

tidyverse

Randy Johnson based on a talk to the Davis R-Users' Group by Michael Levy

January 17, 2017

What is the tidyverse?

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

Tidy data

Put data in data frames

- Each dataset gets a data frame
- Each variable gets a column
- Each observation gets a row

Reuse existing data structures whenever possible!

Tidy APIs

Functions should be consistent and easily (human) readable

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

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
- 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)

## Loading tidyverse: ggplot2
## Loading tidyverse: tibble
## Loading tidyverse: tidyr
## Loading tidyverse: readr
## Loading tidyverse: purrr
## Loading tidyverse: dplyr

## Conflicts with tidy packages -----

## filter(): dplyr, stats
## lag():      dplyr, stats

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

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

Tibbles print politely.

```
tdf

## # A tibble: 10,000 × 2
##       x         y
##   <int>    <dbl>
## 1     1 -0.7642070
## 2     2 -1.1029780
## 3     3 -1.1207650
## 4     4  0.3821556
## 5     5  1.7839275
## 6     6 -0.6526629
## 7     7  2.7817024
## 8     8 -1.4070495
## 9     9  0.6643505
## 10    10 -1.2881764
## # ... 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]),
            tbl = data_frame(abc = letters[1:3], xyz = letters[24:26])
          )
sapply(dfs, function(d) class(d$abc))
```

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

partial matching of names

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

```
## Warning: Unknown column 'a'
## $df
## [1] a b c
## Levels: a b c
##
## $tbl
## NULL
```

type consistency

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

```
## $df
## [1] "factor"
##
## $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!

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

```
## # A tibble: 5 × 2
##   ints    powers
##   <int>   <list>
## 1     1 1 <dbl [1]>
## 2     2 2 <dbl [2]>
## 3     3 3 <dbl [3]>
## 4     4 4 <dbl [4]>
## 5     5 5 <dbl [5]>
```

The pipe %>%

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

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

```
## [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 × 24
##           title year length budget
##           <chr> <int> <int> <int>
## 1 Commune (Paris, 1871), La 2000  555    NA
## 2 Cure for Insomnia, The 1987  5220    NA
## 3 Ebolusyon ng isang pamilyang pilipino 2004  647    NA
## 4 Empire 1964  485    NA
## 5 Farmer's Wife, The 1998  390    NA
## 6 Foolish Wives 1922  384 1100000
## 7 Four Stars 1967  1100    NA
## 8 Hitler - ein Film aus Deutschland 1978  407    NA
## 9 Imitation of Christ 1967  480    NA
## 10 Longest Most Meaningless Movie in the World, The 1970  2880    NA
## # ... with 11 more rows, and 20 more variables: rating <dbl>, votes <int>,
## #   r1 <dbl>, r2 <dbl>, r3 <dbl>, r4 <dbl>, r5 <dbl>, r6 <dbl>, r7 <dbl>,
## #   r8 <dbl>, r9 <dbl>, r10 <dbl>, mpaa <chr>, Action <int>,
## #   Animation <int>, Comedy <int>, Drama <int>, Documentary <int>,
## #   Romance <int>, Short <int>
```

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

```
## # A tibble: 21 × 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                 3.0     12
## 8      Hitler - ein Film aus Deutschland    9.0     70
## 9      Imitation of Christ          4.4      5
## 10 Longest Most Meaningless Movie in the World, The    6.4     15
## # ... with 11 more rows
```

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

```
## # A tibble: 135 × 2
##               title rating
##               <chr>   <dbl>
## 1      Sen to Chihiro no kamikakushi    8.6
## 2      Duck Amuck                     8.4
## 3      Wallace & Gromit: The Wrong Trousers    8.4
## 4      Finding Nemo                     8.3
## 5      Hotaru no haka                    8.3
## 6      Incredibles, The                 8.3
## 7      Mononoke-hime                    8.3
## 8      What's Opera, Doc?               8.3
## 9      Vincent                         8.2
## 10     Wallace & Gromit: A Close Shave    8.2
## # ... with 125 more rows
```

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

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

```
## Source: local data frame [128 x 4]
## Groups: year [54]
##
##   year mpaa avg_budget avg_rating
##   <int> <chr>   <dbl>   <dbl>
## 1  2005 NC-17      NaN    6.700000
## 2  2005 PG    45857143    5.733333
## 3  2005 PG-13  42269333    5.326087
## 4  2005 R    24305882    4.595833
## 5  2004 PG    45126852    5.847619
## 6  2004 PG-13  46288254    6.080180
```

```
## 7    2004      R    19548519    5.848469
## 8    2003     PG    37057692    5.897674
## 9    2003 PG-13    46269491    5.949038
## 10   2003      R    21915505    5.702273
## # ... with 118 more rows
```

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

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

```
## Source: local data frame [156 x 4]
## Groups: year, mpaa [128]
##
```

	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)
full_join(t1, t2, by = "alpha", suffix = c("_t1", "_t2"))
```

```
## # A tibble: 10 × 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
```

tidyr

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

```
who # Tuberculosis data from the WHO
```

```
## # A tibble: 7,240 × 60
##   country iso2 iso3 year new_sp_m014 new_sp_m1524 new_sp_m2534
##   <chr>   <chr> <chr> <int>      <int>      <int>      <int>
## 1 Afghanistan AF AFG 1980         NA         NA         NA
## 2 Afghanistan AF AFG 1981         NA         NA         NA
## 3 Afghanistan AF AFG 1982         NA         NA         NA
## 4 Afghanistan AF AFG 1983         NA         NA         NA
## 5 Afghanistan AF AFG 1984         NA         NA         NA
## 6 Afghanistan AF AFG 1985         NA         NA         NA
## 7 Afghanistan AF AFG 1986         NA         NA         NA
## 8 Afghanistan AF AFG 1987         NA         NA         NA
## 9 Afghanistan AF AFG 1988         NA         NA         NA
## 10 Afghanistan AF AFG 1989         NA         NA         NA
## # ... with 7,230 more rows, and 53 more variables: new_sp_m3544 <int>,
## #   new_sp_m4554 <int>, new_sp_m5564 <int>, new_sp_m65 <int>,
## #   new_sp_f014 <int>, new_sp_f1524 <int>, new_sp_f2534 <int>,
## #   new_sp_f3544 <int>, new_sp_f4554 <int>, new_sp_f5564 <int>,
## #   new_sp_f65 <int>, new_sn_m014 <int>, new_sn_m1524 <int>,
## #   new_sn_m2534 <int>, new_sn_m3544 <int>, new_sn_m4554 <int>,
## #   new_sn_m5564 <int>, new_sn_m65 <int>, new_sn_f014 <int>,
## #   new_sn_f1524 <int>, new_sn_f2534 <int>, new_sn_f3544 <int>,
## #   new_sn_f4554 <int>, new_sn_f5564 <int>, new_sn_f65 <int>,
## #   new_ep_m014 <int>, new_ep_m1524 <int>, new_ep_m2534 <int>,
## #   new_ep_m3544 <int>, new_ep_m4554 <int>, new_ep_m5564 <int>,
## #   new_ep_m65 <int>, new_ep_f014 <int>, new_ep_f1524 <int>,
## #   new_ep_f2534 <int>, new_ep_f3544 <int>, new_ep_f4554 <int>,
## #   new_ep_f5564 <int>, new_ep_f65 <int>, newrel_m014 <int>,
## #   newrel_m1524 <int>, newrel_m2534 <int>, newrel_m3544 <int>,
```

```
## # newrel_m4554 <int>, newrel_m5564 <int>, newrel_m65 <int>,
## # newrel_f014 <int>, newrel_f1524 <int>, newrel_f2534 <int>,
## # newrel_f3544 <int>, newrel_f4554 <int>, newrel_f5564 <int>,
## # newrel_f65 <int>
```

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

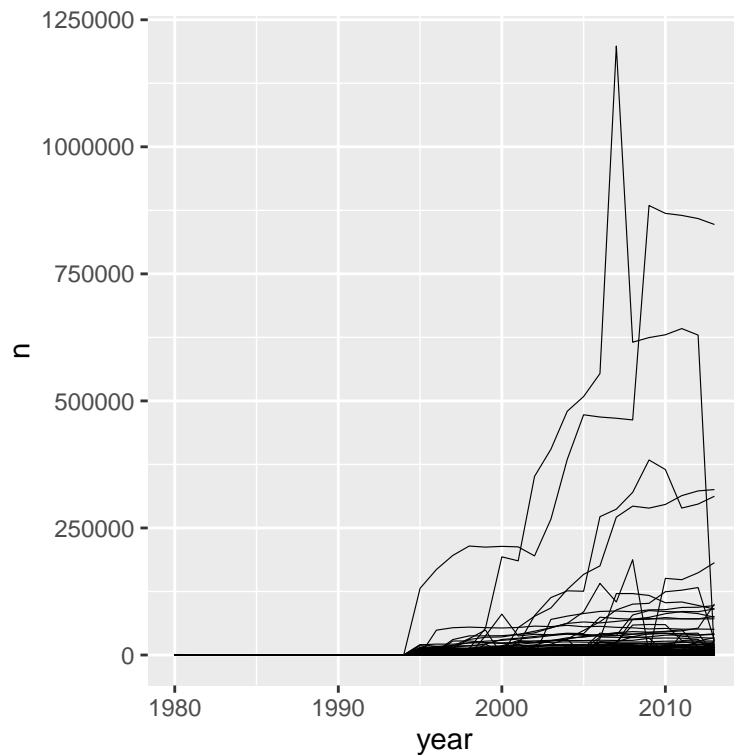
```
## # A tibble: 405,440 × 6
##   country iso2 iso3 year group cases
##   <chr> <chr> <chr> <int> <chr> <int>
## 1 Afghanistan AF AFG 1980 new_sp_m014 NA
## 2 Afghanistan AF AFG 1981 new_sp_m014 NA
## 3 Afghanistan AF AFG 1982 new_sp_m014 NA
## 4 Afghanistan AF AFG 1983 new_sp_m014 NA
## 5 Afghanistan 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 405,430 more rows
```

ggplot2

We will cover this in more detail at a later point, but if you would like to know more, check out one of their 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)
```

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))
```

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

```
##   user  system elapsed
##  3.025   0.103   3.676
```

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

```
##   user  system elapsed
##  0.962   0.081   1.100
```

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

```
##   X int squares letters
## 1 1  1      1      d
## 2 2  2      4      x
## 3 3  3      9      i
```

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

```
## Parsed with column specification:
```

```
## cols(
##   int = col_integer(),
##   squares = col_double(),
##   letters = col_character()
## )

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

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
##  0.042   0.002   0.044
##
## [[2]]
##      user  system elapsed
##  0.050   0.001   0.051
##
## [[3]]
##      user  system elapsed
## 14.903   0.378  15.469
##
## [[4]]
##      user  system elapsed
##  8.797   0.209   9.145
```

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) %>% # str()
  map(~ lm(rating ~ budget, data = .)) %>%
  map_df(tidy, .id = "mpaa-rating") %>%
  arrange(term)

```

```
## Error in as_function(.f, ...): object 'tidy' not found
```

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 × 6
##       dist    funs    samples    mean    var    median
##       <chr> <list>    <list>    <dbl>    <dbl>    <dbl>
## 1   normal <fun> <dbl [100]> 4.928887 1.114385 4.958149
## 2  poisson <fun> <int [100]> 5.470000 5.766768 5.000000
## 3 chi-square <fun> <dbl [100]> 4.521491 9.012409 3.703844

```

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)
```

```
## Error in as_function(.f, ...): object 'augment' not found
```

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"
```

```
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))
```

