COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

Data Science for Business Analytics

Lecture 2

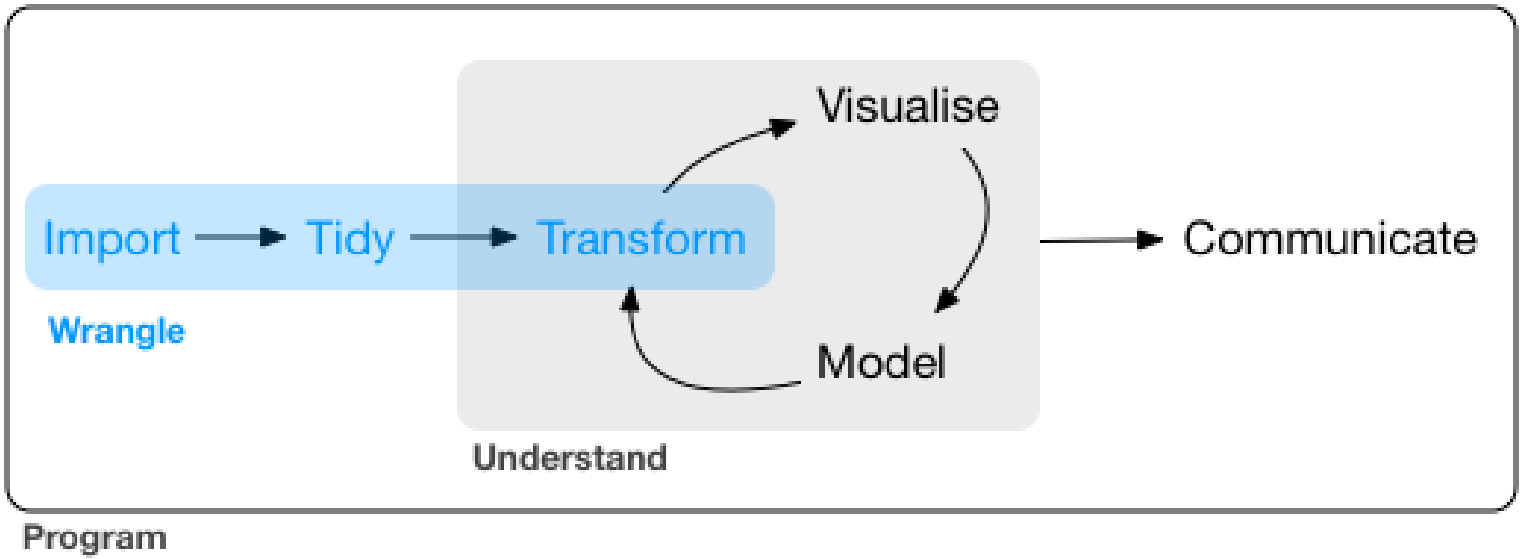
Thibault Vatter

Department of Statistics, Columbia University

02/17/2020

Outline

Today



Most of the material (e.g., the picture above) is borrowed from

R for data science

Data manipulation with `dplyr`

- When working with data you must:
  - Figure out what you want to do.
  - Describe those tasks in the form of a computer program.
  - Execute the program.
- The `dplyr` package makes these steps fast and easy:
  - By constraining your options, it helps you think about your data manipulation challenges.
  - It provides simple "verbs", functions that correspond to the most common data manipulation tasks, to help you translate your thoughts into code.
  - It uses efficient backends, so you spend less time waiting for the computer.

A grammar of data manipulation

- 5 verbs to solve common data manipulation challenges:
  - `filter()` to select observations based on their values.
  - `arrange()` to reorder the observations.
  - `select()` to select variables based on their names.
  - `mutate()` to add variables as functions of existing variables.
  - `summarize()` to collapse many values down to a single summary.
- Two important features:
  - Verbs can be used with `group_by()` to operate groupwise.
  - Verbs work similarly:
    1. First argument is a data frame.
    2. Other arguments describe what to do with it using variable names.
    3. Result is a new data frame.

nycflights13

All 336,776 flights that departed from NYC in 2013 (US BTS):

```
nycflights13::flights
#> # A tibble: 336,776 x 19
#>     year month   day dep_time sched_dep_time dep_delay arr_time
#>    <int> <int> <int>    <int>          <int>     <dbl>    <int>
#>  1  2013     1     1      517            515         2      830
#>  2  2013     1     1      533            529         4      850
#>  3  2013     1     1      542            540         2      923
#>  4  2013     1     1      544            545        -1     1004
#>  5  2013     1     1      554            600        -6      812
#>  6  2013     1     1      554            558        -4      740
#>  7  2013     1     1      555            600        -5      913
#>  8  2013     1     1      557            600        -3      709
#>  9  2013     1     1      557            600        -3      838
#> 10  2013     1     1      558            600        -2      753
#> # ... with 336,766 more rows, and 12 more variables:
#> #   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
#> #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#> #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
#> #   time_hour <dttm>
```

Outline

Filter rows with `filter()`

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

```
filter(flights, month == 1, day == 1)
#> # A tibble: 842 x 19
#>     year month   day dep_time sched_dep_time dep_delay arr_time
#>    <int> <int> <int>    <int>          <int>     <dbl>    <int>
#>  1  2013     1     1      517            515         2      830
#>  2  2013     1     1      533            529         4      850
#>  3  2013     1     1      542            540         2      923
#>  4  2013     1     1      544            545        -1     1004
#>  5  2013     1     1      554            600        -6      812
#>  6  2013     1     1      554            558        -4      740
#>  7  2013     1     1      555            600        -5      913
#>  8  2013     1     1      557            600        -3      709
#>  9  2013     1     1      557            600        -3      838
#> 10  2013     1     1      558            600        -2      753
#> # ... with 832 more rows, and 12 more variables:
#> #   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
#> #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#> #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
#> #   time_hour <dttm>
```

Comparisons

- The standard suite: >, >=, <, <=, !=, and ==.
- Most common mistake:

```
filter(flights, month = 1)
```

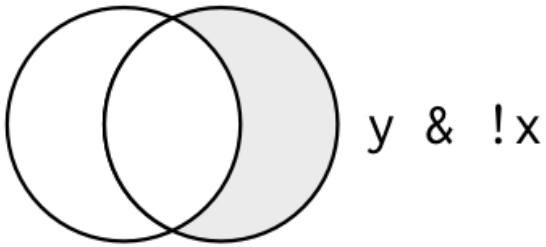
- What happens in the following?

```
sqrt(2) ^ 2 == 2
#> [1] FALSE
1/49 * 49 == 1
#> [1] FALSE
near(sqrt(2) ^ 2,  2)
#> [1] TRUE
near(1 / 49 * 49, 1)
#> [1] TRUE
```

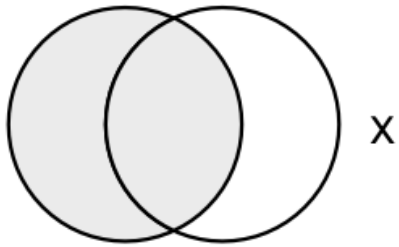
Logical operators

Multiple arguments to `filter()` are combined with:

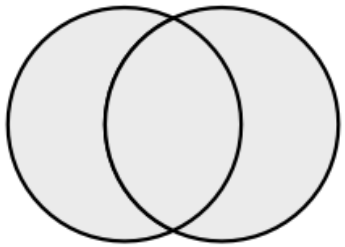
- `&` for "and"
- `|` for "or"
- `!` for "not"



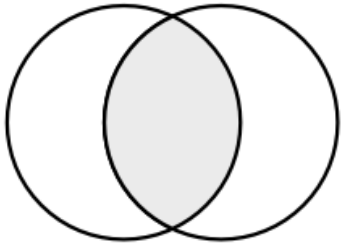
`y & !x`



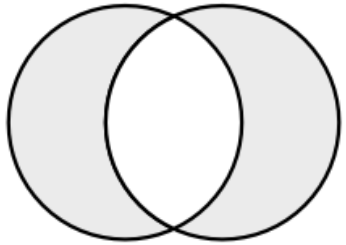
`x`



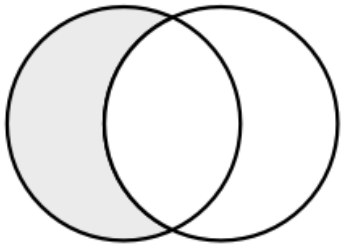
`x | y`



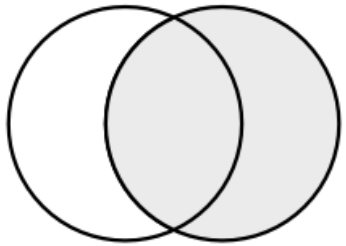
`x & y`



`xor(x, y)`



`x & !y`



`y`

What is this code doing?


```
filter(flights, month == 11 | month == 12)
```

What is this code doing?


```
filter(flights, month == 11 | month == 12)
```

- Literally "finds all flights that departed in November or December".
- `... filter(flights, month == 11 | 12)` ?

What is this code doing?


```
filter(flights, month == 11 | month == 12)
```

- Literally "finds all flights that departed in November or December".
- `... filter(flights, month == 11 | 12)` ?
- No, but a solution:

```
filter(flights, month %in% c(11, 12))
```

De Morgan's law

- `!(x & y)` is the same as `!x | !y`
- `!(x | y)` is the same as `!x & !y`

```
all.equal(
  filter(flights, !(arr_delay > 120 | dep_delay > 120)),
  filter(flights, arr_delay <= 120, dep_delay <= 120)
  )
#> [1] TRUE
```

Missing values and `filter()`


```
df <- tibble(x = c(1, NA, 3))
filter(df, x > 1)
#> # A tibble: 1 x 1
#>       x
#>   <dbl>
#> 1     3
filter(df, is.na(x) | x > 1)
#> # A tibble: 2 x 1
#>       x
#>   <dbl>
#> 1    NA
#> 2     3
```

Outline

Arrange rows with arrange()

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

```
arrange(flights, year, month, day)
#> # A tibble: 336,776 x 19
#>     year month   day dep_time sched_dep_time dep_delay arr_time
#>    <int> <int> <int>    <int>          <int>     <dbl>    <int>
#>  1  2013     1     1      517            515         2      830
#>  2  2013     1     1      533            529         4      850
#>  3  2013     1     1      542            540         2      923
#>  4  2013     1     1      544            545        -1     1004
#>  5  2013     1     1      554            600        -6      812
#>  6  2013     1     1      554            558        -4      740
#>  7  2013     1     1      555            600        -5      913
#>  8  2013     1     1      557            600        -3      709
#>  9  2013     1     1      557            600        -3      838
#> 10  2013     1     1      558            600        -2      753
#> # ... with 336,766 more rows, and 12 more variables:
#> #   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
#> #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#> #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
#> #   time_hour <dttm>
```

arrange() and desc()


```
arrange(flights, desc(arr_delay))
#> # A tibble: 336,776 x 19
#>     year month   day dep_time sched_dep_time dep_delay arr_time
#>    <int> <int> <int>    <int>          <int>     <dbl>    <int>
#>  1  2013     1     9      641            900      1301     1242
#>  2  2013     6    15     1432           1935      1137     1607
#>  3  2013     1    10     1121           1635      1126     1239
#>  4  2013     9    20     1139           1845      1014     1457
#>  5  2013     7    22      845           1600      1005     1044
#>  6  2013     4    10     1100           1900       960     1342
#>  7  2013     3    17     2321            810       911      135
#>  8  2013     7    22     2257            759       898      121
#>  9  2013    12     5      756           1700       896     1058
#> 10  2013     5     3     1133           2055       878     1250
#> # ... with 336,766 more rows, and 12 more variables:
#> #   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
#> #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#> #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
#> #   time_hour <dttm>
```

arrange() and missing values


```
df <- tibble(x = c(5, NA, 2))
arrange(df, x)
#> # A tibble: 3 x 1
#>       x
#>   <dbl>
#> 1     2
#> 2     5
#> 3    NA
arrange(df, desc(x))
#> # A tibble: 3 x 1
#>       x
#>   <dbl>
#> 1     5
#> 2     2
#> 3    NA
```

Outline

Select columns with `select()`

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

```
select(flights, year, month, day)
#> # A tibble: 336,776 x 3
#>     year month   day
#>    <int> <int> <int>
#>  1  2013     1     1
#>  2  2013     1     1
#>  3  2013     1     1
#>  4  2013     1     1
#>  5  2013     1     1
#>  6  2013     1     1
#>  7  2013     1     1
#>  8  2013     1     1
#>  9  2013     1     1
#> 10  2013     1     1
#> # ... with 336,766 more rows
```

All columns between year and day

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

```
select(flights, year:day)
#> # A tibble: 336,776 x 3
#>     year month   day
#>    <int> <int> <int>
#>  1  2013     1     1
#>  2  2013     1     1
#>  3  2013     1     1
#>  4  2013     1     1
#>  5  2013     1     1
#>  6  2013     1     1
#>  7  2013     1     1
#>  8  2013     1     1
#>  9  2013     1     1
#> 10  2013     1     1
#> # ... with 336,766 more rows
```

All columns except from year to day


```
select(flights, -(year:day))
#> # A tibble: 336,776 x 16
#>    dep_time sched_dep_time dep_delay arr_time sched_arr_time
#>       <int>          <int>     <dbl>    <int>          <int>
#>  1      517            515         2      830            819
#>  2      533            529         4      850            830
#>  3      542            540         2      923            850
#>  4      544            545        -1     1004           1022
#>  5      554            600        -6      812            837
#>  6      554            558        -4      740            728
#>  7      555            600        -5      913            854
#>  8      557            600        -3      709            723
#>  9      557            600        -3      838            846
#> 10      558            600        -2      753            745
#> # ... with 336,766 more rows, and 11 more variables:
#> #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
#> #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#> #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

select() and everything()

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

```
select(flights, time_hour, air_time, everything())
#> # A tibble: 336,776 x 19
#>    time_hour           air_time  year month   day dep_time
#>    <dttm>                 <dbl> <int> <int> <int>    <int>
#>  1 2013-01-01 05:00:00      227  2013     1     1      517
#>  2 2013-01-01 05:00:00      227  2013     1     1      533
#>  3 2013-01-01 05:00:00      160  2013     1     1      542
#>  4 2013-01-01 05:00:00      183  2013     1     1      544
#>  5 2013-01-01 06:00:00      116  2013     1     1      554
#>  6 2013-01-01 05:00:00      150  2013     1     1      554
#>  7 2013-01-01 06:00:00      158  2013     1     1      555
#>  8 2013-01-01 06:00:00       53  2013     1     1      557
#>  9 2013-01-01 06:00:00      140  2013     1     1      557
#> 10 2013-01-01 06:00:00      138  2013     1     1      558
#> # ... with 336,766 more rows, and 13 more variables:
#> #   sched_dep_time <int>, dep_delay <dbl>, arr_time <int>,
#> #   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
#> #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#> #   distance <dbl>, hour <dbl>, minute <dbl>
```

More on select

- Helper functions you can use within `select()`:
  - `starts_with("abc")`: matches names that begin with "abc".
  - `ends_with("xyz")`: matches names that end with "xyz".
  - `contains("ijk")`: matches names that contain "ijk".
  - `matches("(.)\\1")`: selects variables that match a regular expression (this one matches any variables that contain repeated characters).
  - `num_range("x", 1:3)` matches `x1`, `x2` and `x3`.
- `select()` can be used to rename variables, but it drops all of the variables not explicitly mentioned. Instead, use `rename()`
- See `?select` for more details.

Outline

Create a narrower dataset

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

```
(flights_sml <- select(flights,
  year:day,
  ends_with("delay"),
  distance,
  air_time))
#> # A tibble: 336,776 x 7
#>     year month   day dep_delay arr_delay distance air_time
#>    <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl>
#>  1  2013     1     1         2        11     1400      227
#>  2  2013     1     1         4        20     1416      227
#>  3  2013     1     1         2        33     1089      160
#>  4  2013     1     1        -1       -18     1576      183
#>  5  2013     1     1        -6       -25      762      116
#>  6  2013     1     1        -4        12      719      150
#>  7  2013     1     1        -5        19     1065      158
#>  8  2013     1     1        -3       -14      229       53
#>  9  2013     1     1        -3        -8      944      140
#> 10  2013     1     1        -2         8      733      138
#> # ... with 336,766 more rows
```

Add new variables with `mutate()`

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

```
mutate(flights_sml,
  gain = arr_delay - dep_delay,
  speed = distance / air_time * 60)
#> # A tibble: 336,776 x 9
#>     year month   day dep_delay arr_delay distance air_time  gain
#>    <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl> <dbl>
#>  1  2013     1     1         2        11     1400      227     9
#>  2  2013     1     1         4        20     1416      227    16
#>  3  2013     1     1         2        33     1089      160    31
#>  4  2013     1     1        -1       -18     1576      183   -17
#>  5  2013     1     1        -6       -25      762      116   -19
#>  6  2013     1     1        -4        12      719      150    16
#>  7  2013     1     1        -5        19     1065      158    24
#>  8  2013     1     1        -3       -14      229       53   -11
#>  9  2013     1     1        -3        -8      944      140    -5
#> 10  2013     1     1        -2         8      733      138    10
#> # ... with 336,766 more rows, and 1 more variable: speed <dbl>
```

Refer to columns just created

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

```
mutate(flights_sml,
  gain = arr_delay - dep_delay,
  hours = air_time / 60,
  gain_per_hour = gain / hours)
#> # A tibble: 336,776 x 10
#>     year month   day dep_delay arr_delay distance air_time  gain
#>    <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl> <dbl>
#>  1  2013     1     1         2        11     1400      227     9
#>  2  2013     1     1         4        20     1416      227    16
#>  3  2013     1     1         2        33     1089      160    31
#>  4  2013     1     1        -1       -18     1576      183   -17
#>  5  2013     1     1        -6       -25      762      116   -19
#>  6  2013     1     1        -4        12      719      150    16
#>  7  2013     1     1        -5        19     1065      158    24
#>  8  2013     1     1        -3       -14      229       53   -11
#>  9  2013     1     1        -3        -8      944      140    -5
#> 10  2013     1     1        -2         8      733      138    10
#> # ... with 336,766 more rows, and 2 more variables: hours <dbl>,
#> #   gain_per_hour <dbl>
```

transmute()


```
transmute(flights,
  gain = arr_delay - dep_delay,
  hours = air_time / 60,
  gain_per_hour = gain / hours)
#> # A tibble: 336,776 x 3
#>     gain hours gain_per_hour
#>    <dbl> <dbl>         <dbl>
#>  1     9 3.78           2.38
#>  2    16 3.78           4.23
#>  3    31 2.67          11.6
#>  4   -17 3.05          -5.57
#>  5   -19 1.93          -9.83
#>  6    16 2.5            6.4
#>  7    24 2.63           9.11
#>  8   -11 0.883        -12.5
#>  9    -5 2.33          -2.14
#> 10    10 2.3            4.35
#> # ... with 336,766 more rows
```

Useful creation functions I

Any vectorized function would work, but frequently useful are:

- Arithmetic operators: `+`, `-`, `*`, `/`, `^`.
  - Vectorized with "recycling rules" (e.g., `air_time / 60`).
  - Useful in conjunction with aggregate functions (e.g., `x / sum(x)` or `y - mean(y)`).
- Modular arithmetic: `%/%` (integer division) and `%%` (remainder), where `x == y * (x %/% y) + (x %% y)`.
  - Allows you to break integers up into pieces (e.g., `hour = dep_time %/% 100` and `minute = dep_time %% 100`)
- Logs: `log()`, `log2()`, `log10()`.
  - Useful for data ranging across multiple orders of magnitude.
  - Convert multiplicative relationships to additive.

Useful creation functions II

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

- Offsets: `lead()` and `lag()`:
  - Refer to lead-/lagging values (e.g., compute running differences `x - lag(x)` or find values change `x != lag(x)`).

```
x <- 1:10
lag(x)
#>  [1] NA  1  2  3  4  5  6  7  8  9
lead(x)
#>  [1]  2  3  4  5  6  7  8  9 10 NA
```

- Cumulative aggregates: `cumsum()`, `cumprod()`, `cummin()`, `cummax()`, `cummean()`.

```
cumsum(x)
#>  [1]  1  3  6 10 15 21 28 36 45 55
cummean(x)
#>  [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5
```

Useful creation functions III

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

- Logical comparisons, <, <=, >, >=, !=
- Ranking functions: `min_rank()`, `row_number()`, `dense_rank()`, `percent_rank()`, `cume_dist()`, `ntile()`

```
y <- c(1, 2, 2, NA, 3, 4)
min_rank(y)
#> [1]  1  2  2 NA  4  5
min_rank(desc(y))
#> [1]  5  3  3 NA  2  1
row_number(y)
#> [1]  1  2  3 NA  4  5
dense_rank(y)
#> [1]  1  2  2 NA  3  4
percent_rank(y)
#> [1] 0.00 0.25 0.25   NA 0.75 1.00
cume_dist(y)
#> [1] 0.2 0.6 0.6  NA 0.8 1.0
```

Outline

Collapse values with `summarize()`

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

```
summarize(flights, delay = mean(dep_delay, na.rm = TRUE))
#> # A tibble: 1 x 1
#>   delay
#>   <dbl>
#> 1  12.6
```

summarize() paired with group_by()

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

```
by_day <- group_by(flights, year, month, day)
summarize(by_day, delay = mean(dep_delay, na.rm = TRUE))
#> # A tibble: 365 x 4
#> # Groups:   year, month [12]
#>     year month   day delay
#>    <int> <int> <int> <dbl>
#>  1  2013     1     1 11.5
#>  2  2013     1     2 13.9
#>  3  2013     1     3 11.0
#>  4  2013     1     4  8.95
#>  5  2013     1     5  5.73
#>  6  2013     1     6  7.15
#>  7  2013     1     7  5.42
#>  8  2013     1     8  2.55
#>  9  2013     1     9  2.28
#> 10  2013     1    10  2.84
#> # ... with 355 more rows
```

Outline

What is this code doing?


```
a1 <- group_by(flights, year, month, day)
a2 <- select(a1, arr_delay, dep_delay)
a3 <- summarize(a2,
                arr = mean(arr_delay, na.rm = TRUE),
                dep = mean(dep_delay, na.rm = TRUE))
filter(a3, arr > 30 | dep > 30)
#> # A tibble: 49 x 5
#> # Groups:   year, month [11]
#>    year month   day   arr   dep
#>   <int> <int> <int> <dbl> <dbl>
#>  1  2013     1    16  34.2  24.6
#>  2  2013     1    31  32.6  28.7
#>  3  2013     2    11  36.3  39.1
#>  4  2013     2    27  31.3  37.8
#>  5  2013     3     8  85.9  83.5
#>  6  2013     3    18  41.3  30.1
#>  7  2013     4    10  38.4  33.0
#>  8  2013     4    12  36.0  34.8
#>  9  2013     4    18  36.0  34.9
#> 10  2013     4    19  47.9  46.1
#> # ... with 39 more rows
```

Same code (no unnecessary objects)

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

```
filter(summarize(select(group_by(flights, year, month, day),
            arr_delay, dep_delay),
    arr = mean(arr_delay, na.rm = TRUE),
    dep = mean(dep_delay, na.rm = TRUE)),
    arr > 30 | dep > 30)
#> # A tibble: 49 x 5
#> # Groups:   year, month [11]
#>     year month   day   arr   dep
#>    <int> <int> <int> <dbl> <dbl>
#>  1  2013     1    16  34.2  24.6
#>  2  2013     1    31  32.6  28.7
#>  3  2013     2    11  36.3  39.1
#>  4  2013     2    27  31.3  37.8
#>  5  2013     3     8  85.9  83.5
#>  6  2013     3    18  41.3  30.1
#>  7  2013     4    10  38.4  33.0
#>  8  2013     4    12  36.0  34.8
#>  9  2013     4    18  36.0  34.9
#> 10  2013     4    19  47.9  46.1
#> # ... with 39 more rows
```

... Or use %>%


```
flights %>%
  group_by(year, month, day) %>%
  select(arr_delay, dep_delay) %>%
  summarize(arr = mean(arr_delay, na.rm = TRUE),
            dep = mean(dep_delay, na.rm = TRUE)) %>%
  filter(arr > 30 | dep > 30)
#> # A tibble: 49 x 5
#> # Groups:   year, month [11]
#>    year month   day   arr   dep
#>   <int> <int> <int> <dbl> <dbl>
#> 1  2013     1    16  34.2  24.6
#> 2  2013     1    31  32.6  28.7
#> 3  2013     2    11  36.3  39.1
#> 4  2013     2    27  31.3  37.8
#> 5  2013     3     8  85.9  83.5
#> 6  2013     3    18  41.3  30.1
#> 7  2013     4    10  38.4  33.0
#> 8  2013     4    12  36.0  34.8
#> 9  2013     4    18  36.0  34.9
#> 10 2013     4    19  47.9  46.1
#> # ... with 39 more rows
```

Basic piping

- `x %>% f` is equivalent to `f(x)`
- `x %>% f(y)` is equivalent to `f(x, y)`
- `x %>% f(y) %>% g(z)` is equivalent to `g(f(x, y), z)`

```
x <- 1:10
y <- x + 1
z <- y + 1
f <- function(x, y) x + y

x %>% sum
#> [1] 55
x %>% f(y)
#>  [1]  3  5  7  9 11 13 15 17 19 21
x %>% f(y) %>% f(z)
#>  [1]  6  9 12 15 18 21 24 27 30 33
```

The argument ("dot") placeholder

- `x %>% f(y, .)` is equivalent to `f(y, x)`
- `x %>% f(y, z = .)` is equivalent to `f(y, z = x)`

```
x <- 1:10
y <- 2 * x
f <- function(z, y) y / z

x %>% f(y, .)
#>  [1] 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
x %>% f(y, z = .)
#>  [1] 2 2 2 2 2 2 2 2 2 2
```

Function composition

- Each of the three options has its own strengths and weaknesses:
  - Nesting, `f(g(x))`:
    - Concise, and well suited for short sequences.
    - Longer sequences harder to read (inside out & right to left).
    - Arguments can get spread out over long distances (see Dagwood sandwich).
  - Intermediate objects, `y <- f(x); g(y)`:
    - Requires you to name intermediate objects.
    - A strength when objects are important, but a weakness when values are truly intermediate.
  - Piping, `x %>% f() %>% g()`:
    - Allows to read code in straightforward left-to-right fashion.
    - Doesn't require to name intermediate objects.
    - Only for linear sequences of transformations of a single object.
- Most code use a combination of all three styles, but. . .
- **Piping is more common in data analysis code!**

Outline

What is happening here?


```
flights %>%
  group_by(year, month, day) %>%
  summarize(mean = mean(dep_delay))
#> # A tibble: 365 x 4
#> # Groups:   year, month [12]
#>     year month   day  mean
#>    <int> <int> <int> <dbl>
#>  1  2013     1     1    NA
#>  2  2013     1     2    NA
#>  3  2013     1     3    NA
#>  4  2013     1     4    NA
#>  5  2013     1     5    NA
#>  6  2013     1     6    NA
#>  7  2013     1     7    NA
#>  8  2013     1     8    NA
#>  9  2013     1     9    NA
#> 10  2013     1    10    NA
#> # ... with 355 more rows
```

Use na.rm = TRUE

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

```
flights %>%
  group_by(year, month, day) %>%
  summarize(mean = mean(dep_delay, na.rm = TRUE))
#> # A tibble: 365 x 4
#> # Groups:   year, month [12]
#>     year month   day  mean
#>    <int> <int> <int> <dbl>
#>  1  2013     1     1 11.5
#>  2  2013     1     2 13.9
#>  3  2013     1     3 11.0
#>  4  2013     1     4  8.95
#>  5  2013     1     5  5.73
#>  6  2013     1     6  7.15
#>  7  2013     1     7  5.42
#>  8  2013     1     8  2.55
#>  9  2013     1     9  2.28
#> 10  2013     1    10  2.84
#> # ... with 355 more rows
```

Or pre-filter the dataset


```
not_cancelled <- flights %>%
  filter(!is.na(dep_delay), !is.na(arr_delay))
not_cancelled %>%
  group_by(year, month, day) %>%
  summarize(mean = mean(dep_delay))
#> # A tibble: 365 x 4
#> # Groups:   year, month [12]
#>     year month   day  mean
#>    <int> <int> <int> <dbl>
#>  1  2013     1     1 11.4
#>  2  2013     1     2 13.7
#>  3  2013     1     3 10.9
#>  4  2013     1     4  8.97
#>  5  2013     1     5  5.73
#>  6  2013     1     6  7.15
#>  7  2013     1     7  5.42
#>  8  2013     1     8  2.56
#>  9  2013     1     9  2.30
#> 10  2013     1    10  2.84
#> # ... with 355 more rows
```

Useful summary functions I

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

- Measures of location: `mean()`, `median()`.
- Measures of spread: `sd()`, `IQR()`, `mad()`.
- Measures of rank: `min(x)`, `quantile(x, 0.25)`, `max(x)`.

```
not_cancelled %>%
  group_by(year, month, day) %>%
  summarize(first = min(dep_time), last = max(dep_time))
#> # A tibble: 365 x 5
#> # Groups:   year, month [12]
#>     year month   day first  last
#>    <int> <int> <int> <int> <int>
#>  1  2013     1     1   517  2356
#>  2  2013     1     2    42  2354
#>  3  2013     1     3    32  2349
#>  4  2013     1     4    25  2358
#>  5  2013     1     5    14  2357
#>  6  2013     1     6    16  2355
#>  7  2013     1     7    49  2359
#>  8  2013     1     8   454  2351
#>  9  2013     1     9     2  2252
#> 10  2013     1    10     3  2320
#> # ... with 355 more rows
```

Useful summary functions II

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

- Measures of position: `first(x)`, `nth(x, 2)`, `last(x)`.

```
not_cancelled %>%
  group_by(year, month, day) %>%
  summarize(first_dep = first(dep_time), last_dep = last(dep_time))
#> # A tibble: 365 x 5
#> # Groups:   year, month [12]
#>     year month   day first_dep last_dep
#>    <int> <int> <int>     <int>    <int>
#>  1  2013     1     1       517     2356
#>  2  2013     1     2        42     2354
#>  3  2013     1     3        32     2349
#>  4  2013     1     4        25     2358
#>  5  2013     1     5        14     2357
#>  6  2013     1     6        16     2355
#>  7  2013     1     7        49     2359
#>  8  2013     1     8       454     2351
#>  9  2013     1     9         2     2252
#> 10  2013     1    10         3     2320
#> # ... with 355 more rows
```

Useful summary functions III

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

- Counts: `n(x)`, `sum(!is.na(x))`, `n_distinct(x)`.

```
not_cancelled %>%
  group_by(dest) %>%
  summarize(carriers = n_distinct(carrier)) %>%
  arrange(desc(carriers))
#> # A tibble: 104 x 2
#>    dest  carriers
#>    <chr>    <int>
#>  1 ATL          7
#>  2 BOS          7
#>  3 CLT          7
#>  4 ORD          7
#>  5 TPA          7
#>  6 AUS          6
#>  7 DCA          6
#>  8 DTW          6
#>  9 IAD          6
#> 10 MSP          6
#> # ... with 94 more rows
```

Useful summary functions IV

- A simple helper function for counts:

```
not_cancelled %>% count(dest)
#> # A tibble: 104 x 2
#>    dest      n
#>    <chr> <int>
#>  1 ABQ     254
#>  2 ACK     264
#>  3 ALB     418
#>  4 ANC       8
#>  5 ATL   16837
#>  6 AUS    2411
#>  7 AVL     261
#>  8 BDL     412
#>  9 BGR     358
#> 10 BHM     269
#> # ... with 94 more rows
```

Useful summary functions V

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

- Counts with an optional weight variable:

```
not_cancelled %>% count(tailnum, wt = distance)
#> # A tibble: 4,037 x 2
#>    tailnum      n
#>    <chr>    <dbl>
#>  1 D942DN    3418
#>  2 N0EGMQ  239143
#>  3 N10156  109664
#>  4 N102UW   25722
#>  5 N103US   24619
#>  6 N104UW   24616
#>  7 N10575  139903
#>  8 N105UW   23618
#>  9 N107US   21677
#> 10 N108UW   32070
#> # ... with 4,027 more rows
```

Useful summary functions VI

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

- Counts of logical values: e.g., `sum(x > 10)`.

```
not_cancelled %>%
  group_by(year, month, day) %>%
  summarize(n_early = sum(dep_time < 500))
#> # A tibble: 365 x 4
#> # Groups:   year, month [12]
#>     year month   day n_early
#>    <int> <int> <int>   <int>
#>  1  2013     1     1       0
#>  2  2013     1     2       3
#>  3  2013     1     3       4
#>  4  2013     1     4       3
#>  5  2013     1     5       3
#>  6  2013     1     6       2
#>  7  2013     1     7       2
#>  8  2013     1     8       1
#>  9  2013     1     9       3
#> 10  2013     1    10       3
#> # ... with 355 more rows
```

Useful summary functions VII

- Proportions of logical values: e.g., `mean(y == 0)`.

```
not_cancelled %>%
  group_by(year, month, day) %>%
  summarize(hour_perc = mean(arr_delay > 60))
#> # A tibble: 365 x 4
#> # Groups:   year, month [12]
#>     year month   day hour_perc
#>    <int> <int> <int>     <dbl>
#>  1  2013     1     1    0.0722
#>  2  2013     1     2    0.0851
#>  3  2013     1     3    0.0567
#>  4  2013     1     4    0.0396
#>  5  2013     1     5    0.0349
#>  6  2013     1     6    0.0470
#>  7  2013     1     7    0.0333
#>  8  2013     1     8    0.0213
#>  9  2013     1     9    0.0202
#> 10  2013     1    10    0.0183
#> # ... with 355 more rows
```

Grouping by multiple variables I

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

```
daily <- group_by(flights, year, month, day)
(per_day   <- summarize(daily, flights = n()))
#> # A tibble: 365 x 4
#> # Groups:   year, month [12]
#>     year month   day flights
#>    <int> <int> <int>   <int>
#>  1  2013     1     1     842
#>  2  2013     1     2     943
#>  3  2013     1     3     914
#>  4  2013     1     4     915
#>  5  2013     1     5     720
#>  6  2013     1     6     832
#>  7  2013     1     7     933
#>  8  2013     1     8     899
#>  9  2013     1     9     902
#> 10  2013     1    10     932
#> # ... with 355 more rows
```

Grouping by multiple variables II

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

```
(per_month <- summarize(per_day, flights = sum(flights)))
(per_year  <- summarize(per_month, flights = sum(flights)))
#> # A tibble: 12 x 3
#> # Groups:   year [1]
#>     year month flights
#>    <int> <int>   <int>
#>  1  2013     1   27004
#>  2  2013     2   24951
#>  3  2013     3   28834
#>  4  2013     4   28330
#>  5  2013     5   28796
#>  6  2013     6   28243
#>  7  2013     7   29425
#>  8  2013     8   29327
#>  9  2013     9   27574
#> 10  2013    10   28889
#> 11  2013    11   27268
#> 12  2013    12   28135
#> # A tibble: 1 x 2
#>    year flights
#>   <int>   <int>
#> 1  2013  336776
```

Ungrouping


```
daily %>%
  ungroup() %>%               # no longer grouped by date
  summarize(flights = n())   # all flights
#> # A tibble: 1 x 1
#>   flights
#>     <int>
#> 1  336776
```

Grouped filters


```
(popular_dests <- flights %>%
    group_by(dest) %>%
    filter(n() > 365))
#> # A tibble: 332,577 x 19
#> # Groups:   dest [77]
#>     year month   day dep_time sched_dep_time dep_delay arr_time
#>    <int> <int> <int>    <int>          <int>     <dbl>    <int>
#>  1  2013     1     1      517            515         2      830
#>  2  2013     1     1      533            529         4      850
#>  3  2013     1     1      542            540         2      923
#>  4  2013     1     1      544            545        -1     1004
#>  5  2013     1     1      554            600        -6      812
#>  6  2013     1     1      554            558        -4      740
#>  7  2013     1     1      555            600        -5      913
#>  8  2013     1     1      557            600        -3      709
#>  9  2013     1     1      557            600        -3      838
#> 10  2013     1     1      558            600        -2      753
#> # ... with 332,567 more rows, and 12 more variables:
#> #   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
#> #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#> #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
#> #   time_hour <dttm>
```

Grouped mutates

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

```
popular_dests %>%
  filter(arr_delay > 0) %>%
  mutate(prop_delay = arr_delay / sum(arr_delay)) %>%
  select(year:day, dest, arr_delay, prop_delay)
#> # A tibble: 131,106 x 6
#> # Groups:   dest [77]
#>     year month   day dest  arr_delay prop_delay
#>    <int> <int> <int> <chr>     <dbl>      <dbl>
#>  1  2013     1     1 IAH          11  0.000111
#>  2  2013     1     1 IAH          20  0.000201
#>  3  2013     1     1 MIA          33  0.000235
#>  4  2013     1     1 ORD          12  0.0000424
#>  5  2013     1     1 FLL          19  0.0000938
#>  6  2013     1     1 ORD           8  0.0000283
#>  7  2013     1     1 LAX           7  0.0000344
#>  8  2013     1     1 DFW          31  0.000282
#>  9  2013     1     1 ATL          12  0.0000400
#> 10  2013     1     1 DTW          16  0.000116
#> # ... with 131,096 more rows
```

Outline

Tidy data

"Happy families are all alike; every unhappy family is unhappy in its own way." —- *Leo Tolstoy*

"Tidy datasets are all alike, but every messy dataset is messy in its own way." —- *Hadley Wickham*

To learn more about the underlying theory, see the Tidy Data paper.

Which representation is "best"?

- First representation?

```
table1
#> # A tibble: 6 x 4
#>   country      year  cases population
#>   <chr>       <int>  <int>      <int>
#> 1 Afghanistan  1999    745   19987071
#> 2 Afghanistan  2000   2666   20595360
#> 3 Brazil       1999  37737  172006362
#> 4 Brazil       2000  80488  174504898
#> 5 China        1999 212258 1272915272
#> 6 China        2000 213766 1280428583
```

- Second representation?

```
table2
#> # A tibble: 12 x 4
#>    country      year type            count
#>    <chr>       <int> <chr>           <int>
#>  1 Afghanistan  1999 cases             745
#>  2 Afghanistan  1999 population   19987071
#>  3 Afghanistan  2000 cases            2666
#>  4 Afghanistan  2000 population   20595360
#>  5 Brazil       1999 cases           37737
#>  6 Brazil       1999 population  172006362
#>  7 Brazil       2000 cases           80488
#>  8 Brazil       2000 population  174504898
#>  9 China        1999 cases          212258
#> 10 China        1999 population 1272915272
#> 11 China        2000 cases          213766
#> 12 China        2000 population 1280428583
```

- Third representation?

```
table3
#> # A tibble: 6 x 3
#>   country      year rate
#> * <chr>       <int> <chr>
#> 1 Afghanistan  1999 745/19987071
#> 2 Afghanistan  2000 2666/20595360
#> 3 Brazil       1999 37737/172006362
#> 4 Brazil       2000 80488/174504898
#> 5 China        1999 212258/1272915272
#> 6 China        2000 213766/1280428583
```

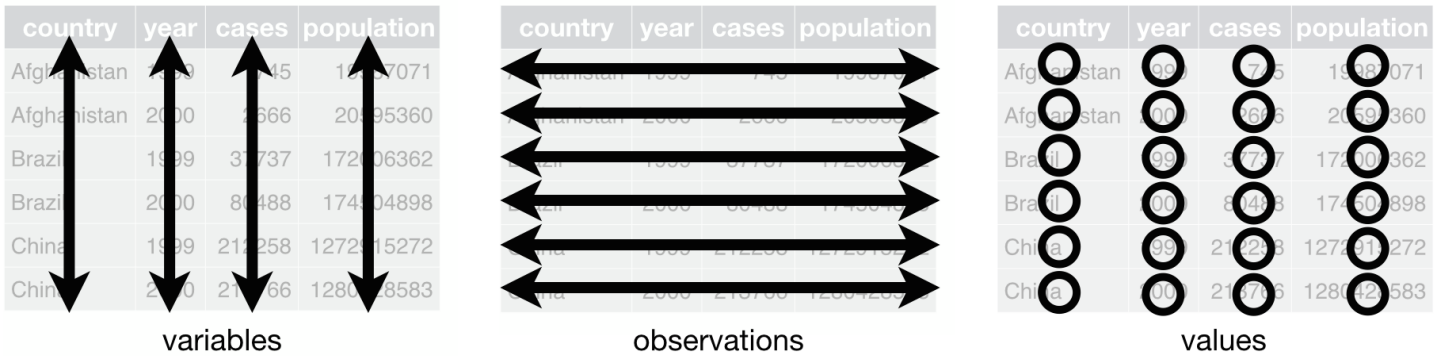
- Fourth representation?

```
table4a  # cases
#> # A tibble: 3 x 3
#>   country     `1999` `2000`
#> * <chr>        <int>  <int>
#> 1 Afghanistan    745   2666
#> 2 Brazil       37737  80488
#> 3 China       212258 213766
table4b  # population
#> # A tibble: 3 x 3
#>   country         `1999`     `2000`
#> * <chr>            <int>      <int>
#> 1 Afghanistan   19987071   20595360
#> 2 Brazil       172006362  174504898
#> 3 China       1272915272 1280428583
```

What makes a dataset tidy?

Three interrelated rules:

- Each variable must have its own column.
- Each observation must have its own row.
- Each value must have its own cell.



Because it's impossible to only satisfy two of the three:

- Put each dataset in a tibble.
- Put each variable in a column.

Why ensure that your data is tidy?

- Why?
  - With consistent data structure, it's easier to learn the tools that work with it because they have an underlying uniformity.
  - Placing variables in columns allows R's vectorized nature to shine.
- Tidy data principles seem obvious, BUT:
  - Most people aren't familiar with them.
  - Data often organized to facilitate something different than analysis.
- Hence, you'll most likely need to do some tidying.

The two steps of tidying

- Figure out what the variables and observations are.
- Resolve one of two common problems:
  - One variable might be spread across multiple columns.
  - One observation might be scattered across multiple rows.

... To fix these problems, you'll need `pivot_longer()` and `pivot_wider()`.

Longer with `pivot_wider()`

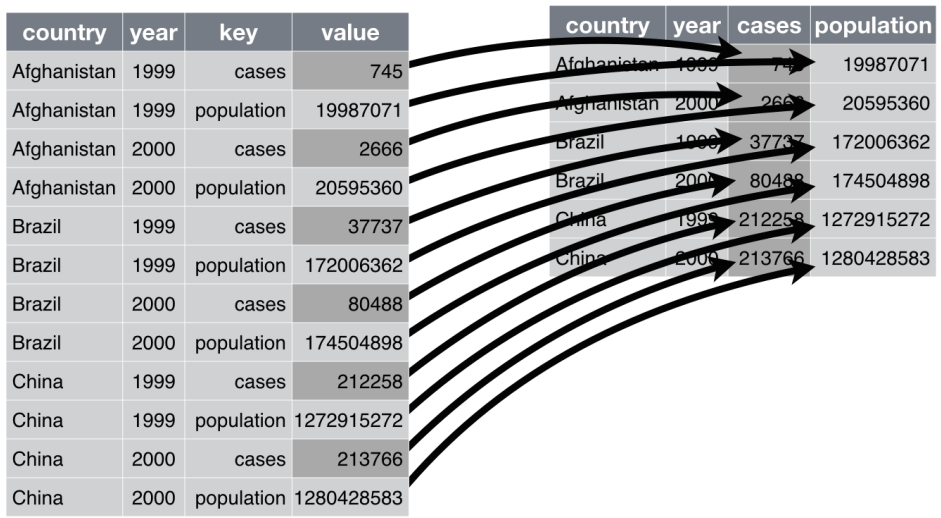

```
table4a
#> # A tibble: 3 x 3
#>   country     `1999` `2000`
#> * <chr>        <int>  <int>
#> 1 Afghanistan    745   2666
#> 2 Brazil       37737  80488
#> 3 China       212258 213766
```

```
table4a %>%
  pivot_longer(c(`1999`, `2000`),
               names_to = "year",
               values_to = "cases")
#> # A tibble: 6 x 3
#>   country     year   cases
#>   <chr>       <chr>  <int>
#> 1 Afghanistan 1999     745
#> 2 Afghanistan 2000    2666
#> 3 Brazil      1999   37737
#> 4 Brazil      2000   80488
#> 5 China       1999  212258
#> 6 China       2000  213766
```

Wider with `pivot_wider()`


```
table2
#> # A tibble: 12 x 4
#>    country      year type            count
#>    <chr>       <int> <chr>           <int>
#>  1 Afghanistan  1999 cases             745
#>  2 Afghanistan  1999 population    19987071
#>  3 Afghanistan  2000 cases            2666
#>  4 Afghanistan  2000 population    20595360
#>  5 Brazil       1999 cases            37737
#>  6 Brazil       1999 population   172006362
#>  7 Brazil       2000 cases            80488
#>  8 Brazil       2000 population   174504898
#>  9 China        1999 cases           212258
#> 10 China        1999 population  1272915272
#> 11 China        2000 cases           213766
#> 12 China        2000 population  1280428583
```

```
table2 %>%
    pivot_wider(names_from = type,
                values_from = count)
#> # A tibble: 6 x 4
#>   country      year  cases population
#>   <chr>       <int>  <int>      <int>
#> 1 Afghanistan  1999    745   19987071
#> 2 Afghanistan  2000   2666   20595360
#> 3 Brazil       1999  37737  172006362
#> 4 Brazil       2000  80488  174504898
#> 5 China        1999 212258 1272915272
#> 6 China        2000 213766 1280428583
```



country	year	key	value
Afghanistan	1999	cases	745
Afghanistan	1999	population	19987071
Afghanistan	2000	cases	2666
Afghanistan	2000	population	20595360
Brazil	1999	cases	37737
Brazil	1999	population	172006362
Brazil	2000	cases	80488
Brazil	2000	population	174504898
China	1999	cases	212258
China	1999	population	1272915272
China	2000	cases	213766
China	2000	population	1280428583

table2

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

Separate a column with `separate()`

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

```
table3
#> # A tibble: 6 x 3
#>   country      year rate
#> * <chr>       <int> <chr>
#> 1 Afghanistan  1999 745/19987071
#> 2 Afghanistan  2000 2666/20595360
#> 3 Brazil       1999 37737/172006362
#> 4 Brazil       2000 80488/174504898
#> 5 China        1999 212258/1272915272
#> 6 China        2000 213766/1280428583
```

```
table3 %>% separate(rate,
                    into = c("cases",
                             "population"))
#> # A tibble: 6 x 4
#>   country      year cases  population
#>   <chr>       <int> <chr>  <chr>
#> 1 Afghanistan  1999 745    19987071
#> 2 Afghanistan  2000 2666   20595360
#> 3 Brazil       1999 37737  172006362
#> 4 Brazil       2000 80488  174504898
#> 5 China        1999 212258 1272915272
#> 6 China        2000 213766 1280428583
```

country	year	rate
Afghanistan	1999	**745** / 19987071
Afghanistan	2000	**2666** / 20595360
Brazil	1999	**37737** / 172006362
Brazil	2000	**80488** / 174504898
China	1999	**212258** / 1272915272
China	2000	**213766** / 1280428583

table3

country	year	cases	population
Afghanistan	1999	**745**	19987071
Afghanistan	2000	**2666**	20595360
Brazil	1999	**37737**	172006362
Brazil	2000	**80488**	174504898
China	1999	**212258**	1272915272
China	2000	**213766**	1280428583

separate() using convert = TRUE


```
table3 %>%
  separate(rate, into = c("cases", "population"), convert = TRUE)
#> # A tibble: 6 x 4
#>   country      year  cases population
#>   <chr>       <int>  <int>      <int>
#> 1 Afghanistan  1999    745   19987071
#> 2 Afghanistan  2000   2666   20595360
#> 3 Brazil       1999  37737  172006362
#> 4 Brazil       2000  80488  174504898
#> 5 China        1999 212258 1272915272
#> 6 China        2000 213766 1280428583
```

Unite two columns with `unite()`


```
table5
#> # A tibble: 6 x 4
#>   country     century year  rate
#> * <chr>       <chr>   <chr> <chr>
#> 1 Afghanistan 19      99    745/19987071
#> 2 Afghanistan 20      00    2666/20595360
#> 3 Brazil      19      99    37737/172006362
#> 4 Brazil      20      00    80488/174504898
#> 5 China       19      99    212258/1272915272
#> 6 China       20      00    213766/1280428583
```

```
table5 %>%
  unite(new, century, year, sep = "")
#> # A tibble: 6 x 3
#>   country     new   rate
#>   <chr>       <chr> <chr>
#> 1 Afghanistan 1999  745/19987071
#> 2 Afghanistan 2000  2666/20595360
#> 3 Brazil      1999  37737/172006362
#> 4 Brazil      2000  80488/174504898
#> 5 China       1999  212258/1272915272
#> 6 China       2000  213766/1280428583
```

country	year	rate
Afghanistan	1999	745 / 19987071
Afghanistan	2000	2666 / 20595360
Brazil	1999	37737 / 172006362
Brazil	2000	80488 / 174504898
China	1999	212258 / 1272915272
China	2000	213766 / 1280428583

country	century	year	rate
Afghanistan	19	99	745 / 19987071
Afghanistan	20	0	2666 / 20595360
Brazil	19	99	37737 / 172006362
Brazil	20	0	80488 / 174504898
China	19	99	212258 / 1272915272
China	20	0	213766 / 1280428583

table6

Missing values and tidy data

- A value can be missing in one of two possible ways:
  - **Explicitly**, i.e. flagged with `NA`.
  - **Implicitly**, i.e. simply not present in the data.

  "An explicit missing value is the presence of an absence; an implicit missing value is the absence of a presence." Hadley Wickham

- Are there missing values in this dataset?

```
stocks <- tibble(
  year   = c(2015, 2015, 2015, 2015, 2016, 2016, 2016),
  qtr    = c(   1,    2,    3,    4,    2,    3,    4),
  return = c(1.88, 0.59, 0.35,   NA, 0.92, 0.17, 2.66)
)
```

Implicit to explicit and conversely

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

- Implicit to explicit:

```
stocks %>%
  pivot_wider(names_from = year,
              values_from = return)
#> # A tibble: 4 x 3
#>     qtr `2015` `2016`
#>   <dbl>  <dbl>  <dbl>
#> 1     1   1.88  NA
#> 2     2   0.59   0.92
#> 3     3   0.35   0.17
#> 4     4  NA      2.66
```

- Explicit to implicit:

```
stocks %>%
  pivot_wider(names_from = year,
              values_from = return) %>%
  pivot_longer(
    cols = c(`2015`, `2016`),
    names_to = "year",
    values_to = "return",
    values_drop_na = TRUE
  )
#> # A tibble: 6 x 3
#>     qtr year  return
#>   <dbl> <chr>  <dbl>
#> 1     1 2015    1.88
#> 2     2 2015    0.59
#> 3     2 2016    0.92
#> 4     3 2015    0.35
#> 5     3 2016    0.17
#> 6     4 2016    2.66
```

Implicit to explicit with `complete()`

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

```
stocks %>% complete(year, qtr)
#> # A tibble: 8 x 3
#>    year   qtr return
#>   <dbl> <dbl>  <dbl>
#> 1  2015     1   1.88
#> 2  2015     2   0.59
#> 3  2015     3   0.35
#> 4  2015     4  NA
#> 5  2016     1  NA
#> 6  2016     2   0.92
#> 7  2016     3   0.17
#> 8  2016     4   2.66
```

Fill in missing values with `fill()`

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

```
treatment <- tribble(
  ~ person,           ~ treatment, ~response,
  "Derrick Whitmore", 1,           7,
  NA,                 2,           10,
  NA,                 3,           9,
  "Katherine Burke",  1,           4
)
treatment %>%
  fill(person)
#> # A tibble: 4 x 3
#>   person           treatment response
#>   <chr>                <dbl>    <dbl>
#> 1 Derrick Whitmore         1        7
#> 2 Derrick Whitmore         2       10
#> 3 Derrick Whitmore         3        9
#> 4 Katherine Burke          1        4
```