APPLICATIONS

OF DATA SCIENCE

The Tidyverse

Applications of Data Science - Class 1

Giora Simchoni

gsimchoni@gmail.com and add #dsapps in subject

Stat. and OR Department, TAU

2023-02-23

I don't need to know about wrangling data, I get by.

So, what's wrong with Excel?

(MS Excel is one amazing software. But it lacks:)

- Structure (or rather, structure is up to the user)
- Types to variables
- Automation (you could learn VBA Excel, but the horror)
- Reproducibility
- Open Source
- Extensibility
- Speed and Scale
- Modeling (there *is* a t-test, but the horror)

Excel might be the most dangerous software on the planet (Forbes)

Excel horror stories

So, what's wrong with base R?

(Base R is one amazing software. But it lacks:)

- Consistency:
  - Function names
  - Function arguments names
  - Function arguments order
  - Function return types (sometimes the same function!)
- Meaningful errors and warnings
- Good choices of default values to arguments
- Speed
- Good and easy visualizations
- One other thing

(In) Consistency - Example 1: Strings

```
# split a string by pattern: strsplit(string, pattern)
strsplit("Who dis?", " ")
```

```
## [[1]]
## [1] "Who"  "dis?"
```

```
# find if a pattern exists in a string: grepl(pattern, string)
grepl("di", "Who dis?")
```

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

```
# substitute a pattern in a string: sub(pattern, replace, string)
sub("di", "thi", "Who dis?")
```

```
## [1] "Who this?"
```

```
# length of a string: nchar(string); length of object: length(obj)
c(nchar("Who dis?"), length("Who dis?"))
```

```
## [1] 8 1
```

(In) Consistency - Example 2: Models

```
n <- 10000
x1 <- runif(n)
x2 <- runif(n)
t <- 1 + 2 * x1 + 3 * x2
y <- rbinom(n, 1, 1 / (1 + exp(-t)))
```

```
glm(y ~ x1 + x2, family = "binomial")
```

```
glmnet(as.matrix(cbind(x1, x2)), as.factor(y), family = "binomial'
```

```
randomForest(as.factor(y) ~ x1 + x2)
```

```
gbm(y ~ x1 + x2, data = data.frame(x1 = x1, x2 = x2, y = y))
```

(Un) Meaningful Errors - Example

```
df <- data.frame(Education = 1:5, Ethnicity = c(2, 4, 5, 2, 1))
table(df$Eduction, df$Ethnicity)
```

```
## Error in table(df$Eduction, df$Ethnicity): all arguments must have the s
length
```

(Bad) Default Values - Example

	A	B	C
1	col1	col2	col3
2	1	0.2	a
3	2	0.3	b
4	3	0.4	c
5	4	0.5	d
6			

```
# In R 3.6... in R 4.0 this was fixed!
df <- read.csv("../data/bad_args_test.csv")
df$col3
```

```
## [1] a b c d
## Levels: a b c d
```

```
df <- read.csv("../data/bad_args_test.csv", stringsAsFactors = FAI
df$col3
```

```
## [1] "a" "b" "c" "d"
```

(No) Speed - Example

```
file_path <- "../data/mediocre_file.csv"
df <- read.csv(file_path)
dim(df)
```

```
## [1] 9180   14
```

```
library(microbenchmark)

microbenchmark(
  read_base = read.csv(file_path),
  read_tidy = read_csv(file_path, col_types = cols()),
  read_dt = data.table::fread(file_path),
  times = 10)
```

```
## Unit: milliseconds
##       expr       min        lq     mean    median        uq       max neva
##  read_base 42.416501 42.694401 43.52487 43.345901 44.041901  45.6394
##  read_tidy 28.360901 29.053501 41.41245 30.191001 32.116101 137.5225
##    read_dt  8.339102  8.696902 14.89041  8.819901  8.943301  63.0822
```

Detour: The OKCupid Dataset

The OKCupid Dataset

- ~60K active OKCupid users scraped on June 2012
- 35K Male, 25K Female (less awareness for non-binary back then)
- Answers to questions like:
  - Body Type
  - Diet
  - Substance Abuse
  - Education
  - Do you like pets?
  - Open questions, e.g. "On a typical Friday night I am..."
  - And the more boring demographic details like age, height, location, sign, religion etc.
- See here for the full codebook

BTW

Letter to the Editor

A Letter to the *Journal of Statistics and Data Science Education* — A Call for Review of "OkCupid Data for Introductory Statistics and Data Science Courses" by Albert Y. Kim and Adriana Escobedo-Land

Tiffany Xiao & Yifan Ma

Pages 214-215 | Published online: 18 Jun 2021

Download citation https://doi.org/10.1080/26939169.2021.1930812

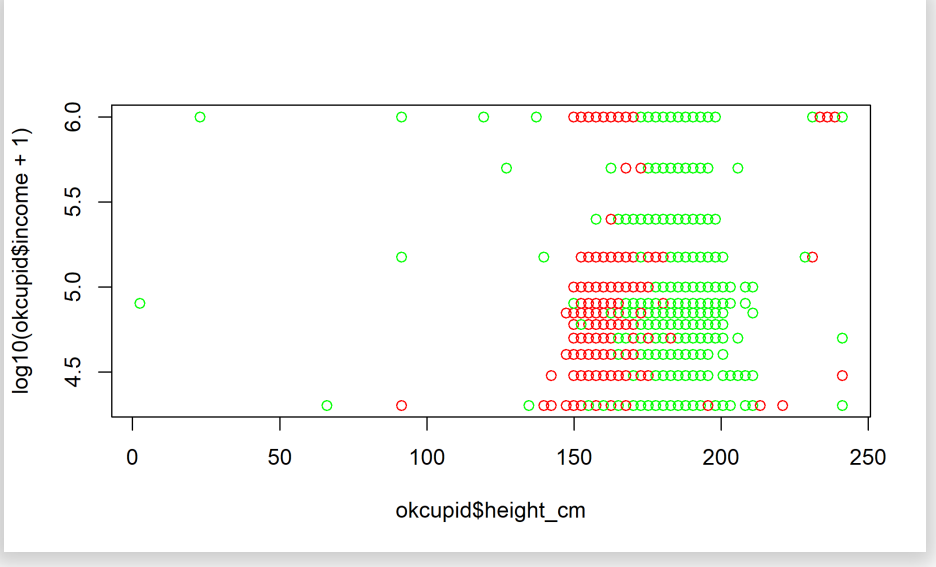
See here.

End of Detour

(Not) Good Vizualizations - Example

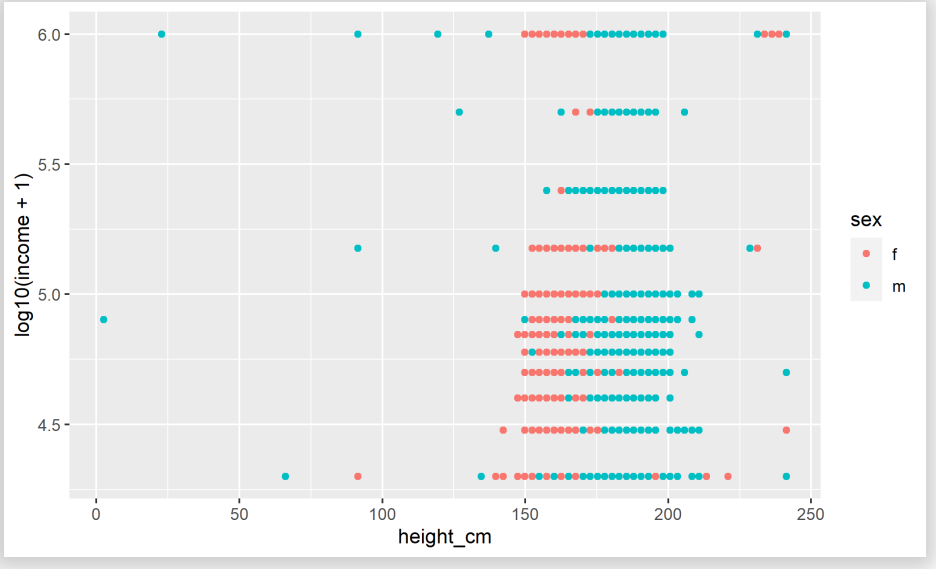
```
okcupid <- read_csv("~/okcupid.csv.zip", col_types = cols())
okcupid$income[okcupid$income == -1] <- NA
okcupid$height_cm <- okcupid$height * 2.54
```

```
plot(okcupid$height_cm, log10(okcupid$income + 1),
     col = c("red", "green")[as.factor(okcupid$sex)])
```



```
ggplot(okcupid, aes(height_cm, log10(income + 1), color = sex)) +
  geom_point()
```

```
## Warning: Removed 48442 rows containing missing values (geom_point()).
```



One other thing

Manager: "Give me the average income of women respondents above age 30 grouped by sexual orientation!"

You:

```
mean_bi <- mean(okcupid$income[okcupid$sex == "f" & okcupid$age >
mean_gay <- mean(okcupid$income[okcupid$sex == "f" & okcupid$age >
mean_straight <- mean(okcupid$income[okcupid$sex == "f" & okcupid$

data.frame(orientation = c("bisexual", "gay", "straight"),
           income_mean = c(mean_bi, mean_gay, mean_straight))
```

```
##   orientation income_mean
## 1    bisexual   133421.05
## 2         gay    86489.36
## 3    straight    85219.74
```

Or the slightly better you:

```
mean_income_function <- function(orientation) {
  mean(okcupid$income[okcupid$sex == "f" & okcupid$age > 30 & okcu
}

mean_bi <- mean_income_function("bisexual")
mean_gay <- mean_income_function("gay")
mean_straight <- mean_income_function("straight")

data.frame(orientation = c("bisexual", "gay", "straight"),
           income_mean = c(mean_bi, mean_gay, mean_straight))
```

```
##   orientation income_mean
## 1    bisexual   133421.05
## 2         gay    86489.36
## 3    straight    85219.74
```

Or the even better you:

```
orientations <- c("bisexual", "gay", "straight")
income_means <- numeric(3)

for (i in seq_along(orientations)) {
  income_means[i] <- mean_income_function(orientations[i])
}

data.frame(orientation = orientations, income_mean = income_means)
```

```
##   orientation income_mean
## 1    bisexual   133421.05
## 2         gay    86489.36
## 3    straight    85219.74
```

Or the best you:

```
okcupid_females_over30 <- with(okcupid, okcupid[sex == "f" & age >
aggregate(okcupid_females_over30$income,
          by = list(orientation = okcupid_females_over30$orientati
          FUN = mean, na.rm = TRUE)
```

```
##   orientation         x
## 1    bisexual 133421.05
## 2         gay  86489.36
## 3    straight  85219.74
```

Manager: "What? Why would bisexual women have a higher income than straight or gay women? Could you add the median, trimmed mean, standard error and n?"

You: 😱

The Tidyverse

What *is* The Tidyverse?

> The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

- `tibble`: the `data.frame` re-imagined
- `readr`: importing/exporting (mostly rectangular) data for humans
- `dplyr` + `tidyr`: a grammar of data manipulation
- `purrr`: functional programming in R
- `stringr`: string manipulation
- `ggplot2`: a grammar of graphics

The above can all be installed and loaded under the `tidyverse` package:

```
library(tidyverse)
```

Many more:

- `lubridate`: manipulating dates
- `tidymodels`: tidy modeling/statistics
- `rvest`: web scraping
- `tidytext`: tidy text analysis (life saver)
- `tidygraph` + `ggraph`: manipulating and plotting networks
- `glue`: print like a boss
- countless `gg` extensions (`ggmosaic`, `ggbeeswarm`, `gganimate`, `ggridges` etc.)

What's so great about the Tidyverse?

- Tidy Data
- Consistency (in function names, args, return types, documentation)
- The Pipe
- Speed (C++ under the hood)
- `ggplot2`
- The Community

Tidy Data

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

Which one of these datasets is tidy? (I)

```
table1
```

```
## # A tibble: 315 × 4
##    religion      yob n_straight n_total
##    <chr>       <dbl>      <dbl>   <dbl>
##  1 atheist      1950         26      29
##  2 buddhist     1950          6       6
##  3 christian    1950         28      32
##  4 hindu        1950          0       0
##  5 jewish       1950         21      24
##  6 muslim       1950          0       0
##  7 unspecified  1950         71      76
##  8 atheist      1951         31      33
##  9 buddhist     1951         11      11
## 10 christian    1951         23      24
## # … with 305 more rows
```

Which one of these datasets is tidy? (II)

```
table2
```

```
## # A tibble: 630 × 4
##    religion    yob type         n
##    <chr>     <dbl> <chr>    <dbl>
##  1 atheist    1950 straight    26
##  2 atheist    1950 total       29
##  3 buddhist   1950 straight     6
##  4 buddhist   1950 total       6
##  5 christian  1950 straight    28
##  6 christian  1950 total       32
##  7 hindu      1950 straight     0
##  8 hindu      1950 total       0
##  9 jewish     1950 straight    21
## 10 jewish     1950 total       24
## # … with 620 more rows
```

Which one of these datasets is tidy? (III)

```
table3
```

```
## # A tibble: 315 × 3
##    religion      yob pct_straight
##    <chr>       <dbl> <chr>
##  1 atheist      1950 26/29
##  2 buddhist     1950 6/6
##  3 christian    1950 28/32
##  4 hindu        1950 0/0
##  5 jewish       1950 21/24
##  6 muslim       1950 0/0
##  7 unspecified  1950 71/76
##  8 atheist      1951 31/33
##  9 buddhist     1951 11/11
## 10 christian    1951 23/24
## # … with 305 more rows
```

Which one of these datasets is tidy? (IV)

```
table4
```

```
## # A tibble: 7 × 91
##   religion    n_total_1950 n_total_1951 n_total_1952 n_total_1953 n_tota
##   <chr>              <dbl>        <dbl>        <dbl>        <dbl>
## 1 atheist               29           33           34           37
## 2 buddhist               6           11           14           16
## 3 christian             32           24           37           47
## 4 hindu                  0            0            0            1
## 5 jewish                24           29           27           23
## 6 muslim                 0            0            0            0
## 7 unspecified           76           79           83           97
## # … with 85 more variables: n_total_1955 <dbl>, n_total_1956 <dbl>,
## #   n_total_1957 <dbl>, n_total_1958 <dbl>, n_total_1959 <dbl>,
## #   n_total_1960 <dbl>, n_total_1961 <dbl>, n_total_1962 <dbl>,
## #   n_total_1963 <dbl>, n_total_1964 <dbl>, n_total_1965 <dbl>,
## #   n_total_1966 <dbl>, n_total_1967 <dbl>, n_total_1968 <dbl>,
## #   n_total_1969 <dbl>, n_total_1970 <dbl>, n_total_1971 <dbl>,
## #   n_total_1972 <dbl>, n_total_1973 <dbl>, n_total_1974 <dbl>, …
```

Why Tidy?

> Happy families are all alike; every unhappy family is unhappy in its own way. (Leo Tolstoy)

> It allows R's vectorised nature to shine. (Hadley Wickham)

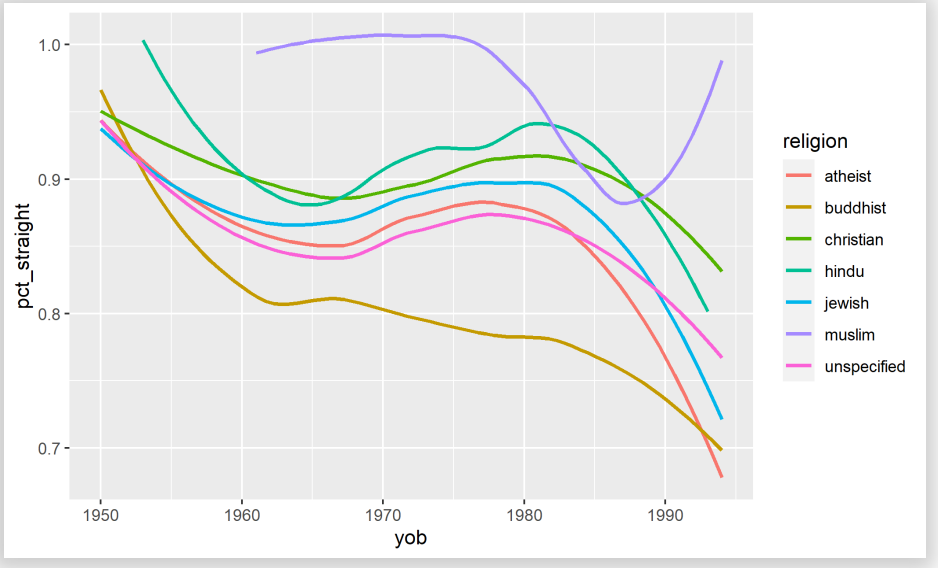
A Tidy dataset will be much easier to transform

```
table1$pct_straight = table1$n_straight / table1$n_total
table1
```

```
## # A tibble: 315 × 5
##    religion      yob n_straight n_total pct_straight
##    <chr>       <dbl>      <dbl>   <dbl>        <dbl>
##  1 atheist      1950         26      29        0.897
##  2 buddhist     1950          6       6        1
##  3 christian    1950         28      32        0.875
##  4 hindu        1950          0       0      NaN
##  5 jewish       1950         21      24        0.875
##  6 muslim       1950          0       0      NaN
##  7 unspecified  1950         71      76        0.934
##  8 atheist      1951         31      33        0.939
##  9 buddhist     1951         11      11        1
## 10 christian    1951         23      24        0.958
## # … with 305 more rows
```

A Tidy dataset will be much easier to plot

```
ggplot(table1, aes(x = yob, y = pct_straight, color = religion)) +
  geom_smooth(method = "loess", formula = y ~ x, se = FALSE)
```



Detour: The `tibble`

The `tibble`: the `data.frame` re-imagined

- Prints nicer:

```
tib1 <- tibble(day = lubridate::today() + runif(1e3) * 30,
               type = sample(letters, 1e3, replace = TRUE),
               quantity = sample(seq(0, 100, 10), 1e3, replace = T
tib1
```

```
## # A tibble: 1,000 × 3
##    day        type  quantity
##    <date>     <chr>    <dbl>
##  1 2023-02-28 l           40
##  2 2023-03-17 g           60
##  3 2023-03-21 i           30
##  4 2023-03-04 w            0
##  5 2023-03-20 c           60
##  6 2023-03-06 u           10
##  7 2023-03-06 o           40
##  8 2023-03-05 s           40
##  9 2023-03-07 e           50
## 10 2023-03-01 r           50
## # … with 990 more rows
```

```
df1 <- data.frame(day = lubridate::today() + runif(1e3) * 30,
                  type = sample(letters, 1e3, replace = TRUE),
                  quantity = sample(seq(0, 100, 10), 1e3, replace
df1
```

```
##              day type quantity
## 1     2023-02-23    b        0
## 2     2023-02-26    g        0
## 3     2023-03-15    x       50
## 4     2023-03-24    q       60
## 5     2023-03-02    a       20
## 6     2023-03-06    g        0
## 7     2023-03-09    s       50
## 8     2023-03-16    o       30
## 9     2023-02-25    e        0
## 10    2023-03-16    n        0
## 11    2023-03-14    w        0
## 12    2023-03-11    l       50
## 13    2023-03-21    s       20
## 14    2023-02-26    b       30
## 15    2023-03-13    z       70
## 16    2023-03-05    q       40
## 17    2023-03-09    n       60
## 18    2023-03-20    t       50
## 19    2023-03-10    z       30
## 20    2023-03-05    a       80
## 21    2023-02-24    i       20
## 22    2023-03-14    a       60
## 23    2023-02-25    r        0
```

- Warns you when you make mistakes (!):

```
tib1$quanitty
```

```
## Warning: Unknown or uninitialised column: `quanitty`.
```

```
## NULL
```

```
df1$quanitty
```

```
## NULL
```

- Can also create via `tribble()`:

```
tribble(
  ~a, ~b, ~c,
  "a", 1, 2.2,
  "b", 2, 4.3,
  "c", 3, 3.4
)
```

```
## # A tibble: 3 × 3
##   a         b     c
##   <chr> <dbl> <dbl>
## 1 a         1   2.2
## 2 b         2   4.3
## 3 c         3   3.4
```

- Can build on top of variables during creation:

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

```
## # A tibble: 5 × 2
##       x     y
##   <int> <dbl>
## 1     1     1
## 2     2     4
## 3     3     9
## 4     4    16
## 5     5    25
```

```
data.frame(x = 1:5, y = x^2)
```

```
## Error in data.frame(x = 1:5, y = x^2): object 'x' not found
```

- Will never turn your strings into factors, will never change your column names:

```
tib1 <- readr::read_csv("../data/another_bad_test_args.csv", col_t
colnames(tib1)
```

```
## [1] "col 1" "col 2" "col 3"
```

```
tib1$`col 3`
```

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

```
df1 <- read.csv("../data/another_bad_test_args.csv")
colnames(df1)
```

```
## [1] "col.1" "col.2" "col.3"
```

```
df1$col.3
```

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

Though one ought to remember a `tibble` is still a `data.frame`:

```
class(tib1)
```

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

```
class(df1)
```

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

End of Detour

Consistency - Example: `stringr`

> a cohesive set of functions designed to make working with strings as easy as possible.

```
strings_vec <- c("I'm feeling fine", "I'm perfectly OK",
                 "Nothing is wrong!")
str_length(strings_vec)
```

```
## [1] 16 16 17
```

```
str_c(strings_vec, collapse = ", ")
```

```
## [1] "I'm feeling fine, I'm perfectly OK, Nothing is wrong!"
```

```
str_sub(strings_vec, 1, 3)
```

```
## [1] "I'm" "I'm" "Not"
```

```r
str_detect(strings_vec, "I'm")
```

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

```r
str_replace(strings_vec, "I'm", "You're")
```

```
## [1] "You're feeling fine" "You're perfectly OK" "Nothing is wrong!"
```

```r
str_split("Do you know regex?", " ")
```

```
## [[1]]
## [1] "Do"     "you"    "know"   "regex?"
```

```r
str_extract(strings_vec, "[aeiou]")
```

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

```r
str_count(strings_vec, "[A-Z]")
```

```
## [1] 1 3 1
```

The Pipe

Remember you?

```
mean_bi <- mean(okcupid$income[okcupid$sex == "f" & okcupid$age >
mean_gay <- mean(okcupid$income[okcupid$sex == "f" & okcupid$age >
mean_straight <- mean(okcupid$income[okcupid$sex == "f" & okcupid$

data.frame(orientation = c("bisexual", "gay", "straight"),
           income_mean = c(mean_bi, mean_gay, mean_straight))
```

```
##   orientation income_mean
## 1    bisexual   133421.05
## 2         gay    86489.36
## 3    straight    85219.74
```

Doesn't this make much more sense?

```
okcupid %>%
  filter(sex == "f", age > 30) %>%
  group_by(orientation) %>%
  summarize(income_mean = mean(income, na.rm = TRUE))
```

```
## # A tibble: 3 × 2
##   orientation income_mean
##   <chr>             <dbl>
## 1 bisexual        133421.
## 2 gay              86489.
## 3 straight         85220.
```

- Read as:
  - Take the OKCupid data,
  - Filter only women above the age of 30,
  - And for each group of sexual orientation,
  - Give me the average income

- Make verbs, not nouns
- Can always access the dataset last stage with " . ":

```
okcupid %>%
  filter(str_count(essay0) > median(str_count(.$essay0), na.rm = T
```

- Operates not just on data frames or tibbles:

```
strings_vec %>% str_to_title()
```

```
## [1] "I'm Feeling Fine"  "I'm Perfectly Ok"  "Nothing Is Wrong!"
```

- No intermediate objects
- Don't strive to make the longest possible pipe (though it is a fun experiment)
- Tools exist for debugging (e.g. `%T>%`, the ViewPipeStep package, ...)

And, if you want to throw in the n, the median:

```
okcupid %>%
  filter(sex == "f", age > 30) %>%
  group_by(orientation) %>%
  summarize(income_mean = mean(income, na.rm = TRUE),
            income_median = median(income, na.rm = TRUE),
            n = n())
```

```
## # A tibble: 3 × 4
##   orientation income_mean income_median     n
##   <chr>             <dbl>         <dbl> <int>
## 1 bisexual        133421.         50000   652
## 2 gay              86489.         40000   664
## 3 straight         85220.         60000 10436
```

And if you want this for the age as well:

```
okcupid %>%
  filter(sex == "f", age > 30) %>%
  group_by(orientation) %>%
  summarize(across(c(income, age),
                   list(mean = ~mean(.x, na.rm = TRUE),
                        median = ~median(.x, na.rm = TRUE))))
```

```
## # A tibble: 3 × 5
##   orientation income_mean income_median age_mean age_median
##   <chr>             <dbl>         <dbl>    <dbl>      <dbl>
## 1 bisexual        133421.         50000     37.8         36
## 2 gay              86489.         40000     40.4         38
## 3 straight         85220.         60000     40.7         38
```

Now *this* is a language for Data Science.

But we're getting ahead of ourselves.

In fact!

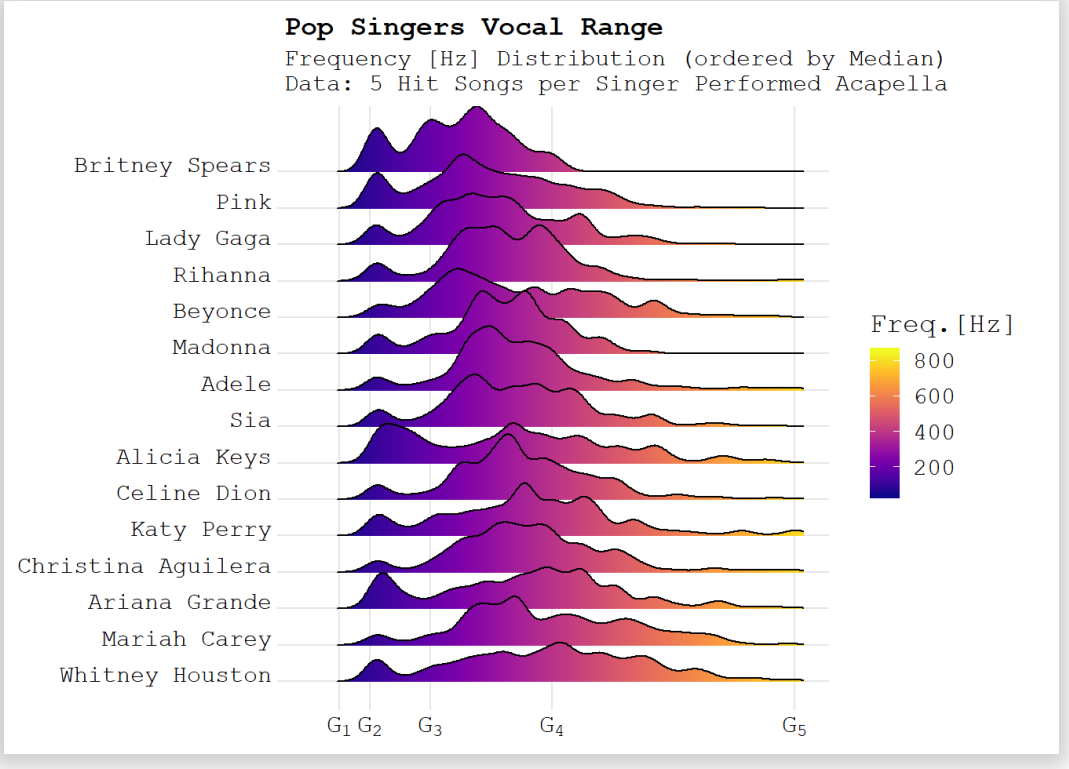
This became so popular that since R 4.1 there is a built-in pipe operator:

```
okcupid |>
  filter(sex == "f", age > 30) |>
  group_by(orientation) |>
  summarize(across(c(income, age),
            list(mean = \(x) mean(x, na.rm = TRUE),
                 median = \(x) median(x, na.rm = TRUE))))
```

```
## # A tibble: 3 × 5
##   orientation income_mean income_median age_mean age_median
##   <chr>             <dbl>         <dbl>    <dbl>      <dbl>
## 1 bisexual        133421.          50000     37.8         36
## 2 gay              86489.          40000     40.4         38
## 3 straight         85220.          60000     40.7         38
```

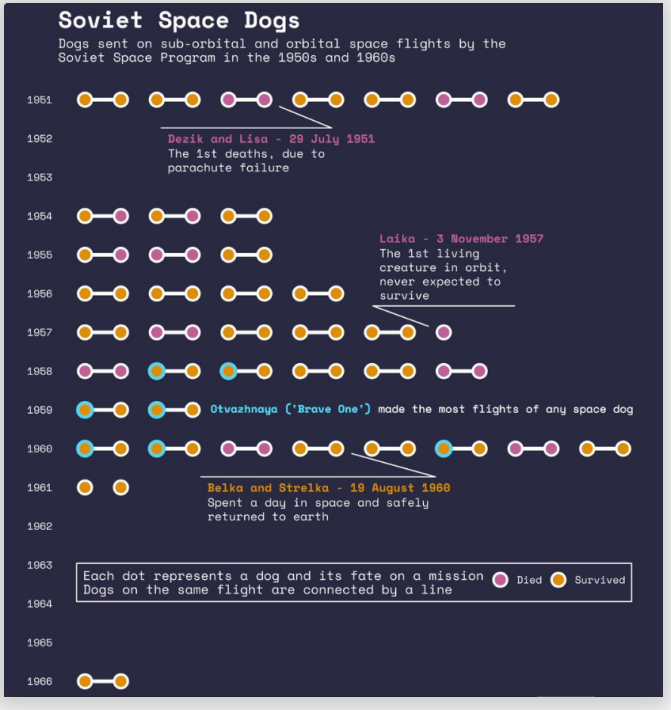
💡 What else changed in R 4.1?

ggplot2



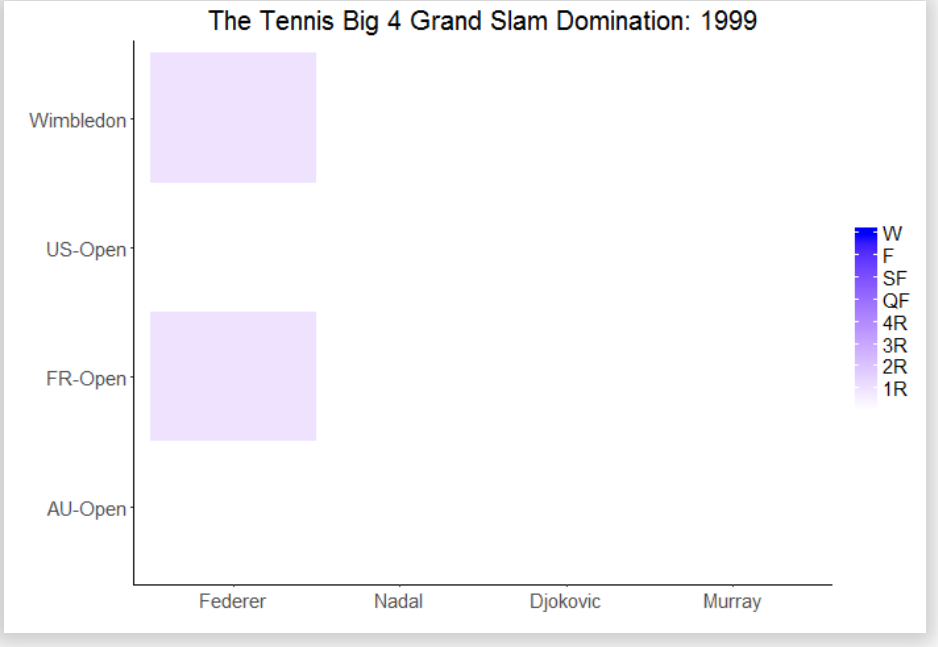
[Ave Mariah / Giora Simchoni]

ggplot2



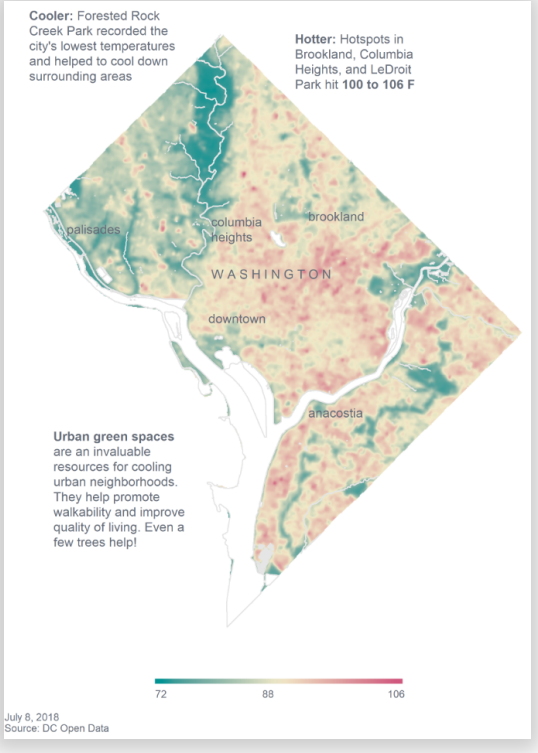
[Soviet Space Dogs / David Smale]

ggplot2



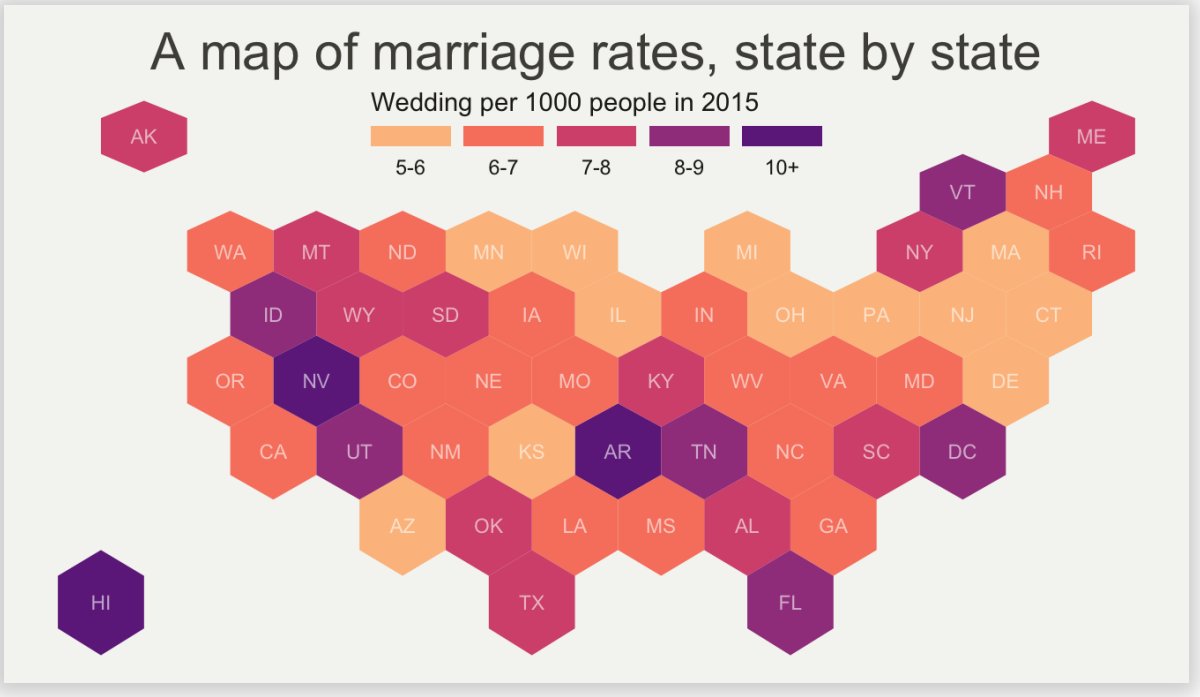
[Federer, Nadal, Djokovic and Murray, Love. / Giora Simchoni]

ggplot2



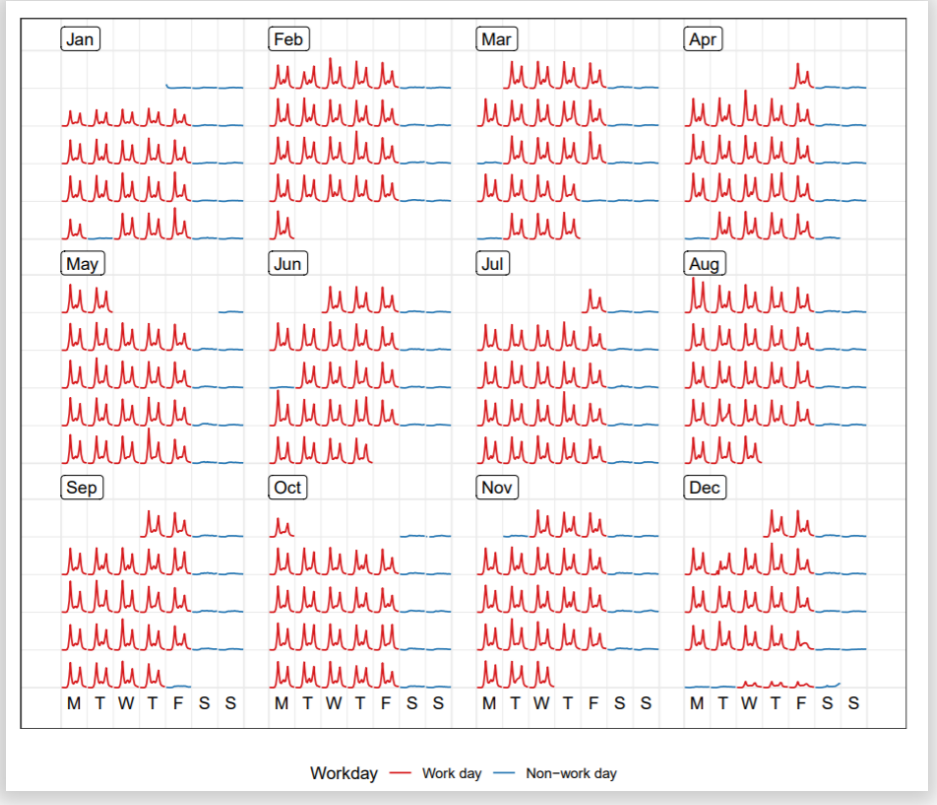
[NYT-style urban heat island maps / Katie Jolly]

ggplot2



A map of marriage rates, state by state / Unkown

ggplot2



[Calendar-based graphics for visualizing people's daily schedules / Earo Wang]

The Community

- 100% Open Source on Github
- Cheatsheet for everything
- Documentation for humans, Packages websites, Webinars, Free Books (start with R4DS)
- Rstudio Community forum
- RLadies worldwide branches (who will pick up the 🥊 and create RLadies TLV?)
- Very strong on Twitter #rstats