Session 3: Into the Tidyverse!

R Basics for Social Sciences

Patrick Kraft
IC3JM / UC3M
05/02/2025

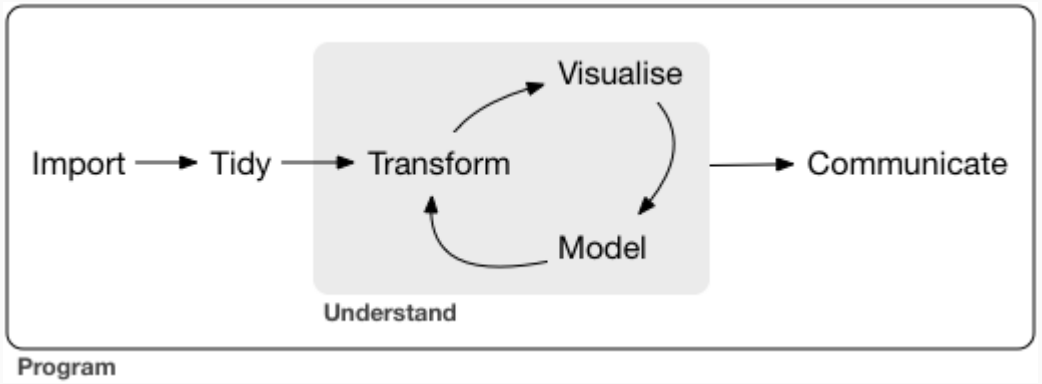
Outline

1. Introducing the tidyverse
2. Data visualization: `ggplot2`
3. Data transformation: `dplyr`
4. Data import: `readr`, `haven`, and `rio`
5. Tidy data: `tidyr`

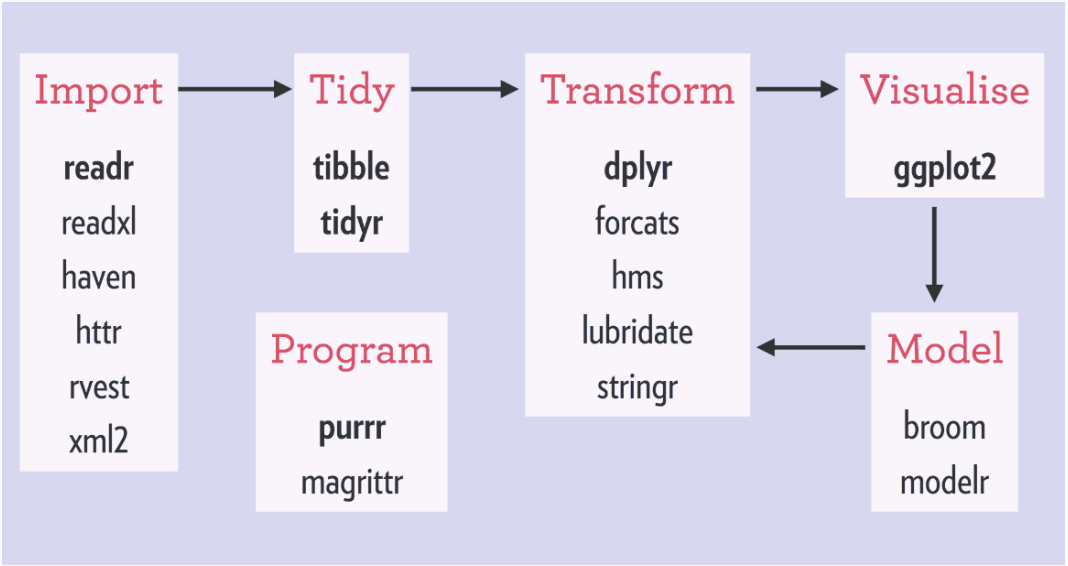
1. Introducing the tidyverse

Why the tidyverse?

Here is a model of the different steps of a typical data science project (from *R for Data Science*):



The tidyverse provides a useful set of tools to facilitate each of these steps!



Using the pipe operator (%>% or |>) in R

How mornings look like in base-r:

```
leave_house(get_dressed(
  get_out_of_bed(wake_up(me)), jacket = TRUE))
```

How mornings look like using the pipe:

```
me %>%
  wake_up() %>%
  get_out_of_bed() %>%
  get_dressed(jacket = TRUE) %>%
  leave_house()
```

Shortcut for pipe operator:

```
Shift + Cmd + M (Mac)
Shift + Ctrl + M (Windows)
```

2. Data visualization

ggplot2

ggplot2: A Layered Grammar of Graphics

Describes all the non-data ink	Theme
Plotting space for the data	Coordinates
Statistical models & summaries	Statistics
Rows and columns of sub-plots	Facets
Shapes used to represent the data	Geometries
Scales onto which data is mapped	Aesthetics
The actual variables to be plotted	Data

First steps

- Load tidyverse

```
library(tidyverse)
```

- Take a look at the `mpg` data frame. Among the variables in `mpg` are:
  - `displ`, a car's engine size, in liters.
  - `hwy`, a car's fuel efficiency on the highway, in miles per gallon (mpg).

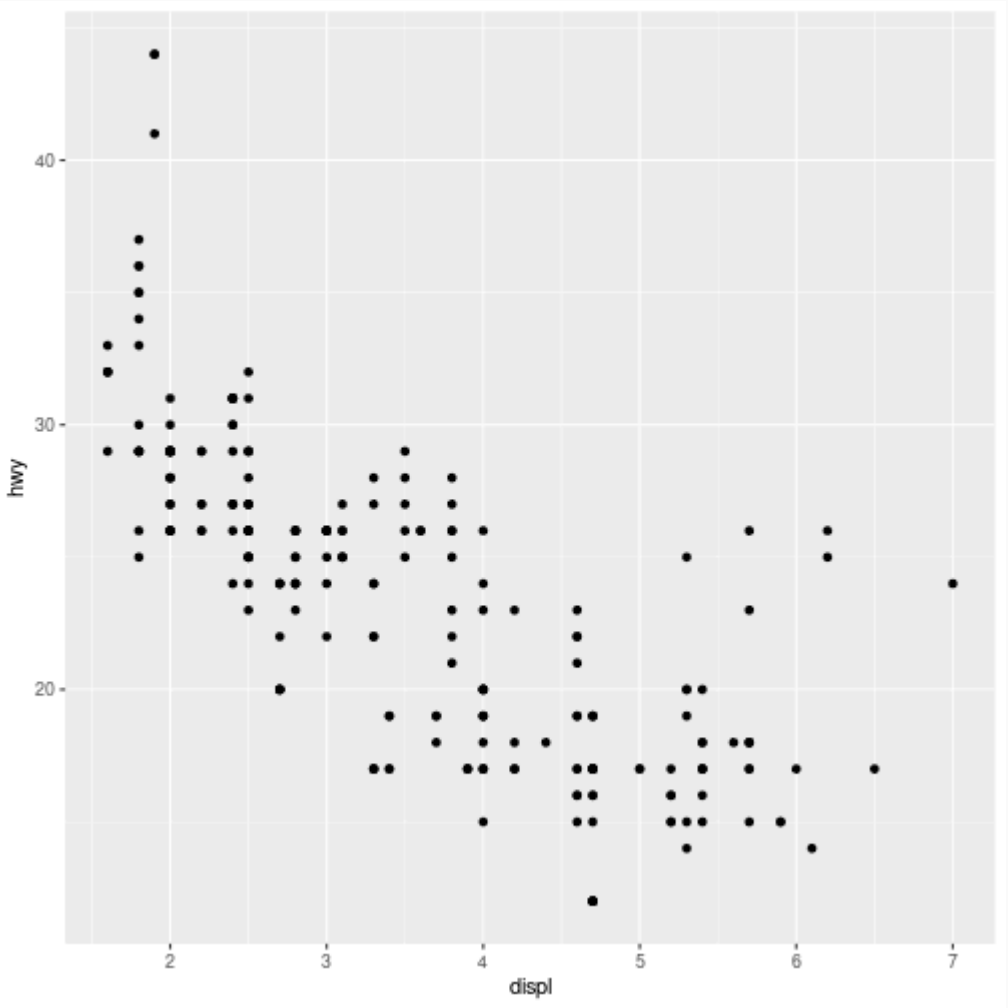
```
mpg
```

```
## # A tibble: 234 × 11
##    manufacturer model      displ  year   cyl trans      drv     cty   hwy fl    class
##    <chr>        <chr>      <dbl> <int> <int> <chr>      <chr> <int> <int> <chr> <chr>
##  1 audi         a4           1.8  1999     4 auto(l5)   f        18    29 p     compact
##  2 audi         a4           1.8  1999     4 manual(m5) f        21    29 p     compact
##  3 audi         a4           2    2008     4 manual(m6) f        20    31 p     compact
##  4 audi         a4           2    2008     4 auto(av)   f        21    30 p     compact
##  5 audi         a4           2.8  1999     6 auto(l5)   f        16    26 p     compact
##  6 audi         a4           2.8  1999     6 manual(m5) f        18    26 p     compact
##  7 audi         a4           3.1  2008     6 auto(av)   f        18    27 p     compact
##  8 audi         a4 quattro   1.8  1999     4 manual(m5) 4        18    26 p     compact
##  9 audi         a4 quattro   1.8  1999     4 auto(l5)   4        16    25 p     compact
## 10 audi         a4 quattro   2    2008     4 manual(m6) 4        20    28 p     compact
```

Creating our first ggplot

- Every `ggplot` consists of (at least) one **dataset**, one **aesthetic mapping**, and one **geometry**:

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point()
```

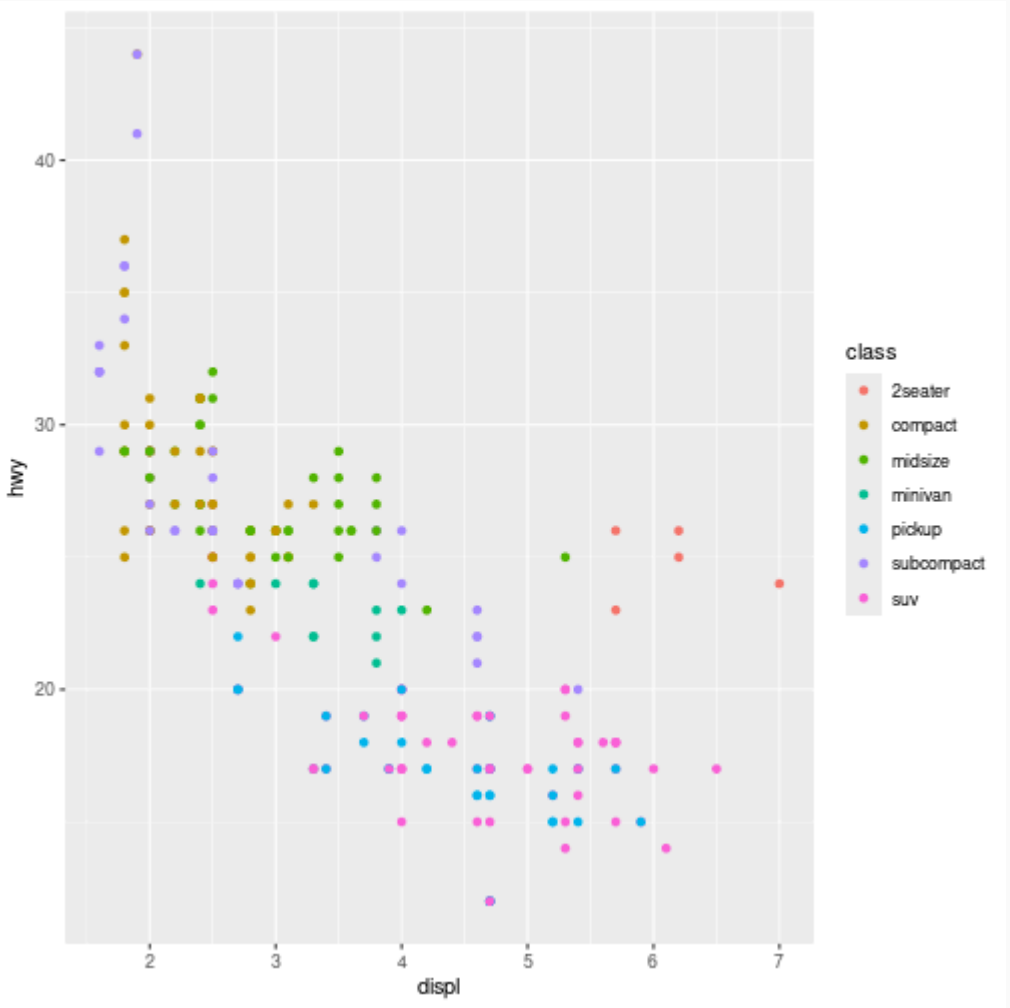


Aesthetic mappings

- There are many different ways to map **data** onto **aesthetics**:

```
ggplot(mpg, aes(x = displ, y = hwy, col = class)) +
  geom_point()
```

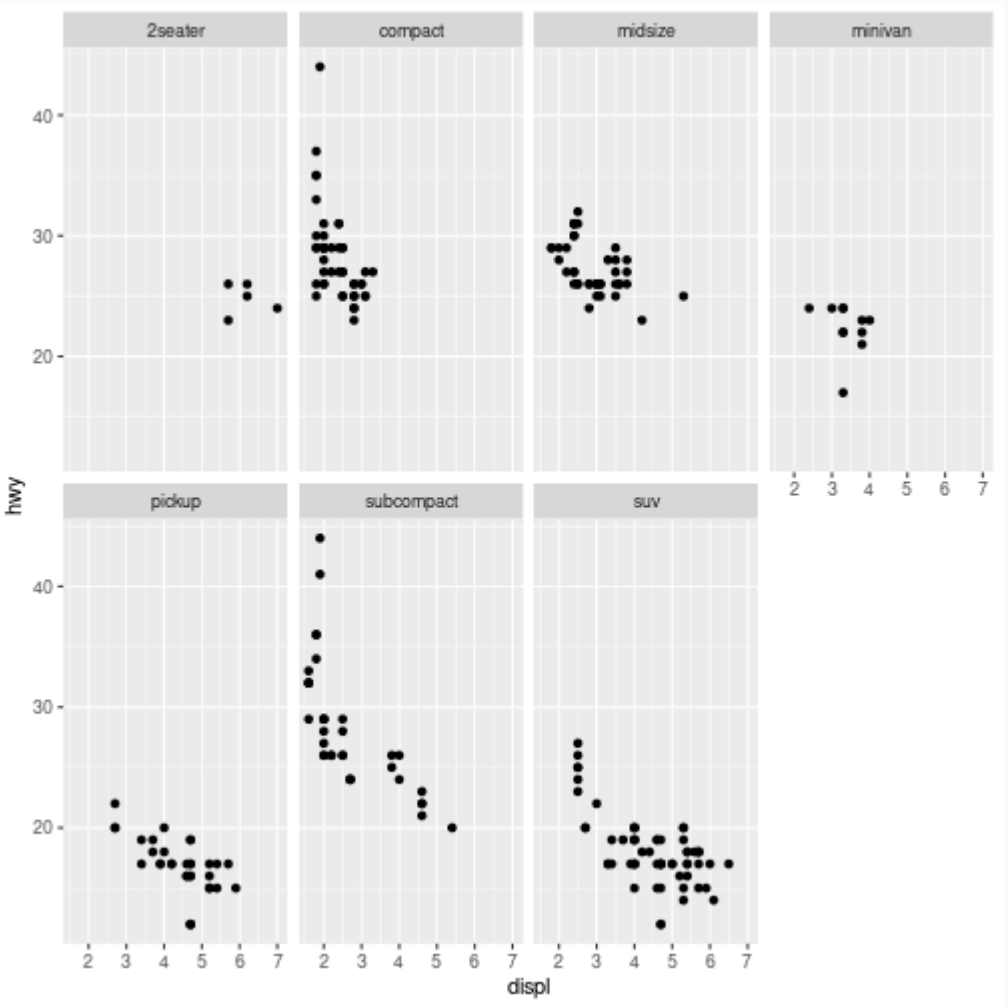
- Other aesthetics: `size`, `shape`, `alpha`, ...



Facets

- Facets determine the rows and columns of sub-plots

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  facet_wrap(~class, nrow = 2)
```

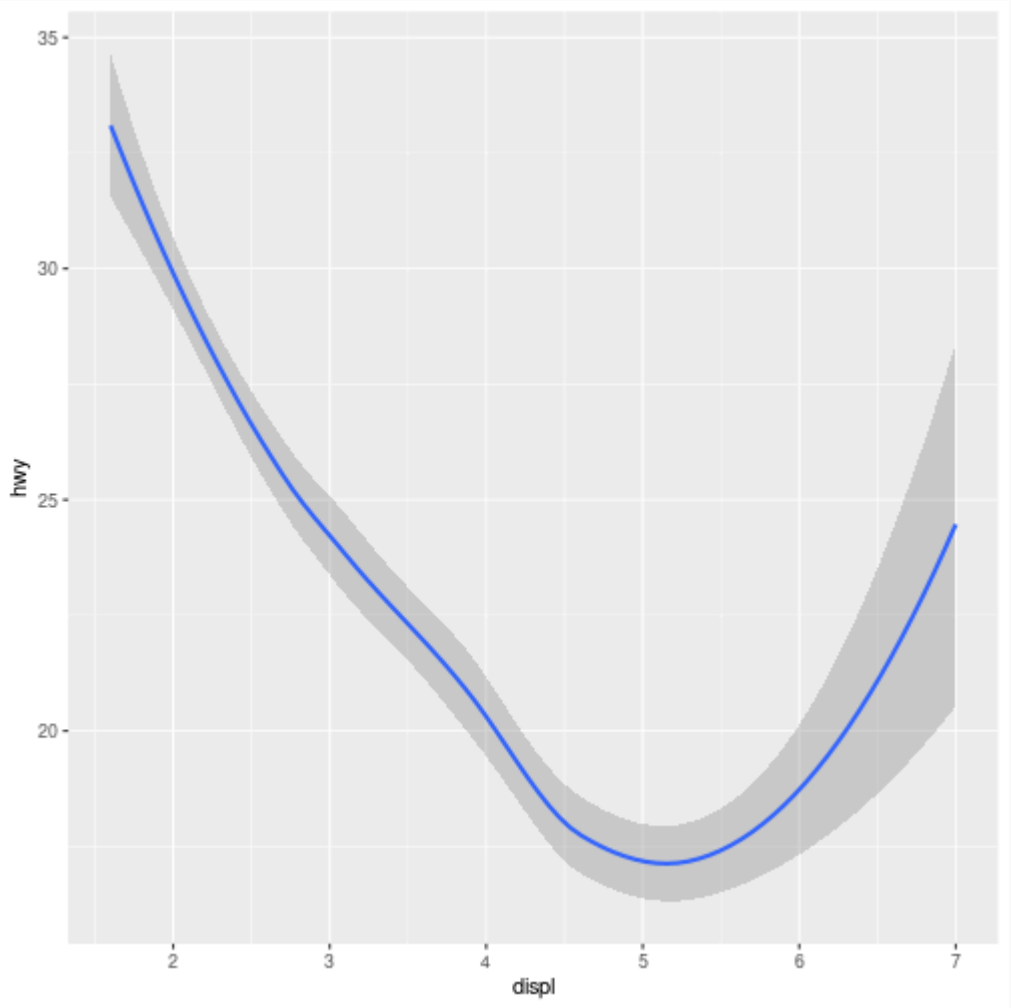


Geometric objects

- Depending on the underlying data, we can choose different **geometries** to visualize the aesthetic mappings:

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x
```

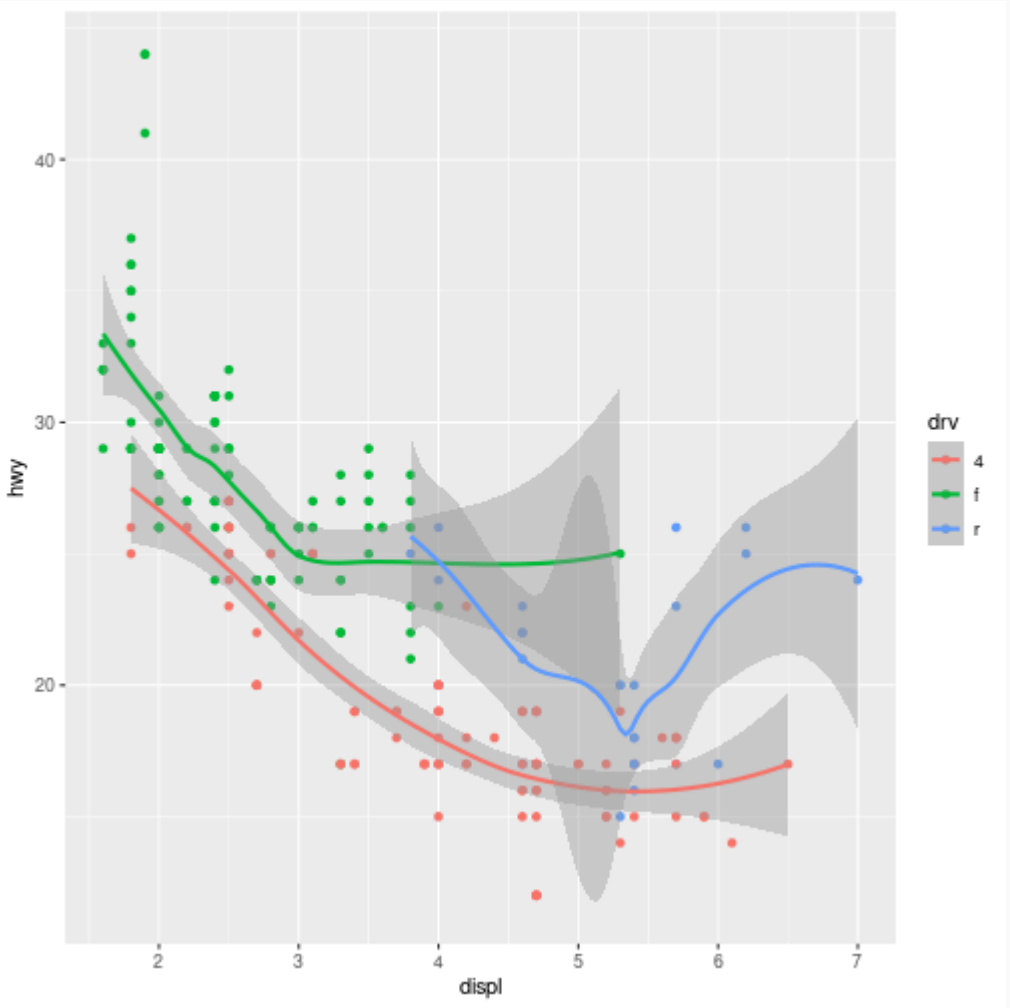


Geometric objects

- Several geometries can be **combined** in a single plot:

```
ggplot(mpg, aes(x = displ, y = hwy, col = drv)) +
  geom_point() +
  geom_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x
```

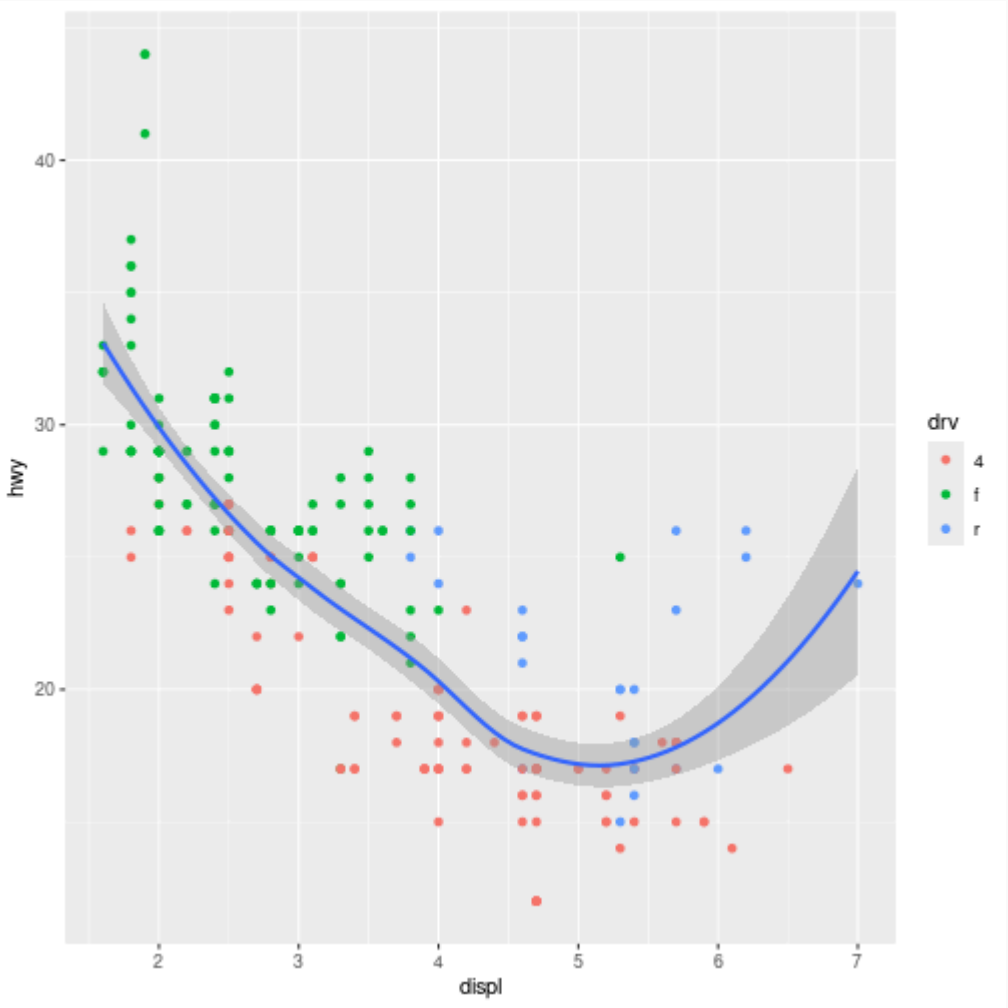


Geometric objects

- We can also specify **aesthetics** for individual **geometries**:

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(aes(col = drv)) +
  geom_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x
```



Statistical transformations

Let's take a look at a different dataset:

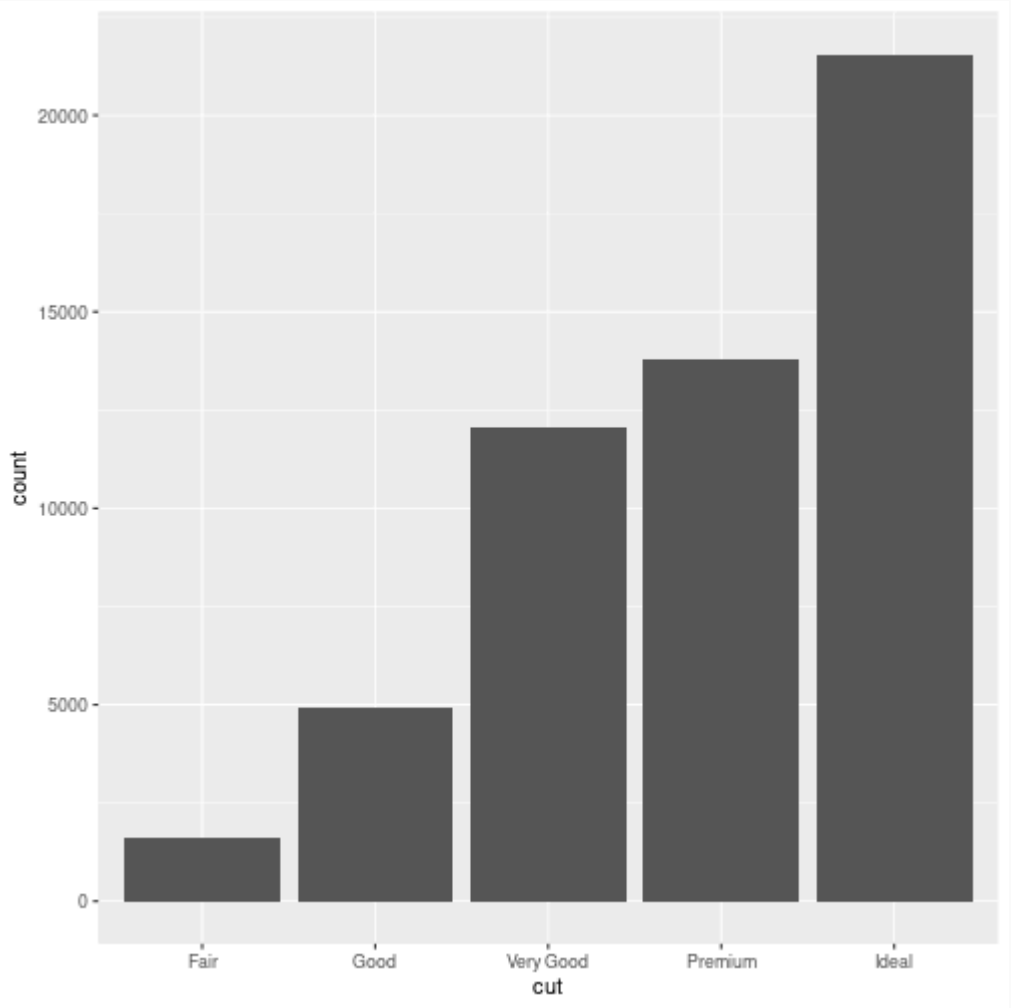
```
diamonds
```

```
## # A tibble: 53,940 × 10
##    carat cut       color clarity depth table price     x     y     z
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
##  2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
##  3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31
##  4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
##  5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75
##  6  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
##  7  0.24 Very Good I     VVS1     62.3    57   336  3.95  3.98  2.47
##  8  0.26 Very Good H     SI1      61.9    55   337  4.07  4.11  2.53
##  9  0.22 Fair      E     VS2      65.1    61   337  3.87  3.78  2.49
## 10  0.23 Very Good H     VS1      59.4    61   338  4     4.05  2.39
## # ℹ 53,930 more rows
```

Statistical transformations

- Some **geometries** perform default statistical transformations:

```
ggplot(diamonds, aes(x = cut)) +
  geom_bar()
```



Statistical transformations

What's going on?

1. **`geom_bar()`** begins with the **`diamonds`** data set

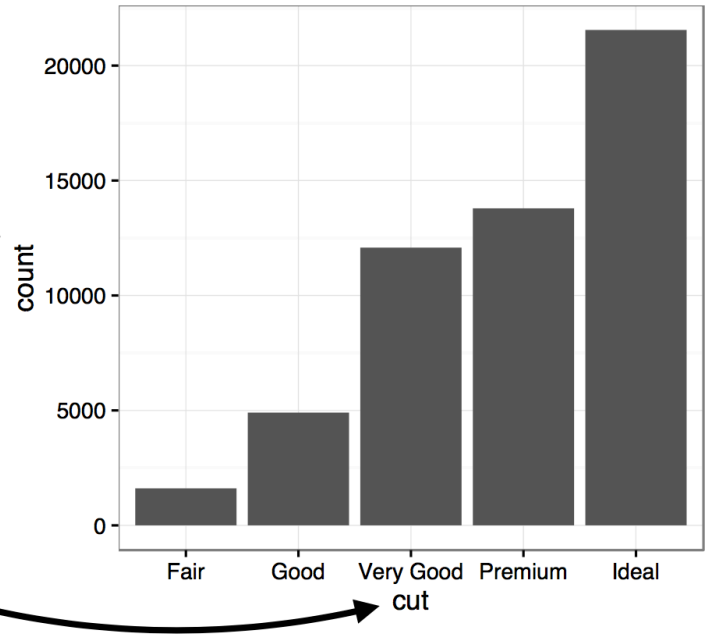
2. **`geom_bar()`** transforms the data with the "count" stat, which returns a data set of cut values and counts.

3. **`geom_bar()`** uses the transformed data to build the plot. cut is mapped to the x axis, count is mapped to the y axis.

carat	cut	color	clarity	depth	table	price	x	y	z
0.23	Ideal	E	SI2	61.5	55	326	3.95	3.98	2.43
0.21	Premium	E	SI1	59.8	61	326	3.89	3.84	2.31
0.23	Good	E	VS1	56.9	65	327	4.05	4.07	2.31
0.29	Premium	I	VS2	62.4	58	334	4.20	4.23	2.63
0.31	Good	J	SI2	63.3	58	335	4.34	4.35	2.75
...	...	...	...	...	...	...	...	...	...

stat_count()

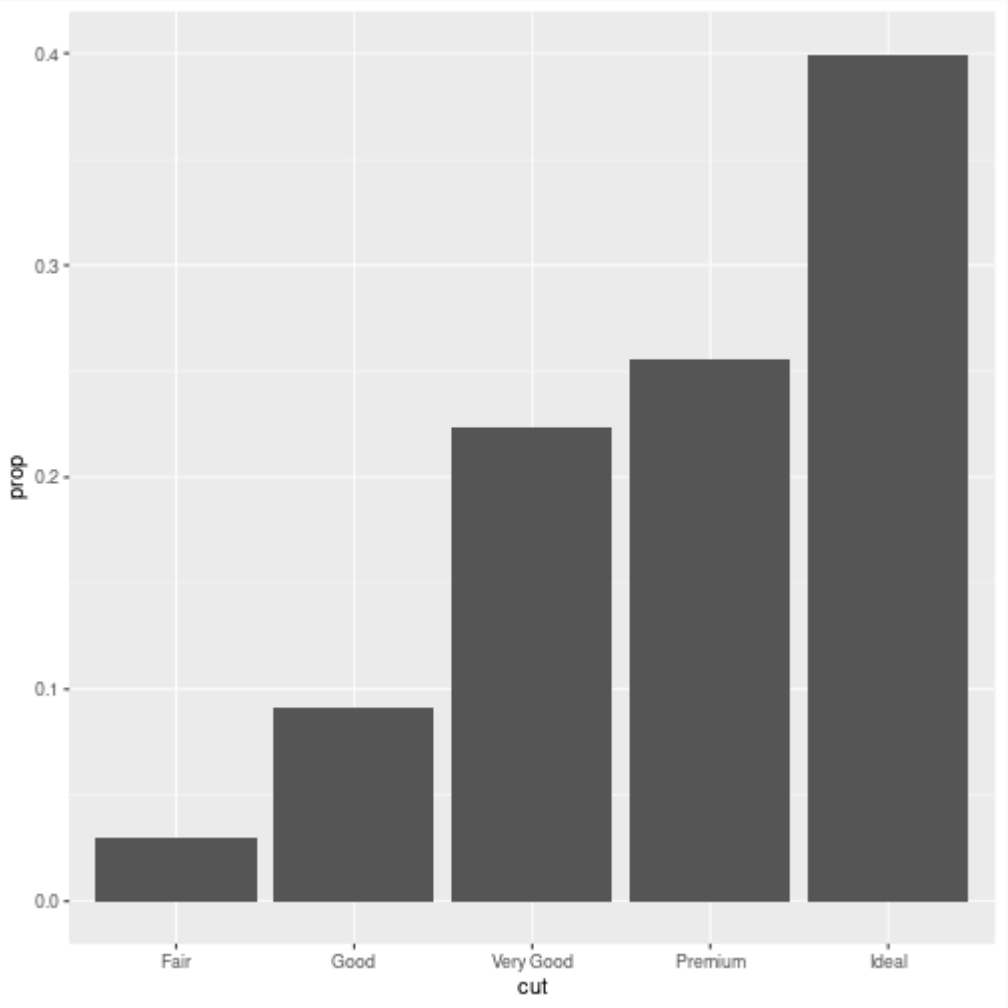
cut	count	prop
Fair	1610	1
Good	4906	1
Very Good	12082	1
Premium	13791	1
Ideal	21551	1



Statistical transformations

- We can override the default mapping, for example if we want to display proportions instead of counts:

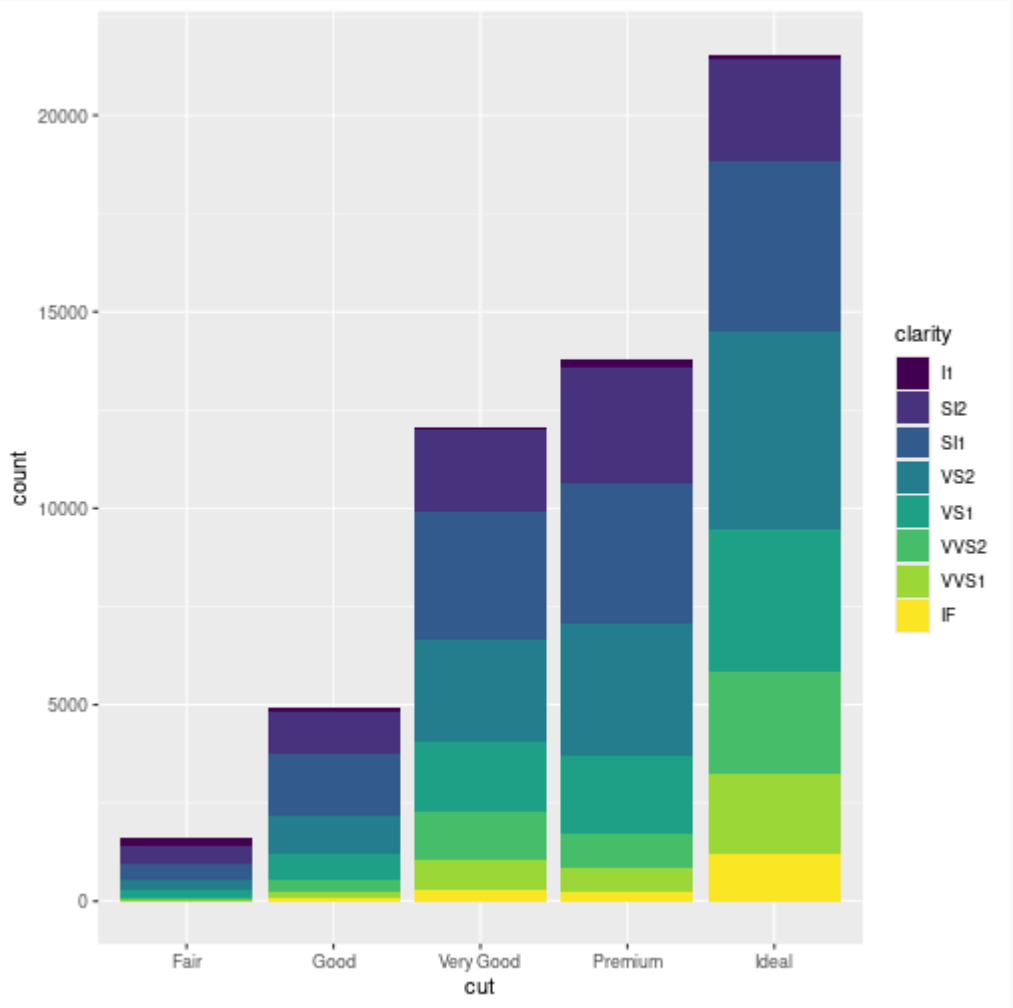
```
ggplot(diamonds, aes(x = cut,
                     y = after_stat(prop),
                     group = 1)) +
  geom_bar()
```



Position adjustments

- We can add a `fill` asthetic to the bar chart that maps to an additional variable:

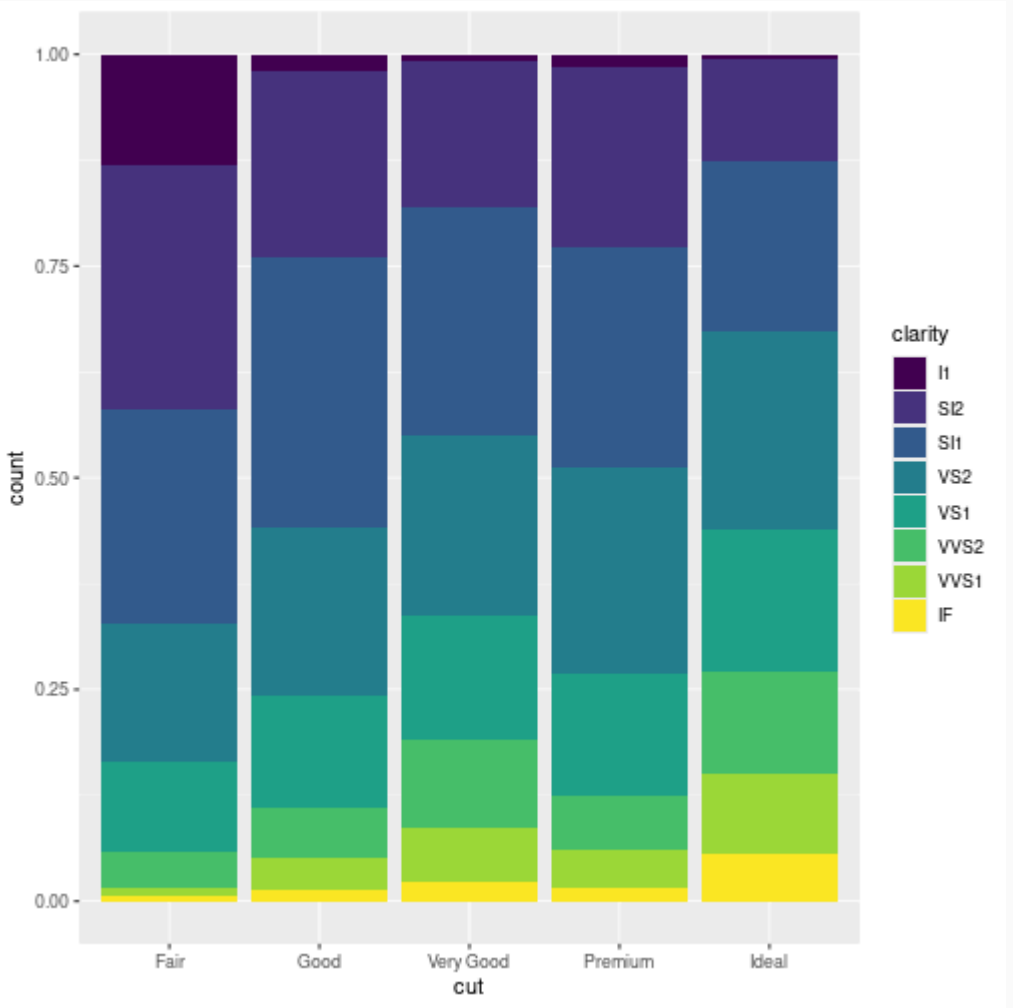
```
ggplot(diamonds, aes(x = cut, fill = clarity)) +
  geom_bar()
```



Position adjustments

- We can adjust the height of each bar to make it easier to compare proportions across groups:

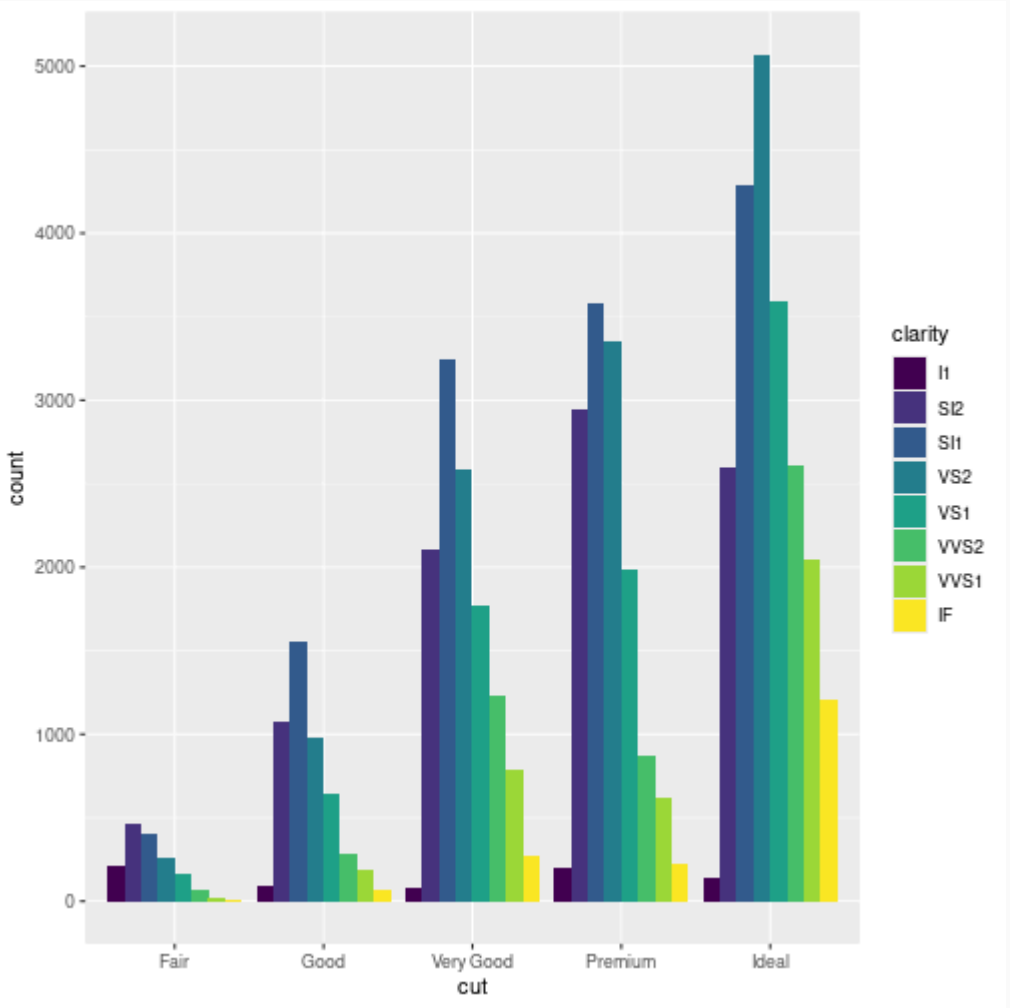
```
ggplot(diamonds, aes(x = cut, fill = clarity)) +
  geom_bar(position = "fill")
```



Position adjustments

- Lastly, we can place bars next to each other instead of stacking them:

```
ggplot(diamonds, aes(x = cut, fill = clarity)) +
  geom_bar(position = "dodge")
```



3. Data transformation

dplyr

Introducing the dplyr package

Key `dplyr` functions

- `filter()`: Pick observations by their values.
- `arrange()`: Reorder the rows.
- `select()`: Pick variables by their names.
- `mutate()`: Create new variables with functions of existing variables.
- `summarise()`: Collapse many values down to a single summary.

All verbs work similarly

1. The first argument is a data frame.
2. The subsequent arguments describe what to do with the data frame, using the variable names (without quotes).
3. The result is a new data frame.

Introducing the dplyr package

For this session, we'll use a data frame that contains all 336,776 flights that departed from New York City in 2013. Let's take a look:

```
library(tidyverse)
library(nycflights13)

flights
```

```
## # A tibble: 336,776 × 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier flight tailnum
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl> <chr>    <int> <chr>
##  1  2013     1     1      517            515         2      830            819        11 UA        1545 N14228
##  2  2013     1     1      533            529         4      850            830        20 UA        1714 N24211
##  3  2013     1     1      542            540         2      923            850        33 AA        1141 N619AA
##  4  2013     1     1      544            545        -1     1004           1022       -18 B6          725 N804JB
##  5  2013     1     1      554            600        -6      812            837       -25 DL          461 N668DN
##  6  2013     1     1      554            558        -4      740            728        12 UA        1696 N39463
##  7  2013     1     1      555            600        -5      913            854        19 B6          507 N516JB
##  8  2013     1     1      557            600        -3      709            723       -14 EV        5708 N829AS
##  9  2013     1     1      557            600        -3      838            846        -8 B6          79 N593JB
## 10  2013     1     1      558            600        -2      753            745         8 AA          301 N3ALAA
## # ℹ 336,766 more rows
## # ℹ 7 more variables: origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
## #   time_hour <dttm>
```

Filter rows with filter()

```
flights %>%
  filter(arr_delay > 120 | dep_delay > 120)
```

```
## # A tibble: 11,422 × 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier flight tailnum
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl> <chr>    <int> <chr>
##  1  2013     1     1      811            630       101     1047            830       137 MQ        4576 N531MQ
##  2  2013     1     1      848           1835       853     1001           1950       851 MQ        3944 N942MQ
##  3  2013     1     1      957            733       144     1056            853       123 UA         856 N534UA
##  4  2013     1     1     1114            900       134     1447           1222       145 UA        1086 N76502
##  5  2013     1     1     1505           1310       115     1638           1431       127 EV        4497 N17984
##  6  2013     1     1     1525           1340       105     1831           1626       125 B6         525 N231JB
##  7  2013     1     1     1540           1338       122     2020           1825       115 B6         705 N570JB
##  8  2013     1     1     1549           1445        64     1912           1656       136 EV        4181 N21197
##  9  2013     1     1     1558           1359       119     1718           1515       123 EV        5712 N826AS
## 10  2013     1     1     1732           1630        62     2028           1825       123 EV        4092 N16911
## # ℹ 11,412 more rows
## # ℹ 7 more variables: origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
## #   time_hour <dttm>
```

Arrange rows with arrange()

```
flights %>%
  arrange(desc(month), day)
```

```
## # A tibble: 336,776 × 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier flight tailnum
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl> <chr>    <int> <chr>
##  1  2013    12     1       13           2359        14      446            445         1 B6         745 N715JB
##  2  2013    12     1       17           2359        18      443            437         6 B6         839 N593JB
##  3  2013    12     1      453            500        -7      636            651       -15 US        1895 N197UW
##  4  2013    12     1      520            515         5      749            808       -19 UA        1487 N69804
##  5  2013    12     1      536            540        -4      845            850        -5 AA        2243 N634AA
##  6  2013    12     1      540            550       -10     1005           1027       -22 B6         939 N821JB
##  7  2013    12     1      541            545        -4      734            755       -21 EV        3819 N13968
##  8  2013    12     1      546            545         1      826            835        -9 UA        1441 N23708
##  9  2013    12     1      549            600       -11      648            659       -11 US        2167 N945UW
## 10  2013    12     1      550            600       -10      825            854       -29 B6         605 N706JB
## # ℹ 336,766 more rows
## # ℹ 7 more variables: origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
## #   time_hour <dttm>
```

Select columns with select()

```
flights %>%
  select(dep_time, sched_dep_time)
```

```
## # A tibble: 336,776 × 2
##    dep_time sched_dep_time
##       <int>          <int>
##  1      517            515
##  2      533            529
##  3      542            540
##  4      544            545
##  5      554            600
##  6      554            558
##  7      555            600
##  8      557            600
##  9      557            600
## 10      558            600
## # ℹ 336,766 more rows
```

Add new variables with mutate()

```
flights_sml ← flights %>%
  select(year:day, ends_with("delay"), distance, air_time) %>%
  mutate(gain = dep_delay - arr_delay,
         speed = distance / air_time * 60)
flights_sml
```

```
## # A tibble: 336,776 × 9
##     year month   day dep_delay arr_delay distance air_time  gain speed
##    <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl> <dbl> <dbl>
##  1  2013     1     1         2        11     1400      227    -9  370.
##  2  2013     1     1         4        20     1416      227   -16  374.
##  3  2013     1     1         2        33     1089      160   -31  408.
##  4  2013     1     1        -1       -18     1576      183    17  517.
##  5  2013     1     1        -6       -25      762      116    19  394.
##  6  2013     1     1        -4        12      719      150   -16  288.
##  7  2013     1     1        -5        19     1065      158   -24  404.
##  8  2013     1     1        -3       -14      229       53    11  259.
##  9  2013     1     1        -3        -8      944      140     5  405.
## 10  2013     1     1        -2         8      733      138   -10  319.
## # ℹ 336,766 more rows
```

Create new variables with transmute()

```
flights %>%
  transmute(gain = dep_delay - arr_delay,
            hours = air_time / 60,
            gain_per_hour = gain / hours)
```

```
## # A tibble: 336,776 × 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
## # ℹ 336,766 more rows
```

Grouped summaries with summarise()

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

```
## # A tibble: 365 × 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
## # ℹ 355 more rows
```

Putting it all together

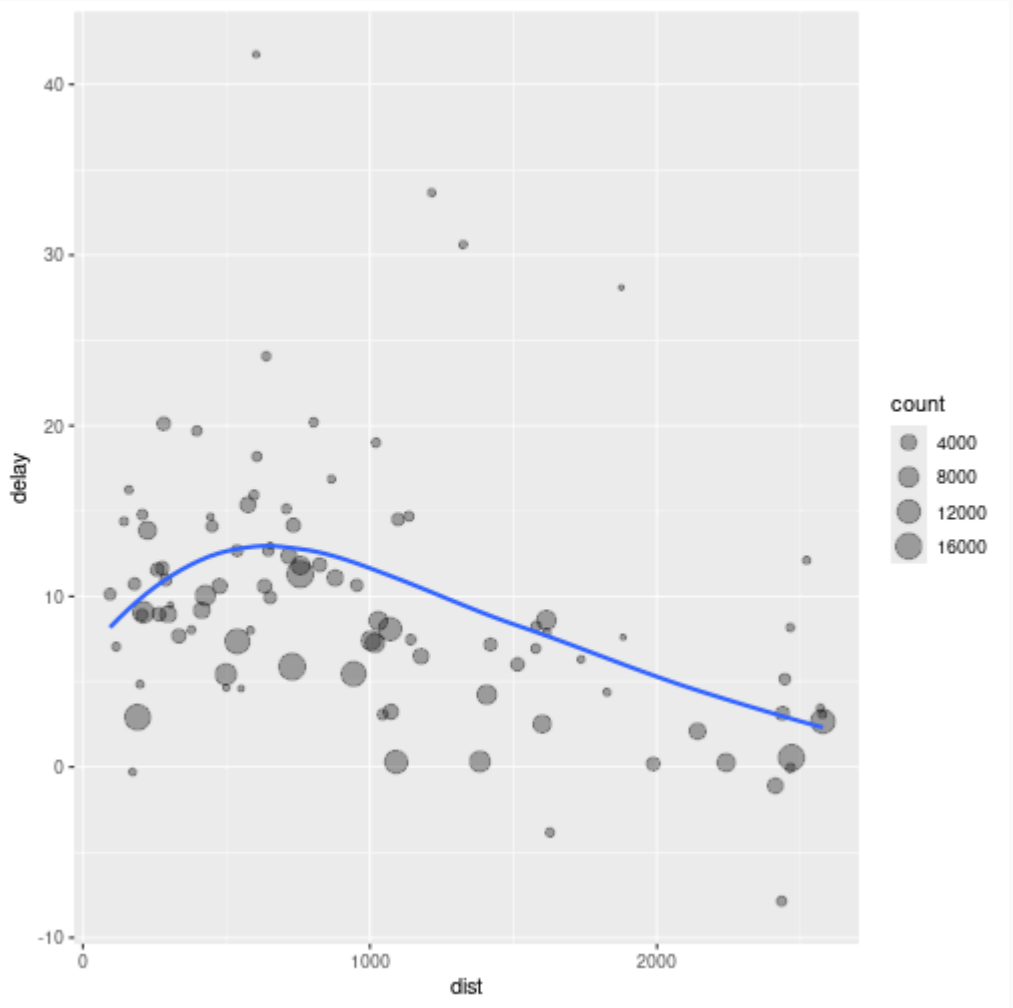
```
flights %>%
  group_by(dest) %>%
  summarise(
    count = n(),
    dist = mean(distance, na.rm = TRUE),
    delay = mean(arr_delay, na.rm = TRUE))
```

```
## # A tibble: 105 × 4
##    dest  count  dist delay
##    <chr> <int> <dbl> <dbl>
##  1 ABQ     254 1826   4.38
##  2 ACK     265  199   4.85
##  3 ALB     439  143  14.4
##  4 ANC       8 3370  -2.5
##  5 ATL   17215  757. 11.3
##  6 AUS    2439 1514.  6.02
##  7 AVL     275  584.  8.00
##  8 BDL     443  116   7.05
##  9 BGR     375  378   8.03
## 10 BHM     297  866. 16.9
## # ℹ 95 more rows
```

Putting it all together & plot!

```
flights %>%
  group_by(dest) %>%
  summarise(
    count = n(),
    dist = mean(distance, na.rm = TRUE),
    delay = mean(arr_delay, na.rm = TRUE)) %>%
  filter(count > 20, dest ≠ "HNL") %>%
  ggplot(aes(x = dist, y = delay)) +
  geom_point(aes(size = count), alpha = 1/3) +
  geom_smooth(se = FALSE)
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x
```



4. Data import:

readr, haven, & rio

readr: turning flat files into data frames

- `read_csv()`: reads comma delimited files
- `read_csv2()`: reads semicolon separated files
- `read_tsv()` reads tab delimited files
- `read_delim()` reads files with any delimiter
- `read_fwf()` reads fixed width files
- `read_table()` reads a common variation of fixed width files where columns are separated by white space

Example: 2016 American National Election Study

```
anes2016a ← read_csv(here::here("data/anes_timeseries_2016.csv"))
```

```
##
```

```
Rows: 4271 Columns: 1196
## [36m[39m Rendering ]8;;file:///home/patrick/Dropbox/Uni/teaching/BasicR/2025/slides/02-Tidyverse.Rmd[34m2025/slide
##
```

```
## [36m[39m Rendering ]8;;file:///home/patrick/Dropbox/Uni/teaching/BasicR/2025/slides/02-Tidyverse.Rmd[34m2025/slide
```

```
── Column specification ────────────────────────────────────────────────────────
## [36m[39m Rendering ]8;;file:///home/patrick/Dropbox/Uni/teaching/BasicR/2025/slides/02-Tidyverse.Rmd[34m2025/slide
```

```
Delimiter: ","
## chr    (3): version, V161010e, V163001b
## dbl (1193): V160001, V160101, V160101f, V160101w, V160102, V160102f, V160102w, V160201, V160201f, V160201w, V1 ...
##
## [36m[39m Rendering ]8;;file:///home/patrick/Dropbox/Uni/teaching/BasicR/2025/slides/02-Tidyverse.Rmd[34m2025/slide
##
```

```
##
## [36m[39m Rendering ]8;;file:///home/patrick/Dropbox/Uni/teaching/BasicR/2025/slides/02-Tidyverse.Rmd[34m2025/slide
```



```
ℹ Use `spec()` to retrieve the full column specification for this data.
```

haven: reads SPSS, Stata, and SAS files

```
library(haven)
anes2016b ← read_dta(here::here("data/anes_timeseries_2016.dta"))
anes2016b
```

```
## # A tibble: 4,271 × 1,196
##    version  V160001 V160101 V160101f V160101w V160102 V160102f V160102w V160201 V160201f V160201w V160202 V160202f
##    <chr>      <dbl>   <dbl>    <dbl>    <dbl>   <dbl>    <dbl>    <dbl>   <dbl>    <dbl>    <dbl>   <dbl>    <dbl>
##  1 ANES201…  300001   0.827    0.888        0   0.842    0.927        0     121       21        0       2        2
##  2 ANES201…  300002   1.08     1.16         0   1.01     1.08         0     123       23        0       1        1
##  3 ANES201…  300003   0.388    0.416        0   0.367    0.398        0     121       21        0       1        1
##  4 ANES201…  300004   0.360    0.385        0   0.366    0.418        0     118       18        0       1        1
##  5 ANES201…  300006   0.647    0.693        0   0.646    0.726        0     113       13        0       2        2
##  6 ANES201…  300007   0.706    0.759        0   0.688    0.725        0     104        4        0       1        1
##  7 ANES201…  300008   3.96     4.25         0   4.62     4.79         0     105        5        0       1        1
##  8 ANES201…  300012   0.962    1.03         0   0.943    1.04         0     104        4        0       2        2
##  9 ANES201…  300018   0.976    1.05         0   1.01     1.07         0     124       24        0       1        1
## 10 ANES201…  300020   0.618    0.664        0   0.600    0.638        0     121       21        0       1        1
## # ℹ 4,261 more rows
## # ℹ 1,183 more variables: V160202w <dbl>, V160501 <dbl+lbl>, V160502 <dbl+lbl>, V161001 <dbl+lbl>,
## #   V161002 <dbl+lbl>, V161003 <dbl+lbl>, V161004 <dbl+lbl>, V161005 <dbl+lbl>, V161006 <dbl+lbl>,
## #   V161007 <dbl+lbl>, V161008 <dbl+lbl>, V161009 <dbl+lbl>, V161010a <dbl+lbl>, V161010b <dbl+lbl>,
## #   V161010c <dbl+lbl>, V161010d <dbl+lbl>, V161010e <chr>, V161010f <dbl>, V161011 <dbl+lbl>,
## #   V161011x <dbl+lbl>, V161012 <dbl+lbl>, V161013a <dbl+lbl>, V161013b <dbl+lbl>, V161014 <dbl+lbl>,
## #   V161015a <dbl+lbl>, V161015b <dbl+lbl>, V161015c <dbl+lbl>, V161015d <dbl+lbl>, V161015e <dbl+lbl>,
```

rio: A Swiss-Army Knife for Data I/O

```
library(rio)
```

```
## Some optional R packages were not installed and therefore some file formats are not supported. Check file support with
```

```
anes2016c ← import(here::here("data/anes_timeseries_2016.dta"))
identical(anes2016b, anes2016c)
```

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

```
tibble(anes2016c)
```

```
## # A tibble: 4,271 × 1,196
##    version  V160001 V160101 V160101f V160101w V160102 V160102f V160102w V160201 V160201f V160201w V160202 V160202f
##    <chr>      <dbl>   <dbl>    <dbl>    <dbl>   <dbl>    <dbl>    <dbl>   <dbl>    <dbl>    <dbl>   <dbl>    <dbl>
##  1 ANES201…  300001   0.827    0.888        0   0.842    0.927        0     121       21        0       2        2
##  2 ANES201…  300002   1.08     1.16         0   1.01     1.08         0     123       23        0       1        1
##  3 ANES201…  300003   0.388    0.416        0   0.367    0.398        0     121       21        0       1        1
##  4 ANES201…  300004   0.360    0.385        0   0.366    0.418        0     118       18        0       1        1
##  5 ANES201…  300006   0.647    0.693        0   0.646    0.726        0     113       13        0       2        2
##  6 ANES201…  300007   0.706    0.759        0   0.688    0.725        0     104        4        0       1        1
##  7 ANES201…  300008   3.96     4.25         0   4.62     4.79         0     105        5        0       1        1
##  8 ANES201…  300012   0.962    1.03         0   0.943    1.04         0     104        4        0       2        2
##  9 ANES201…  300018   0.976    1.05         0   1.01     1.07         0     124       24        0       1        1
```

Other types of data

- `readxl`: reads excel files (both `.xls` and `.xlsx`).
- `DBI`: allows you to run SQL queries against a database and return a data frame

5. Tidy data

tidyr

Two representations of the same data

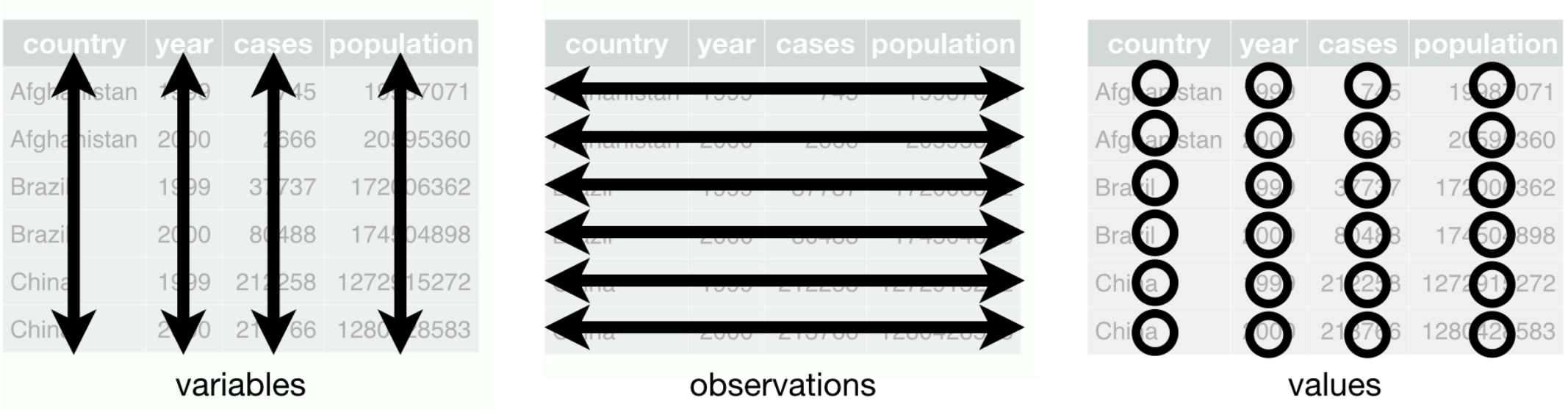
```
table1
```

```
## # A tibble: 6 × 4
##   country      year  cases population
##   <chr>       <dbl>  <dbl>      <dbl>
## 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
```

```
table2
```

```
## # A tibble: 12 × 4
##    country      year type            count
##    <chr>       <dbl> <chr>           <dbl>
##  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
```

What makes data tidy?



A lot of data you will encounter will be **untidy** because:

1. Most people aren't familiar with the principles of tidy data.
2. Data is often organized to facilitate some use other than analysis (e.g., data entry).

Case 1: Pivot to a wider data frame

- Problem: One observation is scattered across multiple rows

```
table2
```

```
## # A tibble: 12 × 4
##    country      year type            count
##    <chr>       <dbl> <chr>           <dbl>
##  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 × 4
##   country      year  cases population
##   <chr>       <dbl>  <dbl>      <dbl>
## 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
```

Case 2: Pivot to a longer data frame

- Problem: One observation is scattered across multiple columns, i.e., some of the column names are not names of variables, but values of a variable.

```
table1
```

```
## # A tibble: 6 × 4
##   country      year  cases population
##   <chr>       <dbl>  <dbl>      <dbl>
## 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
```

```
table1 %>%
  pivot_longer(-country:-year,
               names_to = "type",
               values_to = "count")
```

```
## # A tibble: 12 × 4
##    country      year type            count
##    <chr>       <dbl> <chr>           <dbl>
##  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
```