Columbia University
In the City of New York

Lecture 3: Data Wrangling I

Data Science for Business Analytics

Thibault Vatter <thibault.vatter@unil.ch>

Department of Statistics, Columbia University and HEC Lausanne, UNIL

12.03.2018

Outline

1 Overview

2 Tibbles

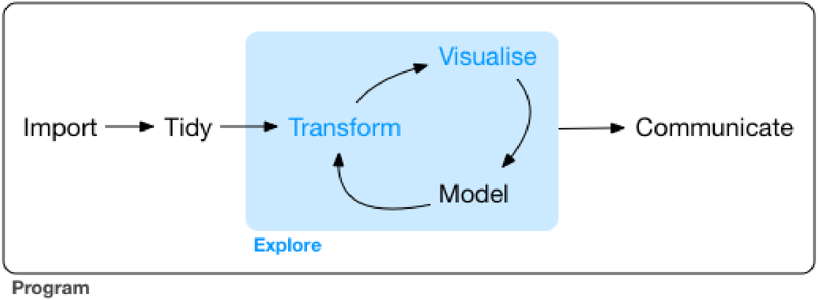
3 Data manipulation

4 The %>% operator

5 More on data manipulation

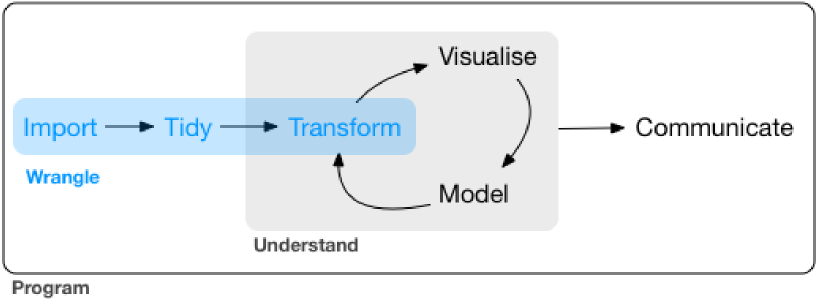
6 Tidy data

Today

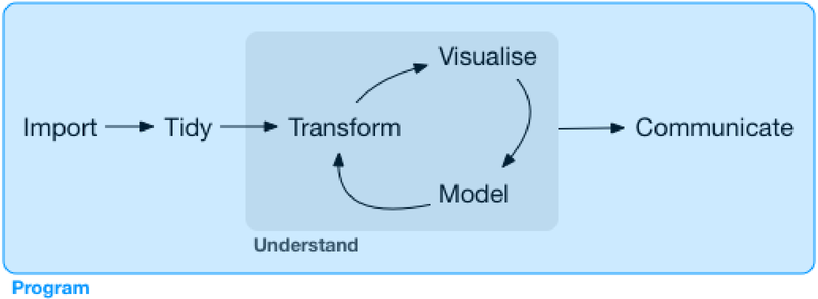


source: R for Data Science (like most figures in what follows)

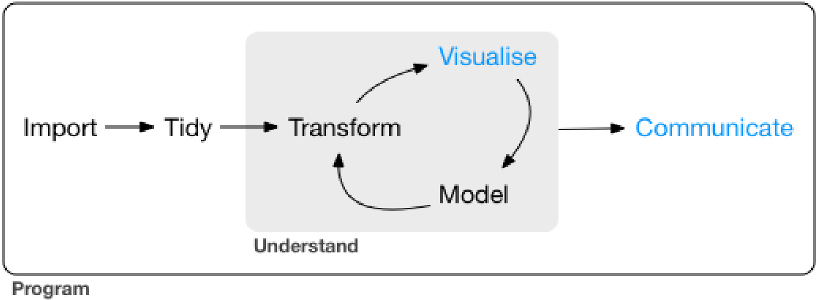
This morning



This morning



This afternoon



Outline

1 Overview

2 Tibbles

3 Data manipulation

4 The %>% operator

5 More on data manipulation

6 Tidy data

What are tibbles?

- Alternative R's traditional `data.frame`.
- Tweak some older behaviours to make life easier.
- Part of the core tidyverse.
- Unifying feature of the tidyverse.
- Most functions from the tidyverse produce tibbles.
- To learn more, see `vignette("tibble")`.

Coerce a data frame to a tibble

```r
as_tibble(iris)
```

```
## # A tibble: 150 x 5
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##           <dbl>       <dbl>        <dbl>       <dbl> <fct>
##  1         5.10        3.50         1.40       0.200 setosa
##  2         4.90        3.00         1.40       0.200 setosa
##  3         4.70        3.20         1.30       0.200 setosa
##  4         4.60        3.10         1.50       0.200 setosa
##  5         5.00        3.60         1.40       0.200 setosa
##  6         5.40        3.90         1.70       0.400 setosa
##  7         4.60        3.40         1.40       0.300 setosa
##  8         5.00        3.40         1.50       0.200 setosa
##  9         4.40        2.90         1.40       0.200 setosa
## 10         4.90        3.10         1.50       0.100 setosa
## # ... with 140 more rows
```

Create from individual vectors

```
tibble(x = 1:5,
y = 1,
z = x ^ 2 + y)
```

```
## # A tibble: 5 x 3
##       x     y     z
##   <int> <dbl> <dbl>
## 1     1  1.00  2.00
## 2     2  1.00  5.00
## 3     3  1.00 10.0
## 4     4  1.00 17.0
## 5     5  1.00 26.0
```

Row-wise tibble creation

```
tribble(
~colA, ~colB,
"a",   1,
"b",   2,
"c",   3
)
```

```
## # A tibble: 3 x 2
##   colA   colB
##   <chr> <dbl>
## 1 a      1.00
## 2 b      2.00
## 3 c      3.00
```

Printing tibbles

```
(df <- tibble(a = lubridate::today() + runif(4e1) * 30,
b = 1:4e1,
c = runif(4e1),
d = sample(letters, 4e1, replace = TRUE)))
```

```
## # A tibble: 40 x 4
##    a              b      c d
##    <date>     <int>  <dbl> <chr>
##  1 2018-04-01     1 0.411  y
##  2 2018-03-13     2 0.821  l
##  3 2018-03-17     3 0.647  s
##  4 2018-03-23     4 0.783  k
##  5 2018-04-02     5 0.553  i
##  6 2018-03-12     6 0.530  t
##  7 2018-04-01     7 0.789  f
##  8 2018-04-03     8 0.0233 s
##  9 2018-03-25     9 0.477  d
## 10 2018-03-24    10 0.732  g
## # ... with 30 more rows
```

Modify default settings

```
print(df, n = 2, width = 30)
```

```
## # A tibble: 40 x 4
##   a              b     c
##   <date>     <int> <dbl>
## 1 2018-04-01     1 0.411
## 2 2018-03-13     2 0.821
## # ... with 38 more rows, and
## #   1 more variable: d <chr>
```

- `options(tibble.print_max = n, tibble.print_min = m)`: if more than `m` rows, print only `n` rows.
- `options(dplyr.print_min = Inf)` to always show all rows.
- `options(tibble.width = Inf)` to always print all columns.
- `package?tibble`

Subsetting tibbles

```
# Extract by name I
df$b
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
## [24] 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
```

```
# Extract by name II
df[["b"]]
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
## [24] 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
```

```
# Extract by position
df[[2]]
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
## [24] 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
```

Compared to a `data.frame`:

- no partial matching
- warning if the column does not exist

Outline

1 Overview

2 Tibbles

3 Data manipulation

4 The %>% operator

5 More on data manipulation

6 Tidy data

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.

`dplyr` **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:

- All verbs operate groupwise with `group_by()`.
- All 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      2.00      830
##  2  2013     1     1      533            529      4.00      850
##  3  2013     1     1      542            540      2.00      923
##  4  2013     1     1      544            545     -1.00     1004
##  5  2013     1     1      554            600     -6.00      812
##  6  2013     1     1      554            558     -4.00      740
##  7  2013     1     1      555            600     -5.00      913
##  8  2013     1     1      557            600     -3.00      709
##  9  2013     1     1      557            600     -3.00      838
## 10  2013     1     1      558            600     -2.00      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>
```

Filter rows with `filter()`

```
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      2.00      830
##  2  2013     1     1      533            529      4.00      850
##  3  2013     1     1      542            540      2.00      923
##  4  2013     1     1      544            545     -1.00     1004
##  5  2013     1     1      554            600     -6.00      812
##  6  2013     1     1      554            558     -4.00      740
##  7  2013     1     1      555            600     -5.00      913
##  8  2013     1     1      557            600     -3.00      709
##  9  2013     1     1      557            600     -3.00      838
## 10  2013     1     1      558            600     -2.00      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?

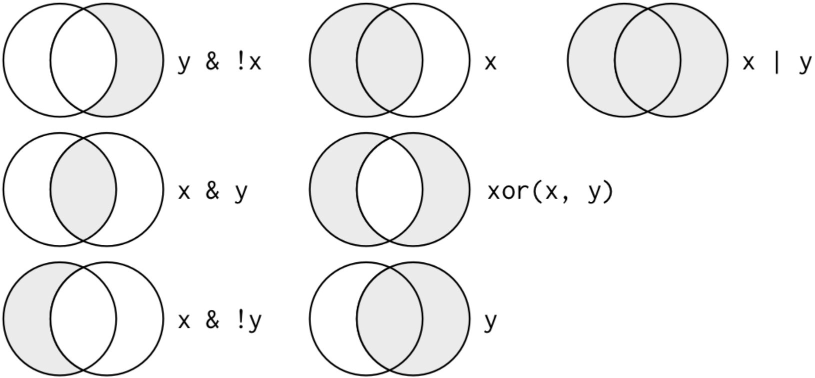
```
sqrt(2) ^ 2 == 2
1/49 * 49 == 1
near(sqrt(2) ^ 2,  2)
near(1 / 49 * 49, 1)
```

```
## [1] FALSE
## [1] FALSE
## [1] TRUE
## [1] TRUE
```

Logical operators

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

- & for “and”
- | for “or”
- ! for “not”



What is this code doing?

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

What is this code doing?

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

Literally “finds all flights that departed in November or December”, but you can’t write `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", but you can't write `filter(flights, month == 11 | 12)`.
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`

```
filter(flights, !(arr_delay > 120 | dep_delay > 120))
filter(flights, arr_delay <= 120, dep_delay <= 120)
```

Missing values

NAs ("not availables") are "contagious":

```
NA > 5
10 == NA
NA + 10
NA / 2
NA == NA
```

```
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
```

To determine if a value is missing':

```
is.na(NA)
```

```
## [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  3.00
```

```
filter(df, is.na(x) | x > 1)
```

```
## # A tibble: 2 x 1
##       x
##   <dbl>
## 1 NA
## 2  3.00
```

Arrange rows with arrange()

```r
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      2.00      830
##  2  2013     1     1      533            529      4.00      850
##  3  2013     1     1      542            540      2.00      923
##  4  2013     1     1      544            545     -1.00     1004
##  5  2013     1     1      554            600     -6.00      812
##  6  2013     1     1      554            558     -4.00      740
##  7  2013     1     1      555            600     -5.00      913
##  8  2013     1     1      557            600     -3.00      709
##  9  2013     1     1      557            600     -3.00      838
## 10  2013     1     1      558            600     -2.00      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  2.00
## 2  5.00
## 3 NA
```

```
arrange(df, desc(x))
```

```
## # A tibble: 3 x 1
##       x
##   <dbl>
## 1  5.00
## 2  2.00
## 3 NA
```

Select columns with `select()`

```
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

```r
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

```r
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      2.00      830            819
##  2      533            529      4.00      850            830
##  3      542            540      2.00      923            850
##  4      544            545     -1.00     1004           1022
##  5      554            600     -6.00      812            837
##  6      554            558     -4.00      740            728
##  7      555            600     -5.00      913            854
##  8      557            600     -3.00      709            723
##  9      557            600     -3.00      838            846
## 10      558            600     -2.00      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()

```
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     53.0  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.

Create a narrower dataset

```r
(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      2.00      11.0      1400     227
##  2  2013     1     1      4.00      20.0      1416     227
##  3  2013     1     1      2.00      33.0      1089     160
##  4  2013     1     1     -1.00     -18.0      1576     183
##  5  2013     1     1     -6.00     -25.0       762     116
##  6  2013     1     1     -4.00      12.0       719     150
##  7  2013     1     1     -5.00      19.0      1065     158
##  8  2013     1     1     -3.00     -14.0       229      53.0
##  9  2013     1     1     -3.00     - 8.00      944     140
## 10  2013     1     1     -2.00       8.00      733     138
## # ... with 336,766 more rows
```

Add new variables with `mutate()`

```
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      2.00     11.0      1400    227     9.00
##  2  2013     1     1      4.00     20.0      1416    227    16.0
##  3  2013     1     1      2.00     33.0      1089    160    31.0
##  4  2013     1     1     -1.00    -18.0      1576    183   -17.0
##  5  2013     1     1     -6.00    -25.0       762    116   -19.0
##  6  2013     1     1     -4.00     12.0       719    150    16.0
##  7  2013     1     1     -5.00     19.0      1065    158    24.0
##  8  2013     1     1     -3.00    -14.0       229     53.0 -11.0
##  9  2013     1     1     -3.00    - 8.00      944    140   - 5.00
## 10  2013     1     1     -2.00      8.00      733    138    10.0
## # ... with 336,766 more rows, and 1 more variable: speed <dbl>
```

Refer to columns just created

```
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      2.00     11.0      1400    227     9.00
##  2  2013     1     1      4.00     20.0      1416    227    16.0
##  3  2013     1     1      2.00     33.0      1089    160    31.0
##  4  2013     1     1     -1.00    -18.0      1576    183   -17.0
##  5  2013     1     1     -6.00    -25.0       762    116   -19.0
##  6  2013     1     1     -4.00     12.0       719    150    16.0
##  7  2013     1     1     -5.00     19.0      1065    158    24.0
##  8  2013     1     1     -3.00    -14.0       229     53.0 -11.0
##  9  2013     1     1     -3.00    - 8.00      944    140   - 5.00
## 10  2013     1     1     -2.00      8.00      733    138    10.0
## # ... 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   9.00 3.78           2.38
##   2  16.0  3.78           4.23
##   3  31.0  2.67          11.6
##   4 -17.0  3.05         - 5.57
##   5 -19.0  1.93         - 9.83
##   6  16.0  2.50           6.40
##   7  24.0  2.63           9.11
##   8 -11.0  0.883        -12.5
##   9 - 5.00 2.33         - 2.14
##  10  10.0  2.30           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

- Offsets: `lead()` and `lag()`:
  - Refer to lead-/lagging values (e.g. to get running differences `x - lag(x)` or find when values change `x != lag(x)`).
  - Useful in conjunction with `group_by()`.

```
x <- 1:10
lag(x)
lead(x)
```

```
##  [1] NA  1  2  3  4  5  6  7  8  9
##  [1]  2  3  4  5  6  7  8  9 10 NA
```

- Cumulative aggregates: `cumsum()`, `cumprod()`, `cummin()`, `cummax()`, `cummean()` (`RcppRoll` package for rolling aggregates).

```
cumsum(x)
cummean(x)
```

```
##  [1]  1  3  6 10 15 21 28 36 45 55
##  [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

- 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)
min_rank(desc(y))
row_number(y)
dense_rank(y)
percent_rank(y)
cume_dist(y)
```

```
## [1]  1  2  2 NA  4  5
## [1]  5  3  3 NA  2  1
## [1]  1  2  3 NA  4  5
## [1]  1  2  2 NA  3  4
## [1] 0.00 0.25 0.25   NA 0.75 1.00
## [1] 0.2 0.6 0.6  NA 0.8 1.0
```

Collapse values with `summarize()`

```r
summarize(flights, delay = mean(dep_delay, na.rm = TRUE))
```

```
## # A tibble: 1 x 1
##   delay
##   <dbl>
## 1  12.6
```

summarize() paired with group_by()

```
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 [?]
##     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

1 Overview

2 Tibbles

3 Data manipulation

4 The %>% operator

5 More on data manipulation

6 Tidy data

What is this code doing?

```r
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)

```r
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
```

The %>% operator

Makes your code more readable by:

- structuring sequences of data operations left-to-right,
- minimizing the need for local variables and function definitions,
- making it easy to add steps anywhere in the sequence of operations.

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)`

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

x %>% sum
```

```
## [1] 55
```

```r
x %>% f(y)
```

```
##  [1]  3  5  7  9 11 13 15 17 19 21
```

```r
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)`

```r
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
```

```r
x %>% f(y, z = .)
```

```
##  [1] 2 2 2 2 2 2 2 2 2 2
```

Subsetting tibbles revisited

```r
df <- tibble(a = lubridate::today() + runif(4e1) * 30,
             b = 1:4e1,
             c = runif(4e1),
             d = sample(letters, 4e1, replace = TRUE))
```

```r
# Extract by name
df %>% .$b
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
## [24] 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
```

```r
# Extract by position
df %>% .[["b"]]
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
## [24] 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
```

Outline

1 Overview

2 Tibbles

3 Data manipulation

4 The %>% operator

5 More on data manipulation

6 Tidy data

What is happening here?

```
flights %>%
  group_by(year, month, day) %>%
  summarize(mean = mean(dep_delay))
```

```
## # A tibble: 365 x 4
## # Groups:   year, month [?]
##     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

```
flights %>%
  group_by(year, month, day) %>%
  summarize(mean = mean(dep_delay, na.rm = TRUE))
```

```
## # A tibble: 365 x 4
## # Groups:   year, month [?]
##     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
```

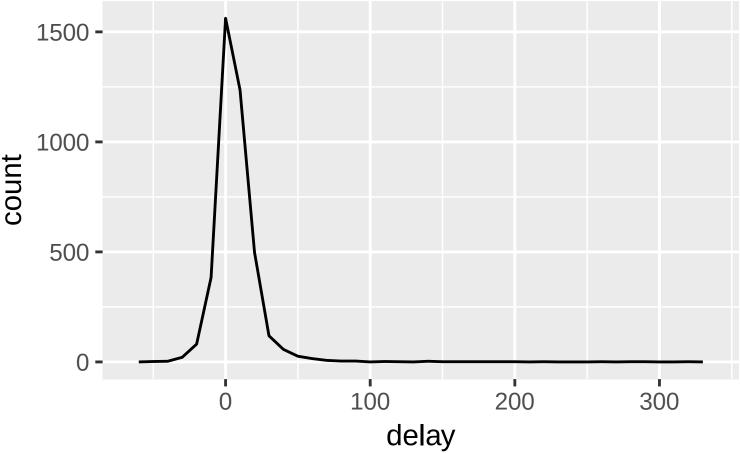
Or pre-filter the dataset

```r
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 [?]
##     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
```

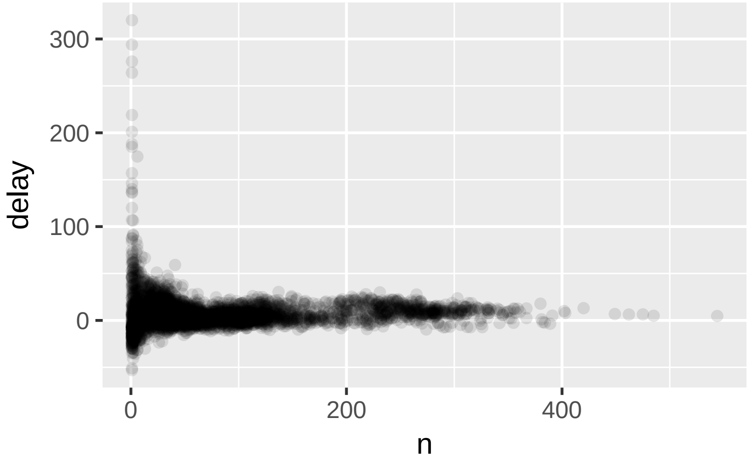
What do you see?

```
delays <- not_cancelled %>%
  group_by(tailnum) %>%
  summarize(delay = mean(arr_delay))
```



Counts

```
delays <- not_cancelled %>%
  group_by(tailnum) %>%
  summarize(delay = mean(arr_delay, na.rm = TRUE), n = n())
```



Useful summary functions I

- Measures of location: `mean()`, `median()`.
- Measures of spread: `sd()`, `IQR()`, `mad()`.

Useful summary functions II

- 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 [?]
##     year month   day  first  last
##    <int> <int> <int>  <dbl> <dbl>
##  1  2013     1     1 517     2356
##  2  2013     1     2  42.0   2354
##  3  2013     1     3  32.0   2349
##  4  2013     1     4  25.0   2358
##  5  2013     1     5  14.0   2357
##  6  2013     1     6  16.0   2355
##  7  2013     1     7  49.0   2359
##  8  2013     1     8 454     2351
##  9  2013     1     9   2.00  2252
## 10  2013     1    10   3.00  2320
## # ... with 355 more rows
```

Useful summary functions III

- 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 [?]
##     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 IV

- 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 V

- 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 VI

- 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 VII

- 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 [?]
##     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 VIII

- 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 [?]
##     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

```
daily <- group_by(flights, year, month, day)
(per_day   <- summarize(daily, flights = n()))
```

```
## # A tibble: 365 x 4
## # Groups:   year, month [?]
##     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

```
(per_month <- summarize(per_day, flights = sum(flights)))
(per_year  <- summarize(per_month, flights = sum(flights)))
```

```
## # A tibble: 12 x 3
## # Groups:   year [?]
##     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      2.00      830
##  2  2013     1     1      533            529      4.00      850
##  3  2013     1     1      542            540      2.00      923
##  4  2013     1     1      544            545     -1.00     1004
##  5  2013     1     1      554            600     -6.00      812
##  6  2013     1     1      554            558     -4.00      740
##  7  2013     1     1      555            600     -5.00      913
##  8  2013     1     1      557            600     -3.00      709
##  9  2013     1     1      557            600     -3.00      838
## 10  2013     1     1      558            600     -2.00      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

```
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   0.000111
##  2  2013     1     1 IAH       20.0   0.000201
##  3  2013     1     1 MIA       33.0   0.000235
##  4  2013     1     1 ORD       12.0   0.0000424
##  5  2013     1     1 FLL       19.0   0.0000938
##  6  2013     1     1 ORD        8.00  0.0000283
##  7  2013     1     1 LAX        7.00  0.0000344
##  8  2013     1     1 DFW       31.0   0.000282
##  9  2013     1     1 ATL       12.0   0.0000400
## 10  2013     1     1 DTW       16.0   0.000116
## # ... with 131,096 more rows
```

Outline

1 Overview

2 Tibbles

3 Data manipulation

4 The %>% operator

5 More on data manipulation

6 Tidy data

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.

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
```

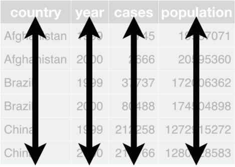
```
table4b  # population
```

```
## # A tibble: 3 x 3
##   country         ‘1999‘     ‘2000‘
## * <chr>            <int>      <int>
## 1 Afghanistan   19987071   20595360
## 2 Brazil       172006362  174504898
## 3 China       1272915272 1280428583
```

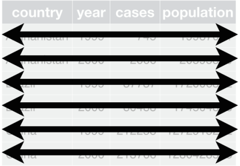
What makes a dataset tidy?

Three interrelated rules:

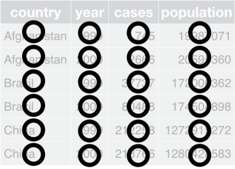
1. Each variable must have its own column.
2. Each observation must have its own row.
3. Each value must have its own cell.



variables



observations



values

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

1. Put each dataset in a tibble.
2. Put each variable in a column.

Why ensure that your data is tidy?

1. If you have a consistent data structure, it's easier to learn the tools that work with it because they have an underlying uniformity.
2. There's a specific advantage to placing variables in columns because it allows R's vectorized nature to shine.

The principles of tidy data seem obvious, BUT:

1. Most people aren't familiar with the principles of tidy data.
2. Data is often organised to facilitate some use other than analysis.

Hence, for most real analyses, you'll need to do some tidying.

The two steps of tidying

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

To fix these problems, you'll need `gather()` and `spread()`.

Gathering with gather()

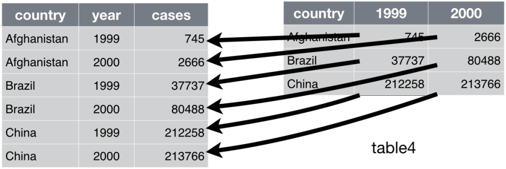
```
table4a
```

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

```
table4a %>% gather(‘1999‘, ‘2000‘, key = "year", value = "cases")
```

```
## # A tibble: 6 x 3
##   country     year   cases
##   <chr>       <chr>  <int>
## 1 Afghanistan 1999     745
## 2 Brazil      1999   37737
## 3 China       1999  212258
## 4 Afghanistan 2000    2666
## 5 Brazil      2000   80488
## 6 China       2000  213766
```

Visual interpretation of `gather()`



country	year	cases
Afghanistan	1999	745
Afghanistan	2000	2666
Brazil	1999	37737
Brazil	2000	80488
China	1999	212258
China	2000	213766

country	1999	2000
Afghanistan	745	2666
Brazil	37737	80488
China	212258	213766

table4

Spreading with `spread()` I

```r
table2
```

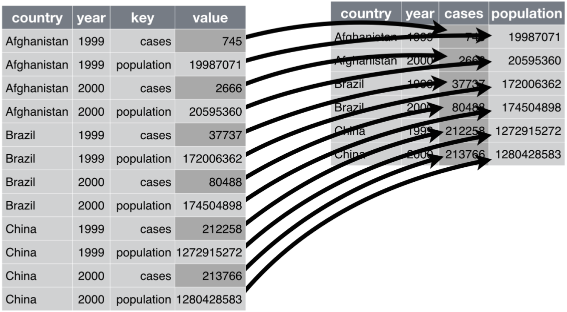
```
## # A tibble: 12 x 4
##    country      year type            count
##    <chr>       <int> <chr>           <int>
##  1 Afghanistan  1999 cases             745
##  2 Afghanistan  1999 population   19987071
##  3 Afghanistan  2000 cases            2666
##  4 Afghanistan  2000 population   20595360
##  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
```

Spreading with `spread()` II

```
table2 %>% spread(key = type, value = 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
```

Visual interpretation of `spread()`



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() I

```
table3
```

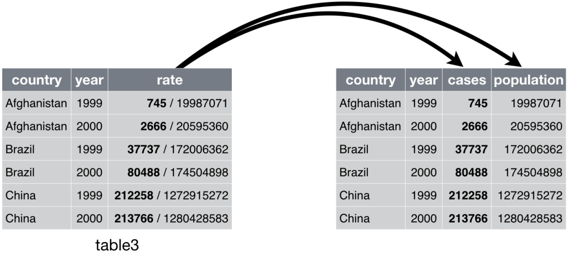
```
## # A tibble: 6 x 3
##   country      year rate
## * <chr>       <int> <chr>
## 1 Afghanistan  1999 745/19987071
## 2 Afghanistan  2000 2666/20595360
## 3 Brazil       1999 37737/172006362
## 4 Brazil       2000 80488/174504898
## 5 China        1999 212258/1272915272
## 6 China        2000 213766/1280428583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

Separate a column with separate() II

```
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
```

Visual interpretation of `separate()`



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()`

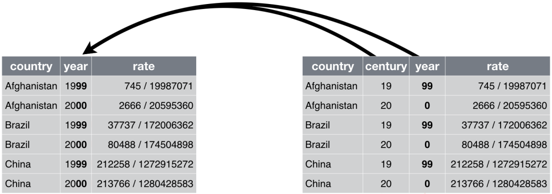
```
table5
```

```
## # 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
```

Visual interpretation of `unite()`



country	year	rate
Afghanistan	19**99**	745 / 19987071
Afghanistan	20**00**	2666 / 20595360
Brazil	19**99**	37737 / 172006362
Brazil	20**00**	80488 / 174504898
China	19**99**	212258 / 1272915272
China	20**00**	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

```
stocks %>%
  spread(year, return)
```

```
## # A tibble: 4 x 3
##     qtr ‘2015‘ ‘2016‘
##   <dbl>  <dbl>  <dbl>
## 1  1.00  1.88  NA
## 2  2.00  0.590  0.920
## 3  3.00  0.350  0.170
## 4  4.00 NA      2.66
```

Explicit to implicit

```
stocks %>%
  spread(year, return) %>%
  gather(year, return, ‘2015‘:‘2016‘, na.rm = TRUE)
```

```
## # A tibble: 6 x 3
##     qtr year  return
## * <dbl> <chr>  <dbl>
## 1  1.00 2015   1.88
## 2  2.00 2015   0.590
## 3  3.00 2015   0.350
## 4  2.00 2016   0.920
## 5  3.00 2016   0.170
## 6  4.00 2016   2.66
```

Implicit to explicit with `complete()`

```
stocks %>% complete(year, qtr)
```

```
## # A tibble: 8 x 3
##    year   qtr return
##   <dbl> <dbl>  <dbl>
## 1  2015  1.00  1.88
## 2  2015  2.00  0.590
## 3  2015  3.00  0.350
## 4  2015  4.00 NA
## 5  2016  1.00 NA
## 6  2016  2.00  0.920
## 7  2016  3.00  0.170
## 8  2016  4.00  2.66
```

Fill in missing values with `fill()`

```r
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      1.00     7.00
## 2 Derrick Whitmore      2.00    10.0
## 3 Derrick Whitmore      3.00     9.00
## 4 Katherine Burke       1.00     4.00
```