11 Resort Hotel Direct          6513
## 12 Resort Hotel Groups          5836
## 13 Resort Hotel Offline TA/TO   7472
## 14 Resort Hotel Online TA      17729
```

order matters when you count

```
# hotel type first
hotels %>%
  count(hotel, market_segment)
```

```
## # A tibble: 14 x 3
##    hotel        market_segment     n
##    <chr>        <chr>          <int>
##  1 City Hotel   Aviation         237
##  2 City Hotel   Complementary    542
##  3 City Hotel   Corporate       2986
##  4 City Hotel   Direct          6093
##  5 City Hotel   Groups         13975
##  6 City Hotel   Offline TA/TO  16747
##  7 City Hotel   Online TA      38748
##  8 City Hotel   Undefined          2
##  9 Resort Hotel Complementary    201
## 10 Resort Hotel Corporate       2309
## 11 Resort Hotel Direct          6513
## 12 Resort Hotel Groups          5836
## 13 Resort Hotel Offline TA/TO   7472
## 14 Resort Hotel Online TA      17729
```

```
# market segment first
hotels %>%
  count(market_segment, hotel)
```

```
## # A tibble: 14 x 3
##    market_segment hotel            n
##    <chr>          <chr>        <int>
##  1 Aviation       City Hotel     237
##  2 Complementary  City Hotel     542
##  3 Complementary  Resort Hotel   201
##  4 Corporate      City Hotel    2986
##  5 Corporate      Resort Hotel  2309
##  6 Direct         City Hotel    6093
##  7 Direct         Resort Hotel  6513
##  8 Groups         City Hotel   13975
##  9 Groups         Resort Hotel  5836
## 10 Offline TA/TO  City Hotel   16747
## 11 Offline TA/TO  Resort Hotel  7472
## 12 Online TA      City Hotel   38748
## 13 Online TA      Resort Hotel 17729
## 14 Undefined      City Hotel       2
```

Your turn!

- Go back to RStudio and continue `AE 04 - Hotels + Data wrangling`.
- Open the R Markdown document and complete Exercises 5 and 6.

mutate

`mutate` to add a new variable

```
hotels %>%
  mutate(little_ones = children + babies) %>%
  select(children, babies, little_ones) %>%
  arrange(desc(little_ones))
```

```
## # A tibble: 119,390 x 3
##    children babies little_ones
##       <dbl>  <dbl>       <dbl>
##  1       10      0          10
##  2        0     10          10
##  3        0      9           9
##  4        2      1           3
##  5        2      1           3
##  6        2      1           3
##  7        3      0           3
##  8        2      1           3
##  9        2      1           3
## 10        3      0           3
## # ... with 119,380 more rows
```

Little ones in resort and city hotels

```
# Resort Hotel
hotels %>%
  mutate(little_ones = children + babies) %>%
  filter(
    little_ones >= 1,
    hotel == "Resort Hotel"
    ) %>%
  select(hotel, little_ones)
```

```
## # A tibble: 3,929 x 2
##    hotel        little_ones
##    <chr>              <dbl>
##  1 Resort Hotel           1
##  2 Resort Hotel           2
##  3 Resort Hotel           2
##  4 Resort Hotel           2
##  5 Resort Hotel           1
##  6 Resort Hotel           1
##  7 Resort Hotel           2
##  8 Resort Hotel           2
##  9 Resort Hotel           1
## 10 Resort Hotel           1
```

```
# City Hotel
hotels %>%
  mutate(little_ones = children + babies) %>%
  filter(
    little_ones >= 1,
    hotel == "City Hotel"
    )  %>%
  select(hotel, little_ones)
```

```
## # A tibble: 5,403 x 2
##    hotel      little_ones
##    <chr>            <dbl>
##  1 City Hotel           1
##  2 City Hotel           1
##  3 City Hotel           2
##  4 City Hotel           1
##  5 City Hotel           1
##  6 City Hotel           1
##  7 City Hotel           1
##  8 City Hotel           1
##  9 City Hotel           1
## 10 City Hotel           1
```

What is happening in the following chunk?

```
hotels %>%
  mutate(little_ones = children + babies) %>%
  count(hotel, little_ones) %>%
  mutate(prop = n / sum(n))
```

```
## # A tibble: 12 x 4
##    hotel        little_ones     n       prop
##    <chr>              <dbl> <int>      <dbl>
##  1 City Hotel             0 73923 0.619
##  2 City Hotel             1  3263 0.0273
##  3 City Hotel             2  2056 0.0172
##  4 City Hotel             3    82 0.000687
##  5 City Hotel             9     1 0.00000838
##  6 City Hotel            10     1 0.00000838
##  7 City Hotel            NA     4 0.0000335
##  8 Resort Hotel           0 36131 0.303
##  9 Resort Hotel           1  2183 0.0183
## 10 Resort Hotel           2  1716 0.0144
## 11 Resort Hotel           3    29 0.000243
## 12 Resort Hotel          10     1 0.00000838
```

`summarise` and `group_by`

`summarise` for summary stats

```
# mean average daily rate for all bookings
hotels %>%
  summarise(mean_adr = mean(adr))
```

```
## # A tibble: 1 x 1
##   mean_adr
##      <dbl>
## 1     102.
```

`summarise` for summary stats

```
# mean average daily rate for all bookings
hotels %>%
  summarise(mean_adr = mean(adr))
```

```
## # A tibble: 1 x 1
##   mean_adr
##      <dbl>
## 1     102.
```

`summarise()` changes the data frame entirely, it collapses rows down to a single summary statistic, and removes all columns that are irrelevant to the calculation.

`summarise()` also lets you get away with being sloppy and not naming your new column, but that's not recommended!

✗

```
hotels %>%
  summarise(mean(adr))
```

```
## # A tibble: 1 x 1
##   `mean(adr)`
##         <dbl>
## 1        102.
```

✓

```
hotels %>%
  summarise(mean_adr = mean(adr))
```

```
## # A tibble: 1 x 1
##   mean_adr
##      <dbl>
## 1     102.
```

group_by for grouped operations

```
# mean average daily rate for all booking at city and resort hotels
hotels %>%
  group_by(hotel) %>%
  summarise(mean_adr = mean(adr))
```

```
## # A tibble: 2 x 2
##   hotel        mean_adr
##   <chr>           <dbl>
## 1 City Hotel      105.
## 2 Resort Hotel     95.0
```

Calculating frequencies

The following two give the same result, so `count` is simply short for `group_by` then determine frequencies

```
hotels %>%
  group_by(hotel) %>%
  summarise(n = n())
```

```
## # A tibble: 2 x 2
##   hotel            n
##   <chr>        <int>
## 1 City Hotel   79330
## 2 Resort Hotel 40060
```

```
hotels %>%
  count(hotel)
```

```
## # A tibble: 2 x 2
##   hotel            n
##   <chr>        <int>
## 1 City Hotel   79330
## 2 Resort Hotel 40060
```

Multiple summary statistics

`summarise` can be used for multiple summary statistics as well

```
hotels %>%
  summarise(
    min_adr = min(adr),
    mean_adr = mean(adr),
    median_adr = median(adr),
    max_adr = max(adr)
    )
```

```
## # A tibble: 1 x 4
##   min_adr mean_adr median_adr max_adr
##     <dbl>    <dbl>      <dbl>   <dbl>
## 1   -6.38     102.       94.6    5400
```

Your turn!

Time to actually play around with the Hotels dataset!

- Go to RStudio and start `AE 04 - Hotels + Data wrangling`.
- Open the R Markdown document and complete Exercises 7 and 8.

We...

have multiple data frames

want to bring them together

Data: Women in science

Information on 10 women in science who changed the world

name
Ada Lovelace
Marie Curie
Janaki Ammal
Chien-Shiung Wu
Katherine Johnson
Rosalind Franklin
Vera Rubin
Gladys West
Flossie Wong-Staal
Jennifer Doudna

Source: Discover Magazine

Inputs

professions dates works

```
professions
```

```
## # A tibble: 10 x 2
##    name               profession
##    <chr>              <chr>
##  1 Ada Lovelace       Mathematician
##  2 Marie Curie        Physicist and Chemist
##  3 Janaki Ammal       Botanist
##  4 Chien-Shiung Wu    Physicist
##  5 Katherine Johnson  Mathematician
##  6 Rosalind Franklin  Chemist
##  7 Vera Rubin         Astronomer
##  8 Gladys West        Mathematician
##  9 Flossie Wong-Staal Virologist and Molecular Biologist
## 10 Jennifer Doudna    Biochemist
```

Inputs

professions | dates | works

```
dates
```

```
## # A tibble: 8 x 3
##   name               birth_year death_year
##   <chr>                   <dbl>      <dbl>
## 1 Janaki Ammal             1897       1984
## 2 Chien-Shiung Wu          1912       1997
## 3 Katherine Johnson        1918       2020
## 4 Rosalind Franklin        1920       1958
## 5 Vera Rubin               1928       2016
## 6 Gladys West              1930         NA
## 7 Flossie Wong-Staal       1947         NA
## 8 Jennifer Doudna          1964         NA
```

Inputs

professions dates works

```
works
```

```
## # A tibble: 9 x 2
##   name              known_for
##   <chr>             <chr>
## 1 Ada Lovelace      first computer algorithm
## 2 Marie Curie       theory of radioactivity,  discovery of elem~
## 3 Janaki Ammal      hybrid species, biodiversity protection
## 4 Chien-Shiung Wu   confim and refine theory of radioactive bet~
## 5 Katherine Johnson calculations of orbital mechanics critical ~
## 6 Vera Rubin        existence of dark matter
## 7 Gladys West       mathematical modeling of the shape of the E~
## 8 Flossie Wong-Staal first scientist to clone HIV and create a m~
## 9 Jennifer Doudna   one of the primary developers of CRISPR, a ~
```

Desired output

```
## # A tibble: 10 x 5
##    name              profession  birth_year death_year known_for
##    <chr>             <chr>            <dbl>      <dbl> <chr>
##  1 Ada Lovelace      Mathematic~         NA         NA first co~
##  2 Marie Curie       Physicist ~         NA         NA theory o~
##  3 Janaki Ammal      Botanist          1897       1984 hybrid s~
##  4 Chien-Shiung Wu   Physicist         1912       1997 confim a~
##  5 Katherine Johnson Mathematic~       1918       2020 calculat~
##  6 Rosalind Franklin Chemist           1920       1958 <NA>
##  7 Vera Rubin        Astronomer        1928       2016 existenc~
##  8 Gladys West       Mathematic~       1930         NA mathemat~
##  9 Flossie Wong-Staal Virologist~      1947         NA first sc~
## 10 Jennifer Doudna   Biochemist        1964         NA one of t~
```

Inputs, reminder

```
names(professions)
```

```
## [1] "name"       "profession"
```

```
names(dates)
```

```
## [1] "name"       "birth_year" "death_year"
```

```
names(works)
```

```
## [1] "name"      "known_for"
```

```
nrow(professions)
```

```
## [1] 10
```

```
nrow(dates)
```

```
## [1] 8
```

```
nrow(works)
```

```
## [1] 9
```

Joining data frames

Joining data frames

```
something_join(x, y)
```

- `left_join()`: all rows from x
- `right_join()`: all rows from y
- `full_join()`: all rows from both x and y
- `semi_join()`: all rows from x where there are matching values in y, keeping just columns from x
- `inner_join()`: all rows from x where there are matching values in y, return all combination of multiple matches in the case of multiple matches
- `anti_join()`: return all rows from x where there are not matching values in y, never duplicate rows of x
- ...

Setup

For the next few slides...

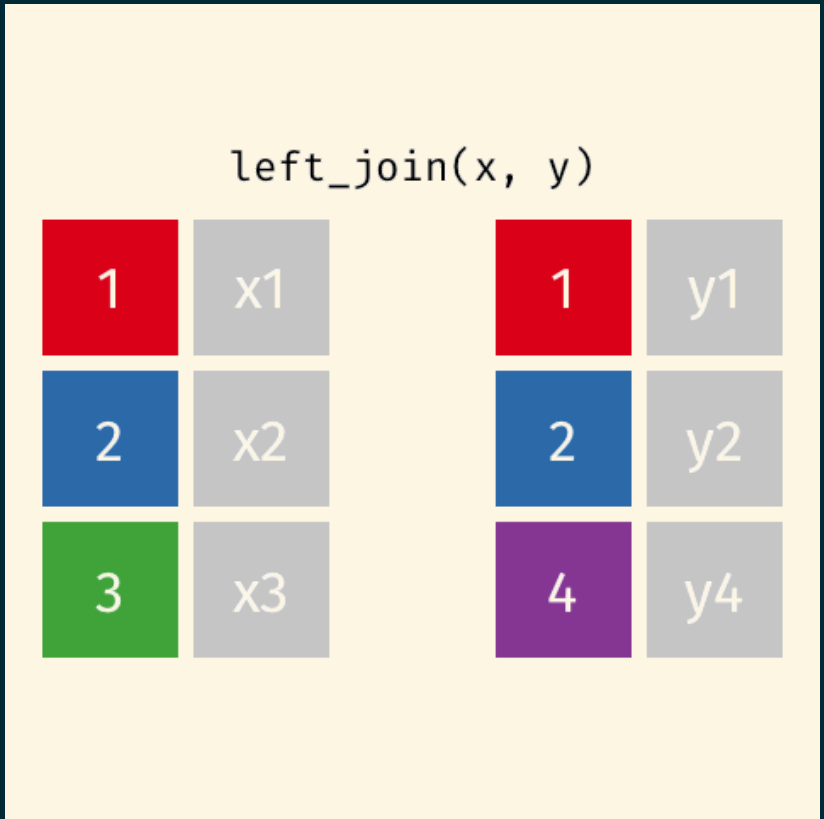
```
x
```

```
## # A tibble: 3 x 2
##      id value_x
##   <dbl> <chr>
## 1     1 x1
## 2     2 x2
## 3     3 x3
```

```
y
```

```
## # A tibble: 3 x 2
##      id value_y
##   <dbl> <chr>
## 1     1 y1
## 2     2 y2
## 3     4 y4
```

left_join()




```
left_join(x, y)
```

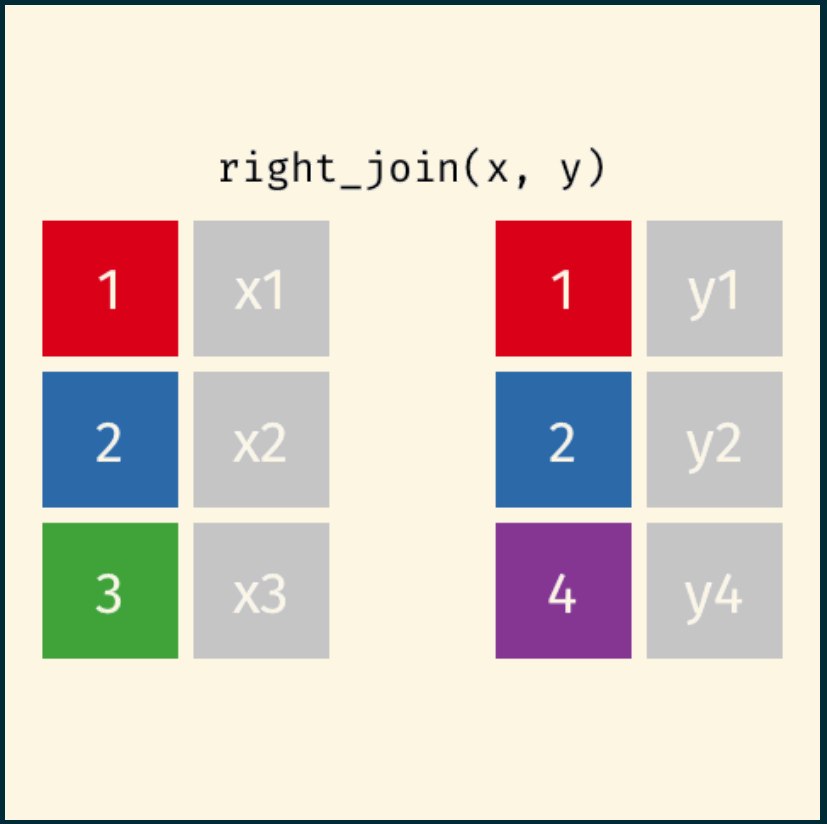
```
## # A tibble: 3 x 3
##      id value_x value_y
##   <dbl> <chr>   <chr>
## 1     1 x1      y1
## 2     2 x2      y2
## 3     3 x3      <NA>
```

left_join()

```
professions %>%
  left_join(dates)
```

```
## # A tibble: 10 x 4
##    name               profession            birth_year death_year
##    <chr>              <chr>                      <dbl>      <dbl>
##  1 Ada Lovelace       Mathematician                 NA         NA
##  2 Marie Curie        Physicist and Chemist         NA         NA
##  3 Janaki Ammal       Botanist                    1897       1984
##  4 Chien-Shiung Wu    Physicist                   1912       1997
##  5 Katherine Johnson  Mathematician               1918       2020
##  6 Rosalind Franklin  Chemist                     1920       1958
##  7 Vera Rubin         Astronomer                  1928       2016
##  8 Gladys West        Mathematician               1930         NA
##  9 Flossie Wong-Staal Virologist and Molec~       1947         NA
## 10 Jennifer Doudna    Biochemist                  1964         NA
```

right_join()




```
right_join(x, y)
```

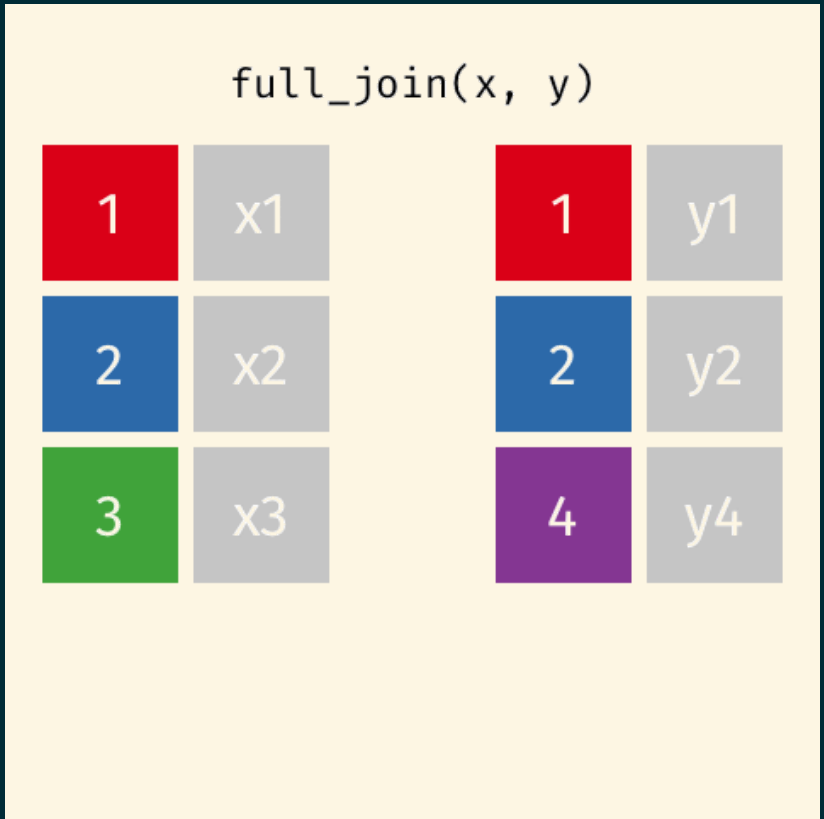
```
## # A tibble: 3 x 3
##      id value_x value_y
##   <dbl> <chr>   <chr>
## 1     1 x1      y1
## 2     2 x2      y2
## 3     4 <NA>    y4
```

right_join()

```
professions %>%
  right_join(dates)
```

```
## # A tibble: 8 x 4
##   name              profession             birth_year death_year
##   <chr>             <chr>                       <dbl>      <dbl>
## 1 Janaki Ammal      Botanist                     1897       1984
## 2 Chien-Shiung Wu   Physicist                    1912       1997
## 3 Katherine Johnson Mathematician                1918       2020
## 4 Rosalind Franklin Chemist                      1920       1958
## 5 Vera Rubin        Astronomer                   1928       2016
## 6 Gladys West       Mathematician                1930         NA
## 7 Flossie Wong-Staal Virologist and Molecu~      1947         NA
## 8 Jennifer Doudna   Biochemist                   1964         NA
```

full_join()




```
full_join(x, y)
```

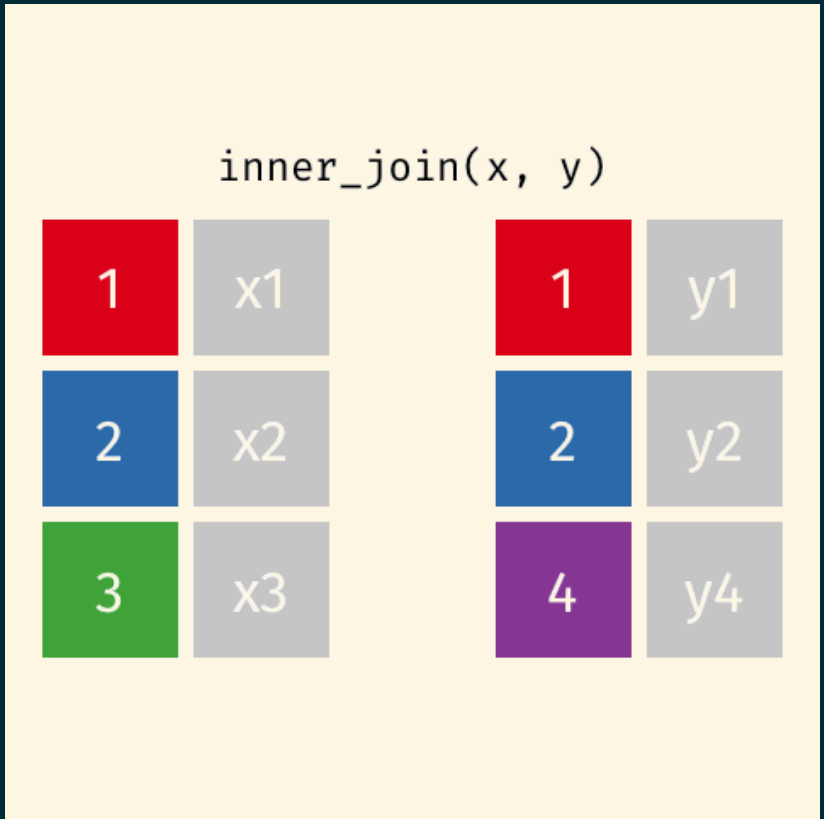
```
## # A tibble: 4 x 3
##      id value_x value_y
##   <dbl> <chr>   <chr>
## 1     1 x1      y1
## 2     2 x2      y2
## 3     3 x3      <NA>
## 4     4 <NA>    y4
```

full_join()

```
dates %>%
  full_join(works)
```

```
## # A tibble: 10 x 4
##    name              birth_year death_year known_for
##    <chr>                  <dbl>      <dbl> <chr>
##  1 Janaki Ammal            1897       1984 hybrid species, biod~
##  2 Chien-Shiung Wu         1912       1997 confim and refine th~
##  3 Katherine Johnson       1918       2020 calculations of orbi~
##  4 Rosalind Franklin       1920       1958 <NA>
##  5 Vera Rubin              1928       2016 existence of dark ma~
##  6 Gladys West             1930         NA mathematical modelin~
##  7 Flossie Wong-Staal      1947         NA first scientist to c~
##  8 Jennifer Doudna         1964         NA one of the primary d~
##  9 Ada Lovelace              NA         NA first computer algor~
## 10 Marie Curie               NA         NA theory of radioactiv~
```

inner_join()




```
inner_join(x, y)
```

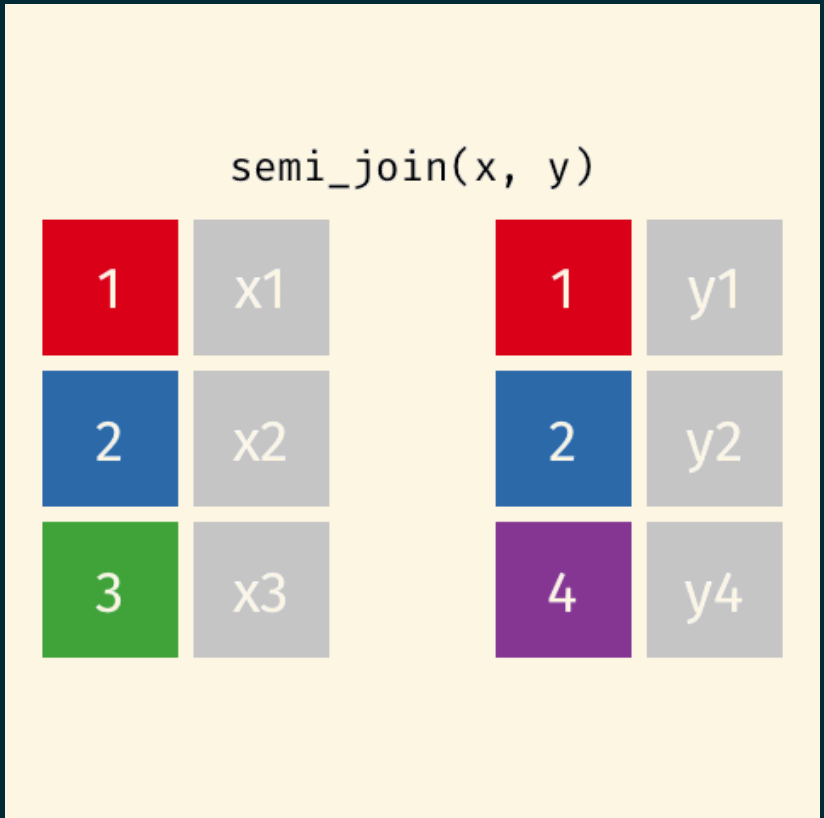
```
## # A tibble: 2 x 3
##      id value_x value_y
##   <dbl> <chr>   <chr>
## 1     1 x1      y1
## 2     2 x2      y2
```

inner_join()

```
dates %>%
  inner_join(works)
```

```
## # A tibble: 7 x 4
##   name              birth_year death_year known_for
##   <chr>                  <dbl>      <dbl> <chr>
## 1 Janaki Ammal            1897       1984 hybrid species, biodi~
## 2 Chien-Shiung Wu         1912       1997 confim and refine the~
## 3 Katherine Johnson       1918       2020 calculations of orbit~
## 4 Vera Rubin              1928       2016 existence of dark mat~
## 5 Gladys West             1930         NA mathematical modeling~
## 6 Flossie Wong-Staal      1947         NA first scientist to cl~
## 7 Jennifer Doudna         1964         NA one of the primary de~
```

semi_join()




```
semi_join(x, y)
```

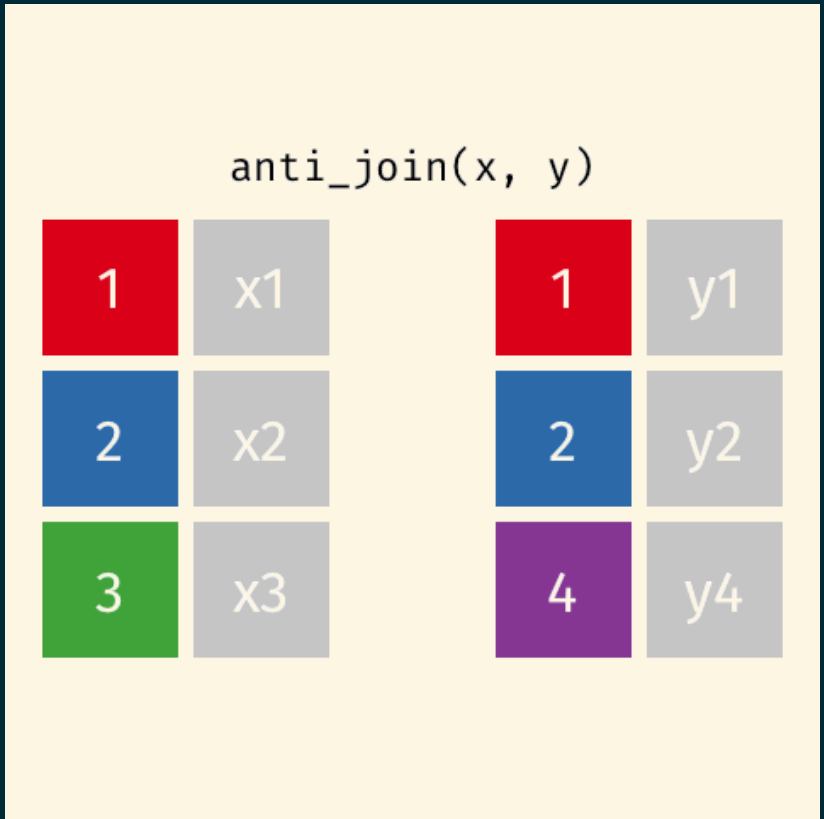
```
## # A tibble: 2 x 2
##      id value_x
##   <dbl> <chr>
## 1     1 x1
## 2     2 x2
```

semi_join()

```
dates %>%
  semi_join(works)
```

```
## # A tibble: 7 x 3
##   name               birth_year death_year
##   <chr>                   <dbl>      <dbl>
## 1 Janaki Ammal             1897       1984
## 2 Chien-Shiung Wu          1912       1997
## 3 Katherine Johnson        1918       2020
## 4 Vera Rubin               1928       2016
## 5 Gladys West              1930         NA
## 6 Flossie Wong-Staal       1947         NA
## 7 Jennifer Doudna          1964         NA
```

anti_join()




```
anti_join(x, y)
```

```
## # A tibble: 1 x 2
##      id value_x
##   <dbl> <chr>
## 1     3 x3
```

anti_join()

```
dates %>%
  anti_join(works)
```

```
## # A tibble: 1 x 3
##   name              birth_year death_year
##   <chr>                  <dbl>      <dbl>
## 1 Rosalind Franklin       1920       1958
```

Putting it altogether

```
professions %>%
  left_join(dates) %>%
  left_join(works)
```

```
## # A tibble: 10 x 5
##    name              profession  birth_year death_year known_for
##    <chr>             <chr>            <dbl>      <dbl> <chr>
##  1 Ada Lovelace      Mathematic~         NA         NA first co~
##  2 Marie Curie       Physicist ~         NA         NA theory o~
##  3 Janaki Ammal      Botanist          1897       1984 hybrid s~
##  4 Chien-Shiung Wu   Physicist         1912       1997 confim a~
##  5 Katherine Johnson Mathematic~       1918       2020 calculat~
##  6 Rosalind Franklin Chemist           1920       1958 <NA>
##  7 Vera Rubin        Astronomer        1928       2016 existenc~
##  8 Gladys West       Mathematic~       1930         NA mathemat~
##  9 Flossie Wong-Staal Virologist~      1947         NA first sc~
## 10 Jennifer Doudna   Biochemist        1964         NA one of t~
```

Case study: Student records

Student records

- Have:
  - Enrolment: official university enrolment records
  - Survey: Student provided info missing students who never filled it out and including students who filled it out but dropped the class
- Want: Survey info for all enrolled in class

Student records

- Have:
  - Enrolment: official university enrolment records
  - Survey: Student provided info missing students who never filled it out and including students who filled it out but dropped the class
- Want: Survey info for all enrolled in class

`enrolment`

```
## # A tibble: 3 x 2
##      id name
##   <dbl> <chr>
## 1     1 Dave Friday
## 2     2 Hermine
## 3     3 Sura Selvarajah
```

`survey`

```
## # A tibble: 4 x 3
##      id name    username
##   <dbl> <chr>   <chr>
## 1     2 Hermine bakealongwithhermine
## 2     3 Sura    surasbakes
## 3     4 Peter   peter_bakes
## 4     5 Mark    thebakingbuddha
```

Student records

In class | Survey missing | Dropped

```
enrolment %>%
  left_join(survey, by = "id")
```

```
## # A tibble: 3 x 4
##      id name.x          name.y  username
##   <dbl> <chr>           <chr>   <chr>
## 1     1 Dave Friday     <NA>    <NA>
## 2     2 Hermine         Hermine bakealongwithhermine
## 3     3 Sura Selvarajah Sura    surasbakes
```

Student records

In class | Survey missing | Dropped

```
enrolment %>%
  anti_join(survey, by = "id")
```

```
## # A tibble: 1 x 2
##      id name
##   <dbl> <chr>
## 1     1 Dave Friday
```

Student records

In class | Survey missing | Dropped

```
survey %>%
  anti_join(enrolment, by = "id")
```

```
## # A tibble: 2 x 3
##      id name  username
##   <dbl> <chr> <chr>
## 1     4 Peter peter_bakes
## 2     5 Mark  thebakingbuddha
```

Case study: Grocery sales

Grocery sales

- Have:
  - Purchases: One row per customer per item, listing purchases they made
  - Prices: One row per item in the store, listing their prices
- Want: Total revenue

Grocery sales

- Have:
  - Purchases: One row per customer per item, listing purchases they made
  - Prices: One row per item in the store, listing their prices
- Want: Total revenue

```
purchases
```

```
## # A tibble: 5 x 2
##   customer_id item
##         <dbl> <chr>
## 1           1 bread
## 2           1 milk
## 3           1 banana
## 4           2 milk
## 5           2 toilet paper
```

```
prices
```

```
## # A tibble: 5 x 2
##   item         price
##   <chr>        <dbl>
## 1 avocado       0.5
## 2 banana        0.15
## 3 bread         1
## 4 milk          0.8
## 5 toilet paper  3
```

Grocery sales

Total revenue | Revenue per customer

```
purchases %>%
  left_join(prices)
```

```
## # A tibble: 5 x 3
##   customer_id item         price
##         <dbl> <chr>        <dbl>
## 1           1 bread          1
## 2           1 milk           0.8
## 3           1 banana         0.15
## 4           2 milk           0.8
## 5           2 toilet paper   3
```

```
purchases %>%
  left_join(prices) %>%
  summarise(total_revenue = sum(price))
```

```
## # A tibble: 1 x 1
##   total_revenue
##           <dbl>
## 1          5.75
```

Grocery sales

Total revenue | Revenue per customer

```
purchases %>%
  left_join(prices)
```

```
## # A tibble: 5 x 3
##   customer_id item         price
##         <dbl> <chr>        <dbl>
## 1           1 bread          1
## 2           1 milk           0.8
## 3           1 banana         0.15
## 4           2 milk           0.8
## 5           2 toilet paper   3
```

```
purchases %>%
  left_join(prices) %>%
  group_by(customer_id) %>%
  summarise(total_revenue = sum(price))
```

```
## # A tibble: 2 x 2
##   customer_id total_revenue
##         <dbl>         <dbl>
## 1           1          1.95
## 2           2          3.8
```

We...

have data organised in an unideal way for our analysis

want to reorganise the data to carry on with our analysis

Data: Sales

We have...

```
## # A tibble: 2 x 4
##   customer_id item_1 item_2       item_3
##         <dbl> <chr>  <chr>        <chr>
## 1           1 bread  milk         banana
## 2           2 milk   toilet paper <NA>
```

Data: Sales

We have...

```
## # A tibble: 2 x 4
##   customer_id item_1 item_2       item_3
##         <dbl> <chr>  <chr>        <chr>
## 1           1 bread  milk         banana
## 2           2 milk   toilet paper <NA>
```

We want...

```
## # A tibble: 6 x 3
##   customer_id item_no item
##         <dbl> <chr>   <chr>
## 1           1 item_1  bread
## 2           1 item_2  milk
## 3           1 item_3  banana
## 4           2 item_1  milk
## 5           2 item_2  toilet paper
## 6           2 item_3  <NA>
```

A grammar of data tidying

The goal of tidyr is to help you tidy your data via

- pivoting for going between wide and long data
- splitting and combining character columns
- nesting and unnesting columns
- clarifying how NAs should be treated

Pivoting data

Not this...

but this!

wide

id	x	y	z
1	a	c	e
2	b	d	f

Wider vs. longer

wider

more columns

```
## # A tibble: 2 x 4
##   customer_id item_1 item_2       item_3
##         <dbl> <chr>  <chr>        <chr>
## 1           1 bread  milk         banana
## 2           2 milk   toilet paper <NA>
```

Wider vs. longer

wider

more columns

```
## # A tibble: 2 x 4
##   customer_id item_1 item_2       item_3
##         <dbl> <chr>  <chr>        <chr>
## 1           1 bread  milk         banana
## 2           2 milk   toilet paper <NA>
```

longer

more rows

```
## # A tibble: 6 x 3
##   customer_id item_no item
##         <dbl> <chr>   <chr>
## 1           1 item_1  bread
## 2           1 item_2  milk
## 3           1 item_3  banana
## 4           2 item_1  milk
## 5           2 item_2  toilet paper
## 6           2 item_3  <NA>
```

pivot_longer()

- `data` (as usual)

```
pivot_longer(
  data,
  cols,
  names_to = "name",
  values_to = "value"
  )
```

pivot_longer()

- `data` (as usual)
- `cols`: columns to pivot into longer format

```
pivot_longer(
  data,
  cols,
  names_to = "name",
  values_to = "value"
  )
```

pivot_longer()

- `data` (as usual)
- `cols`: columns to pivot into longer format
- `names_to`: name of the column where column names of pivoted variables go (character string)

```
pivot_longer(
  data,
  cols,
  names_to = "name",
  values_to = "value"
  )
```

pivot_longer()

- `data` (as usual)
- `cols`: columns to pivot into longer format
- `names_to`: name of the column where column names of pivoted variables go (character string)
- `values_to`: name of the column where data in pivoted variables go (character string)

```
pivot_longer(
  data,
  cols,
  names_to = "name",
  values_to = "value"
  )
```

Customers → purchases

```
purchases <- customers %>%
  pivot_longer(
    cols = item_1:item_3,  # variables item_1 to item_3
    names_to = "item_no",  # column names -> new column called item_no
    values_to = "item"     # values in columns -> new column called item
    )

purchases
```

```
## # A tibble: 6 x 3
##   customer_id item_no item
##         <dbl> <chr>   <chr>
## 1           1 item_1  bread
## 2           1 item_2  milk
## 3           1 item_3  banana
## 4           2 item_1  milk
## 5           2 item_2  toilet paper
## 6           2 item_3  <NA>
```

Why pivot?

Most likely, because the next step of your analysis needs it

Why pivot?

Most likely, because the next step of your analysis needs it

```
prices
```

```
## # A tibble: 5 x 2
##   item         price
##   <chr>        <dbl>
## 1 avocado       0.5
## 2 banana        0.15
## 3 bread         1
## 4 milk          0.8
## 5 toilet paper  3
```

```
purchases %>%
  left_join(prices)
```

```
## # A tibble: 6 x 4
##   customer_id item_no item         price
##         <dbl> <chr>   <chr>        <dbl>
## 1           1 item_1  bread         1
## 2           1 item_2  milk          0.8
## 3           1 item_3  banana        0.15
## 4           2 item_1  milk          0.8
## 5           2 item_2  toilet paper  3
## 6           2 item_3  <NA>         NA
```

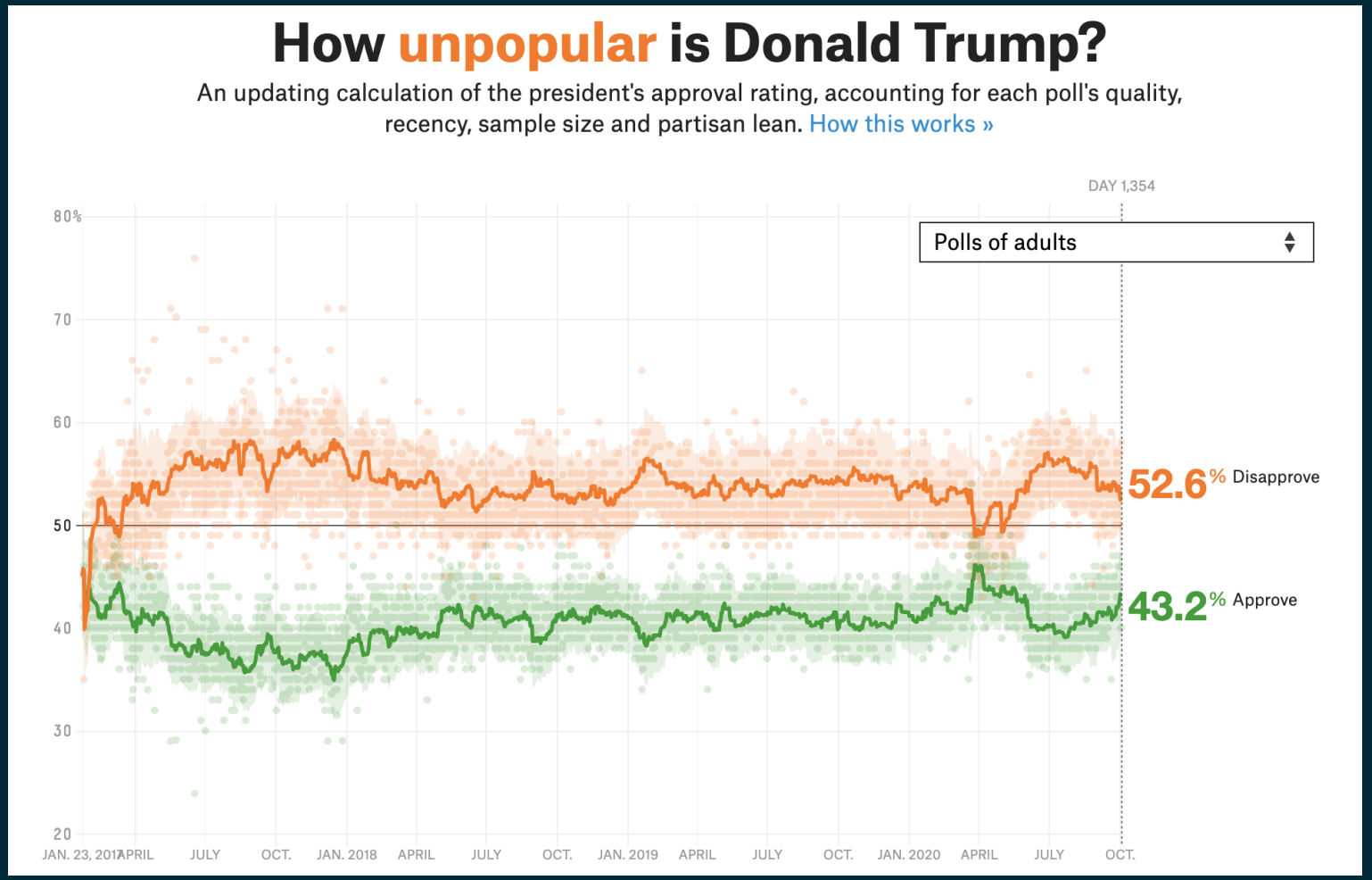
Purchases → customers

- `data` (as usual)
- `names_from`: which column in the long format contains the what should be column names in the wide format
- `values_from`: which column in the long format contains the what should be values in the new columns in the wide format

```
purchases %>%
  pivot_wider(
    names_from = item_no,
    values_from = item
  )
```

```
## # A tibble: 2 x 4
##   customer_id item_1 item_2       item_3
##         <dbl> <chr>  <chr>        <chr>
## 1           1 bread  milk         banana
## 2           2 milk   toilet paper <NA>
```

Case study: Approval rating of Donald Trump



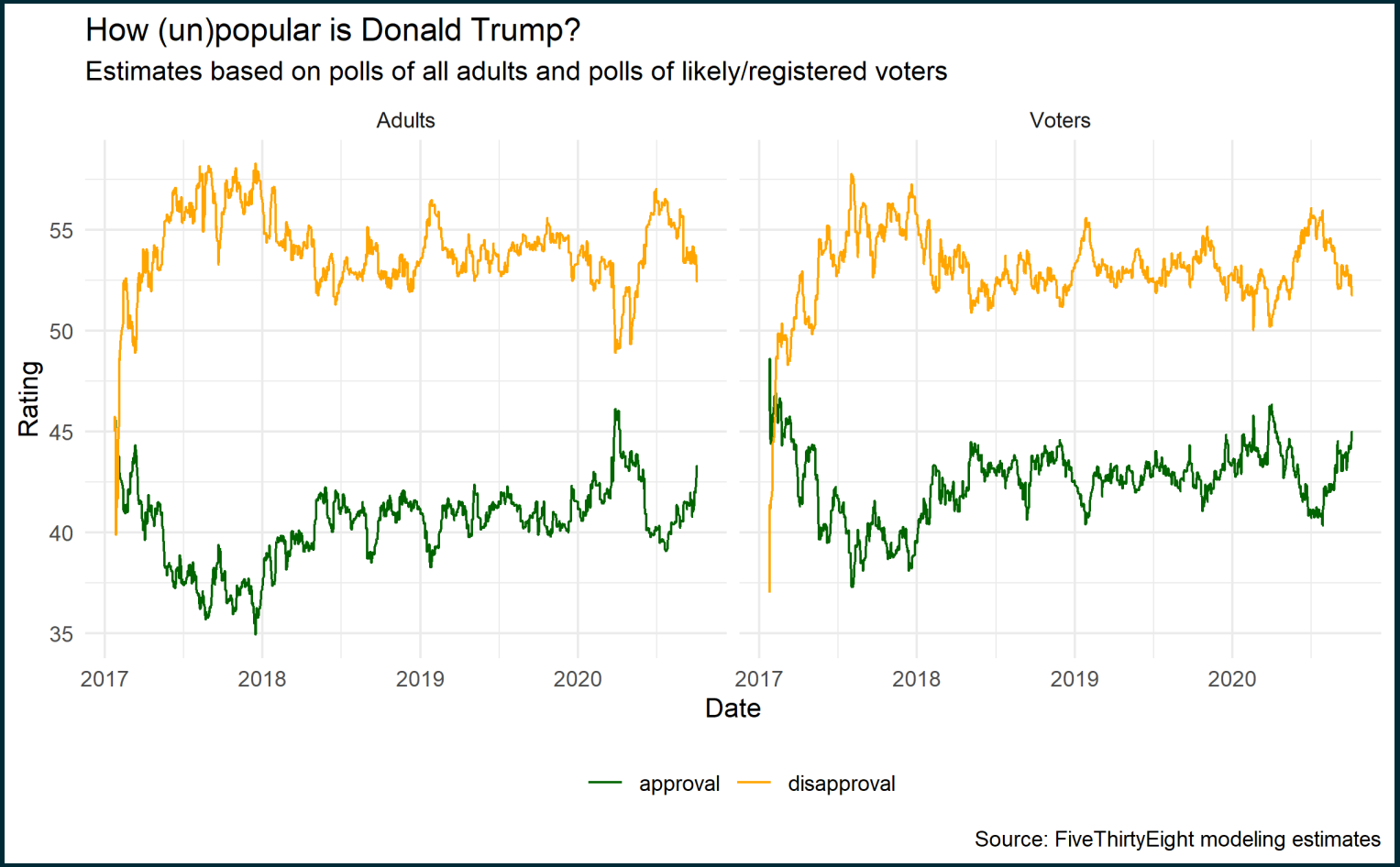
Source: FiveThirtyEight

Data

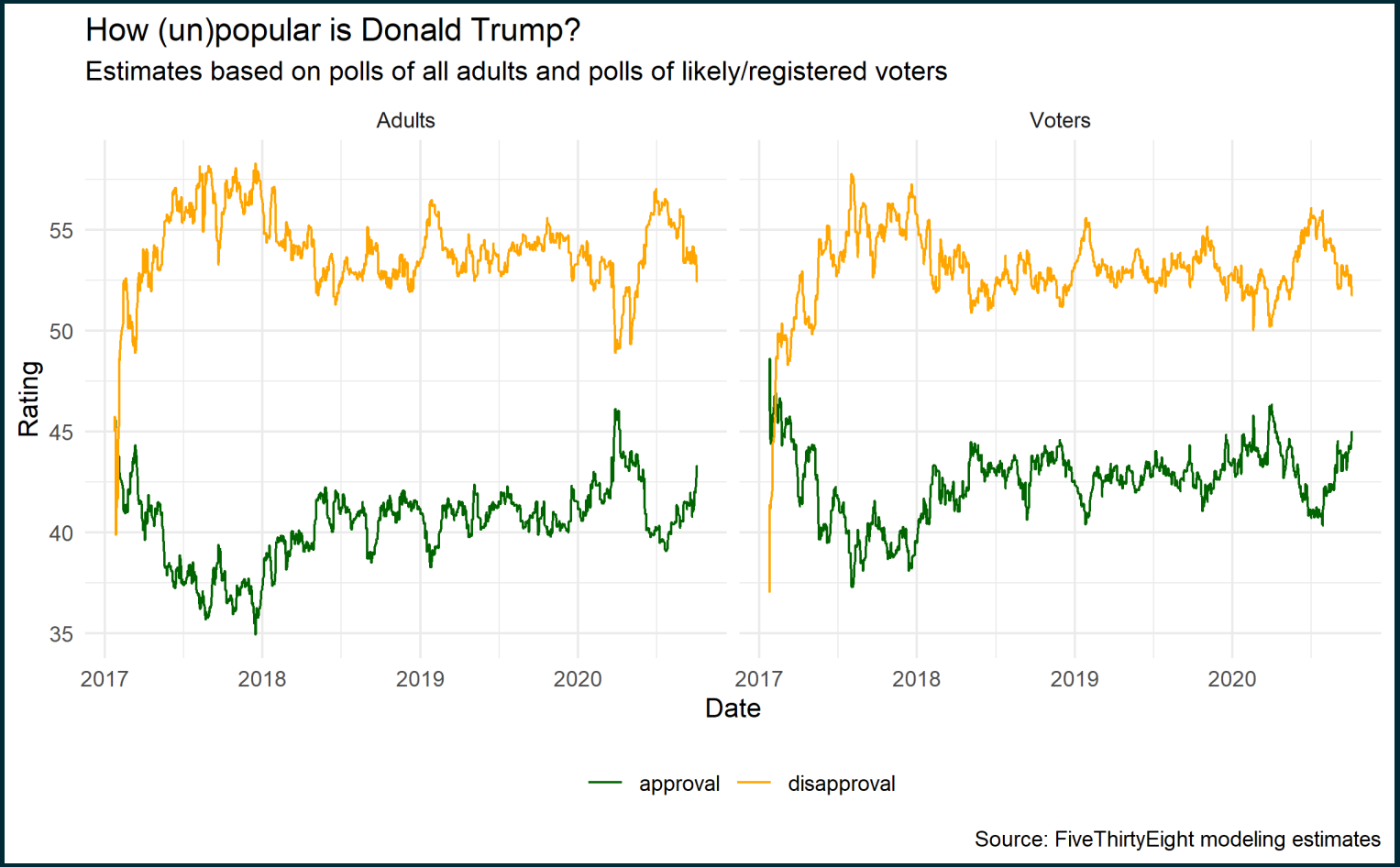
```
trump
```

```
## # A tibble: 2,702 x 4
##    subgroup date       approval disapproval
##    <chr>    <date>        <dbl>       <dbl>
##  1 Voters   2020-10-04     44.7        52.2
##  2 Adults   2020-10-04     43.2        52.6
##  3 Adults   2020-10-03     43.2        52.6
##  4 Voters   2020-10-03     45.0        51.7
##  5 Adults   2020-10-02     43.3        52.4
##  6 Voters   2020-10-02     44.5        52.1
##  7 Voters   2020-10-01     44.1        52.8
##  8 Adults   2020-10-01     42.7        53.3
##  9 Adults   2020-09-30     42.2        53.7
## 10 Voters   2020-09-30     44.2        52.7
## # ... with 2,692 more rows
```

Goal


How (un)popular is Donald Trump?
Estimates based on polls of all adults and polls of likely/registered voters
Adults
Voters
Rating
55
50
45
40
35
2017
2018
2019
2020
Date
approval
disapproval
Source: FiveThirtyEight modeling estimates

Goal



Aesthetic mappings:

- ✓ x = `date`
- ✗ y = `rating_value`
- ✗ color = `rating_type`

Facet:

- ✓ `subgroup` (Adults and Voters)

Pivot

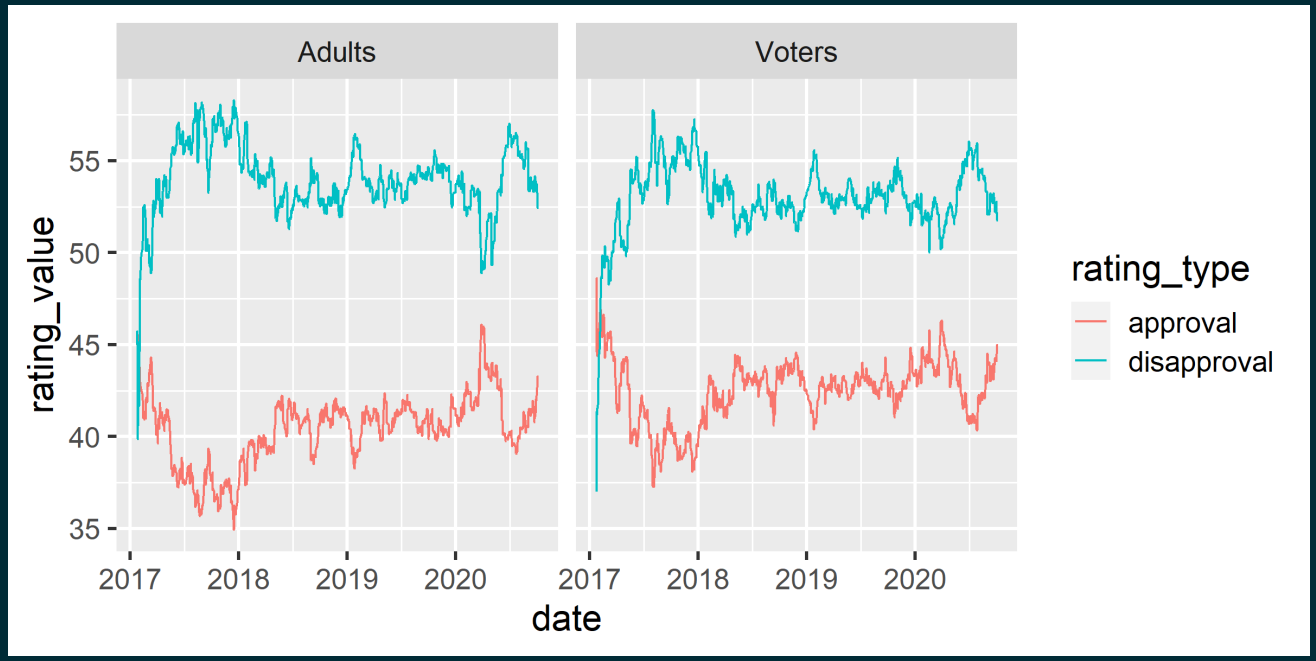
```
trump_longer <- trump %>%
  pivot_longer(
    cols = c(approval, disapproval),
    names_to = "rating_type",
    values_to = "rating_value"
  )

trump_longer
```

```
## # A tibble: 5,404 x 4
##    subgroup date       rating_type rating_value
##    <chr>    <date>     <chr>              <dbl>
##  1 Voters   2020-10-04 approval            44.7
##  2 Voters   2020-10-04 disapproval         52.2
##  3 Adults   2020-10-04 approval            43.2
##  4 Adults   2020-10-04 disapproval         52.6
##  5 Adults   2020-10-03 approval            43.2
##  6 Adults   2020-10-03 disapproval         52.6
##  7 Voters   2020-10-03 approval            45.0
##  8 Voters   2020-10-03 disapproval         51.7
...
```

Plot

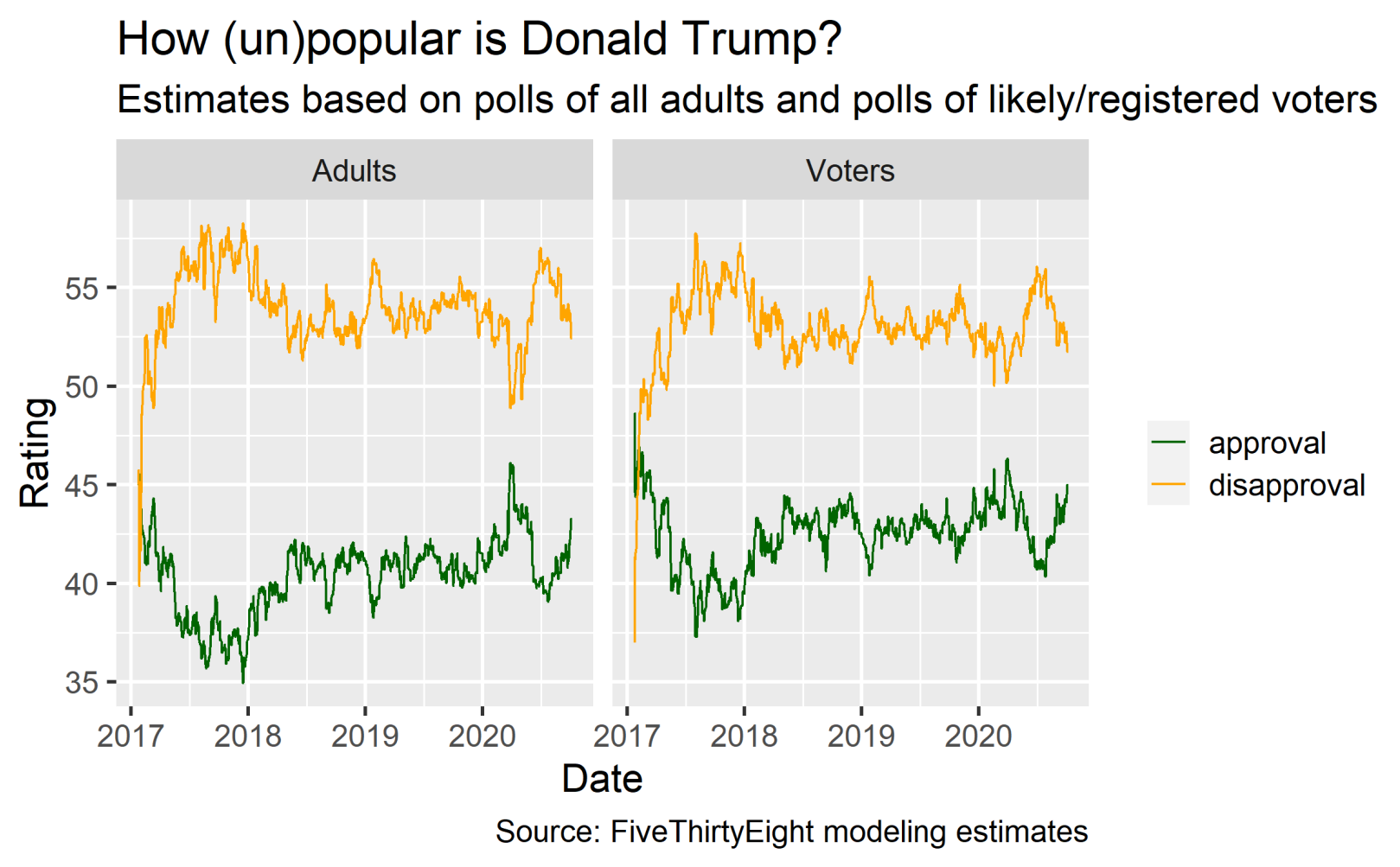
```
ggplot(trump_longer,
       aes(x = date, y = rating_value, color = rating_type, group = rating_type)) +
  geom_line() +
  facet_wrap(~ subgroup)
```



Code Plot

```
ggplot(trump_longer,
       aes(x = date, y = rating_value,
           color = rating_type, group = rating_type)) +
  geom_line() +
  facet_wrap(~ subgroup) +
  scale_color_manual(values = c("darkgreen", "orange")) +
  labs(
    x = "Date", y = "Rating",
    color = NULL,
    title = "How (un)popular is Donald Trump?",
    subtitle = "Estimates based on polls of all adults and polls of likely/registered voters",
    caption = "Source: FiveThirtyEight modeling estimates"
  )
```

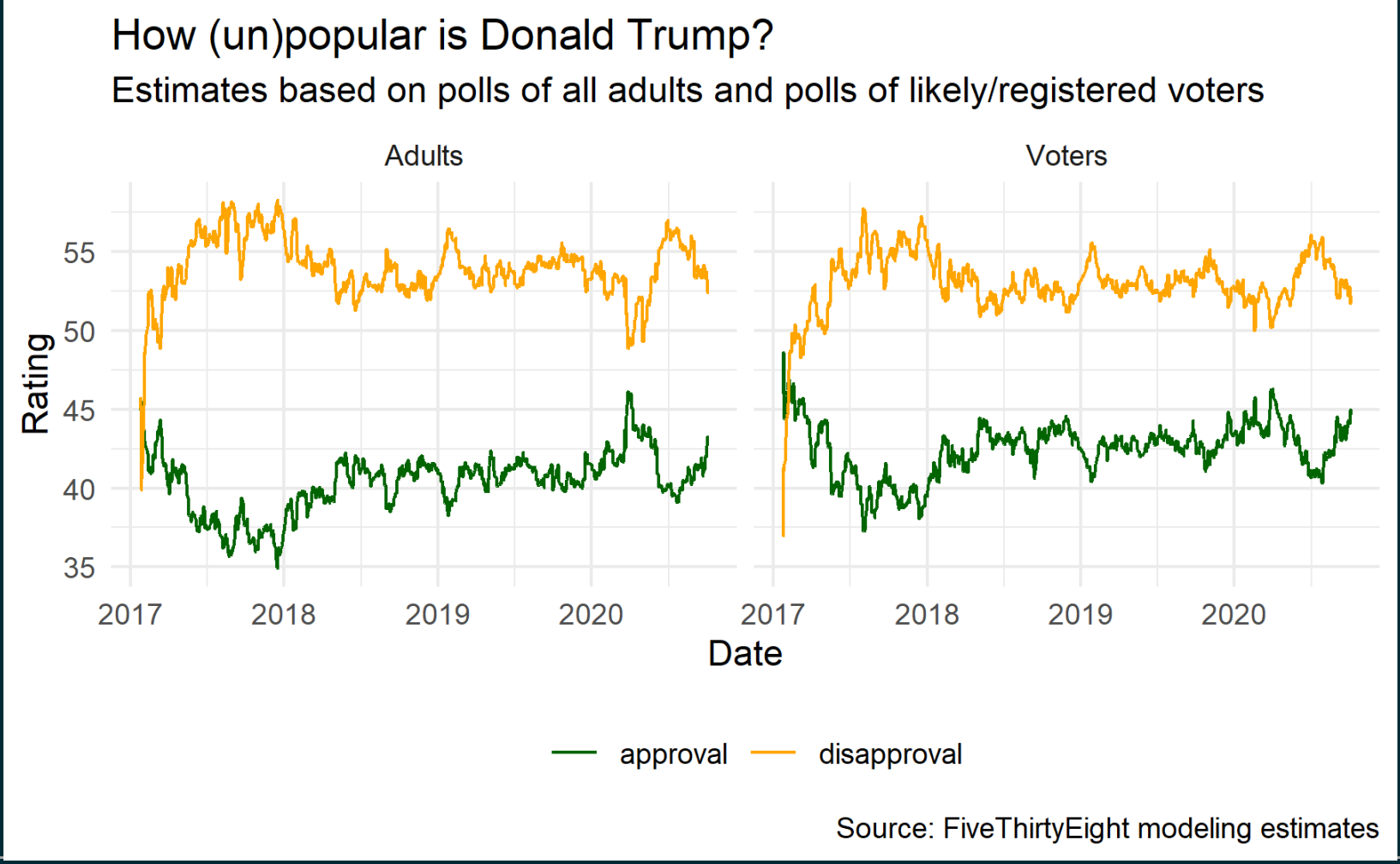
Code Plot


How (un)popular is Donald Trump?
Estimates based on polls of all adults and polls of likely/registered voters
Adults
Voters
Rating
55
50
45
40
35
2017
2018
2019
2020
Date
approval
disapproval
Source: FiveThirtyEight modeling estimates

Code Plot

```
ggplot(trump_longer,
       aes(x = date, y = rating_value,
           color = rating_type, group = rating_type)) +
  geom_line() +
  facet_wrap(~ subgroup) +
  scale_color_manual(values = c("darkgreen", "orange")) +
  labs(
    x = "Date", y = "Rating",
    color = NULL,
    title = "How (un)popular is Donald Trump?",
    subtitle = "Estimates based on polls of all adults and polls of likely/registered voters",
    caption = "Source: FiveThirtyEight modeling estimates"
  ) +
  theme_minimal() +
  theme(legend.position = "bottom")
```

Code Plot


How (un)popular is Donald Trump?
Estimates based on polls of all adults and polls of likely/registered voters
Adults
Voters
Rating
55
50
45
40
35
2017
2018
2019
2020
Date
approval
disapproval
Source: FiveThirtyEight modeling estimates