A Case for the Tidyverse

ECRC Data Science

Presented by Morgan Essex, AG Forslund

25* May 2020

Agenda

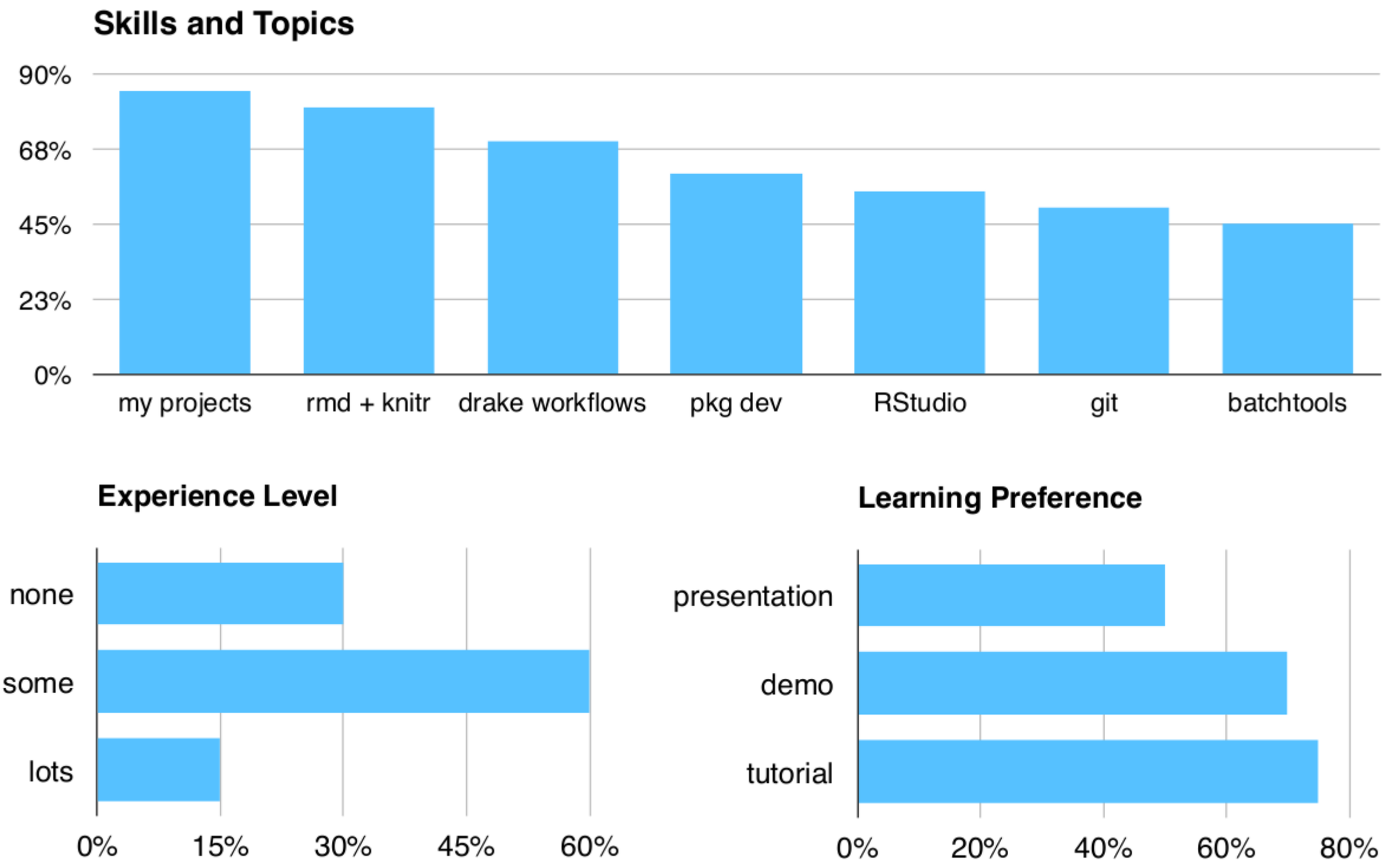
- Results from the survey
- *Theory*: tidyverse philosophy & ecosystem
  - Basic data structures and syntax
  - Core packages and functions
- *Demo*: gut microbiome diversity visualization
- Planning the next meeting(s)

Agenda

- Results from the survey
- *Theory*: tidyverse philosophy & ecosystem
  - Basic data structures and syntax
  - Core packages and functions
- *Demo*: gut microbiome diversity visualization
- Planning the next meeting(s)

Knowledge Sharing Survey Results

21 responses



Follow Up Resources

Slides and code will be available at github.com/sxmorgan/ecrc-data-science

Tutorials will be available on my website at sxmorgan.netlify.app

10 May 2020 — **Gut Microbiome Diversity Visualization**

A short workflow for generating alpha and beta diversity plots using taxonomic gut microbiome data from patients with autoimmune diseases.

5 May 2020 — **A Simple Tidyverse Workflow**

Exploratory data analysis with COVID-19 testing and prognosis in the US.

27 Apr 2020 — **A Case for the Tidyverse**

The tidyverse is a set of actively developed and well-maintained R packages to facilitate the typical data analysis workflow. Here's why you should use it in your projects.

Docs for packages linked in the upper right hand corner of slides

Agenda

- Results from the survey
- *Theory*: tidyverse philosophy & ecosystem
  - Basic data structures and syntax
  - Core packages and functions
- *Demo*: gut microbiome diversity visualization
- Planning the next meeting(s)

What is the Tidyverse?

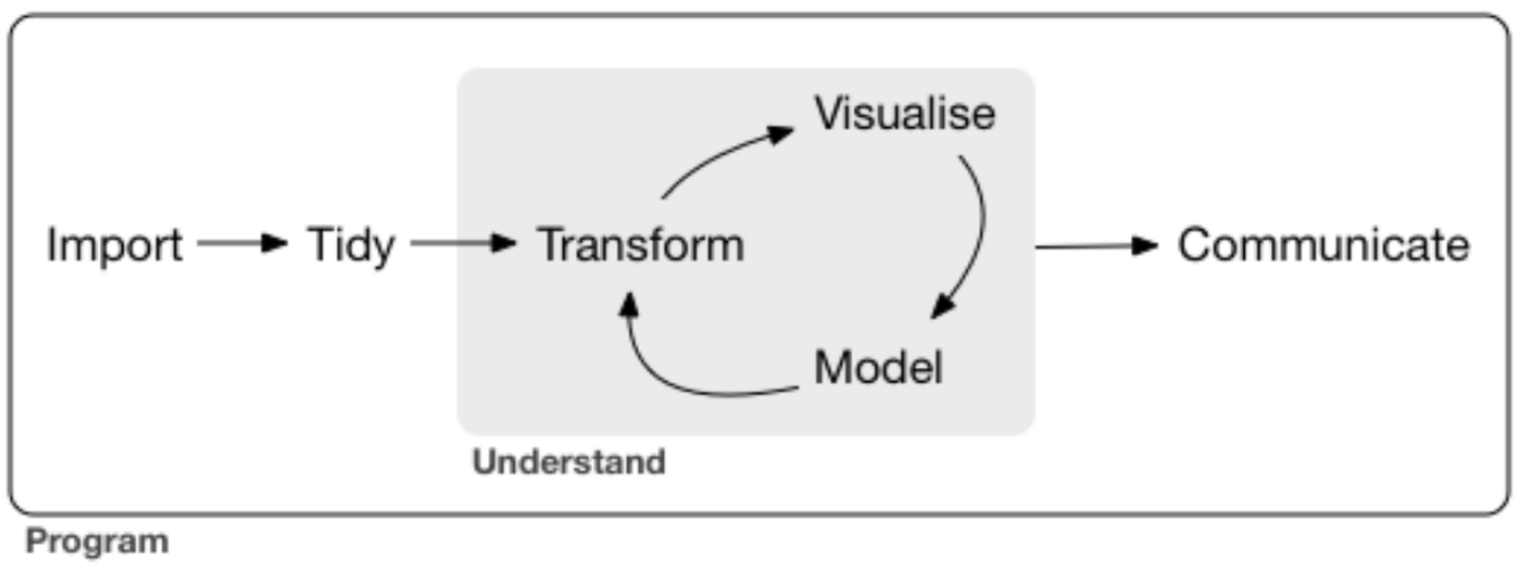
"an opinionated collection of R packages designed for data science sharing an underlying design philosophy, grammar, and data structures"

An *ecosystem* for doing data science

- functional specialization
- flexible and adaptable

A *coherent* set of packages

- designed to be easy to learn

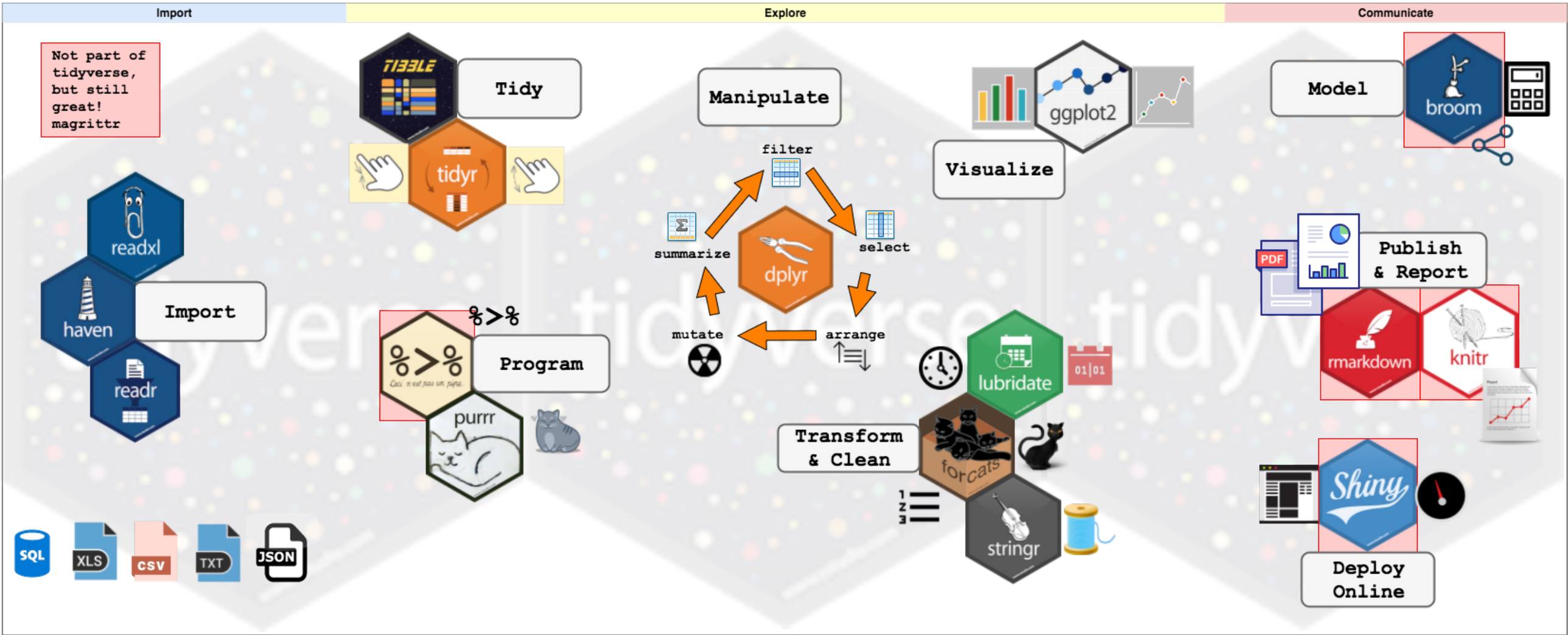


What is the Tidyverse?

"an opinionated collection of R packages designed for data science sharing an underlying design philosophy, grammar, and data structures"

An *ecosystem* for doing data science

- functional specialization
- flexible and adaptable



Data Structures in R

Vectors

- Any number of elements
- Single atomic data type

Matrices

- M rows x N cols
- Single atomic data type

```
> head(mtcars)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

Lists

- Any number of elements
- Complex data types (objects)
- Recursive (lists of lists)

Data frames

- M rows x N cols
- List of same-length vectors
- Multiple atomic data types

The best of both

`library(tibble)`

Tibbles

- "Lazy and surly" data frame
- No rownames
- Smarter console printing

```
> class(mtcars.tbl)
[1] "tbl_df"     "tbl"        "data.frame"
```

```
> mtcars.tbl
# A tibble: 32 x 12
   model              mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
   <chr>            <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
 1 Mazda RX4         21       6  160    110  3.9   2.62  16.5     0     1     4     4
 2 Mazda RX4 Wag     21       6  160    110  3.9   2.88  17.0     0     1     4     4
 3 Datsun 710        22.8     4  108     93  3.85  2.32  18.6     1     1     4     1
 4 Hornet 4 Drive    21.4     6  258    110  3.08  3.22  19.4     1     0     3     1
 5 Hornet Sportabout 18.7     8  360    175  3.15  3.44  17.0     0     0     3     2
 6 Valiant           18.1     6  225    105  2.76  3.46  20.2     1     0     3     1
 7 Duster 360        14.3     8  360    245  3.21  3.57  15.8     0     0     3     4
 8 Merc 240D         24.4     4  147.    62  3.69  3.19  20       1     0     4     2
 9 Merc 230          22.8     4  141.    95  3.92  3.15  22.9     1     0     4     2
10 Merc 280          19.2     6  168.   123  3.92  3.44  18.3     1     0     4     4
# … with 22 more rows
```

- Columns can be lists!

```
> nest(mtcars.tbl, data = -cyl)
# A tibble: 3 x 2
    cyl data
  <dbl> <list>
1     6 <tibble [7 × 11]>
2     4 <tibble [11 × 11]>
3     8 <tibble [14 × 11]>
```

```
# A tibble: 3 x 4
    cyl data                models models
  <dbl> <list>              <list> <list>
1     6 <tibble [7 × 11]>   <lm>   <gg>
2     4 <tibble [11 × 11]>  <lm>   <gg>
3     8 <tibble [14 × 11]>  <lm>   <gg>
```

Pipe operators and syntax

library(*magrittr*)

Typical R syntax has many annoying features

- Operations structured from the inside out
- Involves nested function calls

```
# base R syntax
n.cyl <- length(unique(mtcars$cyl))
```

Whatever is before the %>% will be 'piped'
to the first argument in the next operation

```
# recommended
n.cyl <- mtcars %>%
    use_series(cyl) %>%
    unique() %>%
    length()

# ideal for console
mtcars %>%
    use_series(cyl) %>%
    unique() %>%
    length() -> n.cyl
```

Reading and writing code is more intuitive

Minimizes storage of intermediate variables

%<>% saves in place

-> assigns to variable at the end of a pipe

Easy to add or comment out steps in a sequence of operations

Plays well with base R commands too

Tidy data frames

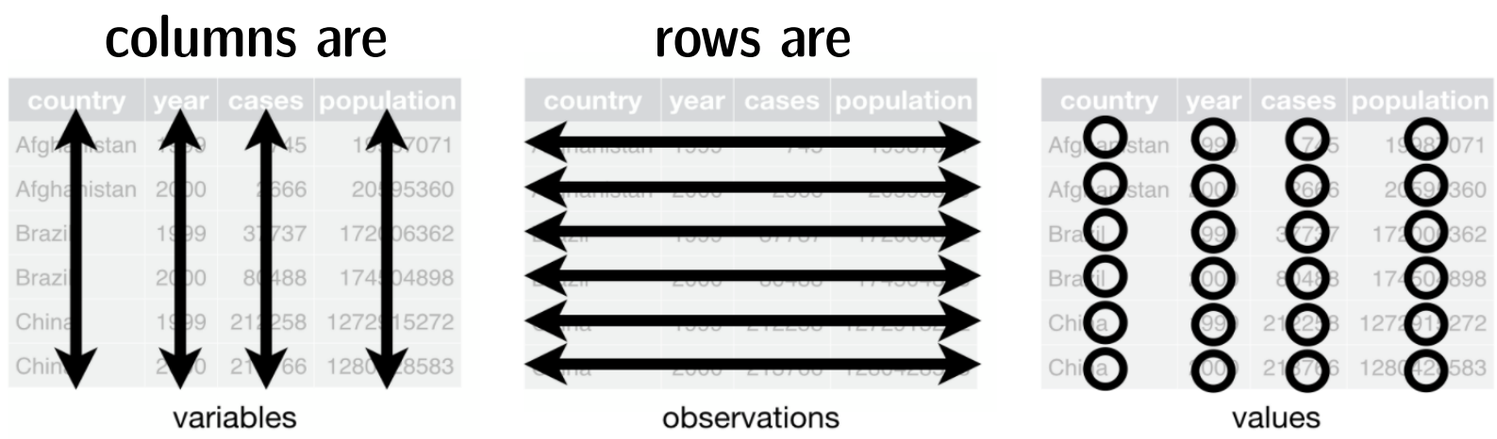
`library(tidyr)`

Maybe you've encountered data in this (or some other weird) format

```
> gapminder.wide
# A tibble: 142 x 13
   country     `1952` `1957` `1962` `1967` `1972` `1977` `1982` `1987` `1992` `1997` `2002` `2007`
   <fct>        <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
 1 Afghanistan   779.   821.   853.   836.   740.   786.   978.   852.   649.   635.   727.   975.
 2 Albania      1601.  1942.  2313.  2760.  3313.  3533.  3631.  3739.  2497.  3193.  4604.  5937.
 3 Algeria      2449.  3014.  2551.  3247.  4183.  4910.  5745.  5681.  5023.  4797.  5288.  6223.
 4 Angola       3521.  3828.  4269.  5523.  5473.  3009.  2757.  2430.  2628.  2277.  2773.  4797.
 5 Argentina    5911.  6857.  7133.  8053.  9443. 10079.  8998.  9140.  9308. 10967.  8798. 12779.
 6 Australia   10040. 10950. 12217. 14526. 16789. 18334. 19477. 21889. 23425. 26998. 30688. 34435.
 7 Austria      6137.  8843. 10751. 12835. 16662. 19749. 21597. 23688. 27042. 29096. 32418. 36126.
 8 Bahrain      9867. 11636. 12753. 14805. 18269. 19340. 19211. 18524. 19036. 20292. 23404. 29796.
 9 Bangladesh    684.   662.   686.   721.   630.   660.   677.   752.   838.   973.  1136.  1391.
10 Belgium      8343.  9715. 10991. 13149. 16672. 19118. 20980. 22526. 25576. 27561. 30486. 33693.
# … with 132 more rows
```

What makes the following "tidy"?

```
> gapminder.long
# A tibble: 1,704 x 3
   country     year  gdpPercap
   <fct>       <chr>     <dbl>
 1 Afghanistan 1952       779.
 2 Afghanistan 1957       821.
 3 Afghanistan 1962       853.
 4 Afghanistan 1967       836.
 5 Afghanistan 1972       740.
 6 Afghanistan 1977       786.
 7 Afghanistan 1982       978.
 8 Afghanistan 1987       852.
 9 Afghanistan 1992       649.
10 Afghanistan 1997       635.
# … with 1,694 more rows
```



Pivoting a data frame

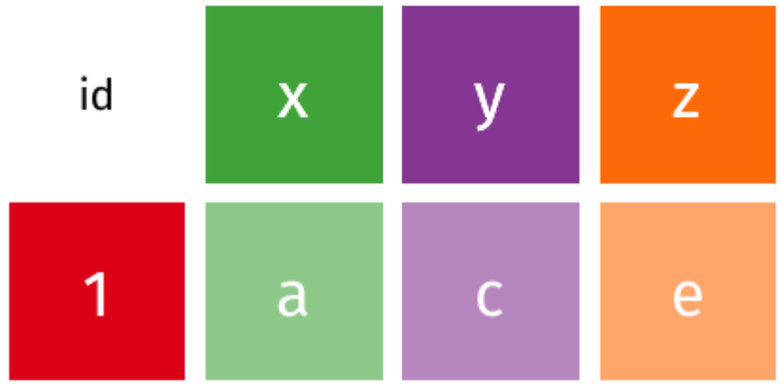
`library(tidyr)`

Aka melting, casting, reshaping

"*gather*" columns into key-value pairs
- more rows, less columns
- creates *longer* data frame

"*spread*" key-value pairs into columns
- more columns, less rows
- creates *wider* data frame

wide



For the tibbles on previous slide:

```
gapminder.long <- gapminder.wide %>%
  pivot_longer(cols = -country,
               names_to = 'year',
               values_to = 'gdpPercap')
```

```
gapminder.wide <- gapminder.long %>%
    pivot_wider(names_from = 'year',
                values_from = 'gdpPercap')
```

Data frame manipulations

`library(dplyr)`

variables = columns, cases = rows

mutate() adds new variables that are functions of existing variables
select() picks variables based on their names*
filter() picks cases based on their values
summarize() reduces multiple values down to a single summary
arrange() changes the ordering of the rows

*select() helpers:
- *starts_with('')* (string prefix), *ends_with('')* (string suffix)
- *contains('')* (string), *matches('[]')* (regular expression)
- *num_range()*, *all_of()*, *any_of()*, *everything()*, *last_col()*

_at() manipulates specific variables using *vars()*
_if() manipulates variables that meet logical condition/function
_all() manipulates all variables

Grouping and Nesting

library(*dplyr*)
library(*tidyr*)

dplyr primarily *transforms* data frames

- manipulations from previous slide
- various *_join()* operations (merge)
- *group_by()* and *ungroup()*

```
> gapminder %>% group_by(country)
# A tibble: 1,704 x 3
# Groups:   country [142]
   country      year gdpPercap
   <fct>       <int>     <dbl>
 1 Afghanistan  1952      779.
 2 Afghanistan  1957      821.
 3 Afghanistan  1962      853.
 4 Afghanistan  1967      836.
 5 Afghanistan  1972      740.
 6 Afghanistan  1977      786.
 7 Afghanistan  1982      978.
 8 Afghanistan  1987      852.
 9 Afghanistan  1992      649.
10 Afghanistan  1997      635.
# … with 1,694 more rows
```

tidyr primarily *restructures* data frames

- *pivot_longer()* and *pivot_wider()*
- *nest()* and *unnest()*

```
> gapminder %>% nest(data = -country)
# A tibble: 142 x 2
   country     data
   <fct>       <list>
 1 Afghanistan <tibble [12 × 2]>
 2 Albania     <tibble [12 × 2]>
 3 Algeria     <tibble [12 × 2]>
 4 Angola      <tibble [12 × 2]>
 5 Argentina   <tibble [12 × 2]>
 6 Australia   <tibble [12 × 2]>
 7 Austria     <tibble [12 × 2]>
 8 Bahrain     <tibble [12 × 2]>
 9 Bangladesh  <tibble [12 × 2]>
10 Belgium     <tibble [12 × 2]>
# … with 132 more rows
```

Agenda

- Results from the survey
- *Theory*: tidyverse philosophy & ecosystem
  - Basic data structures and syntax
  - Basic packages and functions
- *Demo*: gut microbiome diversity visualization
- Planning the next meeting(s)

String and factor manipulations

```
library(stringr)
library(forcats)
```

---

forcats: for dealing with categorical variables (string or factor!)

- excellent in combination with *mutate_at()*

fct_recode() changes levels by hand
fct_relevel() reorder levels by hand
fct_reorder() reorder levels by another variable
fct_collapse() combine factor levels into groups by hand
fct_explicit_na() makes NA a level

stringr: for dealing with strings

- excellent in combination with *filter()*

str_detect() presence/absence of (sub)string (returns T/F)
str_remove() remove matched patterns
str_replace() replace matched patterns
str_split() split strings by patterns
str_to_title() Capitalizes First Letters