tidyverse

sourced from https://github.com/michaellevy/tidyverse_talk/blob/master/tidyverse.md

What is the tidyverse?

~~Hadleyverse~~

The tidyverse is a suite of R tools that follow a tidy philosophy:

Tidy data

Put data in data frames

- Each type of observation gets a data frame
- Each variable gets a column
- Each observation gets a row

Tidy APIs

Functions should be consistent and easily (human) readable

- Take one step at a time
- Connect simple steps with the pipe
- Referential transparency

Okay but really, what is it?

Suite of ~20 packages that provide consistent, user-friendly, smart-default tools to do most of what most people do in R.

- Core packages: ggplot2, dplyr, tidyr, readr, purrr, tibble
- Specialized data manipulation: hms, stringr, lubridate, forcats
- Data import: DBI, haven, httr, jsonlite, readxl, rvest, xml2
- Modeling: modelr, broom

`install.packages(tidyverse)` installs all of the above packages.

`library(tidyverse)` attaches only the core packages.

Why tidyverse?

- Consistency
  - e.g. All `stringr` functions take string first
  - e.g. Many functions take data.frame first -> piping
    - Faster to write
    - Easier to read
  - Tidy data: Imposes good practices
  - Type specificity
- You probably use some of it already. Synergize.
- Implements simple solutions to common problems (e.g. `purrr::transpose`)
- Smarter defaults
  - e.g. `utils::write.csv(row.names = FALSE)` = `readr::write_csv()`

- Runs fast (thanks to `Rcpp`)
- Interfaces well with other tools (e.g. Spark with `dplyr` via `sparklyr`)

tibble

A modern reimagining of data frames.

```
library(tidyverse, quietly = T)
```

```
## Warning: package 'tidyverse' was built under R version 3.6.2
```

```
## -- Attaching packages ---------------------------------------------- tidyverse 1.3.0 --
```

```
## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
```

```
## Warning: package 'tidyr' was built under R version 3.6.2
```

```
## Warning: package 'readr' was built under R version 3.6.2
```

```
## Warning: package 'purrr' was built under R version 3.6.2
```

```
## Warning: package 'dplyr' was built under R version 3.6.2
```

```
## Warning: package 'forcats' was built under R version 3.6.2
```

```
## -- Conflicts ------------------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
tdf = tibble(x = 1:1e4, y = rnorm(1e4))  # == data_frame(x = 1:1e4, y = rnorm(1e4))
class(tdf)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

Tibbles print politely.

```
tdf
```

```
## # A tibble: 10,000 x 2
##        x       y
##    <int>   <dbl>
##  1     1 -0.647
##  2     2 -1.14
##  3     3 -1.03
##  4     4  0.298
##  5     5  1.18
##  6     6 -0.686
##  7     7  0.811
##  8     8  1.59
##  9     9  0.0253
## 10    10 -1.65
## # ... with 9,990 more rows
```

- Can customize print methods with `print(tdf, n = rows, width = cols)`
- Set default with `options(tibble.print_max = rows, tibble.width = cols)`

Tibbles have some convenient and consistent defaults that are different from base R data.frames.

strings as factors

```
dfs = list(
  df = data.frame(abc = letters[1:3], xyz = letters[24:26], stringsAsFactors = FALSE),
  tbl = data_frame(abc = letters[1:3], xyz = letters[24:26])
)
```

```
## Warning: `data_frame()` is deprecated, use `tibble()`.
## This warning is displayed once per session.
```

```
sapply(dfs, function(d) class(d$abc))
```

```
##          df         tbl
## "character" "character"
```

partial matching of names

```
sapply(dfs, function(d) d$a)
```

```
## Warning: Unknown or uninitialised column: 'a'.
```

```
## $df
## [1] "a" "b" "c"
##
## $tbl
## NULL
```

type consistency

```
sapply(dfs, function(d) class(d[, "abc"]))
```

```
## $df
## [1] "character"
##
## $tbl
## [1] "tbl_df"     "tbl"        "data.frame"
```

Note that tidyverse import functions (e.g. `readr::read_csv`) default to tibbles and that *this can break existing code.*

List-columns!

```
a <- tibble(ints = 1:5,
       powers = lapply(1:5, function(x) x^(1:x)))

a[[5,2]]
```

```
## [1]    5   25  125  625 3125
```

```
a
```

```
## # A tibble: 5 x 2
##    ints powers
##   <int> <list>
## 1     1 <dbl [1]>
```

```
## 2     2 <dbl [2]>
## 3     3 <dbl [3]>
## 4     4 <dbl [4]>
## 5     5 <dbl [5]>
```

The pipe %>%

Sends the output of the LHS function to the first argument of the RHS function.

```
1:8 %>%
  sum() %>%
  sqrt()
```

```
## [1] 6
```

```
sqrt(sum(1:8))
```

```
## [1] 6
```

dplyr

Common data(frame) manipulation tasks.

Four core "verbs": filter, select, arrange, group_by + summarize, plus many more convenience functions.

```
library(ggplot2movies)
str(movies)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    58788 obs. of  24 variables:
##  $ title      : chr  "$" "$1000 a Touchdown" "$21 a Day Once a Month" "$40,000" ...
##  $ year       : int  1971 1939 1941 1996 1975 2000 2002 2002 1987 1917 ...
##  $ length     : int  121 71 7 70 71 91 93 25 97 61 ...
##  $ budget     : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ rating     : num  6.4 6 8.2 8.2 3.4 4.3 5.3 6.7 6.6 6 ...
##  $ votes      : int  348 20 5 6 17 45 200 24 18 51 ...
##  $ r1         : num  4.5 0 0 14.5 24.5 4.5 4.5 4.5 4.5 4.5 ...
##  $ r2         : num  4.5 14.5 0 0 4.5 4.5 0 4.5 4.5 0 ...
##  $ r3         : num  4.5 4.5 0 0 0 4.5 4.5 4.5 4.5 4.5 ...
##  $ r4         : num  4.5 24.5 0 0 14.5 14.5 4.5 4.5 0 4.5 ...
##  $ r5         : num  14.5 14.5 0 0 14.5 14.5 24.5 4.5 0 4.5 ...
##  $ r6         : num  24.5 14.5 24.5 0 4.5 14.5 24.5 14.5 0 44.5 ...
##  $ r7         : num  24.5 14.5 0 0 0 4.5 14.5 14.5 34.5 14.5 ...
##  $ r8         : num  14.5 4.5 44.5 0 0 4.5 4.5 14.5 14.5 4.5 ...
##  $ r9         : num  4.5 4.5 24.5 34.5 0 14.5 4.5 4.5 4.5 4.5 ...
##  $ r10        : num  4.5 14.5 24.5 45.5 24.5 14.5 14.5 14.5 24.5 4.5 ...
##  $ mpaa       : chr  "" "" "" "" ...
##  $ Action     : int  0 0 0 0 0 0 1 0 0 0 ...
##  $ Animation  : int  0 0 1 0 0 0 0 0 0 0 ...
##  $ Comedy     : int  1 1 0 1 0 0 0 0 0 0 ...
##  $ Drama      : int  1 0 0 0 0 1 1 0 1 0 ...
##  $ Documentary: int  0 0 0 0 0 0 0 1 0 0 ...
##  $ Romance    : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Short      : int  0 0 1 0 0 0 0 1 0 0 ...
```

```
filter(movies, length > 360)
```

```
## # A tibble: 21 x 24
##    title  year length  budget rating votes    r1    r2    r3    r4    r5    r6
##    <chr> <int>  <int>   <int>  <dbl> <int> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 Comm~  2000    555      NA    7.8    33   0     4.5   4.5   0     0     0
##  2 Cure~  1987   5220      NA    3.8    59  44.5   4.5   4.5   4.5   0     0
##  3 Ebol~  2004    647      NA    8.4     5   0     0     0     0     0     0
##  4 Empi~  1964    485      NA    5.5    46  34.5   0     0     4.5   4.5   4.5
##  5 Farm~  1998    390      NA    8.5    52   0     4.5   0     0     4.5   0
##  6 Fool~  1922    384 1100000    7.6   191   0     4.5   4.5   4.5   4.5   4.5
##  7 Four~  1967   1100      NA    3      12  24.5   0     4.5   0     0     0
##  8 Hitl~  1978    407      NA    9      70   4.5   0     0     0     4.5   4.5
##  9 Imit~  1967    480      NA    4.4     5  44.5   0     0     0    44.5   0
## 10 Long~  1970   2880      NA    6.4    15  44.5   0     0     0     0     0
## # ... with 11 more rows, and 12 more variables: r7 <dbl>, r8 <dbl>, r9 <dbl>,
## #   r10 <dbl>, mpaa <chr>, Action <int>, Animation <int>, Comedy <int>,
## #   Drama <int>, Documentary <int>, Romance <int>, Short <int>
```

```
movies %>%
  filter(length > 360)
```

```
## # A tibble: 21 x 24
##    title  year length  budget rating votes    r1    r2    r3    r4    r5    r6
##    <chr> <int>  <int>   <int>  <dbl> <int> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 Comm~  2000    555      NA    7.8    33   0     4.5   4.5   0     0     0
##  2 Cure~  1987   5220      NA    3.8    59  44.5   4.5   4.5   4.5   0     0
##  3 Ebol~  2004    647      NA    8.4     5   0     0     0     0     0     0
##  4 Empi~  1964    485      NA    5.5    46  34.5   0     0     4.5   4.5   4.5
##  5 Farm~  1998    390      NA    8.5    52   0     4.5   0     0     4.5   0
##  6 Fool~  1922    384 1100000    7.6   191   0     4.5   4.5   4.5   4.5   4.5
##  7 Four~  1967   1100      NA    3      12  24.5   0     4.5   0     0     0
##  8 Hitl~  1978    407      NA    9      70   4.5   0     0     0     4.5   4.5
##  9 Imit~  1967    480      NA    4.4     5  44.5   0     0     0    44.5   0
## 10 Long~  1970   2880      NA    6.4    15  44.5   0     0     0     0     0
## # ... with 11 more rows, and 12 more variables: r7 <dbl>, r8 <dbl>, r9 <dbl>,
## #   r10 <dbl>, mpaa <chr>, Action <int>, Animation <int>, Comedy <int>,
## #   Drama <int>, Documentary <int>, Romance <int>, Short <int>
```

```
movies %>%
  filter(length > 360) %>%
  select(title, rating, votes)
```

```
## # A tibble: 21 x 3
##    title                                          rating votes
##    <chr>                                           <dbl> <int>
##  1 Commune (Paris, 1871), La                         7.8    33
##  2 Cure for Insomnia, The                            3.8    59
##  3 Ebolusyon ng isang pamilyang pilipino             8.4     5
##  4 Empire                                            5.5    46
##  5 Farmer's Wife, The                                8.5    52
##  6 Foolish Wives                                     7.6   191
##  7 Four Stars                                        3      12
##  8 Hitler - ein Film aus Deutschland                 9      70
##  9 Imitation of Christ                               4.4     5
```

```
## 10 Longest Most Meaningless Movie in the World, The    6.4    15
## # ... with 11 more rows
```

```
movies %>%
  filter(Animation == 1, votes > 1000) %>%
  select(title, rating) %>%
  arrange(desc(rating)) -> a
```

`summarize` makes `aggregate` and `tapply` functionality easier, and the output is always a data frame.

```
movies %>%
  filter(mpaa != "") %>%
  group_by(year, mpaa) %>%
  summarize(avg_budget = mean(budget, na.rm = TRUE),
            avg_rating = mean(rating, na.rm = TRUE)) %>%
  arrange(desc(year), mpaa) -> b


cbind
```

```
## function (..., deparse.level = 1)
## .Internal(cbind(deparse.level, ...))
## <bytecode: 0x000000000bd03ec0>
## <environment: namespace:base>
```

`count` for frequency tables. Note the consistent API and easy readability vs. `table`.

```
filter(movies, mpaa != "") %>%
  count(year, mpaa, Animation, sort = TRUE)
```

```
## # A tibble: 156 x 4
##     year mpaa  Animation     n
##    <int> <chr>     <int> <int>
##  1  1999 R             0   366
##  2  2001 R             0   355
##  3  2002 R             0   343
##  4  2000 R             0   341
##  5  1998 R             0   335
##  6  1997 R             0   325
##  7  1996 R             0   310
##  8  1995 R             0   293
##  9  2003 R             0   264
## 10  2004 R             0   196
## # ... with 146 more rows
```

```
basetab = with(movies[movies$mpaa != "", ], table(year, mpaa, Animation))
basetab[1:5, , ]
```

```
## , , Animation = 0
##
##       mpaa
## year   NC-17 PG PG-13 R
##   1934     0  1     0 0
##   1938     0  1     0 0
##   1945     0  0     1 0
##   1946     0  1     0 0
##   1951     0  2     0 0
##
```

```
## , , Animation = 1
##
##       mpaa
## year   NC-17 PG PG-13 R
##   1934     0  0     0 0
##   1938     0  0     0 0
##   1945     0  0     0 0
##   1946     0  0     0 0
##   1951     0  0     0 0
```

joins

dplyr also does multi-table joins and can connect to various types of databases.

```
t1 = data_frame(alpha = letters[1:6], num = 1:6)
t2 = data_frame(alpha = letters[4:10], num = 4:10)
t3 <- full_join(t1, t2, by = "alpha", suffix = c("_t1", "_t2"))

t3
```

```
## # A tibble: 10 x 3
##    alpha num_t1 num_t2
##    <chr>  <int>  <int>
##  1 a          1     NA
##  2 b          2     NA
##  3 c          3     NA
##  4 d          4      4
##  5 e          5      5
##  6 f          6      6
##  7 g         NA      7
##  8 h         NA      8
##  9 i         NA      9
## 10 j         NA     10
```

```
t3 %>% mutate(tot = ifelse(is.na(num_t1),0,num_t1) + ifelse(is.na(num_t2),0,num_t2))
```

```
## # A tibble: 10 x 4
##    alpha num_t1 num_t2   tot
##    <chr>  <int>  <int> <dbl>
##  1 a          1     NA     1
##  2 b          2     NA     2
##  3 c          3     NA     3
##  4 d          4      4     8
##  5 e          5      5    10
##  6 f          6      6    12
##  7 g         NA      7     7
##  8 h         NA      8     8
##  9 i         NA      9     9
## 10 j         NA     10    10
```

Super-secret pro-tip: You can `group_by` %>% `mutate` to accomplish a summarize + join

```
data <- data_frame(group = sample(letters[1:3], 10, replace = TRUE),
           value = rnorm(10))
data
```

```
## # A tibble: 10 x 2
##    group   value
##    <chr>   <dbl>
##  1 a     -0.697
##  2 c     -1.11
##  3 a      0.0278
##  4 a     -0.281
##  5 b     -1.48
##  6 c     -1.18
##  7 a     -0.440
##  8 b     -0.525
##  9 b     -0.379
## 10 c      1.67
```

```
data %>%
  group_by(group) %>%
  mutate(group_average = mean(value))
```

```
## # A tibble: 10 x 3
## # Groups:   group [3]
##    group   value group_average
##    <chr>   <dbl>         <dbl>
##  1 a     -0.697         -0.348
##  2 c     -1.11          -0.209
##  3 a      0.0278        -0.348
##  4 a     -0.281         -0.348
##  5 b     -1.48          -0.796
##  6 c     -1.18          -0.209
##  7 a     -0.440         -0.348
##  8 b     -0.525         -0.796
##  9 b     -0.379         -0.796
## 10 c      1.67          -0.209
```

```
#the alternative is a 2-step process

averages <- data %>% group_by(group) %>% summarize(group_average = mean(value))
data %>% inner_join(averages)
```

```
## Joining, by = "group"
```

```
## # A tibble: 10 x 3
##    group   value group_average
##    <chr>   <dbl>         <dbl>
##  1 a     -0.697         -0.348
##  2 c     -1.11          -0.209
##  3 a      0.0278        -0.348
##  4 a     -0.281         -0.348
##  5 b     -1.48          -0.796
##  6 c     -1.18          -0.209
##  7 a     -0.440         -0.348
##  8 b     -0.525         -0.796
##  9 b     -0.379         -0.796
## 10 c      1.67          -0.209
```

tidyr

Latest generation of `reshape`. `gather` to make wide table long, `spread` to make long tables wide.

We'll use a Tuberculosis dataset from the World Health Organization. This dataset used to be available in the base install of R, but we'll have to get it from an external source. Fortunately, the tidyverse gives us the tools to do that easily.

This data is freely availble at https://www.who.int/tb/country/data/download/en/ ,and a data dictionary can be found at https://extranet.who.int/tme/generateCSV.asp?ds=dictionary

```
who <- read_csv('https://extranet.who.int/tme/generateCSV.asp?ds=notifications')
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   country = col_character(),
##   iso2 = col_character(),
##   iso3 = col_character(),
##   iso_numeric = col_character(),
##   g_whoregion = col_character(),
##   new_sn_sexunk04 = col_logical(),
##   new_sn_sexunk514 = col_logical(),
##   new_sn_sexunk014 = col_logical(),
##   new_sn_sexunk15plus = col_logical(),
##   new_ep_m04 = col_logical(),
##   new_ep_sexunkageunk = col_logical(),
##   rdxsurvey_newinc = col_logical(),
##   rdxsurvey_newinc_rdx = col_logical()
## )
```

```
## See spec(...) for full column specifications.
```

```
## Warning: 105 parsing failures.
##  row                 col           expected actual
## 1250 new_sn_sexunk04     1/0/T/F/TRUE/FALSE    22  'https://extranet.who.int/tme/generateCSV.asp?ds=
## 1250 new_sn_sexunk514    1/0/T/F/TRUE/FALSE    33  'https://extranet.who.int/tme/generateCSV.asp?ds=
## 1250 new_sn_sexunk014    1/0/T/F/TRUE/FALSE    55  'https://extranet.who.int/tme/generateCSV.asp?ds=
## 1250 new_sn_sexunk15plus 1/0/T/F/TRUE/FALSE    632 'https://extranet.who.int/tme/generateCSV.asp?ds=
## 1251 new_sn_sexunk04     1/0/T/F/TRUE/FALSE    23  'https://extranet.who.int/tme/generateCSV.asp?ds=
## .... ................... .................. ...... ........................................
## See problems(...) for more details.
```

```
#let's try that again, but give the proper column types

col_types <- cols(
  .default = col_double(),
  country = col_character(),
  iso2 = col_character(),
  iso3 = col_character(),
  iso_numeric = col_character(),
  g_whoregion = col_character(),
  new_sn_sexunk04 = col_double(),
  new_sn_sexunk514 = col_double(),
  new_sn_sexunk014 = col_double(),
  new_sn_sexunk15plus = col_double(),
  new_ep_m04 = col_double(),
```

```
  new_ep_sexunkageunk = col_double(),
  rdxsurvey_newinc = col_double(),
  rdxsurvey_newinc_rdx = col_double(),
  hiv_ipt_reg_all = col_double(),
  hiv_tbdetect = col_double()
)

who <- read_csv('https://extranet.who.int/tme/generateCSV.asp?ds=notifications', col_types = col_types)
who <- who[,c(1:3, 6, 27:33, 37:43, 47:53, 58:64, 73:79, 84:90)]

str(who)# Tuberculosis data from the WHO
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    8286 obs. of  46 variables:
##  $ country     : chr  "Afghanistan" "Afghanistan" "Afghanistan" "Afghanistan" ...
##  $ iso2        : chr  "AF" "AF" "AF" "AF" ...
##  $ iso3        : chr  "AFG" "AFG" "AFG" "AFG" ...
##  $ year        : num  1980 1981 1982 1983 1984 ...
##  $ new_sp_m014 : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_sp_m1524: num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_sp_m2534: num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_sp_m3544: num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_sp_m4554: num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_sp_m5564: num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_sp_m65  : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_sp_f014 : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_sp_f1524: num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_sp_f2534: num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_sp_f3544: num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_sp_f4554: num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_sp_f5564: num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_sp_f65  : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_sn_m014 : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_sn_m1524: num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_sn_m2534: num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_sn_m3544: num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_sn_m4554: num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_sn_m5564: num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_sn_m65  : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_sn_f014 : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_sn_f1524: num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_sn_f2534: num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_sn_f3544: num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_sn_f4554: num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_sn_f5564: num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_sn_f65  : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_ep_m014 : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_ep_m1524: num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_ep_m2534: num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_ep_m3544: num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_ep_m4554: num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_ep_m5564: num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_ep_m65  : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_ep_f014 : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_ep_f1524: num  NA NA NA NA NA NA NA NA NA NA ...
```

```
##  $ new_ep_f2534: num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_ep_f3544: num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_ep_f4554: num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_ep_f5564: num  NA NA NA NA NA NA NA NA NA NA ...
##  $ new_ep_f65  : num  NA NA NA NA NA NA NA NA NA NA ...
```

```
who %>%
  gather(group, cases, -country, -iso2, -iso3, -year)
```

```
## # A tibble: 348,012 x 6
##    country     iso2  iso3   year group       cases
##    <chr>       <chr> <chr> <dbl> <chr>       <dbl>
##  1 Afghanistan AF    AFG    1980 new_sp_m014    NA
##  2 Afghanistan AF    AFG    1981 new_sp_m014    NA
##  3 Afghanistan AF    AFG    1982 new_sp_m014    NA
##  4 Afghanistan AF    AFG    1983 new_sp_m014    NA
##  5 Afghanistan AF    AFG    1984 new_sp_m014    NA
##  6 Afghanistan AF    AFG    1985 new_sp_m014    NA
##  7 Afghanistan AF    AFG    1986 new_sp_m014    NA
##  8 Afghanistan AF    AFG    1987 new_sp_m014    NA
##  9 Afghanistan AF    AFG    1988 new_sp_m014    NA
## 10 Afghanistan AF    AFG    1989 new_sp_m014    NA
## # ... with 348,002 more rows
```

```
who %>%
  gather(group, cases, -country, -iso2, -iso3, -year) %>%
  filter(!is.na(cases))
```

```
## # A tibble: 73,445 x 6
##    country     iso2  iso3   year group       cases
##    <chr>       <chr> <chr> <dbl> <chr>       <dbl>
##  1 Afghanistan AF    AFG    1997 new_sp_m014     0
##  2 Afghanistan AF    AFG    1998 new_sp_m014    30
##  3 Afghanistan AF    AFG    1999 new_sp_m014     8
##  4 Afghanistan AF    AFG    2000 new_sp_m014    52
##  5 Afghanistan AF    AFG    2001 new_sp_m014   129
##  6 Afghanistan AF    AFG    2002 new_sp_m014    90
##  7 Afghanistan AF    AFG    2003 new_sp_m014   127
##  8 Afghanistan AF    AFG    2004 new_sp_m014   139
##  9 Afghanistan AF    AFG    2005 new_sp_m014   151
## 10 Afghanistan AF    AFG    2006 new_sp_m014   193
## # ... with 73,435 more rows
```

ggplot2

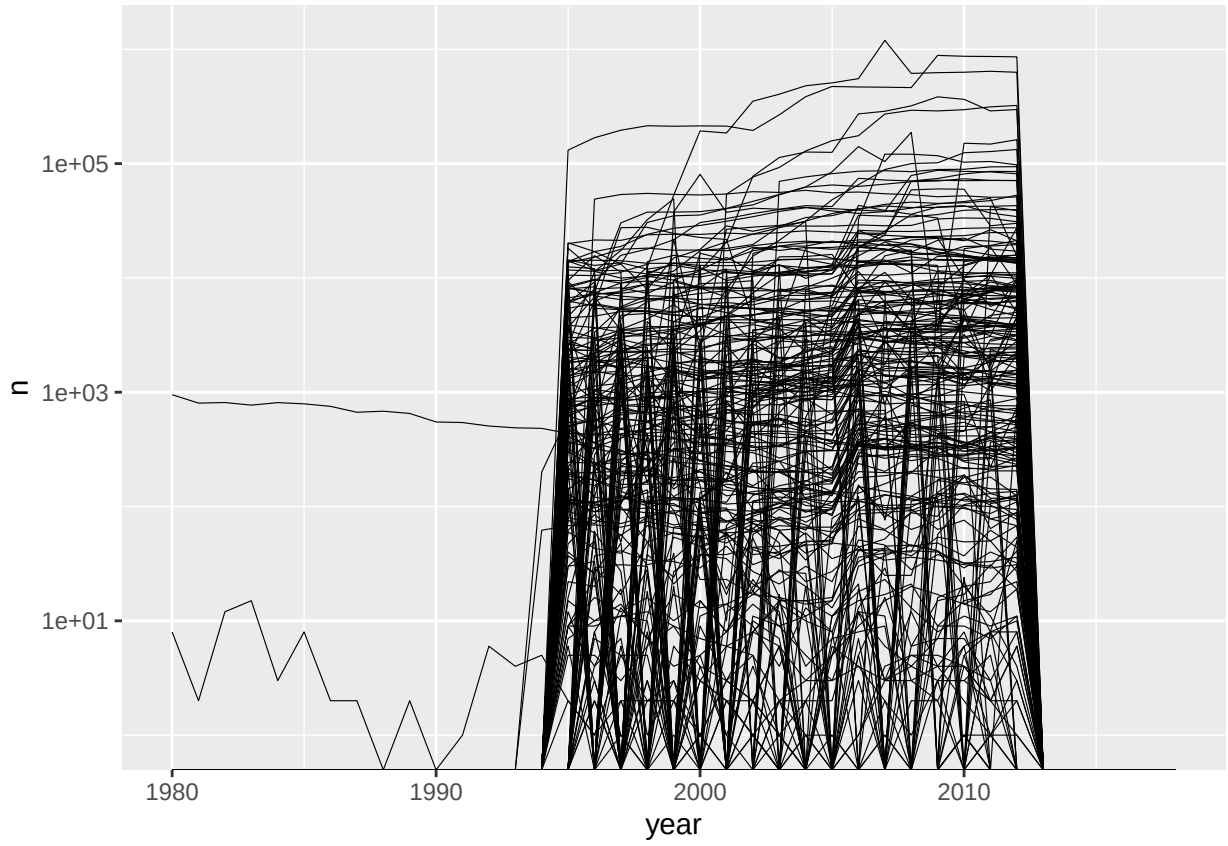
If you don't already know and love it, check out one of our previous talks on ggplot or any of the excellent resources on the internet.

Note that the pipe and consistent API make it easy to combine functions from different packages, and the whole thing is quite readable.

```
who %>%
  select(-iso2, -iso3) %>%
  gather(group, cases, -country, -year) %>%
  count(country, year, wt = cases) %>%
```

```
  ggplot(aes(x = year, y = n, group = country)) +
  geom_line(size = .2) + scale_y_log10()
```

```
## Warning: Transformation introduced infinite values in continuous y-axis
```



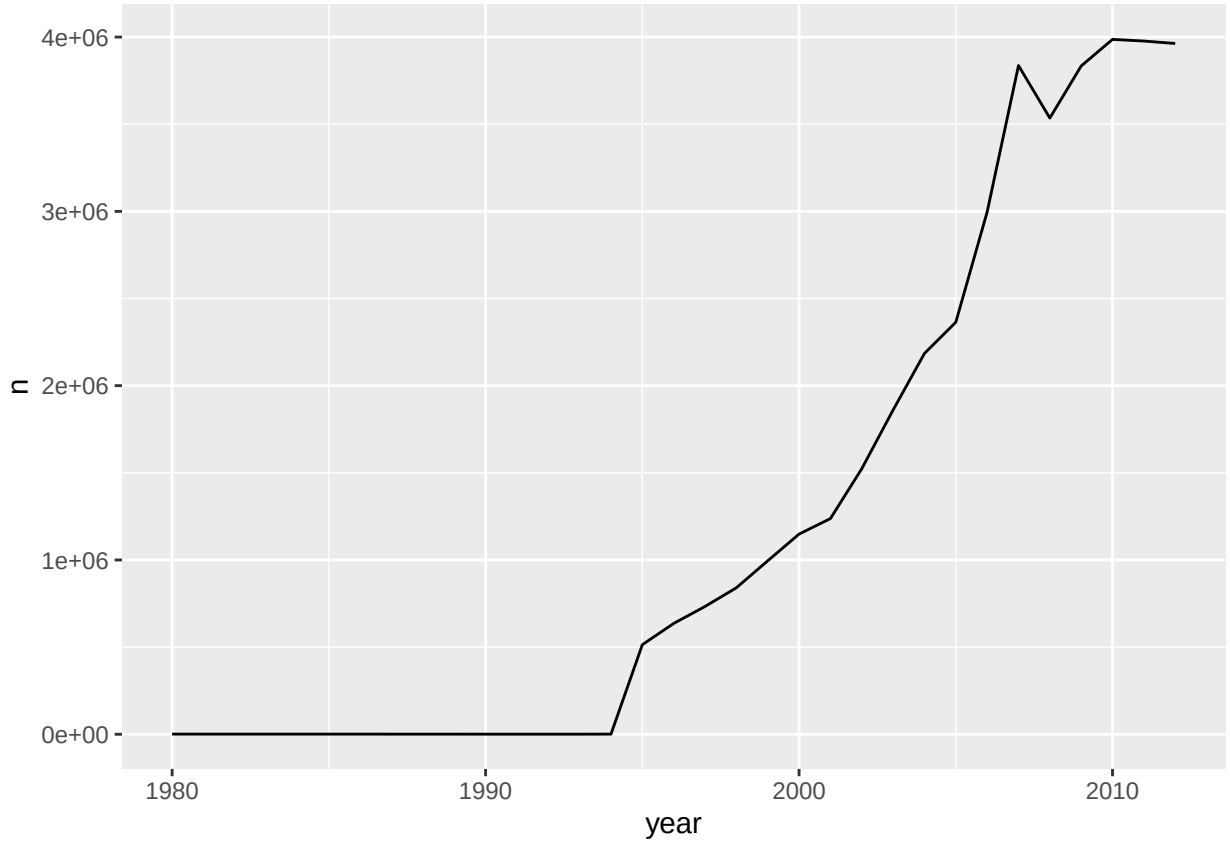

```
who.summary <- who %>%
  select(-iso2) %>%
  gather(group, cases, -country, -iso3, -year) %>%
  filter(!is.na(cases)) %>%
  count(year, wt = cases)

who.summary
```

```
## # A tibble: 33 x 2
##     year     n
##    <dbl> <dbl>
##  1  1980   959
##  2  1981   805
##  3  1982   824
##  4  1983   786
##  5  1984   814
##  6  1985   799
##  7  1986   754
##  8  1987   670
##  9  1988   682
## 10  1989   654
```

```
## # ... with 23 more rows
```

```
who.summary %>%
  ggplot(aes(x=year, y=n)) + geom_line()
```



readr

For reading flat files. Faster than base with smarter defaults.

```
bigdf = data_frame(int = 1:1e6,
                   squares = int^2,
                   letters = sample(letters, 1e6, replace = TRUE))
bigdf
```

```
## # A tibble: 1,000,000 x 3
##      int squares letters
##    <int>   <dbl> <chr>
##  1     1       1 y
##  2     2       4 d
##  3     3       9 h
##  4     4      16 s
##  5     5      25 z
##  6     6      36 j
##  7     7      49 b
##  8     8      64 k
##  9     9      81 n
```

```
## 10    10     100 f
## # ... with 999,990 more rows
```

```
system.time(
  write.csv(bigdf, "base-write.csv")
)
```

```
##    user  system elapsed
##    4.97    0.14    5.18
```

```
system.time(
  write_csv(bigdf, "readr-write.csv")
)
```

```
##    user  system elapsed
##    0.59    0.11    0.72
```

```
read.csv("base-write.csv", nrows = 3)
```

```
##   X int squares letters
## 1 1   1       1       y
## 2 2   2       4       d
## 3 3   3       9       h
```

```
read_csv("readr-write.csv", n_max = 3)
```

```
## Parsed with column specification:
## cols(
##   int = col_double(),
##   squares = col_double(),
##   letters = col_character()
## )
```

```
## # A tibble: 3 x 3
##     int squares letters
##   <dbl>   <dbl> <chr>
## 1     1       1 y
## 2     2       4 d
## 3     3       9 h
```

broom

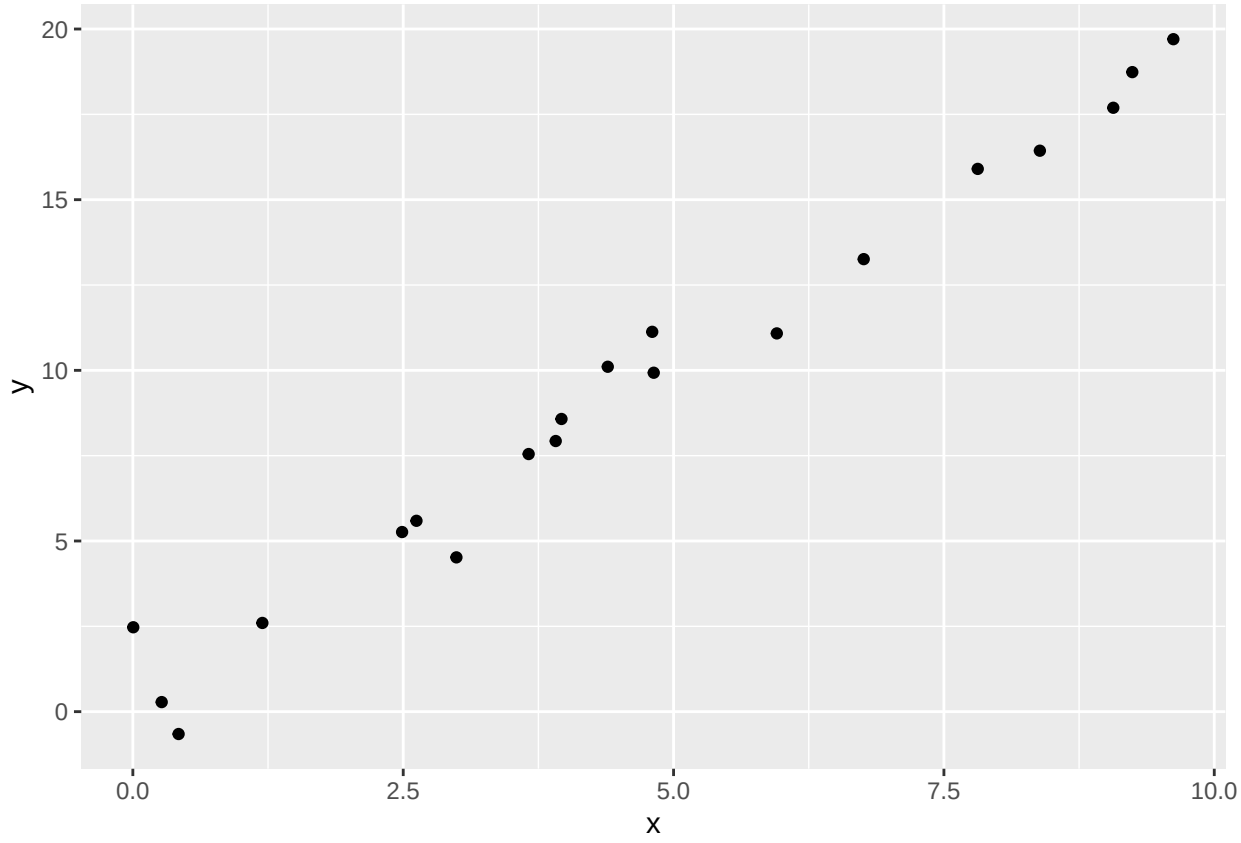
`broom` is a convenient little package to work with model results. Two functions I find useful are `tidy` to extract model results and `augment` to add residuals, predictions, etc. to a data.frame.

```
d = data_frame(x = runif(20, 0, 10),
               y = 2 * x + rnorm(20))
d
```

```
## # A tibble: 20 x 2
##          x      y
##      <dbl>  <dbl>
##  1 1.20     2.60
##  2 3.66     7.55
##  3 3.96     8.58
##  4 3.91     7.93
##  5 0.267    0.280
##  6 4.39    10.1
```

```
##  7 7.81    15.9
##  8 0.00381  2.47
##  9 4.80    11.1
## 10 2.99     4.52
## 11 5.95    11.1
## 12 9.62    19.7
## 13 2.62     5.59
## 14 8.39    16.4
## 15 9.07    17.7
## 16 6.76    13.3
## 17 2.49     5.26
## 18 4.82     9.93
## 19 9.24    18.7
## 20 0.423   -0.655
```

```
qplot(x, y, data = d)
```



tidy

```
library(broom)  # Not attached with tidyverse
```

```
## Warning: package 'broom' was built under R version 3.6.2
```

```
model <- lm(y ~ x, d)
summary(model)
```

```
##
## Call:
## lm(formula = y ~ x, data = d)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.77969 -0.42534  0.04753  0.26450  2.17425
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.29043    0.38899   0.747    0.465
## x            1.97315    0.07065  27.930 2.83e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9469 on 18 degrees of freedom
## Multiple R-squared:  0.9774, Adjusted R-squared:  0.9762
## F-statistic: 780.1 on 1 and 18 DF,  p-value: 2.829e-16
```

```
tidy(model)
```

```
## # A tibble: 2 x 5
##   term        estimate std.error statistic  p.value
##   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)    0.290    0.389      0.747 4.65e- 1
## 2 x              1.97     0.0706    27.9   2.83e-16
```

augment

i.e. The function formerly known as `fortify`.

```
aug = augment(model)
aug
```

```
## # A tibble: 20 x 9
##         y       x .fitted .se.fit   .resid    .hat .sigma    .cooksd .std.resid
##     <dbl>   <dbl>   <dbl>   <dbl>    <dbl>  <dbl>  <dbl>      <dbl>      <dbl>
##  1  2.60  1.20      2.65    0.321 -0.0547 0.115    0.974 0.000246     -0.0615
##  2  7.55  3.66      7.51    0.222  0.0356 0.0551   0.974 0.0000437     0.0387
##  3  8.58  3.96      8.11    0.217  0.464  0.0524   0.967 0.00702       0.504
##  4  7.93  3.91      8.01    0.218 -0.0768 0.0528   0.974 0.000193     -0.0833
##  5  0.280 0.267     0.816   0.373 -0.536  0.155    0.964 0.0349       -0.616
##  6 10.1   4.39      8.96    0.212  1.15   0.0503   0.931 0.0411        1.25
##  7 15.9   7.81     15.7     0.309  0.195  0.107    0.973 0.00285       0.218
##  8  2.47  0.00381   0.298   0.389  2.17   0.169    0.784 0.643         2.52
##  9 11.1   4.80      9.77    0.212  1.36   0.0502   0.914 0.0575        1.48
## 10  4.52  2.99      6.19    0.241 -1.67   0.0647   0.879 0.116        -1.83
## 11 11.1   5.95     12.0     0.232 -0.958  0.0599   0.944 0.0347       -1.04
## 12 19.7   9.62     19.3     0.412  0.426  0.189    0.968 0.0291        0.499
## 13  5.59  2.62      5.46    0.254  0.127  0.0722   0.974 0.000753      0.139
## 14 16.4   8.39     16.8     0.340 -0.405  0.129    0.969 0.0155       -0.458
## 15 17.7   9.07     18.2     0.379 -0.487  0.160    0.966 0.0300       -0.561
## 16 13.3   6.76     13.6     0.260 -0.368  0.0755   0.970 0.00667      -0.404
## 17  5.26  2.49      5.20    0.260  0.0594 0.0752   0.974 0.000173      0.0653
## 18  9.93  4.82      9.79    0.212  0.136  0.0502   0.974 0.000570      0.147
```

```
## 19 18.7   9.24     18.5     0.389  0.211  0.169   0.973 0.00606       0.244
## 20 -0.655 0.423     1.13    0.364 -1.78   0.148   0.855 0.360        -2.04
```

purrr

`purrr` is kind of like `dplyr` for lists. It helps you repeatedly apply functions. Like the rest of the tidyverse, nothing you can't do in base R, but `purrr` makes the API consistent, encourages type specificity, and provides some nice shortcuts and speed ups.

```
df = data_frame(fun = rep(c(lapply, map), 2),
                n = rep(c(1e5, 1e7), each = 2),
                comp_time = map2(fun, n, ~system.time(.x(1:.y, sqrt))))
df$comp_time
```

```
## [[1]]
##    user  system elapsed
##    0.07    0.00    0.06
##
## [[2]]
##    user  system elapsed
##    0.15    0.00    0.15
##
## [[3]]
##    user  system elapsed
##    9.92    0.19   10.31
##
## [[4]]
##    user  system elapsed
##   13.69    0.19   14.20
```

map

Vanilla `map` is a slightly improved version of `lapply`. Do a function on each item in a list.

```
map(1:4, log)
```

```
## [[1]]
## [1] 0
##
## [[2]]
## [1] 0.6931472
##
## [[3]]
## [1] 1.098612
##
## [[4]]
## [1] 1.386294
```

Can supply additional arguments as with `(x)apply`

```
map(1:4, log, base = 2)
```

```
## [[1]]
## [1] 0
##
```

```
## [[2]]
## [1] 1
##
## [[3]]
## [1] 1.584963
##
## [[4]]
## [1] 2
```

Can compose anonymous functions like `(x)apply`, either the old way or with a new formula shorthand.

```
map(1:4, ~ log(4, base = .x))  # == map(1:4, function(x) log(4, base = x))
```

```
## [[1]]
## [1] Inf
##
## [[2]]
## [1] 2
##
## [[3]]
## [1] 1.26186
##
## [[4]]
## [1] 1
```

`map` always returns a list. `map_xxx` type-specifies the output type and simplifies the list to a vector.

```
map_dbl(1:4, log, base = 2)
```

```
## [1] 0.000000 1.000000 1.584963 2.000000
```

And throws an error if any output isn't of the expected type (which is a good thing!).

```
#map_int(1:4, log, base = 2)

   ## Error: Can't coerce element 1 from a double to a integer
```

`map2` is like `mapply` – apply a function over two lists in parallel. `map_n` generalizes to any number of lists.

```
fwd = 1:10
bck = 10:1
map2_dbl(fwd, bck, `^`)
```

```
##  [1]     1   512  6561 16384 15625  7776  2401   512    81    10
```

`map_if` tests each element on a function and if true applies the second function, if false returns the original element.

```
data_frame(ints = 1:5, lets = letters[1:5], sqrts = ints^.5) %>%
  map_if(is.numeric, ~ .x^2)
```

```
## $ints
## [1]  1  4  9 16 25
##
## $lets
## [1] "a" "b" "c" "d" "e"
##
## $sqrts
## [1] 1 2 3 4 5
```

Putting map to work

Split the movies data frame by mpaa rating, fit a linear model to each data frame, and organize the model results in a data frame.

```
movies %>%
  filter(mpaa != "") %>%
  split(.$mpaa) %>%
  map(~ lm(rating ~ budget, data = .)) %>%
  map_df(tidy, .id = "mpaa-rating") %>%
  arrange(term)
```

```
## # A tibble: 8 x 6
##   `mpaa-rating` term        estimate     std.error statistic   p.value
##   <chr>         <chr>          <dbl>         <dbl>     <dbl>     <dbl>
## 1 NC-17         (Intercept)  6.51e+0 0.312            20.8   4.73e-  6
## 2 PG            (Intercept)  5.77e+0 0.137            42.2   1.86e-104
## 3 PG-13         (Intercept)  5.75e+0 0.0781           73.6   8.29e-280
## 4 R             (Intercept)  5.81e+0 0.0524          111.    0.
## 5 NC-17         budget      -6.05e-8 0.0000000184     -3.29  2.16e-  2
## 6 PG            budget       2.03e-9 0.00000000275     0.739 4.61e-  1
## 7 PG-13         budget       3.49e-9 0.00000000144     2.42  1.59e-  2
## 8 R             budget       7.73e-9 0.00000000166     4.66  3.54e-  6
```

List-columns make it easier to organize complex datasets. Can `map` over list-columns right in `data_frame/tibble` creation. And if you later want to calculate something else, everything is nicely organized in the data frame.

```
d =
  data_frame(
    dist = c("normal", "poisson", "chi-square"),
    funs = list(rnorm, rpois, rchisq),
    samples = map(funs, ~.(100, 5)),
    mean = map_dbl(samples, mean),
    var = map_dbl(samples, var)
  )
d$median = map_dbl(d$samples, median)
d
```

```
## # A tibble: 3 x 6
##   dist       funs   samples      mean   var median
##   <chr>      <list> <list>      <dbl> <dbl>  <dbl>
## 1 normal     <fn>   <dbl [100]>  5.01  1.08   5.05
## 2 poisson    <fn>   <int [100]>  5.06  5.45   5
## 3 chi-square <fn>   <dbl [100]>  4.74  8.08   4.42
```

Let's see if we can really make this purrr... Fit a linear model of diamond price by every combination of two predictors in the dataset and see which two predict best.

```
train = sample(nrow(diamonds), floor(nrow(diamonds) * .67))
setdiff(names(diamonds), "price") %>%
  combn(2, paste, collapse = " + ") %>%
  structure(., names = .) %>%
  map(~ formula(paste("price ~ ", .x))) %>%
  map(lm, data = diamonds[train, ]) %>%
  map_df(augment, newdata = diamonds[-train, ], .id = "predictors") %>%
  group_by(predictors) %>%
```

```
  summarize(rmse = sqrt(mean((price - .fitted)^2))) %>%
  arrange(rmse)
```

```
## # A tibble: 36 x 2
##    predictors       rmse
##    <chr>           <dbl>
##  1 carat + clarity 1292.
##  2 carat + color   1484.
##  3 carat + cut     1517.
##  4 carat + z       1539.
##  5 carat + x       1543.
##  6 carat + table   1550.
##  7 carat + y       1550.
##  8 carat + depth   1551.
##  9 clarity + x     1655.
## 10 clarity + y     1666.
## # ... with 26 more rows
```

Type-stability

We have seen that we can use map_lgl to ensure we get a logical vector, map_chr to ensure we get a character vector back, etc. Type stability is like a little built-in unit test. You make sure you're getting what you think you are, even in the middle of a pipeline or function. Here are two more type-stable function implemented in `purrr`.

`flatten`

Like `unlist` but can specify output type, and never recurses.

```
map(-1:3, ~.x ^ seq(-.5, .5, .5)) %>%
  flatten_dbl()
```

```
##  [1]       NaN 1.0000000       NaN       Inf 1.0000000 0.0000000 1.0000000
##  [8] 1.0000000 1.0000000 0.7071068 1.0000000 1.4142136 0.5773503 1.0000000
## [15] 1.7320508
```

```
##  [1]       NaN 1.0000000       NaN       Inf 1.0000000 0.0000000 1.0000000
##  [8] 1.0000000 1.0000000 0.7071068 1.0000000 1.4142136 0.5773503 1.0000000
## [15] 1.7320508
```

`safely`

```
junk = list(letters, 1:20, median)
# map(junk, ~ log(.x))

    ## Error in log(.x): non-numeric argument to mathematical function
```

- `safely` "catches" errors and always "succeeds".
- `try` does the same, but either returns the value or a try-error object.
- `safely` is type-stable. It always returns a length-two list with one object NULL.

```
safe = map(junk, ~ safely(log)(.x))  # Note the different syntax from try(log(.x)). `safely(log)` creat
safe
```

```
## [[1]]
## [[1]]$result
## NULL
##
## [[1]]$error
## <simpleError in .Primitive("log")(x, base): non-numeric argument to mathematical function>
##
##
## [[2]]
## [[2]]$result
##  [1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379 1.7917595 1.9459101
##  [8] 2.0794415 2.1972246 2.3025851 2.3978953 2.4849066 2.5649494 2.6390573
## [15] 2.7080502 2.7725887 2.8332133 2.8903718 2.9444390 2.9957323
##
## [[2]]$error
## NULL
##
##
## [[3]]
## [[3]]$result
## NULL
##
## [[3]]$error
## <simpleError in .Primitive("log")(x, base): non-numeric argument to mathematical function>
```

transpose a list!

Now we could conveniently move on where the function succeeded, particularly using `map_if`. To get that logical vector for the `map_if` test, we can use the `transpose` function, which inverts a list.

```
transpose(safe)
```

```
## $result
## $result[[1]]
## NULL
##
## $result[[2]]
##  [1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379 1.7917595 1.9459101
##  [8] 2.0794415 2.1972246 2.3025851 2.3978953 2.4849066 2.5649494 2.6390573
## [15] 2.7080502 2.7725887 2.8332133 2.8903718 2.9444390 2.9957323
##
## $result[[3]]
## NULL
##
##
## $error
## $error[[1]]
## <simpleError in .Primitive("log")(x, base): non-numeric argument to mathematical function>
##
## $error[[2]]
## NULL
##
## $error[[3]]
## <simpleError in .Primitive("log")(x, base): non-numeric argument to mathematical function>
```

```
map_if(transpose(safe)$result, ~!is.null(.x), median)
```

```
## [[1]]
## NULL
##
## [[2]]
## [1] 2.35024
##
## [[3]]
## NULL
```

stringr

All your string manipulation and regex functions with a consistent API.

```
library(stringr)  # not attached with tidyverse
fishes <- c("one fish", "two fish", "red fish", "blue fish")
str_detect(fishes, "two")
```

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

```
str_replace_all(fishes, "fish", "banana")
```

```
## [1] "one banana"  "two banana"  "red banana"  "blue banana"
```

```
fishes
```

```
## [1] "one fish"  "two fish"  "red fish"  "blue fish"
```

```
str_extract(fishes, "[a-z]\\s")
```

```
## [1] "e " "o " "d " "e "
```

Let's put that string manipulation engine to work. Remember the annoying column names in the WHO data? They look like this new_sp_m014, new_sp_m1524, new_sp_m2534, where "new" or "new_" doesn't mean anything, the following 2-3 letters indicate the test used, the following letter indicates the gender, and the final 2-4 numbers indicates the age-class. A string-handling challenge if ever there was one. Let's separate it out and plot the cases by year, gender, age-class, and test-method.

```
who %>%
  select(-iso2, -iso3) %>%
  gather(group, cases, -country, -year ) %>%
  mutate(group = str_replace(group, "new_*", ""),
         method = str_extract(group, "[a-z]+"),
         gender = str_sub(str_extract(group, "_[a-z]"), 2, 2),
         age = str_extract(group, "[0-9]+"),
         age = ifelse(str_length(age) > 2,
                      str_c(str_sub(age, 1, -3), str_sub(age, -2, -1), sep = "-"),
                      str_c(age, "+"))) %>%
  group_by(year, gender, age, method) %>%
  summarize(total_cases = sum(cases, na.rm = TRUE)) %>%
  ggplot(aes(x = year, y = total_cases, linetype = gender)) +
  geom_line() +
  facet_grid(method ~ age,
             labeller = labeller(.rows = label_both, .cols = label_both)) +
  scale_y_log10() +
```

```
  theme_light() +
  theme(axis.text.x = element_text(angle = 45, vjust = 1, hjust = 1))
```

```
## Warning: Transformation introduced infinite values in continuous y-axis
```